

Display

- [Blogthoughts, tutorials](#)
- [Articlesdev articles](#)
- [Labpet dev projects](#)
- [Hobbiesworkshop fun](#)
- [Wallpaperdesktop images](#)

Welcome to **Pete Brown's 10rem.net**

First time here? If you are a developer or are interested in Microsoft tools and technology, please consider [subscribing to the latest posts](#).

You may also be interested in my [blog archives](#), the [articles section](#), or some of my [lab projects](#) such as the C64 emulator written in Silverlight.

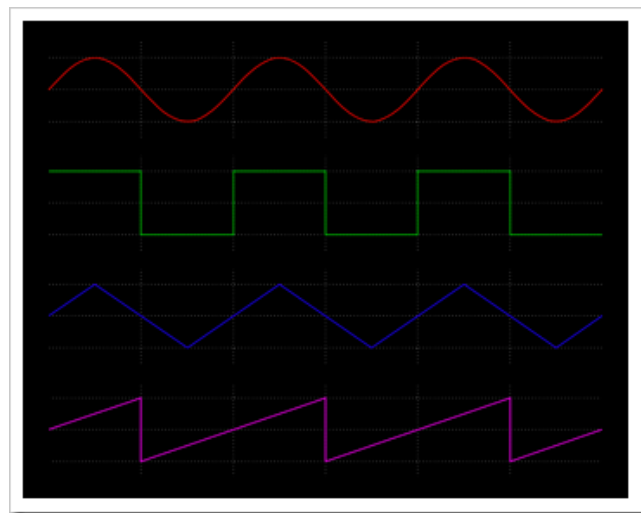
([hide this](#))

27 Sep 2010

Netduino Basics: Using Pulse Width Modulation (PWM)

[.NET](#) [Hardware](#) [geek](#) [Netduino](#) [Micro+Framework](#)

[Pulse Width Modulation \(PWM\)](#) is the process of alternating a signal between two extreme values with little to no intermediate steps. On the Netduino, these values are 0v and 3.3v. In its most basic form, the output resembles a square wave, where both the min and max values have the same duration.



The square wave (which can be created with PWM) is the second down. The others, from the top are: sine wave, triangle wave and sawtooth wave. The place I always ran into these was in synthesizer work. For example, I supported all four waveforms in my [Silverlight Synthesizer](#) project.

For some of you, this may seem insanely simple, but for those of us just starting out in electronics, this is quite an adventure :)

PWM is used to control a number of things from servo motors (PWM is used in my CNC setup) to the level of output in LEDs to, what I intend to use it for shortly, serving as a clock signal for another chip. In my case, the chip will be a MOS 8580 SID chip [salvaged from an old Commodore 64c](#). That'll be detailed in a later post once I get things working.

PWM on the Netduino

Four of the output ports (Digital pins 5, 6, 9 and 10) on the Netduino may be used for PWM signals. I'm using a new Netduino Plus beta, but the [tech specs](#) in this area are identical. PWM in the current firmware release is somewhat limited, but from reading Chris's posts on the forums the 4.1.1 revision of the firmware should have some real advancements in these areas.

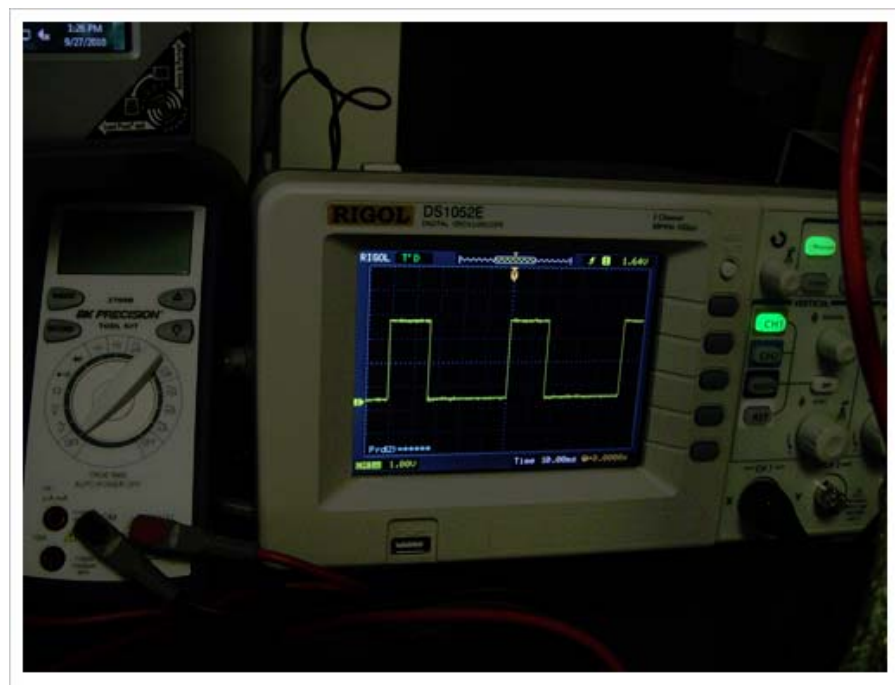
Currently, the standard PWM clock is set to 10Khz. If you set the duty cycle, it will be based on that clock. One of the expected upcoming changes will be to have finer control over the clock, and hopefully support Mhz clock and PWM rates.

Using PWM is pretty simple. Instead of creating an OutputPort, you'll create an instance of the PWM class. Once you do that, you can either set the Duty Cycle or call SetPulse.

In this case, we'll call SetPulse and have the pulse on 1/3 of the time (basically a 33% duty cycle)

```
1  public class Program
2  {
3      public static void Main()
4      {
5          PWM pwm = new PWM(Pins.GPIO_PIN_D5);
6
7          const uint period = 3 * 1000 * 1000;    // 3 ms
8          const uint duration = 1 * 1000 * 1000;  // 1 ms
9
10         pwm.SetPulse(period, duration);
11     }
12 }
13 }
```

Doing that, creates the following waveform:



(the scope lets me save the image to a USB key, but I just unboxed it today. Photo snapshots will need to suffice for now)

The first parameter of SetPulse is the period, the second is the duration. Some definitions:

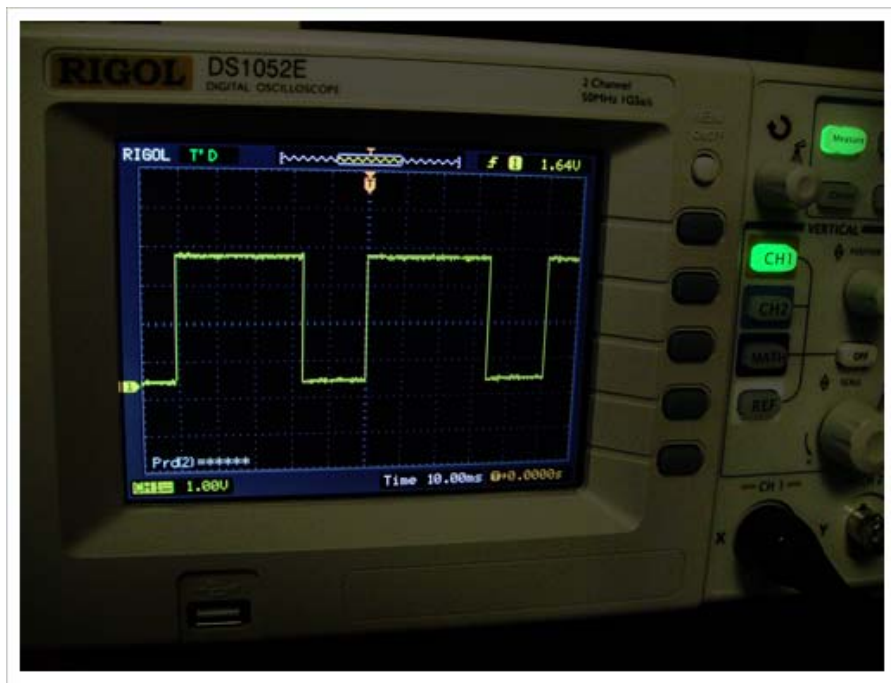
- | | |
|------------|--|
| Duty Cycle | Proportion of "on" time vs. the period. Expressed as a percent with 100% being fully on. This is used only with SetDutyCycle and the default clock rate. |
| Period | "Peak to Peak" time. This is in microseconds (1/1,000,000 second). Used in SetPulse. |

Duration Duration of the "on" time for a cycle. This is also in microseconds. This needs to be less than the Period. Used in SetPulse.

If you want to have the opposite effect; that is, a pulse that is on 2/3 of the time, the code would look like this:

```
1  public class Program
2  {
3      public static void Main()
4      {
5          PWM pwm = new PWM(Pins.GPIO_PIN_D5);
6
7          const uint period = 3 * 1000 * 1000;    // 3 ms
8          const uint duration = 2 * 1000 * 1000;  // 2 ms
9
10         pwm.SetPulse(period, duration);
11     }
12 }
13 }
```

The resulting waveform looks like this. Notice how it is now "on" for 2/3 of the time:



Uses for PWM

In its current form, you can use PWM on the Netduino to control servos (not directly, you'll want to use it to trigger something with more volts/amps) and control the brightness of LEDs. You can also use it as a clock signal for relatively slow clocks. When the new firmware comes out, I hope to use PWM to clock the MOS 8580 SID chip I have sitting in a breadboard on my desk.

About the Netduino

The Netduino is a 100% open source hardware and software platform, pin compatible with the Arduino. It runs the .NET Micro Framework; you write code using Visual Studio 2010. For more information, see the [official Netduino site](http://netduino.com).

(source code for this example is included below)

[Share](#) |