A Simulated Analog Display Using Two LEDs, Version 1.0

By R. G. Sparber

Conclusion

Two LEDs positioned close to each other can convey an almost continuously varying quantity by presenting relative intensity. This strategy is useful when precision is not required. The implementation is software-based, using three lines of code.

Background

Consider the gas gauge on my Honda Fit. It shows Full, has a tick mark at half, and indicates Empty. Funny thing, I've owned this car for eight years and never noticed this sparse look. It has always shown me what I needed to know.

The lesson here is that a high-resolution readout for my gas tank is not warranted.

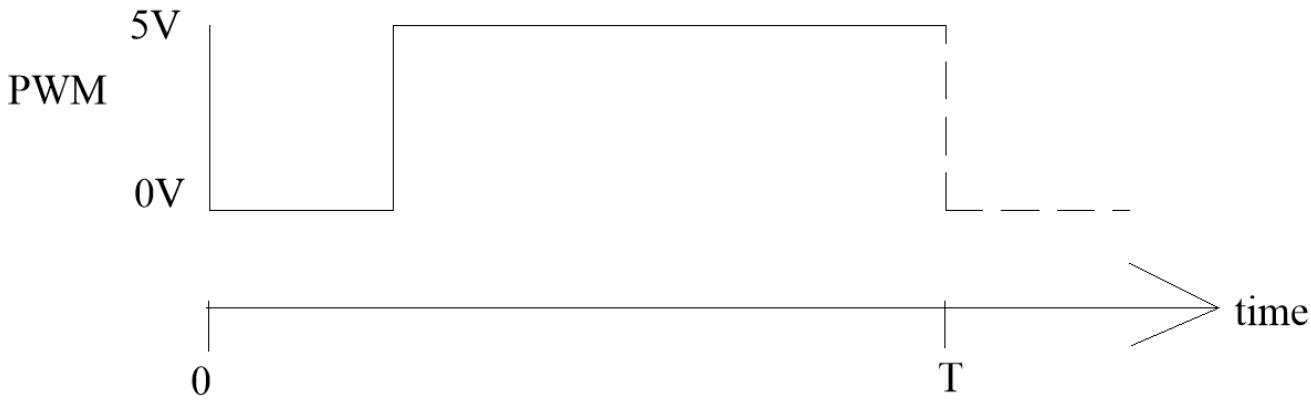
My electric bike has a homemade State Of Charge (SOC) gauge. Can I get two LEDs to tell me how much energy I have left in my battery?

Sure can.

I have come up with many ways to convey this information digitally, but all proved to be too distracting while I ride.

On my latest iteration, I decided to try an analog approach, just like the gas gauge on my car.

Theory



I can dim an LED[2] by driving it with a Pulse Width Modulated (PWM) signal. This signal only has two values, 5 volts and 0 volts, but it can change over time.

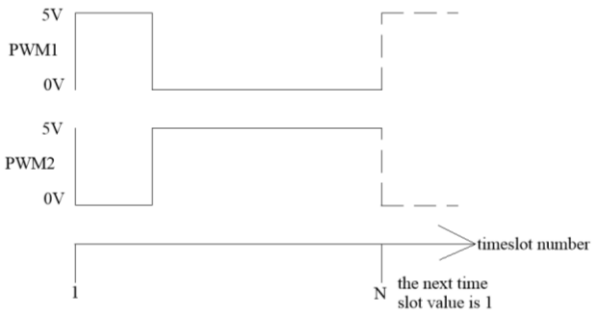
If this signal is always at 5 volts, the LED is at full brightness. If it spends half of its time at 5 volts and the other half at 0 volts, we see half brightness. Yup, the LED is dark when the PWM signal stays at 0 volts. The signal repeats after time T.

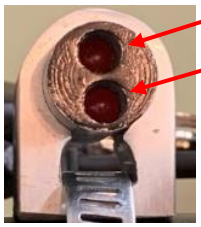
[2] LEDs work best when either full on or off. Trying to dim them by varying the voltage is both complicated and unstable. Dimming them with a PWM signal is the simplest and most predictable approach.

By varying the point where I change voltages, I get any brightness I want. This is because our eyes average the light.

The LED will flicker if it takes too long to repeat the pattern. If this repetition rate is fast enough, the exact value of T doesn't matter. Only the ratio of time spent at 5 volts versus 0 volts is visible on the LED. This ratio is called the duty cycle.

The problem with using a single LED to convey a value is that I have no visual reference. How do I know I'm at half-bright? My answer is to add a second LED driven by the complement of the PWM signal.

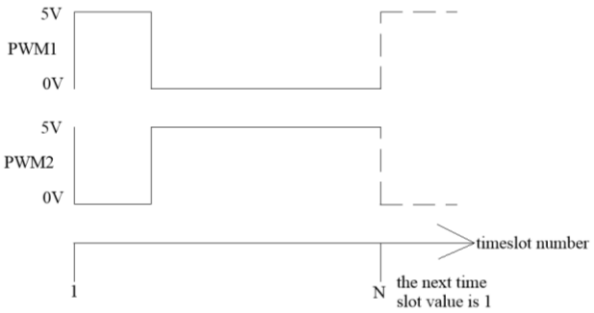




PWM1 drives LED1 and PWM2 drives LED2.

Rather than talk about repetition rate, I will focus on "timeslots." I start with timeslot 1 and count up to some number, call it ***N***. Then the signal repeats, so the next timeslot is 1.

When PWM1 is at 5 volts for timeslots 1 through N, LED1 is at full brightness, and LED2 is dark. When PWM1 is at 0 volts for timeslots 1 through N, LED1 is dark, and LED2 is at full brightness. Not very exciting, but it gets better.



When PWM1 is at 5 volts for timeslots 1 through $\frac{N}{2}$ and then at 0 volts for the remainder of the timeslots, LED1 is on half of the time so is at half brightness.

While LED1 is on, LED2, driven by PWM2, is off. Then, when LED1 turns off, LED2 turns on. The result is that both LEDs are on half of the time. More importantly, the LEDs are at ***equal*** brightness.

As I vary the time slot value where I switch the LEDs, I vary their relative intensity. Given N time slots, I vary this intensity in N steps. For example, if N equals 10, I can set the relative intensity in steps of 10% from 0 to 100%.

One way to think about this dancing of light is to see that the total brightness never changes. Instead, it shifts from top to bottom as the PWM signals change. When my eBike battery is fully charged, the top LED is on, and the bottom LED is off. At half, both are on. When my battery is dead, the top LED is off, and the bottom LED is on.

Now that you know what to expect, here is a video.

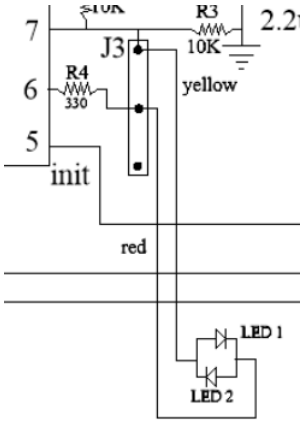
The Hardware

Many embedded systems-on-a-chip[3] contain PWM hardware. If you dig into their spec sheets, you will likely find that the PWM hardware can drive almost any General Purpose Input Output (GPIO) pin. It can also drive a second pin with the inverse of this signal. So, with no external circuitry, we can get PWM1 and PWM2.

If this is an Arduino compatible device, the command

```
writeAnalog(pin,dutyCycle)
```

will connect the PWM hardware to GPIO "`pin`" and set the duty cycle by specifying `dutyCycle` as a value between 0 and 255. However, configuring the device to output PWM2 is not a standard command.



In my application, the inverted signal could only come out on a pin that was already dedicated to another function. Additionally, my connections to the two LEDs were installed and not easy to change.

Time for a software solution!

[3] For example, the ATTiny85.

The Software

First, let me review what is important and what is not important.

1. As long as the PWM repetition rate is fast enough, I won't get flickering. The exact rate is not important.
2. I will get the desired behavior as long as the repetition rate does not change change "too fast." More on this later.
3. I do not want to change the hardware.

Arduino code is structured into four sections:

At the top, I define my variables and constants.

Next comes the function `setup()`, which, as the name implies, is where I do one-time actions needed to make the program run.

This is followed by `loop()`. After leaving `setup()`, `loop()` continuously executes. How long it takes to complete `loop()` depends on the code it contains. Call this the cycle time.

Functions used in `setup()` and `loop()` are defined in the last section.

For my PWM functionality, I define two variables that are each one byte so can hold any value between 0 and 255

- `PWMtimeslot` give it the initial value of 0
- `input` which is set by the user

I also have a constant that is one byte: `fullScale`

No related code is placed in `setup()`, and there are no functions needed.

In loop() I have[4]

```
if(PWMtimeslot >= fullScale)PWMtimeslot = 0
```

Each time loop() executes, this statement tests the current time slot value against the full-scale value. If `PWMtimeslot` is greater than or equal to `fullScale`, the time slot value `PWMtimeslot` is cleared back to zero. This lets me have any number of time slots in a period up to a maximum of 255. I found that `fullScale = 50` gave the smoothest display.

Next, I increment the time slot number by 1 using

```
PWMtimeslot++;
```

The result is that `PWMtimeslot` counts from 1 to `fullScale` and then back to 1 as I cycle through loop().

And, finally, I determine the PWM duty cycle

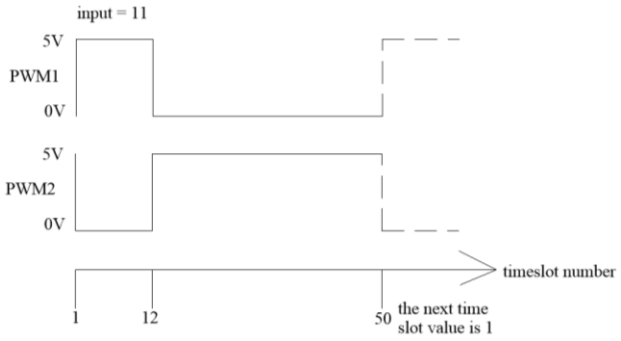
```
If(PWMtimeslot <= input){
      digitalWrite(PWM1,1);
      digitalWrite(PWM2,0);
}else{
      digitalWrite(PWM1,0);
      digitalWrite(PWM2,1);
}
```

where `input` must be equal to or less than the maximum number of timeslots, `fullScale`.

This is saying that when `PWMtimeslot` is less than or equal to `input`, PWM1 is set to 1 and PWM2 is set to 0. Otherwise, we do the opposite.

[4] I know there are more compact ways to code this up, but I don't think they would be as clear to a novice.

Let's try this logic out for `input` equal to 11 and `fullScale` equal to `50`.



Starting at `PWMtimeslot` equal to 1, we test to see if `timeslot` is less than or equal to our `input`, 11. It is, so we set `PWM1` to 1 while `PWM2` is set to 0. We will get this result until `timeslot` equal 12. Then the test is false, so we set `PWM1` to 0 and `PWM2` to 1. We stay in this state until `timeslot reaches 51.` Then this count is set back to 0 and promptly incremented to 1. Rinse and repeat.

If you looked at the two LEDs, the top one would be about ¼ as bright as the bottom LED.

There are a few consequences of this design.

- The software increments the timeslot as fast as possible. Any logic that defines the duration of a timeslot must run slower and will be more complex.
- I can duplicate this code as many times as needed until I run out of output pins. Each PWM generator can have a different number of timeslots.
- Unlike the stock software-controlled PWM hardware, I can have both the true and inverted outputs on any pins.

The one problem with this scheme is that if your application requires a constant repetition rate, this software-based PWM generator will only work if the cycle time of `loop()` is constant. As you change the number of lines of code being executed, the cycle time changes.

In my application, a lot more code runs once every second, yet I don't see any difference in the LEDs.

I welcome your comments and questions.

If you want me to contact you each time I publish an article, email me with “Subscribe” in the subject line. In the body of the email, please tell me if you are interested in metalworking, software plus electronics, kayaking, and/or the Lectric XP eBike so I can put you on the right distribution list.

If you are on a list and have had enough, email me “Unsubscribe” in the subject line. No hard feelings.

Rick Sparber
Rgsparber.ha@gmail.com
Rick.Sparber.org