MODIFICATION DE LA FREQUENCE DU PWM ET DE L'ARDUINO

Réalisé par Bruno PIQUEMAL

24 septembre 2020

Ce guide vous permettra de comprendre et de modifier la fréquence des sorties PWM et de l'Arduino.

Introduction

PWM veut dire Pulse Width Modulation ou Modulation de Largeur d'Impulsion (MLI). Le principe est de créer un signal logique (valant 0 ou 1), à fréquence fixe mais dont le rapport cyclique est contrôlé numériquement. La valeur moyenne obtenue à partir de ce signal est une grandeur analogique et elle est égale au produit du rapport cyclique par l'amplitude maximale du signal.
Le PWM est très utilisé pour envoyer des messages audio, contrôler des moteurs à vitesse variable et aussi pour faire varier le courant pour alimenter des composants, comme des LED par exemple.

La modification de la fréquence de fonctionnement de l'Arduino permet de réduire la consommation de ce dernier. Il est même possible de cumuler basses fréquences de fonctionnement et diminution de la tension d'alimentation pour aboutir à une très basse consommation.

Table des matières

⋆ ⋆ ⋆

A Matériel

Pour ce tutoriel nous allons avoir besoin d'un Arduino UNO et d'un oscilloscope afin de vérifier la fréquence, le duty cycle et la tension de sortie.

B Foncionnement du PWM

La fréquence par défaut de l'Arduino UNO est de 490 Hz pour les pins 3, 9, 10, 11 et de 980 Hz pour les pins 5, 6. Il se peut cependant que l'on veuille augmenter la fréquence ou la diminuer. La fréquence maximale atteinte par l'arduino UNO est de 8 MHz.

Afin de modifier ces valeurs, on va devoir travailler avec les timers internes à l'ATMEGA 328P et donc changer les registres. Il existe trois timers sur ce composant et ils contrôlent des pins différents :

- Timer 0 : pin 5 et pin 6
- Timer 1 : pin 9 et pin 10
- Timer 2 : pin 3 et pin 11

Quatre types de modes existent pour régler le PWM des pins :

- Fast PWM mode
- Phase correct PWM mode
- Frequency and phase correct PWM mode (uniquement pour le Timer 1)
- CTC mode

Attention

La modification des registres peut entraîner des modifications dans le fonctionnement des fonctions millis() et delay(). Veuillez consulter la section "**Pour aller plus loin**".

B.1 Fast PWM mode

Commençons ce tutoriel par le Fast PWM mode. Pour cela, il faut se référer à la datasheet du composant ATMEGA 328P.
Ce mode fait que le Timer compte d'un point bas à un point haut et qu'à partir de ce dernier, on repasse au point bas le long de la verticale. Il y a donc création d'une onde en dent de scie, ce qui permet de générer des hautes fréquences.

Figure 14-6. Fast PWM Mode, Timing Diagram

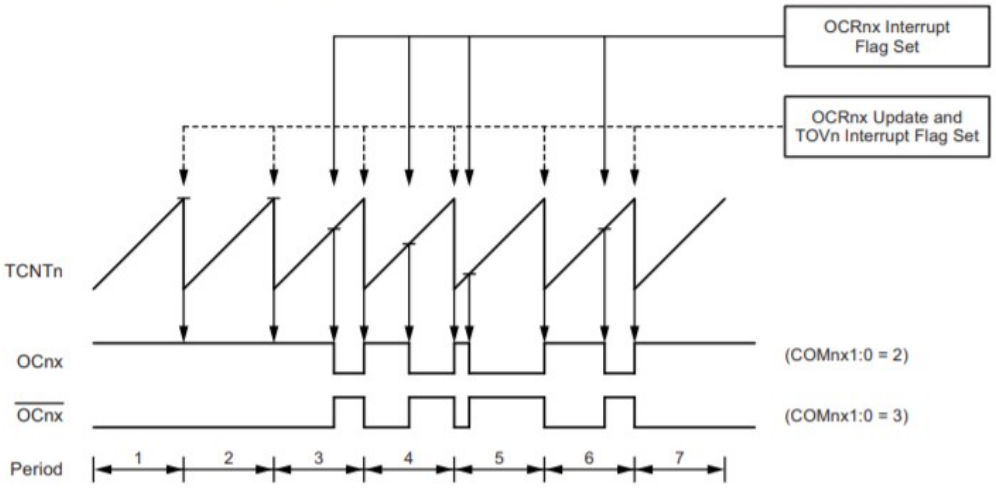

FIGURE 1 – Diagramme des temps du mode Fast PWM

Ce mode est régi par la formule suivante :

$$f_{OCnxPWM} = \frac{f_{clk\_I/O}}{N \times 256}$$

The *N* variable represents the prescale factor (1, 8, 64, 256, or 1024).

FIGURE 2 – Formule pour l'obtention de la fréquence de sortie du mode Fast PWM

Avec $f_{OCnxPWM}$ la fréquence souhaitée, $f_{clkI/O}$ la fréquence du quartz qui est normalement de 16 MHz (à vérifier sur l'arduino).

B.2 Phase Correct PWM mode

Ce mode fait que le Timer compte d'une valeur basse à une valeur haute et de nouveau jusqu'à une valeur basse. Ainsi, l'onde créée est une onde triangulaire ce qui fait que les fréquences seront deux fois moins élevées que celles du Fast PWM mode. Ce mode est apprécié pour le contrôle de moteurs.

Figure 14-7. Phase Correct PWM Mode, Timing Diagram

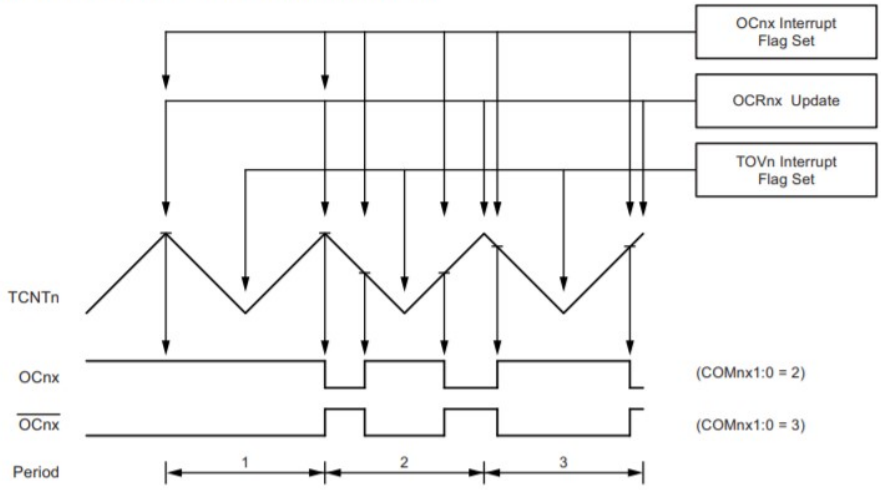

FIGURE 3 – Diagramme des temps du mode Phase Correct

Ce mode est régi par la formule suivante :

$$f_{OCnxPCPWM} = \frac{f_{clk\ I/O}}{N \times 510}$$

The N variable represents the prescale factor (1, 8, 64, 256, or 1024).

FIGURE 4 – Formule pour l'obtention de la fréquence de sortie du mode Phase Correct

Avec $f_{OCnxPCPWM}$ la fréquence souhaitée, $f_{clkI/O}$ la fréquence du quartz qui est normalement de 16 MHz (à vérifier sur l'arduino).

B.3 Frequency and Phase Correct PWM mode (que pour le Timer 1)

Ce mode est similaire à celui du Phase correct PWM mode, il passe d'une valeur basse à une valeur haute et de nouveau à une valeur basse. Toutefois, il permet d'avoir deux valeurs hautes, ce qui fait que si la valeur haute n°2 est inférieure à la valeur haute n°1, lorsque le signal montera jusqu'à la valeur haute n°2 et redescendra, les impulsions auront une période plus courte et donc une fréquence plus élevée.

Figure 15-9. Phase and Frequency Correct PWM Mode, Timing Diagram

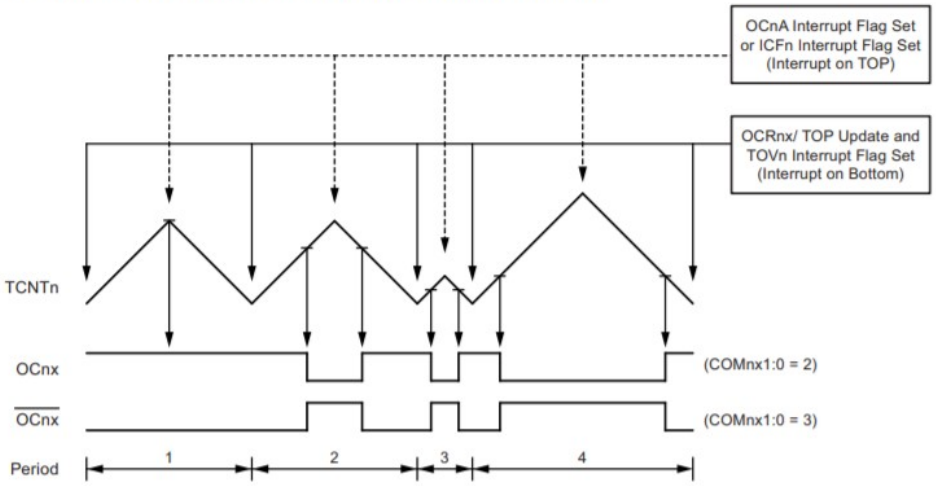

FIGURE 5 – Diagramme des temps du mode Frequency and Phase Correct PWM mode

Ce mode est régi par la formule suivante :

$$f_{OCnxPFCPWM} = \frac{f_{clk\ I/O}}{2 \times N \times TOP}$$

The N variable represents the prescaler divider (1, 8, 64, 256, or 1024).

FIGURE 6 – Formule pour l'obtention de la fréquence de sortie du mode Frequency and Phase Correct PWM

Avec $f_{OCnxPFCPWM}$ la fréquence souhaitée, $f_{clkI/O}$ la fréquence du quartz qui est normalement de 16 MHz (à vérifier sur l'arduino) et TOP étant la valeur haute souhaitée.

B.4 Clear Timer on Compare (CTC) mode

Ce mode permet de placer une valeur de référence et dès que le compteur atteint cette dernière, le bit se met en mode HIGH et il permet d'effectuer des tâches dans le code.
C'est une manière de gagner en temps machine, plutôt que de mettre une fonction « if » on peut utiliser cette méthode pour aller plus vite.

Figure 14-5. CTC Mode, Timing Diagram

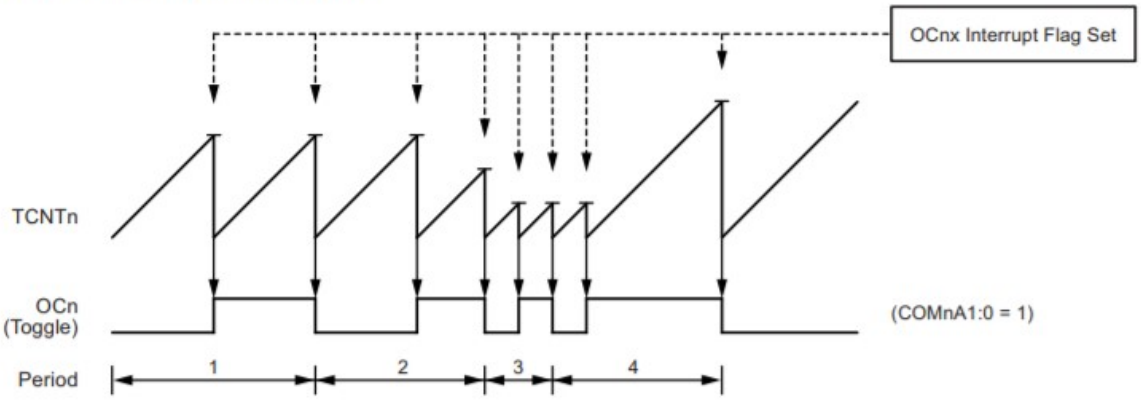

FIGURE 7 – Diagramme des temps du mode CTC

Ce mode est régi par la formule suivante :

$$f_{OCnA} = \frac{f_{clk\_I/O}}{2 \times N \times (1 + OCRnA)}$$

The *N* variable represents the prescaler factor (1, 8, 64, 256, or 1024).

FIGURE 8 – Formule pour l'obtention de la fréquence de sortie du mode CTC

Avec f_{OCnA} la fréquence souhaitée, $f_{clkI/O}$ la fréquence du quartz qui est normalement de 16 MHz (à vérifier sur l'arduino) et OCRnA le duty cycle d'un des pins.

C Exemples d'application du PWM

C.1 Fast PWM mode et Timer 2

Fréquence obtenue : 1.960 kHz

Codage

```
1  void setup() {
2  pinMode(3, OUTPUT);
3  pinMode(11,OUTPUT);
4
5  TCCR2A=0;//reset the register
6  TCCR2B=0;//reset the register
7  TCCR2A=0b10100011; // fast pwm mode
8  TCCR2B=0b00000011; // prescaler 32
9  OCR2A=50;//duty cycle for pin 11
10 OCR2B=50;//duty cycle for pin 3
11
12 }
13 void loop() {
14 // put your main code here, to run repeatedly
15 }
```

FIGURE 9 – Mesures de l'oscilloscope du Fast PWM

C.2 Phase Correct PWM mode et Timer 2

Fréquence obtenue : 490.2 Hz

Codage

```
1  void setup() {
2  pinMode(3, OUTPUT);
3  pinMode(11,OUTPUT);
4
5  TCCR2A=0;//reset the register
6  TCCR2B=0;//reset the register
7  TCCR2A=0b10100001; // phase correct pwm mode
8  TCCR2B=0b00000100; // prescaler 64
9  OCR2A=50;//duty cycle for pin 11
10 OCR2B=50;//duty cycle for pin 3
11
12 }
13 void loop() {
14 // put your main code here, to run repeatedly
15 }
```

FIGURE 10 – Mesures de l'oscilloscope du Phase Correct PWM mode

C.3 CTC mode et Timer 1 – ALLUMAGE DE LED

Fréquence obtenue : 25 Hz

Codage

```
1  volatile int isr1 = 0;
2
3  ISR(TIMER1_COMPA_vect)
4  {
5    // Fonction appele chaque fois que timer compare galise ICR1 ou OCR1A
6    isr1++;
7  }
8
9  void setup()
10 {
11   pinMode (2,OUTPUT);
12   pinMode (9,OUTPUT);
13   TCCR1A = 0b10100000;
14   TCCR1B =0B00001010; //prescaler 8;
15   TCCR1C = 0;
16   TIMSK1 |= 1 << OCIE1A; // bit 4 : Met en place le TimerOutput CompareA
         Match Interrupt
17   OCR1A=39999;//valeur de comparaison.
18   Serial.begin(115200);
19 }
20
21 void loop()
22 {
23       if (isr1 == 50)      //ISR() galise 2 fois par priode
24     {
25       if (digitalRead(2)==HIGH) digitalWrite(2,LOW);
26       else if (digitalRead(2) == LOW) digitalWrite(2,HIGH);
27       isr1=0;
28   }
29 }
```

Cette méthode permet d'utiliser un processus qui est interne au composant ATMEGA 328P. Ainsi, notre action se fait beaucoup plus vite qu'en utilisant une comparaison du type « if . . . else . . . ».

De plus, la fonction ISR() tourne en parallèle de la fonction void loop() ce qui fait qu'il n'y a pas d'interférences ni de latences entre les deux instructions.

L'utilisation des interruptions machines sont donc très utiles pour effectuer des tâches plus rapidement et indépendamment du code entré dans void loop().

D Modi cation de la fréquence de l'Arduino

Une simple méthode pour changer la fréquence de l'horloge interne de l'Arduino est de modi er les diviseurs (prescaler) dans le code.

Cela peut être fait en modi ant le registre CLKPR sur deux lignes comme dans le code ci-dessous. Il est important d'arrêter les interruptions car il s'agit d'une séquence faisant intervenir le temps et des interférences peuvent se créer.

La première ligne authorise la modi cation de la fréquence et la seconde de changer le diviseur parmi la liste suivante :

clock_div_1

clock_div_2

clock_div_4

clock_div_8

clock_div_16

clock_div_32

clock_div_64

clock_div_128

clock_div_256

Codage

```
#include <avr/power.h>

void setup ()
  {
  noInterrupts ();
  CLKPR = bit (CLKPCE);
  CLKPR = clock_div_256;
  interrupts ();

  // disable ADC
  ADCSRA = 0;
  power_all_disable ();
  } // end of setup

void loop () { }
```

Le diviseur par défaut est normalement 1 et parfois il peut être 8. Dans tous les cas il est possible de le modi er à sa guise parmi les diviseurs énoncés plus haut. Il est préférable de mettre le diviseur à 8 si l'on veut alimenter l'Arduino avec des tensions plus basses que 5V.

Les tests suivants ont été fait avec une source d'alimentation de 3.3V :

clock_div_1 - 3.1mA

clock_div_2 - 1.8 mA

clock_div_4 - 1.1 mA

clock_div_8 - 750 µA

clock_div_16 - 550 µA

clock_div_32 - 393 µA

clock_div_64 - 351 µA

clock_div_128 - 296 µA

clock_div_256 - 288 µA

Dans le cas du diviseur égal à 256, on peut alimenter l'Arduino avec une tension allant jusqu'à 1.8V ce qui résulte en une consommation de 160 µA.

Tous ces tests ont été faits sans même avoir utilisé les modes d'endormissement de l'Arduino ce qui veut dire qu'il est possible de consommer très peu en jouant sur la fréquence interne de l'Arduino.

Attention

La modification des diviseurs change l'échelle des temps. Si on choisit un **diviseur égal à 2**, la communication série se fera deux fois plus lentement que le baud rate demandé. L'utilisation du delay(100) fera en réalité une pause de 200 ms.

Une alternative plus courte pour modifier le registre CLKPR est d'utiliser la macro de power.h :

Codage

```
#include <avr/power.h>

void setup ()
  {
  // slow clock to divide by 256
  clock_prescale_set (clock_div_256);

  // disable ADC
  ADCSRA = 0;
  power_all_disable ();
  } // end of setup

void loop () { }
```

E Pour aller plus loin

Il existe donc différentes manières d'aborder le sujet des PWM et de la modification de leurs fréquences. Toutefois pour en savoir plus sur toutes les possibilités auxquelles nous pouvons aboutir, je vous invite à consulter les tutoriels suivants qui sont relativement complets :

http://www.eprojectszone.com/how-to-modify-the-pwm-frequency-on-the-ar

http://www.eprojectszone.com/how-to-modify-the-pwm-frequency-on-the-ardu

http://www.eprojectszone.com/how-to-modify-the-pwm-frequency-on-the-ardui

Aussi, il faut comprendre que des modifications dans les registres entraînent quelques changements dans des fonctions comme millis() ou delay() :

https://playground.arduino.cc/Code/PwmFrequency/

F Bibliographie

https://datasheets.maximintegrated.com/en/ds/DS3231.pdf

http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-M
Datasheet.pdf

https://arduino.stackexchange.com/questions/43139/arduino-timer-ctc-mod