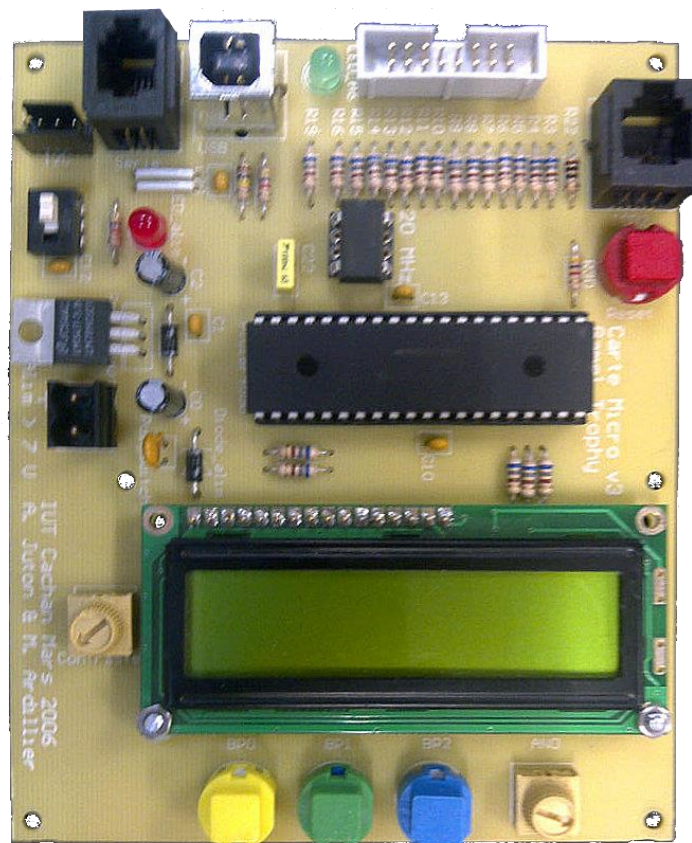


# Carte de commande Microcontrôleur PIC MICROCHIP Gamel Trophy Guide de Mise en œuvre



Source Anthony JUTON – Version 4.0

Modifications Yves GUINAND – Version 2013 et 2014

Modification Joëlle MAILLEFERT – Version 2017 et 2018



## SOMMAIRE

|   |    |
|---|----|
| 1 Introduction.....   | 5  |
| 2 Caractéristiques générales .....  | 5  |
| 2.1 Alimentation.....   | 5  |
| 2.2 Programmation .....   | 6  |
| 2.3 Périphériques internes du microcontrôleur .....                           | 6  |
| 2.4 Périphériques externes .....  | 6  |
| 3 Schémas.....  | 7  |
| 3.1 Schéma d’implantation des composants .....                                | 7  |
| 3.2 Schéma de câblage de la carte .....                                       | 8  |
| 4 Programmation de la carte – généralités.....                                | 10 |
| 4.1 Configuration du microcontrôleur.....                                     | 10 |
| 4.2 Configuration des entrées/sorties Tout ou Rien.....                       | 10 |
| 5 Boutons, Potentiomètre et LED .....   | 11 |
| 5.1 Boutons poussoirs .....   | 11 |
| 5.2 Potentiomètre .....   | 12 |
| 5.3 Led.....  | 12 |
| 6 Afficheur LCD.....  | 13 |
| 7 Les périphériques internes .....  | 15 |
| 7.1 Le convertisseur analogique-numérique .....                               | 15 |
| 7.2 Les PWMs .....  | 16 |
| 7.3 Timers .....  | 17 |
| 8 Connecteur I/O .....  | 19 |
| 9 Périphériques de communication .....  | 20 |
| 9.1 Port série RS232.....   | 20 |
| 9.2 Port I2C.....   | 20 |
| 9.3 Port USB.....   | 20 |
| 10 Tutoriel pour les outils de Développement microchip.....                   | 20 |
| 10.1 Création d’un projet et configuration de MPLAB X IDE .....               | 21 |
| 10.2 Compilation et construction du projet.....                               | 23 |
| 10.3 Programmation de la carte microcontrôleur avec le PickIt3 .....          | 24 |
| 10.4 Débogage du programme avec PickIt3 .....                                 | 24 |
| 11 Bootloader .....   | 25 |
| 11.1 Installer le bootloader USB sur la carte .....                           | 25 |
| 11.2 Créer un projet sous MPLAB destiné à être envoyé par le bootloader ..... | 25 |
| 11.3 Charger le programme sur la carte .....                                  | 26 |



# 1 INTRODUCTION

La carte de commande du Gamel Trophy est utilisée en projet de premier semestre et en TP d'informatique industrielle.

Elle est réalisée autour d'un microcontrôleur MICROCHIP PIC18F4550. Ce microcontrôleur 8 bits est issu d'une famille très populaire, utilisée dans l'informatique embarquée (électroménager, objets connectés, domotique, clés USB, afficheurs, appareils biomédicaux portatifs ...)

**Important - les versions à jour des bibliothèques et des outils logiciels utiles pour la programmation de la carte microcontrôleur sont disponibles ProfGe2 !!!**

Dans la gamme MICROCHIP, le microcontrôleur PIC18F4550 fut choisi pour les raisons suivantes :

- ⇒ Issu de la famille 18Fxxx, il est suffisamment rapide (jusqu'à 12 MIPS) pour notre application, il possède un convertisseur 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 microcontrôleur, et de le programmer
- ⇒ Il est disponible en boîtier DIP, ce qui facilite la conception de la carte et le remplacement du microcontrôleur en cas de panne.
- ⇒ Les outils de développement fournis par MICROCHIP sont fiables, gratuits, puissants et faciles à utiliser.

Nous utiliserons les outils de développement suivants :

- ⇒ Environnement de développement *MPLABX IDE*
- ⇒ Compilateur *MPLAB XC8*
- ⇒ Programmeur / Débogueur *MPLAB PicKit3*

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

## 2 CARACTERISTIQUES GENERALES

### 2.1 ALIMENTATION

On peut alimenter la carte avec une alimentation externe (ce sera la batterie de la gamelle) ou bien à partir du bus USB. Un interrupteur permet de passer d'une source d'alimentation à l'autre. 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 microcontrôleur 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 microcontrôleur 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, ...), voir le chapitre 20. Un bouton Reset (bouton poussoir rouge) permet, en mode autonome, de redémarrer le programme. Son état n'est pas disponible pour une utilisation dans un programme.

Les outils permettant de programmer cette carte, MPLABX IDE et MPLAB XC8 Compiler (version gratuite sans optimisation) ainsi que de nombreux documents, sont disponibles sur le site de MICROCHIP ([www.MICROCHIP.com](http://www.MICROCHIP.com)). Les documents réalisés à l'IUT (tutoriel, schéma de la carte, bibliothèques...) sont disponibles sur ProfGe2

## 2.3 PERIPHERIQUES INTERNES DU MICROCONTROLEUR

La carte est équipée d'un microcontrôleur PIC18F4550 ayant comme périphériques internes :

- ⇒ **4 timers**. Les timers sont des composants utilisés lorsque l'on a besoin d'une mesure de temps précise. Voir le chapitre 7 pour la mise en œuvre du timer0.
- ⇒ **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. Voir le chapitre 7 pour leur mise en œuvre.
- ⇒ **Deux sorties PWM** associées au timer2. Elles sont disponibles sur le connecteur I/O. Voir le 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, voir le chapitre 9 pour la mise en œuvre de cette liaison.
- ⇒ **Une liaison I2C** disponible sur le connecteur 4 points, voir le 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., voir le documentation MICROCHIP pour leur mise en œuvre.

Le microcontrôleur 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 PERIPHERIQUES 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, voir le 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 microcontrôleur, voir le chapitre 5 pour leur mise en œuvre.
- ⇒ **Un potentiomètre** délivrant une tension de 0 à 5V câblé sur l'entrée analogique AN0 du microcontrôleur, voir le chapitre 5 pour sa mise en œuvre.
- ⇒ **Une LED verte** connectée à la sortie Tout Ou Rien A6 du microcontrôleur, voir le 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. Avec les paramètres par défaut de la carte, le fonctionnement du cœur du microcontrôleur **est cadencé à 48 MHz** (un cycle instruction nécessite 4 tops horloge donc 83.3ns).

### 3 SCHEMAS

Les schémas présentés ici sont issus d'un projet *Altium Designer* disponibles sur ProfGe2

#### 3.1 SCHEMA D'IMPLANTATION DES COMPOSANTS

La figure 1 ci-dessous montre l'implantation physique des composants :

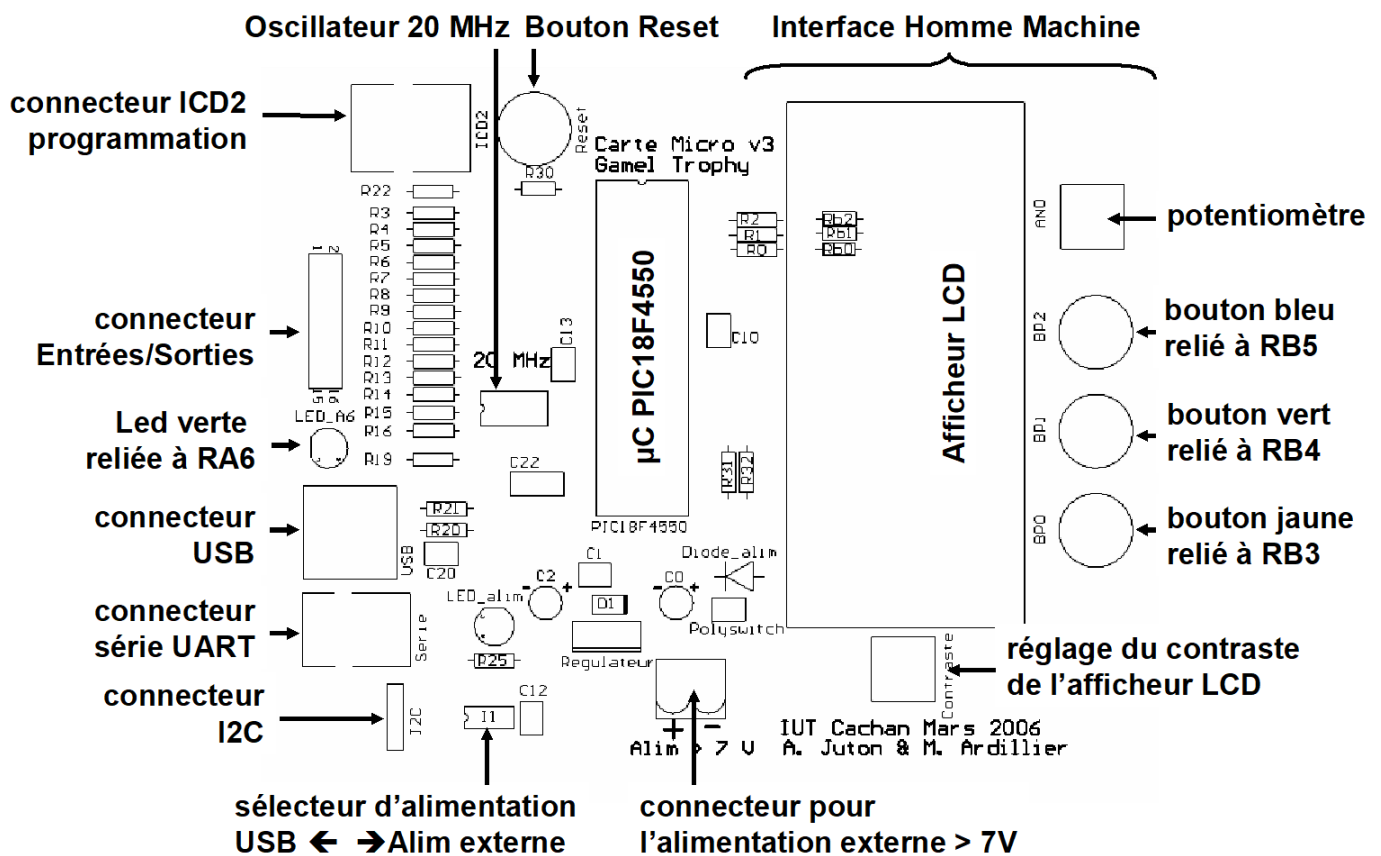


Figure 1 - Schéma d'implantation des composants

### 3.2 SCHEMA DE CABLAGE DE LA CARTE

Les 2 figures suivantes (Figure 2 et Figure 3) montrent le schéma de la carte avec l'interconnexion entre composants :

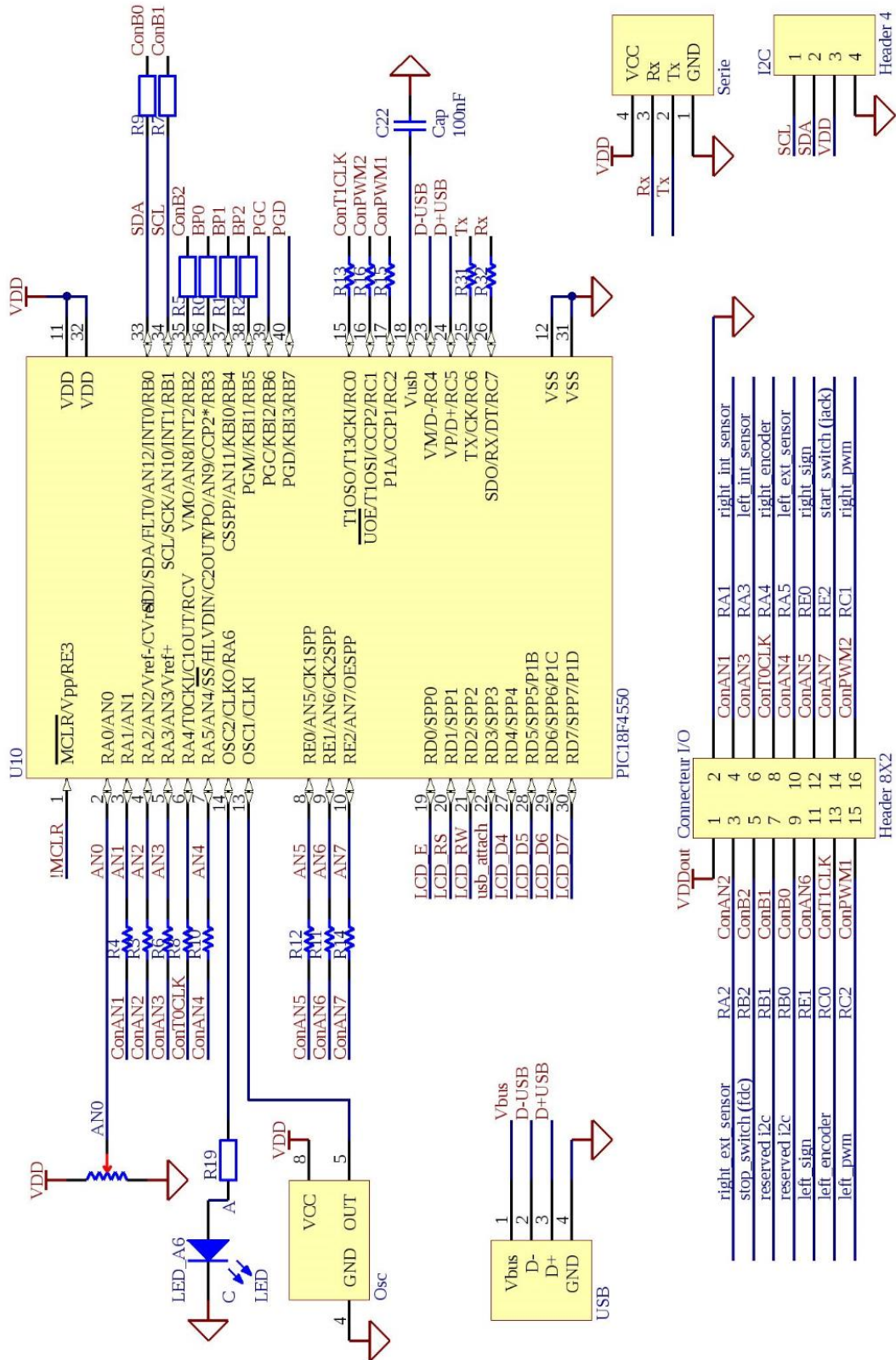


Figure 2 - Interconnexion des composants (page 1)



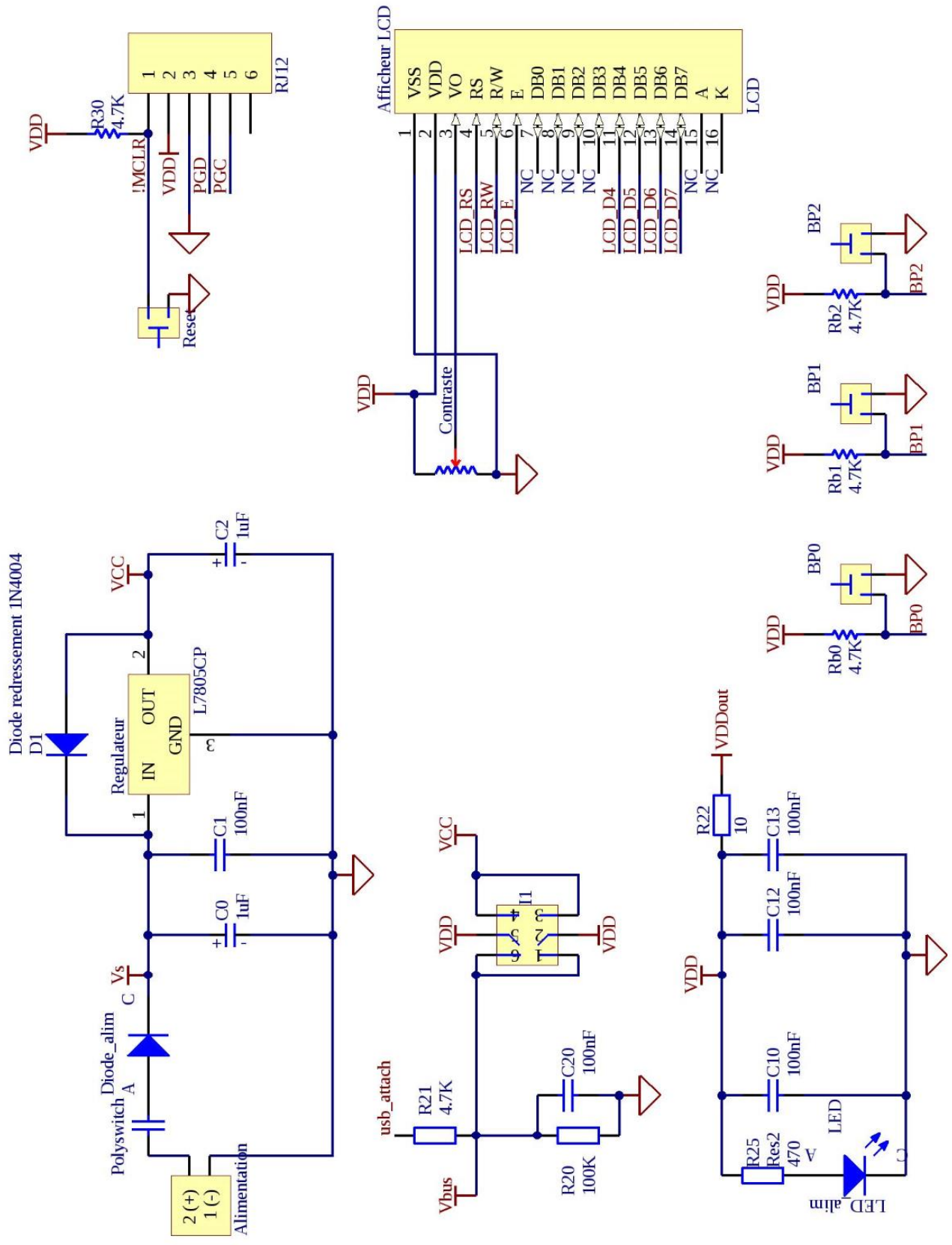


Figure 3 - Interconnexion des composants (Page2)

## 4 PROGRAMMATION DE LA CARTE – GENERALITES

Ce chapitre présente les connaissances indispensables avant de démarrer la programmation.

### 4.1 CONFIGURATION DU MICROCONTROLEUR

Pour s'adapter au mieux aux différentes applications auxquelles est destiné le microcontrôleur PIC18F4550, notamment en termes de consommation d'énergie, de nombreuses configurations sont possibles. Il est présenté ici dans 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 (appelées directives de compilation) 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*** au projet.

Il faut 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, pour le paramétrage des entrées/sorties et les initialisations de périphériques.

Dans l'exemple ci-dessous, on initialise l'écran LCD :

```
#include <xc.h>

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

    lcd_init();
    ...
}
```

### 4.2 CONFIGURATION DES ENTREES/SORTIES TOUT OU RIEN

En plus de la configuration des périphériques internes et externes, en début de programme et parfois pendant le programme, il est nécessaire de 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 & 0xBF;
```

Remarques :

- Les registres du microcontrôleur 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, POTENTIOMETRE ET LED

Pour les TP et les projets, la carte dispose d'une 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 paragraphe 3, Les boutons poussoirs sont câblés de la manière suivante (figure 4) :

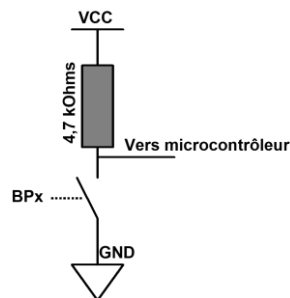


Figure 4 - Câblage des boutons poussoirs

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 microcontrôleur sera VCC, ici 5V. Les entrées du microcontrôleur 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 microcontrôleur qui reçoit alors 0V. La résistance de pull-up est parcourue par un courant valant  $VCC / R$ .

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;
TRISB = TRISB | 0x08; // configuration en entrée de RB3
// à écrire une fois, au début du programme
...
BP0 = PORTBbits.RB3; // à chaque lecture de l'état logique de B3
```

Pour lire plusieurs boutons à la fois, il vaut mieux utiliser un masque (voir le cours II1).

Un changement d'état sur B4 ou B5 peut-être configuré pour provoquer une interruption.

## 5.2 POTENTIOMETRE

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 (Figure 5).

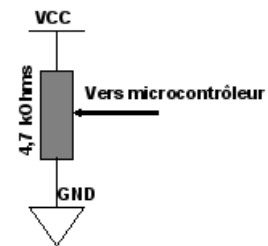


Figure 5 - Câblage du potentiomètre

L'entrée du microcontrôleur n'absorbant pas de courant, le montage est

un simple diviseur de tension. Pour une position *pos* (comprise entre 0

et 1) du curseur du potentiomètre, l'entrée analogique du microcontrôleur reçoit donc la tension suivante :

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

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;
adc_init(0); //initialisation de l'ADC, à écrire au début du programme
...
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 microcontrôleur. Elle est câblée de la manière ci-contre.

Si A6 = 1, le microcontrôleur impose 5 V en sortie, le courant passe alors dans la résistance et la LED : la LED s'allume. La résistance impose un courant  $I_D = (V_{CC} - V_D) / R \sim 6 \text{ mA}$ . C'est le microcontrôleur qui fournit le courant nécessaire. Une sortie Tout Ou Rien peut fournir jusqu'à 25 mA.

Si A6 = 0, le microcontrôleur impose 0V, la LED est alors éteinte.

Allumer la LED\_A6 revient ainsi à modifier l'état logique de A6.

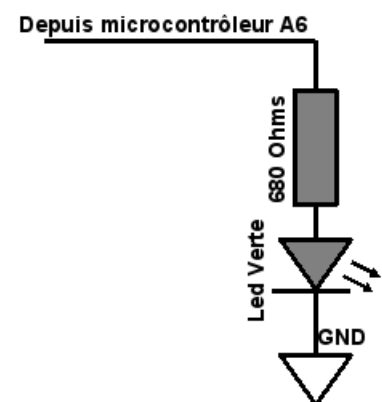


Figure 6 - Câblage de la LED verte

Exemple pour modifier l'état de la LED\_A6 :

```
TRISA = TRISA & 0xBF; // configuration en sortie de A6
                        // à écrire une fois au début du programme
...
PORTAbits.RA6 = 1; // mise à 1 de A6
...
PORTAbits.RA6 = 0; // mise à 0 de A6
```

## 6 AFFICHEUR LCD

Mieux que la LED\_A6, on peut utiliser l'afficheur LCD pour afficher des informations du microcontrôleur 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 microcontrôleur. 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, il faut :

- Copier les fichiers *iut\_lcd.h* et *iut\_lcd.c* dans le dossier du projet
- Ajouter le fichier *uit\_lcd.c* au 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 le programme avant tout emploi d'une autre fonction de la bibliothèque. Une seule exécution de cette fonction est suffisante.

### **void lcd\_position(unsigned char ligne, unsigned 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 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  $\mu$ s environ (à comparer avec les 0,1  $\mu$ s que dure l'exécution d'une instruction simple). Il faut en tenir compte si le programme doit effectuer une tâche rapidement ou à une cadence fixe (asservissement, tâche d'interruption).
- Dans la source de la bibliothèque `iut_lcd.c`, on peut lire comment ces fonctions sont écrites en C et ainsi regarder quel dialogue s'établit entre le microcontrôleur et l'afficheur LCD.

## 7 LES PERIPHERIQUES INTERNES

Le microcontrôleur PIC18F4550 a de multiples périphériques internes. Dans ce chapitre, sont présentés 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-NUMERIQUE

Le PIC18F4550 est muni en interne d'un convertisseur analogique-numérique 10 bits. Par multiplexage, celui-ci peut acquérir jusqu'à 13 nombres, images des tensions analogiques présentes sur différentes broches du boîtier. On peut utiliser celles qui sont disponibles sur le connecteur I/O : AN1 à AN7. La première, AN0, est reliée au potentiomètre nommé AN0.

Pour programmer le convertisseur simplement, une bibliothèque regroupe les fonctions nécessaires. Pour utiliser cette bibliothèque, il faut :

- Copier les fichiers *iut\_adc.h* et *iut\_adc.c* dans le dossier du projet projet
- Ajouter le fichier *iut\_adc.c* au 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). On peut utiliser le nombre d'entrées analogiques que l'on veut, 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 utilisé.

#### **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 permet pas de passer d'une entrée configurée en analogique à une entrée configurée en Tout Ou Rien en cours de programme. Elle ne permet pas non plus d'utiliser les interruptions, ou de modifier les temps d'acquisition, les références... On trouvera toutes les caractéristiques du convertisseur et le rôle de chaque registre le concernant au chapitre 19 de la documentation technique du microcontrôleur 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 deux sorties indépendantes nommées PWM1 et PWM2. Une PWM (pulse width modulation) est un signal carré dont la fréquence est fixe et le rapport cyclique (rapport entre la durée du niveau haut et la période du signal) variable. C'est un signal très utilisé dans la commande de moteur ou la conversion numérique-analogique.

Dans notre cas, les blocs PWMS utilisent pour fonctionner le timer2.

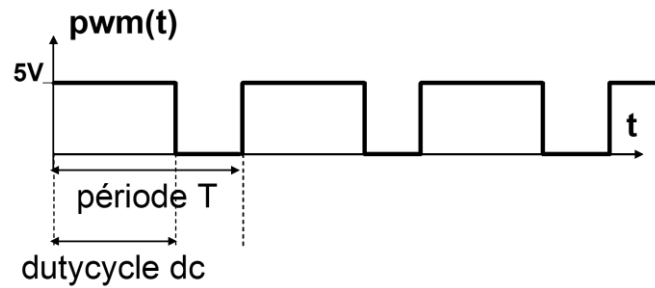


Figure 7 - Principe d'un signal PWM

Pour utiliser les sorties PWM présentes sur les broches C2 (PWM1) et C1 (PWM2), on dispose d'une bibliothèque qui regroupe les fonctions nécessaires. Pour utiliser cette bibliothèque, il faut :

- Copier les fichiers *iut\_pwm.h* et *iut\_pwm.c* dans le dossier du projet
- Ajouter le fichier *iut\_pwm.c* au 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. On choisit en paramètre de cette fonction la période des signaux PWMs, en nombre de pas de **333 ns**. Le paramètre *period* est un entier de 8 bits. Pour  $F_{osc} = 48 \text{ MHz}$  (en utilisant *iut\_init.c*), la fréquence de fonctionnement des PWMs est donnée par la formule :

$$f = (3 \cdot 10^6 / (\text{period} + 1))$$

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

### **void pwm\_setdc1(unsigned int cycles\_état\_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 codé sur 10 bits.

On peut calculer le rapport cyclique avec la formule :

$$\text{rapport cyclique} = \text{cycles\_etat\_haut} / (4 \times (\text{period} + 1))$$

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

```
#include <xc8.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 pour 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 16 bits peut compter de 0 à 65535 ( $2^{16}-1$ ). Un TIMER est un compteur que l'on peut connecter à l'horloge interne du microcontrôleur. 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.

Pour compter un nombre plus important d'événements, on peut pré-diviser la fréquence de l'horloge par un coefficient appelé pré-scalaire.

Pour utiliser les timers, on dispose d'une bibliothèque qui regroupe les fonctions nécessaires. Pour utiliser cette bibliothèque, il faut :

- Copier les fichiers *iut\_timers.h* et *iut\_timer.c* dans le dossier du projet
- Ajouter le fichier *iut\_timer.c* au projet
- Inclure le fichier d'en-têtes *iut\_timer.h* avec l'instruction `#include "iut_timer.h"`

### **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 la valeur du préscalaire (ou pré-diviseur)  
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 Start );**

Cette fonction permet d'écrire la valeur Start dans le TIMER. Ce dernier commencera 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 1 s (la LED change d'état toutes les 0,5s).

```
#include <xc8.h>
#include "iut_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
    // un tic dure 83,3 ns x 128 = 10 us (10,662)
    // pour avoir une temporisation de 0,5s il faut 50000 tics (46895)
    while(1)
    {
        WriteTimer0(0);          // mise à 0 du timer0
        while (ReadTimer0()<46895); // attente 0,5 s
        PORTAbits.RA6 = !PORTAbits.RA6; //inversion led
    }
}
```

## 8 CONNECTEUR I/O

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

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

|                                |  |
|--------------------------------|--|
| <b>1</b>                       | La broche 1 permet de fournir une alimentation de 5V à une autre carte avec un courant limité (moins de 50 mA). Si une carte externe connectée à la carte microcontrôleur est alimentée directement par la batterie, <b>il ne faut surtout pas utiliser cette broche</b> du connecteur.          |
| <b>2</b>                       | La broche 2 est la masse. Les masses d'autres cartes qui communiquent avec celle-ci doivent y être reliées.  |
| <b>3, 4, 6, 10, 11, 12, 14</b> | Ces broches du connecteur sont reliées à des broches du microcontrôleur 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 microcontrôleur.                                   |
| <b>5</b>                       | Cette broche du connecteur est reliée à une broche du microcontrôleur qui peut être au choix configurée comme une entrée analogique (c'est le cas par défaut), comme une entrée d'interruption ou comme un entrée/sortie Tout ou Rien.   |
| <b>7 et 9</b>                  | Ces broches sont reliées à des entrées/sorties Tout ou Rien du microcontrôleur (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 <sup>2</sup> C. |
| <b>8</b>                       | Cette broche est reliée à une broche du microcontrôleur qui peut être configurée au choix comme une entrée/sortie Tout ou Rien ou comme entrée comptage rapide du compteur/TIMER 0.  |
| <b>13</b>                      | Cette broche est reliée à une broche du microcontrôleur qui peut être configurée au choix comme une entrée/sortie Tout ou Rien ou comme entrée comptage rapide des compteurs/TIMERS 1 et 3.  |
| <b>15 et 16</b>                | Ces broches sont reliées à des broches du microcontrôleur 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 (paragraphe 3), on peut 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 microcontrôleur (c'est le cas quand la broche est configurée comme entrée du microcontrôleur), cela ne modifie rien. Par contre, si on souhaite que le microcontrôleur fournisse du courant (allumage d'une diode LED), il faudra tenir compte de la chute de tension dans cette résistance.

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

## 9 PERIPHERIQUES DE COMMUNICATION

### 9.1 PORT SERIE RS232

La liaison RS232 est une liaison série (les bits sont envoyés les uns après les autres) asynchrone (l'horloge cadencant la liaison n'est pas transmise.)

Le port série de la carte microcontrôleur, 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.

Se référer à la documentation de la bibliothèque pour comprendre comment les utiliser.

### 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 cadencant 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. Se référer à la documentation de la bibliothèque pour leur utilisation.

### 9.3 PORT USB

Le microcontrôleur 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. (voir le site Web de MICROCHIP).

## 10 TUTORIEL POUR LES OUTILS DE DEVELOPPEMENT MICROCHIP

Nous utiliserons les outils de développement MICROCHIP suivants :

- ⇒ Environnement de développement *MPLAB X IDE (V4.2 en 2018-2019)*
- ⇒ Compilateur C *MPLAB XC8 (V2.00 en 2018-2019)*
- ⇒ Programmeur / Débogueur *MPLAB PicKit3*

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

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

## 10.1 CREATION D'UN PROJET ET CONFIGURATION DE MPLAB X IDE

Lancer MPLAB X IDE et créer un nouveau projet avec le menu *File* → *New project*

Dans la catégorie *MICROCHIP Embedded*, choisir *Standalone Project* et cliquer sur *Next* (Figure ci-dessous) :

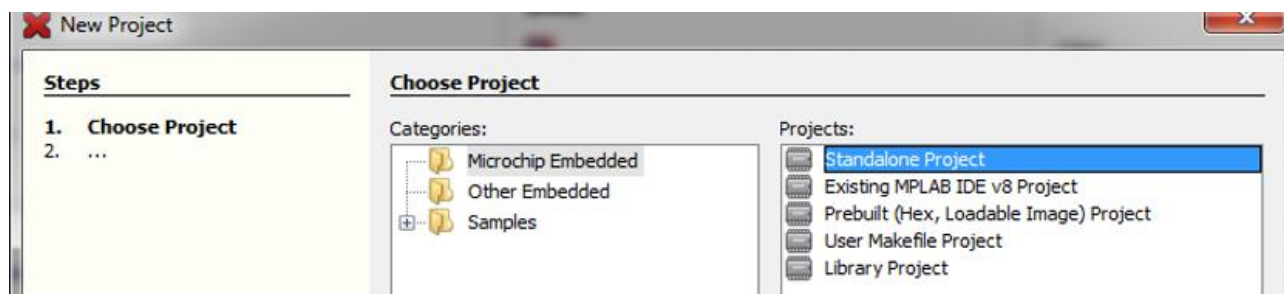


Figure 8 - Création du projet

Choisir la famille *Advanced 8-bit MCUs (PIC18)* et le microcontrôleur *PIC18F4550*, puis cliquer sur *Next* (Figure ci-dessous) :

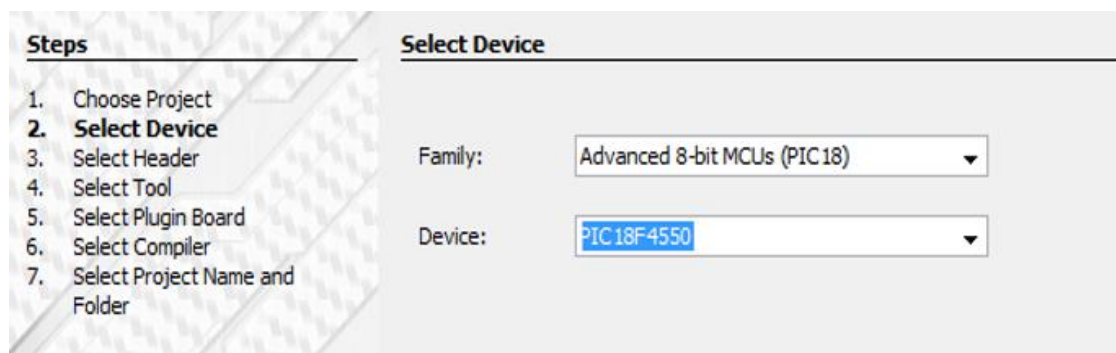


Figure 9 - Choix du composant

Sélectionner le programmeur *PicKit3* et cliquer sur *Next* (Figure 10 ci-dessous) :

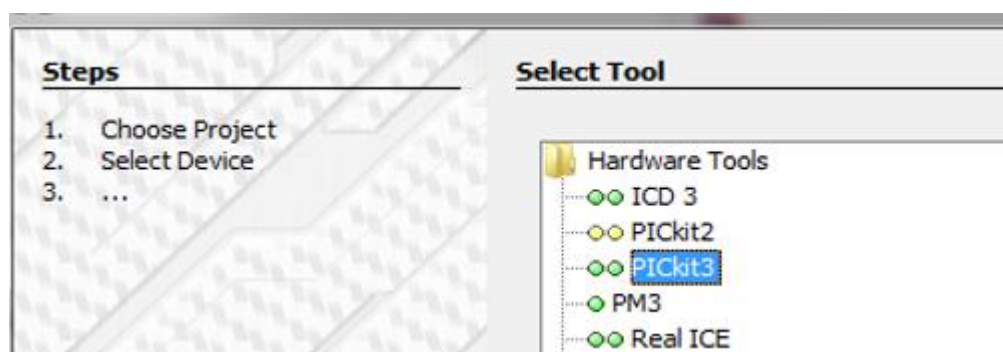


Figure 10 – Choix du programmeur

Sélectionner le compilateur *XC8 (V2.00 en 2018-2019)* et cliquer sur *Next* (Figure 11 ci-dessous) :

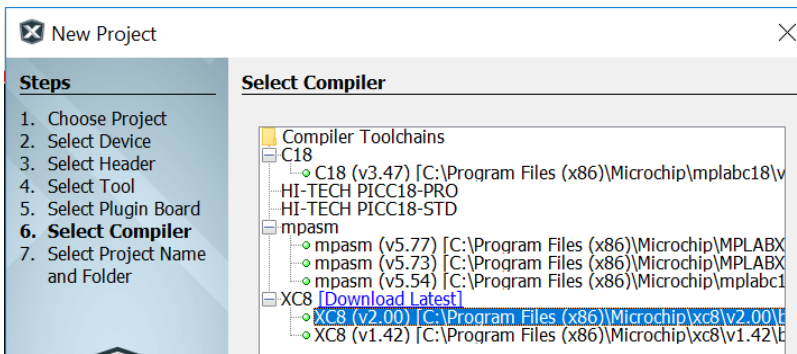


Figure 11 - Choix du compilateur

Choisir un nom de projet (**Project Name**), par exemple **Mon\_Projet**. Ne pas utiliser les caractères spéciaux, les accents ni les espaces. Créer un dossier de travail, par exemple **Dossier\_MICROCHIP** comme emplacement de sauvegarde du projet (**Project Location**). NE PAS UTILISER LE DOSSIER DONNE PAR DEFAULT (Figure 12 ci-dessous) :

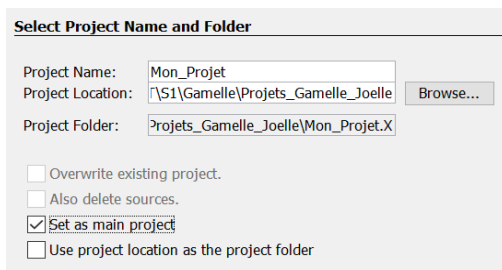


Figure 12 - Sauvegarde du projet

Copier dans le dossier du projet, ici **Dossier\_MICROCHIP\monProjet.X**, les fichiers (\*.c et \*.h) des bibliothèques de l'IUT pour la carte microcontrôleur qu'on trouvera sur ProfGe2

Dans MPLABX, effectuer un clic droit sur *Source Files* puis cliquer sur *Add Existing Item*. Ajouter les quatre fichiers avec l'extension **.c** (iut\_adc.c, iut\_pwm.c, iut\_lcd.c, iut\_init.c). Dans un premier temps, ne pas ajouter iut\_timers.c (voir le chapitre 7.3 pour l'utilisation des TIMERS).

On obtient ceci :

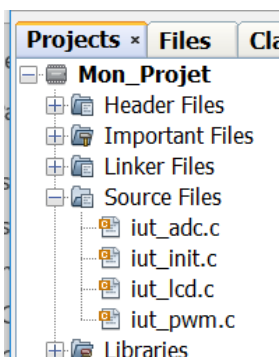


Figure 13 - Arborescence du projet

Effectuer un nouveau clic droit sur *Source Files* puis cliquer sur *New* → *C Source File*, puis *Next*.

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

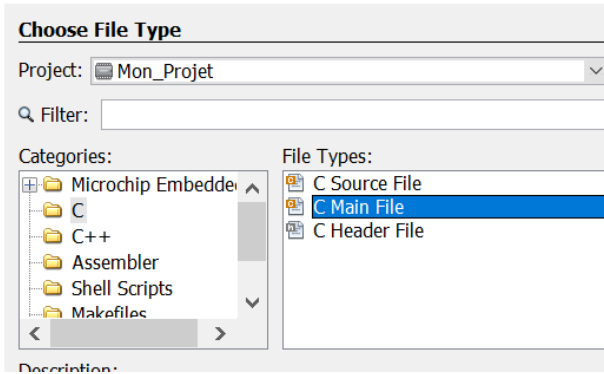


Figure 14 - Création du programme principal

On peut maintenant écrire un premier programme (en effaçant ou en modifiant le modèle de programme proposé)

```
#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...

### Sauvegarder le projet (File → Save All)

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

- iut\_init.c configuration du microcontrôleur (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

Sélectionner *Run → Build Project*  pour compiler et faire l'édition de liens 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 (Figure 15)

```
make -f nbproject/Makefile-default.mk SUBPROJECTS= .b
make[1]: Entering directory 'C:/Dropbox/DUT/S1/Gamelle'
make -f nbproject/Makefile-default.mk dist/default/p
make[2]: Entering directory 'C:/Dropbox/DUT/S1/Gamelle'
make[2]: 'dist/default/production/Test_Gamelle.X.prod
make[2]: Leaving directory 'C:/Dropbox/DUT/S1/Gamelle'
make[1]: Leaving directory 'C:/Dropbox/DUT/S1/Gamelle'

BUILD SUCCESSFUL (total time: 63ms)
Loading code from C:/Dropbox/DUT/S1/Gamelle/Maintenan
Warning: The hex file has the debug bit set. The deb
Loading completed
```

Figure 15 - Résultat de la compilation

### 10.3 PROGRAMMATION DE LA CARTE MICROCONTROLEUR AVEC LE PICKIT3

Pour programmer la carte microcontrôleur, il faut brancher le programmeur PickIt3 et la carte microcontrôleur.

Cliquer ensuite sur *Make and Program Device*



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

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

### 10.4 DEBOGAGE 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 programmeur PickIt3.

Pour mettre au point les programmes, il est indispensable de savoir utiliser le débogueur de l'outil de développement.

#### 10.4.1 Placement d'un point d'arrêt (breakpoint)

Il suffit de cliquer sur le numéro de la ligne pour créer un point d'arrêt dans le programme. La ligne est alors surlignée en rouge. Dans ce cas, lorsqu'on lance l'exécution du programme, celle-ci s'arrête juste avant l'instruction repérée par le point d'arrêt. Ceci permet de vérifier la valeur d'une variable, de la forcer...L'exécution reprend avec un Run.

#### 10.4.2 Débogueur

Cliquer sur *Debug Project*

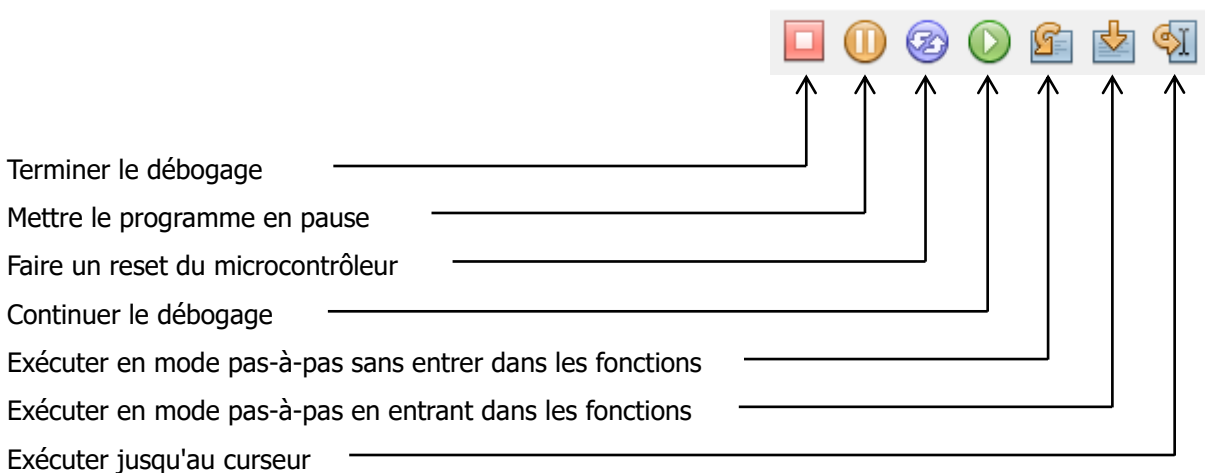


La carte est programmée en mode débogage et le programme est exécuté.

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 :





## 11 BOOTLOADER

Il est possible de programmer la carte sans le programmeur PicKit3, avec un simple câble USB. Pour cela il suffit d'installer un petit programme appelé **bootloader** dans la mémoire flash du microcontrôleur (Figure 16).

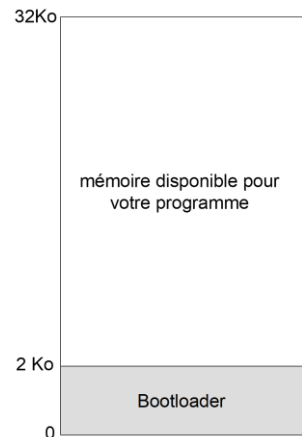
Les ressources pour cela sont sur ProfGe2

### 11.1 INSTALLER LE BOOTLOADER USB SUR LA CARTE

Pour utiliser **bootloader**, il faut d'abord l'installer dans la mémoire du microcontrôleur à l'aide d'un programmeur PicKit3.

⇒ Pour cela, sous MPLAB X, ouvrir le projet bootloader.X présent sur ProfGe2

Figure 16 - Mémoire flash du microcontrôleur



⇒ Programmer alors le microcontrôleur: *Make and Program Device*



Le **bootloader** est désormais installé sur la carte, le programmeur n'est plus nécessaire.

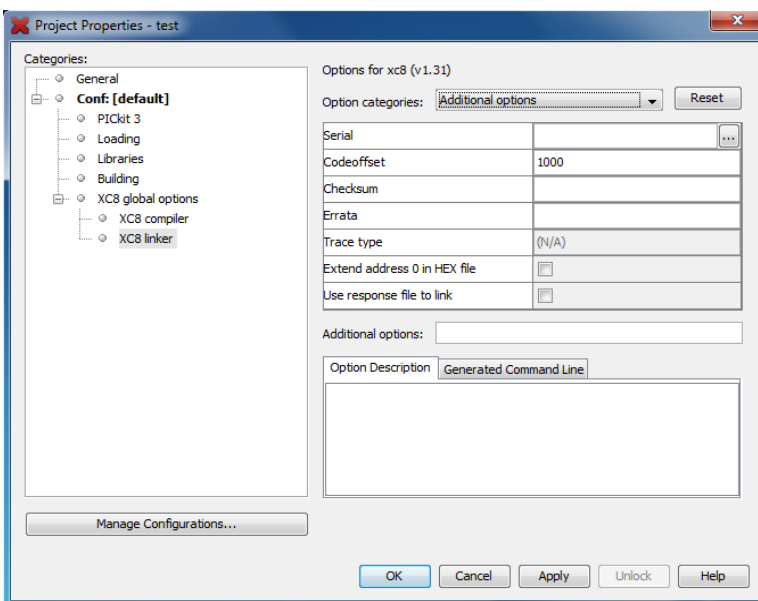
### 11.2 CREER UN PROJET SOUS MPLAB DESTINE A ETRE ENVOYE PAR LE BOOTLOADER

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

⇒ Ajouter le fichier **iut\_bootloader.c** au projet

⇒ Ajouter le fichier de linker **rm18f4550\_HID\_Bootload.lkr** au projet

Ces deux fichiers se trouvent sur ProfGe2



Configurer le projet (Figure 17):

- 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, on compile le programme normalement.

Figure 17 – Configurer le projet pour utiliser le bootloader

### 11.3 CHARGER LE PROGRAMME SUR LA CARTE

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

⇒ Brancher le cordon USB (qui peut aussi alimenter la carte) au PC et à la carte. Appuyer sur les boutons RESET (bouton rouge) et BP1 (bouton vert) en même temps puis relâcher 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.

⇒ Charger le fichier "HEX" du programme normalement présent dans le sous dossier \dist\default\production du projet.

⇒ Programmer la carte et appuyer 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.