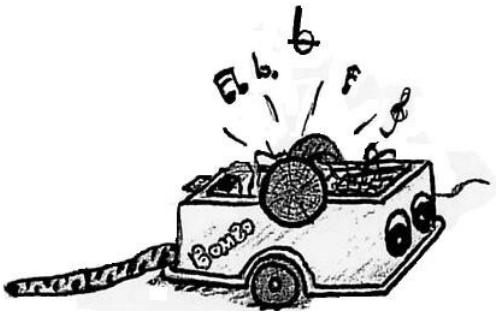
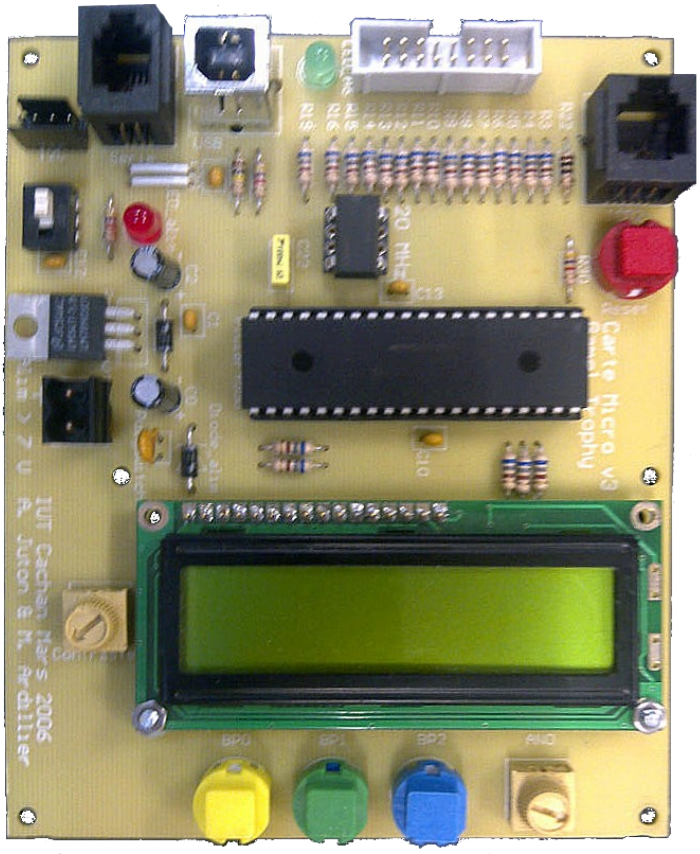


Carte Micro-contrôleur

Gamel Trophy

Guide de Mise en œuvre



Source Anthony JUTON – Version 4.0

Modification Yves GUINAND – Version 2013 et 2014

Modification Joëlle MAILLEFERT – Version 2014

Sommaire

1 INTRODUCTION

La carte micro-contrôleur Gamel Trophy est utilisée en projet de premier semestre et en TP d'informatique industrielle à l'IUT.

Elle est réalisée autour d'un micro-contrôleur Microchip PIC18F4550. Ce micro-contrôleur 8 bits est issu d'une famille très répandue : 2 milliards d'exemplaires vendus par Microchip, 1er fabricant mondial de micro-contrôleurs 8 bits.
De tels micro-contrôleurs sont employés dans l'informatique embarquée des automobiles, de l'électroménager, des télécoms et des systèmes industriels.

Dans la gamme de Microchip, le micro-contrôleur PIC18F4550 fut choisi pour les raisons suivantes :
⇨ Issu de la famille 18Fxxx, c'est un micro-contrôleur rapide (jusqu'à 12 MIPS), possédant des convertisseurs analogique-numérique, des sorties PWM, une liaison série USART, une liaison I2C, une liaison SPI et 4 timers.
⇨ Il comprend un périphérique USB. L'utilisation de la connexion USB permet d'alimenter la carte micro-contrôleur par le bus USB et de le programmer avec un bootloader, ce qui évite d'avoir à utiliser un programmateur.
⇨ Il est disponible en boîtier DIP, ce qui facilite la conception de la carte et le remplacement du micro-contrôleur en cas de panne.
⇨ Les outils de développement fournis par Microchip sont fiables, relativement bon marché, puissants et faciles à utiliser.

Nous utiliserons les outils de développement suivants :
⇨ Environnement de développement *MPLABX IDE*
⇨ Compilateur *MPLAB XC8*
⇨ Programmateur / Débogueur *MPLAB PicKit3*

Ce guide de mise en œuvre vous présente les différents périphériques internes ou externes disponibles sur la carte et les fonctions permettant de les utiliser. Vous y trouverez aussi un tutoriel pour l'utilisation de la chaîne de développement.

2 Caractéristiques générales

2.1 Alimentation

Vous pouvez alimenter la carte avec une alimentation externe ou à partir du bus USB. Une LED rouge indique le bon fonctionnement de l'alimentation 5V.

⇨ L'alimentation externe est une tension continue supérieure à 8V (jusqu'à 18V). Cette tension est ensuite régulée pour fournir 5V au micro-contrôleur et aux différents périphériques. Typiquement la carte consomme 50mA.

⇨ Le bus USB fournit une tension de 5V. Cette tension est moins précise que celle en sortie du régulateur.

2.2 Programmation

On programme le micro-contrôleur par le connecteur RJ12 présent à côté du bouton Reset. Il est relié aux broches MCLR, PGC et PGD pour la programmation et le débogage (mode pas-à-pas, lecture des variables pendant le fonctionnement,…). Cf chapitre 10. Un bouton Reset (bouton poussoir rouge) permet, en mode autonome, de redémarrer le programme. Son état n'est pas disponible pour une utilisation par votre programme.

Les outils permettant de programmer cette carte, MPLABX IDE et MPLAB XC8 Compiler (version sans optimisation) ainsi que de nombreux documents sont disponibles sur le site de Microchip (www.microchip.com). Les documents réalisés à l'IUT (tutoriel, schéma de la carte, bibliothèques…) sont disponibles dans le dossier Y:\COMMUN_E&R_S1_S2\Microchip.

2.3 Périphériques internes du micro-contrôleur

La carte est équipée d'un micro-contrôleur PIC18F4550 ayant comme périphériques internes :

⇨ **4 timers**. Certains timers peuvent être utilisés comme compteurs rapides avec une horloge externe. Deux entrées de comptage rapide sont disponibles sur le connecteur I/O (T0CKI et T13CKI). cf chapitre 7 pour la mise en œuvre du timer1.

⇨ **Un convertisseur analogique/numérique** 10 bits. 7 entrées analogiques sont disponibles sur le connecteur I/O et une entrée analogique est reliée au potentiomètre AN0. cf chapitre 7 pour leur mise en œuvre.

⇨ **Deux sorties PWM** associées au timer2. Elles sont disponibles sur le connecteur I/O. cf chapitre 7 pour leur mise en œuvre.

⇨ **Une liaison série** USART disponible sur le connecteur RJ9. Attention, pour simplifier la carte, cette liaison série est en niveaux logiques 0-5V. Si on veut utiliser une liaison RS232 (niveaux logiques +12V et -12V), il faut donc utiliser un câble adaptant les niveaux des signaux. cf chapitre 9 pour la mise en œuvre de cette liaison.

⇨ **Une liaison I2C** disponible sur le connecteur 4 points. cf chapitre 9 pour la mise en œuvre de la liaison.

⇨ **Une liaison USB** disponible sur le connecteur USB B.

De plus, 3 interruptions externes sont disponibles sur le connecteur I/O, une liaison SPI, un oscillateur interne. cf documentation Microchip pour leur mise en œuvre.
Le micro-contrôleur dispose pour les données d'une mémoire RAM (rapide, mais volatile, 2 Ko octets) et d'une mémoire EEPROM (non volatile mais très lente en écriture, 256 octets). La mémoire programme quant à elle est une mémoire Flash (non volatile, 32 Ko, soit 16384 instructions).

2.4 Périphériques externes

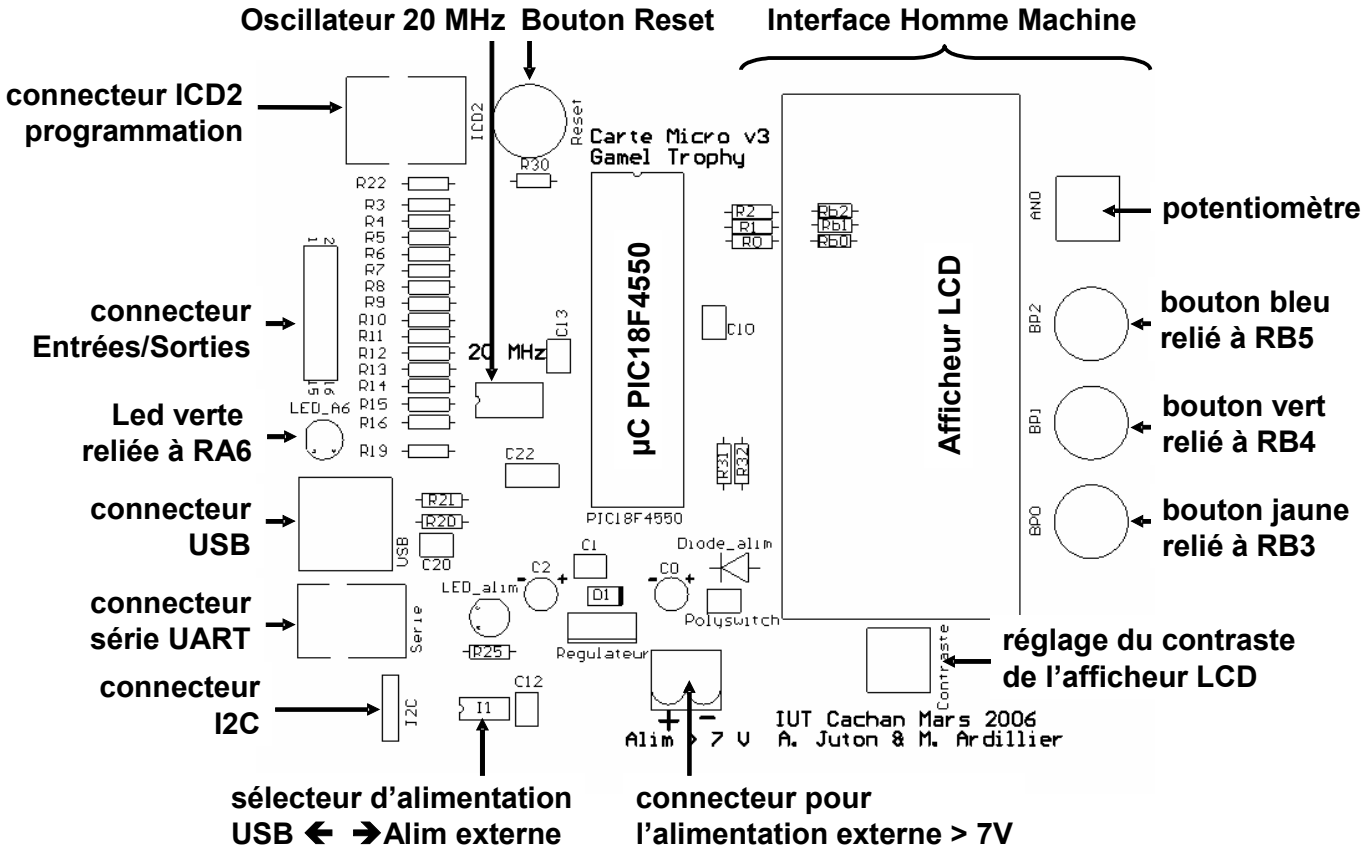
Sur la carte, ont été ajoutés les périphériques externes suivants :
⇨ **Un afficheur LCD** 2x16 caractères alphanumériques câblé sur le port D, avec son potentiomètre de contraste. Cf chapitre 6 pour sa mise en œuvre.
⇨ **3 boutons poussoirs** BP0, BP1 et BP2 câblés sur les entrées Tout Ou Rien B3, B4 et B5 du micro-contrôleur. Cf chapitre 5 pour leur mise en œuvre.
⇨ **Un potentiomètre** délivrant une tension de 0 à 5V câblé sur l'entrée analogique AN0 du micro-contrôleur. Cf chapitre 5 pour sa mise en œuvre.
⇨ **Une LED verte** connectée à la sortie Tout Ou Rien A6 du micro-contrôleur. Cf chapitre 5 pour sa mise en œuvre.

Sur la carte, un oscillateur à 20 MHz dont la fréquence est multipliée en interne permet d'utiliser l'USB Full Speed (nécessité d'une fréquence d'horloge à 96 MHz pour un transfert théorique à 12 Mbps). Avec les paramètres par défaut de la carte, le fonctionnement du cœur du micro-contrôleur **est cadencé à 48 MHz** (un cycle instruction nécessite 4 tops horloge donc 83.3ns).

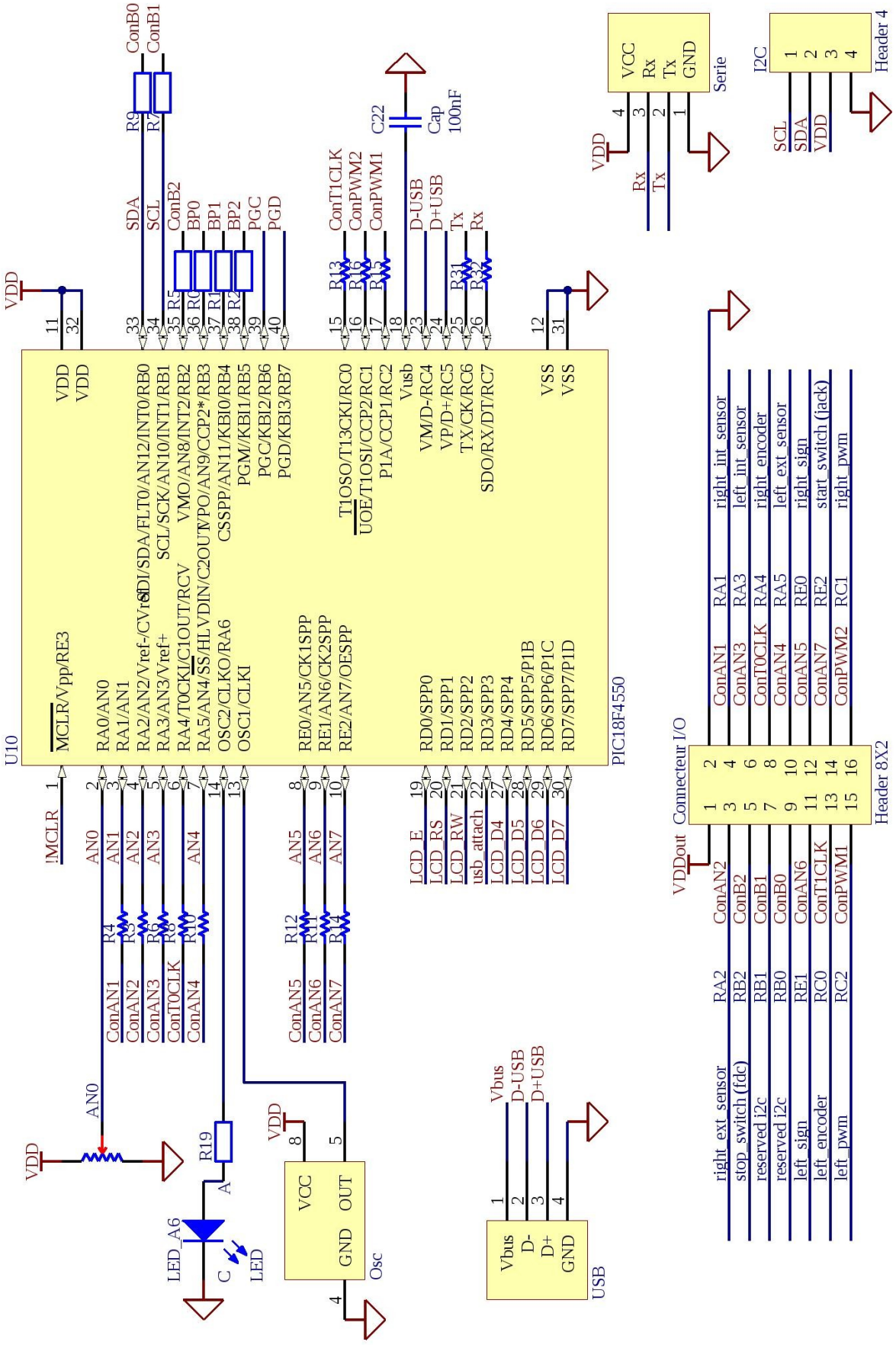
3 SCHÉMAS

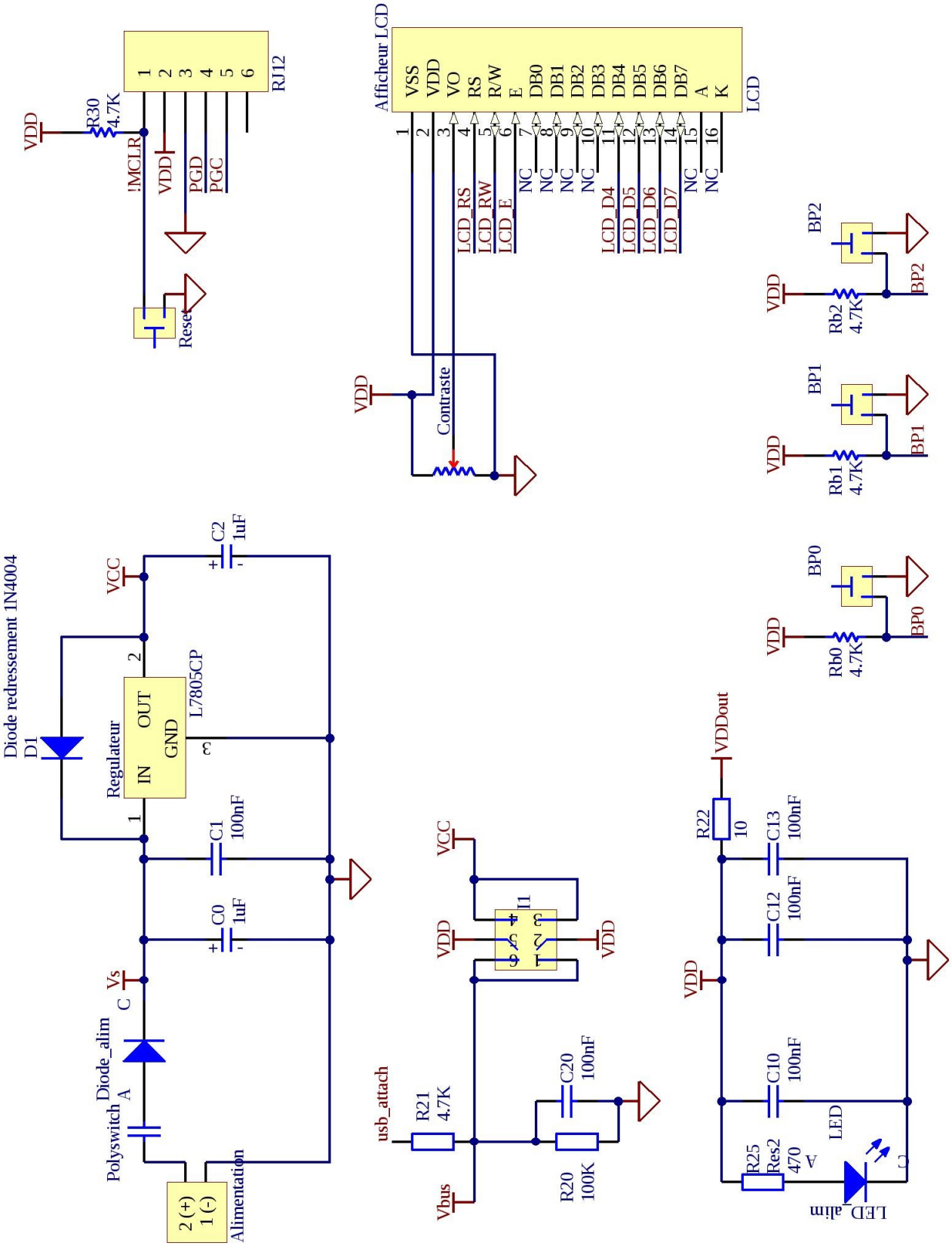
Les schémas présentés ici sont issus d'un projet *Altium Designer* disponible dans le dossier Y:\COMMUN_E&R_S1_S2\Microchip\Carte_micro_v3

3.1 SCHÉMA D'IMPLANTATION DES COMPOSANTS



3.2 SCHÉMA DE CÂBLAGE DE LA CARTE.




RJ12
R30
4.7K
VDD
!MCLR
PGD
PGC
Reset
Afficheur LCD
LCD
VSS
VDD
VO
RS
R/W
E
DB0
DB1
DB2
DB3
DB4
DB5
DB6
DB7
A
K
LCD_RS
LCD_RW
LCD_E
LCD_D4
LCD_D5
LCD_D6
LCD_D7
Contraste
BP2
Rb2
4.7K
BP1
Rb1
4.7K
BP0
Rb0
4.7K
Diode redressement 1N4004
D1
VCC
C2
1uF
Regulateur
L7805CP
IN
OUT
GND
C1
100nF
C0
1uF
Vs
Diode_alim
Polyswitch
Alimentation
2 (+)
1 (-)
usb_attach
R21
4.7K
Vbus
C20
100nF
R20
100K
VDDout
R22
10
C13
100nF
C12
100nF
C10
100nF
R25
Res2
470
LED
LED_alim

4 Programmation de la carte – généralités

Ce chapitre vous présente les quelques connaissances indispensables avant de commencer un programme.

4.1 Configuration du micro-contrôleur

Pour s'adapter au mieux aux différentes applications auxquelles est destiné le micro-contrôleur PIC18F4550, notamment en terme de consommation d'énergie, de nombreuses configurations sont possibles. Vous est présentée ici une configuration de base, largement suffisante pour la majorité des applications. Les curieux pourront trouver plus d'informations sur les configurations dans la documentation technique du PIC18F4550.
La configuration s'effectue dans une zone particulière de la mémoire au moyen d'instructions spéciales de la forme :

```
#pragma config
```

Les différentes valeurs que peuvent prendre les registres de configuration sont indiquées dans le document *MPLAB_XC8_Starter_Guide.pdf*
Pour simplifier la création d'un projet, la configuration par défaut (horloge interne à 48 MHz, A6 en sortie, mode debug, B3, B4 et B5 en entrées TOR…) peut être faite en ajoutant **le fichier *iut_init.c* à votre projet.**

Vous devez inclure le fichier d'en-tête de *xc.h* en tête de programme.
D'autres étapes de configuration doivent se faire dans le code du programme, juste après les définitions des variables.
Vous y mettrez la configuration des entrées/sorties et les initialisations de périphériques (lcd_init(),…

```
#include <xc.h>

void main()
{
      char…  // déclaration/définition des variables

      lcd_init();
      …
}
```

4.2 Configuration des entrées/sorties Tout ou Rien

En plus de la configuration des périphériques internes et externes, en début de programme et parfois pendant le programme, vous aurez à configurer aussi les entrées et sorties Tout Ou Rien.
Pour cela, on utilise les registres TRISx où x est le nom du port. Un bit à 0 correspond à la broche en sortie et un bit à 1 à la broche en entrée.
Exemple pour le PORTA.

A7	A6	A5	A4	A3	A2	A1	A0

Si on veut placer A6 en sortie (broche reliée à la Led verte), il faut mettre le bit 6 de TRISA à 0.

Deux méthodes sont possibles pour cela :

⇨ soit on travaille avec un bit :

```
TRISAbits.TRISA6 = 0;
```

⇨ soit on travaille avec des masques, surtout si on a plusieurs broches à configurer, sans changer l'état des autres bits :

```
TRISA = TRISA & OxBF;
```

Remarques :

- Les registres du micro-contrôleur sont accessibles bit à bit. Par exemple, pour accéder à l'octet du port A, on utilise `PORTA` et pour accéder au port A bit à bit, on utilise `PORTAbits`.
- Le compilateur MPLAB XC8 permet d'écrire des constantes en binaire avec le préfixe `0b`. Par exemple, `0xBF` peut s'écrire `0b10111111`.

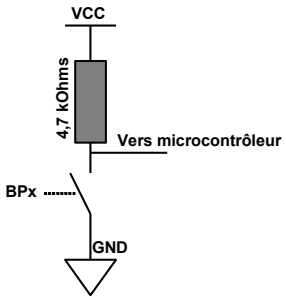
5 Boutons, Potentiomètre et LED

Pour les TPs et les projets, la carte dispose d'une très modeste interface homme-machine composée de :

- trois boutons poussoirs
- un potentiomètre
- une LED
- un afficheur LCD

5.1 Boutons poussoirs

Comme indiqué sur le schéma du chapitre 3, Les boutons poussoirs sont câblés de la manière suivante :



Lorsque le bouton n'est pas enfoncé, il se comporte comme un interrupteur ouvert. Grâce à la résistance de pull-up, le signal reçu par le micro-contrôleur sera VCC, ici 5V. Les entrées du micro-contrôleur n'absorbant pas de courant, la résistance n'est parcourue par aucun courant.
Lorsque le bouton est enfoncé, il se comporte comme un interrupteur fermé, reliant ainsi la masse à l'entrée du micro-contrôleur qui reçoit alors 0V. La résistance de pull-up est parcourue par un courant VCC / R.
Les boutons sont connectés aux broches suivantes :

- le bouton BP0 est relié à la broche B3
- le bouton BP1 est relié à la broche B4
- le bouton BP2 est relié à la broche B5

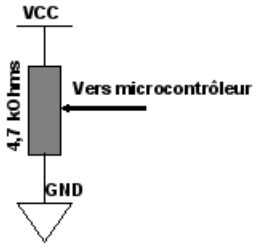
Exemple pour lire l'état d'un bouton poussoir (par exemple BP0) :

```
char BP0=0;
TRISB = TRISB | 0x08;   // configuration en entrée de RB3
...
BP0 = PORTBbits.RB3;    // à chaque lecture de l'état logique de B3
```

Pour lire plusieurs boutons à la fois, il est plus intéressant d'utiliser des masques (cf cours II1).
Un changement d'état sur B4 ou B5 peut-être configuré pour provoquer une interruption.

5.2 POTENTIOMÈTRE

A côté des boutons poussoirs, on trouve un potentiomètre relié à l'entrée analogique AN0 (occupant la broche reliée à l'entrée Tout Ou Rien RA0). Il est câblé de la manière ci-contre.



L'entrée du micro-contrôleur n'absorbant pas de courant, le montage est un simple diviseur de tension. Pour une position *pos* du curseur du potentiomètre, l'entrée analogique du micro-contrôleur reçoit donc la tension suivante :

$$V_{microcontrôleur} = pos \times VCC = pos \times 5 \text{ Volts}$$

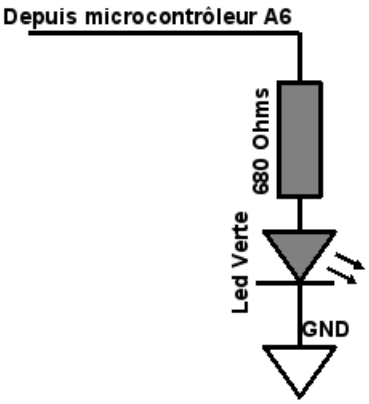
La lecture de la valeur du potentiomètre utilise le convertisseur analogique numérique 10 bits. Pour plus de détails, on se référera au chapitre 7 qui traite de la bibliothèque associée au convertisseur analogique-numérique (ADC) interne.

Exemple pour la lecture de la valeur du potentiomètre :

```
#include "iut_adc.h"
...
int potana=0;
adc_init(0);                    // initialisation de l'ADC
...
potana = adc_read(0) ;  // à chaque lecture de la valeur AN0
```

5.3 LED

Une led verte, appelée LED_A6, est reliée à la sortie Tout Ou Rien A6 du micro-contrôleur. Elle est câblée de la manière ci-contre.



Si A6 = 1, le micro-contrôleur impose 5 V en sortie, le courant passe alors dans la résistance et la LED : la LED s'allume. La résistance impose un courant $I_D = (VCC-V_D) / R \sim 6$ mA. C'est le micro-contrôleur qui fournit le courant nécessaire. Une sortie Tout Ou Rien peut fournir jusqu'à 25 mA.
Si A6 = 0, le micro-contrôleur impose 0V, la LED est alors éteinte.
Allumer la LED_A6 revient ainsi à modifier l'état logique de A6.

Exemple pour modifier l'état de la LED_A6 :

```
TRISA = TRISA & 0xBF;   // configuration en sortie de A6
...
PORTAbits.RA6 = 1;      // mise à 1 de A6
...
PORTAbits.RA6 = 0;      // mise à 0 de A6
```

6 AFFICHEUR LCD

Mieux que la LED_A6, vous pouvez utilisez l'afficheur LCD pour afficher des informations du micro-contrôleur pour l'utilisateur. Il est câblé sur le port D, comme indiqué sur le schéma du chapitre 3. L'afficheur LCD est doté de son propre contrôleur qui interprète les ordres transmis par le micro-contrôleur. Un protocole de dialogue est établi par le fabricant du contrôleur de l'afficheur LCD.
Pour simplifier le travail du programmeur, une bibliothèque regroupe les fonctions les plus utiles. Pour utiliser cette bibliothèque, vous devez :

- copier les fichiers *iut_lcd.h* et *iut_lcd.c* dans le dossier de votre projet
- ajouter ces deux fichiers à votre projet
- inclure le fichier d'en-têtes *iut_lcd.h* avec l'instruction `#include "iut_lcd.h"`

void lcd_init(void);

Cette fonction initialise l'écran LCD. Il faut absolument l'appeler dans votre programme avant tout emploi d'une autre fonction de la bibliothèque. Une seule exécution de cette fonction est suffisante.

void lcd_position(char ligne, char colonne);

Cette fonction positionne le curseur pour l'affichage. Le coin supérieur gauche de l'afficheur est considéré à la ligne 0 et à la colonne 0. Au prochain appel d'une fonction d'affichage comme *lcd_printf* ou *lcd_putc*, le premier caractère s'inscrira à l'endroit désigné par *lcd_position*.

void lcd_putc(char lettre);

Cette fonction affiche un caractère à la position du curseur et décale le curseur d'une case vers la droite pour le prochain affichage.
Les caractères spéciaux ci-dessous sont interprétés :

- `'\n'` pour passer à la ligne
- `'\f'` pour effacer l'écran
- `'\b'` pour reculer d'une case

void lcd_clear(void);

Efface l'écran.

void lcd_printf(const rom char *f, ...);

Cette fonction permet d'effectuer des affichages sur l'écran LCD avec la même syntaxe que la fonction *printf* usuelle du langage C. Rappel des formats les plus couramment utilisés :

- `%d` pour une afficher en base 10 une variable de type `int`
- `%x` pour afficher en hexadécimal une variable de type `int`
- `%f` pour afficher une variable de type `float`
- `%s` pour afficher une chaîne de caractères

Exemple pour l'affichage de "arthur" en bas à droite de l'écran :

```
#include <xc.h>
#include "iut_lcd.h"

void main(void)
{
    lcd_init();               // initialisation de l'afficheur lcd
...
    lcd_position(1, 10);      // positionnement du curseur
    lcd_printf("arthur");     // écriture du texte
...
}
```

Remarques

- Pour que l'affichage d'un nombre soit correct, il est préférable de prévoir le nombre maximum de caractères occupés. Par exemple, une variable de type `int` qui prendrait des valeurs entre 0 et 1023 occupera entre 1 et 4 caractères. On utilisera donc un format qui indique l'occupation de 4 caractères `%4d`. De même, une variable de type `float` comprise entre -1 et 1 et dont on souhaite afficher 2 décimales occupera 5 caractères (1 pour le signe, 1 pour la partie entière, 1 pour le point et 2 pour les décimales). On utilisera le format `%5.2f` (5 caractères et 2 décimales).
- Aucun format de la fonction *lcd_printf* ne permet d'afficher la valeur d'un nombre entier de 8 bits (type `char`), il faut d'abord le transformer en entier de 16 bits (type `int`). Pour cela, on fait ce qu'on appelle un *cast*, c'est-à-dire une conversion explicite de type, en rajoutant entre parenthèses le type voulu. Cela a pour effet de créer un `int` n'ayant que des 0 dans l'octet de poids fort.

  Exemple de syntaxe :

```
char c=10;
...
lcd_printf("%3d", (int)c);
```

- L'écriture sur l'afficheur LCD est une opération relativement lente. L'initialisation prend 25 ms et l'écriture d'un caractère 100 µs environ (à comparer avec les 0,1 µs que dure l'exécution d'une instruction simple). Il faut en tenir compte si votre programme doit effectuer une tâche rapidement ou à une cadence fixe (asservissement, tâche d'interruption).
- Dans la source de la bibliothèque *iut_lcd.c*, vous pouvez lire comment ces fonctions sont écrites en C et ainsi regarder quel dialogue s'établit entre le micro-contrôleur et l'afficheur LCD.

7 LES PÉRIPHÉRIQUES INTERNES

Le micro-contrôleur PIC18F4550 a de multiples périphériques internes. Dans ce chapitre, nous vous présentons les 3 périphériques internes les plus utilisés : le convertisseur analogique-numérique, les sorties PWM et les timers.
Les périphériques de communication sont étudiés au chapitre 9.

7.1 LE CONVERTISSEUR ANALOGIQUE-NUMÉRIQUE

Le PIC18F4550 est muni en interne d'un convertisseur analogique-numérique 10 bits. Par multiplexage, celui-ci peut vous renvoyer jusqu'à 13 nombres, images des tensions analogiques présentes sur différentes broches du boîtier. Vous pouvez utiliser celles qui sont disponibles sur le connecteur I/O : AN1 à AN7. La première, AN0, est reliée au potentiomètre nommé AN0.
Pour utiliser le convertisseur simplement, une bibliothèque regroupe les fonctions dont vous avez besoin. Pour utiliser cette bibliothèque, vous devez :

- copier les fichiers *iut_adc.h* et *iut_adc.c* dans le dossier de votre projet
- ajouter ces deux fichiers à votre projet
- inclure le fichier d'en-têtes *iut_adc.h* avec l'instruction `#include "iut_adc.h"`

void adc_init(char numero_dernier_canal);

Cette fonction initialise le convertisseur analogique-numérique (temps d'acquisition et temps de conversion). Vous pouvez utiliser le nombre d'entrées analogiques que vous souhaitez, à condition de les prendre successives, à partir de AN0 et de donner en paramètre de la fonction *adc_init* le numéro du dernier canal que vous utilisez.

int adc_read(char numero_canal);

Cette fonction déclenche la conversion analogique-numérique sur le canal indiqué en paramètre. Elle renvoie ensuite cette valeur sous forme d'un entier 16 bits. Seuls les 10 bits de poids faible sont utilisés (les 6 bits de poids fort sont toujours nuls). Le temps de conversion est d'environ 25µs.

Exemple effectuant la lecture de la valeur du potentiomètre AN0 (relié à AN0) et la stockant dans un entier :

```
#include <xc.h>
#include "iut_adc.h"
…
void main(void)
{
    int potana = 0; // définition de potana
    adc_init(0);    // initialisation du convertisseur
    while(1)
    {
        potana = adc_read(0); // lecture de AN0
        ...
    }
}
```

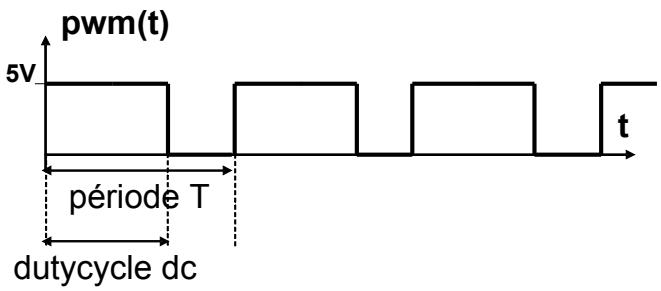
Remarque

La bibliothèque donnée ne vous permet pas de passer d'une entrée configurée en analogique à une entrée configurée en Tout Ou Rien en cours de programme. Elle ne vous permet pas non plus d'utiliser les interruptions, ou de modifier les temps d'acquisition, les références... Vous trouverez toutes les caractéristiques du convertisseur et le rôle de chaque registre le concernant au chapitre 19 de la documentation technique du micro-contrôleur et des bibliothèques plus complètes et plus complexes dans le chapitre 2.2 du guide d'utilisation des bibliothèques *MPLAB_XC8_Peripheral_Libraries .*

7.2 Les PWMs

Le PIC18F4550 est muni en interne de deux blocs générant des PWMs, nommées PWM1 et PWM2. La PWM (pulse width modulation) est un signal carré dont le rapport cyclique (rapport entre la durée du niveau haut et la période du signal) peut varier. C'est un signal très utilisé dans la commande de moteur ou la conversion numérique-analogique.

Les blocs PWMs utilisent pour fonctionner le timer2.



Pour utiliser les sorties PWM présentes sur les broches C2 (PWM1) et C1 (PWM2), vous utiliserez une bibliothèque qui regroupe les fonctions dont vous avez besoin. Pour utiliser cette bibliothèque, vous devez :

- copier les fichiers *iut_pwm.h* et *iut_pwm.c* dans le dossier de votre projet
- ajouter ces deux fichiers à votre projet
- inclure le fichier d'en-têtes *iut_pwm.h* avec l'instruction `#include "iut_pwm.h"`

void pwm_init(char period, char nb_canaux);

Cette fonction initialise les sorties PWMs. C'est cette fonction qui configure le timer2 pour le fonctionnement des PWMs.

Si nb_canaux = 2 alors les deux sorties sont activées, sinon seule la sortie PWM1 est activée. Vous entrerez en paramètre de cette fonction la période de vos signaux PWMs, en nombre de pas de **333 ns**. Le paramètre *period* est un entier de 8 bits. Pour Fosc = 48 MHz (en utilisant *iut_init.c*), la fréquence de fonctionnement des PWMs est donnée par la formule :

$$f = (3.10^6 / (period+1))$$

Par exemple, si *period* = 149 alors la fréquence f sera de 20 kHz.

void pwm_setdc1(unsigned int cycles_etat_haut);

Cette fonction définit le rapport cyclique de PWM1 (C2).

void pwm_setdc2(unsigned int cycles_etat_haut);

Cette fonction définit le rapport cyclique de PWM2 (C1).

Pour ces deux dernières fonctions, *cycles_etat_haut* représente le temps à l'état haut, en nombre de pas de **83,3 ns**. *cycles_etat_haut* est un entier de 10 bits.

On peut calculer le rapport cyclique avec la formule :

rapport cyclique = *cycles_etat_haut* / **(** 4 x (*period* + 1) **)**

Exemple permettant d'obtenir deux signaux PWMs à 20 kHz, de rapport cyclique 0,25 pour PWM1 et de 0,75 pour PWM2 :

```
#include <xc.h>
#include iut_pwm.h

void main(void)
{
    pwm_init(149, 2);   // initialisation de la période 50µs
                        // pour les deux canaux
...
    pwm_setdc1(150);    // 0,25 pour PWM1 (broche C2)
    pwm_setdc2(450);    // 0,75 pour PWM2 (broche C1)
...
}
```

Remarque :

- Les broches configurées en PWM par *pwm_init* restent en PWM tout le programme.
- *period* a une résolution de 333 ns et *cycles_etat_haut* une résolution de 83,3 ns.
- Si le temps à niveau haut est supérieur à la période alors le signal PWM est toujours à 1 (rapport cyclique maximum de 100%).

7.3 TIMERS

Le PIC18F4550 comporte 4 timers (de timer0 à timer3), de 8 ou 16 bits (seul le timer2 est un timer 8 bits). Un timer est en fait un compteur que l'on peut connecter à l'horloge interne du micro-contrôleur. Il compte alors le nombre de cycle instructions (un cycle instructions dure 83,3 ns). Les timers 0 et 1 peuvent aussi être connectés à une horloge externe (reliée via les broches T0CKI et T1CKI). Ils comptent dans ce cas le nombre d'occurrence d'un événement (front montant ou front descendant) se produisant sur cette broche.

Un timer 16 bits peut compter de 0 à 65535 (2^{16}-1). Pour compter un nombre plus important d'événements, on compte de 2 en 2, de 4 en 4, de 8 en 8… ce coefficient de pré-division de l'horloge est appelé pré-scalaire.

Le fonctionnement des timers est expliqué en détail dans la documentation technique du micro-contrôleur (ch 11 à 14) et Microchip fournit des bibliothèques permettant d'utiliser ces timers. Pour utiliser ces bibliothèques, consultez le document *MPLAB_XC8_Peripheral_Libraries.pdf* et incluez le fichier d'en-tête *plib/timers.h*. (On mettra des <> pour indiquer au compilateur que ce fichier est dans le dossier des bibliothèques du compilateur et non dans votre dossier de projet).

Voici quelques mots pour expliquer l'utilisation du timer 0 avec la bibliothèque timers.

void OpenTimer0(unsigned char config);

Cette fonction permet de configurer et de démarrer le timer. Le paramètre de configuration est une association de différents masques validant chacun une caractéristique :

Enable Timer0 Interrupt: TIMER_INT_OFF pour ne pas utiliser les interruptions
TIMER_INT_ON Interrupt enabled
TIMER_INT_OFF Interrupt disabled
Timer Width: T0_16BIT pour utiliser le mode 16 bits du timer
T0_8BIT 8-bit mode
T0_16BIT 16-bit mode
Clock Source: Permet de sélectionner l'horloge du timer (horloge interne ou TxCKI)
T0_SOURCE_EXT External clock source (I/O pin)
T0_SOURCE_INT Internal clock source (TOSC)
External Clock Trigger : T0_EDGE_RISE pour compter les fronts montants
T0_EDGE_FALL External clock on falling edge
T0_EDGE_RISE External clock on rising edge
Prescale Value: Permet de choisir le préscalaire.
T0_PS_1_1 1:1 prescale
T0_PS_1_2 1:2 prescale
T0_PS_1_4 1:4 prescale
T0_PS_1_8 1:8 prescale
T0_PS_1_16 1:16 prescale
T0_PS_1_32 1:32 prescale
T0_PS_1_64 1:64 prescale
T0_PS_1_128 1:128 prescale
T0_PS_1_256 1:256 prescale

void WriteTimer0(unsigned int timer);

Cette fonction permet d'écrire la valeur *timer* dans le timer. Ce dernier recommencera alors à compter, à partir de cette valeur.

unsigned int ReadTimer0(void);

Cette fonction permet de lire la valeur actuelle du timer.

Exemple de programme permettant de faire clignoter une led avec une période de 0,5 s.

```
#include <xc.h>
#include <plib/timers.h>

void main(void)
{
    TRISA = 0xBF;                          // A6 en sortie
    OpenTimer0(   TIMER_INT_OFF &
                  T0_16BIT &
                  T0_SOURCE_INT &
                  T0_PS_1_128 );           //config timer
    while(1)
    {
      WriteTimer0(0);                      // mise à 0 du timer0
      while (ReadTimer0()<46893);          // attente 0,5 s
      PORTAbits.RA6 = !PORTAbits.RA6;      // inversion led
    }                                      // end while
}                                          // end main
```

8 Connecteur I/O

Ce connecteur regroupe l'ensemble des entrées/sorties disponibles du micro-contrôleur. Voici le plan du connecteur :

VDD Out	**1**	**2**	GND
AN2 ou RA2	**3**	**4**	AN1 ou RA1
RB2 ou INT2 ou AN8	**5**	**6**	AN3 ou RA3
RB1 ou INT1	**7**	**8**	RA4 ou T0CKI
RB0 ou INT0	**9**	**10**	AN4 ou RA5
AN6 ou RE1	**11**	**12**	AN5 ou RE0
T13CKI ou RC0	**13**	**14**	AN7 ou RE2
PWM1 ou RC2	**15**	**16**	PWM2 ou RC1

1	La broche 1 permet de fournir une alimentation de 5V à une autre carte avec un courant limité (moins de 50 mA). Si la carte que vous désirez connecter à la carte micro-contrôleur est alimentée directement par la batterie, **il ne faut surtout pas utiliser cette broche** du connecteur.
2	La broche 2 est la masse. Si la carte que vous désirez connecter à la carte micro-contrôleur est alimentée directement par la même batterie que la carte micro-contrôleur, les masses sont reliées via la masse de la batterie. Il est alors déconseillé de relier une nouvelle fois les masses via le connecteur.
3, **4**, **6**, **10**, **11**, **12**, **14**	Ces broches du connecteur sont reliées à des broches du micro-contrôleur qui peuvent être configurées individuellement comme des entrées analogiques ou comme des entrées/sorties Tout ou Rien. Par défaut, ce sont des entrées analogiques du micro-contrôleur.
5	Cette broche du connecteur est reliée à une broche du micro-contrôleur qui peut être au choix configurée comme une entrée analogique (c'est le cas par défaut), comme une entrée interruption (vous verrez cela plus tard) ou comme un entrée/sortie Tout ou Rien.
7 et **9**	Ces broches sont reliées à des entrées/sorties Tout ou Rien du micro-contrôleur (attention, il faut pour cela avoir configuré les entrées analogiques). Elles peuvent servir d'entrées d'interruption. Attention, on ne peut utiliser ces deux broches en même temps que le bus I^2C.
8	Cette broche est reliée à une broche du micro-contrôleur qui peut être configurée au choix comme une entrée/sortie Tout ou Rien ou comme entrée comptage rapide du compteur/timer 0.
13	Cette broche est reliée à une broche du micro-contrôleur qui peut être configurée au choix comme une entrée/sortie Tout ou Rien ou comme entrée comptage rapide des compteurs/timers 1 et 3.
15 et **16**	Ces broches sont reliées à des broches du micro-contrôleur qui peuvent être configurées au choix comme des entrées/sorties Tout ou Rien ou comme sortie PWM (Pulse Width Modulation ou Modulation de Largeur d'impulsion) des blocs PWM 1 et 2.

Attention, sur le schéma de la carte (chapitre 3), vous pouvez voir que les entrées/sorties reliées au connecteur sont protégées par des résistances de 680 Ohms. Tant qu'il n'y a pas de courant fourni ou absorbé par le micro-contrôleur (c'est le cas quand la broche est configurée comme entrée du micro-contrôleur), cela ne modifie rien. Par contre, si vous voulez que le micro-contrôleur fournisse du courant (allumage d'une diode led), il vous faudra tenir compte de la chute de tension dans cette résistance.

Une sortie du micro-contrôleur peut fournir jusqu'à 25 mA mais, toutes sorties comprises, le micro-contrôleur ne peut fournir que 200 mA.

9 Périphériques de communication

9.1 Port série RS232

La liaison RS232 est une liaison série (les bits sont envoyés les uns après les autres) asynchrone (l'horloge cadençant la liaison n'est pas transmise.)
Le port série de la carte micro-contrôleur, présent sur le connecteur RJ9 est en niveaux logiques 0 / 5V. Il ne peut donc pas être connecté tel quel à une liaison RS232 de PC (niveaux logiques +12V / -12V). Pour ce faire, il faudra utiliser un câble adapté. Ils sont disponibles au magasin.

On utilise les fonctions de la bibliothèque XC8 pour établir une communication RS232 :

openUSART
putcUSART
putsUSART ou putrsUSART
getcUSART
getsUSART
BusyUSART
DataRdyUSART

Référez-vous à la documentation de la bibliothèque pour comprendre comment les utiliser (cf Ch 12).

9.2 Port I2C

Les liaisons SPI et I2C sont des liaisons séries synchrones : les bits sont envoyés les uns après les autres sur un même fil. Un autre fil sert à transmettre l'horloge cadençant la transmission. L'horloge étant ainsi imposée exactement pour tous, on peut autoriser des débits bien supérieurs à ceux d'une liaison série asynchrone.
La liaison I2C est de plus un bus, c'est-à-dire qu'il permet de relier plus que 2 périphériques entre eux (jusqu'à 127).

Pour utiliser la liaison I2C, on utilise la bibliothèque C18. Référez-vous à la documentation de la bibliothèque pour comprendre comment les utiliser. (cf Ch 12)

9.3 Port USB

Le micro-contrôleur PIC18F4550 est aussi muni d'un périphérique USB qui permet de communiquer avec le PC. Ce port permet notamment de programmer la carte à l'aide du bootloader. Son utilisation pour communiquer avec le PC est plus complexe. Le mode liaison série virtuelle (Classe CDC du protocole USB) est le plus simple à mettre en œuvre. Plus d'informations sont disponibles sur la note d'application AN956 de Microchip. (cf site Web de Microchip).

10 Tutoriel Outils de Développement

Pour développer des applications pour les micro-contrôleurs Microchip, divers outils de développement sont possibles, fournis par Microchip ou par des sociétés tiers.

Nous utiliserons les outils de développement suivants :

⇨ Environnement de développement *MPLAB X IDE*

⇨ Compilateur C *MPLAB XC8*

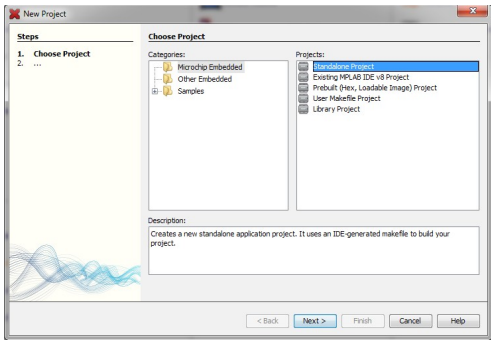
⇨ Programmateur / Débogueur *MPLAB PicKit3*

Pour développer sous MPLAB, vous devez **impérativement** créer un projet, en suivant la démarche ci-dessous.

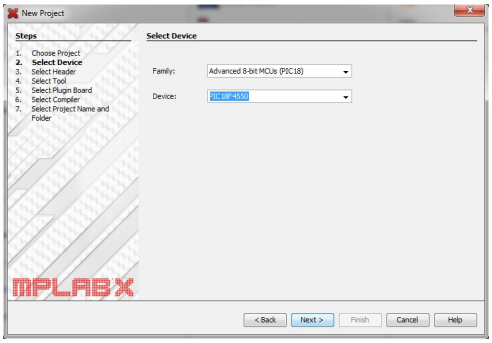
Un projet regroupe un fichier contenant le programme principal (*main*), des bibliothèques à compiler, le tout associé à des bibliothèques du compilateur.

10.1 Création d'un projet et configuration de MPLAB X IDE

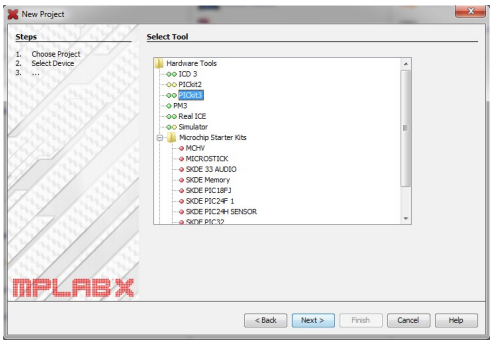
Exécutez MPLAB X et créez un nouveau projet avec le menu *File → New project*



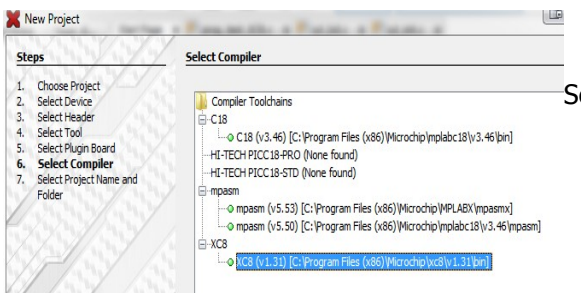
Dans la catégorie *Microchip Embedded*, choisissez *Standalone Project* et cliquez sur *Next*.



Choisissez la famille *Advanced 8-bit MCUs (PIC18)* et sélectionnez le microcontrôleur *PIC18F4550*, puis cliquez sur *Next*.

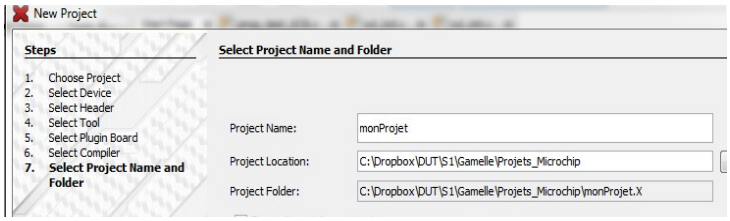


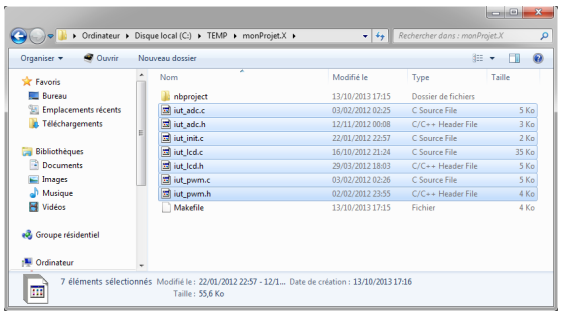
Sélectionnez le programmateur *PicKit3* et cliquez sur *Next*.



Sélectionnez le compilateur *XC8* et cliquez sur *Next*.

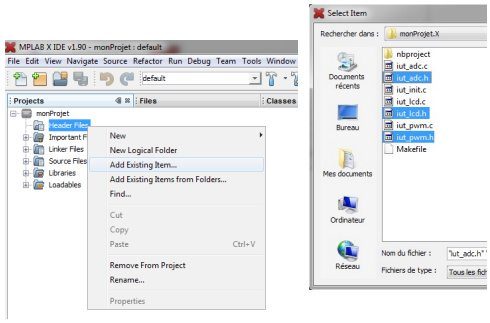
Choisissez un nom de projet (***Project Name***), par exemple ***monProjet.*** Évitez les caractères spéciaux, les accents et les espaces.
Créez un dossier dans votre répertoire de travail, par exemple ***Dossier_Microchip*** comme emplacement de sauvegarde de votre projet (***Project Location***).

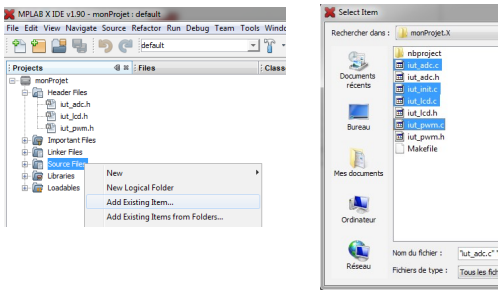




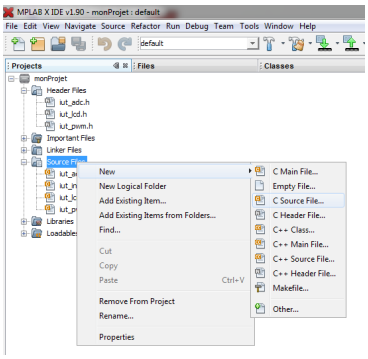
Copiez dans le répertoire du projet, ici ***Dossier_Microchip\monProjet.X,*** les fichiers des bibliothèques de l'IUT pour la carte micro-contrôleur que vous trouverez dans le répertoire :
Y:__E&R_S1_S2_\Microchip\bibliothequesXC8

Dans *MplabX*, effectuez un clic droit sur *Header Files* puis cliquez sur *Add Existing Item*. Ajoutez les trois fichiers avec l'extension ***.h***.

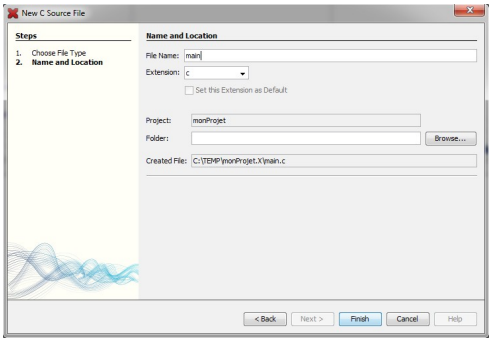




Effectuez un clic droit sur *Source Files* puis cliquez sur *Add Existing Item*. Ajoutez les quatre fichiers avec l'extension ***.c***.

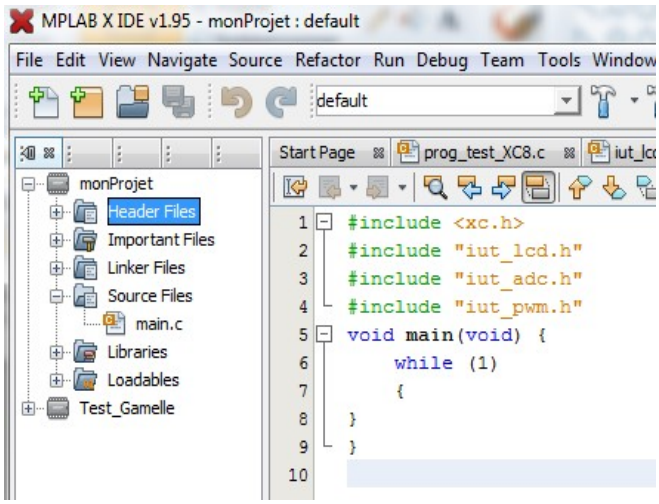


Effectuez un nouveau clic droit sur *Source Files* puis cliquez sur *New → C Source File*.



Choisissez un nom de fichier, par exemple *main*.

Vous pouvez maintenant taper votre premier programme.

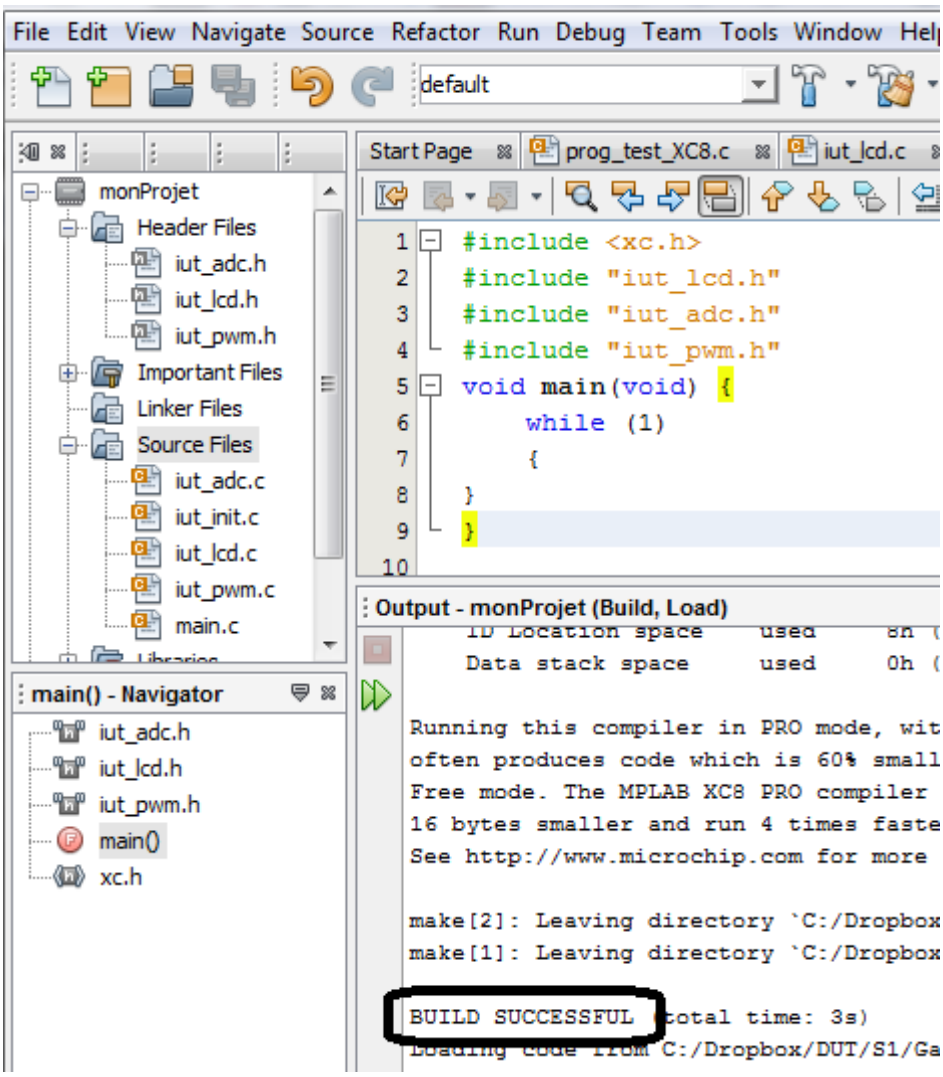



```
#include <xc.h>
#include "iut_lcd.h"
#include "iut_adc.h"
#include "iut_pwm.h"

void main(void) {
    while (1)  {

      }
}
```

Ce programme ne fait strictement rien. Il attend indéfiniment...

Mais après avoir cliqué sur *Build Project* 

vous devez vérifier que vous avez créé correctement votre projet comme ci-dessous :



Sauvegardez votre projet.

Les fichiers des bibliothèques de l'IUT inclus dans le projet ont les rôles suivants :

- iut_init.c configuration du micro-contrôleur (vitesse d'horloge,...)
- iut_lcd.c et iut_lcd.h utilisation simplifiée de l'afficheur LCD
- iut_adc.c et iut_adc.h utilisation simplifiée du convertisseur analogique-numérique
- iut_pwm.c et iut_pwm.h utilisation simplifiée des sorties PWM

10.2 Compilation et construction du projet

Sélectionnez *Run* → *Build Project* pour compiler et linker le projet. S'il existe des messages d'avertissements et d'erreurs, ils apparaîtront dans la fenêtre de sortie (*Output*). Sinon, le message BUILD SUCCESSFUL apparaît dans cette fenêtre de sortie.

10.3 Programmation de la carte microcontrôleur avec PicKit3

Pour programmer la carte microcontrôleur, vous devez brancher le programmateur PicKit3 et la carte microcontrôleur.

Cliquez ensuite sur *Make and Program Device* 

Quand le programme est transféré dans le microcontrôleur, l'exécution doit commencer.

Si le programmateur est déconnecté, la carte microcontrôleur est autonome.

10.4 Débogage du programme avec PicKit3

Le débogage consiste à contrôler l'exécution d'un programme par le PC. Cela regroupe les modes pas-à-pas, la scrutation de variables (watch), et la mise en place de points d'arrêt (breakpoint). La liaison entre le PC et le microcontrôleur est cruciale pour ces actions. Elle se fait par le programmateur PicKit3.

Lorsque votre programme ne fonctionne pas, il est indispensable que vous sachiez utiliser ces outils pour parvenir à retrouver l'erreur dans votre programme ou votre algorithme.

Placement d'un point d'arrêt (breakpoint)

Il suffit de cliquer sur le numéro de la ligne pour créer un point d'arrêt dans votre programme. La ligne est alors surlignée en rouge.

Débogueur

Cliquez sur *Debug Project* 

La carte est programmée en mode débogage et le programme est exécuté.

L'onglet *Variables* permet de visualiser les valeurs des variables.

L'onglet *Breakpoints* permet d'activer ou de désactiver les points d'arrêt.

La barre d'outils du débogueur permet de :

Terminer le débogage

Mettre le programme en pause

Faire un reset du microcontrôleur

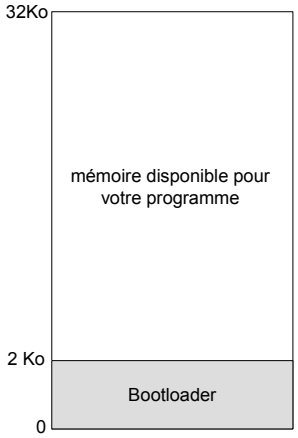
Continuer le débogage

Exécuter en mode pas-à-pas sans entrer dans les fonctions

Exécuter en mode pas-à-pas en entrant dans les fonctions

Exécuter jusqu'au curseur

11 BOOTLOADER



Si vous utilisez la carte micro-contrôleur chez vous, il est possible de la programmer sans programmateur, avec un simple câble USB. Pour cela il suffit d'installer un petit programme appelé ***bootloader*** en mémoire flash.

Vous pouvez copier sur un support personnel tout le répertoire

Y:\COMMUN_E&R_S1_S2\Microchip\Bootloader

11.1 INSTALLER LE BOOTLOADER USB SUR VOTRE CARTE

Avant de pouvoir utiliser le ***bootloader***, il faut avoir installé ce dernier dans la mémoire de votre microcontrôleur à l'aide d'un programmateur PicKit3.

⇨ Pour cela, sous MPLAB X, ouvrez le projet bootloader.X présent dans le dossier

Y:\COMMUN_E&R_S1_S2\Microchip\Bootloader

⇨ Programmez alors simplement le microcontrôleur : *Make and Program Device* 

Le ***bootloader*** est désormais installé sur votre carte. Vous n'avez plus besoin du programmateur.

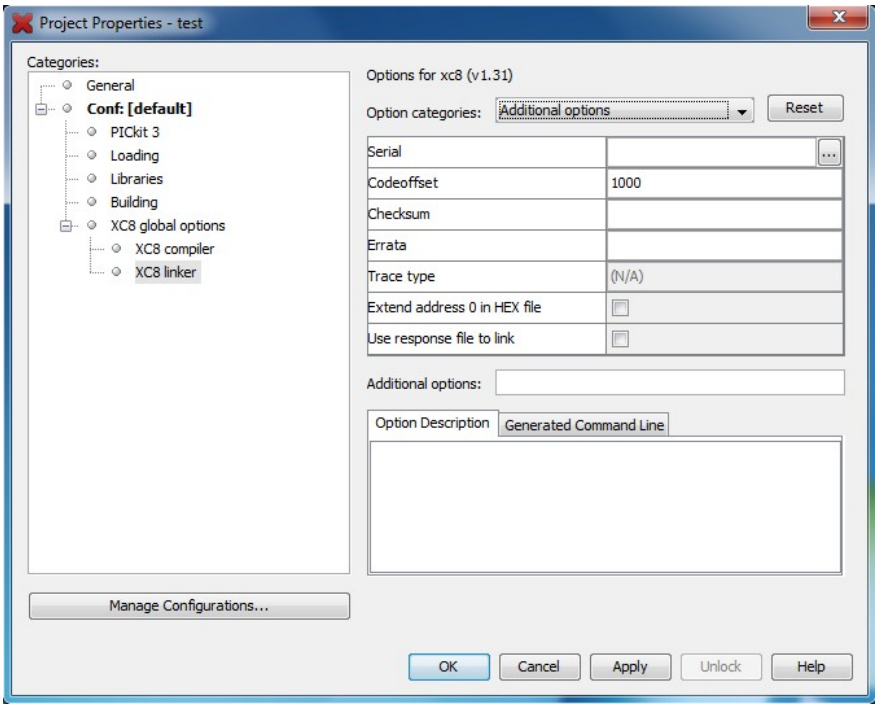
11.2 CRÉER UN PROJET SOUS MPLAB DESTINÉ À ÊTRE ENVOYÉ PAR LE BOOTLOADER

Quelques légères modifications sont à apporter pour que votre projet soit envoyé au microcontrôleur via le ***bootloader***.

⇨ Ajoutez le fichier ***iut_bootloader.c*** à votre projet

⇨ Ajoutez le fichier de linker ***rm18f4550_HID_Bootload.lkr*** à votre projet

Ces deux fichiers se trouvent dans le dossier Y:\COMMUN_E&R_S1_S2\Microchip\Bootloader



Configurer le projet :

- aller dans le menu ***File → Project Properties***
- choisir ***XC8 Linker*** dans les ***Categories***
- choisir ***Additional options*** dans ***Option categories***
- mettre à 1000 le ***Codeoffset***

Ensuite, vous pouvez compiler votre programme normalement.

MPLAB X doit créer un fichier avec l'extension *hex* dans le sous dossier **\dist\default\production** du projet.

11.3 Charger le programme sur la carte

La dernière étape consiste à envoyer le programme sur la carte.

⇨ Branchez le cordon USB (qui peut aussi alimenter la carte) au PC et à la carte. Appuyez sur les boutons RESET (bouton rouge) et BP1 (bouton vert) en même temps puis relâchez le RESET en maintenant BP1 enfoncé. La carte, voyant le bouton BP1 enfoncé dès le démarrage passe en mode bootloader USB et tente de se connecter au PC.

⇨ Lancer le programme ***HIDBootloader (Windows).exe*** qui se trouve dans

Y:\COMMUN_E&R_S1_S2\Microchip\Bootloader\BoatLoader USB Prog

⇨ La carte doit être détectée.

⇨ Chargez le fichier "HEX" de votre programme normalement présent dans le sous dossier \dist\default\production de votre projet.

⇨ Programmez la carte et appuyez sur RESET (bouton rouge) pour démarrer le programme.

Pour programmer la carte de nouveau, il suffit d'appuyer de nouveau sur les boutons Reset et BP1 et de relâcher RESET d'abord.

12 DOCUMENTATIONS

Dans le répertoire Y:\COMMUN_E&R_S1_S2\Microchip\Docs, vous trouverez des documents à propos de la programmation de la carte microcontrôleur GamelTrophy.

Documents IUT

⇨ **Guide de mise en œuvre** : c'est le présent document.

Documents Microchip sur les outils

⇨ **MPLAB_XC8_Starter_guide.pdf** un guide d'apprentissage rapide de XC8 Compiler

⇨ **MPLAB_XC8_C_Compiler_User_Guide.pdf** est le guide d'utilisation de XC8 compiler

⇨ **MPLAB_XC8_Peripheral_Libraries.pdf** est le guide d'utilisation des librairies de XC8 compiler

Documents Microchip sur le PIC18F4550

⇨ **PIC18F4550.pdf** est la datasheet du microcontrôleur PIC18F4550

Vous trouverez aussi dans ce répertoire tous les outils et toutes les bibliothèques pour programmer la carte microcontrôleur GamelTrophy.

Vous trouverez une mine d'informations sur les microcontrôleurs et leur programmation sur le site de Microchip : www.microchip.com.

AIDE MÉMOIRE

Configuration du micro-contrôleur

Votre projet doit obligatoirement comporter le fichier ***iut_init.c*** qui contient la configuration du micro-contrôleur.

En en-tête de vos fichiers :

```
#include <xc.h>
```

Prototypes des différentes fonctions

bibliothèque iut_lcd

```
#include "iut_lcd.h"
```

void lcd_init(void);

void lcd_position(char ligne, char colonne);

void lcd_clear(void);

void lcd_putc(char lettre);

void lcd_printf(const rom char *f, ...);

bibliothèque iut_adc

```
#include "iut_adc.h"
```

void adc_init(char numero_dernier_canal);

int adc_read(char numero_canal);

bibliothèque iut_pwm

```
#include "iut_pwm.h"
```

void pwm_init(char period, char nb_canaux);

void pwm_setdc1(unsigned int cycles_etat_haut);

void pwm_setdc2(unsigned int cycles_etat_haut);

Pour utiliser le bootloader

1- Installer le fichier "bootloader.hex" avec MPlab PicKit3.
   Lancer MPLAB X
   Connecter le programmateur PicKit3
   Ouvrir le projet bootloader.X
   Cliquer sur Make and Program Device
2- Le Boot Loader est prêt à fonctionner : déconnecter MPlab PicKit3.
3- Pour passer en mode "BootLoader", brancher le cordon USB.
4- Appuyer sur BP1 (bouton vert) puis sur RESET (bouton rouge) et relâcher RESET en restant appuyé sur BP1.
5- Lancer le programme "HIDBootloader (Windows).exe" qui se trouve dans "BoatLoader USB Prog".
6- La carte doit être détectée.
7- Ajouter dans votre projet les deux fichiers "iut_bootloader.c" et "rm18f4550_HID_Bootload.lkr".
8- Configurer le projet :
   File -> Project Properties puis choisir "XC8 Linker" dans les "Categories"
   Choisir "Additional options" dans "Option categories"
   Mettre à 1000 le "Codeoffset"
9- Compiler le programme
10- Charger le fichier "HEX" généré avec l'application bootloader.
   Le ficher HEX se trouve dans le dossier "monProjet.X\dist\default\production"
11- Programmer la carte et appuyer sur RESET (bouton rouge) pour démarrer le programme.

Raccourcis clavier

Run project	F6	Run to cursor	F4	Step into	F7
		Continue	F5	Step Over	F8