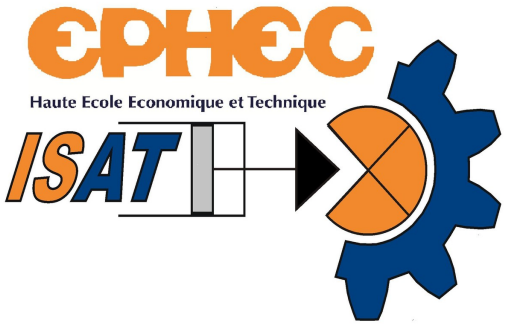


UE A211 : Système Embarqué I

Professeur : COSTA Emile

RAPPORT DE LABORATOIRE

Etudiants : FINYA Alban
KORKUT Caner

TP1 - Prise en main de *MikroC* et *Proteus*

Table des matières

1 Objectif du TP **2**

2 Introduction **2**

3 Schéma de principe **4**

4 Algorigramme **5**

5 Code source **6**

6 Analyse du code source **7**

7 Conclusion **9**

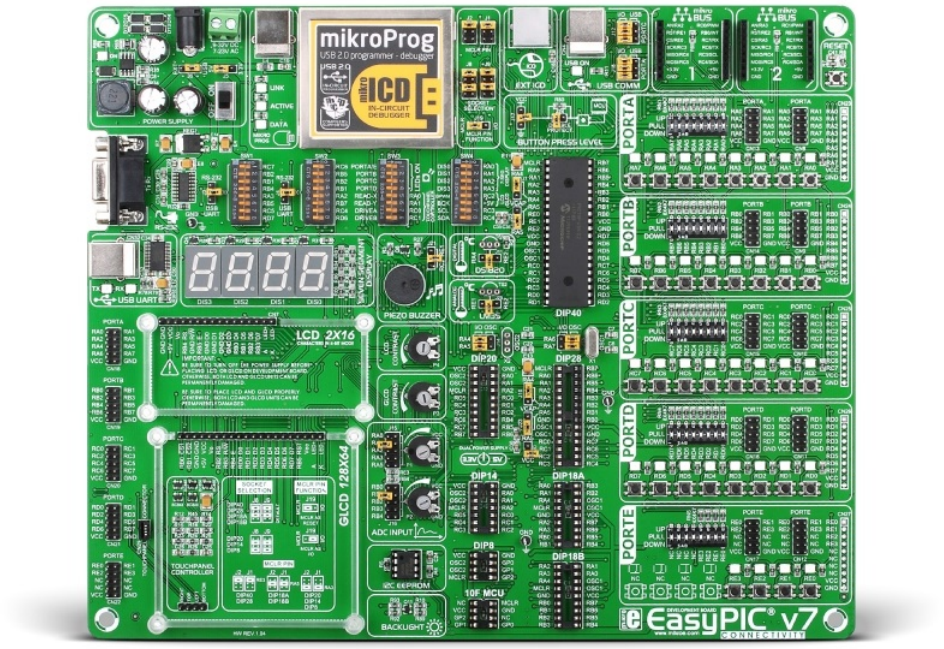


FIGURE 1 – Carte de développement EasyPic 7

1 Objectif du TP

Plusieurs concepts ont été introduits lors de cette séance concernant les microcontrôleurs (que l'on notera μC tout au long des séances). Il a été question d'avoir une approche tantôt matérielle et tantôt logicielle du matériel de cours. L'objectif principal de cette manipulation fut de se familiariser avec l'environnement de programmation *MikroC* et ainsi que la carte de développement *PicKit7*. Ainsi, nous avons pu aborder certains concepts comme la programmation via un programmateur, la nécessité de faire parfois appel à un bootloader, etc. Dans un second temps, il nous a été demandé de réaliser un travail de conception de schéma de principe via le logiciel *Proteus* mis à notre disposition à domicile.

2 Introduction

Dans la programmation orientée systèmes embarqués, l'allumage d'une LED représente un peu le `Hello World !` de la programmation traditionnelle. C'est pourquoi, pour cette première séance de laboratoire, nous avons été amenés à faire clignoter une série de LEDs situés sur la carte de développement *PicKit*.

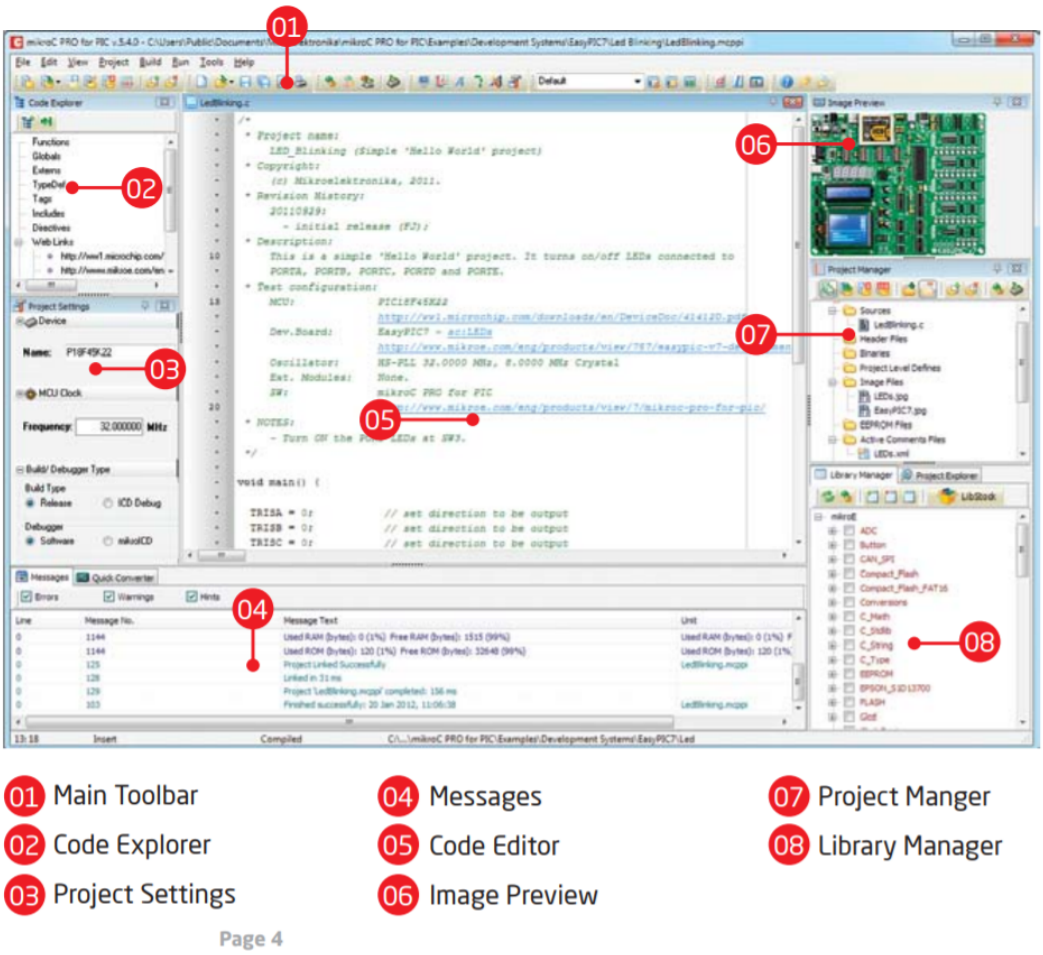


FIGURE 2 – Environnement de développement *MikroC* de MikroElektronika

Pour ce faire, nous avons eu à notre disposition sur la plateforme d'apprentissage Moodle un document [1] disponible au format `.PDF` sur le site officiel de MIKROELEKTRONIKA.

Une fois le code rédigé et compilé, nous avons pu injecter notre programme (fichier au format `.hex`) via le programmateur présent dans la carte de développement. Ensuite, il nous a été demandé de réaliser l'allumage des LEDs dit en *chenille*, à savoir les unes après les autres.

1. *Creating the first project in MikroC PRO for PIC*, 2012, MikroElectronika

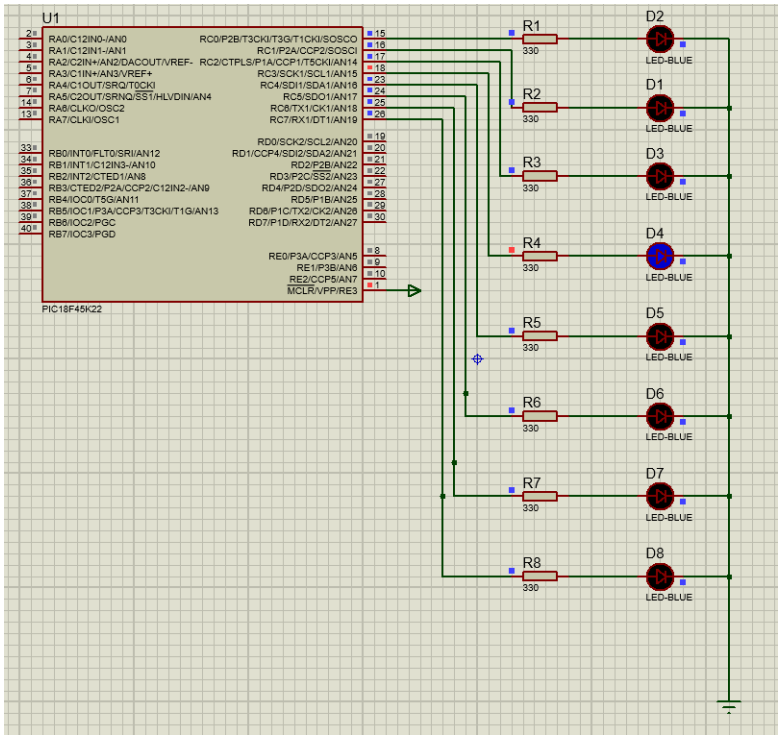
3 Schéma de principe

Le schéma de principe ainsi que la simulation du programme a été réalisé à domicile. Le logiciel **Proteus** nous a été nécessaire afin d'y parvenir. Pour rappel, il s'agit d'un logiciel de conception de CI et de simulation.

Les composants que nous avons utilisés pour réaliser cette manipulation sont les suivants :

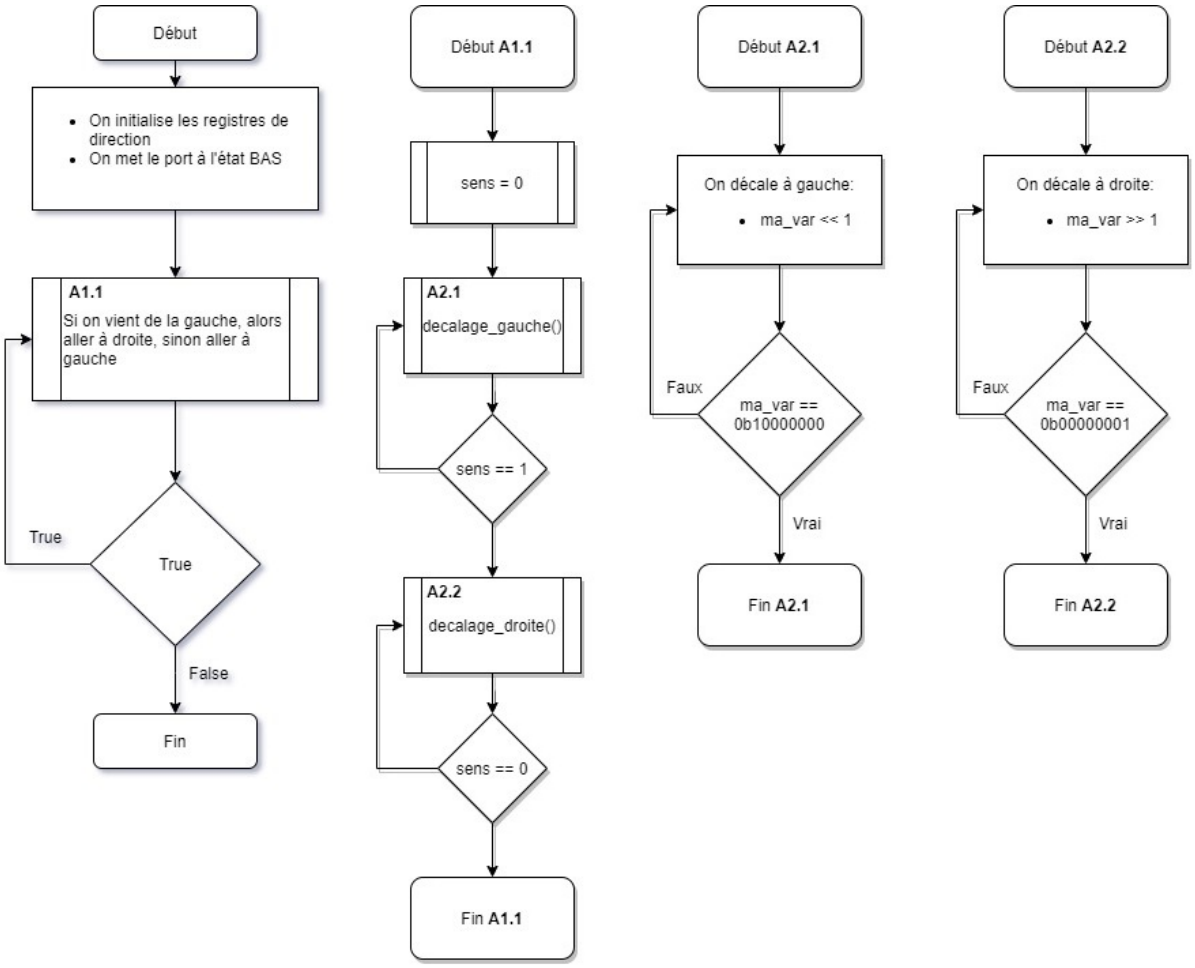
- PIC18F45K22 ;
- 8 LEDs bleues ;
- 8 Résistances 330 Ω ;
- Alimentation 5V.

Il a ensuite été nécessaire de d'injecter notre fichier compilé *Chenillard.hex* au sein du μC. Pour ce faire, nous avons dû sélectionner les paramètres du PIC18F45K22 et de spécifier le fichier source lors de l'exécution de la simulation. Après avoir relié les différents composants entre eux, il nous a fallu raccorder les pins du PORTC afin d'avoir une simulation fonctionnelle :



4 Algorigramme

Nous avons vu en cours qu'il nous fallait créer une boucle infinie contenant notre routine. Notre programme est donc représenté par un algorigramme de différents niveaux. On y retrouve 3 niveaux **A0**, **A1** et **A2** représentés respectivement par les algorigrammes $A0, A1.1, A2.1$ et $A2.2$:



Algorigramme du programme principal

5 Code source

```
// Variables globales
#define TRUE 1
#define FALSE 0

unsigned char mesLeds = 0;
int sens = 0;

// Creation des fonctions
void decalage_droite(){
    while (mesLeds != 0x01) { // Tant que mesLeds != 0b00000001
      mesLeds = mesLeds>>1; // Decalage a droite
      LATC = mesLeds;
      Delay_ms(200);
    }
}

void decalage_gauche(){
    while (mesLeds != 0x80) { // Tant que mesLeds != 0b10000000
      mesLeds = mesLeds<<1; // Decalage a gauche
      LATC = mesLeds;
      Delay_ms(200);
    }
}

// Fonction principale:
void main() {

    TRISC = 0;  // Initialisation du registre de direction du PORT(B)
    en SORTIE digitale
    LATC = 0; // Initialisation des bits du PORT(B) a l'etat BAS
    mesLeds = 1; // Initialisation de l'etat initiale de la chenille
    LATC = mesLeds; // Initialisation de l'etat du PORT(B) a la
    variable "mesLeds"

    // Programme:
    while(1) {

      if (sens == TRUE) { // Si sens est a 1 on decale a gauche
        decalage_gauche();
        sens = FALSE;
      }
```

```
40        else { // Sinon on decale a droite
41          decalage_droite();
42          sens = TRUE;
43        }
44         // Basculement dans l'autre etat
45      }
46  }
```

6 Analyse du code source

Tout d'abord nous pouvons constater que le type booléen n'est pas pris en charge par le compilateur MikroC que nous avons. Pour contourner ce problème, nous avons défini nos propres booléens à l'aide du mot-clé *define*. Pour ce qui est des variables, celles-ci sont déclarées au début du code, avant la fonction principale `void main()` :

```
1  // Variables globales
2  #define TRUE 1
3  #define FALSE 0
4
5  unsigned char mesLeds = 0;
6  int sens = 0;
```

C'est dans cette fonction que tient le programme. On y initialise en premier lieu les registres et leurs états comme suit :

```
8  // Fonction principale:
9  void main() {
10
11     TRISB = 0;  // Initialisation du registre de direction du PORT(B)
       en SORTIE digitale
12     LATB = 0; // Initialisation de tous les bits du PORT(B) a l'etat
       BAS
13     mesLeds = 1; // Initialisation de l'etat initiale de la chenille
14     LATB = mesLeds; // Initialisation de l'etat du PORT(B) a la
       variable "mesLeds"
```

Ensuite, nous avons le programme principal qui tient dans une boucle infinie `while(1)`. Dans ce dernier, nous y initalisons nos deux fonctions de la manière suivante :

```
24 // Creation des fonctions
25 void decalage_droite(){
26     while (mesLeds != 0x01) { // Tant que mesLeds != 0b00000001
```

```
27        mesLeds = mesLeds>>1; // Decalage a droite
28        LATC = mesLeds;
29        Delay_ms(200);
30      }
31  }
32
33  void decalage_gauche(){
34      while (mesLeds != 0x80) { // Tant que mesLeds != 0b10000000
35        mesLeds = mesLeds<<1; // Decalage a gauche
36        LATC = mesLeds;
37        Delay_ms(200);
38      }
39  }
```

Enfin, on peut écrire notre programme dans la boucle `while(1)` qui finalement fait moins de dix lignes :

```
24      // Programme:
25      while(1) {
26        if (sens == TRUE) { // Si sens est a 1 on decale a gauche
27          decalage_gauche();
28          sens = FALSE; // Basculement dans l'autre etat
29        }
30        else { // Sinon on decale a droite
31          decalage_droite();
32          sens = TRUE; // Basculement dans l'autre etat
33        }
34
35      }
```

7 Conclusion

Tout d'abord il nous a semblé pertinent de **comparer la syntaxe utilisée dans ce code avec le langage Arduino** déjà utilisé précédemment dans notre cursus. On peut constater que `TRIS(x)` remplace en quelque sorte la fonction `pinMode(x)` que nous avions auparavant. Il est à noter qu'on s'addresse à des registres de 8 bits et non plus à des pins directement. Pour s'addresser à une pin : $LAT.F(n) = 0$ (où 0/1 représente l'état de la pin) [2]. On remarque également que tout comme dans la plateforme Arduino, une étape de compilation est nécessaire avant d'injecter le programme (au format hexadécimal `.hex`) dans le microcontrôleur.

Ensuite, cette première approche nous a permis de nous familiariser avec l'environnement de développement progressivement. La première étape étant de faire clignoter les LEDs nous a permis de montrer l'importance de découper en *en sous-problèmes* le cahier de charges de départ. Ainsi, il fut facile de réaliser la chenille une fois les bases maîtrisées.

Enfin, la carte de développement EasyPiC 7 nous a été d'une utilité considérable sachant que le programmateur et les connexions y sont intégrés. Dès lors, nous pouvons optimiser notre énergie et notre temps dans notre apprentissage. Sans cette carte, il nous aurait été indispensable d'utiliser un breadbord, des résistances et des LEDs afin de réaliser l'expérience.

2. Avec $0 \leq n \leq 7$