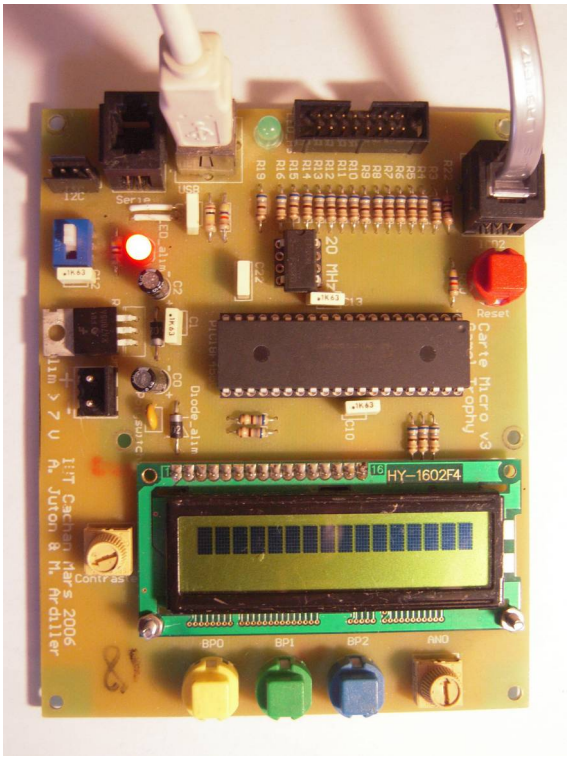
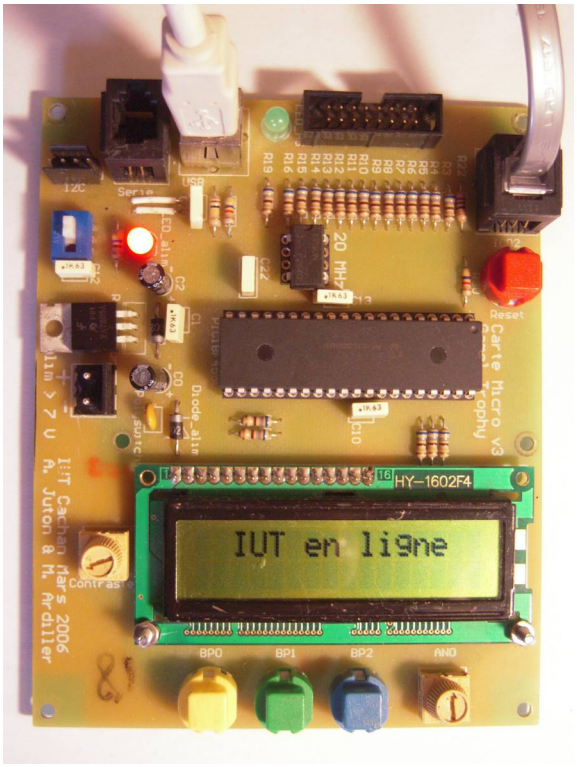
L'écran LCD

L'écran LCD est un afficheur de caractères à 2 lignes et 16 colonnes.



LCD à la mise sous tension



LCD avec le message « IUT en ligne »

Il est piloté par un contrôleur spécialisé Hitachi HD44780. Il ne vous sera pas nécessaire de connaître son fonctionnement. Le dialogue entre le micro-contrôleur et le contrôleur de l'écran est pris en charge par une bibliothèque de fonctions afin de vous simplifier l'utilisation de l'écran. Cependant, si vous voulez en savoir plus vous pouvez consulter sa documentation simple à trouver sur le web.

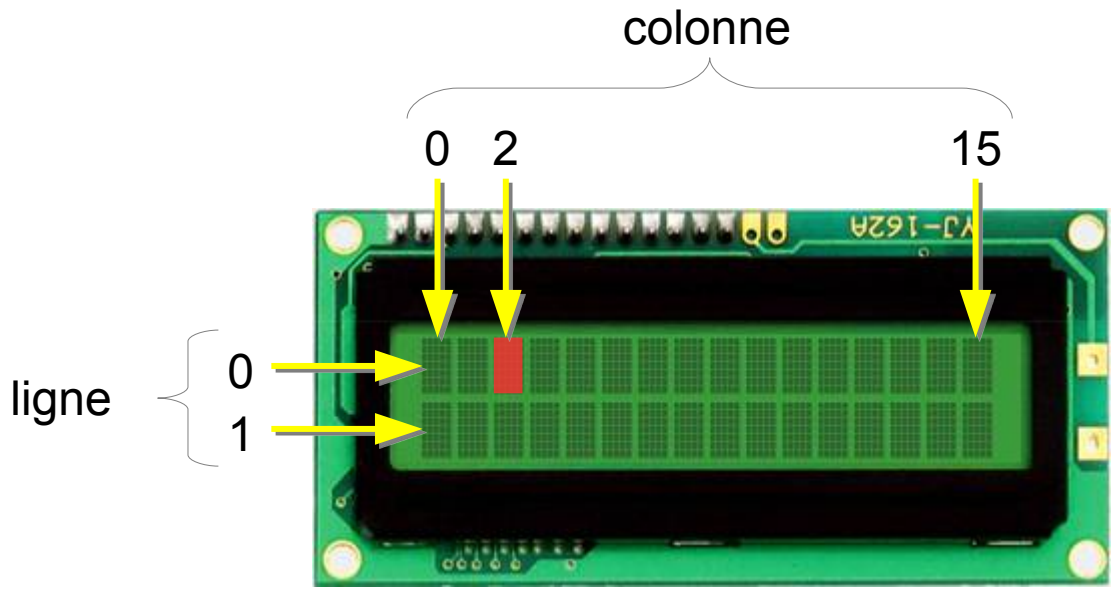
Pour utiliser la bibliothèque de l'écran LCD, il faut d'abord placer les fichiers suivants dans le répertoire du projet :

`iut_lcd.h`
`iut_lcd.c`

puis les ajouter au projet.
Après avoir inclus le fichier d'entêtes de la bibliothèque, il faut initialiser son utilisation par un appel à la fonction `lcd_init` au début du programme.
Exemple :

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

void main(void)
{
    lcd_init();

...

}
```

Avant chaque affichage, il faut placer le curseur à la position souhaitée pour l'affichage. Chaque caractère de l'écran (2 lignes et 16 colonnes) est repéré par ses coordonnées (ligne, colonne). Le caractère en haut à gauche de l'écran a pour coordonnées (0, 0) et celui en bas à droite de l'écran, (1, 15).



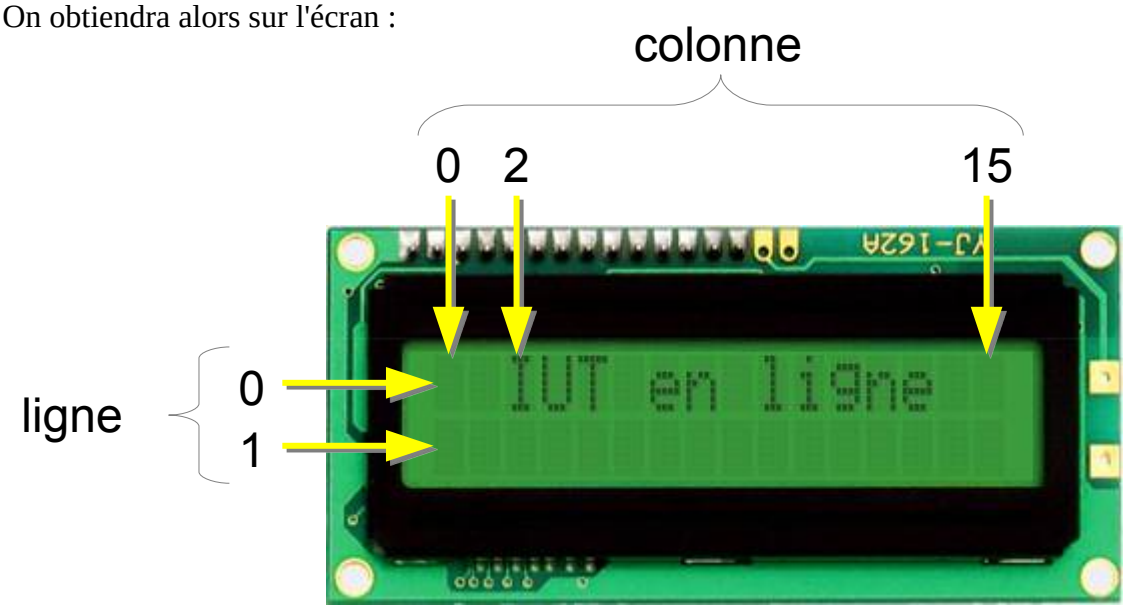
Pour placer le curseur, on utilise la fonction `lcd_position`.
Par exemple, pour positionner le curseur et commencer l'affichage sur la case rouge, ligne numéro 0 et colonne numéro 2, on écrira :

```
lcd_position(0, 2);
```

Pour effectuer un affichage, il faut utiliser la fonction `lcd_printf` assez similaire à la fonction `printf` classique du langage C.
Pour afficher le message `IUT en ligne`, on écrira :

```
lcd_printf("IUT en ligne");
```

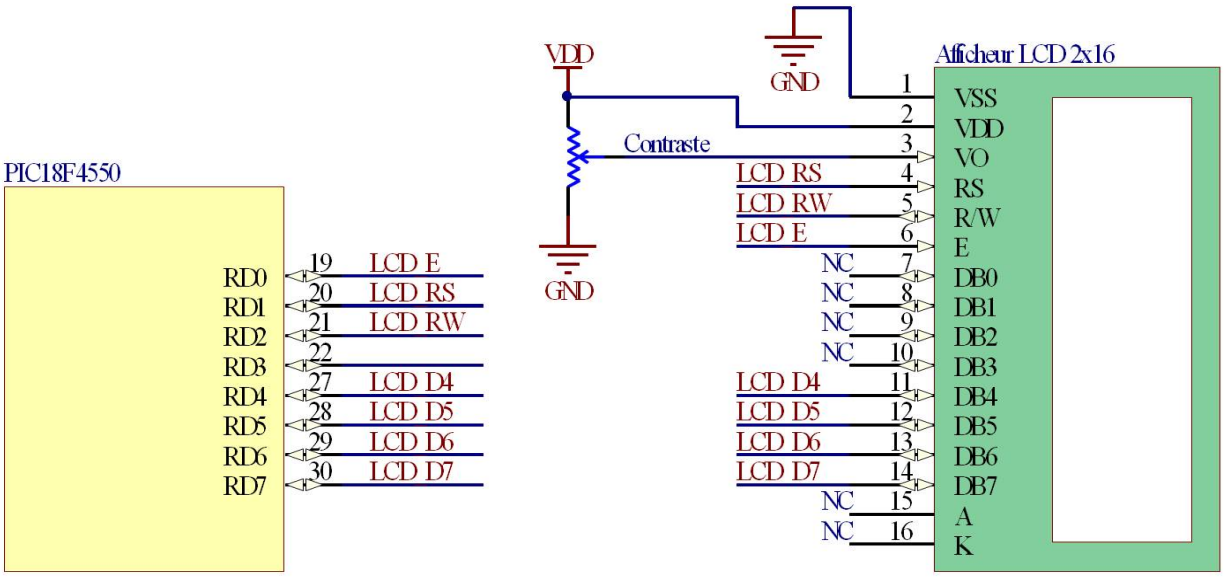
On obtiendra alors sur l'écran :



Exemple d'utilisation de l'écran LCD :
Ce programme détecte les appuis sur les boutons poussoirs et affiche un message différent pour chaque bouton.

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

void main(void)
{
    char bps;

    // initialisation
    TRISB |= 0x38;                  // B3, B4 et B5 en entrées
    lcd_init();                     // on initialise l'écran LCD

    lcd_position(0, 2);
    lcd_printf("IUT en ligne");

    // boucle infinie
    while (1) {                     // on répète infiniment
        bps = PORTB;                // on lit l'état du port B
        if ((bps & 0x08) == 0) {    // si on appuie sur BP0 (B3)
            lcd_position(1, 0);
            lcd_printf("BP0");
        } else {                    // sinon
            lcd_position(1, 0);
            lcd_printf("   ");
        }
        if ((bps & 0x10) == 0) {    // si on appuie sur BP1 (B4)
            lcd_position(1, 6);
            lcd_printf("BP1");
        } else {                    // sinon
            lcd_position(1, 6);
            lcd_printf("   ");
        }
        if ((bps & 0x20) == 0) {    // si on appuie sur BP2 (B5)
            lcd_position(1, 13);
            lcd_printf("BP2");
        } else {                    // sinon
            lcd_position(1, 13);
            lcd_printf("   ");
        }
    }
}
```

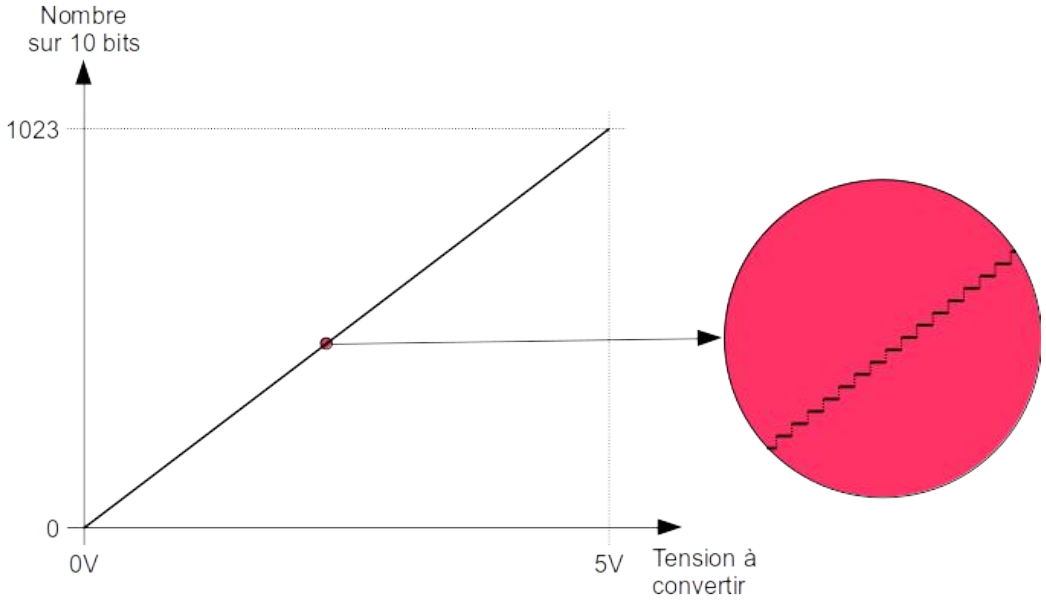
Pour information, le schéma de connexions
L'afficheur LCD est connecté sur le port D du micro-contrôleur, seul port représenté sur le schéma ci-dessous.



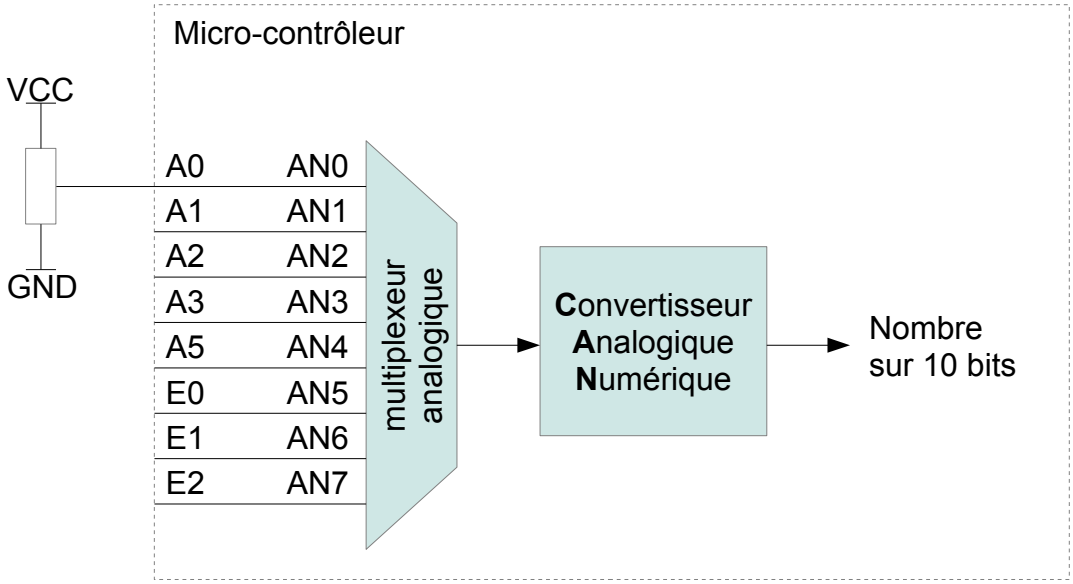
Sans entrer dans les détails, on peut remarquer que la communication entre le micro-contrôleur et le contrôleur de l'écran s'effectue par l'intermédiaire d'un bus de 7 fils. Ce bus comporte 4 fils de données (D7,D6,D5,D4) et 3 signaux de contrôle (E, RS, R/W).
Un potentiomètre permet d'ajuster le contraste de l'afficheur.

Le convertisseur analogique-numérique

Un convertisseur analogique-numérique transforme une tension présente sur une entrée analogique du micro-contrôleur en nombre entier utilisable dans le programme. La conversion suit une loi linéaire.
Notre micro-contrôleur est équipé d'un convertisseur 10 bits, c'est-à-dire que le résultat entier de la conversion sera compris entre 0 et 2^{10}-1 =1023.



Entre les entrées analogiques du micro-contrôleur et le convertisseur analogique-numérique, on trouve un multiplexeur analogique pour aiguiller le signal à convertir.



Une conversion analogique-numérique prend un certain temps, son résultat n'est pas obtenu immédiatement.
Pour effectuer une conversion analogique-numérique, il faut donc :

- positionner le multiplexeur analogique sur la voie à convertir
- attendre le temps d'acquisition nécessaire pour que la tension soit stabilisée
- lancer la conversion
- attendre la fin de la conversion
- lire le résultat de la conversion

L'ensemble de ces opérations dure environ 27µs.
Une bibliothèque de fonctions est fournie pour configurer correctement le convertisseur analogique-numérique puis pour effectuer des conversions sur les différentes voies sans avoir à détailler toutes les étapes nécessaires.
Pour utiliser la bibliothèque du convertisseur analogique-numérique, il faut d'abord placer les fichiers suivants dans le répertoire du projet :

```
iut_adc.h
iut_adc.c
```

puis les ajouter au projet.
Après avoir inclus le fichier d'entêtes de la bibliothèque, il faut initialiser son utilisation par un appel à la fonction `adc_init` au début du programme. Le paramètre à passer à cette fonction d'initialisation correspond au numéro du dernier canal analogique utilisé. On passera « zéro » pour se servir seulement du canal AN0 sur lequel est connecté un potentiomètre.
Exemple :

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

void main(void)
{
    adc_init(0);
    ...
}
```

Pour effectuer une conversion, il suffit d'appeler la fonction `adc_read` avec comme paramètre le numéro du canal à convertir (0 pour AN0, par exemple) et de ranger le résultat dans une variable. Un appel à la fonction dure environ 27µs.

Voici un exemple qui effectue la conversion de la voie du potentiomètre (AN0) et affiche le résultat sur l'écran LCD sous deux formes : en entier (de 0 à 1023) et en volts (de 0 à 5V).

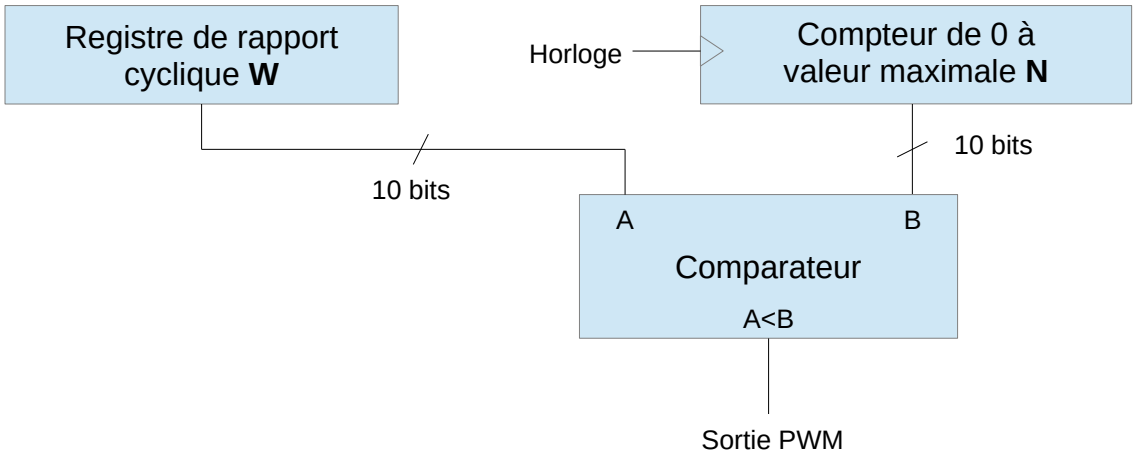
```
#include <xc.h>
#include "iut_lcd.h"
#include "iut_adc.h"

void main(void)
{
    int potar = 0;
    float tension;

    // initialisation
    lcd_init();
    adc_init(0);

    lcd_position(0, 2);
    lcd_printf("IUT en ligne");

    // boucle infinie
    while (1) {
        potar = adc_read(0);
        lcd_position(1, 0);
        lcd_printf("%4d", potar);
        tension = potar * (5.0 / 1023.0);
        lcd_position(1, 10);
        lcd_printf("%5.3fV", tension);
    }
}
```

Remarquez l'utilisation de la fonction `lcd_printf` pour les affichages :

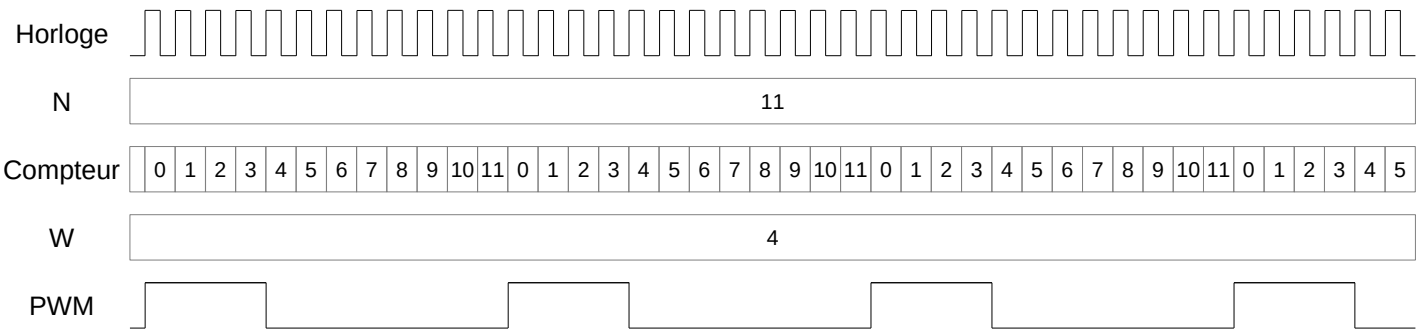
- Pour une variable de type `int` dont la valeur tient toujours sur 4 caractères (0 à 1023), on utilise le format `%4d`
- Pour une variable de type `float` dont la valeur est toujours positive et comprise entre 0 et 5, si on souhaite un affichage à 3 chiffres après la virgule, il faut au maximum 5 caractères, on utilise le format `%5.3f`

Signal à rapport cyclique variable (Pulse Width Modulation)

Un signal PWM est une sortie du micro-contrôleur qui permet, par exemple, de commander un hacheur. En général, la fréquence est fixée et on fait varier le rapport cyclique du signal.
Réaliser un signal PWM avec un circuit numérique nécessite un compteur qui fonctionne en continu. Un tour de compteur correspond à une période du signal PWM. La fréquence peut donc être réglée par le choix de l'horloge du compteur et par la valeur maximale du compteur (N sur la figure). Un registre associé à un comparateur permet de fixer le rapport cyclique. A chaque fin de comptage, la sortie PWM est mise à 1 et elle est mise à 0 quand le compteur atteint la valeur fixée par le registre de rapport cyclique (W).



Exemple de chronogramme :



En réalité, la structure qui se trouve dans le micro-contrôleur est un peu plus compliquée. Seulement les 8 bits de poids forts de la valeur maximale du compteur sont configurables.

Une bibliothèque de fonctions est fournie pour configurer simplement les deux sorties PWM du micro-contrôleur.
Pour utiliser la bibliothèque des sorties PWM, il faut d'abord placer les fichiers suivants dans le répertoire du projet :

```
iut_pwm.h
iut_pwm.c
```

puis les ajouter au projet.
Après avoir inclus le fichier d'entêtes de la bibliothèque, il faut initialiser son utilisation par un appel à la fonction `pwm_init` au début du programme.

Le premier paramètre à passer à cette fonction permet de choisir la fréquence à générer. La valeur est codée sur 8 bits (donc entre 0 et 255, notée N dans la suite) et la fréquence est calculée grâce à la formule suivante :

$$\text{fréquence} = 3.10^6 / (N + 1)$$

Le second paramètre permet de choisir le nombre de sorties (1 ou 2). Si on choisit 1, seule PWM1 (broche C2) est activée, si on choisit 2, les sorties PWM1 et PWM2 (broche C1) sont activées. On peut remarquer que les deux sorties fonctionnent obligatoirement à la même fréquence.
Par exemple, pour activer les deux sorties PWM avec une fréquence de 20kHz, on écrira le code suivant :

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

void main(void)
{
    pwm_init(149, 2);

...

}
```

Pour modifier le rapport cyclique, deux fonctions sont disponibles, une pour chaque sortie PWM. La fonction pwm_setdc1 contrôle le rapport cyclique de PWM1 et pwm_setdc2 celui de PWM2.
Le paramètre à passer à ces fonctions correspond au nombre de cycles à l'état haut. Il est codé sur 10 bits (donc de 0 à 1023, noté W dans la suite). Le rapport cyclique obtenu est lié à la configuration de la fréquence par la formule suivante :

$$\text{rapport cyclique} = W / (4 (N + 1))$$

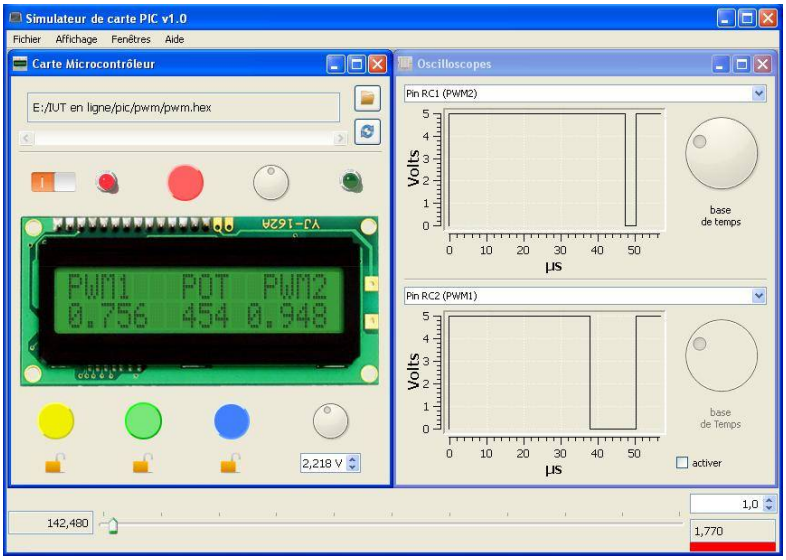
Si W est supérieur à 4 (N + 1) alors le rapport cyclique vaut 1.
Avec la configuration précédente (N=149), la valeur de W doit être comprise entre 0 et 600. On donne ci-dessous quelques valeurs de rapport cyclique en fonction de W :

N=149	W	Rapport cyclique
	0	0
	150	0,25
	300	0,5
	450	0,75
	600	1

Voici un exemple qui effectue la conversion de la voie du potentiomètre (AN0) et utilise la valeur obtenue pour modifier le rapport cyclique de PWM1. Le rapport cyclique de PWM2 évolue de manière opposée.

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

void main(void)
{
    int potar = 0;
    float a1, a2;

    // initialisation
    lcd_init();
    adc_init(0);
    pwm_init(149, 2);

    lcd_position(0, 0);
    lcd_printf("PWM1   POT   PWM2");

    // boucle infinie
    while (1) {
        potar = adc_read(0);
        pwm_setdc1(potar);
        pwm_setdc2(1023 - potar);
        a1 = potar / 600.0;
        if (a1 > 1) a1 = 1;
        a2 = (1023 - potar) / 600.0;
        if (a2 > 1) a2 = 1;
        lcd_position(1, 0);
        lcd_printf("%5.3f", a1);
        lcd_position(1, 6);
        lcd_printf("%4d", potar);
        lcd_position(1, 11);
        lcd_printf("%5.3f", a2);
    }
}
```

Avec le simulateur, pour afficher les signaux PWM dans un oscilloscope virtuel, il faut aller dans le menu *Fenêtre* puis *Oscilloscopes*.



Utilisation d'un timer

Pour compter le temps ou des événements, les micro-contrôleurs disposent de blocs spécialisés, des timers. Un timer est constitué d'un compteur dont on peut ajuster l'horloge, lire la valeur et écrire la valeur. Les timers sont capables de générer des interruptions mais nous ne verrons pas cet aspect là dans ce cours.
Nous allons donner un exemple d'utilisation du Timer0 du PIC18F4550. Pour programmer ce timer, nous nous servirons d'une bibliothèque de fonctions. Il faudra donc inclure le fichier `iut_timers.h` avec le code suivant :

```
#include <iut_timers.h>
```

La fonction qui permet d'initialiser le timer est `OpenTimer0`.

Exemple d'initialisation du timer0 :

```
OpenTimer0(    TIMER_INT_OFF &     // désactive les interruptions
               T0_16BIT &          // configure sur 16 bits
               T0_SOURCE_INT &     // utilise l'horloge interne
               T0_PS_1_256         // divise l'horloge par 256
          );
```

Dans nos exemples, nous désactiverons toujours les interruptions. Nous utiliserons toujours le timer sur 16 bits (comptage de 0 à 65535) car sur 8 bits, le comptage est plus limité (0 à 255). Nous utiliserons l'horloge interne et alors, dans ce cas, le timer reçoit une horloge à 12 MHz pour notre carte électronique. Enfin, nous choisirons le diviseur d'horloge (prescale value) en fonction du temps à compter. Pour le timer0, les valeurs possibles sont 1, 2, 4, 8, 16, 32, 64, 128 ou 256. Pour compter des temps « longs », il est préférable de choisir 256.

Quand le timer est initialisé, il compte continuellement. Il faut donc pouvoir fixer sa valeur à un instant particulier dans le programme, par exemple pour pouvoir le mettre à 0. Pour cela, nous utiliserons la fonction `WriteTimer0`.
Pour mettre le timer à 0, il suffit d'écrire :

```
WriteTimer0(0);
```

Pour lire le timer, la fonction `ReadTimer0` retourne la valeur. On peut ainsi effectuer une boucle pour attendre qu'un certain temps se soit écoulé. Avec la configuration précédente, le timer compte à 12 MHz divisé par 256, c'est-à-dire à 46875 Hz. En partant de 0, une seconde sera écoulée quand il atteindra 46875. Pour attendre une seconde, on écrira :

```
while (ReadTimer0() < 46875);
```

De la même manière, pour attendre une demi seconde, il faudra atteindre 23438. On écrira :

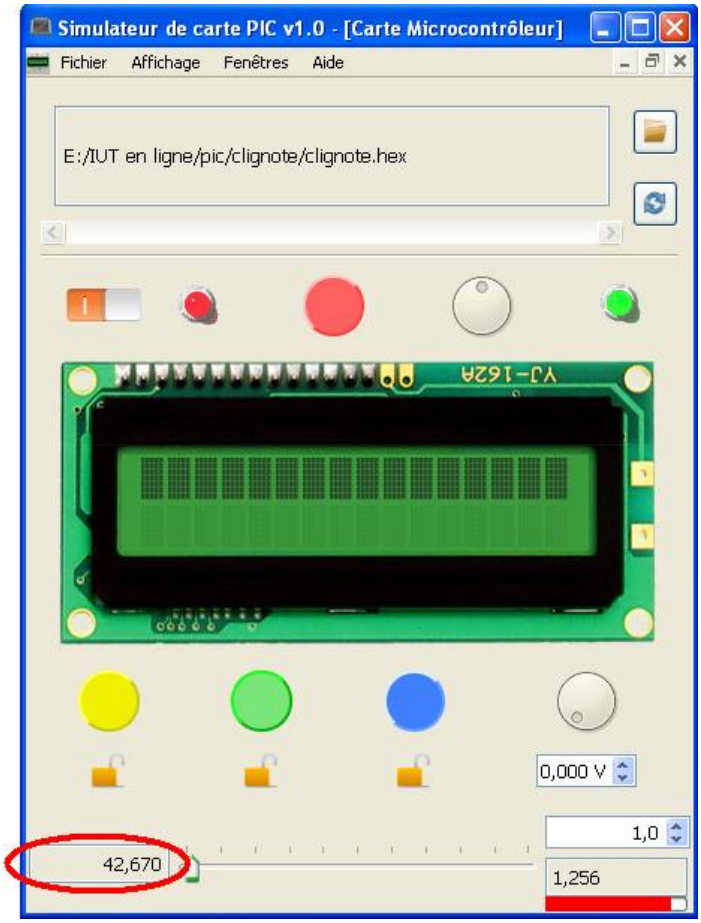
```
while (ReadTimer0() < 23438);
```

On dit que cette méthode est « bloquante » c'est-à-dire qu'aucun autre traitement n'est effectué pendant le temps d'attente.

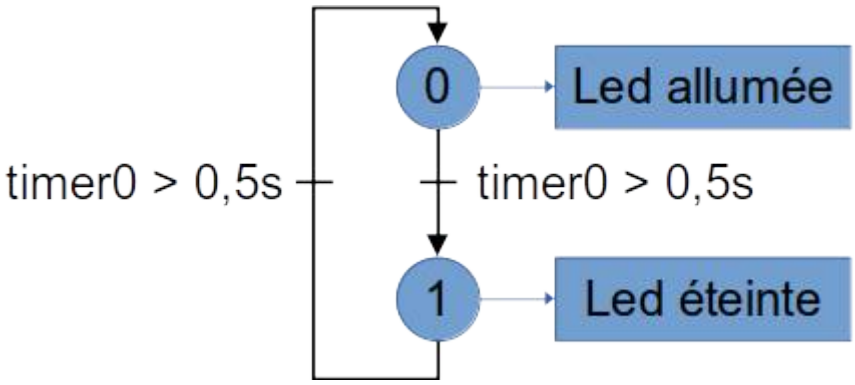
Pour faire clignoter la led raccordée sur A6 à la fréquence de 1 Hz (éteinte pendant 0,5 s et allumée pendant 0,5 s), on peut écrire le programme suivant :

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

void main(void)
{
    // initialisation
    TRISA &= 0xBF;    // A6 en sortie
    OpenTimer0(
               TIMER_INT_OFF &     // désactive les interruptions
               T0_16BIT &          // configure sur 16 bits
               T0_SOURCE_INT &     // utilise l'horloge interne
               T0_PS_1_256         // divise l'horloge par 256
               );

    // boucle infinie
    while (1) {                              // on répète infiniment
        WriteTimer0(0);                      // mise à 0 du timer
        while (ReadTimer0() < 23438);        // attente 0,5 s
        PORTA |= 0x40;                       // on éclaire la LED
        WriteTimer0(0);                      // mise à 0 du timer
        while (ReadTimer0() < 23438);        // attente 0,5 s
        PORTA &= 0xBF;                       // on éteint la LED
    }
}
```

Attention, si le temps écoulé ne semble pas correspondre exactement, la simulation n'est peut-être pas parfaitement en temps réel en fonction des performances de votre ordinateur. Le temps écoulé en simulation pour le micro-contrôleur est indiqué en bas à gauche de la fenêtre du simulateur.



On peut aussi utiliser une méthode « non bloquante » en programmant la machine à états suivante :



Code correspondant :

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

void main(void)
{
     char etat = 0;

     // initialisation
     TRISA &= 0xBF;      // A6 en sortie
     OpenTimer0(
          TIMER_INT_OFF &      // désactive les interruptions
          T0_16BIT &           // configure sur 16 bits
          T0_SOURCE_INT &      // utilise l'horloge interne
          T0_PS_1_256          // divise l'horloge par 256
     );

     while (1) {                                   // on répète infiniment
          switch (etat) {
               case 0:
                    PORTA |= 0x40;                 // on éclaire la LED
                    if (ReadTimer0() >= 23438) {   // si attente terminée
                         WriteTimer0(0);           // mise à 0 du timer
                         etat = 1;                 // on passe à l'état 1
                    }
                    break;
               case 1:
                    PORTA &= 0xBF;                 // on éteint la LED
                    if (ReadTimer0() >= 23438) {   // si attente terminée
                         WriteTimer0(0);           // mise à 0 du timer
                         etat = 0;                 // on passe à l'état 0
                    }
                    break;
          }
     }
}
```