Modulation de largeur d'impulsion

1. Introduction

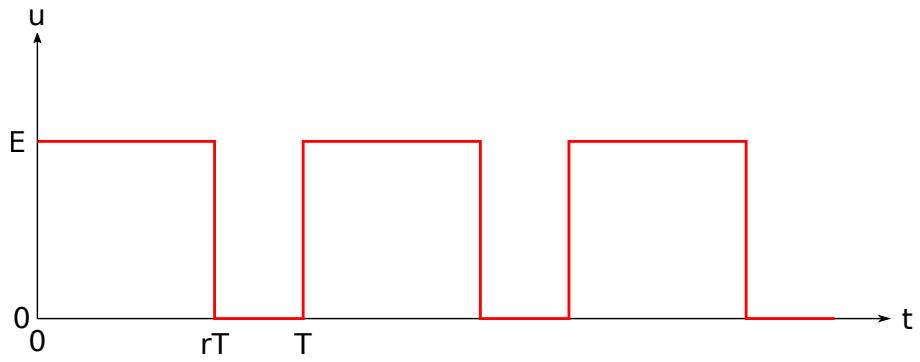
La modulation de largeur d'impulsion (MLI) est une technique utilisée en électronique de puissance pour convertir une tension continue en tension continue (conversion DC-DC) ou une tension continue en tension alternative (conversion DC-AC). L'objectif de ces travaux pratiques est de mettre en œuvre la modulation de largeur d'impulsion à l'aide d'un microcontrôleur. Le signal modulé sera étudié par analyse spectrale et démodulé avec un filtre RC.

Matériel :

- ▷ Carte Arduino MEGA.
- ▷ Plaque d'essai avec borniers bananes.
- ▷ Fils.
- ▷ Condensateur $C = 1{,}0\,\mu\text{F}$.
- ▷ Résistance $R = 15\,\text{k}\Omega$.
- ▷ Carte SysamSP5.

2. Principe

La MLI repose sur la génération d'une tension carrée à rapport cyclique variable, définie sur la figure ci-dessous.



La tension est périodique, de période T, égale à E pendant une durée rT, à 0 pendant $(1-r)T$. Le paramètre r (compris entre 0 et 1) est le *rapport cyclique*.

Exprimer la valeur moyenne de $u(t)$ en fonction de E et r.
Un filtre RC permet d'extraire la valeur moyenne. Faire le schéma de ce filtre et exprimer sa fréquence de coupure.
Pour $T = 1\,\text{ms}$, proposer une valeur pour cette fréquence de coupure.
Proposer une méthode de filtrage similaire utilisant un filtre RL.

En électronique de puissance, la tension u(t) est obtenue à partir d'une tension continue E par un circuit à transistors appelé *hacheur*. Le hacheur est suivi d'un filtre passe-bas. On procède ainsi à la conversion d'une tension continue E en une tension continue ajustable avec le rapport cyclique. Ce principe est utilisé dans les alimentations à découpage (avec un filtre RL). Un système de régulation ajuste automatiquement le rapport cyclique pour que la tension de sortie reste constamment égale à la valeur souhaitée, quelque soit le courant débité. Lorsqu'on alimente une charge inductive, par exemple un moteur électrique, il n'est pas nécessaire d'utiliser un filtre. La vitesse du moteur est contrôlée par le rapport cyclique.

3. Génération d'un signal MLI

3.a. Méthode logicielle

Il s'agit de générer un signal carré de rapport cyclique donné avec une carte Arduino. La carte Arduino MEGA fonctionne avec un microcontrôleur (ATmega2560). Un microcontrôleur contient un microprocesseur (associé à une mémoire) et différentes fonctions permettant d'interagir avec l'extérieur (entrée-sorties numériques, convertisseur A/N, compteurs, etc.). Le logiciel Arduino permet d'écrire le programme exécuté par le microprocesseur en langage C++. Le programme est compilé sur l'ordinateur (c'est-à-dire converti en instructions pour le microprocesseur) puis téléversé vers la carte Arduino.

Une première approche consiste à générer le signal MLI de manière logicielle, en mettant une sortie alternativement au niveau haut et au niveau bas, avec un rapport cyclique choisi. Voici le programme :

generationLogicielle.ino

```
#define SORTIE 11

unsigned int periode = 1000; //période en microsecondes
unsigned int duree_haut;
unsigned int duree_bas;
float rapport_cyclique = 0.3;

void setup() {

  pinMode(SORTIE,OUTPUT);
  duree_haut = periode *rapport_cyclique;
  duree_bas = periode * (1-rapport_cyclique);
}

void loop() {
    int retard = 0;
    digitalWrite(SORTIE,HIGH);
    delayMicroseconds(duree_haut-retard);
    digitalWrite(SORTIE,LOW);
    delayMicroseconds(duree_bas-retard);
}
```

Un programme Arduino est constitué d'un en-tête dans lequel les variables globales sont déclarées. Contrairement au langage Python, le langage C nécessite la déclaration préalable de toutes les variables avec leur type. Le programme ci-dessus comporte une fonction `setup`, qui est exécutée lorsque l'Arduino est initialisé (juste après le téléversement ou après appuie sur la touche RESET de la carte). La fonction `loop` est appelée de manière cyclique. Tant que le microprocesseur n'a pas d'autres tâches à accomplir, le code de cette fonction est exécutée en boucle sans interruption.

Dans la fonction `setup`, la borne 11 est configurée comme une sortie. Les durées sont calculées en fonction de la période et du rapport cylique. Dans la fonction `loop`, on procède à une mise au niveau haut de la sortie puis à une attente pendant la durée `duree_haut`, avant de mettre la sortie au niveau bas et d'attendre pendant une durée `duree_bas`. Lorsque la fonction `delayMicroseconds` est appelée, le microprocesseur exécute un petit programme dont la durée d'exécution est (approximativement) la durée demandée.

Télécharger le fichier ci-dessus et le placer dans un dossier nommé `generationLogicielle`. Ouvrir le fichier avec le logiciel Arduino.
Brancher la carte Arduino avec le câble USB, qui sert à transférer le programme et qui permet d'alimenter la carte.
Dans le menu Outils, sélectionner le port sur lequel se trouve la carte et le type de carte. Téléverser le programme.
Brancher les bornes GND et 11 sur la plaque d'essai. Relier à la borne GND de l'oscilloscope et à la borne de signal de la voie 1.
Observer le signal à l'oscilloscope.
Vérifier le rapport cyclique. La période est-elle égale à celle prévue ?

Pour la durée d'attente programmée avec la fonction `delayMicroseconds`, il faut tenir compte du fait que la fonction `digitalWrite` a une durée d'exécution non négligeable. On peut tenter de corriger ce retard en retranchant une durée `retard` des durées théoriques.

Appliquer un délai de l'ordre de quelques millisecondes pour obtenir à peu près la période souhaitée.

En conclusion, cette méthode logicielle ne permet pas d'obtenir une période de signal très précise. La stabilité de la période n'est pas non plus garantie car le délai appliqué par la fonction `delayMicroseconds` n'est pas parfaitement égal à la durée indiquée et peut varier légèrement d'un appel à l'autre. Par ailleurs, le gros inconvénient de cette méthode est d'occuper le microprocesseur à plein temps. Si l'on souhaite réaliser en parallèle d'autres opérations, il faudra interrompre le programme de la fonction `loop`, ce qui est possible mais viendrait inévitablement compromettre son fonctionnement correct.

Remarque : la fonction `delayMicroseconds` est facile à utiliser mais elle constitue une très mauvaise méthode pour réaliser une durée d'attente car elle occupe le microprocesseur à plein temps. Une méthode bien meilleure consiste à utiliser le mécanisme d'interruption, que nous aborderons en dernière partie.

3.b. Utilisation d'un compteur

Le microcontrôleur comporte des compteurs (appelés communément *timers*) qui peuvent effectuer différentes opérations, en particulier la génération de signaux MLI (connus aussi sous le nom de PWM pour *pulse width modulation*). Le compteur est programmé par le microprocesseur mais il effectue ses opérations indépendamment de celui-ci. La programmation basique d'un compteur pour qu'il génère un signal MLI se fait avec la fonction `analogWrite` de la manière suivante :

```
analogWrite(SORTIE,rapport_cyclique)
```

L'argument `rapport_cyclique` est un entier 8 bits compris entre 0 et 255. La valeur 128 correspond à un rapport cyclique 1/2. Cette fonction ne permet pas de choisir la fréquence du signal MLI. Le programme suivant permet de générer un signal MLI sur la sortie 11 :

generationAnalogWrite.ino

```
#define SORTIE 11

float rapport_cyclique = 0.3;

void setup() {

  pinMode(SORTIE,OUTPUT);
  analogWrite(SORTIE,255*rapport_cyclique);
}

void loop() {
    // on peut faire autre chose ici
}
```

Dans ce cas, la génération du signal MLI n'est plus effectuée par le microprocesseur. Celui-ci est donc libre pour d'autres tâches (en général moins répétitives et plus intelligentes) que l'on peut programmer dans la fonction `loop`.

> Téléverser ce programme. Quelle est la période du signal MLI ? Vérifier le rapport cyclique.
> Réaliser le filtre RC avec les composants fournis. Quelle est sa fréquence de coupure ?
> Observer le signal en sortie du filtre. A-t-on une tension constante ? Vérifier que sa valeur est proportionnelle au rapport cyclique.

Pour faire une étude plus fine des signaux, nous allons faire une numérisation avec la carte SysamSP5 et une analyse spectrale. Le script python ci-dessous fait l'acquisition sur les entrées EA0 et EA1 puis calcule et trace les spectres.

analyseSpectrale.py

```
import numpy
import numpy.fft
```

```
from matplotlib.pyplot import *
import pycanum.main as pycan

can = pycan.Sysam("SP5")
can.config_entrees([0,1],[10,10])
fe= # à choisir
T= # à choisir
te=1/fe
N=int(T/te)
can.config_echantillon(te*1e6,N)
can.acquerir()
t=can.temps()[0]
U=can.entrees()
can.fermer()
u0 = U[0]
u1 = U[1]
N=len(u0)
T=te*N
spectre0 = numpy.absolute(numpy.fft.fft(u0))*2.0/N
spectre1 = numpy.absolute(numpy.fft.fft(u1))*2.0/N
frequences = numpy.arange(N)*1.0/T

figure()
subplot(211)
plot(frequences,spectre0)
#xlim(0,1000)

subplot(212)
plot(frequences,spectre1)
#xlim(0,1000)
xlabel("f (Hz)")

show()
```

Choisir la fréquence d'échantillonnage et la durée de l'acquisition.
Analyser le signal MLI et le signal MLI filtré.
Choisir la fréquence d'échantillonnage afin d'éviter le repliement de spectre.
Quelle est l'influence de la durée T sur le spectre ?
Retrouver la valeur moyenne sur le spectre.

4. Modulation sinusoïdale

4.a. Méthode directe

Il s'agit de moduler le rapport cyclique de manière sinusoïdal, à une fréquence de l'ordre de 1 Hz, beaucoup plus faible que la fréquence du signal MLI. Le signal MLI est généré avec un compteur au moyen de la fonction `analogWrite`. Un appel périodique de cette fonction est effectué dans la fonction `loop`, de manière à faire varier le rapport cyclique. Voici le programme :

modulationAnalogWrite.ino

```
#define SORTIE 11
#define PI 3.1415926
#define NE 1000
uint8_t signal[NE];
unsigned long int periode = 1000000; //période en microsecondes
unsigned long int duree;
unsigned int retard;
unsigned int k;

void setup() {
  pinMode(SORTIE,OUTPUT);
  for (int i=0; i<NE; i++) {
    signal[i] = 255*(0.5+0.3*sin(2*PI/NE*i));
  }
  k=0;
  duree = periode/NE ; // en microsecondes
  retard=10;
}

void loop() {
  analogWrite(SORTIE,signal[k]);
  k+=1;
  if (k==NE) k = 0;
  delayMicroseconds(duree-retard);

}
```

Le signal servant à moduler le rapport cyclique est stocké dans un tableau nommé `signal` de 1000 éléments. On stocke en fait les rapports cycliques codés en entier 8 bits. La durée d'attente entre deux changements du rapport cyclique est égale à la période de modulation (ici 1 s) divisée par le nombre d'échantillons dans la table.

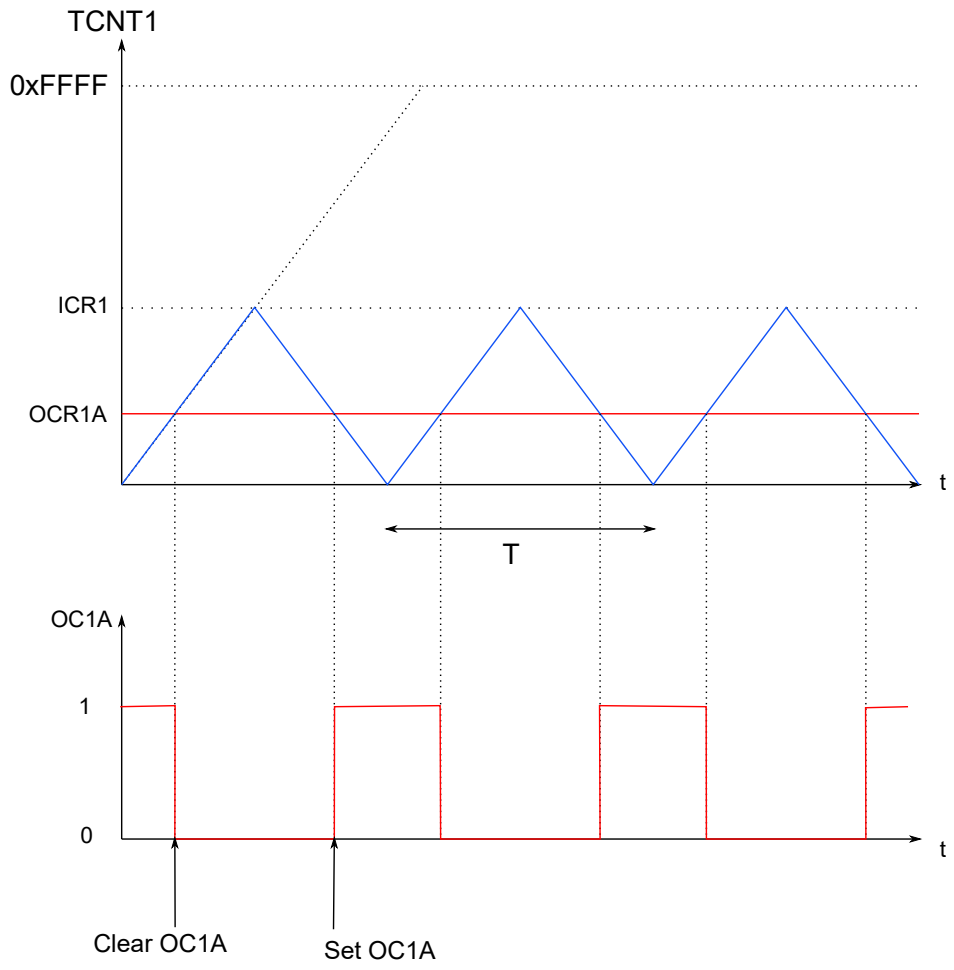
Téléverser ce programme.
Observer le signal en sortie du filtre avec l'oscilloscope.
Faire l'analyse spectrale du signal MLI et du signal filtré.
Expliquer pourquoi le signal en sortie du filtre est sinusoïdal. On dit que le filtre effectue une démodulation.
Mesurer la fréquence du signal en sortie du filtre. Est-elle égale à la valeur attendue ? Ajuster éventuellement la valeur de `retard`, qui sert à compenser le temps d'éxecution des trois lignes qui précèdent.

Cette méthode a deux inconvénients : elle ne permet pas de garantir la fréquence de la modulation et elle occupe le microprocesseur de manière excessive (à cause de la fonction `delayMicroseconds`). Par ailleurs, la fonction `analogWrite` ne permet pas d'augmenter la fréquence du signal MLI, ce qui limite la fréquence du signal modulant.

4.b. Modulation par interruption

Nous allons faire une programmation directe d'un compteur (sans passer par la fonction `analogWrite`) ce qui permettra de choisir la fréquence du signal MLI. Nous allons de plus utiliser des *interruptions* pour déclencher les changements de rapport cyclique. Le même compteur est utilisé pour générer le signal MLI et pour déclencher les interruptions. Lorsque le microprocesseur reçoit un ordre d'interruption, il interrompt provisoirement le programme principal en cours d'exécution pour exécuter un programme secondaire (généralement de courte durée) appelé *service d'interruption*. Les interruptions permettent d'effectuer des tâches avec un chronométrage très précis et garanti. Dans un ordinateur à usage général, les interruptions sont utilisées par le système d'exploitation mais ne sont pas accessibles aux programmes d'application. Sur une carte à microcontrôleur comme l'Arduino, les interruptions permettent d'accomplir des tâches en *temps réel*, ce qui est impossible sur un ordinateur exécutant un système d'exploitation.

Pour comprendre le fonctionnement du programme, il faut comprendre comment fonctionne le compteur. Un compteur 32 bits est en fait un registre 32 bits qui est incrémenté d'une unité à chaque top d'horloge. La fréquence de l'horloge de l'Arduino MEGA est 16 MHz. Cette fréquence étant très élevée, il est possible d'incrémenter le compteur tous les 8, 64, 256 ou 1024 tops d'horloge. Dans ce dernier cas, la fréquence d'incrémentation du compteur est d'environ 15,6 kHz. Le facteur appliqué (1, 8, 64, 256 ou 1024) est appelé *diviseur d'horloge*. Le registre du compteur utilisé est nommé TCNT1. On peut à tout instant lire ce registre pour connaître la valeur du compteur. Lorsque la valeur de TCNT1 atteint la valeur du registre ICR1, le compteur entre dans une phase de décrémentation jusqu'à la valeur nulle, à partir de laquelle l'incrémentation recommence. Pour générer un signal MLI, un troisième registre 32 bits est utilisé : le registre OCR1A. Le signal généré est nommé OC1A. Il est émis sur la sortie 11 de la carte. Lorsque la valeur de TCNT1 est supérieure à OCR1A, OC1A est à l'état bas. Lorsque la valeur de TCNT1 est inférieure à OCR1A, OC1A est à l'état haut. On voit ainsi que le rapport cyclique est le rapport OCR1A/ICR1.



Pour choisir la période T du signal MLI, on joue à la fois sur le diviseur d'horloge et sur la valeur de ICR1, sachant que la période T est égale à deux fois la valeur de ICR1 multipliée par la période de l'horloge et divisée par le diviseur d'horloge.

Le même compteur peut servir à générer des interruptions. Nous choisissons de générer une interruption à chaque fois que TCNT1 atteint la valeur ICR1. Dans le service d'interruption, la valeur de OCR1A est ajustée pour tenir compte de la variation du rapport cyclique.

modulationTimer.ino

```
#define SORTIE 11
#define PI 3.1415926
#define NE 256 // indices de table sur 8 bits
uint32_t accum; // accummulateur de phase 32 bits
uint32_t increm; // incrément de l'accumulateur de phase
#define SHIFT_ACCUM 24 // 32 bits - 8 bits
uint32_t signal[NE];
uint32_t icr;

//Initialisation du compteur pour génération MLI avec interruptions//
void init_pwm_timer1(uint32_t period) { // période en microsecondes
```

```
  uint16_t diviseur[6] = {0,1,8,64,256,1024};
  TCCR1A = (1 << COM1A1); //Clear OC1A on compare match when upcounting, set OC1A on
  TCCR1B = 1 << WGM13; // phase and frequency correct pwm mode, top = ICR1
  int d=1;
  icr = (F_CPU/1000000*period/2);
  while ((icr>0xFFFF)&&(d<6)) { // choix du diviseur d'horloge
     d++;
     icr = (F_CPU/1000000*period/2/diviseur[d]);
  }
  ICR1 = icr; // valeur maximale du compteur
  TIMSK1 = 1 << TOIE1; // overflow interrupt enable
  TCNT1 = 0; // mise à zéro du compteur
  TCCR1B |= d; // déclenchement du compteur
}

//Fonction appelée à l'interruption //
ISR(TIMER1_OVF_vect) { // Timer 1 Overflow interrupt
  accum += increm; // incrémentation de l'accumulateur de phase
  OCR1A = signal[accum >> SHIFT_ACCUM]; // mise à jour du rapport cyclique
}


void setup() {
  pinMode(SORTIE,OUTPUT);
  accum = 0;
  uint32_t period_pwm = 100; // période PWM en microsecondes
  float frequence = 1.0; // fréquence du signal en Hz
  increm = (uint32_t) (((float)(0xFFFFFFFF))*(frequence*1e-6*(float)(period_pwm))); /
  init_pwm_timer1(period_pwm);
  for (int i=0; i<NE; i++) {
    signal[i] = icr*0.5*(1.0+0.5*sin(2*PI/NE*i));
  }
  sei(); // activation des interruptions
}

void loop() {


}
```

La configuration du compteur est faite dans la fonction `init_pwm_timer1`, dont l'agument est la période T en microsecondes. La fonction appelée lors de l'interruption est `ISR(TIMER1_OVF_vect)`. Elle est appelée exactement lorsque le compteur atteint sa valeur maximale. Les valeurs de OCR1A correspondant au signal modulant sinusoïdal sont stockées dans le tableau `signal`. Ce tableau est indexé par un entier 8 bits ; il contient donc 256 échantillons qui représentent exactement une période de la sinusoïde. Afin de pouvoir faire varier très finement la fréquence du signal, on utilise la technique de l'accumulateur de phase. Celui-ci est un entier de 32 bits (non signé) stocké dans la variable

`accum`. À chaque interruption, l'accumulateur est incrémenté d'une valeur donnée par `increm`. Lorsque la valeur 2^{32} est atteinte, l'accumulateur revient automatiquement à zéro (par exemple $2^{32} - 1 + 3 = 2$). L'accumulateur de phase représente un nombre décimal dont la partie entière est l'indice de l'élément à lire dans le tableau `signal`. Pour représenter un nombre à virgule fixe par un entier, il suffit de décider combien de bits sont attribués à la partie entière de ce nombre. Dans le cas présent, il y a 8 bits pour la partie entière et il reste 32-8=24 bits pour la partie décimale. Pour obtenir la partie entière, il suffit d'effectuer un décalage de 24 bits vers la droite, ce qui est fait par l'opération `accum >> SHIFT_ACCUM`. On pourrait obtenir le même résultat avec un accumulateur à virgule flottante (type `float`) mais le temps d'exécution de l'incrémentation serait beaucoup plus long car le microprocesseur de l'arduino ne calcule pas directement en virgule flottante (les calculs en flottants sont mis en place par le compilateur).

Téléverser le programme.
Observer le signal MLI filtré. Pourquoi le résultat est-il meilleur que précédemment ?
Faire varier la fréquence du signal sinsusoïdal modulant et vérifier à l'oscilloscope.
Faire une analyse spectrale pour déterminer la fréquence du signal (attention au choix de la durée d'acquisition).
Expliquer le calcul de l'incrément de l'accumulateur de phase.