CH-1092 Belmont, Switzerland
info@didel.com, www.didel.com

PIC*Génial* - Microcontrôleur PIC 16F84/16F870

Introduction à la programmation avec le PIC*Génial* - suite

9. Structure des programmes et fichiers de référence

Un programme pour PIC commence par l'annonce du processeur. Les différences sont mineures, et le 16F84 convient pour tous les processeurs 14 bits. Le 12C50x convient pour tous les processeurs 12 bits.

Ce qui différencie les PICs, ce sont les registres et les bits. Le fichier 16Fxxx.asi contient toutes les définitions, selon la documentation du fabricant. Ce fichier ne comportant que des assignations de symboles, il est plus rapide d'importer le fichier table de symboles .ref, plutôt que d'insérer le fichier source .asi. On écrit .Ref fichier, l'extension par défaut est .ref.

Il est très utiles de définir quelques macros, pour vérifier en particulier les passages de page, la commutation de banque et la taille des variables et du programme. Les macros doivent être déclarées avant d'être utilisées. Le fichier avec le nom du processeur, le postfixm M et l'extension .asi sera inséré au début.

Un programme a donc les éléments suivants:

```
    .Proc   16F84
    .Ref    16F84      ; ou .Ins   16F84.asi
    .Ins    16F84M.asi
...
    TooLong?
    .End
```

Pour éviter de surcharger les listages avec des éléments bien connus, on peut parenthéser les modules qui ne doivent pas être imprimés avec un .List Cond ... .EndList. Si Cond=0 les lignes n'apparaissent pas dans le fichier de listage.

Les librairies de routines sont formées de trois éléments: les définitions, les variables et les routines elles-mêmes, indépendantes du processeur et des assignations d'entrées-sortie, autant que possible. Un ou plusieurs programmes de test doit être fourni avec chaque librairie. Les routines ont le postfixe R avec l'extension .asi. Les définitions, variables et initialisations sont mises dans un fichier avec le postfixe I et l'extension .asi. Ce fichier est inséré au bon endroit, ou ses éléments sont distribués dans d'autre fichiers insérés et dans le programme principal.

Exemple

Pour communiquer en série, il faut les routines SndSer RecSer, et il est utile de pouvoir transférer de nombres 8 bits en hexa avec les routines SndHex et RecHex. La documentation des routines est

```
Routine SndSer.asi  Envoi série monotâche
  in:   W
  mod: DataSnd
Routine RecSer      Réception série à 9600 b/s monotâche.
  out:  W = DataRec
```

Pour le 16F84, le fichier de routines XserR.asi doit être associé aus fichier de variables XserV.asi de macros XserM.asi; il faut définir sur quel bit de quel port se trouvent TxD et RxD (le même port pour TxD et RxD). Le logiciel fonctionne à 9600 bits/s avec un quartz ou un résonateur à 4 MHz. Un oscillateur RC peut être ajusté avec un oscilloscope (remplacer la résistance 4k7 par une 5.6k et mettre une résistance de 200-1M en parallèle et observer la pin 15 qui doit osciller à 1 MHz +/- 1%), mais la fréquence est sensible à la tension.

Avec le 16F870, le logiciel est plus simple, puisqu'il y a un interface câblé, qui oblige d'utiliser les lignes RC6 et RC7 (connecteur J7 de **PIC*Génial*/870**). Les fichiers sont

Xser7.asm, Xser7V.asi et Xser7R.asi. Le programme de test de ces routines est Xecho.asm. Ce qui est envoyé par le terminal est simplement renvoyé au terminal.

Program **XEcho7** Echo série sur F870

```
.Proc 16F870
.Ref  16F870
.List 0
.Ins  16F870M.asi
.Endlist
```

Constant **PortA et B**

```
DirA   = -1       ; entrées
DirB   = 0        ; sorties
```

Constant **PortC**

```
bTxD           = 6
bRxD  = 7
DirC  = 2'10011000
IniRcSta       = 2'10011000
IniTxSta       = 2'00100100
IniSpBrg       = 10'25
PortSer        = 7          ; id
Ascii
CR    = 16'0D
LF    = 16'0A
BEL   = 16'07
```

```
.APC  Var
.Loc  DebVar
.Ins  XSer7V.asi
.APC  Code
.Loc  0
```

Program **Programme**

```
Start:
      Move        #DirA,W
      Move        W,TrisA
      Move        #DirB,W
      Move        W,TrisB
      Move        #DirC,W
      Move        W,TrisC
      Call        IniSer
; On attends le caractère, on l'affiche sur PortB et on le r
Loop:
      Call        RecSer
      Move        W,PortB
      Call        SndSer
      Jump        Loop
.Ins  XSer7R.asi
.End
```

On pourrait encore augmenter la compacité en créant un fichier avec les définitions. C'est ce que l'on fait quand tous les éléments d'une application ont été testés, et que le câblage et les définitions ne vont plus changer.

Si on veut tester une routine qui envoie l'alphabet, il faut créer le fichier des variables supplémentaires et le fichier contenant la nouvelle routine.

Module **XAlphaV.asi** Pour XalphaR.asi

```
.APC  Var
Alpha: .Blk.16    1          ; Pointeur dans la chaîn
CAlpha:           .Blk.16    1          ; Longueur de
.End
```

Fichier XalphaR.asi Variables dans XalphaV.asi

Routine **SndAlpha** Affiche l'alphabet

```
  in:  -
  mod: Alpha Calpha
SndAlpha:       Move        #"A",W
      Move      W,Alpha
      Move      #26,W
      Move      W,Calpha
TA$:
      Move      Alpha,W
      Inc       Alpha
      Call      SndSer
      DecSkip,EQ Calpha
      Jump      TA$
      Ret
.End
```

Les parties du programme Xecho.asm qui sont motifiées pour créer le programme Xalpha.asm sont les suivantes:

Program **XAlpha7.asm** Envoie l'alphabet

```
....
.APC  Var
.Ins  Xser7V.asi
.Ins  XAlphaV.asi
....
```

```
Loop:
      Call        RecSer
      Call        SndCR
      Call        SndAlpha
      Jump        Loop
.List 0
.Ins  Xser7R.asi
.Ins  XalphaR.asi
.EndList
```

Si maintenant on veut tester une routine qui génère des nombres aléatoire, on écrit pour la boucle principale

Program **XAlea** Test routine aléa entre 1 et max

```
; Affichage série vest terminal
ValMax = 9        ; affichera des valeurs entre 1 et 9
Debut:
      Move        #DirC,W
      Move        W,TrisC
      Call        IniSer
      Move        #ValMax,W
      Move        W,Max
```

```
Loop:
      Move        #Max,W
      Call        Alea
      Call        SndHex
      Call        SndSpace
      Move        #25,W          ; 0.5s
      Call        LongDelai
      Jump        Loop
```

Il faut insérer au bon endroit les fichiers de variables XaleaV.asi et XdelaiV.asi et les routines de librairie XaleaR et XdelaiR. Si la routine "alea" n'est pas satisfaisante, on la remplace dans son fichier, et elle sera appliquée à tous les fichiers qui l'importent.

9.0.1. Fichiers de référence Pic16F84/16F870

L'assembleur connaît les noms des registres réservés, mais pas les noms des bits dans ces registres. Le fichier Pic16F84.asi, donné en détail dans la 2e partie, contient les numéros des bits correspondant à la documentation du fabricant. Le fichier Pic16F84.ref est compactifié pour accélérer l'assemblage. Les définitions ne sont alors plus visibles, et si le message " ^ symbole defini deux fois" apparaît, cela peut vouloir dire que vous avez choisi un symbole déjà défini dans Pic16F84.ref. Ces symboles sont les suivants:
C, D, DC, Z, PD, TO, RP0, RP1, IRP, PS0, PS1, PS2, PSA, RTE, RTS, INTEDG, RBPU, RBIF, INTF, TOIF, RBIE, INTE, TOIE, EEIE, GIE, RD, WR, WREN, WRERR, EEIF

9.1. Fichier de référence du Pic 16F84

Les noms des registres d'un processeur sont connus de l'assembleur. Par contre, les noms des bits dans ces registres doivent être définis par l'utilisateur. Le fabricant a définit les noms de ces bits dans sa documentation; il n'y a pas de raison de changer. Le fichier Pic16F84.ref contient les définitions suivantes, créées par le fichier Pic16F84.asi. Le fichier .ref est plus compact et ne laisse pas de trace dans le listage. On a le choix entre écrire

.Ins Pic16F84.asi ou, ce qui est préférable, car plus rapide à l'exécution

.Ref Pic16F84

Ce fichier est donné ci-dessous. Dans les programmes simples, on tape au début les lignes correaspondant aux quelques bit utilisés, comme on l'a fait par exemple dans le programme RS232 Picgs3, qui ne déclare que le bit TOIF.

```
Constant Pic16F84 Définitions des bits des registres
; F ou Status (adr 3 / 83) Rz -> 0
     C          = 0
     D          = 1
     Z          = 2
     PD         = 3
     TO         = 4
     RP0        = 5          ; Page select
     RP1        = 6
     IRP        = 7
; Option (adr 16'1 / 16'81) Rz -> 1 partout
     PS0        = 0          ; 000 divide per 2 on
     PS1        = 1
     PS2        = 2
     PSA        = 3
     RTE        = 4
     RTS        = 5
     IntEdg     = 6
     RBPU       = 7          ; = 0 pull-up actives

; IntCon (adr 16'B / 16'8B) Rz -> 0
     RBIF       = 0
     INTF       = 1
     TOIF       = 2
     RBIE       = 3
     INTE       = 4
     TOIE       = 5
     EEIE       = 6
     GIE        = 7
; EECon1 (adr 16'8 / 16'88) Rz -> 0
     RD         = 0
     WR         = 1
     WREN       = 2
     WRERR      = 3
     EEIF       = 4
```

9.2. Fichiers de macros du PIC 16F84

Il est pratique de définir des macros pour changer de banque, et éventuellement pour des macroinstructions de comparaisons. Ce fichier doit êre inséere avant le programme. Les macroinstructions qui ne sont pas appelées ne génèrent pas d'instructions; elles rallongent à peine le temps d'assemblage. On a donc avantage à définir un ensemble de macros bien documentées dans un ou plusieurs fichiers. Le fichier 16F84M.asi ne contient que les macros suivantes

Bank1
Bank0
TooLong?
SetPage
exemple pour tester si une page déborde

9.3. Fichier de référence du 16F870

Le fichier de référence 16F870.asi et 16F870.ref est beaucoup plus complet, car le 16F870 a des périphériques supplémentaires. Pour le F873, F877, il faut encore ajouter des définitions.

Le fichier de macros 16F870M.asi a tout un jeu de macros Bankitoj en plus de macros Bank0,1,2,3 qui obligent à deux instructions chaque fois, puisque l'on ne sait pas d'où l'on vient

9.4. Autres processeurs PIC

-- Documentation à terminer

9.4.1. 16F627-628

Nouveau PIC avec flash comme le 16F84, programmable en 5V, avec possibilité d'oscillateur et de clear interne, avec canaux série, PWM, analogiques (comparateurs).

9.4.2. 12C508..CE619

8 pattes (6 I/O). Une feuille de codage spécifique existe (3 instructions en moins).

9.4.3. 16C505

14 pattes (12 I/O)

9.4.4. 12C671..673

8 pattes, 4 entrées analogiques.

9.5. Compatibilité 16F84 - 12C508

voir le document www.didel.com/doc/DopiComp.typo

9.6. Annexe: Feuille de codage du PIC 16F84/16C84 (verte)

Fichiers www.didel.com/doc/Pic84Calm.pdf www.didel.com/doc/Pic50xCalm.pdf