http://www.didel.com/ **info@didel.com** http://www.bricobot.ch/ **info@bricobot.ch**

Programmes de test pour Pics

-- en travail – juillet 2010

Pour apprendre à maîtriser les entrées-sorties d'un microcontrôleur, le plus efficace est d'écrire un programme de test qui vérifie que l'on a bien compris la documentation du fabricant. Cette documentation décrit tous les cas possibles, mais ce que l'on veut faire est en général simple.
Les programmes Txxyy.asm poursuivent cet objectif et ont été écrit pour quelques processeurs. Adapter le programme de test a l'application est facile et permet aussi de vérifier que le câblage est bien traduit dans le logiciel. Ces programmes de test sont la base pour s'assurer que l'on a bien compris la documentation: on complète le programme de test et on vérifie le comportement. De plus, ces programmes de test permettent de reprendre confiance quand dans la mise au point d'un programme tout à coup plus rien ne va. Exécuter le programme de test montre que le processeur et le câblage est bon.

Il faut savoir que les PICs se regroupent par famille et que dans une famille, la gestion des périphériques est identique si elle est implémentée.

Instructions 12 bits (33 instructions)
10F200 - 202 10F204 - 206 10F220 – 222
12F508 – 509 12F505
Instructions 14 bits (35 instructions)
12F629 – **16F630** 12F6xx – 16F676 www.didel.com/pic/PicT630.pdf
16F690 www.didel.com/pic/PicT690.pdf
16F84 16F88
16F628 16F627 www.didel.com/pic/PicT628.pdf
16F870 - ... – **877** www.didel.com/pic/PicT877.pdf
16F882 – 884 www.didel.com/pic/PicT882.pdf
16F946
Les processeurs en gras disposent d'un fichier qui liste et parfois commente les programmes de test à disposition. Ces programmes sont accessibles dans un Zip.
A noter que si les numéros des processeurs ne diffèrent que par la taille mémoire (10F200 et 202) le test n'a pas été dédoublé et n'existe que pour le processeur apparamment le plus utilisé.

Programmes de test existant
Le premier but des fichiers de test est de montrer comment initialiser les ressources internes du processeur, dans des cas simples: lire un canal analogique, envoyer un caractère en série, etc. Ceci aide à choisir dans la doc ce qui est essentiel, et à tester progressivement des fonctionnalités plus complexes.
Le second but est de tester les programmes de la librairie X www.didel.com/pic/LibX.pdf, mais avec seulement un ou deux processeurs. Les fichiers de la LibX sont en principe indépendant du processeur.

Les sources des programmes de test sont dans des dossiers compressés (zip) appelés **Txx.zip**. Par exemple **www.didel.com/pic/T630.zip**
Ces programmes ont été sauvés depuis SmileNG et devraient apparaître correctement s'ils sont lus depuis SmileNG. Le problème des CR-LF et conversion automatique peut faire disparaître les fins de ligne ou les dédoubler. Le copier-collé depuis Word pose souvent problème. Les CR et Tabs ne réagissent pas de la même façon dans les tables. Merci Microsoft!
Attention aux minuscules et majuscules dans les noms de fichier. Merci Unix et JDN! Par contre Calm identifie les minuscules et majuscules, et SmileNG permet de jolies mises en page.

Description générale des programmes de test

Pour chaque processeur, on trouve les programmes de test décrits ci-dessous. Des explications spécifiques sont données dans les fichier de chaque processeur.

Actions sur les entrées sorties

Txx.asm Clignote les sorties avec une période de 0.6sec (a 4 Mhz).
Ce programme montre comment initialiser toute les broches en sortie. Il faut en général désactiver des convertisseurs et sorties multiplexées. Certaines broches sont parfois en entrée seulement, ou en sortie collecteur ouvert, et/ou dépendent de la configuration.
Pour les processeurs avec oscillateur interne, c'est cet oscillateur qui est utilisé, à 4 MHz.

TxxAtoC.asm Copie un port sur un autre port (ici A versC). Certains bit ont plusieurs fonctionnalités, et il peut être utile de vérifier que l'on a configuré correctement le port.

TxxSw.asm Lit un poussoir et supprime les rebonds. A chaque pression, on compte sur un autre port,

TxxSwInter.asm L'activation du poussoir crée une interruption. Les processeurs anciens n'ont qu'une entrée d'interruption, on préfère disposer de registres de configuration pour choisir la provenance de l'interruption.

TxxAna.asm Lit une entrée analogique

Les processeurs qui ont des entrées analogiques en ont plusieurs qui sont multiplexées. Tous les PICs ont la même circuiterie pour le convertisseur, mais le choix des canaux et l'emplacement des registres change.

A l'initialisation il faut :

- activer le convertisseur (bit *AdOn* dans *AdCon0*)
- dire quelles sont les broches utilisées en entrées analogiques. Il faut en général activer des bits dans un registre appelé *AdCon0*, mais avec les 16F87x, seulement un certain nombre de combinaisons sont offertes.

Lorsqu'une lecture est souhaitée, il faut :

- Sélectionner la broche en donnant son adresse dans un registre *AdCon1* et en précisant en même temps comment on veut le résultat dans les 2 mots AdRes (voir plus loin)
- Attendre 20 microsecondes (minimum selon la fréquence d'horloge, valeur dans un tableau) pour que l'aiguillage interne se positionne
- Demander au convertisseur de démarrer en activant le bit *Go/Done* dans le registre *AdCon0*
- Attendre que le bit *Go/Done* passe à un (environ 100 microsecondes à 4 Mhz). Si l'interruption a été initialisée, il y a saut à l'adresse 4.
- Lire la valeur convertie dans *AdResH* et *AdResL*.

Un bit *ADFM* usuellement dans *AdCon1* permet d'aiguiller le résultat 10 ou 12 bits vers les registre *AdResH* et *AdResL*. Dans les cas simples où on se contente de 8 bits de précision, on demande d'avoir les 8 bits de poids fort dans le registre *AdresH*. Les 2 bits restants sont dans les poids fort de l'autre registre que l'on oublie. Si on veut utiliser toute la précision avec des calculs ultérieurs sur 16 bits, on demande une copie dans le registre 16 bits *AdResH – AdResL.*

Le programme de test initialise le 1er cas et affiche *AdResH.* Si on branche un potentiomètre sur l'entrée AN0 on doit voir la valeur passer de 0 à 16'FF.

TxxT0a.asm Copie le timer 0 sur un port

Tous les processeurs PİC des familles 10, 12 et 16 ont le même timer TMR0 très primitif. C'est un compteur qui tourne sans cesse avec une horloge qui est prédivisée par 1, 2, 4, .. 256 selon des bits du registre *Option.* Le test copie *TMR0* sur un port avec un affichage 8 bits, et on voit les bits de poids fort clignoter /les poids faibles clignotent trop vite).

Nota bene: si vous voulez que l'on vous explique tous les bits du registre *Option*, lisez le livre de Tavernier sur les PICs. Notre approche avec ces programmes de test est d'avoir un exemple qui marche, de pouvoir vérifier en lisant la documentation du fabriquant que l'on comprend l'effet des bits utiles à l'application. On modifie et renomme le programme de test pour s'en assurer..

TxxT0b.asm Utilise le timer pour clignoter un port

Le bit *TOİF* dans le registre İ*ntCon* est activé quand ce compteur TMR0 déborde, c'est-à-dire passe de 255 à 0. Tout le reste se fait par soft! Avec le prédiviseur, on décide à quelle période ce compteur déborde (entre 512 microsecondes et 65ms).

Attention! Les O et les 0 sont souvent difficile à distinguer. Microchip a défini TOİF et pas T0İF (T-zero-IF) qui pourrait sembler plus logique.

Pour démarrer une action toutes les 20 ms par exemple. On initialise le prédiviseur à 1/128

(donc 128x256 us = 30 ms pour un tour à 4MHz) et on raccourcit en réinitialisant à la valeur à -20000/128 (l'assembleur calculera la valeur). Etant donné que le TMR0 est un compteur, on initialise avec 256 moins la valeur de comptage, 256-n = -n car on est en 8 bits.

A noter que pour les processeurs avec des instructions 12 bits (F20x F50x) le registre *IntCon* n'existe pas et que le passage par zéro doit être testé dans une boucle assez rapide.

TxxT0Int.asm Le timer agit par interruption pour remettre à zéro un compteur

Le bit *TOİE* dans le registre İ*ntCon* fait que *TOİF* déclenche une interruption, pour autant que le bit *GIE* soit activé. En effet, il y a plusieurs sources possibles d'interruptions, chacunes authorisées par un bit "enable".

Dans la routine d'interruption, il faut sauver W et le registre Status, et rétablir ces deux registres avant l'instructions RETİ de retour d'interruption. Le travail dans l'interruption est le même qu'après la boucle d'attente du programme précédent.

Le programme principal compte lentement sur un port qui a un affichage. L'interruption vient remettre à zéro ce compteur.

Disponible avec **xx** = 200 508 505 627 630 676 690 84 628 870 **877** 882 884

Actions sur le Timer1 www.didel.com/pictest/TestsTMR1.pdf

TxxT1.asm Compte selon TMR1

Actions sur le Timer2 www.didel.com/pictest/TestsTMR2.pdf

TxxT2.asm Compte selon TMR2

TxxAiga.asm Débranchement selon la valeur d'un mot binaire

On lit sur un port un mot binaire valant 0, 1, 2, ... (00000000, 00000001, 00000010, ...) et on veut exécuter des actions différentes pour chaque valeur lue. Si les valeurs sont très différentes, on compare (en faisant une soustraction) et on saute à la tâche à exécuter. Si les valeurs sont consécutives, c'est plus efficace de décompter et tester si on est à zéro.

Disponible avec **xx** = 200 508 505 627 630 676 690 84 628 870 **877** 882 884

TxxAigb.asm Débranchement en passant par une table

S'il y a plus de 3-4 choix, une table est plus efficace, surtout si elle est entièrement en page 0 (adresses 0-FF). Voir le chapitre 4 du cours "PicGénial".

Add W,PCL ajoute le contenu de W au contenu du compteur d'adresse. C'est un jump calculé, très efficace.

Disponible avec **xx** = 200 508 505 627 630 676 690 84 628 870 **877** 882 884

TxxAlpha.asm Envoi de l'alphabet sur la ligne série

La communication série RS232 peut être simulée par soft jusqu'à 2400b/s (voir xx). Sur la pluspart des processeurs à plus de 14 pattes, RS232 5V est supporté avec une vitesse jusqu'à 108400(?) bits/s. Il faut choisir cette vitesse en initialisant le registre *IniSpBrg* selon la vitesse choisie (voir la doc du fabricant). Les registres *IniRcSta* et *IniTxSta.* Permettent de définir différents modes; on nomme toujours ces valeurs initiales au début du programme ou dans un fichier de définitions importé. La routine IniSer fait le travail d'initialisation et c'est utile d'afficher par exemple un + pour montrer que l'initialisation s'est bien faite. A noter que le bit RC6=TxD doit être initialisé à 1 et RC7=RxD est une entrée. Les routine SendCar, SendCR, SendSpace attendent que le caractère précédent soit parti. La routine GetCar attend que l'on pèse sur une touche. Le processeur ne peut rien faire d'autre en parallèle. La routine IfCar par contre n'attend pas. Si une touche est pressée, le Carry est activé et on sait qu'un nouveau code se trouve dans W et RcReg.

La routine İfCar se mets dans une boucle dans laquelle on exécute des tâches qui ne durent pas plus de 0.1 sec, pour que chaque touche pressée soit interceptée et la tâche correspondante exécutée.

Disponible avec **xx** = 690 628 870 **877** 882 884

TxxEeprom.asm Ecriture et lecture en Eeprom

Presque tous les PİCs ont une mémoire dans laquelle on peut écrire des byte de façon permanente. L'écriture prend 10 ms, quelque soit la vitesse du processeur, mais la lecture seulement 8 instructions. İl faut rajouter la modification de l'adresse et le sauvetage du data. La procédure d'écriture demande une séquence d'instruction particulière et le test du bit *WR* dans le registre *EECON1* pour savoir quand on peut écrire le mot suivant.

Disponible avec **xx** = 508 505 627 630 676 690 84 628 870 **877** 882 884

TxxFlash.asm Ecriture et lecture en Flash (si supporté – 690 877 882)