TD 5 - Le C.A.N. du PIC24F

1 Utilisation de l'afficheur LCD de la carte Explorer16

La carte Explorer16 est fournie avec un afficheur LCD 2x16 classique, dit "semi-intelligent" : tout le hardware de contrôle est déporté sur celui-ci, et on communique simplement via un port parallèle. Microchip fournit une bibliothèque de fonctions, déclarées dans "lcd.h", qui permet d'afficher un caractère, d'effacer l'écran, etc.

- Récupérer depuis Moodle le fichier `td3_files.zip`, décompressez le dans votre dossier de travail, et charger le projet.
- Etudier le source, et vérifier le fonctionnement. Il manque cependant une fonction pour afficher toute une chaîne de caractère.
- Ajouter la définition de cette fonction `void LCDPutString(const char* s)` dans le fichier `lcd.c`, ajouter la déclaration dans lcd.h. Cette fonction doit parcourir la chaîne jusqu'au 0 terminal, et envoyer les caractères un par un, via la fonction `LCDPut()`.
- Vérifier le fonctionnement en ajoutant `LCDPutString( "Hello World" );` dans le programme.

2 Utilisation du C.A.N. du PIC24F

2.1 Présentation (voir doc. p.167)

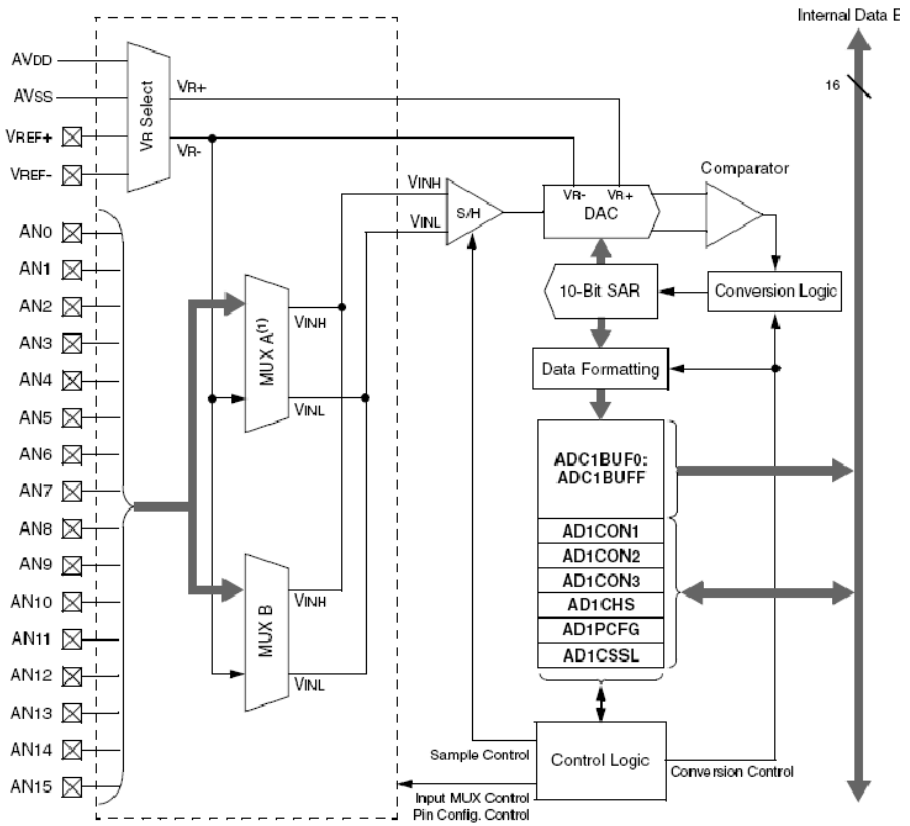
- Successive Approximation (SAR) conversion
- Conversion speeds of up to 500 ksps
- Up to 16 analog input pins
- External voltage reference input pins
- Automatic Channel Scan mode
- Selectable conversion trigger source
- 16-word conversion result buffer
- Selectable Buffer Fill modes
- Four result alignment options
- Operation during CPU Sleep and Idle modes

Ce convertisseur peut fonctionner de plusieurs façons. Il faudra d'abord le configurer, puis on pourra l'utiliser pour faire l'acquisition de tensions. On fonctionnera ici en **scrutation** : il faudra attendre la **fin de la conversion** avant de lire la valeur dans le registre résultat. Ceci se fera en testant le bit DONE du registre `AD1CON1`. On devra ensuite le remettre à 0. Le résultat de la conversion peut ensuite être lu dans l'un des 16 registres 16 bits `ADC1BUF0` à `ADC1BUFF` (voir page suivante).

Pour l'initialisation, vous définirez en fin du source une fonction `void ADC_Init();` , qui sera appelée au début du programme, et dans laquelle vous réaliserez la configuration du convertisseur.

Vous allez le mettre en oeuvre en simulant un capteur de température par le potentiomètre ajustable situé sur la carte (voir schéma de la carte).

Sur quelle entrée du convertisseur est amené le signal du pot. :_____
(voir doc carte Explorer 16, p.42)



2.2 Initialisation du convertisseur

Créer un nouveau projet, un nouveau fichier source, et y copier un squelette de programme. Récupérer dans le fichier `adc_lib.c` le code de la fonction d'initialisation, et copiez le dans votre source. A partir de la doc du PIC, compléter les valeurs.

2.3 Programme à réaliser

Vérifier tout d'abord le bon fonctionnement du convertisseur, en envoyant la valeur lue sur la barre de Dels :

```
int main()
{
   ADC_Init();
   TRISA = 0xff00;
   while(1)
   {
      while( AD1CON1bits.DONE == 0) // attente fin conversion
         ;
      AD1CON1bits.DONE = 0;   // réinit flag
      int res = ADC1BUFX;
      PORTA = res;
   }
}
```

Ensuite, réaliser le cahier des charges suivant :
Afficher HIGH si la température (la tension, ici) est supérieure à un seuil haut (défini par une constante symbolique), afficher LOW si la température est inférieure à un seuil bas, et OK si elle est entre les deux seuils. La difficulté vient du fait qu'il est inefficace d'envoyer la chaîne de caractère de façon continue :

```
     if( res > SEUIL_H )
          LCDPutString( "HIGH" );
     else
          if( res < SEUIL_B )
               LCDPutString( "LOW" );
          else
               LCDPutString( "OK" );
```

Ceci amène une instabilité de l'affichage. Il est préférable de définir une **machine d'états**, et de n'envoyer la chaîne de caractère vers l'afficheur **que** si l'on détecte un changement d'état.

```
while(1)
{
// 1 - Acquisition de la valeur via CAN
     ...

// 2 - positionner 'state' en fonction de la valeur
    if( valeur > ... )
        state = ...;
    else
        ...

// 3 - affichage uniquement si changement
     if( state != old_state )
     {
          old_state = state;
          switch( state )
          {
               case ETAT_0 : ... ; break;
               case ETAT_1:  ... ; break;
               ...
               default : // ERREUR !
          }
     }
}
```

Remarque : dans ce type de système, il est nécessaire d'ajouter un **hystérésis** afin d'éviter l'instabilité. Ajouter cette valeur dans les comparaisons.