



*Cet exercice, le dernier de la série, est un "petit" projet sur 2-3 séances. Nous vous proposons le choix entre deux projets. Bien entendu l'un des deux est plus simple que l'autre.*

## Projet n°1 :

*Implémenter un voltmètre utilisant l'afficheur LCD 2x16 déjà utilisé durant les TP précédents. Ce travail est relativement simple. Voici l'affichage que devra respecter l'afficheur pour une tension d'entrée sur l'ADC valant 3,44V . L'affichage sera mis à jour toutes les 20ms :*

t	e	n	s	i	o	n	:		3	.	4	4			V

## Projet n°2 :

*Le second projet s'adresse principalement aux futurs élèves de majeure SATE et aux autres élèves intrépides curieux de se creuser les méninges. Dans ce projet, il vous sera demandé d'implémenter un modeste analyseur de spectre. Ce projet est directement en lien avec les enseignements "Initiation à la programmation" (1AS1) et "Traitement Numérique du Signal" (1AS2). Votre travail consiste à afficher en pseudo temps réel le spectre d'un signal analogique sur un afficheur graphique ...*



**figure 1 :** exemple de spectre d'une sinusoïde affiché sur l'afficheur graphique GLCD

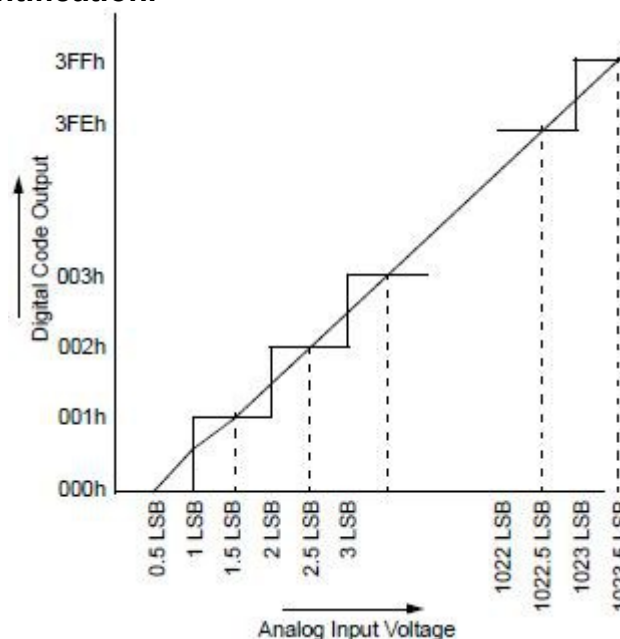
*Mais avant de commencer ces projets, il nous faut vous présenter l'ADC de notre micro-contrôleur ainsi que les registres associés. Quelques rappels sur la TFD serons, nous le pensons, pas un mal .....*

## 1. Comment fonctionne l'ADC de notre MCU ?

Le rôle d'un ADC est de convertir une tension (en Volts) en un mot numérique (sans unité). Les conversions sont souvent réalisées à intervalle de temps régulier, nous parlerons alors de période d'échantillonnage. Par exemple, supposons que nous disposons d'un convertisseur 10 bits ( $2^{10} = 1024$ ), le mot numérique après conversion sera donc forcément compris entre 0 et 1023 (0x000 et 0x3FF). Supposons que pour la partie analogique, la plage d'acquisition d'entrée soit comprise entre 0 et 5V. Après conversion :

- ➔ à 0V correspondra 0 (0x000)
- ➔ à 5V correspondra 1023 (0x3FF)
- ➔ et par exemple à 2,4V correspondra 491 (0x1EB).

La **figure 2** présente la fonction de transfert de notre ADC. En abscisse nous observons la plage d'acquisition analogique d'entrée et en ordonnée la valeur sur 10bits obtenue après conversion. Nous constatons que le principe même d'une **conversion analogique/numérique amène une perte d'information**. Cette "perte" est souvent appelée erreur de quantification.



**figure 2 :** Fonction de transfert analogique/numérique (cf. datasheet PIC18F4550)



*LSB (figure 2) dépend de la plage d'acquisition du signal d'entrée. Par exemple si cette plage est comprise entre  $V_{REF-} = V_{SS} = 0V$  et  $V_{REF+} = V_{DD} = +5V$ , pour un convertisseur 10bits,  $LSB = (V_{REF+} - V_{REF-})/1024 = (5-0)/1024 = 0,004883Volts$ . Cette plage est configurable et dépend bien évidemment de l'application visée.*

Nous venons d'effectuer quelques rapides rappels sur les ADCs, notamment celui rencontré en TP. Nous allons maintenant nous intéresser aux registres associés au convertisseur. Il y en a 5 : **ADCON0**, **ADCON1**, **ADCON2**, **ADRESH**, **ADRESL**

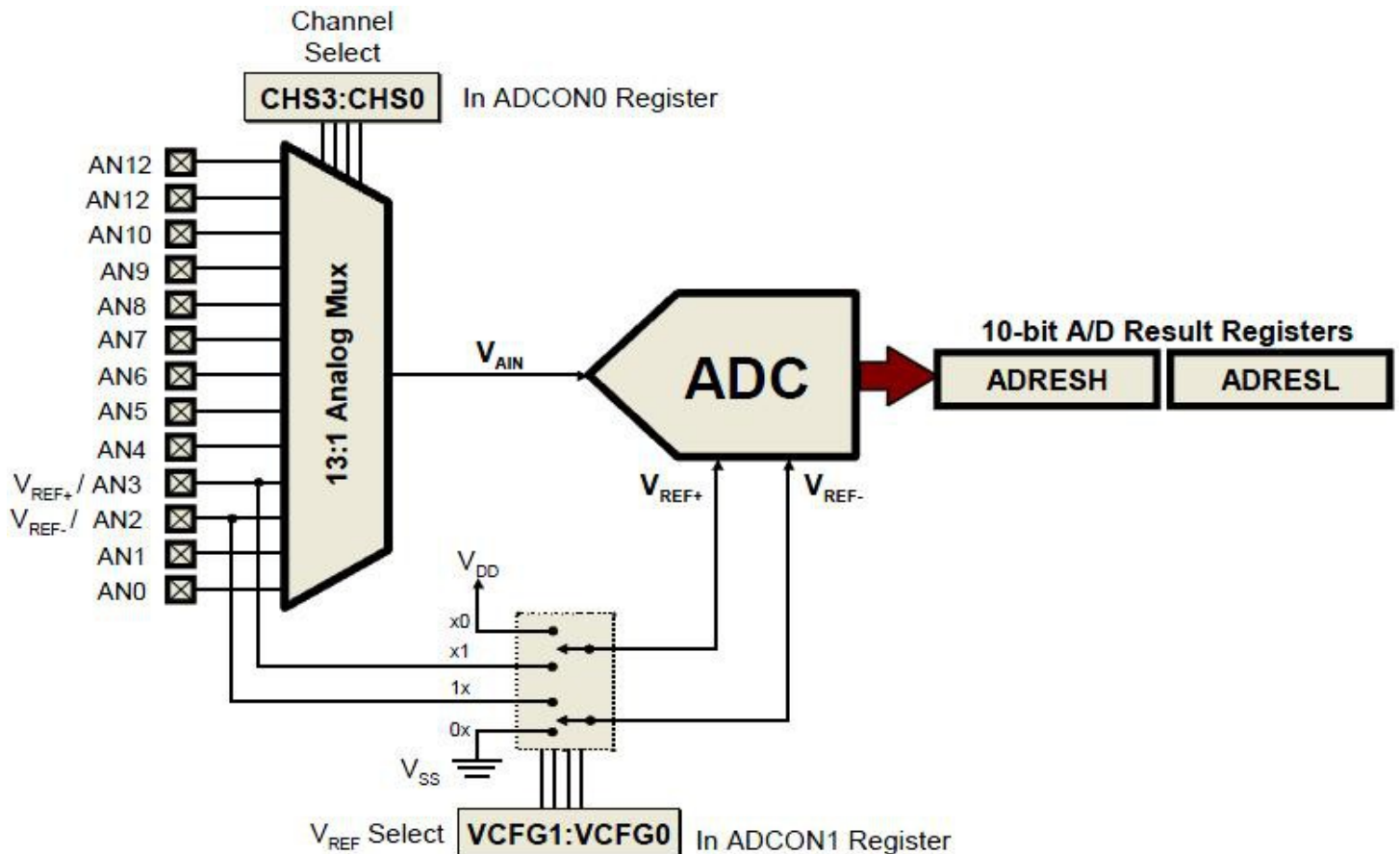


figure 3 : schéma fonctionnel de l'ADC

La **figure 3**, est un schéma simplifié de l'ADC. Nous constatons que nous pouvons utiliser jusqu'à 13 entrées analogiques. La sélection de la voie utilisée pour la conversion est faite grâce aux bits **CHS3:CHS0** du registre **ADCON0**. Le choix des tension de référence pour la plage d'acquisition analogique d'entrée est fait via les bits **VCFG1:VCFG0** du registre **ADCON1**. Nous pouvons également observer les deux registres 8bits **ADRESH** et **ADRESL** dans lesquels est "latchée" la valeur après conversion. Nous verrons dans la suite de la présentation la façon dont sont gérés ces deux registres.

	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/DONE	ADON
bit 7								bit 0
bit 7-6	<b>Unimplemented:</b> Read as '0'							
bit 5-2	<b>CHS3:CHS0:</b> Analog Channel Select bits							
	0000 = Channel 0 (AN0)							
	⋮							
	1100 = Channel 12 (AN12)							
bit 1	<b>GO/DONE:</b> A/D Conversion Status bit							
	<u>When ADON = 1:</u>							
	1 = A/D conversion in progress							
	0 = A/D Idle							

figure 4 : présentation partielle du registre **ADCON0**



Dans le registre **ADCON0**, le bit **GO/DONE** est particulièrement important. Ce bit permet de démarrer une conversion. Tant que la conversion n'est pas finie ce bit reste au niveau haut. Lorsque nous lancerons une conversion (**GO/DONE=1**), il nous faudra donc attendre le passage à '0' du bit **GO/DONE** avant de lire les registres **ADRESH** et **ADRESL**.

## 2. Présentation du registre ADCON2 ?

La plupart des registres de l'ADC sont relativement simples à configurer et/ou utiliser. Seul le registre ADCON2 demande un petit complément d'information. Rappelons qu'une **conversion analogique/numérique ne se fait "malheureusement" pas instantanément**, cela peut pendre du temps, plusieurs "µs" dans notre cas. Ce temps est configurable (via **ADCON2**) et dépend d'un très grand nombre de paramètres liés à l'application. Par exemple les spécifications électriques notifiées par Microchip, la température ambiante, la plage d'acquisition d'entrée ... etc

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

- bit 7      **ADFM**: A/D Result Format Select bit  
             1 = Right justified  
             0 = Left justified
- bit 6      **Unimplemented**: Read as '0'
- bit 5-3    **ACQT2:ACQT0**: A/D Acquisition Time Select bits  
             111 = 20 TAD  
             110 = 16 TAD  
             101 = 12 TAD  
             100 = 8 TAD  
             011 = 6 TAD  
             010 = 4 TAD  
             001 = 2 TAD  
             000 = 0 TAD
- bit 2-0    **ADCS2:ADCS0**: A/D Conversion Clock Select bits  
             111 = FRC (clock derived from A/D RC oscillator)  
             110 = FOSC/64  
             101 = FOSC/16  
             100 = FOSC/4  
             011 = FRC (clock derived from A/D RC oscillator)  
             010 = FOSC/32  
             001 = FOSC/8  
             000 = FOSC/2

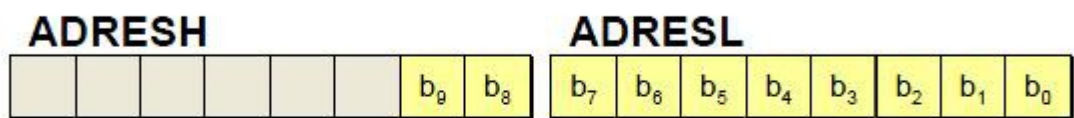
**figure 5 :** présentation du registre **ADCON2**



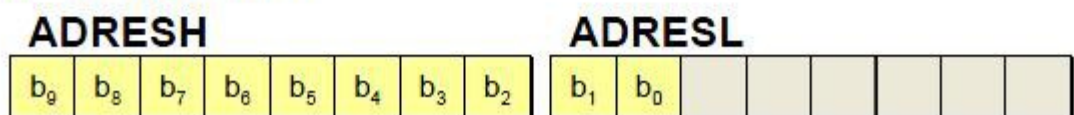


Le bit **ADFM** du registre **ADCON2** permet de sélectionner le mode 10bits ou 8bits. Si nous souhaitons travailler en mode 10bits, il nous suffit de positionner **ADFM** à '1'. Après conversion nous irons alors lire les registres **ADRESH** et **ADRESL** dans lesquels est stockée la valeur convertie (cf. **figure 6**). En mode 8bits (**ADFM=0**), après conversion nous irons seulement lire le registre **ADRESH** (cf. **figure 6**). Dans le cadre de cet exercice, nous travaillerons en mode 8bits, cela est amplement suffisant compte-tenu de la résolution de l'afficheur graphique utilisé (64x128).

bit 7 **ADFM**: A/D Result Format Selection bit  
1 = Right Justified

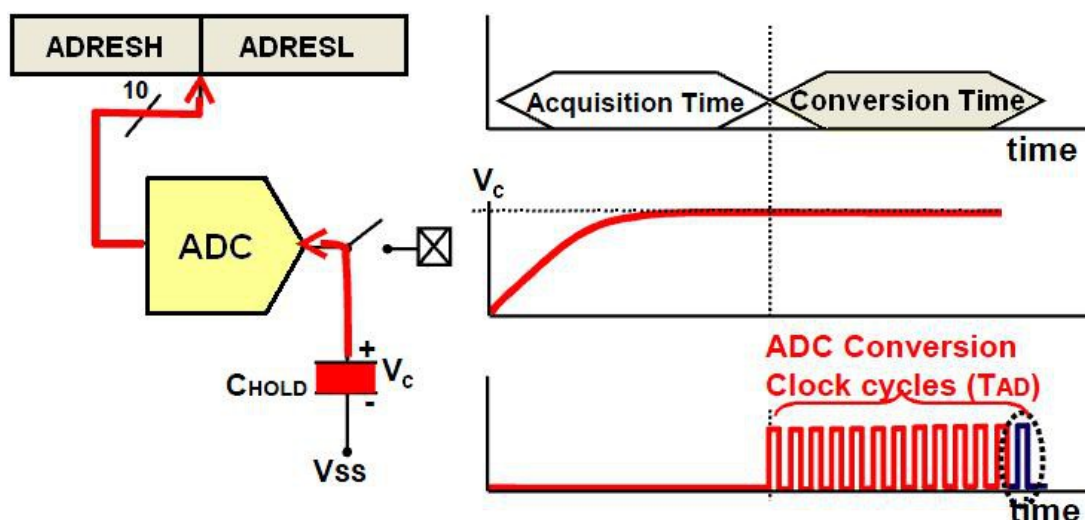


0 = Left Justified



**figure 6** : bit ADFM du registre ADCON2

Avant de présenter la suite du registre **ADCON2**, nous devons effectuer quelques rappels concernant les étapes rencontrées durant une conversion analogique/numérique. Ces étapes concernent notamment les **convertisseurs à approximations successives**, type de convertisseur utilisé par notre MCU. Avant qu'une donnée soit effectivement convertie, nous rencontrons deux cycles (cf. **figure 7**), l'acquisition et la conversion.



**figure 7** : représentation simplifiée des cycles d'acquisition et de conversion

➔ **Temps d'acquisition** : Avant d'effectuer une conversion, nous devons "stabiliser" la tension à convertir de façon à être certain qu'elle ne fluctue pas durant la conversion. Cela est fait en chargeant un condensateur ( $C_{\text{HOLD}} = 25\text{pF}$ ) jusqu'à atteindre la valeur à convertir, puis à ouvrir un interrupteur en entrée afin de maintenir le niveau de tension aux bornes de  $C_{\text{HOLD}}$ . L'ensemble interrupteur et condensateur de maintien est souvent nommé échantillonneur/bloqueur. Ce traitement prend du temps et dépend notamment de la température, de la plage d'acquisition d'entrée ... etc

➔ **Temps de conversion** : Une fois la valeur à convertir stabilisée, nous pouvons la convertir. Il existe différents types de convertisseurs (FLASH, Sigma-Delta ...), celui rencontré ici est à approximations successives. Pour ce type de convertisseurs, le temps de conversion dépend directement de la taille du mot de sortie (10bits dans notre cas). Ce temps est spécifié dans la documentation technique et dure entre 11 et 12  $T_{\text{AD}}$ . Nous devons maintenant savoir à quoi correspond  $T_{\text{AD}}$  ... et du même coup comprendre comment configurer le registre ADCON2 (cf. **figure 5**).



*$T_{\text{AD}}$  (A/D clock Period) est la référence de temps utilisée par le convertisseur pour effectuer les différents traitements internes dont il a la tâche. C'est au programmeur de fixer sa valeur, cependant une valeur minimale est imposée par le constructeur (cf. Table 28-29 datasheet PIC18F4550) :  $T_{\text{ADmin}} = 700\text{ns}$*

Ce sont les bits **ADCS2:ADCS0** de registre **ADCON2** (cf. **figure 5**) qui permettent de fixer  $T_{\text{AD}}$ . Si nous souhaitons obtenir un temps de conversion court, nous avons bien évidemment tout intérêt à nous rapprocher le plus possible de la valeur minimale spécifiée par le constructeur. La valeur de  $T_{\text{AD}}$  est proportionnelle à  $T_{\text{osc}} = 1/F_{\text{osc}} = 20,8\text{ns}$  qui est la référence de temps interne de notre microcontrôleur (cf. **exercice 2**).

bit 2-0	ADCS2:ADCS0: A/D Conversion Clock Select bits
111	= FRC (clock derived from A/D RC oscillator)
110	= FOSC/64
101	= FOSC/16
100	= FOSC/4
011	= FRC (clock derived from A/D RC oscillator)
010	= FOSC/32
001	= FOSC/8
000	= FOSC/2

**figure 8 :** bits ADCS2:ADCS0 du registre ADCON2



*Dans le cadre de notre application, nous devons avoir  $T_{\text{AD}} > 700\text{ns}$ . Nous vous laissons le soin de trouver la bonne valeur à imposer à **ADCS2:ADCS0** pour arriver à nos fins !*

Le dernier paramètre à fixer via les bits **ACQT2:ACQT0** du registre **ADCON2** est le temps d'acquisition  $T_{ACQ}$ . Ce temps est le plus complexe à calculer et peut changer d'une application à une autre. Pour bien assimiler ce qui se cache derrière il faut posséder quelques bases en électronique analogique. Néanmoins, le constructeur fournit des formules permettant d'**estimer** sa durée en fonction de l'application. En se plaçant dans des conditions limites ( $50^{\circ}\text{C}$ ,  $V_{REF+}=5\text{V}$  ...etc), la **valeur maximale de  $T_{ACQ}$  avoisine les  $6,4\mu\text{s}$** .

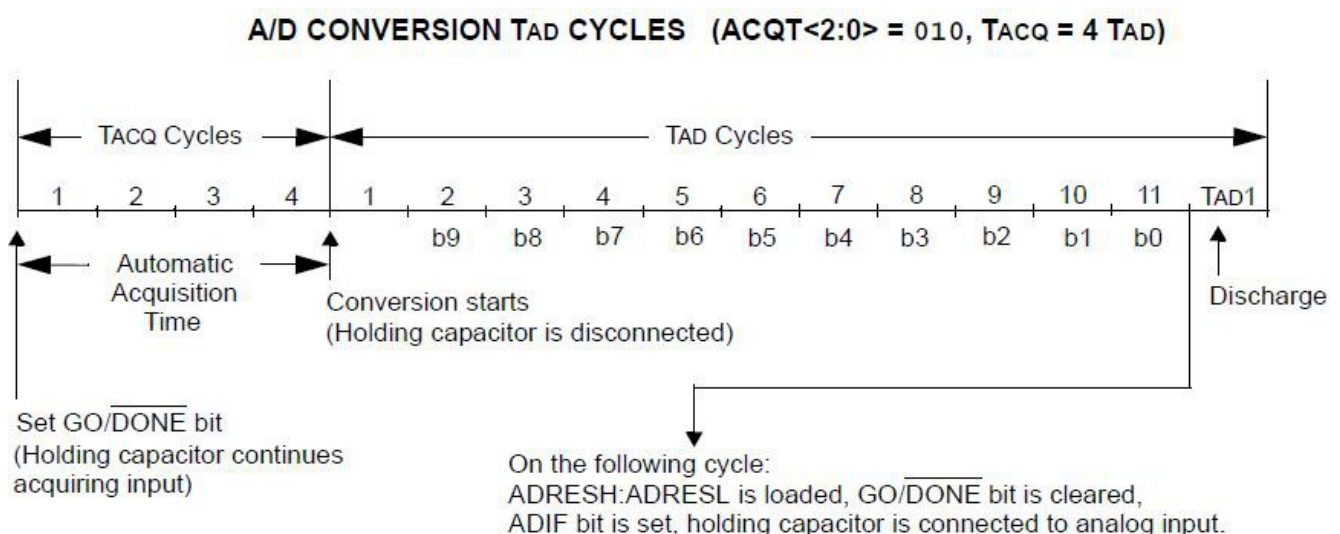
bit 5-3	ACQT2:ACQT0: A/D Acquisition Time Select bits
111	= 20 TAD
110	= 16 TAD
101	= 12 TAD
100	= 8 TAD
011	= 6 TAD
010	= 4 TAD
001	= 2 TAD
000	= 0 TAD

**figure 9** : bits ACQT2:ACQT0 du registre ADCON2



*Dans le cadre de notre application, nous chercherons à avoir  $T_{ACQ} > 6,4\mu\text{s}$ . En respectant cette contrainte, nous garantissons que le convertisseur laissera suffisamment de temps au signal à convertir pour se stabiliser. Nous vous laissons également le soin de choisir les bonnes valeurs à imposer aux bits **ACQT2:ACQT0** !*

Vous trouverez **figure 10** un exemple de conversion A/D (analogique/numérique) présentant les deux cycles (acquisition et conversion). Dans cet exemple,  $T_{ACQ}$  a été fixé à  $4 \times T_{AD}$  (dépend de l'application) et b9-b0 représente la donnée numérique de sortie en court de conversion (cf. **figure 6**).



**figure 10** : exemple de conversion (cf. datasheet PIC18F4550)

### 3. Quelques rappels sur la TFD ?

Avant d'effectuer quelques rappels sur la TFD (Transformée de Fourier Discrète), vous pouvez si vous le souhaitez aller voir sur la plateforme pédagogique de l'école le TD n°2 proposé aux 1A elec en "Traitement **Numérique** du Signal".

L'algorithme de la TFD est du ressort du domaine du TNS. Dès que nous travaillons sur un processeur numérique (MCU, DSP, PC ...) nous ne pouvons plus manipuler des grandeurs continues (domaine du TS – Traitement du Signal) que ce soit dans le domaine temporel ou fréquentiel. Les signaux traités sont forcément discrets. **La TFD est un algorithme permettant de calculer des échantillons spectraux à partir d'un vecteur d'échantillons temporels d'entrée. Le spectre obtenu est donc discret.** Par exemple, pour un vecteur d'entrée de 32 échantillons temporels, on obtiendra une TFD de 32 échantillons spectraux entre 0 et  $f_s$  (fréquence d'échantillonnage). Entre 0 et  $f_s$ , la taille du vecteur d'entrée fixe le nombre de points de la TFD.

$$(1) \quad X\left(\frac{nf_s}{N}\right) = \sum_{k=0}^{N-1} x(kT_s) e^{-j2\pi \frac{kn}{N}}$$

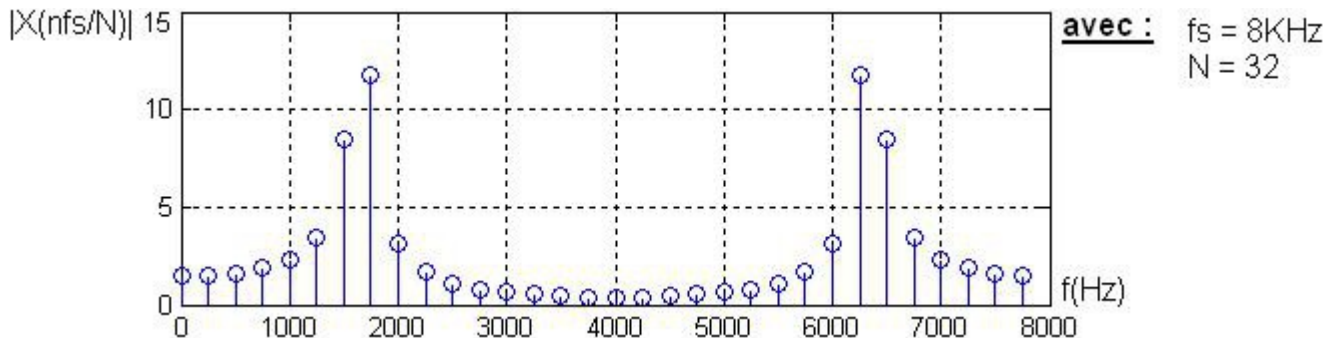
**avec :**  $f_s$  = fréquence d'échantillonnage                       $T_s$  = période d'échantillonnage  
 $n$  = numéro de l'échantillon fréquentiel en cours de calcul  
 $k$  = compteur sur les échantillons temporels  
 $N$  = nombre de points de la TFD = taille du vecteur d'échantillons temporels d'entrée  
 $X(nf_s/N)$  = échantillon spectral numéro "n" à la fréquence  $nf_s/N$   
 $x(kT_s)$  = échantillon temporel numéro "k" du vecteur d'entrée

La formule (1) de la TFD ne nous donne pas le spectre dans sa totalité entre 0 et  $f_s$ . Elle ne permet d'obtenir que l'échantillon numéro "n" correspondant à la fréquence  $nf_s/N$ . Si nous souhaitons obtenir le spectre dans son ensemble, nous devons donc estimer les N échantillons spectraux comme présenté sur la formule (2) :

$$(2) \quad \text{Spectrum} = \left\{ X\left(\frac{nf_s}{N}\right) \right\}_{n=0}^{N-1}$$



La **figure 11**, réalisée sous MATLAB, présente un exemple de spectre obtenu après calcul d'une TFD 32 points sur un signal sinusoïdal de 1650Hz échantillonné à 8KHz.



**figure 11 :** TFD 32 points sur un signal sinusoïdal d'amplitude "1" et de fréquence 1650Hz

La première question à se poser si nous souhaitons implémenter la formule de la TFD est, comment programmer une exponentielle complexe ? Le problème au niveau de notre microcontrôleur est que nous ne pouvons travailler qu'avec des grandeurs scalaires. En effet, le jeu d'instruction de notre MCU ne permet d'effectuer que des opérations arithmétiques sur des nombres scalaires (+, -, x). Nous allons donc tout simplement décomposer l'exponentielle en fonctions trigonométriques. **Nous calculerons ensuite indépendamment la partie réelle  $X_{\text{real}}$  de la TFD (scalaire) et la partie imaginaire  $X_{\text{im}}$  (scalaire), cf. formule 3.**

$$(3) \quad X\left(\frac{nf_s}{N}\right) = \sum_{k=0}^{N-1} x(kT_s) \cos\left(2\pi \frac{kn}{N}\right) + j \sum_{k=0}^{N-1} x(kT_s) \sin\left(2\pi \frac{kn}{N}\right)$$

$$X\left(\frac{nf_s}{N}\right) = X_{\text{real}}\left(\frac{nf_s}{N}\right) + j X_{\text{im}}\left(\frac{nf_s}{N}\right)$$



*Nous chercherons ensuite à calculer le module de chaque échantillon spectral (cf. formule 4). Dans cette formule, chaque valeur calculée ou utilisée est un scalaire. Il n'y a donc aucun problème pour la programmer !*

$$(4) \quad \left| X\left(\frac{nf_s}{N}\right) \right| = \sqrt{X_{\text{real}}^2\left(\frac{nf_s}{N}\right) + X_{\text{im}}^2\left(\frac{nf_s}{N}\right)}$$



*Bien, vous êtes maintenant prêts à attaquer le TP ! ... vous pouvez d'ailleurs dès à présent commencer à compléter les fichiers sources présents juste après ces questions.*

*Cependant, dans un soucis pédagogique, il vous est demandé un petit travail préparatoire. Il sera vérifié en début de séance et vous permettra d'anticiper une perte de temps non négligeable dans l'avancement des TP (nb ★ = difficulté).*

1. (★) Proposez une configuration du Timer0 et de l'interruption associée de façon à exécuter le programme d'interruption de priorité haute toutes les 125µs (8KHz).

2. (★★) Dans le cadre de notre application, proposez une configuration de l'ADC sachant que nous utiliserons seulement la broche RA0 en entrée analogique,  $V_{REF-} = 0V$ ,  $V_{REF+} = 5V$ , mode 8bits ... détaillez les valeurs des registres.

**ADCON0 =**

**ADCON1=**

**ADCON2=**

3. (★) Pour quelles raisons une acquisition A/D n'est-elle pas faite instantanément ?

4. (★★★) Dans le cadre de notre application, combien de temps s'est-il écoulé entre le moment où nous avons lancé une conversion et le moment où nous sommes allés lire la valeur dans le registre ADRESH ?



*Listing du programme **ex6.c**, projet TFD*

```

/*****
@file : ex6.c
@brief : Implémentation d'une TFD et affichage "temps réel" sur GLCD
@author :
last modification :
*****/

/**** Configuration bits - TCY = 83,2ns ****/
#pragma config PLLDIV=2, CPUDIV=OSC1_PLL2, FOSC=HSPLL_HS, BOR=OFF, WDT=OFF, MCLRE=ON

/**** Includes files ****/
#include <p18f4550.h>
#include <delays.h>
#include <GLCD.h>          /**** Librairie "GLCD.lib" développée par un 2Ainfo en 2009-2010 ****/
#include "TNS.h"

/**** Définition des Macros ****/
#define DelayUser_1ms()      Delay1KTCYx(12)          // 12.1000.TCY ~ 1ms
#define TMR0LVALUE          0x00                    /**** à compléter ! ****/
#define TMR0HVALUE          0x00                    /**** à compléter ! ****/

/**** Définition des variables globales ****/
unsigned char DFT_Start = 0;
signed char Inputbuffer[BUFFER_SIZE];

#pragma code
/****
/**** ISR : routine d'interruption de priorité haute ****
/****
#pragma interrupt High_ISR
void High_ISR(void)
{static unsigned char DataIndex = 0;

    if(INTCONbits.TMR0IF)
    {
        /**** Clear Timer0 overflow flag ****/

        /**** à compléter ! ****/

        /**** pré-chargement de TMR0L + chargement automatique de TMR0H ****/

        /**** à compléter ! ****/

        /**** Lancer une conversion ****/

        /**** à compléter ! ****/

        /**** Attendre la fin de conversion ****/

        /**** à compléter ! ****/

        /**** Acquisition de BUFFER_SIZE éléments ****/

        /**** à compléter ! ****/
    }
}
****/
```

```
/** Configuration du vecteur d'interruption de priorité haute **/  
#pragma code high_vector = 0x08  
void interrupt_at_high_vector(void)  
{  
    _asm goto High_ISR_endasm  
}  
  
#pragma code  
/*****/  
/** FONCTION : initialisation des ports A et D **/  
/*****/  
void PORT_Init(void)  
{  
    /** Broche RA0 en entrée et RA1-RA7 en sorties **/  
  
    /** à compléter ! **/  
  
    /** Broches RD0-RD7 en sorties pour la validation d'une conversion A/D **/  
  
    /** à compléter ! **/  
}  
  
/*****/  
/** FONCTION : initialisation du timer0 et de l'IT associée **/  
/*****/  
void TIMER0_Init(void)  
{  
    /** T0CON : Timer0=OFF, mode=16bits, synchro=Tcy, prescaler=ON, prescale value=1:2 **/  
  
    /** à compléter ! **/  
  
    /** TMR0 : Initialisation des registres de pré-chargement **/  
  
    /** à compléter ! **/  
  
    /** Gestion des IT : priority mode, vecteur d'IT de priorité haute, démasquage IT TMR0, TMR0IF=0 **/  
  
    /** à compléter ! **/  
  
    /** Démarrage du Timer0 **/  
  
    /** à compléter ! **/  
}  
  
/*****/  
/** FONCTION : initialisation de l'ADC **/  
/*****/  
void ADC_Init(void)  
{  
    /** ADCON0 : Channel=0, converter module disable **/  
  
    /** à compléter ! **/  
  
    /** ADCON1 : Vref+ = VDD, Vref- = Vss, AN0 only is Analog Input **/  
  
    /** à compléter ! **/  
}
```



```
    /*** ADCON2 : Tadmin=700ns < Tad=32*Tosc (ou 64*Tosc), Left Adjust, TacqSelect=12*Tad > 6,4us ***/

    /*** à compléter ! ***/

    /*** Validation du module ADC ***/

    /*** à compléter ! ***/
}

/*****
*** PROGRAMME PRINCIPAL ***
*****/
void main() {
    unsigned char Spectrum[TFD_SIZE];

    /*** Appel fonction de configuration des ports A et D ***/

    /*** à compléter ! ***/

    /*** Appel fonction de configuration du Timer0 et de l'IT associée ***/

    /*** à compléter ! ***/

    /*** Appel fonction de configuration de l'ADC ***/

    /*** à compléter ! ***/

    /*** Initialisation de l'afficheur graphique GLCD ***/

    /*** à compléter ! ***/

    /*** Démasquage global des IT ***/

    /*** à compléter ! ***/

    while(1){
        /*** Test l'acquisition de 32 échantillons temporels d'entrée***/
        if(????) {
            /*** Masquage global des IT et RAZ du flag buffer d'entrée plein ***/

            /*** à compléter ! ***/

            /*** Algorithme de calcul de la TFD (DFT) ***/

            /*** à compléter ! ***/

            /*** Affichage du spectre sur l'afficheur graphique GLCD-64x128 ***/

            /*** à compléter ! ***/

            /*** Démasquage global des IT ***/

            /*** à compléter ! ***/
        }
    };
}
```



## Listing du programme *TNS.c*

```

/*****
@file : TNS.c
@brief : Librairie pour le TNS. Calcul TFD ...
@author :
last modification :
*****/
#include "TNS.h"

/** Déclaration des variables globales utilisée par la fonction "TNS_dft"      ***/
/** ATTENTION : Elles ne peuvent pas être utilisées en variables locales :   ***/
/** Problème de taille de la pile soft !                                     ***/
rom      char    Sine[SINUS_SIZE/2] = {SINE_TABLE};
unsigned long    X[TFD_SIZE/2];
           long    Xim[TFD_SIZE/2];
           long    Xreal[TFD_SIZE/2];

/*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*/
/*XXXXXXXXXXXXX FONCTIONS PUBLIQUES xxxxxxxxxxxxxx*/
/*XXXXXXXXXXXXX Externes à TNS.c xxxxxxxxxxxxxx*/
/*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*/

/*****
*** FONCTION : Calcul d'une TFD (DFT)                                     ***/
*** @param ptInputBuffer   pointeur sur le vecteur d'échantillons temporels ... ***/
*** @param ptOutputSpectrum pointeur sur le vecteur d'échantillons frequentiels ... ***/
*****/

void TNS_dft(signed char* ptInputBuffer,unsigned char* ptOutputSpectrum){
unsigned      char    i, n, k, kn;
signed        char    Sinus[SINUS_SIZE], Cosinus[SINUS_SIZE];

/** Initialisations avant calcul de la TFD ***/

        /** à compléter ! ***/

/** Calcul de la moitié de la TFD [0,Fs/2]. Si nécessaire, obtention de la deuxième ... ***/

        /** à compléter ! ***/

/** Obtention du spectre complet [0,Fs] ***/

        /** à compléter ! ***/
}

```



## Listing du programme *TNS.h*

```

/*****
@file : TNS.h
@brief : include file pour travailler avec la librairie TNS.c
@author :
last modification :
*****/
#ifndef __TNS_HEADER__
#define __TNS_HEADER__

    /*** Fichiers d'en-tête ***/
    #include <math.h>                                // pour l'API "sqrt" utilisée dans "TNS_dft"

    /*** Déclaration des Tailles des différents vecteurs ***/
    #define TFD_SIZE          32
    #define SINUS_SIZE        32
    #define BUFFER_SIZE       32

    /*** Initialisation des tables ***/
    #define SINE_TABLE         0,25,49,71,90,106,118,125,127,125,118,106,90,71,49,25
    #define TFD_INIT           0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

    /*** Paramétrage de la fonction "TNS_dft" ***/
    #define X_IM_REAL_FACTOR   16
    #define X_FACTOR           6

    /*xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx*/
    /*xxx  Déclarations FONCTIONS PUBLIQUES xxx*/
    /*xxxxxxxxxxx  Externes à TNS.c xxxxxxxxxxxxx*/
    /*xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx*/

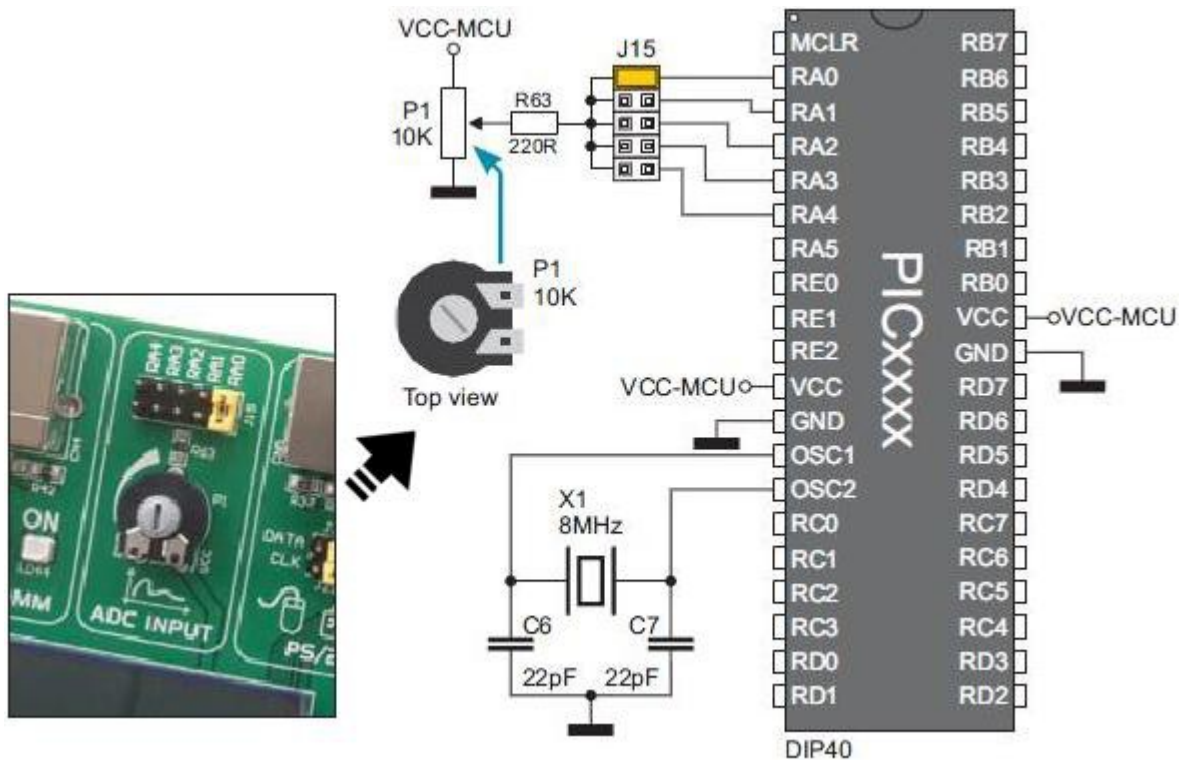
    /*******
    /*** FONCTION : Calcul d'une TFD (DFT)                                     ***/
    /*** @param ptInputBuffer   pointeur sur le vecteur d'échantillons temporels ... ***/
    /*** @param ptOutputSpectrum pointeur sur le vecteur d'échantillons frequentiels ... ***/
    /*******
    extern void TNS_dft(signed char* ptInputBuffer,unsigned char* ptTFDSpectrum);

#endif



```

## 1. Travail pratique (ex6 : partie 1) : conversion A/D

Durant cette première partie de l'exercice votre cahier des charges est de faire l'acquisition d'une grandeur analogique sur la broche RA0 et d'afficher la valeur convertie (8bits) sur le port D. La grandeur analogique sera dans un premier temps imposée par le potentiomètre **P1** présent sur la maquette, n'oubliez donc pas de positionner le cavalier sur la broche RA0 du connecteur **J15** (cf. **figure 12**).



**figure 12** : Schéma de câblage du potentiomètre à l'entrée analogique RA0

- ➔  **Créer un projet ex6 dans votre répertoire de travail (cf. ANNEXE 1)**
- ➔  **Copiez et complétez le listing du programme ex6.c, TNS.c, TNS.h.**
- ➔ Configurez le port D
- ➔ Configurez le Timer0 et démasquez l'interruption associée  
**Projet TFD :  $F_s=2KHz$  et projet Voltmètre :  $F_s=50Hz$**
- ➔ Configurez l'ADC
- ➔ Complétez le programme d'interruption de façon à lancer une conversion puis à afficher la valeur convertie sur le port D



***A vous de jouer maintenant !***



## **2. Travail pratique :** projet Voltmètre

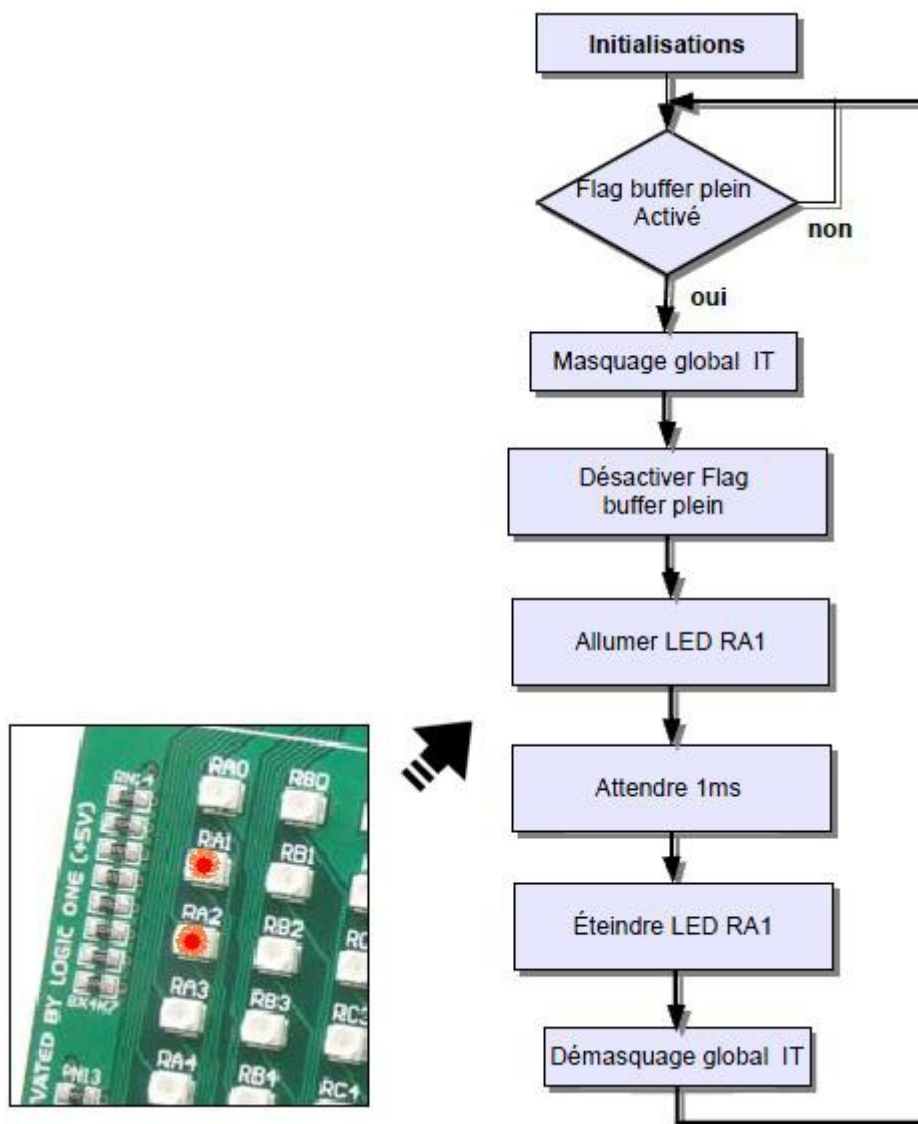
Cette partie ne concerne que les étudiants se lançant dans le développement du projet voltmètre. Votre travail consiste donc à convertir la grandeur à l'entrée de la broche RA0 de l'ADC du MCU, la mettre à l'échelle puis afficher la valeur ainsi mise en forme sur l'afficheur LCD. Voici un exemple d'affichage pour une tension d'entrée valant 3,44V :

t	e	n	s	i	o	n	:		3	.	4	4			V

Afin de gagner du temps dans vos développements, utilisez la librairie standard du C fournie avec la chaîne de compilation (floor, itoa ...).

### 3. Travail pratique : projet TFD

Dans cette seconde partie du projet nous allons effectuer l'acquisition de 32 échantillons d'entrée. Une fois cette acquisition effectuée, nous activerons un flag qui réveillera la boucle principale du main (cf. **figure 13**). Dans cette partie de l'exercice la boucle du main() ne fera qu'allumer la LED connectée à la broche RA1, attendre 1ms puis éteindre la LED. A l'aide des outils de Debug (visualisation des variables) nous nous assurerons également que l'acquisition des 32 échantillons a été bien effectuée. Nous allumerons également la LED connectée à RA2 en entrant dans le programme d'interruption puis nous l'éteindrions avant de le quitter.



**figure 13** : Organigramme de la boucle principale - validation de l'acquisition du buffer d'entrée

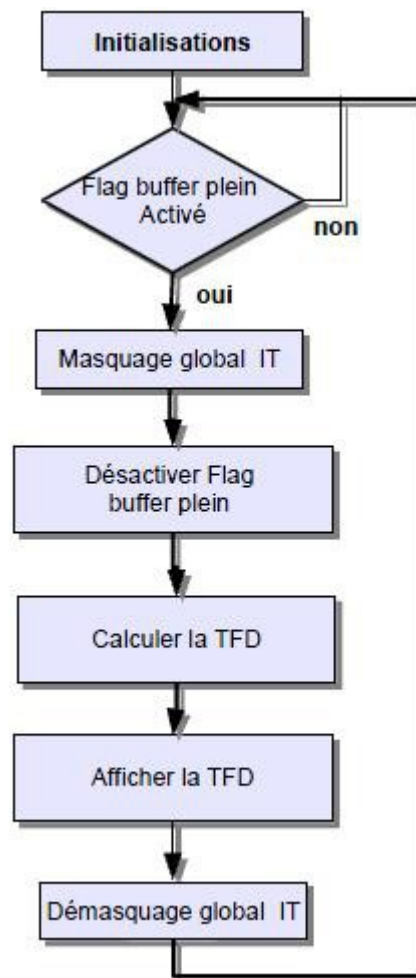


*Observez à l'aide de l'oscilloscope les signaux sur les broches RA1 et RA2 et assurez-vous du bon fonctionnement de votre programme !*

#### 4. Travail pratique : algorithme de la TFD

Nous sommes maintenant capables de faire l'acquisition d'un buffer de 32 échantillons temporels échantillonnés à 8KHz. Nous devons maintenant implémenter l'algorithme calculant une TFD 32 points à partir de ce buffer. Vous trouverez **figure 14** l'organigramme présentant l'architecture du main(). Cependant, il vous faut savoir que cet exercice possède plusieurs objectifs encore inavoués, les voici :

- ➡ Introduire le cours de 2A "DSP et communications PC" en mettant en avant les limites des MCU, notamment les 8bits dans notre cas.
- ➡ Vous forcer à utiliser les outils de Debug proposés par l'IDE de Microchip (MPLAB).
- ➡ Proposer des liens avec des enseignements et des domaines connexes, notamment "Traitement Numérique du Signal" (1AS2) et "Initiation à la programmation" (1AS1)
- ➡ Travailler avec des grandeurs complexes en programmation "C"
- ➡ ...



**figure 14** : Organigramme du main()

Il ne vous reste plus qu'à implémenter l'algorithme de la TFD. Afin de faciliter votre travail, nous travaillerons avec des formats flottants. Cette **TRES MAUVAISE** démarche a également pour but d'introduire l'enseignement de 2A sur les DSP et de vous montrer les limites des MCU 8bits. Soyez également prudent de bien appliquer un facteur démultiplicateur à la fin de l'algorithme afin d'assurer le passage flottants (0 à  $+\infty$ ) vers entiers non signés sur 8bits (0 à 255).



*Nous ne calculerons et n'afficherons que les 16 premiers échantillons de la TFD entre 0 et  $f_s/2$ . Les 16 suivants pouvant si nécessaire être obtenus par symétrie (cf. **figure 11**).*