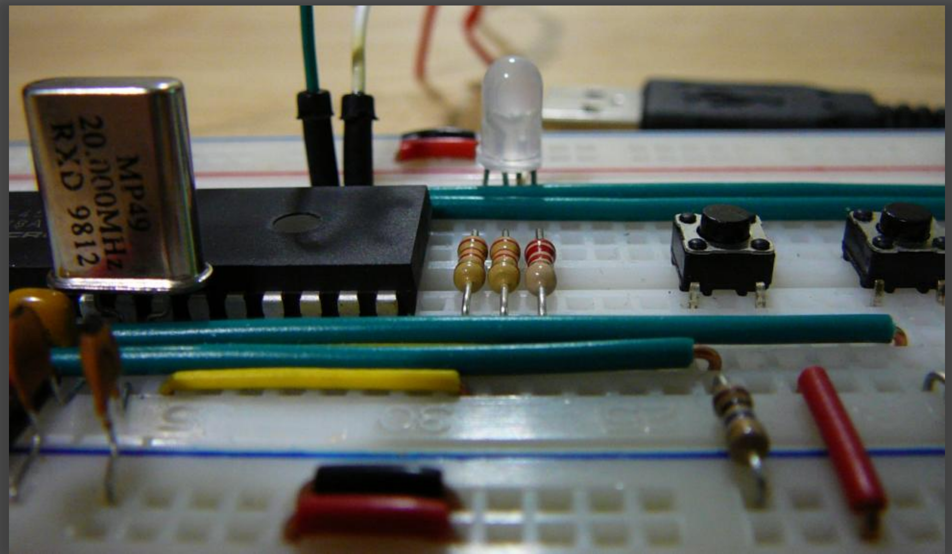
2012

CTD_01

Document tutoriel pour débutant

IFT592 - Projet informatique - Microcontrôleur programmable PIC18F4550



Patrice Bonneau
Université de Sherbrooke
2012-02-20

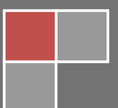


Table des matières

Données de publication ... iv

Historique des révisions ... iv

Sommaire ... iv

Présentation du projet ... 1

Le microcontrôleur programmable ... 1

Le programmateur PICkit 3 ... 3

L'environnement de développement MPLAB ... 3

Démarrage ... 3

Connexion au PIC18F4550 ... 4

Chargement du code dans le microcontrôleur ... 4

Débogage ... 4

Fenêtres d'affichage ... 4

Liste des pièces nécessaires au projet ... 5

Outils à prévoir ... 5

Langage d'assemblage ... 6

Lexique du langage d'assemblage ... 6

Structure du code et nomenclature ... 6

Exemple ... 7

Jeu d'instructions ... 8

Montage électronique ... 11

Les bases ... 11

Les principales composantes ... 11

La planche de montage (breadboard) ... 11

Les résistances ... 12

Les condensateurs ... 13

Les DEL (diodes électroluminescentes) ... 13

L'oscillateur ... 14

Premier programme - Les sorties numériques ... 14

Le montage du prototype ... 14

Le programme en langage d'assemblage ... 16
La configuration initiale ... 16
Définition des constantes ... 17
Déclaration des variables ... 17
Adresses de démarrage et d'interruptions ... 18
Initialisation ... 18
Le coeur du programme ... 19
Chargement du programme dans le microcontrôleur ... 22
Tests sur le sous-programme et les « diviseurs » ... 23
Les macros ... 24
La réinitialisation (reset) ... 26
Montage du prototype ... 26
Valeurs après réinitialisation ... 27
Les minuteurs (timers) ... 28
Le « WatchDog Timer » ... 28
Les entrées numériques ... 29
Vérification du signal par boucle ... 29
Montage ... 29
Code assembleur ... 30
Vérification du signal par interruption ... 32
Montage ... 32
Code assembleur ... 34
Les communications USB ... 37
Détection du périphérique par l'ordinateur ... 42
Code source du microcontrôleur ... 43
Le logiciel interactif ... 47
Fonctionnement ... 48
Explication du code source ... 49
Les entrées analogiques ... 51
Montage électronique ... 52
Programmation du microcontrôleur ... 53

Données de publication

Historique des révisions

version	date	auteur	description
0.1.0	2012-02-06	PB	Présentation du projet Langage d'assemblage
0.1.1	2012-02-12	PB	Modifications mineures de quelques phrases. Ajout du cadencement dans la section « Le microcontrôleur programmable »
0.2.0	2012-02-20	PB	Jeu d'instructions Montage électronique Premier programme - les sorties numériques Les minuteurs (timers)
0.3.0	2012-03-05	PB	Mise à jour de la section « Jeu d'instruction » Réinitialisation Les entrées numériques La gestion des événements et des interruptions
0.3.1	2012-03-20	PB	Corrections suite aux recommandations de Richard St-Denis.
0.4.0	2012-04-22	PB	Les communications USB Les entrées analogiques

Sommaire

Ce document apporte un apprentissage de type tutoriel, pour permettre une première approche simple afin de s'initier à la programmation des microcontrôleurs programmables. Plus précisément, le microcontrôleur PIC18F4550 fabriqué par Microchip. Il couvre la conception du prototype, l'utilisation du programmateur et l'utilisation de l'environnement de développement. Ce document s'adresse autant à des développeurs débutants qu'à des développeurs chevronnés.

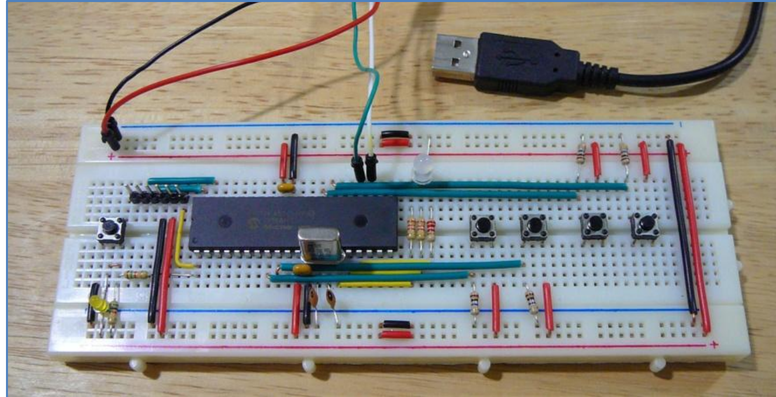


Figure 1

Présentation du projet

Le projet consiste à créer un périphérique USB (Universal Serial Bus) à partir d'un microcontrôleur programmable PIC18F4550 de 40 broches, fabriqué par Microchip. Le périphérique sera un clavier binaire de quatre touches, soit les touches [0], [1], [ENTRÉE] et [EFFACER]. Un cinquième bouton servira à réinitialiser le microcontrôleur. Le prototype peut être vu à la figure 1 ci-dessus.

Le microcontrôleur sera programmé en langage d'assemblage. Le code assemblé lui sera transmis par un programmateur/débogueur appelé « PICkit 3 », spécialement conçu à cette fin et également fabriqué par Microchip. Le périphérique, créé de toutes pièces sur une planche de montage électronique, démontrera les possibilités d'entrée/sortie numérique à l'aide de boutons-poussoirs et de DEL, et agira en tant que clavier de quatre touches. Il montrera également les possibilités d'entrée analogique à l'aide de potentiomètres. La liste complète des pièces et outils nécessaires est présentée plus loin dans ce document.

Le microcontrôleur programmable

Les microcontrôleurs de la famille des PIC sont composés, d'abord et avant tout, d'un microprocesseur de type RISC (Reduced Instruction Set Computer*, c.-à-d. : jeu d'instructions simple*). Les instructions envoyées au micro-processeur sont, à quelques exceptions près, de taille identique, soit 16 bits, et s'exécutent à raison d'un cycle par instruction.

À titre de comparaison, il existe également des microprocesseurs de type CISC (Complex Instruction Set Computer*, c.-à-d. : jeu d'instructions complexe*). Les instructions d'un micro-processeur CISC peuvent s'exécuter sur plusieurs cycles. Ceci est dû au fait qu'elles peuvent se composer de plusieurs sous-instructions. Un jeu d'instructions complexes facilite grandement le travail du programmeur, puisque le code produit est plus petit. Par contre, un microprocesseur RISC permet une meilleure optimisation.

Les microcontrôleurs PIC possèdent une architecture Harvard. À l'instar de l'architecture Von Neumann, qui utilise un seul bus pour l'échange des données sur une mémoire unique, l'architecture Harvard utilise deux bus de communication, sur deux mémoires distinctes. Soit une mémoire pour le programme et une mémoire pour les données. Voyez la différence sur la figure 2.

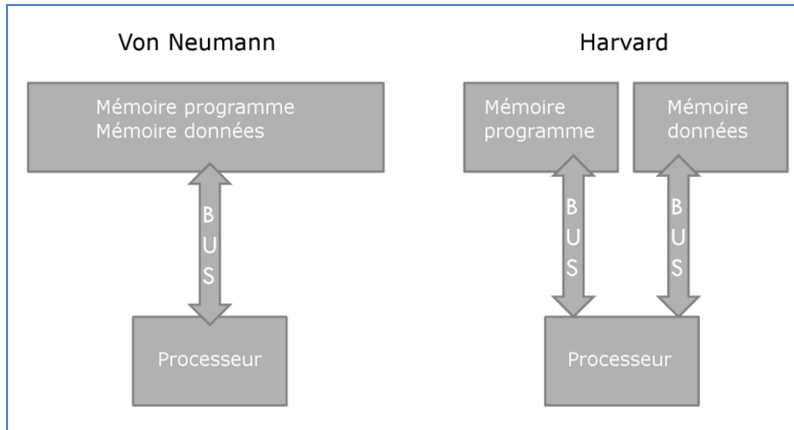


Figure 2

Dans le cas du PIC18F4550, la mémoire pour le programme fait une taille de 32 768 octets. Les instructions s'écrivant sur 16 bits (2 octets), nous pouvons créer des programmes allants jusqu'à 16 384 instructions. Quant à la mémoire pour les données, elle fait une taille de 2 048 octets.

Le PIC18F4550, bien que constitué de 40 broches, n'offre que 35 broches en E/S (entrée/sortie) numérique, dont 13 peuvent être utilisées en entrée analogique. Les cinq autres sont réservées à l'alimentation électrique. Les 35 broches d'E/S sont regroupées en cinq ports de communication bidirectionnels, nommés de A, B, C, D et E.

Il est à noter que chacune des 35 broches peut jouer plusieurs rôles, mais un seul rôle à la fois. Nous reviendrons plus tard sur les rôles qu'elles prennent par défaut lors de la réinitialisation du microcontrôleur. Par contre, il peut être intéressant de noter qu'une broche configurée en E/S numérique porte un nom qui commence par la lettre R, suivie de la lettre correspondante à son port, puis termine par son numéro. Par exemple, la broche qui correspond à la première broche du port A a comme nom RA0. Eh oui, on commence à compter à zéro. Pour une broche configurée en entrée analogique, son nom commence par les lettres AN et termine par son numéro. Par exemple, AN8. Il existe un seul port d'entrée analogique.

Vous pouvez voir sur la figure ci-contre les différents rôles que peuvent prendre chacune des broches du PIC18F4550. Remarquez la demi-lune en haut du microcontrôleur, ainsi que le petit point au dessus du « 1 ». Ceci permet d'identifier physiquement la broche numéro 1 sur votre PIC. Les flèches quant à elles indiquent si une broche est bidirectionnelle, ou le sens de la communication dans le cas d'une broche unidirectionnelle. V_{DD} et V_{SS} indiquent respectivement les bornes positives et négatives de l'alimentation. OSC1 et OSC2 servent pour l'oscillateur et MCLR sert à la réinitialisation du microcontrôleur.

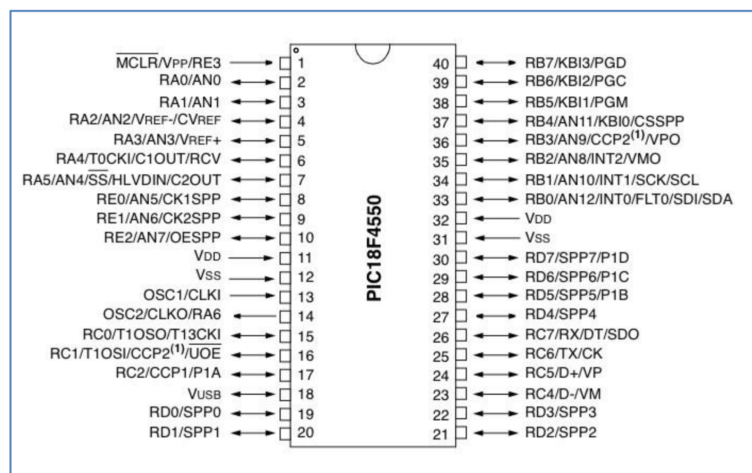
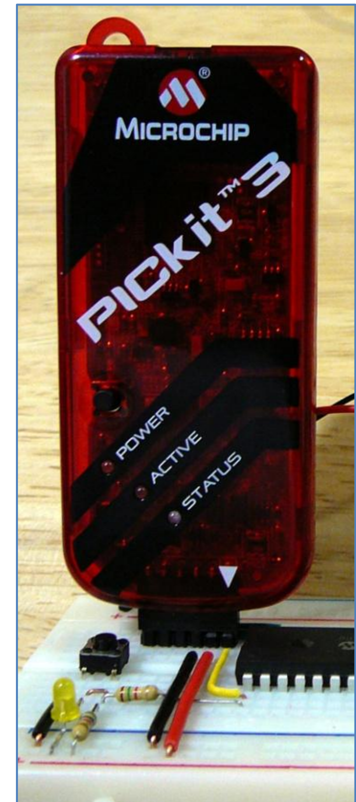


Figure 3

Le PIC18F4550 peut être cadencé de deux manières. Tout d'abord, celle que nous utiliserons, par un oscillateur externe. Cette méthode nous permettra d'atteindre des fréquences allant jusqu'à 48 MHz. La deuxième façon de cadencer le PIC est d'utiliser le bloc oscillateur interne. Ce bloc est constitué de deux oscillateurs. Un premier qui est cadencé à 8 MHz, appelé INTOSC (internal oscillateur) et un deuxième beaucoup plus lent cadencé à 31 kHz, appelé INTRC (internal RC oscillator). Ce dernier est utilisé, entre autres, par les minuteurs.

Le PIC18F4550 est également muni d'un module de communication USB pleinement fonctionnel, qui est entièrement compatible avec les spécifications USB, révision 2.0. Le module supporte autant les communications basse vitesse (1,5 Mbit/s*, c.-à-d. : USB 1.1*) que les communications haute vitesse (12 Mbit/s*, c.-à-d. : USB 2.0*).



Le programmateur PICkit 3

Le programmateur PICkit 3 est l'outil essentiel qui nous permettra de mettre notre code exécutable dans le microcontrôleur. Il est relié à l'ordinateur par le port USB, et au microcontrôleur par un support ICSP (In-Circuit Serial Programming). La connexion au circuit est effectuée à l'aide de six broches. La position de la broche numéro 1 est identifiée par le triangle blanc sur le PICkit. Cette broche doit être reliée à la broche numéro 1 du PIC18F4550. Sur la photo ci-contre, on remarque qu'ils sont reliés par un câble jaune.

Le PICkit 3 ne sert pas uniquement à transférer du code exécutable. Il sert également à déboguer le code présent dans le microcontrôleur. Le débogage, tout comme la programmation du microcontrôleur, s'effectue très aisément avec l'environnement de développement MPLAB. De plus, le PICkit 3 permet d'alimenter le PIC18F4550 dans le cas où ce dernier ne serait pas encore relié à un port USB.

L'environnement de développement MPLAB

MPLAB est l'application qui fait le lien entre le programmeur et le microcontrôleur. Non seulement il permet de taper du code, de l'assembler et de le charger dans le microcontrôleur, mais il offre également un outil de simulation qui permet de déboguer son code de la même manière que s'il était dans le microcontrôleur. Voici quelques-unes de ses fonctionnalités.

Démarrage

MPLAB fonctionne avec des projets. Donc pour commencer, il faut aller dans le menu et cliquer sur « Project >> New... ». Il suffit de donner un nom au projet, et de choisir un répertoire de travail. Notez qu'aucun sous-répertoire ne sera créé dans le répertoire choisi. Pensez à vous créer un sous-répertoire pour contenir les quatre fichiers qui seront créés. Dans le menu « Project », vous trouverez les options de construction et le choix de la version (debug ou release).

Pour commencer à coder, il faut créer un nouveau fichier ASM. Pour ce faire, allez dans le menu et cliquez sur « File >> New » ou appuyez sur CTRL + N. Une nouvelle fenêtre s'ouvrira pour vous permettre de taper votre code. Ne reste plus qu'à enregistrer votre nouveau fichier qui contient le code avant de l'assembler. Pour construire (assembler) le code, il suffit d'appuyer sur F10.

Connexion au PIC18F4550

Une fois que le PICkit 3 est connecté au port USB et au microcontrôleur, il suffit d'aller dans le menu et de cliquer sur « Programmer >> Select Programmer » puis d'y choisir le PICkit 3. Le programmateur tentera de se connecter automatiquement au microcontrôleur.

Si vous voulez alimenter le microcontrôleur à partir du PICkit 3, il faut modifier une option après que le PICkit aura tenté de se connecter une première fois. Pour ce faire, il faut aller dans le menu, sur « Project >> Settings... ». Dans la nouvelle fenêtre, cliquez sur l'onglet « Power » et cochez l'option « Power target circuit from PICkit 3 ». Cliquez sur « OK » et le programmateur tentera de se reconnecter automatiquement. Lorsque la connexion est un succès, la fenêtre « output » affiche « Device ID Revision = nnnnnn ».

Chargement du code dans le microcontrôleur

Une fois connecté au microcontrôleur, il suffit d'aller dans le menu et de cliquer sur « Programmer >> Program ». C'est tout. Notez que vous disposez d'autres options très utiles dans le menu « Programmer ».

Débogage

Pour lancer le débogage, il faut aller dans le menu et cliquer sur « Debugger >> Select tool ». Une fois dans le menu de sélection de l'outil, nous avons deux choix. Premièrement, pour déboguer le code dans l'éditeur, sans toucher au microcontrôleur, il faut choisir « MPLAB SIM ». Deuxièmement, pour déboguer le code directement dans le PIC18F4550, il faut choisir « PICkit 3 », à condition bien sûr qu'il soit connecté au microcontrôleur.

Il existe plusieurs options de débogage via le menu. Les plus utilisées ont des raccourcis-clavier que voici :

- F2 ---> Breakpoint
- F6 ---> Processor reset
- F7 ---> Step into
- F8 ---> Step over
- F9 --- > Run

Fenêtres d'affichage

Plusieurs fenêtres d'affichage sont offertes par MPLAB, spécialement utiles lors du débogage. Elles sont accessibles par le menu « View », dont voici les plus importantes :

- Special Function Registers (SFR) → Permet de voir le contenu de tous les registres.
- Watch → Permet à l'usager de définir les registres et variables dont il veut surveiller le contenu.
- Output → Permet de voir les résultats de connexion et de construction.
- Program Memory → Permet de voir le contenu de l'espace mémoire du programme.

Liste des pièces nécessaires au projet

Voici la liste des pièces nécessaires pour la conception du prototype. Notez que pour le bas prix du microcontrôleur, il peut être bon de s'en procurer plus d'un, surtout si vous désirez faire l'exercice d'en faire communiquer deux ensemble.

Quantité	Description
1	Microchip PIC18F4550
1	Programmateur PICkit 3 (optionnellement, un adaptateur ICSP ZIF 40) **Note** : Un adaptateur ICSP est facultatif et ne sera pas utilisé dans le cadre de ce tutoriel. Il n'a d'utilité que si vous voulez programmer plusieurs microcontrôleurs en série.
1	Planche de montage (breadboard) de 840 points
Quantité variable.	Fils pleins (pas torsadés) de calibre 22. 1 de chaque couleur : noir, rouge, vert.
	Gaine thermorétractable.
1	- Câble USB 2.0. Un bout Male de type « A ». Aucune importance pour l'autre bout, car il sera coupé.
5	Mini interrupteurs momentanés (push button)
1	Oscillateur au cristal de quartz de 20MHz
1 1	DEL jaune de 3mm (5v) DEL RGB de 5mm, cathode commune (5v) **Note** : La DEL RGB peut facilement être remplacée par trois DEL distinctes, de couleurs différentes.
1	Mini potentiomètre, entre 4 et 10 kΩ.
 4 1 3 1	Résistances (les valeurs peuvent différer un peu) 10 mΩ --> Brun, noir, bleu. 5,6 kΩ --> Vert, bleu, rouge. 2,2 kΩ --> Rouge, rouge, rouge. 1,5 kΩ --> Brun, vert, rouge.
 1 2 2	Condensateurs 470 nF. 100 nF. 15 pF.

Outils à prévoir

Voici les outils dont vous aurez besoin pour compléter ce projet. Notez que certaines pinces peuvent se regrouper en une seule.

- Pince coupantes.
- Pince à courber.
- Pince à dénuder.
- Fer à souder.
- Étain.

Optionnellement, prévoir un voltmètre.

Langage d'assemblage

Le langage d'assemblage est, en programmation informatique, un langage de bas niveau. Ce qui signifie qu'il est très proche du langage machine, qui est le seul langage compris par les processeurs qui ne gèrent que des 1 et des 0. Contrairement à un langage de haut niveau, tel que le C++, un programme en langage d'assemblage n'est pas compilé, mais assemblé. C'est pour ainsi dire, une traduction ligne par ligne du code en langage d'assemblage au code en langage machine, afin de créer un exécutable.

Lexique du langage d'assemblage

Terme	Description
Mnémonique	Nom abrégé par lequel une instruction est appelée.
Opérande	Argument d'une instruction. Peut être un registre, une adresse mémoire, un littérale ou une étiquette
Registre	Espace mémoire d'accès très rapide, mais en quantité limitée, situé à l'intérieur du processeur.
Adresse mémoire	Identifie un emplacement dans la mémoire RAM (séparée du processeur).
Littérale	Une valeur donnée explicitement dans le code source d'un programme. Par exemple, dans « `maVariable = 3` », le chiffre 3 est un littérale.
Étiquette	Permet d'identifier textuellement un endroit précis dans le code afin de permettre d'y accéder à l'aide d'instruction de saut (ex. GOTO). Lors de l'assemblage, elle sera convertie en adresse utilisable par le programme.

Structure du code et nomenclature

1. Les étiquettes sont alignées sur la marge gauche, sans tabulation, et se terminent par deux points (:).
2. Les « #DEFINE » sont également alignées sur la marge gauche, sans tabulation.
3. Les mnémoniques sont indentées d'une tabulation.
4. Suite aux mnémoniques (sur la même ligne), les opérandes sont indentés d'une autre tabulation, et sont séparés entre eux par des virgules, sans tabulation supplémentaire.
5. Le code doit se terminer par « END ».
6. Le caractère « ; » est utilisé pour insérer des commentaires de ligne.
7. La base d'un littéral numérique est définie par une lettre majuscule et la valeur est écrite entre apostrophes. Par exemple, pour écrire la valeur 26 dans l'une ou l'autre des bases courantes, nous procédons comme suit :
    - Binaire ---> `B'00011010'`
    - Décimal ---> `D'26'`
    - Hexadécimal ---> `H'1A'`

Exemple

Voici un petit exemple de code. Bien entendu, certaines parties ont été retirées du code fonctionnel original par souci d'économie d'espace. En gros, ce code fait allumer trois fois en alternance les DEL (diode électro luminescente) rouge, bleue et verte.

```
       LIST          P=18F4550,     F=INHX32       ; Définition de processeur
       #include      <P18F4550.INC>                ; Inclusion du fichier des variables
       CONFIG FOSC=HS, WDT=OFF, DEBUG=OFF

#DEFINE DEL_B        PORTC, RC0                    ; DEL bleue
#DEFINE DEL_V        PORTC, RC1                    ; DEL verte
#DEFINE DEL_R        PORTC, RC2                    ; DEL rouge

       CBLOCK 0x00C
              cptr : 1
       ENDC

       ORG    0x000                                ; Adresse de départ après reset
       goto    init

init:
       movlw  D'3'
       movwf  cptr
       goto          main

main:
       bsf    DEL_R                ; Allumer la DEL rouge.
       call   faireUnePause
       bcf    DEL_R                ; Éteindre la DEL rouge.

       bsf    DEL_B                ; Allumer la DEL bleue.
       call   faireUnePause
       bcf    DEL_B                ; Éteindre la DEL bleue.

       bsf    DEL_V                ; Allumer la DEL verte.
       call   faireUnePause
       bcf    DEL_V                ; Éteindre la DEL verte.

       decfsz cptr,1               ; Décrémenter le compteur et vérifier si on a atteint 0.
       goto   main                 ; Si oui, on saute cette ligne, sinon on retourne à « main ».

       END
```

Jeu d’instructions

Chaque instruction est constituée d’une mnémonique et d’un ou plusieurs opérandes, qui précisent le fonctionnement de l’instruction. Voici celles qui seront utilisées dans ce tutoriel. À noter que certaines options de ces instructions ne sont pas abordées ici.

bsf et bcf (1 cycle d’instruction)	
Définition	*bsf (bit set file)* : Met un bit précis à ‘1’ dans un registre donné. *bcf (bit clear file)* : Met un bit précis à ‘0’ dans un registre donné.
Syntaxe	`bsf registre, bit` `bcf registre, bit`
Définition	*registre* : adresse ou identifiant d’un registre *bit* : numéro ou identifiant d’un bit
Exemple	`bsf TRISB, RB2` Contenu du registre TRISB avant l’instruction : 00000000 Contenu du registre TRISB après l’instruction : 00000100

clrf (1 cycle d’instruction)	
Définition	*clrf (clear file)* : Met tous les bits d’un registre à ‘0’.
Syntaxe	`clrf registre`
Définition	*registre* : adresse ou identifiant d’un registre
Exemple	`clrf PORTC` Contenu du registre PORTC avant l’instruction : 00101110 Contenu du registre PORTC après l’instruction : 00000000

movlw & movwf (1 cycle d’instruction)	
Définition	*movlw (move literal to work directory)* : Met une valeur littérale dans le registre de travail « w ». *movwf (move from work directory to file)* : Copie la valeur contenue dans le registre de travail « w » vers un registre. ***Note :*** *Il n’est pas possible de mettre une valeur littérale directement dans un registre.*
Syntaxe	`movlw littéral` `movwf registre`
Définition	*littéral* : Valeur donnée en binaire (B), en décimal (D) ou en hexadécimal (H). *registre* : adresse ou identifiant d’un registre

Exemple	`movlw D’14’ → équivaut à 00001110 en binaire`
	`movwf TRISA`
	
	Contenu du registre TRISA avant l’instruction : 00000000
	Contenu du registre TRISA après l’instruction : 00001110

goto **(2 cycles d’instruction)**	
Définition	*goto* : Poursuit l’exécution du programme à l’endroit identifié par l’étiquette, plutôt que de poursuivre avec l’instruction suivante.
Syntaxe	`goto étiquette`
Définition	*étiquette* : Identifie textuellement un endroit précis dans le code.

Exemple	`        instruction 1`
	`        goto finProgramme`
	`        instruction 2`
	`finProgramme:`
	
	Dans cet exemple, seul l’instruction 1 sera exécutée. À l’instruction « goto », le programme sautera à l’étiquette « finProgramme ».

nop **(1 cycle d’instruction)**	
Définition	*nop (no operation)* : N’exécute rien, mais consomme un cycle d’instruction. Souvent utilisé après une instruction de branchement pour éviter qu’une autre exécution soit chargée dans le pipeline.
Syntaxe	`nop`

Exemple	`        instruction 1`
	`        goto finProgramme`
	`        nop`
	`        instruction 2`
	`finProgramme:`
	
	Dans l’exemple précédent, l’instruction 2 serait chargée inutilement dans le pipeline et pourrait provoquer un événement inattendu. L’exemple ci-dessus montre la bonne façon de faire. Cependant, MPLAB comblera votre oubli lors de l’assemblage, le cas échéant.

decfsz **(1, 2 ou cycles d’instruction)**	
Définition	*decfsz (decrement file skip if zero)* : Décrémente un registre, puis vérifie le résultat. Si le résultat est égal à zéro, alors on saute par-dessus l’instruction suivante. ***Note :*** *Lorsque le résultat est <u>différent</u> de zéro, « decfsz » consomme 1 cycle d’instruction. Lorsque le résultat est <u>égal</u> à zéro, « decfsz » consomme deux cycles d’instruction. Dans ce cas, si l’instruction suivante est une instruction de branchement, alors un cycle d’instruction de plus serait consommé.*
Syntaxe	`decfsz registre`
Définition	*registre* : adresse ou identifiant d’un registre

Exemple	`debut:`
	`        ...`
	`        decfsz maVariable`
	`        goto debut`
	`        ...`
	
	Ce code bouclera tant que « maVariable » sera différente de zéro. Lorsqu’elle atteindra zéro, l’instruction « goto » sera ignorée et le programme poursuivra avec l’instruction qui suit.

btfsc & btfss (1, 2 ou cycles d'instruction)	
Définition	*btfsc (Bit Test f, Skip if Clear)* : Vérifie la valeur d'un bit. Si le résultat est égal à zéro, alors on saute par-dessus l'instruction suivante. *btfsc (Bit Test f, Skip if Set)* : Vérifie la valeur d'un bit. Si le résultat est égal à un, alors on saute par-dessus l'instruction suivante. ***Note :*** *Lorsque le résultat est <u>différent</u> de zéro, « btfsc » et « btfss » consomment 1 cycle d'instruction. Lorsque le résultat est <u>égal</u> à zéro, ils consomme deux cycles d'instruction. Dans ce cas, si l'instruction suivante est une instruction de branchement, alors un cycle d'instruction de plus serait consommé.*
Syntaxe	`btfsc registre, bit` `btfss registre, bit`
Définition	*registre* : adresse ou identifiant d'un registre *bit* : numéro du bit à vérifier (0 à 7) ou identifiant du bit

Exemple	`debut:`
	`      ...`
	`      btfsc monRegistre, 3`
	`      goto debut`
	`      ...`
	Ce code bouclera tant que le bit 3 de « monRegistre » sera différente de zéro. Lorsqu'il atteindra zéro, l'instruction « goto » sera ignorée et le programme poursuivra avec l'instruction qui suit.

Montage électronique

Le montage d'un prototype électronique nécessite de connaitre certaines règles de base. En effet, il serait bien dommage de détruire nos composantes toutes neuves avant même de taper ses premières lignes de code.

Les bases

La première règle de base à connaitre est sans aucun doute qu'aucun lien ne doit être fait directement entre une borne V_{DD} (borne positive) et une borne V_{SS} (borne négative). La raison est simple. S'il n'y a rien pour ralentir le flux des électrons entre les deux bornes, les matériaux surchaufferont dû à la circulation rapide des électrons et certaines composantes risqueront même d'exploser. Cela dit, il arrive que nous devions relier ces deux bornes ensemble, pour diminuer la tension par exemple. Dans une telle situation, une résistance adéquate sera utilisée pour unir les deux bornes, et ainsi limiter le flux des électrons. Cette technique sera utilisée lors de la conception de la réinitialisation du microcontrôleur, où nous devrons provoquer une baisse de tension sur la broche numéro 1.

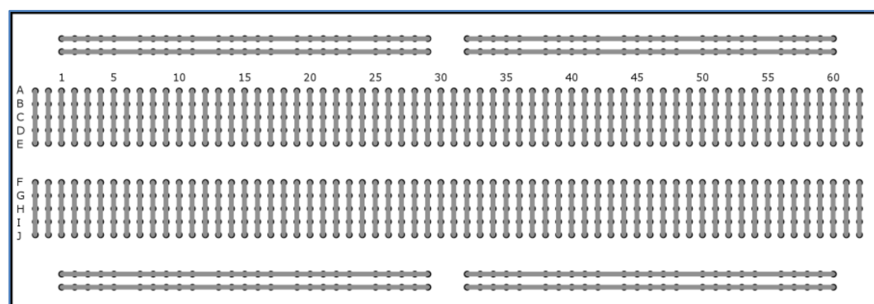
Une autre notion de base importante est de connaitre le sens de la circulation des électrons dans un circuit électronique. Alors à vote avis, les électrons circulent de la borne V_{DD} vers V_{SS}, ou inversement? Eh bien, les deux réponses sont bonnes, dépendamment de si vous faites face à un diagramme ou à un circuit.

Le fait que les diagrammes et les circuits ont des sens de circulation différents vient d'une erreur survenue au tout début des études sur la conduction de l'électricité. Au commencement, les scientifiques ont pensé que les particules qui se déplaçaient dans les métaux étaient chargées positivement et se dirigeaient vers le pôle négatif. C'est ce qu'on appelle le « sens conventionnel du courant ». Plus tard on a mis en évidence que ce sont très majoritairement les électrons, particules chargées négativement, qui se déplacent dans les métaux et qui permettent la circulation des courants électriques. C'est ce qu'on appelle le « sens réel du courant ». C'est donc cette erreur qui fait qu'aujourd'hui, nous utilisons le sens conventionnel dans les diagrammes, par opposition au sens réel.

Les principales composantes

La planche de montage (breadboard)

Les planches de montage sont grandement appréciées pour monter rapidement des prototypes, par le fait qu'elles ne nécessitent aucune soudure, et sont réutilisables. Les fils et composantes sont simplement insérés dans les trous et y sont retenus par pression. Ces trous sont groupés et reliés entre eux par des bandes métalliques situées sous la planche de montage. Pour vous aider à bien comprendre, voici un diagramme montrant, à l'aide de lignes grises, les trous qui sont reliés entre eux.



Les résistances

Les résistances ont pour rôle de ralentir le passage d'un courant électrique. Leur unité de mesure est l'ohm (symbole : Ω). On peut définir la valeur d'une résistance grâce aux anneaux de couleur qu'elle porte. Il existe des résistances à quatre, cinq et six anneaux. La signification de chacun des anneaux est expliquée ci-dessous.

Nombre d'anneaux	Anneau 1	Anneau2	Anneau3	Anneau4	Anneau5	Anneau6
4	Chiffre significatif	Chiffre significatif	Multiplicateur	Tolérance		
5	Chiffre significatif	Chiffre significatif	Chiffre significatif	Multiplicateur	Tolérance	
6	Chiffre significatif	Chiffre significatif	Chiffre significatif	Multiplicateur	Tolérance	Coefficient de température

La valeur correspondante à chacune des couleurs des anneaux (chiffre significatif et multiplicateur) est donnée dans le tableau ci-dessous. Prenez note que les couleurs argent et or ne sont utilisées que pour le multiplicateur. La valeur du multiplicateur représente les puissances de dix (10^n).

Valeur	-2	-1	0	1	2	3	4	5	6	7	8	9
Couleur	Argent	Or	Noir	Brun	Rouge	Orange	Jaune	Vert	Bleu	Violet	Gris	Blanc

Pour l'anneau de la tolérance, on utilise ce tableau. À noter que dans le cas des résistances qu'on utilise habituellement pour les circuits électroniques amateurs, la tolérance change très peu, soit 5 ou 10 %.

Valeur	10 %	5 %	20 %	1 %	2 %	0,5 %	0,25 %	0,10 %	0,05 %
Couleur	Argent	Or	Noir	Brun	Rouge	Vert	Bleu	Violet	Gris

L'anneau du coefficient de température quant à lui est limité aux couleurs ci-dessous. Ce coefficient correspond à l'augmentation ou la diminution de la valeur de résistance, provoquée par une variation de la température ambiante. Cette variation est en référence à la température d'opération normale, spécifiée par le fabricant. Ses valeurs sont exprimées en ppm (partie par million) par degré Celsius de variation.

Valeur	100	50	15	25	10	5	1
Couleur	Brun	Rouge	Orange	Jaune	Bleu	Violet	Blanc

Voici un tableau d'exemple pour des résistances à quatre anneaux, ayant chacune une tolérance de 5% (or).

Valeur	Anneau 1	Anneau2		Anneau3		Anneau4	Explication
47 ohm	Jaune	Violet		Brun		Or	47 x 1 = 47
5 600 ohm (5,6 kΩ)	Vert	Bleu		Rouge		Or	56 x 100 = 5 600
10 000 ohm (10 kΩ)	Brun	Noir		Orange		Or	10 x 1000 = 10 000

Notez que la résistance possède également un effet de bord, qui consiste à une dissipation de l'énergie sous forme de chaleur. On parle ici de l'effet Joule, selon le nom du physicien qui a étudié cet effet aux environs des années 1860. Selon le matériau utilisé, la dissipation peut être légère, telles les petites résistances que nous utiliserons pour notre circuit, ou peut être élevée, tel un élément chauffant.

Pour calculer la valeur nécessaire pour une résistance, on utilise la loi d'Ohm. Cette loi dit que la tension « U » (en volt) aux bornes d'une résistance « R » (en ohms) est proportionnelle à l'intensité du courant électrique « I » (en ampères) qui la traverse. Ce qui se traduit par la formule $\mathrm{U} = R \cdot I$. On peut donc en déduire que $\mathrm{R} = \frac{\mathrm{U}}{\mathrm{I}}$. Donc si nous avons une composante qui fonctionne sur une tension de 5v, à raison de 20 milliampères, nous aurons besoin d'une résistance de 250 ohms, soit $\frac{5}{0{,}02}$.

Les condensateurs

Les condensateurs sont des composants capables d'emmagasiner de l'énergie électrique, sous la forme d'un champ électrostatique, puis de la restituer. Ils sont généralement constitués de deux surfaces conductrices d'électricité (dites armatures), mises face à face et séparées par un isolant polarisable de faible épaisseur (dit diélectrique). Leur unité de mesure est le « farad ». Cela dit, le farad étant une unité trop importante pour un circuit électronique, on utilise généralement des sous-multiples, tels le picofarad, le nanofarad ou encore le microfarad.

Les condensateurs ont plusieurs utilités, mais dans le cadre de notre projet, ils serviront à stabiliser notre alimentation électrique (ils se déchargent lors des chutes de tension et se chargent lors des pics de tension). En effet, bien que le port USB offre une tension de 5v, cela ne signifie pas que nous avons toujours précisément 5v au millième près. Les condensateurs agiront donc un peu comme des tampons qui permettent de protéger le circuit contre ces microvariations de tension.

Les DEL (diodes électroluminescentes)

Les diodes sont des composantes électroniques qui permettent de contrôler la circulation dans un circuit électronique. Elles permettent au courant de circuler uniquement depuis une cathode (électrode reliée à V_{SS}) vers une anode (électrode reliée à V_{DD}). Nous parlons ici du sens réel du courant électrique. Si on connecte une diode en sens opposé, les électrons ne pourront plus circuler.

Une DEL (diode électroluminescente) quant à elle est une diode à part entière, mais qui transforme une partie des électrons en énergie lumineuse. Il existe une variété de couleurs pour les diodes électroluminescentes. Elles comportent généralement une seule couleur par DEL, mais il en existe qui peuvent contenir plusieurs couleurs.

Ces dernières sont soit des DEL avec anode communes, soit des DEL avec cathode commune. Une DEL à cathode commune possède une anode pour chaque couleur, et une même cathode qui est partagée par toutes les couleurs. Donc une DEL de trois couleurs possèdera quatre électrodes. Par déduction, on trouve facilement qu'une DEL à anode commune possède une cathode pour chaque couleur, et une même anode qui est partagée par toutes les couleurs.

Tout comme pour les autres composantes d'un circuit électronique, il peut arriver que nous ayons besoin de mettre une résistance en série avec une DEL. Cependant, l'utilisation d'une résistance avec une DEL n'est généralement nécessaire que lorsque la tension du circuit est supérieure à la tension pour laquelle la DEL a été conçue. Pour ce faire, nous utilisons la formule $R = \frac{\mathrm{U} - \mathrm{U}_{\mathrm{del}}}{\mathrm{I}}$ où « R » est la valeur de la résistance (ohms), « U » est la tension du circuit (volts), « U_{del} » est la tension requise par la DEL (volts) et « I » est l'intensité de la DEL (milliampères).

Par exemple, si nous avons une tension « U » de 5v, et que notre DEL a été conçue pour fonctionner sur une tension « U_{del} » de 1,8v avec une intensité « I » de 20 milliampères, nous aurons $R = \frac{5 - 1{,}8}{0.02} = 160\ ohms$.

L'oscillateur

Un oscillateur est un montage électronique, dont la fonction est de produire un signal périodique. Ce signal alterne donc de manière cyclique entre une tension basse (0v), et une tension haute (ex. 5v). Le signal d'un oscillateur peut être de forme sinusoïdale (souvent à base de transistors) ou rectangulaire (souvent à base de cristal de quartz).

Dans le cadre de notre projet, c'est sur ce dernier que se reposera notre microcontrôleur pour cadencer son horloge, et ainsi faire avancer son compteur ordinal à la fréquence donnée par l'oscillateur.

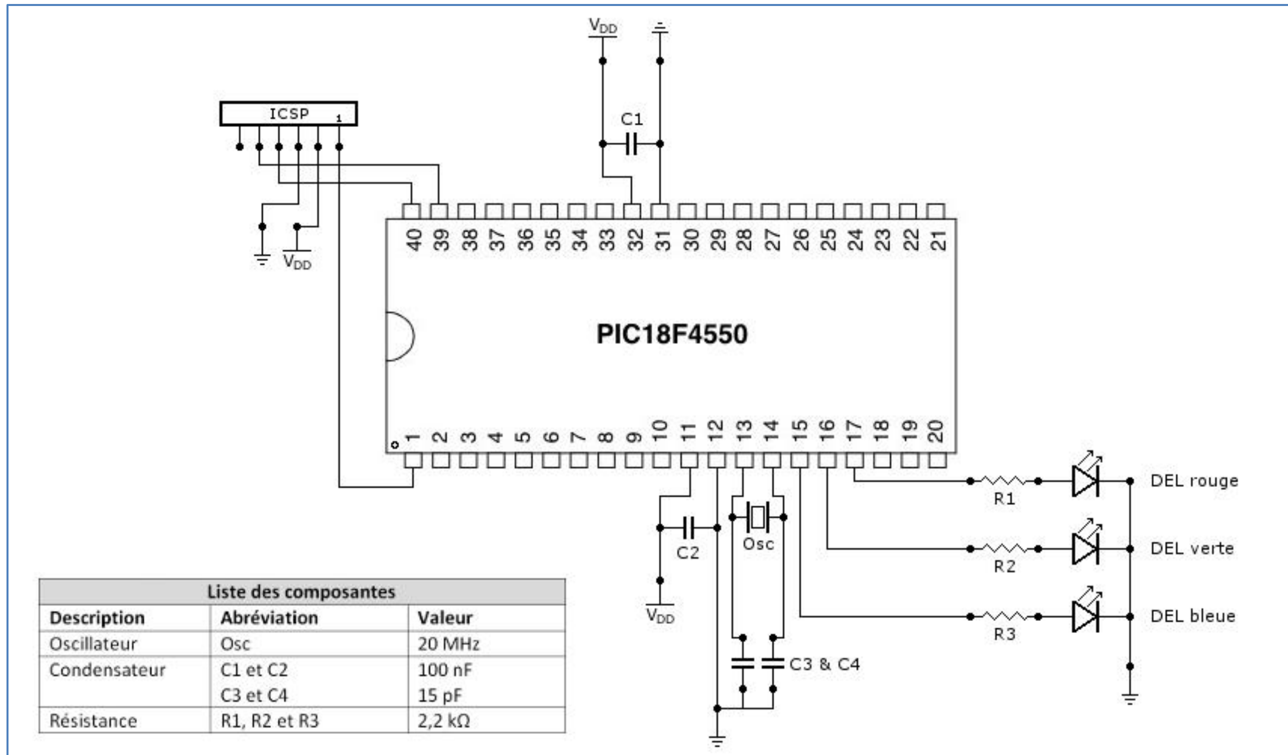
Premier programme - Les sorties numériques

Nous allons donc faire ici notre premier montage, ainsi que notre premier code, ce qui nous permettra de faire allumer trois DEL en alternance.

Le montage du prototype

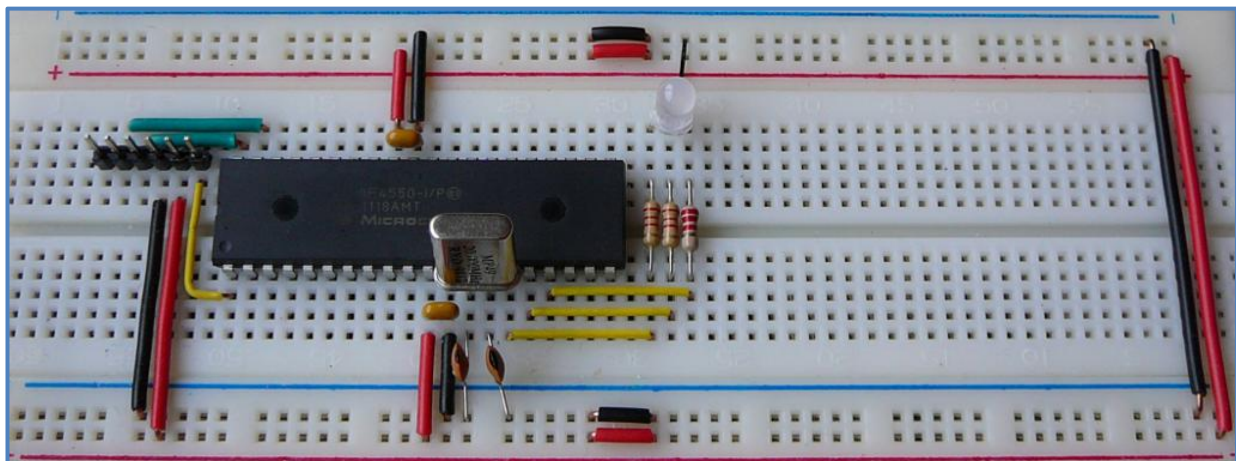
Pour débuter, nous allons connecter au PIC18 l'alimentation V_{DD} et V_{SS}, l'oscillateur et les trois DEL, de même que tous les condensateurs et résistances nécessaires pour faire clignoter les DEL avec les sorties numériques du microcontrôleur. De plus, nous allons également connecter l'ICSP au PIC18, qui n'est rien de plus qu'un ensemble de six broches sur lesquelles se connectera le programmateur PICkit 3 pour transférer notre programme (au format HEX) dans le microcontrôleur.

La page suivante montre le diagramme du montage électronique, de même qu'une photo du résultat sur la planche de montage. Notez que sur le diagramme, deux lignes qui se croisent ne sont pas connectées ensemble, à moins qu'elles soient unies par un point. Vous remarquerez également que j'ai utilisé des résistances pour les DEL, alors qu'elles sont conçues pour fonctionner directement sur le 5v. La raison est que ces DEL ont une forte luminosité, et que les résistances ont pour rôle de diminuer leur intensité.



Liste des composantes		
Description	**Abréviation**	**Valeur**
Oscillateur	Osc	20 MHz
Condensateur	C1 et C2 C3 et C4	100 nF 15 pF
Résistance	R1, R2 et R3	2,2 kΩ

Comme nous l'avons vu précédemment lors de la présentation de la planche de montage, les rangées positives et négatives ne sont pas reliées sur toute la largeur de la planche. Il en est de même pour les rangées inférieures et supérieures. C'est pourquoi une des premières choses à faire est de les relier entre elles. Ensuite, il ne reste plus qu'à procéder au montage. Notez que dans la photo ci-dessous, la broche 1 du PIC18 est connectée à la position G-11 de la planche de montage. De plus, vous remarquerez que j'utilise une DEL RGB, qui contient trois DEL de couleurs différentes en une seule.



Le programme en langage d'assemblage

Voilà! Notre montage est fait. Il ne nous reste plus qu'à démarrer MPLAB, y créer un nouveau projet, puis à programmer le microcontrôleur. Dans ce programme, nous allons utiliser les sorties numériques pour envoyer un signal en alternance à chacune des trois DEL.

Pour créer un délai entre les alternances des DEL, nous utiliserons une boucle qui consommera du temps à ne rien faire d'utile, sinon de perdre du temps en exécutant des instructions de branchement successivement. Ceci nous permettra de faire des expériences afin de bien voir la relation entre le nombre d'instructions par seconde et les configurations des « diviseurs » en lien avec l'oscillateur. Notez que nous verrons plus loin les minuteries (timers) et les interruptions.

La configuration initiale

La toute première ligne de votre code doit être la définition du microcontrôleur, ainsi que la définition du format du fichier HEX qui sera chargé dans le microcontrôleur. Ces informations seront utilisées par MPLAB lors de l'assemblage. Dans notre cas, nous utiliserons bien évidemment le PIC18F4550 et le format du fichier HEX sera « Intel Hex format (INHX32) ». Le code source correspondant se trouve au bas de la page.

La deuxième ligne indispensable est celle de l'inclusion du fichier contenant toutes les définitions permettant un accès facile et convivial aux différents registres, de même qu'à chacun de leurs bits. Dans notre cas, il faut inclure le fichier « P18F4550.INC ». Il est fortement conseillé de jeter un coup d'oeil au contenu de ce fichier.

La troisième et dernière ligne de la configuration initiale est celle qui nous servira à définir les options de configuration du microcontrôleur. Dans notre cas, nous commencerons par définir le mode de l'oscillateur (HS). Nous désactiverons également le « WatchDog Timer » et nous désactiverons le mode de débogage.

Le rôle du « WatchDog Timer » est de s'assurer que le microcontrôleur n'est pas figé, dans une boucle sans fin ou dans tout autre état de blocage. Il permet également de réveiller le microcontrôleur après une mise en veille (instruction « sleep »). Nous verrons plus loin son utilisation en détail.

Voici donc le code source initial. Vous remarquerez que les lignes de commentaires commencent par le caractère « ; ».

```
;*****************************************************************************
;              CONFIGURATION
;*****************************************************************************

       LIST   P=18F4550, F=INHX32          ; Définition de processeur
       #include <P18F4550.INC>             ; Inclusion du fichier des variables

       CONFIG FOSC=HS, WDT=OFF, DEBUG=OFF

; 'CONFIG' précise les paramètres encodés dans le processeur au moment de
; la programmation. Référez-vous au fichier INCLUDE pour les options possibles.
; Dans le cas du fichier <P18F4550.INC>, ces options se retrouvent à partir de
; la ligne 1096.
```

Définition des constantes

Bien que le fichier « P18F4550.INC » offre plusieurs définitions de variables, il peut être utile de définir les nôtres, afin de simplifier le code, ou tout simplement de l'auto commenter.

Par exemple, nous voulons allumer une DEL rouge sur RA0, soit le premier bit du port A. Dans le fichier « P18F4550.INC », on constate que l'adresse du registre PORTA est `H'0F80'`, et que l'assignation du bit correspondant à RA0 est `H'0000'`. Or, pour envoyer un signal sur RA0, il nous faudrait écrire « bsf PORTA, RA0 ».

Il est évident que cette ligne de code est moins explicite que « bsf DEL_ROUGE ». Pour faire notre propre définition, on utilise tout simplement « `#DEFINE   DEL_ROUGE    PORTA, RA0` ».

Note : L'instruction « bsf » met la valeur d'un des bits d'un registre à '1'.

Déclaration des variables

La déclaration des variables est un peu particulière. Non seulement on les déclare, mais on peut également leur assigner une adresse précise en mémoire. Pour ce faire, on utilise « CBLOCK ... ENDC ». Si on ne spécifie pas d'adresse mémoire auquel associer la variable, alors elle sera associée soit à l'adresse `0x000`, soit à la suite de la dernière assignation.

La déclaration de la variable est optionnellement suivie d'un deux- points, puis de la quantité d'espace mémoire à réserver pour cette variable, en terme d'octet.

Pour notre projet, nous utiliserons cinq variables qui nous serviront de compteurs pour les boucles. Nous verrons plus loin pourquoi nous aurons besoin de cinq variables. Notez que dans le code ci-dessous, j'ai assigné les variables à partir de l'adresse `0x00A`, mais j'aurais très bien pu omettre ce paramètre. Vous remarquerez également que j'ai utilisé « LATC » au lieu de « PORTC ». Ceci sera expliqué prochainement.

```
;*****************************************************************************
;              DEFINE
;*****************************************************************************

#DEFINE DEL_BLEUE     LATC, RC0             ; DEL bleue
#DEFINE DEL_VERTE     LATC, RC1             ; DEL verte
#DEFINE DEL_ROUGE     LATC, RC2             ; DEL rouge


;*****************************************************************************
;              DECLARATIONS DE VARIABLES
;*****************************************************************************

       CBLOCK 0x00A
              cptr1 :1
              cptr2 :1
              cptr3 :1
              cptr4 :1
              cptr5 :1
       ENDC
```

Adresses de démarrage et d’interruptions

La première instruction que le microcontrôleur exécutera, tant lors du démarrage qu’à la suite d’une réinitialisation (reset), sera bien évidemment celle située à l’adresse `0x000` de la mémoire du programme. Ensuite il passera naturellement à l’instruction située à l’adresse suivante. Le problème c’est qu’il y a deux adresses importantes qui suivent l’adresse `0x0000`. Ce sont les adresses qui sont appelées lorsque surviennent des interruptions. Il y a l’adresse `0x0008` qui concerne le sous-programme pour les interruptions à haute priorité, puis l’adresse `0x0018` qui concerne la sous-routine pour les interruptions à basse priorité.

Il faudra donc que les instructions de notre programme se situent après ces adresses, mais également que notre programme passe par dessus lors du démarrage pour ne pas exécuter les sous-programmes d’interruption inutilement. Pour ce faire, nous allons mettre à l’adresse `0x000` une instruction de branchement vers le code de notre programme, qui se situera après les adresses d’interruption. Pour forcer l’assignation d’un emplacement mémoire pour une instruction précise, on utilise « ORG », suivi de l’adresse où l’on veut mettre l’instruction qui est située sur la ligne suivante. Dans notre cas, nous ferons un « goto » jusqu’à l’étiquette « init : » où nous procéderons à l’initialisation de notre programme.

```
;*******************************************************************************
;               DEMARRAGE
;*******************************************************************************

        ORG     0x0000                  ; Adresse de départ après un reset
        goto     init
```

Initialisation

Par défaut, les ports d’entrée/sortie du PIC18 sont configurés en entrée. Ceci offre une protection pour le circuit électronique dans le cas d’une erreur de montage ou d’un manque d’attention. Cela dit, chacun des cinq ports d’entrée/sortie du PIC18 dispose de trois registres, de huit bits chacun, pour les échanges de données. Ces registres sont présentés ci-dessous. Les « x » représentent le nom du port désiré, soit A, B, C, D ou E.

- Registre TRISx (définit la direction des données sur le port, c.-à-d. en entrée ou en sortie)
- Registre PORTx (lit le niveau de tension sur les broches du port)
- Registre LATx (registre d’écriture pour la sortie)

Tel que spécifié, le registre TRIS permet de définir individuellement la direction du flux de données sur chacune des broches du port. Pour mettre le flux d’une broche en entrée, on met son bit correspondant à `1`, et pour le mettre en sortie, on le met à `0`.

Prenons par exemple le port D, qui a huit entrées/sorties, soit de RD0 à RD7. Pour mettre les broches RD1, RD3 et RD4 en entrée, puis mettre les broches RD0, RD2, RD5, RD6 et RD7 en sortie, il nous faut mettre dans le registre TRISD la valeur binaire `B’00011010’` avec « `SETF TRISD, B’00011010’` ». Vous remarquerez dans le tableau de la page suivante que les positions des bits d’un registre sont lues de droite à gauche.

Registre TRISD (sur 8 bits)							
RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0
0	**0**	**0**	**1**	**1**	**0**	**1**	**0**

Étant donné que les registres TRIS sont initialisés par défaut à `B'11111111'`, nous aurions simplement pu utiliser l'instruction « bcf » pour mettre à `0` les bits désirés. Par exemple, pour mettre RD2 à `0`, il aurait suffi d'écrire « `bcf TRISD, RD2` ».

Le registre PORT quant à lui permet une lecture de l'état du port, mais également une écriture. Cependant, il n'est pas conseillé d'utiliser PORT pour écrire en sortie. Pour la sortie, il est recommandé d'utiliser le registre LAT qui lui, se verrouille une fois écrit et prévient ainsi toute interférence des autres broches sur le même port.

```
;*****************************************************************************
;              INITIALISATION
;*****************************************************************************

init:
       ; S'assurer que le contenu du port soit à zéro
       clrf    PORTC

       ; Mettre les broches en sortie
       bcf     TRISC, RC0
       bcf     TRISC, RC1
       bcf     TRISC, RC2

       ; S'assurer de démarrer avec les DEL éteints (bits à 0)
       bcf     DEL_VERTE
       bcf     DEL_BLEUE
       bcf     DEL_ROUGE

       goto    main
```

Le coeur du programme

Le coeur du programme se divise en deux parties. D'abord un sous-programme de temporisation, qui simulera une pause de 500 millisecondes, puis le programme principal qui fera alterner les DEL.

Sous-programme de temporisation

Avant de coder l'allumage et l'extinction des DEL, nous allons d'abord nous attaquer au sous-programme de temporisation (délai par boucle). Nous allons donc faire faire assez d'instructions de branchement par le processeur du PIC, pour simuler une pause de 500 millisecondes.

Comme nous utilisons un oscillateur de 20 MHz, et que le processeur nécessite quatre coups d'horloge pour exécuter un cycle d'instruction, nous pouvons déduire que notre PIC exécutera 5 000 000 instructions par

secondes. Question de vous rafraichir la mémoire, 1 Hz signifie 1 oscillation par seconde. Donc 1 MHz est égal à 1 000 000 oscillations par seconde, d'où 20 MHz ÷ 4 = 5 000 000 instructions par secondes.

Nous savons donc que nous avons besoin d' exécuter 2 500 000 cycles d'instructions pour faire une pause de 500 millisecondes. Étant donné que notre variable sera contenue sur un octet, le nombre de fois que nous pourrons itérer sur une boucle sera limité à 256, soit $2^{8\,(bits)}$. Nous devrons donc faire des boucles imbriquées pour obtenir 2 500 000 cycles d'instruction. C'est la raison pour laquelle nous avons déclaré précédemment cinq variables pour les compteurs.

Pour exécuter nos boucles, nous utiliserons l'instruction « decfsz » qui décrémente une variable, suivi de l'instruction « goto » qui nous renverra au début de la boucle. Lorsque la variable atteint zéro, « decfsz » passera par-dessus l'instruction « goto ». La signification de « decfsz » est 'DECrement File, Skip if Zero', qui se traduit par 'DÉCrémenter un Fichier (ou variable), Sauter si Zéro'.

L'instruction « decfsz » nécessite un cycle d'instruction lorsque le résultat est différent de zéro, puis deux lorsque le résultat est égal à zéro. L'instruction « goto » quant à elle nécessite toujours deux cycles d'instruction. Notre code sera donc le suivant.

```
;******************************************************************************
;             SOUS-ROUTINES
;******************************************************************************

pauseParBoucle:                           ; nbr itérations * nbr cycles
                                          ; -------------------------
       movlw  D'12'                       ; 1x 1 cycle.
       movwf  cptr3                       ; 1x 1 cycle.
b3:
       clrf   cptr2                       ; 12x 1 cycle.
b2:
       clrf   cptr1                       ; 12x256x 1 cycle.
b1:
       decfsz cptr1, f                    ; 12x256x(255x 1 cycle, 1x 2 cycles).
       goto   b1                          ; 12x256x(255x 2 cycles, 1x 0 cycle).

       decfsz cptr2, f                    ; 12x(255x 1 cycle, 1x 2 cycles).
       goto   b2                          ; 12x(255x 2 cycles, 1x 0 cycle).

       decfsz cptr3, f                    ; 12x 1 cycle, 1x 2 cycles.
       goto   b3                          ; 12x 2 cycles, 1x 0 cycle.

       return                             ; 1x 2 cycle.
```

En ne tenant compte que des instructions « decfsz » et « goto », la boucle 'b3' exécutera 38 cycles d'instruction (12 + 2 + 12x2). La boucle 'b2' exécutera 9 204 cycles d'instruction (12x[255 + 2 + 255x2]). La boucle 'b1' exécutera 2 356 224 cycles d'instruction (12x256x[255 + 2 + 255x2]). Ce qui nous donne un total de 2 365 466 cycles d'instruction. À cela s'ajoutent les opérations sur les variables, soit 3 086 (1 + 1 + 12 + 12x256), ainsi que 2 cycles d'instruction pour le « return », pour un grand total de 2 368 554 cycles d'instruction.

Il nous manque donc 131 446 cycles d'instruction pour atteindre 500 millisecondes. Nous allons donc ajouter une mini boucle ('miniB1') à l'intérieur de la boucle 'b2' afin de combler une partie de ce manque. Cette boucle s'exécutera 12 fois, et ajoutera 116 736 cycles d'instruction (12x256x[12 + 2 + 12x2]), auxquels s'ajouteront les 12 288 cycles d'instruction (12x256x4) pour l'affectation de la variable « cptr4 » et les deux instructions « nop ». Nous ajouterons donc un total de 129 024 cycles d'instruction.

Il ne nous manque maintenant plus que 2 422 cycles d'instruction. Un dernier ajustement de précision se fera donc à l'intérieur de la boucle 'b3' en y ajoutant une autre mini boucle ('miniB2') qui s'exécutera 66 fois. Elle ajoutera 2 400 cycles d'instruction (12x[66 + 2 + 66x2]) auxquels s'ajouteront les 24 cycles d'instruction (12 + 12) pour l'affectation de la variable « cptr5 » pour un ajout total de 2 424 cycles d'instruction.

Le nouveau code ci-dessous nous donnera donc un grand total de 2 500 002 cycles d'instruction. Nous ne pousserons pas plus loin la précision.

```
;******************************************************************************
;              SOUS-ROUTINES
;******************************************************************************

pauseParBoucle:                           ; nbr itérations * nbr cycles
                                          ; -------------------------
       movlw  D'12'                       ; 1x 1 cycle.
       movwf  cptr3                       ; 1x 1 cycle.
b3:
       clrf   cptr2                       ; 12x 1 cycle.
b2:
       clrf   cptr1                       ; 12x256x 1 cycle.
b1:
       decfsz cptr1, f                    ; 12x256x(255x 1 cycle, 1x 2 cycles).
       goto   b1                          ; 12x256x(255x 2 cycles, 1x 0 cycle).
       ; FIN BOUCLE 'b1'

       movlw  D'12'                       ; 12x256x 1 cycle.
       movwf  cptr4                       ; 12x256x 1 cycle.
       nop                                ; 12x256x 1 cycle.
       nop                                ; 12x256x 1 cycle.
miniB1:
       decfsz cptr4, f                    ; 12x256x(12x 1 cycle, 1x 2 cycles).
       goto   miniB                       ; 12x256x(12x 2 cycles, 1x 0 cycle).
       decfsz cptr2, f                    ; 12x(255x 1 cycle, 1x 2 cycles).
       goto   b2                          ; 12x(255x 2 cycles, 1x 0 cycle).
       ; FIN BOUCLE 'b2'

       movlw  D'66'                       ; 12x 1 cycle.
       movwf  cptr5                       ; 12x 1 cycle.
miniB2:
       decfsz cptr4, f                    ; 12x (66x 1 cycle, 1x 2 cycles).
       goto   miniB                       ; 12x (66x 2 cycles, 1x 0 cycle).
       decfsz cptr3, f                    ; 12x 1 cycle, 1x 2 cycles.
       goto   b3                          ; 12x 2 cycles, 1x 0 cycle.
       ; FIN BOUCLE 'b3'

       return                             ; 1x 2 cycle.
```

Programme principale

Voilà! Le plus difficile est fait. Il ne nous reste plus qu'à allumer une DEL, appeler la sous-routine de pause, éteindre la DEL, puis faire de même pour les autres DEL. Pour allumer une DEL, il suffit d'émettre un signal de sortie en mettant la valeur '1' dans le bit correspondant à la broche au quelle est connecté la DEL. Notez que le PIC18 n'émet par de signal de sortir analogique. Le port émet soit 0v, soit 5v. Ce qui correspond respectivement aux valeurs '0' et '1' dans le registre LATx. Les sorties numériques sont aussi simples que ça.

Vous pouvez constater dans le code ci-dessous que j'ai commencé par faire une pause d'une seconde. Vous remarquerez également que j'ai ajouté l'instruction « sleep » qui met en veille le microcontrôleur à la fin du programme. Si vous n'utilisez pas cette instruction, le microcontrôleur reprendra le programme depuis le début dès qu'il aura atteint la fin du programme (END).

```
;*******************************************************************************
;              PROGRAMME PRINCIPAL
;*******************************************************************************

main:
       call    pauseParBoucle
       call    pauseParBoucle

       bsf     DEL_R
       call    pauseParBoucle
       bcf     DEL_R

       bsf     DEL_B
       call    pauseParBoucle
       bcf     DEL_B

       bsf     DEL_V
       call    pauseParBoucle
       bcf     DEL_V

       sleep
       END
```

Chargement du programme dans le microcontrôleur

Pour charger le programme dans le microcontrôleur, vous devez d'abord l'assembler avec la touche F10. Si vous préférez passer par le menu, il suffit de cliquer sur « Project → Make ».

Une fois le programme assemblé, il faut connecter le câble USB au PICkit 3, puis le PICkit au PIC18. En connectant le PICkit sur l'ICSP, faites attention de bien connecter la broche numéro 1 du PICkit (celle avec le petit triangle blanc) avec la broche numéro 1 du PIC18 (celle avec le petit point). Il faut ensuite choisir le programmateur via le menu, en cliquant sur « Programmer → Select Programmer → PICkit 3 ».

Note : En anglais, aucune distinction n'est faite entre un programmeur (spécialiste chargé du développement de programmes informatiques) et un programmateur (dispositif destiné à commander l'exécution des différentes opérations à effectuer).

La connexion entre MPLAB et le PIC se lancera automatiquement. Naturellement, vous devriez avoir un message d'erreur, car le microcontrôleur doit être alimenté en électricité pour que le PICkit puisse s'y connecter. Nous allons donc aller dans le menu « Programmer → Settings... ». Dans l'onglet « Power », cochez l'option « Power target circuit from PICkit 3 ». Assurez-vous que le voltage est à 5v. La connexion se relancera automatiquement.

Il ne vous reste plus qu'à charger le programme dans le microcontrôleur via le menu « Programmer → Program ». Notez qu'il se peut qu'il y ait un petit délai avant que le microcontrôleur se mette en marche. Les trois DEL devraient s'allumer à tour de rôle, puis le microcontrôleur se mettra en veille. Pour relancer le programme du microcontrôleur, nous concevrons un peu plus tard un bouton de réinitialisation (reset). Pour l'instant, vous pouvez retirer le PICkit de l'ICSP puis le remettre aussitôt, sans débrancher son câble USB.

Tests sur le sous-programme et les « diviseurs »

Maintenant que nous avons un programme fonctionnel, nous pouvons tester notre sous-programme de pause. Allumez une DEL en mettant dix appels au sous-programme entre l'allumage et l'extinction. Compter le temps que la DEL reste allumée. Vous devriez avoir 5 secondes (10 x 500 millisecondes = 5 000 millisecondes = 5 secondes).

« Oscillator Postscaler »

Nous allons maintenant réduire la vitesse de notre horloge en divisant par deux la fréquence de l'oscillateur. Pour ce faire, remplacez la ligne de configuration par :

```
CONFIG FOSC=HS, CPUDIV=OSC2_PLL3, WDT=OFF, DEBUG=OFF
```

Votre DEL devrait maintenant rester allumée environ 10 secondes. Les valeurs possibles pour « CPUDIV » seront vues à la fin de la section suivante.

« PLL Prescaler » et « PLL Postscaler »

Nous allons maintenant monter la fréquence de l'horloge à 48 MHz (limite maximale), en utilisant le PLL (Phase Locked Loop). Le rôle du PLL est de multiplier par 24 chaque phase d'un signal de 4 MHz, pour émettre un signal de 96 MHz à sa sortie. Ce signal sera ensuite divisé par deux pour rester dans les limites du 48 MHz.

L'activation du PLL se fait en remplaçant le type de l'oscillateur par « HSPLL_HS ». Cela dit, nous avons un oscillateur de 20 MHz, mais le PLL demande un signal strictement à 4 MHz. Nous avons vu ci-dessus que l'on pouvait diviser la fréquence de notre oscillateur. C'est ce que nous allons faire ici, mais en utilisant le « PLL prescaler » qui lui, divisera la fréquence uniquement pour le PLL, et non pour le reste du microcontrôleur.

Donc, pour avoir une fréquence de 48 MHz, nous devrons dans un premier temps, diviser la fréquence de notre oscillateur par 5 avec le « PLL Prescaler » pour avoir une fréquence de 4 MHz à l'entrée du PLL, puis diviser la fréquence à la sortie du PLL par 2 avec le « PLL Postscaler » pour obtenir 48 MHz. Pour ce faire, il faut remplacez la ligne de configuration par :

```
CONFIG FOSC=HSPLL_HS, PLLDIV=5, CPUDIV=OSC1_PLL2, WDT=OFF, DEBUG=OFF
```

Maintenant, votre DEL devrait rester allumée un peu plus de 2 secondes (approximativement 208 millisecondes). Si vous le désirez, vous pouvez essayer d'autres valeurs, mais faites attention avec le « PLL Prescaler ». Sa valeur est strictement établie en fonction de l'oscillateur et ne peut être utilisée autrement que pour fournir une fréquence de 4 MHz au PLL. Voici les tableaux des valeurs possibles.

Options pour les deux « Postscaler »		
Options	**Oscillateur**	**PLL**
CPUDIV = OSC1_PLL2	Pas de mise à l'échelle	96 MHz : Division par 2
CPUDIV = OSC2_PLL3	Division par 2	96 MHz : Division par 3
CPUDIV = OSC3_PLL4	Division par 3	96 MHz : Division par 4
CPUDIV = OSC4_PLL6	Division par 4	96 MHz : Division par 6

Options pour le « PLL Prescaler »	
Options	
PLLDIV = 1	Pas de mise à l'échelle [pour un oscillateur de 4 MHz en entrée]
PLLDIV = 2	Division par 2 [pour un oscillateur de 8 MHz en entrée]
PLLDIV = 3	Division par 3 [pour un oscillateur de 12 MHz en entrée]
PLLDIV = 4	Division par 4 [pour un oscillateur de 16 MHz en entrée]
PLLDIV = 5	Division par 5 [pour un oscillateur de 20 MHz en entrée]
PLLDIV = 6	Division par 6 [pour un oscillateur de 24 MHz en entrée]
PLLDIV = 10	Division par 10 [pour un oscillateur de 40 MHz en entrée]
PLLDIV = 12	Division par 12 [pour un oscillateur de 48 MHz en entrée]

Avant de poursuivre, sachez que je travaillerai pour la suite du document à une fréquence de 20 MHz, étant donné que c'est pour cette fréquence que la sous-routine de pause a été conçue. Je vous redonne donc la ligne de configuration correspondante :

```
CONFIG FOSC=HS, WDT=OFF, DEBUG=OFF
```

Les macros

Comme nous pouvons le voir dans notre code, nous effectuons trois fois la même opération, sur trois DEL différentes. Nous allons donc créer une macro qui allumera, puis éteindra une DEL reçue en paramètre, après avoir effectué une pause bien évidemment. Les macros sont généralement situées entre les sections « DEFINE » et « DECLARATIONS DE VARIABLES ».

Il est bon de savoir que les macros ne sont pas des fonctions à proprement dire. Elles servent simplement à simplifier la lecture du code pour le programmeur, et ne diminuent en rien la taille du fichier HEX qui sera chargé dans le microcontrôleur. En effet, lors de l'assemblage, la ligne de code qui fait appel à la macro sera remplacée par le code même de la macro, ainsi que toutes les autres lignes qui font appel à cette même macro.

Pour construire une macro, on tape d'abord son nom, suivi du mot clé « macro », puis optionnellement suivi d'un ou plusieurs paramètres, séparés par une virgule. La macro se termine toujours par une ligne avec le mot clé « endm ». Entre ces deux lignes, on met le code désiré. Voici ce que nous voulons faire pour notre code :

```
;*****************************************************************************
;              MACRO
;*****************************************************************************
allumerDEL macro paramDEL
       bsf    paramDEL
       call   pauseParBoucle
       bcf    paramDEL
       endm

;*****************************************************************************
;              PROGRAMME PRINCIPAL
;*****************************************************************************
main:
       call   pauseParBoucle
       call   pauseParBoucle

       allumerDEL    DEL_ROUGE
       allumerDEL    DEL_BLEUE
       allumerDEL    DEL_VERTE

       sleep
       END
```

Malheureusement, ce code ne fonctionnera pas. Le problème c'est que nos DEL, tel que « DEL_BLEUE » sont constituées elles-mêmes de deux paramètres. C'est-à-dire « `LATC, RC0` ». Alors, pour que notre macro fonctionne, il faudra écrire les paramètres comme ceci :

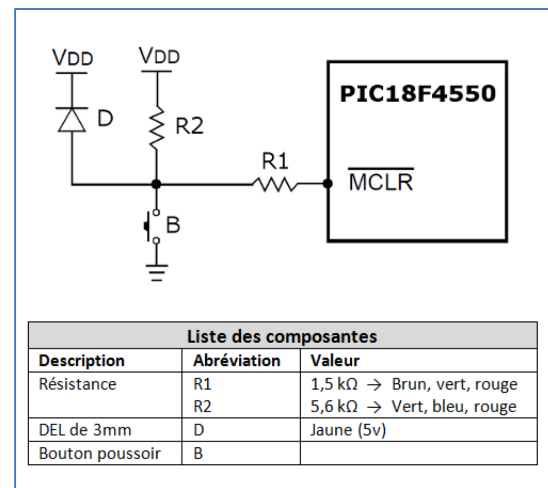
```
;*****************************************************************************
;              MACRO
;*****************************************************************************
allumerDEL macro port, bit
       bsf    port, bit
       call   pauseParBoucle
       bcf    port, bit
       endm

;*****************************************************************************
;              PROGRAMME PRINCIPAL
;*****************************************************************************
main:
       call   pauseParBoucle
       call   pauseParBoucle

       allumerDEL    DEL_ROUGE
       allumerDEL    DEL_BLEUE
       allumerDEL    DEL_VERTE

       sleep
       END
```

La réinitialisation (reset)

Pour effectuer une réinitialisation en cours de fonctionnement, il nous suffit de provoquer une montée de la tension sur la broche 1 du PIC, soit le MCLR (Master CLeaR), mais sans toute fois atteindre les 5 volts. Le microcontrôleur se mettra alors en état de réinitialisation et demeurera dans cet état tant que la tension ne redescendra pas à zéro. Une fois revenu à zéro, le microcontrôleur reprendra son exécution du programme au début de celui-ci, soit à l'adresse `0000h` de la mémoire de programme.

Montage du prototype

Pour faire la réinitialisation, nous utiliserons un montage appelé pont diviseur de tension, afin d'apporter une tension d'environ 4 volts au MCLR. Un pont diviseur de tension permettra de laisser passer une partie du courant vers le PIC18, et une partie du courant vers la borne positive V_{DD}.



Liste des composantes		
Description	**Abréviation**	**Valeur**
Résistance	R1	1,5 kΩ → Brun, vert, rouge
	R2	5,6 kΩ → Vert, bleu, rouge
DEL de 3mm	D	Jaune (5v)
Bouton poussoir	B	

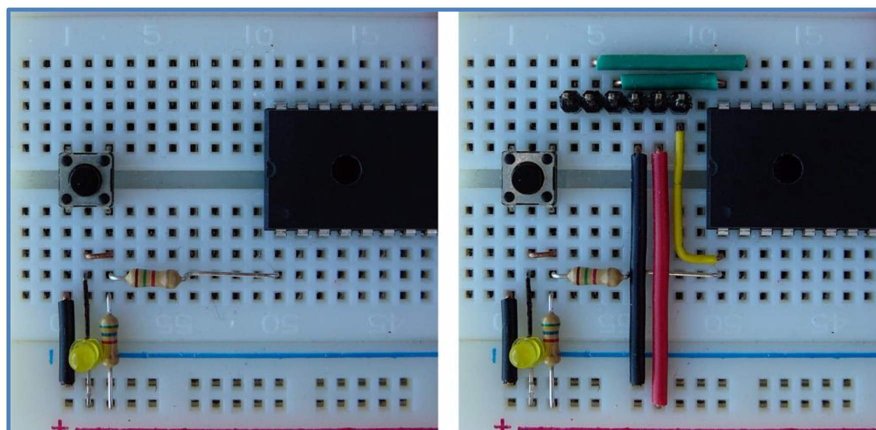
Le diagramme de montage pour le diviseur de tension est montré ci-contre. On remarque que la valeur de la résistance de 'R2' est plus grande que celle de 'R1'. Ceci a pour effet de laisser passer plus de courant au travers de 'R1' et moins au travers de 'R2'. Le MCLR recevra donc moins de 5 volts, mais plus de 2,5 volts. Si la valeur de 'R2' avait été plus petite que celle de 'R1', le MCLR aurait reçu moins de 2,5 volts. Si les deux résistances avaient été égales, le résultat aurait été précisément 2,5 volts.

La formule pour calculer la tension de sortie (V_{out}), à partir de la tension d'entrée (V_{in}) d'un pont diviseur est la suivante :

$V_{out} = V_{in} \cdot \frac{R2}{R1+R2}$

Donc en utilisant les valeurs données dans le diagramme ci-dessus, nous obtenons :

$V_{out} = 5 \cdot \frac{5600}{1500+5600}$ = 3,94 volts à l'entrée du MCLR, ce qui est parfait pour notre montage. Ne reste plus qu'à monter le prototype, puis à l'essayer. Voyez ci-dessous le prototype qui correspond précisément au diagramme (à gauche), de même que le montage qui inclut les autres composantes déjà montées sur le prototype (à droite).



Valeurs après réinitialisation

Lorsqu'on réinitialise le microcontrôleur, certaines valeurs des registres de configuration sont remises à leur état initial, certaines autres conservent les valeurs déterminées par l'usager. Voici le tableau de quelques registres de configuration importants, ainsi que les valeurs prises après une réinitialisation, selon le type de réinitialisation. Voyez la légende ci-contre pour les détails.

Légende	
0	Prend la valeur zéro
1	Prend la valeur un
u	Valeur inchangée
x	Valeur inconnue
-	Bit non implémenté

Registre	Réinitialisation POR (Power-On Reset)	Réinitialisation MCLR Réinitialisation WDT Instruction « RESET » Débordement de la pile	Réveil par le WDT ou par une interruption
TRIS**A**	-111 1111	-111 1111	-uuu uuuu
TRIS**B**	1111 1111	1111 1111	uuuu uuuu
TRIS**C**	11-- -111	11-- -111	uu-- -uuu
TRIS**D**	1111 1111	1111 1111	uuuu uuuu
TRIS**E**	---- -111	---- -111	---- -uuu
PORT**A**	-x0x 0000	-u0u 0000	-uuu uuuu
PORT**B**	xxxx xxxx	uuuu uuuu	uuuu uuuu
PORT**C**	xxxx -xxx	uuuu -uuu	uuuu -uuu
PORT**D**	xxxx xxxx	uuuu uuuu	uuuu uuuu
PORT**E**	0--- x000	0--- x000	u--- uuuu
LAT**A**	-xxx xxxx	-uuu uuuu	-uuu uuuu
LAT**B**	xxxx xxxx	uuuu uuuu	uuuu uuuu
LAT**C**	xx-- -xxx	uu-- -uuu	uu-- -uuu
LAT**D**	xxxx xxxx	uuuu uuuu	uuuu uuuu
LAT**E**	---- -uuu	---- -uuu	---- -uuu

Les minuteurs (timers)

Précédemment, nous avons créé une pause avec une boucle qui consommait du temps à ne rien faire d'utile. Ça fonctionne bien, mais il y a beaucoup mieux.

Le « WatchDog Timer »

Le rôle du WDT (WatchDog Timer) est de s'assurer que le microcontrôleur n'est pas figé, dans une boucle sans fin ou dans tout autre état de blocage. Si un tel cas se produit, le WDT réinitialisera le microcontrôleur. Une autre utilité du WDT est de réveiller le PIC suite à un « sleep ». Dans un tel cas, l'exécution du programme reprendra après le « sleep ». Le WDT n'utilise pas l'horloge générée par l'oscillateur externe, mais l'horloge générée par un des deux oscillateurs internes du microcontrôleur, soit l'INTRC de 31 kHz.

Dans un premier temps, on définit, avec la ligne de configuration, le délai que le WDT doit attendre avant de réinitialiser ou réveiller le microcontrôleur. Pour ce faire, nous établirons un ratio avec la période nominale du WDT, qui est de 4 millisecondes. La configuration s'effectue sur le WDTPS (WDT PostScaler). Ce ratio peut prendre comme valeur une puissance de 2, jusqu'à concurrence de 32 768. Par exemple, pour définir un délai d'environ deux secondes avant la réinitialisation par le WDT, nous ajouterons « WDTPS=512 » à la ligne de configuration. Ce qui donnera précisément 2 048 millisecondes (4 x 512).

Dans un deuxième temps, il nous faut établir à partir de quel moment on commence à compter les 2 048 millisecondes. À vrai dire, cette minuterie est toujours en plein décompte, dès le début du programme. Donc si nous ne faisons rien, notre programme se réinitialisera toujours après 2 048 millisecondes, et ce, même si tout va bien. Nous devons donc nous assurer de remettre régulièrement la minuterie du WDT à zéro, avec l'instruction « CLRWDT ».

Il est donc important ici d'être en mesure de connaître le nombre d'instruction que notre PIC exécute par secondes. En effet, si nous utilisons trop souvent l'instruction « CLRWDT », nous consommons des ressources inutilement. Si nous ne l'utilisons pas assez souvent, nous aurons des réinitialisations non désirées. Cependant, une chose est certaine, il faut réinitialiser cette minuterie avant et après toute portion de code jugée critique.

Il y a une autre instruction qui remet cette minuterie à zéro. Il s'agit de l'instruction « sleep », tel que nous l'utilisons dans notre code assembleur. Cependant, nous n'avons pas encore activé le WDT. C'est pourquoi après s'être exécuté, notre programme ne redémarre pas. Nous allons donc activer le WDT. Ainsi, le « sleep » s'exécutera, puis le WDT réveillera le microcontrôleur après 2 048 millisecondes. Notre programme ne s'arrêtera plus, tant qu'il sera alimenté. Cela dit, n'oubliez pas d'ajouter l'instruction « CLRWDT » dans votre macro, et dans tout autre endroit jugé nécessaire, sinon votre programme sera réinitialisé avant même d'atteindre l'instruction « sleep ». Pour configurer et activer le WDT, il faut remplacer la ligne de configuration par :

```
CONFIG FOSC=HS, WDT=ON, WDTPS=512, DEBUG=OFF, PWRT=OFF
```

Le programme fera maintenant une pause de 2 048 millisecondes à la fin de son exécution. Par contre, n'oubliez par que l'exécution du programme commence par une pause d'une seconde. Donc trois secondes devraient s'écouler entre l'extinction de la dernière DEL, et l'allumage de la première après la réinitialisation.

Pour la suite du tutoriel, désactivez le WDT (WDT=OFF).

Les entrées numériques

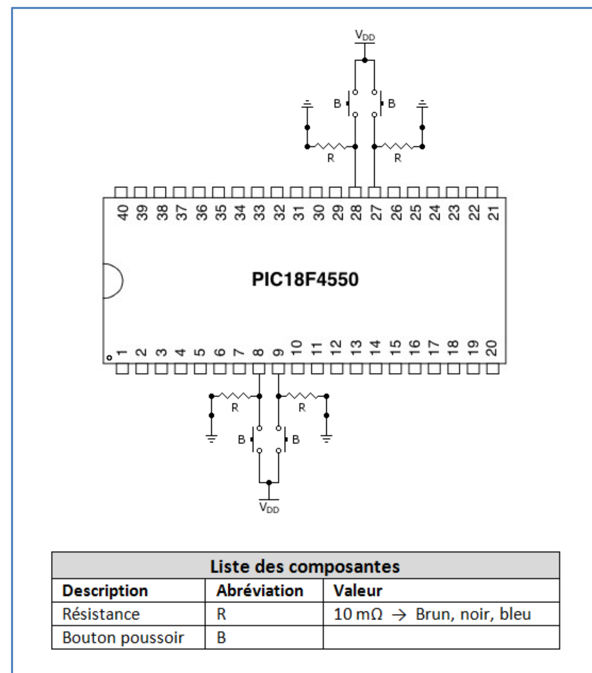
Les entrées numériques ne sont pas plus compliquées que les sorties numériques, à la différence que pour les sorties nous écrivions dans les registres LATx, alors que pour les entrées nous allons lire les registres PORTx. Puisqu'il s'agit d'entrées numériques, les valeurs possibles lors de la lecture seront soit '0', soit '1'.

Vérification du signal par boucle

Pour mettre en pratique les entrées numériques, nous allons ajouter quatre boutons. Deux seront sur le port 'E' et deux sur le port 'D'. Ceux sur le port 'E' occuperont les broches 8 et 9, soit RE0 et RE1. Ceux sur le port 'D' occuperont les broches 27 et 28, soit RD4 et RD5. Nous ferons ensuite une boucle sans fin qui vérifiera constamment s'il y a des signaux en entrée.

Montage

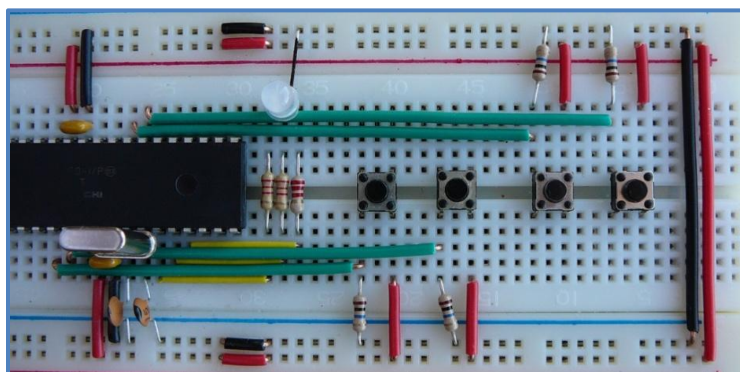
Avant de commencer à coder en assembleur, il nous faut d'abord apporter quelque ajout au prototype sur la planche de montage. Le diagramme ci-contre montre les ajouts à apporter. À noter que les montages précédents n'y apparaissent pas.



Liste des composantes		
Description	**Abréviation**	**Valeur**
Résistance	R	10 mΩ → Brun, noir, bleu
Bouton poussoir	B	

On remarque que les boutons effectuent une connexion entre les broches du microcontrôleur et la borne positive (V_{DD}). Par contre, on y voit également que pour chaque broche, une résistance la relie à la borne négative. Ceci a pour effet d'empêcher les signaux parasites de provoquer un faux signal d'entrée en les redirigeant vers la borne négative.

On peut voir le résultat du prototype dans l'image ci-dessous. À noter que les boutons utilisés ici sont doubles. C'est-à-dire que les deux pattes de gauche sont reliées entre elles, et pareillement pour les pattes de droite. Lorsqu'on appuie sur le bouton, le contact est fait entre les pattes de gauche et celles de droite.



Code assembleur

Il ne nous reste plus qu'à taper quelques lignes de code pour faire fonctionner tout ça. Le programme que nous allons faire fera en sorte que les deux boutons de gauche allumeront la DEL bleue, le troisième bouton allumera la DEL verte, puis le dernier bouton allumera la DEL rouge.

Avant de commencer, il nous faut, dans un premier temps, retirer l'allumage de cinq secondes de la DEL bleu, de même que l'instruction « `sleep` », pour garder dans la partie « `main` » uniquement ce qui suit :

```
;******************************************************************************
;              PROGRAMME PRINCIPAL
;******************************************************************************

main:
       call   pauseParBoucle
       call   pauseParBoucle

       allumerDEL    DEL_ROUGE
       allumerDEL    DEL_BLEUE
       allumerDEL    DEL_VERTE

       END
```

Dans un deuxième temps, nous allons définir nos boutons dans la partie « `DEFINE` » du code source. Bien que nous les utiliserons, du moins pour l'instant, que pour faire allumer nos DEL, nous les nommerons de façon à les réutiliser ultérieurement comme étant les touches d'un clavier binaire, soit les touches [ZÉRO], [UN], [ENTRÉE] puis [EFFACER]. Cela va comme suit :

```
;******************************************************************************
;              DEFINE
;******************************************************************************

#DEFINE BOUTON_0      PORTE, RE0           ; Correspond à la touche [ZÉRO]
#DEFINE BOUTON_1      PORTE, RE1           ; Correspond à la touche [UN]
#DEFINE BOUTON_ENTR   PORTD, RD5           ; Correspond à la touche [ENTRÉE]
#DEFINE BOUTON_EFFA   PORTD, RD4           ; Correspond à la touche [EFFACER]
```

Par défaut, les ports « `AN0` » à « `AN7` » sont toujours définis comme étant analogiques. Or, les ports « `AN5` » et « `AN6` » correspondent respectivement aux broches 8 et 9 du PIC18F4550. Le problème, c'est que ces deux broches correspondent également aux ports « `RE0` » et « `RE1` », soit deux de nos boutons. Nous mettrons donc, du moins pour l'instant, tous les ports en entrée numérique. Pour ce faire, il nous faut mettre à '1' les quatre bits les moins significatifs (les plus à droite) du registre « `ADCON1` », tel que montré à la page suivante. Nous verrons les autres valeurs possibles pour ce registre lorsque nous verrons les entrées analogiques.

```
;******************************************************************************
;              INITIALISATION
;******************************************************************************

       movlw   b'00001111'
       movwf   ADCON1
```

Une façon simple de vérifier si nous avons un signal d'entrée est de vérifier en continu chacune des broches concernées à l'intérieur d'une boucle infinie, et d'agir selon les valeurs lues sur les ports d'entrée. C'est ce qui est fait dans le code suivant. C'est tout! Il ne reste plus qu'à assembler et charger tout ça dans le microcontrôleur.

```
;******************************************************************************
;              PROGRAMME PRINCIPAL
;******************************************************************************

main:
       call   pauseParBoucle
       call   pauseParBoucle

       allumerDEL     DEL_ROUGE
       allumerDEL     DEL_BLEUE
       allumerDEL     DEL_VERTE

boucleBoutons:
       ; ---------------
       ; Tester BOUTON 0
       btfsc   BOUTON_0             ; Si BOUTON_0 == 0, alors sauter ligne suivante.
       goto    allumerDEL_BLEUE

       ; ---------------
       ; Tester BOUTON 1
       btfss   BOUTON_1             ; Si BOUTON_1 == 1, alors sauter ligne suivante.
       goto    eteindreDEL_BLEUE

allumerDEL_BLEUE:
       bsf     DEL_BLEUE
       goto    boucleBoutons

eteindreDEL_BLEUE:
       bcf     DEL_BLEUE

       ; --------------------
       ; Tester BOUTON ENTRÉE
       btfss   BOUTON_ENTR          ; Si BOUTON_ENTR == 1, alors sauter ligne suivante.
       goto    eteindreDEL_VERTE

allumerDEL_VERTE:
       bsf     DEL_VERTE
       goto    boucleBoutons

eteindreDEL_ VERTE:
       bcf     DEL_VERTE
```

```
        ; --------------------
        ; Tester BOUTON_EFFACER
        btfss   BOUTON_EFFA          ; Si BOUTON_EFFA == 1, alors sauter ligne suivante.
        goto    eteindreDEL_ROUGE

allumerDEL_ROUGE:
        bsf     DEL_ROUGE
        goto    boucleBoutons

eteindreDEL_ROUGE:
        bcf     DEL_ROUGE
        goto    boucleBoutons

        END
```

Vérification du signal par interruption

Une interruption, son nom le dit, vient interrompe l'exécution normale d'un programme. Lorsqu'elle survient, le programme est mis en pause, puis le code situé à l'adresse du vecteur d'interruption est exécuté. Après quoi, le programme reprend son cours normal.

Une interruption n'arrête pas le programme en plein milieu d'une instruction en cours, mais attend qu'elle ait terminé de s'exécuter. Ensuite, on empile sur la pile d'adresse la valeur du compteur ordinal, auquel on ajoute 2 afin de redémarrer à l'instruction suivante, puis on branche sur le vecteur d'interruption. À ce moment, c'est à l'usager de s'assurer de sauvegarder les registres d'état, de même que toute autre donnée utile, dès le début de sa routine d'interruption. Viendra ensuite le code à exécuter pour l'interruption, puis l'usager devra remettre les registres d'état dans leur état initiale avant de mettre fin à l'interruption avec l'instruction « `retfie` ». Les étapes à exécuter dans une routine d'interruption sont généralement les suivantes :

1. Sauvegarder l'environnement.
2. Détecter la source de l'interruption (l'événement).
3. Traiter l'événement (c.-à-d. exécuter le code correspondant).
4. Restaurer l'environnement.
5. Mettre fin à l'interruption (`retfie`).

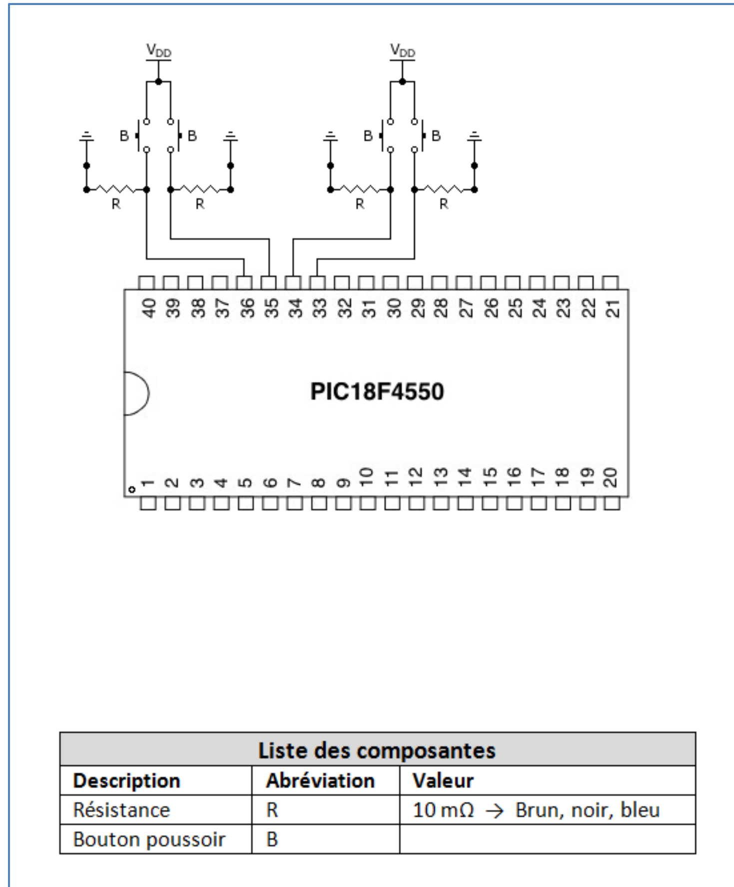
Dans notre cas, nous ne nous préoccuperons pas de l'environnement, car ce ne sera pas nécessaire.

Montage

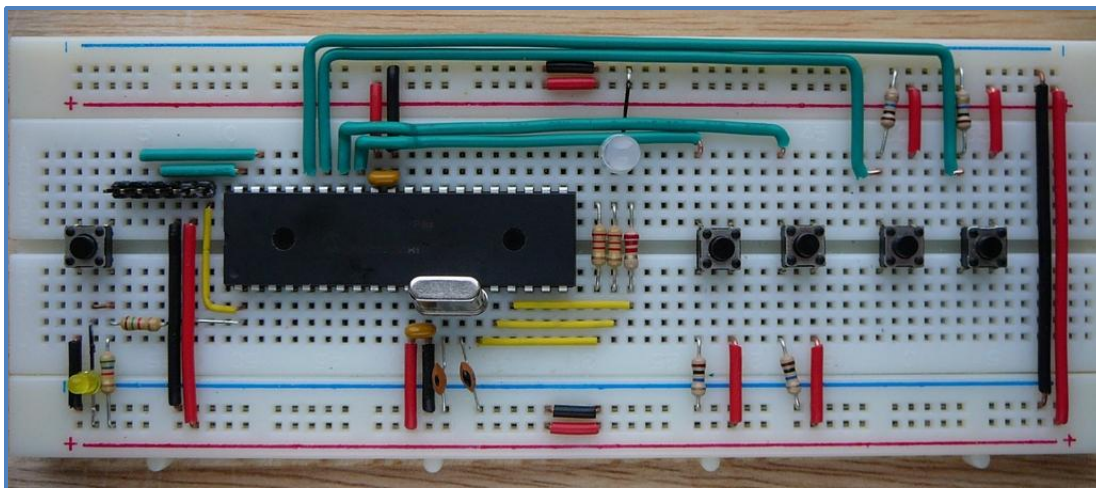
Pour remplacer la boucle sans fin par des interruptions, il nous faut d'abord apporter une modification majeure à notre prototype. Le problème est que les interruptions externes doivent être faites uniquement sur les broches numéro 33 à 40 du PIC18F4550.

Il y a quatre sources d'interruptions externes, soit INT0, INT1 et INT2 (respectivement les broches 33, 34 et 35), puis il y a toutes les entrées restantes du port 'B' (broches 36 à 40 : RB3 à RB7). Les interruptions sur le port 'B' ne peuvent être faites individuellement sur chacune des broches, étant donné qu'il n'existe qu'un seul bit d'état d'interruption pour l'ensemble des broches RBx. INT0, INT1 et INT2 quant à eux ont chacun leur propre bit d'état pour les interruptions.

Pout le montage de notre prototype, nous connecterons donc le bouton [ZÉRO] sur INT0, le bouton [UN] sur INT1, le bouton [ENTRÉE] sur INT2 puis le bouton [EFFACER] sur RB3. Voyez le diagramme ci-dessous, ainsi que le montage qui suit pour les détails.



Liste des composantes		
Description	**Abréviation**	**Valeur**
Résistance	R	10 mΩ → Brun, noir, bleu
Bouton poussoir	B	



Code assembleur

Pour remplacer la boucle sans fin par des interruptions, il nous faut à nouveau modifier le contenu de la partie « main », en remplaçant tout le contenu de la boucle par une instruction « sleep ». Notez que vous pouvez également retirer les définitions des boutons sur les ports 'D' et 'E', dans la section « DEFINE ».

```
;******************************************************************************
;              PROGRAMME PRINCIPAL
;******************************************************************************

main:
        call    pauseParBoucle
        call    pauseParBoucle

        allumerDEL    DEL_ROUGE
        allumerDEL    DEL_BLEUE
        allumerDEL    DEL_VERTE

boucleBoutons:
        sleep
        goto    boucleBoutons

        END
```

Ensuite, il nous faut configurer les interruptions. Deux registres nous concernent ici. Dans un premier temps, il y a le registre INTCON (Interrupt Control). Le premier bit important est le bit no.7, appelé GIE pour Global Interrupt Enable. Par défaut, ce bit est à '0' et aucune interruption n'est permise. Il nous faudra donc le mettre à '1'. Il est bon de se rappeler que ce bit est automatiquement mis à '0' lorsqu'une interruption survient, afin d'empêcher d'autres interruptions d'interrompre notre routine d'interruption. Ce bit est également remis à '1' lors de l'exécution de l'instruction « retfie » qui met fin à la routine d'interruption.

Quatre autres bits sont aussi importants dans ce registre. Il s'agit des bits 3 et 4, qui permettent d'activer respectivement les interruptions sur RBx et sur INT0, de même que les bits 0 et 1, qui sont les bits d'état qui déterminent respectivement s'il y a eu une interruption sur RBx ou sur INT0.

Dans un deuxième temps, il y a le registre INTCON3 qui nous concerne. Dans ce registre, quatre bits sont importants. Ils sont les même que pour RBx et INT0, mais s'appliquent à INT1 et INT2. Donc, les bits 3 et 4 permettent d'activer respectivement les interruptions sur INT1 et sur INT2, et les bits 0 et 1 déterminent respectivement s'il y a eu une interruption sur INT1 ou sur INT2.

Notez qu'il existe d'autres registres qui concernent les interruptions, tels que INTCON2, RCON, PIRx, PIEx et IPRx, mais ceux-ci touchent des options avancées qui ne seront pas abordées ici.

Donc, pour en terminer avec la configuration des interruptions, il nous suffit d'ajouter les lignes de la page suivante dans la partie d'initialisation de notre code.

```
;******************************************************************************
;             INITIALISATION
;******************************************************************************

       [...]

       ; *** Registre INTCON ***
       ; Activer les interruptions GLOBALES, soit le bit no.7
       ; Activer les interruptions sur les ports RB3 à RB7, soit le bit 3
       ; Activer les interruptions sur INT0 (broche no.33), soit le bit 4
       movlw  B'10011000'
       movwf  INTCON

       ; *** Registre INTCON3 ***
       ; Activer les interruptions sur INT1 (broche no.34), soit le bit 3
       ; Activer les interruptions sur INT2 (broche no.35), soit le bit 4
       movlw  B'00011000'
       movwf  INTCON3
```

Maintenant que les interruptions sont activées, il nous faut les traiter. Comme nous l'avions fait pour définir l'instruction à exécuter au démarrage du PIC avec l'instruction « ORG », nous allons définir celle qu'il faut exécuter lorsqu'une interruption survient.

On se souvient que dans la section « Adresses de démarrage et d'interruptions », nous avions mentionné qu'il y avait des adresses importantes qui étaient appelées lors des interruptions. Celle qui nous intéresse est l'adresse `0x0008`, qui correspond à l'adresse des interruptions de haute priorité. Nous allons donc mettre un « goto » à cette adresse qui nous mènera à notre routine d'interruption. Voyez le code ci-dessous.

```
;******************************************************************************
;             DEMARRAGE SUR RESET ET INTERRUPTIONS
;******************************************************************************

       ORG    0x0000                ; Adresse de départ après reset
       goto    init

       ORG    0x0008                ; Adresse du vecteur d'interruptions
       goto   routineInterruption
```

Il ne nous reste plus qu'à faire notre routine, puis c'est tout. Tel que mentionné précédemment, nous ne nous préoccuperons pas de sauvegarder l'environnement, puis que nous n'y toucherons pas. Nous aurons donc à trouver quel bouton a provoqué l'interruption en vérifiant les bits d'état, à allumer la DEL correspondante, à réinitialiser le bit d'état, puis à mettre fin à l'interruption. Le code correspondant est montré à la page suivante. Noter que ce code est ajouté à la suite du code ci-dessus.

```
;****************************************************************************
;              ROUTINES D'INTERRUPTION
;****************************************************************************

       ; RAPPEL pour les bits d'état :            RBx  = INTCON, 0
       ;                                          INT0 = INTCON, 1
       ;                                          INT1 = INTCON3, 0
       ;                                          INT2 = INTCON3, 1

routineInterruption:
       ; Trouver la source de l'interruption
       btfsc          INTCON, 1                   ; Si != INT0, alors sauter l'instruction suivante
       goto           BOUTON_0

       btfsc          INTCON3, 0                  ; Si != INT1, alors sauter l'instruction suivante
       goto           BOUTON_1

       btfsc          INTCON3, 1                  ; Si != INT2, alors sauter l'instruction suivante
       goto           BOUTON_ENTR

       btfsc          INTCON, 0                   ; Si != RBx, alors sauter l'instruction suivante
       goto           BOUTON_EFFA

       ; So on ne trouve pas la source...
       return                                     ; Retour simple, sans réactiver les interruptions


BOUTON_0:
       allumerDEL     DEL_BLEUE
       bcf            INTCON, 1                   ; Remettre le bit d'état de l'interruption à 0
       retfie                                     ; Retour d'interruption

BOUTON_1:
       allumerDEL     DEL_BLEUE
       bcf            INTCON3, 0                  ; Remettre le bit d'état de l'interruption à 0
       retfie                                     ; Retour d'interruption

BOUTON_ENTR:
       allumerDEL     DEL_VERTE
       bcf            INTCON3, 1                  ; Remettre le bit d'état de l'interruption à 0
       retfie                                     ; Retour d'interruption

BOUTON_EFFA:
       allumerDEL     DEL_ROUGE
       bcf            INTCON, 0                   ; Remettre le bit d'état de l'interruption à 0
       retfie                                     ; Retour d'interruption
```

Voilà! Il ne reste plus qu'à assembler, puis à charger le tout dans le microcontrôleur.

Les communications USB

Avant de poursuivre avec les communications USB, il faudra télécharger et installer les éléments ci-dessous. À noter que nous passons maintenant à la programmation en langage C pour la suite des événements. La raison est que la bibliothèque fournie par Microchip, et son modèle de base pour les communications USB, sont en langage C. Voici donc la liste des outils nécessaires, de même que les adresses Internet où on peut les trouver.

Nom	Adresse Internet
MPLAB IDE v8.40 32 bits	**Note importante** : Si vous avez une version de MPLAB plus récente que 8.40, sachez qu'elle ne sera pas compatible avec MPLAB C18, ci-dessous. http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en023073
MPLAB C for PIC18 v3.41 Standard-Eval Version	http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en010014&redirects=c18
USB Framework for PIC18, PIC24 & PIC32	http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2680&dDocName=en547784

Lors de l'installation de l'« *USB framework* », plusieurs modules sont offerts. Le seul dont nous avons vraiment besoin pour notre projet est «USB Demo Projects ».

Une fois l'installation de l'« *USB framework* » terminée, vous remarquerez que le répertoire nouvellement créé contiendra beaucoup de choses. Voici donc une arborescence des seuls fichiers dont nous aurons besoin.

- [DOSSIER] ./**Microchip Solutions v2012-04-03**
  - [DOSSIER] ./Microchip Solutions v2012-04-03/**Microchip**
    - [DOSSIER] ./Microchip Solutions v2012-04-03/Microchip/**Include**
      - **Compiler.h**
      - **GenericTypeDefs.h**
      - [DOSSIER] ./Microchip Solutions v2012-04-03/Microchip/Include/**USB**
        - **usb.h**
        - **usb_ch9.h**
        - **usb_common.h**
        - **usb_device.h**
        - **usb_function_hid.h**
        - **usb_hal.h**
        - **usb_hal_pic18.h**
      - [DOSSIER] ./Microchip Solutions v2012-04-03/Microchip/**USB**
        - [DOSSIER] ./Microchip Solutions v2012-04-03/Microchip/USB/**HID Device Driver**
          - **usb_function_hid.c**
          - **usb_device.c**
          - **usb_device_local.h**
  - [DOSSIER] ./Microchip Solutions v2012-04-03/**USB**
    - [DOSSIER] ./Microchip Solutions v2012-04-03/USB/**Device - HID - Custom Demos**
      - [DOSSIER] ./Microchip Solutions v2012-04-03/USB/Device - HID - Custom Demos/**Firmware**
      - **HardwareProfile.h**
      - **main.c**
      - **USB Device - HID - Simple Custom Demo - C18 - PIC18F Starter Kit 1.mcp**
      - **USB Device - HID - Simple Custom Demo - C18 - PIC18F Starter Kit 1.mcw**
      - **usb_config.h**
      - **usb_descriptors.c**

Libre à vous de tout conserver ou de ne conserver que les fichiers nécessaires afin de libérer de l'espace sur votre disque dur. Maintenant, nous sommes prêts à commencer. On ouvre donc « **USB Device - HID - Simple Custom Demo - C18 - PIC18F Starter Kit 1.mcp** ». Il s'agit d'un projet MPLAB qui contient la base de notre projet.

La première chose à faire, après l'ouverture du projet, c'est de spécifier à MPLAB le périphérique que nous utilisons. Pour ce faire, on clique sur « *Configure→Select Device* », puis on choisit le PIC18F4550.

On peut maintenant passer à la modification du code source des fichiers. Le premier à modifier est « **usb_descriptors.c** ». Nous y définirons les identifiants de notre microcontrôleur en ajoutant le code ci-dessous, tout juste après les sections INCLUDES et CONSTANTS, aux environs de la ligne #164.

```
/** IDENTIFICATION DU PÉRIPHÉRIQUE **********************************/
// Device Vendor Indentifier (VID)
// 0x04D8 = VID de Microchip's
#define USB_VID      0x04D8
// Device Product Indentifier (PID)
// 0x0042 = PID de PIC18F4550
#define USB_PID      0x0042
```

Juste après vient la description du périphérique (`/* Device Descriptor */`). Nous y modifierons deux lignes, telles que montrées en gras dans le code ci-dessous.

```
/* Device Descriptor */
ROM USB_DEVICE_DESCRIPTOR device_dsc=
{
    0x12,                    // Size of this descriptor in bytes
    USB_DESCRIPTOR_DEVICE,   // DEVICE descriptor type
    0x0200,                  // USB Spec Release Number in BCD format
    0x00,                    // Class Code
    0x00,                    // Subclass code
    0x00,                    // Protocol code
    USB_EP0_BUFF_SIZE,       // Max packet size for EP0, see usb_config.h
    USB_VID,                 // Vendor ID
    USB_PID,                 // Product ID
    0x0002,                  // Device release number in BCD format
    0x01,                    // Manufacturer string index
    0x02,                    // Product string index
    0x00,                    // Device serial number string index
    0x01                     // Number of possible configurations
};
```

Nous allons maintenant modifier la description du manufacturier (`//Manufacturer string descriptor`) et la description du produit (`//Product string descriptor`). Le code à modifier devrait maintenant se trouver aux environs des lignes #246 et #253. Modifiez ces lignes pour y mettre les informations de votre choix. La plus importante étant la description du produit, car ces celle-ci qui s'affichera dans votre page de « Périphériques et imprimantes » (sous Windows 7). Veillez à ne pas oublier de modifier la taille des tableaux selon vos nouvelles descriptions. Dans mon cas, il s'agit de « `string[15]` » et de « `string[30]` ». Le code est à la page suivante.

```
//Manufacturer string descriptor
ROM struct{BYTE bLength;BYTE bDscType;WORD string[15];}sd001={
sizeof(sd001),USB_DESCRIPTOR_STRING,
{'P','a','t','r','i','c','e',' ','B','o','n','n','e','a','u'
}};

//Product string descriptor
ROM struct{BYTE bLength;BYTE bDscType;WORD string[30];}sd002={
sizeof(sd002),USB_DESCRIPTOR_STRING,
{'C','l','a','v','i','e','r',' ','b','i','n','a','i','r','e',' ',
'H','I','D',' ','P','I','C','1','8','F','4','5','5','0'
}};
```

Voilà! C'est tout pour les modifications à apporter à ce fichier. Cela dit, si vous tentez de compiler le code maintenant (ctrl-F10), vous aurez une quinzaine d'erreurs au niveau des configurations. Exemple : *configuration setting 'WDTEN' not recognized*. Le problème est que le fichier « main.c » qui est fourni ici n'a pas été conçu pour le PIC18F4550, mais pour le PIC18F_STARTER_KIT_1.

Nous allons donc ouvrir ce fichier et modifier les noms de configurations fautifs. Ici, deux options s'offrent à vous. Soit vous supprimez du code original tout ce qui ne concerne pas le PIC18F4550, soit vous ne remplacez que le code de configuration du PIC18F_STARTER_KIT_1 par celui du PIC18F4550. Pour ma part, j'ai choisi la première option. J'ai donc remplacé les lignes numéro 53 à 246 par le code ci-dessous.

> **Note** : La liste des configurations a été réduite, pour la simple raison que les autres valeurs de configurations sont celles par défaut du microcontrôleur. Il est donc inutile de les redéfinir une seconde fois.

```
#pragma config PLLDIV   = 5          // Oscillateur externe de 20Mhz ÷ 5 = 4Mhz pour le PLL
#pragma config CPUDIV   = OSC1_PLL2  // Horloge du processeur du PIC = PLL de 96MHz ÷ 2 = 48Mhz
#pragma config USBDIV   = 2          // Horloge de module USB = PLL de 96MHz ÷ 2 = 48Mhz
#pragma config FOSC     = HSPLL_HS   // Oscillateur externe de type HS avec activation du PLL
#pragma config VREGEN   = ON         // Activer le régulateur de voltage du module USB
#pragma config WDT      = OFF        // Désactiver le Watchdog timer
#pragma config MCLRE    = ON         // Activer le Master clear sur la broche #1
#pragma config LVP      = OFF        // Désactiver la programmation avec bas voltage
```

Maintenant, il faut effacer tout le contenu de la fonction « `static void InitializeSystem(void)` ». Nous y mettrons notre propre code plus tard. Il faut également effacer entièrement les fonctions suivantes, que nous n'utiliserons pas. Pensez aussi à supprimer les prototypes de fonction correspondants en début de code.

- `void UserInit(void)`
- `WORD_VAL ReadPOT(void)`
- `void BlinkUSBStatus(void)`

Pour terminer, dans la fonction « `static void InitializeSystem(void)` », ne conservez que le code suivant. Vous remarquerez que le code des données entrantes, celui qui est utilisé par le « `switch()` », est à la position [1] du tampon « `ReceivedDataBuffer` ». Il s'agit d'un choix arbitraire.

```
void ProcessIO(void)
{
	// S'assurer que le module USB est fonctionnel avant de poursuivre.
	if((USBDeviceState < CONFIGURED_STATE) || (USBSuspendControl==1))
		return;

	if(!HIDRxHandleBusy(USBOutHandle))		// Vérifier si des données ont été reçues.
	{
		switch(ReceivedDataBuffer[1])		// Appliquer les actions correspondantes.
		{
		default:
			break;
		}
		//Réarmer la sortie (OUT) pour le prochain paquet.
		USBOutHandle = HIDRxPacket(HID_EP,(BYTE*)&ReceivedDataBuffer,64);
	}
}//end ProcessIO
```

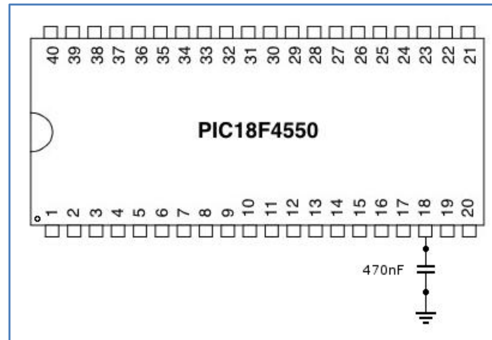
Il nous reste maintenant à modifier le fichier « HardwareProfile.h ». Ce fichier n'est rien d'autre qu'une coquille qui vérifie avec quel microcontrôleur nous travaillons, puis inclus le fichier « *header* » correspondant. Dans notre cas, il inclura le fichier « HardwareProfile - PICDEM FSUSB.h ». Par contre, comme ce dernier ne se retrouve pas dans le projet MPLAB, et que sont contenu n'est pas tout à fait adapté à notre projet, nous ne l'utiliserons pas. Nous transférerons plutôt les quelques lignes utiles directement dans le fichier « HardwareProfile.h ». Nous effacerons donc le contenu de ce dernier, pour n'y laisser que le code ci-dessous, et y revenir plus tard.

```
#ifndef HARDWARE_PROFILE_H
#define HARDWARE_PROFILE_H

    #define tris_self_power     TRISAbits.TRISA2
    #if defined(USE_SELF_POWER_SENSE_IO)
    #define self_power          PORTAbits.RA2
    #else
    #define self_power          1
    #endif

    #define PROGRAMMABLE_WITH_USB_HID_BOOTLOADER

#endif  //HARDWARE_PROFILE_H
```

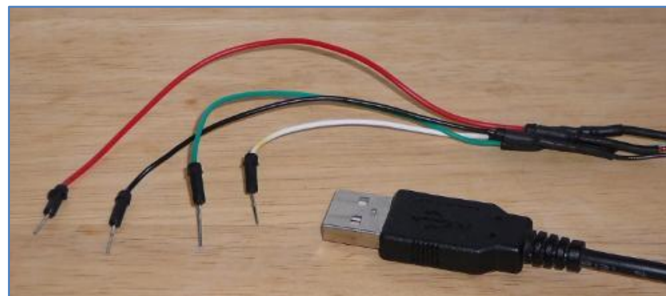
Maintenant, tout devrait compiler. Appuyez sur les touches [CTRL] + [F10] ou cliquez sur le bouton pour lancer la compilation.

Avant de poursuivre avec la programmation, il nous faut faire deux petits ajouts à notre montage électronique. Premièrement, nous avons besoin d'alimenter le module USB du microcontrôleur, qui nécessite une

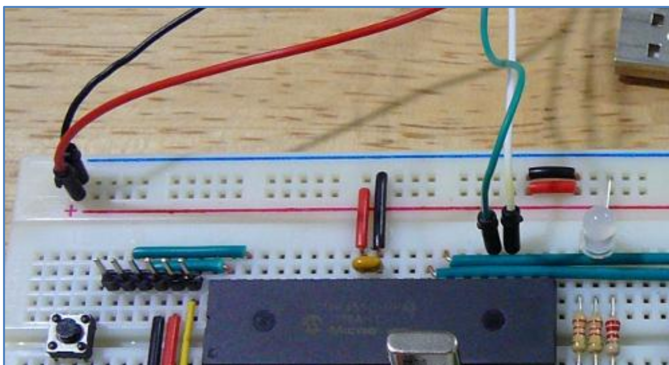
alimentation négative distincte. Nous mettrons donc un condensateur de 470nF entre la borne négative et la broche no.18, tel qu'illustré ci-dessous.



Deuxièmement, il nous faudra un câble USB pour relier notre clavier à un ordinateur. Malheureusement, il n'existe pas de connecteur USB qui soit conçu pour être utilisé sur une planche de montage. Nous devons donc improviser un peu. L'option la plus simple sera de prendre un câble régulier, de couper une extrémité, et d'y faire quelques petites soudures afin d'y mettre de petites tiges droites, qui s'enfonceront dans la planche de montage. Voyez la photo ci-dessous. Notez que j'ai utilisé une gaine thermorétractable pour couvrir les soudures.



Il ne reste plus qu'à connecter votre nouveau câble à la planche de montage. Notez que le câble blanc (Data-) se connecte sur la broche no.23 et que le câble vert (Data+) se connecte sur la broche no.24. Voyez la photo ci-dessous.



Note : Pour la suite du tutoriel, j'ai abandonné les interruptions, pour revenir à la boucle infinie, conformément au code fourni par Microchip. Pensez à modifier votre montage électronique en conséquence. Ceci aura comme effet que nous aurons moins de code à modifier. Cela dit, comme nous avons vu les interruptions plus tôt, rien ne vous empêchera de modifier le code que je vous donne ici lorsque nous aurons terminé ce tutoriel.

Vous pouvez maintenant relier votre câble USB à votre ordinateur. Évidemment, aucune communication ne se fera entre votre clavier et votre ordinateur, mais le microcontrôleur sera alors alimenté par le port USB et il sera détecté par votre ordinateur.

Si votre PICkit 3 n'est pas connecté au clavier, connectez-le. Si le microcontrôleur est alimenté par le programmateur, vous pouvez annuler cette option. Nous sommes maintenant prêts à poursuivre avec la programmation des communications USB.

Détection du périphérique par l'ordinateur

Nous commencerons par nous faire un petit programme, en langage C++, qui nous permettra de récupérer quelques informations de notre microcontrôleur à partir de notre ordinateur. Pour ce faire, nous utiliserons « hidapi.h », que vous pouvez télécharger à l'adresse https://github.com/signal11/hidapi/tree/master/hidapi. Je vous suggère de l'ouvrir et de jeter un coup d'œil au « `struct hid_device_info {…}` », situé en début de code. Ceci vous permettra de vous familiariser avec les infos que vous pourrez récupérer du microcontrôleur.

Commencer maintenant par vous créer un projet C++ « *Console application* » (j'utilise Code::Blocks). Ajoutez-y le fichier « hidapi.h » et modifiez le contenu du fichier « main.c » avec le code ci-dessous.

```
#include <cstdio>
#include "hidapi.h"


int main(int argc, char* argv[])
{
	struct hid_device_info *liste_PeripheriquesHID;
	struct hid_device_info *peripheriqueHID_Courant;

	liste_PeripheriquesHID = hid_enumerate(0x0, 0x0);
	peripheriqueHID_Courant = liste_PeripheriquesHID;
	printf("\n\n");
	printf("[Liste des peripheriques trouves]\n\n");
	while (peripheriqueHID_Courant) {
		printf("    vendor_id: %04hx \n", peripheriqueHID_Courant->vendor_id);
		printf("    product_id: %04hx \n", peripheriqueHID_Courant->product_id);
		printf("    serial_number: %04hx \n", (int)peripheriqueHID_Courant->serial_number);
		printf("    Manufacturer: %ls \n", peripheriqueHID_Courant->manufacturer_string);
		printf("    Product:      %ls \n", peripheriqueHID_Courant->product_string);
		printf("\n\n");
		peripheriqueHID_Courant = peripheriqueHID_Courant->next;
	}
	hid_free_enumeration(liste_PeripheriquesHID);

	return 0;
}
```

Compilez-le, puis exécutez-le. Si votre clavier binaire est connecté à votre ordinateur et que vous y avez chargé le code compilé plus haut, vous devriez avoir un résultat similaire à la copie d'écran que vous pouvez voir à la page suivante.

"D:\Mes documents\Dropbox\[S6] Hiver 2012\IFT592\Clavier binaire\testHidApi\testHidApi.exe"

```
[Liste des peripheriques trouves]

    vendor_id: 046d
    product_id: c521
    serial_number: 1bb0
    Manufacturer: Logitech
    Product:       USB Receiver


    vendor_id: 046d
    product_id: c521
    serial_number: 2b98
    Manufacturer: Logitech
    Product:       USB Receiver


    vendor_id: 04d8
    product_id: 0042
    serial_number: 1b70
    Manufacturer: Patrice Bonneau
    Product:       Clavier binaire HID PIC18F4550
```

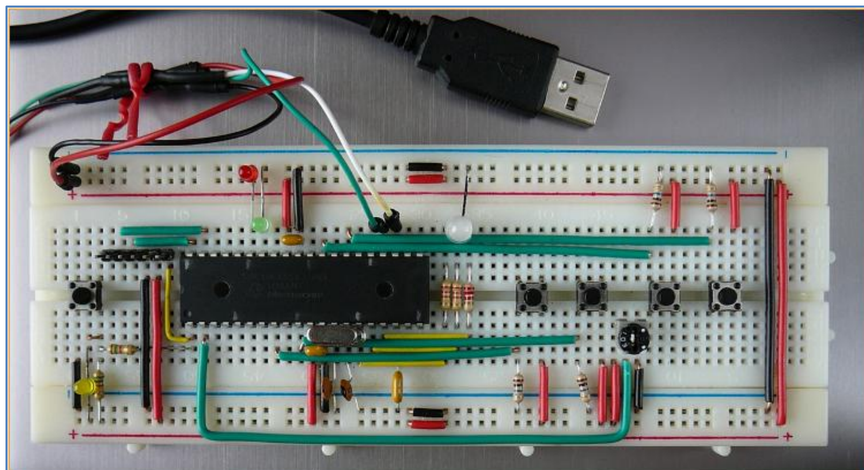
Code source du microcontrôleur

Poursuivons maintenant avec les modifications à apporter au code source du microcontrôleur. Nous avons deux fichiers à modifier. Il y a bien évidemment « main.c », mais nous commencerons par apporter quelques ajouts au fichier « HardwareProfile.h ». Nous y définirons les boutons, les DEL ainsi que les fonctions des DEL. Ouvrez le fichier, puis ajoutez-y le code de la page suivante. Les noms utilisés sont assez significatifs pour se passer d'explication.

```
// Définition des ÉTATS des DEL
#define ETAT_ON                     1
#define ETAT_OFF                    0

// Définition des DEL
#define DEL_BLEUE                   LATCbits.LATC0
#define DEL_VERTE                   LATCbits.LATC1
#define DEL_ROUGE                   LATCbits.LATC2
#define DEL_ACK                     LATBbits.LATB1
#define DEL_TEMOIN                  LATBbits.LATB2

// Définition des fonctions sur les DEL
#define ALTERNER_DEL_BLEUE          DEL_BLEUE = !DEL_BLEUE
#define ALTERNER_DEL_VERTE          DEL_VERTE = !DEL_VERTE
#define ALTERNER_DEL_ROUGE          DEL_ROUGE = !DEL_ROUGE

// Définition des codes des DEL
#define CODE_DEL_BLEUE              0XD0                // Choix arbitraire
#define CODE_DEL_VERTE              0XD1                // Choix arbitraire
#define CODE_DEL_ROUGE              0XD2                // Choix arbitraire

// Définition des codes divers
#define CODE_ACCUSE_RECEPTION       0XB1                // Choix arbitraire
```

```
#define CODE_TEMOIN_LOGICIEL_ON     0XB2                  // Choix arbitraire
#define CODE_TEMOIN_LOGICIEL_OFF    0XB3                  // Choix arbitraire

// Définition des BOUTONS
#define BOUTON_0                    PORTEbits.RE0
#define BOUTON_1                    PORTEbits.RE2
#define BOUTON_ENTREE               PORTDbits.RD6
#define BOUTON_EFFACER              PORTDbits.RD4

// Définition des valeurs des touches du clavier
#define VALEUR_TOUCHE_0             0X27   // = code d'un vrai clavier pour la touche '0'
#define VALEUR_TOUCHE_1             0X1E   // = code d'un vrai clavier pour la touche '1'
#define VALEUR_TOUCHE_ENTREE        0X28   // = code d'un vrai clavier pour la touche 'ENTREE'
#define VALEUR_TOUCHE_EFFACER       0X2A   // = code d'un vrai clavier pour la touche 'BACKSPACE'
```

Vous aurez sans doute remarqué qu'il y a un code pour un accusé de réception, de même que deux codes pour allumer ou éteindre un témoin, qui spécifie si le logiciel interactif, que nous verrons plus tard, est en exécution ou pas. Ceci demande un petit ajout à faire sur la planche de montage. J'ai donc choisi de connecter une DEL verte à la broche no.34 du microcontrôleur, pour jouer le rôle de l'accusé de réception, et une DEL rouge à la broche no.34, pour jouer le rôle de témoin. Voyez la photo ci-dessous qui montre le montage électronique à l'état actuel et sur laquelle nous pouvons voir les deux nouvelles DEL. Nous pouvons également y voir le potentiomètre qui servira lors de l'utilisation des entrées analogiques.



Nous passons maintenant à la modification du fichier « main.c ». Nous commencerons par ajouter une fonction de pause par boucle. Nous ajouterons donc, juste avant la fonction « `int main(void)` », le code ci-dessous.

```
// Création d'une fonction de pause par boucle
unsigned long cptr1;

void pauseParBoucle (void)
{
        cptr1 = 200000;
        do {
                cptr1--;
```

```
	} while(cptr1);
}
```

Ensuite, nous passons à la fonction « `static void InitializeSystem(void)` ». Le code ci-dessous est assez bien commenté pour se passer de détails.

```
	// Mettre tous les ports en entrée (1), à l'exception des
	// trois premières broches du port C, de même que la deuxième
	// et troisième du port B, qui sont en sortie (0) pour les DEL.
	TRISA = 0b11111111;
	TRISB = 0b11111001;
	TRISC = 0b11111000;
	TRISD = 0b11111111;
	TRISE = 0b11111111;

	// Mettre toutes les entrées en mode numérique (bits <3:0> à '1')
	ADCON1 = 0b00001111;

	// Effacer tous les ports
	PORTA = 0b00000000;
	PORTB = 0b00000000;
	PORTC = 0b00000000;
	PORTD = 0b00000000;
	PORTE = 0b00000000;

	// Initialiser le périphérique et les handles d'entrée et de sortie
	USBDeviceInit();
	USBOutHandle = 0;
	USBInHandle = 0;

	// Initialiser le tableau d'envoi de données à 0
	for (cptr1 = 0; cptr1 < 64; cptr1++)
		ToSendDataBuffer[cptr1] = 0x00;

	// Alterner les DEL pour aviser que le microcontrôleur est prêt
	DEL_BLEUE = ETAT_ON;
	pauseParBoucle();
	pauseParBoucle();
	DEL_BLEUE = ETAT_OFF;

	DEL_ROUGE = ETAT_ON;
	pauseParBoucle();
	pauseParBoucle();
	DEL_ROUGE = ETAT_OFF;

	DEL_VERTE = ETAT_ON;
	pauseParBoucle();
	pauseParBoucle();
	DEL_VERTE = ETAT_OFF;
```

Nous passons ensuite à la fonction « `void ProcessIO(void)` », où nous définissons, dans le `switch()`, les actions à prendre selon les messages reçus depuis le logiciel interactif, que nous verrons bientôt. La code à ajouter se trouve à la page suivante.

```
        // DEL bleue
        case CODE_DEL_BLEUE :
               ALTERNER_DEL_BLEUE;
               break;

        // DEL verte
        case CODE_DEL_VERTE :
               ALTERNER_DEL_VERTE;
               break;

        // DEL rouge
        case CODE_DEL_ROUGE :
               ALTERNER_DEL_ROUGE;
               break;

        // DEL ACK (accusé de réception)
        case CODE_ACCUSE_RECEPTION :
               DEL_ACK = ETAT_ON;
               pauseParBoucle();
               DEL_ACK = ETAT_OFF;
               break;

        // DEL_TEMOIN
        case CODE_TEMOIN_LOGICIEL_ON :
               DEL_TEMOIN = ETAT_ON;
               break;

        // DEL_TEMOIN
        case CODE_TEMOIN_LOGICIEL_OFF :
               DEL_TEMOIN = ETAT_OFF;
               DEL_BLEUE = ETAT_OFF;
               DEL_VERTE = ETAT_OFF;
               DEL_ROUGE = ETAT_OFF;
               break;
```

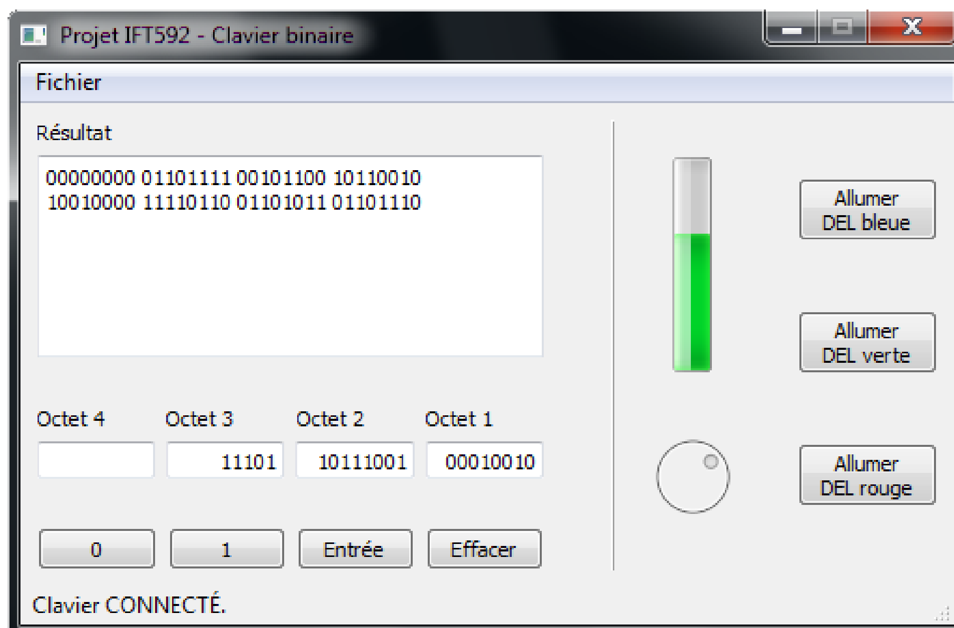
Ensuite, tout juste après l'appel à la fonction « `void ProcessIO(void)` » dans fonction « `int main(void)` », nous ajoutons l'écoute des boutons-poussoirs et les actions à prendre. Vous remarquerez que le code des données sortantes est à la position [2] du tampon « `ToSendDataBuffer` ». Il s'agit d'un choix arbitraire.

```
               // Vérifier l'état des boutons
               if (BOUTON_0 == ETAT_ON) {
                      DEL_BLEUE = ETAT_ON;
                      ToSendDataBuffer[2] = VALEUR_TOUCHE_0;
                      // Transmettre sur le port USB
                      if(!HIDTxHandleBusy(USBInHandle)) {
                             USBInHandle = HIDTxPacket(HID_EP,(BYTE*)&ToSendDataBuffer[0],64);
                      }
                      pauseParBoucle();
                      DEL_BLEUE = ETAT_OFF;
               } else if (BOUTON_1 == ETAT_ON) {
                      DEL_BLEUE = ETAT_ON;
                      ToSendDataBuffer[2] = VALEUR_TOUCHE_1;
                      // Transmettre sur le port USB
                      if(!HIDTxHandleBusy(USBInHandle)) {
                             USBInHandle = HIDTxPacket(HID_EP,(BYTE*)&ToSendDataBuffer[0],64);
```

```
			}

			pauseParBoucle();
			DEL_BLEUE = ETAT_OFF;
		} else if (BOUTON_ENTREE == ETAT_ON) {
			DEL_VERTE = ETAT_ON;
			ToSendDataBuffer[2] = VALEUR_TOUCHE_ENTREE;
			// Transmettre sur le port USB
			if(!HIDTxHandleBusy(USBInHandle)) {
				USBInHandle = HIDTxPacket(HID_EP,(BYTE*)&ToSendDataBuffer[0],64);
			}


			pauseParBoucle();
			DEL_VERTE = ETAT_OFF;
		} else if (BOUTON_EFFACER == ETAT_ON) {
			DEL_ROUGE = ETAT_ON;
			ToSendDataBuffer[2] = VALEUR_TOUCHE_EFFACER;
			// Transmettre sur le port USB
			if(!HIDTxHandleBusy(USBInHandle)) {
				USBInHandle = HIDTxPacket(HID_EP,(BYTE*)&ToSendDataBuffer[0],64);
			}
			pauseParBoucle();
			DEL_ROUGE = ETAT_OFF;
		}
```

Le logiciel interactif

Pour ce qui est de l'interface graphique, nous allons utiliser Qt par Nokia. Cependant, je ne m'attarderai pas à expliquer son fonctionnement ici, étant donné que ce n'est pas le but de ce tutoriel. Par contre, je m'attarderai sur la partie des communications USB. Le code source de ce logiciel se trouve dans le répertoire « interactif ». La capture d'écran ci-dessous montre l'interface graphique en question.



Fonctionnement

Tel qu'on peut le voir dans le coin inférieur gauche de la photo, l'interface vérifie si le clavier est connecté ou non. On remarque également les quatre boutons dans la section de gauche.

Les boutons [0] et [1] appellent les fonctions qui vont ajouter le chiffre correspondant dans les champs des quatre octets au dessus. Les bits sont ajoutés de droite vers la gauche. Ce qui signifie que lorsqu'un bit est ajouté, les bits précédents sont décalés vers la gauche, puis le nouveau bit est ajouté à droite. Ce qui fait en sorte que le bit le plus récent est toujours le moins significatif et le plus ancien est le plus significatif. Lorsque les quatre octets sont pleins (32 bits), les actions des boutons [0] et [1] sont ignorées.

Le bouton [Entrée] appelle une fonction qui complète le nombre de bits à 32 en ajoutant des zéros pour les bits les plus significatifs manquants, puis transfère le résultat dans la boîte de texte au dessus, nommé « Résultat ». Le tout se termine en vidant les quatre octets. Lorsque les quatre octets sont vides, ce bouton ajoute 32 bits ayant la valeur '0' à la section « Résultat ».

Le bouton [Effacer] appelle une fonction qui efface le bit le plus récemment entré dans les octets, soit le plus à droite, puis décale les bits restants vers la droite pour combler le vide. Lorsque les quatre octets sont vides, la ligne la plus récemment ajoutée à la section « Résultat » est effacée. Lorsque cette section est également vide, l'action de ce bouton est ignorée.

Ce sont ces mêmes fonctions qui sont appelées par les boutons ci-dessus sont également appelées par les boutons correspondants du clavier binaire.

La barre de progression verticale, de même que le bouton rotatif, ne peut être utilisée directement via le logiciel. Ils sont modifiés directement par le potentiomètre du clavier binaire. Nous verrons ce point ultérieurement dans ce tutoriel.

Les trois boutons de la section de droite servent à allumer ou éteindre individuellement chacune des trois DEL principales. Lorsque le clavier binaire n'est pas connecté, ces boutons sont désactivés.

Le logiciel interactif écoute les événements du système d'exploitation, en l'occurrence Windows 7. Lorsque le système envoie un message de nouvelle connexion de périphérique, le logiciel récupère les informations, puis vérifie s'il s'agit du microcontrôleur PIC18F4550 à l'aide des numéros d'identification VID et PID, que l'on a vu précédemment. Lorsqu'il reconnait le clavier binaire, il lui envoie un code afin qu'il allume son témoin lumineux, indiquant que le logiciel est actif. Lorsque le logiciel se ferme, il envoie un code au clavier binaire pour lui indiquer de fermer son témoin lumineux.

Lorsque le système d'exploitation envoie un message de nouvelle déconnexion de périphérique, le logiciel récupère les informations, puis vérifie s'il s'agit du microcontrôleur PIC18F4550, toujours à l'aide des numéros d'identification VID et PID. Lorsqu'il reconnait qu'il s'agit du clavier binaire, il change son message dans la barre d'état pour « Clavier NON connecté », puis désactive les boutons de la section de droite concernant les DEL.

Explication du code source

La première chose qu'il faut savoir, c'est que Qt ne gère pas les communications USB. C'est pourquoi nous l'utiliserons conjointement avec « dbt.h » et « setupapi.h ». Le premier sera utilisé pour les détections de périphériques et le second pour les communications USB.

Reconnaissance du clavier binaire

La reconnaissance de la connexion et de la déconnexion du clavier binaire sont traitées dans la fonction « `bool FenetrePrincipale::winEvent(MSG *msg_, long *resultat_)` », basée sur un exemple de Microsoft. On capture d'abord le message, puis on regarde s'il s'agit du changement d'état d'un périphérique avec « `if(typeMessage == WM_DEVICECHANGE)` ». Si oui, on vérifie s'il s'agit d'une connexion ou d'une déconnexion avec un « `switch()` ». Les valeurs respectives sont "DBT_DEVICEARRIVAL et DBT_DEVICEREMOVECOMPLETE.

Dans les deux cas, on vérifie s'il s'agit d'un périphérique de type interface. Je vous rappelle qu'on travaille ici avec une connexion USB de type *Human Interface Device* (HID). S'il s'agit d'un type interface, nous récupérons son nom, puis vérifions s'il contient le VID et PID du microcontrôleur. Si oui, un signal Qt est émis afin que le logiciel lance la fonction appropriée.

Il y a un autre message à traiter. Il s'agit du message de type WM_PAINT. Ce message est lancé lorsque le système, ou une autre application, fait une requête pour peindre une portion de la fenêtre d'une application. Il est plus précisément lancé lors de l'appel de la fonction « UpdateWindow ». Puisque cette dernière est appelée lors du démarrage de notre application, et qu'elle est importante, nous devons la traiter. Voyez le code ci-dessous, pour les détails.

```
bool FenetrePrincipale::winEvent(MSG *msg_, long *resultat_)
{
    int typeMessage = msg_->message;

    if(typeMessage == WM_PAINT) {
        if(!messagePaintRecu) {
            GUID InterfaceClassGuid = HID_CLASS_GUID;
            DEV_BROADCAST_DEVICEINTERFACE filtre;
            ZeroMemory(&filtre, sizeof(filtre));
            filtre.dbcc_size = sizeof(DEV_BROADCAST_DEVICEINTERFACE);
            filtre.dbcc_devicetype = DBT_DEVTYP_DEVICEINTERFACE;
            filtre.dbcc_classguid = InterfaceClassGuid;
            HWND MainWindowHandle = this->effectiveWinId();
            hDevNotify = RegisterDeviceNotification(MainWindowHandle,&filtre,
                                                      DEVICE_NOTIFY_WINDOW_HANDLE);
            messagePaintRecu = true;
        }
    }

    if(typeMessage == WM_DEVICECHANGE) {
        PDEV_BROADCAST_HDR lpdb = (PDEV_BROADCAST_HDR)msg_->lParam;
        switch(msg_->wParam) {
        case DBT_DEVICEARRIVAL:
            if (lpdb->dbch_devicetype == DBT_DEVTYP_DEVICEINTERFACE) {
                PDEV_BROADCAST_DEVICEINTERFACE lpdbv = (PDEV_BROADCAST_DEVICEINTERFACE)lpdb;
                int i = 0;
                QString lectureNom;
```

```
                while(lpdbv->dbcc_name[i] != 0) {
                    lectureNom.append(lpdbv->dbcc_name[i]);
                    i++;
                }

                lectureNom = lectureNom.toUpper();
                if(lectureNom.contains(CLAVIER_HID_ID))
                    emit connexionClavier_USB();
            }
            break;
        case DBT_DEVICEREMOVECOMPLETE:
            if (lpdb->dbch_devicetype == DBT_DEVTYP_DEVICEINTERFACE) {
                PDEV_BROADCAST_DEVICEINTERFACE lpdbv = (PDEV_BROADCAST_DEVICEINTERFACE)lpdb;
                int i = 0;
                QString lectureNom;
                while(lpdbv->dbcc_name[i] != 0) {
                    lectureNom.append(lpdbv->dbcc_name[i]);
                    i++;
                }
                lectureNom = lectureNom.toUpper();
                if(lectureNom.contains(CLAVIER_HID_ID))
                    emit deconnexionClavier_USB();
            }
            break;
        }
    }

    return false;
}
```

Les échanges USB

Avant de procéder à l'échange des données, il faut s'assurer de valider la communication. C'est pourquoi on appelle la fonction « `bool FenetrePrincipale::validerCommunication (void)` » tout de suite après que le clavier binaire ait été détecté, soit depuis la fonction « `void FenetrePrincipale::activerCommunicationUSB (void)` », qui elle-même est connectée au signal « `connexionClavier_USB()` » du code ci-dessus.

Cette fonction, également basée sur un exemple de Microsoft, va vérifier que le clavier binaire est toujours connecté. Si c'est le cas, elle créera les *handles* d'entrée et de sortie, activera les boutons, puis retournera une valeur booléenne pour aviser du succès ou de l'échec de la validation. C'est sur les deux *handles* que s'effectuera l'envoi et la réception avec le clavier binaire.

Une fois la validation effectuée, nous sommes prêts à communiquer avec notre périphérique. L'envoi est le plus simple. On met les données à envoyé dans le tampon « outputBuffer », puis on appel la fonction « `int FenetrePrincipale::envoiUSB (void)` » qui se chargera d'écrire sur le *handle* de sortie, puis d'effacer le tampon de sortie.

La réception quant à elle demande l'utilisation d'un *thread* qui sera en permanence à l'écoute du *handle* d'entrée. Lorsque des données seront disponibles en lecture, le *thread* émettra un signal Qt qui lancera la récupération des données, puis appellera la fonction appropriée. Le code source du *thread* est montré à la page suivante.

```
// Constructeur
ThreadUSB::ThreadUSB(bool *clavierConnecte_, HANDLE *readHandle_, unsigned char *inputBuffer_)
{
    clavierConnecte = clavierConnecte_;
    readHandle = readHandle_;
    inputBuffer = inputBuffer_;
}

// Démarrage du thread
void ThreadUSB::run()
{
    DWORD nbrOctetsLus;

    while (true) {
        if(clavierConnecte && *readHandle != INVALID_HANDLE_VALUE) {
            ReadFile(*readHandle, inputBuffer, 65, &nbrOctetsLus, 0);
            emit receptionUSB();
        }
    }
}
```

Le signal « `emit receptionUSB()` », qui est émis par le *thread*, est connecté à la fonction « `void FenetrePrincipale::receptionDonneesUSB (void)` ». C'est elle récupérera les données, puis appellera la fonction correspondante. Le reste du code ne touchant pas les communications USB, je laisse à vos bons soins le choix de consulter le code source intégral, dans le répertoire « interactif ».

Les entrées analogiques

Les entrées analogiques passent par le convertisseur A/D (de l'anglais Analog/Digital), qui calcul la différence entre la tension sur une broche donnée, et un voltage de référence. Ce voltage peut être celui du microcontrôleur, soit 0v - 5v, soit un voltage défini par vous-même. Dans le cadre de ce tutoriel, nous utiliserons la première option. Avec la seconde option, il aurait fallu apporter la tension négative désirée sur la broche no.4 (VREF-) et la tension positive désirée sur la broche no.5 (VREF+). Cela implique qu'il aurait fallu modifier le circuit pour y faire des variations de tensions, ce qui se serait avéré un peu compliqué.

Le choix quant au voltage de référence est défini dans le registre ADCON1. On y met le bit no.5 à '0' pour l'utilisation de la tension du microcontrôleur (par défaut), et à '1' pour l'utilisation de la tension de référence négative (VREF-) et positive (VREF+).

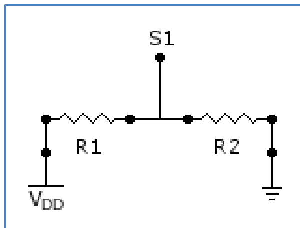
Le bit no.1 du registre ADCON0 est appelé GO/DONE. Lorsque le convertisseur est prêt à travailler, ce bit est à '0'. Lorsque nous voulons lire l'entrée analogique, voir lancer la conversion, il nous suffit de mettre ce bit à '1'. Lorsque le calcul sera terminé, le bit reviendra automatiquement à '0'.

Le résultat de la conversion sera donné sur huit ou dix bits, selon la configuration choisie. Ce qui donnera des valeurs possibles allant respectivement de 0 à 255 et de 0 à 1023. Dans le cas d'une conversion sur huit bits, le résultat se retrouvera dans le registre ADRESH. Dans le cas d'une conversion sur 10 bits, il faudra également

utiliser le registre ADRESL, et faire une concaténation pour lire le résultat. Pour des raisons de simplicité, nous opterons pour la conversion sur 8 bits.

Montage électronique

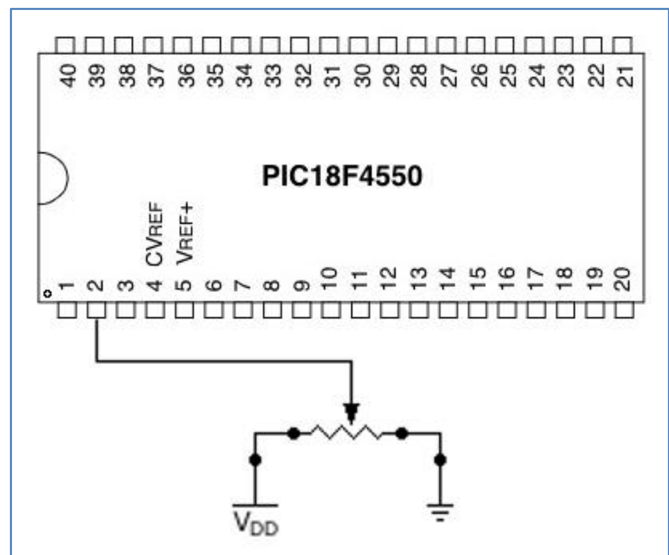
Pour apporter une tension variable sur une broche du microcontrôleur, nous utiliserons un potentiomètre qui agira comme un pont diviseur de tension. Le principe de ce dernier est de ne laisser passer qu'une partie du courant vers la destination.



Jetons un coup d'œil sur le diagramme ci-contre, en se référant au sens conventionnel du courant. Ce qui signifie que l'électricité circule de la borne positive vers la borne négative. V_{DD} représente une tension positive de 5 volts. S1 représente la sortie qui mènera à notre périphérique. En faisant abstraction de la résistance R2 et de la borne négative, la tension sur S1 serait de 5 volts. Or, avec la résistance R2, nous offrons une porte de sortie au courant qui peut maintenant se diriger vers la borne négative. La quantité de courant qui se rendra à S1, ainsi que la quantité de courant qui se rendra à la borne négative, est déterminée par la valeur choisie pour chacune des deux résistances.

Par exemple, deux résistances de même valeur donneront une tension de 2,5 volts sur S1 et l'autre 2,5 volts sera absorbé par la borne négative. Avec R1 = 750 ohms et R2 = 250 ohms, S1 aura une tension de 1,25 volt. Et en inversant les valeurs de R1 et R2, soit R1 = 250 ohms et R2 = 750 ohms, S1 aura une tension de 3,75 volts. Vous trouverez facilement sur Internet des sites qui proposent des outils pour calculer les valeurs pour les résistances d'un pont diviseur de tension.

Dans le cas qui nous intéresse, nous voulons être en mesure de faire varier la tension sur S1. Évidemment, il ne serait par très pratique de devoir changer les résistances de notre clavier binaire directement en cours d'utilisation. Or, c'est ici que le potentiomètre trouve sa place.



Le potentiomètre est en fait une résistance variable à trois bornes. Celle du centre est reliée à un curseur qui parcourt une surface résistante, qui a à ses extrémités les deux autres bornes. Ainsi, lorsque le curseur se retrouve en plein centre (dans le cas d'un potentiomètre linéaire), la résistance entre le curseur et la borne de droite est égale à la résistance entre le curseur et la borne de gauche. C'est le cas équivalent à notre premier exemple ci-dessus. Vous comprenez maintenant qu'en déplaçant le curseur, nous pourrons faire varier la tension sur la broche numéro 2 du microcontrôleur, soit AN0, que nous utiliserons pour ce tutoriel. Notez que n'importe quelle autre broche ANx, à l'exception de AN5 et AN7 que nous utilisons respectivement en tant que RE0 et RE2 pour deux de nos boutons, de même que AN8 et AN10 que nous utilisons respectivement tant que RB2 et RB1 pour le témoin lumineux du logiciel interactif et l'accusé de réception. Reportez-vous à la photo de la page 44 pour la planche de montage.

Programmation du microcontrôleur

Avant d'apporter des modifications au fichier « main.c », nous allons ajouter quelques définitions dans le fichier « HardwareProfile.h ». Nous aurons CONVERT_AD_LIBRE qui nous permettra de savoir si le convertisseur est prêt à travailler, LIRE_POTENTIOMETRE() qui permettra de lancer la conversion, RESULTAT_CONVERSION qui nous permettra de récupérer le résultat lorsque le convertisseur sera à nouveau libre. Finalement, on définit un code pour identifier un envoi analogique.

```
	// Convertisseur analogique/numérique
	#define CONVERT_AD_LIBRE		!ADCON0bits.GO_DONE
	#define LIRE_POTENTIOMETRE()		ADCON0bits.GO_DONE = 1
	#define RESULTAT_CONVERSION		ADRESH
	#define CODE_ENVOI_ANALOGIQUE		0xFF			// Choix arbitraire
```

La première modification que nous apporterons au code de « main.c » est bien entendu l'initialisation. Nous allons donc définir la broche no.2, soit AN0, comme étant notre entrée analogique. Nous allons également mettre AN0 en mode analogique, ce qui nous fera modifier la valeur de ADCON1 que nous avions précédemment. Finalement, nous allons établir la conversion sur 8 bits. Les deux autres options dans le code ci-dessous touchent des options avancées que nous ne verrons pas ici.

```
	// Choisir le canal AN0 (bits <5:2> = 0000)
	// et activer le convertisseur A/D (bit <0> = 1)
	ADCON0 = 0b00000001;

	// Mettre toutes les entrées en numérique,
	// sauf AN0 (bits <3:0> = 1110)
	ADCON1 = 0b00001110;

	// Convertir sur 8 bits, soit alignement à gauche (bit 7 = 0)
	// Temps d'acquisition à 4 Tad (bits <5:3> = 010)
	// Fonctionnement pour 48 MHz avec FOSC/64 (bits <2:0> = 110)
	ADCON2 = 0b00010110;
```

La seconde chose à faire est maintenant d'aller lancer la conversion. Mais avant, nous aurons besoin d'un compteur qui nous permettra de procéder à une conversion (et une transmission) seulement une fois toute les dix itérations de la boucle infinie. Il nous faut donc ajouter la ligne « `unsigned int cptr2;` » tout juste en dessous de la déclaration du compteur « `cptr1` ». De plus, nous ferons une petite pause de quelques millisecondes après la transmission USB. Pour ce faire, nous ajouterons le code ci-dessous juste après la fonction « `void pauseParBoucle (void)` ».

```
void miniPauseParBoucle (void)
{
	cptr1 = 2000;
	do {
		cptr1--;
```

```
	} while(cptr1);
}
```

Voilà! Il ne reste plus qu'à ajouter le code ci-dessous à la boucle infinie de la fonction principale « `main()` », soit tout juste après la vérification de l'état du « `BOUTON_EFFACER` ».

```
		} else if (CONVERT_AD_LIBRE) {
			if (cptr2 > 0) {
				cptr2--;
			} else {
				LIRE_POTENTIOMETRE();
				// Attendre que la conversion analogique-->numérique soit terminée
				while (!CONVERT_AD_LIBRE);
				ToSendDataBuffer[2] = CODE_ENVOI_ANALOGIQUE;
				// Récupérer la valeur
				ToSendDataBuffer[3] = RESULTAT_CONVERSION;
				// Transmettre sur le port USB
				if(!HIDTxHandleBusy(USBInHandle)) {
					USBInHandle = HIDTxPacket(HID_EP,(BYTE*)&ToSendDataBuffer[0],64);
				}
				miniPauseParBoucle();
				cptr2 = 10;
			}
```

Voilà! C'est terminé! Il ne reste plus qu'à compiler, puis à vous amuser.