<http://www.hanhualed.com/List.aspx?cid=434>

220 &Omega; current limiting resistors

|  |  |
| --- | --- |
| Max. Forward Voltage | R:1.9- 2.1v, G:3.2-3.4v, B:3.2-3.4v |
| Max. Forward Current | 20mA |
| color | RGB |
| polarity | common cathode |

InGaN/GaAsp :

Red: [Gallium arsenide phosphide](http://en.wikipedia.org/wiki/Gallium_arsenide_phosphide) (GaAsP)

Green: [Indium gallium nitride](http://en.wikipedia.org/wiki/Indium_gallium_nitride) (InGaN)

Blue: [Indium gallium nitride](http://en.wikipedia.org/wiki/Indium_gallium_nitride) (InGaN)

Chip kit pwm output:

(#/255)\* 3.2 V = Average Voltage

Blue, Green, Cathode, Red

Min PWM value red 128

Tricolor LED connection guide

Introduction

In project 14 the brightness of an LED was controlled via Pulse with modulation (PWM). In this project we will use PWM to control the color of a Tricolor LED. If you are not familiar with PWM it is recommended you review project 14. Project 15 will be very similar to the previous project except the LED being used will change color. Generally LEDs are manufactured to emit one color of light. These colors can range from any color in the rainbow to infrared or even ultraviolet. The LEDs color is determined by the material it is made from and the voltage that is applied to it. Depending on the combination of the material and voltage, different wave lengths of light are emitted. We perceive these different wavelengths of light as colors. The Tricolor LED we will use can only generate red, green and blue wave lengths of light. This is because the Tricolor LED is actually consists of a red, green and blue led all in one bulb. By changing the intensities of each LED inside the bulb, we can generate the illusion of different colors. The same principle also applies when mixing paint. Mixing red and green paint results in what looks like yellow paint even though there is no naturally yellow pigment in the mixture. This phenomenon occurs because our brains simply perceive this combination of red and green pigment the same way we perceive naturally yellow pigment. Similarly, our brains perceive the combination of red and green wave lengths of light, the same way the actual wave length of yellow is perceived.

Inventory…

Building the circuit

Before we begin putting the circuit together the structure of the Tricolor LED must be discussed. Since Tricolor LEDs are essentially a red, green and blue LED it would make sense to assume it will have 3 anodes and 3 cathodes for a total of 6 legs. Realistically this isn’t very practical so the red green and blue LEDs inside the bulb all share a common cathode to save room. This means our Tricolor LED will only have four legs as shown in Fig. 1. The longest leg is the common cathode and should always be connected to ground. Follow the steps below to construct the circuit. Note that the wire colors correspond to the LED color they will be controlling.

1. Put the Tricolor LED in the bread board with the same orientation as shown in the figure.

2. Connect the common cathode of the Tricolor LED (the longest leg) to ground.

3. Connect a 220 &Omega; current limiting resistor to each of the LED’s remaining legs.

4. Connect PWM pin 6 (Red brightness control) to the leftmost leg of the Tricolor LED.

5. Connect PWM pin 5 (Green brightness control) to the leg just to the right of the common cathode.

6. Connect PWM pin 4 (Blue brightness control) to the rightmost leg of the Tricolor LED.

Assembly

Programing the circuit

With the circuit built we will now write some simple code to control the color of the LED. Remember the maximum and minimum value for a PWM pin is 0 to 255. In the code below the PWM pins that control red and blue are set to 200 with the setColorBrightness() function. This will make the tricolor LED purple.

The function to setColorBrightness() may seem unnecessary, since the color can be changed directly with analogWrite(). While this is true, bear in mind that it is easy to mix up what pin number corresponds to the color we want to change. Using the if-else-tree inside of the setColorBrightness() function allows us to modify color values using intuitive color abbreviations, 'r', 'g' and 'b'. Furthermore the if-else-tree provides a good point of comparison for introduction of switch statements, which will be covered in the next section.

Theory about switch statements

Description

Figure comparing path of if-else tree and switch statement

Show application using ‘basic color sweep’ code

Theroy about randomizing color combos

Describe random() and randomseed()

Show application with ‘random RGB’

Switch statements

Switch statements control the flow of a program and are very similar to if-else-trees. They compare a variable to a constant case value. If the variable matches the case value, the code contained within the case is executed, otherwise the case is skipped. In the event none of the cases match the default case will be executed. The default case is optional and not needed for the switch statement to function properly. The following is an example of a switch statement.

In the example the case values were integers and single characters. A case value can be any constant value which can be interpreted as an integer value. This includes the boolean constants HIGH, LOW, INPUT OUTPUT, true and false. Data Types that cannot be used as a case value are doubles, floats, strings and arrays. On top of being careful with the case data type, it is also important to be attentive about including breaks in each case. If you forget to add a break in a specific case, the switch statement will execute that case and all cases below it until it sees a break line.

So far switch statements and if-else-trees appear to do the same thing. While this is true, there are distinct advantages and disadvantages when to using switch statements. We know that they have limited use because they cannot compare dynamic variables and are restricted to certain data types. On the other hand switch statements compare only one case making them very efficient. In Fig. 3. and Fig. 4. The differences in efficiency are illustrated to if-else-trees.

The if-else-tree must evaluate every if statement before and after the targeted case. The switch statement, on the other hand, jumps directly to the targeted case and then exits once it is finished. This saves a lots of time in situations with a large number of cases. Furthermore, the efficiency of switch statements are multiplied inside of loops since only on case is compared ever iteration. This will come in handy when writing the code for a basic color sweep in step 3.

With an understanding of how switch statements work we can begin coding the Chipkit to sweep through combinations of colors. This is will be similar to the code used for creating a breathing LED except the principal will used to fade in and out three colors.

Just like in step 2 one function, changeColorBrightness(), will be used to control which color is being changed. The main difference here is we have replaced the if-else tree with a switch statement and put it inside a while loop. The while loop that increments or decrements the brightness of a color until it reaches the limit of the PWM pin. Each iteration of the loop is delayed so we have time to see the color change. The code should make the LED fade from red, to purple, to blue, to aqua blue, and finally to green. This sweep pattern is determined by the order in which red green and blue are faded in and out. Try rearranging the order to get different color sweep combinations. In the next section we will explore how to randomly select a color combination.

Random() and randomSeed()

In order to select a random color combination a random number must be generated. This can be done using the random() function. The random() function can be used one of two ways. The first is entering a max value as the only argument. This will return a random integer from 0 to the max minus one. The second way is entering a max and min value as the arguments. This will return a random integer from min to max minus one. For example

//Ex

random(11)//returns a random integer from 0 to 10   
random(5,16) // returns a random integer from 5 to 15

Now that we know how to generate a random number we can write the code to generate a random color combination. To begin we should consider all the color combinations we want to implement. All of these combinations are listed below.

Red

Green

Blue

Red and Green

Red and Blue

Green and Blue

Red, Green and Blue

Considering there are seven combinations we want to randomly select one each time. This can be achieved by selecting each combination with a switch statement using the random() function to select the case. Inside of each case we want to set the colors to a random brightness. This will ensure there is a chance for every possible color to be generated. The code for algorithm described is shown below inside of randomLEDColorCombo().

Uploading this code to your chipKit should make the tricolor LED randomly jump to new colors. The time between each jump is determined by the delay inside of the loop() function. You may notice some of the colors appear very dim. This is because inside of each case, random(256) has returned a value near 0. You can try experimenting with different min values for the random function to prevent dim colors. You can also specifically tweak the min values in cases 3 through 6. Currently these cases have the possibility of generating a single primary color, assuming the other colors in the case near zero.

Now that we know how to generate random color combinations and implement color sweeps we can combine the two concepts. In this step we will write some code to randomly select a color sweep patterns. Here the order of the pattern will matter. This means there will be 6 different color patterns which are listed below.

Red, Green, Blue

Red, Blue, Green

Green, Red, Blue

Green, Blue, Red

Blue, Red, Green

Blue, Green, Red

Each pattern will be implemented as a case of a switch statement. Just as in step 4 the case will be selected by using the random() function. Inside of each case we will use the changeColorBrightness() function we wrote in step 3 to fade the colors in or out. The code needed to implement this is shown below inside of the randomColorSweep() function.

Once this code is uploaded the tricolor LED should randomly sweep through different color patterns. The colors either fade in or out after each function call. This is controlled by inverting the value of the global variable brightnessDirection inside of each case. The fade time can be tweaked by changing a variable inside of the randomColorSweep() function. By default it delays five milliseconds between each incrimination of a colors brightness. This is done using delay() within called inside of changeColorBrightness(). If you want to delay for periods less than a millisecond you can comment out the delay()and uncomment delayMicroseconds() for greater precision. Remember that 1 millisecond is equal to 1000 microseconds so, using delay times less than 300 with delayMicroseconds() will make the LED flash rather than fade.

Switch statements