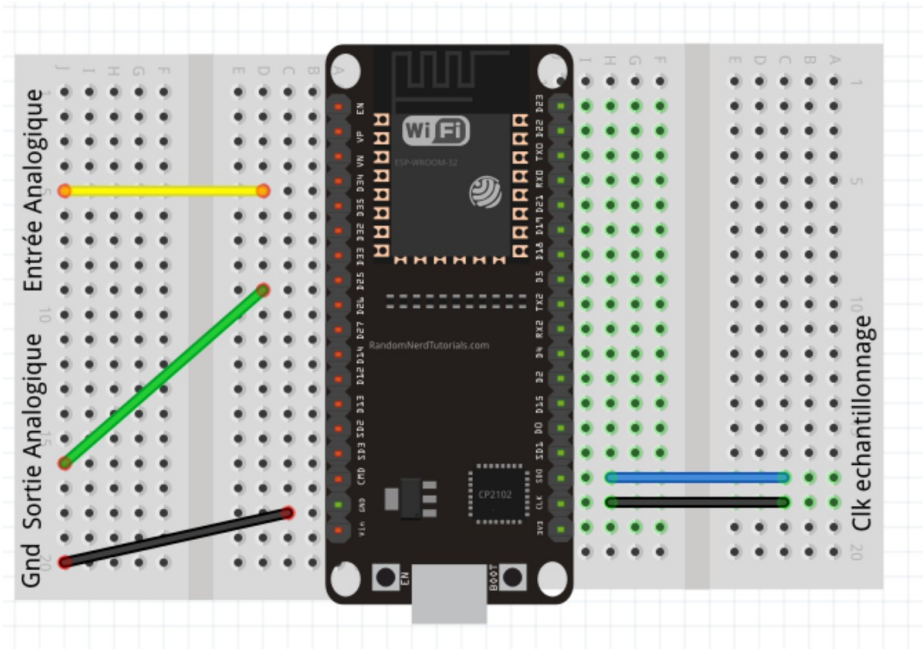
Filtrage Numérique

1 Introduction

Les applications embarquées dans les microcontrôleurs traitent des signaux provenant de différents capteurs (des capteurs de lumière, de températures, etc). Pour extraire des informations de ces signaux et déclencher des actions, ces signaux peuvent nécessiter un filtrage numérique. Par exemple, pour un capteur qui délivre un signal en principe lentement variable mais présentant un bruit important, il est préférable d'échantillonner le bruit puis de procéder à un filtrage passe-bas numérique. Un autre exemple courant est le filtre intégrateur ou le filtre dérivateur.

2 Premier montage

Le montage consiste à connecter une entrée analogique sur un GBF et une sortie analogique sur un oscilloscope. La sortie clk échantillonnage permet de contrôler la fréquence d'échantillonnage.

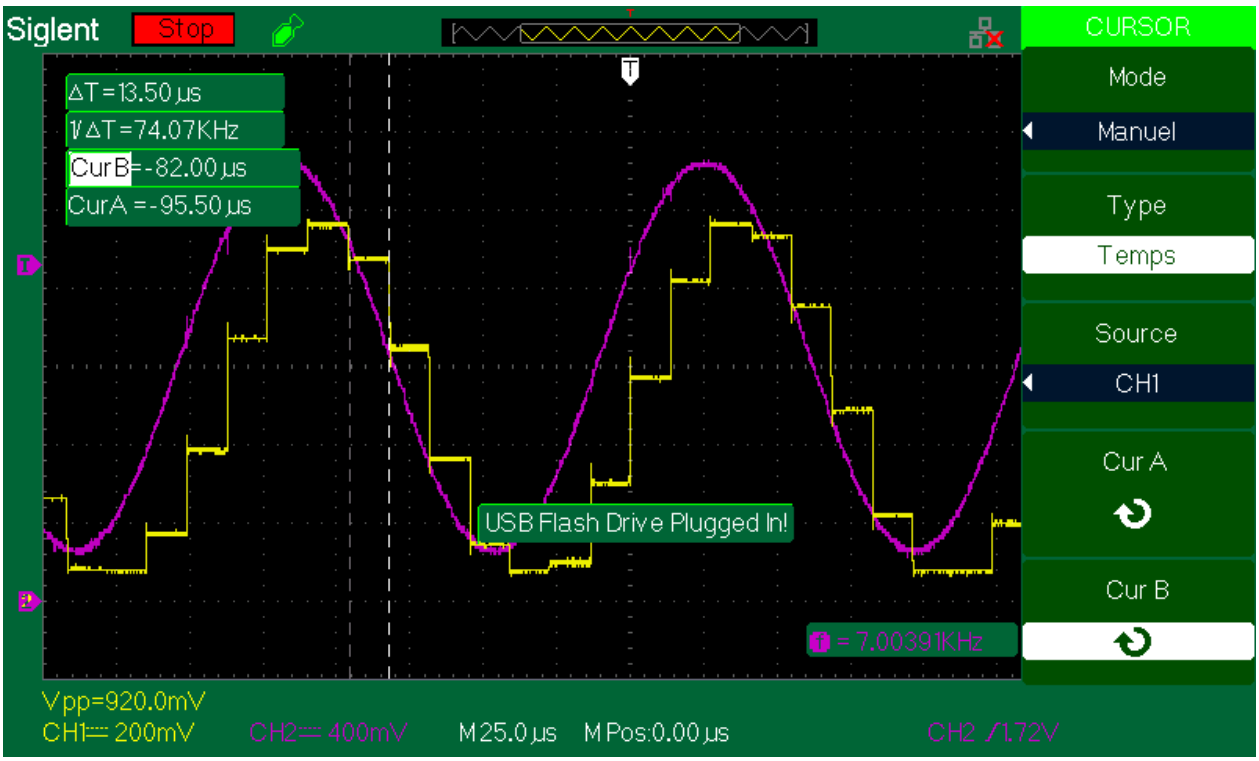


Le premier programme permettra d'estimer la période minimale d'échantillonnage Te.

$$y_n = x_n$$

```
#include <driver/dac.h>
#include <Arduino.h>

#define InAna 34

int x;
int y;

void setup() {
    dac_output_enable(DAC_CHANNEL_1);
```

```
}
void loop() {
    x = analogRead(InAna)/16;
    dac_output_voltage(DAC_CHANNEL_1,x );
}
```



On observe le signal d'entrée analogique (en violet) , c'est un signal sinusoïdale fréquence 7kHz. Le signal de sortie du convertisseur (en jaune).

La période d'échantillonnage est mesurée à **13,5 us**

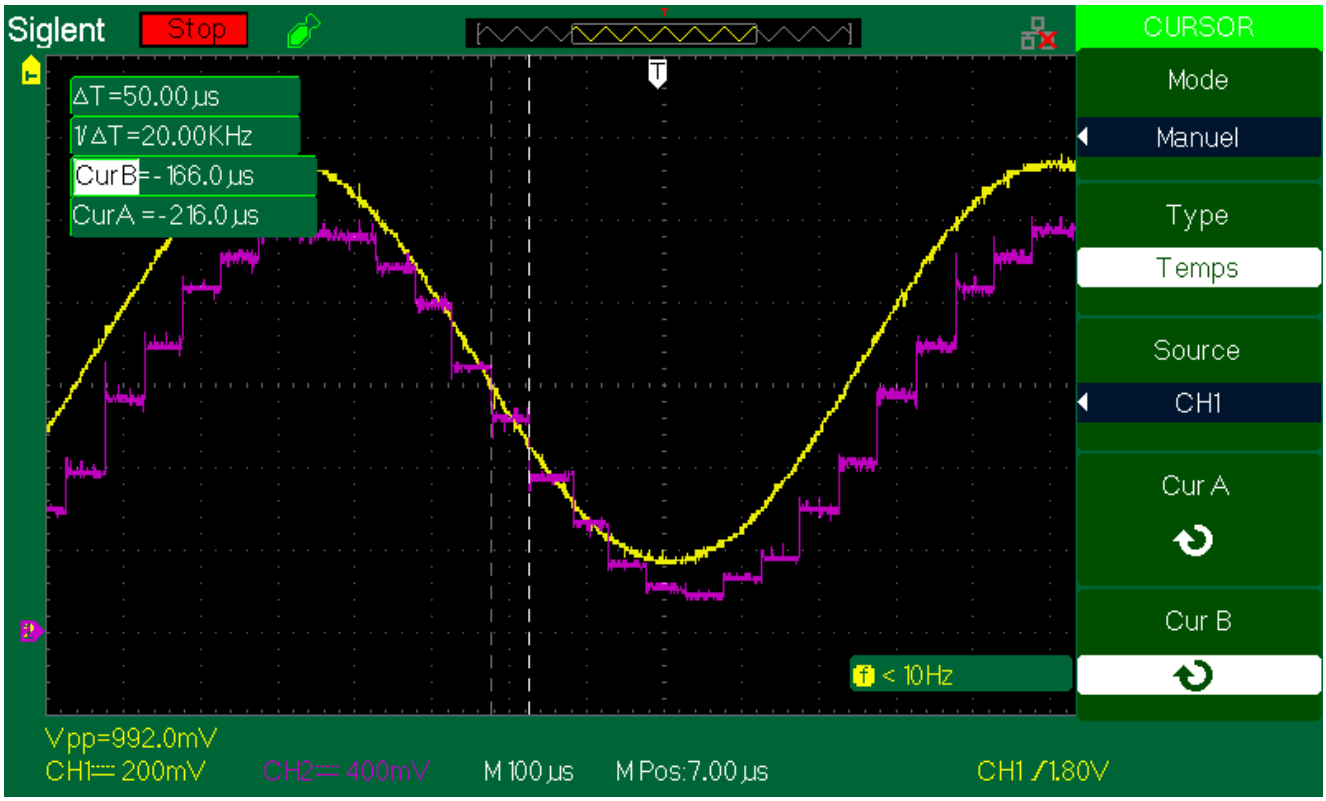
3 Contrôler la période d'échantillonnage

Pour contrôler avec précision la période d'échantillonnage il est nécessaire d'utiliser un timer matériel **hw_timer_t**, pour déclencher périodiquement une interruption. Cette interruption est attaché à la fonction **onTimer()**. Deux variables globales **interruptCounter** et **x** sont utilisées dans cette fonction lancée à chaque interruption. La variable x prend la valeur lue sur l'entrée analogique, interruptCounter prend la valeur 1 puis dans la boucle principale loop(), lorsqu'elle est prise en compte est remise à zéro . L'accès en écriture à la variable **interruptCounter** est protégé par un mutex **timerMux.**

3-1 Le programme complet Te = 50 µs

```
#include <driver/dac.h>
#define LED 15
#define InAna 34

volatile int interruptCounter;
volatile int x;


hw_timer_t * timer = NULL;
portMUX_TYPE timerMux = portMUX_INITIALIZER_UNLOCKED;

void IRAM_ATTR onTimer() {
  portENTER_CRITICAL_ISR(&timerMux);
  interruptCounter++;
  x = analogRead(InAna)/16;
  portEXIT_CRITICAL_ISR(&timerMux);
  digitalWrite(LED, digitalRead(LED) ^ 1);
}

void setup() {

  Serial.begin(9600);
  pinMode(LED, OUTPUT);
  dac_output_enable(DAC_CHANNEL_1);
  timer = timerBegin(0, 80, true);
  timerAttachInterrupt(timer, &onTimer, true);
  timerAlarmWrite(timer, 50, true);
  timerAlarmEnable(timer);
}

void loop() {
  int xn;
  if (interruptCounter > 0) {
    portENTER_CRITICAL(&timerMux);
    interruptCounter--;
    xn = x;
    portEXIT_CRITICAL(&timerMux);
    dac_output_voltage(DAC_CHANNEL_1,xn);
  }
}
```

3-2 Les signaux d’entrée et de sortie



On vérifie bien que la période d’échantillonnage est de 50 µs.

Signal d’entrée : fréquence 1Khz, amplitude 1V, offset 0,65V

4 Filtre passe-bas du premier ordre

3-1 Filtre passe bas du premier ordre τ = 2 Te

$$y(n) = 0{,}333\ x(n) + 0{,}666\ y(n-1)$$

Le programme correspondant pour esp32

```
#include <driver/dac.h>
#include <Arduino.h>

#define InAna 34

short x[2];
short y[2];
unsigned char n;

void setup() {
    dac_output_enable(DAC_CHANNEL_1);
    n = 1;
    y[n-1] = 0;

}
void loop() {
    x[n] = analogRead(InAna)/16;
    y[n] = (x[n] + 2 * y[n-1])/3;   // équation de récurence passe bas 1er ordre
    dac_output_voltage(DAC_CHANNEL_1, y[n]);
    y[n-1] = y[n];                  // yn-1 <- yn
}
```

Réponse indicielle : τ = 2 Te = 28us

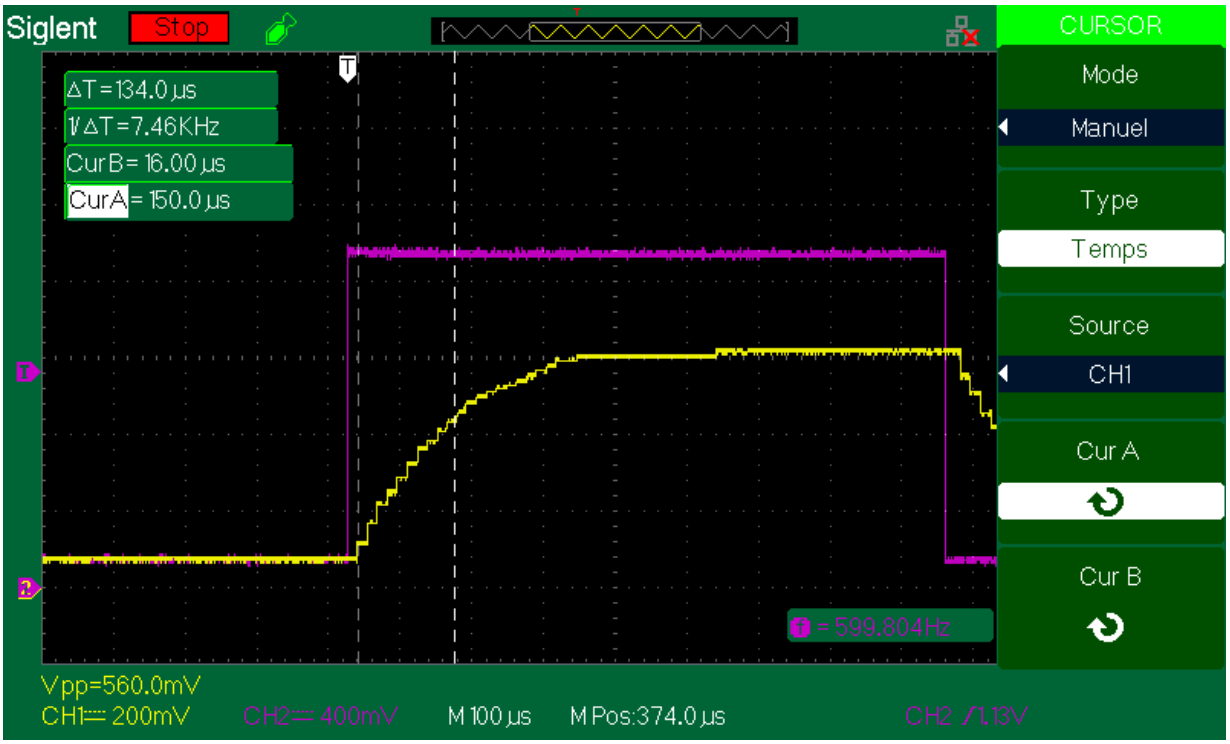
3-2 Filtre passe bas du premier ordre τ = 10 Te

$$y(n) = (x(n) + 10 \times y(n-1)) / 11$$

Le programme correspondant pour esp32

```
#include <driver/dac.h>
#include <Arduino.h>

#define InAna 34

short x[2];
short y[2];
unsigned char n;

void setup() {
    dac_output_enable(DAC_CHANNEL_1);
    n = 1;
    y[n-1] = 0;

}
void loop() {
    x[n] = analogRead(InAna)/16;
    y[n] = (x[n] + 10 * y[n-1])/11;  // équation de récurence passe bas 1er ordre
    dac_output_voltage(DAC_CHANNEL_1, y[n]);
    y[n-1] = y[n];                    // yn-1 <- yn
}
```

Réponse indicielle : τ = 10 Te = 134 us



5 Filtre passe-bas du 2ième ordre

On se propose de synthétiser la transmittance

$$H(p)=\frac{Y(p)}{X(p)}=\frac{\omega_0^2}{\omega_0^2+2m\omega_0 p+p^2}$$

avec

- $\omega_0 = 1000\pi$ *rad/s* $T_0 = 2$ *ms* soit $f_0 = 500$ Hz
- $m = 0{,}1$
- $T_e = 0{,}2$ *ms ou 200 µs soit* $f_e = 5000$ Hz

Filtre de pulsation propre $T_0 = 2\pi/\omega_0 = 10.Te$

4-1 Par la transformation rectangulaire :

Aussi appelé transformation par équivalence de la dérivation

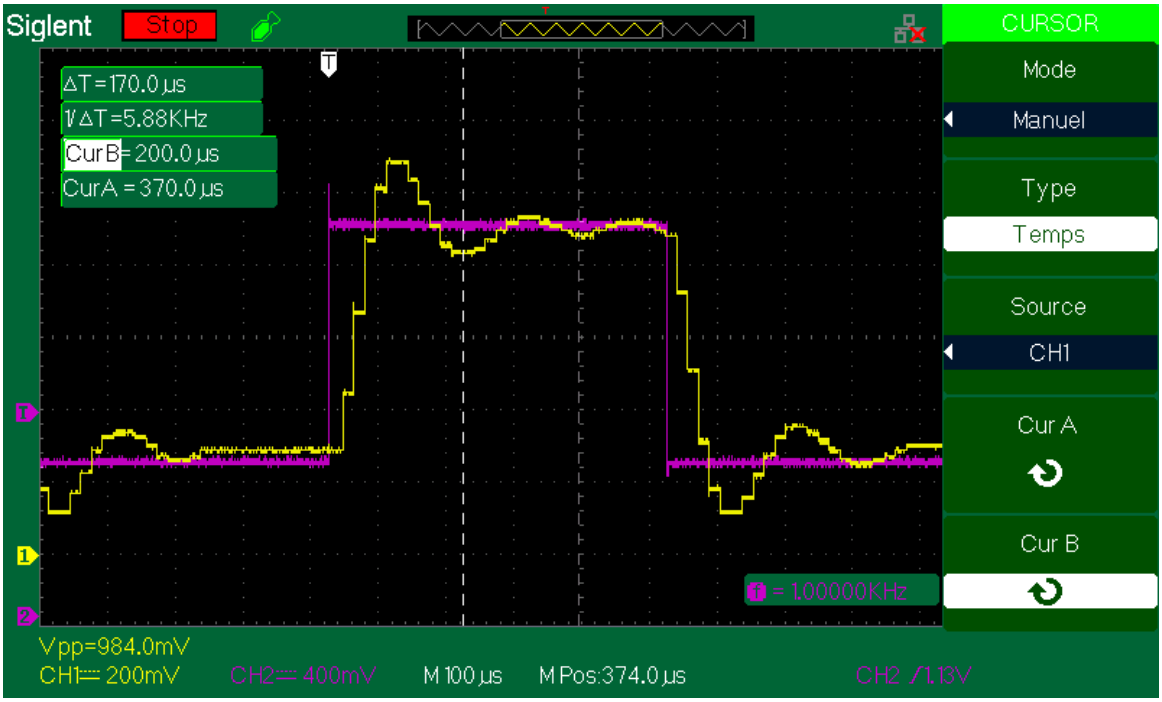
$$p \Leftrightarrow \frac{1-Z^{-1}}{Te}$$

Équation de récurrence :

$$y_n = 0{,}25965\, x_n + 1{,}39805\, y_{n-1} - 0{,}65770\, y_{n-2}$$

```
#include <driver/dac.h>
#include <Arduino.h>

#define InAna 34

short x[3];
short y[3];
unsigned char n;

void setup() {
    dac_output_enable(DAC_CHANNEL_1);
    n = 2;
    y[n-1] = 0;
    y[n-2] = 0;

}
void loop() {
    x[n] = analogRead(InAna)/16;
    y[n] = 0.2596 * x[n] + 1.398 * y[n-1] - 0.6577 * y[n-2];
    dac_output_voltage(DAC_CHANNEL_1, y[n]);
    y[n-2] = y[n-1];
    y[n-1] = y[n];
}
```

période d'échantillonnage : Te=17µs T_0 =170µs GBF : signal carré ~1 kHz



La fréquence échantillonnage n'étant pas suffisamment élevée par rapport à f_0 , le calcul des coefficients par équivalence de la dérivation conduit à une réponse d'une médiocre précision.

4-2 Par la transformation bilinéaire

Aussi appelé transformation par équivalence de l'intégration

$$p \Leftrightarrow \frac{2}{T_e}\frac{1-Z^{-1}}{1+Z-1}$$

Ce qui donne l'équation de récurrence :

$y_n = 0{,}084971 \,.\, (x_n + 2.x_{n-1} + x_{n-2}) + 1.55193.y_{n-1} - 0{,}89181.y_{n-2}$

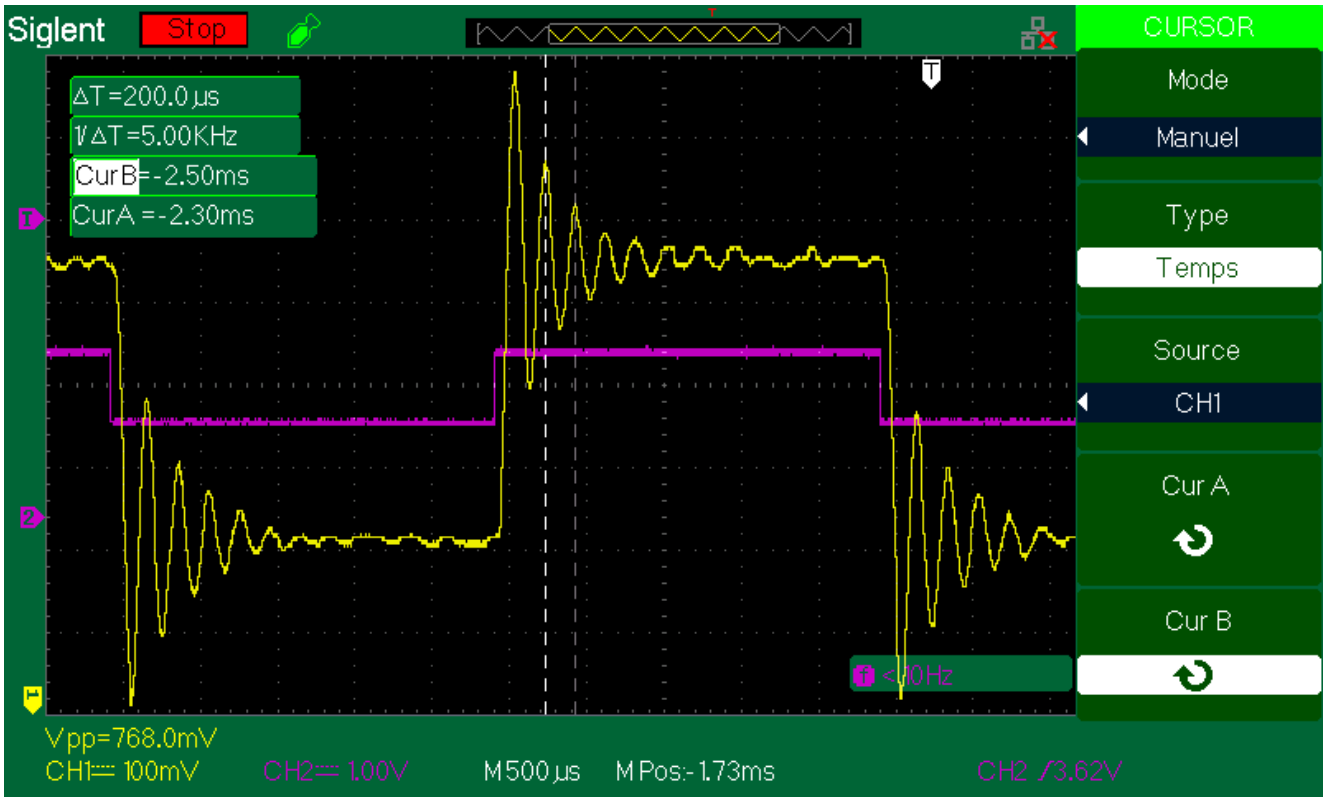
$y_n = 0{,}084971.x_n + 0{,}16994.x_{n-1} + 0{,}084971.x_{n-2} + 1{,}55193.y_{n-1} - 0{,}89181.y_{n-2}$

Le programme correspondant :

```
#include <driver/dac.h>
#include <Arduino.h>

#define InAna 34

float x[3];
float y[3];
unsigned char n;

void setup() {
    dac_output_enable(DAC_CHANNEL_1);
    n = 2;
    y[n-1] = 0;
    y[n-2] = 0;
    x[n-2] = 0;
    x[n-1] = 0;

}
void loop() {
    x[n] = analogRead(InAna)/16;
    y[n]  = 0.08497 * x[n];
    y[n] += 0.16994 * x[n-1];
    y[n] += 0.08497 * x[n-2];
    y[n] += 1.5519 * y[n-1];
    y[n] += -0.8918 * y[n-2];
    dac_output_voltage(DAC_CHANNEL_1, y[n]);
    y[n-2] = y[n-1];
    y[n-1] = y[n];
    x[n-2] = x[n-1];
    x[n-1] = x[n];
}
```

Période d’échantillonnage : Te =20µs T_0 =200µs

GBF : signal carré ~200Hz ampl = 0,43V offset = 0,75



On vérifie la réponse à un échelon d’un système numérique du 2ième ordre (voir cours) pour m=0,1.

On peut comparer les deux transformations (la transformation bilinéaire est plus performante)

6 Calcul des coefficients

http://www.iowahills.com/Index.html