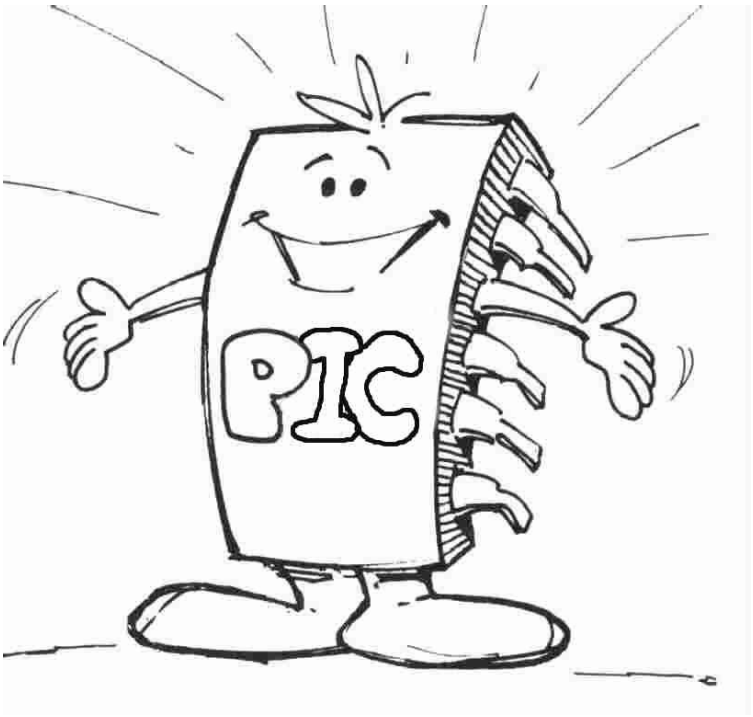
PROGRAMMER LES PIC**12** et PIC**16** EN **C** AVEC LE COMPILATEUR CC5X v2



Equipe de formation sur les microcontrôleurs PIC

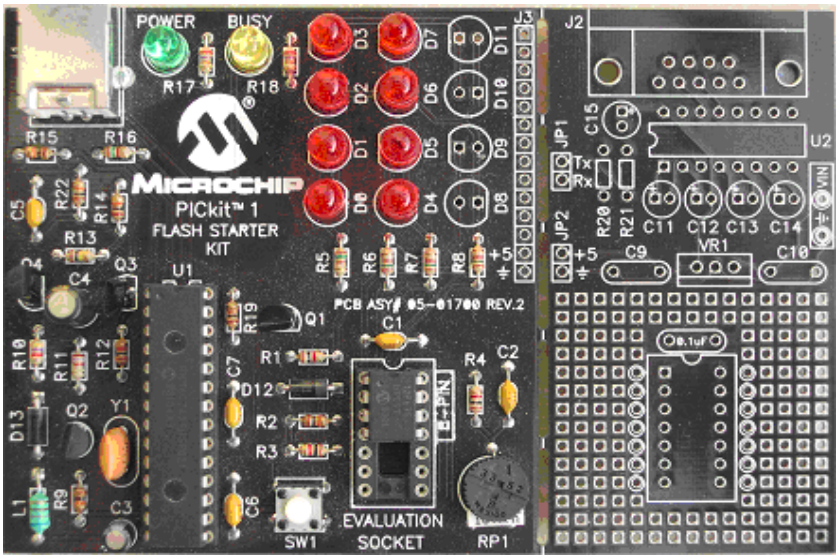
Christian Dupaty
Lycée Fourcade
13120 Gardanne
Académie d'Aix-Marseille
c.dupaty@aix-mrs.iufm.fr

SOMMAIRE

www.microchip.com

http://www.bknd.com/cc5x/

Les exemples de ce cours fonctionnent avec le simulateur de MPLAB 6.40 et sont programmables sur PICKIT1



A lire :
Trucs et astuces sur PIC12/PIC16 :
http://www.microchip.com/download/lit/pline/picmicro/families/12f6xx/40040b.pdf

cc5x-32.pdf documentation complète accompagnant CC5x (en anglais) http://www.bknd.com/cc5x/

Pré requis :
Connaissances élémentaires du langage C ANSI
Structures des microcontrôleurs Microchip PIC12, PIC16
Outils de développement MPLAB, PICKIT1

1. Caractéristiques générales de CC5X

Le compilateur C CC5X de la société BKD produit un code exécutable pour les microcontrôleurs PIC 12 et PIC 16, il s'intègre très simplement dans l'environnement de développement MPLAB de MICROCHIP. La version d'évaluation est limitée à 1KO de code, le type float est codé sur 24 bits. Cette version est donc tout à fait opérationnelle pour le développement d'application sur les microcontrôleurs PIC12 et PIC16 dans la limite d'1KO de programme.

Particularités
Le type int a une taille de un octet(8 bits). Pour disposer de données entières sur 16 bits, il faut les déclarer long. Les variables char sont non signées par défaut.

Types définis dans la version gratuite de CC5 (la version commerciale possèdes des types entiers sur 24 et 32 bits ainsi que des réels à virgule fixe)

```
char a8;              // 8 bits non signé !!!
signed char sc;       // 8 bits signé
int i;                // 8 bits signé   !!!
unsigned int i        // 8 bits non signé
unsigned long i16;    // 16 bits non signé
long i16;             // 16 bits signé !!!
bit pret;             // 0 ou 1

float f;              // nombre réel codé sur 24 bits
```

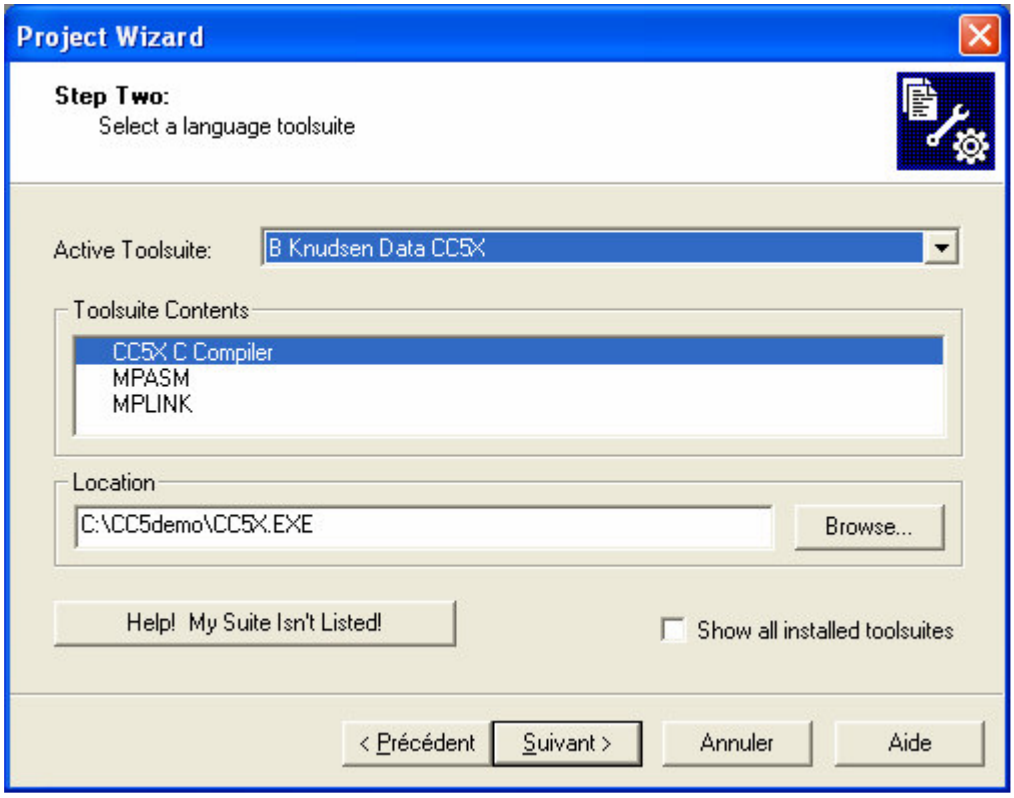
Il faut par ailleurs être très rigoureux lors d'opération avec transtypage et utiliser systématiquement des CAST.

CC5X permet d'approcher au plus prêt le noyau des PICs12 et 16 (gestion des banks, assembleur inline, accès direct aux registres internes, gestion fine des interrutions…
Il est fortement recommandé de lire « CC5X User's Manual » disponible sur
http://www.bknd.com/cc5x/

2. Installation dans MPLAB 6.xx

Le répertoire du compilateur CC5x peut être placé n' importe où sur le disque dur. Il contient l'exécutable du compilateur ainsi que les fichiers de définitions des microcontrôleurs supportés (header). Dans la procédure suivante, CC5x a été placé dans le dossier C:\CC5demo

- Démarrer MPLAB.
- Project – Project Wizard
- Choisir le microcontrôleur cible,par exemple un PIC12F675, (présent sur PICkit1)
- Selectionner l'outil **« B Knudsen Data CC5x »**



- Configurer les noms et chemins des trois outils proposés :

CC5X est C:\CC5demo\CC5x.EXE
L'assembleur et le linker sont ceux installés par MPLAB
MPASM est C:\Program Files\MPLAB IDE\MCHIP_Tools\MPASMWIN.EXE
MPLINK est C:\Program Files\MPLAB IDE\MCHIP_Tools\MPLINK.EXE

- Donner un nom et un chemin au projet. (par exemple TSTC5), il n'y a pas de fichier à inclure au projet pour l'instant.
- Vérifier que le PIC12F675 est bien actif dans Configure – Select Device

Pour les PIC12F675, le debugger peut être le simulateur ou l'ICD2., et le programmateur un PICKIT1 ou l'ICD2. ATTENTION, il n'est pas possible de faire de debug avec le PICKIT1.

Attention, dans les versions 6.30 à 6.60 de MPLAB, le debug par simulateur n'est pas activé par défaut. If faut éditer le fichier C:\Program Files\MPLAB IDE\LegacyLanguageSuites\TLCC5x.INI et remplacer la ligne Target=HEX par Target=COD

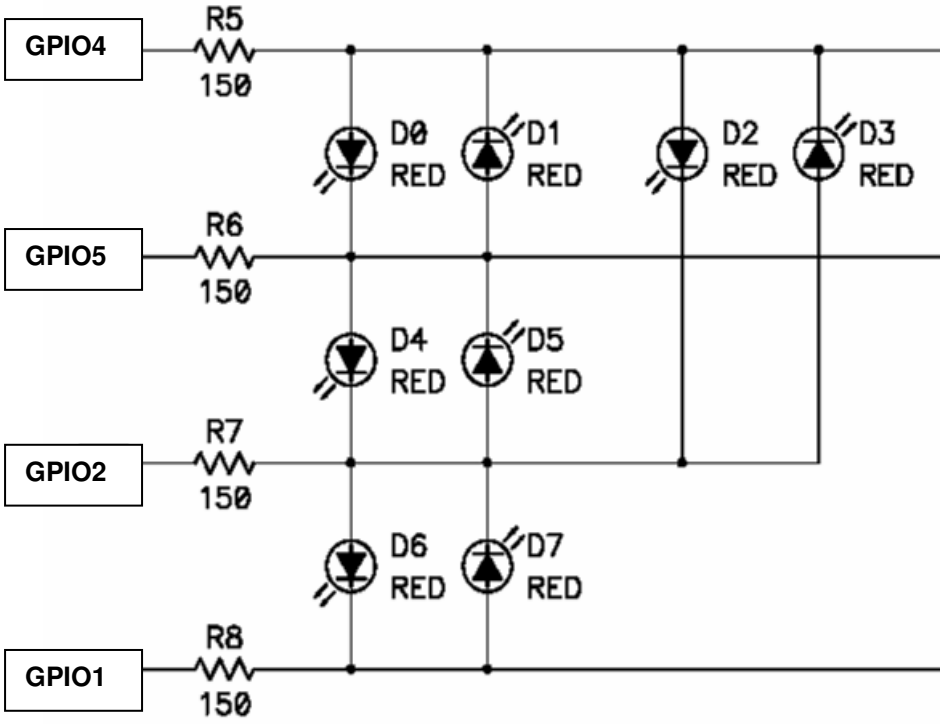
Pour utiliser ICD2 il faut placer en début de programme : #define ICD2_DEBUG
Attention, dans ce cas deux broches sont nécessaires à la connexion avec l'ICD2 et deux pour l'alimentation, il ne reste donc que 4 broches pour l'application.

Le projet est maintenant construit.

3. Exemples de commande d'E/S

Les programmes exemples fonctionnent sur PICKIT1 (sans DEBUGER ICD2) , ce KIT très économique possède 8 LEDs, un potentiomètre un bouton poussoir et est fourni avec un PIC12F675
La mise au point peut se faire à l'aide de simulateur intégré à MPLAB

Le PIC12F675 ayant 8 broches, la commande indépendante des 8 LED nécessite une manipulation particulière du PORT GPIO. . Les lignes de GPIO sont placées en entrée (HiZ) ou en sortie (0 ou 1) en fonction de la LED à allumer.



Par exemple, pour allumer D0, on place 5v sur GPIO4, 0v sur GPIO5 et GPI02 en entrée.

Programme boutonC5.C.
Ce programme allume D0 ou D1 suivant l'état du bouton sur PICKIT1

```
#pragma chip PIC12F675
```

Sélection de la cible, le compilateur inclura automatiquement 12F675.h (définitions des registres et bits du PIC)

```
bit bouton @ GPIO.3;
```

Définition d'un nom de bit (GPIO.3 est la bit 3 de GPIO)

```
void main(void)
{
      GPIO=0;
      CMCON=7;                // fonction comparateur desactivee
      ANSEL=0;                // AN0 - AN3  numeriques
      TRISIO = 0b11001111;    // allume D0 ou D1 sur PICKIT1
      GPIO=0b00010000;        // allume D0 au debut
      while(1)
            {
                  while (bouton);         // attend bouton appuie
                  GPIO.5=1;               // allume D1
                  GPIO.4=0;
                  while (!bouton);        // attend bouton relâche
                  GPIO.5=0;               // allume D0
                  GPIO.4=1;
            }
}
```

Programme LEDC5.C

Ce programme réalise une chenillard sur les LED D0 à D7 avec une tempo logicielle.
Deux tableaux en ROM définissent les 8 états des sorties et sens du port GPIO. Une boucle sans fin recopie ces tableaux dans TRISIO et GPIO avec à chaque boucle appelle d'une fonction tempo.

```
#pragma chip PIC12F675
const unsigned char sortie[]=
{0b00010000,0b00100000,0b00010000,0b00000100,0b00100000,0b00000100,0b000001
00,0b00000010};
const unsigned char sens[]=
{0b11001111,0b11001111,0b11101011,0b11101011,0b11011011,0b11011011,0b111110
01,0b11111001};

unsigned char varchar;  // une variable 8 bits et une 16 bits pour tempo
unsigned long varlong;  // attention : les int sont codés sur 8 bits
unsigned char i;        // indice des tableaux

void tempo(unsigned char n1, unsigned long n2);

void main(void)
{
      varchar=100;            // valeurs produisant une tempo d'environ 0.5s
      varlong=1000;
      GPIO=0;
      CMCON=7;          // fonction comparateur desactivee
      ANSEL=0;          // AN0 - AN3 entrees numeriques
      TRISIO = 0x00;
      GPIO=0b00000000;  // GPIO en sortie
      while(1)
            {
            for (i=0;i<8;i++)
                  {
                  TRISIO=sens[i];
                  GPIO=sortie[i];
                  tempo(varchar,varlong);
                  }
            }
}

void tempo(unsigned char n1, unsigned long n2)
{     unsigned char cpt ;
      while (--n1) {
                        cpt=n2;
                        while (--cpt);
                   }
}
```

Sélection de la cible

Prototype

Chenillement D0 – D7

Page **Erreur ! Signet non défini.**, le listing du programme assembleur généré., avec dans la colonne de gauche le source C correspondnnt

```
           0001
           0002 ; CC5X Version 3.2, Copyright (c) B Knudsen Data
           0003 ; C compiler for the PICmicro family
           0004 ; ************  26. Sep 2004  13:58  *************
           0005
           0006       processor  12F675
           0007       radix  DEC
           0008
     0002  0009 PCL         EQU   0x02
     000A  0010 PCLATH      EQU   0x0A
     0000  0011 Carry       EQU   0
     0002  0012 Zero_       EQU   2
     0005  0013 RP0         EQU   5
     0005  0014 GPIO        EQU   0x05
     0019  0015 CMCON       EQU   0x19
     0085  0016 TRISIO      EQU   0x85
     009F  0017 ANSEL       EQU   0x9F
     0024  0018 varchar     EQU   0x24
     0025  0019 varlong     EQU   0x25
     0027  0020 i           EQU   0x27
     0020  0021 n1          EQU   0x20
     0021  0022 n2          EQU   0x21
     0023  0023 cpt         EQU   0x23
     0020  0024 ci          EQU   0x20
           0025
0000 2816  0026       GOTO main
           0027
           0028   ; FILE ledC5.c
           0029                       ;#pragma chip PIC12F675
           0030                       ;const unsigned char sortie[]=
{0b00010000,0b00100000,0b00010000,0b00000100,0b00100000,0b00000100,0b00000100,0b00000010};
           0031                       ;const unsigned char sens[]=
{0b11001111,0b11001111,0b11101011,0b11101011,0b11011011,0b11011011,0b11111001,0b11111001};
           0032                       ;
           0033 ;unsigned char varchar;      // une variable 8 bits et une 16 bits pour tempo
           0034 ;unsigned long varlong;      // attention : les int sont codés sur 8 bits
           0035                       ;unsigned char i;             // indice des tableaux
           0036                       ;
           0037                       ;void tempo(unsigned char n1, unsigned long n2);
           0038                       ;
           0039                       ;void main(void)
           0040                       ;{
           0041 _const1
0001 00A0  0042       MOVWF ci
0002 018A  0043       CLRF  PCLATH
0003 0820  0044       MOVF  ci,W
0004 390F  0045       ANDLW .15
0005 0782  0046       ADDWF PCL,1
0006 3410  0047       RETLW .16
0007 3420  0048       RETLW .32
0008 3410  0049       RETLW .16
0009 3404  0050       RETLW .4
000A 3420  0051       RETLW .32
000B 3404  0052       RETLW .4
000C 3404  0053       RETLW .4
000D 3402  0054       RETLW .2
000E 34CF  0055       RETLW .207
000F 34CF  0056       RETLW .207
0010 34EB  0057       RETLW .235
0011 34EB  0058       RETLW .235
0012 34DB  0059       RETLW .219
0013 34DB  0060       RETLW .219
0014 34F9  0061       RETLW .249
0015 34F9  0062       RETLW .249
           0063 main
           0064                       ;      varchar=100;          // valeurs produisant une
tempo d'environ 0.5s
0016 3064  0065       MOVLW .100
0017 00A4  0066       MOVWF varchar
           0067                       ;      varlong=1000;
0018 30E8  0068       MOVLW .232
0019 00A5  0069       MOVWF varlong
001A 3003  0070       MOVLW .3
001B 00A6  0071       MOVWF varlong+1
           0072                       ;      GPIO=0;
001C 1283  0073       BCF   0x03,RP0
001D 0185  0074       CLRF  GPIO
```

```
          0075                      ;       CMCON=7;              // fonction comparateur
desactivee
001E 3007 0076       MOVLW .7
001F 0099 0077       MOVWF CMCON
          0078                      ;       ANSEL=0;              // AN0 - AN3 entrees
numeriques
0020 1683 0079       BSF   0x03,RP0
0021 019F 0080       CLRF  ANSEL
          0081                      ;       TRISIO = 0x00;
0022 0185 0082       CLRF  TRISIO
          0083                      ;       GPIO=0b00000000;      // GPIO en sortie
0023 1283 0084       BCF   0x03,RP0
0024 0185 0085       CLRF  GPIO
          0086                      ;       while(1)
          0087                      ;               {
          0088                      ;               for (i=0;i<8;i++)
0025 01A7 0089 m001  CLRF  i
0026 3008 0090 m002  MOVLW .8
0027 0227 0091       SUBWF i,W
0028 1803 0092       BTFSC 0x03,Carry
0029 2825 0093       GOTO  m001
          0094                      ;                       {
          0095                      ;                       TRISIO=sens[i];
002A 3008 0096       MOVLW .8
002B 0727 0097       ADDWF i,W
002C 2001 0098       CALL  _const1
002D 1683 0099       BSF   0x03,RP0
002E 0085 0100       MOVWF TRISIO
          0101                      ;                       GPIO=sortie[i];
002F 0827 0102       MOVF  i,W
0030 2001 0103       CALL  _const1
0031 1283 0104       BCF   0x03,RP0
0032 0085 0105       MOVWF GPIO
          0106                      ;                       tempo(varchar,varlong);
0033 0824 0107       MOVF  varchar,W
0034 00A0 0108       MOVWF n1
0035 0825 0109       MOVF  varlong,W
0036 00A1 0110       MOVWF n2
0037 0826 0111       MOVF  varlong+1,W
0038 00A2 0112       MOVWF n2+1
0039 203C 0113       CALL  tempo
          0114                      ;                       }
003A 0AA7 0115       INCF  i,1
003B 2826 0116       GOTO  m002
          0117                      ;               }
          0118                      ;}
          0119                      ;
          0120                      ;void tempo(unsigned char n1, unsigned long n2)
          0121                      ;{      unsigned char cpt ;
          0122 tempo
          0123                      ;       while (--n1) {
003C 03A0 0124 m003  DECF  n1,1
003D 1903 0125       BTFSC 0x03,Zero_
003E 2845 0126       GOTO  m005
          0127                      ;       cpt=n2;
003F 0821 0128       MOVF  n2,W
0040 00A3 0129       MOVWF cpt
          0130                      ;       while (--cpt);
0041 03A3 0131 m004  DECF  cpt,1
0042 1903 0132       BTFSC 0x03,Zero_
0043 283C 0133       GOTO  m003
0044 2841 0134       GOTO  m004
          0135                      ;                        }
          0136                      ;}
0045 0008 0137 m005  RETURN
          0138
     0000 0139       END
     0000 0140
     0000 0142 ; *** KEY INFO ***
     0000 0144 ; 0x003C   10 word(s)  0 % : tempo
     0000 0145 ; 0x0016   38 word(s)  3 % : main
     0000 0146 ; 0x0001   21 word(s)  2 % : _const1
     0000 0148 ; RAM usage: 8 bytes (4 local), 56 bytes free
     0000 0149 ; Maximum call level: 1
     0000 0150 ; Total of 70 code words (6 %)
```

4. Exemple d'interruption : TIMER0

Le programme LEDITC5.C est identique au précédent, la temporisation étant maintenant réalisée par le TIMER0 en interruption.

Interrupt : spécifie que la fonction est un S/P d'interruption (retour par RETFIE)
#pragma origin 4 : permet de forcer le compilateur à placer le code à une adresse spécifique. Ici 4, adresse de traitement de l'IT sur PIC12F675.
#pragma interruptSaveCheck n : désactive (avec n) le test de sauvegarde W et PCLATCH inutile ici

```
#pragma chip PIC12F675

const unsigned char sortie[]=
{0b00010000,0b00100000,0b00010000,0b00000100,0b00100000,0b00000100,0b000001
00,0b00000010};
const unsigned char sens[]=
{0b11001111,0b11001111,0b11101011,0b11101011,0b11011011,0b11011011,0b111110
01,0b11111001};

#pragma interruptSaveCheck n // pas de test d'IT, les registres W et
                              PCLATCH ne sont pas sauvegardés.
#pragma origin 4             // adresse d'IT sur PIC12

interrupt ITtimer0(void)     // SP de traitement de l'IT TIMER0
{
static unsigned char i,cptIT; // i pointe les LED, cptIT compteur
                              d'interruptions
if (++cptIT>=10)
      {
            TRISIO=sens[i];
            GPIO=sortie[i];
            i++;
            if (i>=8) i=0;
            cptIT=0;
      }
      T0IF=0;
}

void main(void)
{
      GPIO=0;                 // pour init GPIO
      CMCON=7;          // fonction comparateur desactivee
      ANSEL=0;          // AN0 - AN3 entrees numeriques
      T0CS=0;                 // horloge interne sur TIMER0 (dans OPTION)
      PSA=0;                  // prédiviseur sur TIMER0 (dans OPTION)
                              // prédiviseur par 128 lors du reset
      T0IE=1;                 // autorise IT sur débordement TIMER0
      GIE=1;                  // autorise toutes les IT
      while(1);               // boucle sans fin. Attente d'IT
}
```

Attention, les registres W, STATUS et PCLATCH ne sont pas sauvegardés à l'entrée de la fonction d'interruption. Si cela est nécessaire il faut placer en début et fin de fonction d'interruption :
int_saveregister
int_restore-register

5. Exemple sur les réels

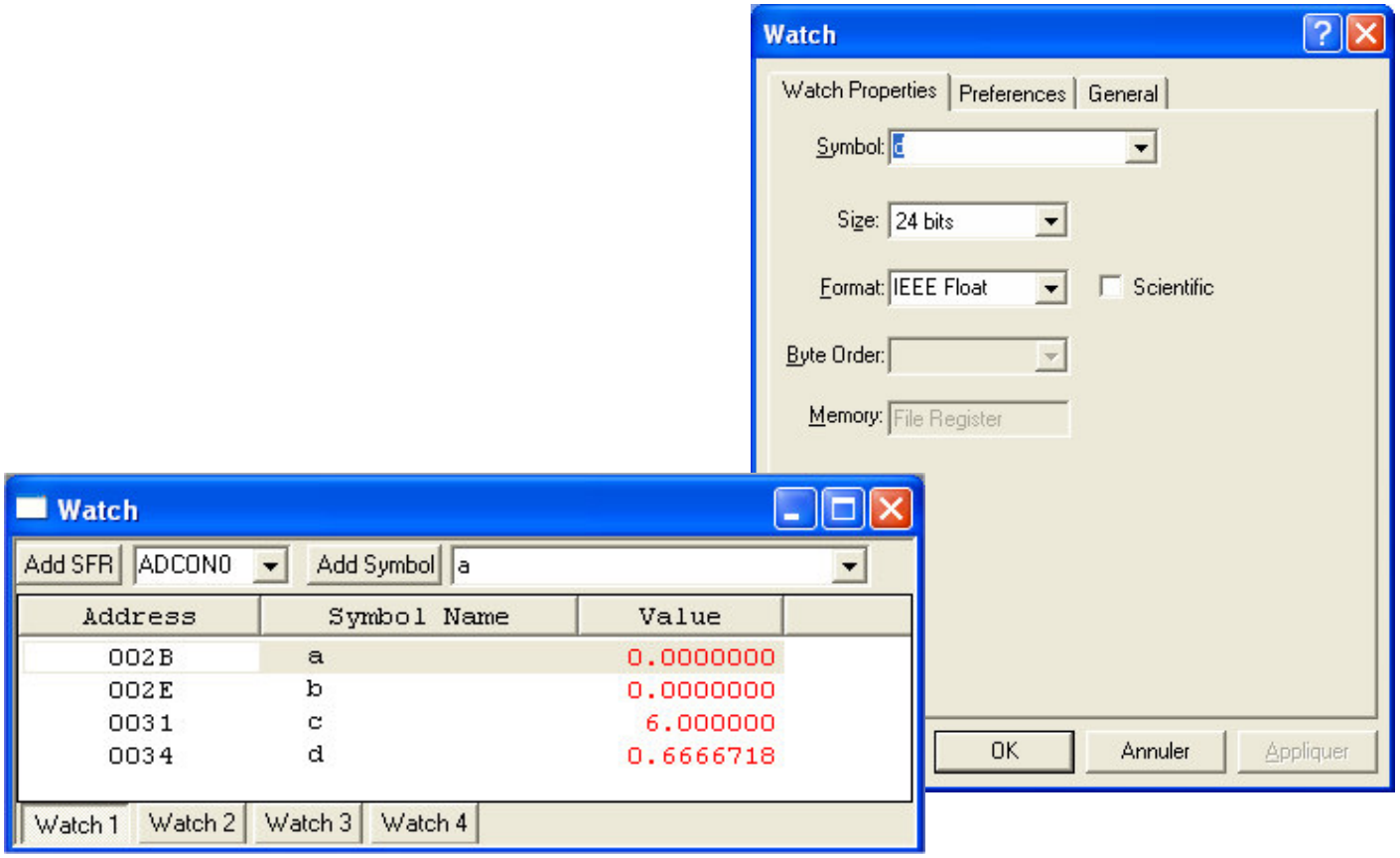
La version démo de CC5x inclue seulement les nombres réels codés sur 24 bits. Le codage n'est pas IEEE754 mais il existe des fonctions de conversion vers le standard (`float24ToIEEE754`).
Pour utiliser les nombres réels il est nécessaire d'inclure le fichier math24f.h.
Le programme tstfloat.c effectue la multiplication et la division de deux nombres réels, l'affichage des résultats dans MPLAB nécessite la conversion IEEE754.

```
// tests sur nombres réels sur PIC12
#pragma chip PIC12F675
#include <math24f.h> // attention le format des reels n'est pas IEEE
float a,b,c,d;

void main(void)
{
a=2.0;            // a et b ne sont pas visibles dans MPLAB car non IEEE
b=3.0;
c=a*b;
float24ToIEEE754(c);  // c est maintenant au format IEEE et visualisable
d=a/b;
float24ToIEEE754(d);
while(1);
}
```

a et b ne sont pas convertis au standard IEEE754 ne sont donc pas visibles.

Pour voir c et d (6.0 et 0.66) cliquer à droite puis sur propriétés et valider IEEE 24 bits



6. Exemple de fonctions mathématiques

Le programme tstfloat2.C montre la mise en œuvre des fonctions mathématiques (ici le calcul d'une racine carrée). Il est nécessaire d'inclure le fichier math24lb.h

```
// tests de fonctions mathématiques avec nombres réels sir PIC12
#pragma chip PIC12F675
#include <math24f.h> // attention le format des reels n'est pas IEEE
#include <math24lb.h> // librairie pour fonctions math
float a,c;

void main(void)
{
a=2.0;                  // a n'est pas visibles dans MPLAB car non IEEE
c=sqrt(a);              // calcul la racine carree
float24ToIEEE754(c);    // c est maintenant au format IEEE et visualisable
while(1);
}
```

7. Exemple de gestion du convertisseur analogique numerique

```
//canC5.C test du CAN sur PIC12F675 et PICKIT1
// La tension issue du potentiomètre RP1 est visualisée sur les LED
#pragma chip PIC12F675
const unsigned char sortie[]=
{0b00010000,0b00100000,0b00010000,0b00000100,0b00100000,0b00000100,0b000001
00,0b00000010};
const unsigned char sens[]=
{0b11001111,0b11001111,0b11101011,0b11101011,0b11011011,0b11011011,0b111110
01,0b11111001};
unsigned long i;

void main(void)
{
      GPIO=0;
      CMCON=7;          // fonction comparateur desactivee
      while(1)
            {
                  ANSEL=1;                // AN0 analogique AN1 - AN3
                                          entrees numeriques
                  TRISIO |= 0x01;         // ANo en entree
                  ADCON0=0b00000011;      // active CAN et SOC
                  while (GO);             // attend ECO
                  i=ADRESH>>5;            // mise à l'echelle pour LED
                  ANSEL=0;                // toutes les entrees numeriques
                  TRISIO=sens[i];         // allume LED numéro i
                  GPIO=sortie[i];
            }
}
```