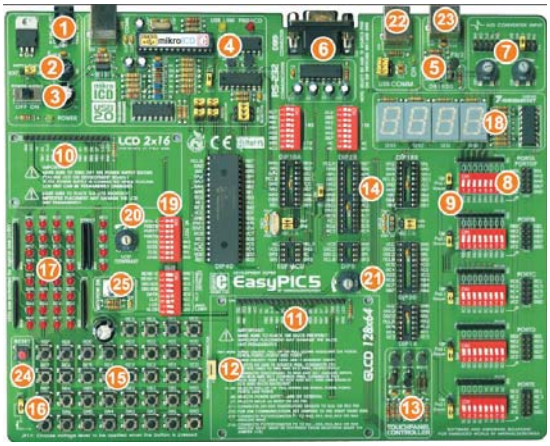
Ministère de l'enseignement supérieur et de la recherche scientifique
Institut Supérieur des Etudes Technologiques de Sousse

Département Génie Electrique
Spécialité : Automatisme Informatique Industrielle – 2ème Année

Fascicule des travaux pratiques

Microcontrôleur

PIC16F877



EasyPIC5

Proposé par : Ali Hmidene

2009/2010

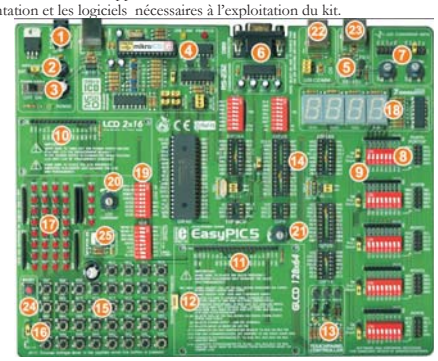
Institut Supérieur des Etudes Technologiques de Sousse
Département Génie Electrique – Licence AII2-S2
Malière : microprocesseur- microcontrôleur

2009 / 2010

TP0 : PRISE EN MAIN DU KIT EasyPIC5

1. Présentation du Kit EasyPIC5

Le kit **EasyPIC5** de la société **MikroElectronika** offre une large gamme d'applications des microcontrôleurs PIC. Cet outil permet à l'étudiant de tester et d'explorer les possibilités offertes par les microcontrôleurs PIC. Outre les applications internes, tous les ports sont accessibles via des connecteurs HE10 permettant ainsi de connecter au kit des périphériques externe tels qu'un ADC, DAC, CAN, RTC, RS485 etc.. De même le kit intègre un programmateur USB haute tension et un ICD Debugger permettant de programmer et tester les applications utilisateurs. Le CD livré avec le kit contient la documentation et les logiciels nécessaires à l'exploitation du kit.



2. Caractéristiques principales du EasyPIC5

2.1. Matérielle

- ⇨ Une alimentation externe AC/DC 8 – 16V (facultative), puisque le kit peut être alimenté par le bus USB.
- ⇨ Un programmateur USB et un ICD debugger.
- ⇨ Un module de communication RS232.
- ⇨ Un port USB pour les microcontrôleurs supportant un module USB.

- ⇨ Un capteur de Température DS1820.
- ⇨ Deux potentiomètres P1 et P2 connectés à travers un sélecteur aux entrées de l'ADC.
- ⇨ Quatre afficheurs 7 segments fonctionnant en mode multiplexé.
- ⇨ Un connecteur 16 broches pour la connexion d'un afficheur LCD.
- ⇨ 36 diodes LEDs connectées aux ports I/O du microcontrôleur.
- ⇨ 36 boutons connectés aux ports I/O du microcontrôleur.
- ⇨ Un connecteur PS/2.
- ⇨ Un connecteur 20 broches pour la connexion d'un afficheur LCD Graphique + un connecteur d'un écran tactile.
- ⇨ Un ensemble des supports des circuits intégrés DIP8, DIP14, DIP18, DIP20, DIP28 et DIP40 permettant au kit de supporter la plupart des microcontrôleurs de microchip.
- ⇨ Un ensemble des DIP Switches et des jumpers pour la sélection des périphériques.

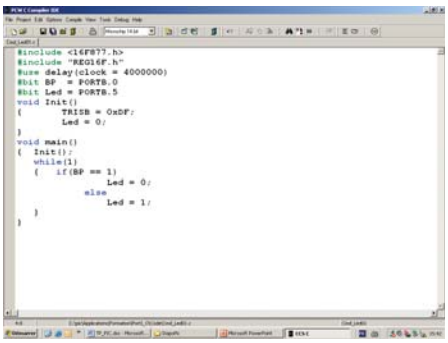
2.2. Logicielle

Un Environnement de Développement Intégré comprenant :

- ⇨ Un compilateur mikroC, mikroBasic ou mikroPascal (Nous avons opté pour le compilateur PIC Compiler).
- ⇨ Un PICflash programmer et mikroICD debugger.
- ⇨ Un Driver USB pour le logiciel PICflash programmer.

3. PICC Compiler

PICC Compiler est un outil de développement complet pour les microcontrôleurs PIC. Ces librairies matérielles permettent au programmeur de développer des applications embarquées de manière simple et rapide.



```
#include <16F877.h>
#include "REG16F.h"
#use delay(clock = 4000000)
#bit BP  = PORTB.0
#bit Led = PORTB.5
void Init()
{
        TRISB = 0xDF;
        Led = 0;
}
void main()
{  Init();
   while(1)
   {   if(BP == 1)
                Led = 0;
           else
                Led = 1;
   }
}
```

L'environnement ***PICC Compiler*** permet :

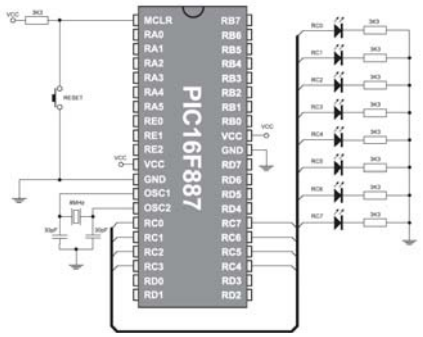
- ⇨ d'écrire votre code source avec un éditeur de texte intégré,
- ⇨ la génération des fichiers listes, assembleurs et Hex compatible avec tous les programmateurs,
- ⇨ l'intégration rapide des librairies,
- ⇨ de donner une aide en ligne des fonctions offertes par le compilateur,

4. Création d'un projet avec PICC Compiler

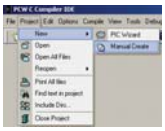
Le compilateur PICC compiler sauvegarde vos applications au sein de projets qui s'apparentent à un fichier «projet» unique (avec l'extension .pjt) ainsi qu'un ou plusieurs fichiers sources (avec l'extension .c)

4.1. Connexion matérielle

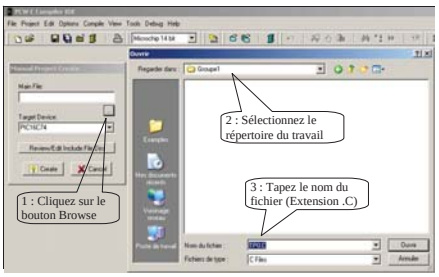
On propose ici le schéma de connexion ci-dessous pour tester le premier programme. Les diodes LEDs sont connectées au PORTC. Dans cet exemple, nous écrivons un programme très simple permettant de changer l'état du PORTC.



4.2. Création d'un nouveau projet

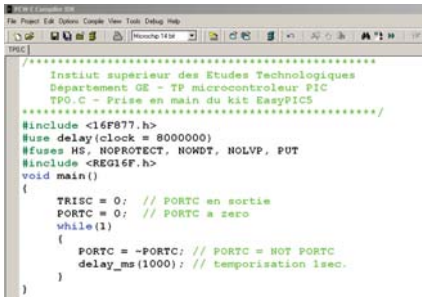


- ⇨ Sélectionner la commande **Manual Create** du sous menu **New,** du menu **Project**.
- ⇨ Une nouvelle fenêtre apparaîtra. Cliquer sur le bouton **Browse**. Dans la fenêtre qui s'ouvre sélectionnez le répertoire du travail (C:\AII2\Groupe1 par exemple) et tapez le nom du fichier (TP0.C par exemple).



Sélectionnez dans la fenêtre **Target Device** le microcontrôleur PIC16F877.

- ⇨ Après avoir renseigné toutes ces informations, cliquez sur le bouton **Create**. A ce stade, une nouvelle fenêtre d'édition s'affiche afin que vous puissiez saisir votre programme. Tapez alors le code suivant :

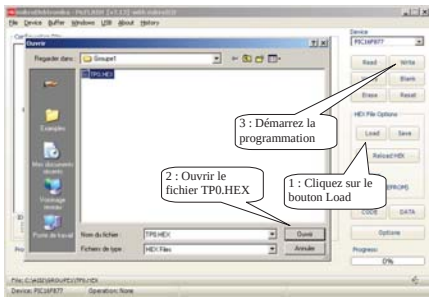


```
/*************************************************
    Instiut superieur des Etudes Technologiques
    Département GE - TP microcontroleur PIC
    TP0.C - Prise en main du kit EasyPIC5
*************************************************/
#include <16F877.h>
#use delay(clock = 8000000)
#fuses HS, NOPROTECT, NOWDT, NOLVP, PUT
#include <REG16F.h>
void main()
{
     TRISC = 0;  // PORTC en sortie
     PORTC = 0;  // PORTC a zero
     while(1)
     {
        PORTC = ~PORTC; // PORTC = NOT PORTC
        delay_ms(1000); // temporisation 1sec.
     }
}
```


4.3. Compilation et chargement du programme

Une fois que vous aurez créé votre projet et écrit le code source, vous pourrez le compiler en choisissant l'option *Compile* du menu **Compile**.
Après la compilation du code, Suivez les étapes suivantes pour programmer le composant.

- ⇨ Connectez le kit EasyPIC5 au PC via le port USB
- ⇨ Démarrez le logiciel **PICFlash**
- ⇨ Cliquez sur le bouton **Load**, et ouvrez le fichier TP0.HEX
- ⇨ Démarrez la programmation du microcontrôleur en cliquant sur le bouton **Write**
- ⇨ Attendez la fin de programmation, puis appuyez sur le bouton **RESET** pour démarrer votre application.



TP1 : PROGRAMMATION DES PORTS D'ENTREES-SORTIES

1. Objectifs

- ⇨ Utiliser l'environnement PICC Compiler
- ⇨ Etre capable de gérer les ports parallèles en C
- ⇨ Etre capable de décomposer un programme en s. programmes (fonctions en C)

2. Environnement de travail

- Système de développement intégré PICC Compiler, comportant un éditeur de texte, un compilateur C et un logiciel **PICFlash** permettant la programmation du composant.
- Une carte d'étude EasyPIC5 permettant l'implémentation et la vérification pratique des applications. Cette carte à base du microcontrôleur 16F877 est connectée au PC via le port USB.

3. Schéma de connexion

Le kit EasyPIC5 est dédié pour les microcontrôleurs PIC de Microchip. Il comprend 8 supports DIP de différentes tailles permettant de piloter la carte par des microcontrôleurs de la famille PIC10, PIC12, PIC16 ou PIC18. De plus 9 DIP switchs destinés à connecter les ports aux différents périphériques.
Dans ce TP on se limite à l'utilisation des boutons poussoirs et de diodes LEDs. Le schéma de la Figure 1-1 présente la connexion de ces périphériques de base aux différents PORTS du microcontrôleur.

4. Gestion des ports parallèles

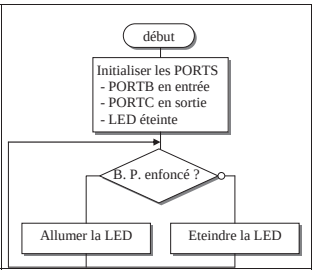
Le programme suivant permet d'allumer la LED RC2 lorsqu'on appui sur le B. Poussoir RB0.

```
// commande d'une LED

void main()
{
     TRISB = 0xFF ;
     TRISC = 0x00 ;
     PORTC = 0 ;
  while(1)
  {
    if((PORTB &0x01)== 0x01)

     PORTC = 0x04 ;
    else

     PORTC = 0x00 ;
  }
}
```



Remarques :

- ⇨ Seule la LED sur RC2 devant être modifiée, on aurait pu écrire : `PORTC=PORTC|0x04;` pour mettre RC2 à 1 et `PORTC=PORTC&0xFB;` pour mettre RC2 à 0.
- ⇨ Très souvent, les masques sont utilisés en C pour les tests ou le positionnement des bits, cependant ***PICC*** permet l'accès individuel aux bits.

Methode 1 :

```
#bit  LED = PORTC.2
#bit  BP  = PORTB.0
LED = 1;   // mettre le bit 2 du PORTC à 1
a  = BP;  // Lire l'état du bit 0 du PORTB
```

Methode 2 :

```
Bit_set(PORTC,2);          // mettre le bit 2 du PORTC à 1
a = Bit_test(PORTB,0);     // Lire l'état du bit 0 du PORTB
```

5. Travail à faire

1. Tapez et exécutez sur kit le programme de commande d'une LED (Page précédente).
2. Réécrire le programme en utilisant l'accès individuel aux bits.
3. Ecrire un programme permettant d'incrémenter le contenu du PORTC à chaque pression sur le bouton RB0.
   a. Est-ce que le compteur s'incrémente convenablement ? Pourquoi ?
   b. Ajoutez l'instruction « **`while( BP == 1);`** » après l'incrémentation du PORTC. Que constatez-vous ?
4. Programmation structurée (utilisation des fonctions)

```
#include <16F877.h>
#include "REG16F.h"
#use delay(clock = 8000000)
#fuses HS, NOWDT,NOLVP
#define duree 10000
unsigned int16 cnt ;
void main(void)
{//--------------------------Initialisation
       PORTC = 0x00;
       TRISC = 0x00;
 //-------------------------
       while(1) {
              PORTC++;
 //-------------------------Temporisation
              cnt = duree;
              while(cnt--);
 //-------------------------
       }
}
```

a. Réécrire le programme ci-contre en le décomposant en trois fonctions : fonction *Initialisation*, fonction *Tempo* avec un paramètre d'appel et une fonction principale *main*.
b. modifier le programme de manière à modifier la durée de temporisation. (passer de 10000 à 40000) si le bouton RB0 est appuyé

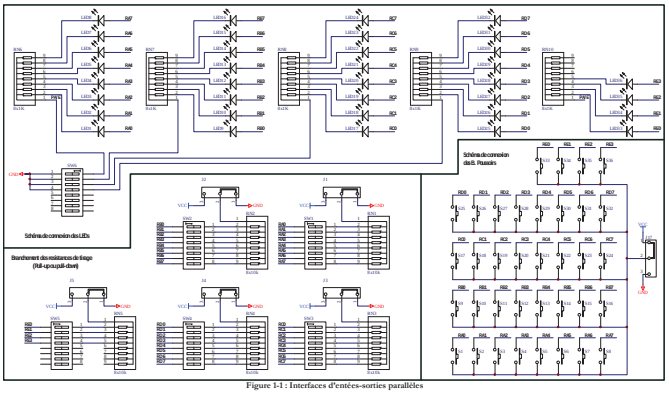


Figure 1-1 : Interfaces d'entées-sorties parallèles

TP2 : PROGRAMMATION DES TIMERS

1. Objectifs

- ⇨ Comprendre les modes de fonctionnement des Timers.
- ⇨ Mettre en œuvre les interruptions des Timers en langage C.

2. Rappel

Les timers sont en fait des compteurs formés généralement d'un **pré-diviseur** suivi d'un registre compteur de 8 ou 16 bits. L'entrée d'horloge peut être interne (mode timer) ou externe (mode compteur d'événements). Lorsque le registre compteur atteint sa valeur maximale et repasse à 0, un bit indicateur (*flag*) sera positionné et une interruption pourra être générée, informant ainsi la CPU du débordement du timer. Il faut bien noter que le bit indicateur de débordement devra être remis à 0 après chaque débordement.

2.1. Timer 0

Le timer 0 est formé d'un **pré-diviseur** suivi d'un registre compteur 8 bits (**TMR0**). 6 bits du registre **OPTION_REG** sont utilisés pour sa configuration. Les deux bits, indicateur de débordement ***T0IF*** et de validation de l'interruption ***T0IE*** se trouvent dans le registre **INTCON**.

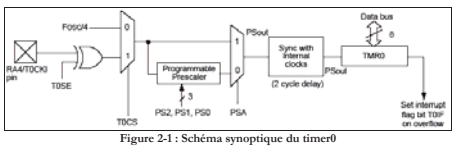


Figure 2-1 : Schéma synoptique du timer0

Registre OPTION_REG

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0

bit 5: **T0CS**: TMR0 Clock Source Select bit
1 = Transition on T0CKI pin
0 = Internal instruction cycle clock (CLKOUT)

bit 4: **T0SE**: TMR0 Source Edge Select bit
1 = Increment on high-to-low transition on T0CKI pin
0 = Increment on low-to-high transition on T0CKI pin

bit 3: **PSA**: Prescaler Assignment bit
1 = Prescaler is assigned to the WDT
0 = Prescaler is assigned to the Timer0 module

bit 2-0: **PS2:PS0**: Prescaler Rate Select bits

Bit Value	TMR0 Rate	WDT Rate
000	1 : 2	1 : 1
001	1 : 4	1 : 2
010	1 : 8	1 : 4
011	1 : 16	1 : 8
100	1 : 32	1 : 16
101	1 : 64	1 : 32
110	1 : 128	1 : 64
111	1 : 256	1 : 128

2.2. Timer 1

Le timer1 est formé d'un **prédiviseur** et d'un registre compteur 16bits (**TMR1H**, **TMR1L**). Ce timer offre des fonctionnalités intéressantes tels que :

- ⇨ la possibilité de connecter un quartz entre les broches RC0 et RC1.
- ⇨ le fonctionnement en mode asynchrone, ce qui lui permet de fonctionner même en mode veille,
- ⇨ bloquer à tout moment le comptage.

Tous les bits de configuration associés au timer1 se trouvent dans le registre **T1CON** excepté du bit indicateur de débordement ***TMR1F*** et du bit de validation de l'interruption ***TMR1E*** qui se trouvent respectivement dans les registres **PIR1** et **PIE1**.

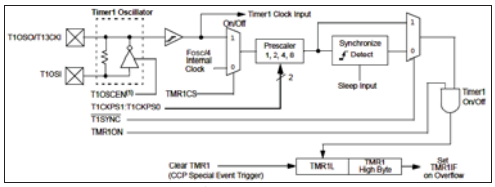


Figure 2-2 : Schéma synoptique du timer1

Registre T1CON

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON

bit 7-6: **Unimplemented:** Read as '0'

bit 5-4: **T1CKPS1:T1CKPS0**: Timer1 Input Clock Prescale Select bits
11 = 1:8 Prescale value
10 = 1:4 Prescale value
01 = 1:2 Prescale value
00 = 1:1 Prescale value

bit 3: **T1OSCEN**: Timer1 Oscillator Enable Control bit
1 = Oscillator is enabled
0 = Oscillator is shut off (The oscillator inverter is turned off to eliminate power drain)

bit 2: **T1SYNC**: Timer1 External Clock Input Synchronization Control bit
This bit is ignored. Timer1 uses the internal clock when TMR1CS = 0.

bit 1: **TMR1CS**: Timer1 Clock Source Select bit
1 = External clock from pin RC0/T1OSO/T1CKI (on the rising edge)
0 = Internal clock (Fosc/4)

bit 0: **TMR1ON**: Timer1 On bit
1 = Enables Timer1
0 = Stops Timer1

2.3. Timer 2

Ce timer est formé d'un prédiviseur, d'un registre compteur 8bits **TMR2** et d'un postdiviseur. Contrairement aux autres timers, le registre **TMR2** déborde chaque fois que son continu dépasse la valeur chargée dans le registre **PR2** ; de même le bit ***T2ON*** ne bloque pas le comptage comme dans le cas du timer1, mais il le met hors service. Les autres registres associés au timer 2 sont : le registre de configuration **T2CON** et les registres **PIR1** et **PIE1** qui contiennent respectivement le bit indicateur de débordement ***TMR2IF*** et le bit autorisation d'interruption ***TMR2IE***.

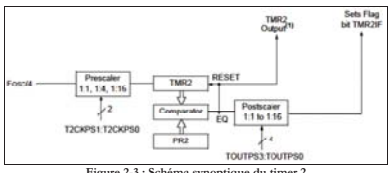


Figure 2-3 : Schéma synoptique du timer 2

Registre T2CON

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0

bit 7: **Unimplemented:** Read as '0'

bit 6-3: **TOUTPS3:TOUTPS0**: Timer2 Output Postscale Select bits
0000 = 1:1 Postscale
0001 = 1:2 Postscale
0010 = 1:3 Postscale
•
•
•
1111 = 1:16 Postscale

bit 2: **TMR2ON**: Timer2 On bit
1 = Timer2 is on
0 = Timer2 is off

bit 1-0: **T2CKPS1:T2CKPS0**: Timer2 Clock Prescale Select bits
00 = Prescaler is 1
01 = Prescaler is 4
1x = Prescaler is 16

Registre INTCON

INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF

Registre PIR1 et PIE1

PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE

2.4. Le mode scrutation

Nous savons maintenant que tous débordement du timer, entraîne le positionnement du bit indicateur de débordement (T0IF dans le cas du timer0 par exemple). Vous pouvez donc utiliser cet indicateur (*flag*) pour déterminer si le timer a débordé ou non.

Exp : `while(T0IF == 0);` *// Attendre le débordement du timer0 pour sortir.*
`T0IF = 0 ;` *// N'oubliez pas de remettre ce bit à 0*

2.5. Le mode interruption

C'est le mode principal d'utilisation des timers. En effet, pour qu'une interruption se déclenche à chaque passage du bit indicateur de débordement à 1 ; il faut positionner à l'avance les bits de validation de l'interruption (voir Figure 2-4).

- Pour que l'interruption timer 0 se déclenche lorsque T0IF passe à 1, il faut positionner les bits T0IE et GIE.
- Pour que l'interruption timer 1 se déclenche lorsque TMR1IF passe à 1, il faut positionner les bits TMR1IE, PEIE et GIE

- Pour que l'interruption timer 2 se déclenche lorsque TMR2IF passe à 1, il faut positionner les bits TMR2IE, PEIE et GIE

Dés que le timer déborde, La CPU se branche automatiquement à une routine précédée par le mot reservé *#int_timer0* par exemple.

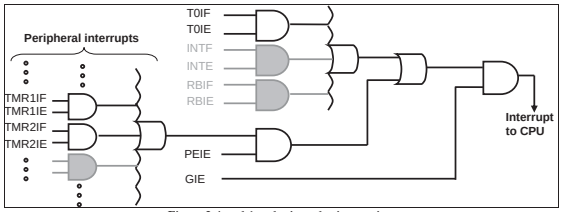


Figure 2-4 : schéma logique des intrruptions

2.6. Exemple de programmation d'une inrerruption (IT)

On veut faire clignoter la LED RC2 à une fréquence approximativement égale à 2 Hz. On admet que la fréquence du Quartz $F_{osc} = 8\ MHz$.

La fréquence de clignotement Fc ≈ 2 Hz ⇨ Tc ≈ 500ms.
Le timer provoque à chaque débordement, une interruption. La période de débordement T_d doit être la moitié de T_c, donc $T_d \approx 250ms$.
La période maximale de débordement du Timer 0 :
$T_{d0max} = 4 \bullet T_{osc} \bullet Prédiv_{max} \bullet 2^8 = 4 \bullet 0{,}125\mu s \bullet 256 \bullet 256 = 32{,}768ms$; Or nous voulons une période de débordement de 250ms environ. Cherchons alors le période maximale de débordement du Timer 1, $T_{d1max} = 4 \bullet T_{osc} \bullet Prédiv_{max} \bullet 2^{16} = 4 \bullet 0{,}125\mu s \bullet 8 \bullet 65536 = 262{,}144ms$; cette valeur pourra être acceptable.
On doit configurer le timer1, en mode timer (horloge interne) avec une valeur de prédivision égale à 8 ; d'où T1CON = 00110001 = 31H.

```
// Mise en oeuvre du timer1
#include <16F877.h>
#use delay(clock = 8000000)
#fuses HS, NOPROTECT, NOWDT, NOLVP, PUT
#include <REG16F.h>
void Init()
{
      TRISC = 0x00 ;          // PORTC en sortie
      T1CON = 0x31 ;          // configuration du Timer 1
      TMR1IE = 1;             // validation de l'IT Timer1
      PEIE = 1 ; // Validation de l'IT Périphériques
      GIE = 1 ;         // validation globale des IT
}
#int_timer1
void interrupt()              // routine d'IT
{
      RC2 = !RC2 ;            // Inversion de l'état de la LED
}
```



```
void main()
{
      Init() ;                // appel de la procédure Init
      while(1);               // boucle infinie
}
```

3. Travail demandé

3.1. Préparation

On veut configurer l'un des timers pour qu'il déborde toutes les 20ms. La fréquence du microcontrôleur F_{osc} = 8 Mhz.

1. Quel est le mode de fonctionnement à choisir (mode timer ou mode compteur) ?
2. Donner pour chacun des timers, les valeurs à charger dans les registres qui lui sont associés.
   a. Pour le Timer 0 : Prediv = ?; TMR0 = ?; OPTION_REG = ?
   b. Pour le Timer 1 : Prediv = ?; TMR1L = ?; TMR1H = ?; T1CON = ?
   c. Pour le Timer 2 : Prediv = ?; PR2 = ?; Postdiv = ?; T2CON = ?
3. Quel est le timer qui convient mieux.

3.2. Manipulation

Le mode Timer

1. Tester le programme de clignotement d'une LED.
2. Modifier le programme pour que la LED clignote à 2 Hz exactement.
3. Réécrire le programme précédent en mode scrutation.
4. Réécrire le programme de clignotement d'une LED, en utilisant le timer2 au lieu du timer1.

Le mode Compteur

1. Ecrire un programme qui incrémente le contenu du registre TMR0 à chaque appui sur le bouton RA4. Afficher le compte sur les LEDs connectées au PORTC.
2. Modifier le programme pour que le registre TMR0 s'incrémente après 4 appuis sur le bouton RA4.

TP3 : AFFICHAGE MUTILPLEXE

1. Objectifs

⇨ Etudier l'affichage multiplexé.
⇨ Mettre en œuvre les Timers pour le rafraîchissement des afficheurs 7 segments.

2. Présentation

2.1. Afficheur 7 segments

Un afficheur 7 segments, est constitué de 7 diodes électroluminescentes (LED) disposées de manière à représenter les chiffres décimaux. Il existe deux types d'afficheurs, selon la mise en commun des anodes (Anodes Communes) ou bien des cathodes (Cathodes communes). Ces afficheurs sont généralement commandés par des décodeurs permettant d'associer à chaque chiffre la combinaison des LEDs à éclairer. Le Tableau 3-1 présente les combinaisons nécessaires à la commande des afficheurs à cathodes communes.

Chiffres	dp	g	f	e	d	c	b	a	Val. en hexa
0	0	0	1	1	1	1	1	1	0x3F
1	0	0	0	0	0	1	1	0	0x06
2	0	1	0	1	1	0	1	1	0x5B
3	0	1	0	0	1	1	1	1	0x4F
4	0	1	1	0	0	1	1	0	0x66
5	0	1	1	0	1	1	0	1	0x6D
6	0	1	1	1	1	1	0	1	0x7D
7	0	0	0	0	0	1	1	1	0x07
8	0	1	1	1	1	1	1	1	0x7F
9	0	1	1	0	1	1	1	1	0x6F

Tableau 3-1 : Décodage BCD vers 7 segments

2.2. Affichage multiplexé

Ce mode de commande a pour but de réduire le nombre de connexions. Cette technique consiste à connecter les afficheurs en parallèle, et à les sélectionner à tour de rôle à une fréquence suffisamment élevée afin d'éliminer le papillotement. La Figure 3-1 présente le schéma de connexion de 4 afficheurs à cathodes communes aux ports A et D du microcontrôleur PIC.
Nous allons présenter les étapes nécessaires à l'affichage de la valeur d'une variable *x*, (prenant *x* = 1234) :

a. conversion binaire_BCD, qui consiste à déterminer les chiffres qui composent le nombre *x* (millième = 1, centaine =2, dizaine =3, unité =4)
b. conversion BCD_7Segments, consiste à chercher pour chaque chiffre la combinaison correspondante (DIS3 ⇦ 0x06, DIS2 ⇦ 0x5B, DIS1 ⇦ 0x4F, DIS0 ⇦ 0x66)

c. mettre sur le PORTD la combinaison correspondante au chiffre des unités (PORTD = 0x66) et sélectionner le 1er afficheur (Activer la ligne RA0)
d. mettre sur le PORTD la combinaison correspondante au chiffre des dizaines (PORTD =0x4F) et sélectionner le 2ème afficheur (Activer la ligne RA1)
e. mettre sur le PORTD la combinaison correspondante au chiffre des centaines (PORTD = 0x5B) et sélectionner le 3ème afficheur (Activer la ligne RA2)
f. mettre sur le PORTD la combinaison correspondante au chiffre des millièmes (PORTD = 0x06) et sélectionner le 3ème afficheur (Activer la ligne RA3)

Les étapes *c* à *f* doivent être exécutées en boucle et sans arrêt, à une fréquence comprise entre 200 et 1000 Hz (soit de 1 à 5ms).

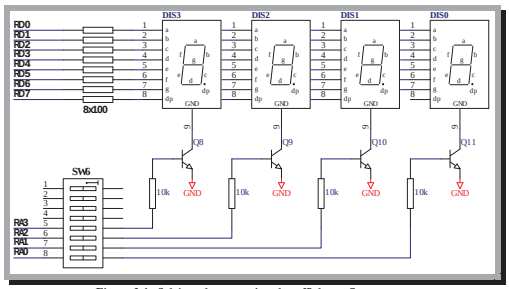


Figure 3-1 : Schéma de connexion des afficheurs 7 segments

```
// Affichage de la valeur 1234
// Programme Aff1234.c
#include <16F877.h>
#use delay(clock = 8000000)
#fuses HS, NOPROTECT, NOWDT, NOLVP, PUT
#include <REG16F.h>

void main()
{
   ADCON1 = 6 ;
   TRISD = 0;
   TRISA = 0;
   PORTA = 0;
   while(1)
   {
     PORTA = 0; PORTD = 0x66; PORTA = 1; // affichage des unités
     delay_ms(4);
     PORTA = 0; PORTD = 0x4F; PORTA = 2; // affichage des dizaines
     delay_ms(4);
     PORTA = 0; PORTD = 0x5B; PORTA = 4; // affichage des centaines
     delay_ms(4);
     PORTA = 0; PORTD = 0x06; PORTA = 8; // affichage des millièmes
     delay_ms(4);
   }
}
```


En réalité, *x* est une variable qui peut prendre n'importe quelle valeur, dans ce cas il faut exécuter les étapes *a* et *b* avant de procéder à l'affichage. Ces deux étapes peuvent être réalisées par une seule fonction appelée BinTo7SEG.

```
-----------------------------------------------------------------
// Programme AFF7SEG.c
#include <16F877.h>
#use delay(clock = 8000000)
#fuses HS, NOPROTECT, NOWDT, NOLVP, PUT
#include <REG16F.h>
char T7SEG[]={0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F} ;
char AFFSEL = 0x01 ;
char TABVAL[4],i;
long x = 9805;

void BinTo7SEG(int val)
{   char m,c,d,u;
     u = val%10;
     TABVAL[0] = T7SEG[u];
     d = (val/10)%10;
     TABVAL[1] = T7SEG[d];
     c = (val/100)%10;
     TABVAL[2] = T7SEG[c];
     m = val/1000;
     TABVAL[3] = T7SEG[m];
}
void main()
{
     TRISD = 0;
     TRISA = 0;
     PORTA = 0;

     while(1)
     {BinTo7SEG(x);
      AFFSEL = 1;
       for (i = 0; i<4 ;i++)
        {
         PORTA = 0; PORTD = TABVAL[i]; PORTA = AFFSEL; AFFSEL <<= 1;
         delay_ms(4);
        }
     }
}
```

3. Travail demandé

1. Ecrire et exécuter le programme AFF7SEG.c.
2. Modifier le programme AFF7SEG.c, afin de réaliser un compteur Module 100, en incrémentant la valeur de *x* à chaque seconde. Que peut-on dire de l'affichage ?
3. On remarque que cette méthode atteint ses limites lorsqu'il y a une ou plusieurs tâches qui s'exécutent en parallèles avec l'affichage. Pour cela, on utilise les interruptions timers.
   a. Calculer les valeurs à donner aux registres T2CON et PR 2 pour avoir une période de débordement de 4ms.
   b. Réécrire le programme précédent en utilisant la méthode interruption, la partie affichage devra être gérée par la routine d'interruption.

TP4 : CHRONOMETRE NUMERIQUE

1. Présentation

La Figure 4-1 présente le schéma d'un chronomètre numérique. Le bouton poussoir START déclenche le comptage, le bouton STOP arrête le comptage, alors que le bouton RESET remet le compteur à zéro. L'afficheur DIS0 affiche les dixièmes de seconde, les afficheurs DIS1 et DIS2 les secondes et l'afficheur DIS3 les minutes.

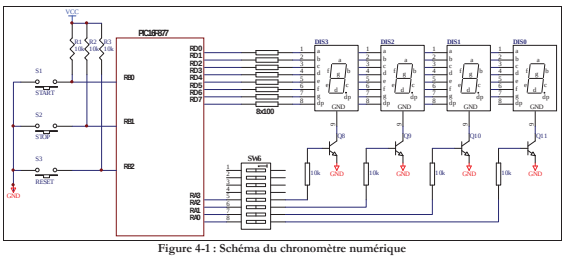


Figure 4-1 : Schéma du chronomètre numérique

2. Démarche à suivre

On propose ici de décomposer le système selon le diagramme de la **Erreur ! Source du renvoi introuvable.**.

- la fonction ***GererConsigne***, : gère les événements issus des Boutons poussoirs et met à jour la variable *ConsFlag*.
- La fonction ***Comptage*** : met à jour les variables *DixSec, Sec et Mins* toutes les 100 ms.
- La fonction ***BinTo7SEG*** : convertit en 7Seg. les valeurs à afficher.
- La fonction ***Affiche*** : gère l'affichage multiplexé (voir TP3) ; cette fonction est appelée dans la routine d'interruption.

3. Travail demandé

Compléter le programme suivant :

```
#include <16F877.h>
#use delay(clock = 8000000)
#fuses HS, NOPROTECT, NOWDT, NOLVP, PUT
#include <REG16F.h>
//------------------- Déclarations des variables globales ---------------
char CNT=25;              // 100ms/4ms = 25
char DixSec=0 ;           // dixièmes des secondes
char Sec=0 ;              // Secondes
char Mins=0 ;             // Minutes
char ConsFlag = 0;        // ConsFlag = 0 ⇨ remise à 0 du compteur
                          // ConsFlag = 1 ⇨ Comptage
                          // ConsFlag = 2 ⇨ Arrêt
char T7SEG[]={0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F} ;
char AFFSEL = 0x01 ;    // sélection des afficheurs
```



```
char TABVAL[4] ;          //tableau contenant les valeurs à envoyer aux affi.
char NumAff=0;            // numéro de l'afficheur
//-------------- Prototypes des fonctions --------------------------
void comptage(char);
void BinTo7SEG(char,char,char);
void Affiche();
//-----------------------------------------------------------------
void interrupt()
{
      ..... à compléter
      TMR2IF = 0 ;
}
void Comptage(char Flag)
{
      ..... à compléter
}
void Affiche()
{
      ..... à compléter
}
void BinToSEG(char DixSc, char Sc, char Mn)
{
      ..... à compléter
}
void GererConsigne()
{
     if( RB0 == 1)
         ConsFlag = 1;
     if( RB1 == 1)
         ConsFlag = 2;
     if( RB2 == 1)
         ConsFlag = 0;
}
void Init()
{    ..... à compléter
     TMR2IE = 1 ;         // validation globale des IT
     PEIE = 1 ;           // validation IT des périphériques
     GIE = 1 ;            // validation globale des IT
}
void main()
{   Init();
     while(1)
      GererConsigne();
}
```

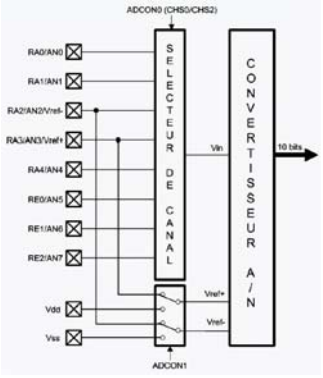
TP5 : CONVERTISSEUR A – N

1. Objectifs

- ⇨ Apprendre à programmer le convertisseur Analogique - Numérique
- ⇨ Utiliser les bibliothèques du PCW Compiler

1. Présentation

Le microcontrôleur PIC16F877 dispose d'un convertisseur analogique numérique 8 voies avec une résolution sur 10 bits. La figure suivante présente le schéma synoptique.



Le convertisseur A/N utilise les broches RA0, RA1, RA2, RA3 et RA5 du port A et les trois broches du port E (RE0 .. RE2) pour former les huit entrées. Les entrées RA2 et RA3 peuvent être utilisées comme niveaux de références.

2. Programmation du convertisseur analogique - numérique

Pour programmer le convertisseur, on dispose de 4 registres :

2.1. Les registres ADRESL et ADRESH

C'est dans ces deux registres que le convertisseur charge le résultat de conversion. Selon la valeur du bit ADFM du registre ADCON1, le résultat peut être justifié à gauche ou à droite.

ADFM = 1

ADRESH								ADRESL							
0	0	0	0	0	0	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0

ADFM = 0

ADRESH								ADRESL							
b9	b8	b7	b6	b5	b4	b3	b2	b1	b0	0	0	0	0	0	0

2.2. Le registre ADCON1

Ce registre permet de déterminer la configuration de chaque broche AN0 .. AN7.

- ***b7 : ADFM*** : A/D result ForMat select
- ***b6 : Inutilisé*** : lu comme « 0 »
- ***b5 : Inutilisé*** : lu comme « 0 »
- ***b4 : Inutilisé*** : lu comme « 0 »
- ***b3 : PCFG3*** : Port ConFiGuration control bit 3
- ***b2 : PCFG2*** : Port ConFiGuration control bit 2
- ***b1 : PCFG1*** : Port ConFiGuration control bit 1
- ***b0 : PCFG0*** : Port ConFiGuration control bit 0

PCFG 3 .. 0	AN7 RE2	AN6 RE1	AN5 RE0	AN4 RA5	AN3 RA3	AN2 RA2	AN1 RA1	AN0 RA0
0000	A	A	A	A	A	A	A	A
0001	A	A	A	A	Vref+	A	A	A
0010	D	D	D	A	A	A	A	A
0011	D	D	D	A	Vref+	A	A	A
0100	D	D	D	D	A	D	A	A
0101	D	D	D	D	Vref+	D	A	A
0110	D	D	D	D	D	D	D	D
0111	D	D	D	D	D	D	D	D
1000	A	A	A	A	Vref+	Vref-	A	A
1001	D	D	A	A	A	A	A	A
1010	D	D	A	A	Vref+	A	A	A
1011	D	D	A	A	Vref+	Vref-	A	A
1100	D	D	D	A	Vref+	Vref-	A	A
1101	D	D	D	D	Vref+	Vref-	A	A
1110	D	D	D	D	D	D	D	A
1111	D	D	D	D	Vref+	Vref-	D	A

A : Analogique **D** : Digitale

2.3. Les registre ADCON0

- ***b7 : ADCS1*** : A/D conversion Clock Select bit 1
- ***b6 : ADCS0*** : A/D conversion Clock Select bit 0 (voir tableau)
- ***b5 .. b3 : CHS2 .. CHS0*** : choix du canal AN0/../AN7
- ***b2 : GO/DONE*** : A/D conversion status bit
- ***b1 : Inutilisé*** : lu comme « 0 »
- ***b0 : ADON*** : A/D ON bit (mettre en ou hors service l'ADC)

Le bit **GO/DONE** est un bit de début de conversion, il sert aussi comme indicateur de fin de conversion puisqu'il se remet à zéro à la fin de la conversion. A la fin de conversion, le bit ADIF du registre PIR1 sera positionné, et une interruption pourra être générée si le bit ADIE est positionné.

ADCS1	ADCS0	Diviseur	Fréquence maximale du quartz
0	0	Fosc/2	1,25Mhz
0	1	Fosc/8	5Mhz
1	0	Fosc/32	20 Mhz
1	1	Osc RC	Si > 1MHz, uniquement en mode « sleep »

2.4. Etapes de programmation du convertisseur

1) Configurez ADCON1 en fonction des pins utilisées en mode analogique, ainsi que les registres TRISA et TRISE si nécessaire.
2) Validez, si souhaité, l'interruption du convertisseur (ADIE)
3) Paramétrez sur ADCON0 le diviseur utilisé
4) Choisissez le canal en cours de digitalisation sur ADCON0
5) Positionnez, si ce n'est pas déjà fait, le bit ADON du registre ADCON0
6) Attendez le temps Tacq (typiquement 19,7µs sous 5V)
7) Démarrez la conversion en positionnant le bit GO du registre ADCON0
8) Attendez la fin de la conversion (test sur le GO/DONE ou ADIF)
9) Lisez les registres ADRESH et si nécessaire ADRESL
10) Attendez un temps équivalent à 2Tad (typiquement 3,2µs)
11) Recommencez au point 7 (ou 4 si vous changez le canal).

3. Schéma de connexion

Le kit EasyPIC5 dispose de trois entrées analogiques. Deux entrées sont délivrées par deux potentiomètres P1 et P2, alors que la troisième est obtenue à partir d'un capteur de température DS1820. Les cavaliers J15, J16 et J11 permettent de connecter les trois entrées analogiques au microcontrôleur selon le schéma de la Figure 5-1.

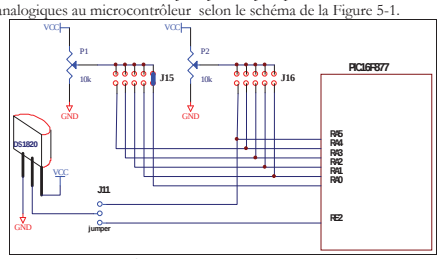


Figure 5-1 : Schéma de connexion des entrées analogiques

4. Travail demandé

Ecrire un programme qui permet de convertir un signal analogique appliqué à l'entrée AN0 (signal délivré par le potentiomètre P1). La valeur convertie sera envoyée sur le port B. La période d'échantillonnage étant fixée par le timer0 à 32 ms environ.

1- Donner les valeurs à charger dans les registres ADCON1, ADCON0, OPTION_REG et TRISB.

2- Compléter le programme suivant :

```
void Init()
{
        ...........
        T0IE = 1;    // validation de l'interruption du Timer0
        PEIE = 1;    // validation de l'interruption des périphériques
        GIE = 1;     // validation Globale
}
#int_Timer0
void Timer0_ISR()
{
      ............
}
void main()
{  Init();
   while(1) ;
}
```

3- Reprendre le programme précédent, en utilisant deux sources d'interruption ; interruption timer0 et interruption ADC.

4- Réécrire le programme précédent en utilisant le mode scrutation.

TP6 : AFFICHEUR LCD

1. Objectifs

- ⇨ Gérer un afficheur LCD sur EasyPIC5
- ⇨ Utiliser les bibliothèques de composants logiciels de PCW Compiler

2. Afficheur LCD

L'afficheur que nous allons utiliser possède 2 lignes 16 caractères, Chaque caractère est inscrit dans une matrice de 5 colonnes de 7 points.
La Norme HD44780, définit une liaison parallèle pour la connexion de l'afficheur. Cette liaison permet d'envoyer de commandes ou des données. Nous présentons ici les différents signaux d'interfaçage de l'afficheur LCD à un microprocesseur ou un microcontrôleur :

- RS (Registrer Select) : indique si la commande envoyée est une commande (RS = 0) ou une donnée (RS = 1)
- R/W (Read-Write) : sens du transfert, vers (R/W = 0) ou à partir de l'afficheur (R/W = 1)
- E (Enable) : permet la validation de donnée sur front descendant.
- D0 à D7 : lignes de données. L'afficheur peut être utilisé en mode 4 bits pour économiser les lignes ; dans ce cas le transfert d'un octet ce fait en deux temps sur les lignes D4 à D7.

Vous trouvez en annexe une description détaillée de toutes les commandes.

2.1. Schéma de connexion

Le kit EasyPIC5, utilise le PORTB pour la commande de l'afficheur LCD, on remarque d'après le schéma de la Figure 6-1 que l'afficheur est utilisé en mode 4 bits et que la ligne R/W est connectée à la masse.

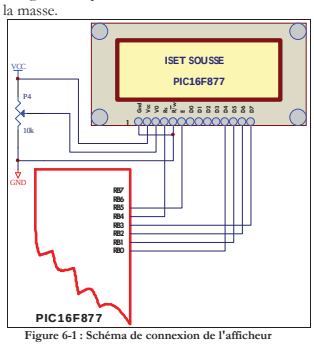


Figure 6-1 : Schéma de connexion de l'afficheur

2.2. Utilisation du driver Lcd4bits.c

On fait appel à un driver écrit en C pour la gestion de l'afficheur. Il comporte les fonctions suivantes :
void lcd_init() : fonction d'initialisation de l'afficheur ; elle devra être appelée au début du programme.
void lcd_gotoXY(int x, int y) : elle permet de positionner le curseur. ***x*** colonne de 1 à 16 et ***y*** ligne de 1 à 2.
void lcd_putc(char c) : cette fonction envoie un caractère à l'LCD. Le caractère '\f' efface l'afficheur. De même cette fonction permet aussi d'envoyer une chaine de caractères constante.
Exemple : `lcd_putc("\fPICC Compiler") ;` : efface l'LCD et affiche le message "PICC Compiler".
La fonction *lcd_putc* peut être utilisée avec la fonction *printf*. Exemple :

```
printf(lcd_putc, "\f X = %u", X) ;
```

void ShowCursor() : permet de faire apparaitre le curseur.
void MaskCursor() : permet de masquer le curseur.

3. Travail à faire

1) Tapez et exécutez le programme suivant :

```
#include <16F877.h>
#use delay(clock = 4000000)
#fuses HS, NOWDT, NOLVP, PUT
#include "REG16F.h"
#include <lcd4bits.c>
void Init()
{
      lcd_init();
}
void main()
{     Init();
      lcd_putc("ISET SOUSSE");
      lcd_gotoXY(1,2);
      lcd_putc("pic16F877");
   while(1)
   {
   }
}
```

2) Reprendre la question 4 du TP précédent en affichant le résultat de conversion sur l'afficheur LCD.
3) On suppose que la tension à l'entrée du convertisseur analogique numérique correspond à la température d'un four, dont la valeur maximale est égale à 100°C. Apportez les modifications nécessaires au programme précédent pour afficher la température sur l'LCD.
4) Modifier le programme précédent en affichant la température avec un chiffre après la virgule.

TP7 : COMMUNICATION SERIE ASYNCHRONE

1. Objectifs

- ⇨ Mettre en œuvre la Communication série Asynchrone (USART)
- ⇨ Utiliser les bibliothèques du compilateur PCW

2. Communication série

Le microcontrôleur PIC16F877 dispose de deux modules de communication série :

- ✓ Le module USART : c'est un module qui fonctionne en mode Synchrone et Asynchrone, il est souvent utilisé en mode asynchrone pour connecter le microcontrôleur à un PC.
- ✓ Le module MSSP (Master Synchronous Serial Port), ce module travaille en mode SPI ou en mode I2C. Il permet de connecter au microcontrôleur des afficheurs, des mémoires RAM et EEPROM séries, des détecteurs de température, des horloges en temps réel etc.

3. Communication série asynchrone

3.1. Présentation

La communication série RS232 est généralement utilisée pour le transfert de données entre le microcontrôleur est le PC. Comme les niveaux des tensions ne sont pas compatibles, on insère le circuit MAX232 pour l'adaptation des niveaux des tensions. Pour rendre le système plus flexible, le microcontrôleur est connecté au MAX232 via deux groupes de switch SW7 et SW8 (voir Figure 7-1). Il est à remarquer que les broches RC2 et RB0 sont connectées respectivement aux lignes RTS et CTS pour permettre l'implémentation du contrôle du flux matériel.

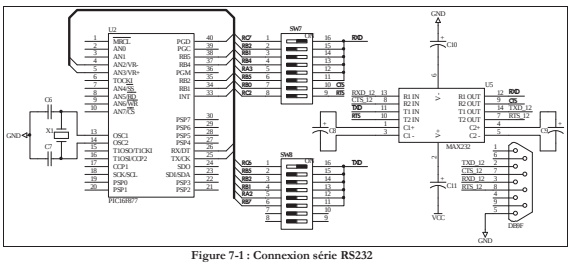


Figure 7-1 : Connexion série RS232

3.2. Librairie RS232

La librairie RS232 contient toutes les fonctions permettant de gérer la communication asynchrone :

- ➢ **#use rs232(options)**
- ➢

Les Options sont séparés par des virgules :

STREAM=id	Associe un identificateur de flux (stream) identifier avec le port série RS232. Cet identificateur est utilisé avec la fonction fputc.
BAUD=x	Définie le débit de transmission (1200, 2400,…)
XMIT=pin	Définie la broche de transmission (pin_C6 par exemple)
RCV=pin	Définie la broche de réception (pin_C7 par exemple)
FORCE_SW	Génère les fonctions E/S série logicielles. Cette option est utilisée surtout lorsque le microcontrôleur n'intègre pas un module série.
PARITY=X	Définie la parité N, E, ou O.
BITS =X	Nombre de bits de données (8 ou 9).

- ➢ **Fonctions d'émission**

Syntaxe : putc (***cdata***)
putchar (***cdata***)
value = fputc(***cdata***, ***stream***)
Cette fonction permet d'envoyer le caractère *cdata* sur le port série. Pour la fonction fputc, il faut ajouter l'identificateur du flux *stream*.

Syntaxe : puts (***string***).
value = fputs (***string***, ***stream***)
Cette fonction permet d'envoyer une chaine de caractères sur la liaison RS232. La chaine string peut être une chaine constante ou un tableau de caractères se terminant par le caractère null ('\0'). Après l'envoie de la chaine *string*, les caractères CR(13) et LF(10) sont aussi envoyés.
Syntaxes : printf (***string***)
printf (***cstring***, ***values***...)
printf (***fname***, ***cstring***, ***values***...)
fprintf (***stream***, ***cstring***, ***values***...)
Cette fonction permet d'envoyer une chaine de caractère formatée sur la liaison RS232.

string : peut être une constante de type chaine ou un tableau de char terminé par le caractère null.

values : est une liste de variables séparées par des virgules.

cstring : doit être composée de chaines de caractères constante et d'arguments de mise en forme des valeurs représenté par le symbole %.

Le formatage des variables :

%C	caractère
%S	Chaine ou caractère
%U	Entier non signé (8 bits)
%x	Octet en Hexadécimal (minuscule)
%X	Octet en Hexadécimal (majuscule)
%D	Entier singé (8 bits)
%e	Réel en exposant.
%f	Réel
%Lx	Double octets en Hexadécimal (minuscule)
%LX	Double octets en Hexadécimal (majuscule)
%lu	Entier long non signé (16 ou 32 bits)
%ld	Entier long signé (16 ou 32 bits)

fname : nom de la fonction utilisée pour l'envoie du caractère (par défaut c'est la fonction *putc*).

stream : définie l'identificateur du flux (par défaut *stdout*).

➢ **Fonctions de réception**

Syntaxe : value = kbhit()
value = kbhit (***stream***)
La fonction *kbhit* renvoie la valeur booléenne "*true*" s'il y a un caractère dans le buffer de réception, et "*false*" dans le cas contraire.

Syntaxe : value = getc()
value = fgetc(***stream***)
value=getch()
value=getchar()
Cette fonction permet de lire un caractère se trouvant dans le buffer de réception.

Syntaxe : gets (***string***)
value = fgets (***string***, ***stream***)
Cette fonction permet de lire les caractères d'une chaine jusqu'à la réception du caractère CR (13).

4. Travail demandé

4.1. Préparation

On veut établir une communication série entre le PC et le kit EasyPIC5 selon les paramètres suivants : 9600 bps, parité paire, un bit de stop et sans contrôle de flux

1- Calculez la durée d'émission d'un bit lors d'une transmission.
2- Calculez la durée d'émission d'une trame
3- Si on émet le caractère B
- Déterminez son code ASCII
- Déterminez et justifiez la valeur du bit de parité pour l'émission de ce caractère.
- Représentez le chronogramme de la trame correspondant à ce caractère.

4.2. Manipulation

1- Ouvrir l'**`hyper terminal`** de Windows, avec la configuration suivante : 9600 bps, 8bits de données, pas de parité, un bit de stop et sans contrôle de flux.
2- Le programme suivant renvoi au PC les caractères tapés au clavier (mode écho) :

```
#include <16F877.h>
#include "REG16F.h"
#use delay(clock = 4000000)
#use rs232(baud = 9600, xmit = pin_C6, rcv = pin_C7)
 char c;
void main()
{
    while(1)
   {    if(kbhit())
         {
            c = getc();
            putc(c);
         }
   }
}
```

3- Programmation d'un code : écrire un programme permettant de recevoir à partir du PC une chaine de caractères se terminant par CR (13). La chaine reçue sera comparée à la chaine constante "ISETSO" ; en cas d'égalité le programme renvoi un accusé de réception ACK (06) et allume la LED RD0 pendant 500ms.

ANNEXES

Compilateur C pour Pic - PCW Compiler -

1. Pré processeur directives

Toutes les directives du pré processeur commencent par « **#** » suivies d´une commande.

#ASM et ENDASM : Cette directive est utilisée lorsqu'on veut insérer un code assembleur.

#BIT : Crée un bit à un emplacement mémoire donné.
Syntaxe : ***#Bit id = x.y*** (*x* : adresse de l'octet et *y* : numéro du bit)
Exemple : #BIT T0IF = 0x0B.2

#BYTE : définie l'emplacement mémoire d'une variable de 8 bits
Syntaxe : ***#Byte id = x*** (*x* est l'adresse de la variable *id*)
Exemple : #BYTE PORTA = 5

#DEFINE : définie une constante
Syntaxe : ***#Define id Val ou #DEFINE id(x, y) Expression***
Exemples : #DEFINE PORTA 5
#DEFINE h(x) (x<<4)
a =h(a) ; // a = (a<<4)

#DEVICE : spécifie le processeur utilisé
Syntaxe : ***#Device Chip Options***
Chip : circuit utilisé, exemple PIC16F877
Options : *=Y ;Y∈{5, 8, 16} ; Y=16 pour les circuits 14 bits
ADC=ZZ ; ZZ∈{8, 10, 11, 16} ; nombre de bits qui définissent la résolution de l'ADC (pour le Pic16F877, le résultat de conversion A/D est codée sur 8 ou 10 bits).
Exemple : #DEVICE PIC16F877 *=16 ADC = 8

#FUSES : Directive de configuration du registre CONFIG
Syntaxe : ***#Fuses Options***
Options : LP, XT, HS, RC, CPD, NOCPD, WDT, NOWDT, PROTECT, NOPROTECT, PUT, NOPUT, BROWNOUT, NOBROWNOUT, LVP, NOLVP, WRT, NOWRT
Exemple : #fuses XT, NOWDT, NOPROTECT, PUT, NOLVP

#INCLUDE : Inclure un fichier
Syntaxe : ***#Include <Nom_fichier> ou #Include "Nom_fichier"***

#INLINE : informe le compilateur que la fonction juste après cette directive est implémentée en INLINE.
Exemple : #inline
Swapbyte (int &x, int &y)
{int t ;
t = x ; x = y ; y = t ; }

#INT_XXX : Indique que la fonction suivante est une routine d'interruption.
Syntaxe : ***#INT_AD*** : fin de conversion analogique numérique
#INT_CCP 1 : interruption capture compare 1
#INT_CCP 2 : interruption capture compare 2
#INT_EEPROM : fin d'écriture sur l'EEPROM

#INT_EXT : interruption INT (broche RB0)
#INT_I2C : interruption I2C
#INT_PSP: interruption « Parallel Slave Port »
#INT_RB : interruption sur changement d'état des broches RB4..RB7
#INT_RDA : réception de donnée sur l'RS232
#INT_RTCC ou #INT_TIMER0 : débordement du timer0
#INT_SSP : interruption SPI ou I2C
#INT_TBE : Buffer de transmission de l'RS232 vide
#INT_TIMER1 : débordement du timer1
#INT_TIMER2 : débordement du timer2

Exemple : #INT_AD
```
Isr_ad()            // Routine d'interruption du convertisseur
{                   // Analogique numérique
   ....
}
```

#LOCATE : définie l'adresse d'une variable
Syntaxe : ***#Locate id = Adresse***
Exemple : float x ;
#locate x = 0x22 ;

#PRIORITY : classer les interruptions par priorité
Syntaxe : ***#Priority Ints***
Exemple : #priority ad , rtcc, rb

#USE DELAY : Cette directive définie la fréquence du quartz , pour permettre au compilateur de calculer les temporisations (delay_us(), delay_ms())
Syntaxe : ***#Use Delay (clock = Vitesse)*** // Vitesse = fréquence du quartz en Hz
ou ***#Use Delay (clock = Vitesse, restart_wdt)***
Exemple : #use_delay (clock = 4000000, restart_wdt)

#USE RS232 : configuration du port série
Syntaxe : ***#Use RS232 (baud = x, xmit = pin, rcv = pin)***
Exemple : #use rs232 (baud = 9600, xmit = PIN_C6, rcv = PIN_C7)

2. Déclaration des variables

Syntaxe : ***Type var1,…., varn ;***
Différents types

int1	Définie un 1 bit
int8	Entier codé sur 8 bits
int16	Entier codé sur 16 bits
int32	Entier codé sur 32 bits
char	Caractère codé sur 8 bits
float	Nombre en virgule flottante sur 32 bits
short	Equivalent à int1
Int	Equivalent à int8
long	Equivalent à int16

Les mots-clé **const** et **static** peuvent être employés après le type.
Exemples : int a, b, c ;
int tab[3][2]
char *c ;


```
enum boolean {false, true} ;
boolean j ;
int const X = 20 ;
int const Tab[4] = {23, 200, 54, 2} ;
struct donnee
 {    int       a[2] ;
      boolean   keyhit ;
      int       tow : 2 ;  // deux bits
 }
```

3. Les fonctions

CCS supporte le passage des paramètres par valeur, par adresse et par référence. Le passage des paramètres par référence est plus efficace pour les fonctions implémentée en inline.

Exemples :
```
funct_a(int*x,int*y)
{ if(*x'=5) *y=*x+3; }
funct_a(&a,&b);

funct_b(int&x,int&y)
{ if(x'=5) y=x+3; }
funct_b(a,b);
```

Ce compilateur n.accepte pas comme paramètre les pointeurs sur les chaînes de caractères constantes (***const char***). Mais avec le paramètre **char**, il accepte les chaînes de caractères constantes comme paramètres d.appel.

Exemple :
```
void lcd_putc (char c)
{ # # .   }
lcd_putc("Hello Word") ;
```

4. Instructions de contrôle

if (expr) trait1; [else trait2;]	if (x==25) x=1; else x=x+1;
while (expr) traitement;	while(get_rtcc()!=0) putc('n');
do traitement while (expr);	do { putc(c=getc()); } while (c!=0);
for (expr1;expr2;expr3) trait;	for(i=1;i<=10;++i) printf("%u\r\n",i);
switch (expr) { case cexpr: trait1; case [default:traitn] ... }	switch (cmd) { case 0: printf("cmd 0"); break; case 1: printf("cmd 1"); break; default: printf("bad cmd"); break; }
return [expr];	return (5);
goto label;	goto label ;
label: trait;	label : i = 5 ;

5. Les constantes

123	Décimal
0123	Octal
0x123	Hexadécimal
0b010010	Binaire

'x'	Caractère
'\010'	Caractère octal
'\xA5'	Caractère hexadécimal
'\c'	Caractères spéciaux : \n ligne vide (\x0a) \r retour chariot (\x0d) \t tabulation
"abcdef"	Chaîne de caractères

6. Les opérateurs

+	Opérateur d'addition
&	Opérateur AND
^	Opérateur OU exclusif
|	Opérateur OU inclusif
?:	Opérateur conditionnel
- -	Décrément
/	Opérateur de division
==	Egalité
>	Opérateur supérieur
++	Incrément
!=	Inégalité
<	Opérateur inférieur
<<	Opérateur décalage à gauche
<=	Opérateur inférieur ou égal
&&	Opérateur AND logique
!	Opérateur NON logique
||	Opérateur OU logique
%	Opérateur modulo
*	Opérateur multiplication
~	Opérateur complément à un
>>	Opérateur décalage à droite
->	Pointeur
-	Opérateur soustraction
sizeof	Détermine le nombre d'octets d'un opérande

7. Les fonctions prédéfinies

7.1. Fonctions RS232

Le port série asynchrone est considéré comme entrée/sortie standard.
getc() putc() getch() gets() puts() printf() kbhit() set_uart_speed() getchar() putchar()
Syntaxe : set_uart_speed(Vitesse) ; // Vitesse : 100 - 115200

7.2. Entrées/sorties logiques

output_low(*pin****)*** ; Exemple : output(PIN_A2); // mettre RA 2 à 0
output_high(*pin****)*** ; Exemple : output(PIN_B0); // mettre RB0 à 1
output_bit(*pin, val****)*** : *val* prend les valeurs 0 ou 1.
output_X(*val*) : écrire *val* sur le port X ; X ∈ {A, B, C, D, E}

input(*pin****)*** : la valeur retournée est égale à 0 ou 1 (false ou true)
input_X() : lire l'état du port X.
port_b_pullups(*val****)*** : *val* prend la valeur true ou false
set_tris_X(*val****)*** : définie la direction du port X ; chaque bit de l'octet *val* correspond à un bit du port X (1 : entrée , 0 : sortie). Exemple : set_tris_b(0x0F) ;

7.3. Contrôle du processeur

sleep() : mettre le microcontrôleur en mode veille
disable_interrupts(*niveau****) enable_interrupts(****niveau****)*** inhibe ou valide les interruptions.
Exemples : disable_interrupts(GLOBAL) ; //inhibe toutes les interruptions, sans avoir besoin d'inhiber les interruptions spécifiques. Elle met à 0 le bit GIE (Global Interrupt Enable) du registre INTCON.
Enable_interrupts(GLOBAL) ; // les interruptions peuvent êtres autorisées. Elle met à 1 le bit GIE.
Enable_interrupts(int_ad) ; //valider l'interruptions de l'ADC, cette instruction n´a pas d'effet si le bit GIE n'est pas positionné.
ext_int_edge(*front****)*** : l'interruptions externe RB0, peut être active sur front montant ou descendant (L_TO_H ou H_TO_L).
Exemple : ext_int_edge(H_TO_L) ; // active sur front descendant
ext_int_edge(L_TO_H) ; // active sur front montant
goto_address(*adresse****)*** : faire un saut à l'*adresse* spécifiée

7.4. Manipulation des bits et des octets

shift_right(*adresse,octets,val****) shift_left(****adresse,octets,val****)*** : faire un décalage à droite ou à gauche d'un seul bit.
adresse : adresse du premier octet
octets : espace de travail en d'octets
val : le bit d'entré (*val* prend la valeur 0 ou 1)
Exemple : int buffer[3] ; // tempon de 3 octets
Shift_left(buffer,3,1) // décaler vers la gauche 3 octets d'un seul bit
rotate_right(*adresse,octets****) rotate_left(****adresse,octets****)*** : faire la rotation à droite ou à gauche d'un nombre fini d'octets.
adresse : adresse du premier octet
octets : nombre d'octets
Exemple : int16 x ;
rotate_right(&x,2) ;
bit_clear(*var,bit****) bit_set(****var,bit*) : mettre à un 0 ou à 1 un *bit* de la variable *var* ; *var* peut être de 8 ou 16 ou 32 ; *bit* est compris entre 0 .. 7 ou 0 .. 15 ou 0 .. 31.
Exemple : int x ; x = 5 ;
bit_set(x, 3) ; // x = 13
bit_set(*5, 2) ; //mettre le bit 2 du port A à 1 ;
//(5 est l'adresse du port A).
bit_test(*var,bit****)*** : renvoie l'état d'un *bit* de la variable *var*.
Exemple : if(bit_test(*5, 1))
printf("RA 1 = 1") ;

swap(*octet)* : permute les quartets haut et bas d'un octet.
Exemple : int x ; x = 0x23 ;
swap(x) ; // x = 0x32
make8(*var, offset)* : extrait un octet à partir d'une variable de 16 ou 32 bits
offset prend les valeurs 0, 1, 2 ou 3.
Exemple : int32 x ; int y ; x = 0x24FA ;
y = make8(x, 1) ; // y = 0x24
make16(*var1, var1_)* : construire une variable de 16 bits à partir de deux octets.
Exemple : int x0 ; int x1 ; long y ;
y = make16(x1, x0) ;
make32(*va1, var2, var3, var4)* : construire une variable de 32 bits à partir des variables de 8 et de 16 bits.
Exemple : int x ; long y ; int32 z ;
x = 0x12 ; y = 0xFA43 ;
z = make32(x, y) ; // z = 0x0012FA43
z = make32(y, x, x) ; // z = 0xFA431212

7.5. Standard C Math

abs(*x*)	Renvoie la valeur absolue
acos(*x*) asin(*x*) atan(*x*) atan2(*x, y*)	Arc cos, Arc sin Arc tan ou Arc tan de x/y
ceil(*x*)	Arrondi par excès à l'entier le plus proche
cos(*x*) sin(*x*) tan(*x*)	Fonctions trigonométriques
exp(*x*)	Fonction exponentielle
floor(*x*)	Arrondi par défaut à l'entier le plus proche
sinh(*x*) cosh(*x*) tanh(*x*)	Fonctions hyperboliques
log(*x*) log10(*x*)	Logarithme népérien ou décimal
pow(*x, y*) sqrt(*x*)	Fonction puissance, racine carré
fmod(*x, y*)	Renvoie la partie entière d'une fraction
modf(*Real, &integral*)	Renvoie la partie fractionnaire d'un réel, la partie entière dans le deuxième argument
div(*num, denum*) ldiv(*num, denum*)	Renvoie la partie entière et le reste du quotient

8. Manipulation des caractères

i8val=***atoi(****string)*	convertie une chaîne de caractères en entier de 8 bits
i16val=***atol(****string)*	convertie une chaîne de caractères en entier de 16 bits
i32val=***atoi32(****string)*	convertie une chaîne de caractères en entier de 32bits
fval=***atof(****string)*	convertie une chaîne de caractères en un réel
cval=***tolower(****cdata)* cval= ***toupper(****cdata)*	renvoie la minuscule du caractère *cdata* renvoie la majuscule du caractère *cdata*
boolval=***isalnum(****x)*	teste si *x* ∈ {'0' .. '9', 'A' .. 'Z', 'a' .. 'z'}
boolval=***isalpha(****x)*	teste si *x* ∈ {'A' .. 'Z', 'a' .. 'z'}
boolval=***isdigit(****x)*	teste si *x* ∈ {'0' .. '9'}
boolval=***islower(****x)*	teste si *x* ∈ {'a' .. 'z'}
boolval=***isupper(****x)*	teste si *x* ∈ {'A' .. 'Z'}
boolval=***isxdigit(****x)*	teste si *x* ∈ {'0' .. '9', 'A' .. 'F', 'a' .. 'f'}
ival=***strlen(****string)*	renvoie le nombre de caractères dans une chaîne

ptr=***strcpy(****str1, str2)*	copie la chaîne de caractères *str2* dans *str1*
ptr=***strncpy(****str1,str2,n)*	copie les *n* premiers caractères de *str2* dans *str1*
ival=***strcmp(****str1,str2)*	compare les deux chaînes *str1* et *str2*
ival=***strncmp(****str1,str2,n)*	compare les *n* premiers caractères de deux chaînes
ptr=***strcat(****str1,str2)*	concaténer *str1* et *str2* , le résultat dans *str1*
ptr=***strncat(****str1,str2,n)*	ajoute *n* caractères de str2 à str1
ptr=***strstr(****str1,str2)*	cherche *str2* dans *str1*, et renvoie un pointeur sur *str2* dans *str1*
ptr=***strchr(****str1,c)*	cherche dan *str1* la première position du caractère *c*
ptr=***strrchr(****str1,c)*	opère de même que ***strchr,*** mais en partant de la fin de *str1*
ptr=***strpbrk(****str1,str2)*	cherche le premier caractère dans *str1* qui se trouve aussi dans *str2*
strlwr(*string)*	convertie une chaîne de caractères en minuscule
sprintf(*string,format,val)*	identique à printf() sauf qu'elle renvoie le résultat dans *string*

Delays

delay_us(*val)* ***delay_ms(****val)* ; attente en miro ou milliseconde.
delay_cycles(*val)* : attente de *val* cycle machine (Tc = 4 * Tosc)

9. Conversion analogique numérique

Read_adc(*mode)*
mode : ADC_START_AND_READ ; paramètre par défaut
ADC_START_ONLY
ADC_READ_ONLY
set_adc_channel(*canal)* : définie la broche d'entrée de l'ADC
canal = 0 : Entrée RA0) ; *canal* = 1 : entrée AN1,… , *canal* = 7 : entrée RE2
setup_adc(*mode)* : définie l'horloge de l'ADC
mode : ADC_OFF : met l'ADC hors tension
ADC_CLOCK_INTERNAL : Pic sans quartz (RC interne)
ADC_CLOCK_DIV_2 : F_{oscMax} = 1,25Mhz
ADC_CLOCK_DIV_8 : F_{oscMax} = 5Mhz
ADC_CLOCK_DIV_32 : F_{oscMax} = 20Mhz
setup_adc_ports(*val)*
val : ALL_ANALOG : RA0, RA1,RA2,RA3,RA5, RE0,RE1,RE2 entrées analogiques
NO_ANALOGS : toutes les broches du port A et E sont des entrées numériques
A_ANALOG : RA0..RA5 sont des entrées analogiques et RE0..RE2 sont des entrées numériques.
Pour les autres combinaisons voir le ficher device.h (exp :16F877.h)

10. Les timers

get_timerX() : renvoie la valeur courante du timerX. Cette valeur est codée sur 8 bits pour le timer0 et timer2, sur 16 bits pour le timer1.
set_timerX(*val)* : initialise le timerX.

11. Timer0 ou RTCC

Les fonctions de configurations du **TIMER0** (appelé aussi **RTCC**)
set_rtcc(*val)* ou ***set_timer0(****val)* : charge le registre *TMR0* par la valeur *val*.
setup_counters(*rtcc_state, ps_state)* : configure le registre OPTION_REG, équivalente à la fonction ***setup_timer_0(****mode)*.
- *rtcc_state* : RTCC_INTERNAL, RTCC_EXT_L_TO_H ou RTCC_EXT_H_TO_L

- *ps_state* : RTCC_DIV_YYY ; YYY∈ {1, 2, 4, 8, 16, 32, 64, 128, 256}
- *mode* : peut prendre les valeurs ci-dessus, séparées par le symbole «|».
Exemple : setup_timer_0(RTCC_INTERNAL|RTCC_DIV_32) ;

12. Timer1

Setup_timer_1(*mode)* : configuration du timer 1
mode : T1_DISABLED : timer1 bloqué
T1_INTERNAL : horloge interne (fosc/4)
T1_CLOCK_OUT : valide l'oscillateur (quartz branché entre RC0 et RC1)
T1_EXTERNAL : horloge externe (entrée RC0)
T1_EXTERNAL_SYNC : horloge externe synchronisée sur Fosc (entrée RC0)
T1_DIV_BY_1, T1_DIV_BY_2, T1_DIV_BY_4, T1_DIV_BY_8 : pré diviseur
On peut regrouper ses constantes en utilisant l'opérateur « | »
Exemple : setup_timer_1(T1_INTERNAL|T1_DIV_BY_4) ;

13. Timer2

Setup_timer_2(*mode, période, posdivi)* : configuration du timer2
mode : T2_DISABLED : timer2 hors tension
T2_DIV_BY_1, T2_DIV_BY_4, T2_DIV_BY_16 : valeur du pré diviseur
Période : charger le registre PR2 par une valeur comprise entre 0 et 255
Posdiv : post-diviseur (valeur comprise entre 1et 16)
Exemple : setup_timer_2(T2_DIV_BY_4, 145, 12) ;

14. Module Capture Compare PWM (CCP)

Setup_ccpX(*mode)* : configuration du module CCP (X = 1 ou 2)
mode : CCP_OFF : module CCP hors tension
Module CCP en mode capture :
CCP_CAPTURE_FE : Capture sur front montant
CCP_CAPTURE_RE : Capture sur front descendant
CCP_CAPTURE_DIV_4 : Capture après 4 impulsions
CCP_CAPTURE_DIV_16 : Capture après 16 impulsions
Module CCP en mode compare :
CCP_COMPARE_SET_ON_MATCH : mettre la sortie à 1 en cas d'égalité
CCP_COMPARE_CLR_ON_MATCH : mettre la sortie à 0 en cas d'égalité
CCP_COMPARE_INT : génère une interruption en cas d'égalité.
CCP_COMPARE_RESET_TIMER : remet timer1 à zéro et lance la conversion A/D.
Module CCP en mode PWM :
CCP_PWM : configure le module en mode PWM
Set_pwmX_DUTY(*val)* : charge le rapport cyclique, *val* est un entier codé sur 8 bits ou 16 bits (*val* < 1024).
En mode capture le contenu du timer1 est copié dans la variable CCP_X (CCP_X_LOW et CCP_X_HIGH)
En mode compare, l'événement prend action lorsque le timer1 devient égal à CCP_X.
La fréquence du signal PWM est fixée par le timer2, sans tenir compte du post-diviseur.
Th = 4 * Tosc * Prédiviseur*(PR2 + 1)

AFFICHEUR LCD

1. Description

Un afficheur LCD est caractérisé par le nombre de lignes, ainsi que le nombre de caractères. 2 lignes 16 caractères par exemple. Chaque caractère est inscrit dans une matrice de 5 colonnes de 8 points. La plupart des caractères n'utilisent que les 7 rangées supérieures de la matrice; la rangé inférieure est prévue pour la visualisation du curseur.

2. Possibilités de l'afficheur

L'afficheur est en mesure de visualiser 192 caractères :
- de $00 à $ 07 : 8 caractères définissables par l'utilisateur
- de $20 à $7F : 96 caractères ASCII (majuscules, minuscules, chiffres, signes)
- de $A0 à $DF: 64 caractères japonais (alphabet kana)
- de $E0 à $FF : 32 caractères spéciaux: accent, lettres grecques, ...

3. Brochage de l´LCD

N° DE BROCHE	SIGNAL	NIVEAU
1	VSS	Masse
2	VDD	+ 5 V
3	V_{LC}	< 2,5 V
4	RS	0 = Instruction 1 = caractère.
5	R/W	0 = écriture 1 = lecture
6	E	Front descendant
7	D0	Logique positive
8	D1	Logique positive
9	D2	Logique positive
10	D3	Logique positive
11	D4	Logique positive
12	D5	Logique positive
13	D6	Logique positive
14	D7	Logique positive

4. Description des différentes broches

D7 à D0 : Bus de données bidirectionnel 3 états (Haute impédance lorsque E=0)
E : Entrée de validation (ENABLE); elle est active sur front descendant. Il est important ici de tenir compte des 2 seuils, durée de commutation importantes en pratique; lorsque RS et R/W ont atteint un niveau stable, il doit se passer un intervalle de 140 ns minimum avant que la ligne "E" ne passe au niveau haut. Cette ligne doit ensuite, être maintenue à ce niveau pendant 450 ns au moins et les données doivent rester stables sur le bus de données jusqu'au début du flanc descendant de ce signal. Lorsque E=0, les entrées du bus de l'afficheur sont à l'état haute impédance.
(*front montant Latch de l'état RS et R/W, front descendant latch de la donnée*)
R/W : Lecture ou écriture. (READ/WRITE)
Lorsque R/W est au niveau bas, l'afficheur est en mode "écriture", et lorsque R/W est au niveau haut, l'afficheur est en mode "lecture".

RS : Sélection du registre. (REGISTER SELECT)
Grâce à cette broche, l'afficheur est capable de faire la différence entre une commande et une donnée. Un niveau bas indique une commande et un niveau haut indique une donnée.
VLC : Cette tension permet le réglage du contraste de l'afficheur. C'est une tension négative et tournant autour de -1,5 V. (selon l'angle de visualisation)

5. Fonctionnement

5.1. Apparition des caractères sur l'afficheur

Apres avoir défini le sens de déplacement, les caractères apparaissent au dessus du curseur (qu'il soit visualisé ou non).

Adresse	gauche droite
haut	$00 ..... $0F ……
bas	$40 .....$4F ……

L'adresse 00 correspond à la ligne du haut à gauche, 0F à droite (afficheur 16 caractères).
L'adresse 40 correspond à la ligne du bas à gauche, 4F à droite (afficheur 16 caractères).

5.2. Principe de fonctionnement

Le principe de fonctionnement est simple, pour visualiser un caractère, il suffit de le présenter sur le bus de donnée (codé en ASCII), de mettre RS au niveau haut (caractère), R/W au niveau bas (écriture), et de provoquer un front descendant sur l'entrée de validation de l'afficheur (E).

5.3. Tableau des différentes commandes de l'afficheur

COMMANDE	RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0	DESCRIPTION
EFFACER L'AFFICHAGE	0	0	0	0	0	0	0	0	0	1	Efface l'ensemble de la mémoire de données. Met le curseur en position Home, à l'adresse 00.
CURSEUR EN POSITION HOME	0	0	0	0	0	0	0	0	1	*	Met le curseur en position Home.
MANIERE DE VISUALISER LES CARACTERES	0	0	0	0	0	0	0	1	ID	S	Détermine le sens de déplacement du curseur après apparition d'un caractère **(ID)** et le déplacement collectif d'une posi-tion de l'ensemble de l'affichage **(S)**.
MARCHE/ARRET DE L'AFFICHAGE DU CURSEUR	0	0	0	0	0	0	1	D	C	B	Met l'affichage en ou hors fonction **(D)**. Met le curseur en ou hors fonction **(C)**. Fait clignoter le caractère situé au-dessus du curseur **(B)**.
DECALAGE	0	0	0	0	0	1	S/C	R/L	*	*	Déplace le curseur ou l'ensemble de l'affichage.

FONCTION SET	0	0	0	0	1	DL	N	F	*	*	Indique la largeur du bus de données (**DL**). Nombre de lignes (**N**). Matrice (**F**).
ADRESSE DU GENERATEUR DE CARACTERES CGRAM	0	0	0	1	Caractère A5 .. A3			Rangée A2 .. A0			Définit l'adresse de la mémoire du générateur de caractères.
ADRESSE DE LA DDRAM	0	0	1	Adresse (A6..A0)							Définit l'adresse de la mémoire de données
INDICATEUR *BUSY*, LECTURE D'ADRESSE	0	1	BF	Adresse (A6 .. A0) position du curseur							Lit l'indicateur Busy (**BF**) et l'adresse de la position du curseur.
ECRITURE DE DONNEES dans CG ou DDRAM	1	0	Données (D7..D0)								Ecrit des données respectiv-ement dans la mémoire de données ou dans le générateur de caractères.
LECTURE DE DONNEES du CG ou DDRAM	1	1	Données (D7..D0)								Lit les données de la mémoire de données ou du générateur de caractères.

5.4. Description des différentes commandes.

Commande	0	1
ID	Déplacement vers la gauche	Déplacement vers la droite
S	L'affichage ne bouge pas	L'affichage est décalé
D	Affichage OFF	Affichage ON
C	Absence du curseur	Visualisation du curseur
B	Absence de clignotement du caractère	Clignotement du caractère
S/C	Déplacement du curseur	Déplacement de l'affichage
R/L	Décalage vers la gauche	Décalage vers la droite
DL	4 bits	8 bits
N	Ligne du haut	2 lignes validées

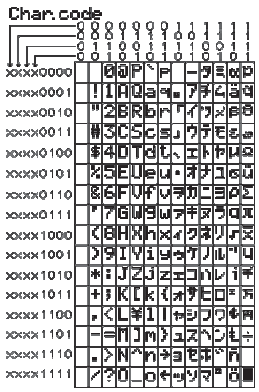
Le bit noté **F** permet de définir la matrice des caractères suivant le tableau ci dessous.

N	F	Nombre DE LIGNE	MATRICE
0	0	1	5 * 7
0	1	1	5 * 10
1	*	2	5 * 7

5.5. Temps d'exécution des instructions

Instruction	Temps d'exécution	Remarque
Effacement de l'affichage	1.64 ms	/
Curseur au début de la ligne	40µs à 1.64 ms	Dépends de la "distance" du curseur par rapport à l'origine
Lecture de busy flag	1 Enable cycle	2 cycles en mode 4 bits
Lecture/Ecriture dans le générateur de caractère	120 µs	/
Toutes les autres instructions	40 µs	/

5.6. Tableau de caractères



6. Initialisation de l'afficheur

Après la mise sous tension de l'afficheur, la ligne supérieur devrait être totalement sombre, celle du bas complètement claire. Si tel n'était pas le cas, il faudra augmenter (négativement) la valeur de la tension VLC. Il est temps maintenant d'envoyer les premières commandes d'initialisation.

La première commande à envoyer est la commande permettant de définir le mode de dialogue avec l'afficheur **(DL),** et le nombre de lignes sélectionnées **(N)**. Pour cela, on met RS et R/W au niveau bas et on présente sur le bus de données le code correspondant au choix de l'utilisateur (cette commande efface la ligne supérieure). Après un front descendant de l'entrée E, l'information sera prise en compte. A partir de maintenant, l'ordre des commandes n'a plus d'importance.

Envoyons par exemple la commande $01 (effacement de l'afficheur), puis la commande $0E, soit D=1, C=1, B=0, pour mettre le curseur en fonction.

Ensuite, il faut définir le sens de déplacement du curseur, pour cela, envoyons la commande $06, soit ID=1 et S=0.

Ceci étant fait, on peut maintenant commencer à envoyer les premiers caractères à afficher. Il faut faire passer la ligne RS au niveau haut et envoyer sur le bus de données les codes des caractères à afficher. Après chaque front descendant de l'entrée E, le caractère sera affiché au dessus de la position du curseur.

ATTENTION : *Avant toute procédure d'écriture, il est impératif de tester l'indicateur Busy. Pour cela il faut aller lire l'adresse de la position du curseur en mettant **RS** au niveau bas, **R/W** au niveau haut, et effectuer un ET logique avec la valeur $80 afin de permettre le test du bit **BF** de l'indicateur **Busy**. Lorsque **BF=0**, il est possible d'envoyer le caractère suivant à afficher.*

Fichier REG16F.h

```
#byte   TMR0    = 0x01
#byte   STATUS  = 0x03
#byte   PORTA   = 0x05
#byte   PORTB   = 0x06
#byte   PORTC   = 0x07
#byte   PORTD   = 0x08
#byte   PORTE   = 0x09
#byte   INTCON  = 0x0B
#byte   PIR1    = 0x0C
#byte   PIR2    = 0x0D
#byte   TMR1L   = 0x0E
#byte   TMR1H   = 0x0F
#byte   T1CON   = 0x10
#byte   TMR2    = 0x11
#byte   T2CON   = 0x12
#byte   SSPBUF  = 0x13
#byte   SSPCON  = 0x14
#byte   CCPR1L  = 0x15
#byte   CCPR1H  = 0x16
#byte   CCP1CON = 0x17
#byte   RCSTA   = 0x18
#byte   TXREG   = 0x19
#byte   RCREG   = 0x1A
#byte   CCPR2L  = 0x1B
#byte   CCPR2H  = 0x1C
#byte   CCP2CON = 0x1D
#byte   ADRESH  = 0x1E
#byte   ADCON0  = 0x1F

#byte   OPTION_REG   = 0x81
#byte   TRISA   = 0x85
#byte   TRISB   = 0x86
#byte   TRISC   = 0x87
#byte   TRISD   = 0x88
#byte   TRISE   = 0x89
#byte   PIE1    = 0x8C
#byte   PIE2    = 0x8D
#byte   PCON    = 0x8E
#byte   SSPCON2 = 0x91
#byte   PR2     = 0x92
#byte   SSPADD  = 0x93
#byte   SSPSTAT = 0x94
#byte   TXSTA   = 0x98
#byte   SPBRG   = 0x99
#byte   ADRESL  = 0x9E
#byte   ADCON1  = 0x9F
#byte   EEDATA  = 0x10C
#byte   EEADR   = 0x10D
#byte   EEDATH  = 0x10E
#byte   EEADRH  = 0x10F
#byte   EECON1  = 0x18C
#byte   EECON2  = 0x18D
#bit    GIE     = INTCON.7
#bit    PEIE    = INTCON.6
#bit    T0IE    = INTCON.5
#bit    INTE    = INTCON.4
#bit    RBIE    = INTCON.3
#bit    T0IF    = INTCON.2
#bit    INTF    = INTCON.1
#bit    RBIF    = INTCON.0
#bit    PSPIE   = PIE1.7
#bit    ADIE    = PIE1.6
#bit    RCIE    = PIE1.5
#bit    TXIE    = PIE1.4
#bit    SSPIE   = PIE1.3
#bit    CCP1IE  = PIE1.2
#bit    TMR2IE  = PIE1.1
#bit    TMR1IE  = PIE1.0
#bit    EEIE    = PIE2.4
#bit    BCLIE   = PIE2.3
#bit    CCP2IE  = PIE2.0
#bit    PSPIF   = PIR1.7
#bit    ADIF    = PIR1.6
#bit    RCIF    = PIR1.5
#bit    TXIF    = PIR1.4
#bit    SSPIF   = PIR1.3
#bit    CCP1IF  = PIR1.2
#bit    TMR2IF  = PIR1.1
#bit    TMR1IF  = PIR1.0
#bit    EEIF    = PIR2.4
#bit    BCLIF   = PIR2.3
#bit    CCP2IF  = PIR2.0
#bit    TMR1ON  = T1CON.0
#bit    TMR2ON  = T2CON.2
#bit    SPEN    = RCSTA.7
#bit    RX9     = RCSTA.6
#bit    SREN    = RCSTA.5
#bit    CREN    = RCSTA.4
#bit    ADDEN   = RCSTA.3
#bit    RX9D    = RCSTA.0
#bit    ADCS1   = ADCON0.7
#bit    ADCS0   = ADCON0.6
#bit    GO      = ADCON0.2
#bit    ADON    = ADCON0.0
#bit    RA5     = PORTA.5
#bit    RA4     = PORTA.4
#bit    RA3     = PORTA.3
#bit    RA2     = PORTA.2
#bit    RA1     = PORTA.1
#bit    RA0     = PORTA.0
#bit    RB7     = PORTB.5
#bit    RB6     = PORTB.4
#bit    RB5     = PORTB.5
#bit    RB4     = PORTB.4
#bit    RB3     = PORTB.3
#bit    RB2     = PORTB.2
#bit    RB1     = PORTB.1
#bit    RB0     = PORTB.0
#bit    RC7     = PORTC.5
#bit    RC6     = PORTC.4
#bit    RC5     = PORTC.5
#bit    RC4     = PORTC.4
#bit    RC3     = PORTC.3
#bit    RC2     = PORTC.2
#bit    RC1     = PORTC.1
#bit    RC0     = PORTC.0
#bit    RD7     = PORTD.5
#bit    RD6     = PORTD.4
#bit    RD5     = PORTD.5
#bit    RD4     = PORTD.4
#bit    RD3     = PORTD.3
#bit    RD2     = PORTD.2
#bit    RD1     = PORTD.1
#bit    RD0     = PORTD.0
#bit    RE1     = PORTE.1
#bit    RE0     = PORTE.0
#bit    RE2     = PORTE.2
```


Driver lcd4bits.c

```
// Ce fichier est extrait un ficher lcd.c du compilateur PCW
struct lcd_pin_map {
           boolean unused;
           boolean rw;
           boolean enable;
           boolean rs;
           int     data : 4;
         } lcd;

#byte lcd = 6        // port B
#define lcd_type 2           // 0=5x7, 1=5x10, 2=2 lines
#define lcd_line_two 0x40    // LCD RAM address for the second line


byte CONST LCD_INIT_STRING[4] = {0x20 | (lcd_type << 2), 0xc, 1, 6};
                             // These bytes need to be sent to the LCD
                             // to start it up.
STRUCT lcd_pin_map const LCD_WRITE = {0,0,0,0,0};

void lcd_send_nibble( byte n ) {
      lcd.data = n;
      delay_cycles(10);
      lcd.enable = 1;
      delay_cycles(20);
      lcd.enable = 0;
}
void lcd_send_byte( byte address, byte n ) {
      lcd.rs = 0;
      lcd.rs = address;
      delay_cycles(10);
      lcd.rw = 0;
      delay_cycles(10);
      lcd.enable = 0;
      lcd_send_nibble(n >> 4);
      lcd_send_nibble(n & 0xf);
      delay_ms(2);
}
void lcd_init() {
    byte i;
    set_tris_b(LCD_WRITE);
    lcd.rs = 0;
    lcd.rw = 0;
    lcd.enable = 0;
    delay_ms(15);
    for(i=1;i<=3;++i) {
       lcd_send_nibble(3);
       delay_ms(5);
    }
    lcd_send_nibble(2);
    for(i=0;i<=3;++i)
       lcd_send_byte(0,LCD_INIT_STRING[i]);
}
```



```
void lcd_gotoxy( byte x, byte y) {
   byte address;

   if(y!=1)
     address=lcd_line_two;
   else
     address=0;
   address+=x-1;
   lcd_send_byte(0,0x80|address);
}

void lcd_putc( char c) {
   switch (c) {
     case '\f'   : lcd_send_byte(0,1);
                   delay_ms(2);
                                          break;
     case '\n'   : lcd_gotoxy(1,2);       break;
     case '\b'   : lcd_send_byte(0,0x10);  break;
     default     : lcd_send_byte(1,c);     break;
   }
}
void ShowCursor()
{
        lcd_send_byte(0,0x0E);
}
void MaskCursor()
{
        lcd_send_byte(0,0x0C);
}
char lcd_getc( byte x, byte y) {
   char value;

    lcd_gotoxy(x,y);
    lcd.rs=1;
    value = lcd_read_byte();
    lcd.rs=0;
    return(value);
}
```