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

DIDEL

Liste des fichiers de cette notice www.didel.com/picg/Picg.html
Fichier www.didel.com/picg/picg87x/Picg1.pdf

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

Introduction à la programmation avec le PIC*Génial* en vue de développer des robots, jeux, décodeurs, automates, etc.

Table des matières

1. Introduction

La famille des processeurs PIC 12F675 et 16Fxxx couvre une large gamme d'applications temps réel, et convient très bien aussi pour les loisirs: programmer un petit robot, une horloge, une commande de train ou une servocommande est facile, et l'assembleur ne mérite pas la réputation qu'on lui fait. On utilise de moins en moins les circuits TTL ou CMos pour câbler de la logique. Les circuits programmables (FPGA, PLD) sont leur remplacement direct, avec des outils de mise en oeuvre compliqués. Les microcontrôleurs, en particulier les PIC, Scenix, AVR qui ont une mémoire Flash incorporée, sont faciles à utiliser et à mettre en oeuvre, et peuvent traiter beaucoup d'information, surtout si on sait exploiter leurs possibilités.

Le kit "**PIC***Génial*" permet de se familiariser avec la programmation des processeurs Microchip-PIC, qui ont tous à peu près le même répertoire d'instructions. Les PICs 16F84A/16F627/16F870/16F676/12F675 sont particulièrement faciles à mettre en oeuvre. **PIC***Génial* permet de bien comprendre toutes les instructions, et la technique de programmation parfois particulière du PIC, en exécutant et adaptant une suite de programmes didactiques. L'expérience acquise, et la compréhension des techniques très efficaces de programmation que l'on peut appliquer avec les PICs permettent de développer des applications très performantes. La carte WdPicPro2 connectée au port parallèle d'un PC permet de programmer tous les PIC 16F et 12F.

La programmation du PIC est considérablement facilitée par l'assembleur CALM (Common Assembly Language for Microcontrolers) et l'environnement de développement Smile NG ou Picolo, qui s'exécute sur un PC avec Windows 95, 98, NT, 2000. Le programme est édité, assemblé et téléchargé avec un outil unique très convivial. Les erreurs de syntaxe sont signalées dans le source, et l'optimisation d'un paramètre qui demande plusieurs assemblages successifs est on ne peut plus efficace. A cause de cette facilité de développement, un simulateur n'est pas nécessaire. Dans les applications temps réel visées, un simulateur v(par ailleur disponible) est de toute façon trop limité (et les émulateurs sont d'un prix qui les réserve aux applications professionnelles).

L'assembleur CALM est explicite et facilite considérablement l'apprentissage de la fonctionnalité du PIC, et l'écriture de programmes performants. Les Basic Stamps ont beaucoup de succès, mais sont terriblement limités dans leur vitesse d'exécution et la complexité des programmes (100 instructions) que l'on peut écrire. Les programmes Basic apparaissent bien peu satisfaisants pour celui qui a appris le CALM et comprend l'architecture du PIC.

Les PICs 12Fxx 16Fxx présentent l'avantage d'être reprogrammable, et de conserver le code en Flash EPROM une fois programmé. Le programme édité et assemblé avec Smile NG est téléchargé en quelques secondes par le logiciel IC-Prog, puis programmé et exécuté immédiatement. L'éditeur de Smile NG et du Picolo est puissant et permet une mise en évidence des titres et commentaires qui aide à la relecture des programmes. Les notations CALM sont cohérentes et facilitent la compréhension des instructions parfois délicates du PIC.

La cartes PicgPro1 permet de programmer tous les PICs que le logiciel IC-Prog sait faire. L'alimentation se fait avec un transfo 12 Volts alternatif ou 15-18V continu. Du 5V est généré pour alimenter l'une des cartes PicgDev ou la carte de votre application. Le courant maximum est de 200 mA environ. La règle à respecter est que les transistors régulateurs ne doivent pas chauffer excessivement. Un refroidisseur peut être ajouté, mais le refroidisseur du régulateur 12V doit être isolé par rapport au régulateur 5V. Attention, le connecteur parallèle (Centronics) ne fournit pas d'alimentation, mais il y a suffisamment de courant qui fuit par les lignes de données pour allumer les diodes lumineuses et donner l'impression que le module est alimenté. La tension résiduelle de 2,5 à 4V suffit pour faire tourner le PIC, mais pas pour le programmer.

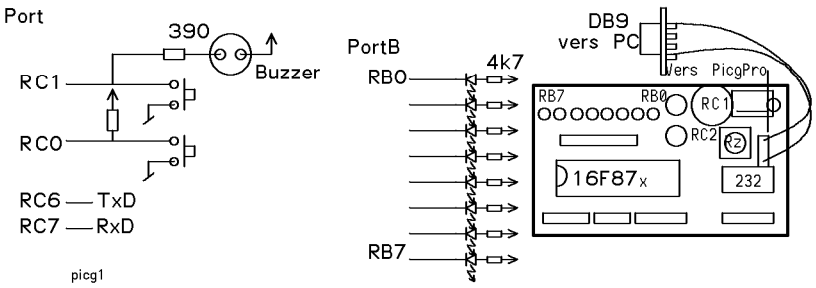
Les cartes de développement WdPicgxxx sont liées à la carte de programmation PicgPro par un câble à 5 fils. Les expériences de cette notice montrent combien il est facile de s'interfacer sur les entrées-sorties du PIC. Le PIC qui a été programmé peut aussi être déplacé de son socle et mis dans un montage personnel, sur une carte imprimée ou wrappée. Le plus efficace est toutefois de prévoir dans chaque montage un petit connecteur pour la programmation. La carte à programmer est reliée à la carte PicgPro par un câble plat de 5 à 20 cm de long et la programmation peut se faire et refaire directement sur votre robot, horloge ou autre montage.

1.1. Environnement de développement

Pour s'exercer avec le PIC et tester des modules de programmes, il faut disposer de lampes, interrupteurs. La carte PicgDev87x offre un maximum de facilités pour s'exercer et tester des application propres. Il ne serait fait mention par la suite que du 16F870, mais les programmes s'appliquent sans autres aux 16F873 et 16F876 (boitiers 28 broches), ainsi qu'aux 16F871 et 16F877 (boitiers 40 broches).

Le 16F870 dispose de trois ports appelés A, B et C, programmables en entrée ou en sortie. La carte PicgDev87x affiche l'état du port B sur 8 diodes lumineuses, allumées si la sortie est à zéro. Le port C lit deux interrupteurs sur RC3 et RC4; le processeur lit un zéro si l'interrupteur est fermé. La ligne RC4 est liée aussi au buzzer. Le terme buzzer est ambigu: certains buzzers font un son fixe tant que l'entrée est active; notre buzzer est un haut-parleur avec une membrane attirée ou relâchée suivant que le courant passe dans la bobine ou pas. Si elle est initialisée en entrée, le processeur lira 1 ou 0 selon que la touche est relâchée ou pressée (il y a inversion). Si le port est initialisé en sortie, le buzzer est opérationnel, sauf si le poussoir de gauche est maintenu pressé; la ligne est alors court-circuitée à la masse, le courant atteint plus de 20 mA, mais l'échauffement dans le processeur est acceptable pour une durée brève de court-circuit.

Côté processeur, le schéma est très simple. Les processeurs 16F87x ont besoin d'un résonateur 4MHz (max 20 MHz). La résistance R1 sur le Clr (80-120k) est nécessaire pour garantir l'état 1 sur l'entrée de remise à zéro du processeur. Le programme démarre automatiquement à la fin du chargement. Le poussoir de remise à zéro permet de redémarrer le programme.



picg1 *Fig. 1 Schéma simplifié du WdPicg87x*

1.2. Smile NG / Picolo / IC-Prog

Pour pouvoir développer, il faut charger l'éditeur-assembleur Smile NG (picdoc:SmileNG|) ou Picolo (picgdoc:Picolo|) et le logiciel programmateur IC-Prog (picgdoc:IcProg|)

Pour installer SmileNG, il suffit de décomprimer le fichier www.didel.com/picg/SmileNG7.zip en mettant le tout dans un répertoire de C:

Pour installer Picolo, il suffit de décomprimer le fichier www.didel.com/picg/Picolo7.zip en mettant le tout dans un répertoire de n'importe quel disque.

Lorsque la carte WdPicg87x est branchée sur le programmateur WDPicPro2, les diodes doivent clignoter, car le 16F870 a déjà été programmé. Chargez le programme Pict870.asm, assemblez avec F5 pour créer le binaire, reprenez le binaire sous IC-Prog, et programmez. Cela fonctionne? Changez la valeur initiale Motif = 16'CC (2'011001100) contre 16'55 (2'01010101). Le clignotement est différent.

Si tout ne s'est pas bien passé, il faut vérifier que la LED verte, qui indique une tension de programmation de 12V, s'allume. Le câble parallèle Centronics ne doit pas être trop long (1,5m est en général acceptable).

1.3. L'éditeur de Smile NG et Picolo

L'éditeur ne pose pas de problèmes à celui qui est familier avec les outils d'un PC. La spécialité de Smile NG est de décoder quelques séquences de mise en page. Ces séquences "boa" (elles commencent par une Barre Oblique Arrière) sont interprétées dès qu'elles sont complètes. Tapez \prog;xxx|yyy au début d'une ligne et vous aurez des gros caractères encadrés pour les xxx de cette ligne, et du gras pour la suite de la ligne, après la barre verticale. Sous SmileNG, la touche F8 permet de passer du mode avec les séquences boa interprétées au mode avec les séquences d'ordre éditables. Sous Picolo, la séquence boa et son interprétation son toujours présentes. La liste des commandes est donnée à la section 8 www.didel.com/picg/picg87x/Picg8.pdf. On les verra progressivement dans les exemples de programmes. L'assembleur ne fait pas de différence entre minuscules et majuscules. Ceci permet de définir des symboles abbrégés plus lisibles, avec une majuscule en milieu de mot, sans que cela fasse erreur si la majuscule a été oubliée. Par contre les séquences "boa" de mise en page le sont (il faut les taper en minuscules)

1.4. L'assembleur CALM de Smile NG

L'assembleur CALM, utilisé au LAMI-EPFL depuis 1975 (8080, 6800 et 2650 à l'époque) définit un ensemble de pseudo-instructions et d'instructions qui séparent clairement l'opération et les opérandes. La syntaxe "Move source,destination" suit les habitudes de Motorola, et est donc inverse de celle d'Intel. Voir www.didel.com/picg/doc/AsBiz.doc pour plus de détails.

Les pseudo-instructions (section 8) permettent d'importer des fichiers de définition, déclarer des tables de nombres. Les macros sont commodes pour alléger les notations, spécialement dans les tableaux. La définition complète du langage CALM se trouve dans deux brochures à disposition.

L´assembleur traduit le programme source en un programme binaire exécutable par le processeur. L´assembleur signale les erreurs de syntaxe, les instructions illégales, mais c´est à vous de programmer l´algorithme correct, et de développer une méthode de mise au point qui permet de retrouver rapidement ses erreurs.

1.5. Principes de la programmation en assembleur

Les notions de cette section, un peu rébarbatives, seront reprises dans les exemples des sections suivantes. Une première lecture en diagonale, et une relecture après quelques exercices est probablement la meilleure approche.

Pour ceux qui n´ont jamais programmé un microprocesseur en assembleur, il faut savoir que le grand principe de fonctionnement des processeurs est de lire une liste d´instructions en mémoire, à des adresses consécutives. Une autre zone mémoire contient les variables. Dans le cas du PIC 16F84/16F870, les instructions ont toutes 14 bits et sont en mémoire "flash" électriquement programmable et effaçable. Les variables ou registres ont 8 bits et sont dans une autre zone mémoire qui commence à l´adresse 0; les premiers registres ont une fonction bien précise et un nom attribué par le fabricant. Les positions suivantes, à partir d´une adresse AdVar qui dépends du processeur (définie dans le fichier de référence), et jusqu´à FinVar-1, sont à disposition du programmeur.

L´exécution du programme dépend d´un compteur d´adresse PC qui pointe les instructions et est en général incrémenté pour prendre les instructions dans l´ordre. Des instructions permettent de sauter à une adresse quelconque, et une caractéristique du PIC est de pouvoir passer par dessus l´instruction suivante (Skip) si une certaine condition a lieu (un résultat nul par exemple). Toutes les instructions du PIC prennent un mot de la mémoire programme (instructions 14 bits pour le 16F84/16F870), et s´exécutent en 1 microseconde (avec un oscillateur à 4MHz). En cas de saut ou autre modification du compteur d´adresse, un 2e cycle de 1 microseconde est nécessaire. C´est donc très facile de calculer les temps d´exécution.

Les instructions agissent sur les registres internes, qui contiennent des mots de 8 bit. Ces registres sont liés aux entrées/sorties (PortA et PortB), commandent des modes et états internes, ou sont utilisables pour des variables. Les noms des registres d´état, de mode du processeur et des ports ont été choisis par le fabricant, et sont connus par l´assembleur (pour les registres identiques dans tous les PICS. Les variables du programme doivent être déclarées par le programmeur.

Les adresses des instructions et des variables sont des mots binaires. On les écrira en hexadécimal (0,1,..9,A,B,C,D,E,F pour les chiffres hexa ayant pour équivalent décimal 0..15) avec un 16´ comme préfixe aux nombres hexa (16´1A est équivalent à 1x16+10 = 26 en décimal). Les contenus des registres sont le plus souvent des bits (pour commander un moteur). On notera 2´10110010 pour un mot binaire, et 16´B2 pour le même mot binaire s´il y a de bonnes raisons d´utiliser l´hexadécimal. Quand on doit répéter des boucles, le plus naturel est de raisonner en décimal. On l´écrit sans signe particulier, et la valeur maximale que l´on peut mettre dans un registre est 255, codé par l´assembleur 2´11111111 = 16´FF. Il ne faut pas oublier que l´assembleur calcule mieux que vous. N´hésitez pas à écrire (12*15)/3 si cela correspond à votre algorithme. Ne convertissez pas vous-même de binaire en décimal ou de décimal en binaire: utilisez toujours la notation la plus naturelle et la plus expressive. Pour les codes Ascii qui représentent les lettres et caractères tapés au clavier et affichés sur un écran, la notation: "B" est la valeur du code Ascii de la lettre B majuscule (peut importe sa valeur, l´assembleur et les circuits d´affichage la connaissent).

Il faut encore bien distinguer les constantes et les variables. Si un registre a l´adresse 16´2B contient par exemple la variable "Vitesse" qui correspond à la vitesse d´un moteur, il être initialisé avec une valeur qui est une constante, par exemple 12 (décimal), que l´on nommera par exemple IniVit. Le transfert dans une variable s´écrit en donnant le nom de la variable (en fait son adresse). Au début du programme, il y a des déclarations qui permettent de savoir que Vitesse = 16´2B (la position mémoire contenant la variable Vitesse est à l´adresse 16´1B) et -iniVit = 12 (ceci n´initialise pas la variable Vitesse!).

L´assembleur traduit le 12 décimal en binaire lorsque cette valeur est mise dans une instruction. Ces assignations ne sont pas des instructions, mais simplement un moyen de travailler avec des noms, et non pas avec des nombres. Il faut bien distinguer quand il est nécessaire de travailler au niveau de la machine, pour des bits et des adresses mémoire, ou au niveau de l´application, pour des valeurs comme une vitesse.

On préfère déclarer les variables et tableaux de variables en annonçant la place nécessaire avec la pseudo-instruction .Blk.16, comme cela sera vu dans la section 1.2.6. L´emplacement des variables dans la zone des registres est annoncé par un .Loc DebVar (DebVar=16´0C pour le 16F84 et DebVar=16´20 pour le 16F870). Par exemple, .Blk.16 1 réserve un mot (en fait de 8 bits mais à cause de l´architecture d´instructions 12 ou 14 bits du PIC, il faut déclarer les variables 8 bits avec un .16 et pas un .8 (anomalie de l´assembleur). La pseudo .Blk.16 5 réserve un tableau de 5 variables. Il ne faut pas oublier de mettre un .Loc 0 au début du programme, car c´est le même compteur dans l´assembleur qui pointe les variables, puis le programme (on peut naturellement faire mieux dans les gros programmes).

A noter encode que pour déclarer des variables, on utilise souvent la notation plus simple .16 1 qui réserve une position (et la remplit avec la valeur 1 qui sera perdue. Pour réserver une suite de 5 positions, il ne faut pas écrire .16 5, mais .Blk.16 5. On peut remplir de la mémoire avec la pseudo .Fill.16 valeur,longueur; c´est ce qui est fait en fin de programmes pour accélérer la programmation: on remplit la fin de la mémoire avec des 16´FF = -1.

Dans le programme, une constante est précédée du signe #. On parle d´adressage par valeur immédiate (adressage immédiat) car la constante est dans le programme, on sait ce qui est transféré. Avec les variables, on transfère un contenu qui peut varier selon l´instant d´exécution. Si on oublie le signe #, et que l´on écrit "Move IniVit,W" au lieu de "Move #IniVit,W", le processeur ira chercher dans la position mémoire 12 la valeur à mettre dans W. Cela peut être une valeur qui par hasard fait marcher l´application correctement. Il faut donc être attentif pour éviter ce genre d´erreur.

JDN 30 novembre 1903