Comment changer la fréquence PWM d'Arduino

: Guide épique Robert Brun·11 juin 2021

Le microcontrôleur possède plusieurs temporisateurs qui peuvent exécuter différentes fonctions, telles que la génération d'un signal PWM . Pour que le temporisateur génère un signal PWM, il doit être préconfiguré en éditant le registre du temporisateur. Lorsque nous travaillons dans l'IDE Arduino, les minuteries sont configurées à notre insu dans la bibliothèque Arduino.h et obtiennent en fait les paramètres souhaités par les développeurs. Et ces paramètres ne sont pas très bons : la fréquence PWM par défaut est faible et les minuteries ne sont pas utilisées à leur plein potentiel. Regardons le PWM standard de l'ATmega328 (Arduino UNO/ Nano / Pro Mini) :

Minuteur	Épingles	Fréquence	Résolution
Minuterie 0	D5 et D6	976Hz	8 bits (0-255)
Minuterie 1	D9 et D10	488Hz	8 bits (0-255)
Minuterie 2	D3 et D11	488Hz	8 bits (0-255)

En fait, tous les temporisateurs peuvent facilement émettre un signal PWM de 64 kHz , et le temporisateur 1 – c'est même 16 bits, et à la fréquence qui lui a été donnée Arduino, pourrait fonctionner avec une résolution de 15 bits au lieu de 8, et cela, soit dit en passant, 32768 gradations de remplissage au lieu de 256 ! Alors pourquoi cette injustice ? La minuterie 0 est en charge de la synchronisation et est réglée de manière à ce que les millisecondes s'écoulent avec précision. Les autres minuteries sont ramenées à zéro pour éviter que l'amateur d'Arduino n'ait des problèmes inutiles. Cette approche est généralement compréhensible mais aurait fait au moins quelques fonctions standard pour une fréquence plus élevée, eh bien, sérieusement ! D'accord, s'ils ne l'ont pas fait, nous le ferons.

Réglage de la fréquence PWM via les registres

La génération PWM est réglée via les registres de temporisation. Ensuite, vous trouverez des “morceaux” de code prêts à l'emploi, que vous devez insérer dans setup(), et la fréquence PWM sera reconfigurée (le pré-délimiteur et le mode minuterie changeront). Vous pouvez toujours travailler avec le signal PWM avec la analogWrite()fonction, contrôlant le remplissage du PWM sur les broches standard.

Modification de la fréquence PWM sur l'ATmega328 (Arduino UNO/Nano/Pro Mini)

Broches D5 et D6 (Timer 0) – 8 bits

pwmD5D6.ino

```
// Pins D5 and D6 are 62.5kHz
TCCR0B = 0b00000001; // x1
TCCR0A = 0b00000011; // fast pwm
// Pins D5 and D6 - 31.4 kHz
TCCR0B = 0b00000001; // x1
TCCR0A = 0b00000001; // phase correct
// Pins D5 and D6 - 7.8 kHz
TCCR0B = 0b00000010; // x8
TCCR0A = 0b00000011; // fast pwm
// Pins D5 and D6 - 4 kHz
TCCR0B = 0b00000010; // x8
TCCR0A = 0b00000001; // phase correct
// Pins D5 and D6 - 976 Hz - default
TCCR0B = 0b00000011; // x64
TCCR0A = 0b00000011; // fast pwm
// Pins D5 and D6 - 490 Hz
TCCR0B = 0b00000011; // x64
TCCR0A = 0b00000001; // phase correct
// Pins D5 and D6 - 244 Hz
TCCR0B = 0b00000100; // x256
TCCR0A = 0b00000011; // fast pwm
// Pins D5 and D6 - 122 Hz
TCCR0B = 0b00000100; // x256
TCCR0A = 0b00000001; // phase correct
// Pins D5 and D6 - 61 Hz
TCCR0B = 0b00000101; // x1024
TCCR0A = 0b00000011; // fast pwm
// Pins D5 and D6 - 30 Hz
TCCR0B = 0b00000101; // x1024
TCCR0A = 0b00000001; // phase correct
```

Broches D9 et D10 (Timer 1) – 8 bits

pwmD9D108B.ino

```
// Broches D9 et D10 - 62,5 kHz
TCCR1A = 0b00000001 ; // 8 bits
TCCR1B = 0b00001001 ; // x1 pwm rapide
// Broches D9 et D10 - 31,4 kHz
TCCR1A = 0b00000001 ; // 8 bits
TCCR1B = 0b00000001 ; // phase x1 correcte
// Broches D9 et D10 - 7,8 kHz
TCCR1A = 0b00000001 ; // 8 bits
TCCR1B = 0b00001010 ; // x8 rapide pwm
// Broches D9 et D10 - 4 kHz
TCCR1A = 0b00000001 ; // 8 bits
```

```
TCCR1B = 0b00000010 ; // phase x8 correcte
// Broches D9 et D10 - 976 Hz
TCCR1A = 0b00000001 ; // 8 bits
TCCR1B = 0b00001011 ; // pwm rapide x64
// Broches D9 et D10 - 490 Hz - par défaut
TCCR1A = 0b00000001 ; // 8 bits
TCCR1B = 0b00000011 ; // phase x64 correcte
// Broches D9 et D10 - 244 Hz
TCCR1A = 0b00000001 ; // 8 bits
TCCR1B = 0b00001100 ; // x256 pwm rapide
// Broches D9 et D10 - 122 Hz
TCCR1A = 0b00000001 ; // 8 bits
TCCR1B = 0b00000100 ; // phase x256 correcte
// Broches D9 et D10 - 61 Hz
TCCR1A = 0b00000001 ; // 8 bits
TCCR1B = 0b00001101 ; // x1024 pwm rapide
// Broches D9 et D10 - 30 Hz
TCCR1A = 0b00000001 ; // 8 bits
TCCR1B = 0b00000101 ; // phase x1024 correcte
```

Broches D9 et D10 (minuterie 1) – 10 bits

pwmD9D1010B.ino

```
// Broches D9 et D10 - 15,6 kHz 10 bits
TCCR1A = 0b00000011 ; // 10 bits
TCCR1B = 0b00001001 ; // x1 pwm rapide
// Broches D9 et D10 - 7,8 kHz 10 bits
TCCR1A = 0b00000011 ; // 10 bits
TCCR1B = 0b00000001 ; // phase x1 correcte
// Broches D9 et D10 - 2 kHz 10 bits
TCCR1A = 0b00000011 ; // 10 bits
TCCR1B = 0b00001010 ; // x8 rapide pwm
// Broches D9 et D10 - 977 Hz 10 bits
TCCR1A = 0b00000011 ; // 10 bits
TCCR1B = 0b00000010 ; // phase x8 correcte
// Broches D9 et D10 - 244 Hz 10 bits
TCCR1A = 0b00000011 ; // 10 bits
TCCR1B = 0b00001011 ; // pwm rapide x64
// Broches D9 et D10 - 122 Hz 10 bits
TCCR1A = 0b00000011 ; // 10 bits
TCCR1B = 0b00000011 ; // phase x64 correcte
// Broches D9 et D10 - 61 Hz 10 bits
TCCR1A = 0b00000011 ; // 10 bits
TCCR1B = 0b00001100 ; // x256 pwm rapide
// Broches D9 et D10 - 30 Hz 10 bits
TCCR1A = 0b00000011 ; // 10 bits
TCCR1B = 0b00000100 ; // phase x256 correcte
```

```
// Broches D9 et D10 - 15 Hz 10 bits
TCCR1A = 0b00000011 ; // 10 bits
TCCR1B = 0b00001101 ; // x1024 pwm rapide
// Broches D9 et D10 - 7,5 Hz 10 bits
TCCR1A = 0b00000011 ; // 10 bits
TCCR1B = 0b00000101 ; // phase x1024 correcte
```

Broches D3 et D11 (Timer 2) – 8 bits

pwmD3D118B.ino

```
// Broches D3 et D11 - 62,5 kHz
TCCR2B = 0b00000001 ; // x1
TCCR2A = 0b00000011 ; // pwm rapide
// Broches D3 et D11 - 31,4 kHz
TCCR2B = 0b00000001 ; // x1
TCCR2A = 0b00000001 ; // phase correcte
// Broches D3 et D11 - 8 kHz
TCCR2B = 0b00000010 ; // x8
TCCR2A = 0b00000011 ; // pwm rapide
// Broches D3 et D11 - 4 kHz
TCCR2B = 0b00000010 ; // x8
TCCR2A = 0b00000001 ; // phase correcte
// Broches D3 et D11 - 2 kHz
TCCR2B = 0b00000011 ; // x32
TCCR2A = 0b00000011 ; // pwm rapide
// Broches D3 et D11 - 980 Hz
TCCR2B = 0b00000011 ; // x32
TCCR2A = 0b00000001 ; // phase correcte

// Broches D3 et D11 - 980 Hz
TCCR2B = 0b00000100 ; // x64
TCCR2A = 0b00000011 ; // pwm rapide
// Broches D3 et D11 - 490 Hz - par défaut
TCCR2B = 0b00000100 ; // x64
TCCR2A = 0b00000001 ; // phase correcte
// Broches D3 et D11 - 490 Hz
TCCR2B = 0b00000101 ; // x128
TCCR2A = 0b00000011 ; // pwm rapide
// Broches D3 et D11 - 245 Hz
TCCR2B = 0b00000101 ; // x128
TCCR2A = 0b00000001 ; // phase correcte
// Broches D3 et D11 - 245 Hz
TCCR2B = 0b00000110 ; // x256
TCCR2A = 0b00000011 ; // pwm rapide
// Broches D3 et D11 - 122 Hz
TCCR2B = 0b00000110 ; // x256
TCCR2A = 0b00000001 ; // phase correcte
```

```
// Broches D3 et D11 - 60 Hz
TCCR2B = 0b00000111 ; // x1024
TCCR2A = 0b00000011 ; // pwm rapide
// Broches D3 et D11 - 30 Hz
TCCR2B = 0b00000111 ; // x1024
TCCR2A = 0b00000001 ; // phase correcte
```

Exemple d'utilisation

exemple1.ino

```
void setup() {
  // Pins D5 and D6 - 7.8 kHz
  TCCR0B = 0b00000010; // x8
  TCCR0A = 0b00000011; // fast pwm
  // Pins D3 and D11 - 62.5 kHz
  TCCR2B = 0b00000001; // x1
  TCCR2A = 0b00000011; // fast pwm
  // Pins D9 and D10 - 7.8 kHz 10bit
  TCCR1A = 0b00000011; // 10bit
  TCCR1B = 0b00000001; // x1 phase correct
  analogWrite(3, 15);
  analogWrite(5, 167);
  analogWrite(6, 241);
  analogWrite(9, 745); // yes, range 0-1023
  analogWrite(10, 345); // yes, range 0-1023
  analogWrite(11, 78);
}
void loop() {
}
```

Si vous changez la fréquence sur les broches D5 et D6, vous perdrez les fonctions de temps (millis(), delay(), pulseIn(), setTimeout(), etc.), elles ne fonctionneront pas correctement. De plus, les bibliothèques qui les utilisent cesseront de fonctionner !

Changé les frequences Timer 0

Si vous voulez ou avez vraiment besoin d'un PWM overclocké sur le temporisateur système (zéro) sans perte de fonctions temporelles, vous pouvez les corriger comme suit :

timerO.ino

```
#define micros() (micros() >> CORRECT_CLOCK)
#define millis() (millis() >> CORRECT_CLOCK)
```

```
void fixDelay(uint32_t ms) {
  delay(ms << CORRECT_CLOCK);
}
```

Les définitions doivent être placées avant de brancher les bibliothèques afin qu'elles entrent dans le code et remplacent les fonctions. La seule chose est que vous ne pouvez pas corriger le retard dans une autre bibliothèque de cette façon. Vous pouvez utiliser fixDelay() pour vous-même comme indiqué ci-dessus.

La chose la plus importante est CORRECT_CLOCK. Il s'agit d'un nombre entier égal au rapport entre le diviseur de minuterie par défaut et le nouveau défini (pour l'accélération PWM). Par exemple, nous réglons le PWM à 8 kHz. Dans la liste ci-dessus, nous voyons que le diviseur par défaut est 64, et 7,8 kHz sera 8, ce qui est huit fois plus petit. CORRECT_CLOCK est défini en conséquence.

correctclock.ino

```
#define CORRECT_CLOCK 8
void fixDelay(uint32_t ms) {
  delay(ms << CORRECT_CLOCK);
}
void setup() {
  pinMode(13, 1);
  // Pins D5 and D6 - 4 kHz
  TCCR0B = 0b00000010; // x8
  TCCR0A = 0b00000001; // phase correct
}
void loop() {
  digitalWrite(13, !digitalRead(13));
  fixDelay(1000);
}
```

Bibliothèques pour travailler avec PWM

En plus de jouer manuellement avec les registres, il existe des bibliothèques prêtes à l'emploi qui vous permettent de modifier la fréquence PWM de l'Arduino. Jetons un coup d'œil à certains d'entre eux :

PWM library (GitHub) - une bibliothèque puissante qui vous permet de modifier la fréquence PWM sur les microcontrôleurs ATmega48 / 88 / 168 / 328 / 640 / 1280 / 1281 / 2560 / 2561, dont 328 sur UNO/Nano/Mini et 2560 sur un Arduino Méga.

- Vous permet de définir n'importe quelle fréquence PWM, pré-retard, TOP
- Un seul canal est disponible lorsque vous travaillez avec des minuteries 8 bits (par exemple, sur l'ATmega328, uniquement D3, D5, D9 et D10)
- Permet de travailler avec des temporisateurs 16 bits à une résolution plus élevée (16 bits au lieu du standard 8)

- La bibliothèque est très compliquée, elle ne peut donc pas être déchiquetée en morceaux.
- Voir exemples dans le dossier avec la bibliothèque !

GyverPWM library(GitHub) - la bibliothèque que nous avons écrite avec mon ami. La bibliothèque permet un travail très flexible avec PWM sur le microcontrôleur ATmega328 (nous ajouterons Mega plus tard):

- Vous permet de définir n'importe quelle fréquence PWM dans la plage de 250 Hz à 200 kHz
- Sélection de bits : 4-8 bits pour les temporisateurs 8 bits, 4-16 bits pour les temporisateurs 16 bits (à 4 bits, la fréquence PWM est de 1 MHz)
- Sélection du mode PWM : Fast PWM ou Phase-correct PWM (favorable aux moteurs)
- Génération de fréquences de méandres de 2 Hz à 8 MHz sur la broche D9 avec une précision maximale
- Un seul canal est disponible lorsque vous travaillez avec des minuteries 8 bits (par exemple, sur un ATmega328, uniquement D3, D5, D9 et D10)
- Il existe des fonctions pour reconfigurer les sorties PWM standard sans perdre le PWM
- La bibliothèque est écrite de manière très simple, et vous pouvez en prendre des morceaux de code
- Voir exemples dans le dossier avec la bibliothèque !

From:
https://www.fablab37110.chanterie37.fr/ - **Castel'Lab le Fablab MJC de Château-Renault**

Permanent link:
https://www.fablab37110.chanterie37.fr/doku.php?id=start:arduino:pwm&rev=1671543995

Last update: **2023/01/27 16:08**