La programmation des PIC en C

Structure des programmes, utilisation des entrées sorties

Réalisation : HOLLARD Hervé.
http://electronique-facile.com
Date : 01 août 2004
Révision : 2.1

Sommaire

Introduction

Les microcontrôleurs PIC de la société Microchip, sont depuis quelques années dans le "hit parade" des meilleures ventes. Ceci est en partie dû à leur prix très bas, leur simplicité de programmation, les outils de développement que l'on trouve sur le NET.

Aujourd'hui, développer une application avec un PIC n'a rien d'extraordinaire, et tous les outils nécessaires sont disponibles gratuitement. Voici l'ensemble des matériels qui me semblent les mieux adaptés.

Ensemble de développement (éditeur, compilateur, simulateur) :
MPLAB de MICROCHIP **http://www.microchip.com**
Logiciel de programmation des composants:
IC-PROG de Bonny Gijzen **http://www.ic-prog.com**
Programmateur de composants:
PROPIC2 d'Octavio Noguera voir notre site **http://electronique-facile.com**

Pour la programmation en assembleur, beaucoup de routines sont déjà écrites, des didacticiels très complets et gratuits sont disponibles comme par exemple les cours de **BIGONOFF** dont le site est à l'adresse suivante **http://abcelectronique.com/bigonoff**.

Les fonctions que nous demandons de réaliser à nos PIC sont de plus en plus complexes, les programmes pour les réaliser demandent de plus en plus de mémoires. L'utilisateur est ainsi à la recherche de langages "évolués" pouvant simplifier la tâche de programmation.

Depuis l'ouverture du site **http://electronique-facile.com**, le nombre de questions sur la programmation des PIC en C est en constante augmentation. Il est vrai que rien n'est aujourd'hui disponible en français.

Mon expérience dans le domaine de la programmation en C due en partie à mon métier d'enseignant, et à ma passion pour les PIC, m'a naturellement amené à l'écriture de ce **didacticiel**. Celui-ci se veut **accessible à tous** ceux qui possèdent une petite expérience informatique et électronique (utilisation de Windows, connaissances minimales sur les notions suivantes : la tension, le courant, les résistances, les LEDs, les quartz, l'écriture sous forme binaire et hexadécimale.).

Structure de ce document

Ce document est composé de chapitres. Chaque chapitre dépend des précédents. Si vous n'avez pas de notion de programmation, vous devez réaliser chaque page pour progresser rapidement.

Le type gras sert à faire ressortir les termes importants.

La couleur **bleue** est utilisée pour vous indiquer que ce **texte** est **à taper** exactement sous cette forme.

La couleur **rouge** indique des **commandes informatiques** à utiliser.

Le matériel nécessaire

Les deux logiciels utilisés lors du premier fascicule.

Un programmateur de PIC comprenant un logiciel et une carte de programmation. Vous trouverez tout ceci sur notre site.

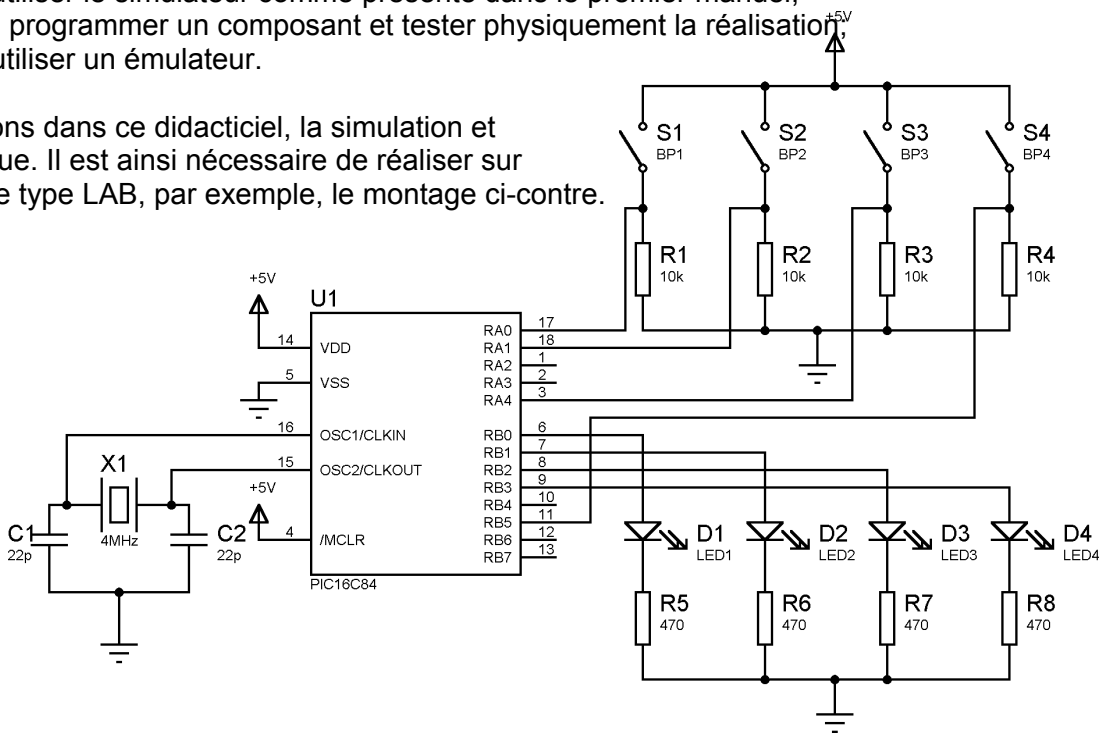
Un PIC 16F84, un quartz de 4MHz, deux condensateurs de 22pF, 4 leds rouges, 4 résistances de 470 ohms, 4 interrupteurs, 4 résistances de 10 Kohms. Une platine d'essai sous 5 volts.

La platine d'essai

Pour tester les programmes proposés il est possible :

-d'utiliser le simulateur comme présenté dans le premier manuel;
-de programmer un composant et tester physiquement la réalisation;
-d'utiliser un émulateur.

Nous utiliserons dans ce didacticiel, la simulation et le test physique. Il est ainsi nécessaire de réaliser sur une platine de type LAB, par exemple, le montage ci-contre.



But à atteindre

Ce didacticiel vous permettra de **comprendre la structure d'un programme en C**, ainsi que la **structure des entrées et des sorties** du PIC 16F84.
Vous serez à la fin de ce document capable de **manipuler** simplement **les pattes du 16F84**, capable de **simuler** les pattes du PIC et de **programmer un composant**.

Pour atteindre ce but, nous allons réaliser un programme de gestion d'éclairage constitué de 4 boutons poussoirs et de 4 leds (Voir page 4). Chaque bouton poussoir pourra commander la led qui lui est associée.

Premières règles d'écriture en C

Voici un premier programme que nous allons commenter pour en dégager quelques règles.

```
void main(void)
{
// debut
TRISB = 0b11110000;
TRISA = 0xff;
RB1 = 0; RB1=1; }
```

Règle 1 : Les **instructions propres au** langage **C** doivent être écrites en **minuscule** (void main(void)).
Règle 2 : Les **instructions particulières aux microcontrôleurs** doivent être écrites en **majuscule** (TRISB).
Règle 3 : Les **retours à la ligne** et les **espaces** servent uniquement à aérer le code. Ils **peuvent ne pas être utilisés**.
Règle 4 : Les **accolades délimitent un groupement** d'instruction.
Règle 5 : Le signe **//** indique un **commentaire**.
Règle 6 : Le **point virgule** indique la **fin d'une instruction**.
Règle 7 : Le mot clé **0b** indique que la suite est écrite en **binaire**.
Règle 8 : Le mot clé **0x** indique que la suite est écrite en **hexadécimal**.
Règle 9 : L'**absence** de mot clé indique une écriture en **décimal**.

Sous MPLAB :

- Pour simuler en pas à pas sous MPLAB, il faut des retours à la ligne car seule la dernière instruction de la ligne est affichée, mais elles sont toutes simulées.

- Les instructions propres au langage C sont en gras et bleu foncé.

Ce qui donne sous MPLAB:

```
void main(void)
{
// debut
TRISB = 0b11110000;
TRISA = 0xff;
RB1 = 1;
RB1=0;
}
```

La notion de fonction

En assembleur (comme en basic), les lignes de codes sont écrites les unes à la suite des autres avec des sauts à la ligne, et sont exécutées dans cet ordre. Pour ne pas effectuer certaines instructions, il faut utiliser des instructions de sauts.

En C (comme en Pascal) les instructions de sauts sont quasiment inexistantes. Il faut écrire **plusieurs petits programmes** que l'on appelle "**FONCTION**". Une de ces fonctions est appelée **fonction principale**. C'est cette fonction qui **s'exécute** de façon autonome **en premier**.

La structure d'une fonction principale est la suivante

```
void main(void)
{
// espace réservé à la fonction principale
}
```

Le mot clé "main" signifie fonction principale. Les mots void seront expliqués lorsque l'on abordera l'utilisation des fonctions.

L'initialisation des E/S

Chaque patte du PIC16F84 peut être **configurée indépendamment** des autres en entrée ou en sortie.

Dans la RAM du 16F84 existent des bytes particuliers appelés REGISTRES. Nous allons nous intéresser à deux de ces registres : TRISA et TRISB.

Les états des bits de ces registres sont représentatifs de la fonction des pattes du 16F84. Chaque bit représente une patte du 16F84 comme suit.

Registre TRISA :

Nom de la patte	RA4	RA3	RA2	RA1	RA0
Bit correspondant	4	3	2	1	0

Les bits 5 à 7 n'ont pas d'utilité.

Registre TRISB :

Nom de la patte	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0
Bit correspondant	7	6	5	4	3	2	1	0

Si un **bit est à 1**, la patte associée est en **entrée**. Si un **bit est à 0**, la patte est en **sortie**.
A la **mise sous tension** tous ces bits sont à **1**.

Il est donc nécessaire au début de la fonction principale de mettre chaque bit à la bonne valeur. Nous allons utiliser pour cela le signe =.

Pour notre système, les pattes RA0, RA1, RA4, RB5 sont en entrées et les pattes RB0, RB1, RB2, RB3 sont en sorties. Nous allons donc par exemple écrire:

```
TRISA = 0b11111111;
TRISB = 0b11110000;
```

Attention : Les bits sont toujours écrits dans l'ordre suivant : bit7, bit6, … bit0.

Ces instructions sont valables quel que soit le compilateur pour PIC. Il se peut tout de même que les mots clés TRISA et TRISB soient remplacés par d'autres dans un autre compilateur.

La manipulation de TRISA et TRISB peut se faire à volonté dans le programme.

Manipulation des pattes en sorties

Lorsqu'une patte est en sortie, il est facile de lui attribuer un état grâce à deux registres : PORTA et PORTB.

Chaque bit représente une patte du 16F84 comme suit.

Registre PORTA :

Nom de la patte	RA4	RA3	RA2	RA1	RA0
Bit correspondant	4	3	2	1	0

Les bits 5 à 7 n'ont pas d'utilité.

Registre PORTB :

Nom de la patte	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0
Bit correspondant	7	6	5	4	3	2	1	0

Si un **bit est à 1**, **la patte** associée **est à 1**. Si un **bit est à 0, la patte** associée **est à 0**.
A la **mise sous tension** tous ces bits sont à **1**.

Il est donc possible d'éclairer la LED1 et d'éteindre les autres grâce à l'instruction suivante :

```
PORTB = 0b00000001;
```

Il est aussi possible de manipuler une patte sans toucher aux autres. Pour éclairer la LED 2, il faut mettre RB1 à 1. Ce qui donne en C :

```
PORTB.1 = 1;
```

Ces instructions sont valables quel que soit le compilateur utilisé. Il se peut tout de même que les mots clés PORTA et PORTB soient remplacés par d'autres dans un autre compilateur.

Avec certains compilateurs, il est possible de remplacer PORTB.1, par exemple, par RB1. Ce qui donne:

```
RB1 = 1;
```

Au stade où nous en sommes, nous pouvons comprendre le programme du fascicule 1.

Ci dessous, un exemple similaire que vous pouvez simuler comme vu dans le fascicule 1.

```
// La LED1 du composant clignote
// Attention de respecter les majuscules
void main(void)
{
  PORTB = 0;              // Initialisation des pattes du microcontrôleur
  TRISB = 0b11110000;

  for (;;)  {             // La suite du programme s'effectue en boucle
            RB0 = 1;
            RB0 = 0;   }
}
```

Explication 1 : Lorsqu'une patte est mise en sortie, son état est aléatoire. Il est donc dans notre cas possible d'avoir certaines LEDS allumées. Nous allons éviter ce problème en affectant une valeur aux sorties avant de les déclarer en sortie.
Explication 2 : Il n'est pas nécessaire de configurer TRISA car il est à 1 à la mise sous tension.
Explication 3 : L'instruction FOR (;;){ Instructions } sera expliquée dans le fascicule 3. Elle permet de répéter les instructions entre les accolades infiniment.

Manipulation des pattes en entrées

Lorsqu'une patte est en entrée, il est facile de connaître son état grâce aux deux registres : PORTA et PORTB.

Si un **bit est à 1**, **la patte** associée **est à 1**. Si un **bit est à 0, la patte** associée **est à 0**.

Il n'est pas possible de modifier un bit du PORTA ou PORTB si ce bit représente une patte en entrée.

Voici le programme à réaliser :

- Si on appuie sur le bouton poussoir 1, la led 1 s'allume. Si on relâche ce bouton poussoir, la led s'éteint (on recopie donc l'état de RA0 dans RB0).
- Les autres boutons poussoirs fonctionnent de façon similaire.

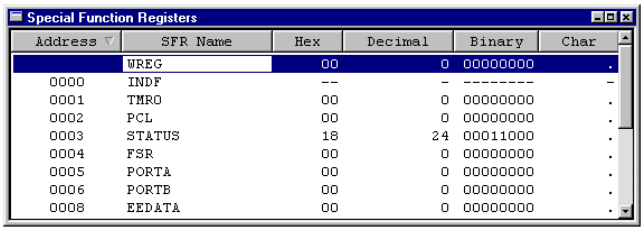
```
// Attention de respecter les majuscules et minuscules
void main(void)
{
   PORTB = 0;                    // Initialisation des pattes du microcontrôleur
   TRISB = 0b11110000;

   for (;;) {                    // La suite du programme s'effectue en boucle
            RB0 = RA0;
            RB1 = RA1;
            RB2 = RA4;
            RB3 = RB5;    } }
```

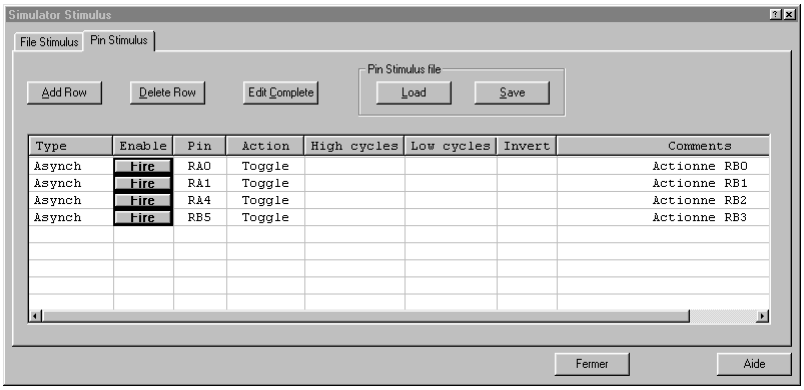
Ce programme se passe de commentaires. Nous allons dans un premier temps le simuler. Nous allons devoir attribuer des états aux pattes RA0, RA1, RA4, RB5 pendant la simulation. Pour cela nous allons nous servir de stimulis.

Simulation avancée et programmation

Réaliser les opérations du fascicule 1 jusqu'à la fin de la compilation (opération Build).

Ouvrir la fenêtre "Special Function Registers" du menu "Voir". Afin de visualiser les sorties (le PORTB) comme ci dessous.





Ouvrir la fenêtre "Stimulus" du menu "Debugger", cliquer sur l'onglet "Pin Stimulus". Créer 4 lignes grâce à la commande "Add Row". Remplir les 4 lignes comme ci-contre.

Avec une telle configuration, à chaque action sur le bouton "Fire" lors de la simulation, l'état de la patte associée change.

Organiser la page de façon à faire apparaître les deux fenêtres précédentes, cliquer sur "Reset" puis sur "Animate". Actionner "fire" pour faire évoluer les sorties. Pour arrêter, cliquer sur "Halt", "F5" ou aller chercher la commande "Halt" du menu "Debugger"

Le test physique

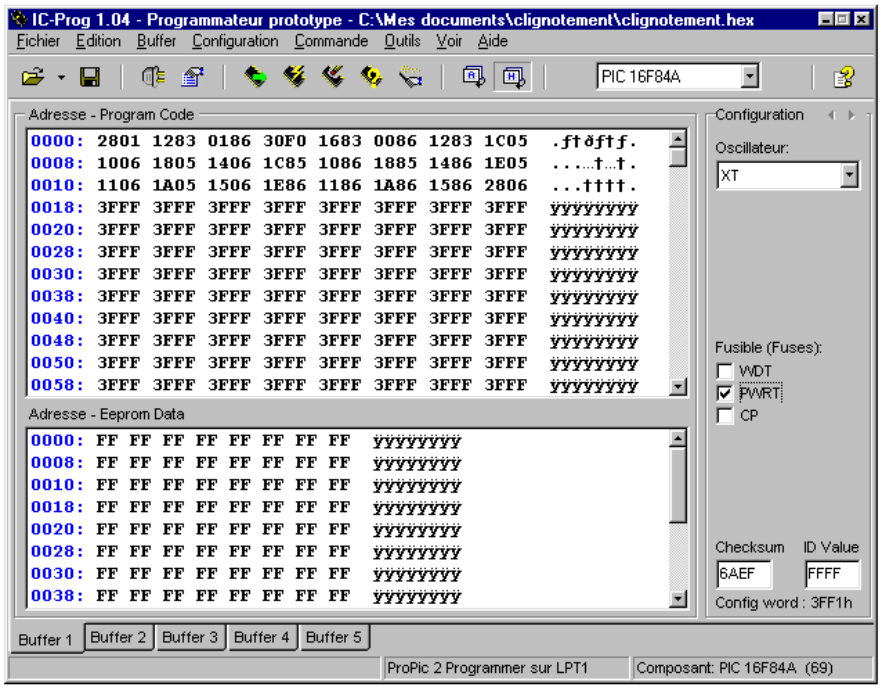
La prochaine étape est de programmer un 16F84 afin de tester physiquement l'application. Pour cela, vous pouvez utiliser l'ensemble proposé dans le site électronique-facile.com.

Lors de la compilation, un fichier avec l'extension "hex" a été créé. Ce fichier contient le code destiné au PIC. Ouvrir ce fichier avec votre programmateur, choisir comme destination un PIC16F84A.

Certaines options de fonctionnement doivent être signalées au programmateur qui informera le PIC lors de la programmation. Pour cela:

- Choisir un oscillateur de type XT (quartz inférieur ou égal à 4MHz).
- Décocher l'option WDT (remise à 0 en cas de problème).
- Cocher l'option PWRT (retard à la mise sous tension pour attendre que l'alimentation se stabilise).
- Décocher l'option CP (code de protection pour éviter la lecture du programme).

Vous devez sous IC-Prog obtenir la fenêtre ci-dessous.



Programmer le composant et le tester sur la platine.

Les équivalences des entrées et sorties

Pour faciliter la lecture d'un programme, il est possible de remplacer des noms désignant des bytes, des bits, des expressions par d'autres.

Pour remplacer un mot par un autre : #define nouveau_mot ancien_mot
Pour remplacer une expression par une autre : #define nouvelle_expression ancienne_expression

- Voici le programme final à essayer. Il est identique au précédent, mais plus lisible.

```
// Attention de respecter les majuscules et minuscules
//------------------------------E/S-------------------------------------------
#define sortie PORTB
#define inter1 RA0
#define inter2 RA1
#define inter3 RA4
#define inter4 RB5
#define led1 RB0
#define led2 RB1
#define led3 RB2
#define led4 RB3
//----------------------------Fonction principale----------------------------
void main(void)
{
    sortie = 0;                     // Initialisation des pattes du microcontrôleur
    TRISB = 0b11110000;

    for (;;)  {                     // La suite du programme s'effectue en boucle
                led1 = inter1;
                led2 = inter2;
                led3 = inter3;
                led4 = inter4;
              }
}
```

Nous arrivons au bout de cette deuxième partie. Il est évident qu'à ce niveau le développement est encore limité. Avec le troisième didacticiel, vous serez enfin capable de :

- utiliser la mémoire;
- faire des choix;
- créer des boucles.

Vous pourrez facilement concevoir la plupart des applications ne faisant pas intervenir la notion de temps. Ce fascicule vous permettra enfin de rentrer complètement dans le monde de la programmation des PIC en C.