



Table des matières

INTRODUCTION.....	4
Première partie: LES OUTILS.....	5
Les outils logiciels.....	5
L'éditeur de texte CONTEXT.....	7
Le compilateur SDCC.....	8
Assembleur et Linker GPUTILS.....	8
Le Bootloader TinyBootloader.....	9
Le répertoire PROJET.....	9
RESUME DE LA PREMIERE PARTIE.....	9
CONTENU DU CD DE DEVELOPPEMENT.....	10
Deuxième partie: Mise en oeuvre des outils logiciels.....	11
Le bootloader.....	11
Le module processeur.....	11
Le module downloader.....	11
Configurer Context.....	13
Troisième partie: Le matériel.....	16
La platine de développement.....	16
Un aperçu du 18F252.....	18
Première application: Une led clignote !!.....	21
Le montage.....	21
Le programme.....	22
Résumé.....	26
Optimisation.....	27
Deuxième application: Un chenillard.....	29
Le montage.....	29
Le programme.....	30
Résumé.....	32
Troisième application: Un télérupteur.....	34
Le montage.....	34
Le programme.....	35
Résumé.....	38
Quatrième application: Gestion d'un afficheur LCD.....	38
Le montage.....	38
Fonctionnement de l'afficheur.....	42
Les instructions.....	42
Initialisations de l'afficheur.....	44
Le programme.....	44
Création d'une bibliothèque.....	52
Résumé.....	54
Cinquième application: Utilisation de l'UART du PIC.....	56
Le montage.....	56
Un outil supplémentaire HYPERTERMINAL.....	56
Le programme	58
Résumé.....	60

INTRODUCTION

Bienvenue dans le monde des logiciels OPEN SOURCE, ce livre est le fruit de quelques années de pratique de la programmation des microcontrôleurs PIC.

Il devrait vous permettre de découvrir comment concevoir des applications à base de PIC et de les programmer en C avec SDCC. Cet outil, mis à la disposition de tous, est digne de rivaliser avec la plupart des compilateurs commerciaux. L'utiliser, c'est non seulement accéder à la programmation, mais aussi contribuer à son évolution et rendre hommage à tous ceux qui ont passé du temps à le développer.

Ce livre enfin, diffusé sous licence GPL, est le début de l'aventure des logiciels OPEN SOURCE pour moi. Familier des compilateurs commerciaux depuis plusieurs années, et ayant démarré la programmation en assembleur, le passage à l'OPEN SOURCE est un moyen de découvrir un nouveau monde. Ce pas à pas de la programmation des PIC est ma contribution au monde du libre, juste un moyen de rendre hommage à tous ceux qui consacrent du temps pour mettre à la disposition de tous les outils de la création.

Première partie: LES OUTILS

Les outils logiciels

Un certain nombre d'outils logiciels sont nécessaires pour développer des applications en C. Un éditeur d'abord, qui nous permettra d'écrire les programmes et de sauvegarder les sources pour une utilisation ultérieure. Un compilateur ensuite, qui assurera la translation du code en C vers du code en assembleur, un assembleur et un linker enfin qui convertiront le code assembleur en code hexadécimal compatible avec le processeur utilisé, dans notre cas les PIC de la série 18.

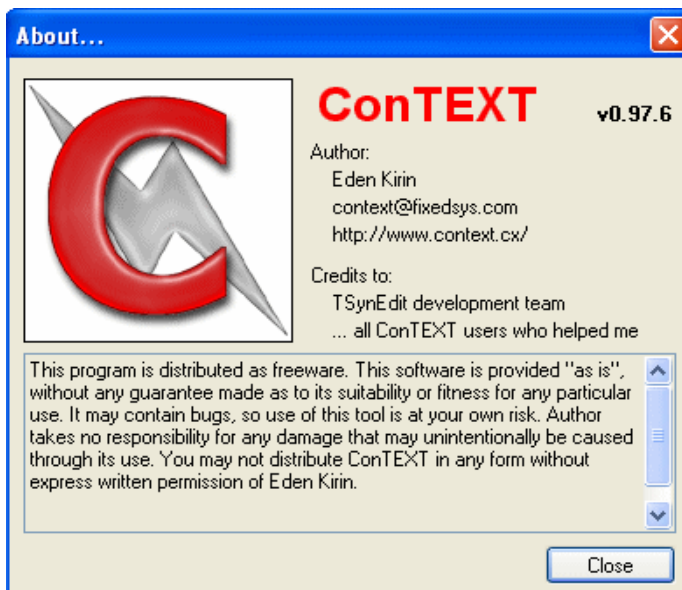
Enfin, une fois ce code généré, il faudra le transférer dans le processeur. Pour cela, deux méthodes peuvent être utilisées; le programmeur ou le bootloader. Le programmeur recevra le processeur et permettra de transférer le programme depuis le PC. Cette méthode, bien qu'utilisée la plupart du temps comporte quelques inconvénients. Le premier et non des moindre est de vous obliger à déplacer le processeur de la carte cible (celle qui vous permet de tester votre programme) au programmeur à chaque changement du programme même minime soit il. Le temps de programmation étant non négligeable (de l'ordre de 2 minutes pour un 18F458) ces manipulations deviennent rapidement fastidieuses, surtout pour les débutants qui ont malheureusement tendance à faire quelques erreurs.

La deuxième méthode, celle que je préconise est celle du bootloader. Le bootloader est un petit module logiciel qui est placé une fois pour toute dans le processeur. Il permet d'établir un dialogue avec le PC via une petite interface et de placer le programme en mémoire en un temps record. Le processeur, dans ce cas, est directement placé sur la carte cible et n'en bouge plus. Les modifications du programme peuvent être réalisées pratiquement 'inline' et le développement est bien sûr beaucoup plus rapide.

Dans les pages qui suivent nous utiliserons cette méthode après l'avoir décrite en détail dans ce chapitre.

Les outils et les méthodes décrits dans ce livre ne sont probablement pas la panacée en matière de développement mais elles ont fait leurs preuves depuis de nombreuses années dans mon atelier. Cet ouvrage est un manuel 'pas à pas' qui je l'espère vous amènera à développer des applications plus lourdes que celles que je réalise. Une fois familiarisé avec ces méthodes et ces outils, vous aurez le loisir de choisir des outils plus performants et de développer vos propres méthodes. Mais pour le moment, soyez patients, un apprentissage rapide de la programmation passe avant tout par le suivi strict des consignes qui vous sont données. **C'est le gage de votre réussite !!**

L'éditeur de texte CONTEXT



Context est un éditeur de texte freeware développé par Eden Kirin.

Il permet d'utiliser la plupart des langages avec la colorisation syntaxique correspondante. Des touches de fonction permettent de compiler, d'assembler et de linker les applications depuis l'éditeur. Ces fonctions font de Context le rival des environnements de développement intégré.

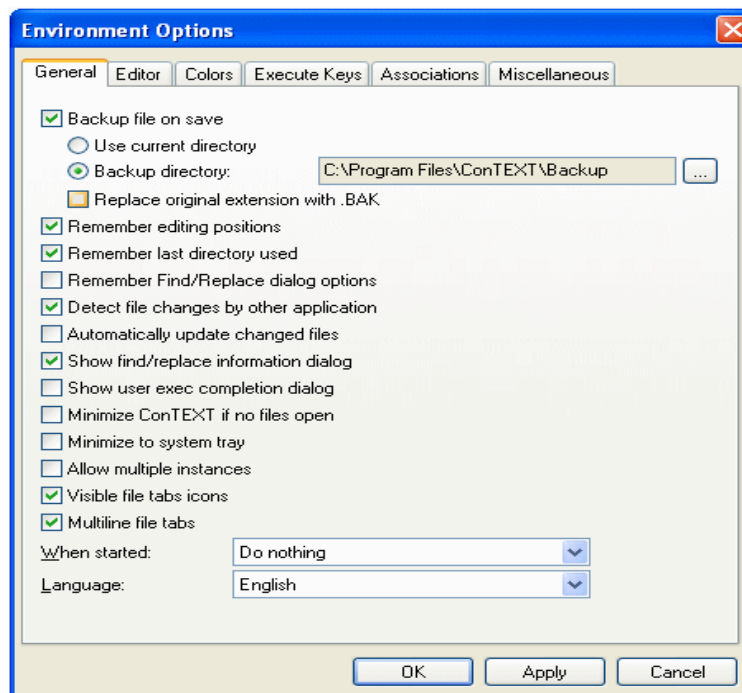
Context est livré sous forme d'un exécutable que vous devrez installer sur votre disque, l'assistant d'installation vous guidera tout au long de ce processus.

L'exécutable d'installation peut être téléchargé à l'adresse suivante:

www.context.cx

Une fois installé, vous devrez démarrer Context pour le paramétrer en langue française, pour cela, lancez Context puis dans le menu **Options/Environment options** sélectionnez Français dans la case **Language**.

Après avoir quitté Context puis redémarré, vous aurez un éditeur de texte entièrement en français. Nous verrons un peu plus bas dans ce chapitre comment configurer context pour utiliser les touches de fonction pour compiler et charger votre application.



Le compilateur SDCC

SDCC est la contraction de Small Device C Compiler. Ce compilateur OPEN SOURCE initialement développé pour les processeurs INTEL de la série 51 a été porté pour les PIC par Scott DATTALO. Il permet aujourd'hui de développer des applications fiables sur les PIC de la série 18 au détriment d'une taille de code un peu importante. Compte tenu de la taille de la mémoire de ces processeurs, la taille du code n'est pas un élément important surtout pour des débuts dans la programmation en C. SDCC est en constante évolution, notamment pour les PIC. Nous vous conseillons de télécharger et d'installer régulièrement la dernière version.

L'installateur Windows peut être téléchargé à l'adresse suivante:

<http://sdcc.sourceforge.net/>

Une fois installé, vous disposerez dans le dossier Program Files de windows d'un dossier nommé SDCC. Ce compilateur étant un compilateur en ligne de commande, vous ne trouverez pas de raccourci dans le menu démarrer de windows. Nous verrons au chapitre 'Mise en oeuvre' comment l'utiliser avec Context.

Assembleur et Linker GPUTILS

GPUTILS est un ensemble d'outils OPEN SOURCE pour les PIC. Un assembleur, un éditeur de liens et un simulateur composent ce package. GPUTILS est l'alternative à MPLAB, ces outils ont été développés à l'origine pour permettre le travail avec les PIC sous LINUX.

L'exécutable d'installation peut être téléchargé à l'adresse suivante:

<http://gputils.sourceforge.net/>

Choisissez le package DOS qui se compose d'un fichier ZIP et décompressez le dossier GPUTILS à la racine du disque C de votre ordinateur.

Vous disposez maintenant d'un assembleur et d'un éditeur de lien.

Le Bootloader TinyBootloader

Ce bootloader a été développé par Claudiu Chiculita. Il est probablement le plus compact en taille (moins de 100 mots) et a été porté sur de nombreux PIC. Le package logiciel est composé d'un downloader, programme windows permettant de transférer le code dans le processeur cible, et d'un module écrit en assembleur que vous devrez programmer dans votre PIC.

L'ensemble du package logiciel peut être téléchargé à l'adresse suivante:

<http://www.etc.ugal.ro/cchiculita/software/picbootloader.htm>

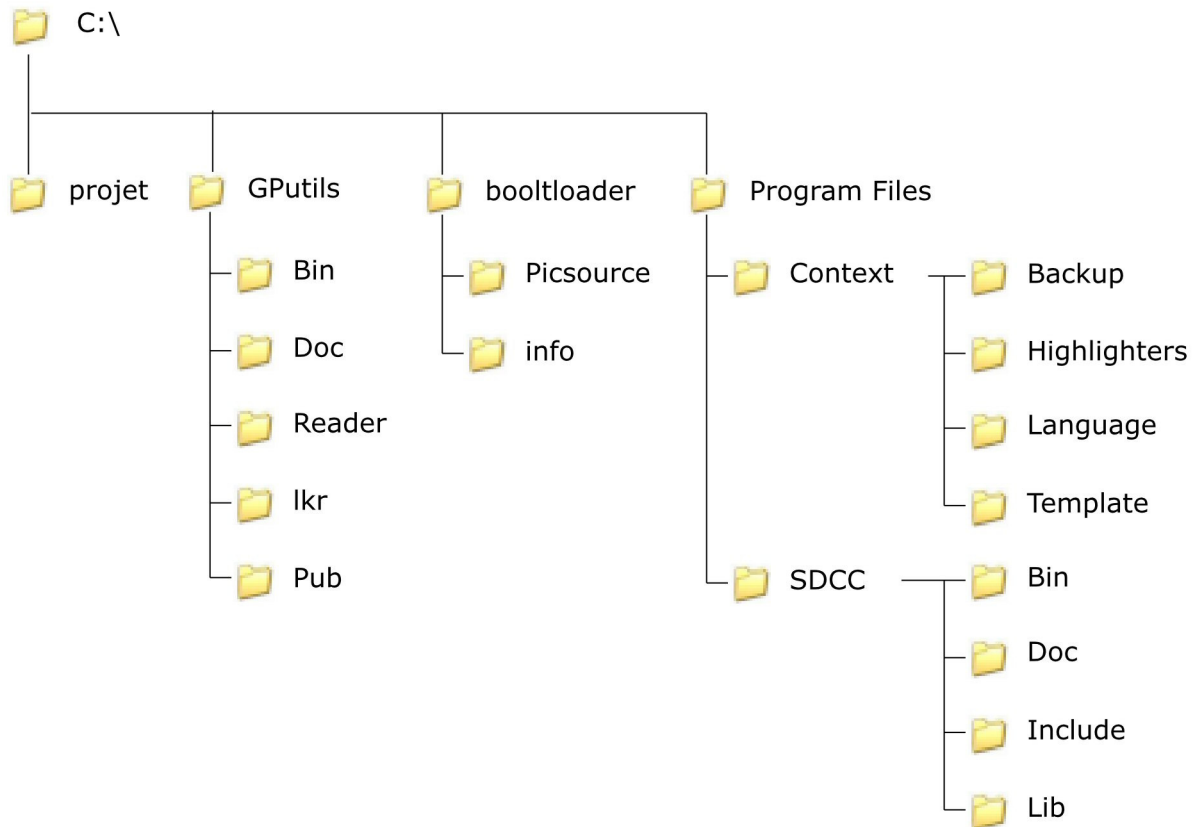
Créez un dossier 'bootloader' à la racine du disque C et sauvegardez l'ensemble du bootloader dans ce dossier.

Le répertoire PROJET

Il s'agit maintenant de créer un dossier dans lequel vous sauvegarderez les différents programmes que vous allez créer avec ces outils logiciels. Pour cela, créez un nouveau dossier à la racine du disque C nommé 'projet'. Vous créerez à l'intérieur de ce dossier les différents dossiers de vos futures applications.

RESUME DE LA PREMIERE PARTIE

Vous disposez maintenant de l'ensemble des outils logiciels vous permettant de développer avec des PIC. Nous vous proposons ci-dessous l'arborescence des répertoire de votre disque avec les outils qui ont été installés et les différents dossiers et sous dossiers s'y rapportant. Vérifiez que l'arborescence des répertoires correspond à celle de votre disque, vous pourrez ensuite passer à la mise en oeuvre de ces différents outils dans la deuxième partie de cet ouvrage.



CONTENU DU CD DE DEVELOPPEMENT

Les utilisateurs de notre kit de développement pourront utiliser ce CD pour installer les outils décrits dans la première partie de cet ouvrage. Les versions proposées dans ce CD sont les versions à la date de la création du CD. Pour SDCC en particulier, nous vous conseillons de télécharger la dernière version sur le site. Pour le reste des outils, vous trouverez sur notre site, dans la rubrique réservée au développement en C les indices des dernières versions logicielles de tous ces outils. Nous vous engageons à visiter régulièrement cette page et à vérifier la disponibilité de nouvelles versions.

Deuxième partie: Mise en oeuvre des outils logiciels

Le bootloader

Nous avons vu dans la première partie de cet ouvrage que le bootloader était composé d'un module logiciel pour windows, le 'downloader' et d'un module logiciel pour le processeur. Nous allons voir les caractéristiques de chacun de ces modules et leur utilisation.

Le module processeur

Il s'agit d'un programme écrit en assembleur et placé en haut de mémoire du processeur. Ce module doit être adapté au processeur avec lequel il est utilisé et à la fréquence d'horloge à laquelle le processeur fonctionne. Le package logiciel fourni par Claudiu Chiculita est composé d'un certain nombre de fichiers hex déjà adaptés pour certains processeurs. Vous trouverez ces fichiers dans le dossier [Picsource/18F](#) pour les PIC de la série 18F. Le fichier [tinybld_18F252_20Mhz.hex](#) par exemple est le fichier destiné à un 18F252 fonctionnant à 20 Mhz. Nous avons adapté le programme assembleur du bootloader à un certain nombre de configuration type des processeurs que nous utilisons, les différents fichiers HEX ainsi que le source les accompagnant peuvent être téléchargés sur notre site à l'adresse suivante:

www.pitect.org/kitdev/tiny/modules/

Il s'agit ensuite de choisir le processeur que vous utilisez ainsi que la fréquence d'horloge à laquelle il fonctionne.

Une fois le module logiciel sélectionné, il suffit de programmer le processeur avec ce fichier à l'aide de votre programmeur habituel.

Note aux utilisateurs de notre kit de développement

Notre kit de développement est livré avec un PIC 18F252 ou 18LF252 préprogrammé, vous n'aurez donc pas à programmer le processeur avant utilisation. Il est tout de même recommandé de disposer d'un programmeur de PIC et de mettre en oeuvre les conseils de ce chapitre.

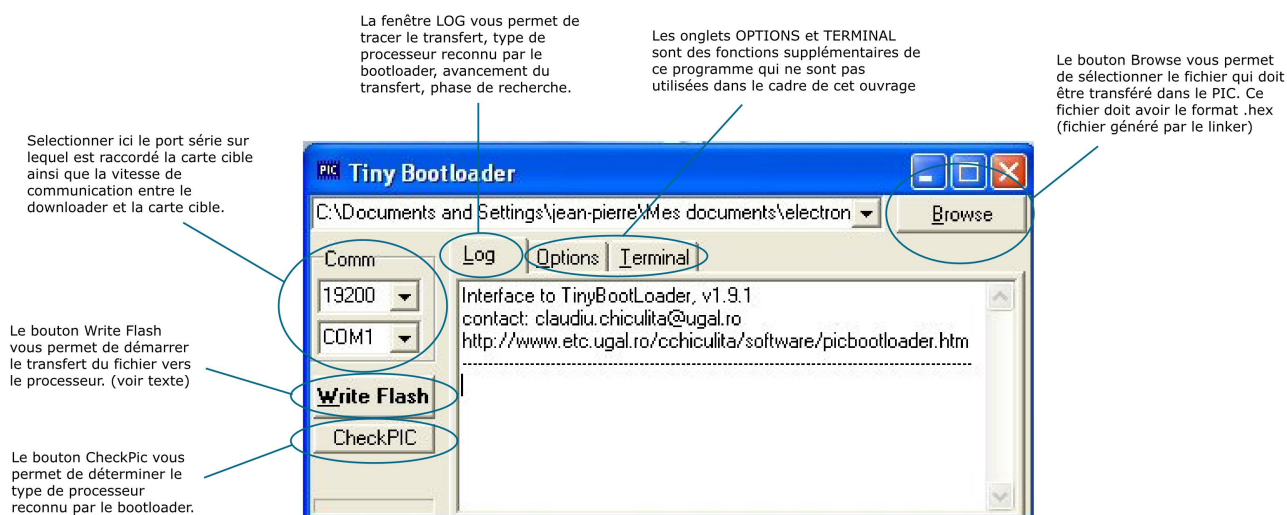
Le module downloader

Nous allons maintenant tester le module de transfert du programme du PC vers le processeur. Pour cela, vous devrez disposer d'une platine de test avec le PIC et la circuiterie indispensable (alimentation, reset, quartz) ainsi que d'un convertisseur RS232/TTL raccordé sur un port série de votre PC.

Note aux utilisateurs de notre kit de développement

Les utilisateurs de notre kit de développement disposent de l'ensemble de ces éléments sur la platine de développement livrée avec le kit. Pour tester le fonctionnement du downloader, placer la platine du PIC sur la plaque de test, raccorder un câble droit entre un port com de votre PC et la platine et alimenter la platine de test avec un adaptateur de 9 à 16V. La led présence tension de la platine s'allume.

Une fois votre montage sous tension, lancer le programme [tinybldWin.exe](#) qui se trouve dans le dossier c:\bootloader\



Il s'agit maintenant de tester le fonctionnement du bootloader. Pour cela, nous mettons à votre disposition un fichier .HEX qui pourra être transféré dans le PIC et ce quel que soit le modèle. Ce fichier test.hex peut être téléchargé sur notre site à l'adresse suivante:

www.pictec.org/kitdev/tiny/test/

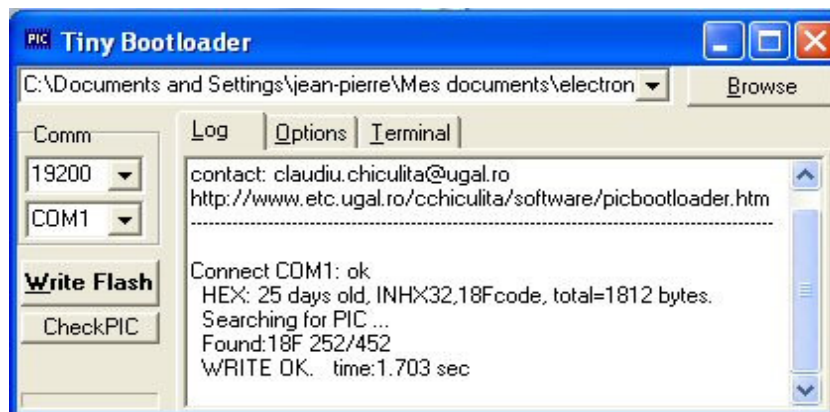
Enregistrez le fichier dans votre dossier 'Mes documents' puis sélectionnez le à l'aide du bouton Browse du programme Tiny Bootloader. Sélectionnez le port COM sur lequel est raccordé votre application puis la vitesse de transfert (**19200 bauds pour les utilisateurs de notre kit de développement**).

Pour démarrer le transfert, la chronologie des opérations est la suivante:

- Faire un reset du PIC,
- dans les deux secondes qui suivent, cliquer sur le bouton Write Flash.

Le reset du PIC peut être réalisé de deux façons, soit en interrompant l'alimentation, soit, si votre carte est équipée, en actionnant le bouton Reset. C'est cette solution qui a été retenue sur notre carte de développement.

Si tout se passe correctement, vous obtiendrez l'écran suivant:



Dans le cas contraire, vous obtiendrez l'écran suivant:



Dans la plupart des cas, cet écran est obtenu lorsque le temps entre le reset et le click sur le bouton Write Flash est trop long. Tout ceci bien sûr à condition que vous soyez sûrs de votre carte cible et du module que vous avez programmé dans le PIC. Pour les utilisateurs de notre kit de développement, il faut savoir que toutes nos cartes sont testées avant l'envoi, il n'y a donc pas de doute sur le fonctionnement de la liaison entre le PIC et le PC.

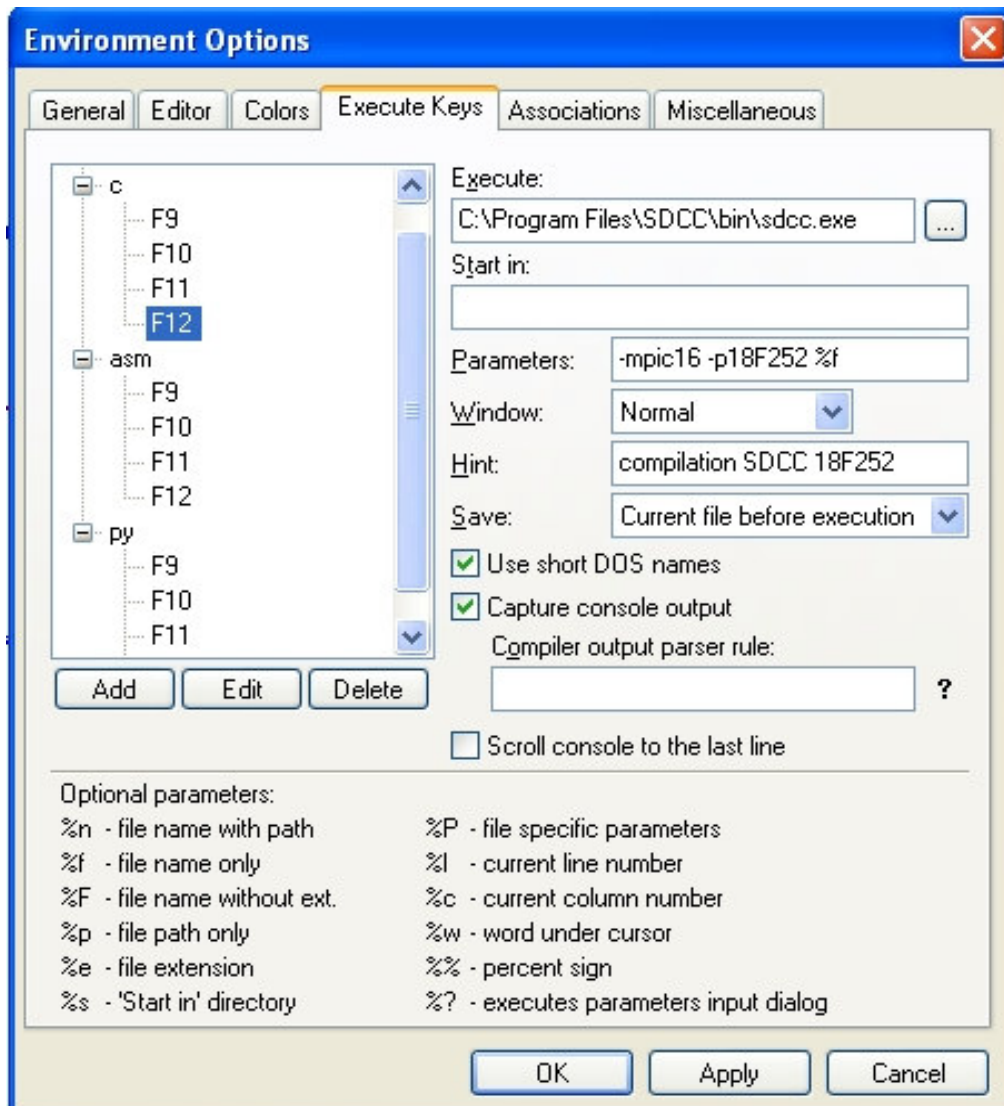
Une fois familier du fonctionnement de ce bootloader, vous apprécierez sa facilité d'utilisation et le temps gagné par rapport à l'utilisation d'un programmeur standard. Les fonctions annexes de terminal RS232 que nous n'utiliserons pas dans le cadre de cet ouvrage vous serviront probablement pour la mise au point d'applications utilisant l'UART du PIC (équivalent du port série d'un PC).

Configurer Context

Les touches utilisateurs permettent à Context d'être pratiquement un IDE. Ces touches permettent de compiler le programme C écrit avec SDCC sans sortir de l'éditeur. Pour programmer les touches utilisateur, allez dans le menu **Options, Options d'environnement** puis cliquez sur l'onglet **Touches d'exécution**. Nous allons affecter une touches d'exécution permettant d'invoquer le SDCC quand un programme portant l'extension .C est ouvert dans l'éditeur.

Dans la partie gauche de la fenêtre, cliquez sur le bouton Ajouter puis taper C dans la fenêtre texte. Nous venons d'affecter 4 touches utilisateurs à tous les sources portant l'extension .c ouverts dans l'éditeur de texte.

Ces 4 touches peuvent soit être invoquées par les touches de fonction (F9,F10,F11,F12) soit en cliquant sur les touches utilisateurs dans le fenêtre principale. Ces touches sont numérotées de 1 à 4.
Cliquez maintenant sur F12 et complétez la fenêtre comme indiqué ci-dessous:



Cliquez ensuite sur OK, Context vous permettra désormais de compiler en C avec SDCC.

Nous allons faire un test avec une application toute simple.

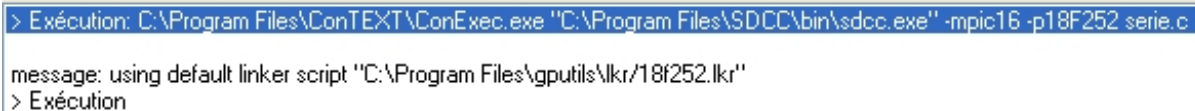
```
#define __18F252
#include <pic18F252.h>
#pragma stack 0x200 100
unsigned int i;
void main(void) {
```

```

TRISB = 0;
while(1)
{
    PORTBbits.RB7=1;
    for (i=1;i<1000;i++);
    PORTBbits.RB7=0;
    for (i=1;i<1000;i++);
}
}

```

Après avoir recopié ce source dans Context, cliquez sur la touche utilisateur N°4 ou sur F12. Voici le résultat que vous obtiendrez dans la partie basse de l'écran de Context.



```

> Exécution: C:\Program Files\ConTEXT\ConExec.exe "C:\Program Files\SDCC\bin\sdcc.exe" -mpic16 -p18F252 serie.c
message: using default linker script "C:\Program Files\gputils\lkr\18f252.lkr"
> Exécution

```

Cet écran vous indique que la compilation a réussi. Vous trouverez dans le dossier dans lequel vous avez sauvegardé ce fichier de test un fichier .hex qui peut être directement téléchargé dans le PIC.

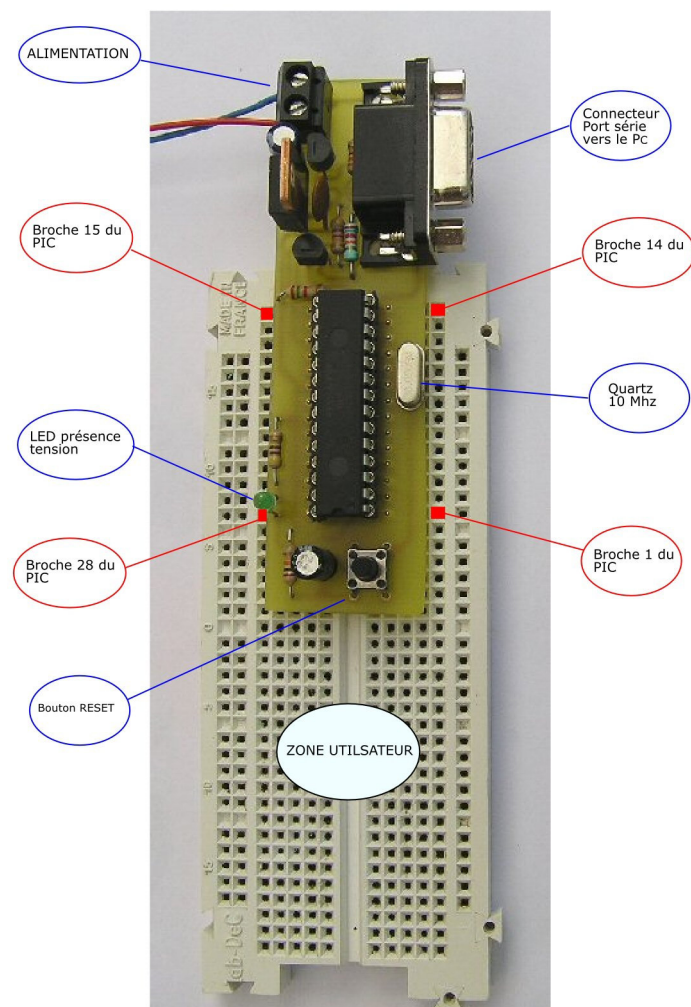
Troisième partie: Le matériel

La platine de développement

La platine que nous utiliserons dans le cadre de cet ouvrage intègre l'ensemble des composants 'minimum' permettant de faire fonctionner un PIC. Ses caractéristiques sont les suivantes:

- alimentation de 9 à 20 Volts continu
- Port de communication RS232
- Fréquence de l'oscillateur 10 Mhz
- circuit de reset avec bouton

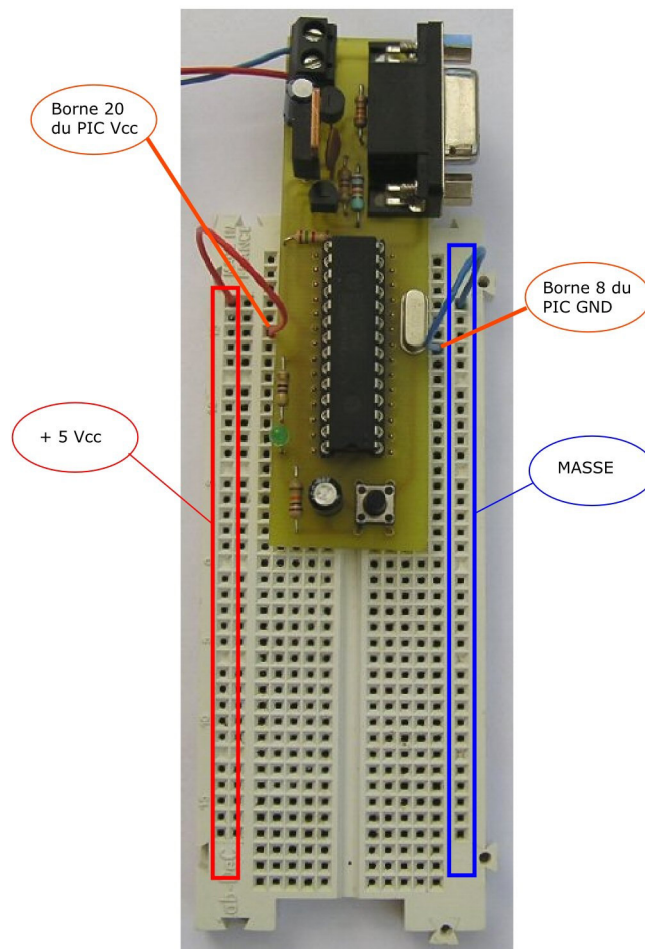
Le PIC est alimenté en 5 Volts par l'intermédiaire d'un régulateur 7805, l'adaptation des niveaux RS232 vers le PIC est réalisée à l'aide de deux transistors. La fréquence de l'oscillateur a été choisie pour permettre, à terme, de faire fonctionner le PIC à 40 Mhz.



VUE DE LA PLATINE

On commencera par insérer la platine de développement sur la plaque de test en prenant bien soin de placer la broche 14 du PIC sur la première rangée de la platine comme indiqué sur la photo ci-dessus. La platine sera centrée sur la plaque test de façon à laisser un trou libre de chaque côté.

Les rangées situées de chaque côté de la platine seront utilisées comme zone d'alimentation, le +5 Vcc à gauche, la masse à droite. L'alimentation 5 V sera récupérée sur les bornes du PIC, la masse étant connectée à la broche 8, le + 5 V est connecté à la broche 20. Raccorder les broches comme indiqué sur la photo ci-dessous.



On dispose maintenant de deux rangées de connections de part et d'autre de la plaque test, une rangée reliée à la masse, une rangée reliée au + 5 Vcc.

Ceci nous permettra d'alimenter les composants de la zone utilisateur.

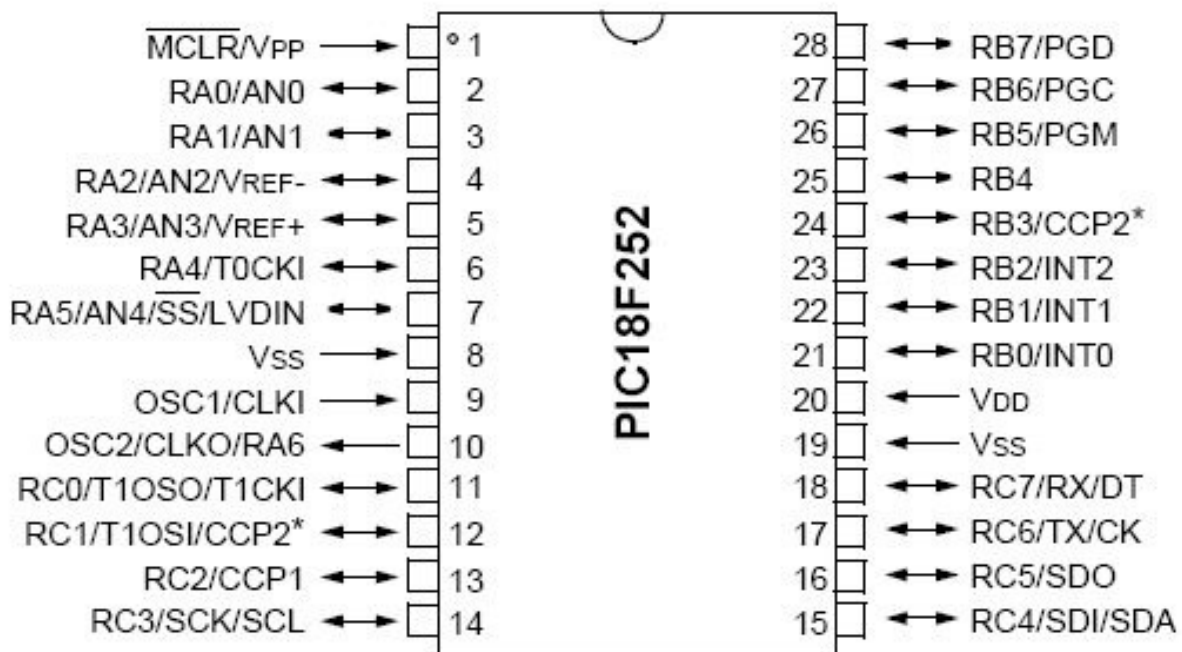
Votre platine de développement est maintenant prête à recevoir votre première application programmée avec SDCC. Il convient toutefois, au préalable de la réalisation de notre premier programme d'analyser le brochage du PIC que nous allons utiliser et de définir ses caractéristiques principales.

Les informations données dans le chapitre suivant sont valables pour tous les PIC de la série 18F et pour la plupart aux PIC de la série 16F. L'architecture des microcontrôleurs MICROCHIP est toujours à peu près identiques.

Un aperçu du 18F252

Le brochage du 18F252 peut être trouvé dans la documentation MICROCHIP sur le site du constructeur:

www.microchip.com



Nous allons voir, grâce au schéma ci-dessous comment peut être décrite l'architecture de ce microcontrôleur. Dans un premier temps, nous nous limiterons aux fonctions de base du 18F252, essentiellement les ports d'entrées sorties et leur utilisation. Dans la deuxième partie de cet ouvrage, nous verrons comment utiliser quelques fonctions spécifiques du 18F252.

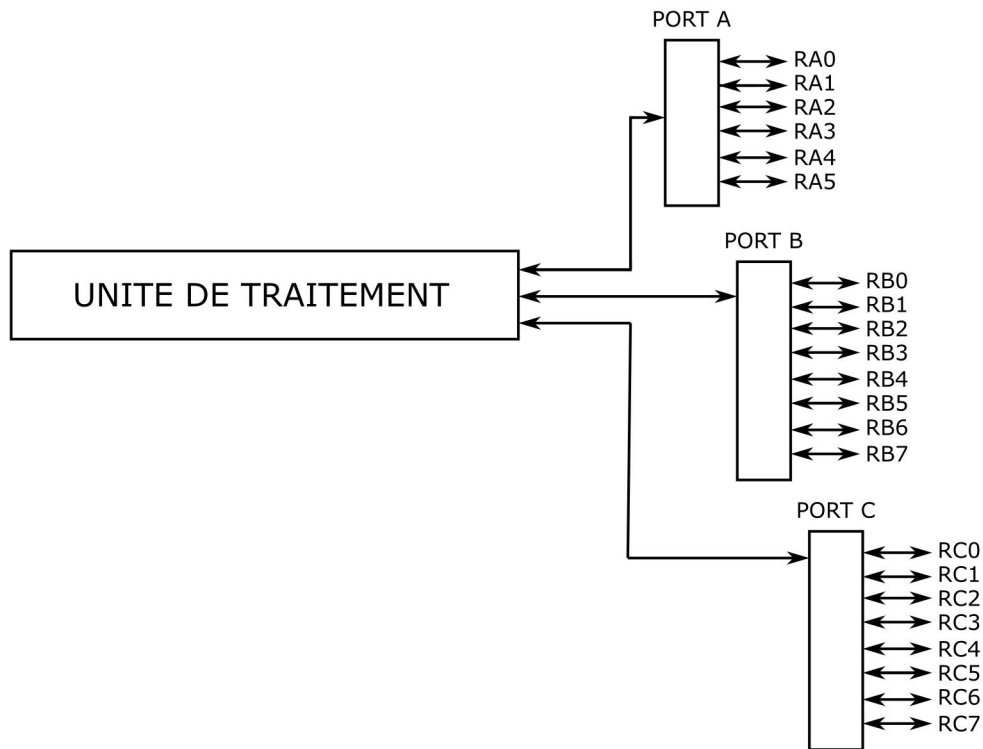
Le 18F252, dans ses fonctions de base, est constitué d'une unité de traitement (celle dans laquelle nous allons transférer notre programme) et d'une unité de commande (celle qui permet de raccorder des éléments extérieurs). Cette unité de commande est composée de 3 ports, les ports sont des ensembles d'entrées ou de sorties qui permettent d'amener à l'unité de traitement des informations extérieures ou de commander des dispositifs. Généralement, les ports sont composés de 8 bornes qui peuvent être configurées indépendamment en entrées (permettant de recevoir des informations) ou en sorties (permettant de transmettre des informations). L'unité de traitement traite ces informations en fonction du programme que vous aurez écrit.

Nous allons voir dans le premier exemple comment configurer une ligne de port en

sortie, raccorder sur cette ligne une led et la faire clignoter.

Il s'agit d'une fonction très simple qui n'a pas besoin d'acquérir des informations de l'extérieur (c'est l'unité de traitement qui va définir les périodes d'allumage et d'extinction) mais cet exemple va nous permettre de définir les bases d'un programme écrit en C et compilé avec SDCC.

ARCHITECTURE SIMPLIFIEE DU 18F252



On remarque que le port A du 18F252 n'a que 6 entrées/sorties (RA0 à RA5). Les ports B et C ont 8 entrées/sorties.

NOTA

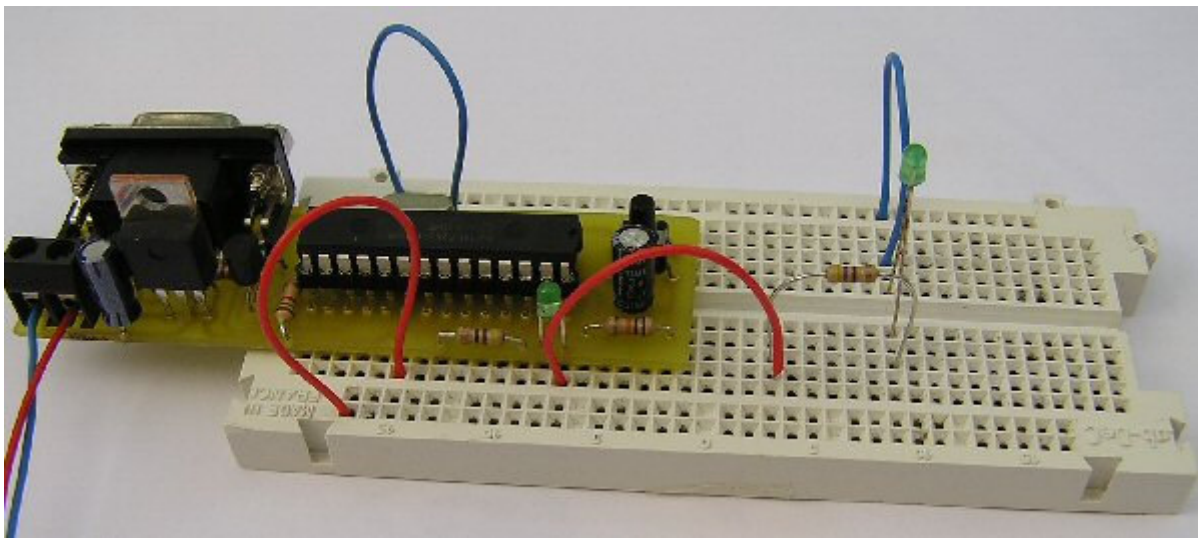
Dans notre platine de développement, les bornes RC6 et RC7 sont utilisées pour le transfert du programme avec le bootloader, on évitera donc d'utiliser ces broches dans nos montages.

Première application: Une led clignote !!

Voilà une application qui ne fera rêver personne !! Toutefois il s'agit de l'exemple le plus simple à réaliser qui va nous permettre de définir les bases de la programmation en C sur PIC. Outre le fait que l'architecture d'un programme est toujours identique quelle que soit la complexité du système, nous allons voir dans ce chapitre qu'il est possible d'écrire un programme de multiples façons pour obtenir le même résultat. Il y a tout de même des moyens d'écrire un code plus compact et plus lisible en utilisant les outils du langage C. C'est cet aspect que nous privilégierons dans cet exemple.

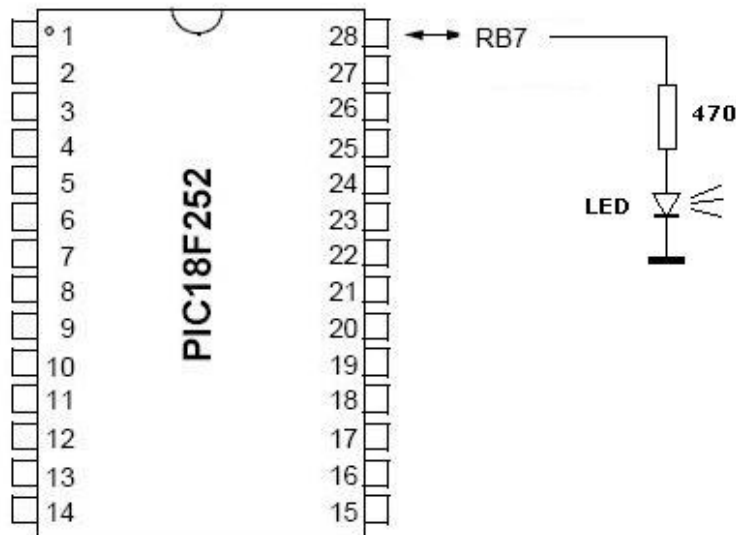
Le montage

Une résistance de 470 ohms et une led suffiront pour réaliser ce montage.



La led est câblée sur le port RB7 du PIC. Veillez à respecter la polarité de la LED, anode coté résistance (la patte la plus longue), cathode coté masse.

Voici le schéma de principe du montage ainsi réalisé.



Le programme

Dans le dossier '**Projet**' créez un nouveau dossier que vous nommerez '**led**', ce sera le dossier de ce projet. Lancez **Context** et créez un nouveau fichier puis enregistrez le tout de suite dans le dossier **C:\Projet\led** en lui donnant l'extension **.C**

Nous allons commencer par commenter le source du programme pour indiquer:

- le nom du programme,
- l'auteur du programme,
- la cible (type du processeur et fréquence),
- la date de la dernière version.

Il ne s'agit pas la d'une coquetterie mais juste d'un moyen de retrouver dans 6 mois pour quel PIC vous avez écrit ce programme. Croyez moi, lorsque l'on commence à utiliser de manière régulière les PIC il est indispensable d'avoir un minimum d'organisation.

Ces remarques vont figurer dans le programme sous forme de commentaire. En C il y a deux façons d'écrire les commentaires, sur une seule ligne et n'importe où dans la ligne en faisant précéder le commentaire de **//**, lorsqu'il s'agit d'un bloc de commentaires écrits sur plusieurs lignes, le début de la zone commentaire est signalé par **/*** la fin de la zone commentaire est signalée par ***/**

Le compilateur n'interprétera pas tout ce qui est écrit dans cette zone.

```
// clignotant à LED sur port RB7
// Jean-Pierre MANDON 2006
// Pic 18F252 à 10 Mhz
// version du 04/08/2006
```

Ceci est la première façon d'écrire des commentaires sur plusieurs lignes, nous aurions pu écrire cette zone de commentaires de la façon suivante:

```
/* clignotant à LED sur port RB7
   Jean-Pierre MANDON 2006
   Pic 18F252 à 10 Mhz
```

On remarque que Context gère la colorisation syntaxique, de ce fait les commentaires apparaîtront toujours en gris dans le source.

Nous allons maintenant insérer les lignes indiquant au compilateur le type de processeur utilisé. Dans notre cas, il s'agit d'un 18F252, les lignes suivantes devront être insérées:

```
#define __18F252
#include <pic18F252.h>
```

La première directive **#define** indique au compilateur que le processeur utilisé est un 18F252, la directive **#include** permet d'inclure à notre projet le fichier de définition du 18F252 qui se trouve à l'emplacement suivant: **C:\program files\SDCC\include\pic16**

La colorisation syntaxique ici encore est définie en vert pour les directives.

Nous allons maintenant écrire le corps du programme. Un programme en langage C est toujours composé d'une boucle **main** (principal). Le programme démarre toujours au début du **main**. Tous les autres modules que vous trouverez dans un programme en langage C ne sont que des fonctions appelées par le **main** ou des bibliothèques.

Nous allons écrire le début de la boucle **main** sans nous préoccuper pour le moment de ce qui est écrit autour (void). Nous verrons la signification de ces termes plus loin dans cet ouvrage.

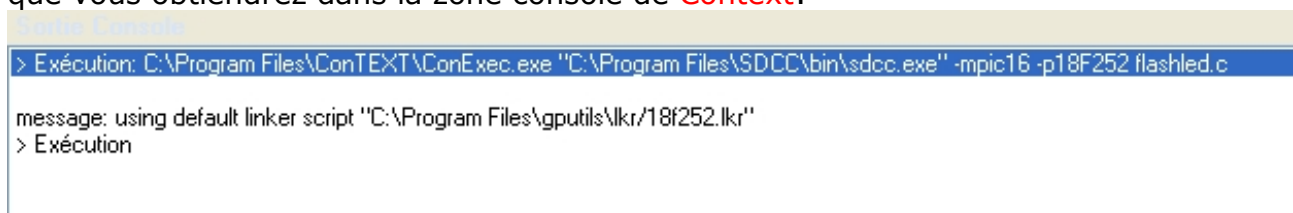
```
void main(void) {
```

Le corps principal du programme est maintenant défini. L'accolade ouverte indique le début du **main**, en fin du **main** (donc en fin du programme) il faudra placer une accolade fermée qui indiquera la fin de la boucle **main**.

Essayons tout de suite de terminer ce programme, pour cela placer une accolade fermée sur la ligne suivante.

```
}
```

et maintenant compilez ce programme en cliquant sur la touche utilisateur 4. Voici ce que vous obtiendrez dans la zone console de **Context**:



```
> Exécution: C:\Program Files\ConTEXT\ConExec.exe "C:\Program Files\SDCC\bin\sdcc.exe" -mpic16 -p18F252 flashled.c
message: using default linker script "C:\Program Files\gputils\lkr\18f252.lkr"
> Exécution
```

Votre programme a été compilé sans erreur, la syntaxe est donc correcte. Même si ce programme ne fait rien il nous permet de voir que les directives sont correctement définies et que le corps du programme est juste (présence de la boucle **main**). Un programme en langage C ne peut être compilé si il ne comporte pas de boucle **main**.

Voici l'ensemble du programme qui ne fait rien (!!)

```
// clignotant à LED sur port RB7
// Jean-Pierre MANDON 2006
// Pic 18F252 à 10 Mhz
// version du 04/08/2006
```

```
#define __18F252
#include <pic18F252.h>
```

```
void main(void) {  
}
```

Nous allons maintenant ajouter ce qui est nécessaire pour faire clignoter la led. Pour cela il y a plusieurs action à exécuter:

1. indiquer à l'unité de traitement du PIC que RB7 est utilisée en sortie,
2. éteindre la led,
3. attendre un moment,
4. allumer la led,
5. attendre un moment,
6. retourner en (2)

Pour indiquer au PIC que RB7 est utilisé en sortie nous allons utiliser le registre TRISB. Ce registre est le registre de direction du PORT B, il existe le même registre pour le port C (TRISC) et pour le port A (TRISA).

Le registre TRISB est composé de 8 bits correspondants au 8 lignes du port B, le bit 0 correspond à la ligne RB0, le bit 1 à la ligne RB1 etc...

Pour indiquer qu'une ligne est utilisée en sortie, il suffit de mettre le bit correspondant du registre TRISB à 0, si la ligne est utilisée en entrée, le bit correspondant du registre de direction sera mis à 1. Dans notre cas, la ligne RB7 étant utilisée en sortie, le bit 7 du registre TRISB sera mis à 0. Pour cela, nous allons écrire:

```
TRISBbits.TRISB7=0;
```

Nous allons ensuite mettre à 0 la ligne RB7 pour éteindre la led. En effet, l'état des lignes d'entrées sorties est indéfini au démarrage du PIC, il convient toujours d'initialiser les états dans le début de votre programme.

```
PORTBbits.RB7=0;
```

Avec ces deux instructions nous avons initialisé le PIC il s'agit maintenant d'écrire la partie exécution du programme.

Nous avons vu que les étapes 2 à 6 de notre programme se renouvelaient indéfiniment, nous allons donc insérer une boucle qui s'exécutera de manière permanente:

```
while (1)    {  
            }
```

L'instruction **while** (tant que) exécute le corps de programme situé entre les accolades tant que la condition définie entre parenthèses est vraie. Nous avons placé la valeur 1 entre parenthèses, la valeur 1 est l'équivalent de condition vraie, donc notre boucle **while** sera exécutée de manière permanente par le PIC.

Il faut maintenant allumer la led, ceci équivaut à mettre la ligne RB7 à 1.

```
PORTBbits.RB7=1;
```

La led est maintenant allumée. Nous devons maintenant attendre un moment avant de l'éteindre. Pour cela nous allons créer une boucle compteur, il s'agit d'une variable qui va être incrémentée un certain nombre de fois, le but étant de perdre un peu de

temps mais sans avoir de tâche particulière à réaliser.

```
for (i=0;i<10000;i++) { };
```

La boucle for compte de 0 à 9999 et exécute le corps de programme situé entre les accolades. Comme il n'y a rien entre les accolades, aucune action n'est exécutée. Le but est juste de perdre un peu de temps. Ce temps correspondra à la période d'allumage de notre led. Nous avons volontairement placé des accolades mais si aucune action ne doit être exécutée, elles peuvent être omises. On écrira donc plus simplement:

```
for (i=0;i<10000;i++);
```

Il faut maintenant éteindre la led, ceci équivaut à mettre la ligne RB7 à 0.

```
PORTBbits.RB7=0;
```

et ensuite perdre de nouveau un peu de temps.

```
for (i=0;i<10000;i++);
```

Notre programme est juste à une exception près, nous avons utilisé pour notre compteur dans la boucle for une variable i mais nous n'avons pas déclaré cette variable. En effet, chaque fois qu'une variable est utilisée dans un programme, le compilateur doit lui réserver une place mémoire. Pour que cette réservation soit faite il faut déclarer la variable en tête de programme. Nous allons déclarer que cette variable est un entier.

Voici le résumé du programme de pilotage de notre led:

```
// clignotant à LED sur port RB7
// Jean-Pierre MANDON 2006
// Pic 18F252 à 10 Mhz
// version du 04/08/2006

#define __18F252
#include <pic18F252.h>

void main(void) {
    int i;                                // déclaration de la variable i

    TRISBbits.TRISB7=0;                  // RB7 est utilisé en sortie
    PORTBbits.RB7=0;                     // RB7 est mis à 0 (led éteinte)

    while (1) {                          // exécuter toujours
        PORTBbits.RB7=1;                 // RB7=1 (led allumée)
        for (i=0;i<10000;i++);          // faire 10000 fois rien
        PORTBbits.RB7=0;                 // RB7=0 (led éteinte)
        for (i=0;i<10000;i++);          // faire 10000 fois rien
    }                                    // retourner (while)
}
```

Compilez ce programme puis transférez le dans le 18F252 à l'aide du bootloader. La

led clignote à une vitesse très rapide. Testez maintenant le même programme en remplaçant la valeur 10000 dans les boucles for par une valeur de 30000. La vitesse de clignotement est beaucoup moins élevée.

Testez ensuite des valeurs différentes pour la 1ère boucle for et pour la seconde boucle. On voit que le rapport cyclique change (temps led allumée supérieur au temps led éteinte). Si vous dépassez la valeur fatidique de 32767 le compilateur va générer une erreur, en effet une variable entière est comprise entre -32767 et +32767, notre variable i ne peut donc varier qu'à l'intérieur de ces limites. Comment faire alors pour obtenir des délais plus longs ?

Il suffit d'imbriquer deux boucles à l'aide d'une autre variable:

```
for (j=0;j<10;j++) { for (i=0;i<30000;i++) };
```

ou plus simplement:

```
for (j=0;j<10;j++) for (i=0;i<30000;i++);
```

on obtiendra un temps environ 10 fois supérieur. Il faudra veiller à déclarer la variable j en tête de programme.

```
int i,j;
```

Testez de nouveau le programme en imbriquant les deux boucles.

Résumé

- les commentaires sont précédés du signe // ou des signes de début /* et de fin */
- la définition du processeur utilisé est obtenue en utilisant la directive **#define**
 - **#define __18F252** // pour un 18F252
 - **#define __18F458** // pour un 18F458
- le fichier de définition du processeur doit être inclus **#include <18F252.h>**
- Un programme en langage C est toujours composée d'une boucle principale main
- Les registres de direction des ports d'entrées sorties des PIC sont TRISA, TRISB et TRISC (pour le 18F252),
- les registres d'entrées sorties sont PORTA, PORTB et PORTC,
- les variables doivent être déclarées en tête de programme,
- une variable de type entier peut varier de -32767 à +32767,
- une boucle while (condition) est exécutée tant que condition est vraie,
- si condition est la valeur 1, condition est toujours vraie,

- une boucle for à la forme `for (i=0;i<10000;i++)`
 - `i=0` est la valeur de départ,
 - `i<10000` est la condition d'arrêt,
 - `i++` incrémente la variable de 1 à chaque passage.

Optimisation

Nous allons voir ici comment optimiser un peu notre programme, tant sur le plan de la lisibilité que sur celui de la taille du code. Notre led est connectée sur RB7, pour l'allumer nous avons utilisé l'instruction suivante:

```
PORTBbits.RB7=1;
```

Cette forme d'écriture est un peu lourde et difficile à lire. Nous allons définir un nom pour notre led et indiquer au compilateur que chaque fois qu'il trouvera ce nom dans le programme il devra le remplacer par `PORTBbits.RB7`.

Pour cela nous allons utiliser la directive `#define`:

```
#define led PORTBbits.RB7
```

et dans la suite du programme nous pourrons écrire:

```
led=1;
```

OU

```
led=0;
```

Maintenant voyons l'aspect fonctionnel de notre programme, la led prend successivement les états 1 ou 0. La valeur suivante de l'état de notre led est donc une fonction ou exclusif de l'ancien état. Nous allons réduire la taille du code en utilisant une fonction Ou exclusif, en langage C cette fonction est notée comme ci-dessous:

```
i = i^1;
```

La boucle while de notre programme devient alors:

```
while (1)
{
    led=led^1;
    for (i=0;i<30000;i++);
}
```

Le programme complet et optimisé devient:

```
// clignotant à LED sur port RB7
// Jean-Pierre MANDON 2006
// Pic 18F252 à 10 Mhz
// version du 04/08/2006
```

```
#define __18F252
```

```
#include <pic18F252.h>
```

```
#define led PORTBbits.RB7
```

```
void main(void) {
```

```
int i;
```

```
TRISBbits.TRISB7=0;
```

```
PORTBbits.RB7=0;
```

```
while (1)    {  
    led=led^1;  
    for (i=0;i<32767;i++);  
}
```

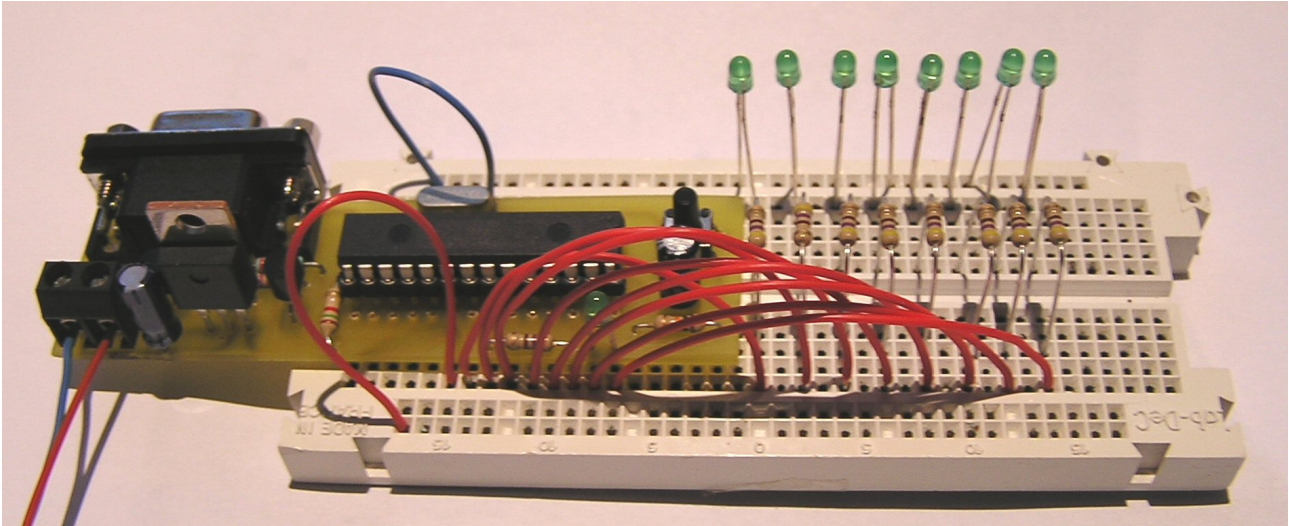
Cette forme est plus facile à lire et optimisée en taille de code.

Deuxième application: Un chenillard

Ce chenillard va nous permettre de manipuler les ports d'entrées sorties et de découvrir les fonctions mathématiques et de rotation de bits en C.

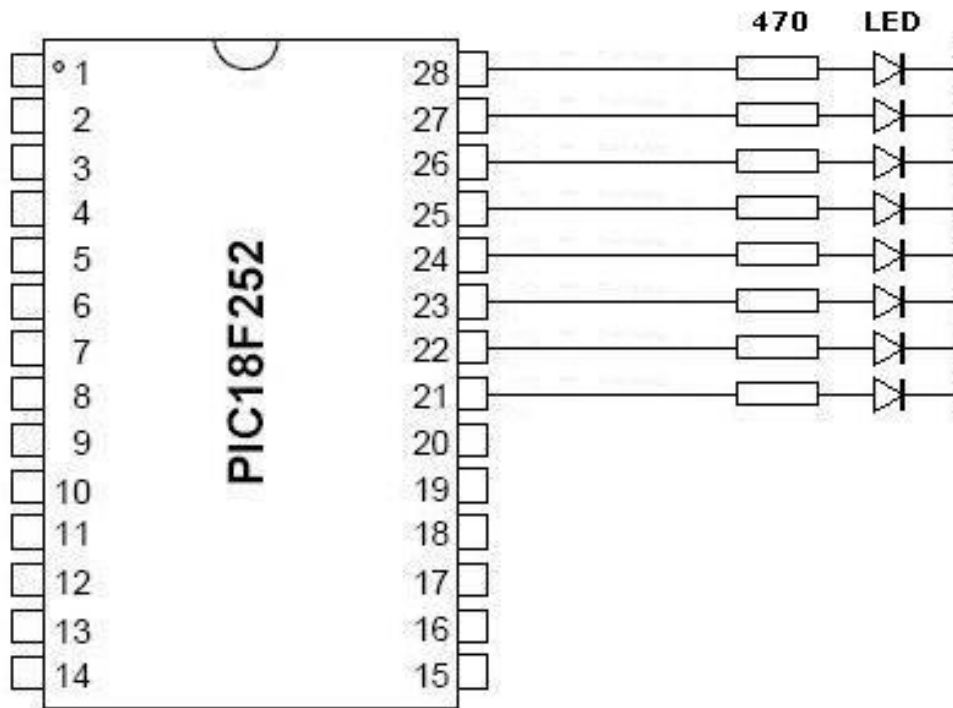
Le montage

Huit résistances de 470 ohms et huit leds sont nécessaires pour réaliser ce montage.



Les leds sont câblées sur les ports RB0 à RB7 du PIC.

Voici le schéma de principe du montage ainsi réalisé.



Le programme

Dans le dossier '**Projet**' créez un nouveau dossier que vous nommerez '**chenillard**', ce sera le dossier de ce projet. Lancez **Context** et créez un nouveau fichier puis enregistrez le tout de suite dans le dossier **C:\Projet\chenillard** en lui donnant l'extension **.C**

Les 8 leds étant câblées sur le PORTB, l'objectif est de faire allumer les leds successivement de la led raccordée à RB0 jusqu'à la led raccordée sur RB7. Nous avons vu que le port B était composé de 8 lignes constituant un PORT. Le registre associé à ce port s'appelle PORTB et est composé de 8 bits. Il faut donc mettre à 1 chaque bit de ce registre successivement. Pour cela deux méthodes peuvent être utilisées, la rotation de bit par la droite ou des multiplications par deux. Nous allons voir ici ces deux méthodes.

Voici le texte du programme utilisant la multiplication:

```
// chenillard à LED sur port B
// Jean-Pierre MANDON 2006
// Pic 18F252 à 10 Mhz
// version du 04/08/2006

#define __18F252
#include <pic18F252.h>

void main(void) {
    int i,valeur;

    TRISB=0;
    valeur=1;

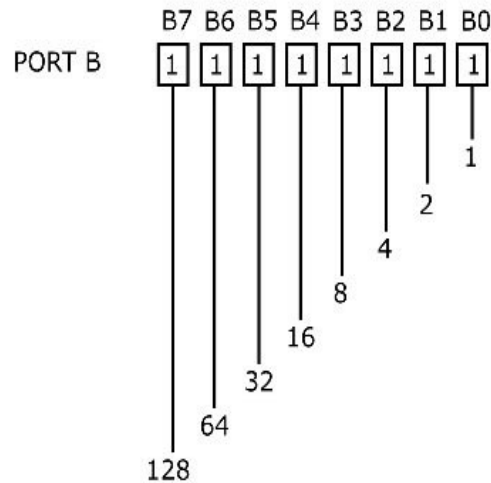
    while (1)
    {
        valeur=valeur*2;
        if (valeur>128) valeur=1;
        PORTB=valeur;
    }
}
```

```

        for (i=0;i<32767;i++);
    }
}

```

Voici les valeurs successives que doit prendre le registre PORT B pour obtenir le résultat que nous recherchons:



Si le registre a la valeur 1, le bit 0 du registre est à 1, si le registre a la valeur 2, le bit 1 du registre est à 1 etc jusqu'à 128.

Dans notre programme, la variable valeur est un entier qui prend les différentes valeurs 1,2.....128,256 est qui est copiée dans le registre PORTB. Nous avons utilisé cette variable intermédiaire pour pouvoir gérer le débordement. Ce débordement intervient quand la variable prend la valeur 256 et on remet à 1 cette variable dans ce cas. Cette manipulation ne peut être faite directement sur le registre PORTB qui a une taille de 8 bits et donc des valeurs comprises entre 0 et 255.

Une autre possibilité pour obtenir le même résultat est d'utiliser la fonction de décalage à gauche de bits. Dans ce cas le programme devient:

```

// chenillard à LED sur port B
// Jean-Pierre MANDON 2006
// Pic 18F252 à 10 Mhz
// version du 04/08/2006

#define __18F252
#include <pic18F252.h>

void main(void) {
    int i;

    TRISB=0;
    PORTB=1;

    while (1)
    {
        for (i=0;i<32767;i++);
        PORTB=PORTB<<1;
        if (PORTB==0) PORTB=1;
    }
}

```

La fonction de décalage à gauche est représentée par le signe << , le nombre suivant ce signe représente la taille du décalage, 1 indique un décalage de 1 bit, 2 de 2 bits

etc.

La même fonction existe pour les décalages à droite, elle est représentée par le signe `>>`.

Lorsque 8 décalages sont réalisés, le registre PORTB a alors la valeur 0, on le réinitialise à 1 pour redémarrer le comptage au début.

Nous avons introduit ici l'instruction de comparaison if, sa forme est:

```
if (condition)    {  
                  }
```

La condition figurant entre parenthèses est une condition binaire qui peut être égale à 1 ou à 0. Ainsi:

if (1) est toujours vraie

if (0) est toujours faux

Dans notre cas, la condition est `PORTB==0` qui signifie 'est-ce que le registre PORTB est égal à 0'. A noter la différence entre l'opérateur d'affectation `PORTB=0` et l'opérateur de comparaison `PORTB==0`, cette notation est caractéristique du langage C et des langages dérivés du C.

Testez maintenant le programme en remplaçant la ligne:

```
PORTB=PORTB<<1;
```

par

```
PORTB=PORTB<<2;
```

Le résultat obtenu, allumage successif des leds 1,3,5,7 illustre le décalage de deux bits vers la gauche.

Testez maintenant le programme en remplaçant la ligne:

```
if (PORTB==0) PORTB=1;
```

par

```
if (PORTB==16) PORTB=1;
```

Le résultat obtenu, remise à zéro du chenillard après l'allumage de la 4ème led montre comment évolue la valeur du registre PORTB.

Résumé

-Une opération de comparaison est réalisée avec la fonction if

La forme est `if (condition) { }`

Le bloc placé entre accolades est exécuté si la condition est vraie.

- La comparaison de deux valeurs en C est réalisée avec l'opérateur `==`

`if (PORTB==0) { }`

- L'affectation d'une valeur est réalisée avec l'opérateur `=`

`PORTB=0;`

- Un décalage est réalisé à l'aide de la fonction `>>` pour les décalages à droite

`<<` pour les décalages à gauche

Le nombre suivant l'opérateur indique le nombre de décalage.

- Une rotation de un bit vers la gauche est l'équivalent d'une multiplication par 2

- Une rotation de un bit vers la droite est l'équivalent d'une division par 2
- Une multiplication est l'équivalent d'une rotation de un bit vers la gauche
- Une division est l'équivalent d'une rotation de un bit vers la droite

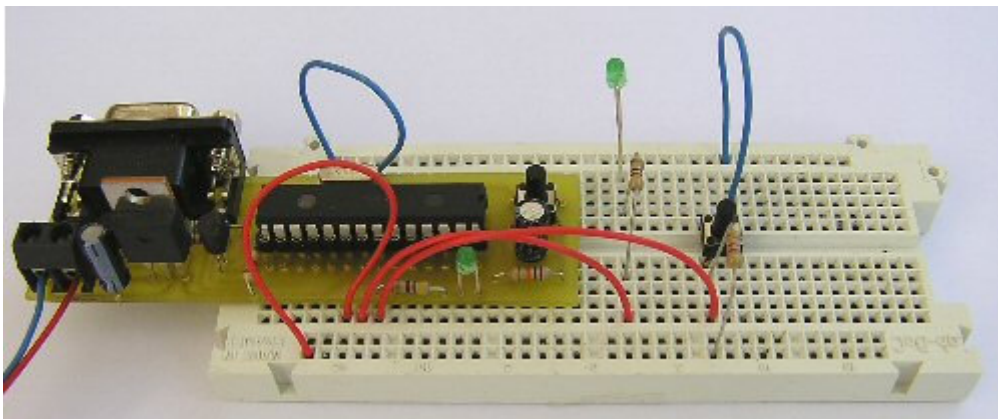
Troisième application: Un télérupteur

La fonction télérupteur est une fonction mémoire simple qui va nous permettre de mettre en oeuvre les entrées et de voir comment tester un poussoir mécanique.

Le montage

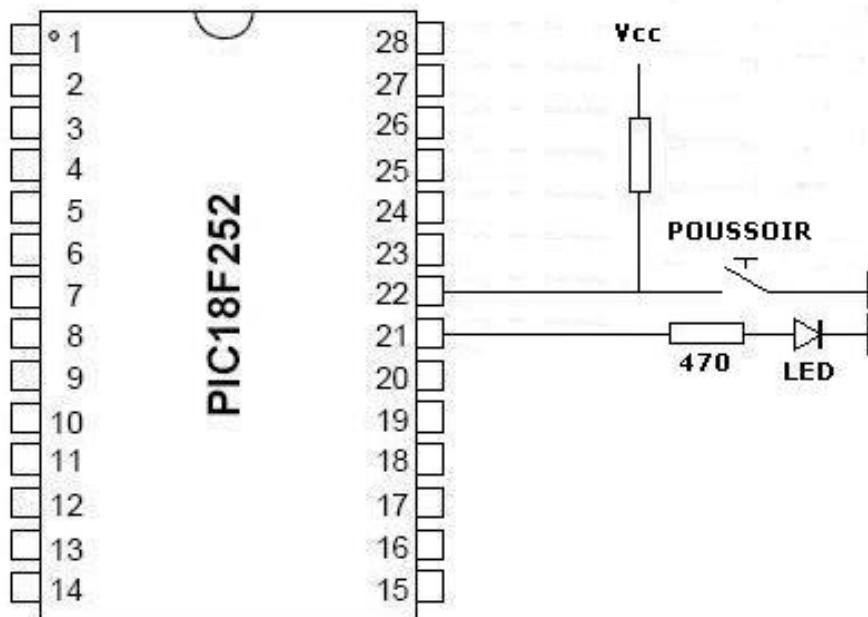
Pour réaliser ce montage, vous aurez besoin des composants suivants:

- une led
- une résistance de 470 ohms
- une résistance de 4,7 kohms
- un poussoir (celui que nous avons utilisé a été récupéré sur un magnétoscope).



La led est câblée sur le port RB0 du PIC, le bouton poussoir est câblé sur le port RB1 avec une résistance de pull up de 4,7 kohms.

Voici le schéma de principe du montage ainsi réalisé.



Le programme

Dans le dossier '**Projet**' créez un nouveau dossier que vous nommerez '**telerupteur**', ce sera le dossier de ce projet. Lancez **Context** et créez un nouveau fichier puis enregistrez le tout de suite dans le dossier **C:\Projet\telerupteur** en lui donnant l'extension **.C**

L'objectif de ce programme est de réaliser le fonctionnement tel qu'il est décrit ci-dessous:

- Si la led est allumée, un appui sur le poussoir éteint la led,
- si la led est éteinte, un appui sur le bouton poussoir l'allume.

Il convient donc:

- de détecter une action sur le bouton poussoir,
- de vérifier quel est l'état du port associé à la led,
- d'inverser cet état.

Le poussoir étant raccordé sur RB1, il faudra configurer RB1 en entrée via le registre TRISB. RB0 sera configuré en sortie (alimentation de la led).

A noter que le poussoir est câblé sur RB1 à travers une résistance de Pull-Up imposant un niveau 1 (+Vcc) lorsque le poussoir n'est pas actionné. L'entrée RB0 sera donc en permanence à 1 et passera à 0 lors d'une action sur le bouton.

Voici le programme réalisant la fonction souhaitée:

```
// Télérupteur
// Jean-Pierre MANDON 2006
// Pic 18F252 à 10 Mhz
// version 1.0 du 13/08/2006
```

```

#define __18F252
#include <pic18F252.h>

#define led PORTBbits.RB0
#define poussoir PORTBbits.RB1

void main(void) {

TRISB=0x02;
PORTB=0;

while (1)    {
                if (poussoir==0) led=led^1;
            }
}

```

Nous avons utilisé la notation hexadécimale 0x02 pour la configuration du registre TRISB. Lorsqu'une valeur est notée en hexadécimal, sa représentation doit être précédée de 0x.

Après avoir compilé ce programme et l'avoir transféré dans la platine de développement, on s'aperçoit qu'il ne réalise pas la fonction souhaitée. En effet, lorsque l'on appuie sur le poussoir, la led diminue en intensité mais ne s'éteint jamais ou très rarement. Si l'on analyse la ligne qui détecte le passage à 0 du poussoir, il ne semble pas y avoir d'erreur. En fait, le processeur étant très rapide, il exécute cette instruction environ 1 million de fois par seconde, il voit donc le poussoir à 0 un million de fois par seconde et fait changer l'état du port RB0 à ce rythme. Pour régler ce problème, nous allons insérer une boucle d'attente après le test, ceci permettra de ralentir le test et de figer l'appui sur le bouton pendant environ 20 millisecondes. Il s'agit du temps typique d'appui sur une touche.

Le programme devient:

```

// Télérupteur
// Jean-Pierre MANDON 2006
// Pic 18F252 à 10 Mhz
// version 1.1 du 13/08/2006

#define __18F252
#include <pic18F252.h>

#define led PORTBbits.RB0
#define poussoir PORTBbits.RB1

void main(void) {
int i;

TRISB=0x02;
PORTB=0;

while (1)    {
                if (poussoir==0) led=led^1;
                for (i=0; i<30000; i++);
            }
}

```

Cet exemple, réalisé avec un poussoir peut être également reproduit avec n'importe quel capteur que vous aurez choisi, capteur de présence infra-rouge, fin de course ou tout autre dispositif tout ou rien.

A noter que d'une manière générale, la commutation d'un interrupteur mécanique provoque un certain nombre de rebonds qui se traduisent par une succession de 1 et

de 0 sur l'entrée d'un processeur. L'intégration dans le programme d'une boucle de temporisation permet d'absorber ces rebonds. En contrepartie, la mise en place de cette temporisation prend du temps machine et le processeur est bloqué pendant cette phase du programme. Nous verrons dans le volume 2 de cet ouvrage d'apprentissage les techniques d'interruptions qui permettent de s'affranchir de cette perte de temps.

Il est tout de même possible d'éviter cette perte de temps de la façon suivante:

```
// Télérupteur
// Jean-Pierre MANDON 2006
// Pic 18F252 à 10 Mhz
// version 1.2 du 14/08/2006

#define __18F252
#include <pic18F252.h>

#define led PORTBbits.RB0
#define poussoir PORTBbits.RB1

void main(void) {
int i;

TRISB=0x02;
PORTB=0;

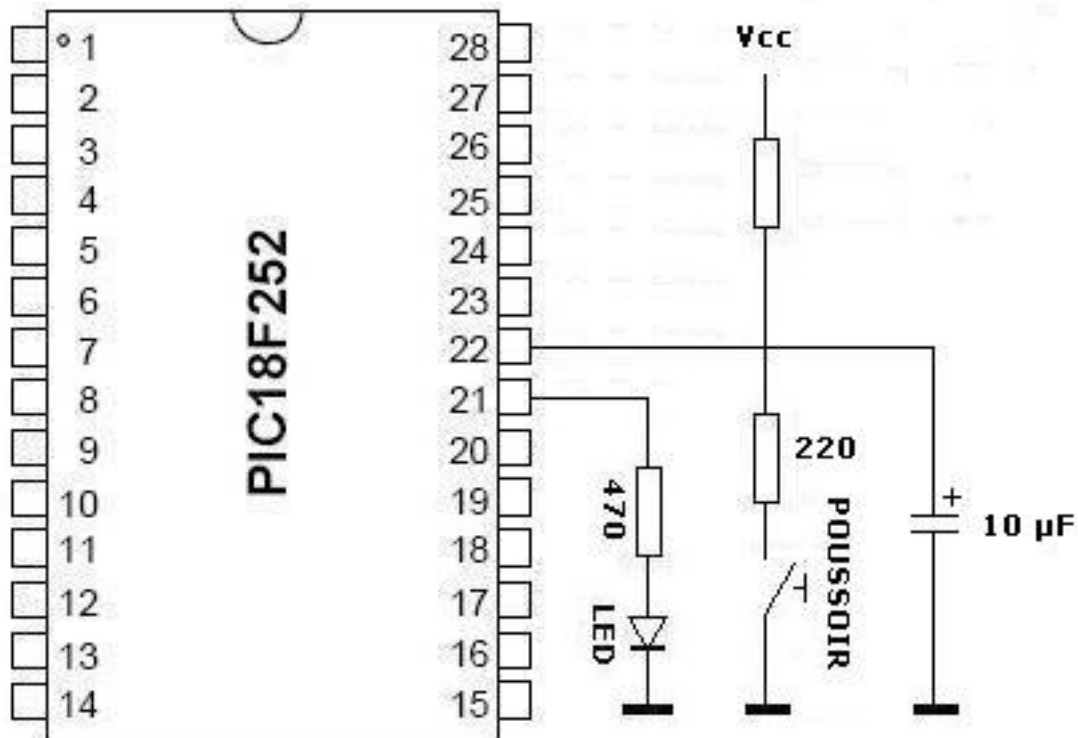
while (1)
{
if (poussoir==0) led=led^1;
while (poussoir==0);
}
}
```

Dans ce cas, on attend que l'utilisateur ait relâché le bouton pour continuer d'exécuter la suite du programme, lorsque l'on met en oeuvre cette technique, il est facile d'observer le phénomène de rebond mécanique expliqué plus haut dans ce paragraphe. Une fois sur deux, la led ne s'allume ou ne s'éteint pas.

Une autre solution pour intégrer ce rebond mécanique est de le traiter à l'extérieur du processeur par un réseau RC. Cette solution est utilisée notamment lorsqu'on a affaire à des capteurs très sensibles qui se déclenchent de manière intempestive.

Le réseau RC par R1 et C1 absorbe les baisses de tension et le niveau bas sur l'entrée du PIC n'est obtenu qu'au bout de la constante de temps du réseau RC.

$T=RC$



Pour les valeurs données dans le schéma ci-dessus, la constante de temps est de 2,2 millisecondes.

Résumé

- En C, les nombres hexadécimaux sont précédés de la notation **0x**.
- Un bouton poussoir mécanique se ferme de manière fiable au bout d'environ 20 mS, pendant ces 20 mS un phénomène de rebond mécanique existe (succession de passage à 1 et à 0).
- Un processeur exécute plusieurs milliers d'instructions par secondes.
- Un filtre électronique peut être réalisé à l'aide d'un réseau RC pour absorber les variations de tensions rapides.

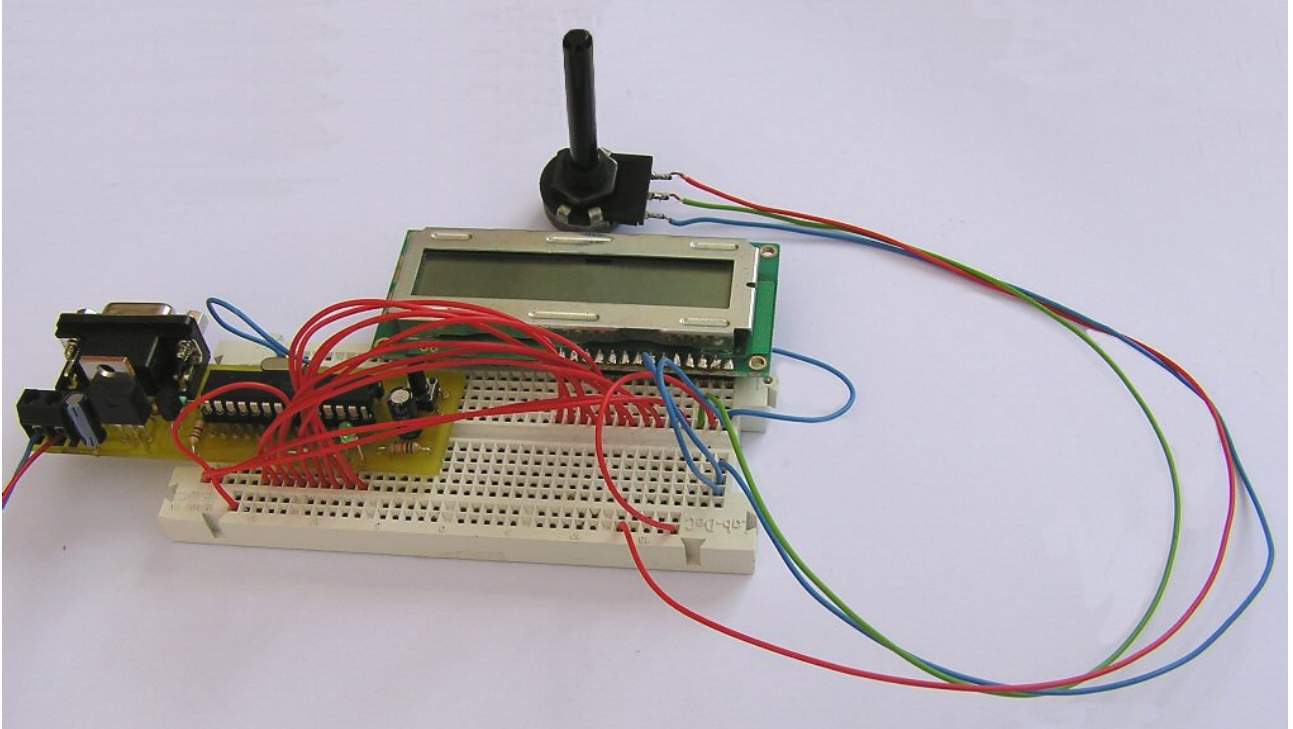
Quatrième application: Gestion d'un afficheur LCD

Les afficheurs LCD sont présents dans de nombreux appareils du commerce. Devenus très simples à gérer depuis l'apparition des circuits du type HD44780, ils peuvent être facilement intégrés dans une application à base de PIC. Nous allons voir comment gérer ces afficheurs et réaliser une bibliothèque de fonctions spécifiques.

Le montage

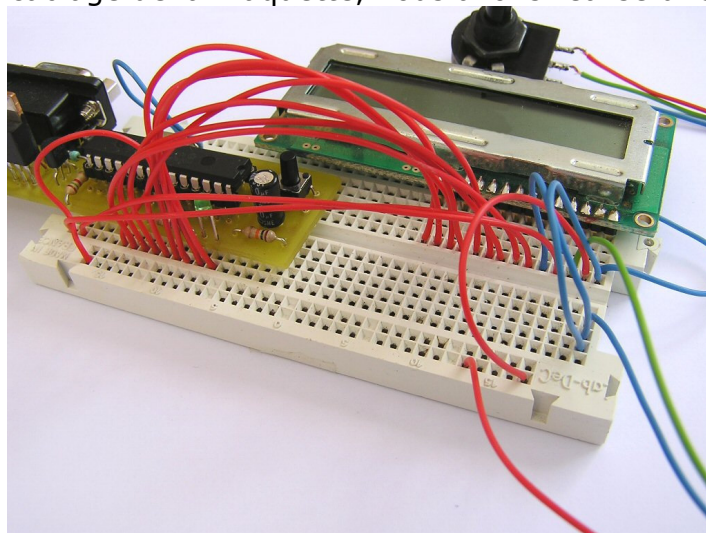
Pour réaliser ce montage, vous aurez besoin des composants suivants:

- un afficheur LCD compatible HD44780, une ou deux lignes de 16 caractères,
- un potentiomètre de 10 kohms,

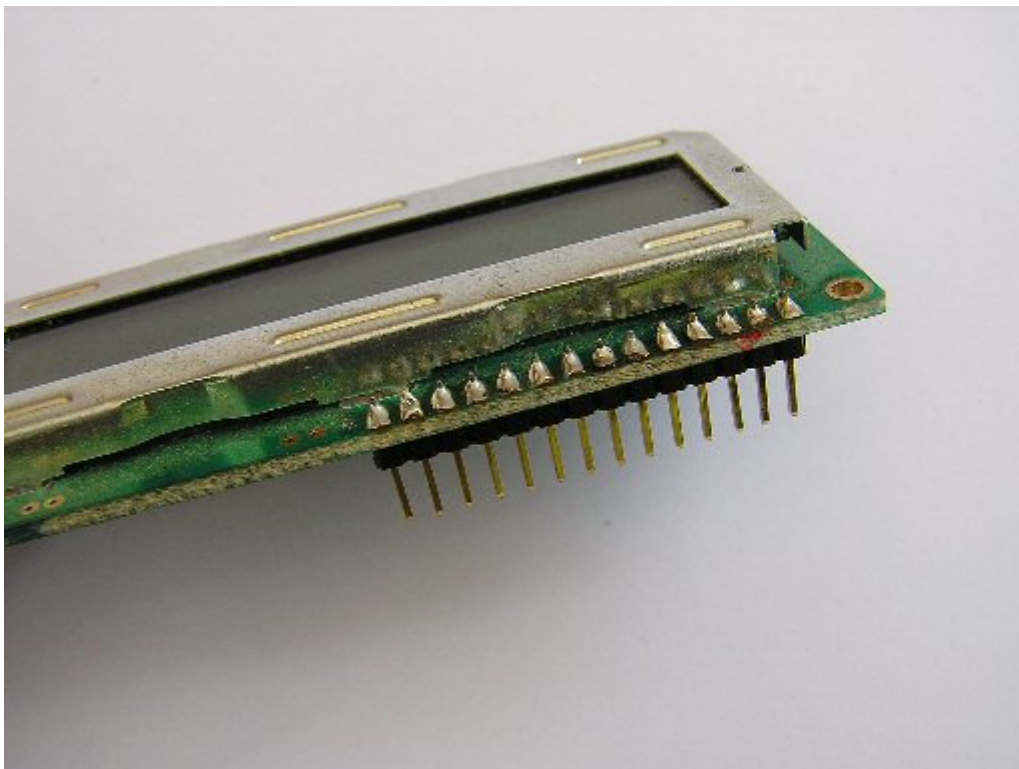


Vue d'ensemble du montage

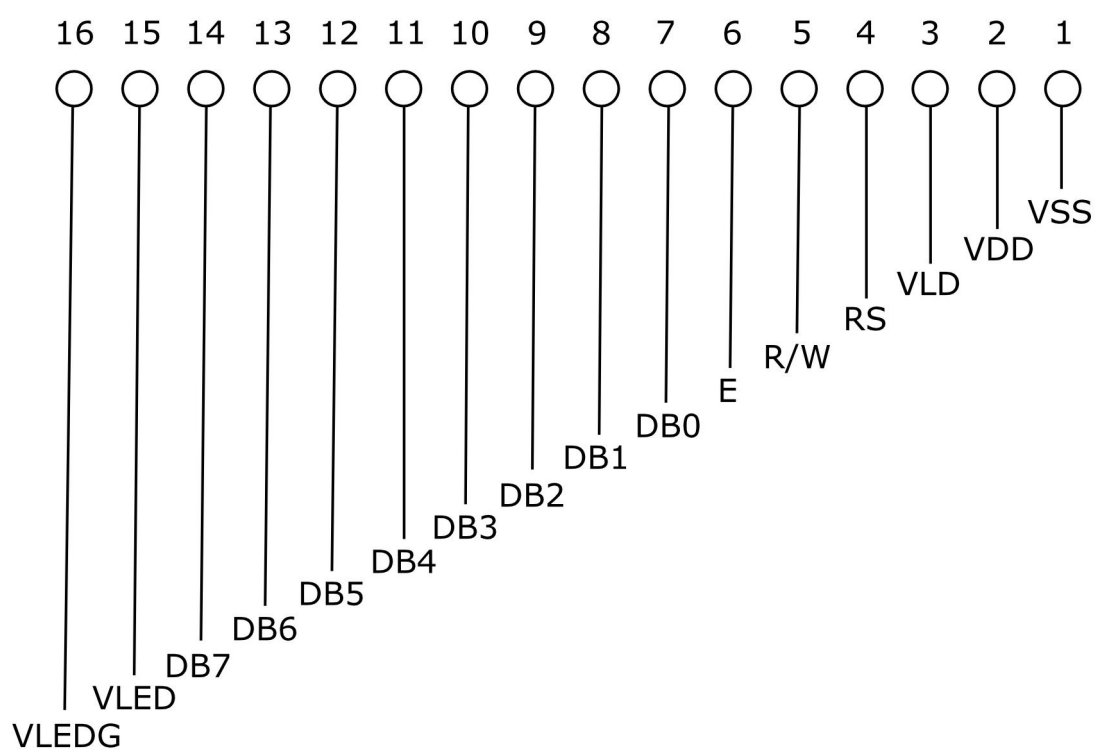
Afin de simplifier le câblage de la maquette, nous avons réalisé une photo de détail:



Au préalable, l'afficheur LCD a été équipé d'une barette de picot pour permettre son insertion sur la plaque d'essai.



Ce connecteur est standard sur l'ensemble des afficheurs compatibles HD44780, généralement la broche 1 est repérée sur le circuit imprimé. Le brochage du connecteur est le suivant:



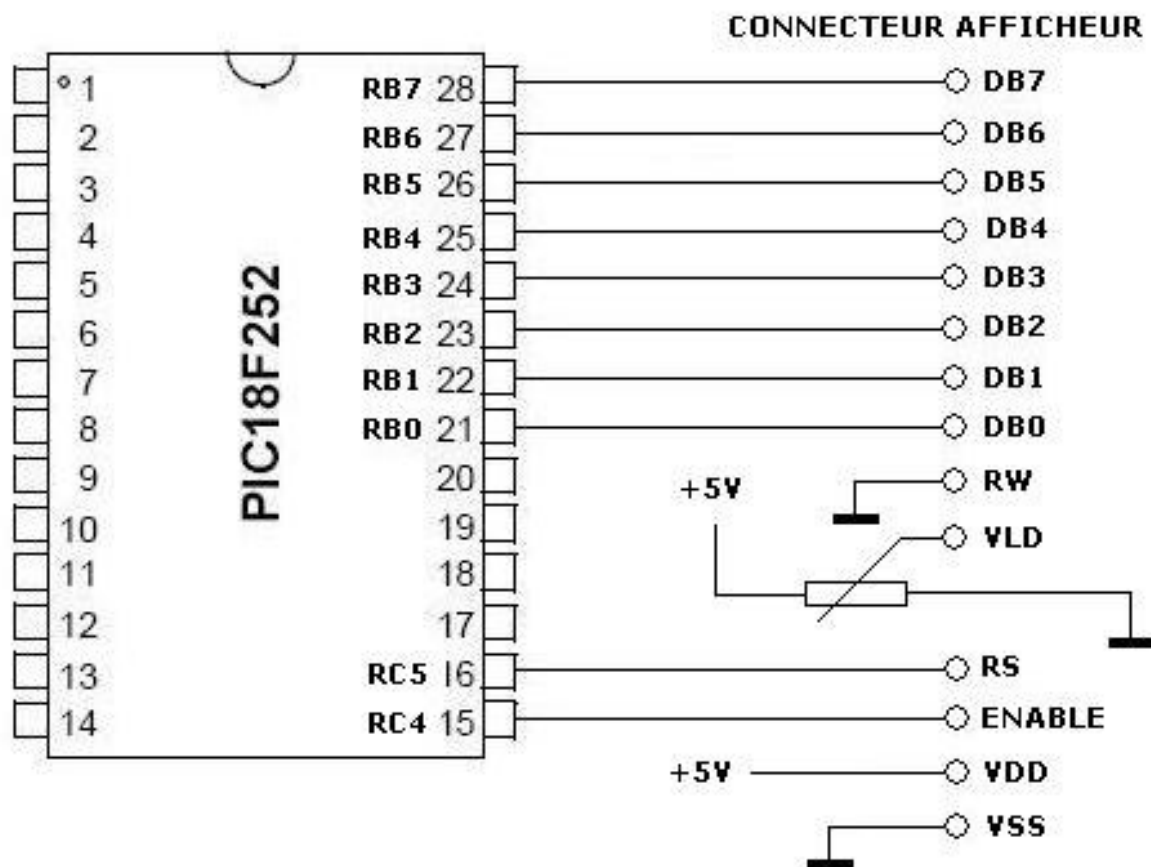
Les broches 15 et 16 n'existent pas sur la plupart des afficheurs, il s'agit de

l'alimentation du rétro-éclairage.

DESCRIPTION DES BROCHES DE RACCORDEMENT

VSS	0 volt de l'alimentation
VDD	+5V
VLD	tension à appliquer pour gérer le contraste
RS	Register select (mode instruction ou donnée)
R/W	Read/Write (écriture ou lecture)
E	Enable (Sélection de l'afficheur)
DB0 – DB7	Data 0 à 7 (données ou instructions)
VLED	tension positive du rétro-éclairage
VLEDG	0 volt du rétro-éclairage

Voici le schéma du montage que nous avons réalisé sur la plaque d'essai.



La valeur du potentiometre servant au réglage du contraste de l'afficheur est de 10 kohms. D'autres valeurs peuvent être utilisées.

Fonctionnement de l'afficheur

L'afficheur dans notre montage n'a été câblé qu'en écriture (RW=0), nous nous limiterons donc à ce type de fonctionnement.

L'afficheur reçoit deux types de commande, soit des instructions permettant de gérer l'afficheur proprement dit (effacement de l'écran, affichage du curseur, clignotement du curseur etc...) soit des données qui seront affichées à la position courante du curseur.

La sélection du mode instruction est réalisée en mettant la broche RS à 0 (register select) et en envoyant ensuite l'octet de commande sur le port de données (DB0 à DB7). Lorsque la broche RS est à 1, les valeurs envoyées sur les lignes DB0 à DB7 seront affichées sous forme de caractères ASCII à la position courante du curseur.

Les instructions

[Clear display](#)

Cette instruction permet d'effacer l'écran et de ramener le curseur à la position 0 (en haut à gauche).

RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	0	0	0	1

[Return home](#)

Cette instruction permet ramener le curseur à la position 0 (en haut à gauche).

RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	0	0	1	x

La valeur x pour DB0 indique que quel que soit l'état de ce bit, l'instruction sera exécutée.

[Shift mode](#)

Cette instruction permet de déterminer le sens de déplacement du curseur.

RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	0	1	I/D	S

Si I/D=1 le sens de déplacement du curseur est vers la droite.

Si S=1 le curseur est déplacé à chaque écriture de caractère.

[Display on/off](#)

Cette instruction permet de fixer l'état de l'afficheur et du curseur.

RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	1	D	C	B

Si D=1, l'afficheur est allumé.

Si C=1, le curseur est visible.

Si B=1, le curseur est clignotant.

Function set

Cette instruction permet de définir la longueur des données, du nombre de ligne et de la police de caractères.

RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	1	DL	N	F	x	x

DL=1 mode 8 bits, dans ce cas les 8 lignes DB0 à DB7 sont câblées sur le processeur. Si DL=0, l'afficheur est en mode 4 bits, les lignes DB0 à DB3 sont câblées sur le processeur et les 8 bits sont transmis en 2 phse, d'abord les 4 bits de poids faible ensuite les 4 bits de poids fort.

N=1 l'afficheur a deux lignes, N=0 l'afficheur n'a qu'une ligne.

F=1 les matrices de caractères ont une résolution de 5x10, si F=0 les matrices ont une résolution de 5x7.

Write data

Cette instruction permet de fixer l'adresse d'écriture des données. Les lignes de caractères affichées sont stockées dans une zone mémoire, pour écrire à un endroit donné, il suffit de modifier l'adresse de la mémoire d'affichage. Cette mémoire est appelée DDRAM.

RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	1	ADD6	ADD5	ADD4	ADD3	ADD2	ADD1	ADD0

Voici l'adresse de chaque caractère dans la zone mémoire de la première ligne.

ADD	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0A	0x0B	0x0C	0x0D	0x0E	0x0F
CHAR	B	O	N	J	O	U	R	—	—	—	—	—	—	—	—	—

Voici l'adresse de chaque caractère dans la zone mémoire de la deuxième ligne.

ADD	0x40	0x41	0x42	0x43	0x44	0x45	0x46	0x47	0x48	0x49	0x4A	0x4B	0x4C	0x4D	0x4E	0x4F
CHAR	A	—	T	O	U	S	—	—	—	—	—	—	—	—	—	—

Nous nous sommes limités dans cette description aux instructions les plus simples. Il est tout de même important de noter que ces afficheurs ont des possibilités plus étendues, notamment, la programmation de caractères spécifiques. Nous n'aborderons pas ces points dans cet ouvrage, le but étant avant tout de réaliser une bibliothèque de fonctions permettant d'utiliser facilement ce type d'afficheur.

Les lecteurs intéressés par ces fonctions non documentées ici se reporteront à la data sheet de l'afficheur.

A noter également qu'un certain nombre d'afficheur ont des formats d'instructions légèrement divergents, nous avons testés par exemple des afficheurs EPSON à base de HD44780 qui sont des afficheurs une ligne de 16 caractères mais qui devaient être adressés comme des afficheurs à deux lignes mais avec des groupes de 8 caractères.

Si vous utilisez des afficheurs de surplus, procédez à quelques essais pour vérifier la

compatibilité 100%. Nous verrons dans ce chapitre comment adapter une bibliothèque pour la rendre un peu plus universelle.

Initialisations de l'afficheur

Nous allons voir ici quelles sont les actions à effectuer pour mettre l'afficheur dans une configuration correcte avant d'écrire le texte.

1. Il faut, dans un premier temps définir le mode fonctionnement de l'afficheur. Dans notre cas, le mode 8 bits sera choisi (les lignes DB0 à DB7 de l'afficheur sont câblées sur le port B), donc le bit DL sera mis à 1 dans l'instructions `function set`. Il faut ensuite définir le nombre de lignes de l'afficheur, dans notre cas il s'agit d'un afficheur à 2 lignes de caractères, le bit N sera mis à 1. Enfin nous choisirons une matrice de caractère standard de 5x7, donc F=0.
Nous allons donc écrire en premier la valeur 0x38 sur le PORTB.
2. Nous allons ensuite allumer l'écran sans afficher de curseur. Pour cela l'instruction `display on/off` sera utilisée avec D=1.
Nous écrirons la valeur 0x0C sur le PORTB.
3. La dernière opération à réaliser pour initialiser l'afficheur est de déterminer le sens de déplacement du curseur. L'instruction `shift mode` sera initialisée avec la valeur 0x06.
PORTB=0x06
4. Pour finir, nous effacerons l'écran avec l'instruction `Clear display`.
PORTB=0x01

Une fois cette séquence d'initialisation réalisée, l'afficheur est prêt à recevoir des données destinées à être écrites à partir de la position 0 sur la 1ère ligne.

Le programme

Dans un premier temps, nous allons écrire les routines d'initialisation et écrire le caractère A sur l'écran.

```
// Afficheur LCD
// Jean-Pierre MANDON 2006
// Pic 18F252 à 10 Mhz
// version 1.0 du 20/08/2006

#define __18F252
#include <pic18F252.h>

#define DATA PORTB
#define RS PORTCbits.RC5
#define enable PORTCbits.RC4

void main(void) {
    int i;

    TRISB=0x00;           // port B en sortie
    PORTB=0;
    TRISCbits.TRISC5=0;   // RC5 = Register select
    TRISCbits.TRISC4=0;   // RC4 = enable de l'afficheur

    RS=0;                 // sélectionner le mode instructions
    PORTB=0x38;           // instruction function set, DL=1, N=1, F=0
    enable=1;
```

```

for (i=0;i<1000;i++);           // envoi d'une impulsion sur la broche enable
enable=0;                       // instruction display on/off, D=1
PORTB=0x0C;                     // envoi d'une impulsion sur la broche enable
enable=1;                       // instruction shift mode, I/D=1
for (i=0;i<1000;i++);           // envoi d'une impulsion sur la broche enable
enable=0;                       // instruction clear display
PORTB=0x01;                     // envoi d'une impulsion sur la broche enable
enable=1;                       // envoi d'une impulsion sur la broche enable
for (i=0;i<1000;i++);           // envoi d'une impulsion sur la broche enable
enable=0;

RS=1;

DATA='A';                       // on envoie A sur l'afficheur
enable=1;
for (i=0;i<1000;i++);           // envoi d'une impulsion sur la broche enable
enable=0;

while (1);

}

```

On voit que la validation d'une instruction est réalisée en mettant la broche enable à 1 puis à 0 après une temporisation. Cette série d'instructions étant répétée de nombreuses fois dans notre programme de façon identique, nous allons créer une fonction permettant d'éviter cette répétition. Cette fonction, que nous appellerons `set_enable` exécutera la même série d'instructions que celle que nous voyons ici.

```

void set_enable(void) {
int i;

enable=1;
for (i=0;i<1000;i++);
enable=0;
}

```

Pour appeler cette fonction à partir de notre programme:

```
set_enable();
```

ce qui va nous permettre d'écrire le programme de la façon suivante:

```

// Afficheur LCD
// Jean-Pierre MANDON 2006
// Pic 18F252 à 10 Mhz
// version 1.1 du 20/08/2006

#define __18F252
#include <pic18F252.h>

#define DATA PORTB
#define RS PORTCbits.RC5
#define enable PORTCbits.RC4

// Validation d'une instruction sur l'entrée enable du HD44780

void set_enable(void) {
int i;

enable=1;
for (i=0;i<1000;i++);
enable=0;
}

```

```

}

void main(void) {
int i;

TRISB=0x00;           // port B en sortie
PORTB=0;
TRISCBits.TRISC5=0;   // RC5 = Register select
TRISCBits.TRISC4=0;   // RC4 = enable de l'afficheur


RS=0;                 // sélectionner le mode instructions
PORTB=0x38;           // instruction function set, DL=1, N=1, F=0
set_enable();
PORTB=0x0C;           // instruction display on/off, D=1
set_enable();
PORTB=0x06;           // instruction shift mode, I/D=1
set_enable();
PORTB=0x01;           // instruction clear display
set_enable();

RS=1;

DATA='A';             // on envoie A sur l'afficheur
enable=1;
for (i=0;i<1000;i++); // envoi d'une impulsion sur la broche enable
enable=0;

while (1);
}

```

Notre programme se trouve plus simple à lire, mais surtout beaucoup moins important en taille de code. La version 1.0 avait une taille de 392 octets, après la création de la fonction `set_enable`, la taille du code est de 348 octets. On voit que l'optimisation d'un programme est souvent lié à la façon d'écrire le code source.

La fonction `set_enable` n'a pas de paramètres d'entrée ni de paramètres de sortie. Elle est donc déclarée `void` en entrée (`void enable`) et `void` en paramètres (`set_enable (void)`). Le mot clé `void` signifie vide en anglais.

Lorsqu'une fonction renvoie des valeurs et en reçoit elle est écrite sous la forme suivante:

```
int rotation(int nombre_de_tours) { }
```

Cette fonction reçoit en paramètres d'entrée le nombre de tours à exécuter sous forme d'un entier et retourne une valeur en sortie. Son appel pourra être écrit dans la boucle `main` sous la forme suivante:

```
retour=rotation(10);
```

La variable `retour` contiendra la valeur transmise en sortie par la fonction `rotation`. On peut écrire simplement:

```
rotation(10);
```

Dans ce cas, la valeur de sortie de la fonction ne sera pas exploitée.

Pour améliorer encore la lisibilité de notre programme, nous allons créer une fonction d'initialisation de l'afficheur LCD. Celle-ci prendra en charge toutes les actions décrites ci-dessus. Cette opération n'a pour but que l'amélioration de la lisibilité.

```

// Afficheur LCD
// Jean-Pierre MANDON 2006
// Pic 18F252 à 10 Mhz
// version 1.2 du 20/08/2006

#define __18F252
#include <pic18F252.h>

#define DATA PORTB
#define RS PORTCbits.RC5
#define enable PORTCbits.RC4

// Validation d'une instruction sur l'entrée enable du HD44780

void set_enable(void) {
int i;

enable=1;
for (i=0;i<1000;i++);
enable=0;
}

// initialisation du HD44780

void init_LCD(void) {

RS=0; // sélectionner le mode instructions
PORTB=0x38; // instruction function set, DL=1, N=1, F=0
set_enable();
PORTB=0x0C; // instruction display on/off, D=1
set_enable();
PORTB=0x06; // instruction shift mode, I/D=1
set_enable();
PORTB=0x01; // instruction clear display
set_enable();
}

void main(void) {

TRISB=0x00; // port B en sortie
PORTB=0;
TRISCbits.TRISC5=0; // RC5 = Register select
TRISCbits.TRISC4=0; // RC4 = enable de l'afficheur

init_LCD();

RS=1;

DATA='A'; // on envoie A sur l'afficheur
set_enable();

while (1);

}

```

Nous voyons ici que la fonction `init_LCD` n'est qu'une recopie des instructions que nous exécutons avant dans la boucle `main`. Cette fonction fait appel à la fonction `enable`, il est donc possible de faire des appels imbriqués de fonctions.

Nous allons enfin ajouter à notre programme une fonction `writetchar` qui permettra d'écrire un caractère sur l'écran sans avoir à connaître toutes les subtilités du câblage de la carte d'essais.

```

void writechar(char caractere) {
    RS=1;                                // affichage
    DATA=caractere;                     // caractere à afficher sur le PORTB
    set_enable();
}

```

La boucle `main` devient:

```

void main(void) {
    TRISB=0x00;                          // port B en sortie
    PORTB=0;
    TRISCbits.TRISC5=0;                  // RC5 = Register select
    TRISCbits.TRISC4=0;                  // RC4 = enable de l'afficheur
    init_LCD();
    write_char('A');
    while (1);
}

```

Il ne reste maintenant qu'à compléter les fonctions par ce qui peut nous être utile pour afficher des caractères sur l'afficheur. Nous avons déterminé les fonctions suivantes:

- `void set_display(int état)`
cette fonction nous permettra de fixer l'état du curseur (on, off, clignotant) et de l'afficheur (on,off).
- `void write_string(char* texte, short indent, short line)`
Cette fonction nous permettra d'écrire une chaîne de caractères sur l'afficheur, justifiée à droite ou à gauche, sur la première ou la deuxième ligne.
- `void clear(void)`
Cette fonction permet d'effacer l'écran et de ramener le curseur en position 0.

Un certain nombre de fonctions seront associées à celles qui sont citées ci-dessus, parmi celles-ci:

- `int length(char* texte)`
qui calcule la longueur d'une chaîne de caractères.
- `void setpos(short position, short line)`
Qui permet de déterminer la position d'affichage en modifiant l'adresse dans la DDRAM.

Voici nos fonctions telles qu'elles apparaissent dans la version 1.3 du programme afficheur:

```

#define cursor_blink 0x01
#define cursor_on    0x02
#define display_on   0x04

#define right        1
#define left         2

// fonction de calcul de la longueur d'une chaîne de caractères

int length(char* texte) {
    int i,longueur;

```

```

i=0;
longueur=0;
while (texte[i++]!=0) longueur++;
return(longueur); }

// fonction position sur l'afficheur

void set_pos(short position, short line) {
RS=0;
if (line==1) DATA=0x80+position;
else        DATA=0xC0+position;
set_enable(); }

// fonction d'effacement de l'écran
void clear(void) {
RS=0;
DATA=0x01;
set_enable(); }

// fonction de définition de l'état de l'affichage
void set_display(int etat) {

RS=0;
DATA=0x08+etat;
set_enable(); }

// Fonction d'écriture d'une chaîne de caractères justifiée à droite ou à gauche

void write_string(char* texte, short indent, short line) {
int i;
int longueur;
if (line==1) {
    RS=0;
    DATA=0x80;
    set_enable();
}
else {
    RS=0;
    DATA=0xC0;
    set_enable();
}
longueur=length(texte);
if (indent==right) set_pos(16-longueur,line);
else set_pos(0,line);

RS=1;
i=0;
while (texte[i]!=0) {
    DATA=texte[i];
    set_enable();
    i++;
}
}

```

Quelques notions sont nouvelles dans cette façon d'écrire le programme. Pour les fonctions d'état de l'affichage, nous avons défini des masques sur les valeurs suivantes:

```

#define cursor_blink 0x01
#define cursor_on    0x02
#define display_on   0x04

```

Ces valeurs correspondent à la mise à 1 du bit 0 pour cursor_blink (curseur clignotant), à la mise à 1 du bit 1 pour cursor_on (curseur allumé) et à la mise à 1 du bit 2 pour display_on (afficheur allumé).

Il suffit ensuite d'appeler la fonction set_display sous la forme:

```
set_display(cursor_blink+cursor_on);
```

ou simplement

```
set_display(cursor_on)
```

si l'on ne veut afficher que le curseur sans qu'il clignote.

Dans les fonctions write_string et length, le paramètre chaîne de caractère est déclaré comme ci-dessous:

```
char* texte;
```

Il s'agit d'un passage de valeur par pointeur de la chaîne de caractère. En effet, une chaîne de caractère est une zone mémoire dans laquelle sont stockés les caractères, il est donc indispensable de transmettre l'adresse de la zone mémoire et non la valeur puisqu'on ne connaît pas sa longueur. L'indication * permet de déterminer que c'est un pointeur qui est récupéré et affecté à la chaîne de caractère. A noter que cette chaîne de caractère n'est en fait qu'un tableau à une dimension.

Voici la version 1.3 de notre programme d'affichage:

```
// Afficheur LCD
// Jean-Pierre MANDON 2006
// Pic 18F252 à 10 Mhz
// version 1.3 du 20/08/2006

#define __18F252
#include <pic18F252.h>

#define DATA PORTB
#define RS PORTCbits.RC5
#define enable PORTCbits.RC4

#define cursor_blink 0x01
#define cursor_on    0x02
#define display_on   0x04
#define right        1
#define left         2

// Validation d'une instruction sur l'entrée enable du HD44780
void set_enable(void) {
    int i;

    enable=1;
    for (i=0;i<1000;i++);
    enable=0;
}

// fonction de calcul de la longueur d'une chaîne de caractères
int length(char* texte) {
    int i,longueur;

    i=0;
    longueur=0;
    while (texte[i++]!=0) longueur++;
    return(longueur); }

// fonction position sur l'afficheur
```

```

void set_pos(short position, short line) {

RS=0;
if (line==1) DATA=0x80+position;
else          DATA=0xC0+position;
set_enable(); }

// fonction d'effacement de l'écran
void clear(void) {

RS=0;
DATA=0x01;
set_enable(); }

// fonction de définition de l'état de l'affichage
void set_display(int etat) {

RS=0;
DATA=0x08+etat;
set_enable(); }

// Fonction d'écriture d'une chaîne de caractères justifiée à droite ou à gauche
void write_string(char* texte, short indent, short line) {
int i;
int longueur;

longueur=length(texte);
if (indent==right) set_pos(16-longueur,line);
else set_pos(0,line);
RS=1;
i=0;
while (texte[i]!=0)    {
                        DATA=texte[i];
                        set_enable();
                        i++;
                        }
}

// initialisation du HD44780
void init_LCD(void) {

RS=0;                                     // sélectionner le mode instructions
DATA=0x38;                               // instruction function set, DL=1, N=1, F=0
set_enable();
DATA=0x0C;                               // instruction display on/off, D=1
set_enable();
DATA=0x06;                               // instruction shift mode, I/D=1
set_enable();
DATA=0x01;                               // instruction clear display
set_enable();
}

void main(void) {
int i;

TRISB=0;
PORTB=0;
TRISCbits.TRISC5=0;
TRISCbits.TRISC4=0;
PORTC=0;

```

```

for (i=0;i<30000;i++);
init_LCD();
set_display(display_on);
clear();

write_string("VIVE",left,1);
write_string("SDCC",right,1);
write_string("ET LE LANGAGE C",left,2);

while (1)    {}
}

```

Création d'une bibliothèque

A partir des différentes fonctions que nous avons créées, nous allons créer une bibliothèque réutilisable sans avoir à ré-écrire les fonctions dans le code source.

Pour cela, nous allons créer un fichier portant l'extension .c qui contiendra le code de toutes les fonctions.

Dans un premier temps, recopier l'ensemble des fonctions propres à l'afficheur dans un fichier nommé lcd.c

```

// bibliothèque de fonctions pour la gestion des afficheurs HD44780
// version 1.0 du 20/08/2006

#define cursor_blink    0x01
#define cursor_on       0x02
#define display_on      0x04
#define right           1
#define left            2

// Validation d'une instruction sur l'entrée enable du HD44780
void set_enable(void) {
int i;

enable=1;
for (i=0;i<1000;i++);
enable=0;
}

// fonction de calcul de la longueur d'une chaîne de caractères
int length(char* texte) {
int i,longueur;

i=0;
longueur=0;
while (texte[i++]!=0) longueur++;
return(longueur); }

// fonction position sur l'afficheur
void set_pos(short position, short line) {

RS=0;
if (line==1) DATA=0x80+position;
else        DATA=0xC0+position;
set_enable(); }

// fonction d'effacement de l'écran
void clear(void) {

RS=0;
DATA=0x01;
set_enable(); }

```

```

// fonction de définition de l'état de l'affichage
void set_display(int etat) {
    RS=0;
    DATA=0x08+etat;
    set_enable(); }

// Fonction d'écriture d'une chaîne de caractères justifiée à droite ou à gauche
void write_string(char* texte, short indent, short line) {
    int i;
    int longueur;

    longueur=length(texte);
    if (indent==right) set_pos(16-longueur,line);
    else set_pos(0,line);
    RS=1;
    i=0;
    while (texte[i]!=0) {
        DATA=texte[i];
        set_enable();
        i++;
    }
}

// initialisation du HD44780
void init_LCD(void) {
    RS=0; // sélectionner le mode instructions
    DATA=0x38; // instruction function set, DL=1, N=1, F=0
    set_enable();
    DATA=0x0C; // instruction display on/off, D=1
    set_enable();
    DATA=0x06; // instruction shift mode, I/D=1
    set_enable();
    DATA=0x01; // instruction clear display
    set_enable();
}

```

Nous avons créé maintenant une bibliothèque de fonctions pour la gestion des afficheurs, celle-ci pourra être utilisée sans ré-écrire les fonctions dans le source. Pour l'utiliser il suffira d'inclure la bibliothèque par une directive **#include**

```

// Afficheur LCD
// Jean-Pierre MANDON 2006
// Pic 18F252 à 10 Mhz
// version 1.4 du 20/08/2006

#define __18F252
#include <pic18F252.h>

#define DATA PORTB
#define RS PORTCbits.RC5
#define enable PORTCbits.RC4

#include <lcd.c>

void main(void) {
    int i;

    TRISB=0;
    PORTB=0;
    TRISCbits.TRISC5=0;
    TRISCbits.TRISC4=0;
    PORTC=0;
    for (i=0;i<30000;i++);
}

```

```

init_LCD();
set_display(display_on);
clear();

write_string("VIVE",left,1);
write_string("SDCC",right,1);
write_string("ET LE LANGAGE C",left,2);

while (1)    {}
}

```

Résumé

- une variable déclarée `short` est un entier non signé sur 8 bits (0..255),
- une fonction sans passage de paramètres est déclarée `void`,
`void test(void) { }`
- Une fonction retournant une valeur en sortie est déclarée:
`int test(void) { }`
dans ce cas la valeur de retour est renvoyée à l'intérieur de la fonction:
`return(5);` ou `return(i);`
- une fonction à laquelle est transmise un paramètre en entrée et qui retourne une valeur en sortie est déclarée:
`int test(int ligne) { }`
- le source d'un module externe peut être ajouté au programme en utilisant la directive `#include "nom_du_programme.c"`

Cinquième application: Utilisation de l'UART du PIC

L'UART (universal asynchronous receiver transmitter) est probablement le périphérique le plus universel pour tous ceux qui veulent faire communiquer des systèmes entre eux. Sur PC on l'appelle port COM mais il s'agit en fait d'un UART avec un convertisseur de tension permettant de respecter la norme RS232 pour les signaux. Dans ce chapitre, nous allons voir comment utiliser l'UART du PIC et le faire dialoguer avec un terminal sur PC.

Le montage

Dans un premier temps, nous nous contenterons de la platine d'essais pour tester notre programme. Pour relier le port COM d'un PC à l'UART d'un PIC, l'adaptation des signaux est nécessaire, en effet, les sorties du PIC sont compatibles TTL (0 ou 5 V) et le port COM du PC est compatible RS232 (-12V ou +12V). Notre platine de développement intègre déjà les composants nécessaires à cette conversion de signaux puisqu'elle est reliée au PC pour le téléchargement des programmes. Sur vos réalisations finales, il sera nécessaire d'intégrer un convertisseur de signaux (MAX232 par exemple) ou de vous équiper d'un câble de conversion tel que celui que nous proposons par exemple.



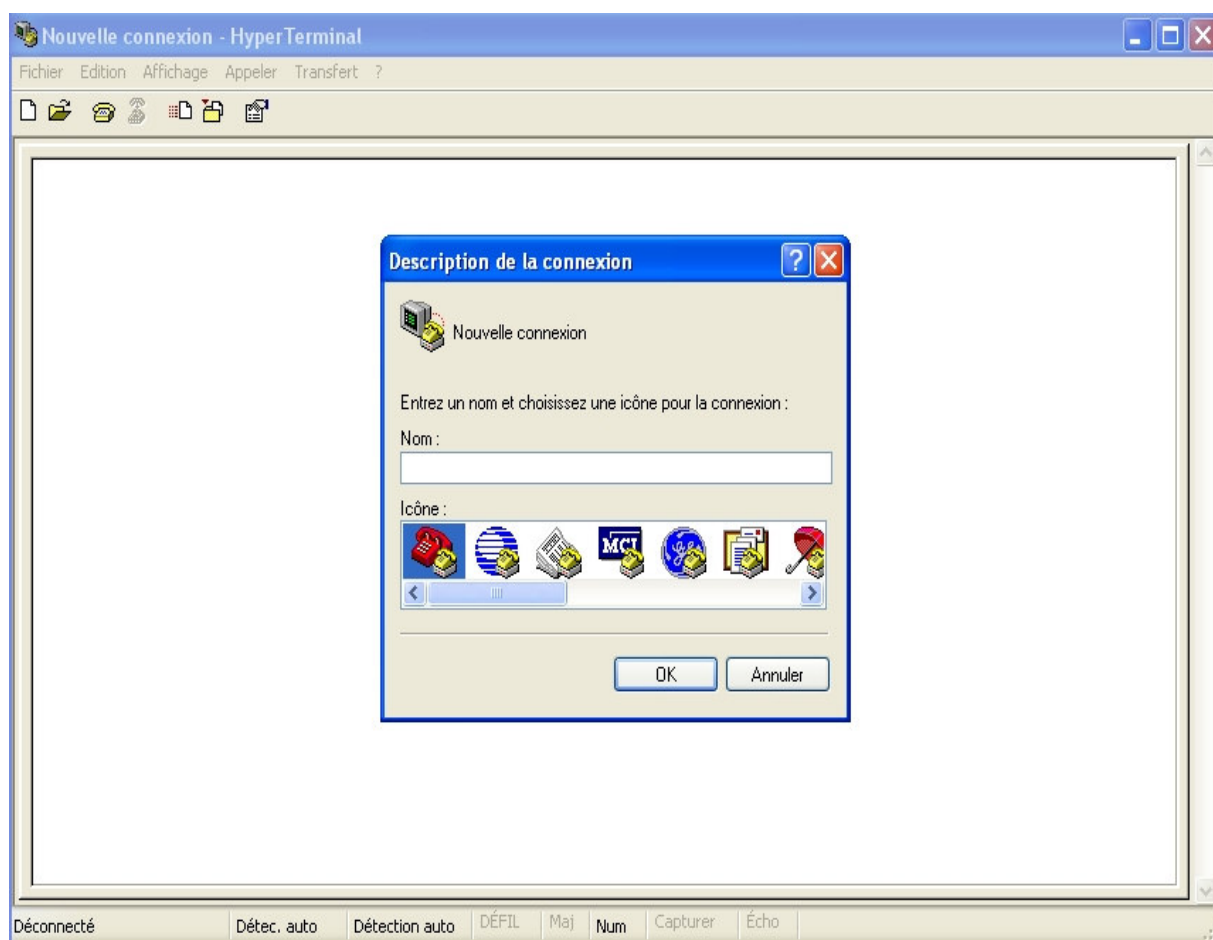
Notre câble C232

Un outil supplémentaire HYPERTERMINAL

Un outil indispensable dès que l'on commence à travailler avec des applications communicantes est un terminal sur PC. Windows est livré en standard avec une application permettant de réaliser du monitoring de port série, il s'agit d'hyperterminal. Ce programme conviendra parfaitement pour réaliser les essais que nous souhaitons faire dans ce chapitre.

Le source du programme que nous vous proposons est très simple et affiche sur l'UART du PIC un message puis se met en attente d'une entrée de l'utilisateur. La vitesse de communication est de 19200 bauds. Il va donc falloir configurer hyperterminal pour fonctionner à cette vitesse.

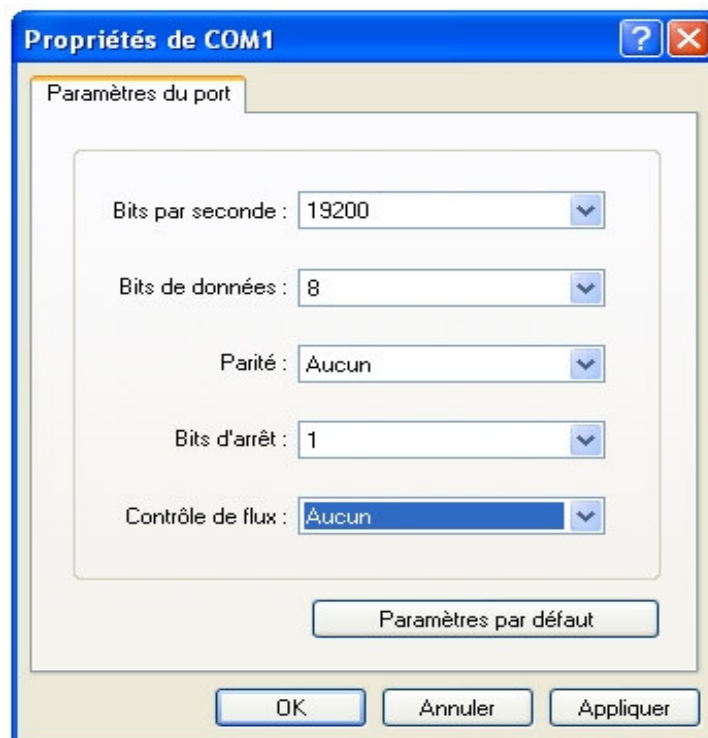
Lancer Hyperterminal (menu accessoires, communication, hyperterminal), vous obtenez l'écran suivant:



Entrez un nom pour la nouvelle connexion, personnellement je définis les noms en fonction de la vitesse de communication et du port utilisé. Dans ce cas la, 19200_COM1 permet de s'y retrouver plus vite ensuite.



Sélectionnez ensuite le port sur lequel est raccordé votre platine de développement.



Sélectionnez ensuite la vitesse et le format des données sur le port série. Dans notre cas la vitesse est de 19200 bauds.

Vous obtenez ensuite l'écran hyperterminal tel qu'il apparaîtra dorénavant lorsque vous lancerez 19200_com1.

Le programme

L'utilisation de l'UART du PIC est liée à la définition des différents paramètres de communication. Parmi ceux-ci on trouve la vitesse, le format, le nombre de stop bit et la parité. Nous avons regroupé l'ensemble du paramétrage dans une bibliothèque `uartsdcc.c` qui comporte une fonction `setup_UART()` qui s'occupe de positionner les différents registres. Il suffit de transmettre à cette fonction une étiquette pour initialiser correctement l'UART. Ces étiquettes sont définies en tête de la bibliothèque `uartsdcc.c`.

```
// SPBRG value for FOSC 10 Mhz
#define B1200_10    0,129
#define B2400_10    0,64
#define B9600_10    1,64
#define B19200_10   1,32
#define B38400_10   1,16
```

Nous voyons ci-dessus que le paramétrage de l'UART est dépendant de la vitesse d'horloge du processeur. Dans notre cas, le quartz de 10 Mhz nous donne les valeurs permettant de configurer l'UART.

Il suffit donc d'appeler la fonction `setup_UART` avec la valeur `B19200_10`

```
setup_UART(B19200_10);
```

Outre cette fonction d'initialisation, notre bibliothèque gère également la fonction de réception des octets arrivant sur le port série à l'aide d'un buffer rotatif. La taille de ce buffer est fixé par l'étiquette `BUFFER` qui est fixée à 10 par défaut.

La réception des caractères est traitée par interruptions. Les caractères reçus sont donc automatiquement placés dans un buffer sans que le programme principal ne soit arrêté. Etudions maintenant le source de notre programme de test qui va nous éclairer sur les possibilités de cette bibliothèque.

```
// test file for sdcc UART library
// SDCC version / small device c compiler
// Jean-Pierre MANDON 2006
// www.pictec.org

#define __18F252
#include <pic18F252.h>

#include "isr.c"
#include "uartsdcc.c"

void main(void)
{
    int i;
    TRISC=0x80;                // set RX,TX pin direction
    setup_UART(B19200_10);     // setup UART
    INTCONbits.GIE=1;         // enable interrupt
    for (i=0;i<2000;i++);     // delay but i don't know why it is necessary (?)

    putc('A');                // write a character on the UART
    print_str("HELLO\n\r");    // write a string on the UART
    print_hex8(255);           // convert and write an hexadecimal value (8 bits)
    print_str("\n\r");         // line feed, carriage return

    while(1)
    {
        if (receivedchar) {   // a character has been received
            print_str("character="); //
            putc(getch());     // send it on UART
        }
    }
}
```

Dans ce programme, on commence par inclure les librairies nécessaires. `Isr.c` est la librairie qui gèrent les interruptions sur le 18F252, `uartsdcc.c` est la librairie de gestion de l'UART.

La première des opérations comme dans tout programme est de définir le sens des entrées/sorties. Dans notre cas, seule la ligne RC7 est positionnée en entrée, il s'agit de la broche de réception de l'UART. On appelle ensuite la routine d'initialisation de l'UART qui est définie dans `uartsdcc.c` avec le paramètre `B19200_10`. Ceci nous permettra de travailler avec la platine de test qui est équipée d'un quartz de 10 Mhz à la vitesse de 19200 bauds.

Pour que la réception fonctionne, il s'agit maintenant de mettre en service les interruptions. Ceci est réalisé par la ligne `INTCONbits.GIE=1`, `GIE` étant l'abréviation de `Global Interrupt Enable`.

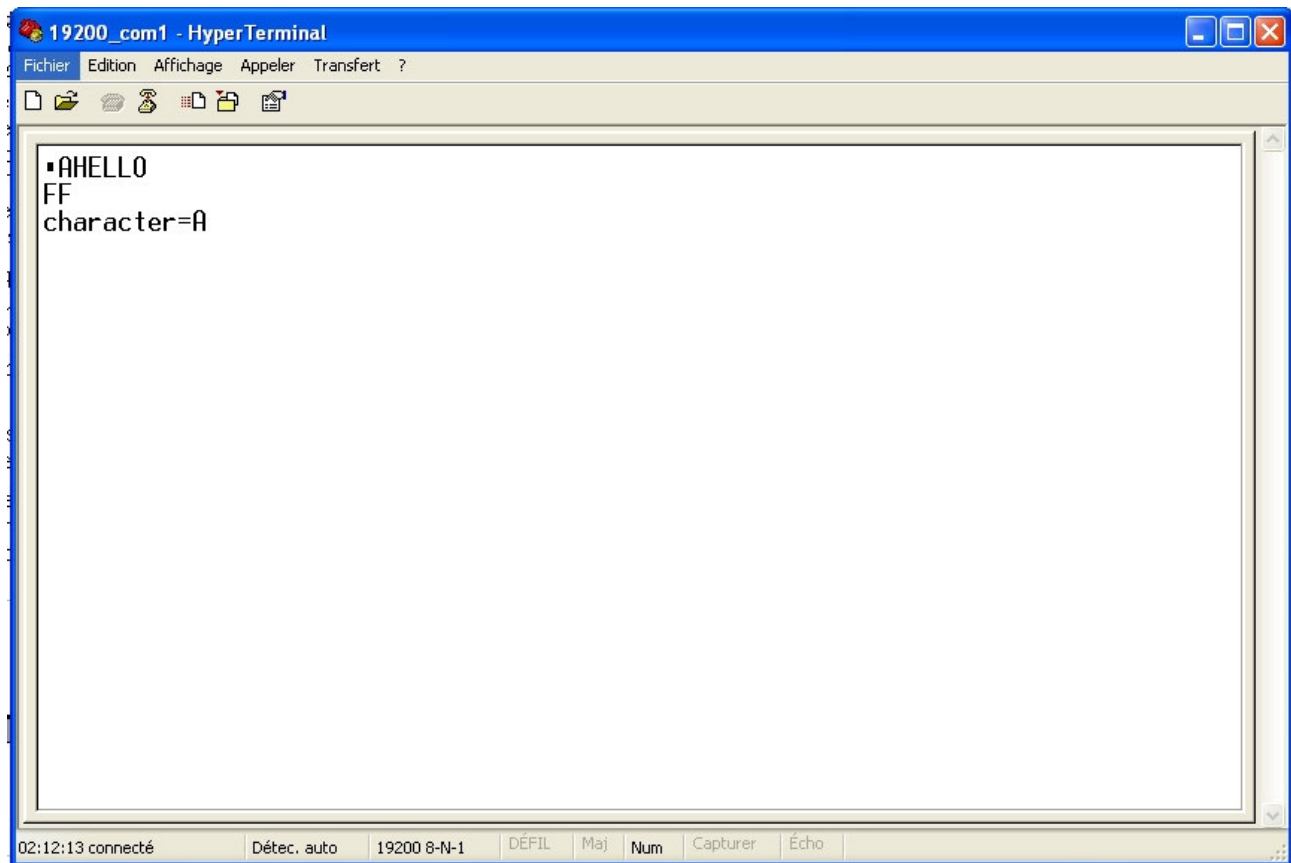
Nous pouvons maintenant tester l'UART, il est prêt à fonctionner.

La fonction `putc` envoie sur l'UART le caractère passé en paramètre. La fonction `print_str` fait la même chose avec une chaîne de caractère. La fonction `print_hex8` affiche en hexadécimal sur le terminal la valeur passée en paramètre.

Pour la réception, la fonction `receivedchar` vous permet de savoir si un caractère est disponible, si c'est le cas, ce caractère est extrait du buffer de réception par la fonction `getch()`.

Une fonction disponible dans la librairie `uartsdcc.c` n'est pas utilisée dans notre exemple, il s'agit de la fonction `clear_buffer()`. Cette fonction permet d'effacer le contenu du buffer de réception.

Au lancement de notre programme, on obtient sur hyperterminal l'écran suivant:



Le caractère A tapé sur le clavier est renvoyé par le PIC en écho sur sa broche TX.

Résumé

L'UART est probablement le périphérique le plus utile pour peu que l'on souhaite rendre communicantes nos applications. Je m'en sers également souvent pour le débogage, il permet d'afficher des valeurs de manière simple et rapide. Il est ainsi plus facile de tracer les points d'un programme et de déterminer d'où vient l'erreur.

- La configuration de l'UART est réalisée au moyen de la fonction `setup_UART`,
- Les valeurs possibles pour la vitesse de l'UART sont définies dans `uartsdcc.c`,
- La fonction `putc('A')`; envoie le caractère A sur la broche TX de l'UART,
- la fonction `print_str("TOTO")`; envoie la chaîne TOTO sur la broche TX de l'UART,

- la fonction `print_hex8(255)` envoie la valeur FF sur la broche TX de l'UART,
- La fonction `clear_buffer()` efface le contenu du buffer de réception,
- la fonction `getch()` renvoie le dernier octet du buffer,
- la fonction `receivedchar` prend la valeur `1` si le buffer n'est pas vide.