ECE 1001/1002 Introduction to Robotics
Lab #5: Analog Outputs and PWM

Objectives

Introduction to the analog control, while loops

You have made your Arduino control things digitally, that is, you can turn things on or off. Because it is an electronic computer, Arduino can turn things on/off very quickly. If fact, it can switch digital signals so quickly that we experience it as a continuous (analog) change in signals.

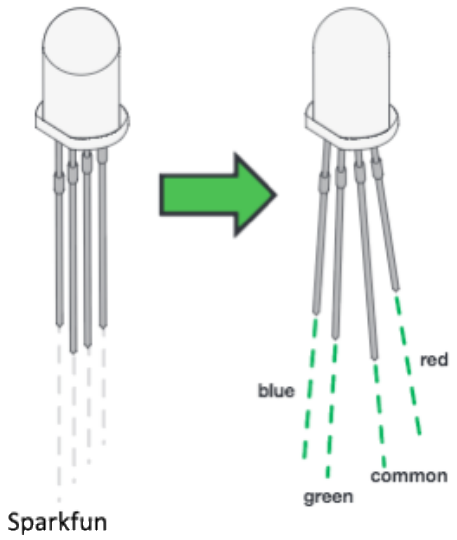
The music output is kind of that situation—the Arduino doesn't make a sinusoidal signal—instead it makes a square wave (the output is a boxy on/off) but it sounds similar to a continuous, sinusoidal wave to us. Some of that is because our senses are analog, and have a kind of filtering capacity to smooth things out when they occur more quickly than our senses respond (most nerve responses are limited to about 1/8 second…125ms). Some of that is due to other equipment limitations, the speaker for instance, tends to smooth out square wave signals.

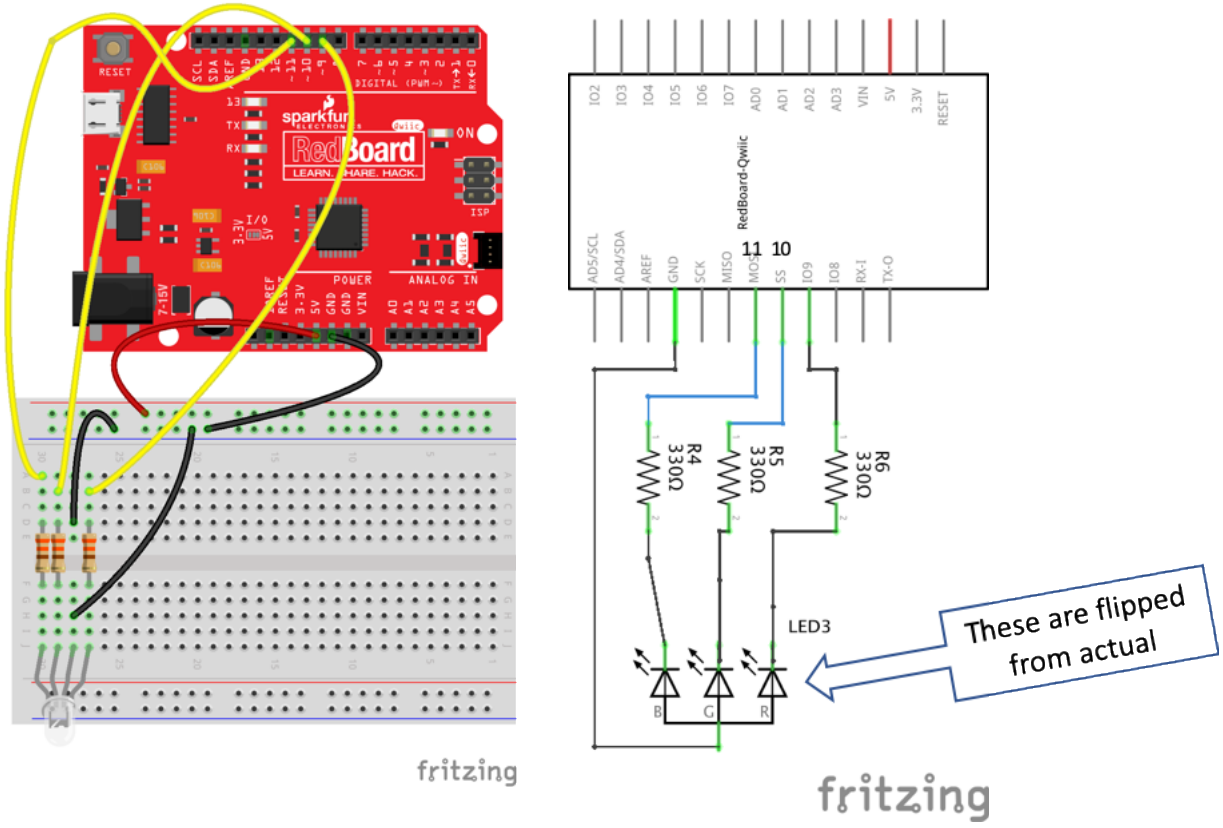
Electronic controls often make use of the tendency of digital signals to smooth out into more analog-type signals.

Requirements and Signoffs

RGB LED circuit

Build the circuit below. It uses the RGB LED which can display red, green, blue, or a mixture of the colors; (it's actually just three small LEDs in a single package). The package looks like this. Note the arrangement of the leads, you must connect them in the right order. "common" means "ground" or GND.

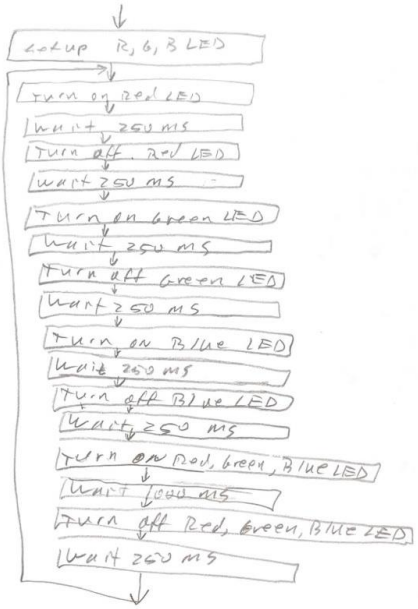




The LED color control pins are connected to 330Ω resistors which are then connected to Arduino pins 9, 10, 11.

Digital Light Control by Software

Open the program "Lab5_part_a" and look at it. The three color LEDs are controlled separately, each by their own Arduino pin (9, 10 or 11). It looks like lab 1, just more LEDs.
(for the next few programs no flowchart is shown, they are similar)



The colors should blink on and off in order: red, green, blue, then all three stay on for 1 second. Notice how when all three colors are on, you see a blend of the colors, maybe a bluish-green color, but it's a *blend*. Because of the mechanical structure of the LED (the frosted cover) and the blending our eye does, it looks smoothly blended and brighter than any individual LED.

Open the program "Lab5_part_b" and look at it. Now all three LEDs are turned on for a 14ms delay, then the LEDs are all turned off for 1ms before the main loop repeats. If you think about it, does this means the LEDs are on for 93% of the time, and off for 7% of the time?

Question 1) Describe on your sign-off paper what you see.

Question 2) How does this compare to the last part of program a where all three LEDs were turned on? Does it look about the same?

Open the program "Lab5_part_c" and look at it. Here the three LEDs are turned on for 2ms, then off for 13ms. The total time the program takes to repeat is the sum of the delays:

$$time_{on} + time_{off} = time_{total} = 2ms + 13\ ms = 15\ ms$$

This is called a 13% duty cycle, since the LEDs are on for 13% of the total time:

$$\text{part c } \%\ duty\ cycle = \frac{time_{on}}{time_{total}} = \frac{2ms}{15ms} = 13\%$$

Question 3) How does the light look compared to the part b program? For comparison, Part b was a 93% duty cycle:

$$\text{part b } \%\ duty\ cycle = \frac{time_{on}}{time_{total}} = \frac{14ms}{15ms} = 93\%$$

Open the program "Lab5_part_d" and look at it. Here the duty cycle is 47%%: 7ms on, 8 ms off.

Question 4) What difference do you see? Can you detect any flicker in programs a, b, c, or d?

Probably not for several reasons. Mostly, your eye can only see longer light flashes; these short ones tends to blur together. It is similar to the way that the individual colors blend when all on.

As the duty cycle is decreased, the light seems to get dimmer and dimmer, and as long as the total cycle time is less than about 20 ms there seems to be not much flicker[1]. It's a smooth change in brightness.

This is actually a common control method known as "pulse-width-modulation" or PWM. It's so common that the Arduino has a sophisticated PWM capability built-in on pins 3, 5, 6, 9, 10, 11 (the ones with a ~).

[1] Actually, the cones which see color head-on through your eyes are a bit slower than the rods which see black/white everywhere. For my eyes, the flicker is gone looking directly at the LED at about 24 ms total time, but looking through the side of my eye (rods for peripheral vision) the images flickers down to about 18 ms total time. It's quick, easy and interesting to try for yourself!

While we did implement PWM just by quickly turning digital signals on and off in programs b, c, d, the built-in PWM control in the Arduino is better because it lets us control signals more precisely and with easier programming. The PWM on Arduino has 256 increments of control, and the on/off cycle is 490 Hz (2ms) or faster.

Open the program "Lab5_part_d" and look at it. This part implements analog instead of digital output on the LED pins. The control part looks like this:

analogWrite(9, 64);

For using PWM, the "syntax" is *analogWrite(pin, value)* where 0 < value < 255. In part e, the value is set at 64, which means the duty cycle will be 64/255=25% (when the LED is on of course). The LED will be at about 25% power; now it blinks on for 1 second, off for 0.2 seconds.

Question 5) Try changing the PWM value to other (integer) numbers between 0 and 255, and describe what happens. 0 is off, and 255 is all the way on.

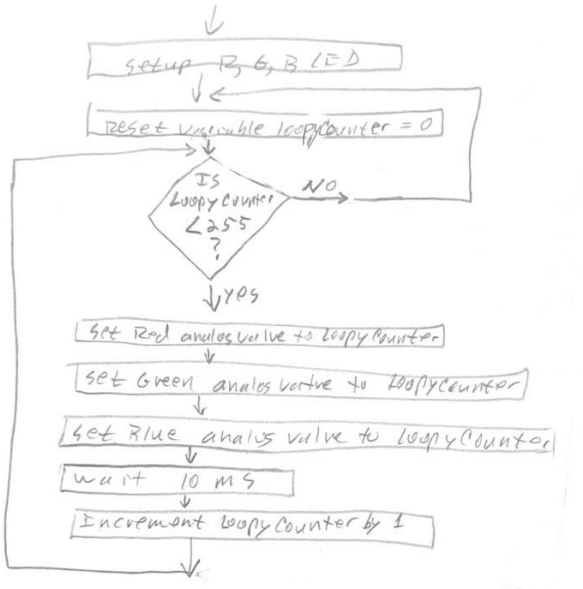
This implements a light dimmer, if you have an LED dimmer at home, this is most likely exactly how it works. Probably more or less the same thing for changing speed on many (small) motors and other (modern) power devices you use.

Fancy computer light control

Now that we have PWM control in place, it becomes easier to do all kinds of fancy crazy things controlling the LEDs (or whatever!).

First there needs to be additional programming complexity, which enable more complicated output control.

Open the program "Lab5_part_f" and look at it. Two new programming concepts are introduced here, variables and while() loops.



First the while() loop. A while loop will do whatever is inside the curly braces {} as long as the condition between the parenthesis () true. Like this:

```
While (condition is true)
{
Do things
}
```

When the Arduino comes to a while statement, it will check the (condition). If the condition evaluates as TRUE, then the Arduino will do the things between the { } braces. When the Arduino reaches the end } brace, it goes back up to the while statement, evaluates the (condition) and, if TRUE, repeats what is between the braces. If the Arduino evaluates the (condition) and finds it is FALSE, it skips around the {} braces and does whatever is next after the } brace.

The other new programming concept involves variables, which are named memory storage on the Arduino. To use a variable, first you name the bit of memory you want to use, and tell the Arduino something about it; this "declares" the variable. Here we have declared a variable of the type "int" (an integer which can be numbers like 5, -5, 0, 43, -79 etc). It's been named "loopyCounter." Any set of upper/lower case characters without a space will work as a name. But you need to be careful and not misspell it, the Arduino IDE rejects spelling errors but it doesn't always tell you what or where you made the spelling mistake. The variable declaration step occurs (in this program) right at the top:

```
int loopyCounter;
```

The semicolon after the declaration is required. Now the Arduino knows to store some integer number in a memory location called loopyCounter.

The main loop is this:

```
void loop()
{
 loopyCounter = 0;

  while (loopyCounter < 256)
  {
    analogWrite(9, loopyCounter);
    analogWrite(10, loopyCounter);
    analogWrite(11, loopyCounter);
    delay(10);
    loopyCounter = loopyCounter + 1;
  }

}
```

The first thing that happens in the main loop is a value is assigned to the variable loopyCounter, a value of 0 is stored in the memory. You can say the variable is initialized. The single equals sign is not a test (a condition or test of equivalence is two equals ==) A single equal sign means take whatever is on the right and make the thing on the left be the same. So here, 0 is stored in loopyCounter with the single equal sign.

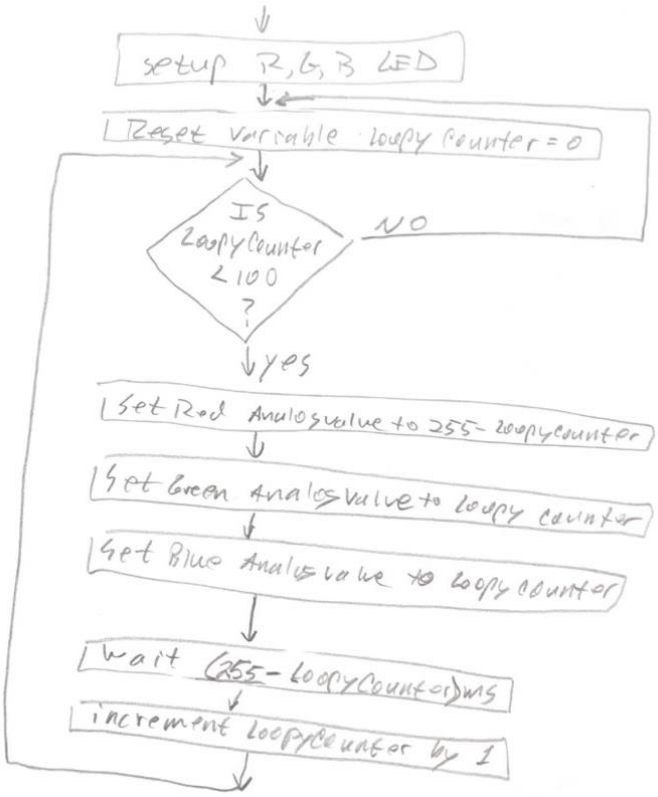
Then there is the while () loop. The Arduino checks the condition in the parenthesis, if it's true, it does what's in the curly braces. Here the Arduino will write each of the LEDs with whatever value loopyCounter is set to, then it will delay 10 ms. Then the program adds 1 to the value of loopyCounter and stores that back in loopyCounter. It's said that loopyCounter was incremented by one. After the first time through the while

loop, when the conditional test is made, loopyCounter has a value of 1, which is less than 256, so the loop will execute again.

This means the PWM value set by loopyCounter will gradually increase from 0 (off) to 255 (maximum brightness). It will take 255*10ms=2.5 seconds go around the loop, gradually (and smoothly!) increasing the LED brightness.

Question 6) Play with the values, and try a few things. Describe what happens.

Open the program "Lab5_part_g" and look at it. The program is the same, but now there is some math in the analogWrite and delay statements. Red gets smaller as loopyCounter increases, while blue and green increase. The delay gets shorter as loopyCounter increases.



Show your instructor the result with part g, and hand in your signoff paper.

To receive credit for this lab, you *must* get the sign-off before the due date (see Canvas), during class time or with a TA during their hours. The instructor will not signoff outside of class hours. Partial credit is allowed.