

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 utilisés dans les correcteurs.

Un filtre numérique est un algorithme de calcul qui fait correspondre à une suite d'échantillons $x(n)$ d'entrée à une autre suite d'échantillons $y(n)$ de sortie.

Le calcul de la valeur de la sortie $y(n)$ à l'instant n est effectué à partir de

- des échantillons de sorties calculés précédemment y_{n-1} y_{n-2} y_{n-3}
- des échantillons obtenus précédemment de l'entrée x_{n-1} x_{n-2} x_{n-3}
- de la nouvelle valeur qui vient d'être obtenue de l'entrée x_n

L'échantillon est calculé avec la formule dont la forme générale est la suivante :

$$y_n = a_1 \cdot y_{n-1} + a_2 \cdot y_{n-2} + \dots + a_p \cdot y_{n-p} + b_0 \cdot x_n + b_1 \cdot x_{n-1} + \dots + b_q \cdot x_{n-q}$$

Les a_i et b_i sont des coefficients qui définissent la nature du filtre.

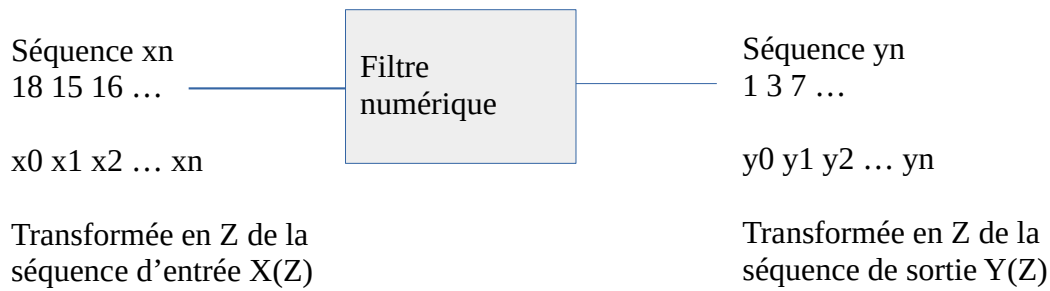
1-1 Filtres non récurrents ou à réponse impulsionnelle finie FIR

Ce sont des filtres pour lesquels la sortie ne dépend que des entrées et pas des sorties précédentes. Leur réponse à une impulsion s'annule au bout d'un certain temps. Ils n'ont pas d'équivalent analogique.

1-2 Filtres récurrents ou à réponse impulsionnelle infinie IIR

Ce sont des filtres pour lesquels la sortie dépend des entrées et des sorties précédentes. Leur réponse à une impulsion s'annule au bout d'un temps infini.

1-3 Transmittance en Z d'un filtre



La transmittance en Z du filtre est alors définie par

$$T(Z) = \frac{Y(Z)}{X(Z)}$$

Puisque $X(Z)$ et $Y(Z)$ sont des polynômes contenant des puissances négatives de Z , il en est de même pour $T(Z)$.

1-4 Passage de $T(Z)$ à la relation de récurrence

Pour passer de la transmittance en Z à la relation de récurrence il suffit d'appliquer les équivalences suivantes :

$$X_{n-i} \Leftrightarrow X(z) \cdot z^{-i} \quad z^{-i} \text{ est l'opérateur retard}$$

$$Y_{n-i} \Leftrightarrow Y(z) \cdot z^{-i}$$

Pour la relation générale suivante

$$y_n = a_1 \cdot y_{n-1} + a_2 \cdot y_{n-2} + \dots + a_p \cdot y_{n-p} + b_0 \cdot x_n + b_1 \cdot x_{n-1} + \dots + b_q \cdot x_{n-q}$$

La transmittance peut s'écrire

$$Y(z) = a_1 \cdot Y(z) \cdot z^{-1} + a_2 \cdot Y(z) \cdot z^{-2} + \dots + a_p \cdot Y(z) \cdot z^{-p} + b_0 \cdot X(z) + b_1 \cdot X(z) \cdot z^{-1} + \dots + b_q \cdot X(z) \cdot z^{-q}$$

puis en factorisant

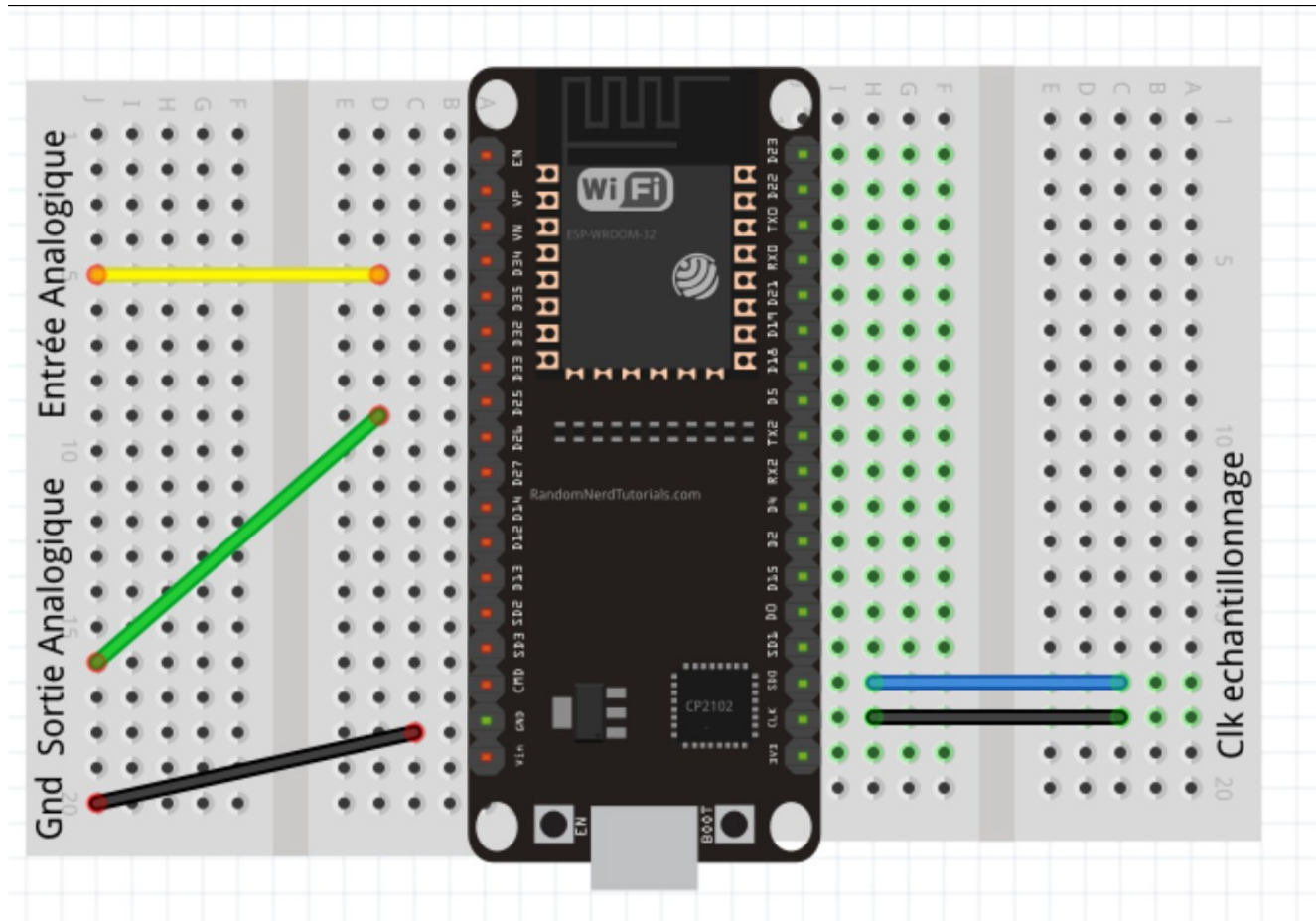
$$Y(z) \cdot (1 - a_1 \cdot z^{-1} - a_2 \cdot z^{-2} - \dots - a_p \cdot z^{-p}) = X(z) \cdot (b_0 + b_1 \cdot z^{-1} + \dots + b_q \cdot z^{-q})$$

Mise en forme

$$T(z) = \frac{Y(z)}{X(z)} = \frac{(b_0 + b_1 \cdot z^{-1} + \dots + b_q \cdot z^{-q})}{(1 - a_1 \cdot z^{-1} - a_2 \cdot z^{-2} - \dots - a_p \cdot z^{-p})}$$

2 Premier montage

Le montage consiste à connecter l'entrée analogique sur un GBF et la 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 T_e .

$$y_n = b_0 \cdot x_n$$

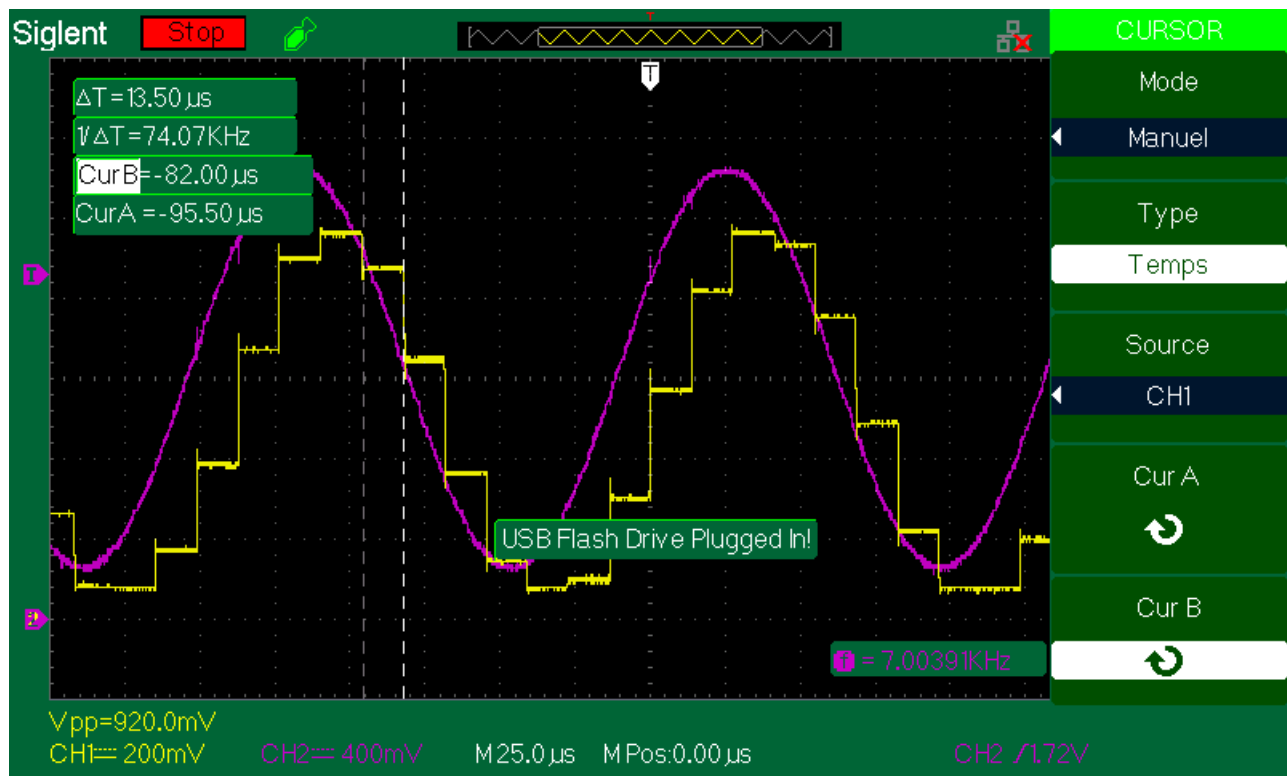
```
#include <driver/dac.h>
#include <Arduino.h>

#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

L'ESP32 possède deux groupes de temporisateurs, chacun avec deux temporisateurs matériels à usage général. Tous les temporisateurs sont basés sur des compteurs 64 bits et des prédiviseurs 16 bits. Le prédiviseur est utilisé pour diviser la fréquence du signal de base (80 MHz), qui est ensuite utilisée pour incrémenter / décrémenter le compteur de temporisation.

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 .

Enfin, nous devons déclarer une variable de type **portMUX_TYPE** , que nous utiliserons pour assurer la synchronisation entre la boucle principale et l'ISR, lors de la modification d'une variable partagée. L'accès en écriture à la variable **interruptCounter** est protégé par un mutex **timerMux**.

Nous initialiserons le timer avec un appel à la fonction **timerBegin** , qui retournera un pointeur vers une structure de type **hw_timer_t** , qui est celle de la variable globale **timer** que nous avons déclarée.

```
timer = timerBegin(0, 80, true);
```

En entrée, cette fonction reçoit le numéro du temporisateur que nous voulons utiliser (de 0 à 3, car nous avons 4 temporisateurs matériels), la valeur du prédiviseur et un indicateur indiquant si le compteur doit compter (**true**) ou décroître (**false**).

Ainsi, si nous divisons par 80 (en utilisant 80 comme valeur de prédiviseur), nous obtiendrons un signal avec une fréquence de 1 MHz qui incrémentera le compteur de la minuterie 1 000 000 fois par seconde. le compteur sera incrémenté à chaque microseconde.

Nous devons lier le temporisateur à une fonction de gestion, qui sera exécutée lorsque l'interruption sera générée. Cela se fait avec un appel à la fonction **timerAttachInterrupt** .

```
timerAttachInterrupt(timer, &onTimer, true);
```

Ensuite, nous utiliserons la fonction **timerAlarmWrite** pour spécifier la valeur du compteur pour laquelle l'interruption du temporisateur sera générée. Ainsi, cette fonction reçoit comme première entrée le pointeur vers le temporisateur, comme deuxième la valeur du compteur pour laquelle l'interruption doit être générée, et comme troisième un indicateur indiquant si le temporisateur doit se recharger automatiquement lors de la génération de l'interruption.

```
timerAlarmWrite(timer, 50, true);
```

Nous terminons notre configuration en activant la minuterie avec un appel à la fonction **timerAlarmEnable** , en passant en entrée notre variable de minuterie.

```
timerAlarmEnable(timer);
```

Le code ISR

La routine de service d'interruption doit être une fonction qui renvoie void et ne reçoit aucun argument.

Notre fonction sera aussi simple que d'incrémenter le compteur d'interruptions qui signalera à la boucle principale qu'une interruption s'est produite. Cela se fera dans une section critique, déclarée avec les macros `portENTER_CRITICAL_ISR` et `portEXIT_CRITICAL_ISR`, qui reçoivent toutes les deux comme paramètres d'entrée l'adresse de la variable globale `portMUX_TYPE` que nous avons déclarée comme variable globale.

Mise à jour: La routine de gestion des interruptions doit avoir l'attribut `IRAM_ATTR`, pour que le compilateur place le code dans IRAM.

3-1 Le programme complet $T_e = 50 \mu s$

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

volatile int interruptCounter;
volatile int x;

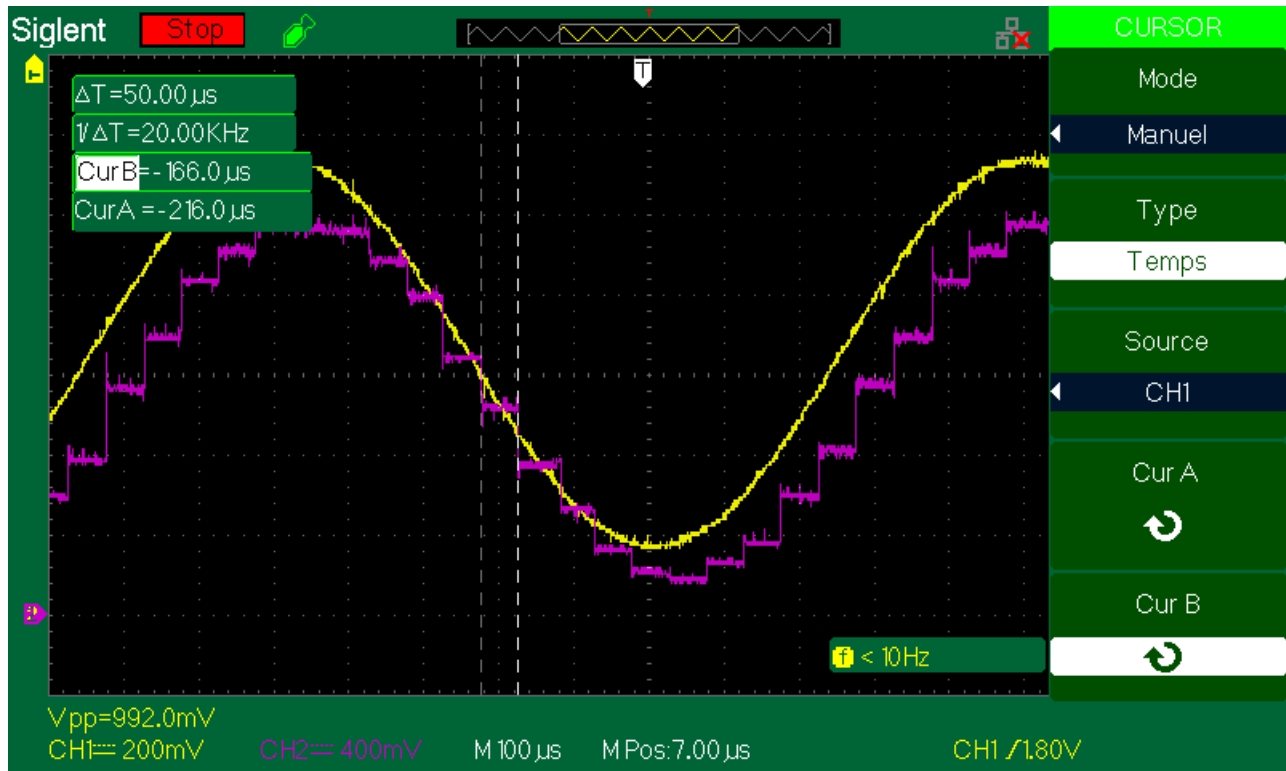
hw_timer_t * timer = NULL;
portMUX_TYPE timerMux = portMUX_INITIALIZER_UNLOCKED;

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 $50 \mu s$.

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 $\tau = 2 T_e$

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

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 récurrence passe bas 1er ordre
    dac_output_voltage(DAC_CHANNEL_1, y[n]);
    y[n-1] = y[n]; // yn-1 <- yn
}
```

Réponse indicielle : $\tau = 2 T_e = 28\mu s$



3-2 Filtre passe bas du premier ordre $\tau = 10 T_e$

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

Le programme correspondant pour esp32

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

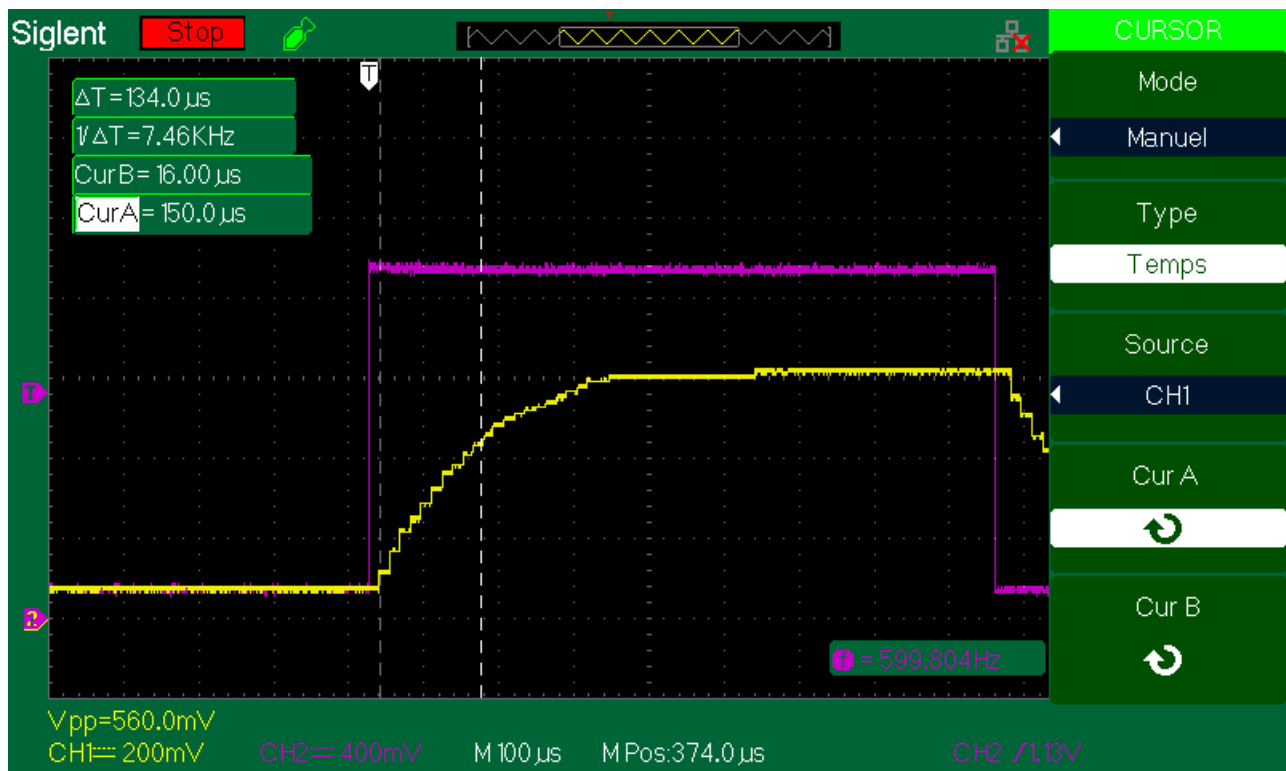
#define InAna 34

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 récurrence passe bas 1er ordre
    dac_output_voltage(DAC_CHANNEL_1, y[n]);
    y[n-1] = y[n];                // yn-1 <- yn
}
```

Réponse indicielle : $\tau = 10 T_e = 134 \mu s$



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

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

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

4-1 Par la transformation rectangulaire :

Aussi appelé transformation par équivalence de la dérivation

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

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 : $T_e = 17\mu\text{s}$ $T_0 = 170\mu\text{s}$ GBF : signal carré ~1 kHz

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 :

$$y_n = 0,084971 \cdot (x_n + 2x_{n-1} + x_{n-2}) + 1,55193y_{n-1} - 0,89181y_{n-2}$$

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

Le programme correspondant :

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

#define InAna 34

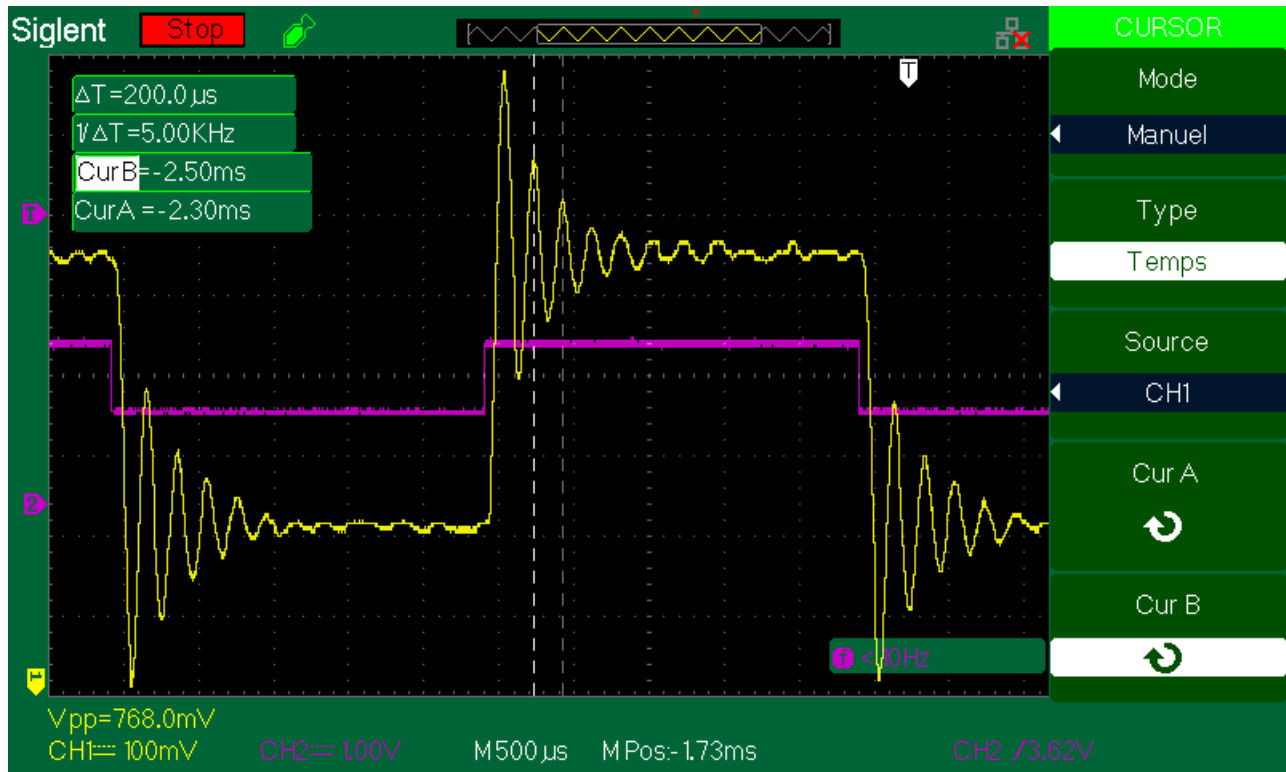
float x[3];
float y[3];
unsigned char n;

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 : $T_e = 20\mu s$ $T_0 = 200\mu s$

GBF : signal carré $\sim 200\text{Hz}$ $\text{ampl} = 0,43\text{V}$ $\text{offset} = 0,75$



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>