Lab Sheet: 3

Objective: LED dimmer; controlling Servo and DC motors - using PWM.

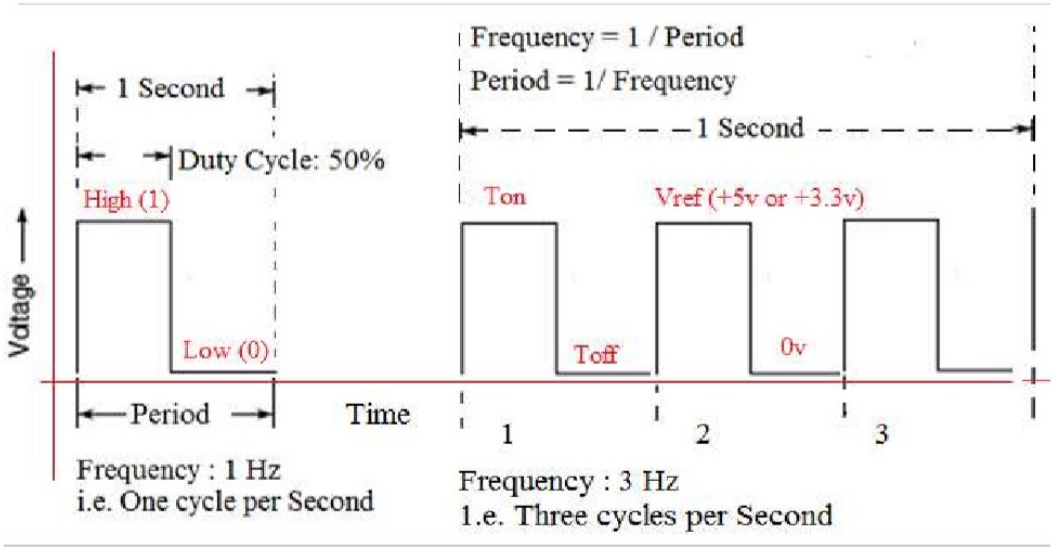
Hardware Requirements: Raspberry Pi; SD card at least 4GB, Monitor with HDMI (High definition multimedia interface) connector, Micro USB power supply, USB keyboard, USB mouse, breadboard, switches, servo motor, DC motor, LM293D motor driver, resistors and jumper wires.

Here we discuss what is PWM? and how it is working? then we configure GPIO pins to generate required PWM signals. Using this technique, one may control the brightness of an LED or else may control the speed of a motor. Raspberry Pi pins gives either +3.3V (when turned High) or 0v (when turned LOW), and the output become a square wave. If you want to dim an LED, you need to change the voltage; ofcourse, you cannot supply a voltage between +3.3 and 0V through GPIO pins. But, you may change the ON and OFF time of the LED (may change the duty cycle), so that the brightness can be controlled.

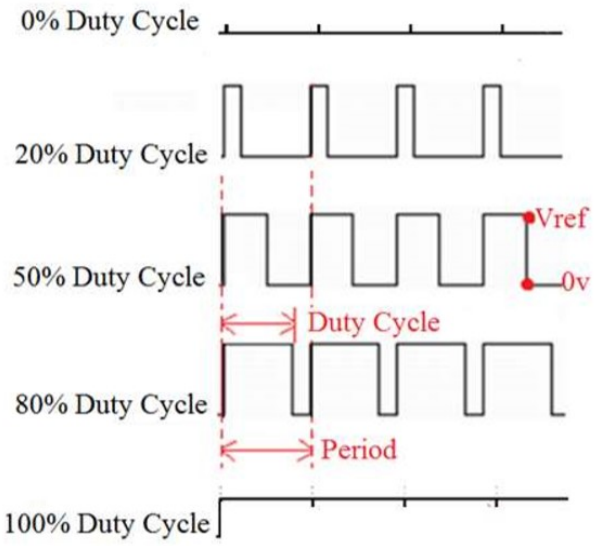
Pulse Width Modulation: PWM stands for pulse width modulation; Analog voltages or current can control Analog devices (such as DC-Motor, Stepper Motor, etc.) directly. For example, in a simple Analog radio, the volume knob is connected to a variable resistor; when you turn the knob, the resistance goes up or down and the current flowing through the resistor increases or decreases. This changes the amount of current driving the speakers, thus increasing or decreasing the volume. If you control Analog circuits digitally, system costs and power consumption can be reduced.

Pulse Width Modulation (PWM), is a technique that modulates the width of a rectangular wave. It generates a wave of varying duty cycle called PWM signal, which is used to control various analog devices through digital pins of a processor (i.e. this is a method of digital control for analog devices). Where a direct digital signal is used all the way to

control analog devices without any digital to analog signal conversion (DAC).



Duty Cycle: There are two important parameters that affect the behaviour of a PWM signal; the first one is the frequency of the pulse and the second is the width of the pulse. The PWM signal oscillates according to given frequency and duty cycle. The high portion of the signal is called on-time (Ton), and the low portion is called off-time (Toff). Duty cycle is a percentage of time that the signal is HIGH over a period of time that determines how much energy to be delivered to an analog device. If a signal is half of the time High (ON) and half of the time Low (OFF), then we say that the signal has a duty cycle of 50% (become a square wave). The below figure shows signals with different duty cycles. Square wave forms are symmetrical their duty cycle is half of its period (50%). If the duty cycle of a waveform is greater or less than 50% then the resulting waveform is a rectangular waveform which is considered as non-symmetrical.



Period and Frequency: Period (sometimes called clock rate) is a length of one complete cycle (Ton+Toff) or time taken by one cycle. Frequency refers to the number of cycles occurs in a second, which is measured in hertz (Hz); The rate of different waves, such as sound waves, radio waves, and light waves, etc. are measured in Hertz. Audible sound waves may have a frequency of roughly 20 Hz to 20,000 (20 KHz). Therefore, the sound we hear is a cycle of waves which can occur 20 to 20,000 times per second. A Rectangular waveform has a duty cycle of 25% which takes 10 ms, Find the frequency.

(25/peroid)*100=10 ms; 25/(100*Period)=10; 1/(4*peroid) =10 then period=40 ms (100%); F =1/period; F = 1/40 ms =0.025 ms

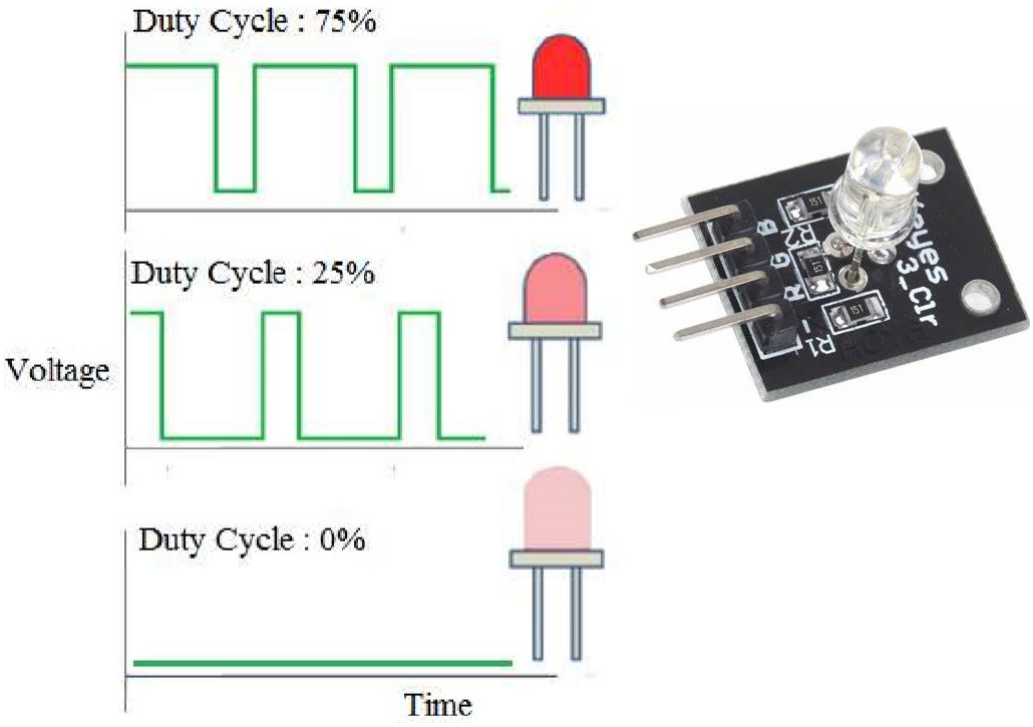
- 1ms = 1 KHz (Thousand)
- 1us = 1 MHz (Million)
- 1ns = 1 GHz (Billion)
- 1ps = 1 THz (Trillion)

Hence, 0.025 ms = 0.025x1000 = 25 Hz

In Raspberry Pi, you may notice that some of the GPIO pins have other letters in parentheses after their names. Those pins have some special purpose. For example, GPIO0 and GPIO1 are labelled SDA and SCL These are the clock and data lines, respectively, for a serial bus type called I2C that is popular for communicating with peripherals such as temperature sensors, LCD displays. Yet another type of serial communication is also possible through GPIO9 to GPIO11 (MISO, MOSI, and SCLK). This type of serial interface is called SPI. Finally, GPIO18 and GPIO21 are labelled PWM, those pins can be used for pulse width modulation (PWM). Rasberry Pi has two PWM channels PWM0 and PWM1; GPIO12 and GPIO18 are used as PWM0 channel; GPIO13 and GPIO19 are used as PWM1 channel;

1. LED-Dimmer: Design an LED-Dimmer that fade in and out. Brightness or intensity of the LED should be controlled in a sequential manner so that it can glow from bright to dim and in reverse. The brightness of an LED is directly proportional to the current flowing through it, but it would be rather difficult to accurately control the current flow. The best method is to use PWM that vary the time, of which current or voltage is supplied to the LED by adjusting its duty cycle. With 0% duty cycle, LED is switched off (receives no voltage); with 100% duty cycle it is turned on (receives all voltage).This means that if we vary the duty cycle 0% to 100% back and forth the LED would fade in and out.

If the frequency of the signal is fast enough, then there will be no visible flicker, and the brightness will be proportional to the duty cycle. According to the persistence of vision by turning an LED on and off rapidly, we can trick the brain into seeing an average value of brightness.



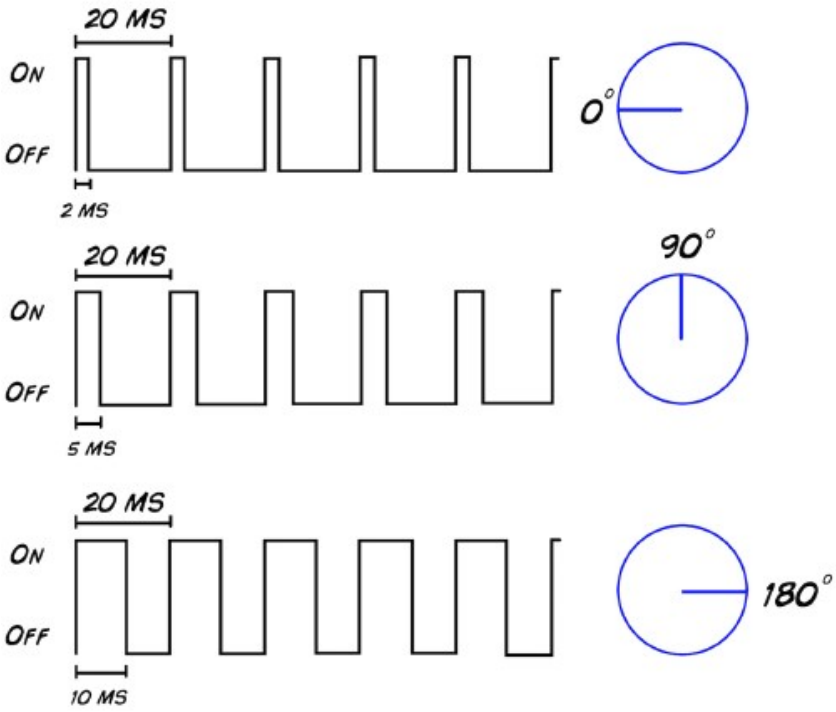

```
#Controlling an LED with a switch
import RPi.GPIO as GPIO
import time
ledpin=12 #PWM pin is connected to LED
GPIO.setwarnings(False)                    #disable warnings
GPIO.setmode(GPIO.BOARD);                  #  follow physical pin numbering
GPIO.setup(ledpin, GPIO.OUT)
dimmer=GPIO.PWM(ledpin, 50)                # pin number,  frequency
#frequency of the PWM signal to be generated =50Hz
dimmer.start(0)
#start PWM signal generation with 0% duty cycle
```

```
Try:
        while (True):
            for duty in range (0, 100, 5) :  # Increase duty cycle  from 0 to 100
                dimmer.ChangeDutyCycle(i)        # change dutycycle
                time.sleep(0.2)

            for duty in range (100, 0, -5) : # decrease duty cycle  from 100 to 0
                dimmer.ChangeDutyCycle(i)     # change dutycycle
                time.sleep(0.2)

except KeyboardInterrupt:                     # exception handling
        dimmer.stop()
        GPIO.cleanup()                        # resets GPIO pins
```

We created an object dimmer of the class PWM which belongs to RPi.GPIO library; pin number specifies the pin through which required PWM signal to be generated; frequency specifies the frequency of PWM signal. Hence we are generating a software PWM signal, any GPIO pin can be used for connecting the device to be controlled. ChangeFrequency() function may be used to change the frequency; this function has not been used in the above program; but we may use it for changing frequency; using stop() function, you may stop PWM signal generation.

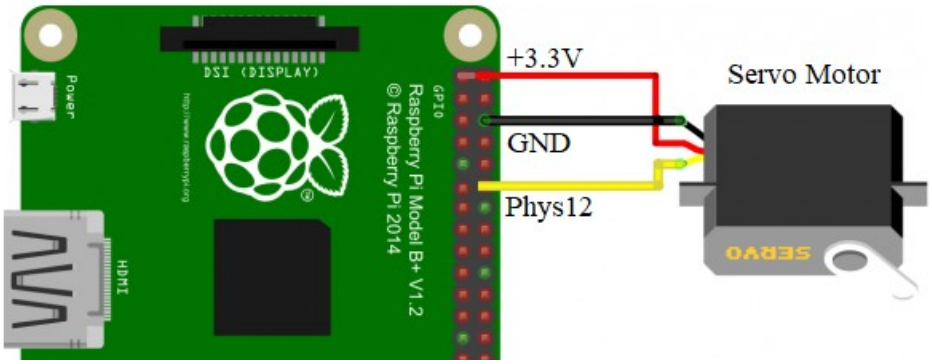
Servo Motor: Motor converts electrical energy into mechanical energy; we have a small (in size and weight) and cheap motors called servo motors; there are two types of servo: AC and DC. AC servos can handle higher current surges and used in industrial machinery. DC servos are better suited for smaller applications. Servos are used in robots, operating grippers, etc**.** Generally, it requires a DC a supply of 4.8V to 6 V; it may usually turn or run 90 degrees in either direction for a total of 180 degrees (servo motors rotates 180 degrees only).

A servo motor can be controlled by sending a pulse of variable or modulated width (using PWM technique) through its control wire. There is a minimum pulse, a maximum pulse, and a repetition rate; width of the pulse applied to the motor is varied and send for a fixed amount of time; that the pulse width determines the angular position of the servo motor. i.e. the PWM signal sent to the motor determines position of the shaft, and based on the duration of the pulse sent via the control wire; the motor will turn to the desired position.



The servo motor expects to see a pulse every 20 milliseconds (ms) and the length of the pulse will determine how far the motor turns. For example, a 5ms pulse will make the motor turn to the 90-degree position. Shorter than 2ms moves it to 0 degrees, and any longer than 5ms will turn the servo to 180 degrees, as diagrammed above. The maximum amount of force the servo can exert is called the torque rating of the servo. Servos will not hold their position forever, though the position pulse must be repeated to instruct the servo to stay in position.

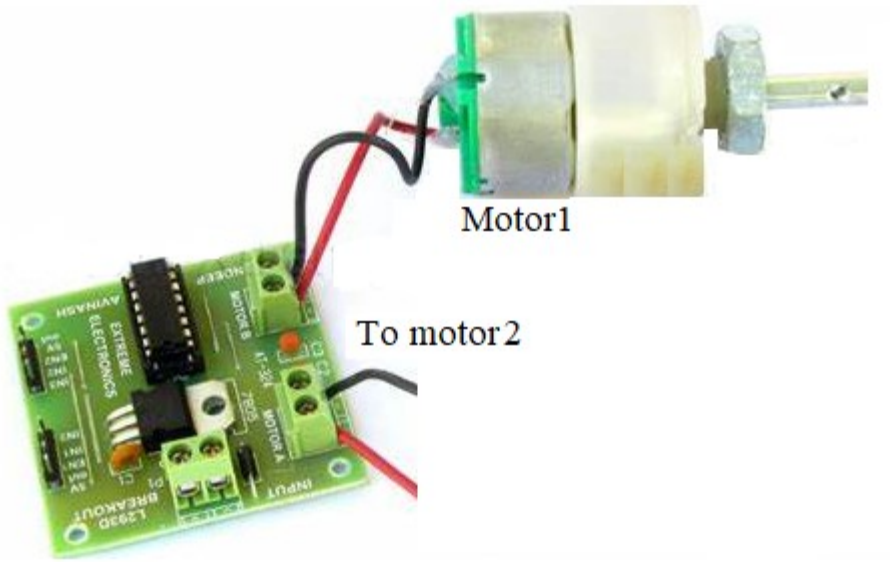
2. Generate a PWM signal with a frequency of 50Hz through the pin GPIO18 (Phys12) and drive your servo motor; that the motor expects a pulse in every 20ms (period), that means 50 pulses per second or Hertz. Connect the black wire of the motor to the GND pin of Pi; red wire to the 3.3V; and the yellow or orange wire to the GPIO pin. If you want to play it safe, you may have a 1kΩ resistor between the data pin (yellow/orange) and the GPIO pin.

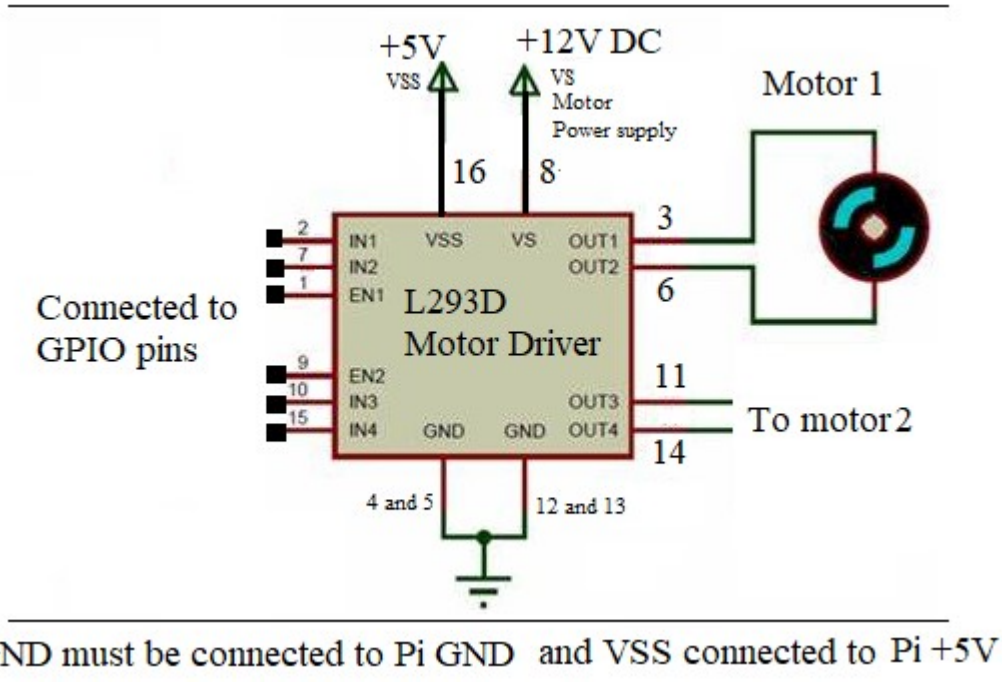



```
#Interfacing Servo motor with Pi
import RPi.GPIO as GPIO
import time
motor=12                        # pin to be connected to motor: GPIO18
GPIO.setwarnings(False)         # disable warnings
GPIO.setmode(GPIO.BOARD);       # follow physical pin numbering
```

```
GPIO.setup(motor, GPIO.OUT)
servo=GPIO.PWM(motor, 50)
 # frequency of  the PWM signal to be generated : 50Hz
servo.start(0) #start PWM signal generation with 0% duty cycle

Try:
        while (True):
                for duty in range (0, 100, 5) :
                    servo.ChangeDutyCycle(i) # changing dutycycle
                    time.sleep(0.2)

except KeyboardInterrupt:
    print('Motor stopped')
    motor.stop()
    GPIO.cleanup() # resets GPIO pins
```

Interfacing DC Motor using LM293D: DC motors runs on direct current; small DC motors are used in toys, tools and appliances; microcontrollers or microprocessors alone cannot drive DC motors directly; so we need an additional mechanism to control the speed and direction of the motors. Such an additional device is called driver (or controller), which acts as an interface between the processors and the DC motors. For example, L293D chip, the motor driver is designed to control two motors at once; which is provided with two sets of, direction control inputs (InpA1, InpA2, InpB1, and InpB2), outputs (OutA1, OutA2, OutB1, OutB2) and enable inputs (EN1 and EN2). Unlike servo motors, DC motors rotates in 360 degrees.





GND must be connected to Pi GND and VSS connected to Pi +5V

3. Control a DC motor with a switch. If the switch is pressed (ON), then rotate the motor in clockwise direction for 5 seconds and in anticlockwise for another five seconds; otherwise turn the motor OFF. Note that here we use LM293 motor driver (controller) to interface the DC motor with our Raspberry Pi. Configure two GPIO pins as output pins to be connected with two control inputs (Inp1 and Inp2)

of the motor driver; and configure one more pin as input pin to be connected with a switch. As a beginner, connect the E1 (Enable1) input of LM293 to +5v pin of Pi, so that the motor may run in full speed. If you connect the EN1 to any GPIO pin through which the PWM signal is to be generated, then speed of the motor can be controlled.

In1	In2	Direction
0	0	No rotation
1	0	Forward
0	1	Backward

```
#Controlling a DC  motor with a switch
import RPi.GPIO as GPIO
import time
in1=40 # pin to be connected with In1 of Motor
in2=38 # pin to be connected with In2 of Motor
switch=36 # on/off  switch
GPIO.setwarnings(False)  #disable warnings
GPIO.setmode(GPIO.BOARD); # Physical pin numbering
GPIO.setup(in1, GPIO.OUT)
GPIO.setup(in2, GPIO.OUT)
GPIO.setup(switch, GPIO.IN)
```

```
while (True):
        status= GPIO.input(switch)
        if (switch) :                          #if switch is ON then rotate
                # Rotate motor in clockwise
                GPIO.output(In1, True)
                GPIO.output(In1, False)
                time.sleep(5)                  #after a delay of 5 seconds
                # Rotate motor in anticlockwise
                GPIO.output(In1, False)
                GPIO.output(In1, True)
        else:                                  #else turn the motor OFF
                GPIO.output(In1, False)
                GPIO.output(In1, False)
GPIO.cleanup()
```

Assignment 1: Run a DC motor at 2 different speeds; generate a PWM signal with 90% duty cycle when the switch is pressed; otherwise generate a signal with 30% duty cycle.

Assignment 2: Write a Python code that detects a keypress and use it to control the motors associated with a Robot.