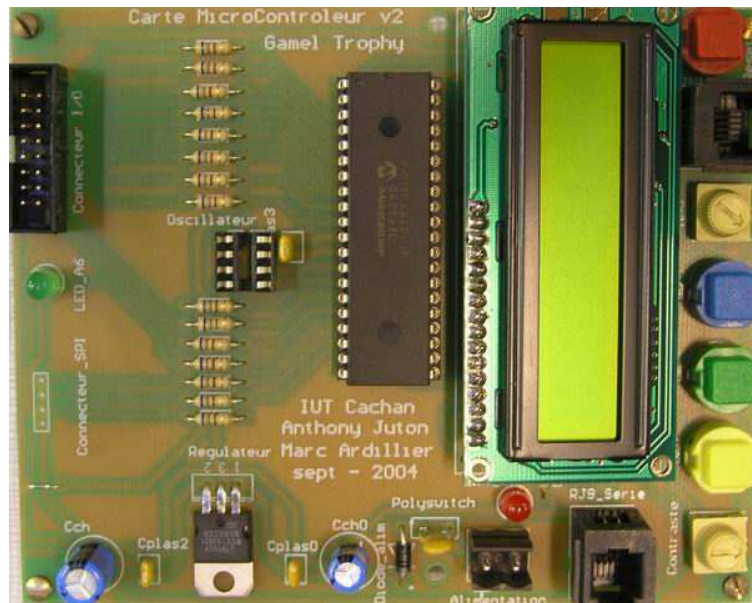


Carte Microcontrôleur Gamel Trophy

Guide de Mise en œuvre



SOMMAIRE

Sommaire	2
1 Introduction	3
2 Caractéristiques générales.....	4
2.1 Alimentation.....	4
2.2 Programmation	4
2.3 Périphériques internes du microcontrôleur	4
2.4 Périphériques externes	5
3 Schémas	6
3.1 schéma d'implantation des composants	6
3.2 schéma de câblage de la carte.	7
4 Programmation de la carte – généralités	8
4.1 Configuration du microcontrôleur	8
4.2 Configuration des entrées/sorties Tout ou Rien	8
5 Boutons, Potentiomètre et Led	10
5.1 Boutons poussoirs	10
5.2 Potentiomètre	11
5.3 Led.....	11
6 Afficheur LCD	13
7 Les périphériques internes	15
7.1 Le convertisseur analogique.....	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 SPI / I2C.....	20
10 Tutorial Outils de Développement.....	21
10.1 Création d'un projet et configuration de MPLAB.....	21
10.2 Compilation et construction du projet :	25
10.3 Exécution du programme et débogage avec MPLAB ICD 2 :	25
10.4 Exécution du programme en mode autonome	27
11 Documentation.....	28
Aide mémoire.....	29

1 INTRODUCTION

La carte microcontrôleur Gamel Trophy est utilisée en projet de premier semestre et en TP d'informatique industrielle à l'IUT. Elle est aussi le support de TP d'informatique industrielle en licence professionnelle.

Elle est réalisée autour d'un microcontrôleur Microchip PIC18F4320. Ce microcontrôleur 8 bits est issu d'une famille très répandue : 2 milliards d'exemplaires vendus par Microchip, 2nd fabricant mondial de microcontrôleurs 8 bits derrière Motorola.

De tels microcontrôleurs sont employés dans l'informatique embarquée des automobiles, de l'électroménager, des télécoms et des systèmes industriels. Ainsi, vous trouverez des microcontrôleurs Microchip entre autres dans les autos BMW ou Toyota, les magnétoscopes JVC, les consoles SEGA, les téléphones Motorola ou Nokia, les systèmes automatisés AllenBradley ou Honeywell.

Dans la gamme de Microchip, le microcontrôleur PIC18F4320 fut choisi pour les raisons suivantes :

- ⇒ Issu de la famille 18Fxxx, c'est un microcontrôleur rapide (jusqu'à 10 MIPS), possédant des convertisseurs analogiques numériques, des sorties PWM, une liaison série USART, une liaison I2C, une liaison SPI et 4 timers.
- ⇒ Il possède une horloge interne à 8 MHz, ce qui limite le nombre de composant sur la carte et améliore ainsi la fiabilité.
- ⇒ 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, relativement bon marché, puissants et faciles à utiliser.

Nous utiliserons les outils de développement suivants :

- ⇒ Environnement de développement *MPLAB IDE v7.20*
- ⇒ Compilateur C *MPLAB C18 Compiler v3.0*
- ⇒ Programmeur / Débogueur *MPLAB ICD2*

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 tutorial pour l'utilisation de la chaîne de développement.

2 CARACTERISTIQUES GENERALES

2.1 ALIMENTATION

La carte est alimentée par une tension continue supérieure à 8 V (jusqu'à 18 V). Cette tension est ensuite régulée pour fournir 5 V au microcontrôleur et aux différents périphériques. Typiquement la carte consomme 100 mA. Une led rouge indique le bon fonctionnement de l'alimentation 5 V.

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,...). Cf chapitre 10. Un bouton Reset 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, MPLAB IDE et MPLAB C18 Compiler (version limitée à 60 jours) ainsi que de nombreux documents (tutorial, schéma de la carte, datasheet du PIC18F4320, librairies...) sont disponibles dans le dossier Y:\commun\Microchip.

2.3 PERIPHERIQUES INTERNES DU MICROCONTROLEUR

La carte microcontrôleur est équipée d'un microcontrôleur PIC18F4320 ayant comme périphériques internes :

⇒ **4 timers**. Certains timers peuvent être utilisés comme compteur rapide avec une horloge externe. Deux entrées compteur rapide sont disponibles sur le connecteur I/O (timer0 et timer1). 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 ANA0. 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 série. Attention, pour simplifier la carte, cette liaison série est en niveaux logiques 0-5V. Si on veut utiliser le format 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/SPI** disponible sur le connecteur 4 points. cf chapitre 9 pour la mise en œuvre de la liaison.

De plus, 3 interruptions externes sont disponibles sur le connecteur I/O. cf documentation Microchip pour leur mise en œuvre.

Le microcontrôleur dispose pour les données d'une mémoire RAM (rapide, mais volatile, 512 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, 8 Ko, soit 4000 instructions).

2.4 PERIPHERIQUES EXTERNES

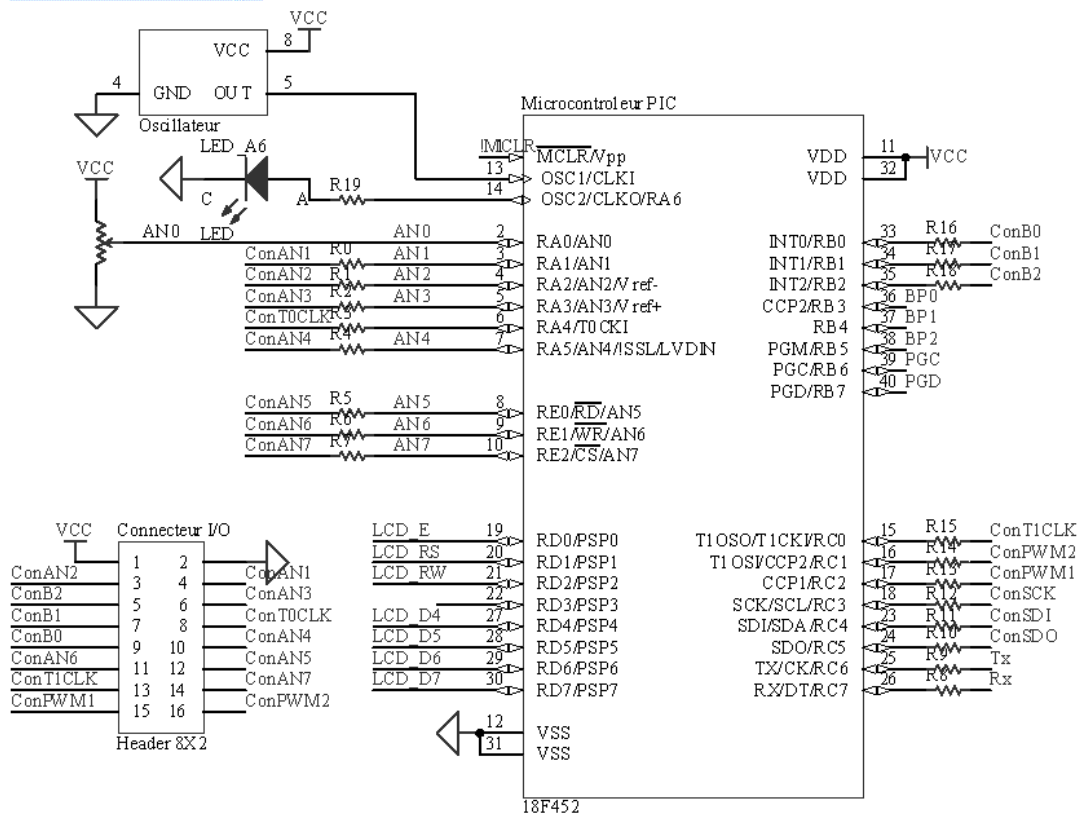
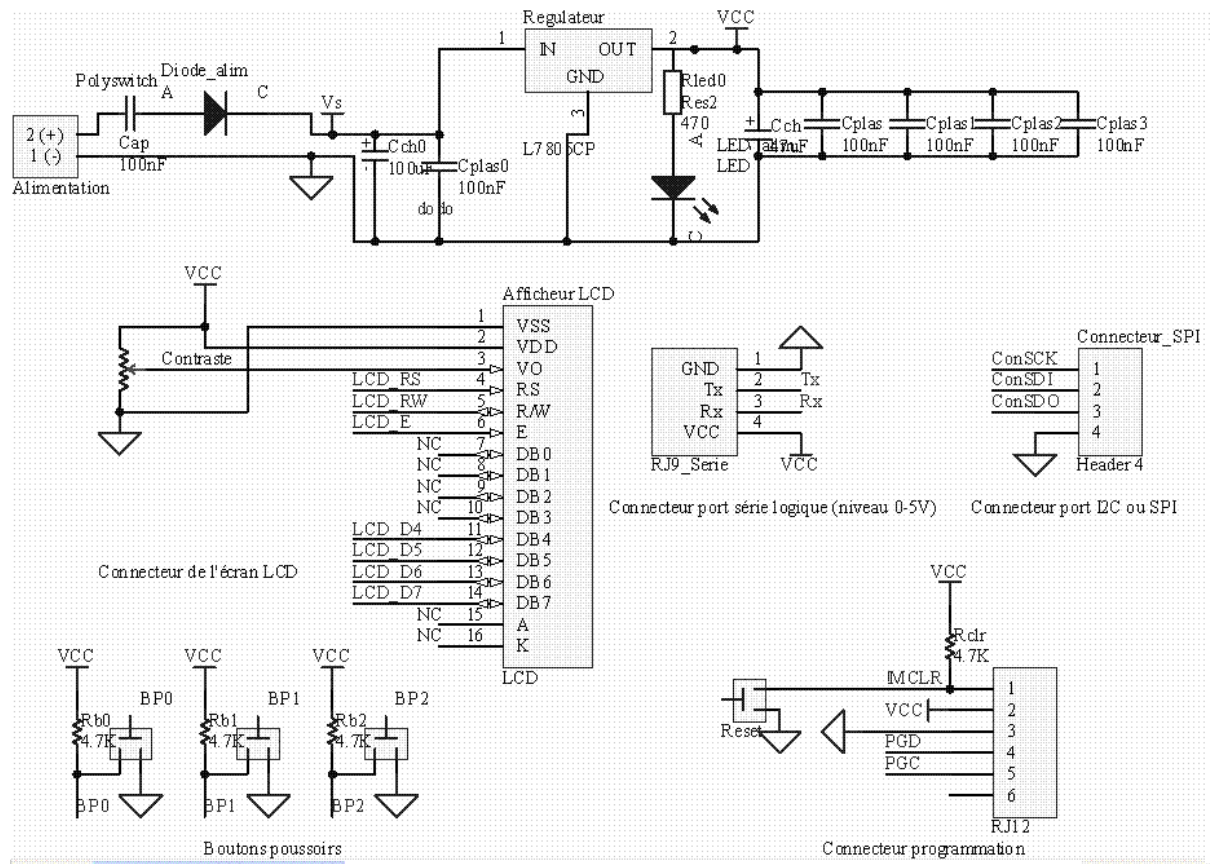
Sur la carte, ont été ajoutés les périphériques externes suivants :

- ⇒ **Un afficheur LCD** 2x16 caractères alphanumériques câblé sur le port D, avec son potentiomètre de contraste. 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 microcontrôleur. Cf chapitre 5 pour leur mise en œuvre.
- ⇒ **Un potentiomètre** délivrant une tension de 0 à 5V câblé sur l'entrée analogique AN0 du microcontrôleur. Cf chapitre 5 pour leur mise en œuvre.
- ⇒ **Une led verte** est connectée à la sortie Tout ou Rien A6 du microcontrôleur. Elle n'est pas utilisable lorsqu'on travaille avec un oscillateur externe dont on multiplie la fréquence par la PLL interne. Cf chapitre 5 pour sa mise en œuvre.

Un support DIP8 est prévu pour l'utilisation d'un oscillateur externe. Pour des applications nécessitant une exécution rapide des instructions, on pourra ainsi fonctionner à 40 MHz avec un oscillateur à cette fréquence ou avec un oscillateur à 10 MHz grâce à la PLL interne du microcontrôleur.

Le plus souvent, on utilisera l'horloge interne à 8 MHz du microcontrôleur.

3.2 SCHEMA DE CABLAGE DE LA CARTE.



4 PROGRAMMATION DE LA CARTE – GENERALITES

Ce chapitre vous présente les quelques connaissances indispensables avant de commencer un programme sur la carte microcontrôleur Gamel Trophy.

4.1 CONFIGURATION DU MICROCONTROLEUR

Pour s'adapter au mieux aux différentes applications auxquelles est destiné le microcontrôleur PIC18F4320, 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 particulières dans la documentation liée au PIC18F4320 (cf chapitre 11).

Une partie de la configuration se fait en ROM, c'est-à-dire dans une zone non modifiable par le programme. Pour écrire dans cette zone particulière, on insère la fonction de configuration entre les mots-clés suivants :#pragma romdata et #pragma. On utilise pour la configuration des masques s'ajoutant à la configuration par défaut. Pour le sens des différents octets de configuration, voir AN_programming18F4320.

Pour simplifier la création d'un projet, la configuration par défaut (horloge interne, A6 en sortie, mode debug,...) peut être faite en incluant l'en-tête gamelinit.h à votre projet.

```
| #include "gamelinit.h"
```

D'autres étapes de configuration se font en début d'exécution du programme, dans des zones de mémoire accessibles en écriture. Cette configuration se fait juste après les déclarations/définitions des variables.

Vous y mettez la configuration de la vitesse d'horloge (OSCCON=OSCCON|0b01110000) et les initialisations de périphériques internes et externes (lcd_init(), adc_init(...), pwm_init(...)).

```
| void main()  
| {  
|   char... // déclaration/définition des variables  
|   OSCCON=OSCCON|0b01110000;  
|   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, vous aurez à configurer aussi les entrées et sorties Tout ou Rien.

Pour cela, on utilise la fonction 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 TRISA.

RA7	RA6	RA5	RA4	RA3	RA2	RA1	RA0
-----	-----	-----	-----	-----	-----	-----	-----

Si on veut placer RA6 en sortie (broche reliée à la Led verte), il faut mettre le bit 6 de TRISA à 0. 2 méthodes 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 :

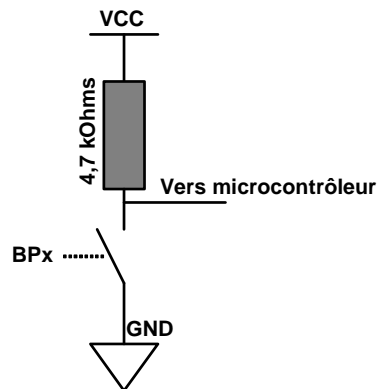
```
| TRISA = TRISA & 0b10111111;
```

5 BOUTONS, POTENTIOMETRE ET LED

Pour les TPs et projets, la carte dispose d'une très modeste interface homme-machine composée de 3 boutons poussoirs, d'un potentiomètre en entrée et d'une Led et d'un afficheur LCD en sortie.

5.1 BOUTONS POUSSOIRS

Comme indiqué sur le schéma de 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 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 VCC/R .

Les boutons sont connectés aux broches suivantes

- BP0 B3
- BP1 B4
- BP2 B5

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

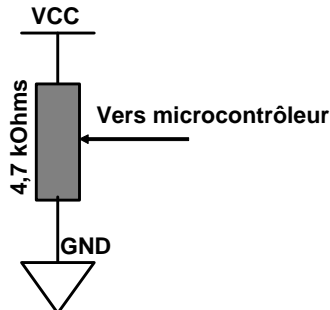
```
char BP0=0 ;
BP0 = PORTBbits.RB3 ; //lecture de l'état logique de B3
```

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 POTENTIOMETRE

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



L'entrée du microcontrôleur n'absorbant pas de courant, le montage est un simple diviseur de tension. Pour une position *alpha* du curseur du potentiomètre, l'entrée analogique du microcontrôleur peut donc lire la tension suivante :

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

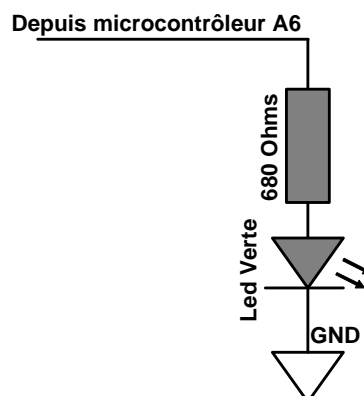
La lecture de la valeur du potentiomètre utilise le convertisseur analogique numérique. Pour plus de détails, on se référera au chapitre 11 qui traite de la librairie associée au convertisseur ADC interne.

Exemple pour la lecture de la valeur du potentiomètre.

```
#include "gameladc.h"
...
int potana=0 ;
adc_init(0) ; //initialisation du convertisseur
potana = adc_read(0) ; //lecture de la valeur AN0
```

5.3 LED

Une led verte, appelée led_A6, est relié à la sortie Tout ou Rien A6 du microcontrôleur. Elle est câblée de la manière suivante :



Si $A6 = 1$, le microcontrôleur impose 5 V en sortie, le courant passe alors dans la résistance et la diode Led : la diode s'allume. La résistance impose un courant $I_D = (V_{CC} - V_D) / R = 5 \text{ mA}$. 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.

Exemple pour allumer la led_A6

```
TRISA = 0xBF ;  
PORTAbits.RA6 = 1 ;
```

6 AFFICHEUR LCD

Mieux que la Led_A6, vous pouvez utiliser l'afficheur LCD pour envoyer des informations du microcontrôleur à 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 instructions du microcontrôleur. Un protocole de dialogue est établi par le fabricant du contrôleur de l'afficheur LCD.

Pour simplifier le travail du programmeur, une librairie regroupe les fonctions les plus utiles : gamelcd :

void lcd_init(void) : Cette fonction initialise l'afficheur. Il faut absolument l'appeler dans votre programme avant tout emploi d'une autre fonction de la librairie. Une seule exécution de cette fonction est suffisante.

void lcd_gotoxy(char x, char y) : Cette fonction positionne le curseur sur l'afficheur. x correspond à la ligne, y à la colonne et (1,1) est le coin haut gauche. Au prochain appel d'une fonction d'affichage lcd_put..., le premier caractère s'inscrira à l'endroit désigné par lcd_gotoxy.

void lcd_puti(int nombre) : Cette fonction affiche un nombre entier (int) sur l'afficheur

void lcd_putc(char lettre) : Cette fonction affiche un caractère sur l'afficheur

void lcd_puts(char* message) : Cette fonction affiche la chaîne de caractères (string) désignée par le pointeur message.

Pour utiliser la librairie, vous devez inclure en en-tête de votre fichier principal les déclarations de ces différentes fonctions. Ceci est fait par l'instruction #include gamelcd.h.

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

```
#include "gamelcd.h"
...
char texte[]="arthur"; //définition d'une chaîne de
                        //caractères en RAM.
lcd_init();           // initialisation de l'afficheur
lcd_gotoxy(2,10);    //déplacement du curseur
lcd_puts(texte);     //écriture du texte
```

Remarques

⇒ Les fonctions lcd_putc et lcd_puts comprennent les caractères spéciaux suivant :

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

⇒ Pour 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, en rajoutant entre parenthèse le type nouveau voulu. Cela a pour effet de créer un int n'ayant que des 0 dans l'octet de poids fort.

Exemple de syntaxe :

```
char arthur=10 ;  
...  
lcd_puti( (int)arthur )
```

⇒ 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 0,5 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 librairie gamelcd.c, vous pouvez 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 PIC18F4320 a de multiples périphériques internes. Dans ce chapitre, nous vous présentons les 3 périphériques internes les plus utilisés : le convertisseur analogique, les sorties PWM et les timers.

Les périphériques de communication sont étudiés au chapitre 9.

7.1 LE CONVERTISSEUR ANALOGIQUE

Le PIC18F4320 est muni en interne d'un convertisseur analogique numérique 10 bits. Par multiplexage, celui-ci peut vous renvoyer jusqu'à 8 nombres images des tensions analogiques présentes aux broches A0 à A3, A5 et E0 à E2. La première, AN0, est reliée au potentiomètre nommé AN0. Toutes les autres sont disponibles sur le connecteur I/O.

Pour utiliser le convertisseur simplement, une librairie, `gameladc`, regroupe les fonctions dont vous avez besoin :

`void adc_init(char numero_lastchannel_used)` est une fonction qui 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 "channel" que vous utilisez.

`int adc_read(char numero_channel)` est une fonction qui déclenche la lecture de la valeur analogique présente au "channel" indiqué en paramètre. Elle renvoie ensuite cette valeur sous forme d'entier 16 bits. Seuls les 10 derniers bits sont significatifs.

L'utilisation de la librairie demande que soit inclus en-tête de votre fichier source le fichier de déclaration de ces fonctions, `gameladc.h`

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

```
#include "gameladc.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
}
```

Remarques

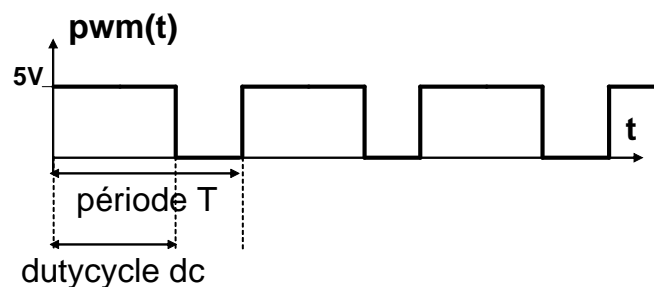
⇒ L'acquisition suivie de la conversion durent environ $40 \mu\text{s}$, un temps relativement grand comparé au $0,5 \mu\text{s}$ nécessaire à l'exécution d'une instruction simple.

⇒ La librairie donnée ne vous permet pas de repasser une entrée analogique 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 datasheet et des librairies plus complètes et plus complexes dans le chapitre 2.2 du guide d'utilisation des librairies MPLAB_C18_librairies.pdf.

7.2 LES PWMS

Le PIC18F4320 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 aux broches C2 (PWM1) et C1 (PWM2), vous utiliserez la librairie gamelpwm qui regroupe les fonctions dont vous avez besoin.

void pwm_init(char period, char nb_canaux) est une fonction qui initialise les sorties PWM. Si $nb_canaux = 2$ alors les deux sorties sont activées, sinon seule la sortie PWM1 est activée. Vous entrez en paramètre de cette fonction la période de votre PWM, en nombre de cycles instruction, un cycle instruction durant $0,5 \mu\text{s}$. $period$ est un entier de 8 bits. C'est cette fonction qui configure le timer2 pour le fonctionnement des PWMs.

void pwm_setdc1(unsigned int dutycycle1) définit le rapport cyclique de PWM1

void pwm_setdc2(unsigned int dutycycle2) définit le rapport cyclique de PWM2

$dutycycle$ est le temps pendant lequel PWM est haut, en nombre de cycle horloge, un cycle horloge durant $0,125 \mu\text{s}$. $dutycycle$ est un entier de 10 bits.

L'utilisation de la librairie demande que soit inclus en-tête de votre fichier source le fichier de déclaration de ces fonctions, gamelpwm.h

Exemple permettant de sortir en C1 et C2 deux PWMs à 20 kHz, de rapport cyclique moitié.

```
#include gamelpwm.h
...
void main(void)
{
    pwm_init(100,2); //initialisation de la periode 50
                    //µs, pour les deux canaux
    pwm_setdc1(200); //définition du temps à niveau
    pwm_setdc2(200); //haut(dutycycle) : 25 µs
}
```

Remarque :

- ⇒ Les broches configurées en PWM par `pwm_init(...)` restent en PWM tout le programme.
- ⇒ Attention : la période a une résolution de 0,5 µs et `dutycycle` une résolution de 0,125 µs.
- ⇒ Si le temps à niveau haut (`dutycycle x 0,125µs`) est supérieur à la période (`period x 0,5µs`) alors la `pwm` est toujours à 1 (rapport cyclique maximum de 100%).

7.3 TIMERS

Le PIC18F4320 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 µC : il compte alors le nombre de cycle instructions (un cycle instructions dure 0,5 µs). 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 prédivison de l'horloge est appelé préscaleur.

La librairie gameltimer comporte simplement les fonctions nécessaires à l'utilisation du timer1 avec une horloge interne. Pour utiliser les fonctions avancées des timers (horloge externe notamment) ou les autres timers, vous devrez utiliser les bibliothèques C18 dont la documentation est fournie avec le compilateur (cf Ch 11)

void timer1_init(char prescaleur) cette fonction permet d'initialiser le timer1 avec l'horloge interne. La résolution (la durée entre deux incrémentations du timer) est égale à un cycle instruction (0,5 µs lorsque l'on utilise l'horloge interne du µC). On peut augmenter cette durée en utilisant un

préscalaire. La résolution du timer1 est alors : $\text{prescal} \times 4 \times \text{TOSC}$ (avec TOSC, la période de l'horloge, 0,125 μs dans notre cas).

Prescalaire	Prescal	résolution	Durée maximum mesurée
0	1	0,5 μs	~ 33 ms
1	2	1 μs	~ 65 ms
2	4	2 μs	~ 131 ms
3	8	4 μs	~ 262 ms

Attention le paramètre *prescalaire* n'est pas la valeur du prescalaire mais un nombre associé (cf tableau ci-dessus)

void timer1_write(unsigned int value) Cette fonction permet d'écrire la valeur *value* dans le timer. Ce dernier recommencera alors à compter, à partir de cette valeur.

int timer1_read(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 gameltimer.h
...
void main(void)
{
    timer1_init(3); //configuration du timer1, prescal=8
    TRISA = TRISA & 0xBF; //A6 est configurée en sortie
    while(1)
    {
        PORTA=PORTA | 0x40; //mise à 1 de A6, la led s'allume
        while ( timer1_read() < 62500 ) ; //attente 125 ms
        timer1_write(0); //remise a zero du timer1
        PORTA=PORTA & 0xBF; //mise à 0 de A6, la led s'eteint
        while ( timer1_read() < 62500 ) ; //attente 125 ms
        timer1_write(0); //remise a zero du timer1
    }
}
```

8 CONNECTEUR I/O

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

VCC	1	2	GND
AN2 ou A2	3	4	AN1 ou A1
B2 ou INT2	5	6	AN3 ou A3
B1 ou INT1	7	8	T0CKI
B0 ou INT0	9	10	AN4 ou A5
AN6 ou E1	11	12	AN5 ou E0
T1CKI ou C0	13	14	AN7 ou E2
PWM1 ou C2	15	16	PWM2 ou C1

ANx ou Lx signifie que cette broche du connecteur est reliée à une broche du microcontrôleur qui peut être configurée au choix en entrée analogique ou en entrée/sortie Tout ou Rien.

PWMx ou Cx signifie que cette broche du connecteur est reliée à une broche du microcontrôleur qui peut être configurée au choix en sortie PWM (Pulse Width Modulation ou Modulation de largeur d'impulsions) ou en entrée/sortie Tout ou Rien.

Bx ou INTx signifie que cette broche du connecteur est reliée à une broche entrée/sortie Tout ou Rien du microcontrôleur qui peut déclencher ou non une interruption externe.

TxCKI caractérise une connexion à une entrée compteur rapide du microcontrôleur. La broche 13 du connecteur peut aussi être configurée en entrée/sortie Tout ou Rien C0.

Pour plus de détails sur les entrées/sorties spécifiques, référez-vous au chapitre précédent sur les périphériques internes du microcontrôleur.

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 microcontrôleur (c'est le cas quand la broche est configurée comme entrée du microcontrôleur), cela ne modifie rien. Par contre, si vous voulez que le microcontrôleur fournisse du courant (allumage d'une diode led), il vous 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.

Enfin, sur le connecteur, vous pouvez voir une broche d'alimentation 5V (VCC) et une broche reliée à la masse (GND). Ces broches sont issues de l'alimentation de la carte microcontrôleur. Elles peuvent fournir jusqu'à 200 mA environ, ce qui est suffisant pour une carte de capteurs mais certainement pas pour une carte d'alimentation de moteurs.

Sur vos cartes périphériques, si vous utilisez une alimentation propre, issue de la batterie, ne reliez pas la broche VCC.

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 μ C, 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 C18 pour établir une communication RS232 :

```
openUSART  
putcUSART  
putsUSART  
getcUSART  
getsUSART  
Busy USART  
DataRdyUSART
```

Référez-vous à la documentation de la librairie pour comprendre comment les utiliser. (cf Ch 11)

9.2 PORT SPI / 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. Référez-vous à la documentation de la librairie pour comprendre comment les utiliser. (cf Ch11)

10 TUTORIAL OUTILS DE DEVELOPPEMENT

Pour développer des applications pour les microcontrôleurs 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 IDE v7.20*
- ⇒ Compilateur C *MPLAB C18 Compiler version 3.0*
- ⇒ Programmeur / Débogueur *MPLAB ICD2*

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. Vous aurez aussi à rajouter à votre projet le fichier linker, indiquant notamment l'emplacement de votre programme et de vos données en mémoire.

10.1 CREATION D'UN PROJET ET CONFIGURATION DE MPLAB

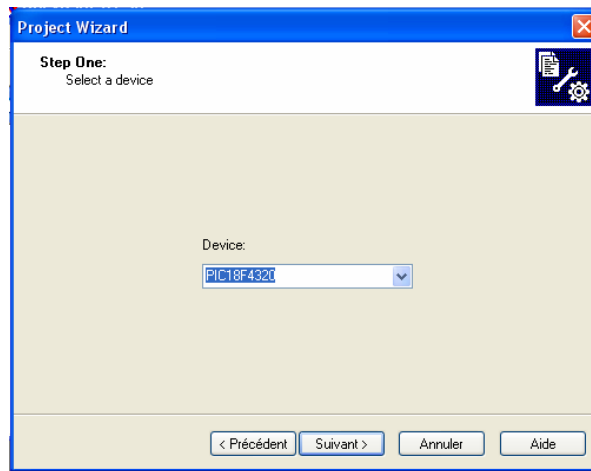
Etape 1 : Créer un dossier, y ajouter les librairies IUT

Sur le disque local (E:\), créez un dossier associé à votre projet. Copiez dans ce dossier les librairies que vous souhaitez utiliser (vous les trouverez dans le répertoire Y:\commun\Microchip\Librairies IUT.), ainsi que le fichier de configuration gamelinit.h.

En fin de séance, vous recopierez bien sûr votre dossier-projet dans votre répertoire personnel (sur U:\).

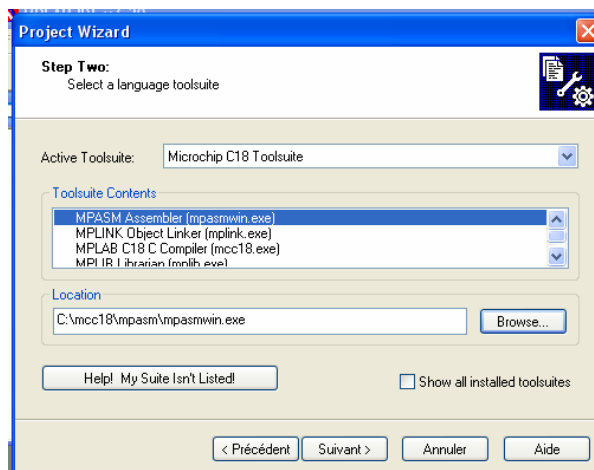
Etape 2 : Création du projet

Sélectionnez *Project>Project Wizard* pour créer un nouveau projet. Cliquez sur *suivant*. Ensuite choisissez votre microcontrôleur, appelé ici *device (composant)* ou parfois *target (cible)*. Le microcontrôleur de votre carte est un 18f4320.



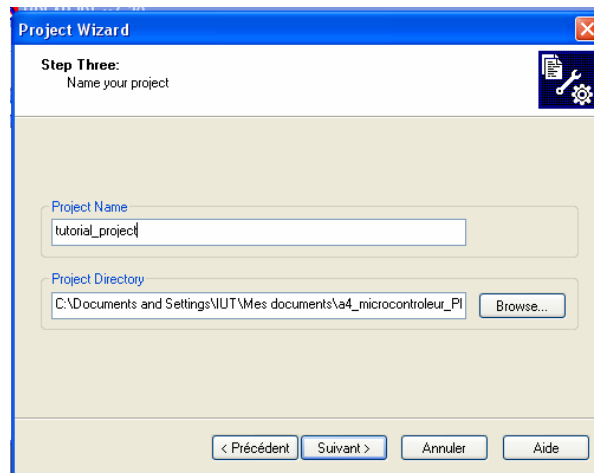
Ensuite cliquez sur *suivant*. Vous devez choisir ici quel compilateur C vous souhaitez utiliser. Dans le menu déroulant, choisissez *Microchip C18 Toolsuite*. Dans la fenêtre *Toolsuite Contents*, vous trouverez tous les éléments du compilateur. Pour chaque élément, vous devez indiquer son emplacement sur le disque. Sur les PCs de salle de projet, vérifiez que les emplacements sont les suivants :

- ⇒ MPASM (Assembleur) : d:\mcc18\mpasm\mpasmwin.exe
- ⇒ MPLINK (Linker) : d:\mcc18\bin\mplink.exe
- ⇒ MPLAB C18 (compilateur C) : d:\mcc18\bin\mcc18.exe
- ⇒ MPLIB (gestion des bibliothèques) : d:\mcc18\bin\mplib.exe



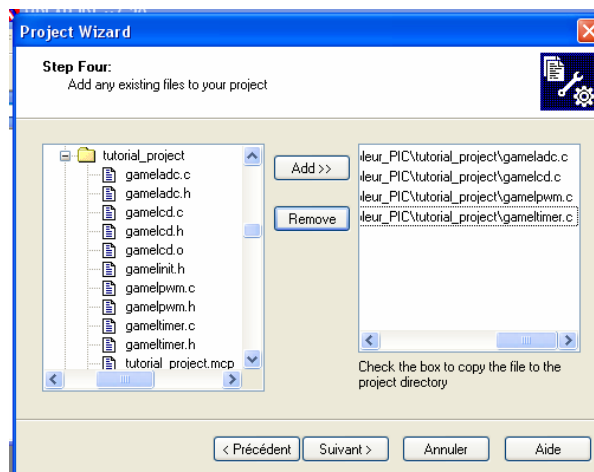
Ensuite, cliquez sur *suivant*.

Dans cette nouvelle fenêtre, vous devez indiquer le nom de votre projet et l'emplacement sur le disque du dossier associé à ce projet (Vous pouvez utiliser le bouton *Browse* pour parcourir le disque et indiquer l'emplacement). Choisissez un nom de projet en lien avec la fonction de ce projet.



Ensuite, cliquez sur *suivant*.

Dans la fenêtre qui apparaît, vous devez inclure les fichiers sources de votre projet. (les fichiers contenant les définitions des fonctions (gamelxx.c)).



Enfin, cliquez sur *suivant* puis *terminer*.

Etape 3 : Configuration des options de compilation du projet

La dernière étape consiste à fournir les options de compilation du projet. Sélectionnez *Project>Build Options>Project*. Entrez les chemins d'accès aux fichiers d'en-tête (d:\mcc18\h en salle de projet), aux bibliothèques de MPLAB C18 (d:\mcc18\lib en salle de projet) et aux fichiers linker (d:\mcc18\lkr en salle de projet) comme indiqué sur la figure ci-dessous.



Ajout de fichiers sources et linker au projet

Comme tout bon programmeur, vous allez écrire des fonctions qu'il faudra compiler. La première d'entre elles est la fonction `main()`. Les définitions de ces fonctions (appelées aussi corps des fonctions), pour être compilées, doivent être incluses dans le projet.

Pour créer un nouveau fichier, cliquez sur *File>New*. Ensuite, enregistrez ce fichier dans votre répertoire avec un nom choisi intelligemment suivi de l'extension `.c` pour un fichier contenant les définitions des fonctions (*File>Save As...*). Pour ajouter ce fichier source C, dans la fenêtre *project* (accessible par *View>project* si elle n'est pas visible), cliquez sur *Source Files* avec le bouton droit de la souris et sélectionnez *Add Files*.

Remarque 1: les fichiers d'en-tête n'ont pas besoin d'être compilés. Ils sont spécifiés comme provenant du dossier du compilateur (`#include < xxx.h >`) ou du dossier du projet (`#include " xxx.h "`), il n'est donc pas nécessaire d'ajouter ceux-ci au projet.

Le linker Mplink, qui construit votre projet pour votre microcontrôleur, a besoin d'un script de linkage. Sélectionnez *Linker Scripts* avec le bouton droit de la souris et *Add Files*. Recherchez le fichier `18f4320i.lkr` dans le dossier linker de C18 (`d:\mcc18\lkr` en salle de projet) de MPLAB C18. Ce fichier linker permet de créer un projet utilisant le débogueur MPLAB ICD 2.

10.2 COMPILATION ET CONSTRUCTION DU PROJET :

Sélectionnez *Project>Build All* 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 SUCCEEDED apparaît dans cette fenêtre de sortie. Vous pouvez faire apparaître la fenêtre de sortie en cliquant sur *View>Output*.

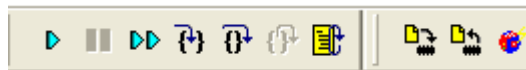
10.3 EXECUTION DU PROGRAMME ET DEBOGAGE AVEC MPLAB ICD 2 :

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 débogueur MPLAB ICD2. Ce débogueur sert aussi à envoyer le programme dans le microcontrôleur. Il est donc débogueur et programmeur.


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.

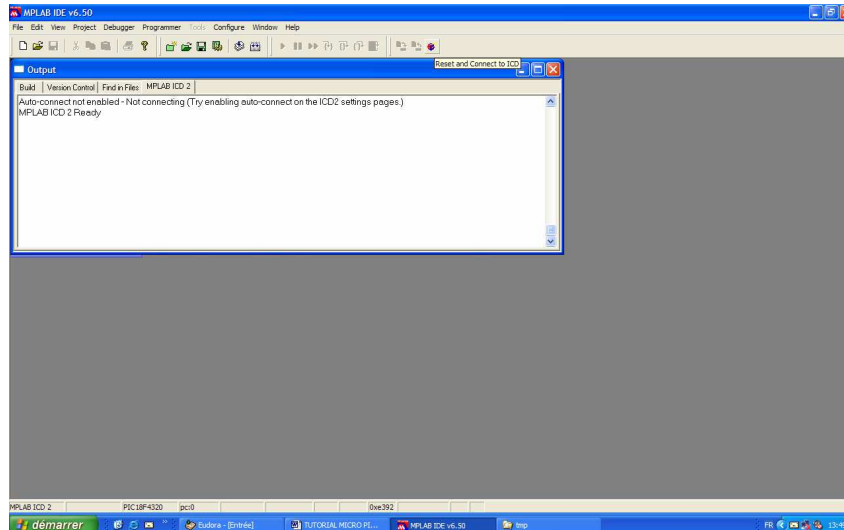
Etape 1 : Sélection du débogueur

Sélectionnez *Debugger>Select Tool>MPLAB ICD 2* pour choisir l'ICD2 comme outil de communication entre PC et μ C. Vous devez voir apparaître la barre d'outils suivante :




Etape 2 : Connexion au microcontrôleur

Pour vous connecter au microcontrôleur avec MPLAB ICD 2, sélectionnez *Debugger>Connect* ou sélectionnez l'icône 





La fenêtre de sortie montre que le programmeur MPLAB ICD 2 passe des tests de vérification et affiche le mot READY si aucune erreur ne s'est produite.

Etape 3 : Programmation

Pour programmer le microcontrôleur, sélectionnez *Debugger>Program* ou sélectionnez l'icône 


Etape 4 : Lancement du programme

Pour lancer le programme, sélectionnez *Debugger>Run*, ou l'icône 


Pour lancer le programme de nouveau, sélectionnez *Debugger>Reset* ou l'icône 


Etape 5 : Débogage

Pour retrouver les éventuelles erreurs de votre programme, vous devez utiliser les fonctionnalités suivantes, dites de débogage :

 **Halt** (F5) permet de stopper l'exécution du programme. Une flèche verte vous indique alors l'instruction qu'allait exécuter le microcontrôleur au moment où vous l'avez stoppé. Cette fonction permet de savoir dans quelle boucle s'est perdu votre microcontrôleur le cas échéant.

mode pas à pas

 **Step into** (F7) permet de lancer l'exécution de l'instruction suivante, indiquée par la flèche verte. Si l'instruction est un appel de fonction, seule la première instruction de la fonction sera exécutée.

 **Step over** (F8) permet de lancer l'exécution de l'instruction suivante, indiquée par la flèche verte. Si l'instruction est un appel de fonction, L'ensemble de la fonction sera exécuté.

Ces deux dernières fonctions *step over* et *step into* sont très utiles pour exécuter le programme pas à pas et voir ainsi l'évolution des variables instruction après instruction.

points d'arrêt

Vous pouvez ajouter aussi des **points d'arrêt** dans le fichier source de votre programme en cliquant sur le bouton droit dans la marge à gauche puis en sélectionnant *Set Breakpoint*.

Quand vous lancez le programme, il s'exécute jusqu'à ce point d'arrêt. Cela permet ainsi de n'exécuter qu'une partie du programme. Vous pouvez mettre autant de point d'arrêt que vous le souhaitez.

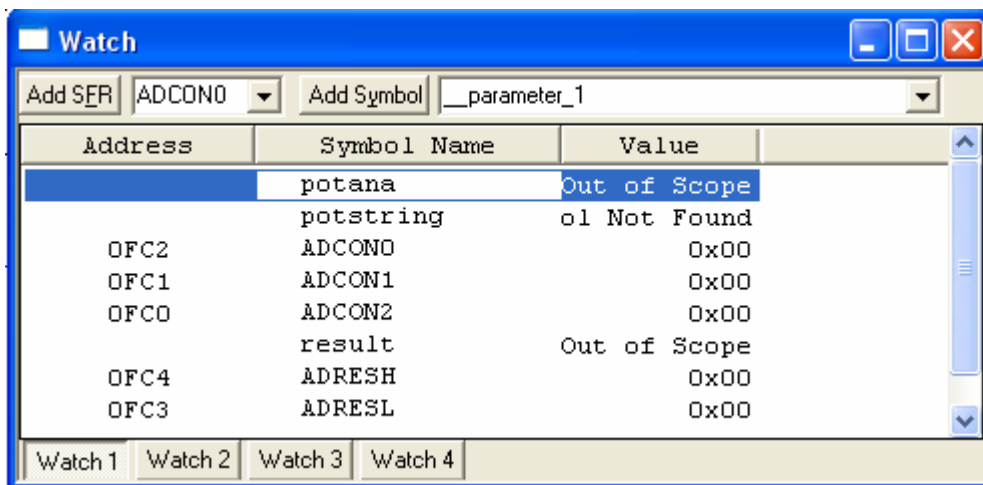
C'est très utile notamment pour sauter les temporisations (delay).

Après un reset, vous pouvez aussi mettre un point d'arrêt au début de votre main() pour qu'il s'y arrête après les étapes d'initialisation.

Semblable aux points d'arrêt, la fonction *run to cursor*, disponible dans le même menu contextuel permet de faire s'exécuter le programme jusqu'à l'endroit où est placé votre curseur.

Scrutation des variables

Pour voir l'évolution des variables en mode pas à pas, vous pouvez au choix glisser la souris sur la variable ou alors utiliser la fenêtre Watch (*View>Watch*). Vous y sélectionnez alors la variable que vous souhaitez voir et cliquez sur *Add Symbol*. Vous pouvez aussi visualiser les registres internes du microcontrôleur en choisissant dans la colonne de gauche et en cliquant sur *Add SFR*.



10.4 EXECUTION DU PROGRAMME EN MODE AUTONOME

Lorsque vous avez vérifié que votre programme fonctionne en mode connecté (avec débogueur), vous pouvez alors le lancer en mode autonome (sans débogueur, déconnecté du PC).

Pour cela, vous désélectionnez le débogueur (*Debugger>Select Tool>None*) et sélectionnez le programmeur (*Programmer>Select Programmer>MPLAB ICD2*). Le MPLAB ICD2 sert alors uniquement à programmer votre microcontrôleur.

Vous chargez le programme comme précédemment et débranchez le programmeur. Ensuite, pour relancer le programme, vous utiliserez le bouton *reset* de la carte microcontrôleur.

11 DOCUMENTATION

Dans le répertoire Y:\commun\Microchip\documentation, 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.
- ⇒ **automates.pdf** : cours sur les automates (machines à état) de Jacques Weber.
- ⇒ **automates_c.pdf** : exercices corrigés sur les automates de Jacques Weber.

Documents Microchip sur les outils

- ⇒ **MPLAB_IDE_quick_start_guide.pdf** est un guide d'apprentissage rapide de MPLAB IDE
- ⇒ **MPLAB_IDE_user_guide.pdf** est le guide d'utilisation de MPLAB IDE
- ⇒ **MPLAB_C18_GettingStarted.pdf** est un guide d'apprentissage rapide de C18 Compiler
- ⇒ **MPLAB_C18_Userguide.pdf** est le guide d'utilisation de C18 compiler
- ⇒ **MPLAB_C18_Libraries.pdf** est le guide d'utilisation des bibliothèques de C18 compiler

Documents Microchip sur le PIC18F4320

- ⇒ **PIC18F4320.pdf** est la datasheet du microcontrôleur PIC18F4320
- ⇒ **AN_programming18F4320.pdf** est une note d'application sur la programmation de ce microcontrôleur.

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 MEMOIRE

Configuration du microcontrôleur

en en-tête du fichier principal

```
#include <p18f4320.h> // ce fichier contient les adresses des différents périphériques du µC.  
#include "gamelinit.h" // ce fichier contient un partie de la configuration du µC.
```

dans la zone d'initialisation du programme

```
OSCCON=OSCCON | 0b011110000; // cette instruction configure l'horloge interne à 8 MHz.
```

Prototypes des différentes fonctions

bibliothèque gamelcd

```
void lcd_init(void)  
void lcd_gotoxy(char x, char y)  
void lcd_puti(int nombre)  
void lcd_putc(char lettre)  
void lcd_puts(char* message)
```

bibliothèque gameladc

```
void adc_init(char numero_lastchannel_used)  
int adc_read(char numero_channel)
```

bibliothèque gamelpwm

```
void pwm_init(char period, char nb_canaux)  
void pwm_setdc1(unsigned int dutycycle1)  
void pwm_setdc2(unsigned int dutycycle2)
```

bibliothèque gameltimer

```
void timer1_init( char prescalaire )  
void timer1_write( unsigned int value )  
int timer1_read( void )
```

Raccourcis clavier

Build All	Ctrl+F10
Reset	F6
Run	F9
Halt	F5
Step into	F7
Step Over	F8