Word Count : 1104

Video, Working with LEDs & Buttons

In this video, we will learn about Pulse Width Modulation and implement it on LEDs. Later we will learn to work with Multiple LEDs and individually control their brightness. Finally, we will learn to work with Buttons.

You have now learned to switch an LED ON and OFF using Python. What if i told you that we could do much more than turning the LED ON & OFF. PWM, also known as Pulse Width Modulation, can be used to control the brightness of an LED. PWM is a method of controlling the average power delivered to the device by essentially switching the power ON and OFF at a very high frequency. Due to the human eye’s inability to see very fast switching, we perceive this as less power. With this concept, we can control the brightness of LEDs, speed of motors, and so much more. In a PWM Signal, one period is the total sum of ON Time and OFF time. The effective power delivered depends on the duty cycle or power cycle, which is the fraction of the total period when the system is powered ON. So, if the duty cycle is 50%, it means that the LED will be ON with Half the brightness.

GPIO Zero comes with a class for implementing PWM in an LED called PWMLED. Let’s see it in action. Open the LED\_PWM.py code from the repository and run the script. Now watch how the LED reacts. You can see that the brightness varies from none to half to full brightness on each passing second. In the script, first, we imported the PWMLED and sleep class from the gpiozero and time modules, respectively. Then we created an instance of this class by assigning it to a variable. We have entered the GPIO number, ie 17 as the argument for the class. Then we created an infinite while loop. Inside the while loop, we use the dot operator to set the value of the variable inside the instance of the PWMLED class called value. We can provide any values from 0 to 1 to this, where 0 means 0% duty cycle for the PWM signal, which means OFF. While 1 means 100% duty cycle, which means full brightness or ON. In this example, by setting the “value” to 0.5, a PWM signal with 50% duty cycle is created, effectively making the LED glow in half brightness. The sleep(1) instruction gives a delay of 1 second between each PWM Changes. Now, I will give you an activity. Tinker with this code, and make the LED gradually pulse like shown here. You will need to fine-tune the delay, and the number of PWM changes to achieve this effect.

Now let's say you want to control multiple LEDs at the same time, then GPIOZERO comes with a great inbuilt class called LEDBoard. It even has inbuilt methods to control multiple LEDs brightness using PWM. We will wire up one more LED to the setup. Here we are connecting the positive leg of the Green LED to GPIO 3 through a current limiting resistor. This LED will be sharing the same Ground Pin as the Red LED. Now open the LED Board.py script and run the code. You can see that the following sequence of events occur

1. Both LEDs lights up then

2. Both LEDs are off

3. Later the Red LED is on while Green one is off

4. Now the Green one is on, and the Red LED is off

And this cycle continues.

In the code, you can see that we have imported the sleep and LEDBoard classes from the gpiozero and time modules, respectively. Then we assigned the class LEDBoard to the variable “leds” with the LED GPIO S , 17, and 3 as the arguments of the class. Now inside the infinite loop, we have used the “on()” method and “off()” method to switch on and off all the LEDs. Next comes the interesting part. Here we have set “value” to a tuple, where each element in the tuple are mapped to the LED in the same order as assigned earlier. In this case, as the first element is 1, the first LED, that is, the LED connected to GPIO 17 will be ON, and the other one will be OFF, as the second element is 0.

To control brightness of multiple LEDs using PWM, we need to modify this code like the following. First, we need to enable the PWM option in the class by passing an extra argument called “pwm=True” while creating the object. Now, you are free to put any values from 0 to 1 for each LED, as shown here. Here is how it translates to the LED Output.

Next, let's learn about Buttons. There are many types of buttons, aka switches available. We will work with tactile push buttons as these are the most commonly available and breadboard-friendly buttons. Connecting the Raspberry Pi’s GPIO to a momentary tactile push button switch is a fairly simple circuit.

We connect one side of the button to an input pin on the Raspberry Pi. In this case, we use GPIO 2. On the other side of the switch, we connect to Ground through a resistor. The resistor is used as a current limiting resistor to protect our input pin by limiting the amount of current that can flow.

Now open the Button LED.py script from the repository and run the code. Now you can see the Button is not pressed string, printing continuously on the shell. Now push the button, and you can see the LED light up and the message in the shell changing. The code is self-explanatory. The only thing that stands out is the “is\_pressed” method. The following figure shows the most commonly used button methods and their functions. Please check out the resources section to know more about them.

I will now give you an activity to do. Create a reaction time game using 2 buttons and an LED. Basically, the LED should randomly light up, and whoever presses the button faster wins. It’s a 2-player game, thus use 2 push buttons. I will give you a hint. You will need to use the inbuilt module called random and its method to create random sleep times for the LED.

Summary

In this video, we have covered the following

● PWM in LEDs

● Controlling Multiple LEDs

● Working with Buttons

In the next video, we will learn to interface and work with Light Sensors and Motion Sensors.