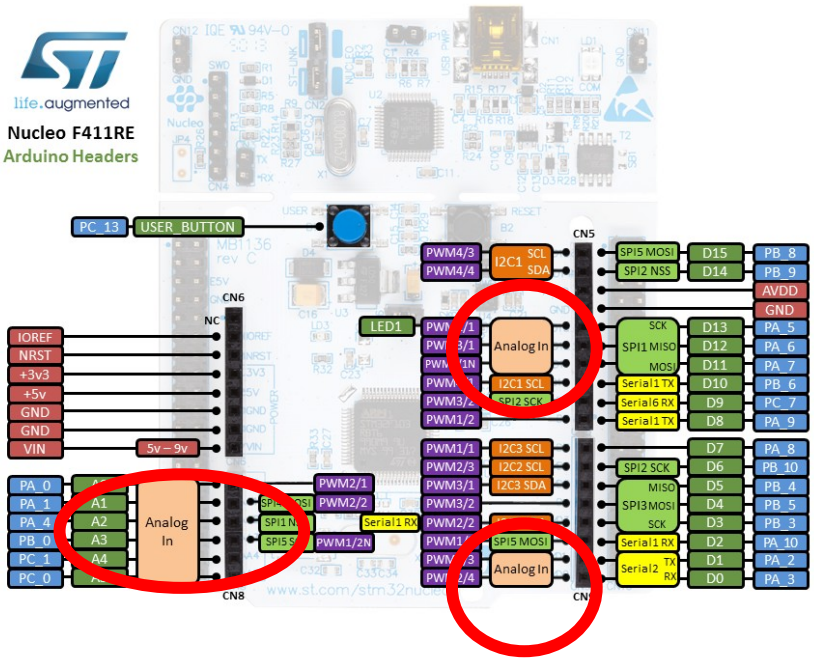
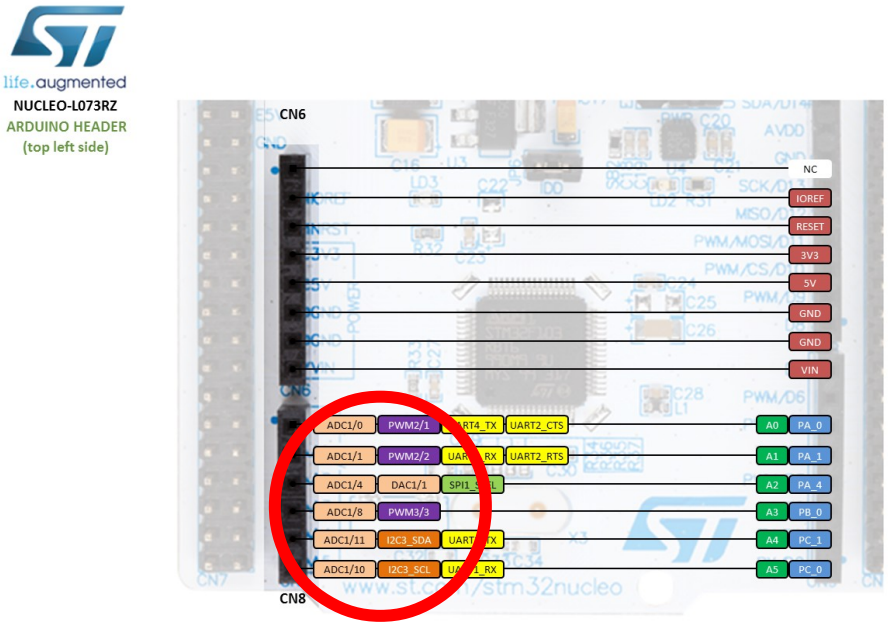
STM32 ADC/DAC

Objectifs : Mise en œuvre du convertisseur analogique numérique
Matériel : Ce TP peut utiliser une NUCLEO-F411RE, mais n'importe quelle autre carte NUCLEO conviendra pour les exercices sur l'ADC en revanche tous les STM32 ne disposent pas de DAC, **les TPs DAC on été réalisés sur NUCLEO-F073RZ.**
Logiciel : MBED
Conserver les corrigés de tous les exercices, ils serviront pour les révisions

Connecteurs "Arduino"



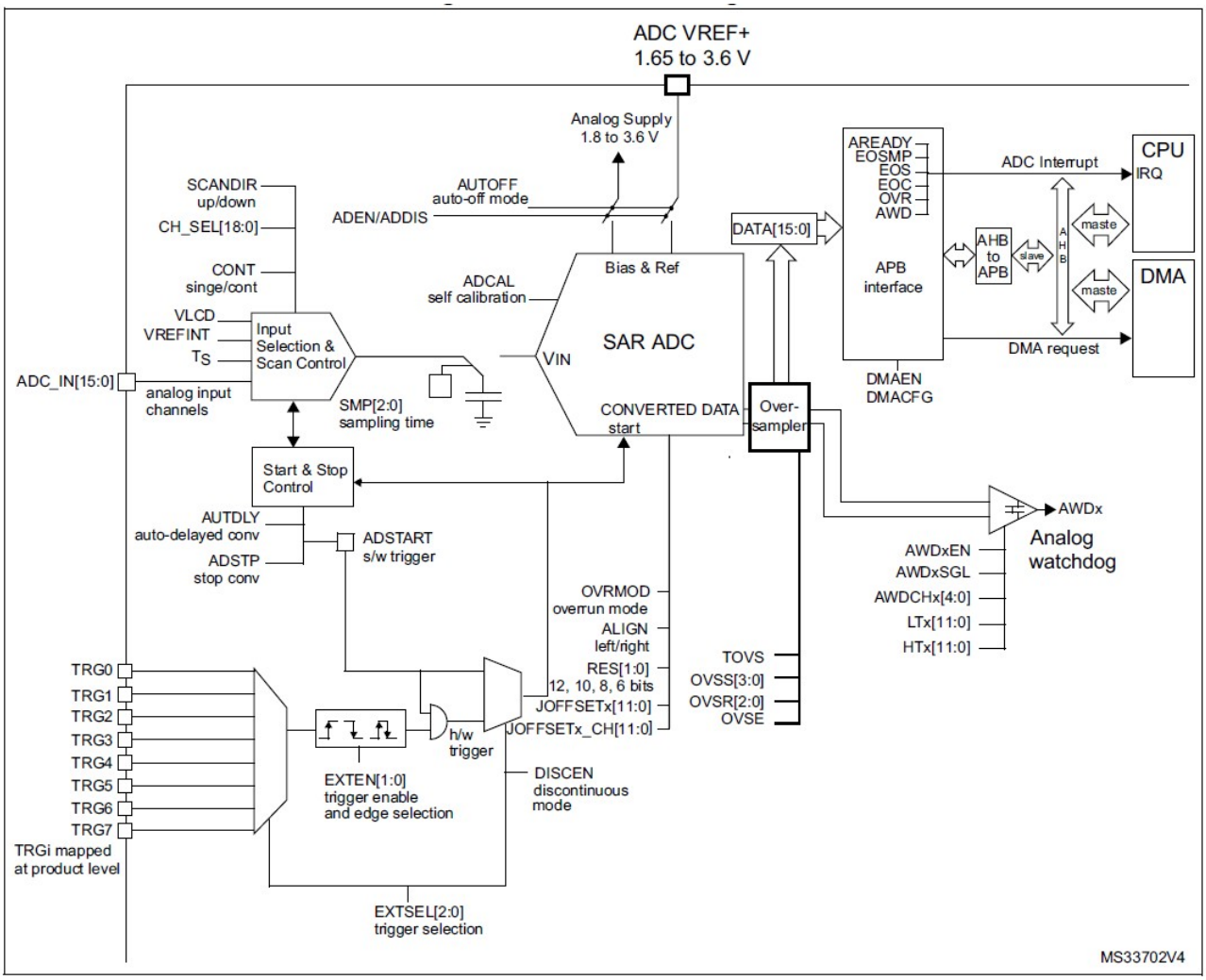


1 Convertisseur analogique numérique du STM32L0

L'ADC 12 bits du STM32 est un convertisseur analogique-numérique à approximations successives. Il a jusqu'à 19 canaux multiplexés lui permettant de mesurer les signaux de 16 sources externes et de 3 sources internes. La conversion A / N des différents canaux peut être effectuée en mode simple, continu, balayage. Le résultat de l'ADC est stocké dans un registre 16 bits. Un sur-échantillonneur intégré permet d'augmenter le nombre de bits effectif (ENOB)

Sur les cartes NUCLEO la tension de référence de l'ADC (VREF+) est par défaut connectée à AVDD, qui elle même est par défaut connectée à VDD=3,3v
Attention, VDD est produit par un régulateur avec une tolérance de +/- 2%.

Le temps de conversion est au maximum de 0,87uS sur 12bits pour un STM32 sur carte NUCLEO



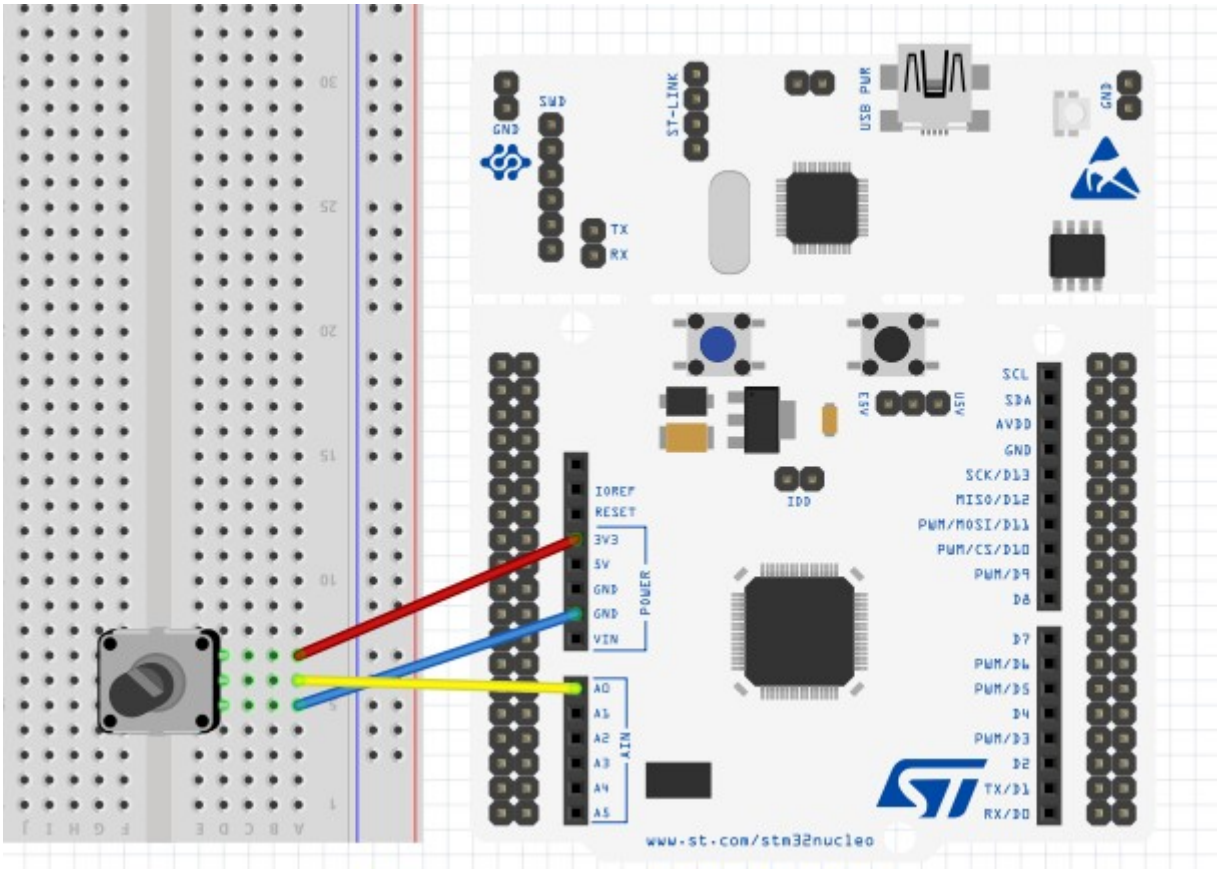
a) Après conversion, exprimer le nombre N dans le registre DATA en fonction de VREF+, et du nombre de bits de l'ADC
b) Expliquer le rôle de
- CH_SEL[18:0]
- ADC_IN[15:0]
- TRG0 à TRG7
- EOC
- CPU IRQ
- DMA
- START

c) Over-sampler
Traduire le texte ci dessous :

> The oversampling unit performs data preprocessing to offload the CPU. It can handle multiple conversions and average them into a single data with increased data width, up to 16-bit.
>
> It provides a result with the following form, where N and M can be adjusted:

$$\text{Result} = \frac{1}{M} \times \sum_{n=0}^{n=N-1} \text{Conversion}(t_n)$$

Mise en œuvre



Câbler un potentiomètre (1KΩ à 100KΩ), extrémités 3,3v et 0v , curseur connecté sur A0 (AN0)

Programmer le uC avec le programme suivant depuis MBED.

```
#include "mbed.h"
#define VREF 3.3

Serial pc(SERIAL_TX, SERIAL_RX);
AnalogIn  adin(A0);
DigitalOut maled(LED1);

int main(void)
{
    while (1)
    {
        if(adin > 0.3) maled = 1;
        else maled = 0;
        pc.printf("valeur en %% de VREF+ : %3.3f\n", adin.read()*100.0);
        pc.printf("valeur brute  0x%04X \n", adin.read_u16());
        pc.printf("tension : %f \n",adin.read()*VREF);
        wait(0.5);
    }
}
```

`AnalogIn  adin(A0);` création d'une instance de la classe AnalogIn, le paramètre représente le nom de la broche utilisée

`read();` méthode de la classe AnalogIn retournant un nombre réel 0.0 pour une tension de 0v et 1.0 pour une tension égale à VREF+
Cette méthode est appelée par défaut.

`read_u16());` méthode de la classe AnalogIn retournant un nombre entier sur 16bits (0x0000 pour 0v et 0xFFFF pour VREF+). Cette méthode utilise le sur-échantillonnage et la décimation matérielle permettant de passer d'un résultat sur 12bits à un résultat sur 16bits.

1.1 Exercice

Tracer N(16) = f(Vin), nombre sur 1-bits en fonction de la tension d'entrée.
Calculer le quantum.

1.2 Exercice , thermomètre

On désire réaliser un thermomètre à l'aide d'une thermistance NTCLE100E3103JB0 CTN 10 KOhms, référence Farnell : 1187031
Afin d'améliorer la précision il est possible de comparer VREF+ à la tension AVDD théorique qui ne sera jamais exactement 3,3v

```
AnalogIn adc_vref(ADC_VREF);
```

La tension de calibration usine peut être obtenue par :

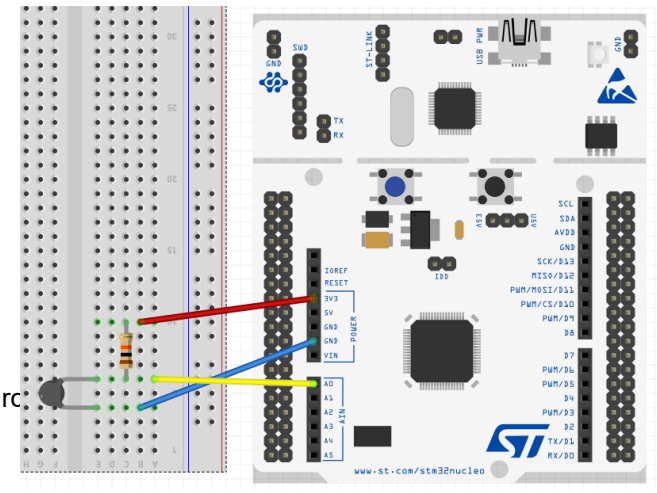
```
unsigned short int vref_cal;
vref_cal =  *(__IO uint16_t *)((uint32_t)0x1ff800f8);
```

La tension de référence actuelle par :

```
VDDA = 3  * (vref_cal / adc_vref);
```

Afficher la température ambiante sur un terminal

Quelle est la **résolution** du thermomètre en degrés centigrades



1.3 Exercice Mesure de la température interne du uC.

Le STM32 dispose d'un capteur de température interne. Le capteur de température interne est généralement utilisé pour mesurer la température du cœur du STM32 afin d'éviter une destruction par "effet joule".

La tension de sortie du capteur de température change linéairement avec la température. Le capteur n'est pas calibré (jusqu'à 45 ° C d'une puce à l'autre). Pour améliorer l'exactitude de la mesure des valeurs d'étalonnage (TS_CAL1 et TS_CAL2) sont stockées dans la mémoire du microcontrôleur pendant la production.

```
const uint16_t TS_CAL1 = *(uint16_t *)(0x1FFF75A8);
const uint16_t TS_CAL2 = *(uint16_t *)(0x1FFF75CA);
```

Temperature (°C) = ((130 °C – 30 °C)/(TS_CAL2 – TS_CAL1)) * (TS_DATA – TS_CAL1) + 30 °C
TS_DATA est le nombre obtenu après conversion sur ADC_TEMP

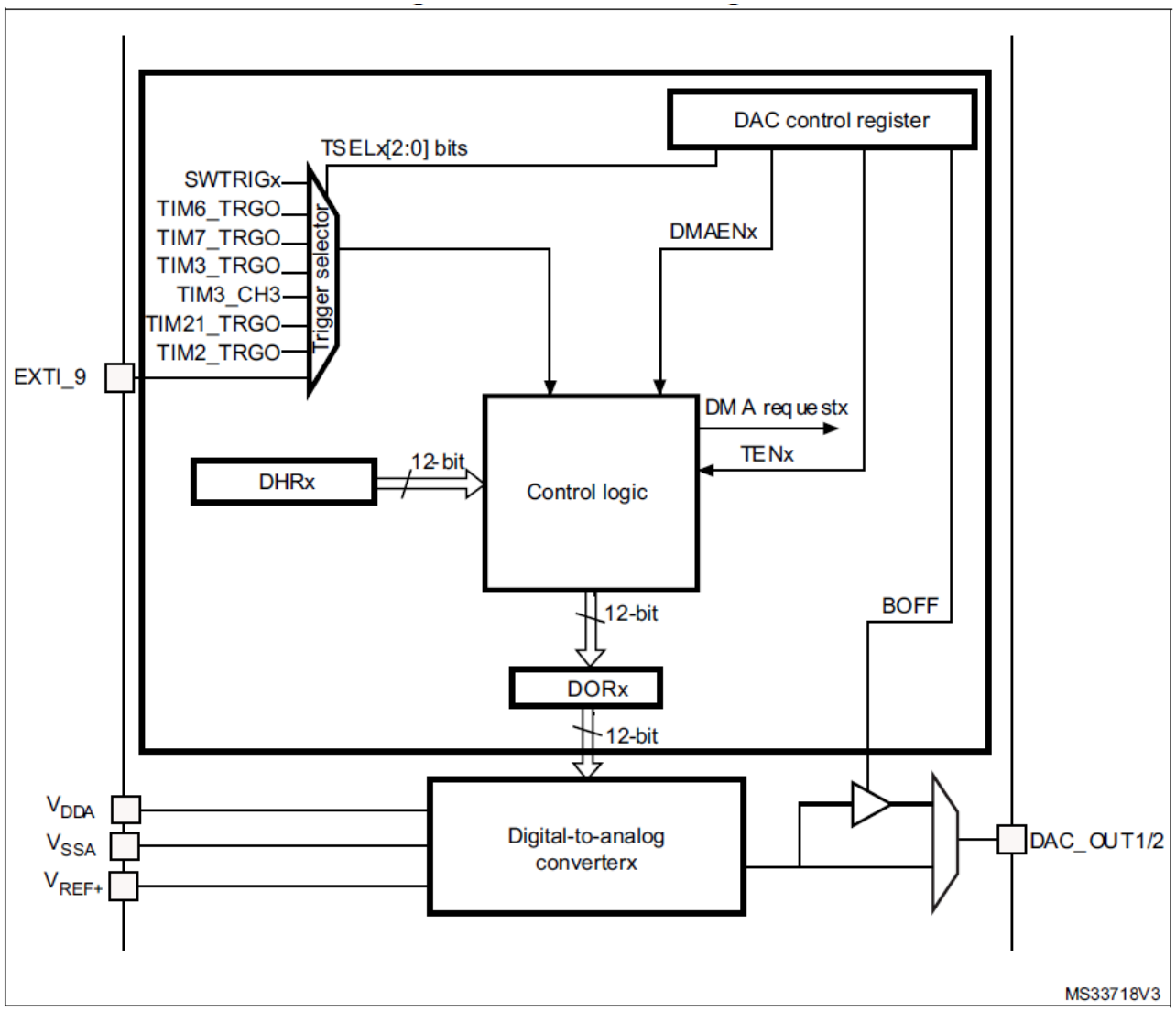
A partir de ce programme

```
#include "mbed.h"

AnalogIn adc_temp(ADC_TEMP);  // instance sur le capteur de température interne

int main()
{
    printf("\nSTM32 temperature interne\n");
    while(1) {
        printf("ADC Temp = %f\n", adc_temp.read());
        printf("\033[1A");  // controle sur terminal ASCII
        wait(1.0);
    }
}
```

Afficher la valeur de la température interne exacte en introduisant une correction avec les valeurs de calibration.

2 Convertisseur numérique analogique (DAC)

Certains STM32 comme le STM32 L073RZ disposent d'un Convertisseur Numérique Analogique 12 bits (Digital Analog Converter ou DAC). **! Le STM32F411RE ne possède pas de DAC !**



La tension de référence du DAC est VREF+

a) Donner l'équation de la tension de sortie DAC_OUT en fonction de N et VREF+
b) Donner la valeur du quantum q pour VREF+=3.3v

c) Programmer le STM32 avec le programme ci-dessous et placer un oscilloscope sur la broche PA4)
Interpréter les résultats, le quantum est-il mesurable ? Pourquoi ?
Combien de « marches » sont visibles ? Pourquoi ?

```
#include "mbed.h"

#if !DEVICE_ANALOGOUT
#error Ce uC ne dispose pas de DAC
#else
AnalogOut dac_out(PA_4);
#endif

int main()
{
    printf("Sawtooth example\n");

    while(1) {
        for (int i = 0; i < 0xFFFF; i += 5000) {
            dac_out.write_u16(i);
            wait_us(10);
        }

        dac_out.write_u16(0xFFFF);
        wait_us(100);

        for (int i = 0xFFFF; i > 0; i -= 5000) {
            dac_out.write_u16(i);
            wait_us(10);
        }

        dac_out.write_u16(0);
    }
}
```

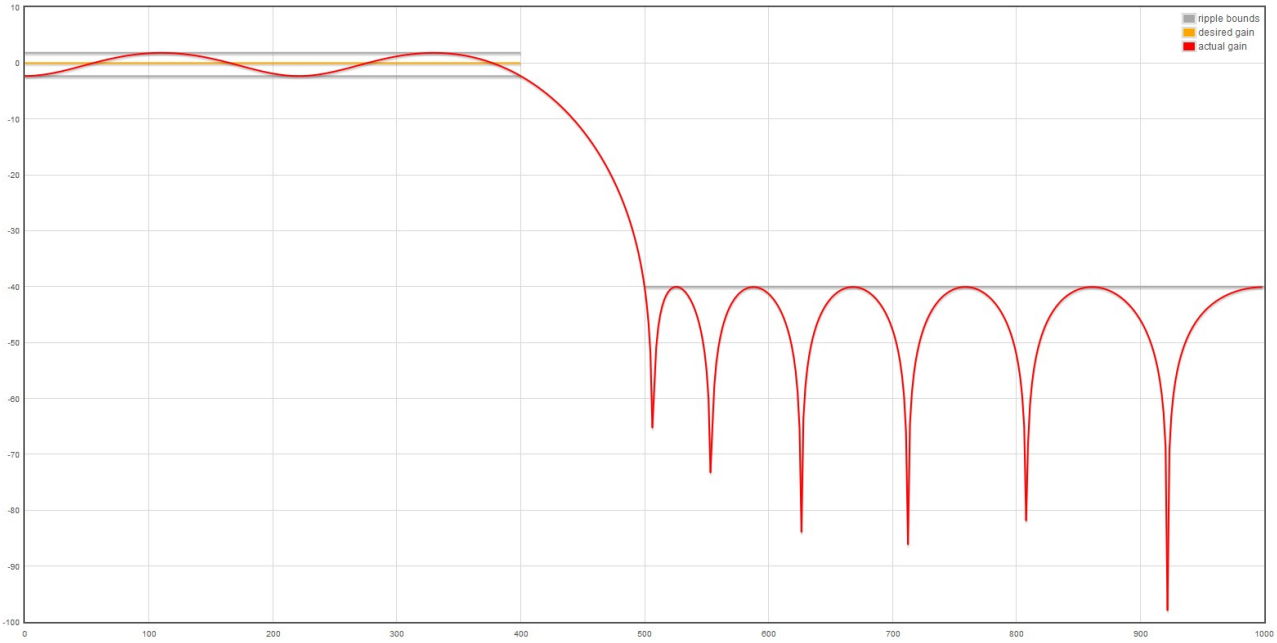
d) Expliquer le rôle des méthodes out.write et out.write_u16

2.1 Exercice, recopie

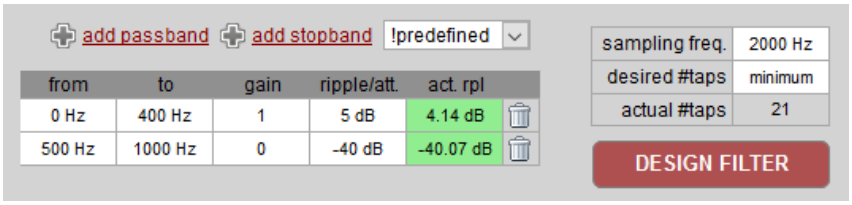
Réaliser un programme recopiant la tension sur AN0 vers PA4
On placera sur AN0 un GBF produisant une tension sinusoïdale d'amplitude crête à crête de 1v, de valeur moyenne 1v et de fréquence 100Hz
Observer en correspondance les deux signaux AN0 et PA4.
Augmenter la fréquence à 1KHz , que constatez vous ?
Mesurer alors la somme des temps de conversion.
D'après le théorème de Shannon quelle sera la fréquence maximum théorique des signaux en entrée ?

2.2 Exercice, filtre passe bas

On se propose de réaliser un simple filtre passe bas FIR de type y=a1.y(n-1)+a2.y(n-2)+a3.y(n-3)+.....
Le site http://t-filter.engineerjs.com/ permet le design et le calcul des coefficients du filtre.



Entrer les paramètres du filtre puis DESIGN FILTER



a) Créer un projet vide dans MBED,

Créer et copier les fichiers suivants dans le projet

filtre.h

```
#ifndef FIRFILTER_H_
#define FIRFILTER_H_

#define nbcoeffs ????    // inscrire ici le nombre de coefficients du filtre


class FirFilter
{
    public:
      FirFilter();
      void put(double value) ;
      double get() ;

    private:
        double history[nbcoeffs];
        int current_index_;
};

#endif
```

filtre.cpp

```
#include "filtre.h"
#include "coefficients.h"

FirFilter::FirFilter()
{
    current_index_=0;
    for(int i = 0; i < nbcoeffs; ++i)   history[i] = 0.0;
}

void FirFilter::put(double value)
{
    history[current_index_++] = value;
    if(current_index_ == nbcoeffs)  current_index_ = 0;
}

double FirFilter::get()
{
    double output = 0.0;
    int index = current_index_;
    for(int i = 0; i < nbcoeffs; ++i)
    {
      if(index != 0)  --index;
      else index = nbcoeffs - 1;
      output += history[index] * coeffs[i];
    }
    return output;
}
```

coefficients.h

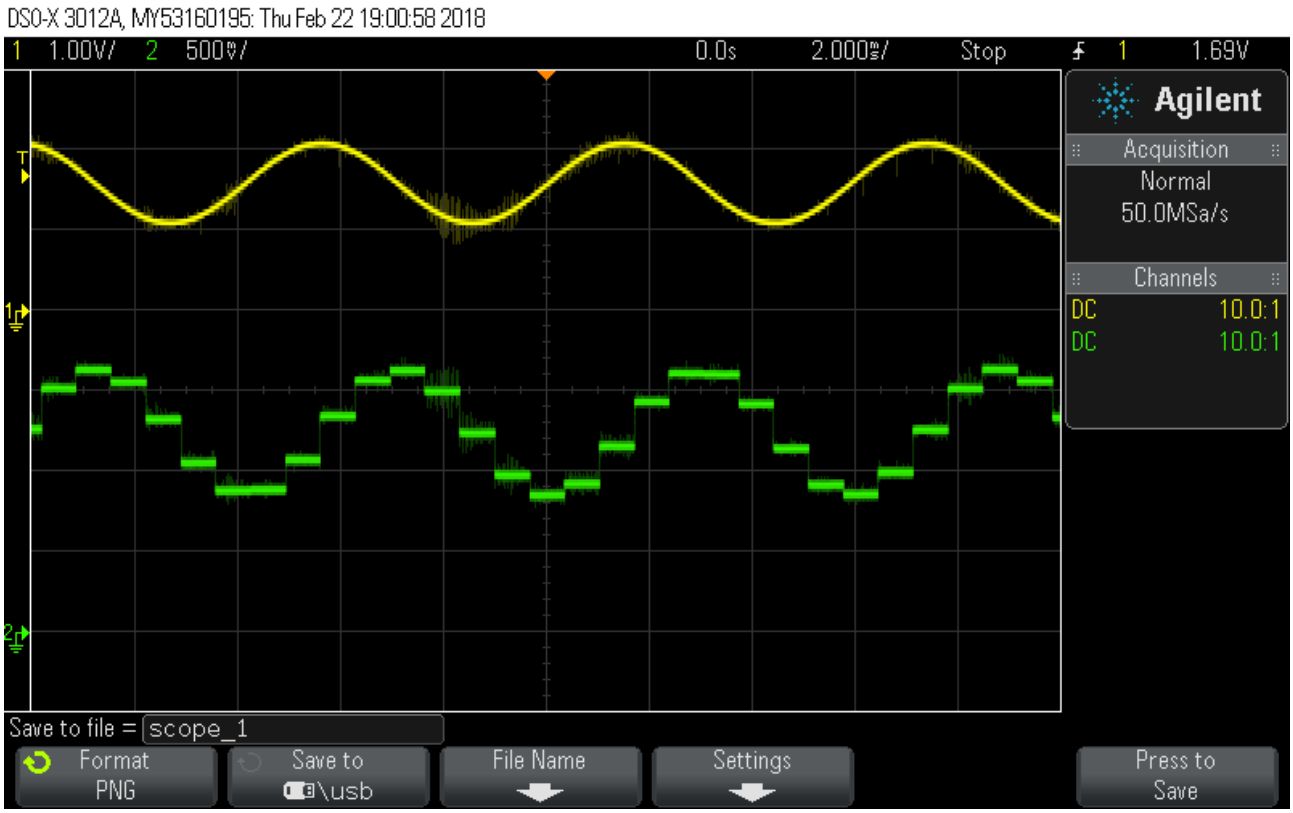
```
#ifndef COEFFS_H_
#define COEFFS_H_

double coeffs[] = {



Récupérer les coefficients sur le site en ligne et les copier ici
 ....
};

#endif
```

Dans main.cpp

```
#include "mbed.h"
#include "filtre.h"  // http://t-filter.engineerjs.com/

AnalogOut sortie(PA_4);
AnalogIn  entree(A0);
DigitalOut led(LED1);
Serial pc(USBTX,USBRX);
FirFilter monfiltre;

int main() {
   double e,s;

 while(1)
    {
        // temps de calcul mesure 150uS
            e=entree.read();
            monfiltre.put(e);
            s=monfiltre.get();
            sortie.write(s);
            led=0;
            wait_us(350); // fe#2000Hz
            led=1;
    }
}
```

b) Dessiner l'algorigramme de main.cpp, ainsi que des méthodes put et get de la classe FirFilter.
c) Commenter chaque ligne du programme

Résultat attendu



d) Tester le filtre à l'aide d'un GBF et d'un oscilloscope
e) Placer en sortie un filtre analogique RC de fréquence de coupure > 500Hz afin d'éliminer en partie les harmoniques dues à l'échantillonnage
f) Relever la fréquence de coupure du filtre numérique à -3dB.
g) Tracer le diagramme de Bode du gain pour des fréquences comprises entre 10Hz et 100Hz