Controlling The Real World With Computers

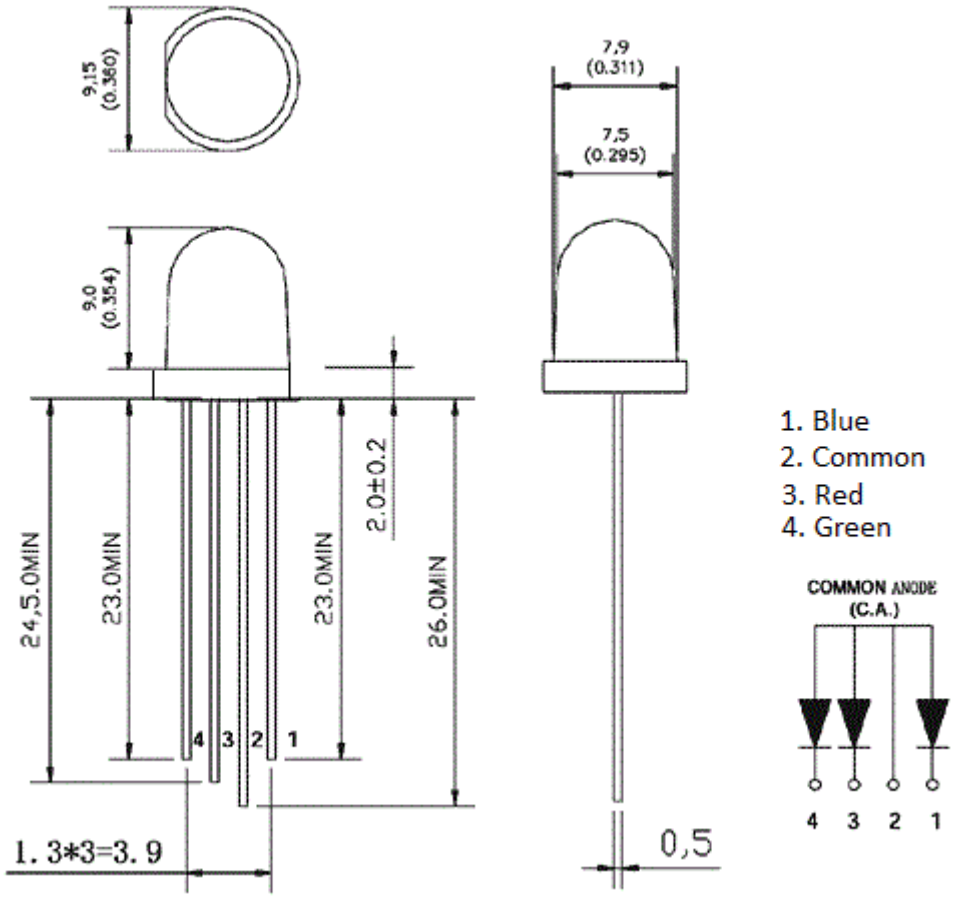
:::. Control And Embedded Systems .::

Table Of Contents

Previous: Programming Part 3 Next: Programming Experiment 2

Putting It All Together - Controlling The Hardware With The Software
Programming Experiment 1

We will next take a look at a way to make an interesting display using a RGB (red, green, blue) LED which is actually three LEDs in a single package. It contains a red, green and blue LED. A huge number of colors can be obtained from it by varying the brightness of each of the primary colors it contains. The one used in this experiment looks like this:



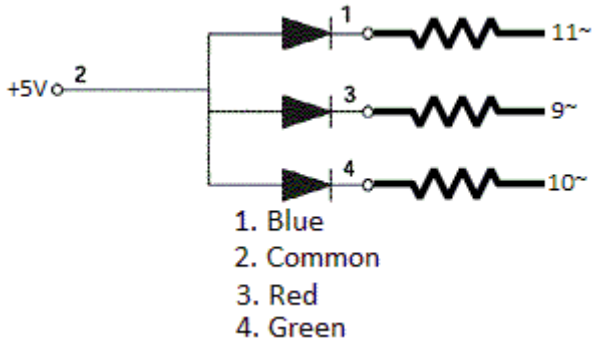
Notice that the anodes of the three LEDs are common in the package,which. should be connected to +5 volts on the UNO.

2

Connect # 3 on the LED through a 150 ohm resistor to 9~ on the UNO,
4 through a 150 ohm resistor to 10~ on the UNO and # 1 through a 150 ohm resistor to 11~ on the UNO Now load BrightTest here, compile and run it. It looks like this:

```
/*
BrightTest
Turns one color on 1 second, then next, etc.
*/

int Red_Pin_9 = 9;
int Green_Pin_10 = 10;
int Blue_Pin_11 = 11;

void setup()
{
	pinMode(Red_Pin_9, OUTPUT);
	pinMode(Green_Pin_10, OUTPUT);
	pinMode(Blue_Pin_11, OUTPUT);
}

int RedOn = 0, GreenOn = 0, BlueOn = 0;
int RedOff = 255, GreenOff = 255, BlueOff = 255;

void loop()
{
	SetLED(RedOn, GreenOff, BlueOff, 1000); // red on, green off, blue off 1 second
	SetLED(RedOff, GreenOn, BlueOff, 1000); // red off, green on, blue off 1 second
	SetLED(RedOff, GreenOff, BlueOn, 1000); // red off, green off, blue on 1 second
	SetLED(RedOn, GreenOn, BlueOn, 3000); // all on 3 seconds

} // end void loop()

void SetLED(int RedValue, int GreenValue, int BlueValue, int DelayTime)
{
	analogWrite(Red_Pin_9, RedValue);
	analogWrite(Green_Pin_10, GreenValue);
	analogWrite(Blue_Pin_11, BlueValue);
	delay(DelayTime);
} // end void SetLED(..)
```

This test turns each LED on for 1 second, then all three on for 3 seconds. Pay careful attention to the brightness of each color. Note that, whereas in the blink program we made the pin high to turn on an LED, here the pin is taken low to turn on an LED section. Since these are PWM analog pins, 0 is full low and 255 is full high. What's happening here is that the cathodes of each of the LEDs is being pulled low by the UNO to turn on a particular section. That completes the circuit for the LED:



Pulling the current limiting resistor of a color low completes the circuit, turning that color on. You might note that red is not quite as bright as green and blue. Increase its brightness by decreasing its resistor to 100 ohms. Now load ThreeSweep here. Pin number variables are declared above setup, then setup turns off all three LEDs by taking the outputs high, by outputting 0xff, which is 255, to the pins:

```
int Red_PWM_Pin_9 = 9;
int Green_PWM_Pin_10 = 10;
int Blue_PWM_Pin_11 = 11;

void setup()
{
        pinMode(Red_PWM_Pin_9, OUTPUT);
        pinMode(Green_PWM_Pin_10, OUTPUT);
        pinMode(Blue_PWM_Pin_11, OUTPUT);
        analogWrite(Red_PWM_Pin_9, 0xff);
        analogWrite(Green_PWM_Pin_10, 0xff);
        analogWrite(Blue_PWM_Pin_11, 0xff);
}
```

The brightness variables are then set to different values so that the colors will move up and down differently. This is also accomplished by making the step sizes and signs different for each color. One color will move up in intensity at one rate while another moves down in intensity at another rate. All three maximums are set at 255. Any of these, along with the delay used each time the steps are taken, can be changed. The best way to see the effects is to experiment with the numbers:

```
// these could be set to different values
int RedSweepValue = 0; // start red full off
int GreenSweepValue = 200; // start green almost full on
int BlueSweepValue = 127; // start blue half on

// these could also start at different values
int RedStep = 3;
int GreenStep = -1;
int BlueStep = 1;
int MaxRedSweep = 255;
int MaxGreenSweep = 255;
int MaxBlueSweep = 255;
int DelayTime = 10;
```

The first that happens in loop() is to output the values to their respective analog ports:

```
void loop()
{
        // we are sinking rather than sourcing
        // so XOR with 11111111 to invert
        if(MaxRedSweep)
                analogWrite(Red_PWM_Pin_9, 0xff ^ RedSweepValue);
        if(MaxGreenSweep)
                analogWrite(Green_PWM_Pin_10, 0xff ^ GreenSweepValue);
        if(MaxBlueSweep)
                analogWrite(Blue_PWM_Pin_11, 0xff ^ BlueSweepValue);
}
```

Notice two things. First, a value is not sent out unless there is a maximum sweep value for the color. The term if (MaxRedSweep) means to do the following (analogWrite) if MaxRedSweep is not 0. If you make a maximum low, there will be no output for that color. Try making one of them 0 and see what happens. Making one less than 255 will still cause it to work, but not get as bright. It will also change faster, since it doesn't have as far to go. Also, the largest value analogWrite will accept is 255, so maximums cannot be greater than that. The other thing to take note of is the XOR with 0xff, which inverts the bits and causes the duty cycle to be related to the low rather than high pulse. Taking something low is called sinking in electronics. Remember that XOR yields an on state if the inputs are different. Recall the following from the Boolean section:

0xD4 ^ 0x8D = 0x59 (this says 0xD4 XOR 0x8D = 0x59)
0xD4 11010100
0x8D 10001101
0x59 01011001 (Think, "This bit will be on only if the two above it are not the same.")
0xDA ^ 0x87 = 0x5D (this says 0xDA XOR 0x87 = 0x5D)
0xDA 11011010
0x87 10000111
0x5D 01011101
0xAB ^ 0xFC = 0x57 (this says 0xAB XOR 0xFC = 0x57)
0xAB 10101011
0xFC 11111100
0x57 01010111

If one of the values is 0xff, all of the bits of the other value is inverted. For example, green starts at 200, which is 11001000 or 0xc8. So if we XOR it with 0xff, we get an inversion of the bits:

0xC8 ^ 0xFF = 0x64 (this says 0xC8 XOR 0xFF = 0x64)
0xC8 1010 1011
0xFF 1111 1111
0x64 0101 0100

The bottom value is the top value with all of the bits inverted, and the pulses are said to sink rather than source. One purpose of this exercise is to help students get used to pulling a device low to turn it on, which is very common in embedded and control systems.

Next, the values are incremented or decremented by the step values, then checked for being too high or too low:

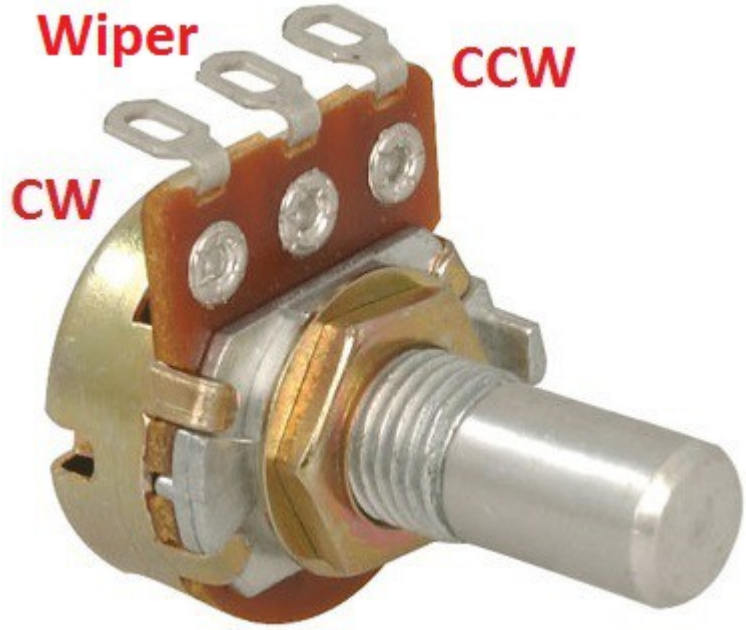
```
RedSweepValue += RedStep; // step up when positive, down when negative
GreenSweepValue += GreenStep;
BlueSweepValue += BlueStep;

// red
if(MaxRedSweep)
{
	// last value used was 255 - incremented from 255 to 256
	if (MaxRedSweep < RedSweepValue)
	{
		RedSweepValue = MaxRedSweep - 1; // last used for PWM was 255 so
start back down
		RedStep *= -1; // and set step to -1 so we will go back down
	}

	else if (0 > RedSweepValue) // last value used was 0 - decremented from 0 to -1
	{
		RedSweepValue = 1; // last used for PWM was 0 so start back up
		RedStep *= -1; // and set step to 1 so we will go back up
	}
}
```

Testing for too high or too low is also done only if there is a non-zero maximum value.

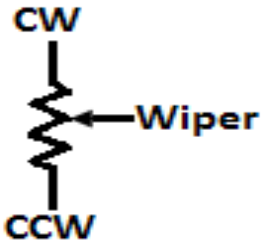
At the end of all of the tests, there is a delay by the number of milliseconds indicated by the DelayTime variable. Next, we will make a color changer. Load ColorChanger here and you will see the following:

```
/*
ColorChanger
Read analog values from potentiometers and output to color channels
*/

int Red_PWM_Pin_9 = 9;
int Green_PWM_Pin_10 = 10;
int Blue_PWM_Pin_11 = 11;

void setup()
{
	pinMode(Red_PWM_Pin_9, OUTPUT);
	pinMode(Green_PWM_Pin_10, OUTPUT);
	pinMode(Blue_PWM_Pin_11, OUTPUT);
}

int RedAnalog_Pin_A0 = A0; // read red analog from AD converter A0
int GreenAnalog_Pin_A1 = A1; // read green analog from AD converter A1
int BlueAnalog_Pin_A2 = A2; // read blue analog from AD converter A2

int RedAnalogValue = 0; // holds analog from AD A0
int GreenAnalogValue = 0; // holds analog from AD A1
int BlueAnalogValue = 0; // holds analog from AD A2

void loop()
{
	// Read the analog value and map to the analog variable
	// The analogRead() output will range from 0 to 1023
	// Map that to 0 to 255 for PWM using the built-in map function
	RedAnalogValue = map(analogRead(RedAnalog_Pin_A0), 0, 1023, 0, 255);
	GreenAnalogValue = map(analogRead(GreenAnalog_Pin_A1), 0, 1023, 0, 255);
	BlueAnalogValue = map(analogRead(BlueAnalog_Pin_A2), 0, 1023, 0, 255);

	// XOR with 11111111 to invert and write for PWM output
	analogWrite(Red_PWM_Pin_9, RedAnalogValue ^ 0xff);
	analogWrite(Green_PWM_Pin_10, GreenAnalogValue ^ 0xff);
	analogWrite(Blue_PWM_Pin_11, BlueAnalogValue ^ 0xff);

} // end void loop()

// end ColorChanger
```

This is basically a simplified version of ThreeSweep. It accepts analog input for the three colors using analogRead(..) to read analog inputs A0, A1 and A2. This reads analog values from some potentiometers. The result will be a number from 0 to 1023, representing 0 to 5 volts at the analog input. The program uses the map function to map the 0 to 1023 value to the 0 to 255 values the PWM analog channels can use. The resulting value is put into the PWM output with an XOR of 0xff to invert the pulses. A potentiometer looks like this, where CW is clockwise, CCW is counterclockwise and Wiper is, well, the wiper:



And the schematic representation looks like this:



9

Use a combination of alligator clip leads and breadboard jumper wires to connect all clockwise terminals to +5 volts and all counterclockwise terminals to ground. Connect the wipers to A0 for the red control, A1 for green and A2 for blue. Compile ColorChanger. You should be able to change the intensity of each color by turning the shafts. A color should be off at with the control fully counterclockwise, and full on with it fully clockwise.

A challenge for next time:

Use the potentiometers to modify the step sizes for each color in ThreeSweep. Remember to keep the polarities and map to something like 1 to 10. Experiment (play).

Table Of Contents

Previous: Programming Part 3 Next: Programming Experiment 2