HANDS-ON LAB INSTRUCTION SHEET – Learning Kit MODULE 6

RESISTIVE SENSOR CONTROL OF RGB LEDS

NOTES:
If you did not finish Modules 1 through 5, be sure to finish them NOW *before starting this Module or you will fall behind the rest of the class.* ***Labs MUST be done in order.***

BILL OF MATERIALS

(1) **Arduino UNO R3 Microcontroller & USB Cable**
(3) **RED** Light-Emitting Diodes (LED) – *Or one each Red, Yellow, and Green LEDs*
(1) **RGB** Red-Green-Blue Light-Emitting Diode (LED) – *Common Cathode*
(1) **Photoresistor** - *sometimes called a* **Photocell** *or* **LDR (Light Detecting Resistor)**
(1) **1KΩ (1000 Ohm)** Resistor
(3) **220Ω** Resistors (*actually any value **220Ω** through **1KΩ** would work*)
(1) **10KΩ (10,000 Ohm)** Potentiometer – if available

BACKGROUND REVIEW AND THEORY

In Lab 5 we have seen that either resistor voltage divider networks or a potentiometer can provide a variable, analog, input that the Arduino can then use to provide different levels of control rather than that of a simple switch.

There are many sensors that provide a variable resistance change. These include the simple potentiometer, the Joystick Controller which is actually an X-axis and a Y-axis potentiometer pair, a thermistor (changes resistance according to temperature), a resistive strain gage (changes resistance when compressed according to the load on it), and a Cadmium-Sulfide (CdS) photocell or LDR (light-detecting resistor that changes resistance based on light).

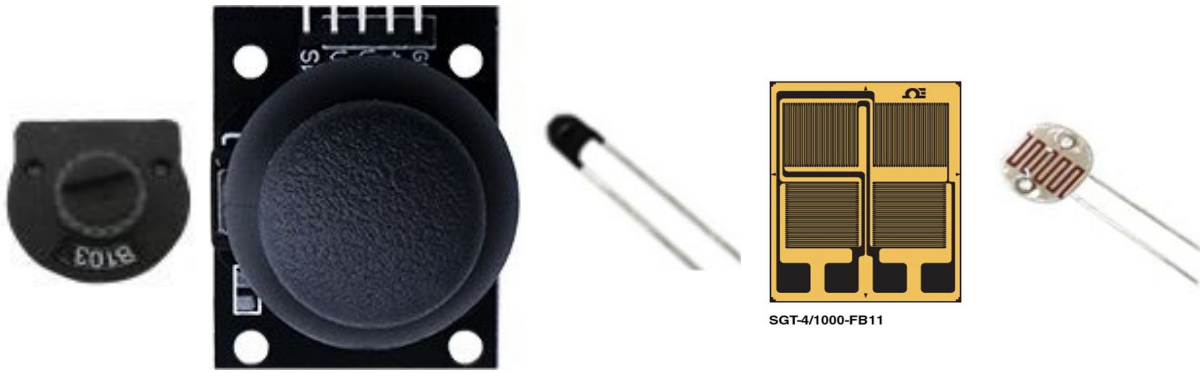

Figure 6.1 Potentiometer, Joystick Controller, Thermistor, Strain Gage and CdS Photocell (LDR)

In this lab we will continue our use of a CdS photocell or LDR - variable resistance sensors that respond to levels of light – from Lab 5 to provide a variable analog input to the Arduino microcontroller. We will use those changes to control PWM (pulse-width modulated) digital I/O output signals to change the color displayed by a RGB LED which can create millions of possible color outputs.

Revision 211021: This document is available electronically at:
http://www.CharlesRubenstein.com/222/**06Lab_LK.pdf**

Lab 5F, 5G & 5H REVIEW: PHOTOCELL ANALOG INPUTS

As we saw in Lab 5, an input is any signal *entering* an electrical system and both **digital** and **analog** devices can be inputs. Digital signals are either a ZERO (0v) OFF or a 1 (+5v) ON input. Digital inputs can come from switches and Keyboards or O/1 outputs from specially designed sensor devices.

READING ANALOG INPUTS

Each digital input or output needs a pinMode command in the **setup** function:

pinMode (pinNumber, INPUT); *// Make sure to use ALL CAPS for **INPUT***

*However, analog inputs are defined inside the **loop** function using analogRead:*

"NAME" = analogRead(pin#); // Note the spelling of **analogRead**

Example:

val = analogRead(5); // read value of Analog Input Pin #5 and store in **val**

*Recall that when we name variables and functions using two-word names the first name is NOT capitalized, but the second name is – e.g., **firstSecond**.*

We can connect analog inputs using any of the six analog Input pins # A0 through # A5 – when we specify these pins we do NOT need to use the "A" in front of 0-5 since the analogRead or digitalRead commands define which is being used. The Analog Input at Analog Port 'pin#' is read and its value stored in the variable space reserved for 'NAME' - an analog can be any value between 0 and 5 volts, but the analog-to-digital converter (ADC) internal to the Arduino will convert this into discrete values from 0 through 1023.

From Lab 5F: Resistor Controlled "#12" LED Blinking

Varying the Blink Rate using an Analog Input at A0 and an LED at I/O #12

```
        int analogPin = 0;      // val = pin #A0
        int val;            // val = make sure that A0 is an integer
void setup() {
        pinMode(12, OUTPUT); // LED at digital I/O pin #12 }
void loop() {
        val = analogRead(analogPin); // Read A0, set delay value
        digitalWrite(12, HIGH); delay(val); // Turn ON LED, wait
        digitalWrite(12, LOW);  delay(val); // Turn OFF LED, wait }
```

Cadmium Sulfide (CdS) Photoresistor or Photocells

*Photoresistors – or Light Detecting Resistors (LDR) are specially designed resistors whose 'dark' and 'light' resistance values are great enough for the device to act as a light detector, or even as an **amount of light** **detector.*** A photoresistor is typically put into a voltage divider circuit with either a fixed **Pull-down** resistor circuit as we saw in Lab 5.

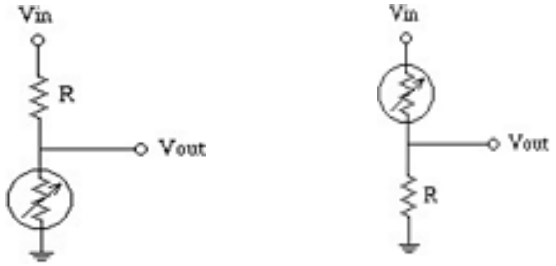

Figure 6.2 (Fig 5.10). Photoresistor Inputs with Pull-Up and Pull-Down Resistors

The CdS photocells in the Elegoo Kit have a Dark resistance of about 50,000 Ohms and a LIGHT resistance of about 500 Ohms – they are typically put into the **Pull-down** resistor voltage

divider circuit using a 1000 Ohm pull-down resistor. In darkness we then have a 50K and 1K voltage divider representing a potentiometer with V_{out} equaling nearly zero volts:

V_{out}DARK = 5v x 1/51 = 0.098 volts

In very bright light the resistance is 500 Ohms and the 1000 Ohms pull-down resistor makes a voltage divider with a roughly 3.33 volt maximum output:

V_{out}VERY BRIGHT = 5v x 2/3 = 3.33 volts

In moderately bright light the resistance is 1000 Ohms and the 1000 Ohms pull-down resistor makes a voltage divider with a roughly 2.5 volt maximum output:

V_{out}MODERATLEY BRIGHT= 5v x 1/2 = 2.5 volts

If the maximum analog digitized value for 5 volts is 1023, these translate into:

A0 DARK = (1023 x 0.098) / 5 = **20**
A0 VERY BRIGHT LIGHT = (1023 x 3.33) / 5 = **675**
A0 MODERATLEY BRIGHT LIGHT = (1023 x 2.5) / 5 = **511.5**

Thus the analog voltage into **A0** *INCREASES with light*.

In Lab 5 we used the analog input values directly to set the delay times for the BLINK sketch. The Arduino can't always use these V_{out} values directly. We often need to process them and use processed values to do things…

In the Elegoo Photocell Lab 10 example for controlling the output latch to create an eight-LED display according to light levels, the code shows dividing the incoming analog input values (see above calculations) by 57 to set up nine levels of LED display, and sets a new range of levels from 0 to about 18 (17.95) instead of 0 through 1023. We will review this code set in Lab 7, but note that when the photocell equals the 1K pull-down resistor, V_{out} would be 1023/2 = 511.5 which, when divided by 57 is greater than 8 and is used to light all eight LEDs. If it were 50K, then V_{out} would be 1023/51 = 20 which, when divided by 57 is much less than 8.

EXCERPTS FROM Elegoo Lab 10 Photocell Sketch – ***NOTE ONLY SOME LINES SHOWN!***:

```
int lightPin = 0;
void loop()  {  int reading  = analogRead(lightPin);
  int numLEDSLit = reading / 57;  // 56.83 = 1023 / 9 LED possibilities 0-8 Lit / 2 when LDR=1K
// When numLEDSLit > 8 the code that follows lights ALL the LEDs…
  if (numLEDSLit > 8) numLEDSLit = 8;
  leds = 0;   // no LEDs lit to start
  for (int i = 0; i < numLEDSLit; i++)
  {   leds = leds + (1 << i);  // sets the i'th bit   }
  updateShiftRegister();
}
```

Don't worry if you don't fully follow this, we discuss it further in Lab 7 Controlling Eight-LED Displays.

ABOUT LAB 6

This lab continues our investigation on how we can use the Arduino to sense analog inputs and use their values to control digital and/or pulse-width modulated (PWM) outputs to simulate an analog output voltage by adjusting the duty cycle of a digital output.

We will also introduce the RGB LED and show how it can be used with variable resistive sensors (or a potentiometer) to create millions of colors.

Revision 211021: This document is available electronically at:
http://www.CharlesRubenstein.com/222/**06Lab_LK.pdf**

6A. PRINTING VALUES ON THE IDE SERIAL MONITOR

One of the really important Arduino IDE functions is that it includes a built-in Serial Port Monitor that can be used to view the inputs and/or results of calculations done by your sketch. There are several new commands that make that happen. They include the **Serial.begin**, **Serial.print**, and the **Serial.println**:

THE "Serial.function" COMMANDS

Three of the most commonly used Serial Monitor commands are shown here with their syntax and an example of how they are used:

Serial.print ("Content ") ;
Start printing a line of text. Begins the line with '**Content** ' – note the space after the word and before the ending quotes – and waits for additional commands.
Serial Monitor Display:
Content

Serial.print (variable);
Print the value of "variable" without any spacing before or after its current value, e.g., XXX.
Serial Monitor Continued Display:
Content XXX

Serial.println ("content2.") ;
Print the text '**content2.**' – note there is no space before or after the word and before the end quotes – and then start additional printing on the next line. The 'ln' tells the monitor to do a 'carriage return' and 'line feed' as if it were a printer.
Serial Monitor Continued Display:
Content XXX content2.
(next line goes here…)

6A.1. Download **serial.txt** and copy its contents (below) into the IDE, then wire the CdS photocell and 1K resistor as noted for a pull-down resistive sensor input.

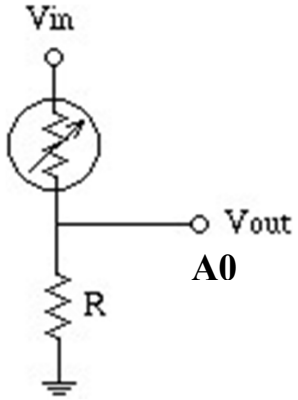

Fig. 6.3 Pull-Down Sensor Circuit

```
// serial.txt  Uses: analogRead , Serial.begin , Serial.print, Serial.println
int sensorVal = 0; int sensorPin = 0; // Global Parameters
void setup ( )  {
    Serial.begin (9600) ; // Initialize the Serial Port
    pinMode(sensorPin, INPUT);     } // End Setup function
void loop ( )    {
sensorVal = analogRead(sensorPin); // get reading
Serial.print ("voltage = "); // print text
Serial.print (sensorVal*10); // print reading times ten
Serial.println (" millivolts"); // print text, next line
        delay(100);     } // Repeat the Loop function
```

After wiring the circuit, save the sketch, upload it, and open the IDE's Serial Monitor. You should see the onboard BLUE TX LED flashing on the Arduino for each 'analogRead' and see the output as shown in Figure 6.4.

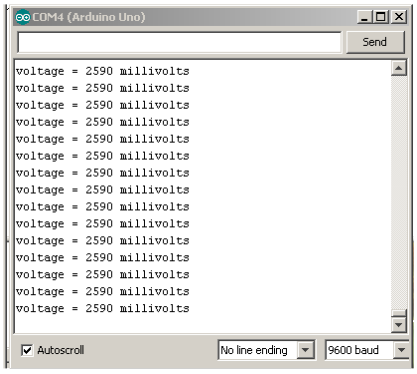

Fig. 6.4 Serial Monitor Screen

Question 6A.1: Take a screen shot – call it **millivolts.jpg**

Revision 211021: This document is available electronically at:
http://www.CharlesRubenstein.com/222/**06Lab_LK.pdf**

6A.2. *We can use the LDR sensor input and convert our input readings to a real voltage using floating point variables. The variable **sensorVal** then prints as the actual input voltage Vout on the Serial Monitor using the float commands:*

float "NAME";
In the sketch sensor.txt, below, we set "voltage" as the name of the input signal.
float input2volts = 5.0/1023.0; // Map the binary input to voltage
Then we map the 0.0 to 5.0 volt analog input voltage into the digital range of 0 to 1023 such that we see each digital number is 4.888 x 10^{-3} and we can, in the Loop function, then multiple 5.0/1023.0 to recover the original input voltage.

Download **serial.txt** and copy its contents (below) into the IDE. You already have Figure 6.3, the CdS photocell and 1K resistor, wired for a pull-down resistive sensor input.

```
// sensor.txt  Uses: floating point math (float)
int sensorVal;          // variable declaration only
int sensorPin = 0;      // variable declaration, assigned to pin #0
float voltage;          // Voltage of the input signal
float input2volts = 5.0/1023.0;  // Map the binary input to voltage
void setup ( ) {
  Serial.begin (9600) ;          // Setup the Serial Monitor @ 9600 BAUD
}
void loop ( ) {
  sensorVal = analogRead(sensorPin);      // Get the input from A0
  voltage = float(sensorVal)*input2volts ;  // Calculate the actual voltage
  Serial.print ("sensorVal = ") ;
  Serial.print (sensorVal); Serial.print (",") ;  Serial.print ("actual voltage = ") ;
  Serial.print (voltage) ;  Serial.println (" volts");
  delay(100)
}         // Repeat
```

Using the code of "sensor.ino" above take a screen shot or photo or two of the simulated sensor input range using a CdS Photocell.

Question 6A.2: Do a screen shot of the Monitor, label it: **sensorVal.jpg**

Each Serial Monitor output line should read: sensorVal = **xxxx**, actual voltage = **yyyy** volts
*where **xxxx** is the sensorVal variable used in the program and where **yyyy** is the actual voltage coming into the analog port*.

6A.3. *We can use the LDR sensor input and measure the actual input voltage and compare it to the Serial Monitor values of the input voltage from your CdS photocell.*

Using Dark, Ambient light and Bright light levels, confirm these output readings with your multimeter:
Question 6A.3a: DARK; sensorVal = _____ ,voltage = ____ volts
Question 6A.3b: AMBIENT; sensorVal = _____ ,voltage = ____ volts
Question 6A.3c: BRIGHT; sensorVal = _____ ,voltage = ____ volts

Revision 211021: This document is available electronically at:
http://www.CharlesRubenstein.com/222/**06Lab_LK.pdf**

6B. CONTROLLING THREE LEDS

Before we investigate the RGB LED and how to control it, let's try to control three individual RED LEDs (or a Red, Yellow and Green LED). We will also see how to use the 'define' command to make our code easier to read.

THE DEFINE COMMAND

One of the important new commands we will use in this Lab is the **define** command. The **define** command creates a way to name "**NAME**" the I/O port you want to use in a sketch to make its use more easily understandable without adding comments. The **define** command:

#define "NAME" "value"

assigns the NAME to a specific "**value**" - in the program, the "value" will be compiled rather than the NAME. NOTE there is no ";" at the end of the line and ***Name*** *is always case sensitive!*

Examples (insert in sketch BEFORE the setup function):

```
#define LED 13       // sets LED to digital I/O 13
#define BLUE 10      // sets BLUE to digital I/O 10
#define BUTTON 7 // sets BUTTON to digital I/O 7
```

So… in our setup function we could use the following code:

void setup() { **pinMode(13, OUTPUT); pinMode(9, OUTPUT); pinMode(10, OUTPUT); … }**

Or, if we use **define** before setup, in the 'global parameters' part of the header, we would use:

```
#define L 13
#define RED 11
// And so on and then go to the setup function:
```

void setup() { **pinMode(L, OUTPUT**); **pinMode(RED, OUTPUT); pinMode(GREEN, OUTPUT); … }**

6B.1. The full sketch BlinkThree.txt, using define is:

```
/* Blink Three: Uses define.
Turn the on board "L" LED and three (3) External LEDs ON and
then OFF for one second */
// Global Parameters:
#define L 13
#define RED 11
#define GREEN 9
#define BLUE 10 // this could also be yellow
// Setup Function:
void setup() {
  pinMode(L, OUTPUT); pinMode(RED, OUTPUT);
  pinMode(GREEN, OUTPUT); pinMode(BLUE, OUTPUT); …
// External LEDs on Digital I/O Ports #9-#11
}
// Loop Function:
void loop () {
digitalWrite(L, HIGH); digitalWrite(RED, HIGH); digitalWrite(GREEN, HIGH); digitalWrite(BLUE,
HIGH);
          // sets "L" and all external LEDs "9-11" ON
 delay(1000);     // wait for a second
digitalWrite(L, LOW); digitalWrite(RED, LOW); digitalWrite(GREEN, LOW); digitalWrite(BLUE, LOW);
          // sets "L" and all external LEDs "9-11" OFF
 delay(1000);     // wait for a second
}                          // repeat
```

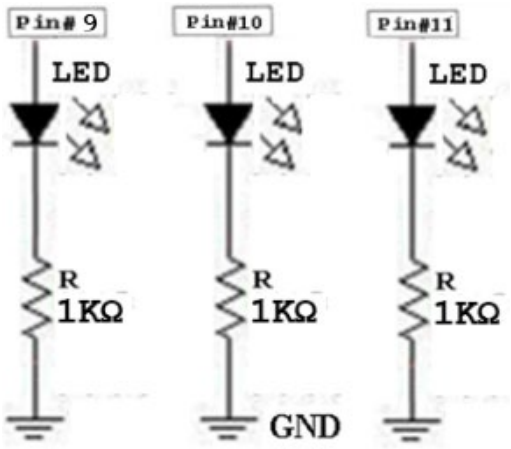

Fig. 6.5 Blink Three: Three External LEDs

NOTE: We are using pins 9, 10, and 11 as these are all PWM (~) pins.

6B.1. Wire the schematic (using 220 ohm to 1K resistors). Did all three external LEDs blink?

Revision 211021: This document is available electronically at:
http://www.CharlesRubenstein.com/222/**06Lab_LK.pdf**

6B.2. *Instead of controlling three individual LEDs from our Arduino, could we use a single three-LED (RGB) device in our circuit? If so, how does it work?*

There are two variations of the RGB LED. The RGB LED in the Elegoo kit has the individual LED *cathodes* tied together. This is called the *Common Cathode* configuration, where pin 2 – the Common Cathode/Negative Terminals of all the internal LEDs are tied to ground. Writing **HIGH** to a digital I/O pin turns one of the LEDs **ON**.

The wiring for a common-cathode RGB LED is as shown here:

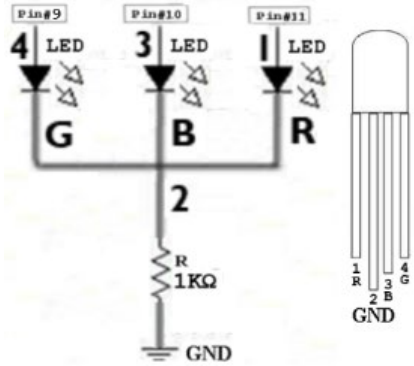

Fig 6.6 Elegoo Common-Cathode RGB LED

*NOTE: If you are using another kit, you might have a Common Anode which has all the anodes of the internal LEDS tied together meaning Pin 2 must be connected to +5 volts. In Common Anode devices, Writing a **LOW** to a digital I/O pin turns one of the LEDs **ON** as that completes the circuit from +5v to ground through the Arduino. If that is the case, please email me immediately! And note that this is the correct wiring diagram:*

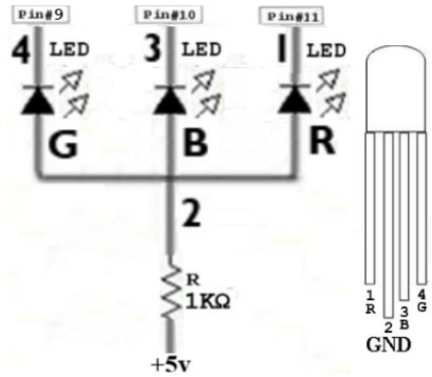

Fig 6.7 Common-Anode RGB LED

*Regardless of which device type you have, instead of the original three LEDS with three 1K resistors, you have one RGB LED and only require **one 1K resistor** to use it!*

The Blink Three sketch can now run and you will NOT see the individual red, green and blue LEDs turn on, but rather all three will blink on and off at the same time.

What color do you see when BlinkThree turns all three RGB LEDs on at the same time?

Question 6B.2. The color when all RGB LEDs are on is: _____

6B.3. To see the individual LEDs turn on and off we will need to modify the Blink Three code so that each LED is turned on and off at one time using the BlinkInTurn sketch:

```
// BlinkInTurn: Turn each LED in an RGB LED ON and then OFF.
// Global Parameters:
#define L 13
#define RED 11
#define GREEN 9
#define BLUE 10 // this could also be yellow
// Setup Function:
void setup() {
  pinMode(L, OUTPUT); pinMode(RED, OUTPUT);
  pinMode(GREEN, OUTPUT); pinMode(BLUE, OUTPUT);
}
void loop () {
// Turn on Red, Green, Blue LEDs one at a time
// set only RED LED ON
digitalWrite(GREEN,LOW); digitalWrite(BLUE, LOW); digitalWrite(RED,HIGH);
delay(1000);    // wait for a second
// set only GREEN LED ON
digitalWrite(GREEN,HIGH); digitalWrite(BLUE,LOW); digitalWrite(RED,LOW);
 delay(1000);   // wait for a second
// set only BLUE LED ON
digitalWrite(GREEN,LOW); digitalWrite(BLUE,HIGH); digitalWrite(RED,LOW);
 delay(1000);   // wait for a second
}          // Repeat
```

Of course, if desired we could include some code that would turn asll three LEDs on at once before cycling…

6B.3. were you able to see the individual colors flash at one second intervals?
ANS: _____YES _____ NO

We have seen that we can turn ON each individual LED in the RGB LED to select between the colors **Red** = Red, **Green** = Green, and **Blue** = Blue.

What LEDs would we need to turn on at one time to show the following colors?

Question 6B.4a Yellow: ___ RED ___ GREEN ___ BLUE
Question 6B.4b Purple: ___ RED ___ GREEN ___ BLUE
Question 6B.4c Cyan: ___ RED ___ GREEN ___ BLUE
Question 6B.4d White: ___ RED ___ GREEN ___ BLUE

Other colors and hues are available as well if we could only modify the intensity of each of the LEDs in the RGB LED…

It was not accidental that we used digital I/O Ports 9, 10 and 11 in Lab D3. If you look closely, you will notice each of these ports (*as well as digital I/O Ports 3, 5, and 6*) all have a "~" in front of their port numbers. These six digital I/O Ports are NOT the same as all the rest. They, and ONLY they, can provide the standard digital output of a zero or one, but they are also capable of providing a Pulse-Width Modulated (PWM) output signal.

6C. PULSE WIDTH MODULATION

The Arduino uses PWM to *simulate an analog voltage output* yielding varying widths of digital pulses. We need to use PWM because although there *is* an *analog-to-digital converter* (ADC) for our six analog inputs (#A0-#A5), there is no *digital-to-analog converter* (DAC) in the Arduino and many output devices require an 'analog' type of signal to work. These include LED (controls light intensity), motors (controls speed) and audio output devices (creates a simulatedanalog sound).

The Arduino can turn its digital output ports On and OFF 1000 or more times a second using the standard digitalWrite commands, e.g.,

digitalWrite(pin#, HIGH); delay(2);
digitalWrite(pin#, LOW); delay(2);

to create a 50% duty cycle with 2 milliseconds ON and then 2 milliseconds off. The frequency of the output is 1/T where the T is the total time per cycle. In this case, we see a 4 millisecond ON/OFF period for a frequency of 1/0.004 seconds = 250 cycles per second (Hertz). Our ears can hear up to 15,000 to 20,000 Hertz, which requires a much smaller period!

THE analogWrite Command

Once we define the digital I/O Pin using one of the PWM digital I/O pins (3, 5, 6, 9, 10 or 11) we can define the duty cycle of the output. To simulate an analog output voltage, the microcontroller varies the duty cycle of one of these output ports using a technique called Pulse Width Modulation or PWM using the analogWrite command with integer values from 0 to 255.

The use is:

analogWrite(pin#, "value");

where "value" can be any integer from 0 through 255 where it corresponds to the duty cycle of the output pulse where 0 = 0% or OFF, 255 = 100% or ON, and 127 = 50% simulates a 2.5 volt output.

Using the **analogWrite** command with the value **127** in our code:

analogWrite(pin#, 127);

we can create a **50% duty cycle** output which is called Square Wave and yields 2.5 volts at the I/O port.

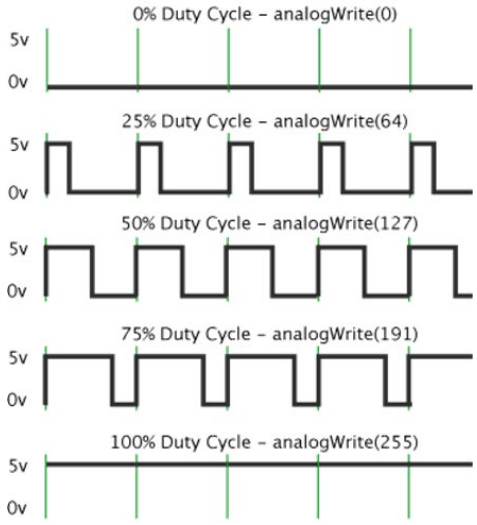

Figure 6.7 Duty Cycles and PWM outputs

Revision 211021: This document is available electronically at:
http://www.CharlesRubenstein.com/222/**06Lab_LK.pdf**

The overall effect of Pulse Width Modulation (PWM) on an LED comes from the fact that LED 'sees' the *average voltage output*:

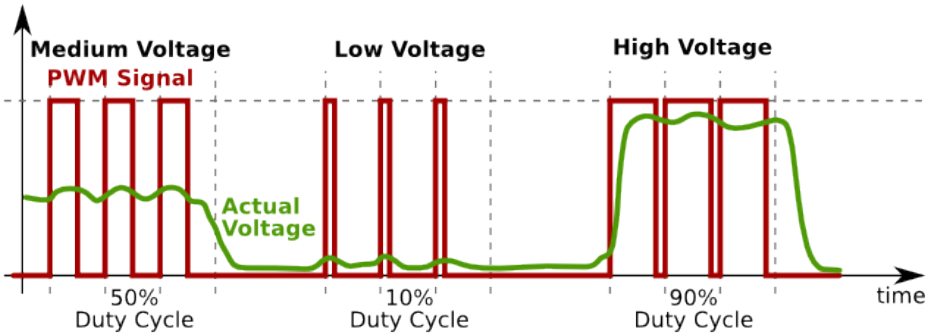

Figure 6.8 PWM Effective Output Voltages

Thus, if the full voltage (100% = 255) would be 5.0 volts output, a 50% duty cycle generates the equivalent of a 2.5 volt output and 10% (26) would generate an approximately 0.5 volts output. The effect of this modulation is seen as a dimming *or fading* of the LED's light output.

Use your RGB LED connected to Pin 9 (GREEN LED) with the example code sketch example:
File → Examples → 01.Basics →Fade
Only Pin 9 is used in that sketch.

6C.1a. What happens when you run the FADE sketch?
ANS: _____

6C.1b. Fading/Dimming Rate Change
Which lines of code in Fade are needed to be changed, added, or deleted, to change the rate of the fading in and out in this circuit?
ANS: _____
*(Hint - there are at least two different ways to do this.. The easiest is to set the **fadeAmount** to a higher value, e.g., **10**)*

6C.1c. Save the sketch as **FadeRate**.

Lab 6D: Simple RGB LED Color Mixing

Starting with the sketch of USK Circuit 3: **Examples → USK → Circuit_03**
which we can save as **colorMix** we note a few changes to the *Global Parameters* section of the sketch – just before the setup function - where instead of **int** we are using **const int** which requires that the integer value defined NEVER changes:

```
const int RED_PIN = 9;
const int GREEN_PIN = 10;
const int BLUE_PIN = 11;
```

We can use this technique instead of using 'define' if the values are constant integers.

We should also note that the Loop function has, up until now, been the only set of codes we used in the sketch after our Setup was accomplished.

In most computer programs, we use 'subroutines' or 'macros' which are snippets of code that we "call" as needed *FROM* the Loop function main program. In this color mixing application we will develop two separate subroutines: **mainColors** and **showSpectrum**.

So… our new sketch has the following structure:

Header/Global Parameters
- Setup Function
- Loop Function
  - mainColors - a subroutine Function
  - showSpectrum – a subroutine Function which includes **showRGB**

Note that the order of the other functions does not matter. They will be 'called' from Loop (etc.) as needed. When called, Loop will go to that function and stay in it until it has completed its task and then returns to the Loop function at the next line of code. The Loop function will control when and how the other functions are run, if at all. In the current sketch, BOTH subroutine functions are run, one after the other. However, commenting out one of them in the Loop function means it is there, but will NOT be run…

The Header/Global Parameters, Setup and Loop look like this:

```
// Global Parameters
const int RED_PIN = 9;
const int GREEN_PIN = 10;
const int BLUE_PIN = 11;
int DISPLAY_TIME = 100;  // In milliseconds

void setup( ) {
  pinMode(RED_PIN, OUTPUT);  pinMode(GREEN_PIN, OUTPUT);
  pinMode(BLUE_PIN, OUTPUT);
}
void loop() {
  mainColors( );              // The function mainColors() steps through 8 colors
  showSpectrum( );   // The function showSpectrum() steps through all colors
}

mainColors() {
// code to step through 8 colors:  Red, Green, Blue, Yellow, Cyan, Purple and White, goes here
}
showSpectrum() {   // code creates "x" = int color, goes to showRGB(int color) with "x"
// code showRGB(int color) steps through ALL color combinations of Red, Green, and Blue
// with each going from 0 through 255 in intensity
}
```

The subroutine functions are a bit long and will not be repeated here. You should be able to understand what they are doing and how they are doing it from the comments on the code.

The **mainColors() subroutine function** steps through 8 colors: **Red, Green, Blue, Yellow, Cyan, Purple and White.** The **showSpectrum() subroutine function** steps through ALL possible color combinations of Red, Green, and Blue with each going from 0 through 255 in intensity. It does this using the subroutine **showRGB(int color)** which is embedded inside showSpectrum that actually steps through all color combinations.

QUESTION 6D.1: If each RGB LED has 256 different brightness levels, how many unique colors can you create?
ANS _____
QUESTION 6D.2: What was the overall effect of color mixing?
ANS 6D.2 _____

DO NOT BREAK DOWN YOUR RGB LED CIRCUIT!
It will start here in Lab 7!