Modulation de largeur d'impulsion

1. Introduction

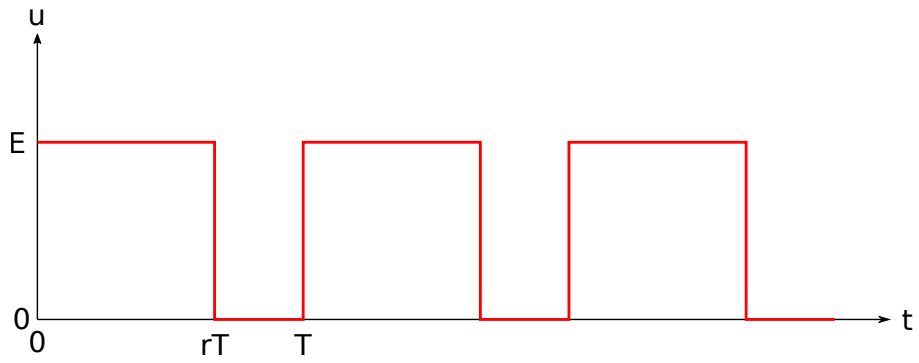
La modulation de largeur d'impulsion (MLI) est une technique utilisée en électronique de puissance pour convertir une tension continue en tension continue (conversion DC-DC) ou une tension continue en tension alternative (conversion DC-AC). L'objectif de ces travaux pratiques est de mettre en œuvre la modulation de largeur d'impulsion à l'aide d'un microcontrôleur (carte Arduino). 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.
- Condensateurs $C = 1{,}0\,\mu\mathrm{F}$ et $C = 100\,\mathrm{nF}$.
- Résistance $R = 15\,\mathrm{k}\Omega$.
- Carte SysamSP5.

On dispose du script analyse-spectrale.py, qui effectue la numérisation sur la voie EA0 de la carte SysamSP5 et trace le spectre du signal.
Les programmes Arduino utilisés sont disponibles dans le fichier : arduino.zip. Télécharger ce fichier et le décompresser. Chaque programme Arduino doit être placé dans un dossier portant le même nom que le fichier source.
Un microcontrôleur est un circuit comportant un microprocesseur, une mémoire et différentes interfaces permettant de communiquer avec l'extérieur. Il comporte en particulier un ou plusieurs compteurs (timers) permettant de générer des signaux périodiques ou de déclencher des actions périodiques. Une carte Arduino comporte un microcontrôleur, une horloge à quartz, un circuit de stabilisation de la tension d'alimentation, un port USB, un circuit permettant de communiquer avec le PC via ce port et les bornes de connexion aux différents ports d'entrée-sortie du microcontrôleur.

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*.

[1] Exprimer la valeur moyenne de $u(t)$ en fonction de E et r.

[2] Un filtre RC permet d'extraire la valeur moyenne. Faire le schéma de ce filtre et exprimer sa fréquence de coupure. Comment faut-il choisir cette fréquence de coupure en fonction de la période T.

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 de consigne, quel que 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.

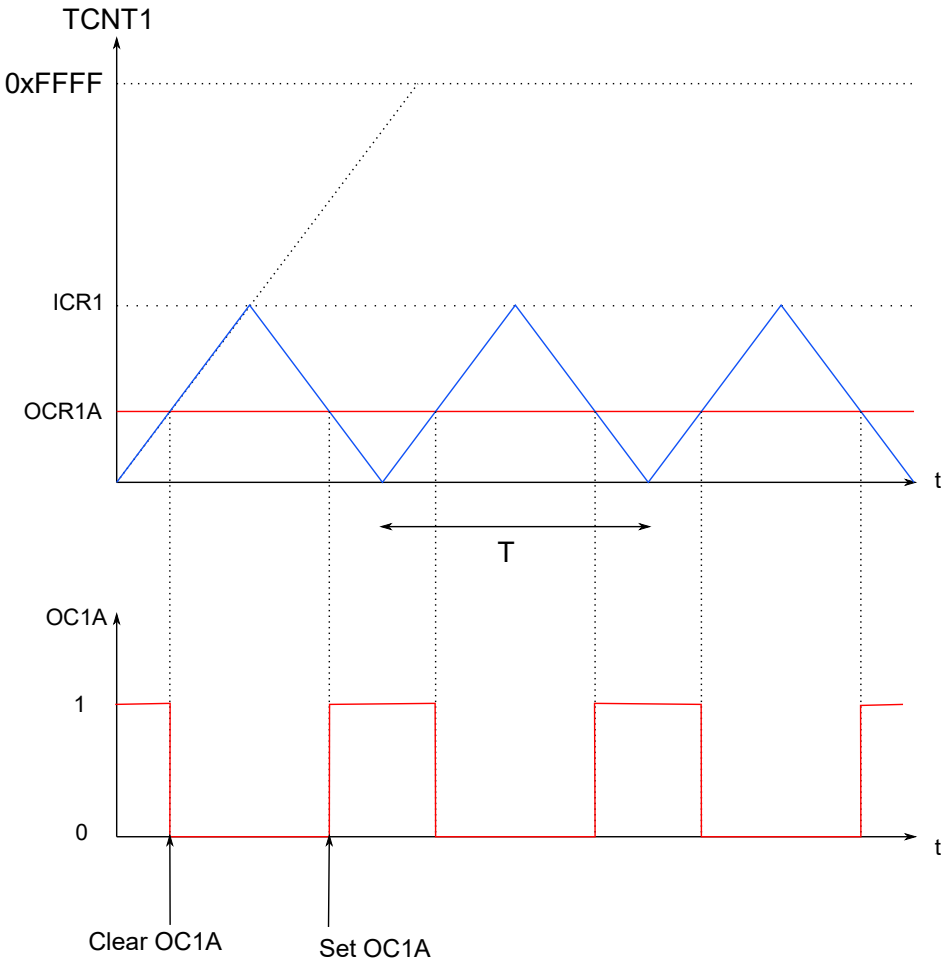
3. Génération du signal

3.a. Principe

Le microcontrôleur comporte des compteurs (ou *timers*) qui peuvent effectuer différentes opérations, en particulier la génération de signaux MLI (connus aussi sous le nom de PWM pour *pulse width modulation*). Un compteur est programmé par le microprocesseur mais il effectue ses opérations indépendamment de celui-ci.

Le microcontrôleur de l'arduino MEGA (ATmega 2560) comporte quatre compteurs 16 bits (Timers 1,3,4,5). Un compteur 16 bits comporte un registre 16 bits qui est incrémenté d'une unité à chaque top d'horloge (un registre est un emplacement de la mémoire réservé à un usage particulier). La fréquence de l'horloge de l'Arduino MEGA est $16\,\mathrm{MHz}$. 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 $15{,}6\,\mathrm{kHz}$. Le facteur appliqué (1, 8, 64, 256 ou 1024) est appelé *diviseur d'horloge*. Le registre du compteur 1 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 16 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. Le signal carré émis a un rapport cyclique égal à OCR1A/ICR1. En modifiant la valeur stockée dans le registre OCR1A, il sera donc possible de moduler le rapport cyclique.



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.

3.b. Programme arduino

La manière la plus simple de générer un signal MLI (ou PWM) est d'utiliser la fonction `analogWrite(sortie,rapport)`. L'argument `sortie` désigne le numéro de la sortie et `rapport` est le rapport cyclique exprimé sous la forme d'un nombre entier 8 bits, compris entre 0 et 255. 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() {

}
```

La fonction `setup` est exécutée lors du démarrage de la carte (ou après appuie sur la touche RESET). La génération du signal MLI est programmée dans cette fonction (avec un rapport cyclique fixe). La fonction `loop` est exécutée de manière cyclique.

[3] Ouvrir le fichier avec le logiciel Arduino.

[4] 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 (Mega 2560). Téléverser le programme.

[5] 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.

[6] Mesurer la période. Vérifier le rapport cyclique pour différentes valeurs programmées. La fonction `analogWrite` ne permet pas de modifier la fréquence du signal MLI. Par ailleurs, c'est une fonction d'exécution relativement lente car elle accomplie la configuration complète du compteur. Pour effectuer une modulation du rapport cyclique, il est préférable de programmer directement le compteur, ce que fait le programme suivant :

generationTimer.ino

```
#define SORTIE 11

uint32_t icr;
uint32_t period;

//Initialisation du Timer1 pour génération MLI
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 com
  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]);
  }
  Serial.print("diviseur_1= "); Serial.println(d);
  Serial.print("icr_1 = "); Serial.println(icr);
  float periode_effective = icr*2*d*1000000.0/F_CPU;
  Serial.print("periode_1 (us) = "); Serial.println(periode_effective);
  ICR1 = icr; // valeur maximale du compteur
  TCNT1 = 0; // mise à zéro du compteur
  TCCR1B |= d; // déclenchement du compteur
}
```

```
void rapport_cyclique(float r) {
  OCR1A = icr*r;
  Serial.print("OCR1A = "); Serial.println(OCR1A);
}

void setup() {
  Serial.begin(9600);
  pinMode(SORTIE,OUTPUT);
  period = 1000; // période en microsecondes
  init_pwm_timer1(period);
  rapport_cyclique(0.1);
}

void loop() {

}
```

La fonction `init_pwm_timer1` configure le Timer 1 pour qu'il génère un signal MLI dont la période est donnée en microsecondes (sur la sortie 11). Une fois le compteur lancé, le changement du rapport cyclique se fait simplement en modifiant la valeur stockée dans le registre OCR1A. La fonction `rapport_cyclique` permet de faire cela à partir d'un rapport cyclique donné sous la forme d'un `float` compris entre 0 et 1. La fonction de configuration affiche le diviseur d'horloge, la valeur de ICR et la période effective.

[7] Téléverser ce programme sur la carte. Ouvrir le moniteur série (dans le menu Outils) et ajuster la vitesse de transmission à 9600 bauds (9600 bits par secondes). Relever les valeurs du diviseur d'horloge, de ICR et de OCR1A.

[8] Observer le signal à l'oscilloscope pour vérifier la période et le rapport cyclique.

[9] Effectuer une analyse spectrale afin de déterminer la fréquence au dixième de hertz près. Est-elle conforme à la valeur programmée ? Comment expliquer l'écart ?

[10] Pour cette fréquence, combien y-a-t-il de valeurs différentes possibles du rapport cyclique ? Le fait que le rapport cyclique programmé par la fonction `analogWrite` est un entier 8 bits est-il pertinent pour ce compteur 16 bits ?

[11] Refaire l'étude pour une fréquence de $10\,\mathrm{kHz}$.

[12] Revenir à une fréquence de $1\,\mathrm{kHz}$ et réaliser le filtre RC passe-bas permettant d'obtenir la valeur moyenne (deux condensateurs sont disponibles). Observer l'ondulation résiduelle en sortie. Choisir le condensateur qui donne l'ondulation la plus faible et donner la valeur de l'amplitude de l'ondulation. Confirmer l'ordre de grandeur de cette valeur par un calcul théorique simple.

4. Modulation du rapport cyclique

4.a. Principe

La modulation de largeur d'impulsion consiste à effectuer une modulation du rapport cyclique, par un signal dont la fréquence est beaucoup plus faible que celle du signal MLI (porteuse). La démodulation s'effectue avec un filtre passe-bas dont la fréquence de coupure est très faible par rapport à la fréquence de la porteuse et supérieure à la fréquence du signal modulant.

4.b. Modulation par temporisation

Le signal modulant (par exemple une sinusoïde) consiste en un tableau contenant NE échantillons du rapport cyclique répartis sur une période. On choisit une période d'échantillonnage TE. La période de la modulation est donc TE*NE.
Une première méthode pour effectuer la modulation consiste à utiliser la fonction `delayMicroseconds` pour fixer le temps d'attente entre deux changements consécutifs du rapport cyclique.

modulation-delay.ino

```
#define SORTIE 11
#define NE 100 // nombre d'échantillons du signal

uint32_t icr;
uint32_t period;
float echant[NE]; // tableau contenant les échantillons du signal
uint32_t periode_echant;
uint16_t indice;

//Initialisation du Timer1 pour génération MLI
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 com
  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]);
  }
  Serial.print("diviseur_1= "); Serial.println(d);
  Serial.print("icr_1 = "); Serial.println(icr);
  float periode_effective = icr*2*d*1000000.0/F_CPU;
  Serial.print("periode_1 (us) = "); Serial.println(periode_effective);
  ICR1 = icr; // valeur maximale du compteur
  TCNT1 = 0; // mise à zéro du compteur
  TCCR1B |= d; // déclenchement du compteur
}

void rapport_cyclique(float r) {
  OCR1A = icr*r;
}

void definir_signal(float amp) { // amplitude inférieure à 1
  for (int i=0; i<NE; i++) echant[i] = 0.5*(1+amp*sin(2*PI/NE*i));
}

void setup() {
  Serial.begin(9600);
  pinMode(SORTIE,OUTPUT);
  period = 1000; // période de la porteuse en microsecondes
  init_pwm_timer1(period);
  definir_signal(0.5);
  periode_echant = 10000;// microsecondes
  indice = 0;
}

void loop() {
```

```
  delayMicroseconds(periode_echant);
  indice += 1;
  if (indice==NE) indice=0;
  rapport_cyclique(echant[indice]);

}
```

Le tableau des valeurs du rapport cyclique est calculé dans la fonction `definir_signal`, appelée dans la fonction `setup`. La modulation est effectuée dans la fonction `loop`. Cette fonction est exécutée de manière cyclique mais il existe un petit délai entre deux exécutions consécutives.

[13] Téleverser ce programme sur la carte et observer le signal MLI à l'oscilloscope.

[14] Effectuer la démodulation avec le filtre RC. Véfifier la fréquence du signal sinusoïdal.

[15] Effectuer une analyse spectrale afin de déterminer cette fréquence au dixième de hertz près.

[16] Refaire les mesures pour une période d'échantillonnage dix fois plus petite. Pourquoi la fréquence du signal démodulé est-elle inférieure à celle prévue ? Conclure sur la précision de la méthode.

4.c. Modulation à l'aide d'un compteur

Pour obtenir un réglage précis de la période d'échantillonnage, l'utilisation de la fonction `delayMicroseconds` ne convient pas. Cette fonction a de plus l'inconvénient d'être bloquante, ce qui rend le microprocesseur indisponible pendant l'attente.
Pour effectuer un échantillonnage précis, il faut utiliser un compteur. Un compteur permet en effet de déclencher périodiquement une interruption du programme principal. Nous allons utiliser pour cela le Timer 3 et préciser qu'une interruption doit être déclenchée lorsque la valeur du compteur (registre TCTNT3) dépasse la valeur maximale ICR3 et commence sa phase de décroissance (*Overflow interrupt*). À chaque fois que cet évènement se produit, le programme en cours d'exécution par le microprocesseur est interrompu afin d'exécuter une fonction d'interruption, dans laquelle on effectuera la modification du rapport cyclique du signal généré par le Timer 1.

modulation-timer.ino

```
#define SORTIE 11
#define NE 100 // nombre d'échantillons du signal

uint32_t icr;
uint32_t period;
volatile float echant[NE]; // tableau contenant les échantillons du signal
uint32_t periode_echant;
volatile uint16_t indice;

//Initialisation du Timer1 pour génération MLI
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 com
  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]);
  }
  Serial.print("diviseur_1= "); Serial.println(d);
  Serial.print("icr_1 = "); Serial.println(icr);
  float periode_effective = icr*2*d*1000000.0/F_CPU;
  Serial.print("periode_1 (us) = "); Serial.println(periode_effective);
  ICR1 = icr; // valeur maximale du compteur

  TCNT1 = 0; // mise à zéro du compteur
  TCCR1B |= d; // déclenchement du compteur
}

// initialisation du Timer3 pour l'échantillonnage du signal modulant
void init_pwm_timer3(uint32_t period) { // période en microsecondes
  uint16_t diviseur[6] = {0,1,8,64,256,1024};
  TCCR3A = (1 << COM1A1); //Clear OC1A on compare match when upcounting, set OC1A on com
  TCCR3B = 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]);
  }
  Serial.print("diviseur_3= "); Serial.println(d);
  Serial.print("icr_3 = "); Serial.println(icr);
  float periode_effective = icr*2*d*1000000.0/F_CPU;
  Serial.print("periode_3 (us) = "); Serial.println(periode_effective);
  ICR3 = icr; // valeur maximale du compteur
  OCR3A = icr*0.5;
  TIMSK3 = 1 << TOIE1; // overflow interrupt enable
  TCNT3 = 0; // mise à zéro du compteur
  TCCR3B |= d; // déclenchement du compteur
}

//Fonction appelée à l'interruption //
ISR(TIMER3_OVF_vect) { // Timer 3 Overflow interrupt
  indice += 1;
  if (indice==NE) indice=0;
  OCR1A = icr*echant[indice];
}

void definir_signal(float amp) { // amplitude inférieure à 1
  for (int i=0; i<NE; i++) echant[i] = 0.5*(1+amp*sin(2*PI/NE*i));
}


void setup() {
  Serial.begin(9600);
  pinMode(SORTIE,OUTPUT);
  period = 1000; // période de la porteuse en microsecondes
  init_pwm_timer1(period);
  definir_signal(0.5);
  periode_echant = 10000;// microsecondes
  indice = 0;
  init_pwm_timer3(periode_echant);
}

void loop() {
```

```
}
```

L'entête de la fonction appelée lors de l'interruption est `ISR(TIMER3_OVF_vect)` (*interrupt service routine*). Les variables utilisées dans cette fonction doivent être déclarées avec l'attribut `volatile`.

[17] Téleverser ce programme sur la carte.

[18] Effectuer une analyse spectrale du signal démodulé au dixième de hertz près pour un signal modulant de 1 Hz. Comparer à la méthode précédente et conclure.

[19] Générer un signal MLI de fréquence 10 kHz avec un signal modulant de 50 Hz, en prenant soin de bien choisir le condensateur du filtre RC.

[20] Modifier la fonction `definir_signal` afin de générer un signal en dent de scie, comportant une rampe montante entre 0 et 5 V, suivi d'un retour à zéro.

5. Modulation par un signal externe

Dans cette partie, le signal modulant est délivré par un générateur de fonctions. La voie A du générateur (SIGLENT) est branchée sur l'entrée analogique A0 de la carte Arduino. On utilise le convertisseur analogique-numérique intégré au microcontrôleur (convertisseur 10 bits) au moyen de la fonction `analogRead`. Le programme est similaire au précédent, mais la fonction d'interruption est chargée d'effectuer la conversion A/N et de modifier le rapport cyclique en conséquence.

modulation-timer-externe.ino

```
#define SORTIE 11
#define ENTREE A0


uint32_t icr;
uint32_t period;
uint32_t periode_echant;


//Initialisation du Timer1 pour génération MLI
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 com
  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]);
  }
  Serial.print("diviseur_1= "); Serial.println(d);
  Serial.print("icr_1 = "); Serial.println(icr);
  float periode_effective = icr*2*d*1000000.0/F_CPU;
  Serial.print("periode_1 (us) = "); Serial.println(periode_effective);
  ICR1 = icr; // valeur maximale du compteur

  TCNT1 = 0; // mise à zéro du compteur
```

```
  TCCR1B |= d; // déclenchement du compteur
}

// initialisation du Timer3 pour l'échantillonnage du signal modulant
void init_pwm_timer3(uint32_t period) { // période en microsecondes
  uint16_t diviseur[6] = {0,1,8,64,256,1024};
  TCCR3A = (1 << COM1A1); //Clear OC1A on compare match when upcounting, set OC1A on com
  TCCR3B = 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]);
  }
  Serial.print("diviseur_3= "); Serial.println(d);
  Serial.print("icr_3 = "); Serial.println(icr);
  float periode_effective = icr*2*d*1000000.0/F_CPU;
  Serial.print("periode_3 (us) = "); Serial.println(periode_effective);
  ICR3 = icr; // valeur maximale du compteur
  OCR3A = icr*0.5;
  TIMSK3 = 1 << TOIE1; // overflow interrupt enable
  TCNT3 = 0; // mise à zéro du compteur
  TCCR3B |= d; // déclenchement du compteur
}

//Fonction appelée à l'interruption //
ISR(TIMER3_OVF_vect) { // Timer 3 Overflow interrupt
  OCR1A = icr*analogRead(ENTREE)/1023.0;
}
```

```
void setup() {
  Serial.begin(9600);
  pinMode(SORTIE,OUTPUT);
  period = 1000; // période de la porteuse en microsecondes
  init_pwm_timer1(period);

  periode_echant = 10000;// microsecondes
  init_pwm_timer3(periode_echant);
}

void loop() {

}
```

[21] Régler le générateur de fonctions afin qu'il délivre un signal sinusoïdal d'une fréquence de l'ordre du hertz, avec un décalage (offset) ajustée de telle sorte que la tension soit comprise entre 0 et 5 V. ATTENTION : si cette condition n'est pas respectée, le convertisseur A/N de l'arduino peut être détruit.

[22] Téleverser ce programme sur la carte et brancher la sortie du générateur sur l'entrée analogique A0.

[23] Comparer le signal de commande et le signal en sortie du filtre. Augmenter la fréquence.

La fonction `analogRead` est trop lente pour une utilisation optimale du convertisseur A/N. Le programme suivant effectue une programmation directe du convertisseur, ce qui permet d'effectuer la conversion avec une période d'échantillonnage égale à la période de la porteuse. Dans ce cas, un seul compteur (Timer 1) est suffisant : il sert à la fois à générer le signal MLI et les interruptions qui effectuent la conversion A/N et le changement du rapport cyclique.

modulation-timer-externe-rapide.ino

```
#define SORTIE 11
#define ENTREE A0


uint32_t icr;
uint32_t period;
uint32_t periode_echant;


//Initialisation du Timer1 pour génération MLI
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 com
  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]);
  }
  Serial.print("diviseur_1= "); Serial.println(d);
  Serial.print("icr_1 = "); Serial.println(icr);
  float periode_effective = icr*2*d*1000000.0/F_CPU;
  Serial.print("periode_1 (us) = "); Serial.println(periode_effective);
  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 3 Overflow interrupt
  ADMUX = 0b01000000 | 0; // voie A0
  ADCSRA |= 0b01000000; // start ADC
  while (ADCSRA & 0b01000000);
  OCR1A = icr* (ADCL | (ADCH << 8))/1023.0;
}


void adc_init(uint8_t prescaler) {
  ADCSRA = 0;
  ADCSRA |= (1 << ADEN); // enable ADC
  ADCSRA |= prescaler ;
  ADCSRB = 0;
}

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

```
  pinMode(SORTIE,OUTPUT);
  period = 500; // période de la porteuse en microsecondes
  init_pwm_timer1(period);
  adc_init(4);
}

void loop() {

}
```

[24] Tester ce programme avec différentes formes de signal de commande.

[25] Avec un signal de commande sinusoïdal, jusqu'à quelle fréquence fonctionne-t-il correctement ?

[26] Pour un signal de commande en créneau d'une fréquence de $1\,\mathrm{Hz}$, choisir au mieux la fréquence du signal MLI et le filtre RC. Que se passe-t-il lorsqu'on augmente la fréquence du signal de commande. Expliquer.