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/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.

1.05

1. Introduction

Le PIC 16F84 et sa version améliorée 16F870 font parties d'une famille de processeurs qui 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 la famille "74" 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 traîter 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 16F84/16F870 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. Une carte complémentaire permet de programmer tous les PIC 16C et 12C, même en boîtier microminiature.

La programmation du PIC est considérablement facilitée par l'assembleur CALM (Common Assembly Language for Microcontrolers) et l'environnement de développement Smile NG, 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 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.

Le PIC 16F84/16F870 présente l'avantage essentiel 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 à travers le circuit de programmation PicgeDe dans le module d'exécution PicgExe et exécuté immédiatement. L'éditeur de Smile NG 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 existent pour plus de 10 processeurs. Le passage d'un processeur à un autre en est considérablement facilité.

La carte PicgeDe permet de programmer tous les PICs, en particulier le 16F84/16F870 de la carte PicgExe. Le connecteur imprimante du PC se branche directement sur cette carte, mais pour éviter que la rigidité de ce câble fasse basculer sans cesse votre montage, un câble plat intermédiaire à 16 broches est utilisé. Huit fils suffisent si vous regardez le

schéma; vous pouvez faire votre propre câble, mais assez court car l'interface électrique du port parallèle est mal spécifié, et les câbles de plus d'un mètre ne sont pas fiables. L'alimentation se fait avec un transfo 12 Volts alternatif ou 15-18V continu. Du 5V est généré pour alimenter la carte PicgExe 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 de 3,5 à 4V suffit pour faire tourner le PIC, mais pas pour le programmer.

La carte de développement PicgeDev, qui permet de programmer le PIC, est liée à la carte d'expérimentation PicgExe par un câble à 5 fils. Les expériences de cette notice montrent combien il est facile de s'interfacer sur les 13 entrées-sorties du PIC. Le 16F84/16F870 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 PicgeDev par un câble plat de 20 à 40cm de long et la programmation peut se faire et refaire directement sur votre robot, horloge ou autre montage.

Pour programmer les PIC miniatures, la carte astucieuse PicgePro vous permet d'économiser les socles très coûteux habituellement nécessaires et de programmer presque tous les PIC.

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 PicgExe, dont le montage est décrit dans le document "Mise en oeuvre du kit" et le schéma avec toutes ses options dans le document "Montage personnels avec **PIC***Génial*" (www.didel.com/picg/Picgm.pdf) offre un maximum de facilités pour s'exercer et tester des application propres.

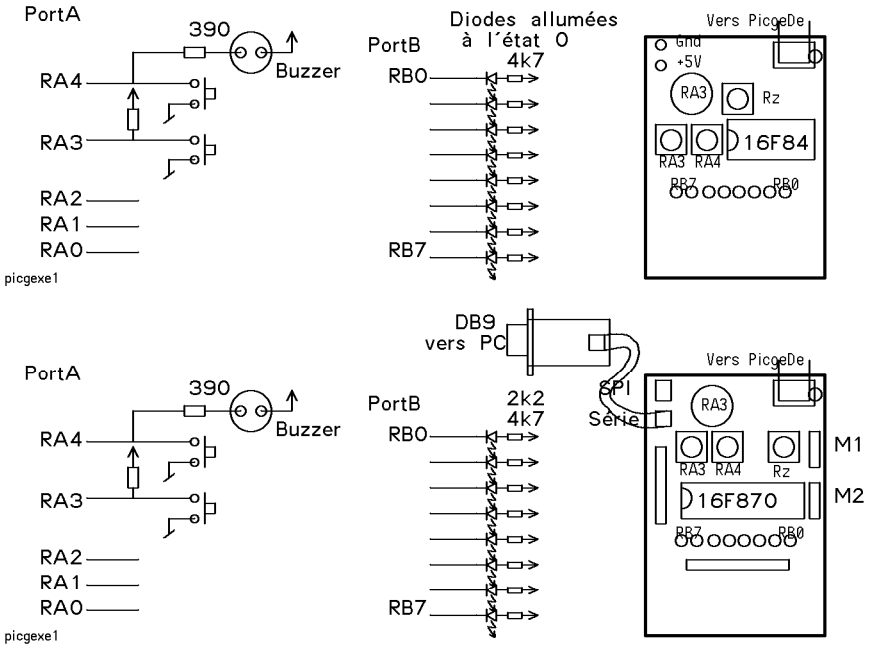
Le 16F84 dispose de deux ports appelés A et B, programmables en entrée ou en sortie. Le 16F870 a un port supplémentaire mis à disposition sur des connecteurs d'extension. La carte PicgExe est prévue pour afficher le port B sur 8 diodes lumineuses. Deux poussoirs sont câblés sur le port A. Un buzzer permet de programmer des sons. Une photodiode infrarouge et un photo récepteur en option permettent de programmer des transmissions infrarouge. Une autre option est le moteur pas-à-pas switec, qui permet de programmer une horloge, ou mettre au point un mécanisme astucieux. Le PIC 16F84/16F870 est pré-programmé; dès la première mise sous tension, vous devez voir toutes les LED (Light Emitting Diodes) clignoter (programme PicTest.asm).

La carte PicgExe est très simple. Ce qui nous intéresse sont les circuits branchés sur les entrées-sorties du processeur. Le port B commande 8 diodes lumineuses. Attention, un 0 en sortie allume la LED. Le port A lit deux interrupteurs sur RA3 et RA4; le processeur lit un zéro si l'interrupteur est fermé. La ligne RA4 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.

Des options permettent de tester une communication infrarouge, de brancher un moteur pas-à-pas switec, des moteurs continu commandés en PWM et des interfaces série. Les algorithmes sont décrits plus loins, et compètent la notice "Montage personnels avec **PIC***Génial*" (www.didel.com/picg/Picgm.pdf).

Côté processeur, le schéma est très simple. Sur la carte avec 16F84, le processeur a besoin d´une résistance et d´un condensateur pour osciller à 4MHz. La place est prévue pour un résonateur 4MHz, ou unquartz à 4 MHz qui nécessite deux condensateurs de 22 pF (la résistance R2 de 4k7 est enlevée). Avec l´oscillateur RC (4,7k / 22pF), la fréquence est supérieure de 10 à 20% à 4 MHz. Pour régler la fréquence, on peut ajouter un condensateur de 3 à 6 pF, ou mettre un résistance de 56 kOhm avec unpotentiométre de 200-600k en parallèle pour un réglage fin. 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.

La carte PicgExe870 est compatible, mais le résonateur 4MHz est standard, pour faciliter la liaisons série. Des modules série et moteur ont été développées avec des connecteurs directement compatibles. Ces modules peuvent aussi être branchés sur un PicgExe F84, mais il faut mettre ces connecteurs sur la sone universelle et tirer quelques fils.



picgexe1 *Fig. 1 Schéma simplifié du* **PIC***Génial*

1.2. Smile NG

Smile NG vous est livré sur disquette dans une version PIC uniquement. D´autres processeurs sont supportés par Smile NG. Votre disquette contient le source des programmes de cette notice, que l´on trouve aussi sous www.didel.com/picg/Picg.html

Installez Smile NG sur votre disque en copiant le répertoire Smilepicg sur la disquette dans c: (cela doit être un répertoire direct dans C:). Ajoutez un raccourci à C:smilepicg/smileng.exe sur le bureau. Connectez le **PIC***Génial* sur le câble parallèle Centronics.

Si vous êtes sour Windows NT ou Windows 2000, un driver spécial doit être installé. Le fichier dans www.didel.com/picg/Pariodrv.zip doit être installé.

Appelez SmileNG. Chargez, depuis le répertoire PICGenial, le programme PicTest.asm, expliqué plus loin.

Vérifier que la configuration de chargement est sur "16F84-RC PicGénial" ou "F870-XT PicGénial".

Assemblez et chargez avec F5 ou en cliquant l'icône en haut au milieu. La lampe verte de PicgeDe s'allume pendant la programmation, et les deux lignes RB6, RB7, affichées sur les diodes rouge du module, transfèrent l'information en série. Attention, si ces lignes sont court-circuitées ou trop chargées, le chargement ne peut pas se faire.

Le programme PicTest clignote les diodes lumineuses de la carte, mais en alternance. Cela fonctionne? Heureux? Changez la valeur initiale 16'CC (2'011001100) contre 16'55 (2'01010101). Le clignotement est différent.

Si tout ne s'est pas bien passé, il faut remonter patiemment la filière. Pendant le téléchargement du programme, la diode verte, qui indique une tension de programmation de 12V, doit être allumée. Le câble parallèle Centronics ne doit pas être trop long (1,5m est en général acceptable). L'imprimante doit être LPT1, comme on peut le vérifier dans un sous-menu de "Configure". La tension de sortie des régulateurs doit être de 13,1V et 5V. Le mode "PicSmart test" dans "Tools-Configure-Download" permet d'activer les 3 lignes utilisées pour la programmation et de vérifier si les LEDs correspondantes s'activent. Pour tester les signaux de l'interface, il ne faut pas connecter la carte PicgExe. Ne modifiez rien dans ce panneau de configuration, même si cela ne marche pas; vour risquez de neutraliser définitivement votre PIC.

Le fichier "Installation.txt" sur la disquette et en www.didel.com/picg/Installation.txt est un peu plus explicite. Le fichier d'aide est encore bien incomplet pour vous guider dans la richesse de Smile NG si vous voulez tout comprendre. Mais il y a encore la complexité du PC et les anomalies d'un "compatible" à l'autre. Seul un gourou expérimenté pourra vous aider.

1.3. L'éditeur de Smile NG

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. La touche F8 permet de passer du mode avec les séquences boa interprétées au mode avec les séquences d'ordre éditables. La liste des commandes est donnée à la section 8 www.didel.com/picg/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 toujours les taper en minuscule)

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.

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 qui dépends du processeur, 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, 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 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 registres supplémentaires doivent être déclarés). 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, il y a aussi une 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. Le registre a l'adresse 16'1B, que l'on va nommer par exemple "Vitesse", car il correspond à la vitesse d'un moteur, doit être initialisé avec une valeur qui est une constante, par exemple 12 (décimal), que l'on nommera VitInit. 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 = 16'1B (la position mémoire contenant la variable Vitesse est à l'adresse 16'1B) et VitInit = 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).

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 VitInit,W" au lieu de "Move #VitInit,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 24 avril 1901