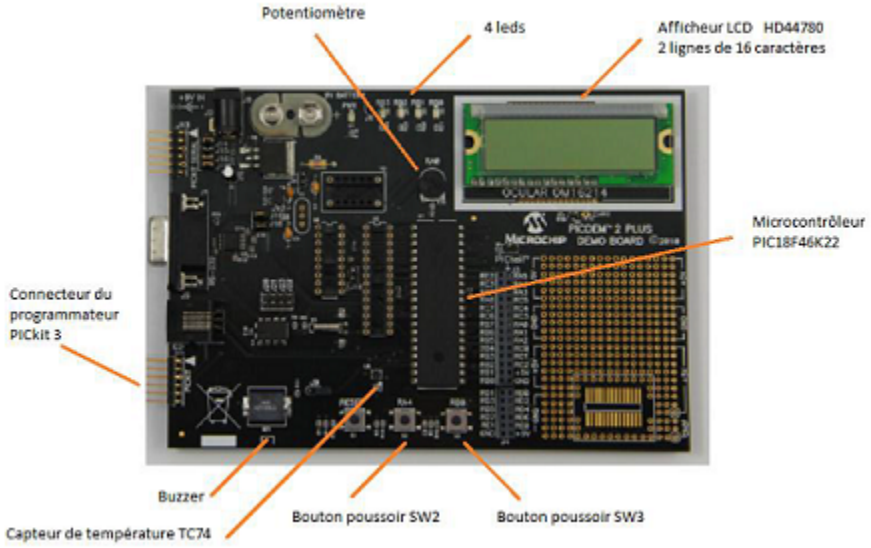


FIGURE 1 – Carte de développement PICDEM2+

1 Introduction

L'objectif de ce mini-projet est d'étudier le fonctionnement de la carte **PICDEM2+** équipée d'un microcontrôleur **Microchip PIC18F46K22** ainsi que différents périphériques qui y sont attachés. Le programmateur sera celui du **Pickit 3**.

De nombreux exemples utilisant chacun un périphérique particulier (LEDs, bouton poussoir, potentiomètre, buzzer, afficheur LCD et capteur de température) seront étudiés. Un dernier exemple récapitulera les précédents et introduira quelques notions supplémentaires.

Tous les programmes seront écrits en **langage C** (compilateur **Microchip XC8**) et l'environnement de développement sera l'IDE **MPLABX**, basé sur Netbeans.

2 Carte de prototypage

Une image de la carte **PICDEM2+** se trouve en Figure 1 et montre les différents composants et connecteurs qui seront utilisés durant ces travaux pratiques.

En annexe, vous trouverez le guide d'utilisation de la carte **PICDEM2+** et les datasheets du **PIC18F46K22**, de l'écran LCD **HD44780U** et du capteur de température **TC74**.

3 Préliminaires

3.1 Préparation de la carte PICDEM2+

La carte **PICDEM2+** est équipée en standard d'un microcontrôleur **PIC16F1937**. Si tel est le cas, il faut le remplacer par le **PIC18F46K22** qui se trouve dans la boîte noire.

Une vérification des cavaliers de la carte est également nécessaire comme indiqué dans le tableau ci-dessous. L'utilisation de ces cavaliers est décrite en page 13 du manuel d'utilisation de la carte.

Cavalier	État
J6	ON
J7	ON
J9	ON
J10	OFF
J12	ON
J<14 :17>	ON
J<18 :19>	ON
J<20 :23>	OFF
J24	OFF

3.2 Installation de l'IDE MPLABX

Si **MPLABX** n'est pas installé, le télécharger depuis ce lien : `http://www.microchip.com/pagehandler/en-us/family/mplabx/` (rubrique MPLABX IDE/downloads archive). Il s'agit de la version 2.26.

Désarchiver *MPLABX-v2.26-windows-installer.zip*.

Lancer l'exécution de l'assistant d'installation *MPLABX-v2.26-windows-installer.zip* et suivez les étapes par défaut.

3.3 Installation du compilateur C XC8

Si le compilateur **XC8** n'est pas installé, le télécharger depuis l'adresse `http://www.microchip.com/pagehandler/en_us/devtools/mplabxc/` (rubrique MPLAB XC Compilers/downloads archive). Il s'agit de la version 2.05.

Lancer l'exécution de l'assistant d'installation *xc8-v2.05-full-install-windows-installer.exe* et suivez les étapes par défaut.

Si **MPLABX** est en cours d'exécution lors de l'installation de **XC8**, ce dernier sera automatiquement reconnu par **MPLABX**.

4 Fichiers importants

Dans le CD d'accompagnement, on trouvera dans le répertoire **"PICDEM2 Plus"**, plusieurs fichiers importants car réutilisables dans chaque projet :

1. "Header files" qui contient "general.h", "leds.h", "lcd.h", "uart.h", "i2c.h" et "spi.h"
2. "Source files" qui contient "configbits.c", "lcd.c", "uart.c", "i2c.c" et "spi.c"

4.1 general.h

Ce fichier contient les définitions couramment utilisées ainsi que quelques types :

```
#ifndef _GENERAL_H
#define _GENERAL_H

// Booleans
#define TRUE            1           // allow TRUE to equal 1
#define FALSE           0           // allow FALSE to equal 0
#define forever         while(TRUE) // endless loop

// Following definitions are equivalent (depending on the context), but SET or CLEAR are sufficient
#define SET             1           // allow SET to equal 1
#define CLEAR           0           // allow CLEAR to equal 0

#define ENABLE          SET         // allow ENABLE to equal 1
#define DISABLE         CLEAR       // allow DISABLE to equal 0

#define READ            SET         // allow READ to equal 1
#define WRITE           CLEAR       // allow WRITE to equal 0

#define ON              SET         // allow ON to equal 1
#define OFF             CLEAR       // allow OFF to equal 0

#define INP             SET         // PORT bit is in input mode
#define OUTP            CLEAR       // PORT bit is in output mode

#define OUTP_HIGH       SET         // high level
#define OUTP_LOW        CLEAR       // low level

// Internal oscillator frequency
#define _XTAL_FREQ      1000000UL   // default XTAL frequency is 1MHz (FOSC=250kHz)
                                    // (used by inline function _delay)
                                    // TCY = (1 / (_XTAL_FREQ / 4)) = 4µs

// New typedefs
typedef unsigned char   UINT8_T;    // UINT8_T means byte (8-bit)
typedef unsigned int    UINT16_T;   // UINT16_T means word(16-bit)
typedef unsigned char   BOOL;       // BOOL means byte (8-bit)
typedef char            INT8_T;     // INT8_T means 2's complement (1 sign bit + 7-bit)
typedef int             INT16_T;    // INT16_T means 2's complement (1 sign bit + 15-bit)

#endif /* _GENERAL_H */
```

4.2 leds.h

Ce fichier contient les définitions relatives à l'utilisation des leds :

```
#ifndef _LEDS_H
#define _LEDS_H

#define LED0_DIR        TRISBbits.TRISB0        // direction of RB0 (OUTP)
#define LED1_DIR        TRISBbits.TRISB1        // direction of RB1 (OUTP)
#define LED2_DIR        TRISBbits.TRISB2        // direction of RB2 (OUTP)
#define LED3_DIR        TRISBbits.TRISB3        // direction of RB3 (OUTP)

#define LED0_STATE      LATBbits.LATB0          // state of led0 (ON/OFF)
#define LED1_STATE      LATBbits.LATB1          // state of led1 (ON/OFF)
#define LED2_STATE      LATBbits.LATB2          // state of led2 (ON/OFF)
#define LED3_STATE      LATBbits.LATB3          // state of led3 (ON/OFF)

#endif /* _LEDS_H */
```

4.3 i2c.h

Ce fichier contient les définitions relatives à l'utilisation du bus I2C ainsi que les prototypes des fonctions contenues dans "i2c.c" :

```
#ifndef _I2C_H
#define _I2C_H

#include "general.h"


#define I2C_SCL_DIR         TRISCbits.TRISC3        // Direction bit for I2C clock (SCL)
#define I2C_SDA_DIR         TRISCbits.TRISC4        // Direction bit for I2C serial data (SDA)

#define I2C_SCL             LATCbits.RC3            // I2C SCL pin
#define I2C_SDA             LATCbits.RC4            // I2C SDA pin

#define TC74_ADDRESS        0b01001101              // TC74 address

#define I2C_READ            READ                    // I2C read mode
#define I2C_WRITE           WRITE                   // I2C write mode


void i2c_init(void);
void i2c_waitForIdle(void);
void i2c_start(void);
void i2c_repStart(void);
void i2c_stop(void);
UINT8_T i2c_read(void);
void i2c_write(UINT8_T data);
void i2c_ACK(void);
void i2c_NAK(void);

#endif /* _I2C_H */
```

4.4 lcd.h

Ce fichier contient les définitions relatives à l'utilisation de l'afficheur LCD ainsi que les prototypes des fonctions utilitaires contenues dans le fichier "lcd.c" :

```
#ifndef _LCD_H
#define _LCD_H

#include "general.h"

#define NB_LINES    2                       // number of display lines
#define NB_COL      16                      // number of characters per line

#define LCD_PWR     LATDbits.LATD7          // LCD ON/OFF line
#define LCD_EN      LATDbits.LATD6          // LCD enable line
#define LCD_RW      LATDbits.LATD5          // LCD read/write line
#define LCD_RS      LATDbits.LATD4          // LCD register select line
#define LCD_DAT0    LATDbits.LATD0          // data bit 0
#define LCD_DAT1    LATDbits.LATD1          // data bit 1
#define LCD_DAT2    LATDbits.LATD2          // data bit 2
#define LCD_DAT3    LATDbits.LATD3          // data bit 3

#define LCD_PWR_DIR TRISDbits.TRISD7        // direction bit for the power line
#define LCD_EN_DIR  TRISDbits.TRISD6        // direction for EN control line
#define LCD_RW_DIR  TRISDbits.TRISD5        // direction for RW control line
#define LCD_RS_DIR  TRISDbits.TRISD4        // direction for RS control line
#define LCD_D0_DIR  TRISDbits.TRISD0        // direction for data line 0
#define LCD_D1_DIR  TRISDbits.TRISD1        // direction for data line 1
#define LCD_D2_DIR  TRISDbits.TRISD2        // direction for data line 2
#define LCD_D3_DIR  TRISDbits.TRISD3        // direction for data line 3

#define LCD_ON()    LCD_PWR = ENABLE        // turns the display on
#define LCD_OFF()   LCD_PWR = DISABLE       // turns the display off


void LCDInit(void);                                             // LCD initialization service
void LCDClear(void);                                            // screen clearing service
void LCDGoto(UINT8_T Pos,UINT8_T Ln);                           // cursor positioning service
void LCDPutChar(UINT8_T Data);                                  // outputs ASCII character
void LCDPutByte(UINT8_T Val);                                   // outputs binary value
void LCDWriteStr(const char  *Str);                             // outputs strings to LCD
void LCDWriteHexa(UINT8_T column, UINT8_T row, UINT8_T value);  // output hexadecimal value at specified position
void LCDWriteInt(UINT8_T column, UINT8_T row, UINT8_T value);   // output positive decimal value (from 00 to 99) on 2 digit
void LCDWriteFloat(UINT8_T column, UINT8_T row, float value);   // output positive float value as x.yy, at specified positi
void LCDShiftDisplay(INT8_T shiftCount);                        // shift display 'shiftCount' on the left

#endif /* _LCD_H */
```

4.5 configbits.c

Ce fichier est très important. Comme en assembleur (directive CONFIG), il permet d'informer le programmateur du **Pickit 3** des valeurs des bits de configuration (voir Annexe).

Ce fichier est à intégrer dans le fichier "main.c" de chaque projet, après les directives *#include*.

N'étant pas un fichier d'entête, on ne peut que faire du copier/coller (enfin si, on peut faire *#include "configbits.c"* mais ce n'est pas la convention).

5 Exercices

5.1 Chenillard

Il s'agit de concevoir un chenillard (chaque LED s'allume séquentiellement), avec une attente d'une seconde entre chaque changement de LED.

Pour cela, votre programme devra être capable de contrôler des LEDs et utiliser une temporisation logicielle.

1. Ouvrir le logiciel MPLABX.
2. Créer un projet et le nommer TP_Chenillard.
3. Écrire un programme en langage C permettant d'allumer séquentiellement chaque LED sans temporisation.
4. Simuler le programme sur la carte PCIDEM2+ en utilisant le mode "Debugger". Commenter.
5. Écrire une fonction delai_1sec permettant d'effectuer une temporisation software sachant que le temps d'exécution d'une instruction dure 4 µs.
6. Insérer cette fonction dans votre programme, puis tester-le directement sur la carte PICDEM2+.
7. Commenter les résultats obtenus.

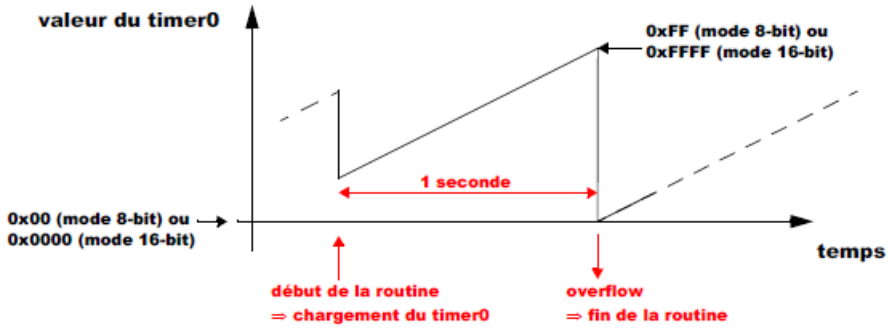
Remarques :
— **Pour cet exercice, le cavalier J6 doit être sur ON.**
— **Il faut impérativement commencer votre programme par l'instruction *Nop()*, qui est une fonction équivalente à l'instruction *NOP* en assembleur, car le debugger a une fâcheuse tendance à commencer l'exécution du programme à la deuxième instruction, et ce de manière aléatoire.**

5.2 Utilisation d'un timer

Pour cet exercice, l'objectif principal est le même que précédemment, et le programme principal gardera la même structure, mais la routine de temporisation se basera sur l'utilisation d'un timer pour fixer la routine de temporisation à **exactement 1 seconde** (à quelques µs près).

C'est le timer 0 qui sera utilisé (cf. page 159 de la datasheet du PIC18F46K22). Il sera activé et configuré de manière à ce que sa période totale soit au moins égale à 1 seconde (registre T0CON).

Au début de la routine de temporisation, il faudra charger le timer à une valeur adéquate, puis attendre qu'il ait atteint sa valeur maximale (ou overflow). Cet évènement peut être détecté en testant le flag d'overflow TMR0IF dans le registre INTCON.



1. Ouvrir le logiciel MPLABX.
2. Créer un projet et le nommer TP_Chenillard_timer.
3. Écrire un fichier header et un fichier source afin d'initialiser le timer 0.
4. Écrire un programme en langage C permettant d'allumer séquentiellement chaque LED en utilisant le timer 0 comme temporisation hardware.
5. Tester le programme directement sur la carte PICDEM2+.
6. Commenter les résultats obtenus.

5.3 Utilisation des interruptions

La carte de prototypage comporte un bouton poussoir qui permet de forcer une entrée du microcontrôleur (RB0/INT0) à l'état bas lors d'un appui (cf. schéma électrique pages 11-14 de la documentation de la carte).

Ce bouton sera utilisé pour mettre en marche ou arrêter le timer 0 à chaque appui, en modifiant le bit TMR0ON dans le registre T0CON.Comme cet évènement peut arriver à n'importe quel moment, il sera traité comme une interruption.

La routine d'interruption devra à chaque appel modifier la valeur d'une variable "marche_arret", et selon la valeur de celle-ci arrêter ou mettre en marche le timer 0.

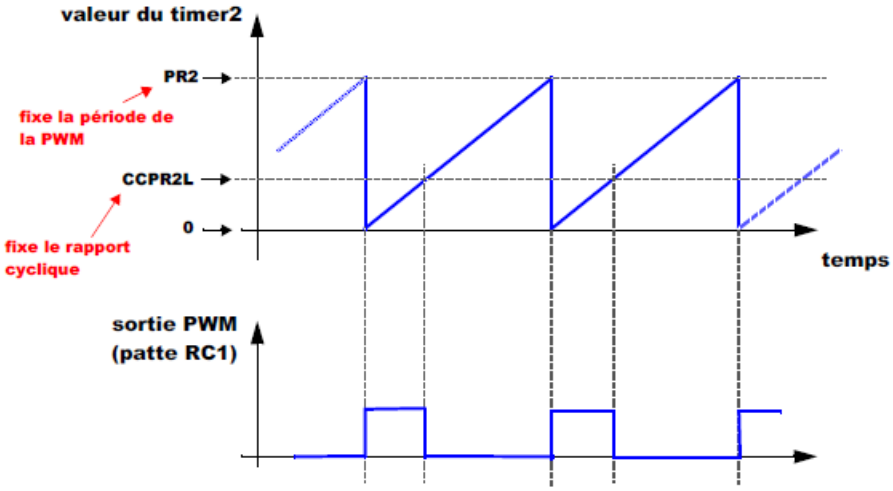
Le programme principal devra configurer le microcontrôleur de manière à ce qu'une interruption soit déclenchée à chaque front descendant détecté sur la patte RB0 (interruption INT0, configurée par les bits INT0IE dans INTCON, et INTEDG0 dans INTCON2).

1. Ouvrir le logiciel MPLABX.
2. Créer un projet et le nommer TP_Chenillard_interruption.
3. Écrire un fichier header et un fichier source afin d'initialiser le timer 0.
4. Écrire un fichier header et un fichier source afin d'initialiser le bouton.
5. Écrire un programme en langage C permettant d'interrompre ou de redémarrer le chenillard par appui sur le bouton poussoir.
6. Tester le programme directement sur la carte PICDEM2+.
7. Commenter les résultats obtenus.

5.4 Génération d'un signal PWM

Le but de cet exercice est de générer à l'aide d'un buzzer un signal de fréquence 440 Hz (le LA de la gamme). Puis, après temporisation, on passe au LA de l'octave précédent (police, gendarmerie, pompiers ou SAMU).

Le programme doit générer un signal carré de fréquence déterminée par l'intermédiaire du module PWM contrôlé par le timer 2. Le fonctionnement simplifié est le suivant :



La phase d'initialisation permet de :
- Mettre à jour le rapport cyclique de la PWM (CPPR1L = 128, bits<2 :10> du rapport cyclique)
- Configurer le timer 2 (T2CON) sans prescaler ni postcaler (FOSC = 250 kHz)
- Configurer le mode PWM et les deux bits<1 :0> du rapport cyclique
- Indiquer que le signal issu de la PWM sort sur la broche 2 du port C (broche connectée au buzzer)

1. Ouvrir le logiciel MPLABX.
2. Créer un projet et le nommer TP_Buzzer_PWM.
3. Écrire un programme en langage C permettant d'utiliser le buzzer.
4. Tester le programme directement sur la carte PICDEM2+.
5. Commenter les résultats obtenus.

Remarque : Pour cet exemple, le cavalier J9 doit être sur ON.

5.5 Voltmètre

Le but de cet exercice est d'utiliser l'écran LCD pour afficher la tension aux bornes du potentiomètre monté sur la carte de prototypage.

Ce dernier permet d'appliquer sur l'entrée RA0/AN0 du microcontrôleur une tension comprise entre 0 et V_{DD}. Il faudra donc utiliser ici le convertisseur analogique/numérique pour mesurer la tension d'entrée, puis afficher sa valeur sur l'écran LCD.

— L'entrée RA0 sera configurée en entrée (registre TRISA).
— Le bit n°0 du registre ANSEL devra être mis à 1 pour désactiver le buffer d'entrée numérique de la patte RA0, car elle sera utilisée comme entrée analogique.
— Le registre ADCON1 sera configuré de manière à ce que les tensions de référence de l'ADC soient $V_{SS}(0V)$ et $V_{DD}(\approx 3V)$.
— Le registre ADCON2 sera configuré de manière à ce que le résultat de la conversion soit "left-justified", que la fréquence de fonctionnement du convertisseur f_{AD} soit égale à $f_{OSC}/64$, et que le temps d'acquisition soit égal à $20T_{AD}$.
— Le registre ADCON0 sera utilisé pour sélectionner l'entrée analogique RA0/AN0, mettre en route le convertisseur, lancer une nouvelle conversion, et tester la fin de la conversion en scrutant (méthode dite de "polling") le bit $GO/\overline{DONE}$.

1. Ouvrir le logiciel MPLABX.
2. Créer un projet et le nommer TP_voltmetre.
3. Écrire un programme en langage C permettant d'afficher la tension aux bornes du potentiomètre sur l'écran LCD.
4. Tester le programme directement sur la carte PICDEM2+.
5. Commenter les résultats obtenus.

Remarques :
— **Dans les directives *#pragma*, il faut configurer le bit PBADEN à ON pour pouvoir utiliser l'entrée analogique du potentiomètre.**
— **Il faut autoriser la broche 0 du port A (à laquelle est relié le potentiomètre) à devenir une entrée analogique.**

5.6 Affichage de la température

Le but de cet exercice est de mesurer la température de la salle de classe à l'aide du capteur TC74 monté sur la carte de prototypage et d'afficher la valeur sur un écran LCD.

Pour cela, on utilisera le protocole de communication du bus I2C dont le fonctionnement est le suivant :

1. Envoyer à l'esclave une *start condition* pour démarrer une nouvelle trame
2. Envoyer à l'esclave son adresse afin qu'il le reconnaisse
3. Envoyer à l'esclave une commande
4. Envoyer à l'esclave une *repeated start condition*
5. Envoyer à l'esclave son adresse afin qu'il se reconnaisse
6. Écouter l'esclave qui envoie le résultat de la commande effectuée en 3
7. Envoyer à l'esclave un *acknowledge* si la trame n'est pas terminée, sinon un *non-acknowledge* dans le cas contraire
8. Répéter les étapes 4 à 7 si nécessaire
9. Envoyer à l'esclave une *stop condition*

Manipulations :

1. Ouvrir le logiciel MPLABX.
2. Créer un projet et le nommer TP_temperature.
3. Écrire un programme en langage C permettant d'afficher la température sur l'écran LCD.
4. Tester le programme directement sur la carte PICDEM2+.
5. Commenter les résultats obtenus.

5.7 Utilisation de la liaison UART

L'exercice est assez simple : il consiste à transmettre un message, à le recevoir et à l'afficher sur l'afficheur LCD. Pour cela, nous utiliserons l'EUSART en mode UART avec transmission/réception asynchrone (1 bit de START, 8 bits de données, 1 bit de STOP).

La transmission se fait de manière classique (avec une temporisation entre l'envoi de chaque caractère), la réception utilise les interruptions.

L'horloge du PIC étant de 1 MHz, la transmission et la réception se font à 9600 Bauds, afin de minimiser l'erreur (ici 0.16 %). Les bits BRGH et BRG16 des registres TXSTA1 et BAUDCON1 étant à 1, la transmission se fait en haute vitesse, et la formule pour calculer la fréquence d'émission est (voir le document PIC18F4xK22 Silicon Errata) :

$$SPBRG = (\_XTAL\_FREQ/4 * 9600) - 1$$

1. Ouvrir le logiciel MPLABX.
2. Créer un projet et le nommer TP_UART.
3. Écrire un programme en langage C permettant d'afficher le message transmis par l'UART sur l'écran LCD, à savoir le débit de la liaison série ainsi qu'un compteur comptant indéfiniement.
4. Tester le programme directement sur la carte PICDEM2+.
5. Commenter les résultats obtenus.

Remarque : Pour cet exemple, il faut connecter la broche RC6 (TxD) à la broche RC7 (RxD).

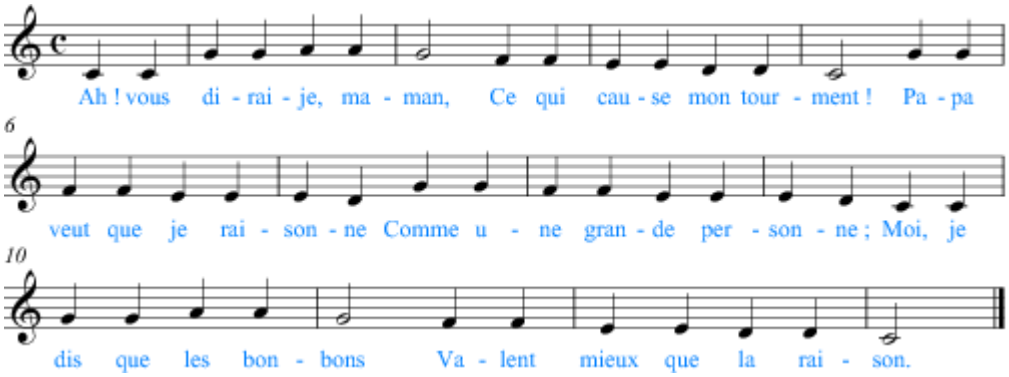
5.8 Karaoké

Ce projet simule un "karaoké". Le buzzer joue la mélodie, les LEDs suivent le rythme, l'afficheur LCD affiche les paroles.

Pour cela, nous utiliserons le timer 0 comme timer, et non plus comme compteur.
Pour reprogrammer la durée de la temporisation du timer 0, son compteur n'ira plus de 0 à 65535, mais de N à 65535. Plus N est grand, plus courte sera la temporisation. Les bits de poids forts du compteur devront obligatoirement être réinitialisés avant les bits de poids faibles, car c'est l'écriture dans TMR0L qui déclenche la validation de TMR0H.

L'écran LCD affiche 2 lignes de 16 caractères. En fait, chaque ligne peut en compter jusqu'à 40. Si l'on veut afficher de telles lignes, il faut utiliser la fonction *LCDShiftDisplay()* en indiquant le nombre voulu de décalages vers la gauche.

1. Ouvrir le logiciel MPLABX.
2. Créer un projet et le nommer TP_karaoke.
3. Écrire un programme en langage C permettant de jouer une mélodie tout en affichant les paroles sur l'afficheur LCD. La chanson que vous jouerez sera *Ah ! Vous dirai-je maman*, une comptine mise en musique par W.A. Mozart dont la partition est la suivante :



4. Tester le programme directement sur la carte PICDEM2+.
5. Commenter les résultats obtenus.

Remarque : Voici un tableau présentant les fréquences de chaque note de musique :

Fréquences des hauteurs (en Hertz)

Note\octave	0	1	2	3	4	5	6	7
Do	32,70	65,41	130,81	261,63	523,25	1046,50	2093,00	4186,01
Do♯	34,65	69,30	138,59	277,18	554,37	1108,73	2217,46	4434,92
Ré	36,71	73,42	146,83	293,66	587,33	1174,66	2349,32	4698,64
Ré♯	38,89	77,78	155,56	311,13	622,25	1244,51	2489,02	4978,03
Mi	41,20	82,41	164,81	329,63	659,26	1318,51	2637,02	5274,04
Fa	43,65	87,31	174,61	349,23	698,46	1396,91	2793,83	5587,65
Fa♯	46,25	92,50	185,00	369,99	739,99	1479,98	2959,96	5919,91
Sol	49,00	98,00	196,00	392,00	783,99	1567,98	3135,96	6271,93
Sol♯	51,91	103,83	207,65	415,30	830,61	1661,22	3322,44	6644,88
La	55,00	110,00	220,00	440,00	880,00	1760,00	3520,00	7040,00
La♯	58,27	116,54	233,08	466,16	932,33	1864,66	3729,31	7458,62
Si	61,74	123,47	246,94	493,88	987,77	1975,53	3951,07	7902,13