J.D. Nicoud, Mouette 5
CH-1092 Belmont, Switzerland
Tel +41 21 728-6156, Fax 728-6157
Email info@didel.com, www.didel.com

DIDEL

Fichier: www.didel.com/doc/DopicBug.pdf

1. Déverminage de programmes PIC sans émulateur

Le déverminage d´un programme peut être assez pénible, avec ou sans système de développement coûteux. La premiière recommandation est de bien structurer son programme avec des routines réutilisables, et de tester chaque routine avec un programme de test spécifique. La 2e recommandation est de relire attentivement un nouveau module de programme en vérifiant les #, les 16´ 2´, les compteurs de boucle, les noms et fonctions des variables, les paramètres en entrées et sorties des routines, les registres modifiés.

1.1. Deux types de programmes et trois techniques

On dévermine un programme en repérant par où il passe, et en observant l´état des variables. Si on est en temps réel, il faut insérer un nombre minimal d´instructions pour ne pas ralentir. Si on teste un algorithme, on peut arrêter l´exécution du programme en test et appeler un module qui affiche et éventuellement modifie des variables avant de continuer l´exécution. Mais ce qui s´exécute dans le PIC ne doit pas prendre trop de place, car il n´y a que 1 ou 2 k de programme.

Pour repérer par où le programme passe sans ralentir de façon significative son exécution du programme, on peut générer des impulsions sur une ligne de libre ou afficher une valeur sur un port commandant des Leds. Si l´on n´a pas de contraintes temps réel, on peut générer des sons avec un haut-parleur ou faire clignoter des Leds. Le plus efficace est d´insérer des points d´arrêt, avec 2 solutions: affichage LED et poussoir pour continuer l´exécution, ou liaison avec un PC exécutant un Hyperterminal ou un programme ad-hoc. Un émulateur comme il existe avec le 16F877 ou le Scenix/Ubicom, intervient à un plus bas niveau dans le processeur, et permet de stopper le processeur et examiner les registres sans ajouter du code dans la mémoire. Il faut un processeur spécial avec des fonctions internes supplémentaires et une carte de liaison avec le PC, d´où son prix élevé.

1.2. Marques impulsionnelles

Si on développe avec l´oscilloscope un programme en temps réel, il faut disposer d´au moins deux lignes libres en sortie du processeur, appelées S0 et S1. L´une sert à synchroniser l´oscilloscope. L´autre permet de marquer des instants. Appelons bS0 et bS1 les bits correspondants sur le portA. Les macros S0On, S0Off, S1On S1Off, vues plus loin activent et désactivent ces lignes.

Pour repérer l´instant d´échantillonnage dans une routine série, on insérera les instructions tramées. Le câblage est donné dans la figure 1, avec copie de ce que l´on observe à l´oscilloscope.

```
Routine RecSer  Réception série à 9600 .. 2400 b/s
; On a >1 stop bit (>100 us) pour traiter l´info
  out:   RecShiDa = W
  mod:  CRec CAtt
RecSer:
 On attend le Start bit
  A$: SkipIfStopBit
      Jump        A$
  B$: SkipIfStartBit
      Jump        B$
s0on
s0off
Lecture des data
      Move        #8,W
      Move        W,CRec
 Attente 1-1/2 période
      Move        #(3*RateAdjust)/2,W
 L$:
      Move        W,CAtt
 W$:  Dec         CAtt
      Skip,EQ
      Jump        W$
      SetC
s1on
      SkipIfRxOne
s1on
      ClrC
      RRC         RecShiDa        ; LSB first
 Attente 1 période
      Move        #RateAdjust,W
      DecSkip,EQ  CRec
      Jump        L$
Le stop bit doit encore passer
s1on  ; pour vérifier la durée stop bit
; W a été initialisé dans la dernière boucle
      Move        W,CAtt          ; Pour arriver au stop bi
 VV$: Dec         CAtt
      Skip,EQ
      Jump        VV$
s1off
; On peut travailler pendant le stop bit
      Move        RecShiDa,W
      Ret
```

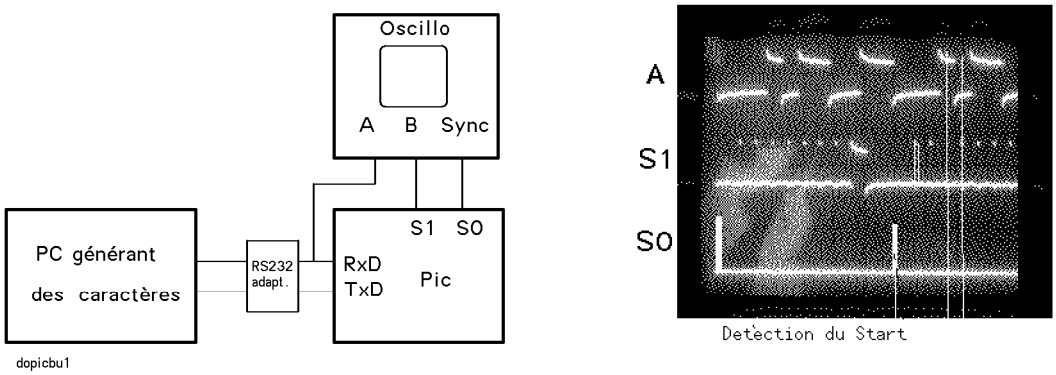

Fig. 1 Câblage de l´oscillo et image typiques (caractère reçu "4" = 16´34)

Sans oscilloscope à mémoire, le maintien de la touche génère une fréquence de répétition adéquate pour l´observation. L´oscilloscope est synchronisé avec S0 ou avec le start bit avec un réglage du "hold off" supérieur à 1ms..

Pour dépanner un programme de réception infrarouge, ou simplement vérifier qu´il fait bien ce que l´on veut et que les marges de sécurité sont symétriques, cette technique est la seule.

On peut la perfectionner en générant des impulsions multiples, et pour éviter de devoir taper des longues séquences de S1On S1Off, une macro que l´on peut appeler Mark, suivie d´un paramètre qui est le nombre d´impulsions, simplifie l´écriture. Cette macro est donnée en annexe avec un programme de test. Si on n´a pas d´oscilloscope, on peut compter le nombre d´impulsions avec un autre PIC qui a un affichage 8 bits, et avoir un espion qui dit combien d´impulsions successives ont été comptées (en annexe).

1.3. Macros conditionnelles

Après la mise au point, on doit enlever toutes les marques, faciles à repérer si on les a décalées, mais souvent on croit que c´est au point et il y a encore des fautes cachées, ou on veut vérifier les timings avec ou sans marques. Les macros conditionnelles utilisent un paramètre, appelé ici Debug. Les instructions ne sont générées que si ce paramètre est différent de zéro. Si le paramètre est zéro, il n´y a tout simplement pas d´instructions générées. La macro S1On s´écrit:

```
.Macro    S1On
  .If     Debug
          Set       PortA:#bS1
  .Endif
.Endmacro
```

A noter encore que les macros SxOn/off ne sont pas utilisables si des lignes du même port sont en collecteur ouvert (I2C par exemple), ou si des sorties sont trop chargées (un moteur demandant trop de courant est directement sur une sortie). Il faut dans ce cas entretenir une copie de l´état du port en mémoire, et toutes les fois que cette copie est modifiée, il faut remettre à jour le port. Ceci permet d´effectuer les Set/Clr bits sur la copie en mémoire, mais évidemment cela coûte trois instructions pour toutes les écritures sur le port, à travers tout le programme, et s´il y a des interruptions qui utilisent le même port, cela peut être la catastrophe. Donnons quand même une macro en exemple:

```
.Macro    S1On
          Set       CopiePortA:#bS1
          Move      CopiePortA,W
          Move      W,PortA
.Endmacro.
```

Une autre solution dans le cas de l´I2C par exemple, est de changer de banque, et remettre la bonne banque au retour. La macro qui tient compte de la banque et ne modifie pas les autres bits du registre n´est pas évidente. Il faut tester dans quelle banque on est pour y revenir, et ne pas lire le port, car on lit les entrées et sorties en collecteur ouvert sur le chip, et non pas l´état interne qui ne doit surtout pas être modifié dans le cas du

collecteur ouvert. L'état du registre interne doit être connu. S'il n'est pas toujours le même, il faut en garder une copie en mémoire, et travailler avec cette copie. Les macros S1On S1Off qui utilisent InitA comme valeur du port s'écrivent:

```
.MacroS1Off
.Localmacro  A
      TestSkip,BC   F:#RP0
      Jump          A
      Move          #IniA+2**bS1,W   ; garder bits 0 et
      Move          W,PortA
      Jump          B
 A:   Clr           F:#RP0
      Move          #IniA+2**bS1,W   ; garder bits 0 et
      Move          W,PortA
      Set           F:#RP0
 B:
.Endmacro
```

```
.MacroS1Off
.Localmacro  A
      TestSkip,BC   F:#RP0
      Jump          A
      Move          #IniA,W    ; garder bits 0 et 1 à zéro
      Move          W,PortA
      Jump          B
 A:   Clr           F:#RP0
      Move          #IniA,W    ; garder bits 0 et 1 à zéro
      Move          W,PortA
      Set           F:#RP0
 B:
.Endmacro
```

1.4. Affichage sur un port

Si on dispose d'un port avec un affichage non utilisé par le programme, cela ne coûte qu'une ou deux instructions pour afficher l'état d'une variable sur cet affichage. Avec le 16F84 qui n'a qu'un port 8 bit complet, il faut le réserver pour cet affichage, et utiliser les lignes du port A pour câbler les entrées et sorties d'un capteur ou moteur à gérer. Cela sera facile de les déplacer quand les routines seront au point.

Si l'on veut afficher l'état d'une variable à un instant donné, il suffit d'écrire les deux instructions

```
      Move      Var,W
      Move      W,PortB
```

Si le registre W ne doit pas être modifié, on rajoute deux instructions:

```
      Move      W,SavW
      Move      Var,W
      Move      W,PortB
      Move      SavW,W
```

Le registre SavW doit naturellement avoir été déclaré avec les autres variables. Il y a encore le flag EQ/NE qui peut être modifié par ces instructions, et on ne mettra pas la macro qui génère ces 4 instructions entre un test et le skip conditionnel associé.

La macro (appelée ici V) ne doit pas toujours afficher la même variable; le nom de la variable sera le paramètre de la macro et on écrira V Var pour afficher la variable Var. Dans une macro, %1 désigne le premier paramètre, %2 le 2e, etc. (jusqu'à 8). La macro V est:

```
.Macro    V                ; Appel: V nom de variable
          Move      W,SavW
          Move      %1,W
          Move      W,PortB
          Move      SavW,W
.Endmacro
```

La mise au point se fait en déplaçant la macro V et en affichant la même ou d'autres variables. Il ne peut y avoir qu'une macro V par programme, et faut chaque fois assembler et charger le programme, on a donc avantage à tester des petits modules.

1.5. Attente sur un interrupteur

En attribuant une ligne du PortA à un poussoir (bit bSw), avec B toujours réservé pour afficher une valeur 8 bits, on peut avancer dans le programme en pas à pas à chaque action du poussoir et afficher sur le port un numéro d'ordre ou une variable.

L'astuce est d'afficher un numéro repère quand on agit sur le poussoir, et la valeur de la variable quand on relâche. Il n'y a plus de temps réel, mais c'est simple et prend peu de place en mémoire. Plusieurs marques peuvent être introduites et on peut afficher plusieurs variables consécutives à un endroit critique. La macro, appelée M, a 2 paramètre, le numéro d'ordre et la variable.

```
.Macro    M                  ; Appel: M #numéro,nom de variable
.Localmacro A,B
          Move      W,SavW
 A:       Call      AttReb
          TestSkip,BC   PortA:#bSw      ; passe à zéro quand on presse
          Jump      A
          Move      %1,W
          Move      W,PortB
 B:       Call      AttReb
          TestSkip,BS   PortA:#bSw      ; attend qu'on relâche
          Jump      B
          Move      %2,W
          Move      W,PortB
          Move      SavW,W
.Endmacro
```

La routine AttReb est nécessaire à cause des rebonds de contact: il faut attendre quelques millisecondes entre deux échantillonnages. Pour ne pas utiliser de variable, cette routine s'écrit:

```
AttReb:             ; Durée 256*12us = 3 ms
          Clr       W
  W$:     Jump      APC+1     ; 2 us
          Jump      APC+1     ; 2 us
          Jump      APC+1     ; 2 us
          Jump      APC+1     ; 2 us
          Add       #-1,W     ; 1 us
          Skip,EQ             ; 1 us
          Jump      W$        ; 2 us
          Ret
```

Pour éviter de rajouter ces définitions de macros et routines dans chaque programme à tester, il y a avantage à en faire des fichiers librairies, et insérer ces fichiers. Malheureusement, les macros doivent être insérées au début, avant toute utilisation et les routines après le début du programme, en général à la fin, ce qui oblige à préparer deux fichiers. Définissons donc un fichier XDebugM.asi (M pour macros) et un fichier XDebugR.asi (R pour routines) dans lesquels nous pourrons ajouter nos macros et routines de déverminage, sans trop nous préoccuper si elles sont toutes utilisées dans un programme donné. Les routines prennent de la place en mémoire, mais pas les macros, qui ne génèrent du code que lorsqu'elles sont appelées.

Le programme de test de ces macros, routines et façons de structurer le programme est le suivant. Le programme incrémente la variable Compte et fait décaler un "1" dans la variable Decale (en exécutant ce programme, on voit qu'il y a un bug à corriger: le 1 devrait réapparaître après un tour).

Program **XDebug.asm** **Test macros DebutM.asi et routines DebugR.asi**

```
.Proc 16F84
Ports
bSw   = 4         ; switch sur PB4, actif à zéro
DirA  = 2'10000
DirB  = 0         ; sorties
Variables
.Loc  16'0C
Compte:           .Blk.16     1
Decale:           .Blk.16     1
SavW: .Blk.16     1
.Ins  XDebugM.asi
.Loc  0
Initialisation
          Move      #DirA,W
          Move      W,TrisA
          Move      #DirB,W
          Move      W,TrisB
          Clr       Compte
          Move      #2'10000000,W
          Move      W,Decale
Boucle
          Inc       Compte
M #1,Compte
          RRC       Decale
M #2,Decale
          Jump      Loop
.Ins  XDebugR.asi
.End
```

A noter que le numéro de marque doit être précédé d'un #. Si le # n'est pas mis, c'est une variable qui sera sélectionnée. On pourrait donc dans ce programme de test remplacer les 2 macros M par une seule:

```
M  Compte,Decale
```

et voir le contenu de Compte quand on presse, et celui de Decale quand on relâche.

1.6. Envoi sur le port série

La solution la plus flexible si l'on n'a pas de sévère contraintes de place et de temps est de communiquer en série avec un PC chargé avec un programme émulateur de terminal (Hyperteminal par exemple). Il suffit de disposer des macros et routines suivantes pour bien aider à la mise au point.

```
Macros :  (voir  www.didel.com/doc/XBugSerM.asi)
          V  Var    Affiche [Var] (en hexadécimal: deux quatrets)
          W  #N,Var          Affiche[NN]Var\space;, et attend une frappe du clavier

Routines:  (voir  www.didel.com/doc/XTrSerR.asi)
          SndSer    Envoie en série le contenu de W
          SndHex    Envoie en hexa Ascii le contenu de W
          RecSer    Attend un caractère du clavier (résultat dans W)
```

Il faur encore insérer les définitions, variables et macros. Les routines série sont expliquées dans le document

www.didel.com/doc/dopicser.pdef, qui donne différentes variantes selon le processeur et la vitesse.

La structure du programme de test est tellement semblable au programe précédent qu'il n'est pas nécessaire de répéter. Il se trouve sous www.didel.com/doc/XBugSer.asm dans le cas d'un 16F84 avec quartz 4 MHz, avec Tx et Rx câblés sur RA0 et RA1.

2. Annexe

La macro XmarkM.asi (www.didel.com/doc/XmarkM.asi) est assez astucieusement écrite pour générer un train de 1 à 8 impulsions.

Macro XMarkM.asi

```
; bMark          = 3          ; portA
; valeur = nbre d'impulsions de 1 us toutes les 2 us
.MacroMark       ; Appel par Mark valeur
.If  %1.HS.1       ; HS = Higher or Same
      Set        PortA:#bMark
      Clr        PortA:#bMark
 .If  %1.HS.2
      Nop
      Set        PortA:#bMark
      Clr        PortA:#bMark
  .If  %1.HS.3
      Nop
      Set        PortA:#bMark
      Clr        PortA:#bMark
   .If  %1.HS.4
      Nop
      Set        PortA:#bMark
      Clr        PortA:#bMark
    .If  %1.HS.5
     Nop
     Set         PortA:#bMark
     Clr         PortA:#bMark
     .If  %1.HS.6
     Nop
     Set         PortA:#bMark
     Clr         PortA:#bMark
      .If  %1.HS.7
     Nop
     Set         PortA:#bMark
     Clr         PortA:#bMark
       .If  %1.HS.8
     Nop
     Set         PortA:#bMark
     Clr         PortA:#bMark
        .Endif
      .Endif
     .Endif
    .Endif
   .Endif
  .Endif
 .Endif
.Endmacro
```

Le programme de test est le suivant (www.didel.com/doc/Xmark.asm)

Program **XMark.asm** **Envoi marques sur PortA:#0**

```
; Répétition toutes les 100 us
.Proc 16f84
```

Constant **Ports** **PortA**

```
bS0   = 2        ; synchro
bMark = 3        ; Y sur silo
DirA  = 2'00000
.Ins  XmarkM.asi ; macros
.Loc  0
Deb:
      Move       #DirA,W              ; sorties
      Move       W,TrisA
Loop:
      Set        PortA:#bS0
Mark 1
      Clr        PortA:#bS0
      Call       Wait
Mark 8
      Call       Wait
Mark 3
      Call       Wait
      Jump       Loop
Wait:  ; Attente 100 us
      Move       #20,W
 W$:  Add        #-1,W
      Skip,EQ
      Jump       W$
      Ret
```

Le programme qui compte les marques avec un PIC 16F84 et affiche sur le portB utilise le compteur/timer (www.didel.com/doc/XAfMark.asm)

Program **XAfMarks** 200101

```
; Affiche le nombre d´impulsions reçues sur le portB
; Un silence de 30 us doit séparer l´émission
; de deux trains d´impulsions.
.Proc 16F84
TOIF  = 2
bSw   = 4
bIn   = 3       ; Entrée marque
DirA  = 2´10000
DirB  = 0       ; Affichage marque
.Loc  16´C
SavMark:        .Blk.16     1
.Loc  0
Deb:
      Move      #DirA,W
      Move      W,TrisA
      Move      #DirB,W
      Move      W,TrisB
      Move      #2´00101000,W       ; T0Ck, no p
      Move      W,Option
      Clr       PortB
Loop:
      Clr       TMR0
   V$:Move      TMR0,W
      Skip,NE
      Jump      V$
;Les marques arrivent en 8x2us max (macro XMarkM.asi)
      Move      #3,W
   W$:Add       #-1,W
      Skip,NE
      Jump      W$
      Dec       TMR0,W
      Add       TMR0,W        ; x 2
      Add       W,PCL
      Nop       ; Pas sensé arriver ici
      Set       PortB:#0
      Jump      Z$
      Set       PortB:#1
      Jump      Z$
      Set       PortB:#2
      Jump      Z$
      Set       PortB:#3
      Jump      Z$
      Set       PortB:#4
      Jump      Z$
      Set       PortB:#5
      Jump      Z$
      Set       PortB:#6
      Jump      Z$
      Set       PortB:#7
      Jump      Z$

Z$:   Jump      Loop
.End
```