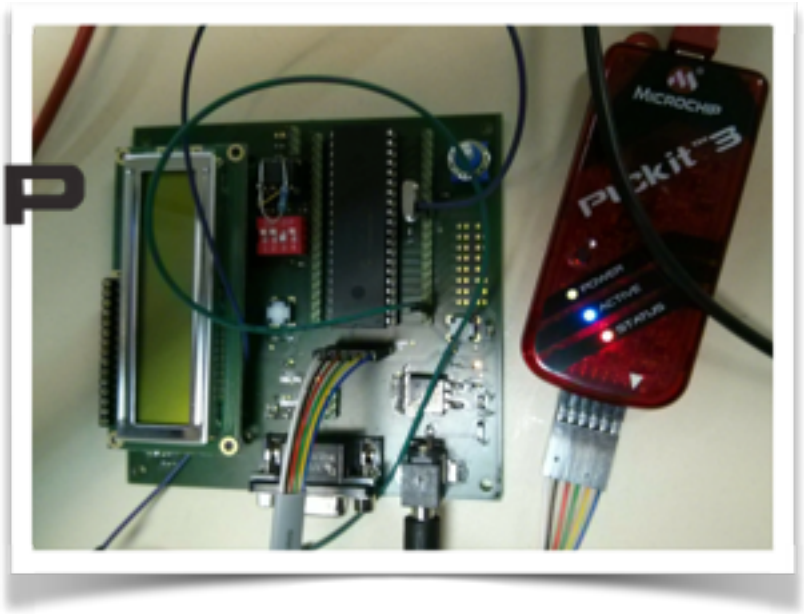
Outils de développement logiciel

M1B Série 1 Groupe D



BONZEL Pierre EVEILLARD Antoine RENOLLEAU Cyril

I. Généralités

Dans ce projet d'outils de développement logiciel nous avions comme objectif de réaliser plusieurs programmes grâce au logiciel MPLAB. Ces programmes avaient pour but de gérer l'allumage des leds d'une maquette comportant un PIC18F46K22, un écran LCD, un potentiomètre et 2 boutons poussoirs.

Dans un premier temps, avec l'aide d'un encadrant, nous avons codé un bandeau de leds clignotant. Puis nous avons réaliser un chenillard simple suivi d'un système de type K2000 : un chenillard ayant deux sens. Enfin nous avons programmé le PIC de tel sorte que les leds représentent une barre de chargement.

Ces différents programmes nous ont amenés à mieux maitriser le logiciel ainsi que la programmation du PIC et nous avons alors pu commencer notre sujet qui est de réaliser un dé électronique grâce à un chenillard utilisant les 6 leds du port A. Lorsque l'on appuie sur le bouton poussoir le défilement des leds s'arrête puis redémarre lors de la 2ème activation du bouton.

Les différents programmes que nous avons cités possèdent une base commune qui est une fonction pour faire clignoter les leds avec l'aide d'une temporisation. Nous avons souhaité maitriser précisément la durée de la temporisation et non pas utiliser une boucle avec laquelle on ne peut précisément la déterminer. Pour cela nous utilisons les macros __delay_ms() ou encore __delay_ns() grâce à l'ajout dans le main.h d'une commande définissant la variable _XTAL_FREQ fréquence de l'oscillateur :

- **#define _XTAL_FREQ 4000000**

Cette fonction possède des limites, en effet il n'est pas possible avec une fréquence XTAL de 400000Hz d'effectuer un delay supérieur à 200ms, car au delà, l'argument est déclaré comme trop large et une erreur est retournée à la compilation. Cependant ici ce n'est pas gênant car les 200 ms suffisent à rendre visible le clignotement des leds.

Pour programmer le Pic nous avons besoin de son fichier de configuration. Ce fichier est généré par MPLAB X après avoir sélectionner les réglages désirés. Par rapport au fichier généré par défaut nous avons modifié les paramètres :

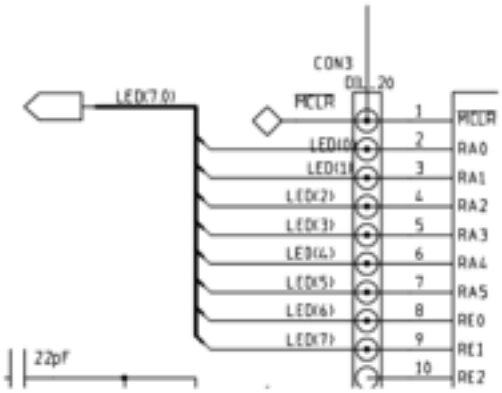
- **#pragma config FOSC = XT** : Indique que l'oscillateur est externe et est de type cristal/résonateur.
- **#pragma config WDTEN = OFF** : Désactivation du watchdog timer

Pour tout les exercices suivants, le fichier config présent en annexe 1 sera utilisé.

II. Bandeau de leds clignotant

Le but de ce premier exercice est de faire s'allumer et s'éteindre périodiquement les 8 diodes présentes sur la maquette. Comme on peut le voir sur l'extrait du schéma technique de cette dernière, les leds sms sont connectées au PIC sur le port A et E. Respectivement :

- Leds 0 à 5 : PORTA - Broches RA0 à RA5
- Leds 5 à 7 : PORTE - Broches RE0 & RE1



Pour allumer tout le port, et donc accéder aux deux leds du port E, on va utiliser la notion de masque. La réalisation et l'exécution de ceux-ci se trouvent dans la méthode leds.
Ce code comporte une deuxième fonction essentielle (program) qui permet de charger la valeur 00 pour éteindre toutes les leds puis de les rallumer après une temporisation en chargeant la valeur FF.

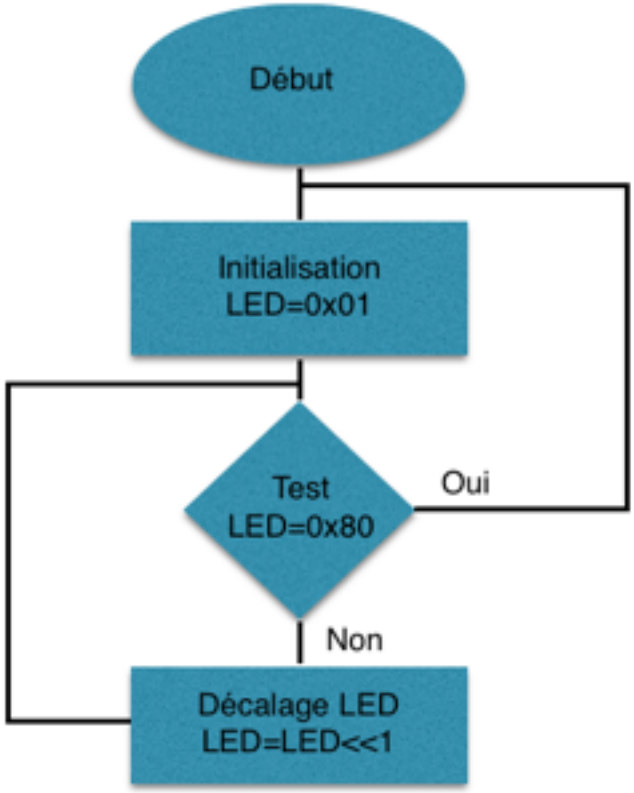
On peut retrouver ci-dessous le main.c du programme commenté.
Le main.h présentant moins d'intérêt se trouve en annexe 2.

```c
//Importation des librairies & hearders
#include <stdio.h>              //Input-Output
#include <stdlib.h>             //Standard
#include <pic18f46k22.h>        //Correspondant a notre PIC
#include "config.h"
#include "main.h"

#define out 0x00
#define in 0xFF

void setup(){
    TRISA=out;                  //Initialisation du port A en sortie
    TRISE=out;                  //Initialisation du port E en sortie
          }
void leds(unsigned char port){ //Prend en argument, la valeur en binaire a afficher sur les 8 leds
    LATA=port; //Permet de récupérer la valeur du port A dans LATA : bien en cas de cours circuit
    if(port & 0x40){ //Masque selectionnant le bit 6 de l'agument d'entré port
        LATE=LATE | 0x01; //Masque qui correspond a RE0
    }
    else LATE=LATE & ~0x01;
    if(port & 0x80){
        LATE=LATE | 0x02;
    } else LATE=LATE & ~0x02;
}
void program(){
    leds(0xFF);                 //Allumage de toutes les leds
    __delay_ms(190);            //Temporisation de 190ms
    leds(0x00);                 //Extinction de toutes les leds
    __delay_ms(190);            //Temporisation de 190ms
}
int main() {
    setup();                    //Paramètre les ports
    while(1){                   //Boucle while toujours vrai, permet de cacher le main
        program();              //Appel de la fonction programme
    }
    return (EXIT_SUCCESS);
}
```

III. Chenillard



Lors du deuxième exercice nous avons réalisé un chenillard suivant l'organigramme ci-contre.
Pour se faire nous avons donc repris les fonctions leds et setup du premier exercice et déclaré une fonction chenillard qui décale la led de la valeur 0x00 à la valeur FF. Une fois cette valeur atteinte on repars à la première led. Les étapes correspondantes sont décrites dans le programme ci-dessous.

```
16
17  void setup(){
18      TRISA=out;                      // Parametrer le port A en sortie
19      TRISE=out;                      // Parametrer le port E en sortie
20  }
21
22  void leds(unsigned char port){      // Fonction d'allumage des leds
23      LATA=~port;                     // Permet de lire la valeur du port dans un registre
24      if(~port & 0x40){               // Selectionne la valeur 65 du port
25          LATE=LATE | 0x01;
26      }
27      else LATE=LATE & ~0x01;
28      if(~port & 0x80){
29          LATE=LATE | 0x02;
30      } else LATE=LATE & ~0x02;
31  }
32
33  void chenillard(){
34  unsigned char Leds=0x01;
35      while(1){
36          leds(Leds);
37          Leds=Leds << 1;
38          if (Leds==0xFF){
39              Leds=0x01;
40          }
41      }
42  }
43  void program(){
44        chenillard();
45  }
46
47  int main() {
48      setup();                        //Initialisation de port via la fonction setup
49      while(1){
50          program();
51      }
52      return (EXIT_SUCCESS);
53  }
```

IV. Chenillard dans les deux sens : K2000

Pour réaliser un K2000, nous devons allumer successivement chaque led dans un sens puis dans l'autre. Pour cela nous reprenons donc la fonction led (unsigned char) et nous plaçons un delay de 190ms permettant une temporisation entre deux clignotements de leds. Il nous faut donc ensuite créer une variable « sens » dans la fonction program qui permettra de donner le sens de déplacement des diodes que l'on veut allumer.
On peut retrouver ci-dessous ces deux fonctions, le reste (*main.h, main(), setup(), #define et #include*) étant identiques aux programmes précédents.

```
21  void leds(unsigned char port){
22      port=~port;
23      LATA=port&0x3F;
24      if(port & 0x40){
25          LATE=LATE | 0x01;
26      }
27      else LATE=LATE & ~0x01;
28      if(port & 0x80){
29          LATE=LATE | 0x02;
30      }
31      else LATE=LATE & ~0x02;
32      __delay_ms(190);
33      return;
34  }
35  void program(){
36      unsigned char Leds=0x01;     //Initialisation de Leds a  1
37      unsigned char Sens=0;        //Initialisation de sens a  0
38      while(1){
39          leds(Leds);
40          if (Leds == 0x80){       //Si Leds=10000000 (valeur de la dernière led)
41              Sens=1;              //On attribu 1 a Sens
42          }
43          if (Leds == 0x01){       //Si Leds=000000001 (valeur de la première led)
44              Sens=0;              //On attribu 0 a Sens
45          }
46          if (Sens) Leds=Leds >> 1;    //Si on a un sens(=1) on déplace la led allumé
47          else Leds=Leds << 1;
48      }
49  }
```

V. Barre de chargement

Pour réaliser une barre de chargement nous nous sommes aidé des fonctions réalisées dans les exercices précédents. En effet nous réutilisons la fonction led (unsigned char) mais cette fois nous déplaçons le delays dans la fonction program. Ce programme doit permettre d'allumer les diodes tout en gardant les précédentes allumées. Pour cela nous avons eut l'idée d'incrémenter la valeur précédente par 2 puissance i pour le chemin allé et de décrémenter la valeur par 2 puissance j pour le chemin de retour.
On peut retrouver ces deux fonctions ci-dessous, le reste (*main.h, main(), setup(), #define et #include*) étant identiques aux programmes précédents.

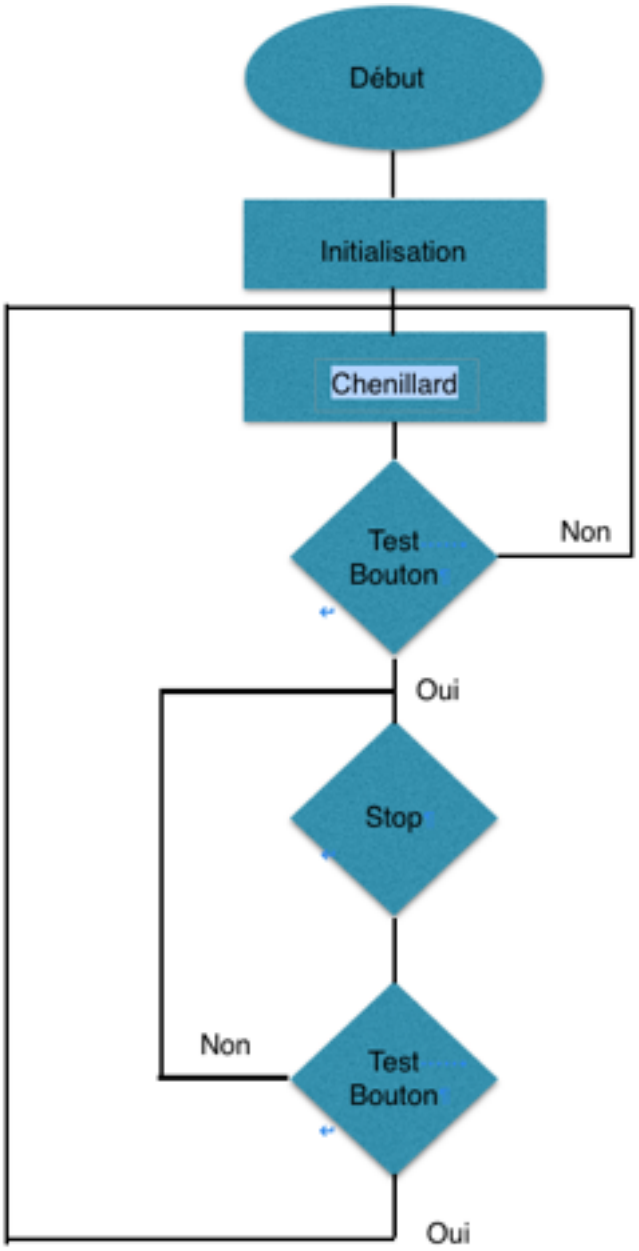
```
#include <stdio.h>                  // Initialisation des biliotheques
#include <stdlib.h>
#include <pic18f46k22.h>
#include "config.h"
#include "main.h"
#include "math.h"

#define out 0x00
#define in 0xFF

void setup(){
    TRISA=out;                      // Parametrer le port A en sortie
    TRISE=out;                      // Parametrer le port E en sortie
}
void leds(unsigned char port){      // Fonction d'allumage des leds
    LATA=~port;                     // Permet de lire la valeur du port dans un registre
    if(~port & 0x40){               // Selectionne la valeur 65 du port
        LATE=LATE | 0x01;
    }
    else LATE=LATE & ~0x01;
    if(~port & 0x80){
        LATE=LATE | 0x02;
    } else LATE=LATE & ~0x02;
}
void program(){
    unsigned char imLed=0x00;       //Initialisation d'imLed a  0
    int i=0, j=0;                   //Initialisation de i et j pour la boucle for suivante
        for(i=0;i<9;i++){           //Boucle for pour l'alumage des diodes de gauche a  droite
            leds(imLed);            //Allumage de la diode definie par la valeur imLed
            imLed=imLed+((unsigned char)pow((double)2,(double)i));  //Incrementation de la valeur de imLed par 2^i
            __delay_ms(100);        //Ajout d'un delais pour ralentir le clignotement
                if (imLed==0xFF){   //Si on arrive a la derniere diode de valeur 0xFF
                    for (j=9;j>-1;j--){ //On reparcours les leds dans l'autre sens (de droite a  gauche
                        imLed=imLed-((unsigned char)pow((double)2,(double)((j-1))));
                                    //Decrementation de la valeur de imLed par 2^j
                    leds(imLed);    //On allume la diode de valeur imLed
                        __delay_ms(100); //Ajout d'un delais pour ralentir le clignotement
                    }
                }
        }
}

int main(int argc, char** argv) {
    setup();                        //Appel de la fonction qui parametre les ports
    while(1){                       //Boucle while toujours verifier pour ne pas sortir du programme
        program();                  //Appel de la fonction programme
    }
    return (EXIT_SUCCESS);
}
```

VI. Sujet 2 : Réalisation d'un dé



Notre sujet consiste à réaliser un dé électronique en utilisant les leds comme faces du dé et le bouton poussoir pour arrêter le défilement des valeurs. A partir de cet énoncé nous en avons déduit l'organigramme ci-contre.

La difficulté est donc qu'une pression sur le bouton poussoir arrête le chenillard et qu'une deuxième pression sur ce même bouton le fait repartir à partir de cette même valeur.

Notre première idée était de faire un appel récursif de la fonction chenillard après l'arrêt. C'était la solution la plus logique et la plus simple à mettre en oeuvre. Malheureusement l'appel récursif n'est pas possible avec XC8.

Nous avons donc dû raisonner de manière itérative en séparant l'initialisation du chenillard pour qu'il commence à la première led, puis les cas suivants en mémorisant la valeur du port lors de l'arrêt. Cette valeur étant ensuite renvoyée, après le test du bouton dans la fonction led.
Nous nous sommes par la suite heurté à plusieurs autres problèmes.

Dans un premier temps nous avons du paramétrer le bouton poussoir en entrée numérique et non pas analogique. Après étude de la datasheet nous avons trouvé la commande :

- **ANSELBbits.ANSB0=0**

Ensuite nous avons configurer l'interruption de manière à ce qu'elle soit active sur front montant afin de ne pas avoir à faire de fonction de debouncing (annule les rebonds du bouton poussoir) :

- INTCON2bits.INTEDG0=1

La dernière difficulté a été de récupérer la valeur au moment de la pression du bouton poussoir pour redémarrer à la position correspondante. Nous avons donc dans un premier temps initialiser le chenillard et pris la première led comme position de départ. Puis lors de l'appel de la fonction stop, le programme enregistre la valeur retournée par la fonction chenillard dans une variable intermédiaire (stopled1). Pour finir, on reprend le chenillard à la même position en lui faisant prendre comme argument la valeur stopled1 précédemment enregistrée.

L'ensemble du programme commenté est disponible ci-dessous:

```c
#include "main.h"

void setup(){
    TRISA=out;                      //Configuration du port A en sortie
    TRISB=in;                       //Configuration du port B en entree
    ANSELBbits.ANSB0=0;             //Les broches sont configurees comme des entrees/sorties numerique
    INTCON2bits.INTEDG0=1;          //Interruption active sur front montant
    INTCONbits.INT0IF=0;            //Flag correspondant a RB0 initialise a 0
    INTCONbits.GIE=1;               //Active les interruptions globales
}
void leds(unsigned char port){
    port=~port;
    LATA=port;
    __delay_ms(25);
}
void stop(){                        //Fonction pour arrÃªter le defilement des leds en cas d'appui sur le bouton
    INTCONbits.INT0IF=0;            //Flag correspondant a RB0 initialise a 0
    while(push!=1){                 //Tant que le flag est different de 1 on reste dans la boucle while
        __delay_ms(1);
    }
}
unsigned char init(){               //Fonction d'initialisation du programme
      unsigned char stopled;        //Premier cycle: Start, Stop, Repries
      INTCONbits.INT0IF=0;          //Flag correspondant a RB0 initialise a 0
      stopled=chenillard(0x01);     //stopled prend la valeur de chenillard(0x01)
      stop();                       //Appel de la fonction stop en cas d'appui sur le bouton(changement de valeur de INT0IF
    return stopled;                 //Retourne la valeur stopled
}
unsigned char chenillard(unsigned char Leds){
    unsigned char tmp;              //Initialisation d'une variable tmp
    INTCONbits.INT0IF=0;            //Flag correspondant a RB0 initialise a 0
        while(push == 0 ){          //Tant que le bouton n'est pas presse, les leds chenillent
            leds(Leds);
            Leds=Leds << 1;
            if (Leds==0x40){
                Leds=0x01;
            }
            tmp=Leds;
        }
    return tmp;
}
void program(unsigned char firststop){
      unsigned char stopled1;       //Initialisation d'une variable stopled1
      INTCONbits.INT0IF=0;          //Flag correspondant a RB0 initialise a 0
      stopled1 = chenillard(firststop); //stopled1 prend la valeur du chenillard au premier arret
      stop();                       //appel de la fonction stop
      chenillard(stopled1);         //Reprise du chenillard Ã  la diode ou l'on s'Ã©tait arrete (stopled1)
      stop();                       //Appel de la fonction stop
}
int main() {
    unsigned char firststop;        //Initialisation d'une variable firstop
    setup();                        //Initialisation de port via la fonction setup
    firststop = init();             //Initialisation de la premiere valeur dans firstop
    while(1){
        program(firststop);
    }
    return (EXIT_SUCCESS);
}
```

VII. Conclusion

Ce TP nous a permis de découvrir la programmation des PIC grâce au logiciel MPLAB. En effet, nous avons appris à générer le fichier de configuration d'un PIC grâce à MPLAB, nous avons également appris à réaliser des programmes au début peu complexes tel que le chenillard, puis plus compliqués avec la réalisation d'un dé électronique.
Nous avons notamment vu qu'il était important de savoir lire une datasheet afin d'utiliser les informations contenues dans celle-ci, dès que l'on veut faire interagir le pic avec d'autres composants. En effet, pour coder un programme fonctionnel, il faut par exemple être capable d'initialiser les ports en entrée ou en sortie, les interruptions, les flags, ou encore paramétrer correctement les boutons poussoirs etc.
Enfin, ce TP nous a finalement permis d'appliquer nos connaissances en informatique, et notamment en C, sur un projet physique, plus concret que lorsqu'on observe les résultats sur une console.

VIII.Annexes

a. Annexe 1 - Configuration Bit Setting (config.h)

```
// PIC18F46K22 Configuration Bit Settings
#include <xc.h>

// CONFIG1H
#pragma config FOSC = XT        // Oscillator Selection bits (XT oscillator)
#pragma config PLLCFG = OFF     // 4X PLL Enable (Oscillator used directly)
#pragma config PRICLKEN = ON    // Primary clock enable bit (Primary clock is always enabled)
#pragma config FCMEN = OFF      // Fail-Safe Clock Monitor Enable bit (Fail-Safe Clock Monitor disabled)
#pragma config IESO = OFF       // Internal/External Oscillator Switchover bit (Oscillator Switchover mode disabled)

// CONFIG2L
#pragma config PWRTEN = OFF     // Power-up Timer Enable bit (Power up timer disabled)
#pragma config BOREN = SBORDIS  // Brown-out Reset Enable bits (Brown-out Reset enabled in hardware only (SBOREN is disabled))
#pragma config BORV = 190       // Brown Out Reset Voltage bits (VBOR set to 1.90 V nominal)

// CONFIG2H
#pragma config WDTEN = OFF      // Watchdog Timer Enable bits (Watch dog timer is always disabled. SWDTEN has no effect.)
#pragma config WDTPS = 32768    // Watchdog Timer Postscale Select bits (1:32768)

// CONFIG3H
#pragma config CCP2MX = PORTC1  // CCP2 MUX bit (CCP2 input/output is multiplexed with RC1)
#pragma config PBADEN = ON      // PORTB A/D Enable bit (PORTB<5:0> pins are configured as analog input channels on Reset)
#pragma config CCP3MX = PORTB5  // P3A/CCP3 Mux bit (P3A/CCP3 input/output is multiplexed with RB5)
#pragma config HFOFST = ON      // HFINTOSC Fast Start-up (HFINTOSC output and ready status are not delayed by the oscillator
#pragma config T3CMX = PORTC0   // Timer3 Clock input mux bit (T3CKI is on RC0)
#pragma config P2BMX = PORTD2   // ECCP2 B output mux bit (P2B is on RD2)
#pragma config MCLRE = EXTMCLR  // MCLR Pin Enable bit (MCLR pin enabled, RE3 input pin disabled)

// CONFIG4L
#pragma config STVREN = ON      // Stack Full/Underflow Reset Enable bit (Stack full/underflow will cause Reset)
#pragma config LVP = ON         // Single-Supply ICSP Enable bit (Single-Supply ICSP enabled if MCLRE is also 1)
#pragma config XINST = OFF      // Extended Instruction Set Enable bit (Instruction set extension and Indexed Addressing mode

// CONFIG5L
#pragma config CP0 = OFF        // Code Protection Block 0 (Block 0 (000800-003FFFh) not code-protected)
#pragma config CP1 = OFF        // Code Protection Block 1 (Block 1 (004000-007FFFh) not code-protected)
#pragma config CP2 = OFF        // Code Protection Block 2 (Block 2 (008000-00BFFFh) not code-protected)
#pragma config CP3 = OFF        // Code Protection Block 3 (Block 3 (00C000-00FFFFh) not code-protected)

// CONFIG5H
#pragma config CPB = OFF        // Boot Block Code Protection bit (Boot block (000000-0007FFh) not code-protected)
#pragma config CPD = OFF        // Data EEPROM Code Protection bit (Data EEPROM not code-protected)

// CONFIG6L
#pragma config WRT0 = OFF       // Write Protection Block 0 (Block 0 (000800-003FFFh) not write-protected)
#pragma config WRT1 = OFF       // Write Protection Block 1 (Block 1 (004000-007FFFh) not write-protected)
#pragma config WRT2 = OFF       // Write Protection Block 2 (Block 2 (008000-00BFFFh) not write-protected)
#pragma config WRT3 = OFF       // Write Protection Block 3 (Block 3 (00C000-00FFFFh) not write-protected)

// CONFIG6H
#pragma config WRTC = OFF       // Configuration Register Write Protection bit (Configuration registers (300000-3000FFh) not w
#pragma config WRTB = OFF       // Boot Block Write Protection bit (Boot Block (000000-0007FFh) not write-protected)
#pragma config WRTD = OFF       // Data EEPROM Write Protection bit (Data EEPROM not write-protected)

// CONFIG7L
#pragma config EBTR0 = OFF      // Table Read Protection Block 0 (Block 0 (000800-003FFFh) not protected from table reads exec
#pragma config EBTR1 = OFF      // Table Read Protection Block 1 (Block 1 (004000-007FFFh) not protected from table reads exec
#pragma config EBTR2 = OFF      // Table Read Protection Block 2 (Block 2 (008000-00BFFFh) not protected from table reads exec
#pragma config EBTR3 = OFF      // Table Read Protection Block 3 (Block 3 (00C000-00FFFFh) not protected from table reads exec

// CONFIG7H
#pragma config EBTRB = OFF      // Boot Block Table Read Protection bit (Boot Block (000000-0007FFh) not protected from table
```

b. Annexe 2 - main.h - 8 leds clignotantes.

```
/*
 * File:   main.h
 */

#ifndef MAIN_H
#define MAIN_H

#ifndef _XTAL_FREQ
#define _XTAL_FREQ 4000000
#endif

void setup();
void delay();
void leds(unsigned char);
void program();
int main();

#endif  /* MAIN_H */
```

c. Annexe 3 - main.h - réalisation d'un dé.

```
7
8  #ifndef MAIN_H
9  #define MAIN_H
10
11 #ifndef _XTAL_FREQ
12 #define _XTAL_FREQ 4000000
13 #endif
14
15 #include <stdio.h>
16 #include <stdlib.h>
17 #include <pic18f46k22.h>
18 #include "config.h"
19
20 #define out 0x00
21 #define in 0xFF
22 #define AVECIT                          // mode Interrupt sur RB0 ou Pooling bloquant
23 #define push INTCONbits.INT0IF
24
25 unsigned char chenillard(unsigned char);
26 unsigned char init(void);
27 int main(void);
28 void program(unsigned char);
29 void stop(void);
30 void setup(void);
31 void leds(unsigned char);
32
33
34 #endif  /* MAIN_H */
```