**Practical No. 01**

**Aim:-** Displaying Blinking LED Patterns With Raspberry PI.

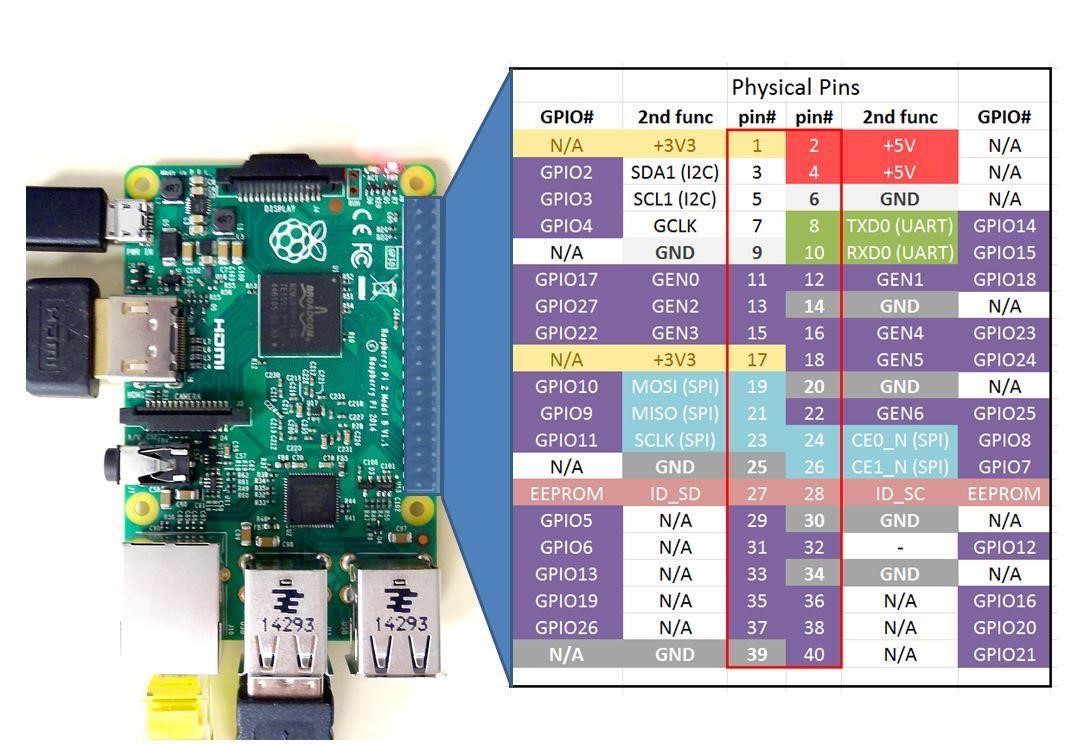
# Hardware Guide:-

Along with the basic setup you will require the following components to get started with the GPIO pins as follows:

LED Module

Connecting wires (F-F)

Before learning this lesson, you must understand the pin numbering system of the GPIO pins



# Connection:-

1. Connect the GPIO22 (i.e. Physical Pin 15) Pin od raspberry pi to the LED Modole Pin.
2. Connect another end of resistor to the positive end (anode) of LED Pin 4 3. Connect the negative end (cathode) of LED to Ground of raspberry pi Pin 6.

4. Then Power on your raspberry pi.

**Software Guide:**

# Code in Python 3:-

#Blink LED Program

#Connect the LED to GPIO 22 Pin #LED Blink Progarm

#Connect the LED to GPIO22 (i.e. Physical Pin15)

#import GPIO and time library import RPi.GPIO as GPIO from time import sleep

GPIO.setmode(GPIO.BCM) #set the Pin mode you will be working with ledPin = 22 #this is GPIO22 pin i.e

Physical Pin15 #setup the ledPin(i.e. GPIO22) as output

GPIO.setup(ledPin, GPIO.OUT)

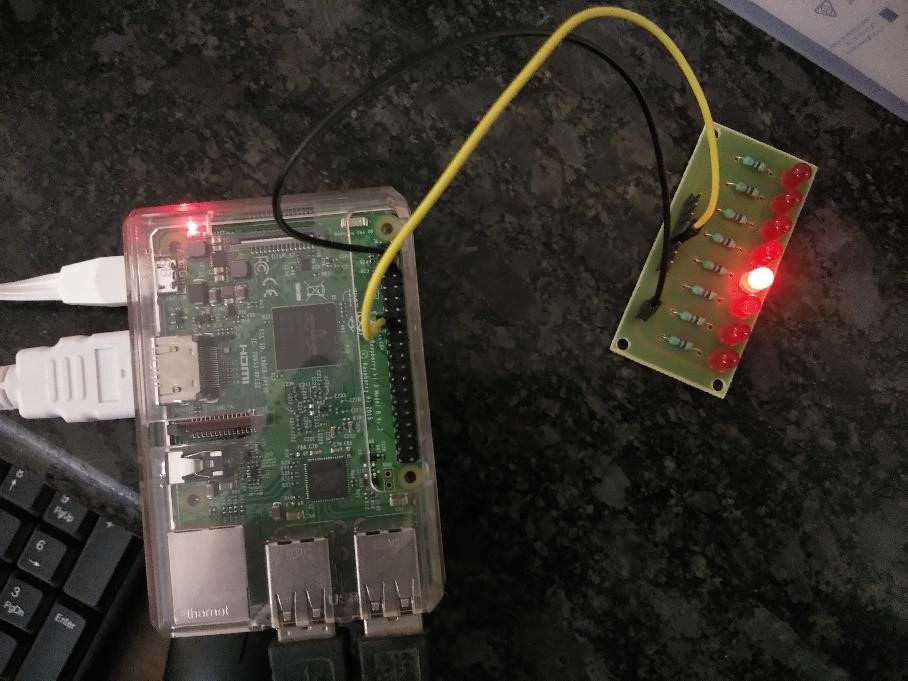
GPIO.output(ledPin, False)

try: while True:

GPIO.output(ledPin, True) #Set the LED Pin to HIGH print("LED ON") sleep(1) #Wait for 1 sec GPIO.output(ledPin, False) #Set the LED Pin to LOW print("LED OFF") sleep(1) #wait for 1 sec finally:

#reset the GPIO Pins GPIO.output(ledPin, False) GPIO.cleanup() #end of code.

## Output:-





**Practical No. 02**

**Aim :-** Using 7 Segment Display to Display Time.

## Hardware Guide:-

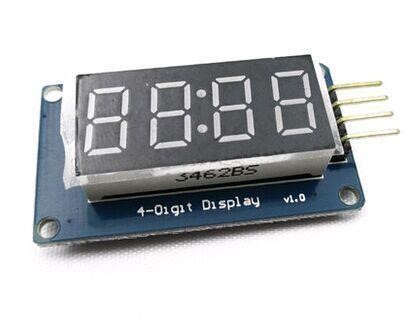
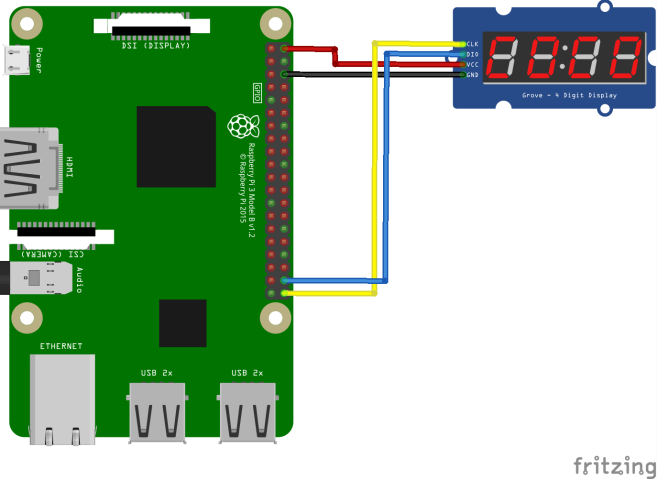
Along with the basic setup you will require the following components to get started with the GPIO pins as follows:

7 Segment Display

Connecting wires (F-F)

Before learning this lesson, you must understand the pin numbering system of the GPIO pins

**Connection:-**



1. Connect the Pin2 (5V) of Rpi to Vcc pin of Module
2. Connect Pin 6 (GND) of Rpi to GND of Module
3. Connect Pin38 (GPIO20) of Rpi to DIO of Module
4. Lastly connect Pin 40 (GPIO21) of Rpi to CLK of Module

**Software Guide:**

# Code in Python 3:- Code for tm1637

import math import RPi.GPIO as IO import threading

from time import sleep, localtime

# from tqdm import tqdm

# IO.setwarnings(False)

IO.setmode(IO.BCM)

HexDigits = [0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d,

0x07, 0x7f, 0x6f, 0x77, 0x7c, 0x39, 0x5e, 0x79, 0x71]

ADDR\_AUTO = 0x40

ADDR\_FIXED = 0x44

STARTADDR = 0xC0

# DEBUG = False

class TM1637: \_\_doublePoint = False

\_\_Clkpin = 0

\_\_Datapin = 0

\_\_brightness = 1.0 # default to max brightness

\_\_currentData = [0, 0, 0, 0]

def \_\_init\_\_(self, CLK, DIO, brightness): self.\_\_Clkpin = CLK self.\_\_Datapin = DIO self.\_\_brightness = brightness IO.setup(self.\_\_Clkpin, IO.OUT)

IO.setup(self.\_\_Datapin, IO.OUT)

def cleanup(self):

"""Stop updating clock, turn off display, and cleanup GPIO""" self.StopClock() self.Clear() IO.cleanup()

def Clear(self):

b = self.\_\_brightness point = self.\_\_doublePoint self.\_\_brightness = 0 self.\_\_doublePoint = False data = [0x7F, 0x7F, 0x7F, 0x7F] self.Show(data) # Restore previous settings: self.\_\_brightness = b

self.\_\_doublePoint = point

def ShowInt(self, i): s = str(i) self.Clear() for i in range(0, len(s)): self.Show1(i, int(s[i]))

def Show(self, data): for i in range(0, 4): self.\_\_currentData[i] = data[i]

self.start()

self.writeByte(ADDR\_AUTO)

self.br()

self.writeByte(STARTADDR) for i in range(0, 4): self.writeByte(self.coding(data[i]))

self.br()

self.writeByte(0x88 + int(self.\_\_brightness))

self.stop()

def Show1(self, DigitNumber, data): """show one Digit (number 0...3)""" if(DigitNumber < 0 or DigitNumber > 3): return # error

self.\_\_currentData[DigitNumber] = data

self.start()

self.writeByte(ADDR\_FIXED)

self.br()

self.writeByte(STARTADDR | DigitNumber)

self.writeByte(self.coding(data)) self.br()

self.writeByte(0x88 + int(self.\_\_brightness))

self.stop()

def SetBrightness(self, percent): """Accepts percent brightness from 0 - 1""" max\_brightness = 7.0 brightness = math.ceil(max\_brightness \* percent) if (brightness < 0): brightness = 0 if(self.\_\_brightness != brightness): self.\_\_brightness = brightness self.Show(self.\_\_currentData)

def ShowDoublepoint(self, on): """Show or hide double point divider""" if(self.\_\_doublePoint != on): self.\_\_doublePoint = on

self.Show(self.\_\_currentData)

def writeByte(self, data): for i in range(0, 8):

IO.output(self.\_\_Clkpin, IO.LOW) if(data & 0x01):

IO.output(self.\_\_Datapin, IO.HIGH) else:

IO.output(self.\_\_Datapin, IO.LOW) data = data >> 1

IO.output(self.\_\_Clkpin, IO.HIGH)

# wait for ACK

IO.output(self.\_\_Clkpin, IO.LOW)

IO.output(self.\_\_Datapin, IO.HIGH)

IO.output(self.\_\_Clkpin, IO.HIGH)

IO.setup(self.\_\_Datapin, IO.IN)

while(IO.input(self.\_\_Datapin)): sleep(0.001) if(IO.input(self.\_\_Datapin)): IO.setup(self.\_\_Datapin, IO.OUT)

IO.output(self.\_\_Datapin, IO.LOW)

IO.setup(self.\_\_Datapin, IO.IN)

IO.setup(self.\_\_Datapin, IO.OUT)

def start(self):

"""send start signal to TM1637"""

IO.output(self.\_\_Clkpin, IO.HIGH)

IO.output(self.\_\_Datapin, IO.HIGH)

IO.output(self.\_\_Datapin, IO.LOW)

IO.output(self.\_\_Clkpin, IO.LOW)

def stop(self):

IO.output(self.\_\_Clkpin, IO.LOW)

IO.output(self.\_\_Datapin, IO.LOW)

IO.output(self.\_\_Clkpin, IO.HIGH)

IO.output(self.\_\_Datapin, IO.HIGH)

def br(self): """terse break""" self.stop() self.start()

def coding(self, data): if(self.\_\_doublePoint): pointData = 0x80 else:

pointData = 0

if(data == 0x7F):

data = 0 else:

data = HexDigits[data] + pointData

return data

def clock(self, military\_time): """Clock script modified from: https://github.com/johnlr/raspberrypi-tm1637""" self.ShowDoublepoint(True) while (not self.\_\_stop\_event.is\_set()):

t = localtime() hour = t.tm\_hour if not military\_time:

hour = 12 if (t.tm\_hour % 12) == 0 else t.tm\_hour % 12

d0 = hour // 10 if hour // 10 else 0 d1 = hour % 10 d2 = t.tm\_min // 10 d3 = t.tm\_min % 10 digits = [d0, d1, d2, d3] self.Show(digits)

# # Optional visual feedback of running alarm:

# print digits

# for i in tqdm(range(60 - t.tm\_sec)):

for i in range(60 - t.tm\_sec): if (not self.\_\_stop\_event.is\_set()):

sleep(1)

def StartClock(self, military\_time=True):

# Stop event based on: http://stackoverflow.com/a/6524542/3219667

self.\_\_stop\_event = threading.Event() self.\_\_clock\_thread = threading.Thread( target=self.clock, args=(military\_time,)) self.\_\_clock\_thread.start()

def StopClock(self): try:

print 'Attempting to stop live clock' self.\_\_stop\_event.set() except: print 'No clock to close'

if \_\_name\_\_ == "\_\_main\_\_": """Confirm the display operation"""

display = TM1637(CLK=21, DIO=20, brightness=1.0)

display.Clear()

digits = [1, 2, 3, 4] display.Show(digits)

print "1234 - Working? (Press Key)" scrap = raw\_input()

print "Updating one digit at a time:"

display.Clear() display.Show1(1, 3) sleep(0.5) display.Show1(2, 2) sleep(0.5) display.Show1(3, 1) sleep(0.5) display.Show1(0, 4) print "4321 - (Press Key)"

scrap = raw\_input()

print "Add double point\n" display.ShowDoublepoint(True)

sleep(0.2) print "Brightness Off" display.SetBrightness(0) sleep(0.5) print "Full Brightness" display.SetBrightness(1) sleep(0.5) print "30% Brightness" display.SetBrightness(0.3)

sleep(0.3)

# See clock.py for how to use the clock functions!

# Code for Clock:-

#!/usr/bin/env python

# -\*- coding: utf-8 -\*-

from time import sleep

import tm1637

try:

import thread except ImportError:

import \_thread as thread

# Initialize the clock (GND, VCC=3.3V, Example Pins are DIO-20 and CLK21)

Display = tm1637.TM1637(CLK=21, DIO=20, brightness=1.0)

try:

print "Starting clock in the background (press CTRL + C to stop):"

Display.StartClock(military\_time=True)

Display.SetBrightness(1.0)

while True:

Display.ShowDoublepoint(True)

sleep(1)

Display.ShowDoublepoint(False) sleep(1)

Display.StopClock() thread.interrupt\_main() except KeyboardInterrupt: print "Properly closing the clock and open GPIO pins"

Display.cleanup()

**Output:-**



**Practical No. 03**

**Aim :-** Capturing Image Using Raspberry Pi And Pi Camera**.**

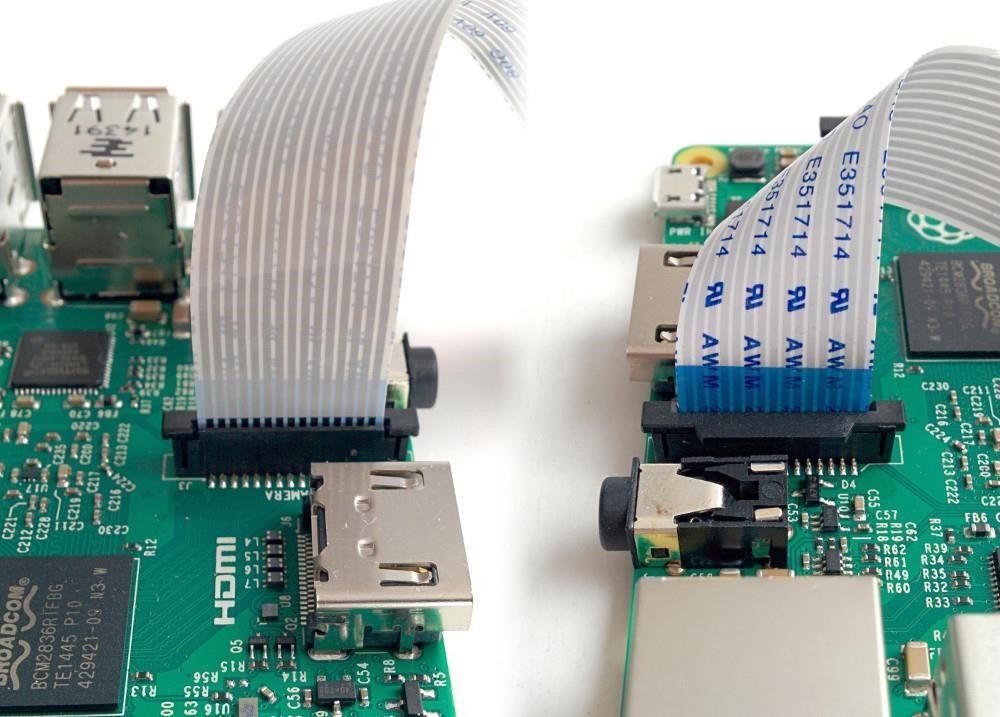
# Hardware Guide:-

Along with the basic setup you will require the following components to get started with the GPIO pins as follows:

Camera Module.

## Hardware Coonection:-

First of all, with the Pi switched off, you'll need to connect the Camera Module to the Raspberry Pi's camera port, then start up the Pi and ensure the software is enabled.



**Software Guide:**

**Command In CMD:-**

$ sudo raspistill -o /home/pi/Desktop/image.jpg

This command will capture an image and store it at the specified location (here the location specified is /home/pi/Desktop) with the specified name (here the name is ‘image.jpg’).

**Practical No. 04**

**Aim :-** GPS Module Interfacing with Raspberry Pi

**Hardware Guide:-**

For completing this lesson, you will require the following things along with your initial raspberry pi setup

1. GPS module
2. USB to TTL converter
3. Connecting wires

**GPS Module:**

Global Positioning System (GPS) makes use of signals sent by satellites in space and ground stations on Earth to accurately determine their position on Earth.

Radio Frequency signals sent from satellites and ground stations are received by the GPS. GPS makes use of these signals to determine its exact position.

The signals received from the satellites and ground stations contain time stamps of the time when the signals were transmitted.

Using information from 3 or more satellites, the exact position of the GPS can be triangulated.



GPS receiver module gives output in standard (National Marine Electronics Association) NMEA string format. It provides output serially on Tx pin with default 9600 Baud rate.

This NMEA string output from GPS receiver contains different parameters separated by commas like longitude, latitude, altitude, time etc. Each string starts with ‘$’ and ends with carriage return/line feed sequence.

E.g.

$GPGGA,184237.000,1829.9639,N,07347.6174,E,1,05,2.1,607.1,M,-64.7,M,,0000\*7D $GPGSA,A,3,15,25,18,26,12,,,,,,,,5.3,2.1,4.8\*36

$GPGSV,3,1,11,15,47,133,46,25,44,226,45,18,37,238,45,26,34,087,40\*72

$GPGSV,3,2,11,12,27,184,45,24,02,164,26,29,58,349,,05,26,034,\*7F

$GPGSV,3,3,11,21,25,303,,02,11,071,,22,01,228,\*40

$GPRMC,184237.000,A,1829.9639,N,07347.6174,E,0.05,180.19,230514,,,A\*64

**Wiring up your Circuit:**

1. Connect the VCC Pin of GPS Module to 3.3V Pin of USB to TTL converter
2. Connect the GND Pin of GPS Module to GND Pin of USB to TTL converter
3. Connect the Tx Pin of GPS Module to Rx Pin of USB to TTL converter
4. Connect the Rx Pin of GPS Module to Tx Pin of USB to TTL converter.
5. Lastly connect the USB to TTL converter to USB port of Raspberry Pi.

**Software Guide:**

Open Terminal Window and type the following command to know to which USB port the GPS module is attached: ls /dev/ttyUSB\*

We can find whether our GPS module is working properly and the connections are correct by typing the following command: sudo cat /dev/ttyUSB\*

(Here replace \* with the port number to which GPS module is attached. You should be seeing a lot of text pass by. That means it works. Type Ctrl + c to return.)

**Use 'gpsd':**

You can always just read that raw data, but its much nicer if you can have some Linux software prettify it. We'll try out gpsd which is a GPS-handling Daemon (background-helper)

**Installing a GPS Daemon (gpsd)**

The first step is installing some software on your Raspberry Pi that understands the serial data that your GPS module is providing via /dev/ttyUSB0.

Thankfully other people have already done all the hard work for you of properly parsing the raw GPS data, and we can use (amongst other options) a nice little package named 'gpsd', which essentially acts as a layer between your applications and the actual GPS hardware, gracefully handling parsing errors, and providing a common, well-defined interfaces to any GPS module.

To install gpsd, make sure your Pi has an Internet connection and run the following commands from the console:

## 1. sudo apt-get update 2. sudo apt-get install gpsd gpsd-clients python-gps

And install the software as it prompts you to do.

**Raspbian Jessie systemd service fix:**

Note if you're using the Raspbian Jessie or later release you'll need to disable a systemd service that gpsd installs. This service has systemd listen on a local socket and run gpsd when clients connect to it, however it will also interfere with other gpsd instances that are manually run (like in this guide). You will need to disable the gpsd systemd service by running the following commands:

## 1. sudo systemctl stop gpsd.socket 2. sudo systemctl disable gpsd.socket

Should you ever want to enable the default gpsd systemd service you can run these commands to restore it (but remember the rest of the steps in this guide won't work!):

## 1. sudo systemctl enable gpsd.socket 2. sudo systemctl start gpsd.socket

**Try out 'gpsd'**

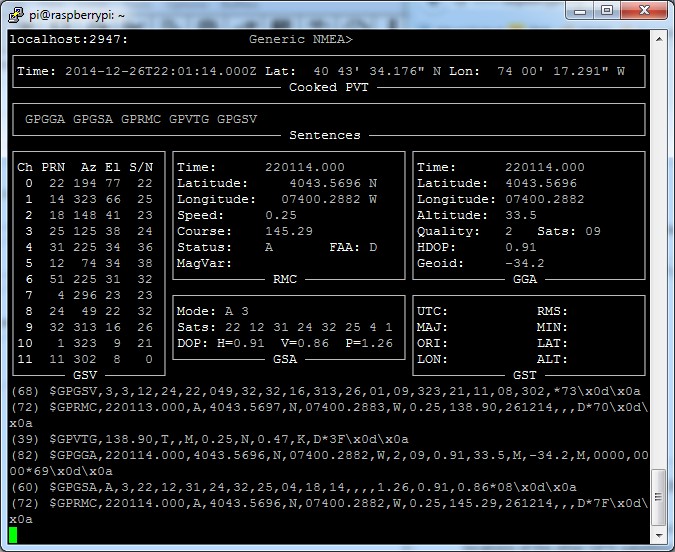
After installing gpsd and disabling the gpsd systemd service as mentioned above you're ready to start using gpsd yourself.

Start gpsd and direct it to use USB. Simply entering the following command(Here we are assuming that GPS module is connected to USB0):

## 1. sudo gpsd /dev/ttUSB0 -F /var/run/gpsd.sock

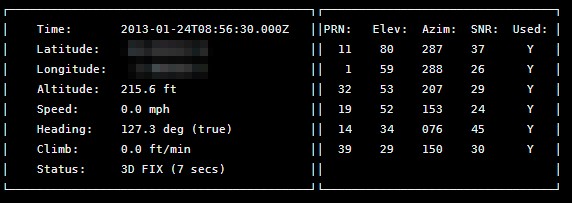
... which will point the gps daemon to our GPS device on the /dev/ttyAMA0 console

Try running **gpsmon** to get a live-streaming update of GPS data!



or cgps which gives a less detailed, but still quite nice output

1. cgps -s



You can abort gpsd by the following command

1. sudo killall gpsd

**Using Python**

Let’s extract Latitude, Longitude and time information from NMEA GPGGA string received from GPS module using Python. And print them on console (terminal). By using these latitude and longitude, locate the current position on Google Map.

**Code:**

'''

GPS Interfacing with Raspberry Pi using Pyhton

http://www.electronicwings.com

'''

import serial #import serial pacakge

from time import sleep

import webbrowser #import package for opening link in browser import sys #import system package

def GPS\_Info():

global NMEA\_buff global lat\_in\_degrees global long\_in\_degrees nmea\_time = [] nmea\_latitude = [] nmea\_longitude = []

nmea\_time = NMEA\_buff[0] #extract time from GPGGA string nmea\_latitude = NMEA\_buff[1] #extract latitude from GPGGA string nmea\_longitude = NMEA\_buff[3] #extract longitude from GPGGA string

print("NMEA Time: ", nmea\_time,'\n')

print ("NMEA Latitude:", nmea\_latitude,"NMEA Longitude:", nmea\_longitude,'\n')

lat = float(nmea\_latitude) #convert string into float for calculation longi = float(nmea\_longitude) #convertr string into float for calculation lat\_in\_degrees = convert\_to\_degrees(lat) #get latitude in degree decimal format long\_in\_degrees = convert\_to\_degrees(longi) #get longitude in degree decimal format

#convert raw NMEA string into degree decimal format def convert\_to\_degrees(raw\_value): decimal\_value = raw\_value/100.00 degrees = int(decimal\_value)

mm\_mmmm = (decimal\_value - int(decimal\_value))/0.6 position = degrees + mm\_mmmm position = "%.4f" %(position) return position

gpgga\_info = "$GPGGA,"

ser = serial.Serial ("/dev/ttyUSB0") #Open port with baud rate

GPGGA\_buffer = 0 NMEA\_buff = 0 lat\_in\_degrees = 0

long\_in\_degrees = 0

try: while True:

received\_data = (str)(ser.readline()) #read NMEA string received

GPGGA\_data\_available = received\_data.find(gpgga\_info) #check for NMEA GPGGA string if (GPGGA\_data\_available>0):

GPGGA\_buffer = received\_data.split("$GPGGA,",1)[1] #store data coming after "$GPGGA,"

NMEA\_buff = (GPGGA\_buffer.split(',')) #store comma separated data in buffer

GPS\_Info() #get time, latitude, longitude

print("lat in degrees:", lat\_in\_degrees," long in degree: ", long\_in\_degrees, '\n')

map\_link = 'http://maps.google.com/?q=' + lat\_in\_degrees + ',' + long\_in\_degrees

#create link to plot location on Google map

print("<<<<<<<<press ctrl+c to plot location on google maps>>>>>>\n")

#press ctrl+c to plot on map and exit

print("------------------------------------------------------------\n")

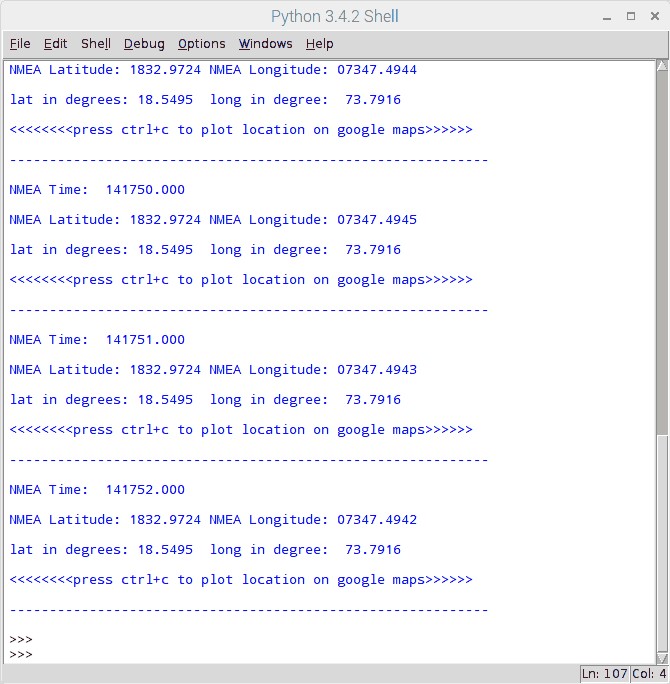
except KeyboardInterrupt:

webbrowser.open(map\_link) #open current position information in google map

sys.exit(0)

#end of file

The output of python code is as follows:



**Practical No. 05**

# Aim:- RFID Module Interfacing with Raspberry Pi

**Hardware Guide:**

For completing this lesson, you will require the following things along with your initial raspberry pi setup

1. RFID module
2. USB to TTL converter
3. Connecting wires

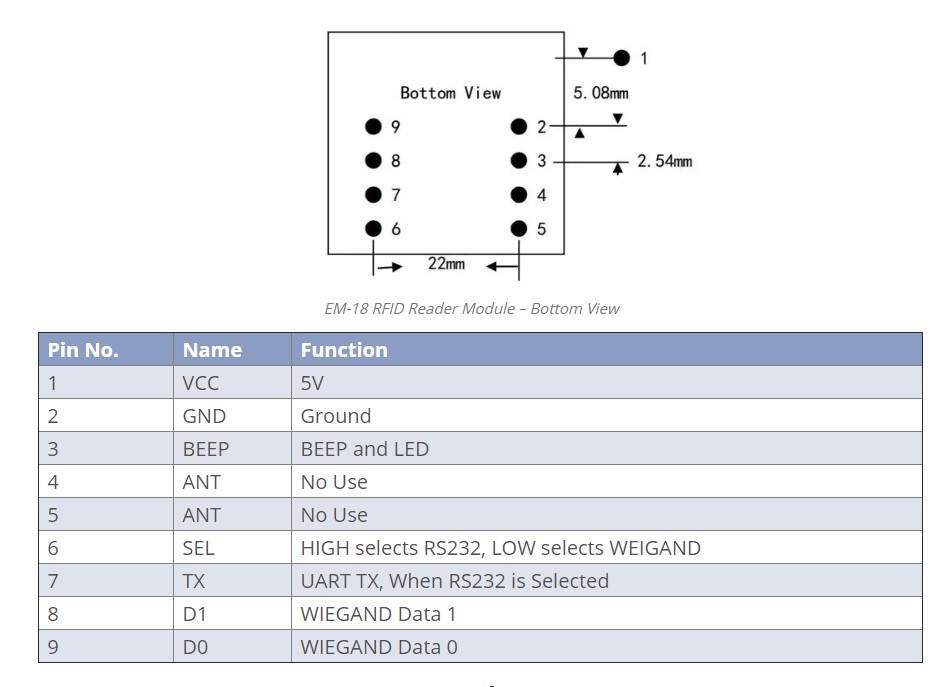
**RFID Module:-**

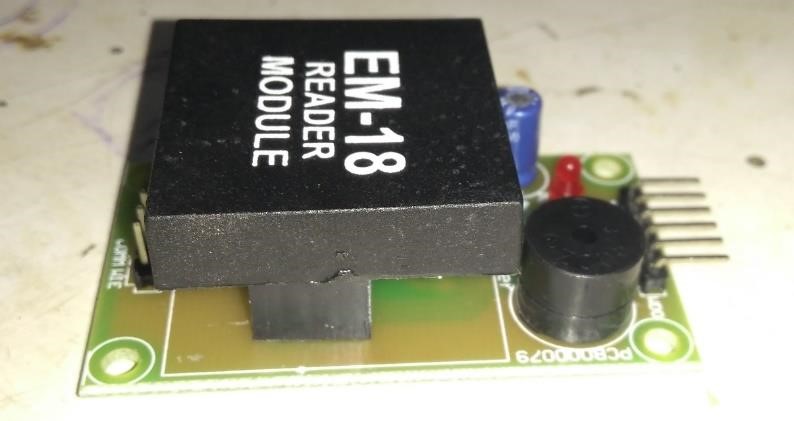
RFID (Radio Frequency Identification) uses electromagnetic fields to read, monitor and transfer data from tags attached to different objects. It is not necessary that the cards are to be in visibility of the reader, it can be embedded in the tracked object. The tags can be actively powered from a power source or can be passively powered form the incoming electromagnetic fields.

EM-18 RFID reader module is one of the commonly used reader and can read any 125KHz tags. It features low cost, low power consumption, small form factor and easy to use. It provides both UART and Wiegand26 output formats. It can be directly interfaced with microcontrollers using UART and with PC using an RS232 converter.

The module radiates 125KHz through its coils and when a 125KHz passive RFID tag is brought into this field it will get energized from this field. These passive RFID tags mostly consist of CMOS IC EM4102 which can get enough power for its working from the field generated by the reader.

Pinout of the RFID module is given as follows:

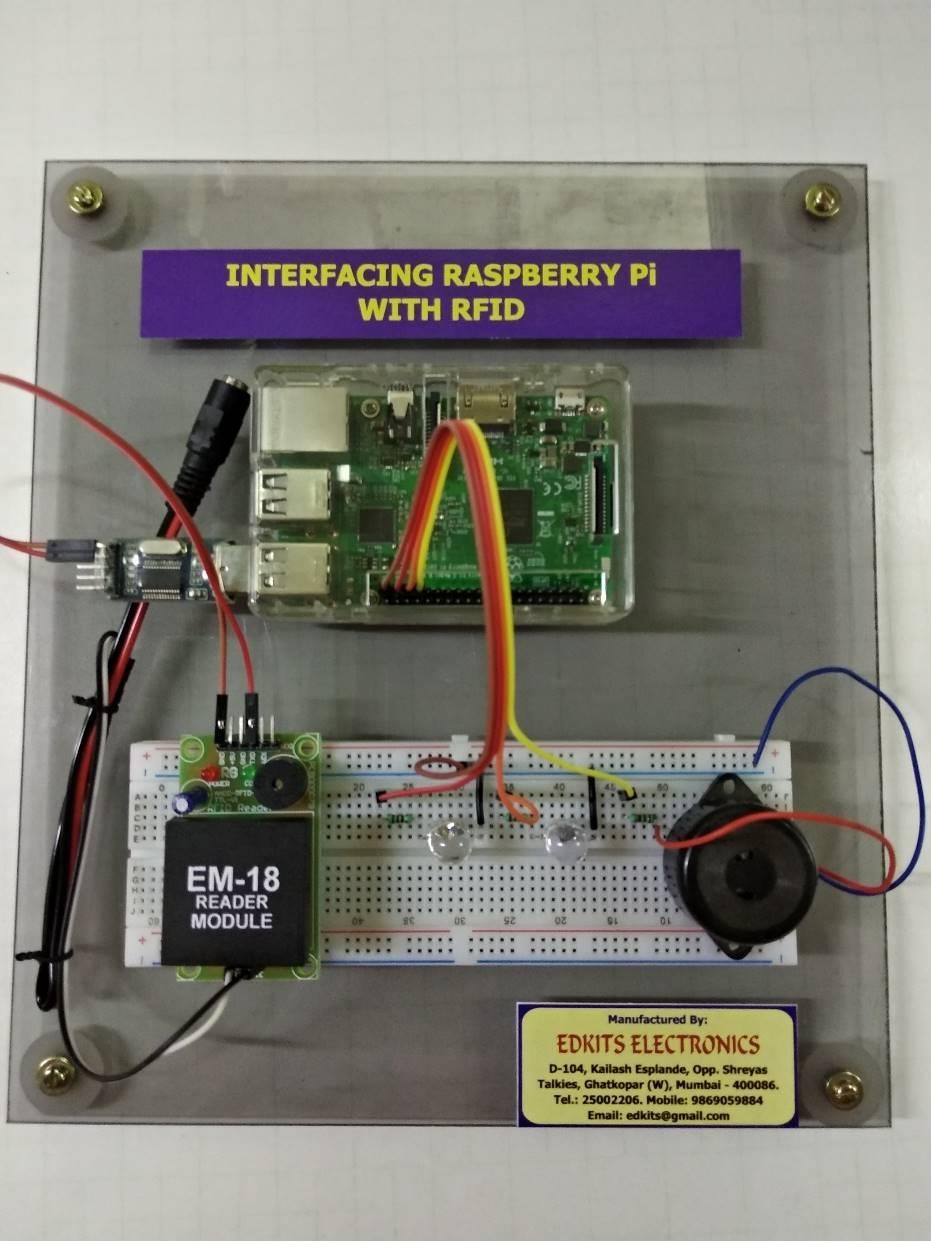




**Wiring up your Circuit:**

1. Connect TX pin of Module to Rx Pin of USB to TTL converter 2. Connect the GND Pin of Module to GND Pin of USB to TTL converter

1. Connect the positive of 5V external supply to VCC pin of module.
2. Connect the negative/GND of 5V external supply to GND of Module
3. Finally connect the USB to TTL converter to USB of raspberry Pi
4. Connect the green LED to Pin37 of raspberry Pi



1. Connect the red LED to Pin35 of raspberry pi 8. And connect the buzzer to Pin33 of raspberry Pi.

**Software Guide:**

After connecting the Module to raspberry pi via USB to TTL converter, check for the port number to which it is being connected by following command: ls /dev/ttyUSB\*

Now open Python 2 Idle and write the following code and test the working of RFID

**Code:**

import RPi.GPIO as GPIO import time

import serial #import serial module

GPIO.setmode(GPIO.BOARD)

greenLED = 37 redLED = 35

buzzer = 33

GPIO.setup(greenLED, GPIO.OUT)

GPIO.setup(redLED, GPIO.OUT)

GPIO.setup(buzzer, GPIO.OUT)

GPIO.output(greenLED, False)

GPIO.output(redLED, False)

GPIO.output(buzzer, True) time.sleep(0.1)

GPIO.output(buzzer, False) time.sleep(0.1)

GPIO.output(buzzer, True) time.sleep(0.1)

GPIO.output(buzzer, False) time.sleep(0.1)

def read\_rfid ():

ser = serial.Serial ("/dev/ttyUSB0") #Open named port ser.baudrate = 9600 #Set baud rate to 9600 data = ser.read(12) #Read 12 characters from serial port to data ser.close () #Close port return data #Return data

try:

while True:

id = read\_rfid () #Function call

print (id) #Print RFID

if id=="400034E165F0": #replace the ID number with your ID number print("Acces Granted") GPIO.output(greenLED, True)

GPIO.output(redLED, False) GPIO.output(buzzer, False)

time.sleep(2) else:

print("Access Denied") GPIO.output(greenLED, False)

GPIO.output(redLED, True) GPIO.output(buzzer, True) time.sleep(2)

GPIO.output(greenLED, False)

GPIO.output(redLED, False) GPIO.output(buzzer, False) finally:

GPIO.cleanup()

**Practical No. 06**

# Aim:- Setting up Wireless Access Point using Raspberry Pi

The Raspberry Pi can be used as a wireless access point, running a standalone network. This can be done using the inbuilt wireless features of the Raspberry Pi 3 or Raspberry Pi Zero W, or by using a suitable USB wireless dongle that supports access points.

Note that this documentation was tested on a Raspberry Pi 3, and it is possible that some USB dongles may need slight changes to their settings. If you are having trouble with a USB wireless dongle, please check the forums.

To add a Raspberry Pi-based access point to an existing network, see this section.

In order to work as an access point, the Raspberry Pi will need to have access point software installed, along with DHCP server software to provide connecting devices with a network address. Ensure that your Raspberry Pi is using an up-to-date version of Raspbian (dated 2017 or later).

Use the following to update your Raspbian installation:

1. sudo apt-get update
2. sudo apt-get upgrade

Install all the required software in one go with this command:

1. sudo apt-get install dnsmasq hostapd

Since the configuration files are not ready yet, turn the new software off as follows:

1. sudo systemctl stop dnsmasq
2. sudo systemctl stop hostapd

**Configuring a static IP**

We are configuring a standalone network to act as a server, so the Raspberry Pi needs to have a static IP address assigned to the wireless port. This documentation assumes that we are using the standard 192.168.x.x IP addresses for our wireless network, so we will assign the server the IP address 192.168.4.1. It is also assumed that the wireless device being used is wlan0.

To configure the static IP address, edit the dhcpcd configuration file with:

* sudo nano /etc/dhcpcd.conf

Go to the end of the file and edit it so that it looks like the following:

interface wlan0 static ip\_address=192.168.4.1/24 nohook wpa\_supplicant

Now restart the dhcpcd daemon and set up the new wlan0 configuration:

* sudo service dhcpcd restart

Configuring the DHCP server (dnsmasq)

The DHCP service is provided by dnsmasq. By default, the configuration file contains a lot of information that is not needed, and it is easier to start from scratch. Rename this configuration file, and edit a new one:

* sudo mv /etc/dnsmasq.conf /etc/dnsmasq.conf.orig
* sudo nano /etc/dnsmasq.conf

Type or copy the following information into the dnsmasq configuration file and save it:

interface=wlan0 # Use the require wireless interface - usually wlan0 dhcp-range=192.168.4.2,192.168.4.20,255.255.255.0,24h

So for wlan0, we are going to provide IP addresses between 192.168.4.2 and 192.168.4.20, with a lease time of 24 hours. If you are providing DHCP services for other network devices (e.g. eth0), you could add more sections with the appropriate interface header, with the range of addresses you intend to provide to that interface.

There are many more options for dnsmasq; see the dnsmasq documentation for more details.

**Configuring the access point host software (hostapd)**

You need to edit the hostapd configuration file, located at /etc/hostapd/hostapd.conf, to add the various parameters for your wireless network. After initial install, this will be a new/empty file. • sudo nano /etc/hostapd/hostapd.conf

Add the information below to the configuration file. This configuration assumes we are using channel 7, with a network name of NameOfNetwork, and a password AardvarkBadgerHedgehog. Note that the name and password should not have quotes around them. The passphrase should be between 8 and 64 characters in length.

interface=wlan0 driver=nl80211 ssid=NameOfNetwork hw\_mode=g

channel=7 wmm\_enabled=0 macaddr\_acl=0 auth\_algs=1 ignore\_broadcast\_ssid=0

wpa=2 wpa\_passphrase=AardvarkBadgerHedgehog

wpa\_key\_mgmt=WPA-PSK wpa\_pairwise=TKIP

rsn\_pairwise=CCMP

We now need to tell the system where to find this configuration file.

* sudo nano /etc/default/hostapd

Find the line with #DAEMON\_CONF, and replace it with this:

* DAEMON\_CONF="/etc/hostapd/hostapd.conf"

**Start it up**

Now start up the remaining services:

* sudo systemctl start hostapd
* sudo systemctl start dnsmasq

**ADD ROUTING AND MASQUERADE**

Edit /etc/sysctl.conf and uncomment this line:

* net.ipv4.ip\_forward=1

Add a masquerade for outbound traffic on eth0:

* sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE

Save the iptables rule.

* sudo sh -c "iptables-save > /etc/iptables.ipv4.nat"

Edit /etc/rc.local and add this just above "exit 0" to install these rules on boot. • iptables-restore < /etc/iptables.ipv4.nat

Reboot

Using a wireless device, search for networks. The network SSID you specified in the hostapd configuration should now be present, and it should be accessible with the specified password. If SSH is enabled on the Raspberry Pi access point, it should be possible to connect to it from another Linux box (or a system with SSH connectivity present) as follows, assuming the pi account is present:

* ssh pi@192.168.4.1

By this point, the Raspberry Pi is acting as an access point, and other devices can associate with it. Associated devices can access the Raspberry Pi access point via its IP address for operations such as rsync, scp, or ssh.

**Using the Raspberry Pi as an access point to share an internet connection (bridge)**

One common use of the Raspberry Pi as an access point is to provide wireless connections to a wired Ethernet connection, so that anyone logged into the access point can access the internet, providing of course that the wired Ethernet on the Pi can connect to the internet via some sort of router. To do this, a 'bridge' needs to put in place between the wireless device and the Ethernet device on the access point Raspberry Pi. This bridge will pass all traffic between the two interfaces. Install the following packages to enable the access point setup and bridging.

* sudo apt-get install hostapd bridge-utils

Since the configuration files are not ready yet, turn the new software off as follows: • sudo systemctl stop hostapd

Bridging creates a higher-level construct over the two ports being bridged. It is the bridge that is the network device, so we need to stop the eth0 and wlan0 ports being allocated IP addresses by the DHCP client on the Raspberry Pi.

* sudo nano /etc/dhcpcd.conf

Add denyinterfaces wlan0 and denyinterfaces eth0 to the end of the file (but above any other added interface lines) and save the file.

Add a new bridge, which in this case is called br0.

* sudo brctl addbr br0

Connect the network ports. In this case, connect eth0 to the bridge br0.

* sudo brctl addif br0 eth0

Now the interfaces file needs to be edited to adjust the various devices to work with bridging. sudo nano /etc/network/interfaces make the following edits.

Add the bridging information at the end of the file.

# Bridge setup auto br0 iface br0 inet manual

bridge\_ports eth0 wlan0

The access point setup is almost the same as that shown in the previous section. Follow the instructions above to set up the hostapd.conf file, but add bridge=br0 below the interface=wlan0 line, and remove or comment out the driver line. The passphrase must be between 8 and 64 characters long.

interface=wlan0 bridge=br0 #driver=nl80211 ssid=NameOfNetwork hw\_mode=g

channel=7 wmm\_enabled=0 macaddr\_acl=0 auth\_algs=1 ignore\_broadcast\_ssid=0

wpa=2 wpa\_passphrase=AardvarkBadgerHedgehog

wpa\_key\_mgmt=WPA-PSK wpa\_pairwise=TKIP

rsn\_pairwise=CCMP

Now reboot the Raspberry Pi.

There should now be a functioning bridge between the wireless LAN and the Ethernet connection on the Raspberry Pi, and any device associated with the Raspberry Pi access point will act as if it is connected to the access point's wired Ethernet.

The ifconfig command will show the bridge, which will have been allocated an IP address via the wired Ethernet's DHCP server. The wlan0 and eth0 no longer have IP addresses, as they are now controlled by the bridge. It is possible to use a static IP address for the bridge if required, but generally, if the Raspberry Pi access point is connected to a ADSL router, the DHCP address will be fine.

**Practical No. 07**

# Aim:- Raspberry Pi based Oscilloscope.

One of the most important tools in Electrical/Electronic engineering is **The Oscilloscope**.

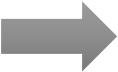
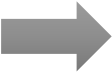
An oscilloscope is a laboratory instrument commonly used to display and analyze the waveform of electronic signals. In effect, the device draws a graph of the instantaneous signal voltage as a function of time.

In this project we will seek to replicate the signal visualization capabilities of the oscilloscope using the Raspberry Pi and an analog to digital converter module.

Replicating the signal visualization of the oscilloscope using the Raspberry Pi will require the following steps;

1. Perform Digital to analog conversion of the Input signal
2. Prepare the resulting data for representation
3. Plot the data on a live time graph

**Analog Signals Signal Processing Visualize using the Graph**



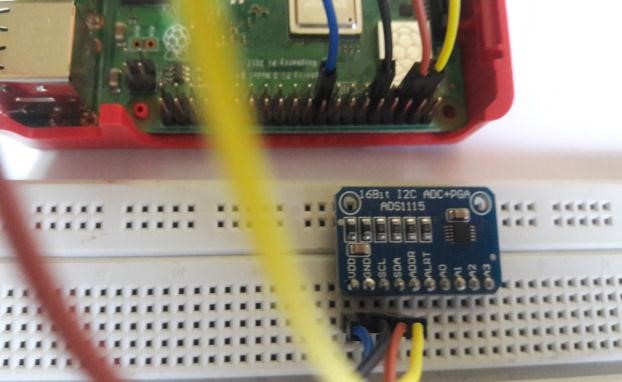
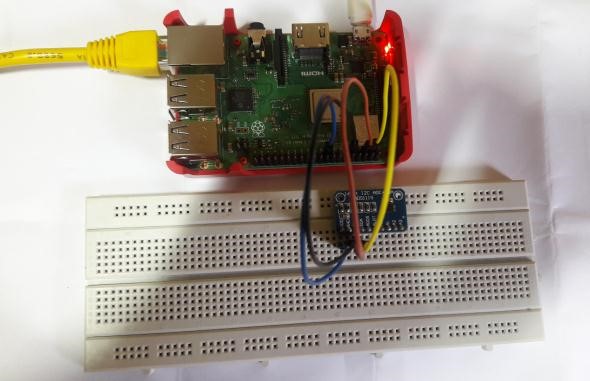
* **Hardware Requirements** 
  1. Raspberry Pi Model A/B/B+
  2. ADS1115 ADC
  3. Breadboard
  4. Jumper Wires

**ADS1115 ADC chip** is used to convert the analog input signals to digital signals which can be visualized with the Raspberry Pi. This chip is important because the Raspberry Pi does not have an onboard analog to digital converter (ADC).

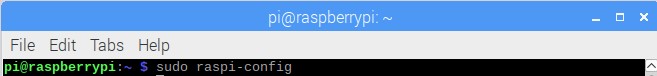
* **Software Requirements** 
  1. Raspbian Stretch OS
  2. Adafruit module for interfacing with the ADS1115 ADC chip 3. Python Module **matplotlib** used for data visualization

**1. Connect your ADC with Raspberry Pi's GPIO Pins.**

|  |  |  |
| --- | --- | --- |
| **ADS1115 ADC** | **Pin Number** | **GPIO Number** |
| **VDD** | **Pin 17** | **3.3v** |
| **GND** | **Pin 9** | **GND** |
| **SCL** | **Pin 5** | **GPIO 3** |
| **SDA** | **Pin 3** | **GPIO 2** |

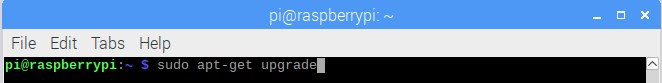
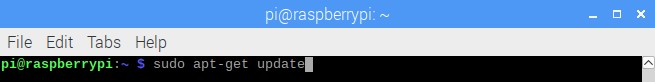


**Step 1: Enable Raspberry Pi I2C interface**



**Go to Interfacing Options**  **I2C**  **Enable (Yes)**

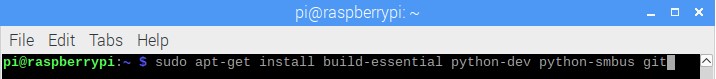
**Step 2: Update the Raspberry pi**



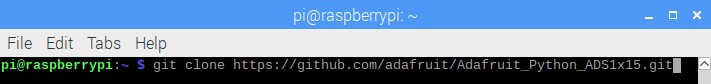
**Step 3: Install the Adafruit ADS1115 library for ADC**

To install the dependencies starting with the Adafruit python module for the ADS115 chip,

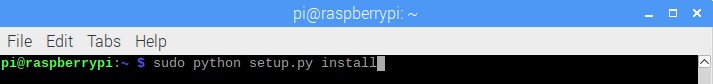
Ensure you are in the Raspberry Pi home directory ($ cd ~)



Next, clone the Adafruit git folder for the library by running



Change into the cloned file’s directory and run the setup file



**Step 4: Test the library and 12C communication.**

Now,it is important to test the library and ensure the ADC can communicate with the raspberry pi over I2C. To do this use an example script that comes with the library.

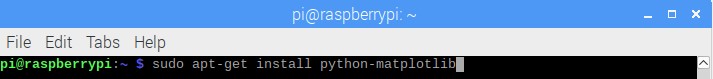
$ cd examples

$ python simpletest.py

If the I2C module is enabled and connections good, it should display the data as below

If an error occurs, check to ensure the ADC is well connected to the PI and I2C communication is enabled on the Pi.

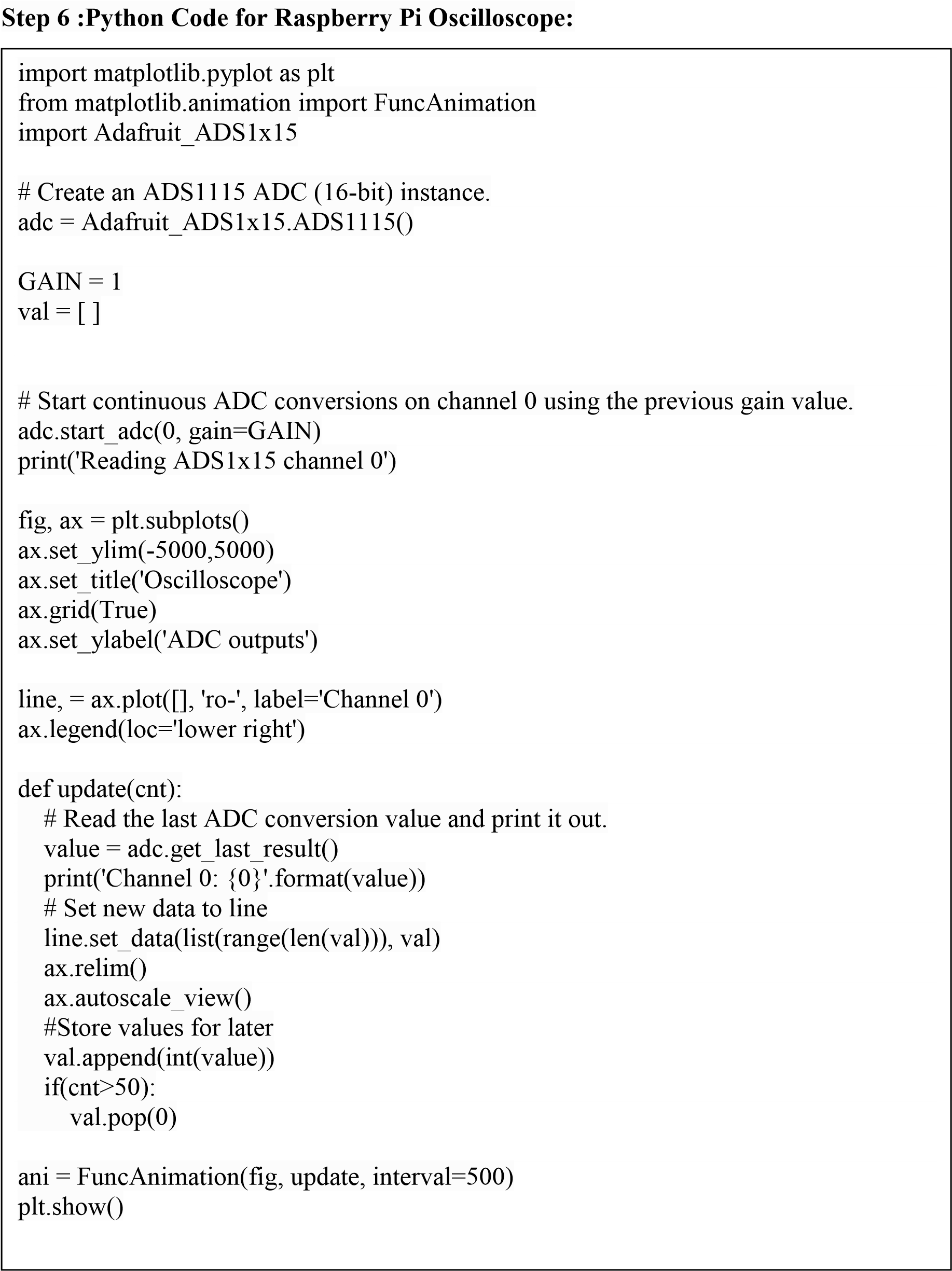
**Step 5: Install *Matplotlib***



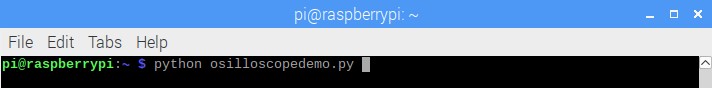
With all the dependencies installed, we are now ready to write the code.

At this stage it is important to switch to a monitor or use the VNC viewer (or Remote Desktop

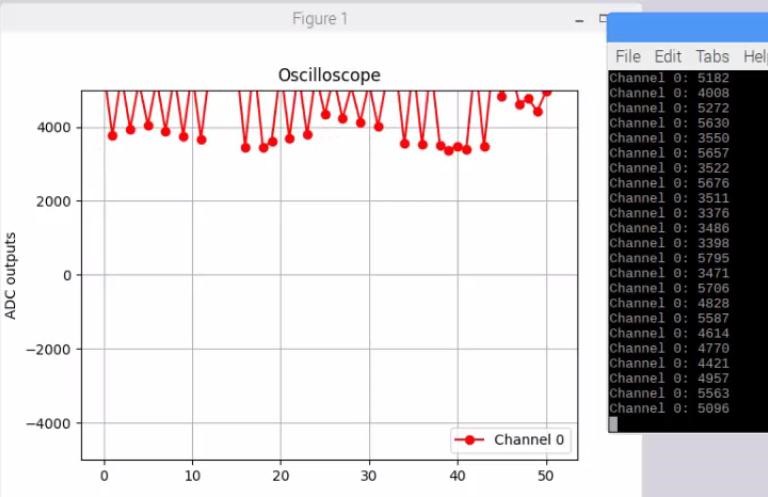
Connection), anything through which you can see your Raspberry Pi’s desktop, as the graph being plotted won’t show on the terminal.



**Save the code and run using**



**ADC data being printed on the terminal,and related Plot is also visible.**



**Practical No. 08**

**Aim:** Controlling Raspberry pi with whatsapp.

|  |  |
| --- | --- |
|  | **Step 1: Installation**  Update the packages with |
| sudo apt-get update sudo apt-get upgrade |
| Update firmware |
| sudo rpi-update |
| Prepare the system with the necessary components to Yowsup |
| sudo apt-get install python-dateutil sudo apt-get install python-setuptools sudo apt-get install python-dev sudo apt-get install libevent-dev sudo apt-get install ncurses-dev |
| Download the library with the command |
| git clone git://github.com/tgalal/yowsup.git |
|  |

|  |  |
| --- | --- |
|  | navigate to the folder |
| cd yowsup |
| and install the library with the command |
| sudo python setup.py install |
| **Step 2: Registration**  After installing the library we have to register the device to use WhatsApp. Yowsup comes with a cross platform command-line frontend called yowsup-cli. It provides you with the options of registration, and provides a few demos such as a command line client.  WhatsApp registration involves 2 steps. First you need to request a registration code. And then you resume the registration with code you got.  Request a code with command |
| python yowsup-cli registration --requestcode sms --phone 39xxxxxxxxxx --cc 39 --mcc 2 22 --mnc 10 |
| Replace with your data , cc is your country code in this example 39 is for Italy, mcc is Mobile Country Code mnc is Mobile Network Code  You should receive on your phone a sms message with a code like xxx-xxx  Send a message to request registration with this command, (replace xxx-xxx with code you received) |
| python yowsup-cli registration --register xxx-xxx --phone 39xxxxxxxxxx --cc 39 |
| If all goes well, we should get a message like this |
| status: ok |

|  |  |
| --- | --- |
|  | kind: free  pw: xxxxxxxxxxxxxxxxxx=  price: € 0,89 price\_expiration: 1416553637 currency: EUR cost: 0.89 expiration: 1445241022 login: 39xxxxxxxxxxx  type: existing |
| **Warning**  WhatsApp requires the registration of a number, and with that number you can use WhatsApp on only one device at a time, so it is preferable to use a new number.  WhatsApp can be used on one device at a time and if you will make many attempts to register the number, it could be banned. We recommend you using Telegram.  **Step 3: Utilization**  Create a file to save your credentials |
| sudo nano /home/pi/yowsup/config |
| with this content |
| ## Actual config starts below ##  cc=39 #if not specified it will be autodetected phone=39xxxxxxxxxx password=xxxxxxxxxxxxxxx= |
| Ok, we're ready for the test, Yowsup has a demo application in  /home/pi/yowsup/yowsup/demos  Navigate to yowsup folder |

|  |  |
| --- | --- |
|  | cd /home/pi/yowsup |
| Start yowsup-cli demos with the command |
| yowsup-cli demos --yowsup --config config |
| You can see Yowsup prompt  If type "/help" you can see all available commands  First use the '/L' command for login; to send a message type |
| /message send 39xxxxxxxxxx "This is a message sent from Raspberry Pi" |
| replace xxx with the recipient number  If you respond with a message it will be displayed on Raspberry. |