Créer un projet pour la carte PICDEM avec MPLAB v6.6 et L'ICD2

Tout d'abord, créez un projet par la commande **Project->Project Wizard…** :

1/ La première étape permet de choisir le type de micro utilisé (en tp il s'agit d'un 16F877).

2/ La deuxième étape permet de choisir le langage utilisé par votre projet (**Select a language toolsuite).**

Pour l'assembleur, choisir : `Microchip MPASM Tool Suite` (choix par défaut)
Pour le C, choisir : `HI-TECH PICC Tool Suite`.

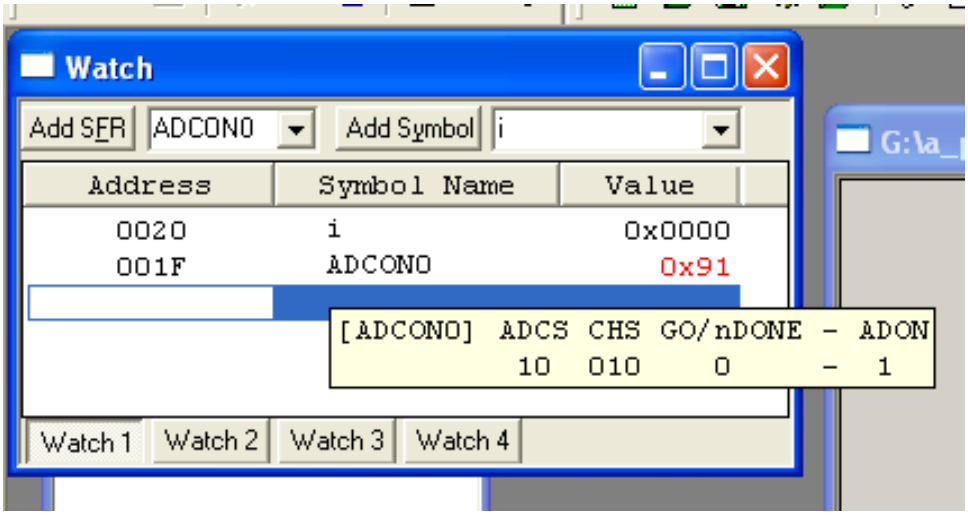
3/ Vous devez entrez ensuite le nom du projet et le nom du répertoire qui va le contenir. Pour s'y retrouver parmi les nombreux fichiers qui vont être générés, il est important de créer un nouveau répertoire par projet. Le nom du répertoire devrait être le même que le nom du project. Ensuite sauver l'espace de travail (workspace) sous le même nom que le projet et dans le même répertoire.
Par exemple le projet `tp1` donnera lieu à la création d'un répertoire `c:\pic\tp1`, d'un fichier projet `tp1.mcp` et d'un fichier workspace `tp1.mcw` se trouvant tous les deux dans le répertoire `c:\pic\tp1`.

4/ Vous pouvez à ce stade ajouter des fichiers existants dans votre projet. Il est souvent préférable de copier ces fichiers dans le répertoire du projet. Pour cela cocher la case se trouvant à gauche du nom du chemin du fichier ajouté.
Vous pouvez ajouter à cette étape les fichiers biosdemx.h et biosdemx.lib de la librairie de tp.
(Vous pourrez également le faire par la suite par la commande **Project->Add Files to Project).**
5/ Créez ensuite un fichier source (assembleur ou C) par le menu **File->New**, sauvez le dans le répertoire du projet avec l'extension correspondant au langage (.asm ou .c).
Il faut ensuite ajouter ce fichier au projet par le menu :
Project->Add Files to Project ou bien en cliquant avec le bouton droit sur la ligne `Source Files` de la fenêtre Project (Menu `View->Project` pour la faire apparaître) . Egalement dans cette fenêtre, il est possible de modifier les options d'assemblage ou de compilation d'un fichier par le menu `Build Options…`

5/ Il faut alors assembler ou compiler votre projet par les commandes
Project->Make ou `Project->Build All`.
La fenêtre `Output` montre alors les erreurs ou avertissements éventuels et si le message `BUILD SUCCEEDED` apparaît c'est que le compilation a réussit. Sinon, en cliquant sur la ligne d'erreur, le pointeur de la souris est positionné sur la ligne du fichier source correspondante.

6/ Vous pouvez alors choisir un outil pour la mise au point par **Debugger->Select tool**. Sélectionner MPLAB SIM pour travailler en simulation ou bien MPLAB ICD2. Dans ce cas, il faut télécharger votre programme sur la carte cible par la commande
Debugger->Program.

7/ Vous pouvez alors lancer votre programme (**Run**) ou l'exécuter en pas à pas (step over, step into).
Lorsque le programme est arrêté vous pouvez obtenir la valeur d'une variable en passant la souris dessus dans votre fichier source.
Pour examiner en permanence des variables ou des registres, vous pouvez créer une "**Watch Window**" par le menu **`View->Watch`** et y faire apparaître la valeur des différentes cases mémoires que vous souhaitez observer. Les valeurs dans cette fenêtre sont mise à jour lors de chaque arrêt du programme (pas à pas, point d'arrêt ou stop).
Dans la watch window vous pouvez fixer la base utilisée pour représenter chaque variable par le menu click droit->Properties. La base par défaut est l'*hexadécimal*.
En activant la fonction "SFR Bitfield MouseOver" par un click droit dans la watch window, vous obtiendrez le détail des bits d'un registre dédié (SFR) en passant la souris dessus.



Quelques particularités du langage C pour PIC16 de Hi-Tech

Le langage PICC de Hi-TECH comporte certaines caractéristiques **non standards** qui le rendent particulièrement bien adapté à l'écriture d'applications embarquées sur µC PIC. En voici quelques unes, vous trouverez les plus de détails dans le manuel du langage (fichier manuel.pdf).

- bits de configurations :
On peut fixer la valeur des bits de configurations directement dans le source C en utilisant la macro `__CONFIG( )` définie dans `pic.h`
Exemple : `__CONFIG(0x3539);`

- Accès direct aux bits de la mémoire Data :
On peut manipuler directement des bits en mémoire en utilisant des variables de type **`bit`**
(Attention les variables de type bit doivent être de la classe statique (globale ou locale statique)
Exemple :

```
{ static bit b,c ;
.............
if (b==1)
     {
        c=!c ;
        b=b || c ;
     }
.............
}
```

Ces variables sont de type booléen, les opérateurs qui s'appliquent normalement sur ces variables sont les opérateurs booléens `(==,&&,||,!)` et pas les opérateurs de manipulation de bit du C.
Le fichier `pic1687x.h` (inclus automatiquement par l'inclusion de `pic.h`) utilise abondamment le type bit pour définir tous les bits utiles du PIC. L'écriture des programmes en est grandement facilitée.
Exemple :

```
#include <pic.h>
.................
TRISB1 = 0 ;  /* RB1 en sortie (bit 1 de TRISB à 0) */
        ..................
if (RB0==0)   /* Test du bit 0 de PORTB */
     {
        RB1 = 1 ;   /* Mise à 1 du bit 1 de PORTB */
     }
else
     {
        RB1 = 0 ;  /* Mise à 0 du bit 1 de PORTB */
     }
          .................
```

Quelques particularités du langage C pour PIC16 de Hi-Tech

- Placer des variables ailleurs qu'en banque 0 :

PICC place systématiquement les variables (statiques ou automatiques) en banque 0. Pour placer une variable dans une autre banque, il faut précéder sa déclaration d'un qualifieur `bank1`, `bank2` ou `bank3`. (Il n'y a pas de qualifieur bank0 qui est la banque par défaut). Attention seules les variables de la classe statique peuvent être placées ailleurs qu'en banque 0. L'utilisation des variables dans les autres banques est ensuite transparente.

Exemple :

```
bank1 int i ;       /* globale en banque 1 */
bank2 char tab[5];  /* globale en banque 2 */
int j ;             /* globale en banque 0 */
.............

void main ( void)
     {
       static bank3 unsigned char t=255 ; /* locale statique en
banque 3 */

       for ( i=0; i <5 ; i++)
          {
             tab[i] = t ;
          }
       ........................

     }
```

Remarque :
Lorsque le message "`Can't find 0x?? words for psect rbss_0 in segment BANK0`" apparaît lors de l'édition de liens, c'est qu'il n'y a plus assez de place dans la banque 0. Vous devez alors essayer de placer certaines variables dans les autres banques.

La librairie biosdem2

Afin de faciliter l'écriture des premiers Tp, une librairie de fonctions spécifiques à la carte PICDEM2+ a été crée. Une librairie est un ensemble de fonctions pré-compilées que vous pouvez utiliser comme les fonctions de la librairie standard du C après avoir ajouté cette librairie à votre projet. Le fichier à inclure est `biosdem2x.lib` (où x est le numéro de la version). Un source utilisant la librairie devra inclure le fichier en-tête `biosdem2x.h`.

Liste des fonctions et macros de biosdem221 (version 2.1)

Fonctions d'attente

`DelayUs(x);` : Macro qui bloque l'exécution du programme pendant x microsecondes (x est de type unsigned char (0-255))

`void DelayMs(unsigned char x);` : bloque l'exécution du programme pendant x millisecondes.

En appelant plusieurs fois ces fonctions il est possible de bloquer l'exécution du programme pendant un temps arbitrairement long (au prix d'une légère perte de précision).

Fonctions d'écriture sur l'afficheur lcd

`void lcd_init(void);` : Initialisation du module lcd de la carte picdem2+. Cette fonction doit être obligatoirement appelé une fois avant toute utilisation des autres fonctions lcd.

`void lcd_clear(void);` : Efface l'écran lcd et positionne le curseur sur la première position de la première ligne.

`lcd_goto(unsigned char pos);` : Positionne le curseur à la position `pos`.
Cette fonction utilise deux constantes `LCD_LINE_ONE` et `LCD_LINE_TWO` pour fixer la première position de la première ligne et de la deuxième ligne (respectivement).
Exemple : `lcd_goto(LCD_LINE_TWO+2);` place le curseur au troisième caractère de la 2ème ligne.

`void lcd_putchar(char);` : Affiche un caractère à la position du curseur.

`void lcd_puts(const char * s);` : Affiche une chaîne de caractère à partir de la position du curseur.

`unsigned char lcd_printf(const char * f, ...);` : Cette fonction est similaire à la fonction printf de la libraire standard du C, mais elle affiche sur l'écran lcd à partir de la position du curseur. Elle retourne le nombre de caractères réellement affichés.
Les formats reconnus sont :
`%d` décimal signé
`%u` décimal non signé
`%x` ou `%X` hexadécimal
`%o` octal
`%c` caractère

`unsigned char lcd_small_printf(const char * f, int v);` : Cette fonction est similaire à la fonction lcd_printf, mais elle ne prend en compte qu'un seul format % obligatoire pour afficher un seul paramètre de type int ou char. Elle retourne le nombre de caractères réellement affichés. L'intérêt de cette fonction est sa taille réduite par rapport à lcd_printf ce qui permet d'utiliser la librairie avec la version freeware de picc ou pour des pics à la taille mémoire limité. En la combinant avec lcd_puts et lcd_putchar il est possible de se passer complètement de lcd_printf.
Par exemple :

```
char c;
int x ;
….
lcd_printf("x=%d,c=%c toto!",x,c);
```

peut se remplacer par :

```
lcd_small_printf("x=%d,c=",x);//un seul %,une seule variable
lcd_putchar(c);  // remplace le %c
lcd_puts(" toto!");
```

Les formats reconnus sont (un et un seul par appel) :
`%d` décimal signé
`%u` décimal non signé
`%x` ou `%X` hexadécimal
`%o` octal

`unsigned char lcd_printf1(const char * f, ...);` : Cette fonction est identique à la fonction lcd_printf mais utilise un buffer fourni par l'utilisateur à la place du buffer de 17 caractères utilisé par lcd_printf. Pour utiliser cette fonction, il faut déclarer une variable globale nommée `printf1_buffer` en banque 0 par :

```
far char printf1_buffer[N]  ;
```

où est N est la place nécessaire.
(Cette fonction permet de limiter la place RAM utilisée au strict nécessaire et peut également servir à exploiter la chaîne affichée sans utiliser un appel supplémentaire à `sprintf()`).

Fonctions de conversion analogique/numérique :

`void adc_on(void);` : Démarre le convertisseur analogique/numérique du PIC.
(Dans la version 1.0 de la librairie seule RA0 est utilisable en entrée analogique).

`unsigned char adc_read_8b(unsigned char ch);` : Lance une conversion du canal ch et retourne le 8 bits les plus significatifs du résultat. (Dans la version 2.1 de la librairie seule RA0 est utilisable donc ch est obligatoirement égal à 0).

Fonctions d'accès aux périphériques i2c :

`void i2c_init(void);` Initialise le module i2c du pic dans le mode master , adresse 7 bits à une vitesse de 100kHz. Cette fonction doit être appelée une fois avant toute utilisation d'un périphérique i2c.

`signed char tc74_read(void) ;` Cette fonction retourne la température en °C relevée par le thermomètre TC74A0 présent sur la carte picdem2.