NERDYTECHY

How to Change the PWM Frequency Of Arduino: Epic Guide

Arduino / By Robert Brown / 5 Comments / June 11, 2021

The microcontroller has several timers that can perform different functions, such as generating a PWM signal. In order for the timer to generate a PWM signal, it has to be pre-configured by editing the timer register. When we work in the Arduino IDE, the timers are configured without our knowledge in the Arduino.h library, and actually get the settings the developers wanted. And these settings are not very good: the default PWM frequency is low, and the timers are not used to their full potential. Let's look at the standard PWM of the ATmega328 (Arduino UNO/Nano/Pro Mini):

Timer	Pins	Frequency	Resolution
Timer 0	D5 and D6	976 Hz	8 bits (0- 255)
Timer 1	D9 and D10	488 Hz	8 bits (0-255)
Timer 2	D3 and D11	488 Hz	8 bits (0-255)

In fact, all timers can easily give out **64 kHz PWM signal**, and timer 1 – it is even 16 bits, and at the frequency that was given to him Arduino, could work with a resolution of 15 bits instead of 8, and that, by the way, 32768 gradations of filling instead of 256! So why this injustice? Timer 0 is in charge of timing and is set so that the milliseconds are ticking precisely. The other timers are combed to zero to prevent the Arduino-enthusiast from having unnecessary problems. This approach is generally understandable but would have made at least a couple of standard functions for a higher frequency, well, seriously! Okay, if they didn't, we will.

Contents [show]

still work with the PWM signal with the `analogWrite()` function, controlling the filling of the PWM on the standard pins.

Changing the PWM Frequency on the ATmega328 (Arduino UNO/Nano/Pro Mini)

Pins D5 and D6 (Timer 0) – 8 bits

```
// 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
```

Pins D9 and D10 (Timer 1) – 8 bits

```
// Pins D9 and D10 - 62.5 kHz
TCCR1A = 0b00000001; // 8bit
TCCR1B = 0b00001001; // x1 fast pwm
// Pins D9 and D10 - 31.4 kHz
TCCR1A = 0b00000001; // 8bit
TCCR1B = 0b00000001; // x1 phase correct
// Pins D9 and D10 - 7.8 kHz
TCCR1A = 0b00000001; // 8bit
TCCR1B = 0b00001010; // x8 fast pwm
// Pins D9 and D10 - 4 kHz
TCCR1A = 0b00000001; // 8bit
TCCR1B = 0b00000010; // x8 phase correct
// Pins D9 and D10 - 976 Hz
TCCR1A = 0b00000001; // 8bit
TCCR1B = 0b00001011; // x64 fast pwm
// Pins D9 and D10 - 490 Hz - default
TCCR1A = 0b00000001; // 8bit
TCCR1B = 0b00000011; // x64 phase correct
// Pins D9 and D10 - 244 Hz
TCCR1A = 0b00000001; // 8bit
TCCR1B = 0b00001100; // x256 fast pwm
// Pins D9 and D10 - 122 Hz
TCCR1A = 0b00000001; // 8bit
TCCR1B = 0b00000100; // x256 phase correct
// Pins D9 and D10 - 61 Hz
TCCR1A = 0b00000001; // 8bit
TCCR1B = 0b00001101; // x1024 fast pwm
// Pins D9 and D10 - 30 Hz
TCCR1A = 0b00000001; // 8bit
TCCR1B = 0b00000101; // x1024 phase correct
```

Pins D9 and D10 (Timer 1) – 10 bit

```
// Pins D9 and D10 - 15.6 kHz 10bit
TCCR1A = 0b00000011; // 10bit
TCCR1B = 0b00001001; // x1 fast pwm
```

```
TCCR1A = 0b00000011; // 10bit
TCCR1B = 0b00000010; // x8 phase correct
// Pins D9 and D10 - 244 Hz 10bit
TCCR1A = 0b00000011; // 10bit
TCCR1B = 0b00001011; // x64 fast pwm
// Pins D9 and D10 - 122 Hz 10bit
TCCR1A = 0b00000011; // 10bit
TCCR1B = 0b00000011; // x64 phase correct
// Pins D9 and D10 - 61 Hz 10bit
TCCR1A = 0b00000011; // 10bit
TCCR1B = 0b00001100; // x256 fast pwm
// Pins D9 and D10 - 30 Hz 10bit
TCCR1A = 0b00000011; // 10bit
TCCR1B = 0b00000100; // x256 phase correct
// Pins D9 and D10 - 15 Hz 10bit
TCCR1A = 0b00000011; // 10bit
TCCR1B = 0b00001101; // x1024 fast pwm
// Pins D9 and D10 - 7.5 Hz 10bit
TCCR1A = 0b00000011; // 10bit
TCCR1B = 0b00000101; // x1024 phase correct
```

Pins D3 and D11 (Timer 2) – 8 bits

```
// Pins D3 and D11 - 62.5 kHz
TCCR2B = 0b00000001; // x1
TCCR2A = 0b00000011; // fast pwm
// Pins D3 and D11 - 31.4 kHz
TCCR2B = 0b00000001; // x1
TCCR2A = 0b00000001; // phase correct
// Pins D3 and D11 - 8 kHz
TCCR2B = 0b00000010; // x8
TCCR2A = 0b00000011; // fast pwm
// Pins D3 and D11 - 4 kHz
TCCR2B = 0b00000010; // x8
TCCR2A = 0b00000001; // phase correct
// Pins D3 and D11 - 2 kHz
TCCR2B = 0b00000011; // x32
TCCR2A = 0b00000011; // fast pwm
// Pins D3 and D11 - 980 Hz
TCCR2B = 0b00000011; // x32
TCCR2A = 0b00000001; // phase correct
// Pins D3 and D11 - 980 Hz
TCCR2B = 0b00000100; // x64
TCCR2A = 0b00000011; // fast pwm
// Pins D3 and D11 - 490 Hz - default
TCCR2B = 0b00000100; // x64
TCCR2A = 0b00000001; // phase correct
// Pins D3 and D11 - 490 Hz
TCCR2B = 0b00000101; // x128
TCCR2A = 0b00000011; // fast pwm
// Pins D3 and D11 - 245 Hz
TCCR2B = 0b00000101; // x128
TCCR2A = 0b00000001; // phase correct
// Pins D3 and D11 - 245 Hz
TCCR2B = 0b00000110; // x256
TCCR2A = 0b00000011; // fast pwm
// Pins D3 and D11 - 122 Hz
TCCR2B = 0b00000110; // x256
TCCR2A = 0b00000001; // phase correct
// Pins D3 and D11 - 60 Hz
TCCR2B = 0b00000111; // x1024
TCCR2A = 0b00000011; // fast pwm
// Pins D3 and D11 - 30 Hz
TCCR2B = 0b00000111; // x1024
TCCR2A = 0b00000001; // phase correct
```

Example of Usage

```
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
```

```
}
void loop() {
}
```

Important!
If you change the frequency on pins D5 and D6, you will lose the time functions (`millis()`, `delay()`, `pulseIn()`, `setTimeout()`, etc.), they will not work correctly. Also, the libraries that use them will stop working!
If you really want or need an overclocked PWM on the system (zero) timer without loss of time functions, you can correct them as follows:

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

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

Defines should be placed before plugging in the libraries so that they get into the code and substitute functions. The only thing is that you can not correct the delay in another library this way. You can use `fixDelay()` for yourself as written above.

The most important thing is `CORRECT_CLOCK`. This is an integer equal to the ratio of the default timer divider and the new one set (for PWM acceleration). For example, we set the PWM to 8 kHz. From the list above, we see that the default divider is 64, and 7.8 kHz will be 8, which is eight times smaller. CORRECT_CLOCK is set accordingly.

```
#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);
}
```

Libraries for Working with PWM

In addition to fiddling with the registers manually, there are ready-made libraries that allow you to change the PWM frequency of the Arduino. Let's take a look at some of them:

PWM library (GitHub) – a powerful library that allows you to change the PWM frequency on ATmega48 / 88 / 168 / 328 / 640 / 1280 / 1281 / 2560 / 2561 microcontrollers, of which 328 is on UNO/Nano/Mini and 2560 is an Arduino Mega.

- Allows you to set any PWM frequency, pre-delay, TOP
- Only one channel is available when working with 8-bit timers (for example, on the ATmega328, only D3, D5, D9, and D10)
- Allows to work with 16-bit timers at a higher resolution (16 bits instead of the standard 8)
- The library is very complicated, so it can't be shredded into pieces.
- See examples in the folder with the library!

GyverPWM library (GitHub) – the library, which we wrote together with my friend. The library allows very flexible work with PWM on microcontroller ATmega328 (we will add Mega later):

- Allows you to set any PWM frequency in the range of 250 Hz – 200 kHz
- Bit selection: 4-8 bits for 8-bit timers, 4-16 bits for 16-bit timers (at 4 bits, the PWM frequency is 1 MHz)
- PWM mode selection: Fast PWM or Phase-correct PWM (favorable for motors)
- Generation of meander frequencies from 2 Hz to 8 MHz on pin D9 with maximum accuracy
- Only one channel is available when working with 8-bit timers (for example, on an ATmega328, only D3, D5, D9, and D10)
- There are functions to reconfigure the standard PWM outputs without losing the PWM

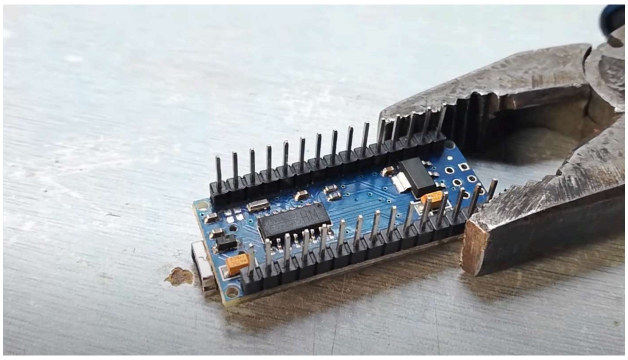
By following the steps above, you can change the PWM frequency on your Arduino. This can be useful for controlling motors or other devices that require a specific frequency to function correctly. Thanks for reading!

← Previous Post

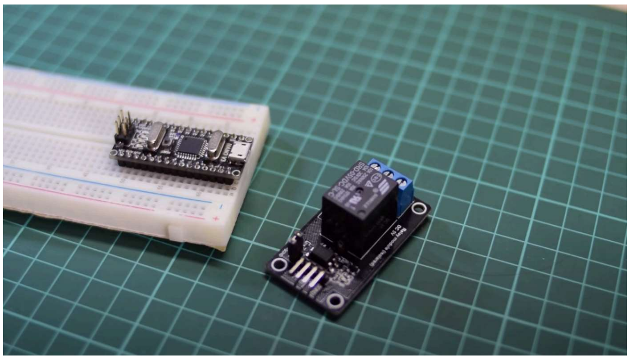
Next Post →

Related Posts

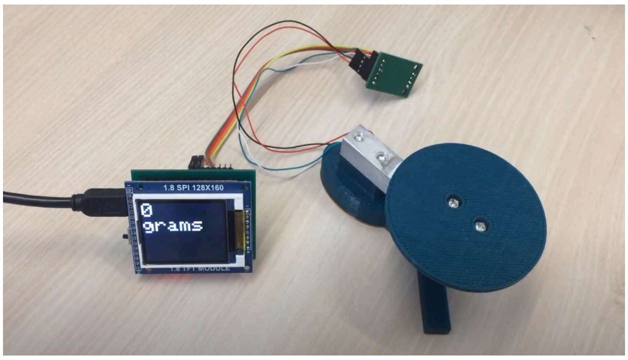
How to Build a Plex Server (Longread)



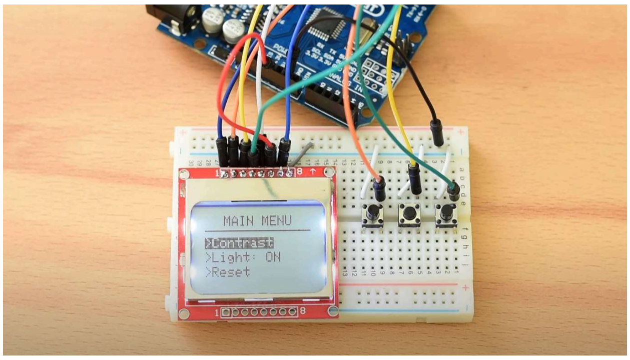
Arduino Nano Pinout for Beginners



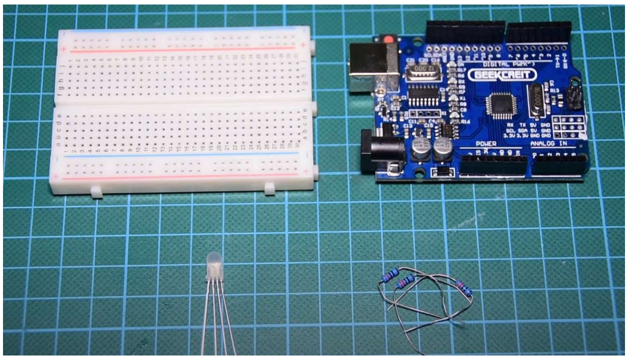
Arduino Relay Tutorial for Beginners



Connecting HX711 Load Cell to Arduino



Connecting Nokia 5110 LCD with Arduino



How to Connect LED to Arduino: Epic Guide

5 thoughts on "How to Change the PWM Frequency Of Arduino: Epic Guide"

Thanks for this. I'm driving some motors with a Mega board and they whine worse than my ex-wife. There is a big jump from the 4 Khz to the 32 Khz values. Is there the possibility to set the frequency somewhere between 16 and 24 Khz.
Thanks,
John

Reply

ROBERT BROWN
APRIL 20, 2022 AT 6:41 AM

Hi John,
You can set PWM frequency using PWM library, check this example.

Reply

DAVE
APRIL 20, 2022 AT 1:47 AM

Great article. Did you add support or the Mega? I went to GitHub but it is written in Russian (?). Thanks in advance.

Reply

ROBERT BROWN
APRIL 20, 2022 AT 6:44 AM

Hi Dave,
I'll update this article when I have free time

Reply

SAM
MAY 10, 2023 AT 7:20 PM

I appreciated that the article provided step-by-step instructions and even included code snippets to simplify the process. The author also took the time to explain the importance of changing the PWM frequency and the benefits it can provide.

Overall, I found this article to be a great resource for anyone looking to change the PWM frequency of their Arduino. It was well-written, easy to understand, and provided all the necessary information needed to complete the task successfully. I highly recommend giving it a read!

Reply

Leave a Comment

Your email address will not be published. Required fields are marked *

Name*

Email*

Website

Save my name, email, and website in this browser for the next time I comment.

Post Comment

Search...

Recent Posts

- Ohm's Law Calculator Online
- How to Send Command to Raspberry Pi Over Internet?
- What to Do After Replacing Camshaft Sensor?
- Does Solder Expire?
- Film vs. Ceramic Capacitor: What's the Difference?

Recent Posts

- Ohm's Law Calculator Online
- How to Send Command to Raspb
Internet?
- What to Do After Replacing Cams
- Does Solder Expire?
- Film vs. Ceramic Capacitor: What's

Categories

- Calculators
- Pinouts
- Reviews
- Tutorials
- Useful Info
- Versus

Information

- About Me
- Contact Me
- Privacy Policy
- Terms and Conditions
- Sitemap
- Jobs

Tags

Arduino Batteries Ham Radio
Multimeters Orange Pi
Oscilloscopes Raspberry Pi
Sensors Soldering