Episode 1.4 – Pulse Width Modulation

Welcome to the Geek Author series on Computer Organization and Design Fundamentals. I'm David Tarnoff, and in this series we are working our way through the topics of Computer Organization, Computer Architecture, Digital Design, and Embedded System Design. If you're interested in the inner workings of a computer, then you're in the right place. The only background you'll need for this series is an understanding of integer math, and if possible, a little experience with a programming language such as Java.

In this episode, we will show how a binary signal can be used to give the appearance of an analog output. We will then use this understanding to create a digital dimmer switch on the Arduino open source platform.

Have you ever seen an LED dim? Well, actually, you haven't. LEDs are binary devices for the most part – they're either on or off. I can hear the protestations now. "Wait a minute, Tarnoff. I have definitely seen an LED dim." Well…no.

At the end of our last episode, we discussed the two measurements used to fully describe a periodic pulse train, the period and the positive going pulse width. There is a third way to describe them, although by itself, it won't fully describe the signal. The ***duty cycle*** represents the percentage of time that a periodic pulse train is a logic one. For example, if the pulses of a periodic pulse train are coming once a second, then its frequency is 1 Hz. If each pulse has a pulse width, t_w, of one quarter of a second, then it is a logic one for 25% of the duration of the full period, T. This makes the duty cycle of the signal 25%.

The formula used to calculate the duty cycle is simply the ratio of the pulse width to the period. Since the pulse width can take on values from from 0 to T, then the duty cycle has a range from 0%, which is a constant logic zero, to 100%, which is a constant logic one. Since both t_w and T have units of seconds, the units cancel out and the resulting fraction is a unitless value between 0 and 1. Multiply this fraction by 100% and you get the duty cycle as a percentage.

Let's consider an example. When looking at a fixed light source, the viewer cannot detect the difference between a persistent light and one that is blinking at a high frequency. The blinking frequency at which the light appears to turn solid varies between individuals, but it is usually between 50 and 90 Hz. This critical rate, referred to as the ***flicker fusion threshold*** or ***critical flicker fusion rate***, can also change based on the movement of the light source or its location within the viewer's field of vision.[1]

That brings us to the not so dimmable LED. By sending a periodic pulse train to an LED using a frequency above the flicker fusion threshold, the LED will appear steady yet dimmer than it would when the same LED is sent a constant logic one. The brightness of the LED with respect to the full on state is equivalent to the duty cycle. For example, to make an LED shine half as bright as it would have with an input of a constant logic one, the duty cycle should be 50%. The frequency is irrelevant as long as it is higher than the human eye can detect.

Let's try this with some real numbers. Assume that a periodic pulse train with a frequency of 1 kHz, or 1,000 cycles/second, is sent to an LED. What should the pulse width (t_w) be to make the light emitted from the LED appear to be three-quarters of its full brightness?

First, we need to determine the period. From Episode 1.3, we know that the period in seconds/cycle is the inverse of the frequency, which is measured in cycles/second. By taking the inverse of 1,000 Hz, we see that a full period lasts one-thousandth of a second.

Remember that the duty cycle is the percentage of time that the signal is high over the full duration of one period. Since we want the positive going pulse to be three-quarters of the period, then the pulse width must be three-quarters of one-thousandth of a second. This equals 0.00075 seconds or 750 microseconds.

This strategy for using a digital signal to give the appearance of an analog signal is called ***Pulse-Width Modulation*** or ***PWM***. It turns out that PWM isn't just sorcery only available to engineers and scientists. With a tiny investment and a little software, anyone can create PWM signals.

Have you ever heard of ***Arduino***? If you haven't, it's a line of low-cost circuit boards that allows students, hobbyists, engineers, and, well, just about anyone to create programmable devices to interact with users and their environment.[2] The Internet contains countless tutorials on the use of Arduino boards, so this introduction will be but a taste of what can be done.

The typical Arduino board has a set of conductive pins along the edge of the board that serve different functions. A number of these pins can be used as digital outputs. Circuitry on the board allows programmers to send a PWM signal to any of these digital outputs using a single command. The command is ***analogWrite()***. This command is called a function, and like many functions, it needs some information in order to work. This information is usually a set of numbers referred to as parameters. The first parameter is a whole number representing the number of the pin to which we want to send the PWM signal. The second parameter is the one we are interested in. It is an integer from 0 to 255. Later in this series, we will explain the significance of the number 255. For now, it is enough to say that this second parameter represents the duty cycle of the PWM signal. Passing a 0 for this parameter will set the duty cycle of the PWM signal to 0% while passing 255 will set the duty cycle to 100%. Basically, the programmer needs only to multiply the desired duty cycle by 255 and round it to the closest integer in order to appropriately set the pulse width of the PWM output.

Note that of the parameters passed to the analogWrite() function, none of them identify a frequency. Remember that as long as the frequency of the periodic pulse train is above the flicker fusion threshold, the LED will appear solid to the viewer. That said, the default frequencies of these digital output pins on the Arduino range from 500 Hz to 1,000 Hz, which is above the flicker fusion threshold for most everyone.

Oh, and one more thing. Have you ever seen those strings of LEDs that transition across an entire spectrum of color? They're using pulse-width modulation! Each of those colored LEDs is actually three LEDs: one red, one green, and one blue. They are mounted close together so that they appear as a single light source. A different PWM signals is sent to each of the three LEDs so as to independently vary the intensity of red, green, and blue, thus creating a full spectrum of color.

In the next episode, we will be going back to the basics – counting, or more importantly, how the computer counts. Until then remember that while the scope of what makes a computer is immense, it’s all just ones and zeros.

References:

1 – Davis, J., Hsieh, Y.-H., & Lee, H.-C. (2015, February 03). _Humans perceive flicker artifacts at 500 Hz_. *Scientific Reports*. Retrieved February 05, 2019, from https://www.nature.com/articles/srep07861

2 – Arduino. (n.d.). _Getting Started | Foundation > Introduction_. Retrieved February 05, 2019, from Arduino: https://www.arduino.cc/en/guide/introduction