Arduino Function Generator

April 22, 2019

1 Introduction

In this lab, you will construct a function generator using the Arduino microprocessor. The Arduino Uno provides analog output using pulse width modulation (PWM): the output is always at the digital logic level (0 or VCC) but the duty cycle varies according to the specified one-byte output level (from 0 to 255). For an analog write output level of 0, the duty cycle is 0% (always off). For 255, the duty cycle is 100% (always on). For 127, the duty cycle is 50% (half on, half off).

The software you develop for your Arduino microprocessor will provide PWM output for the intended function, and a low-pass RC filter will convert the PWM output to a constant or varying analog output value.

2 Introduction to PWM

- **Step 1:** Upload the "Examples/01.Basics/Fade" Arduino sketch.

  On the proto-shield, connect the LED 1 to Arduino output pin 5 and change the input pin used in the sketch to 5 as well. In the Fade example as provided by the Arduino Integrated Development Environment (IDE), the output is sent to pin 9, but later we will be using high speed PWM output which is only available on pins 5 and 6. So it is best to simply change this now at the beginning.

  You should see the LED gradually become brighter and then dimmer.

- **Step 2:** To slow things down, adjust the delay (during fading) from 30 milliseconds to 100 milliseconds.

  Connect the oscilloscope to the output of pin 5. Describe what you see. What is the frequency of the PWM output on Pin 5 (i.e. one over the time interval between rising edges of the PWM output.)

- **Step 3:** The PWM frequency will ultimately limit the frequency of the output functions we can generate. To speed up the PWM frequency we will use low-level commands to adjust the divisor used for the clock which drives the PWM output on these pins. Place the following command in the setup function for your sketch:

```
// set 3 LSB bits of TCCR0B to 1, setting clock divisor to 1,
// for fastest PWM mode on pins 5,6;
TCCR0B = (TCCR0B & 0xb1111000) | 0b001;
```

You should see the PWM frequency increase substantially. This has the unintended side effect of speeding up the "micros" function as well. We can't have that! So when speeding up this clock, you'll want to use a corrected version of the micros function for measuring time intervals, like this one:

```
// fix micros to give correct answer,
// even after we mess with the clock speed.
unsigned long corrected_micros(){
  unsigned long divisor_mode = TCCR0B & 0x7;
  unsigned long fudge = 1;
  if (divisor_mode == 1){ fudge = 64; }
  return micros()/fudge;
}
```

3 Function Generator

Your task is to design a function generator with three different simultaneous outputs: square wave, sine wave, and saw tooth function.

Every function should have a software specified frequency which is independent of the other functions. The square wave function may have a fixed amplitude, but the sine wave and square wave should have software specified (independent) amplitudes as well.

It will be hard to make this work reliably using busy wait such as provide by the `delay` function as in the Fade example. This is a busy wait because the processor does nothing else (but wait) during this time. If all you are doing is one thing (as in the Fade example) this technique is easy and effective. But you want to update three different functions at three different frequencies, the busy wait version becomes problematic. Instead, you should use something like:

```
void loop(){
   unsigned long time = corrected_micros();
   unsigned period1 = 1000; // period of function 1 in microseconds
   float phase1 = (time \% period1) / ((float) period1);
   //...
}
```

The microprocessor enters the loop function at an unpredictable time. The busy wait implicitly assumes this unpredictability is small compared to the wait time, it will have noticable timing jitter if you try to run at the fastest possible rate, another defect of this approach! In this approach, the current time (corrected for the fact that we have sped up of the clock) is reported by the hardware each time you enter the loop. You should calculate the appropriate output level for each function and update the PWM output accordingly. Conceptually, you are simply being told the current time t, and setting the output to $f(t)$.

You will also need to design an appropriate RF filter to smooth the PWM output into a continuous function. Try to keep the source impedance of your function generator to something like 100 Ω.

What is the highest frequency of each waveform you can generate without major distortion?

4 Common Pitfalls

Common mistakes and symptoms during this lab include:

- Mixing types (unsigned long and unsigned) inappropriately. This can mess up the timing logic based on micros(), which returns unsigned long.
- Leaving the LED in the filter circuit (a good idea at beginning, to provide visual feedback) and then wondering later why the sine function is distorted.

5 Lab Write-Up

The lab report from this section should include a short description of your design (both hardware and software), oscilloscope shots of your wave form output, discussion of its limitations, and the complete source listing for your sketch.