LE PIC 16F84: L'ESSENTIEL

DIABLOTRONIC

Auteur: Galodé Alexandre

Date: 11/11/2004

INTRODUCTION

Ce livre, qui s'apparente plus à un aide-mémoire, a été écrit afin de corriger nombre de problèmes rencontrés au cours de mon cursus, notamment des informations contradictoires, voire fausses , que ce soit sur le net, dans des livres, ou autre.

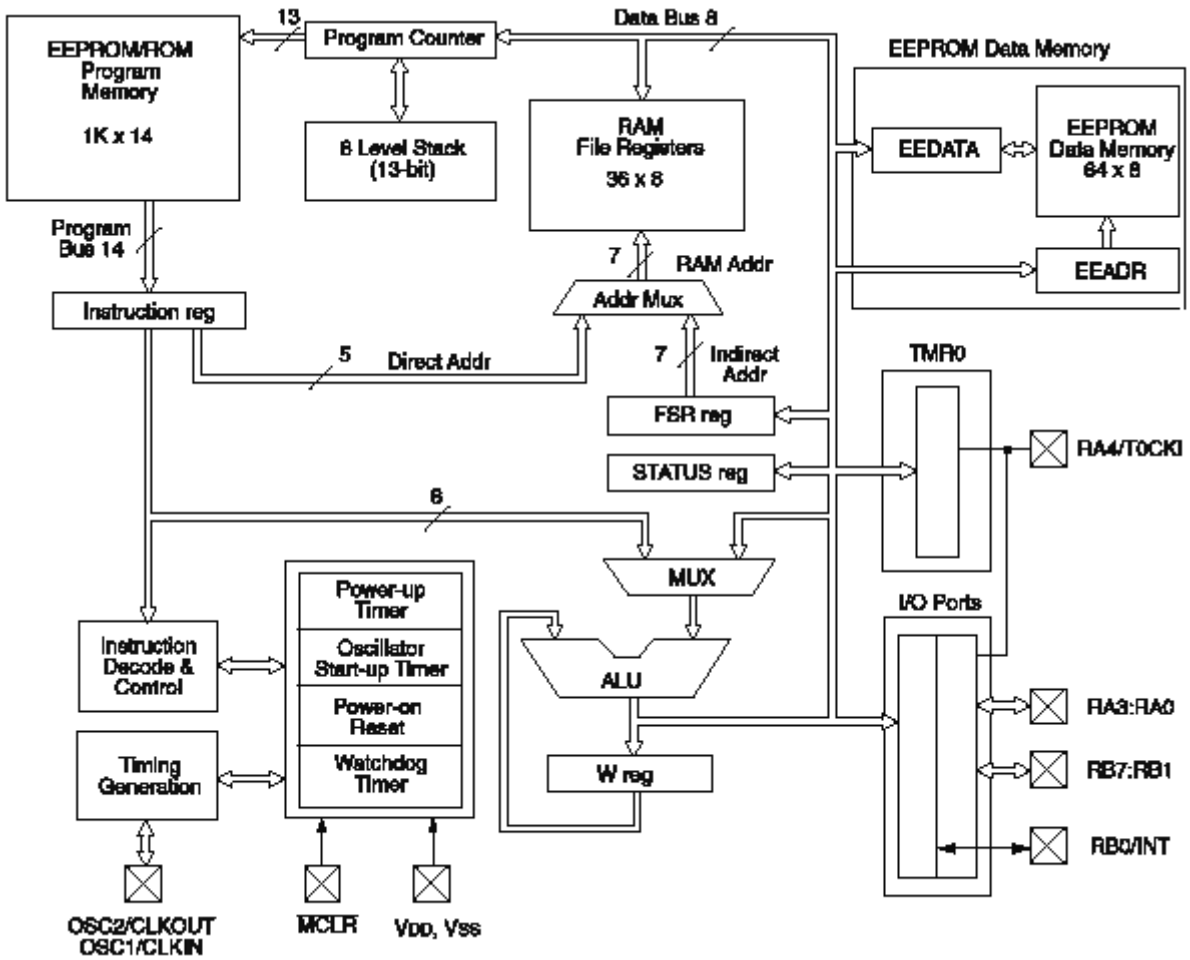
Le but de ce livre est donc de constituer en quelque sorte un livre de références pour comprendre, programmer et tester le PIC 16F84. Un livre à la fois, simple, compréhensible, mais aussi, je l'espère, efficace. Le but n'est pas ici de faire un livre avec des tonnes de documentations ne servant jamais à rien, mais au contraire, un livre avec uniquement les informations utiles et nécessaires à la compréhension et à la mise en œuvre du PIC 16F84. Nous ne nous attarderons donc pas sur les schémas détaillés du PIC 16F84, mais uniquement sur des représentations schématiques, suffisantes à son utilisation.

Il vous sera cependant nécessaire d'avoir un minimum de connaissances en électronique et informatique afin de comprendre l'intégralité de ce livre. J'espère qu'il vous permettra de fabriquer de nombreux systèmes, de toutes sortes, avec pour principale limite, votre imagination.

Ce livre s'adresse donc aussi bien aux amateurs avertis, qu'aux étudiant(e)s et/ou professionnels.

Bonne lecture !!!

STRUCTURE INTERNE D'UN PIC



SOMMAIRE

CHAPITRE 1 : LE PIC 16F84

1- LE PIC 16F84

1.1- Présentation du microcontrôleur :

Deux exemples de microcontrôleurs

Aujourd'hui, les microcontrôleurs sont partout : ordinateurs, portable…

Assez facile d'utilisation, pour la plupart, et programmable de fort nombreuses fois (plus de 1000), ou définitivement (OTP : One Time Programmable), leur souplesse d'utilisation a séduit rapidement les divers constructeurs de divers domaines. Aujourd'hui, un des géants mondiaux s'appelle Microchip.

Mais il ne faut surtout pas confondre les microcontrôleurs et les microprocesseurs. Pour résumer, on peut dire qu'un microcontrôleur est un ordinateur extrêmement miniaturisé et possédant donc assez peu de mémoire, et dont le processeur est relativement simple, alors qu'un microprocesseur ne fait qu'exécuter des instructions qui lui sont communiquées, puis renvoie les résultats.

Les principaux problèmes des microcontrôleurs sont la taille de leur mémoire et le nombre limité de périphériques qu'ils peuvent recevoir en même temps. Cependant, le nombre de ces derniers peut parfois être augmenté en associant, sur les mêmes pattes un périphérique d'entrée et un de sortie, permettant alors de doubler le nombres de périphériques connectables…

Les microcontrôleurs sont tellement miniatures, que la plupart du temps, ils sont implantés sur l'application même qu'ils sont censés piloter, comme par exemple un clavier d'ordinateur, ou la souris.

Ces systèmes sont alors appelés « systèmes embarqués ». Ils exécutent tout le temps, en boucle, le même programme.

Leur champ d'application ne connaît comme limite, que l'ingéniosité des divers concepteurs. Grâce à eux, la réception radio est de plus en plus fine. Leur avenir s'annonce radieux en ce début de XXI éme siècle. Pour l'an 2000, le bilan était très positif :

- Communication : 30%
- Consommation générale : 27%
- Industrie automobile : 18%
- Périphériques informatiques : 15%
- Industrie : 10%

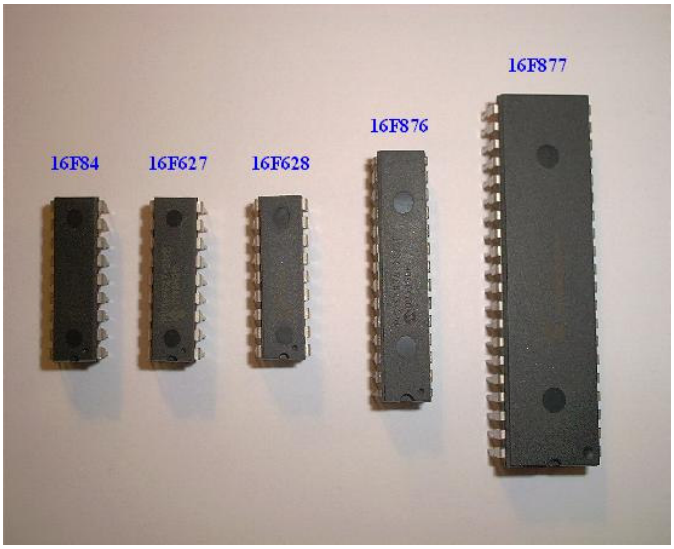
Nous pouvons en effet rencontrer des microcontrôleurs tous les jours dans les appareils que nous utilisons.

La classifications générales de puces se fait par nombre de bits : 4, 8, 16, 32 bits.

Un microcontrôleur se décompose en diverses parties : la mémoire de programme, la mémoire de données, le processeur, les ressources auxiliaires.

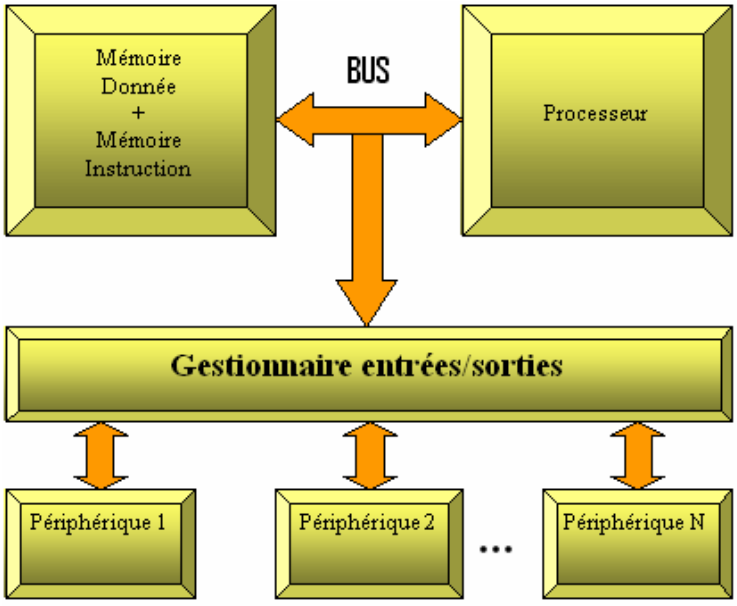
Les modèles les plus vendus sont les PIC de chez Microchip, en raison d'un excellent rapport qualité/prix, et surtout le PIC 16F84, qui est le sujet de ce livre, et qui est utilisé par toute la communauté électronique, que ce soit amateur, ou même par certains professionnels. Facile à programmer et à utiliser, son prix tourne aux alentours des 7.5€, avec une cadence de fonctionnement pouvant aller jusqu'à 20 MHz.

Chez les PIC, il y a la gamme de base (16C5X : 33 instructions de 12 bits, DIL 18 et 28), la gamme intermédiaire (16C/FXXX : 72 modèles DIL 18 à 68, 35 instructions de 14 bits), la gamme mini (12CXXX : DIL 8), la gamme supérieure (58 instructions de 16 bits), la gamme avancée (18CXXX : 10 MIPS à 40 MHz, 77 instructions de 16 bits, DIL28 à 80, 16 Kmots de 16 bits).



Des exemples de la famille PIC de chez Microchip

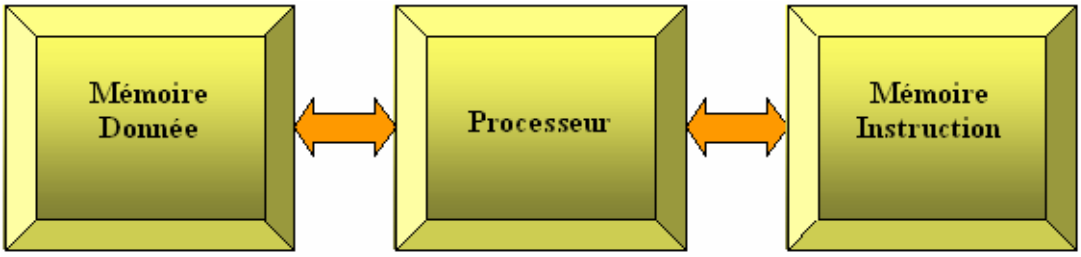
Jusqu'à une certaine époque, les microcontrôleurs respectaient l'architecture Von Neumann, inventeur de l'ENIAC, premier ordinateur au monde. Cependant, celle-ci présente des inconvénients. En effet, la vitesse d'exécution est limitée, les instructions et les données transitaient par le même bus. Pour résumer, son principal défaut était le fait qu'elle ne possédait qu'un bus pour, simultanément, la mémoire programme et la mémoire donnée.



L'architecture de Von Neumann

D'où l'architecture Harvard, utilisée maintenant par les PIC. Sa particularité tient dans le fait qu'il y a deux mémoires accessibles en même temps par le processeur, par l'intermédiaire de deux bus spécifiques.

L'un sert pour les données, et l'autre pour les instructions. De ce fait, les deux peuvent être accessibles en même temps, d'où un gain de vitesse, au niveau exécution.



L'architecture Harvard

Ceci aidant, il existe depuis quelques années, un nouveau type de mémoires dites « flash », avec écriture et effacement électrique des données dans la mémoire. Elle est de type RAM, mais est associée à une mémoire E²PROM pour des données auxiliaires.

Cependant, d'autres architectures existent, mais néanmoins moins répandues, comme par exemple, l'architecture Flynn développée en 1996.

Chaque microcontrôleur possède des registres. Il s'agit de mémoires intervenant dans les opérations de l'UAL (Unité Arithmétique et Logique). Celle-ci requiert deux opérandes (instructions) pour réaliser une opération.

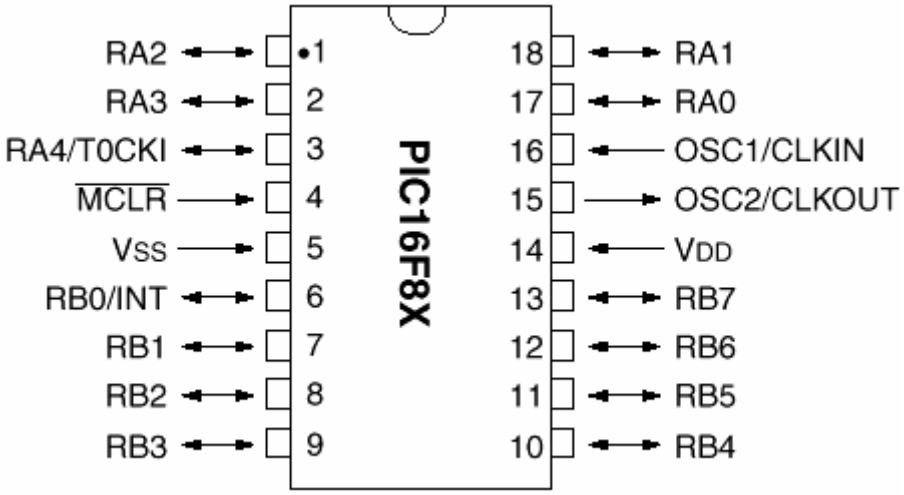
Quand un processeur exécute une instruction, il le fait en deux étapes : d'abord une phase de recherche, puis une phase d'exécution.

Tous les microcontrôleurs, ou presque, possèdent des entrées/sorties programmables. Celles-ci se configurent simplement sur les PIC en entrant une valeur à l'adresse du port correspondant.

En plus du programme, les microcontrôleurs ont besoin de ce qu'on appelle un mot de configuration, tenant compte du type d'oscillateur (RC, XT,…), de l'activation ou non du Watchdog (qui empêche un

dysfonctionnement), d'une temporisation au démarrage ou non, et du code P (non utilisé et explicité dans ce livre, le code P, pour Protection, sert à verrouiller la programmation du PIC ; c'est-à-dire que pour le programmer, il faut un mot de passe).

1.2- Caractéristiques du PIC 16F84 :



Ce microcontrôleur, que nous surnommerons « le pic » pour des raisons de commodité, dans ce livre, possède 13 broches configurables, réparties sur deux ports : le port A et le port B.

***Remarque :* On ne peut affecter que deux valeurs différentes de configuration à chaque patte : un '1' pour la mettre en entrée, ou un '0' pour une sortie.**

Le port A possède 5 broches (nommées RA1 à RA4), mais la quatrième, également appelée TOCKI peut servir pour une éventuelle temporisation externe.

Le port B, lui, possède 8 broches (de RB0 à RB7) ; mais la broche RB0 peut également servir comme interruption éventuelle (un peu comme un garde sur un évènement)

La broche 4, le MCLR barre, sert à indiquer au PIC s'il est en fonctionnement normal (un '1' logique) ou alors s'il est en cours de programmation (un '0' logique). Cette broche sert également à un éventuel Reset du PIC.

Ne reste que 4 broches : l'alimentation (la 5 (0 volts) et la 14(+5 volts))
l'oscillateur(pattes 16 et 15)

Outre ces caractéristiques, il faut savoir que le PIC 16F84 n'utilise pas de signaux analogiques, mais uniquement numériques (logique). De

plus, il faut savoir, qu'il ne fait pas la différence entre les niveaux logiques TTL et CMOS. Il est donc alors préférable d'utiliser les deux extrêmes de signaux logiques TTL (0 et 5 volts), pour éviter tout problème possible.

Voici enfin, les caractéristiques du PIC 16F84 fournit par Microchip :

- Mémoire de programme : 1KO, type Flash
- Mémoire de données RAM : 68 octets
- Mémoire de données E²PROM : 64 octets
- Niveau de la pile : 8
- Jeux d'instruction RISC : 35 de 14 bits
- Temps d'exécution des instructions normales : 4*Tosc
- Temps d'exécution des instructions de saut : 8*Tosc
- Cause d'interruption : 4
- Fréquence max de travail : 10 MHz
- Lignes E/S numérique : 13
- Temporisateur : un pour l'utilisateur, un pour le Watchdog
- Tension d'alimentation : 2 à 6 V continu
- Tension de programmation : 12 à 14 V continu
- Boîtier : DIL 18

Enfin, autre point fort du PIC : la consommation, car en tant que système embarqué, il faut que ce microcontrôleur consomme peu.

Ainsi, des 2 mA de consommation en fonctionnement normal, il peut passer à 10µA en fonction veille ou sommeil, comme par exemple sur un téléphone portable, quand on ne s'en sert pas, la lumière reste éteinte : il est en veille. En revanche, au moindre appui sur un bouton (interruption externe, broche RB0 ici), il se remet en marche. Le Pic fonctionne de la même manière.

***Remarque :* il faut cependant savoir que le PIC peut fournir jusqu'à 20 mA par sortie, soit, si tous les ports sont en sortie, un débit de 260 mA. Il est aisé de comprendre alors que les 2 mA sont une consommation moyenne et que celle-ci varie selon l'utilisation du PIC**

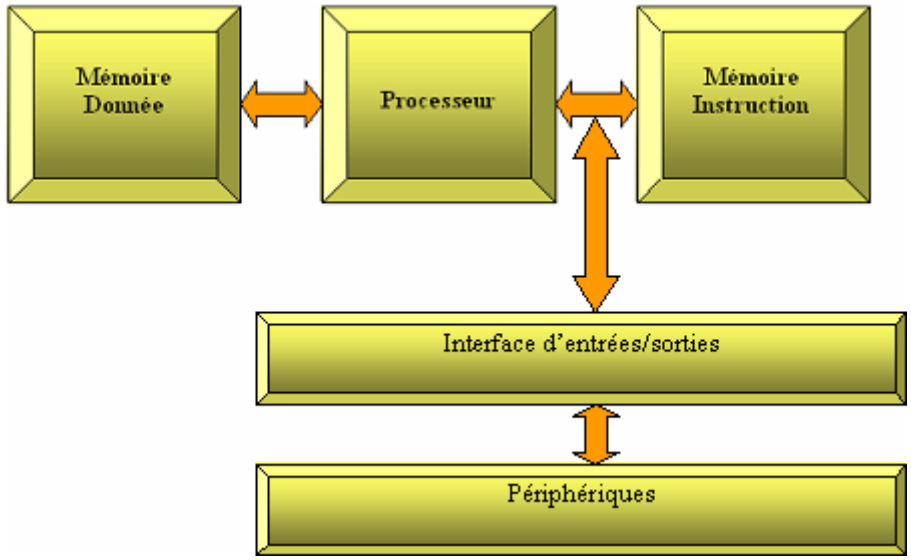
1.3- Fonctionnement du PIC 16F84 :

Dans cette partie du livre, nous traiterons des principaux systèmes du PIC, plus ou moins utiles. Nous illustrerons ces systèmes par des schémas, les plus clairs et les plus compréhensibles possible. Certains de ces systèmes, étant des paramètres configurables, devront faire l'objet d'attention selon que vous déciderez où non de les mettre en œuvre.

Nous verrons donc :

›les entrées/sorties
›les différents types d'oscillateurs possibles (pour l'horloge)
›le Reset
›la mémoire E²PROM
›la mémoire flash, utilisée dans le PIC
›les interruptions
›le TMR0
›le Watchdog
›le RTCC

1.3.1- Les entrées/sorties :



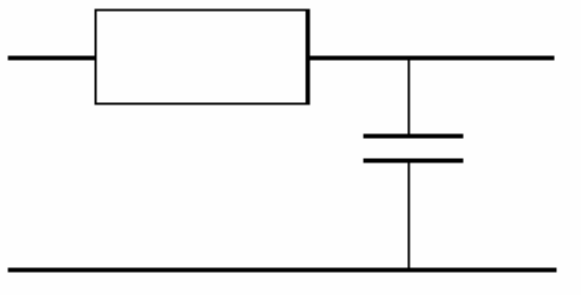
Les entrées/sorties sont l'interface entre le système embarqué et le monde extérieur.

Les entrées/sorties sur le Pic sont réparties sur deux ports : le port A (5 E/S) et le port B (7 E/S). Pour configurer ces dernières, il s'agit de rentrer un '1' pour la mettre en entrée, ou un '0' pour une sortie. Quand chaque bit est déterminé, on obtient alors le mot binaire à rentrer dans le registre de configuration.

1.3.2- Les oscillateurs :

Ils peuvent être de plusieurs types. Nous verrons donc ceux-ci en détail, afin de pouvoir choisir le meilleur possible.

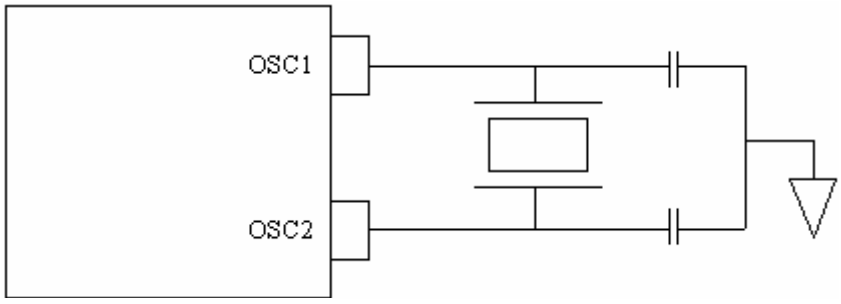
●RC :



Cet oscillateur coûte peu cher à fabriquer, mais n'est pas très précis. Il peut convenir si votre système n'a pas besoin d'une horloge précise. La constante de temps est alors : ζ=RC

Remarque : $F=1/\zeta$

●Systèmes à quartz :



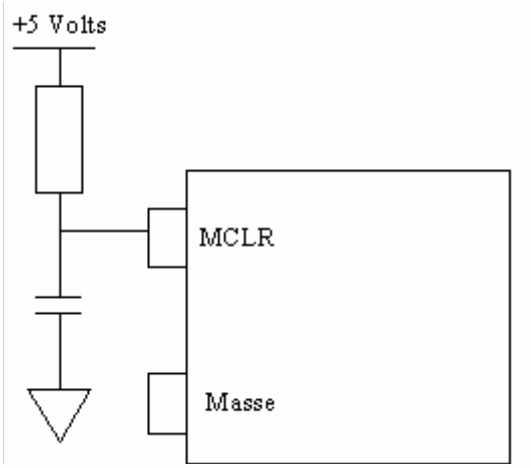
Ce système peut se diviser en trois sous-systèmes : HS, XT et LP

La plupart du temps, l'oscillateur utilisé sera un XT, un quartz à 4 MHz. Cependant, le Pic peut monter jusqu'à 20 MHz en utilisant un mode HS (High speed). Quand au mode LP, il s'agit du mode de fonctionnement en basse fréquence, peu utilisé.

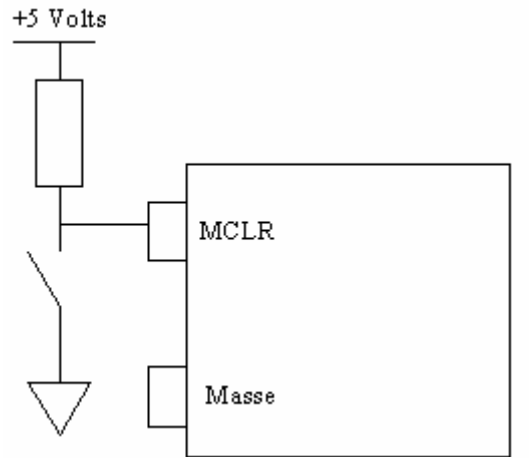
1.3.3- Le reset :

Ce système est relativement important. En effet, lorsque le système a des ratés, ou plante, il est utile de pouvoir le réinitialiser. Il est également possible et utile parfois d'effectuer un reset lors de la mise sous tension.

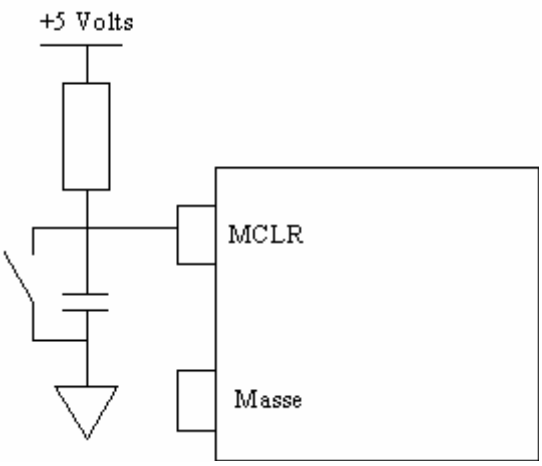
Le reset s'effectue en mettant un '0' logique sur la patte 4 du microcontrôleur. Il existe plusieurs types de système de reset : le manuel, l'automatique, et un mixte.



Le reset automatique (condensateur de 1µF et une résistance de 1 K)



Le reset manuel (avec une résistance de 1K)



Le modèle mixte, permettant un reset automatique lors de la mise sous tension et également un interrupteur de reset manuel

1.3.4- La mémoire E²PROM :

L'E²PROM(Electrically Erasable Programmable Read Only Memory) est une mémoire interne au Pic. Il s'agit d'une mémoire non volatile dans laquelle le PIC peut stocker des données, comme par exemple les résultats d'une acquisition.

1.3.5- La mémoire flash :

La mémoire flash est le nouveau type de mémoire E²PROM. Bien plus souple que les premières générations de ces dernières, la mémoire flash permet une écriture/effacement de toute la mémoire ou que d'une partie au choix. Ce type de mémoire possède beaucoup de caractéristiques intéressantes.

Caractéristiques de la mémoire Flash
1-Pas de différence significative entre les différents types de mémoires existantes sur le marché
2-Utilisation simplifié par rapport aux autres mémoires E²PROM
3-Faible coût des outils de développement
4-Simplification du débogage
5-Possibilité de misa a jour du Firmware
6– Grande plage de tension de programmation.

Caractéristiques	Mémoire flash
Alimentation	2-5,5 volts
Tension de programmation	Vdd
Autoprogrammable	OUI
Débogage en circuit	OUI
Technologie	0,5µm
Cycles d'effacement/écriture	100K(données), 1K(programme)
EEPROM Données	OUI
Temps de programmation/cycle	1-2 ms

1.3.6- Les Interruptions :

Une interruption ou IT est un sous programme prioritaire, occupant l'adresse 04. Quand une interruption intervient, le programme principale finit sa tâche en cours, puis va effectuer le sous-programme d'interruption.

Il existe quatre modes d'interruptions :
► Une modification de RB0 (en entrée)
► La fin de l'écriture en E_PROM
► Le débordement du timer interne
► Une modification de l'état des pattes RB4 à RB7 (configurées en entrées)

C'est le registre INTCON (interruption control) qui sert à gérer les interruptions.

1.3.7- Le TMR0 :

Il s'agit d'un temporisateur interne de 8 bits, qui peut être initialisé à une valeur donnée. A chaque passage de FF à 00 (en hexa), le bit de débordement est activé. Il faut alors le remettre à zéro, pour pouvoir détecter un autre débordement (non automatique).

Il possède deux modes de fonctionnement possible, dont le choix s'effectue par la mise à 1 ou à 0 du bit TOSC (voir chap. nécessaire à la programmation), l'entrée horloge devenant alors la patte RA4, en mode dit « TOCKI ».

Ces deux modes sont :

► temporisateur interne (peut alors servir pour des fonctions de temps)
► Compteur d'évènements (peut servir pour compter des évènements extérieurs par l'intermédiaire de RA4)

Remarque : la patte RA4 doit être définie en entrée dans le cas du compteur d'évènements.

1.3.8- Le Watchdog :

Littéralement le « chien de garde », le Watchdog est un système de surveillance du bon déroulement du programme. Il s'agit d'un compteur, qui est réinitialisé régulièrement dans le cas d'un fonctionnement normal. Mais dans le cas d'un dysfonctionnement, le compteur va jusqu'au bout et déclenche alors un reset interne, par débordement, réinitialisant le Pic.

Le compteur peut fonctionner à la fréquence de l'oscillateur, ou bien à une fréquence spécifique, désignée par la fréquence de l'oscillateur modifiée par un diviseur.

Remarque : il n'est pas obligatoire de l'utiliser. Son utilisation est à activer ou non, lors de la programmation du PIC

1.3.9- Le RTCC:

Il s'agit d'une horloge interne destinée au fonctionnement du timer 0 (TMR0) dans le cas d'un fonctionnement de ce dernier sur horloge interne.

CHAPITRE 2 : PROGRAMMATION DU PIC 16F84

2.1- Nécessaire à la programmation du PIC :

Quand on veut programmer un PIC, il est nécessaire d'avoir un minimum de documents avec soi. Cependant, si ces derniers sont importants pour programmer en langage assembleur, nous verrons plus tard qu'en langage C, certains deviennent quasi inutiles.

Ces documents, peu nombreux, traitent des tableaux de pages de programmation, de la répartition des ports en entrées/sorties et du type d'oscillateur utilisé (en effet, dans le cas par exemple d'un oscillateur RC, l'instruction SLEEP est alors utilisable)

Dans un premier temps, nous verrons donc les tableaux de programmation, et nous spécifierons les parties inutiles et utiles, en détaillant ces dernières.

Voici le tableau de programmation 0 :

	ADRESSES HEXA	NOM	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Page 0	00	INDF	Utilise le contenu de FSR pour adresser la mémoire							
	01	TMR0	Horloge/compteur en temps réel							
	02	PLC	Octet de moindre poids du PC							
	03	STATUS	IRP	IRP1	RP0	TO#	PD#	Z	DC	C
	04	FSR	Adressage indirect avec INDF							
	05	PORTA	-	-	-	RA4/ Tock	RA3	RA2	RA1	RA0
	06	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0/ INT
	07		Non implémenté, lu comme étant à 0							
	08	EEDATA	Registre de données E²PROM							
	09	EEADR	Registre d'adresses E²PROM							
	0A	PCLATH	-	-	-	S'écrit sur les 5 bits de PCH				
	0B	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF

Dans ce tableau, nous ne nous réfèrerons qu'au ligne : STATUS
TRISA
TRISB

Les autres lignes entrant dans des modes spécifiques du PIC, ou servant juste à indiquer dans quel état se trouve le PIC, nous ne les détaillerons pas.

***Remarque* : Il est à noter que le port A se trouve à l'adresse 05 et le port B à l'adresse 06**

STATUS :

Ce registre contient tous les bits qui servent pour nous renseigner sur l'état du PIC, et pour une partie de la programmation. Seul trois bits nous intéressent : le C, le RP1 et le RP0.

Le bit C (pour carry, retenue en français), est à un ou zéro selon que l'opération effectuée nécessitait une retenue ou non.

Le RP0 et le RP1 sont les bits permettant d'avoir accès au tableau de programmation, vu précédemment (et donc au bit système des registres) et aux différents espaces mémoires.

Ils respectent le tableau suivant :

RP1	RP0	PAGE	ADRESSES Hexa
0	0	O	00 à 7F
0	1	1	80 à FF
1	0	2	100 à 17F
1	1	3	180 à 1FF

***Remarque :* Les adresses 0C à 2F sont réservées pour les registres banalisés. Stockés en RAM, ils n'ont pas de but précis et peuvent être utilisés pour diverses applications.**

TRISA :

Le TRISA est le registre permettant de définir les entrées/sorties du Port A. En remplaçant les RAx par leur valeur correspondante (1 pour une entrée, 0 pour une sortie), on obtient alors le mot binaire à rentrer dans TRISA. Quand au bit 7, 6 et 5, on peut leur donner la valeur que l'on veut, selon que ça nous arrange ou non.

TRISB :

Le TRISB fonctionne de la même manière que le TRISA, mais avec 2 bits supplémentaires

Remarque : Bien que dans notre tableau page 0, il est spécifié PORTA et PORTB et que comme nous allons le voir juste après, dans le tableau page 1, il est indiqué TRISA et TRISB, ceux-ci, sont respectivement identiques. De ce fait, nous pouvons rentrer le code binaire de programmation des E/S dans la page 0 ou 1. Il est cependant conseillé de passer à la page 1 pour les programmer pour éviter certains problèmes.

Nous allons à présent voir le tableau de programmation 1 :

	ADRESSES HEXA	NOM	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Page 1	80	INDF	Utilise le contenu de FSR pour adresser la mémoire							
	81	OPTION	RBPU#	INTDG	TOCS	TOSE	PSA	PS2	PS1	PS0
	82	PLC	Octet de moindre poids du PC							
	83	STATUS	IRP	IRP1	RP0	TO#	PD#	Z	DC	C
	84	FSR	Adressage indirect avec INDF							
	85	TRISA	-	-	-	Configuration du PORT A				
	86	TRISB	Configuration du PORT B							
	87		Non implémenté, lu comme étant à 0							
	88	EECON1	-	-	-	EEIF	WRERR	WREN	WR	RD
	89	EECON2	Registre de commande E²PROM							
	0A	PCLATH	-	-	-	Configuration du PORT A				
	0B	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF

Les registres qui vont nous intéresser ici sont : EECON
OPTION
INTCON

Nous ne nous attarderons pas, mais ces registres EECON servent à contrôler l'écriture et la lecture dans l'E²PROM. Ils doivent être utilisés avec les registres EEDATA et EEADR du tableau de programmation 0.

A noter la présence de TRISA et TRISB, qui ont fait l'objet d'une remarque précédemment.

OPTION :

Ce registre permet de définir le choix d'un diviseur et son affectation au Watchdog ou au RTCC

Le choix de ce diviseur, se fait par l'intermédiaire de PS2, PS1, et PS0, selon le tableau suivant :

PS2	PS1	PS0	Rapport de division	
			WDT	RTCC
0	0	0	:1	2
0	0	1	2	4
0	1	0	4	8
0	1	1	8	16
1	0	0	16	32
1	0	1	32	64
1	1	0	64	128
1	1	1	128	256

L'affectation au Watchdog (WDT) ou à l'horloge temps réel (RTCC) se fait par l'intermédiaire du bit PSA.

Dans le cas de l'affectation du diviseur au RTCC, le bit TOSE permet de choisir entre un déclenchement sur front montant ou sur front descendant.

Le bit INTEDG permet lui, de choisir le déclenchement de l'interruption (sur front montant ou sur front descendant).

Enfin, le bit /RBPU, permet de choisir de la mise en place ou non d'une résistance de rappel entre RB4 et RB7.

Remarque : Dans le cas d'un éventuel reset, ou à la mise sous tension, tous les bits sont par défaut à '1'.

Tout ceci est résumé dans le tableau suivant :

Bit PSA	0 = Choix du RTCC
	1 = Choix du WDT
Bit TOSE	0 = Front montant
	1 = Front descendant
Bit TOCS	0 = Horloge interne
	1 = Front sur la broche RTCC
Bit INTEDG	0 = interruption (RB0) validé sur front descendant
	1 = interruption (RB0) validé sur front montant
$\overline{\text{Bit RBPU}}$	0 = Connection d'une résistance de rappel entre RB4 et RB7
	1 = Aucune résistance de rappel

INTCON :

Bit du registre INTCON	NOM	Fonction
Bit 7	GIE (Global Interrupt Enabled Bit)	Bit de validation général des interruptions (à 1 pour activer les interruptions)
Bit 6	EEIE (E²PROM Write Complète Interrupt Enabled Bit)	Interruption de la fin d'écriture en E²PROM
Bit 5	TOIE (Timer 0 Interrupt Enabled Bit)	Interruption pour le Timer 0
Bit 4	INTE (INTerrupt pin Enabled Bit)	Interruption sur la patte RB0
Bit 3	RBIE (RB port change Interrupt Enabled bit)	Interruption sur les pattes RB4 à RB7
Bit 2	TOIF (Timer 0 Interrupt Flag bit)	Signal que l'interruption vient du débordement du Timer 0
Bit 1	INTF (INTerrupt pin Flag bit)	Signal que l'interruption provient d'un changement d'état de RB0
Bit 0	RBIF (poRt B Interrupt Flag bit)	Signal que l'interruption provient des pattes RB4 à RB7

Ce tableau résume la façon dont on peut activer et vérifier les interruptions actives.

2.2- Choix d'un langage de programmation :

Quand on choisit de programmer un microcontrôleur, il faut savoir que sauf exceptions, cas rare, nous avons le choix entre plusieurs langages de programmation. En ce qui concerne le PIC, les programmes se développent avec le logiciel MPLAB (dont nous verrons le fonctionnement plus loin dans ce livre), qui nous permet de programmer le PIC dans divers langages. Parmi ceux-ci, les principaux sont le C et le BASIC pour les langages hauts niveaux, et l'assembleur pour les langages bas niveaux.

Les langages bas niveaux sont les plus proches de la machine, généralement l'équivalent de la « langue natale » du système. L'assembleur en fait partie. Proche de la machine, ce langage permet d'optimiser le programme. Il reste cependant difficile à mettre en œuvre pour des programmes complexes, mais abordable pour des programmes de base. Adopté par nombre d'industriels comme langage de référence, il reste cependant difficilement réparable pour une tierce personne.

Les langages hauts niveaux sont des langages faciles à comprendre et à mettre en œuvre, avec un minimum de connaissance. Ce type de langage facilite la maintenance lors de problèmes, car il est aisé pour quelqu'un, pris au hasard, de comprendre un programme haut niveau. Ce type de langage est, en effet, plus proche de l'utilisateur que de la machine. Ces langages utilisent ce qu'on appelle des « compilateurs », sorte de convertisseurs, dictionnaires automatiques. Ces compilateurs font la conversion directe entre le langage haut niveau et le langage assembleur, nous épargnant ainsi une difficulté supplémentaire.

En comparant ces deux types de langage, et dans la mesure où les compilateurs sont, de nos jours, performant, il apparaît comme plus pratique de choisir le langage C, plutôt que le langage assembleur, tant pour la simplicité de programmation, que pour la facilité d'utilisation.

Ainsi, nous verrons dans ce livre, les bases de l'assembleur pour PIC (Instructions de commande) à fin d'informations, et éventuellement de comprendre certains programmes.

Puis nous nous attarderons sur le langage C, langage de programmation choisi. Nous commencerons par voir les bases

classiques du C général, puis nous verrons ensuite le C PIC, spécifique au PIC.

2.3- Le langage assembleur (bases) :

Le langage assembleur est ce qu'on appelle un langage « bas niveau », c'est-à-dire que c'est un langage extrêmement proche du fonctionnement du système. Très difficile à appréhender, il est cependant nécessaire d'avoir des bases en assembleur, car, cela permet d'optimiser les programmes et de gagner de la place, chose importante dans certains cas, car celle-ci est précieuse sur le PIC.

Il faut savoir que quand, sur les documentations techniques, on lit le mot « instruction », celles-ci font référence à une instruction assembleur. C'est le langage par défaut du PIC, sa « langue natale ».

Cependant, nous nous attarderons bien plus sur le langage C, successeur à l'assembleur, et bien plus compréhensible. Nous ne verrons donc en assembleur, que les 35 instructions du PIC avec quelques explications sur chacune d'entre elles.

Voici donc pour commencer les instructions du PIC. Il n'y en a que 35, et le PIC ne se programme qu'avec elles. Les registres (W et f) contiennent des littérals (k), un peu comme un dossier (registres) contient des fichiers (littérals), mais où ces derniers seraient du texte.

Mnémonique	Opérande 1 (source)	Opérande 2 (cible)	Opération réalisée	Nombre de cycle
addlw	k		Ajoute k à w	1
addwf	f	d	Ajoute f à w. Si d=0, le résultat est mis dans w; si d=1, le résultat est mis dans f	1
andlw	k		Réalise un ET logique entre w et k	1
andwf	f	d	f&w, si d=0, le résultat est mis dans w, dans f sinon	1
bcf	f	b	Mise à O du bit b de f	1

bsf	**f**	**b**	**Mise à 1 du bit b de f**	**1**
btfsc	**f**	**b**	**Si le bit b de f est nul, l'instruction suivante est ignorée**	**1-2**
btfss	**f**	**b**	**Si le bit b de f est à 1, l'instruction suivante est ignorée**	**1-2**
call	**k**		**Appel le sous programme d'étiquette k. A utiliser avec le mnémonique *return***	**2**
clrf		**f**	**Charge la valeur 00 dans f**	**1**
clrw			**Charge la valeur 00 dans w**	**1**
clrwdt			**Efface la valeur du chien de garde**	**1**
comf	**f**	**d**	**Complément à 1 de f. Si d=0, le résultat est mis dans w; dans f sinon**	**1**
decf	**f**	**d**	**Décrémente f d'une unité. Si d=0, le résultat est mis dans w; si d=1, le résultat est mis dans f**	**1**
decsfz	**f**	**d**	**Décrémente f d'une unité. Si le résultat est 0, l'instruction suivante est ignorée. Si d=0,**	**1-2**

			le résultat est placé dans w, dans f si d=1	
goto	k		Branchement inconditionnelle à l'étiquette k	2
incf	f	d	Incrémente f d'une unité. Si d=0, le résultat est mis dans w; si d=1, le résultat est mis dans f	1
incsfz	f	d	Incrémente f d'une unité. Si le résultat est 0, l'instruction suivante est ignorée. Si d=0, le résultat est mis dans w; dans f sinon	1-2
iorlw	k		OU logique entre k et w	1
iorwf	f	d	f +w, si d=0, le résultat est mis dans w, dans f sinon	1
movf	f	d	Déplace f vers w si d=0, vers f si d=1	1
movlw	k		Déplace k dans w	1
movwf		f	Déplace w dans f	1
nop			Instruction vide	1
retfie			Retour de sous programme d'interruption	2

retlw	**k**		**Retour de Sous programme, avec chargement de k dans w**	**2**
return			**Retour de sous programe**	**2**
rlf	**f**	**d**	**Rotation d'un bit vers la gauche de f. Si d=0, le résultat est mis dans w, dans f sinon**	**1**
rrf	**f**	**d**	**Rotation d'un bit vers la droite de f. si d=0, le résultat est mis dans w, dans f sinon**	**1**
sleep			**Mise en sommeil du pic (basse consommation)**	**1**
sublw	**k**		**Soubstrait w à k (k-w)**	**1**
subwf	**f**	**d**	**f-w, si d=0, le résultat est mis dans w; dans f sinon**	**1**
swapf	**f**	**d**	**Inversion des 4 bits de poids forts, avec les 4 bits de poids faibles**	**1**
xorlw	**f**	**d**	**f+w exclusif, si d=0, le résultat est mis dans w, dans f sinon**	**1**
xorlw	**k**		**OU exclusif entre k et w**	**1**

Remarque : *la fonction sleep ne fonctionne que si l'oscillateur est de type RC*

2.4- Le langage C :

Les bases du langage C :

Dans cette partie, nous verrons les bases du langage C pour ordinateur, en nous limitant toutefois aux points communs avec le langage C PIC, c'est-à-dire aux commandes communes.

Il est cependant à spécifier, que pour des raisons de commodité, nous verrons quelques points en plus, afin d'expliquer clairement le langage C ; points existant en C PIC, mais sous formes différentes. Gardons donc à l'esprit que les bases du C sont ici données, afin d'apprendre à maîtriser ce langage haut niveau afin de savoir, plus tard, programmer les PICs avec le plus de facilité possible. A noter tout de même, que la meilleure école pour maîtriser un langage reste la pratique de celui-ci et l'étude de programme.

Un programme en langage C se compose de plusieurs parties :
-Un appel de librairies
-Une fonction « main », correspondant au programme principal
-Eventuellement de sous-programmes (auxquels cas, à déclarer)

Les librairies :

Elles constituent une sorte d'encyclopédie pour le programme principal. En effet, elles contiennent toutes les infos et directives dont a besoin le programme principal : le main. Le main a accès à ces directives par l'intermédiaire d'instructions spécifiques aux librairies en question.

Pour résumer, et faire simple, le main appelle les fonctions dans les librairies par leur nom. On les appelles avec la commande « #include<nom.h>».

La fonction main :

Il s'agit du programme principal. C'est dans ce programme qu'on écrira notre code informatique. Cette fonction peut faire appel à des sous fonctions, lesquelles peuvent être situées dans des librairies ou derrière le main.

Remarque : ***Une fonction est composée de l'en-tête comprenant son nom et éventuellement ses paramètres, et d'un bloc, commençant par {et finissant par }, indiquant le code correspondant à l'en-tête***

Les sous-programmes, ou sous-fonctions :

Il s'agit de programmes secondaires, sensés aider le main dans son travail. L'utilité de créer des sous fonction apparaît surtout lorsqu'une tache se répète plusieurs fois dans le même programme : plutôt que d'écrire par exemple 3 fois le même code, on définit une seule fois le code, dans un sous programme, et on l'appelle trois fois. On économise ainsi de la place (optimisation du code), chose très importante pour un microcontrôleur. La création possible de sous programme doit être une des choses à avoir à l'esprit lorsqu'on cherche à optimiser le code. Lorsque la possibilité se présente, il faut alors juger s'il est plus judicieux de créer un sous programme, ou alors de laisser le code tel quel, en se référençant à la longueur du programme.

Les bases du langage C :

Quelques règles :

-Une ligne se finit toujours (sauf cas particulier que nous verrons) par un point virgule « ; »

-Il est jugé appréciable d'écrire avec l'aide des tabulations, pour avoir un code lisible.

Les commentaires :

Lorsque l'on désire faire des commentaires en langage C, il y a deux solutions :

-Pour une ligne seule, on utilise un double slash : « // »
Il n'est pas nécessaire de mettre un point virgule à la fin du Commentaire.

- Pour un paragraphe, ou plusieurs lignes consécutives, on utilise Un « /* » pour indiquer le début du commentaire, et un « */ » pour en indiquer la fin.

Exemple : a=b ; //voici un commentaire sur une ligne
*/*Voici un commentaire*
sur plusieurs lignes/*

***Remarque :* Le commentaire de paragraphe peut également servir pour mettre de coté une partie de code sans l'effacer**

Les variables :

Dans un programme, on utilise souvent des variables pour effectuer des opérations (ex : a=b). Il faut savoir que le langage C fait la différence entre majuscule et minuscule ; et que les variables sont la première chose à définir dans un programme. Elles peuvent principalement être de trois types : int (pour les entiers), float (pour les chiffres à virgule), char (pour les caractères). *Les variables doivent être obligatoirement définies au début du bloc.*

Exemple : int a, b ;
float c ;

***Remarque :* Il est possible d'affecter une valeur à une variable lors de sa définition (ex : int a=2 ;//créé a comme entier, et lui donne la valeur 2)**

***Remarque :* quand on veut rentrer un caractère dans un char, il faut utiliser des guillemets (ex : char c ; c=''A'' ;)**

Les formes d'écriture :

Les deux utiles sont :
-décimal (format par défaut)
-hexadécimal (il faut marquer 0x devant la valeur pour spécifier le type)

***Remarque :* On peut utiliser pour les float l'exponentiel. Il s'écrit « e » (ex : 1.6e+3)**

Les opérateurs :

Les opérateurs sont :
+ : addition
- : soustraction
* : multiplication
/ : division
% : modulo (ex : 11%3=2)

= : égale
!= : différent
>,< : supérieur et inférieur

***Remarque :* Si on fait une opération sur des floats dont le résultat est un float, lequel est placé directement dans un int, le résultat sera de type entier (on ne verra rien après la virgule)**

Il existe certaines subtilités sur les opérateurs. Ainsi mettre ++ ou – équivaut à, respectivement, ajouter ou soustraire '1' à la valeur ou variable.

De même, marquer a=b, équivaut à dire que a est égal à b ; mais marquer a==b, consiste à comparer a à b, pour savoir s'ils sont égaux.

Les opérateurs logiques :

Ils servent à effectuer tous types d'opérations logiques. Il y a le ET (&&), le OU inclusif (||) et le NON (!).

Les opérateurs conditionnels :

Ex : si X=Y alors A=B, sinon A=C
Se traduit par :

```
IF(X=Y)
{
        A=B ;
}
ELSE
{
        A=C ;
}
```

***Remarque :* les incollades sont obligatoires. Elles indiquent que ce qu'il y a à l'intérieur fait partie de l'instruction rentrée avant. Dans le cas ou il n'y aurait qu'un seule ligne (comme ici), on peut toutefois les enlever.**

Ex : Surveille A ; si A=1, alors B=1 ; Si A=2, alors B=2…
Se traduit par:

```
SWITCH (A)
{
        CASE '1': B=1;
                  BREAK;
        CASE '2': B=2;
                  BREAK;
}
```

Ex : fait A=B*C, tant que A supérieur à 3
Se traduit par :

```
DO
{
        A=B*C ;
}
WHILE(A>3) ;
```

Ex : La boucle FOR, étant difficile à traduire, nous allons la présenter, et expliquer son fonctionnement.

```
FOR(i=0 ;i<4 ;i++)
{
        A=B*C ;
}
```

La boucle FOR présente trois paramètres p1, p2, et p3 :
FOR(p1 ;p2 ;p3)

p1 ne sert que lorsque l'on rentre dans la boucle. Il sert à initialiser des valeurs. P& peut être vide s'il ne sert à rien.

p2 est la condition qui permettra de mettre fin à la boucle.

p3 est un paramètre où l'on effectue des opérations sur la variable de condition. Ces opérations peuvent être de différents types.

La fonction FOR fonctionne ainsi. On rentre dans la boucle, le paramètre p1 est effectué. Puis, on vérifie p2. Si elle est fausse, on

effectue les action entre incollades ; puis on effectue p3, et on retourne vérifier p2. Si p2 est vérifiée, alors, on sort de la boucle FOR, sinon, on recommence les actions, et p3, puis p2, jusqu'à ce que p2 soit vérifiée.

Exemple de programme pour résumer tout ce qu'on a vu :

```
/* Création d'un programme qui attribue une lettre à une variable selon le résultat d'une opération*/

//déclaration des librairies
#include<stdio.h>

//programme principal
Void main(void)
{
	//déclaration de variable
	Int A, b, c=3 ;
	Char car ;

	//boucle FOR
	For(b=0 ;b<3 ;b++)
	{
		A=b*c ;
		Switch(A)
		{
			Case '0' : car='A' ;
				Break ;
			Case '3' : car='B' ;
				Break ;
			Case '6' : car='C' ;
				Break ;
		}
	}
}
```

Le C pour PIC:

Le C pour PIC est quelque peu différent de celui pour ordinateur. Ces différences viennent du compilateur C. En effet, puisque ce dernier fait la conversion C-assembleur et que les instructions dans ce dernier sont limitées (processeur RISC), le langage C est grandement simplifié, mais les concepteurs du compilateur ont légèrement modifié les instructions, qui sont heureusement assez intuitives, et rapidement assimilées.

Les nouveautés :

La différence entre un ordinateur et un PIC est dans les E/S. De ce fait, en langage C PIC, on donnera, si on le souhaite et c'est préférable, des noms aux pattes, ou aux ports complets, entre l'appel des librairies et la déclaration de sous-fonctions et/ou main.

Les règles :

Il est nécessaire d'écrire les instructions de C en minuscule, et celles propres au PIC (comme les TRISA et TRISB) en majuscule.

L'appel des librairies :

En C PIC, les librairies s'appellent ainsi : #include ''nom.H''

La déclaration des pattes et ports :

En C PIC, une patte se désigne par son port, et son numéro (lesquels commence à 0 jusqu'à 7). Par exemple, la patte 3 (en fait la quatrième) du port B sera : PORTB.3

Une patte seule se définit par la commande ''bit'' :
bit fname@PORTB.3, affecte le nom fnam à la patte 3 du port B.

Un port complet se définit par la commande ''char'' :
char sname@PORTA, affecte le nom snam au port A.

Ainsi, dans le main et les sous-programmes, on utilisera snam à la place de port A, et fnam, à la place de PORTB.3

Les affectations :

La première chose à faire dans le main, en C PIC, est de définir les valeurs dans TRISA et TRISB. Si leurs déclarations étaient compliquées en assembleur, elles sont grandement simples en C PIC. Ainsi il suffit de marquer : TRISA=0x00 (par exemple, en hexadécimal, la valeur étant à déterminer). On peut également rentrer la valeur en décimal (ex : TRISA=27), ou en binaire (ex :TRISA=0b00100110).

En ce qui concerne, la façon dont on utilise les pattes, elle est très simple. Comme vu précédemment, elles se désigne par le port auxquelles elles appartiennent, et leur numéro de patte. De plus elles peuvent être affectées à une valeur logiques (en sortie, ex : PORTA.2=1 ;), ou être lues (en entrées, ex : c=PORTB.7)

Remarque : ***Ces paragraphes sur le cours C a été écrit d'après ce que j'ai compris et retenu, comme principal, d'après mes cours de langage C et ceux de Hervé Hollard, dont vous trouverez l'adresse dans la section 9. Si les explications de ce cours, constituant les bases de la programmation C PIC sont suffisantes pour programmer pour ceux connaissant déjà le C, je conseille tout de même ne serait-ce que de lire, sinon d'imprimer les cours de Hervé Hollard, car ils constituent une base claire et solide du langage C (général et PIC), surtout pour les débutants et les novices.***

2.5- Exemple de programme :

La Fonction Delay :

Cette fonction, très utile dans un programme, a pour but de créer, comme son nom l'indique, une fonction Delay, c'est-à-dire, une temporisation. Il s'agit en réalité d'une sous-fonction, donc à rentrer en dehors du main. Quand on en a besoin dans la fonction main, il suffit alors de taper le nom de la sous fonction, en indiquant entre parenthèses le paramètre en milliseconde.

Instruction d'appel : *Delay (500) ; //pour une tempo de .5 s*

Code de la fonction :

```
Void Delay(int mill)
{
        OPTION=2 ;
        Do
        {
                TMR0=0 ;
                While(TMR0<125)
                        ;
        }While(--mill>0) ;
}
```

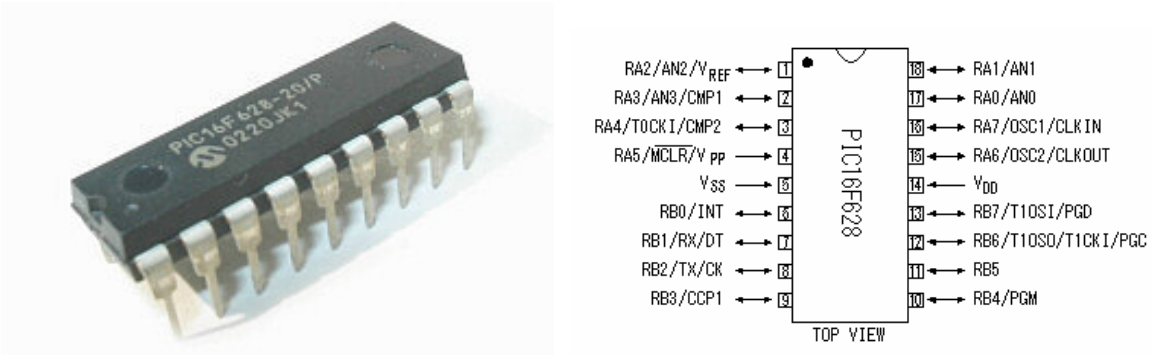
Et bien sur, il ne faut pas oublier de définir la sous-fonction avant le main :

Void Delay(int mill) ;

CHAPITRE 3 : LE PIC 16F628

3.1 Le futur PIC 16F84

3- LE PIC 16F628



Le PIC 16F628

A titre d'informations, une faible partie de ce livre est dédiée au remplaçant du PIC 16F84 : le 16F628

Compatible et avec plein de nouvelles fonctions, parions qu'il saura se faire une place dans le monde des microcontrôleurs

3.1- Le futur PIC 16F84 :

Le PIC 16F84 commençant à être un peu âgé (car existant depuis de nombreuses années, il a déjà été modifié à plusieurs reprises, d'où les lettres A et P lisibles sur certains boîtiers), Microchip a d'ores et déjà conçu et commercialisé son remplaçant : le PIC 16F628.

Avec un boîtier identique au 16F84, le 16F628 reprend les mêmes fonctions que son prédécesseur, en lui rajoutant des fonctions et des avantages au passage. De ce fait, les deux sont parfaitement identiques, et on pourra remplacer, dans un circuit donné, un 16F84 par un 16F628.

Quand à sa programmation, elle est identique à celle d'un 16F84 et par conséquent, pas besoin de réinvestir dans un programmateur.

Les améliorations notables sont :

- mémoire flash multipliée par deux
- mémoire E²PROM multipliée par deux
- jusqu'à 16 E/S disponible selon les utilisations (avec un min de 13)
- Deux timers 8 bit au lieu d'un
- Un timer 16 bits
- Deux comparateurs analogiques
- Une interface série
- Une référence de tension interne
- Un module de comparaison
- Dix sources d'interruptions au lieu de 4
- Une horloge interne

Il est bien sûr évident que si l'on garde le même nombre de pattes, toutes ces fonctions sont réparties. Ainsi chaque patte possède plusieurs fonctions possibles, mais dont une seule est utilisable à la fois.

Sans pour autant nous attarder sur ce merveilleux microcontrôleur, nous pouvons préciser que bien qu'étant donné toutes ces nouvelles fonctions, sa programmation et sa configuration seront quelque peu plus difficile, nous devons admettre que dans l'ensemble, nous sommes gagnants. En effet, le simple fait d'avoir une mémoire doublée, permet déjà d'ouvrir de nouvelles possibilités d'applications, tout comme la présence des deux comparateurs analogiques.

Et bien qu'il soit totalement compatible avec le PIC 16F84, il est à regretter que si l'on veut disposer des 3 entrées supplémentaires, cela entraîne une complication de programmation comme le constateront ceux qui le mettront en œuvre.

CHAPITRE 4 : PROGRAMMER LE PIC

4.1 Un programmateur de PIC

4.2 Les logiciels de programmation

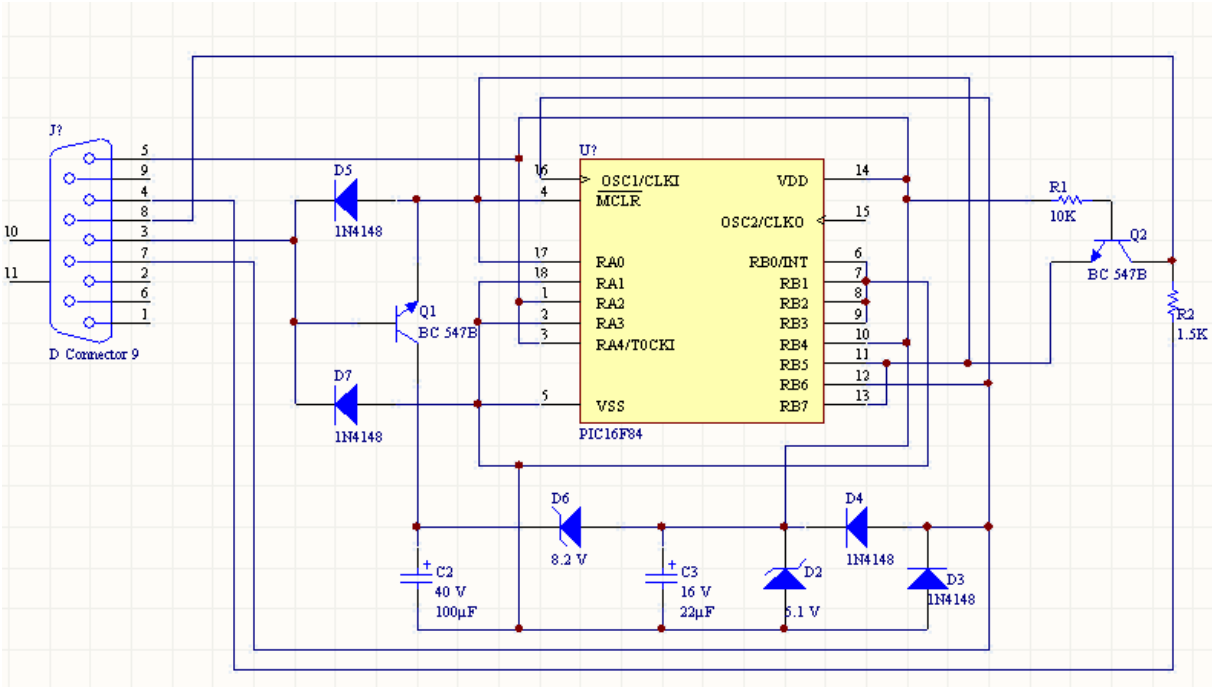
4- Programmer le PIC

4.1- un programmateur de PIC :

Nous allons voir ici, un programmateur de PIC fort simple. Bien que certains disent qu'il commence à vieillir pour la simple raison qu'il se branche sur un port série, je pense qu'il reste le meilleur programmateur. En effet, pas besoin de lui fournir une alimentation externe comme certains programmateurs. De plus, étant relativement simple, et ancien, on peut trouver de nombreuses docs sur le net, notamment, pour ceux que ça intéresse, sur son fonctionnement. Il s'agit d'un programmateur de PIC 16F84, de type JDM

Nous ne verrons ici, que la façon de le fabriquer, sans nous attarder sur son fonctionnement. Nous verrons donc son schéma structurel, le PCB et le schéma d'implantation.

Le structurel :

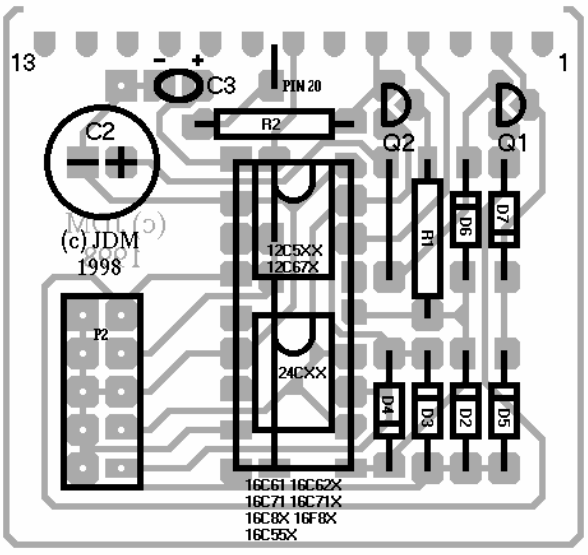


Il vous faudra quelques composants pour le fabriquer :

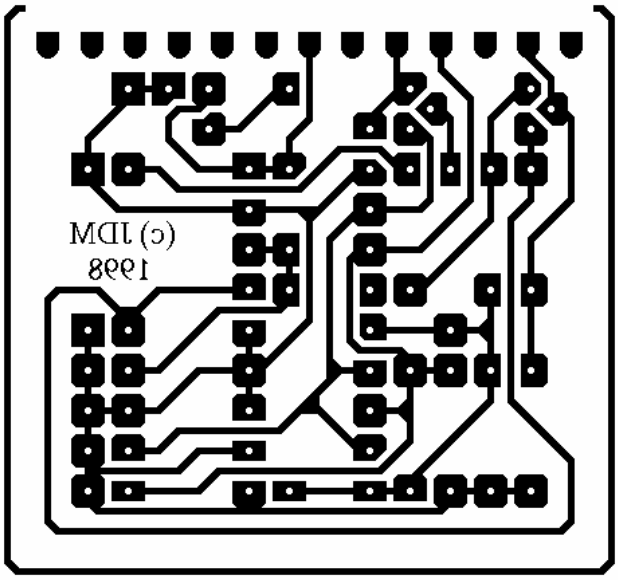
R1	10K	Resistor
R2	1.5K	Resistor
D2	5.1V/0.5W	Zener
D3	1N4148 or 1N4448	Diode
D4	1N4148 or 1N4448	Diode
D5	1N4148 or 1N4448	Diode
D6	8.2V/0.5W	Zener
D7	1N4148 or 1N4448	Diode
C2	100µF/25V	Capacitor electrolytic
C3	22µF/16V or 47µF/6.3V	Capacitor tantal
Q1	BC547B	Transistor NPN
Q2	BC547B	Transistor NPN
P1	DS25 (female)	25 pol DSUB connector
P2		Connector for In Circuit Programming

Le PCB vient d'un site internet (http://www.jdm.homepage.dk/pcb2.htm), mais pour ceux qui ne dispose pas d'accès, voici les PCB, tels qu'ils sont mis sur le net. A noter que le schématique du site est avec un connecteur parallèle, alors, que le schématique de ce livre est pour un connecteur série.

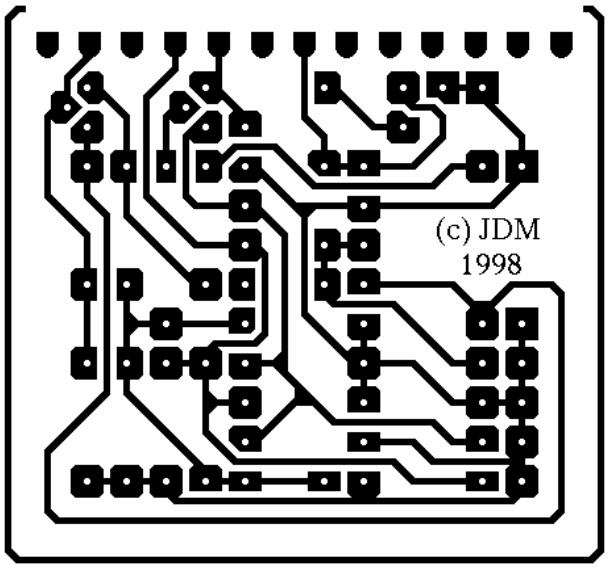
Implantations des composants :



Vue de dessus (par transparence) :



Vue côté cuivre :



Pour informations, le cadre fait 3,8 cm de large et 3,6 cm de haut. Attention donc, quand vous imprimez le typon.

4.2- Les logiciels de programmation :

Afin de programmer le PIC, nous aurons besoin de deux logiciels (MPlab et IC-Prog), ainsi que d'un compilateur (CC5Xfree).

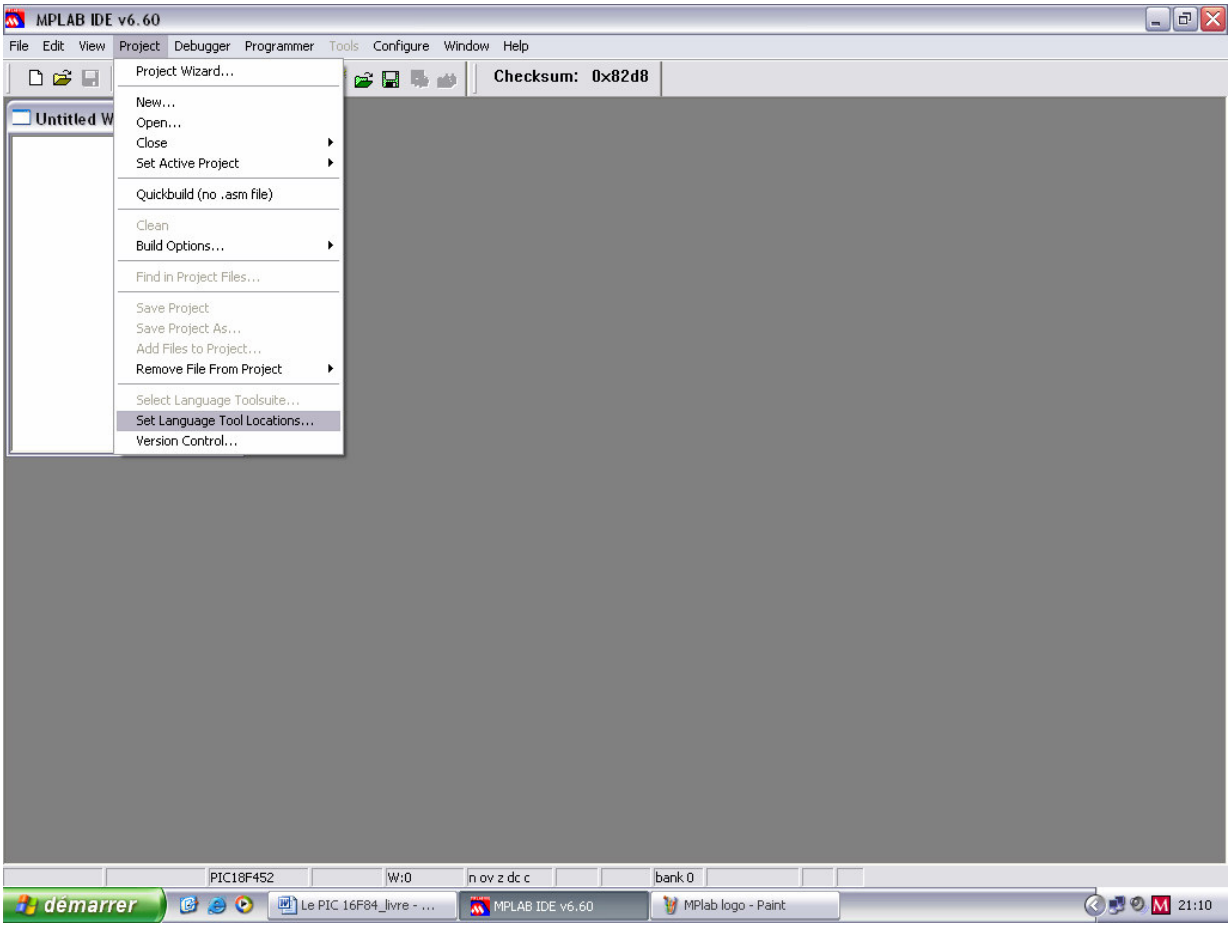
Nous verrons donc le logiciel de programmation avec son compilateur (MPlab et CC5Xfree), et le logiciel pour télécharger le programme dans le PIC (IC-Prog)

MPlab est un logiciel gratuit de programmation, développé et proposé par Microchip, le fabricant du microcontrôleur PIC.

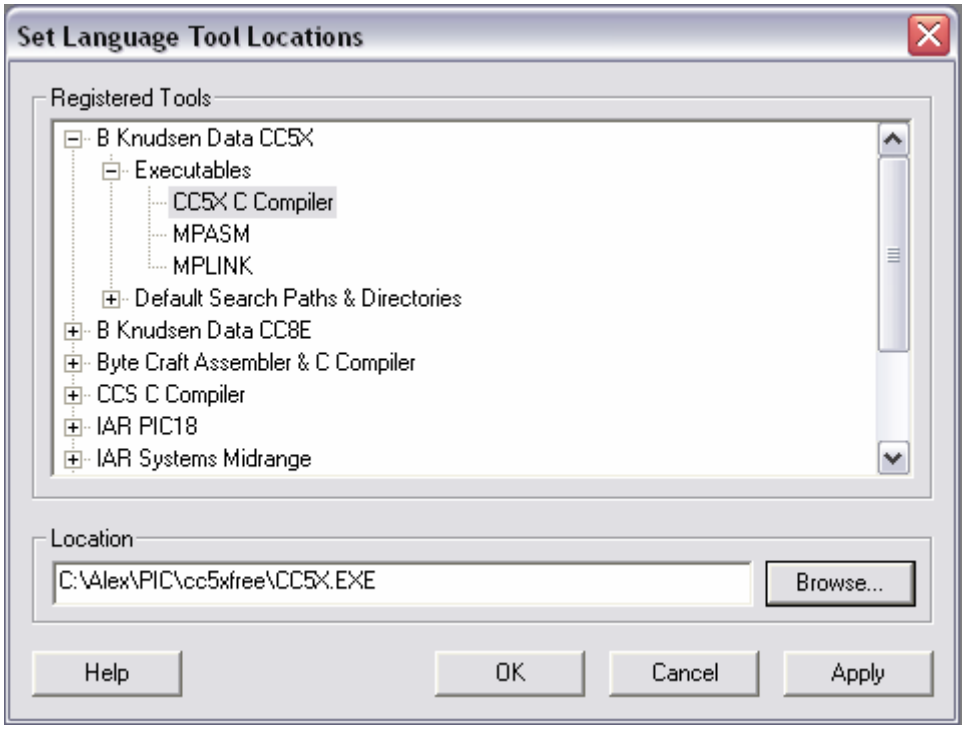
Les étapes que nous verrons dans l'immédiat consistent à déclarer le compilateur dans MPlab. De fait, cette opération n'est à effectuer qu'une seule fois. Ces étapes seront illustrées en images.

La première chose à faire est d'ouvrir le logiciel MPlab. Bien, maintenant, nous allons pouvoir commencer.

Etape 1 : ouvrez le menu ''Project'' et cliquez sur ''Set Language Tool Locations

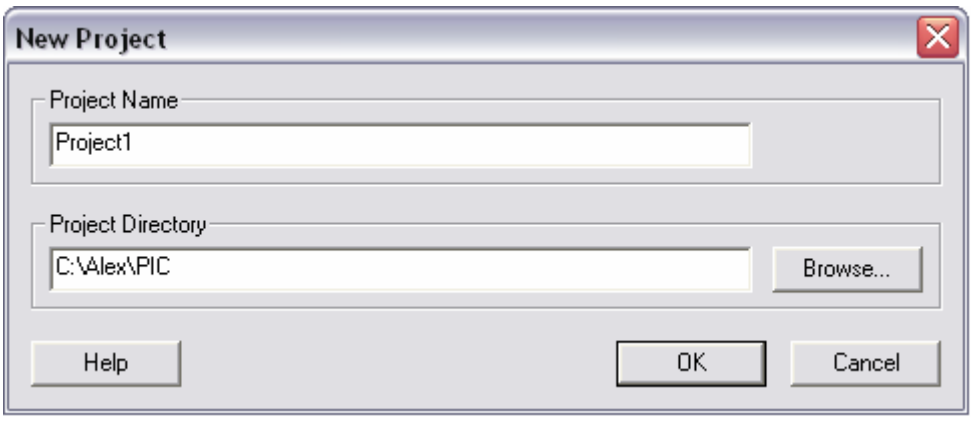


Etape 2 : il faut préciser à MPlab où se trouve le compilateur CC5X, qui nous permettra d'écrire en C.

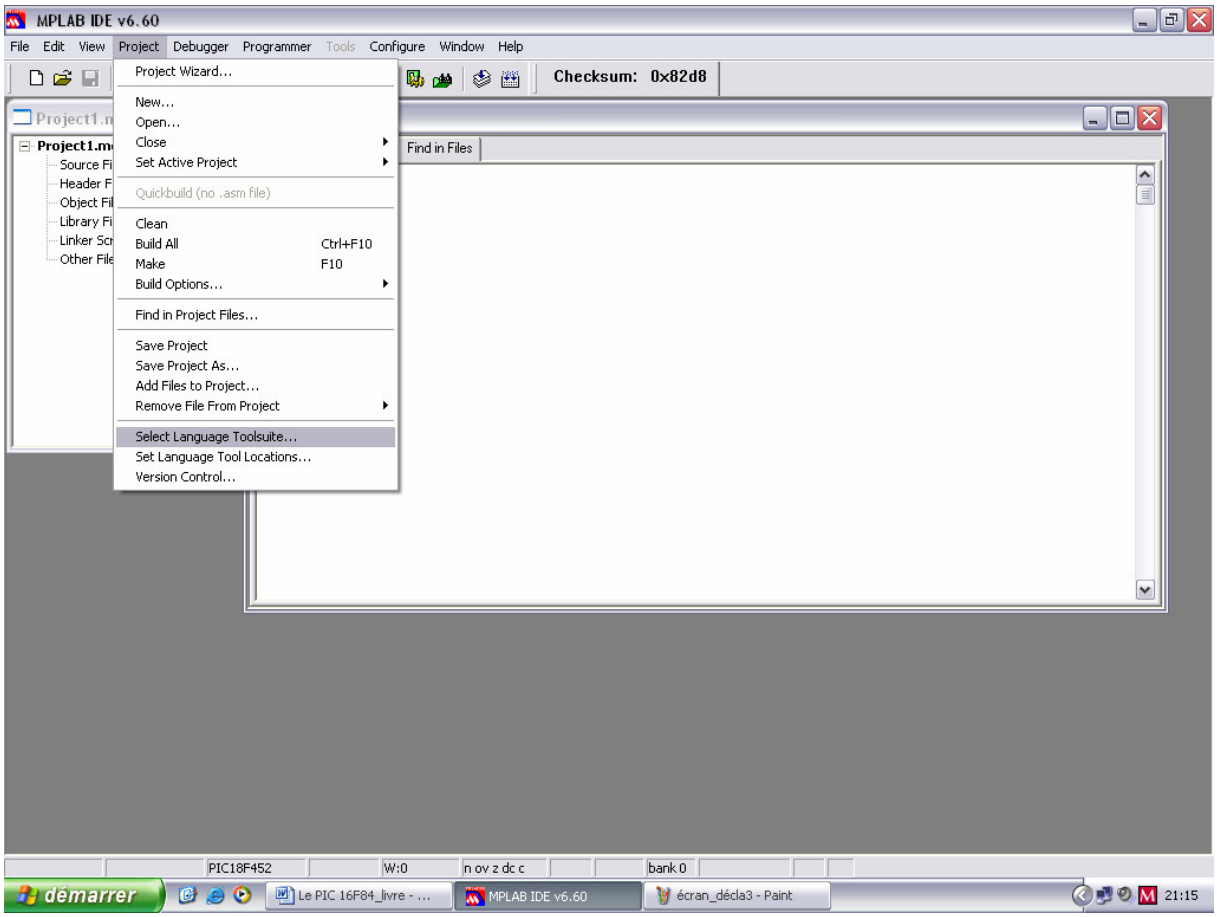


Vous devez cliquer, comme vous pouvez le voir sur ''B Knudsen Data CC5X'', puis sur ''Executables'', et enfin sélectionner ''CC5X C Compiler''. Ensuite vous devez cliquer sur ''Browse'' pour lui dire où trouver le compilateur, en lui indiquant le chemin.

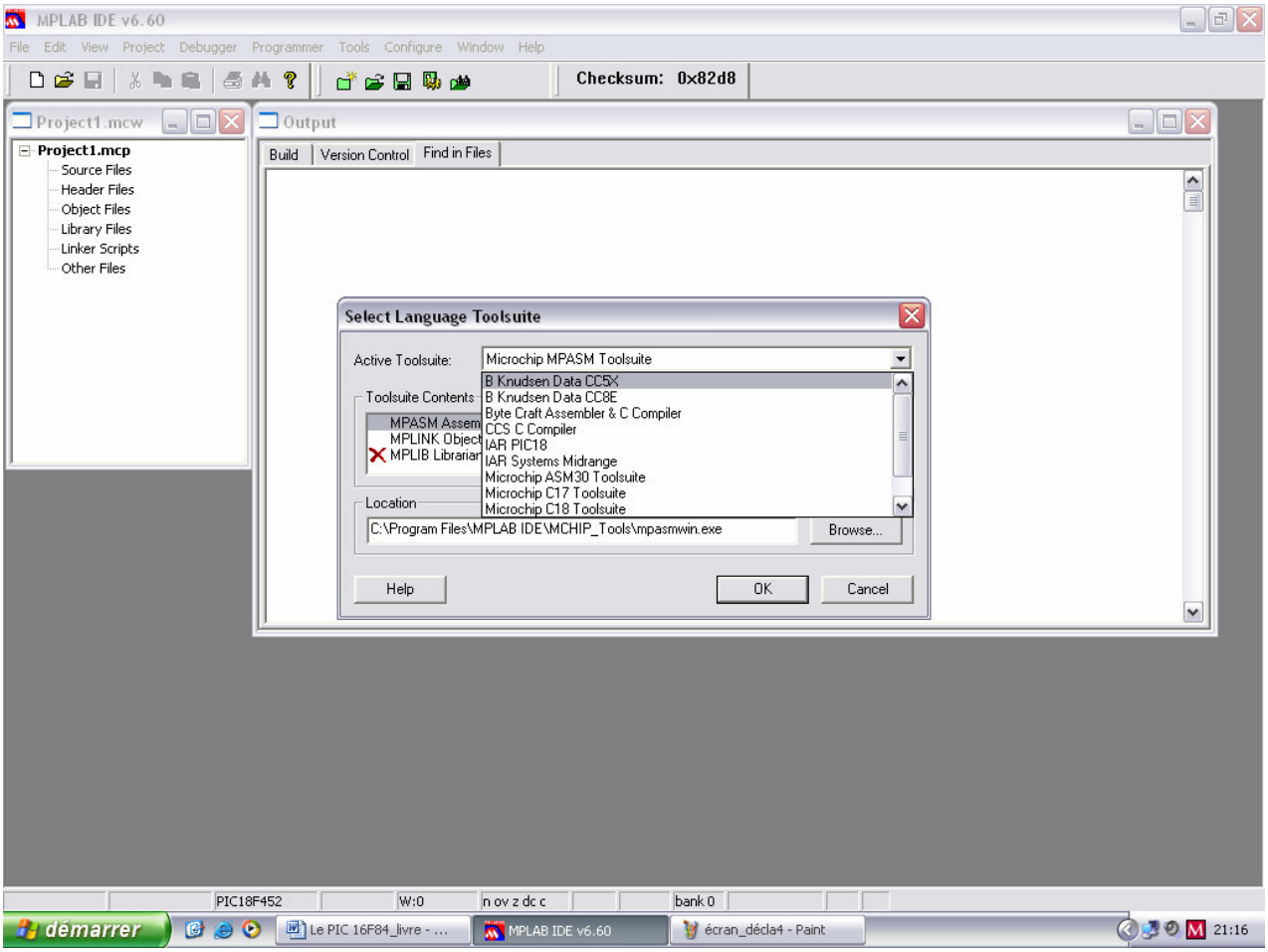
Etape 3 : la troisième étape consiste à déclarer un nouveau projet, en cliquant sur ''Project'', puis sur ''new''

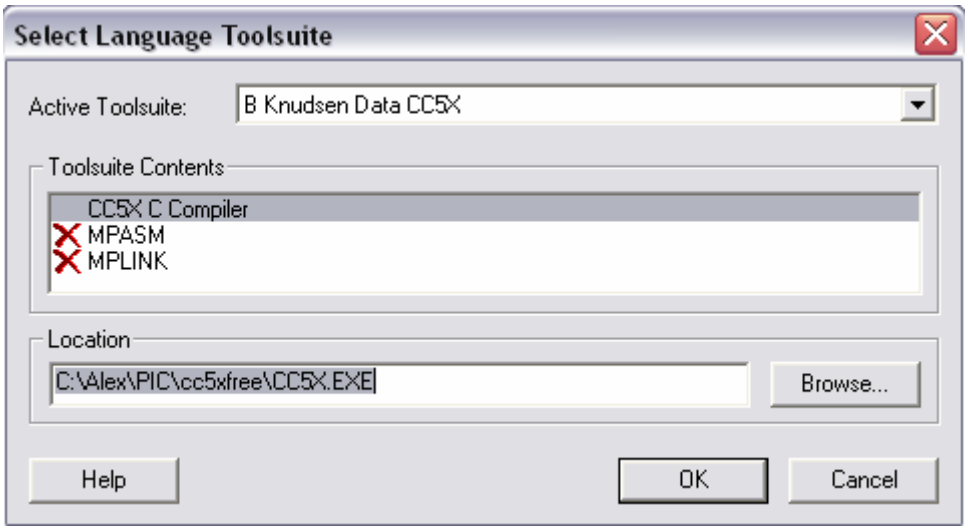


Etape 4 : après avoir dit à MPlab où était le compilateur CC5X, il faut lui préciser, que c'est le compilateur que l'on veut utiliser. Pour cela cliquez sur le menu ''Project'', puis sur ''Select Language Toolsuite''.

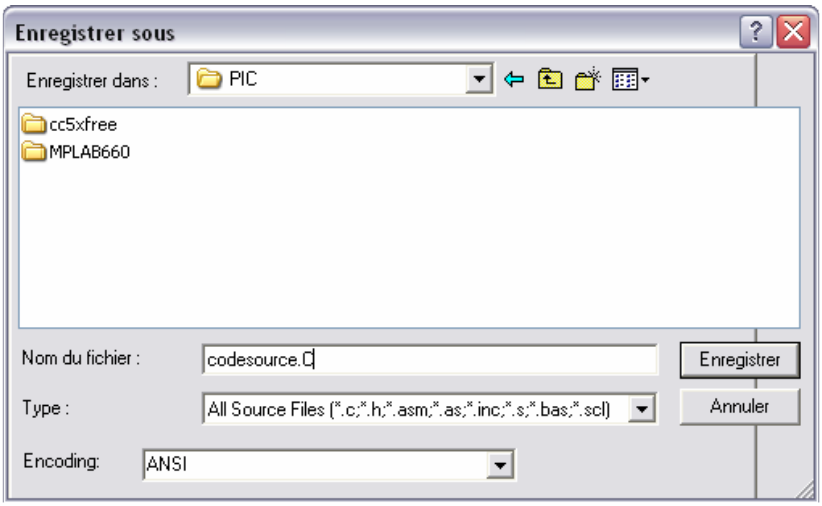


Vous obtenez alors la fenêtre suivante, dans laquelle vous devez sélectionner dans le menu déroulant ''active toolsuite'' la ligne ''B Knudsen Data CC5X''. Cliquez alors sur ''CC5X C Compiler'' et vérifiez que la ligne ''Location'' indique le bon chemin à MPlab.

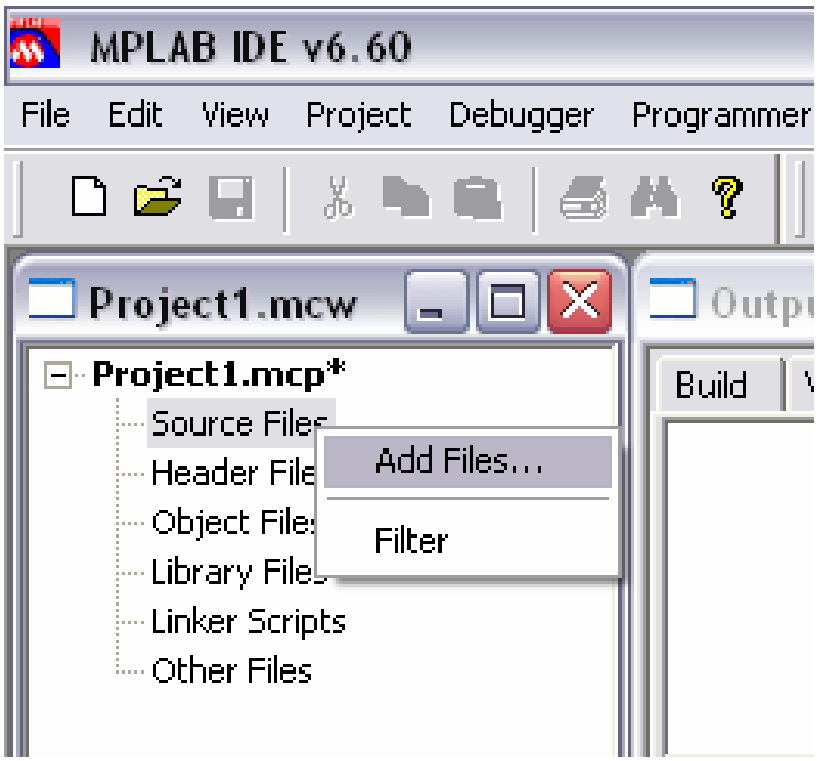




Voilà, nous avons configuré le compilateur ; maintenant, nous allons voir comment créer la fenêtre pour le code source. Cela est très simple : cliquez sur ''File'', puis sur ''New''. Une fenêtre apparaît alors. Cliquez alors sur ''File'', puis sur ''Save As''. Rentrez alors le nom que vous voulez, suivit de ''.C'', afin de préciser le type de document que vous allez écrire.

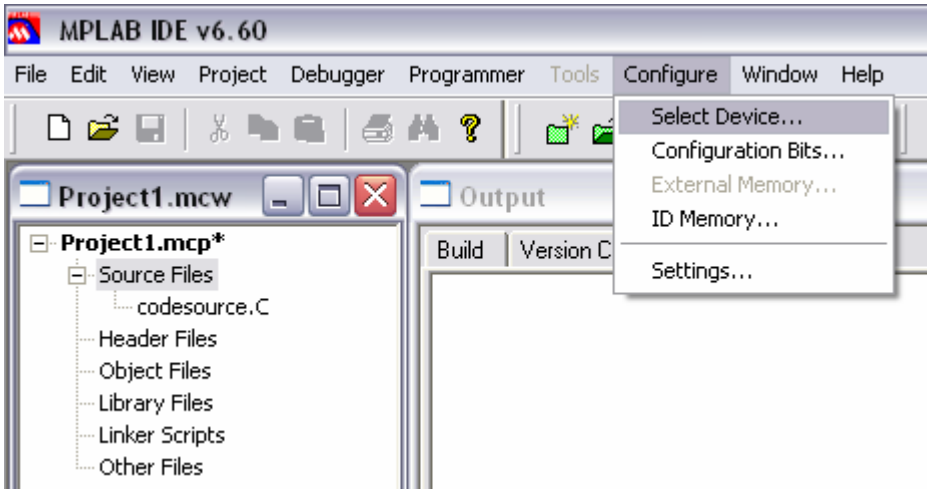


Il ne vous reste alors plus qu'à lier le code source à votre projet :

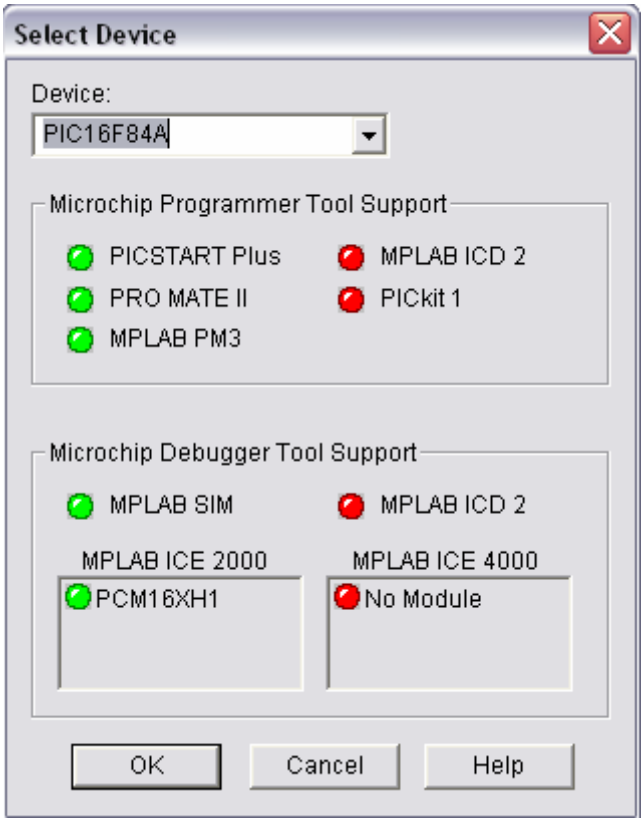


Les derniers paramètres :il s'agit de préciser à MPlab le composant auquel est destiné le programme.

Pour ce faire, cliquez sur ''Configure'', puis sur ''select device''

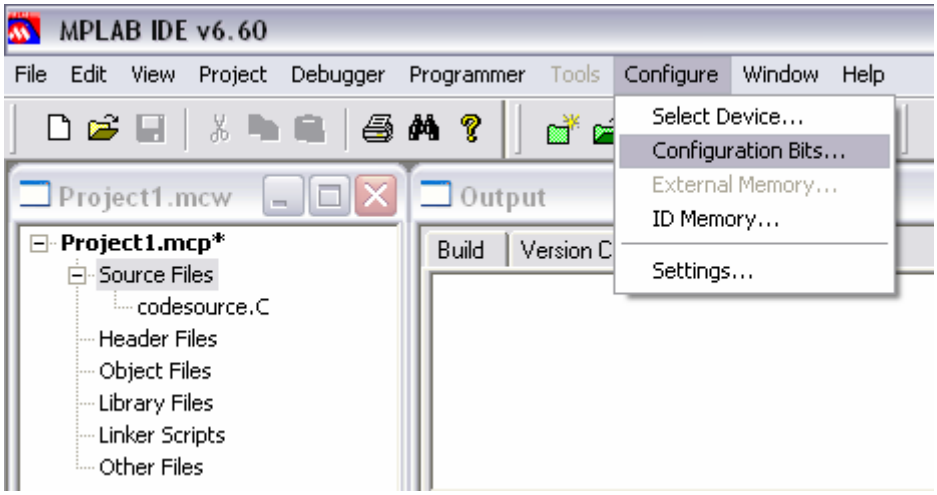


Apparaît alors l'écran suivant :

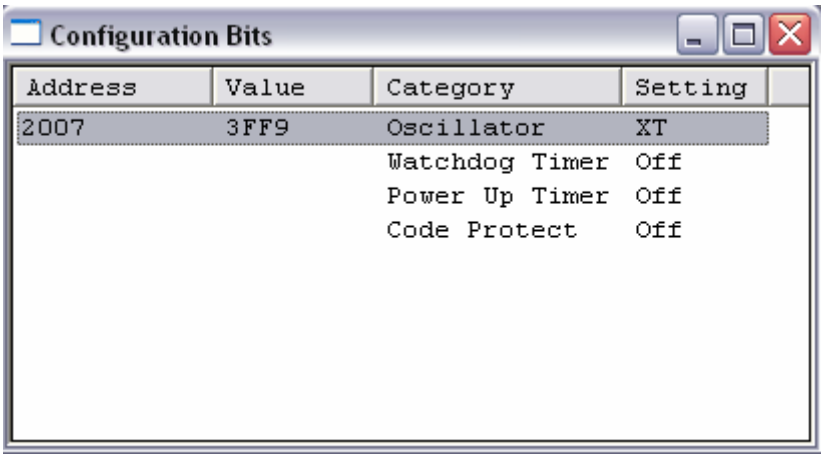


Maintenant, vous devez définir les paramètres de configuration : le Watchdog, le type d'oscillateur , le power up timer (mises sous-tension retardée) et le code P (code protect).

Cliquez sur ''Configure'', ensuite sur ''Configuration Bits'' :

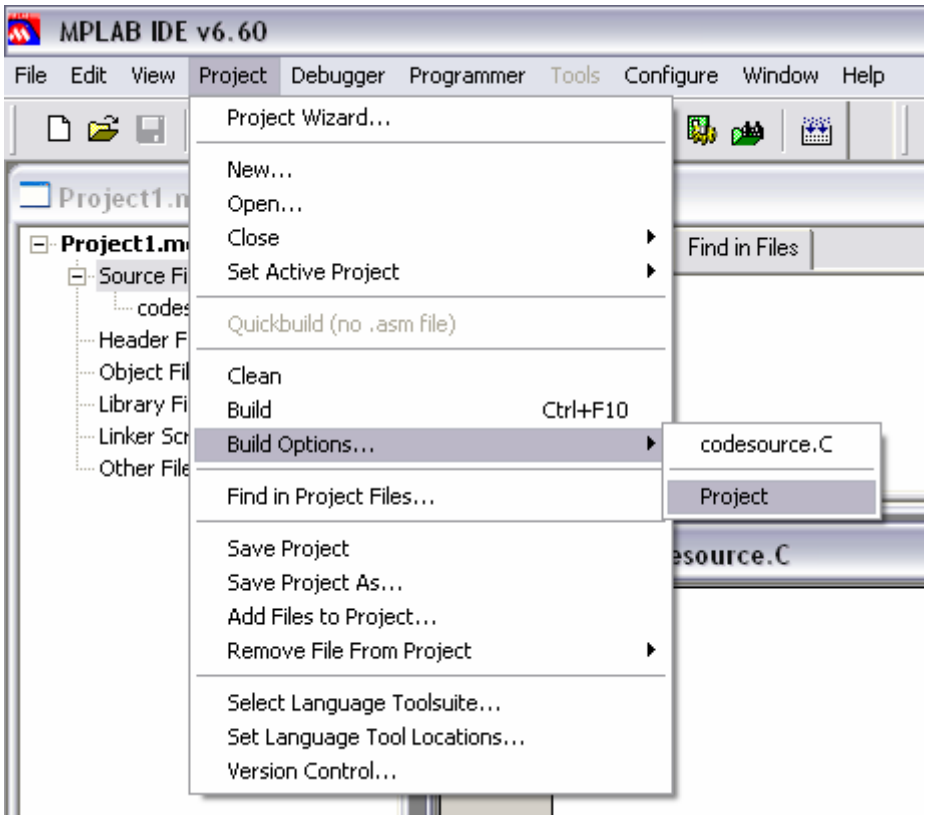


Veuillez alors à rentrer les mêmes valeurs que dans cette capture d'écran. A noter, que cette fenêtre correspond avec un quartz.

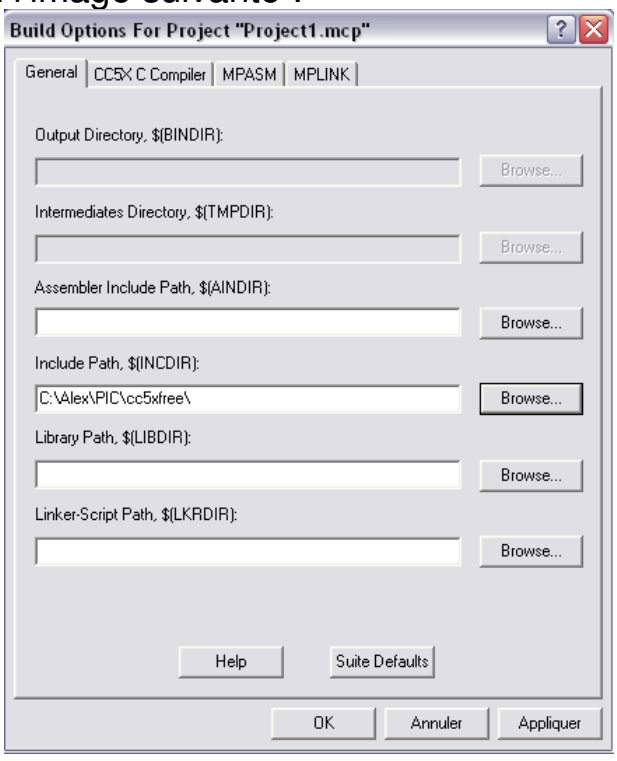


Comme nous l'avons vu précédemment dans la partie sur le langage C, nous aurons besoin de librairies. Celles-ci sont propres au compilateur. Il est donc nécessaire de définir l'endroit où elles se trouvent.

Pour cela, cliquez sur le menu ''Project'', puis sur ''Build Options'' et enfin sur ''Project''.



Et vous obtenez l'image suivante :



Il faut alors que vous indiquiez dans le champ ''Include Path'' le chemin du dossier du compilateur CC5X.

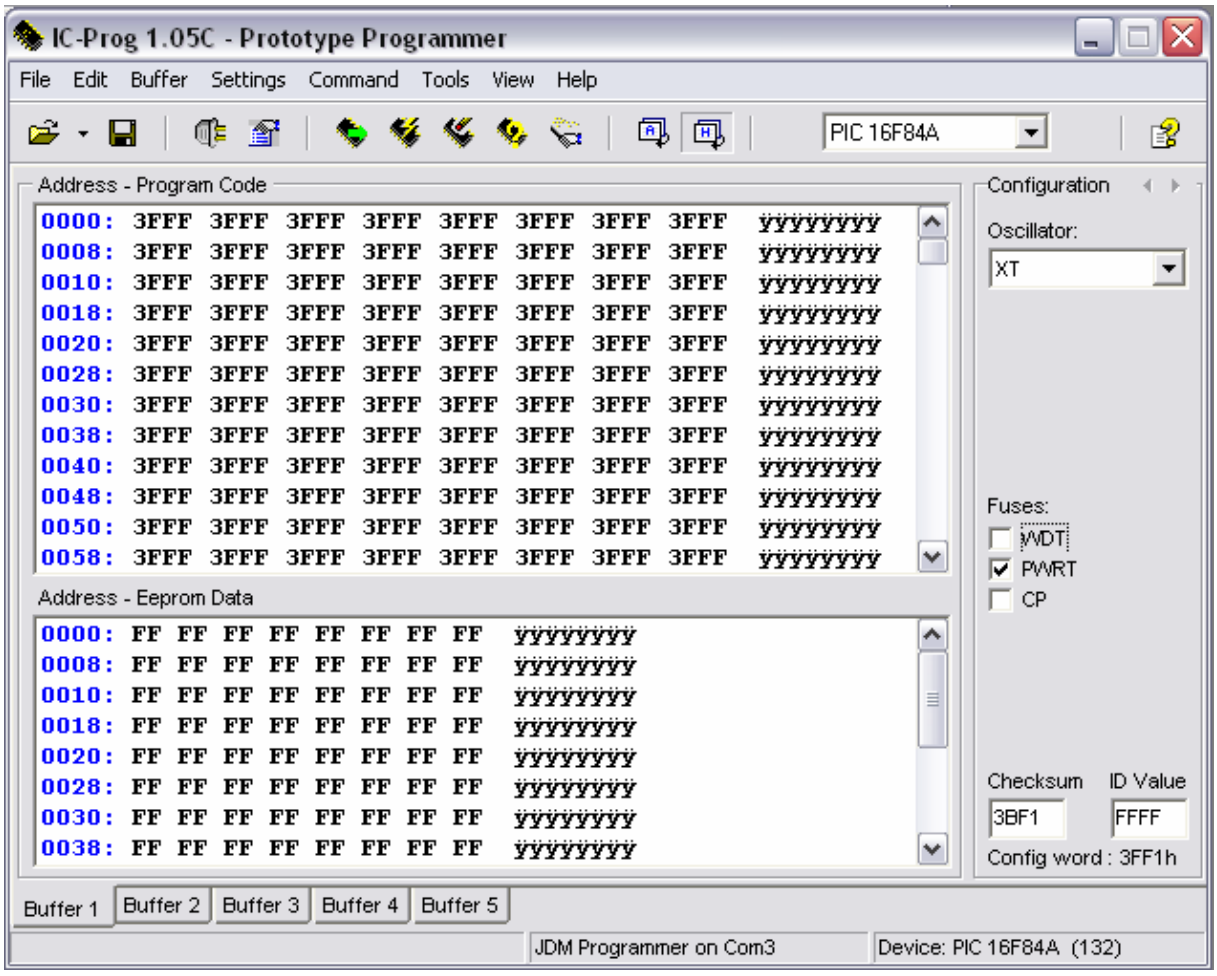
Voilà, votre logiciel est enfin configuré pour programmer.

Une indication utile toutefois. Quand votre programme sera terminé, pour le compiler et vérifier que tout va bien (si ce n'est pas le cas, il vous affichera un message d'erreur), il faut utiliser le bouton ''build'', correspondant à l'icône : 

Nous allons maintenant voir le logiciel, qui permet, par l'intermédiaire du programmateur vu précédemment, de télécharger notre programme dans le PIC.

<u>Remarque :</u> C'est le ''.hex'' que l'on doit ouvrir. Si celui-ci n'est pas présent dans votre dossier, ou si MPlab vous indique lors de la compilation qu'il n'a pas pu être généré, alors ouvrez le menu Windows, et cliquez sur ''Exécuter''. Dans le champ qui s'ouvre tapez alors la ligne :
regsvr32 ''C:\Program Files\MPLAB IDE\dlls\MPProgram.dll''
Cette ligne devrait résoudre le problème.

Nous allons commencer par ouvrir le logiciel ICPROG :



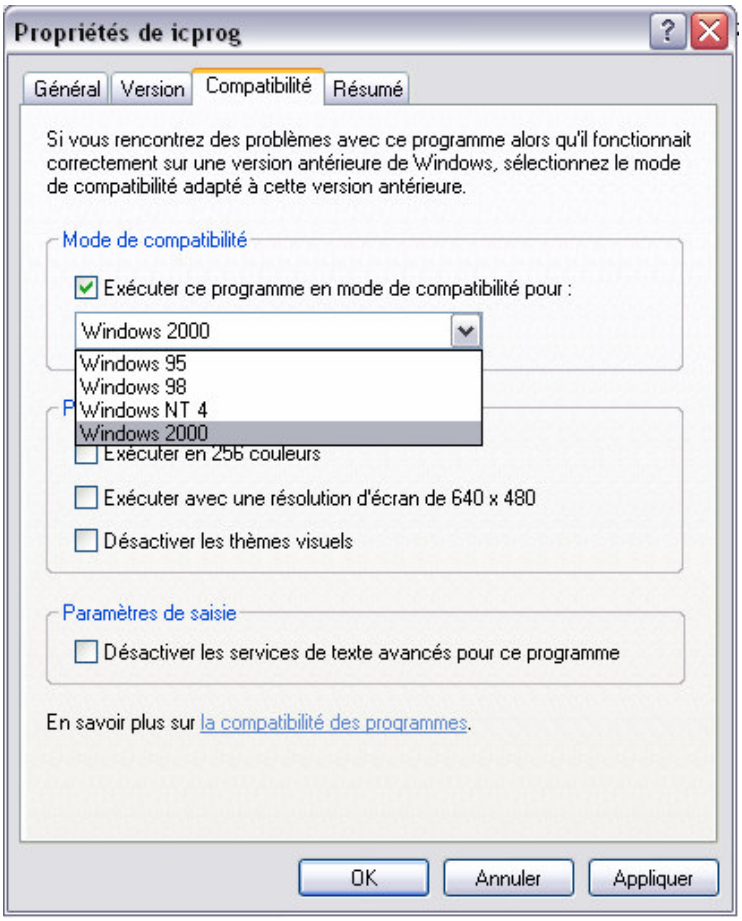
Faite attention à bien choisir un oscillateur de type ''XT'', et de ne cocher que le PWRT. Enfin chose importante, sélectionner le PIC 16F84 comme composant.

A ce stade, vous êtes prêts à programmer, ne vous reste alors plus qu'à télécharger votre programme. Mais avant de voir comment faire nous allons faire une remarque importante.

<u>Remarque :</u> Toutes les étapes pour ICPROG vues jusqu'ici sont bonnes pour toutes les versions de Windows. Cependant, pour windows 2000, NT, et XP, il est nécessaire de rajouter des drivers spéciaux à ICPROG. Voici comment faire :

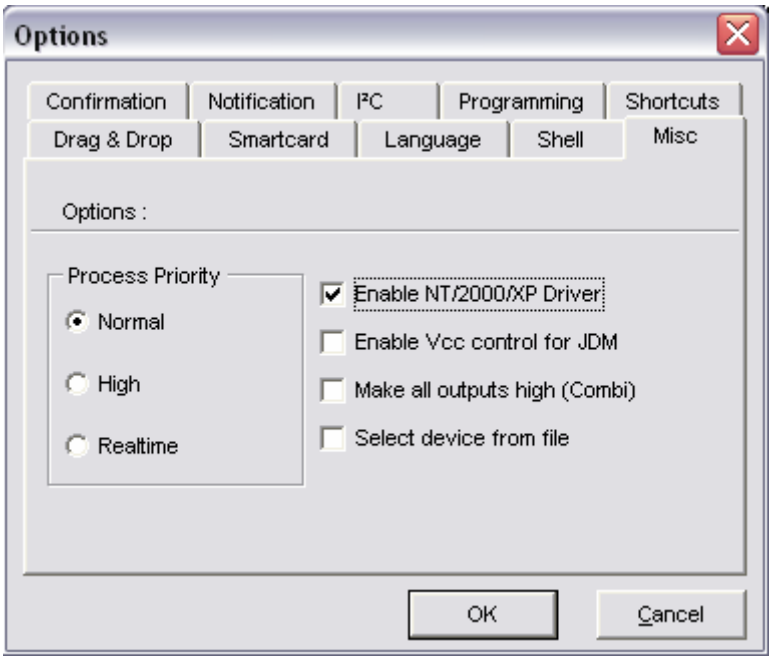
Lors de la première mise sous tension, vous obtiendrez des messages d'erreurs de violations. Refermez alors le logiciel et allez dans le répertoire où se trouve l'exécutable. Faites un clique droit sur l'exécutable, et ouvrez les propriétés.

Allez alors dans compatibilité, cochez la première case, et demandez à exécuter le programme en mode de compatibilité pour windows 2000, puis validez.

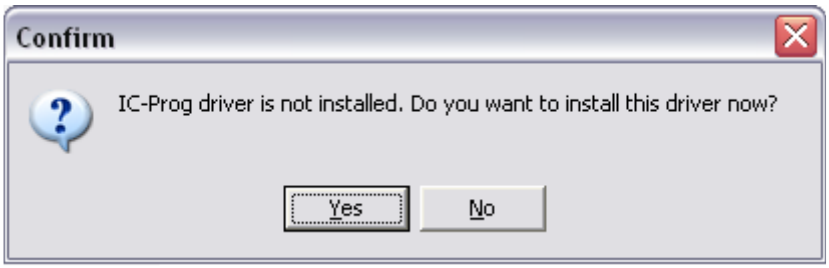


Rouvrez alors ICPROG,. Vous obtiendrez un nouveau message d'erreur. Cliquez sur OK, puis cliquez sur le menu ''Settings'', puis sur ''Options''.

Dans le menu qui apparaît, allez dans l'onglet ''MISC'', et cochez l'option ''Enable NT/2000/XP Drivers''

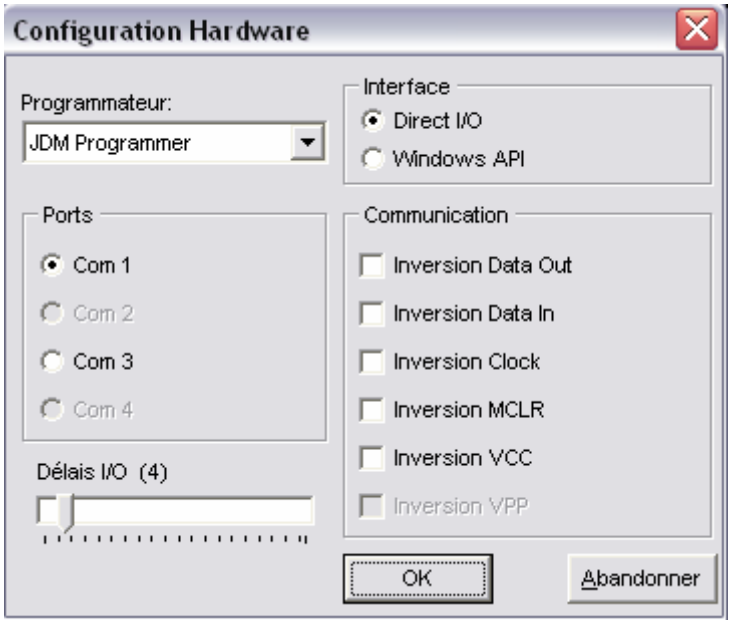


Vous devez alors redémarrer ICPROG pour que les changements prennent effet. Au redémarrage, il vous affiche cette fenêtre :



Après vous être assuré que le driver (icprog.sys) est présent dans le même dossier que l'exécutable, cliquez sur YES.

Maintenant, préciser à ICPROG, le programmateur que vous utilisez. Pour ce faire, appuyez sur F3, et réglez les paramètres comme suit :



Voilà, ICPROG est configuré pour Windows NT, 2000 et XP.

<u>Remarque :</u> avec un ordinateur portable, il se peut que la programmation échoue (erreur de type :''Echec de la vérification à l'adresse 0000h''). Ceci peut venir de votre ordinateur ne fournissant pas les bonnes tensions pour la programmation, ou alors, dans certains un câble série trop long (la résistance du fil entraînant une chute de tension entre le port série et le programmateur).

Bien. Maintenant, nous allons voir comment télécharger un programme dans le PIC.

Voici à quoi ressemble la barre des tâches de ICPROG :

Nous détaillerons ces icônes une à une :

1-Ouvrir un fichier (n'ouvrez que des fichiers .hex, les seuls aptes à être téléchargés)
2-Enregistrez (ne sert que rarement)
3-Hardware configurations (configuration déjà effectuée)

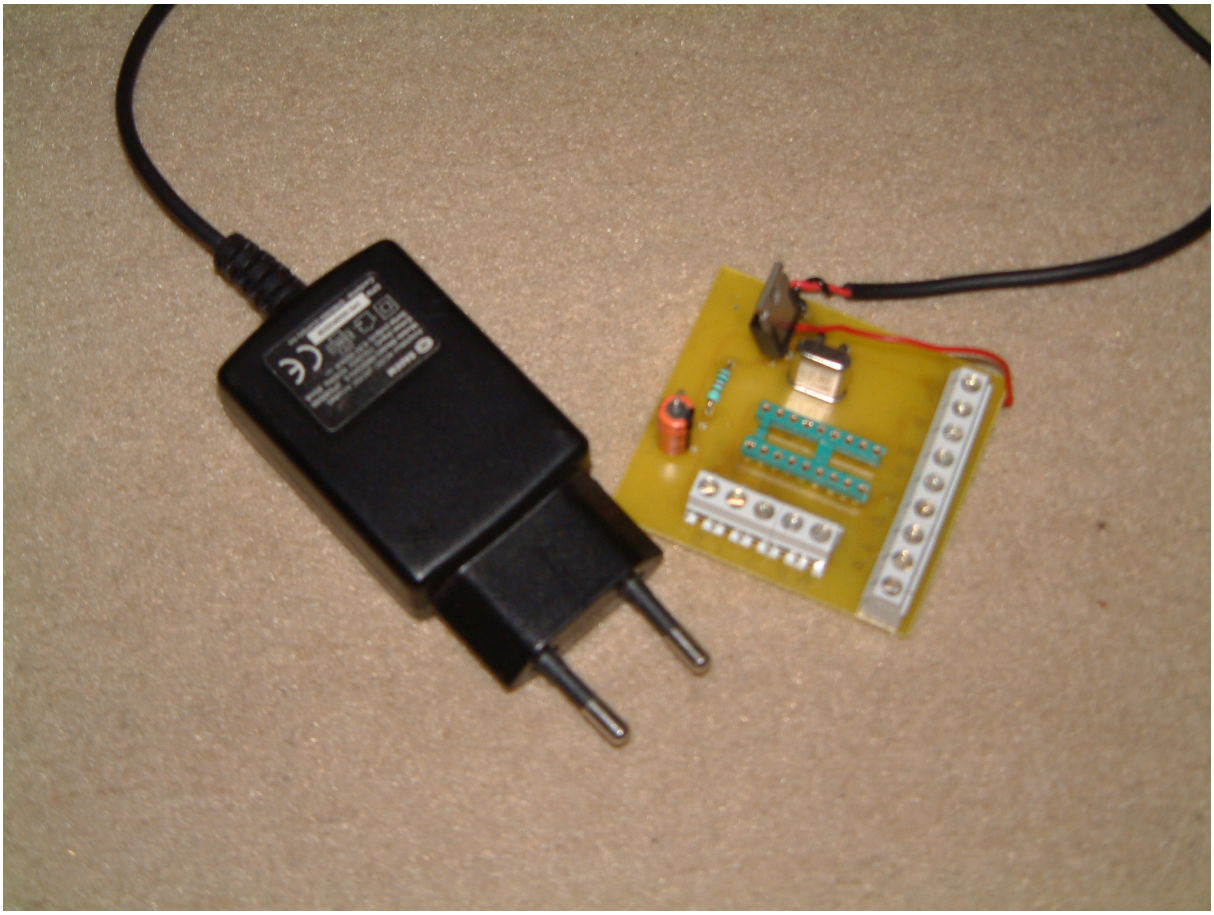
4-Options configurations (configuration déjà effectuée)
5-Sert à lire le code présent dans le composant. Le résultat s'affiche dans la fenêtre sous forme de code hexa (si pas d'erreurs).
6-Sert à télécharger le programme dans le composant
7-Sert à effacer le contenu du composant
8-Sert à vérifier que le composant à bien été programmé
9-Assistant SmartCard (ne sert pas)
10-Sert à voir le code en assembleur
11-Sert à voir le code en hexadécimal

Télécharger le programme :

Pour télécharger un programme, il faut commencer par ouvrir un fichier .hex. Pour cela, cliquez sur l'icône 1 et allez chercher votre fichier . hex. Puis, réglez votre oscillateur, et vos paramètres comme indiqués plus haut. Effacer le composant en cliquant sur l'icône 7, puis programmez le avec l'icône 6.

Remarque : ***Dans certains cas, avant d'importer votre .hex, il est préférable de vérifier s'il n'y à rien sur votre composant, à l'aide de l'icône 5.***

CHAPITRE 5 : UN TESTEUR DE PIC

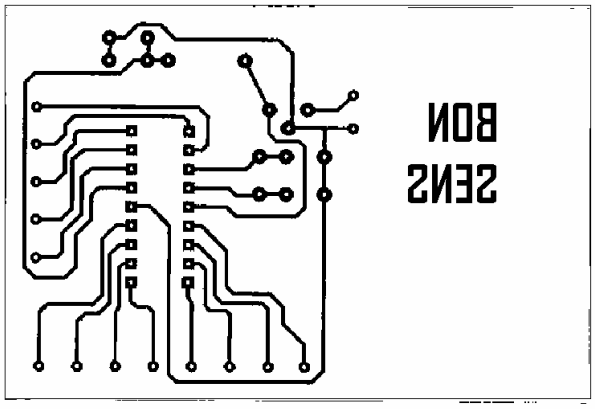


Dans cette partie, nous verrons un testeur de PIC, le principe, et le but de celui-ci.

Ce testeur, composé avec uniquement quelques composants, nous permet de tester nos programmes, et de vérifier le bon fonctionnement de ces derniers.

Son principe est simple. Le testeur possède sur sa plaque un quartz, un régulateur, un système de reset mixte et deux séries de borniers. Alimenté par un transformateur 220V-6V, le testeur délivre, grâce au régulateur (un 7805) une tension de 5 volts au PIC. Le système Reset assure le rôle qui lui est alloué, à la mise sous tension, et lorsque l'utilisateur le veut. Le quartz, cadencé à 4 Mhz, sert à délivrer le signal d'horloge au PIC. Les séries de borniers, aux nombre de deux, assurent une connectivité des ports avec l'extérieur (une série de borniers par ports).

En voici le typon (vu de dessus, attention à l'impression sur calque) :

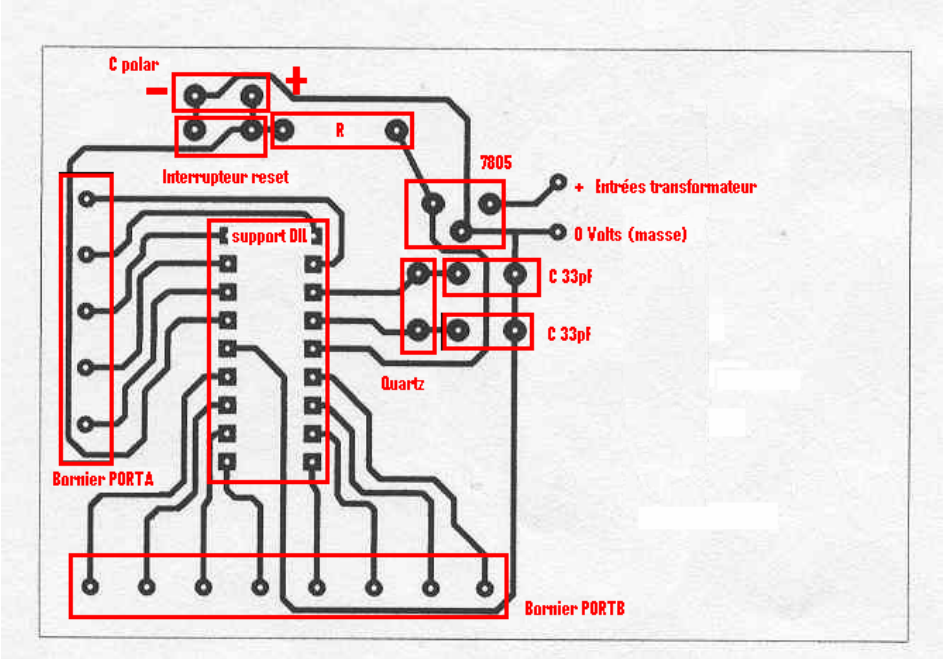


Vous pouvez scanner et imprimer, sur calque, ce typon, afin de pouvoir tirer une plaque. Passer par un ordinateur, pour vous assurer de la bonne échelle. Le cadre fait 7,8 cm de large et 5, 35 cm de haut. Attention au sens, une fois tiré, vous devez pouvoir lire « bon sens » coté cuivre.

Voici la liste des composants dont vous aurez besoin, en dehors, bien sûr, d'une plaque et du PIC :

-un quartz de 4 MHz
-5 borniers double à souder sur CI
-1 bornier triple à souder sur CI
-un support DIL 18 pattes
-un transformateur externe 220V- 6V
-un régulateur 7805
-une résistance de 1K
-un condensateur polarisé 1µF
-2 condensateurs 33pF
-un interrupteur poussoir

Voici l'implantation des composants (vu de dessus) :



Le but de ce testeur est donc de permettre de tester notre programme complet ou partiel en nous permettant de connecter facilement les périphériques, au choix ; le principal inconvénient de ce testeur étant sa fréquence de fonctionnement fixe, à 4 MHz.

<u>Petite astuce</u> : Vous pouvez ôter le régulateur et brancher le circuit sur une prise USB d'un ordinateur. En effet, la prise délivre une tension de 5V régulée. Cependant, cette manip reste réservée aux personnes possédant un minimum de connaissances. En effet, un mauvais branchement peut détruire votre PIC.

CHAPITRE 6 : EN BREF, TOUT CE QUI EST UTILE

Le but de cette partie est simple : y rassembler tous les éléments vus, et qui sont nécessaires à la programmation du PIC. Bonne programmation.

Tableaux de programmation

	ADRESSES HEXA	NOM	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Page 0	00	INDF	Utilise le contenu de FSR pour adresser la mémoire							
	01	TMR0	Horloge/compteur en temps réel							
	02	PLC	Octet de moindre poids du PC							
	03	STATUS	IRP	IRP1	RP0	TO#	PD#	Z	DC	C
	04	FSR	Adressage indirect avec INDF							
	05	PORTA	-	-	-	RA4/ Tock	RA3	RA2	RA1	RA0
	06	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0/ INT
	07		Non implémenté, lu comme étant à 0							
	08	EEDATA	Registre de données E²PROM							
	09	EEADR	Registre d'adresses E²PROM							
	0A	PCLATH	-	-	-	S'écrit sur les 5 bits de PCH				
	0B	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF

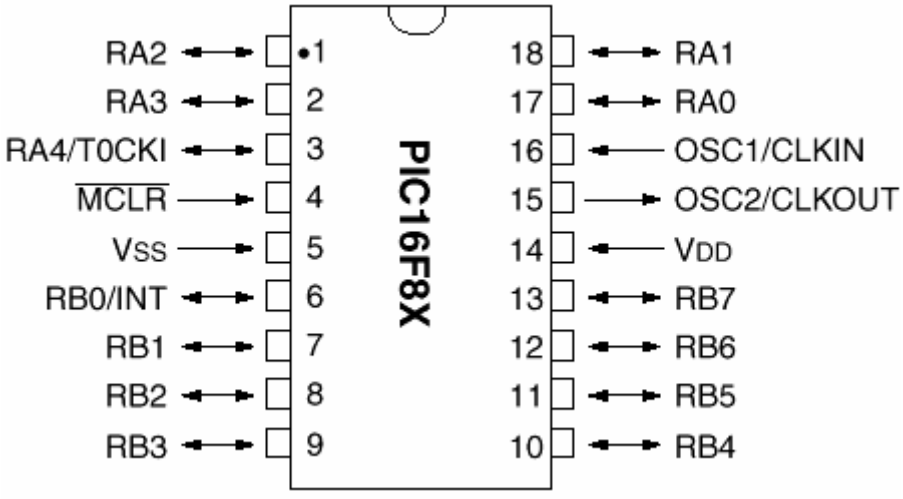
	ADRESSES HEXA	NOM	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Page 1	80	INDF	Utilise le contenu de FSR pour adresser la mémoire							
	81	OPTION	RBPU#	INTDG	T0CS	T0SE	PSA	PS2	PS1	PS0
	82	PLC	Octet de moindre poids du PC							
	83	STATUS	IRP	IRP1	RP0	TO#	PD#	Z	DC	C
	84	FSR	Adressage indirect avec INDF							
	85	TRISA	-	-	-	Configuration du PORT A				
	86	TRISB	Configuration du PORT B							
	87		Non implémenté, lu comme étant à 0							
	88	EECON1	-	-	-	EEIF	WRERR	WREN	WR	RD
	89	EECON2	Registre de commande E²PROM							
	0A	PCLATH	-	-	-	Configuration du PORT A				
	0B	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF

Configuration du registre OPTION :

PS2	PS1	PS0	Rapport de division	
			WDT	RTCC
0	0	0	:1	2
0	0	1	2	4
0	1	0	4	8
0	1	1	8	16
1	0	0	16	32
1	0	1	32	64
1	1	0	64	128
1	1	1	128	256

Bit PSA	0 = Choix du RTCC
	1 = Choix du WDT
Bit TOSE	0 = Front montant
	1 = Front descendant
Bit TOCS	0 = Horloge interne
	1 = Front sur la broche RTCC
Bit INTEDG	0 = interruption (RB0) validé sur front descendant
	1 = interruption (RB0) validé sur front montant
$\overline{\text{Bit RBPU}}$	0 = Connection d'une résistance de rappel entre RB4 et RB7
	1 = Aucune résistance de rappel

Répartitions des pattes du PIC



Programmer le Port A et le Port B :

Configuration du Port A 1:entrée 0:sortie							
-	-	-	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

Configuration du Port B 1:entrée 0:sortie							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

CHAPITRE 7 : CONCLUSION

Ainsi, comme nous avons pu le voir tout au long de ce livre, le PIC16F84, bien que vieillissant un peu, dispose encore de nombreuses ressources et risque de faire parler de lui encore un certain temps.

Difficile à appréhender en premier lieu, celui-ci s'impose comme l'élément indispensable pour de nombreux montages, tant par son rapport qualité/prix, que par ses capacités ; son principal handicap étant le nombre limité de ses entrées/sorties.

Il est programmable avec un minimum de difficultés, ce qui fait de lui le microcontrôleur parfait pour les amateurs débutants. Quand aux utilisateurs confirmés, ils apprécient l'ensemble de ses fonctions à leurs justes valeurs, et savent repousser les limites du PIC par de petites astuces, toutes plus ingénieuses les unes que les autres.

De même, le matériel de programmation et de test, requiert le minimum de connaissances que l'on peut demander à un amateur.

De fait, l'amateur appréciera utiliser le PIC 16F84 tant il saura lui simplifier la vie pour tous ses montages, quels qu'ils soient.

Pour conclure, nous pouvons dire que dans bien des cas, le PIC 16F84 nous permettra d'arriver à nos fins avec un minimum de difficultés, et un maximum de simplicité.

CHAPITRE 8 : LEXIQUE

Adresse : correspond à l'endroit où l'on peut trouver les informations recherchées

Architecture : terme servant à définir la façon dont est organisée la structure d'un Circuit Intégré, tel un microcontrôleur

Bit : unité de base en informatique

BUS : sorte « d'autoroutes » pour les signaux, les BUS servent à transporter les informations entre deux parties du Circuit Intégré

Code : nom donné au texte constituant un programme informatique

DIL : norme de Circuit Intégré, définissant notamment les écarts entre les pattes

E²PROM : Electrically Erasable Programmable Read Only Memory ; se dit d'une mémoire qui peut s'écrire et surtout s'effacer électriquement

Flash : nouveau type de mémoire, avec de nouvelles possibilités

Microcontrôleur : Circuit Intégré (CI), contenant à la fois un processeur, des mémoires, et des entrées/sorties externes

Octet : ensemble de 8 bits

Oscillateur : se dit d'un composant, ou ensemble de composants capables de générer un signal régulier

PCB : nom donné aux typon, servant à tirer les plaques de circuit imprimé

Port : se dit d'un ensemble, d'un groupement d'entrées/sorties

RAM : Random Access Memory, aussi appelée mémoire vive, ce type de mémoire perd toutes ses informations stockées, lorsqu'elle cesse d'être alimentée

RISC : Reduced Instruction Set Computer ; se dit d'un processeur, ou microcontrôleur, possédant un nombre d'instructions de programmation réduits, limités

Schéma d'implantation : schéma montrant la façon, et le sens si nécessaire, dont doivent être montés les composants sur le circuit imprimé

Schéma Structurel : schéma montrant les représentations schématiques des composants, avec leurs liaisons électriques, correspondant aux pistes du PCB

Télécharger un programme : action de copier le code du PC dans le microcontrôleur

CHAPITRE 9 : LIENS INTERNET UTILES

Hervé Hollard, cours de langage C :

- http://perso.club-internet.fr/hhollard/
-http://perso.club-internet.fr/hhollard/prog_pic_c.htm

Adresse pour le programmateur de PIC :

-http://www.jdm.homepage.dk/newpic.htm
-http://www.jdm.homepage.dk/pcb2.htm

Adresse de Microchip :

http://www.microchip.com/

Adresse du compilateur CC5X :

http://www.bknd.com/cc5x/index.shtml

Adresse de l'auteur:

http://diablotronic.bzh.bz

CHAPITRE 10 : ANNEXES

Génération de musique :

```
/*************************************************************/
/****************Génération de son avec PIC**************/
/*************************************************************/
```

/*Ce programme permet de générer du son avec un PIC, par modulation de fréquence, de signal carré, sur un HP. Le HP est branché sur la patte 0 du PORTA. Au niveau de la ss-fct° HP, nous pouvons voir une boucle de test sur dur. Des problèmes apparaissent pour un dur>100; d'où cette boucle de test, permettant d'aller jusqu'à 200. Donc, si vous voulez augmentez cette capacité, n'oubliez pas de recréer une boucle de test sur 200.

A noter que ce programme ne possède pas d'algorithme afin de corriger les durées des notes en fonction de leur fréquence.
Par conséquent, pour une durée donnée, plus la note sera aigu (fréquence plus rapide), moins elle sera audible longtemps.
N'oubliez pas de tenir compte de ce petit problème. Peut-être ce problème sera-t-il modifié plus tard. La durée dépend également
de votre fréquence d'horloge. Seul des tests vous permettront de déterminer les bon temps.

Les notes définies, sont dans l'ordre, celles de la clé de sol. Pour d'autres notes, i faudra se référer au tableau de fréquence et faire vous même le calcul. Calculez la période (T=1/F en ms), puis divisez par 0,02. Vous aurez alors la valeur équivalente du soft.

La tempo inter() est à mettre entre chaque note jouée, afin de pouvoir faire une distinction correcte de chaque note. Exemple d'appel: HP(SOL, 75);*/

```
bit son @ PORTA.0;

const int do=180;
const int ré=170;
const int mi=150;
const int fa=145;
const int sol=128;
const int la=107;
const int si=101;
const int do=93;
const int ré=86;


void delay (int mill);
void inter (void);
void HP (int note, int dur);

void main (void)
{
        TRISA=0x00;
        TRISB=0xFF;

}
```

```
void delay (int mill)
{
	OPTION=0 ;
	do
	{
		TMR0=0 ;
		while(TMR0<1)
			;
	}while(--mill>0) ;
}

void inter (void)
{
	int i, j=500;
	for(i=0;i<50;i++)
	delay(j);
}


void HP (int note, int dur)
{
	int i;

	if(dur>100)
	{
		for (i=0;i<dur-100;i++)
		{
			son=1;
			delay(note);
			son=0;
			delay(note);
		}
		for(i=0;i<100;i++)
		{
			son=1;
			delay(note);
			son=0;
			delay(note);
		}
	}

	else

		for (i=0;i<dur;i++)
		{
			son=1;
			delay(note);
			son=0;
			delay(note);
		}
}
```

Gestion d'un LCD avec HD44780 :

```
/**************************************/
/****gestion d'un afficheur LCD****/
/**************************************/

/*Cette fonction permet d'utiliser un afficheur LCD parallèle en 8 bits. Attention tout de même car des différences peuvent apparaître d'un écran à l'autre, notamment au niveau des adresses mémoires, selon le nombres de lignes de l'écran, vous aurez donc à changer quelques valeurs dans 'initialisation.

   remarque sur le LCD: •quand RSOFF est actif, les données( DATA) sont pr la gestion du LCD.
                        quand RSON est actif, les données, sont pour l'affichage (code hexa)
                        •Pour qu'une donnée soit prise en compte, il faut la placer sur DATA, puis
                        appeler valid_DATA (création d'un front descendant)*/


//Declaration des librairies nécessaires

#include "MATH16.H"
#include "MATH24F.H"


//Renommage des E/S

char data@PORTB;

#define ENABLE PORTA.1=1
#define DISABLE PORTA.1=0
#define RSON PORTA.0=1
#define RSOFF PORTA.0=0


//Declaration des fonctions utiles

void delay(int mill);
void valid_DATA(void);
void init_LCD(void);


void delay(int mill)
{
        OPTION=2;
        do
        {
                TMR0=0;
                while(TMR0<125)
                        ;
        }while(--mill>0);
}


void valid_DATA(void)
{
        delay(1000);
```

```
        ENABLE;
        delay(1000);
        DISABLE;
        delay(1000);
}



void init_LCD(void)
{
        RSOFF;
        delay(1000);
        data=0x30;
        valid();
        valid();
        valid();
        data=0x38;
        valid();
        data=0x0C;
        valid();
        data=0x01;
        valid();
        data=0x06;
        valid();
        delay(1000);
}
```

Conversion numérique hexadécimal :

/*Ce petit sous programme permet de convertir un nombre décimal en son code hexa, ce qui est fort pratique, lorsqu'on doit afficher le résultat d'une opération (effectué en décimal) sur un afficheur LCD (code a envoyé en hexa)*/

```
void conv_nbr(int val)
{
	switch(val)
	{
		case 0: aff=0xFE;
				break;
		case 1: aff=0x98;
				break;
		case 2: aff=0xB7;
				break;
		case 3: aff=0xBD;
				break;
		case 4: aff=0xD9;
				break;
		case 5: aff=0xED;
				break;
		case 6: aff=0xEF;
				break;
		case 7: aff=0xB8;
				break;
		case 8: aff=0xFF;
				break;
		case 9: aff=0xF9;
				break;
		case 10: aff=0xFB;
				 break;
		case 11: aff=0x8F;
				 break;
		case 12: aff=0x87;
				 break;
		case 13: aff=0x9F;
				 break;
		case 14: aff=0xF7;
				 break;
		case 15: aff=0xE3;
				 break;
	}
}
```

Gestion d'un clavier 12 touches :

```
/************************************************/
/******Gestion d'un clavier 12 touches*******/
/************************************************/

/* Ce petit sous programme vous permettra de connaitre quel touche est enfoncée. */

char data@PORTB;

int clavier (void)
{

TRISB=0x0F;

int i, nb, touche, rd, c;

for (i=0;i<4;i++)
{
        data=(0x10)*2^i;
        rd=data&0x0F;

        switch(rd)
        {
                case 0x00:break;

                case 0x01:c=1;
                        break;

                case 0x02:c=2;
                        break;

                case 0x04:c=3;
                        break;
        }

        nb=3*i+c;

        if(nb<=9)
                touche=nb
        if(nb==10)
                touche='*';
        if(nb==11)
                touche=0;
        if(nb==12)
                touche='#';
}

return (touche);

}
```