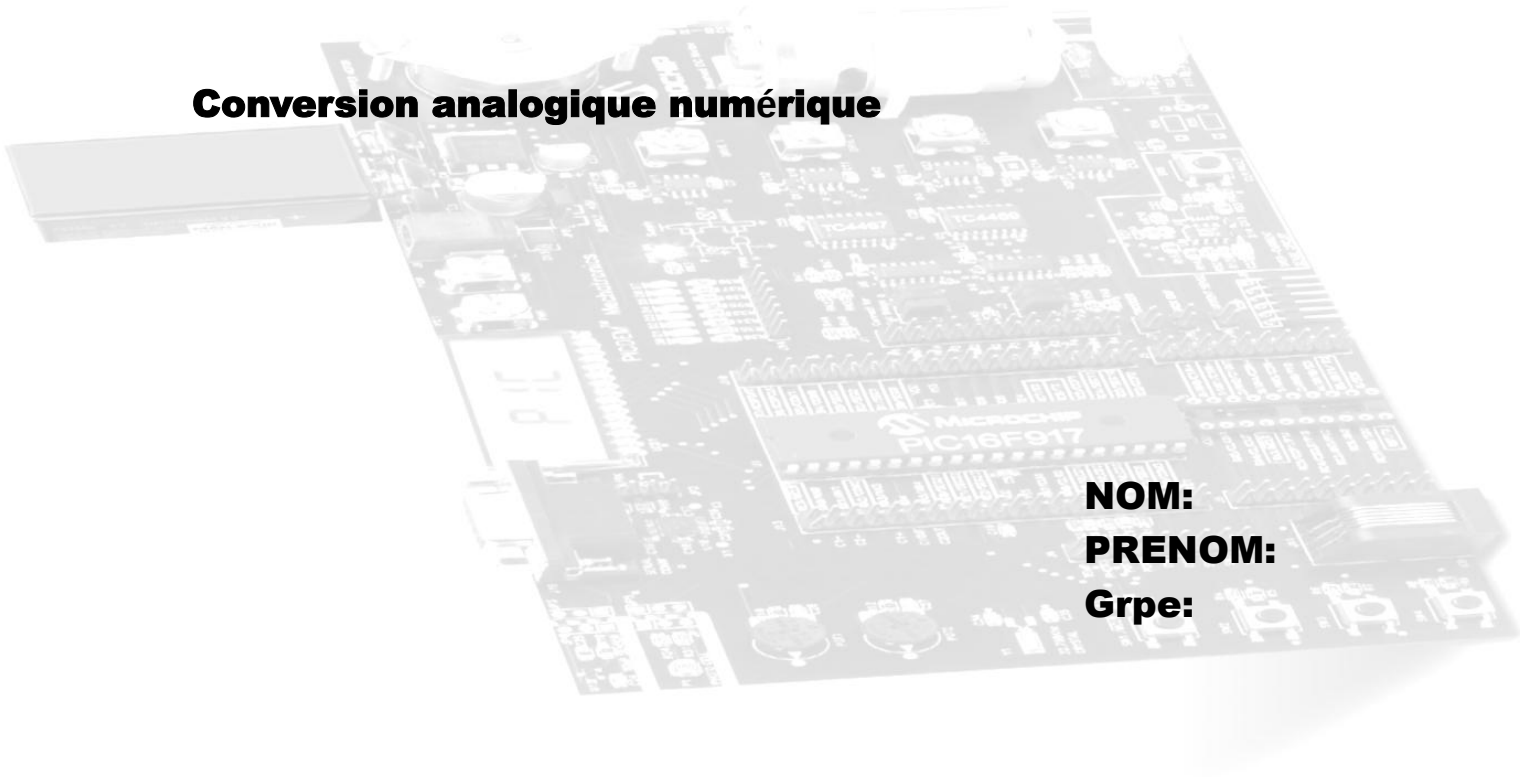
Cours de PIC



Conversion analogique numérique

NOM:

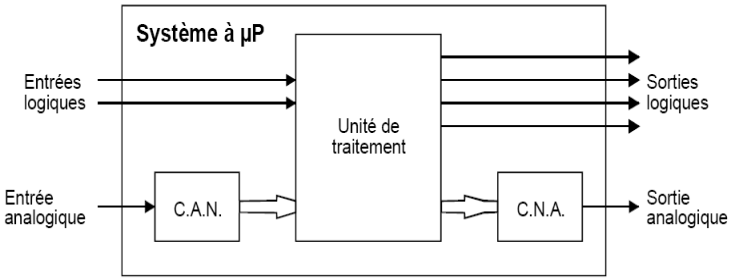
PRENOM:

Grpe:

CAN:Principes et caractéristiques

Objectif du convertisseur analogique numérique

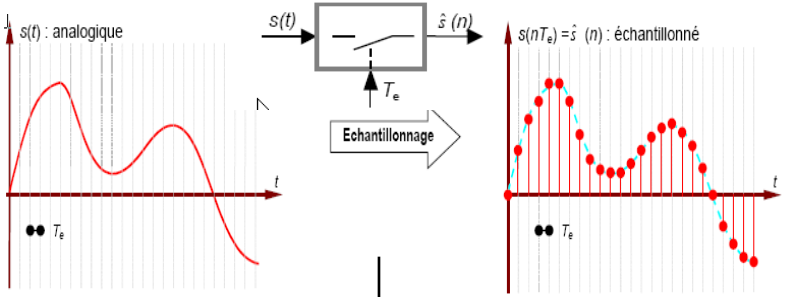
- convertir une tension analogique, comprise entre deux tension de référence Vref- et Vref+, en une valeur numérique N sur n bits
- Un système de commande comportant un microprocesseur peut se schématiser de la manière suivante

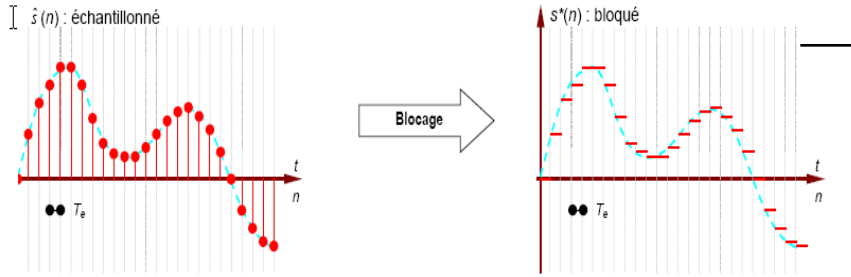


Un exemple

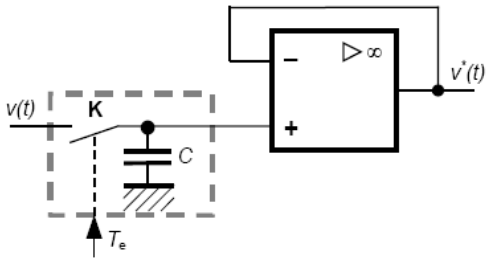
Les étapes de conversion

- L'échantillonnage
  - ✓ action de prélever la valeur du signal à intervalle fixe et répétitif
  - ✓ Fréquence des échantillons: Te
- Le blocage
  - ✓ Temps de conversion non nul
  - ✓ Maintien de la valeur pendant ce temps





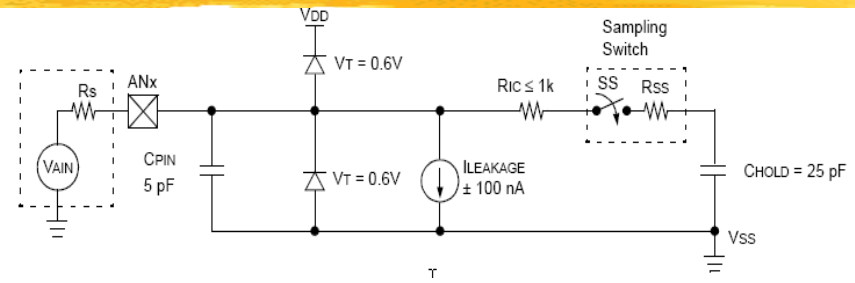
L'échantillonneur bloqueur:schéma de prinicpe



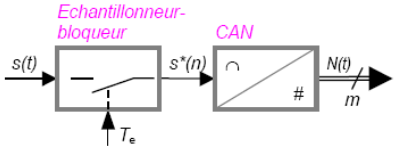
CAN:Principes et caractéristiques

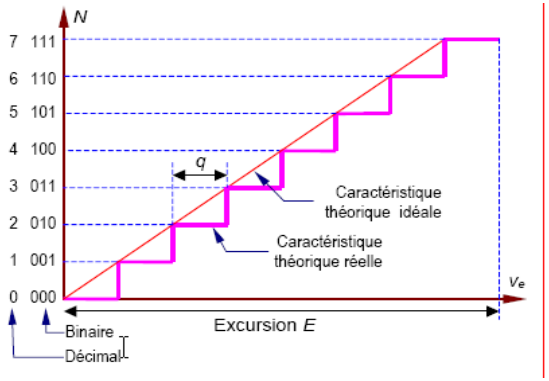
L'échantillonneur bloqueur du PIC

- ✓ Temps d'établissement de l'échantillonneur bloqueur = temps de chargement du condo

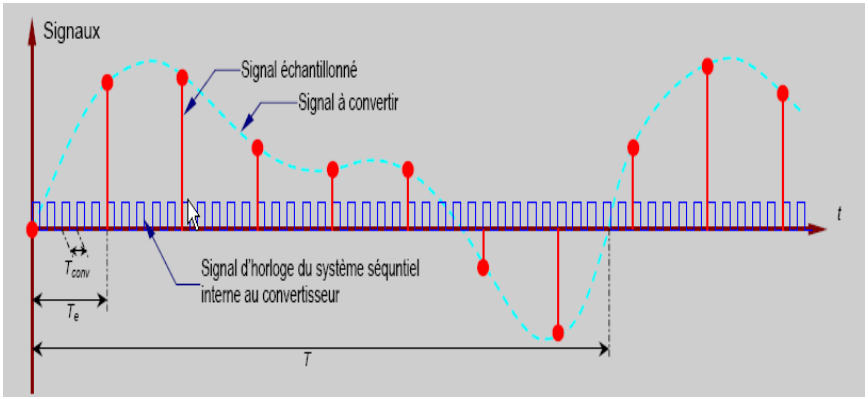


- La quantification
  - ✓ Réalisé par le convertisseur
  - ✓ Convertir une valeur analogique en une valeur numérique sur m bits
  - ✓ Introduit des distordions
    - Erreur de quantification, erreur de gains





Les différents signaux d'un CAN



➢Le signal à convertir (si périodique T)

➢Signal échantilloné (période d'échantillonnage Te)

➢L'horloge du convertisseur Tconv

✓Pour le PIC il faut 12 coup d'horloge pour réaliser une conversion => on peut en déduire la fréquence max d'échantillonnage

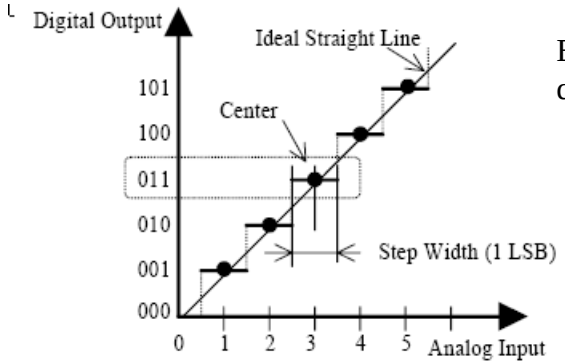
CAN:Principes et caractéristiques

Les caractéristiques principales d'un CAN

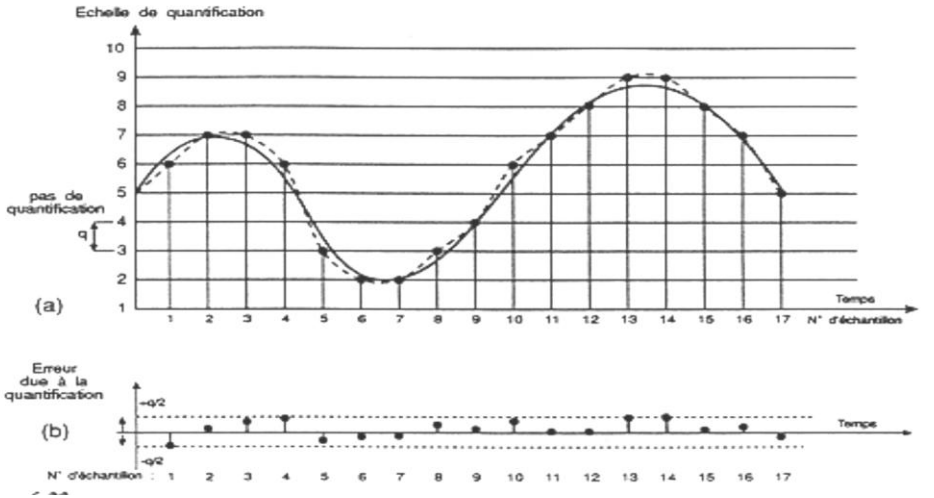
La résolution et quantum

- **Quantum ou LSB:** Incrément de tension d'entrée produisant une variation d'une unité du résultat de la conversion lsb.
  - q=(Vref+ - Vref-)/ 2^n pour un convertisseur n bits => 1lsb =2^n / (Vref+ - Vref-)
- Résolution: nombre de « points » =2^n

La fonction de transfert



Erreur de quantification



Le temps de conversion

La pleine échelle

- (*Full Scale Range* FSR) : c'est la tension maximum acceptable
- FSR= 2Nq pour un convertisseur linéaire.

Programmation du CAN

Les étapes à mettre en œuvre

- Initialisation des paramètres du convertisseur
- Sélection de la voie à convertir
- Attente de la stabilisation de l'échantillonneur bloqueur
  - ✓ (chargement du condensateur)
- Lancement d'une conversion
- Attente Fin de conversion
- Lecture donnée

ADCON0 REGISTER

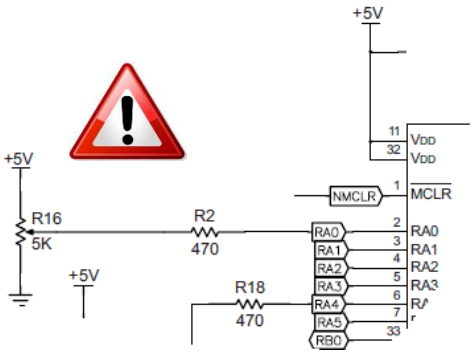
U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	CHS3	CHS2	CHS1	CHS0	GO/$\overline{DONE}$	ADON
bit 7							bit 0

ADCON1 REGISTER

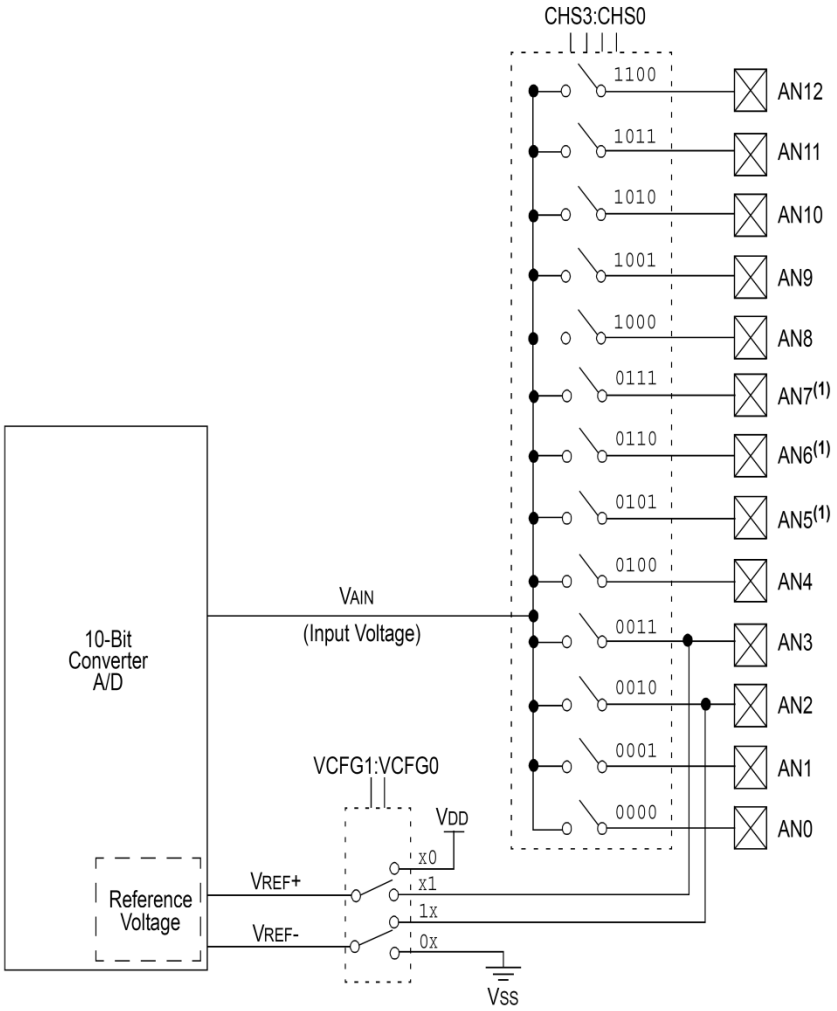
U-0	U-0	R/W-0	R/W-0	R/W-0(1)	R/W(1)	R/W(1)	R/W(1)
—	—	VCFG1	VCFG0	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

ADCON2 REGISTER

R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	—	ACQT2	ACQT1	ACQT0	ADCS2	ADCS1	ADCS0
bit 7							bit 0



Potentiomètre de la carte PICDEM2+ connecté à AN0

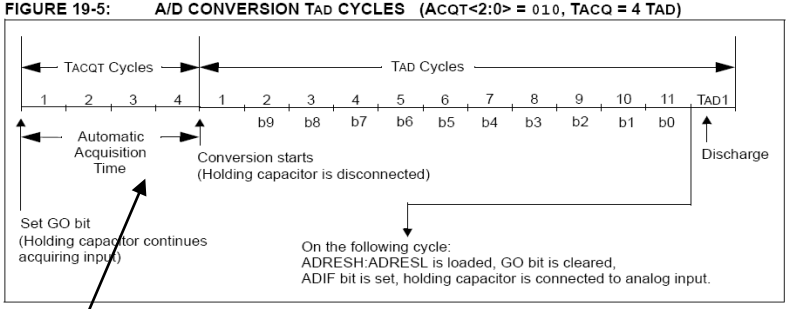


Programmation du CAN

Etape 1:Initialisation

- Validation du convertisseur: registre ADCON1 , bit **ADON=1**
- Choix des tensions de références
  - ✓ 4 possibilités
    - haute : Vdd ou externe par l'entrée Vref+ (AN3) (1.8V mini)
    - basse : Vss ou externe par l'entrée Vref- (AN2)
  - ✓ Nos choix: Vdd et Vss
  - ✓ Registre ADCON1 bits **VCFG1: VCFG0**
- Mode Analogique pour les broches d'entrées
  - ✓ Registre ADCON1, bits **PCFG3:PCFG0**
  - ✓ Voir tableau p224 doc du PIC18Fxxx
- Fréquence d'horloge de conversion
  - ✓ Tad=temps de conversion min d'un bit
  - ✓ Données constructeur: **Tadmin >= 0.7µs**
  - ✓ Tbitconv=kTosc > Tadmin => k=……
  - ✓ k le + petit compatible avec les bits ADCS2:ADCS0 de ADCON2
- Temps d'établissement de l'échantillonneur bloqueur Tacq
  - ✓ Temps mini entre fin d'acquisition et début nouvelle acquisition
  - ✓ Ou encore entre un changement de voie et le début nouvelle acquisition
  - ✓ Dépend de l'impédance de la source d'entrée
    - Estimation: 2.8 us
  - ✓ Temps obtenu par soft ou hardware
  - ✓ Temps hardware réglable grâce à ACQT2:ACQT0 registre ADCON2
    - Multiple de Tad ex : 010=> 2*Tad de tempo
  - ✓ Optionnel dans le cas d'une seul voie
    - le temps des instructions de votre programme assure en général un temps suffisant
- Cadrage à gauche ou à droite (explications pages + loin)
  - ✓ Bit **ADFM** du registre ADCON2

FIGURE 19-5: A/D CONVERSION TAD CYCLES (ACQT<2:0> = 010, TACQ = 4 TAD)



Exemple de fonction d'initialisation

```
Void adc_init (void)
{
ADCON0=0x09;
ADCON1=0x0C;
ADCON2=0x94;
TRISAbits.TRISA0 = 1;
}
```

Programmation du CAN

Etape 2: Sélection de la voie à acquérir

- registre ADCON1
  - ✓ **Bits CHS0 à CHS2** : Choix du canal à convertir voir doc

ADCON1 REGISTER

U-0	U-0	R/W-0	R/W-0	R/W-0(1)	R/W(1)	R/W(1)	R/W(1)
—	—	VCFG1	VCFG0	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

Etape 3: Temps d'acquisition de l'échantillonneur bloqueur

- Temps à respecter entre
  - ✓ un changement de voie et le début nouvelle acquisition
  - ✓ la fin d'acquisition et le début d'une nouvelle acquisition
- Mode automatique: ACQT2:ACQT0 registre ADCON2
  - ✓ tempo automatique après une demande de conversion
- Mode « manuel »: ACQT2:ACQT0=000
  - ✓ Prévoir le cas échéant une tempo software (delays ou autre)

ADCON2 REGISTER

R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	—	ACQT2	ACQT1	ACQT0	ADCS2	ADCS1	ADCS0
bit 7							bit 0

Etape 4: Lancement d'une conversion

- **Bits GO de ADCON0**
  - ✓ Mettre ce bit à 1 pour démarrer la conversion

ADCON0 REGISTER

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	CHS3	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	ADON
bit 7							bit 0

Etape 5: Attente fin de conversion

- Fin de conversion si bit GO retombe à 0
- Plusieurs méthodes pour tester le bit GO
- Méthode POLLING (scrutation)
  - ✓ 2 approches: bloquante ou non
- Méthode INTERRUPTION
  - ✓ Solution hardware
  - ✓ Le CAN interrompt le PIC

```
- Demande conversion GO=1
Faire
        - Lecture registre ADCON1
Tant que ((bit ADGO = 1) :   // Boucle tant que le bit GO/DONE =1
- Retourner ADRESH          // Retour résultat.
```

Méthode la+ simple (toléré!!!! – voir transp suivant)

Etape 6: lecture des données

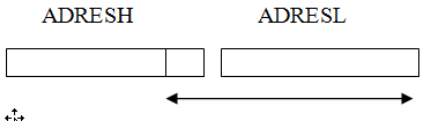
- Résultat dans les registres ADRESH et ADRESL
- Suivant la valeur du bit ADFM le résultat se trouve sur 10 bits cadrés à droite ou à gauche ce qui permet d'exploiter le résultat sur 8 ou 10 bits (voir schémas de ADRESH et ADRESL précédents).

Programmation du CAN

Comprendre le cadrage : ADFM

- Permet facilement de lire un résultat de 8 bits sur un convertisseur de 10 bits

Si ADFM = 1 Résultat sur 10 bits cadrés à droite. (B0 à B9)
B0 → B7 dans ADRESL (L).
B8 → B9 dans les deux bits de poids faibles de ADRESH (H).



Si ADFM = 0 Résultat sur 10 bits cadrés à gauche. (B0 à B9
B2 → B9 dans ADRESH (H).
B0 → B1 dans les deux bits de poids forts de ADRESL (L).

- Le dernier cas permet de récupérer le résultat de la conversion sur 8 bits (poids forts de la conversion) en lisant uniquement le registre ADRESH.
- L'initialisation du registre ADCON1 est faite une seule fois en début de programme

- Conversion sur 8 bits:
  - ✓ res=ADRESH; //avec unsigned **char** res; et ADCON2bits.ADFM=0

- Conversion sur 10 bits:
  - ✓ La variable res déclarée en: unsigned **int** res;

```
res=ADRESH; // recupère poids fort de res et mets sur poids faible de res
res=res<<8;  // poids faible devient poids fort
Res=res+ADRESL // complte avec le pids faible
```

Programmation du CAN

Comprendre le polling

- ***Technique 1***: boucle directe sur le bit GO => ***while(*ADCON0bits.GO *==1);***

```
// Directives de compilation
#include <p18cxxx.h>
// configuration
#pragma config OSC = HS   //Oscillateur externe
#pragma config WDT = OFF  //Watch Dog inhibé.
#pragma config LVP = OFF  //Low Voltage ICSP dévalidé.
#pragma config PBADEN = OFF
#include <stdio.h>
#include "xlcd200v.h"

#define GO ADCON0bits.GO


unsigned char res;
```

```
void main(){
ADCON0=0x09;
ADCON1=0x0C;
ADCON2=0x94;
TRISAbits.TRISA0 = 1;
OpenXLCD(OPEN_PICDEM_LCD);
stdout = _H_USER ;
clearXLCD();
for(;;){
        GO=1;
        while(GO==1);
        res=ADRESH;
        gotoXLCD(LCD_LINE_ONE);
        printf("%x                ",res);}
}
```

Toléré pour le CAN

- Inconvénient technique 1: cette méthode est BLOQUANTE pour le programme, par conséquence déconseillée sur le plan du principe de fonctionnement
  - ✓ Cependant dans le cas du convertisseur la durée de conversion est de 12Tad. En supposant Tad=1us, le programme resterait bloqué 12us. Ce temps reste petit et peut-être négligé dans bon nombre d'applications! L'étude se fait au cas par cas et dépend des contraintes temps –réels de l'application

- ***Technique 2:*** Polling en exploitant la boucle infini for(;;) + un IF pour le test de fin de conversion
  - ✓ Demande une approche plus structurée => **utilisation d'une machine d'état conseillée**

Programmation du CAN

Utilisation de la librairie C18 standart

- Fonctions de haut niveau disponible
  - ✓ NE PAS OUBLIER #include <adc.h>

TABLE 2-1: A/D CONVERTER FUNCTIONS

Function	Description
BusyADC	Is A/D converter currently performing a conversion?
CloseADC	Disable the A/D converter.
ConvertADC	Start an A/D conversion.
OpenADC	Configure the A/D convertor.
ReadADC	Read the results of an A/D conversion.
SetChanADC	Select A/D channel to be used.

- Initialiser le convertisseur
  - ✓ Complément d'info sur la doc

OpenADC(ADC_FOSC_8 &ADC_RIGHT_JUST &ADC_4_TAD ,
ADC_CH0 &ADC_INT_OFF &ADC_VREFPLUS_VDD &ADC_VREFMINUS_VSS, 14);

- Changer de voie **SetChanADC**(ADC_CH1);
  - ✓ Attention: vérifier que le temps d'établissement de l'échantillonneur est respecté avant la demande de conversion
  - ✓ Voir dans la doc « A/D acquisition time select » pour la fonction **OpenADC**

- Lancer la conversion **ConvertADC**()

- Attendre la fin de conversion
  - ✓ Technique 1: while(**BusyADC**());
  - ✓ Technique 2 **: if** (**BusyADC**()) { ……..};

- Lire la valeur Variable=**ReadADC**();

Programmation du CAN

- **Un exemple**

```
// Directives de compilation

#include <p18cxxx.h>

// configuration
#pragma config OSC = HS    //Oscillateur externe
#pragma config WDT = OFF   //Watch Dog inhib.
#pragma config LVP = OFF   //Low Voltage ICSP devalid.
#pragma config PBADEN = OFF //Si PBADEN=ON RB4:RB0 analogiques

#include <adc.h>
#include <stdio.h>
#include "xlcd200v.h"

#define GO ADCON0bits.GO
#define S2  PORTAbits.RA4
char cpt,b=0;
unsigned char res;

void main()
{
        OpenADC(ADC_FOSC_4 &
                        ADC_LEFT_JUST &
                        ADC_4_TAD,
                        ADC_CH0 &
                        ADC_INT_OFF &
                        ADC_VREFPLUS_VDD &
                        ADC_VREFMINUS_VSS,
                        14);
        TRISAbits.TRISA4 = 1; //RA4 en entrEe
        OpenXLCD(OPEN_PICDEM_LCD);
        stdout = _H_USER ;
        clearXLCD();
```

```
for(;;)
        {
        if(S2==0)
                {
                if(b==0)
                        {
                        ConvertADC(); // Lancement de la conversion
                        while(BusyADC()==1); // Attente fin de conversion
                        b=1;
                        res=ReadADC()>>8; // ou res=ReadADC() si res unsigned int
                        gotoXLCD(LCD_LINE_ONE);
                        printf("%x                    ",res);
                        }
                }
        else
                {
                if(b==1)
                        {
                        b=0;
                        }
                }
        }
```