10

LED Fader Project

This is the first of three projects designed to make use of Python and the GPIO pins to control the color of light coming from an RGB LED. The project combines the use of the Tkinter library to create a user interface and the RPi.GPIO libraries' PWM feature to control the brightness of the three channels of the LED (red, green, and blue).

Figure 10-1 shows the LED hardware built onto breadboard and Figure 10-2, the user interface used to control it on your Raspberry Pi.

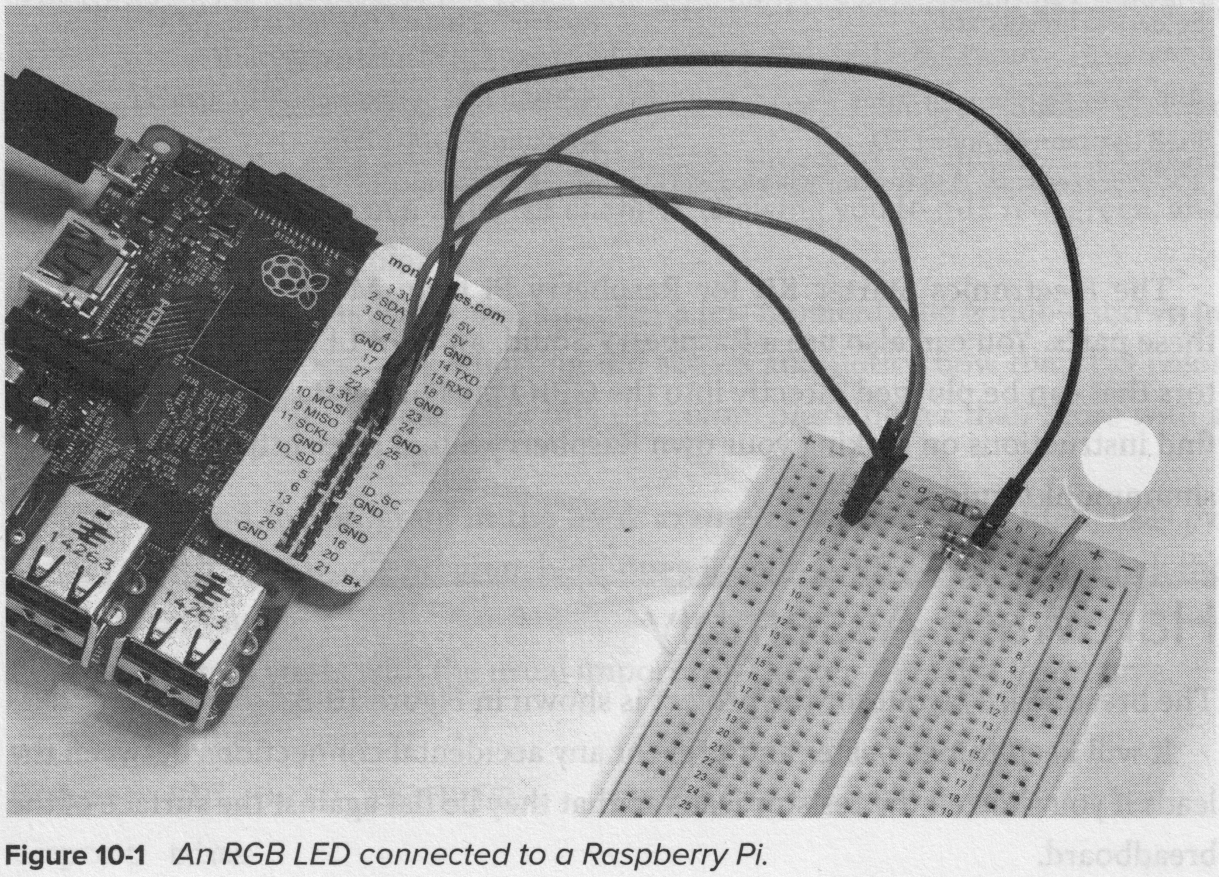

Figure 10-1 *An RGB LED connected to a Raspberry Pi.*

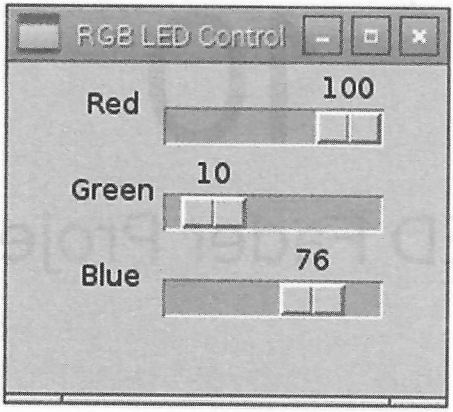

Figure 10-2 *A Tkinter user interface for controlling the LED.*

What You Need

To build this project, you will need the following parts. Suggested part suppliers are listed, but you can also find these parts elsewhere on the Internet.

Part	Suppliers
Solderless breadboard	Adafruit (Product 64), Sparkfun (SKU PRT-00112), Maplin (AG09K)
Female-to-male jumper wires	Adafruit (1954), Sparkfun (PRT-09385)
RGB common cathode LED	Sparkfun (COM-105)
3 × 470Ω resistor (1kΩ resistor will also work)	MCM Electronics (34-470)

The Electronics Starter Kit for Raspberry Pi from MonkMakes includes all these parts. You can also use a Raspberry Squid, an RGB LED with built-in resistors that can be plugged directly into the GPIO pins of the Raspberry Pi. You can find instructions on making your own Raspberry Squid here: https://github.com/simonmonk/squid.

Hardware Assembly

The breadboard layout for the project is shown in Figure 10-3.

It will keep things neater and prevent any accidental connections between the leads if you shorten the resistor leads so that they lie flat against the surface of the breadboard.

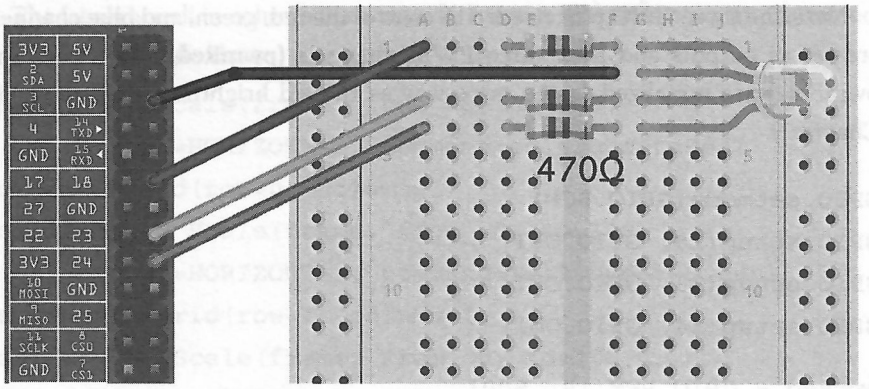

Figure 10-3 *The breadboard layout for an RGB LED.*

The RGB LED will have one leg that is longer than the others. This is the "common" lead. When you buy your RGB LED, make sure that it is specified as being "common cathode." This means that the negative terminals of each of the red, green, and blue LED elements are all connected together.

Software

The software for this project has some similarity with the experiment in Chapter 9, where you controlled the brightness of a single red LED by typing in a value between 0 and 100. However, in this project, instead of entering a number, Tkinter is used to create a user interface that has three sliders in a window. Each slider controls the brightness of a different channel, allowing you to mix red, green, and blue light to make any color.

Run the program (as superuser) and after a few moments the window shown in Figure 10-2 will appear. Try adjusting the sliders and notice how the LED color changes. LEDs with a diffuse body mix the colors much better than those with a clear body.

You can find the program in the book examples as the file 10_RGB_LED.py. Rather than list the whole program here, open it up in IDLE while I go through the code in sections.

The program starts with the usual imports of Tkinter, RPi.GPIO, and time.

```
from Tkinter import *
import RPi.GPIO as GPIO
import time
```

Next, the three GPIO pins needed to control the red, green, and blue channels are set as outputs and then three PWM channels (pwmRed, pwnGreen, and pwmBlue) are initialized in the same way as the led brightness experiment in Chapter 9.

```
GPIO.setmode(GPIO.BCM)
GPIO.setup(18, GPIO.OUT)
GPIO.setup(23, GPIO.OUT)
GPIO.setup(24, GPIO.OUT)

pwmRed = GPIO.PWM(18, 500)
pwmRed.start(100)

pwmGreen = GPIO.PWM(23, 500)
pwmGreen.start(100)

pwmBlue = GPIO.PWM(24, 500)
pwmBlue.start(100)
```

The user interface uses a grid layout to set the positions of the three labels inside the __init__ method:

```
class App:

    # this function gets called when the app is created
    def __init__(self, master):
        # A frame holds the various GUI controls
        frame = Frame(master)
        frame.pack()

        # Create the labels and position them in a grid layout
        Label(frame, text='Red').grid(row=0, column=0)
        Label(frame, text='Green').grid(row=1, column=0)
        Label(frame, text='Blue').grid(row=2, column=0)
```

The labels are all in column 0, on rows 0, 1, and 2. The sliders are implemented by the Tkinter Scale class.

```
scaleRed = Scale(frame, from_=0, to=100,
      orient=HORIZONTAL, command=self.updateRed)
scaleRed.grid(row=0, column=1)
scaleGreen = Scale(frame, from_=0, to=100,
      orient=HORIZONTAL, command=self.updateGreen)
scaleGreen.grid(row=1, column=1)
scaleBlue = Scale(frame, from_=0, to=100,
      orient=HORIZONTAL, command=self.updateBlue)
scaleBlue.grid(row=2, column=1)
```

Each Scale object is constructed with "from_" (with underscore after the name) and "to" parameters that specify the range of values that the scale can set, so when the slider is far left, the value will be 0 and when it's all the way over to the right, the value will be 100.

Each of the scales also has a "command" attribute where the name of a method is specified. This method will be called whenever the slider position is changed. For the red channel this method is called "updateRed."

```
def updateRed(self, duty):
    # change the led brightness to match the slider
    pwmRed.ChangeDutyCycle(float(duty))
```

Whenever the update function for a particular channel is changed, it just changes the PWM duty of that channel to the new slider value of between 0 and 100.

The remainder of the code initializes the window and starts the Tkinter main loop running. Note that the loop is set running inside a try/finally block so that if the window is closed, the GPIO pins are automatically cleaned up (set to inputs).

Summary

This is a simple project to get you started with some GPIO programming. In the next chapter, you will use a display module that uses a I2C serial interface to connect to the Raspberry Pi and make a digital clock.