piRover Builds with K2

piRover – Pulse Width Modulation (PWM) Rev 1.0

Overview:

In this activity you will create a Pulse Width Modulated (PWM) port on the Pi. A PWM port is an output port on the GPIO header that when started, creates a square waveform as it alternates from low to high.

The instructor will review the concepts of PWM including frequency and duty cycle. You then use GPIO code to configure a port first as an output and then as a PWM output.

You will use this code to create a dimming function for the LEDS. Here the pushbutton switch will control then intensity of the LED module. This will require you to modify the duty cycle of the PWM port as the pushbutton switch is depressed.

The requirements for the solution are listed below.

- The user will see a welcome message indicating that this is the LED Dimming activity.
- The user will be prompted to press the pushbutton switch to increase the intensity of the LED light.
- When the LED light is fully on, a message is displayed.
- As the user continues to press the pushbutton, the light dims.
- When the LED light is fully off, a message is displayed.

Prerequisites:

Prior to beginning the instruction provided in this lesson you must have completed the following:

1. piRover Traffic Light

Performance Outcomes:

1. Configure a GPIO output port as a PWM port.
2. Identify frequency and duty cycle related to PWM.
3. Create analog output by controlling a PWM port's duty cycle.

Resources:

1. Arduino PWM output and its uses

Materials:

1. piRover

piRover Builds with K2

Part 1 – Set Up

1. Prepare your workspace for this activity.

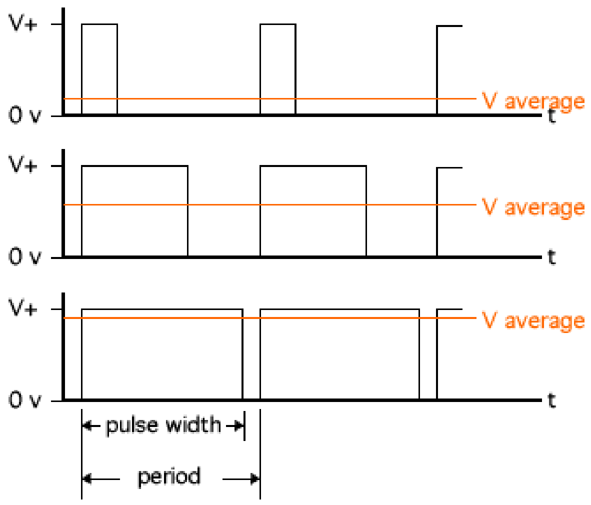
   a. Connect to your piRover using VNC.
   b. Access your piRover folder
   c. Create a **09.PWMIntro** directory
   d. Change to the 09.PWMIntro directory
   e. Download the file below.

```
wget -O pwm_intro.py http://bit.ly/K2-piRover-pwm_intro
```

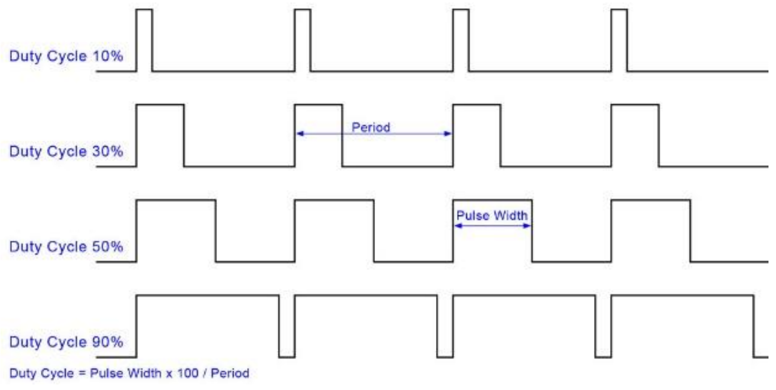
Part 2 – Investigate the PWM

2. Review concepts related to PWM and analog output using the Arduino PWM output and its uses resource. Yes, this discusses Arduino and not Raspberry Pi, but PWM concepts are the same.

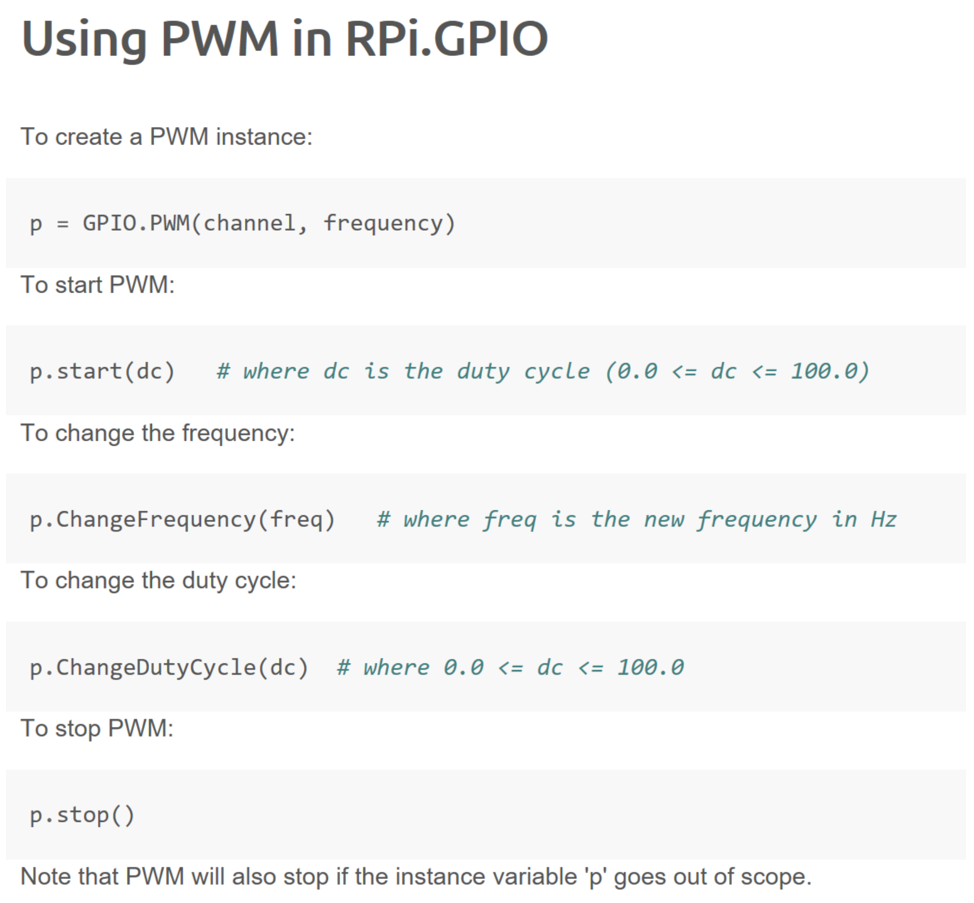
3. Your goal with this solution is to provide analog control (light dimming) using a digital signal. An analog value varies between a minimum and a maximum. The light will vary between off and its maximum intensity. This variable output can be produced by switching the port high and low for varied times. The average value of the voltage over time creates this analog value. See the diagram below.



2. The attribute that is associated with the various waveforms above is Duty Cycle which is the percentage of time the output is high compare to the period of the wave. See sample duty cycles on the following page.



3. The frequency of the waveform is the number of cycles in one second. While this is an additional attribute of PWM, it is normally set during initialization and then is never changed. The duty cycle on the on other hand, is the attribute that is modified to produce the varied output.
4. Review the RPi.GPIO's implementation of a PWM port (see https://sourceforge.net/p/raspberry-gpio-python/wiki/PWM/)



Using PWM in RPi.GPIO

To create a PWM instance:

```
p = GPIO.PWM(channel, frequency)
```

To start PWM:

```
p.start(dc)   # where dc is the duty cycle (0.0 <= dc <= 100.0)
```

To change the frequency:

```
p.ChangeFrequency(freq)   # where freq is the new frequency in Hz
```

To change the duty cycle:

```
p.ChangeDutyCycle(dc)  # where 0.0 <= dc <= 100.0
```

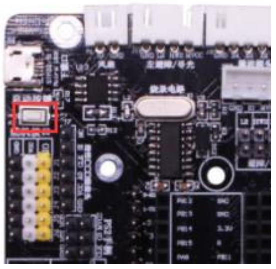
To stop PWM:

```
p.stop()
```

Note that PWM will also stop if the instance variable 'p' goes out of scope.

5. . The pushbutton switch is located in the front left corner of the controller board in front of the header used for servo connetions.

Start button



6. Review the documentation in the [Yahboom Expansion Board Manual]() to determine which GPIO pin is connected to the pushbutton switch on the GPIO board. You will find this on page 26.
7. The start button K2 is connected to pin 24. See Table 1. Review Table 1 to be sure that you understand how this connection detail is determined. You will be expected to locate information on pinouts on your own in future activities.

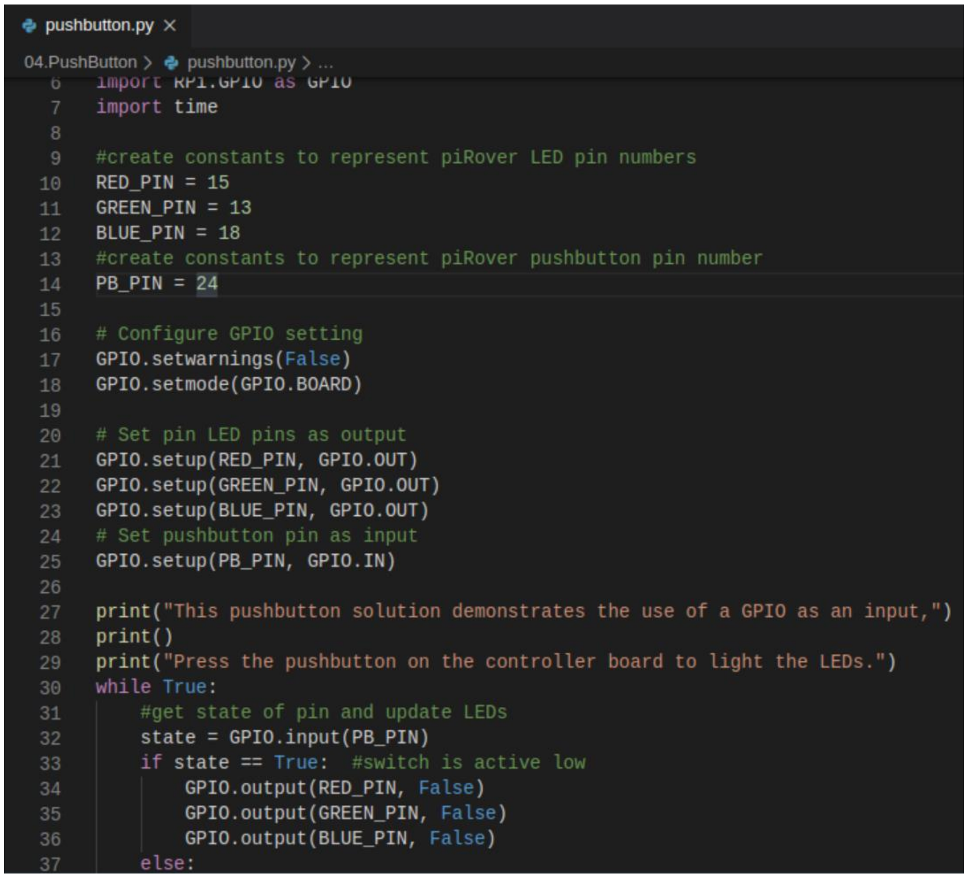
Input	Board Pin	GPIO Reference
Start button	24	GPIO8

Table 1

8. Your goal in this activity is use the GPIO.input() function to read the state of the push button switch and control the LED based on its value. You will need to configure pin 24 as a GPIO input.

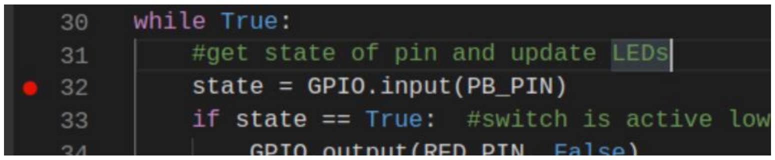
Part 2 – Investigate the software – pushbutton.py

1. Open the pushbutton.py in the VS Code editor.



```
pushbutton.py ×
04.PushButton > pushbutton.py > ...
 6  import RPi.GPIO as GPIO
 7  import time
 8
 9  #create constants to represent piRover LED pin numbers
10  RED_PIN = 15
11  GREEN_PIN = 13
12  BLUE_PIN = 18
13  #create constants to represent piRover pushbutton pin number
14  PB_PIN = 24
15
16  # Configure GPIO setting
17  GPIO.setwarnings(False)
18  GPIO.setmode(GPIO.BOARD)
19
20  # Set pin LED pins as output
21  GPIO.setup(RED_PIN, GPIO.OUT)
22  GPIO.setup(GREEN_PIN, GPIO.OUT)
23  GPIO.setup(BLUE_PIN, GPIO.OUT)
24  # Set pushbutton pin as input
25  GPIO.setup(PB_PIN, GPIO.IN)
26
27  print("This pushbutton solution demonstrates the use of a GPIO as an input,")
28  print()
29  print("Press the pushbutton on the controller board to light the LEDs.")
30  while True:
31      #get state of pin and update LEDs
32      state = GPIO.input(PB_PIN)
33      if state == True:  #switch is active low
34          GPIO.output(RED_PIN, False)
35          GPIO.output(GREEN_PIN, False)
36          GPIO.output(BLUE_PIN, False)
37      else:
```

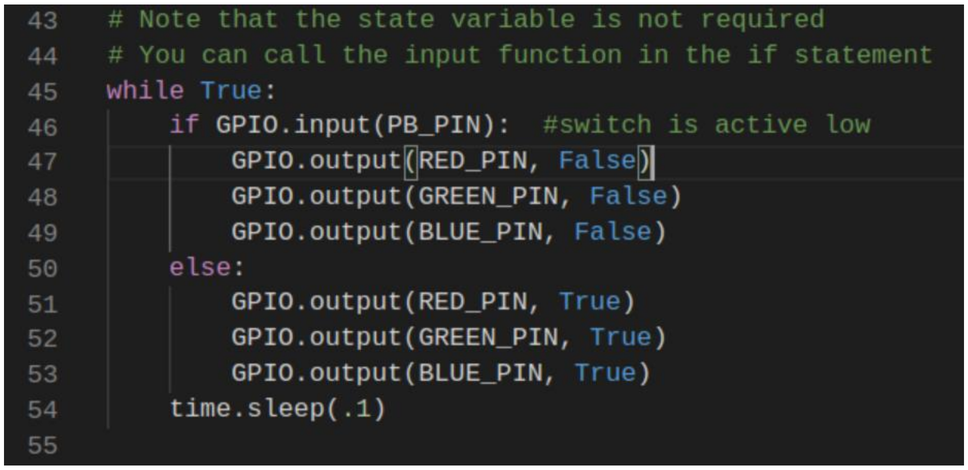
2. Run the pushbutton.py code. Does it function?
3. Note the new pin definition on line 14 and refer to table 1.
4. Inspect line 25. Note that this is the same setup() function used for the LED outputs only the second parameter is changed to GPIO.IN.
5. Inspect line 32. Is this similar to the input() function used in user_blink.py? What is the data type for the state variable?
6. Inspect lines 33 through 40. What logic level is read by the GPIO.input() function when the switch is depressed?
7. Insert a breakpoint on line 32.



```
30  while True:
31      #get state of pin and update LEDs
32      state = GPIO.input(PB_PIN)
33      if state == True:  #switch is active low
34          GPIO.output(RED_PIN, False)
```

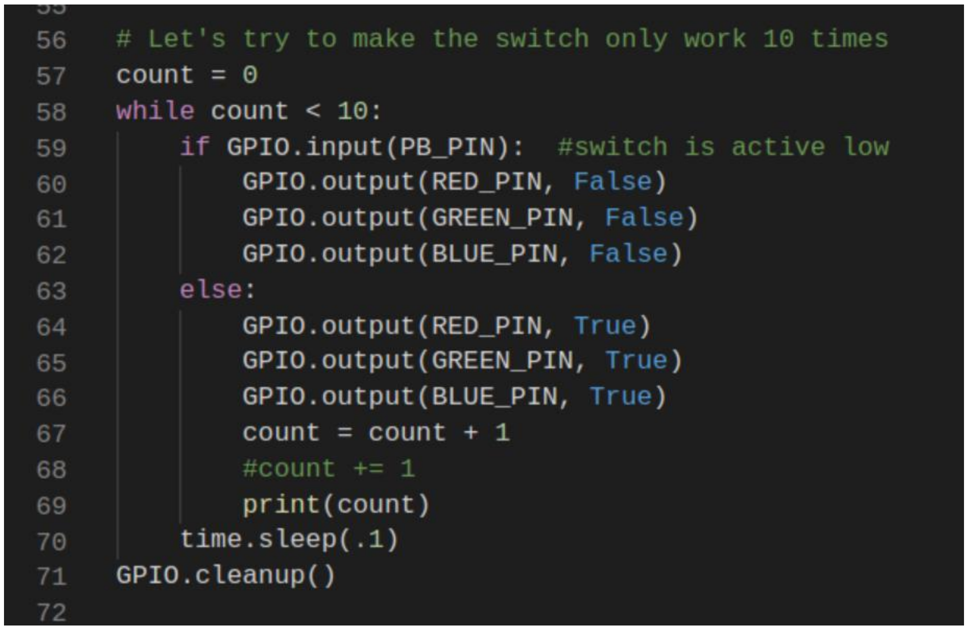
8. Run the debugger (F5) and verify the value and data type of the state variable as you single step (F10) the program and press the pushbutton switch.

9. Follow along with the instructor as he or she swaps out the code on lines 45 through 54. Do you understand why the switch variable can be eliminated from this revised code?



```
43  # Note that the state variable is not required
44  # You can call the input function in the if statement
45  while True:
46      if GPIO.input(PB_PIN):  #switch is active low
47          GPIO.output(RED_PIN, False)
48          GPIO.output(GREEN_PIN, False)
49          GPIO.output(BLUE_PIN, False)
50      else:
51          GPIO.output(RED_PIN, True)
52          GPIO.output(GREEN_PIN, True)
53          GPIO.output(BLUE_PIN, True)
54      time.sleep(.1)
55
```

10. Follow along with the instructor as she or he swaps out the code for lined 57 through 71. Run the code. Does it function as expected? Explain the behavior.

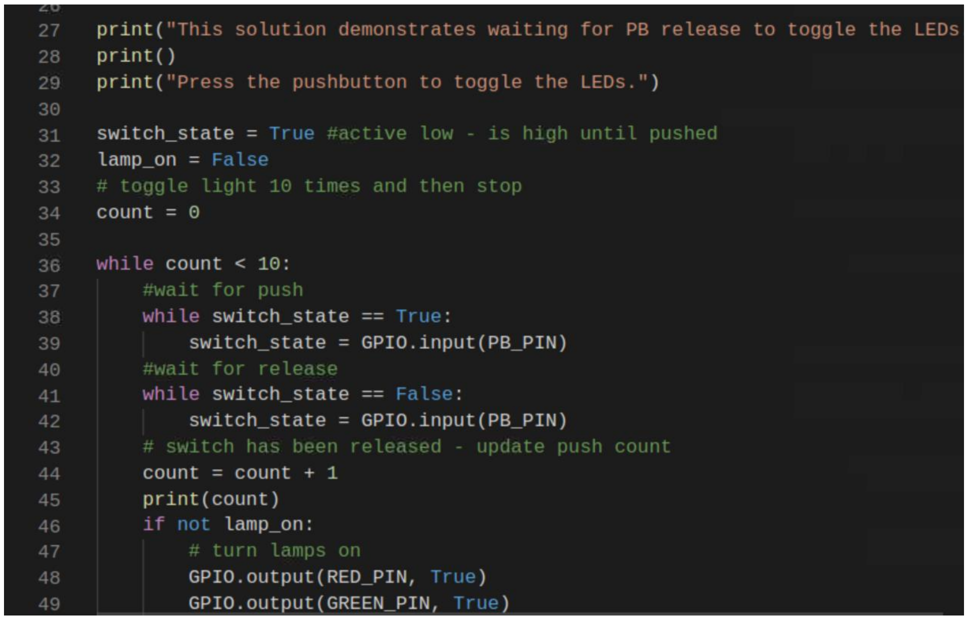


```
56  # Let's try to make the switch only work 10 times
57  count = 0
58  while count < 10:
59      if GPIO.input(PB_PIN):  #switch is active low
60          GPIO.output(RED_PIN, False)
61          GPIO.output(GREEN_PIN, False)
62          GPIO.output(BLUE_PIN, False)
63      else:
64          GPIO.output(RED_PIN, True)
65          GPIO.output(GREEN_PIN, True)
66          GPIO.output(BLUE_PIN, True)
67          count = count + 1
68          #count += 1
69          print(count)
70      time.sleep(.1)
71  GPIO.cleanup()
72
```

11. The instructor will describe the purpose of the GPIO.cleanup() function on line 71. What happens if you make an error and indent line 71 so that it is part of the while loop?

Part 3 – Investigate the software – pushbutton_toggle.py

12. Open the pushbutton_toggle.py file. Run the code to understand the function. How does this code differ from the prior pushbutton code as it cycles 10 times?



```
27  print("This solution demonstrates waiting for PB release to toggle the LEDs
28  print()
29  print("Press the pushbutton to toggle the LEDs.")
30
31  switch_state = True #active low - is high until pushed
32  lamp_on = False
33  # toggle light 10 times and then stop
34  count = 0
35
36  while count < 10:
37      #wait for push
38      while switch_state == True:
39          switch_state = GPIO.input(PB_PIN)
40      #wait for release
41      while switch_state == False:
42          switch_state = GPIO.input(PB_PIN)
43      # switch has been released - update push count
44      count = count + 1
45      print(count)
46      if not lamp_on:
47          # turn lamps on
48          GPIO.output(RED_PIN, True)
49          GPIO.output(GREEN_PIN, True)
```

13. In general, you should be very cautious about using while loops in your code. They can produce infinite loops that lock up the execution of your code.
14. Review the while loops on lines 38 through 42. Can you describe the function of this code? Set a breakpoint at line 38 and test.
15. Use your mouse to click on a button or item on any menu in your system How does this code duplicate the function that you experience with your computer mouse?

Part 4 – Investigate the software – pushbutton_cycle.py

1. Open the pushbutton_cycle.py file. Run the code to understand the function.
2. Note the use of the f-string in the print() function on line 31. Use the debugger to determine the data type of the state variable on line 28. The f-string format allows this non-string data to be easily combined to create a string message.
3. Read the comment on line 33. Which lines of code constrain the value of the I variable to 0, 1, 2, and 3?
4. Modify the code so that the counter I goes from 0 to 5. Add an else condition at the end of the selection structure (if) staring on line 50 so that white light is on for counts 4 and 5.
5. Comment out the code starting at line 33 "### i controls LED cycle". Replace the code by uncommenting the block of code starting around line 75.

6. Run the revised code. How does this code differ from the prior toggle code?
7. The wait for push and release code is at the bottom of this loop. Does this change the behavior of the code?
8. Examine the push and release code in this loop and compare it to the earlier code. What changes have been made? What does the "pass" statement due? Can this line just be eliminated? Try commenting the "pass" statement out and run the code. What error occurs and why?

9. Once you completed the investigating the three solutions, copy your 04.PushButton folder up to your cloud service as a backup (OneDrive, Google Drive, or DropBox)

Assessment:

Download the PushButtonActivity.docx file from the Moodle section. Complete this document by entering your responses in the spaces provided. Submit your completed activity document as either a .docx or a .pdf file to Moodle along with any other files in this week's zip file.