Engineering Principles Name___________________________________

Electrical Engineering

Color Wheel 4.5.1 Software: Pin Maps

Date_____________________ Team __________

Problem statement

Develop a persistence-of-vision color wheel utilizing 8 RGB diodes and an Arduino UNO R3 board.

Programming: Getting Started

1. Download blinkRGBW.ino from Canvas.
2. Create a new file using the blinkRGBW.ino file as a starting point (save blinkRGBW.ino to your drive under another file name).
3. Add comments to the beginning of your file including your group number, names of members of your group and a brief description of the project (see the above problem statement).

Programming: Constants

Constants can be used to give a name to a value without using programming memory. Use the **const** modifier in front of the **int** type modifier and set a value

```
const int name = value;
const int ledPin = 14;
```

Arduino uses the `digitalWrite`(*pin*, *value*), `digitalRead`(*pin*, *value*) and `pinMode`(*pin, value*) functions to program its I/O pins. The *pin* argument expects a number representing the physical hardware pin. When reading code it is less confusing to use a descriptive name for the pin rather than just a pin number. Using constants also makes the code easier to maintain. By using **constants** we can define a **pin map** in one place in the code and then use the reference everywhere else. If the **pin map** changes, we only have to update our code in one place.

4. Create a **pin map** using `const` declarations. Create constants for the eight row pins, six color pins and the magnetic senor analog I/O pin. Talk with your group to make sure that your aliases agree with how the project will be connected. The digital I/O pins are numbered from 0 through 13. The analog I/O pins are A0 through A5.

Example:

```
const int red1=12;
const int red2=11;
const int row0=0;
const int row1=1;
```

5. Create **constants** for the three LED colors: red = 0, blue = 1 and green = 2.

Color Wheel 4.5.1 Software: Pin Maps

Date____________________ Team __________

Programming: Arrays

type arrayName [*size*] = { *value1*, *value2*, … , *lastValue* } ;

Arrays are used to access lists of numbers using an index or counter to tell the code which element you want. C uses square brackets [ ] for the array indices. Arrays are defined by using one of the variable types followed by the name of the array. In the declaration specify the size of the array by entering that number in square brackets. Declare these arrays outside of any of your functions so that all the code in your project file will have access to it. (When using arrays the number will range from 0 to 1-size.)

Example:

```
const int rowPin[8] = {
        row0, row1, row2, row3, row4 ,row5 ,row6 ,row7
};
```

6. Create an int array that contains the row pin aliases in order (see above).
7. Make two more arrays (colorPin1 and colorPin2) for the two sets of common color pins.

Programming: Functions

Functions make code easier to read, write and debug. A function is a collection of statements that we may want to use multiple times in our code. Functions may accept values in an arguments list. They may also return a value.

```
returnType functionName (type1 argument1, type2 argument2, … ) {
        program statement1;
        program statement2;
        ---
        return(returnValue);
}
```

If the function doesn’t return a value or use arguments, use `void` to indicate that.

8. Write a function that will turn all six of the color pins off. Give it an appropriate name. It won’t have any arguments so the first line will look something like this:

   ```
   void clearColorPins(void) {

   }
   ```

   To turn off a colorPin use the `digitalWrite()` command to set the pin value to `LOW`. This function should have a line for each of the six pins.

9. Write another function to turn on one **pair** of the color pins. The function should accept an `int` argument called `color` that tells which color we want to turn on. Before turning the color pins

on, call the function written above so that we don't ever have two different colors on at the same time.

```
void setColorPin(int color) {
      clearColorPins();
      digitalWrite(colorPin1[color],HIGH);
      …;
}
```

10. Write functions to set and clear the individual row pins. Row Pins should always be set to a LOW output. Toggling the pin mode from OUTPUT to INPUT turns each LED on and off

```
void setRowPin(int row){
      pinMode(rowPin[row],OUTPUT);
}

void clearRowPin(int row) {
      pinMode(rowPin[row],INPUT);
}
```

Programming: Putting it all together

11. In your `setup()` function, write code to set each of the I/O pins to their proper modes, but don't turn on any lights yet. Color pins should be set as OUTPUT with a value of LOW. Row pins should be set as INPUT with a value of LOW. The magnetic sensor pin should be set as an INPUT, but its value should be written as HIGH – this enables a pull-up resistor inside Arduino that the sensor needs.
12. Plug at least one LED with a resistor on the common pin matching the pin-mapping you set above.
13. Write code to turn on and off the LED's using the functions you created above.
14. Test that you can get RED, GREEN, BLUE and WHITE from the LED's. Try to get other colors as well.
15. Test the LED in several positions to make sure that you can turn on all 8 LEDs when we get the hardware.

Finishing Up

16. Save your code and upload it to your group's file space on Canvas. Use a new name so that prior versions of the code are not over-written.