3 mars 2025

Expérience : 3. Systèmes discrets

Opérateur de retard et modulation à largeur d'impulsion (MLI ≡ PWM)

1 Matériel

— 1 × générateur de fonction
— 1 × oscilloscope
— 1 × Arduino Uno/Genuino
— 1 × Olimexino 85
— 1 × ordinateur avec Processing
— 1 × câble USB qui fait office de ligne série de transmission
— 1 × résistance de 220[kΩ]
— 3 × capacités $C = 68, 6.8$ et 100[nF] respectivement.

2 Objectif

L'objectif consiste en la réalisation de l'opérateur de retard q^{-1} et d'un échantillonneur-bloqueur (Sample and Hold : S/H). Il s'agit également de présenter les sorties dites analogiques de l'Arduino Uno/Genuino et de l'Olimexino 85. Ce sont les sorties dites analogiques des microcontrôleurs AVR. Pour obtenir une valeur analogique la sortie est modulée en largeur d'impulsion (MLI=PWM) et c'est la valeur moyenne de ce signal qui donne la valeur analogique. Un filtre RC produit ainsi le signal analogique de sortie.

3 Schéma

— L'entrée #2 de l'Olimexino reçoit le signal du générateur.
— La sortie #1 pilote la résistance du circuit RC.
— L'Arduino prend la valeur A0 qui provient du générateur de fonction. Il reçoit également en A1 la sortie du filtre RC. Ces deux valeurs sont envoyées sur la ligne série TTY à l'ordinateur central. L'Arduino remet à jour et échantillonne les deux signaux A0 et A1 de telle sorte à pouvoir opérer à 9600 bauds avec l'ordinateur central.

— L'ordinateur central ne fait que représenter graphiquement les deux suites de valeurs reçues de la ligne série. Il opère donc à 9600 bauds.

— C'est dans l'Olimexino 85 que l'opérateur de retard q^{-1} est réalisé. La période d'échantillonnage est totalement découplée de la transmission série entre l'Arduino et l'ordinateur central. Elle est garantie par un délai dans la boucle principale du code de l'Olimexino (cf. ci-dessous).

— Un oscilloscope est utilisé pour examiner la nature du signal analogique avant le filtre RC et après le filtre RC et également le signal du générateur.

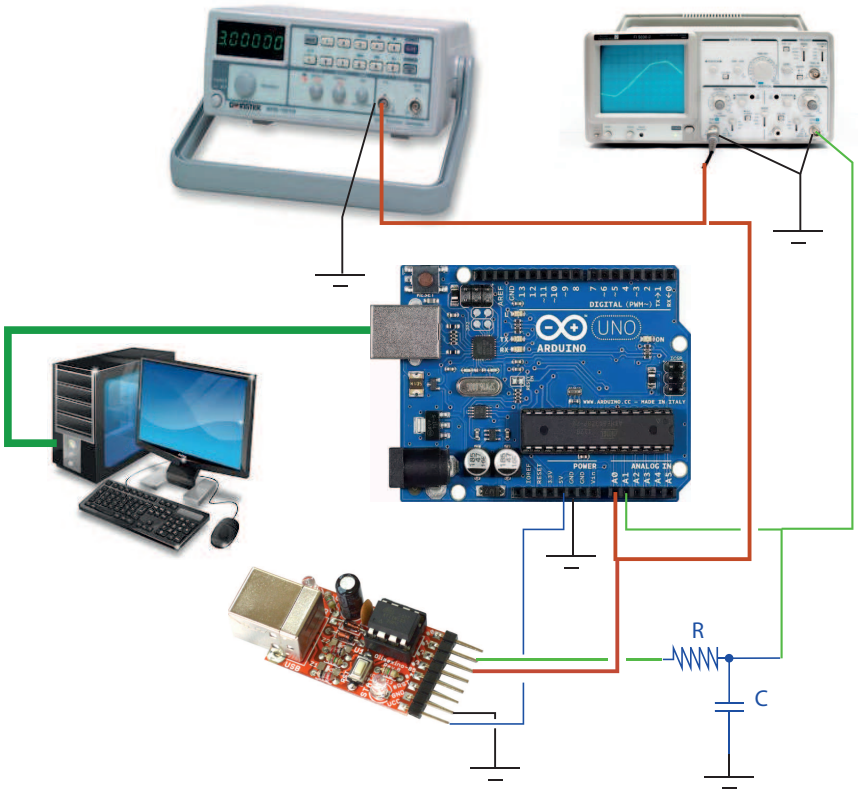

FIGURE 1 – Schéma général de l'expérience.

4 Code de l'Arduino

La période d'échantillonnage n'est pas explicitement garantie. Elle est fixée indirectement par la cadence de la ligne série. Celle-ci est fixée à 9600 Bauds.
L'Arduino se comporte uniquement comme oscilloscope, c.-à-d. qu'il lit les entrées A0 et A1 et transmet ces deux valeurs sur la ligne série. On utilise une chaîne de caractères pour transmettre ces deux valeurs en un coup avec un espace unique qui sépare les valeurs.

```
int i=0;

void setup() {
  Serial.begin(9600);
}

void loop() {
  int entreeX = analogRead(A0);
  int entreeY = analogRead(A1);
  char buffer [50];
  i=sprintf (buffer, "%d %d\n" , entreeX, entreeY);
      for(int l= 0; l<=i; l++)
      Serial.print(buffer[l]);
 delay(1);
}
```

5 Code de l'Olimexino 85

5.1 Code pour le PWM standard à ≈ 500 [Hz]

```
int ADC_k = 0; // valeur a l'echantillon k
int ADC_k1 = 0; // valeur a l'echantillon k-1;

// Mise en place
void setup() {
  pinMode(1, OUTPUT);    // broche 1 comme sortie
  analogWrite(1, LOW);
}

void loop() {
  analogWrite(1, (ADC_k1/4));     // PWM de la valeur de from 0 to 255 (max) sur la broche 1
                                  // qui est egalement la LED
  ADC_k = analogRead(1);    // lecture de la broche 2. La valeur est comprise entre 0..1023
  delay(400); // delai  pour implementer la periode d'echantillonnage
  ADC_k1 = ADC_k; // l'operateur 'retard' est code
}

/* Attention : la broche physique 2 de l'Olimexino est indexee par l'index 1.
 * La broche physique 2 correspond a ADC1, et donc index 1 egalement, ainsi on ecrit
 * analogRead(1) pour lire la broche 2
  */
```

5.2 Code pour le PWM à haute fréquence ≈ 30 [kHz]

```
int ADC_k = 0; // valeur a l'echantillon k
int ADC_k1 = 0; // valeur a l'echantillon k-1;

// Mise en place
void setup() {

  pinMode(1, OUTPUT);    // broche 1 comme sortie

// Configuration du mode MLI haute frequence 30 kHz.
// Nous n'entrons pas dans les details de cette sequencen d'instruction
// se referer a la fiche technique de l'ATTiny.
    TCCR0A = 2<<COM0A0 | 2<<COM0B0 | 3<<WGM00;
    TCCR0B = 0<<WGM02 | 1<<CS00;
    TCCR1 = 0<<PWM1A | 0<<COM1A0 | 1<<CS10;
    GTCCR = 1<<PWM1B | 2<<COM1B0;

  analogWrite(1, LOW);
}

void loop() {

  analogWrite(1, (ADC_k1/4));     // MLI (PWM) de la valeur de from 0 to 255
                                  // qui est egalement la LED
```

```
  ADC_k = analogRead(1);    // lecture de la broche 2. La valeur est comprise entre 0..1023
  delay(20000); // delai  pour implementer la periode d'echantillonnage 0.2 [s]
  ADC_k1 = ADC_k; // l'operateur 'retard' est code
}

/* Attention : la broche physique 2 de l'Olimexino est indexee par l'index 1.
 * La broche physique 2 correspond a ADC1, et donc index 1, ainsi on ecrit
 * analogRead(1) pour lire la broche 2
  */
```

6 Code dans Processing

```
mport processing.serial.*;

Serial myPort;        // Le port serie
int xPos = 0;         // Position horizontale dans le graphique
float inY1 = 300.0;
float inY2 = 300.0;
float oldY1;
float oldY2;
String[] list;

void setup () {
  // taille de la fenetre:
  size(900, 600);
  //size(1800,900); //en auditoire
  // Ouverture de la ligne serie et configuration de celle-ci pour 9600 Bauds
  myPort = new Serial(this, Serial.list()[1], 9600);
  myPort.bufferUntil('\n');
  // on colorie le fond de l'ecran en noir
  background(0);
}

void draw () {
  // on dessine le signal en dessinant seulement entre deux echantillons:
  strokeWeight(4);
  stroke(255,255, 0);
  line(xPos-2, height-oldY1, xPos, height-inY1);
  stroke(0,255,255);
  line(xPos-2, height-oldY2, xPos, height-inY2);
  oldY1 = inY1;
  oldY2 = inY2;
  // si fin de l'ecran, on recommence et on efface
  if (xPos >= width) {
    xPos = 0;
    background(0);
  } else {
    // increment de deux pixels pour chaque echantillon:
    xPos+=2;
```

```
  }
}

void serialEvent (Serial myPort) {
  String inString = myPort.readString();
  String[] valueArray = split(inString,' ');
  if (inString != null) {
    inY1=float(trim(valueArray[0]))+40.0;
    inY2=float(trim(valueArray[1]))+40.0;
    inY1 = map(inY1, 0, 1023, 0, height);
    inY2 = map(inY2, 0, 1023, 0, height);
  }
}
```

7 Expérience et résultats

L'expérience consiste à coder l'Olimexino 85 afin que celui-ci réalise un retard après la saisie du signal analogique par l'échantillonneur. La sortie est un signal à modulation de largeur d'impulsion (MLI) d'un signal issu d'un bloqueur situé en amont du modulateur MLI. Un filtre RC externe élimine les oscillations et lisse le signal MLI pour le transformer en un signal analogique. Un échantillonneur-bloqueur avec une représentation de 0 à 1023 valeurs discrètes est ainsi réalisé. La valeur 0 correspond à une entrée analogique de 0 [V] et 1023 correspond à une valeur de 5 [V].

7.1 Implémentation du retard

Le retard est obtenu en utilisant une variable interne qui est mise à jour par rapport à la valeur obtenue avant le retard. La période d'échantillonnage est obtenue par l'entremise d'un délai. Nous verrons des méthodes basée sur les interruptions dans les manips ultérieures.
Le système possède une seule condition initiale dans la variable

```
ADC_k1
```

Le code est donné à la Section 5.1 et les instructions essentielles sont résumées ci-dessous :
Initialisation :

```
int ADC_k = 0; // valeur a l'echantillon k
int ADC_k1 = 0; // valeur a l'echantillon k-1;
```

Induction :

```
void loop() {
//...
analogWrite(ADC_k1);
ADC_k = analogRead(1);
delay(400);
ADC_k1 = ADC_k;
//...
}
```

7.2 La sortie analogique par MLI (modulation de largeur d'impulsion)

7.2.1 Fréquence standard ≈ 500 [Hz]

Le générateur envoie un signal sinusoïdal à l'entrée analogique de l'Olimexino.
Un filtre RC lisse la sortie "analogique" de l'Olimexino (lisse le signal MLI) avec comme valeurs

$$R = 220[\mathrm{k}\Omega]$$
$$C = 0.15[\mu\mathrm{F}]$$

Les résultats sont représentés à la figure 2.

7.2.2 Fréquence haute ≈ 30 [kHz]

On modifie le code de l'Olimexino 85 afin d'augmenter la fréquence du MLI en la forçant à ≈ 30 kHz. Cette partie est technique et elle est décrite dans la fiche technique de l'AVR ATTiny. Il s'agit de configurer le temporisateur-compteur de manière adéquate.

7.2.3 Absence de filtre RC

En absence de filtre RC, la modulation en largeur d'impulsion est visible comme en (figure 5), obtenue pour fréquence de modulation de 500 [Hz]. On peut également constater la MLI en haute fréquence en augmentant la résolution temporelle de l'oscilloscope comme en figure 6.

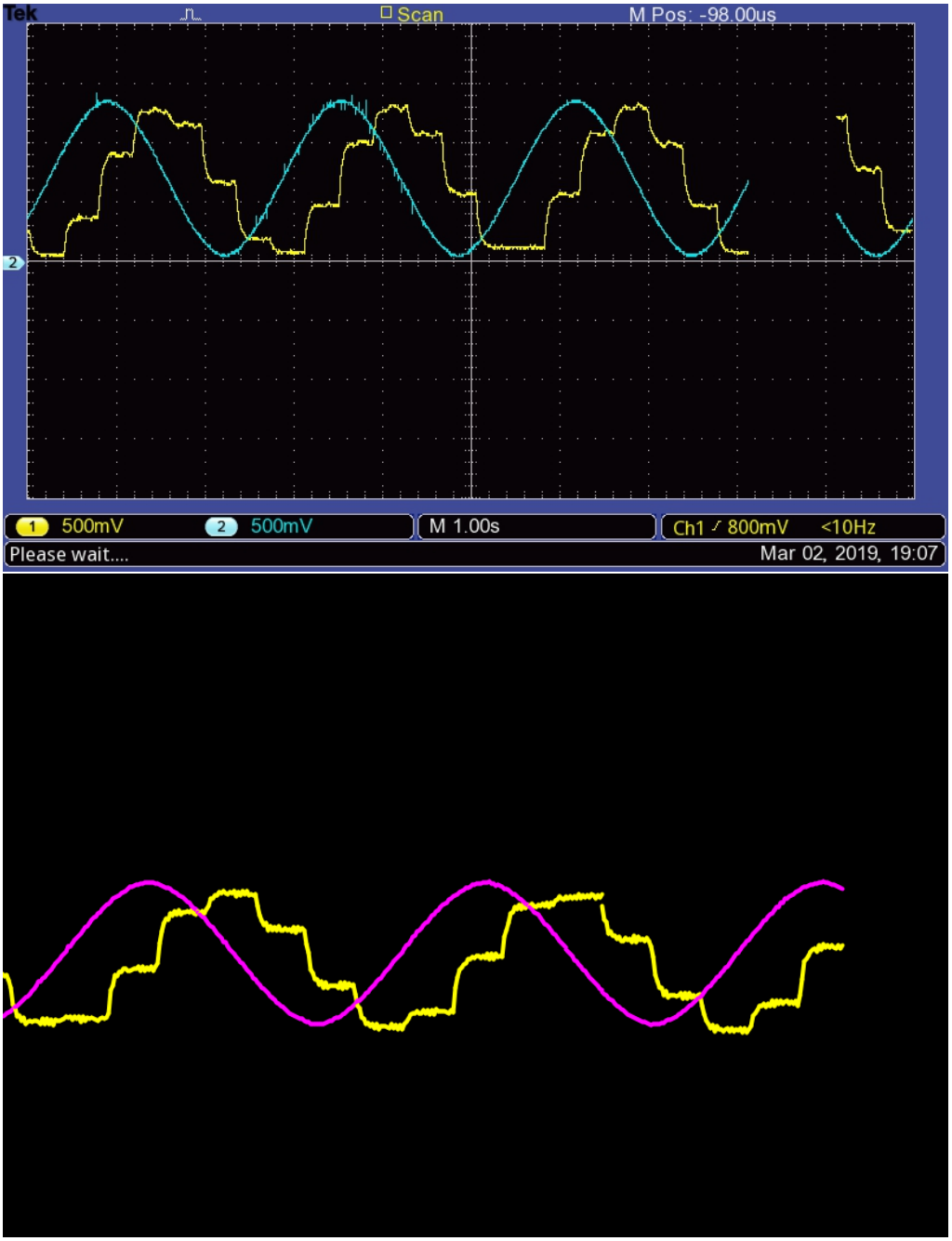

FIGURE 2 – Signal sinusoïdal à la sortie du générateur en bleu (oscilloscope) et en magenta (Processing). En jaune, la sortie du filtre RC avec $R = 220[\text{k}\Omega]$ et $C = 0.15[\mu\text{F}]$. L'Arduino est utilisé en oscilloscope lorsqu'il est couplé à Processing. Le filtre est un peu trop prononcé car les flancs ne sont pas assez raides.

.

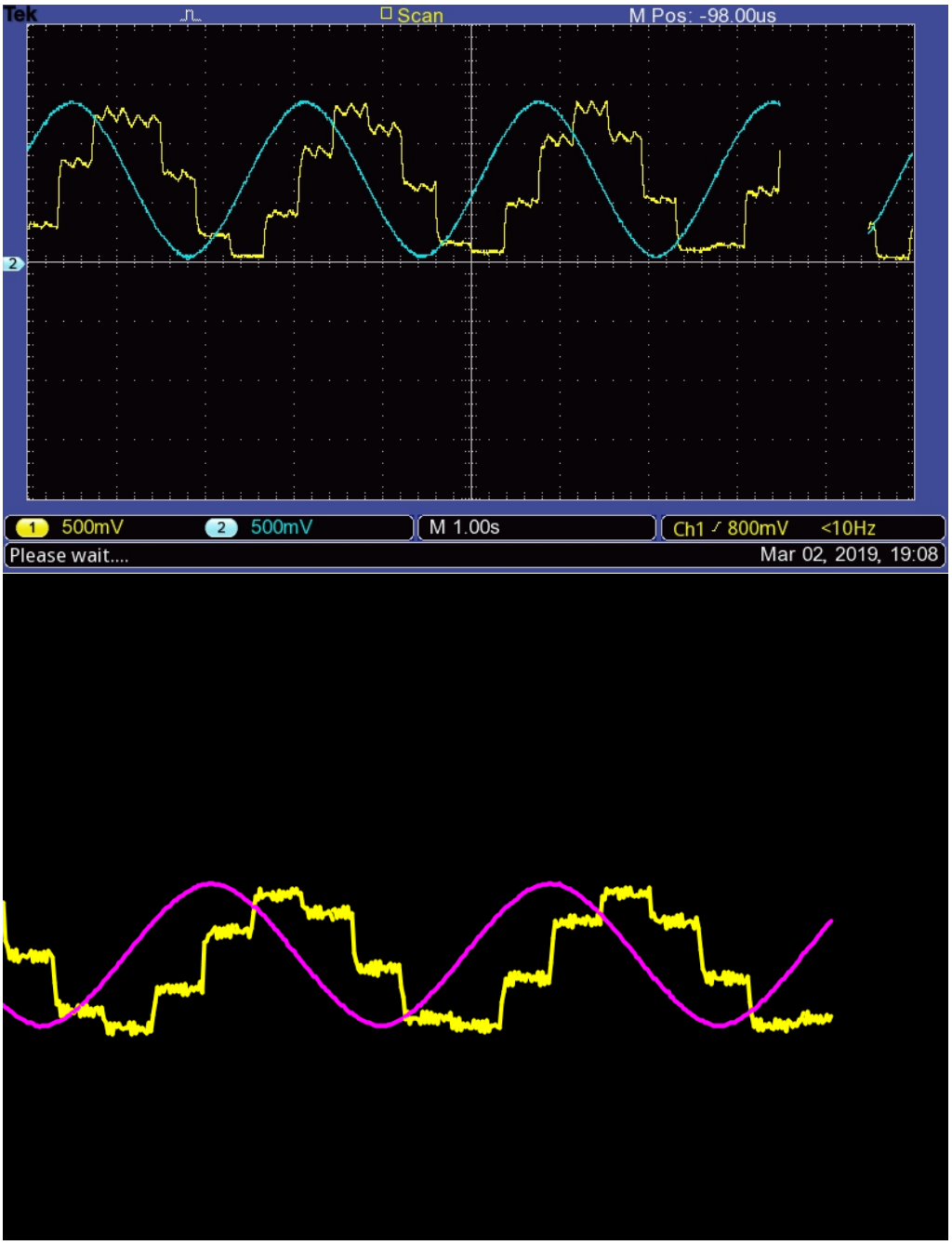

FIGURE 3 – Signal sinusoïdal à la sortie du générateur en bleu (oscilloscope) et en magenta (Processing). En jaune, la sortie du filtre RC avec $R = 220[\text{k}\Omega]$ et $C = 68[\text{nF}]$. L'Arduino est utilisé en oscilloscope lorsqu'il est couplé à Processing. Bien que les flancs soient rapides, le filtre RC n'arrive pas à lisser suffisament le MLI.

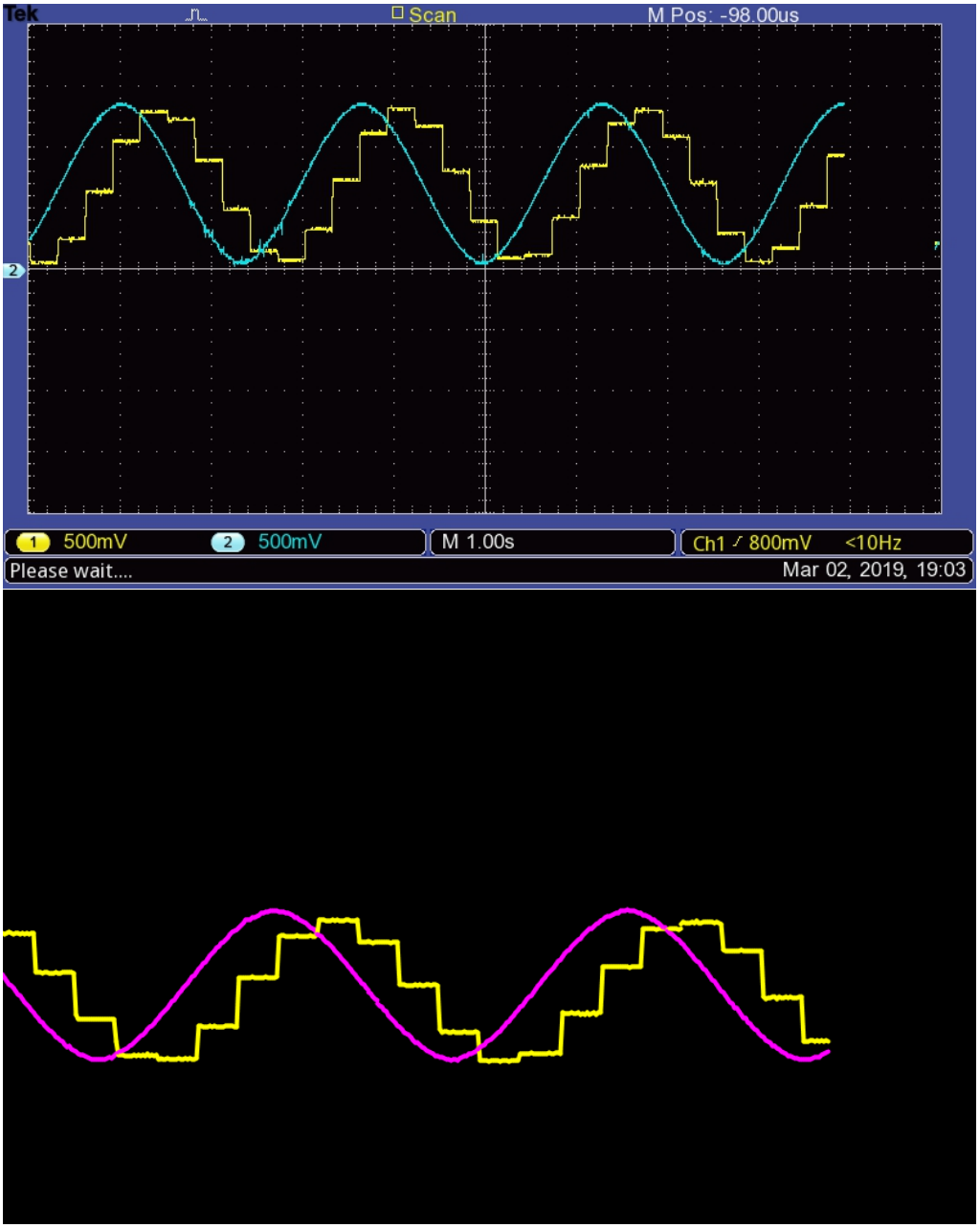

FIGURE 4 – Signal sinusoïdal à la sortie du générateur en bleu (oscilloscope) et en magenta (Processing). En jaune, la sortie du filtre RC avec $R = 220[\text{k}\Omega]$ et $C = 6.8[\text{nF}]$. L'Arduino est utilisé en oscilloscope lorsqu'il est couplé à Processing. Les flancs sont raides et le filtrage est suffisant.

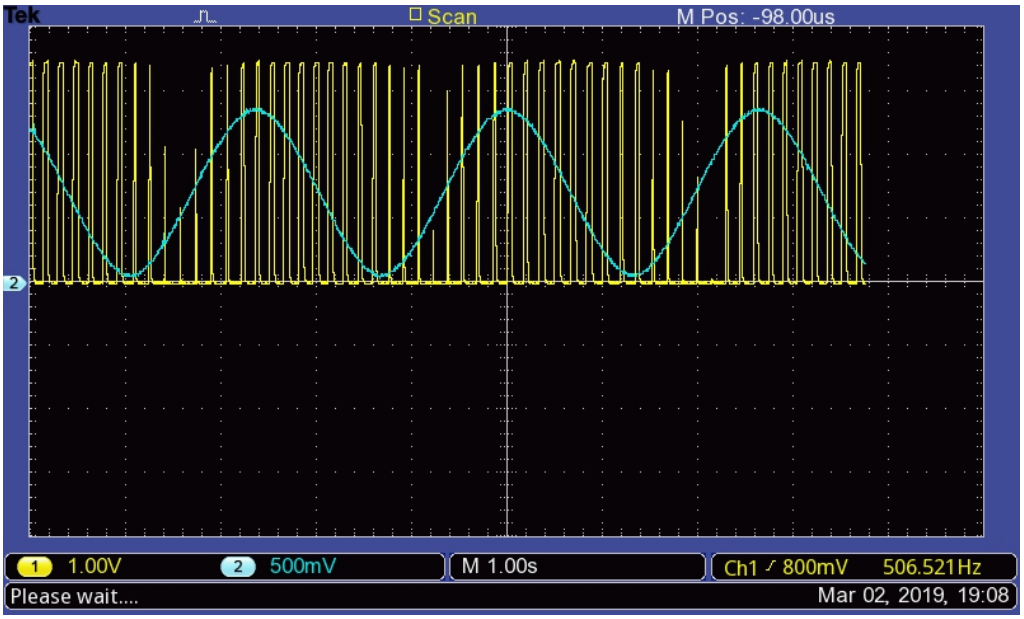

FIGURE 5 – En absence de filtre RC, la modulation à largeur d'impulsion est visible.

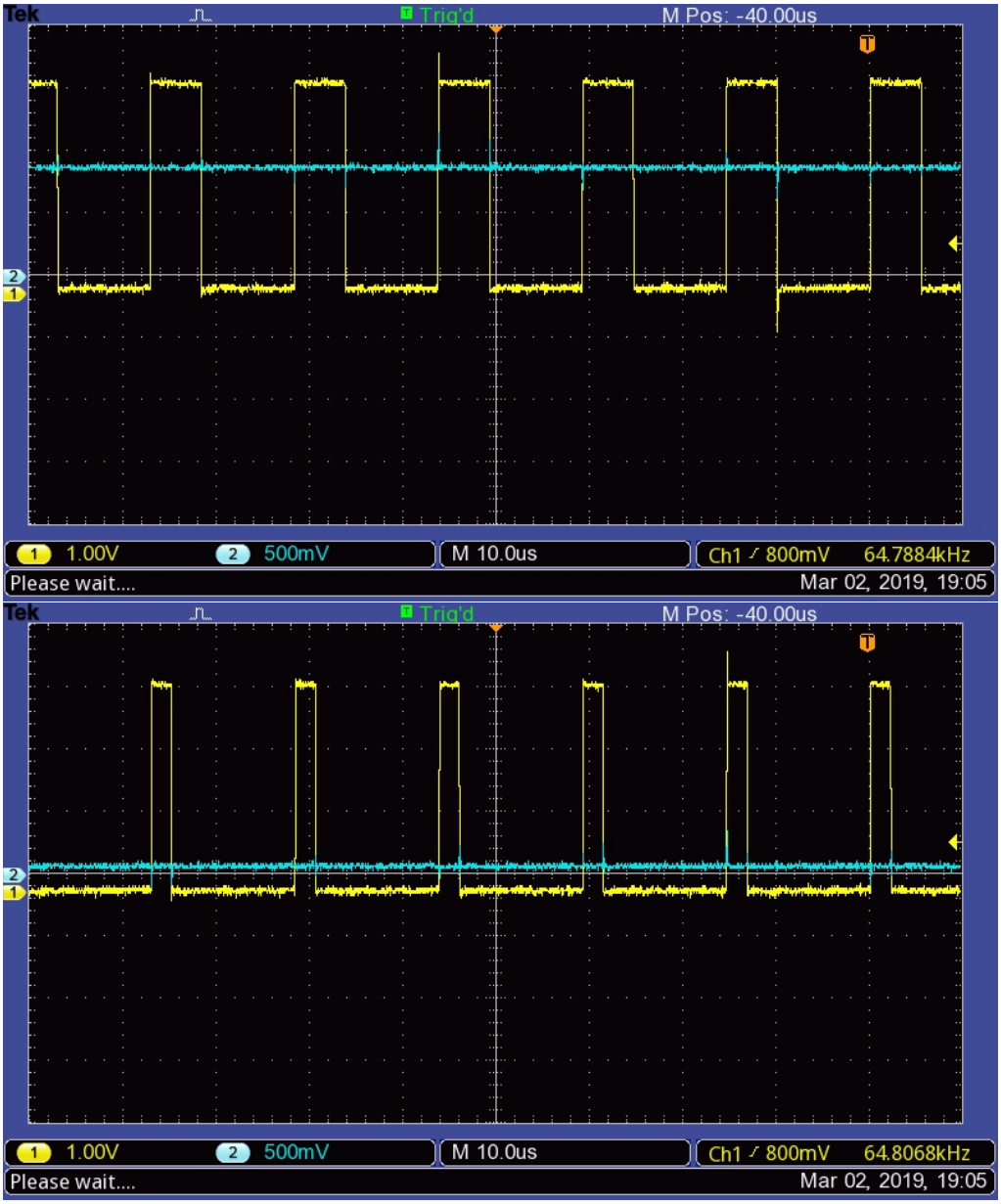

FIGURE 6 – La modulation à largeur d'impulsion est rendue plus nette en augmentant la résolution temporelle de l'oscilloscope. Plus le signal bleu est au milieu de la plage possible, plus le rapport cyclique est proche de 50 %. C'est ce qui se passe sur le graphique du haut. En bas, on constate que la modulation est effectuée de telle sorte que la valeur moyenne du signal jaune soit égale au signal lentement variable bleu qui apparaît constant pour la fenêtre temporelle choisie.