piRover Builds with K2

piRover – Pulse Width Modulation (PWM) Rev 1.3

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 the 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

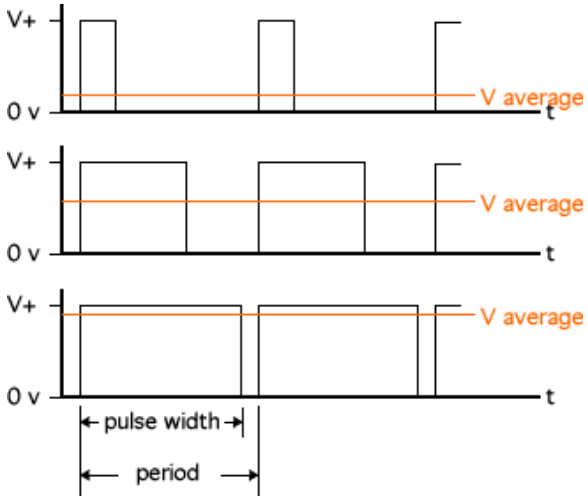
1. Prepare your workspace for this activity.
    a. Prepare your workspace for this activity. Connect to your piRover using VNC and open VS Code
    b. Create a directory for this week's work if this was not in the prior session
    c. Using the VS Code terminal window, change to this week's directory.
    d. Download the starter files for the activity using the wget instructions below.

```
wget https://k2controls.github.io/piRover01/lessons/30/pwm_intro.py
```

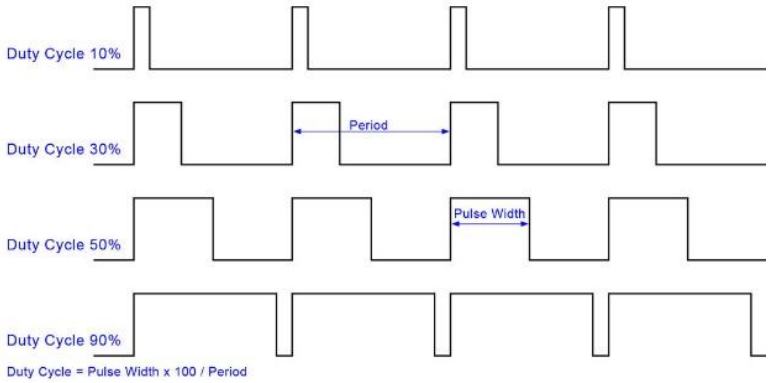
Part 2 – Investigate the PWM Documentation

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 compared 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.
4. The duty cycle on the other hand, is the attribute that is modified to produce the varied output. This varied output can be averaged over time to create an analog output voltage.
5. 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.

Part 3 – Investigate the PWM Code

6. Open the pwm_intro.py file in VS Code. The starter solution is shown below. Review the code and calls to the PWM object. Note that the pin must be defined as an output first.

```
RED_PIN = 15
GREEN_PIN = 13
BLUE_PIN = 18

# Configure GPIO setting
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BOARD)

# Set LED as output
GPIO.setup(RED_PIN, GPIO.OUT, initial=False)
GPIO.setup(GREEN_PIN, GPIO.OUT, initial=False)
GPIO.setup(BLUE_PIN, GPIO.OUT, initial=False)

### Example 1 ###
# Set Red LED as PWM to blink
# Adjust both frequency and duty cycle to investigate
freq = 1
dc = 50
pwm_red = GPIO.PWM(RED_PIN, freq)
pwm_red.start(dc)
while(True):
    pass
```

7. Run the code and recognize that once the while(True) loop runs, nothing is running in the foreground code. Adjust both the frequency and the duty cycle and rerun the code to see the effects of these settings. Compare the visual LED display to the waveforms shown in the PWM documentation.

8. Comment out the ### Example ### code and enter the following Example 2 code below.

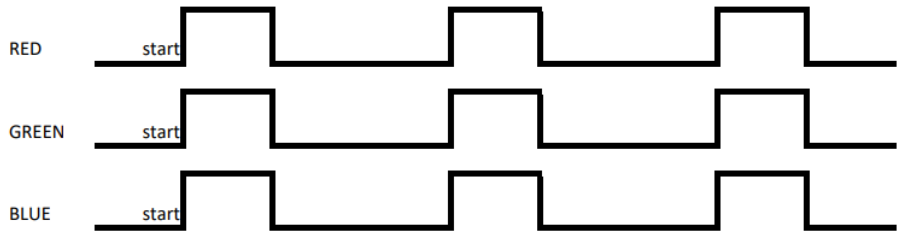
```
## Example 2 ###
Use buzzer as alternative
Adjust both frequency and duty cycle to investigate
PB pin is buzzer pin - Be careful!
PB_PIN = 24
BUZZER_PIN = 24
freq = 1
dc = 50
GPIO.setup(BUZZER_PIN, GPIO.OUT, initial=True)
pwm_buzzer = GPIO.PWM(BUZZER_PIN, freq)
pwm_buzzer.start(dc)
while(True):
    pass
```

9. Run the code. Adjust both the frequency and the duty cycle and rerun the code to hear effects of these settings. Compare the results to the waveforms shown in the PWM documentation.

10. Comment out the Example 2 code. Copy the Example 1 code below Example 2 and rename it to Example 3. Modify the code to include all three LED lamps as shown below.

```
### Example 3 ###
# Set Red, Green, Blue LEDs as PWM
# Blink white. Then blink red,green,blue
freq = .33      # once every 3 seconds
dc = 33         # one second pulse every 3 seconds
pwm_red = GPIO.PWM(RED_PIN, freq)
pwm_green = GPIO.PWM(GREEN_PIN, freq)
pwm_blue = GPIO.PWM(BLUE_PIN, freq)
pwm_red.start(dc)
pwm_green.start(dc)
pwm_blue.start(dc)
while(True):
    pass
```

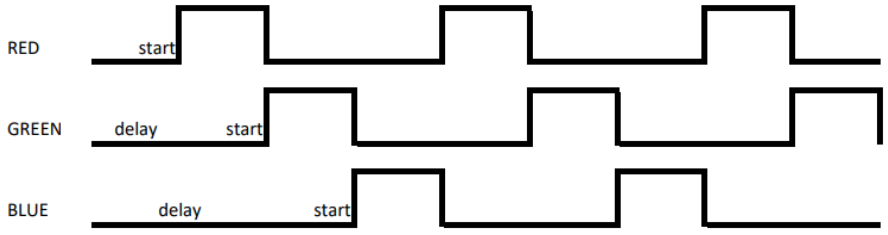
11. The RGB PWM output is represented by the waveforms shown below.



12. Modify the code to include delays prior to starting the Green and the Blue channels.

```
### Example 3 ###
# Set Red, Green, Blue LEDs as PWM
# Blink white. Then blink red,green,blue
freq = .33      # once every 3 seconds
dc = 33         # one second pulse every 3 seconds
pwm_red = GPIO.PWM(RED_PIN, freq)
pwm_green = GPIO.PWM(GREEN_PIN, freq)
pwm_blue = GPIO.PWM(BLUE_PIN, freq)
pwm_red.start(dc)
time.sleep(1) # delay start of green (1 sec)
pwm_green.start(dc)
time.sleep(1) # delay start of blue (2 secs)
pwm_blue.start(dc)
while(True):
    pass
```

13. The revised RGB PWM output is represented by the waveforms shown below.



14. Comment out the Example 3 code. Copy the Example 3 code below and rename it to Example 4. Modify to include the "for loops" to modify the frequency of the pulses. Run the code and note the impact of modifying frequency. Note: Normally frequency is NOT changed.

```
### Example 4 ###
# Use for loop to vary the frequency of the pulse
freq_start = 1
freq_end = 50
dc = 50
pwm_red = GPIO.PWM(RED_PIN, freq_start)
pwm_green = GPIO.PWM(GREEN_PIN, freq_start)
pwm_blue = GPIO.PWM(BLUE_PIN, freq_start)
pwm_red.start(dc)
pwm_green.start(dc)
pwm_blue.start(dc)
print("Lamp is blink at 1 Hz - wait for 10 sec")
time.sleep(10)
print("Now vary the frequency up")
for i in range(freq_start, freq_end):
    pwm_red.ChangeFrequency(i)
    pwm_green.ChangeFrequency(i)
    pwm_blue.ChangeFrequency(i)
    time.sleep(.5)
print("Now vary the frequency down")
for i in range(freq_end, freq_start, -1):
    pwm_red.ChangeFrequency(i)
    pwm_green.ChangeFrequency(i)
    pwm_blue.ChangeFrequency(i)
    time.sleep(.5)
```

the code to include all three LED lamps as shown below.

15. Comment out the Example 4 code. Copy the Example 4 code below and rename it to Example 5. Modify the "for loops" to modify the duty cycle of the pulses rather than the frequency. Run the code and note the impact of modifying duty cycle. Note: Normally it is the duty cycle that is modified when using PWM outputs.

```
### Example 5 ###
# Use for loop to vary the duty cycle of the pulse
freq = 50
dc = 0
pwm_red = GPIO.PWM(RED_PIN, freq)
pwm_green = GPIO.PWM(GREEN_PIN, freq)
pwm_blue = GPIO.PWM(BLUE_PIN, freq)
pwm_red.start(dc)
pwm_green.start(dc)
pwm_blue.start(dc)
print("Pulse is on but dc is zero to lamp is off")
print("Increase Lamp intensity using dc")
for i in range(0, 100, 5):
    pwm_red.ChangeDutyCycle(i)
    pwm_green.ChangeDutyCycle(i)
    pwm_blue.ChangeDutyCycle(i)
    time.sleep(.5)
print("Lamp is on full")
print("Now dim")
for i in range(100, 0, -5):
    pwm_red.ChangeDutyCycle(i)
    pwm_green.ChangeDutyCycle(i)
    pwm_blue.ChangeDutyCycle(i)
    time.sleep(.5)
```

16. Finally, complete example 6 to configure the push button as an input and use this input to control the duty cycle. The code listing is shown on the following page/

```
### Example 6 ###
# set up Push Button to control PWM
PB_PIN = 24
# Set pushbutton pin as input
GPIO.setup(PB_PIN, GPIO.IN)

freq = 50
dc = 0
pwm_red = GPIO.PWM(RED_PIN, freq)
pwm_green = GPIO.PWM(GREEN_PIN, freq)
pwm_blue = GPIO.PWM(BLUE_PIN, freq)
pwm_red.start(dc)
pwm_green.start(dc)
pwm_blue.start(dc)

state = False
while True:
    state = GPIO.input(PB_PIN)
    if not state: #button is pushed
        if dc < 100:
            dc = dc + 5
            print(f"Duty cycle = {dc}")
            pwm_red.ChangeDutyCycle(dc)
            pwm_green.ChangeDutyCycle(dc)
            pwm_blue.ChangeDutyCycle(dc)
    time.sleep(.5)
```

Part 4 – Investigate the Variable Scope and the Global keyword

17. Follow the instructor as he or she converts PWM code into user functions. See the code listing on the following page. The PWM variable must be global to enable each function to set and adjust.

```
#create constants for LED GPIO pin
RED_PIN = 15
GREEN_PIN = 13
BLUE_PIN = 18

# pwm_red = None
# pwm_green = None
# pwm_blue = None

def init():
    # global pwm_red, pwm_green, pwm_blue
    freq = 1
    dc = 0      # start as "off"

    # Configure GPIO setting
    GPIO.setwarnings(False)
    GPIO.setmode(GPIO.BOARD)

    # Set LED as output
    GPIO.setup(RED_PIN, GPIO.OUT, initial=False)
    GPIO.setup(GREEN_PIN, GPIO.OUT, initial=False)
    GPIO.setup(BLUE_PIN, GPIO.OUT, initial=False)

    pwm_red = GPIO.PWM(RED_PIN, freq)
    pwm_green = GPIO.PWM(GREEN_PIN, freq)
    pwm_blue = GPIO.PWM(BLUE_PIN, freq)

    pwm_red.start(dc)
    pwm_green.start(dc)
    pwm_blue.start(dc)

def blink(frequency, duty_cycle):
    pwm_red.ChangeFrequency(frequency)
    pwm_green.ChangeFrequency(frequency)
    pwm_blue.ChangeFrequency(frequency)

    pwm_red.ChangeDutyCycle(duty_cycle)
    pwm_green.ChangeDutyCycle(duty_cycle)
    pwm_blue.ChangeDutyCycle(duty_cycle)

# main code is here
init()
blink(1, 50)
while(True):
    pass
```

Assessment:

Submit the pwm_intro.py and pwm_in_functions.py along with other files in this week's zip file.