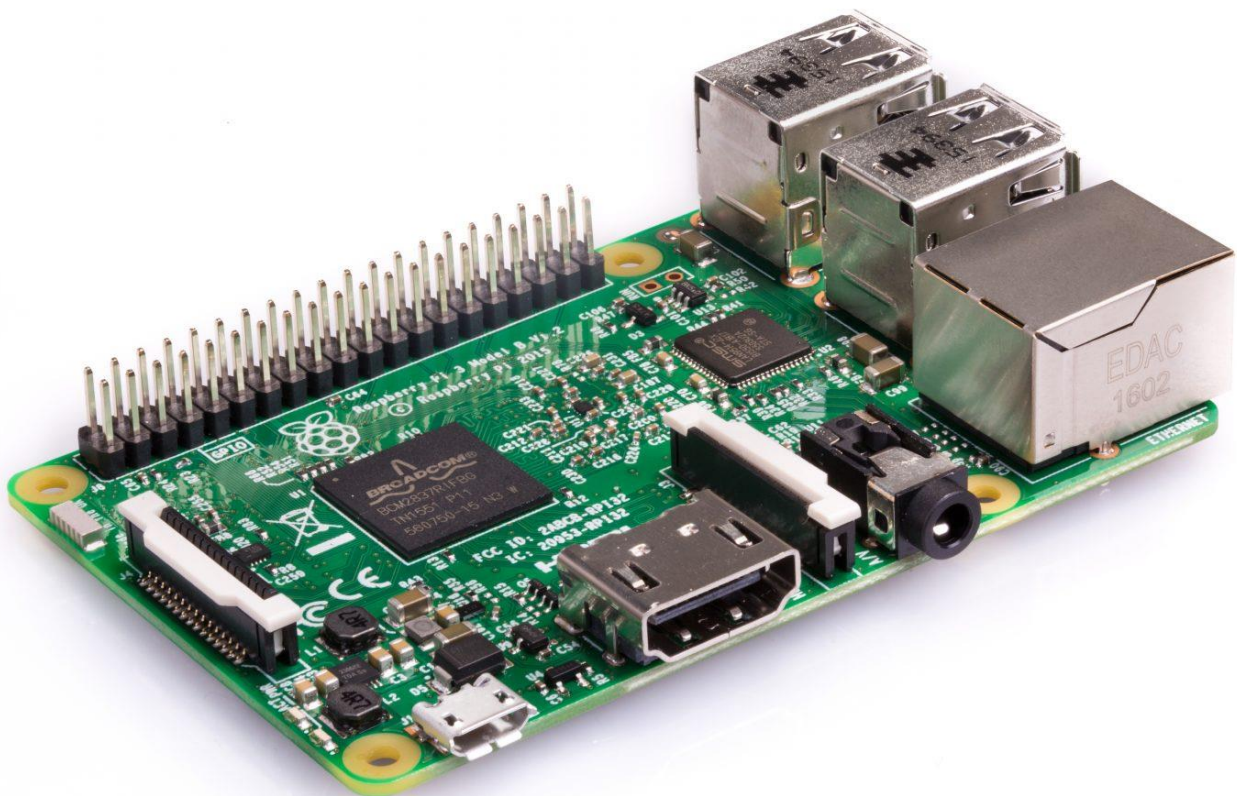
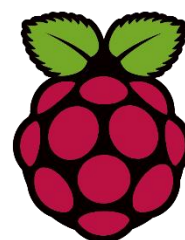


**EDKITS ELECTRONICS**

# Raspberry Pi

## Guide for TYBSc IT





## Table of Contents

<b>Raspberry Pi Hardware Preparation and Installation .....</b>	<b>3</b>
<b>Displaying different LED Patterns with Raspberry Pi .....</b>	<b>6</b>
<b>Displaying Time over 4 Digit 7 Segment Display using Raspberry Pi .....</b>	<b>13</b>
<b>Fingerprint Sensor interfacing with Raspberry Pi .....</b>	<b>16</b>
<b>GPS Module interfacing with Raspberry Pi .....</b>	<b>26</b>
<b>RFID Module interfacing with Raspberry Pi .....</b>	<b>32</b>
<b>Capturing Images with Raspberry Pi and Pi Camera .....</b>	<b>36</b>
<b>Visitor Monitoring with Raspberry Pi and Pi Camera .....</b>	<b>39</b>
<b>IOT Based Home Automation using raspberry Pi.....</b>	<b>43</b>
<b>Installing Windows 10 IOT Core on Raspberry Pi.....</b>	<b>47</b>
<b>Building Google Assistant with Raspberry Pi .....</b>	<b>49</b>
<b>Setting up Wireless Access Point using Raspberry Pi.....</b>	<b>58</b>
<b>Oscilloscope using Raspberry Pi.....</b>	<b>63</b>
<b>LED Matrix Module interfacing with Raspberry Pi.....</b>	<b>69</b>

# Raspberry Pi Hardware Preparation and Installation

## **Hardware Guide:**

For getting started with raspberry pi for the first time you will require the following hardware

1. Raspberry Pi (latest Model)
2. Monitor or TV
3. HDMI cable
4. Ethernet cable
5. USB keyboard
6. USB mouse
7. Micro USB power supply
8. 8GB or larger microSD card
9. SD Card Reader

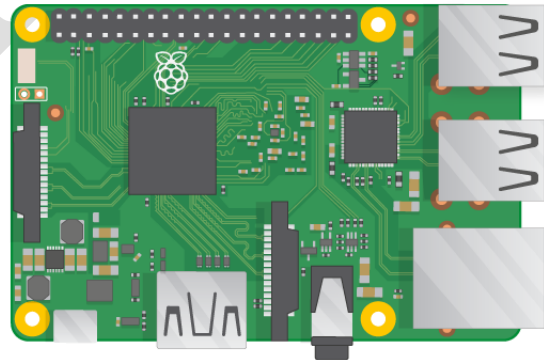
## **Raspberry Pi 3 Model B:**

The Raspberry Pi 3 is the third generation Raspberry Pi. It replaced the Raspberry Pi 2 Model B in February 2016. Compared to the Raspberry Pi 2 it has:

- A 1.2GHz 64-bit quad-core ARMv8 CPU
- 802.11n Wireless LAN
- Bluetooth 4.1
- Bluetooth Low Energy (BLE)

Like the Pi 2, it also has:

- 4 USB ports
- 40 GPIO pins
- Full HDMI port
- Ethernet port
- Combined 3.5mm audio jack and composite video
- Camera interface (CSI)
- Display interface (DSI)
- Micro SD card slot (now push-pull rather than push-push)
- Video Core IV 3D graphics core
- The Raspberry Pi 3 has an identical form factor to the previous Pi 2 (and Pi 1 Model B+) and has complete compatibility with Raspberry Pi 1 and 2.



### **Monitor or TV:**

A monitor or TV with HDMI in can be used as a display with a Raspberry Pi. Most modern television sets and monitors have an HDMI port, and are the easiest to get working with the Raspberry Pi. You can use an HDMI cable to connect the Raspberry Pi directly to the television or monitor.

Some older monitors have a DVI port. These work well with the Raspberry Pi, although you'll need an HDMI-to-DVI adapter to attach to an HDMI cable, or a one-piece HDMI-to-DVI cable. Some old monitors have a VGA port. These can be trickier to use as you'll need an HDMI-to-VGA converter, which can change digital video to analogue video. A simple port adapter won't work.

### **HDMI to HDMI Cable:**

Connect Raspberry Pi to a Monitor or TV with a HDMI to HDMI cable.

### **Ethernet cable:**

Ethernet cable will allow your Pi to connect with the internet. It is also useful for headless setup of Raspberry Pi

### **USB Keyboard and Mouse:**

Any standard USB keyboard and mouse can be used with the Raspberry Pi. This plug and play devices will work without any additional driver. Simply plug them into the Raspberry Pi and they should be recognised when it starts up.

### **Power Supply:**

It is recommended that you use a 5V, 2A USB power supply for all models of Raspberry Pi.

### **SD Card:**

The latest version of Raspbian, the default operating system recommended for the Raspberry Pi, requires an 8GB (or larger) micro SD card. SD card will store the operating systems as well as all the file and applications created by you.

## **Installation Guide:**

Now since you have all the required hardware, we will now learn how to get the operating system onto your microSD card so that you can start using software on your Raspberry Pi

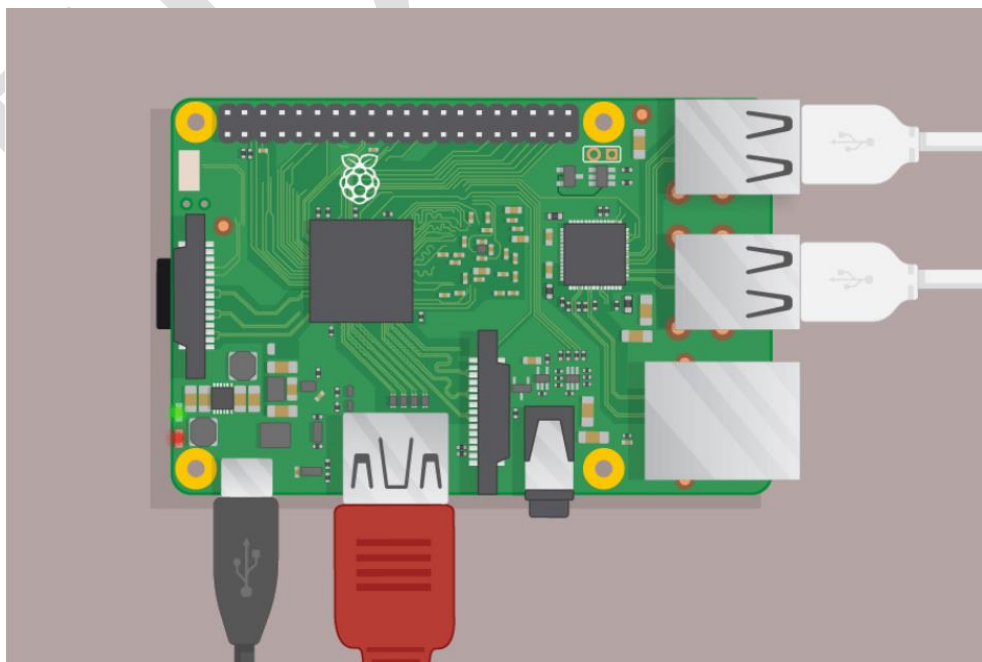
### **Get Raspbian OS on your microSD card:**

Raspbian comes pre-installed with plenty of software for education, programming and general use. It has Python, Scratch, Sonic Pi, Java, Mathematica and more.

1. To download Raspbian log on to [raspberrypi.org](http://raspberrypi.org) and click on the download, then click on Raspbian and lastly download the RASPBIAN JESSIE WITH DESKTOP file. You can choose either the Torrent file or ZIP file.
2. The downloaded file will be in zip format. To unzip the file, you will require an unzip tool. You can use any unzipping tool viz. WINRAR, 7ZIP etc. After unzipping the file, you will find a disc image file in the unzipped folder.
3. Now format the SD Card before writing the disc image file on the SD card. You can use SD Formatter tool or any other tool of your wish.
4. To write the image file of the operating system on the SD card you will require a Disk Imager tool. For this you can use Win32 Disk Imager tool.
5. Once the image is written on the SD Card, your untitled SD card will now have the name boot. Your SD Card will now hold the Raspbian Operating system required for the first-time setup.

### **Plugging in your Raspberry Pi:**

1. Begin by placing your SD card into the SD card slot on the Raspberry Pi. It will only fit one way.
2. Next, plug your keyboard and mouse into the USB ports on the Raspberry Pi.
3. Make sure that your monitor or TV is turned on, and that you have selected the right input (e.g. HDMI 1, DVI, etc).
4. Connect your HDMI cable from your Raspberry Pi to your monitor or TV.
5. If you intend to connect your Raspberry Pi to the internet, plug an Ethernet cable into the Ethernet port, or connect a WiFi dongle to one of the USB ports (unless you have a Raspberry Pi 3).
6. When you're happy that you have plugged all the cables and SD card in correctly, connect the micro USB power supply. This action will turn on and boot your Raspberry Pi.



# Displaying Different LED Patterns with Raspberry Pi

After setting up the raspberry pi and having hands on practice with the Linux commands, you are now familiar with raspberry pi. Now it's time to work with the GPIO pins of the raspberry pi to have an external interface with the raspberry pi.

## **Hardware Guide:**

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

1. LED
2. Resistor
3. Connecting wires
4. Breadboard

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

## **GPIO?**

One powerful feature of the Raspberry Pi is the row of GPIO (general purpose input/output) pins along the top edge of the board.

These pins are a physical interface between the Pi and the outside world. At the simplest level, you can think of them as switches that you can turn on or off (input) or that the Pi can turn on or off (output). Of the 40 pins, 26 are GPIO pins and the others are power or ground pins (plus two ID EEPROM pins which you should not play with unless you know your stuff!)



## **What are they for? What can we do with them?**

You can program the pins to interact in amazing ways with the real world. Inputs don't have to come from a physical switch; it could be input from a sensor or a signal from another computer or device, for example. The output can also do anything, from turning on an LED to sending a signal or data to another device. If the Raspberry Pi is on a network, you can control devices that are attached to it from anywhere\*\* and those devices can send data back. Connectivity and control of physical devices over the internet is a powerful and exciting thing, and the Raspberry Pi is ideal for this.



## How the GPIO pins work?

### Output

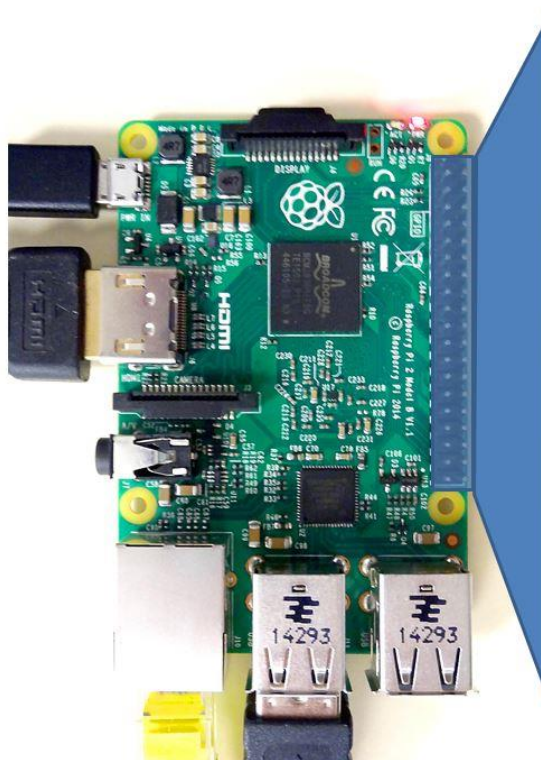
When we use a GPIO pin as an output. Each pin can turn on or off, or go HIGH or LOW in computing terms. When the pin is HIGH it outputs 3.3 volts (3v3); when the pin is LOW it is off.

### Input

GPIO outputs are easy; they are on or off, HIGH or LOW, 3v3 or 0v. Inputs are a bit trickier because of the way that digital devices work. Although it might seem reasonable just to connect a button across an input pin and a ground pin, the Pi can get confused as to whether the button is on or off. It might work properly, it might not. It's a bit like floating about in deep space; without a reference, it would be hard to tell if you were going up or down, or even what up or down meant!

Therefore, you will see phrases like "pull up" and "pull down" in Raspberry Pi GPIO tutorials. It's a way of giving the input pin a reference so it knows for certain when an input is received.

**Warning: Randomly plugging wires and power sources into your Pi, however, may kill it. Bad things can also happen if you try to connect things to your Pi that use a lot of power.**



Physical Pins					
GPIO#	2nd func	pin#	pin#	2nd func	GPIO#
N/A	+3V3	1	2	+5V	N/A
GPIO2	SDA1 (I2C)	3	4	+5V	N/A
GPIO3	SCL1 (I2C)	5	6	GND	N/A
GPIO4	GCLK	7	8	TXD0 (UART)	GPIO14
N/A	GND	9	10	RXD0 (UART)	GPIO15
GPIO17	GEN0	11	12	GEN1	GPIO18
GPIO27	GEN2	13	14	GND	N/A
GPIO22	GEN3	15	16	GEN4	GPIO23
N/A	+3V3	17	18	GEN5	GPIO24
GPIO10	MOSI (SPI)	19	20	GND	N/A
GPIO9	MISO (SPI)	21	22	GEN6	GPIO25
GPIO11	SCLK (SPI)	23	24	CE0_N (SPI)	GPIO8
N/A	GND	25	26	CE1_N (SPI)	GPIO7
EEPROM	ID_SD	27	28	ID_SC	EEPROM
GPIO5	N/A	29	30	GND	N/A
GPIO6	N/A	31	32	-	GPIO12
GPIO13	N/A	33	34	GND	N/A
GPIO19	N/A	35	36	N/A	GPIO16
GPIO26	N/A	37	38	N/A	GPIO20
N/A	GND	39	40	N/A	GPIO21

### A note on pin numbering:

When programming the GPIO pins there are two different ways to refer to them: GPIO numbering and physical numbering.

GPIO numbering:

These are the GPIO pins as the computer sees them. The numbers don't make any sense to humans, they jump about all over the place, so there is no easy way to remember them. You will need a printed reference or a reference board that fits over the pins.

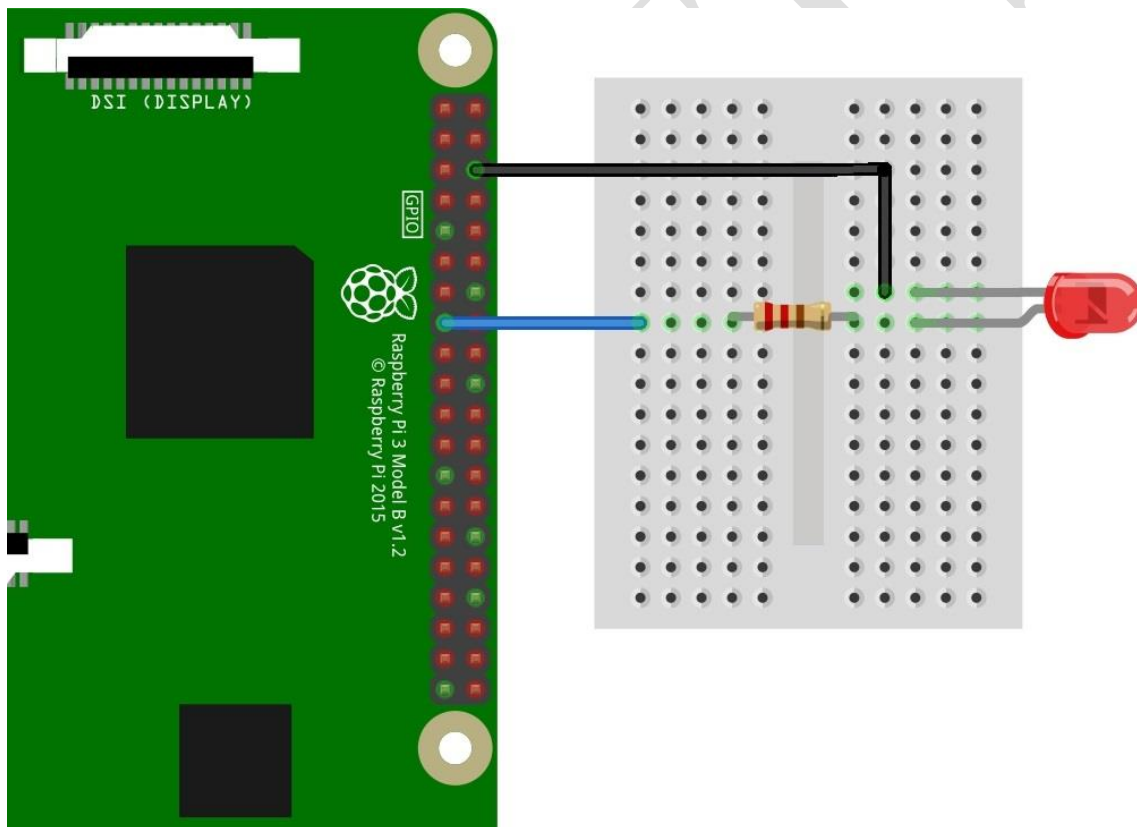
Physical numbering:

The other way to refer to the pins is by simply counting across and down from pin 1 at the top left (nearest to the SD card).

### **Wiring up your Circuit:**

1. Connect the GPIO22 (i.e. Physical Pin 15) Pin of raspberry pi to one end of the resistor.
2. Connect another end of resistor to the positive end (anode) of LED
3. Connect the negative end (cathode) of LED to Ground of raspberry pi.
4. Then Power on your raspberry pi

**Circuit Diagram:**



### **Software Guide:**

Raspbian OS comes with many preinstalled programming environments. Here we will be using Python for coding.



To open Python, click on the application Menu, navigate to Programming, then click on Python 3 (IDLE) an Integrated Development Environment for Python 3. After opening the IDE, go to files and open new file to start your code.

### Code for single LED Blink:

```
#Blink LED Program
#Connect the LED to GPIO 22 Pin
#LED Blink Program
#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
```

After writing your code save it on a desired location and run it to enjoy the fun.

Great! You have completed your first GPIO program. Now you are confident enough to go forward to explore and interface new things with raspberry Pi.

If you were unable to blink the LED, don't worry, recheck your connection and debug the error so that you don't repeat it again.

Now connect 8 LEDs in the same format to the pin numbers given in the program below

**Code for LED pattern:**

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BOARD)
led1 = 29
led2 = 31
led3 = 33
led4 = 35
led5 = 36
led6 = 37
led7 = 38
led8 = 40

#setup the ledPin(i.e. GPIO22) as output
GPIO.setup(led1, GPIO.OUT)
GPIO.setup(led2, GPIO.OUT)
GPIO.setup(led3, GPIO.OUT)
GPIO.setup(led4, GPIO.OUT)
GPIO.setup(led5, GPIO.OUT)
GPIO.setup(led6, GPIO.OUT)
GPIO.setup(led7, GPIO.OUT)
GPIO.setup(led8, GPIO.OUT)
GPIO.output(led1, False)
GPIO.output(led2, False)
GPIO.output(led3, False)
GPIO.output(led4, False)
GPIO.output(led5, False)
GPIO.output(led6, False)
GPIO.output(led7, False)
GPIO.output(led8, False)

def ledpattern(ledVal1, ledVal2, ledVal3, ledVal4, ledVal5, ledVal6, ledVal7, ledVal8):
    GPIO.output(led1, ledVal1)
    GPIO.output(led2, ledVal2)
    GPIO.output(led3, ledVal3)
    GPIO.output(led4, ledVal4)
    GPIO.output(led5, ledVal5)
    GPIO.output(led6, ledVal6)
    GPIO.output(led7, ledVal7)
    GPIO.output(led8, ledVal8)

def patterOne():
    for i in range(0, 3):
        ledpattern(1, 0, 1, 0, 1, 0, 1, 0)
```

```

time.sleep(1)
ledpattern(0, 1, 0, 1, 0, 1, 0, 1)
time.sleep(1)

def patternTwo():
    for i in range (0, 5):
        ledpattern(1, 0, 0, 0, 0, 0, 0, 0)
        time.sleep(0.1)
        ledpattern(0, 1, 0, 0, 0, 0, 0, 0)
        time.sleep(0.1)
        ledpattern(0, 0, 1, 0, 0, 0, 0, 0)
        time.sleep(0.1)
        ledpattern(0, 0, 0, 1, 0, 0, 0, 0)
        time.sleep(0.1)
        ledpattern(0, 0, 0, 0, 1, 0, 0, 0)
        time.sleep(0.1)
        ledpattern(0, 0, 0, 0, 0, 1, 0, 0)
        time.sleep(0.1)
        ledpattern(0, 0, 0, 0, 0, 0, 1, 0)
        time.sleep(0.1)
        ledpattern(0, 0, 0, 0, 0, 0, 0, 1)
        time.sleep(0.1)

def patternThree():
    for i in range (0, 5):
        ledpattern(0, 0, 0, 0, 0, 0, 0, 1)
        time.sleep(0.1)
        ledpattern(0, 0, 0, 0, 0, 0, 1, 0)
        time.sleep(0.1)
        ledpattern(0, 0, 0, 0, 0, 1, 0, 0)
        time.sleep(0.1)
        ledpattern(0, 0, 0, 0, 1, 0, 0, 0)
        time.sleep(0.1)
        ledpattern(0, 0, 0, 1, 0, 0, 0, 0)
        time.sleep(0.1)
        ledpattern(0, 0, 1, 0, 0, 0, 0, 0)
        time.sleep(0.1)
        ledpattern(0, 1, 0, 0, 0, 0, 0, 0)
        time.sleep(0.1)
        ledpattern(1, 0, 0, 0, 0, 0, 0, 0)
        time.sleep(0.1)

def patternFour():
    for i in range (0, 5):
        ledpattern(0, 1, 1, 1, 1, 1, 1, 1)
        time.sleep(0.1)
        ledpattern(1, 0, 1, 1, 1, 1, 1, 1)
        time.sleep(0.1)

```

```

    ledpattern(1, 1, 0, 1, 1, 1, 1, 1)
    time.sleep(0.1)
    ledpattern(1, 1, 1, 0, 1, 1, 1, 1)
    time.sleep(0.1)
    ledpattern(1, 1, 1, 1, 0, 1, 1, 1)
    time.sleep(0.1)
    ledpattern(1, 1, 1, 1, 1, 0, 1, 1)
    time.sleep(0.1)
    ledpattern(1, 1, 1, 1, 1, 1, 0, 1)
    time.sleep(0.1)
    ledpattern(1, 1, 1, 1, 1, 1, 1, 0)
    time.sleep(0.1)

def patternFive():
    for i in range (0, 5):
        ledpattern(1, 1, 1, 1, 1, 1, 1, 0)
        time.sleep(0.1)
        ledpattern(1, 1, 1, 1, 1, 1, 0, 1)
        time.sleep(0.1)
        ledpattern(1, 1, 1, 1, 1, 0, 1, 1)
        time.sleep(0.1)
        ledpattern(1, 1, 1, 1, 0, 1, 1, 1)
        time.sleep(0.1)
        ledpattern(1, 1, 1, 0, 1, 1, 1, 1)
        time.sleep(0.1)
        ledpattern(1, 1, 0, 1, 1, 1, 1, 1)
        time.sleep(0.1)
        ledpattern(1, 0, 1, 1, 1, 1, 1, 1)
        time.sleep(0.1)
        ledpattern(0, 1, 1, 1, 1, 1, 1, 1)
        time.sleep(0.1)

try:
    while True:
        patterOne()
        patternTwo()
        patternThree()
        patternFour()
        patternFive()

finally:
    #reset the GPIO Pins
    GPIO.cleanup()

```

# Displaying Time over 4 Digit 7 Segment Display using Raspberry Pi

In this lesson we will interface 4-digit seven segment display with raspberry pi.

## **Hardware Guide:**

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

1. TM1637 4-digit seven segment Display board
2. Connecting wires

### **TM1637 4 Digit seven segment Display Board:**

This is a common anode 4-digit tube display module which uses the TM1637 driver chip; Only 2 connections are required to control the 4-digit 8-segment displays

Here is the module



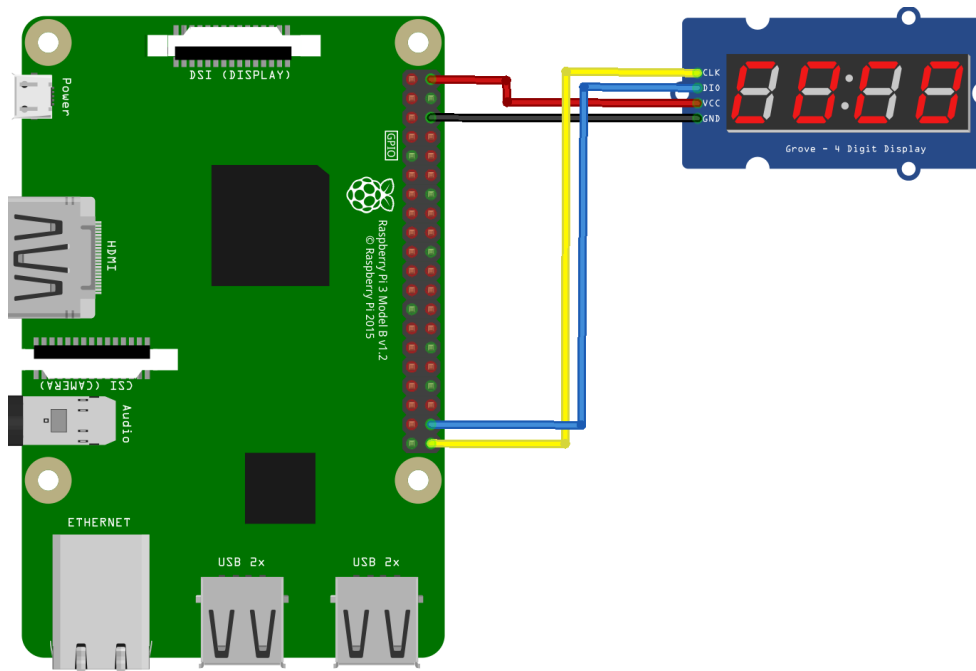
Features of the module:

1. Display common anode for the four red LED
2. Powered supply by 3.3V/5V
3. Four common anode tube display module is driven by IC TM1637

## **Wiring up your Circuit:**

Hook up your circuit as follows:

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



fritzing

## Software Guide:

1. Now to download libraries, open Web Browser on your Raspberry Pi and log on to the following link: <https://github.com/timwaizenegger/raspberrypi-examples/tree/master/actor-led-7segment-4numbers> . Click on the actor-led-7segment-4numbers.zip folder and Now click on Download Button to download the file.
2. Now on your rpi move to /home/pi/Downloads/ location to find the zip file downloaded.
3. Unzip the file and try to execute the different example codes present in that folder in Python 2 Idle.
4. Now open Python 2 Idle, create a new file, write the code given below and save it in the same folder i.e. actor-led-7segment-4numbers since the code below is depended on tm1637.py file which is present in the same folder.

### Code:

**#Program to display Time on 4-digit Seven segment display**

```
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()
```

(Note: the code given above is the edited form of clock.py program in 'actor-led-7segment-4number' folder.)

# Fingerprint Sensor interfacing with Raspberry Pi

In this lesson we will learn to interface fingerprint sensor with raspberry pi. For this we will be using the USB port of raspberry pi.

## **Hardware Guide:**

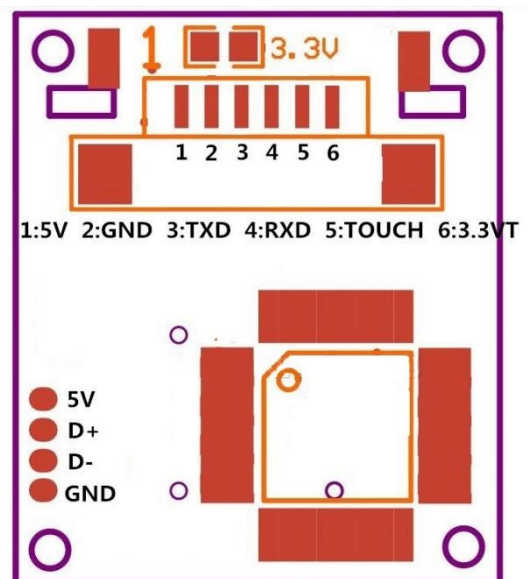
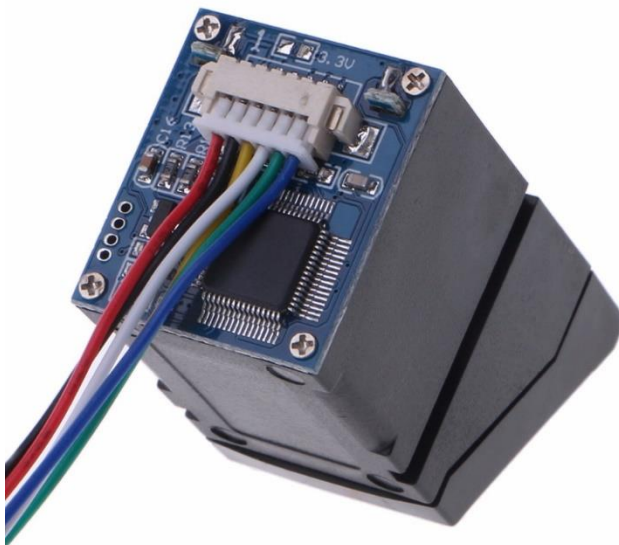
For completing this lesson, you will require the following things along with your initial raspberry pi setup

1. Fingerprint Sensor
2. USB to TTL/UART converter
3. Connecting wires
4. Push Buttons
5. 16x2 LCD
6. LED
7. Breadboard

## **Fingerprint Sensor:**

It is an intelligent module which can freely get fingerprint, image processing, verified fingerprint, search and storage, and it can work normally without upper monitor's participatory management.

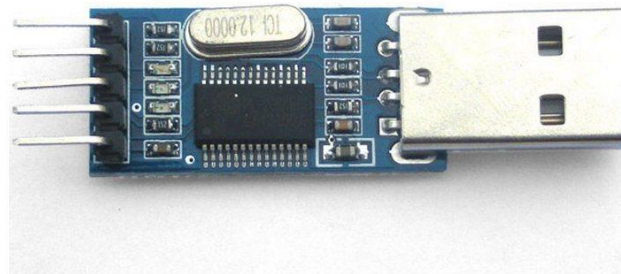
Fingerprint processing includes two parts: fingerprint enrolment and fingerprint matching (the matching can be 1:1 or 1: N). Enrolling fingerprint, user needs to enter the finger 2-4 times for every one finger, process finger images with many times, store generate templates on module. When fingerprint matching, enrol and process verified fingerprint image and then matching with



module (if match with appoint templates on the module, named fingerprint verification, for 1:1 matching method; if match with many templates on the module, named fingerprint search method also named 1: N) system will return the matching result, success or failure.

#### **USB to TTL converter:**

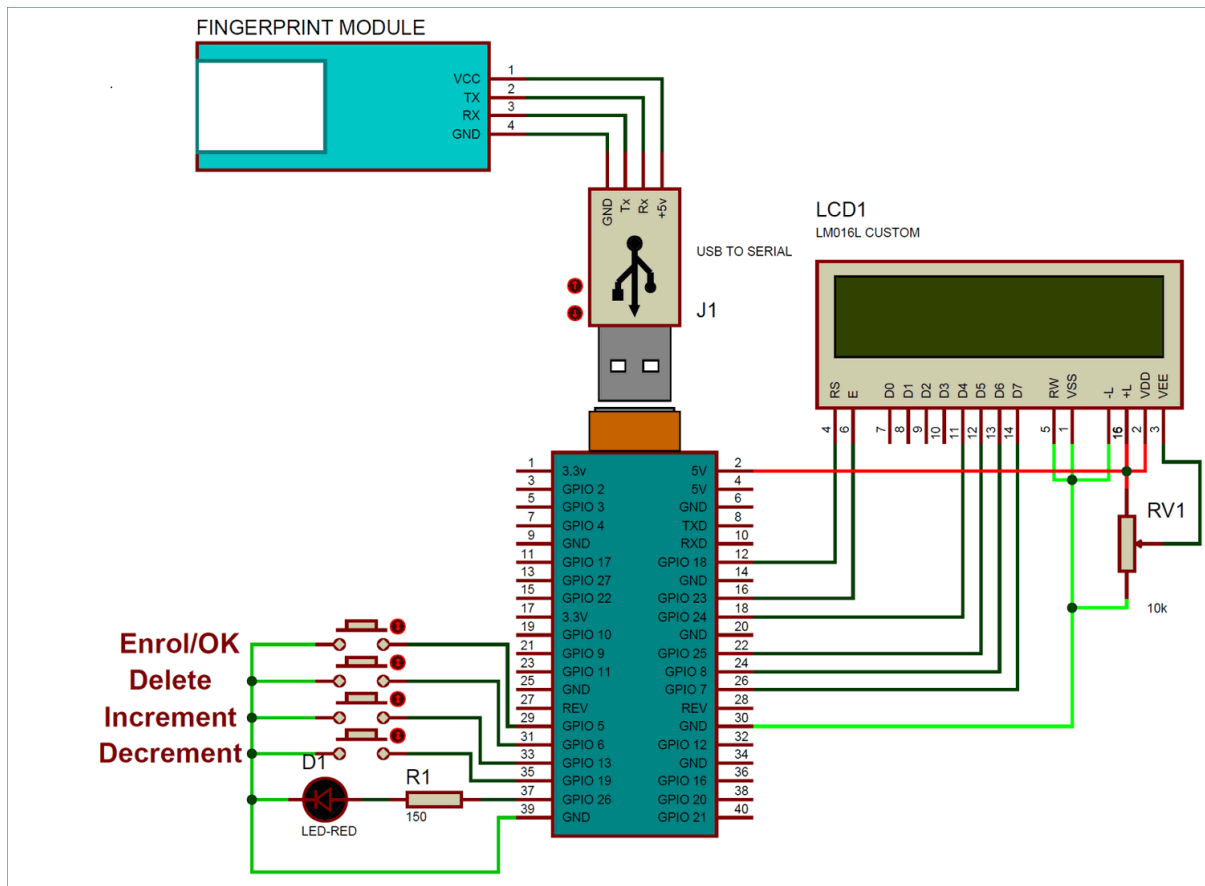
Here we have used a fingerprint module which works on UART. So here we have interfaced this fingerprint module with Raspberry Pi using a USB to Serial converter.



#### **Wiring up your circuit:**

In this Raspberry Pi Finger Print sensor interfacing project, we have used a 4 push buttons: one for enrolling the new finger pring, one for deleting the already fed finger prints and rest two for increment/decrement the position of already fed Finger prints. A LED is used for indication that fingerprint sensor is ready to take finger for matching. Here we have used a fingerprint module which works on UART. So here we have interfaced this fingerprint module with Raspberry Pi using a USB to Serial converter.

So, first of all, we need to make the all the required connection as shown in Circuit Diagram below. Connections are simple, we have just connected fingerprint module to Raspberry Pi USB port by using USB to Serial converter. A 16x2 LCD is used for displaying all messages. A 10k pot is also used with LCD for controlling the contrast of the same. 16x2 LCD pins RS, EN, d4, d5, d6, and d7 are connected with GPIO Pin 18, 23, 24, 25, 8 and 7 of Raspberry Pi respectively. Four push buttons are connected to GPIO Pin 5, 6, 13 and 19 of Raspberry Pi. LED is also connected at pin 26 of RPI.

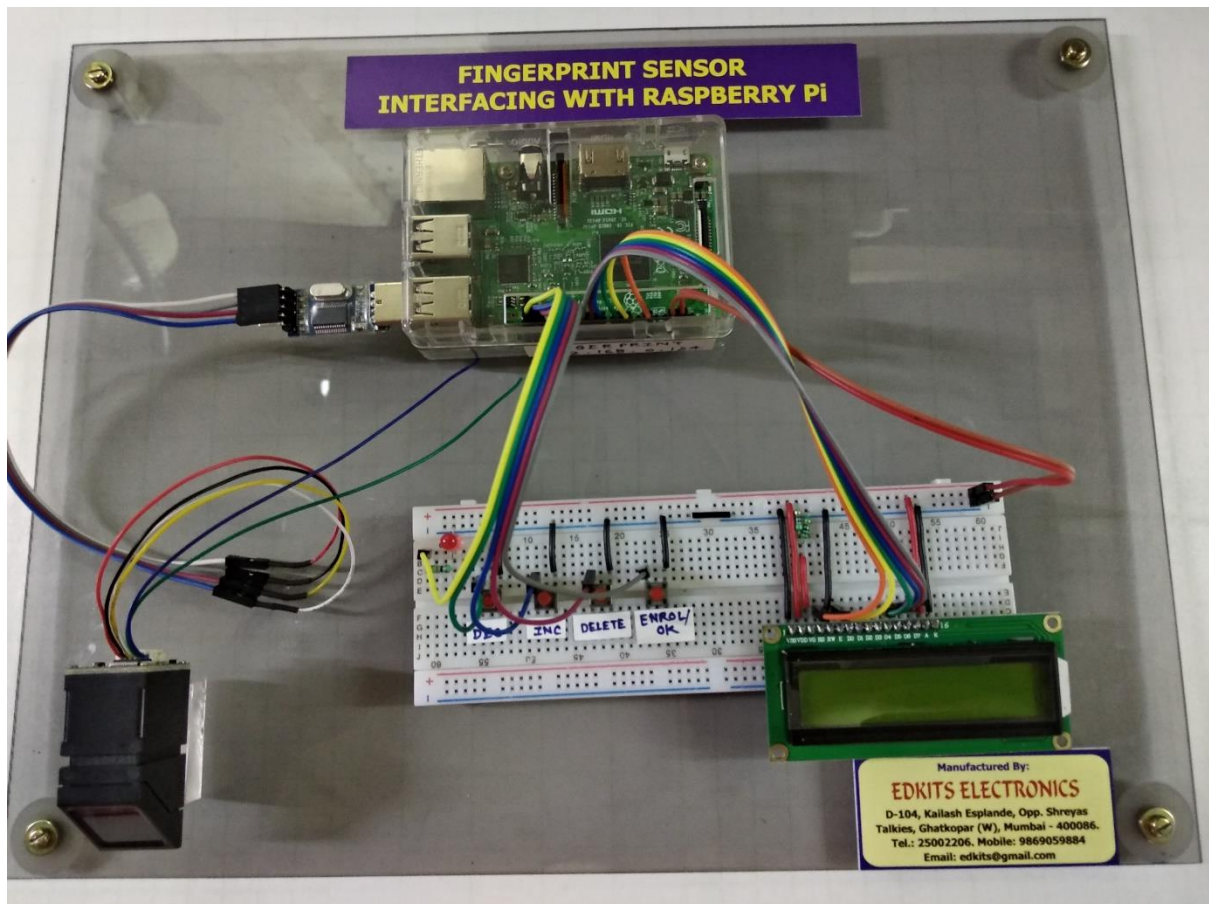


## Software Guide:

After making all the connections we need to power up Raspberry Pi and get it ready with terminal open. Now we need to install fingerprint library for Raspberry Pi in python language by following the below steps.

1. To install this library, root privileges are required. So first we enter in root by given command: `sudo bash`
2. Then download some required packages by using given commands:
  - a. `wget -O - http://apt.pm-codeworks.de/pm-codeworks.de.gpg | apt-key add -`
  - b. `wget http://apt.pm-codeworks.de/pm-codeworks.list -P /etc/apt/sources.list.d/`
3. After this, we need to update the Raspberry pi and install the downloaded finger print sensor library:
  - a. `sudo apt-get update`
  - b. `sudo apt-get install python-fingerprint`
  - c. now exit root by typing `exit`
4. After installing library now, we need to check USB port on which your finger print sensor is connected, by using given the command: `ls /dev/ttyUSB*` (or `lsusb`)

Now in your python code replace the USB port number with the one you got on the screen after executing the command in step4.



### Code:

You can download the sample codes from the GitHub link:

<https://github.com/bastianraschke/pyfingerprint> on your raspberry pi. After downloading execute the python codes present in the example folder using Python 2 Idle. After testing the example codes, now open python 2 Idle and write the following code:

```
import time                                     #import libraries
from pyfingerprint.pyfingerprint import PyFingerprint
import RPi.GPIO as gpio

RS =18                                           #define pins for LCD, buttons and LED
EN =23
D4 =24
D5 =25
D6 =8
D7 =7

enrol=5
delet=6
inc=13
```

```

dec=19
led=26

HIGH=1
LOW=0

gpio.setwarnings(False)                                #set the pins as output pins
gpio.setmode(gpio.BCM)
gpio.setup(RS, gpio.OUT)
gpio.setup(EN, gpio.OUT)
gpio.setup(D4, gpio.OUT)
gpio.setup(D5, gpio.OUT)
gpio.setup(D6, gpio.OUT)
gpio.setup(D7, gpio.OUT)

gpio.setup(enrol, gpio.IN, pull_up_down=gpio.PUD_UP)
gpio.setup(delet, gpio.IN, pull_up_down=gpio.PUD_UP)
gpio.setup(inc, gpio.IN, pull_up_down=gpio.PUD_UP)
gpio.setup(dec, gpio.IN, pull_up_down=gpio.PUD_UP)
gpio.setup(led, gpio.OUT)

try:                                                      #initialize the fingerprint sensor
    f = PyFingerprint('/dev/ttyUSB0', 57600, 0xFFFFFFFF, 0x00000000)

    if ( f.verifyPassword() == False ):
        raise ValueError('The given fingerprint sensor password is wrong!')

except Exception as e:
    print('Exception message: ' + str(e))
    exit(1)

def begin():                                              #defining LCD initialize function
    lcdcmd(0x33)
    lcdcmd(0x32)
    lcdcmd(0x06)
    lcdcmd(0x0C)
    lcdcmd(0x28)
    lcdcmd(0x01)
    time.sleep(0.0005)

def lcdcmd(ch):                                          #defining LCD command functions
    gpio.output(RS, 0)
    gpio.output(D4, 0)
    gpio.output(D5, 0)
    gpio.output(D6, 0)
    gpio.output(D7, 0)
    if ch&0x10==0x10:
        gpio.output(D4, 1)

```



```

if ch&0x20==0x20:
    gpio.output(D5, 1)
if ch&0x40==0x40:
    gpio.output(D6, 1)
if ch&0x80==0x80:
    gpio.output(D7, 1)
gpio.output(EN, 1)
time.sleep(0.005)
gpio.output(EN, 0)
# Low bits
gpio.output(D4, 0)
gpio.output(D5, 0)
gpio.output(D6, 0)
gpio.output(D7, 0)
if ch&0x01==0x01:
    gpio.output(D4, 1)
if ch&0x02==0x02:
    gpio.output(D5, 1)
if ch&0x04==0x04:
    gpio.output(D6, 1)
if ch&0x08==0x08:
    gpio.output(D7, 1)
gpio.output(EN, 1)
time.sleep(0.005)
gpio.output(EN, 0)

```

def lcdwrite(ch):

#defining function for writing on LCD

```

gpio.output(RS, 1)
gpio.output(D4, 0)
gpio.output(D5, 0)
gpio.output(D6, 0)
gpio.output(D7, 0)
if ch&0x10==0x10:
    gpio.output(D4, 1)
if ch&0x20==0x20:
    gpio.output(D5, 1)
if ch&0x40==0x40:
    gpio.output(D6, 1)
if ch&0x80==0x80:
    gpio.output(D7, 1)
gpio.output(EN, 1)
time.sleep(0.005)
gpio.output(EN, 0)
# Low bits
gpio.output(D4, 0)
gpio.output(D5, 0)
gpio.output(D6, 0)
gpio.output(D7, 0)

```

```

if ch&0x01==0x01:
    gpio.output(D4, 1)
if ch&0x02==0x02:
    gpio.output(D5, 1)
if ch&0x04==0x04:
    gpio.output(D6, 1)
if ch&0x08==0x08:
    gpio.output(D7, 1)
gpio.output(EN, 1)
time.sleep(0.005)
gpio.output(EN, 0)
def lcdclear():
    lcdcmd(0x01)

def lcdprint(Str):
    l=0;
    l=len(Str)
    for i in range(l):
        lcdwrite(ord(Str[i]))
#defining function to print string on LCD

def setCursor(x,y):
    if y == 0:
        n=128+x
    elif y == 1:
        n=192+x
    lcdcmd(n)
#defining function for cursor position

def enrollFinger():
    lcdcmd(1)
    lcdprint("Enrolling Finger")
    time.sleep(2)
    print('Waiting for finger...')
    lcdcmd(1)
    lcdprint("Place Finger")
    while ( f.readImage() == False ):
        pass
    f.convertImage(0x01)
    result = f.searchTemplate()
    positionNumber = result[0]
    if ( positionNumber >= 0 ):
        print('Template already exists at position #' + str(positionNumber))
        lcdcmd(1)
        lcdprint("Finger ALready")
        lcdcmd(192)
        lcdprint(" Exists ")
        time.sleep(2)
        return
    print('Remove finger...')

```

```

lcdcmd(1)
lcdprint("Remove Finger")
time.sleep(2)
print('Waiting for same finger again...')
lcdcmd(1)
lcdprint("Place Finger")
lcdcmd(192)
lcdprint(" Again ")
while ( f.readImage() == False ):
    pass
f.convertImage(0x02)
if ( f.compareCharacteristics() == 0 ):
    print "Fingers do not match"
    lcdcmd(1)
    lcdprint("Finger Did not")
    lcdcmd(192)
    lcdprint(" Mactched ")
    time.sleep(2)
    return
f.createTemplate()
positionNumber = f.storeTemplate()
print('Finger enrolled successfully!')
lcdcmd(1)
lcdprint("Stored at Pos:")
lcdprint(str(positionNumber))
lcdcmd(192)
lcdprint("successfully")
print('New template position #' + str(positionNumber))
time.sleep(2)

def searchFinger():                                     #defining function for searching finger stored
try:
    print("Waiting for finger...")
    while( f.readImage() == False ):
        #pass
        time.sleep(.5)
        return
    f.convertImage(0x01)
    result = f.searchTemplate()
    positionNumber = result[0]
    accuracyScore = result[1]
    if positionNumber == -1 :
        print('No match found!')
        lcdcmd(1)
        lcdprint("No Match Found")
        time.sleep(2)
        return
    else:

```

```

        print('Found template at position #' + str(positionNumber))
        lcdcmd(1)
        lcdprint("Found at Pos:")
        lcdprint(str(positionNumber))
        time.sleep(2)

except Exception as e:
    print('Operation failed!')
    print('Exception message: ' + str(e))
    exit(1)

def deleteFinger():                                #defining function to delete fingerprint
    positionNumber = 0
    count=0
    lcdcmd(1)
    lcdprint("Delete Finger")
    lcdcmd(192)
    lcdprint("Position: ")
    lcdcmd(0xca)
    lcdprint(str(count))
    while gpio.input(enrol) == True: # here enrol key means ok
        if gpio.input(inc) == False:
            count=count+1
            if count>1000:
                count=1000
            lcdcmd(0xca)
            lcdprint(str(count))
            time.sleep(0.2)
        elif gpio.input(dec) == False:
            count=count-1
            if count<0:
                count=0
            lcdcmd(0xca)
            lcdprint(str(count))
            time.sleep(0.2)
    positionNumber=count
    if f.deleteTemplate(positionNumber) == True :
        print('Template deleted!')
        lcdcmd(1)
        lcdprint("Finger Deleted");
        time.sleep(2)

begin()                                            # main program starts from here
lcdcmd(0x01)
lcdprint("FingerPrint ")
lcdcmd(0xc0)
lcdprint("Interfacing ")
time.sleep(3)

```

```
lcdcmd(0x01)

flag=0
lcdclear()

while 1:
    gpio.output(led, HIGH)
    lcdcmd(1)
    lcdprint("Place Finger")
    if gpio.input(enrol) == 0:
        gpio.output(led, LOW)
        enrollFinger()
    elif gpio.input(delet) == 0:
        gpio.output(led, LOW)
        while gpio.input(delet) == 0:
            time.sleep(0.1)
        deleteFinger()
    else:
        searchFinger()
```

Note: you can download the code from the following link:

<https://circuitdigest.com/microcontroller-projects/raspberry-pi-fingerprint-sensor-interfacing>

# 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

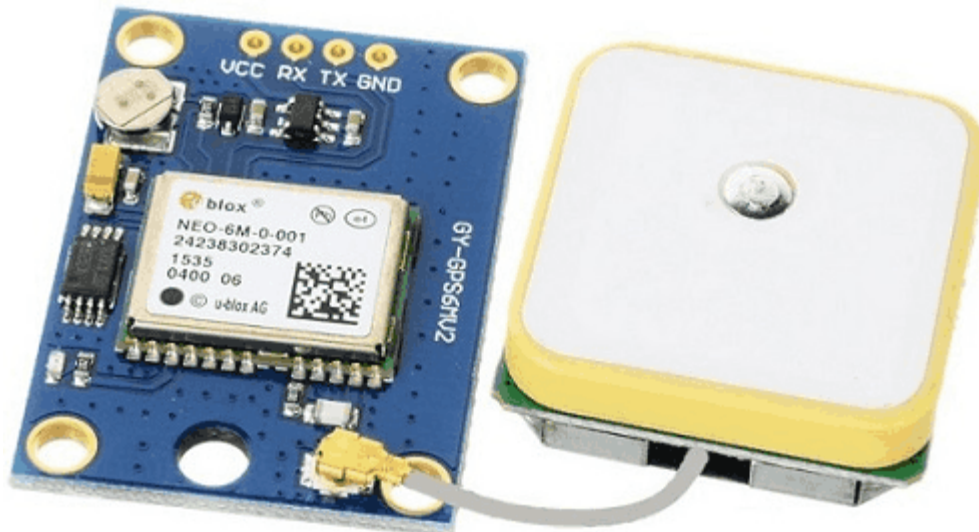
### **GPS Module:**

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 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'

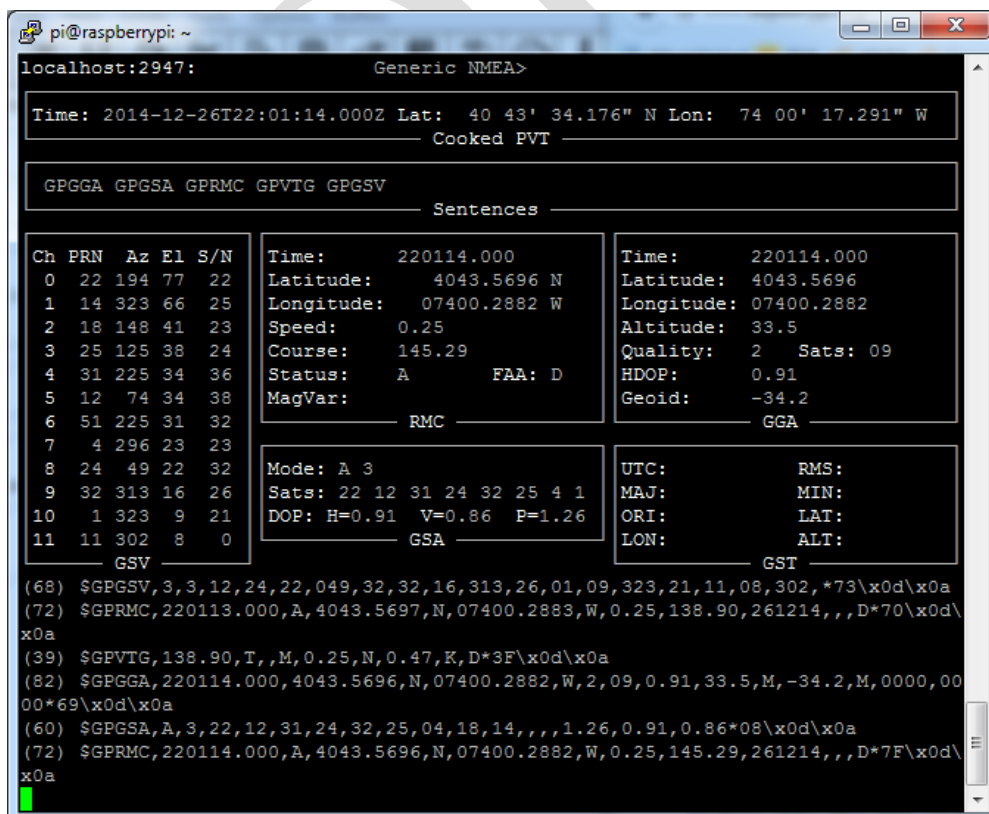
After installing gpsd and disabling the gpsd systemd service as mentioned above you're ready to start using gpsd yourself.

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/ttyUSB0 -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!



```
pi@raspberrypi: ~
localhost:2947: Generic NMEA>
Time: 2014-12-26T22:01:14.000Z Lat: 40 43' 34.176" N Lon: 74 00' 17.291" W
Cooked PVT
GPGLGA GPGLSA GPRMC GPVTG GPGSV
Sentences
Ch PRN Az El S/N Time: 220114.000 Latitude: 4043.5696 N Time: 220114.000
1 14 323 66 25 Longitude: 07400.2882 W Longitude: 07400.2882
2 18 148 41 23 Speed: 0.25 Altitude: 33.5
3 25 125 38 24 Course: 145.29 Quality: 2 Sats: 09
4 31 225 34 36 Status: A FAA: D HDOP: 0.91
5 12 74 34 38 MagVar: Geoid: -34.2
6 51 225 31 32 RMC GGA
7 4 296 23 23
8 24 49 22 32 Mode: A 3 UTC: RMS:
9 32 313 16 26 Sats: 22 12 31 24 32 25 4 1 MAJ: MIN:
10 1 323 9 21 DOP: H=0.91 V=0.86 P=1.26 ORI: LAT:
11 11 302 8 0 GSA LON: ALT:
GSV GST
(68) $GPGSV,3,3,12,24,22,049,32,32,16,313,26,01,09,323,21,11,08,302,*73\x0d\x0a
(72) $GPRMC,220113.000,A,4043.5697,N,07400.2883,W,0.25,138.90,261214,,D*70\x0d\x0a
(39) $GPVTG,138.90,T,,M,0.25,N,0.47,K,D*3F\x0d\x0a
(82) $GPGLGA,220114.000,4043.5696,N,07400.2882,W,2,09,0.91,33.5,M,-34.2,M,0000,00
00*69\x0d\x0a
(60) $GPGLSA,A,3,22,12,31,24,32,25,04,18,14,,,1.26,0.91,0.86*08\x0d\x0a
(72) $GPRMC,220114.000,A,4043.5696,N,07400.2882,W,0.25,145.29,261214,,D*7F\x0d\x0a
x0a
```

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

1. cgps -s

Time:	2013-01-24T08:56:30.000Z	PRN:	Elev:	Azim:	SNR:	Used:
Latitude:		11	80	287	37	Y
Longitude:		1	59	288	26	Y
Altitude:	215.6 ft	32	53	207	29	Y
Speed:	0.0 mph	19	52	153	24	Y
Heading:	127.3 deg (true)	14	34	076	45	Y
Climb:	0.0 ft/min	39	29	150	30	Y
Status:	3D FIX (7 secs)					

You can abort gpstd by the following command

1. `sudo killall gpstd`

## 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 Python
http://www.electronicwings.com
'''
import serial          #import serial package
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)       #convert 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

gppga_info = "$GPGGA,"
ser = serial.Serial ("/dev/ttyUSB0") #Open port with baud rate
GPPGA_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
        GPPGA_data_available = received_data.find(gppga_info) #check for NMEA GPGGA string
        if (GPPGA_data_available>0):
            GPPGA_buffer = received_data.split("$GPGGA,",1)[1] #store data coming after "$GPGGA,"
            NMEA_buff = (GPPGA_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)

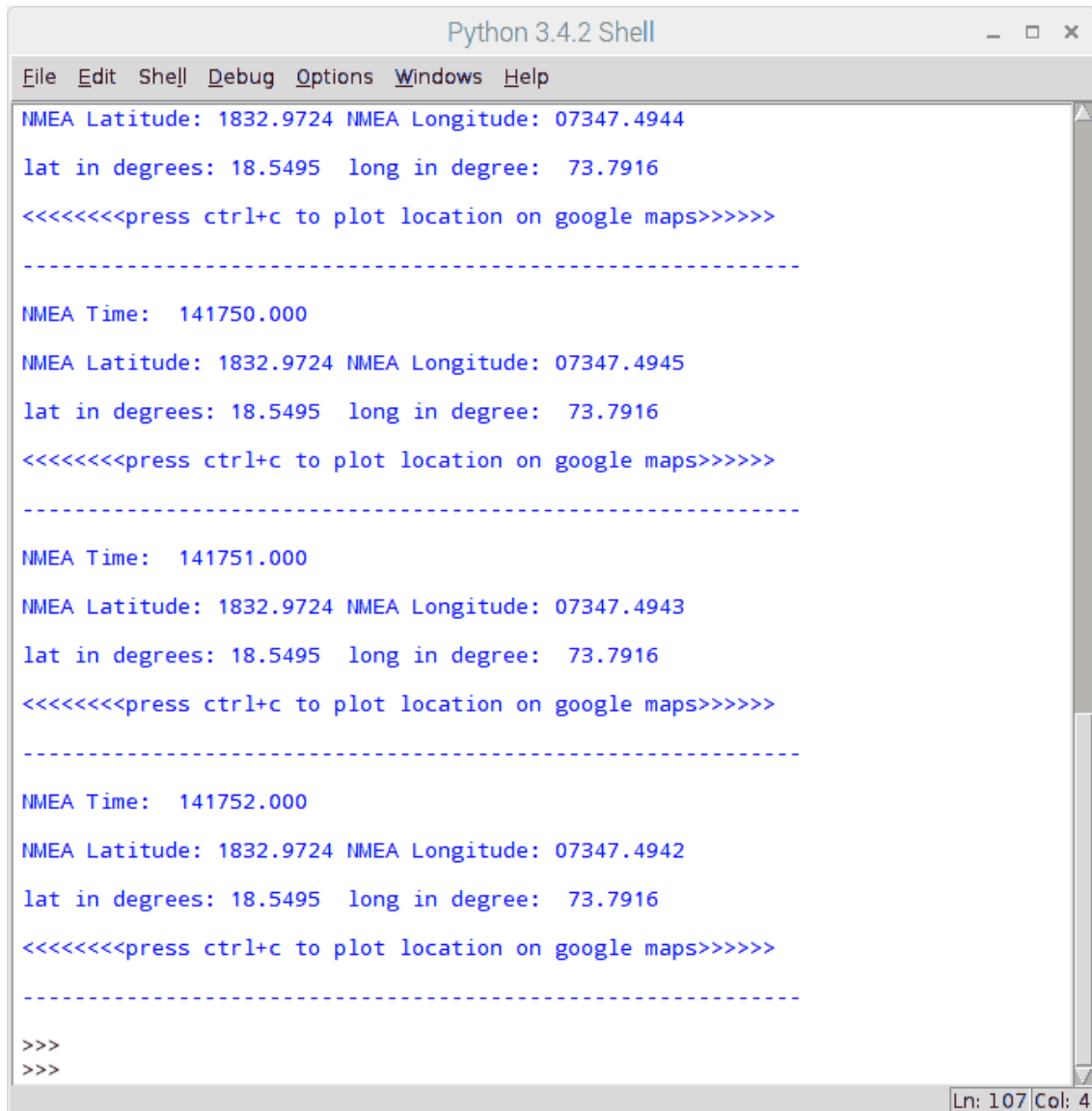
#end of file

```

You can download or copy the code from the following link:

<http://www.electronicwings.com/raspberry-pi/gps-module-interfacing-with-raspberry-pi>

The output of python code is as follows:



```
Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
NMEA Latitude: 1832.9724 NMEA Longitude: 07347.4944
lat in degrees: 18.5495 long in degree: 73.7916
<<<<<<<press ctrl+c to plot location on google maps>>>>>>>
-----
NMEA Time: 141750.000
NMEA Latitude: 1832.9724 NMEA Longitude: 07347.4945
lat in degrees: 18.5495 long in degree: 73.7916
<<<<<<<press ctrl+c to plot location on google maps>>>>>>>
-----
NMEA Time: 141751.000
NMEA Latitude: 1832.9724 NMEA Longitude: 07347.4943
lat in degrees: 18.5495 long in degree: 73.7916
<<<<<<<press ctrl+c to plot location on google maps>>>>>>>
-----
NMEA Time: 141752.000
NMEA Latitude: 1832.9724 NMEA Longitude: 07347.4942
lat in degrees: 18.5495 long in degree: 73.7916
<<<<<<<press ctrl+c to plot location on google maps>>>>>>>
-----
>>>
>>>
```

**Note:** Please ensure that GPS module is visible to open sky or else it will not be able to produce desired output.

# 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

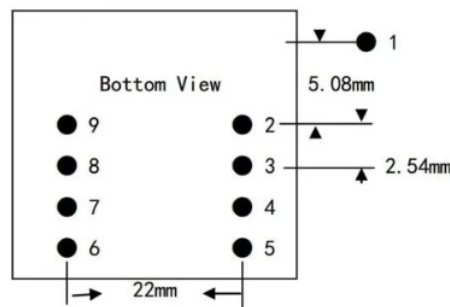
### **RFID Module:**

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 from 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.

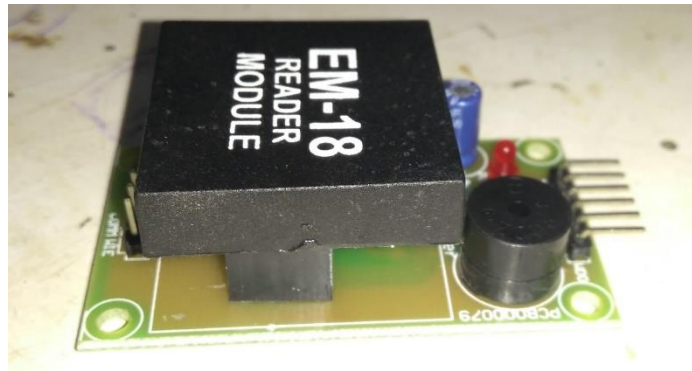
Pinout of the RFID module is given as follows:



EM-18 RFID Reader Module – Bottom View

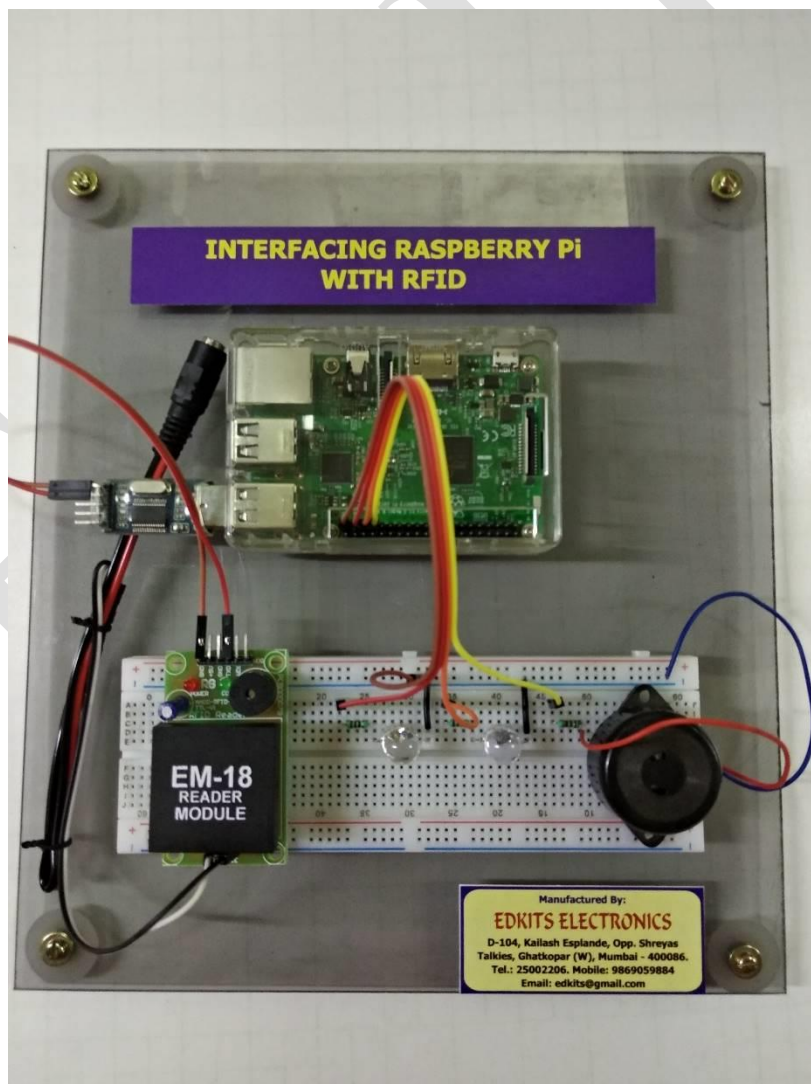
Pin No.	Name	Function
1	VCC	5V
2	GND	Ground
3	BEEP	BEEP and LED
4	ANT	No Use
5	ANT	No Use
6	SEL	HIGH selects RS232, LOW selects WIEGAND
7	TX	UART TX, When RS232 is Selected
8	D1	WIEGAND Data 1
9	D0	WIEGAND Data 0





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



7. 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()
```

# Capturing Images with Raspberry Pi and Pi Camera

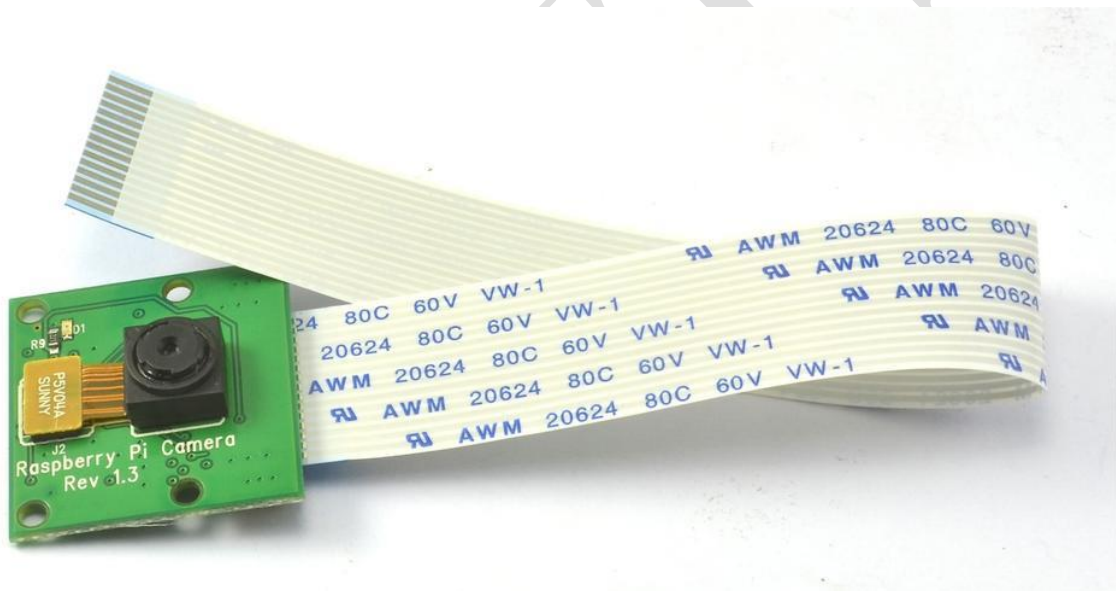
The Camera Module is a great accessory for the Raspberry Pi, allowing users to take still pictures and record video in full HD.

## **Hardware Guide:**

For completing this lesson, you will require the Camera Module along with your initial raspberry pi setup.

### **Camera Module:**

The Raspberry Pi Camera Board plugs directly into the CSI connector on the Raspberry Pi. The camera is supported in the latest version of Raspbian, the Raspberry Pi's preferred operating system.



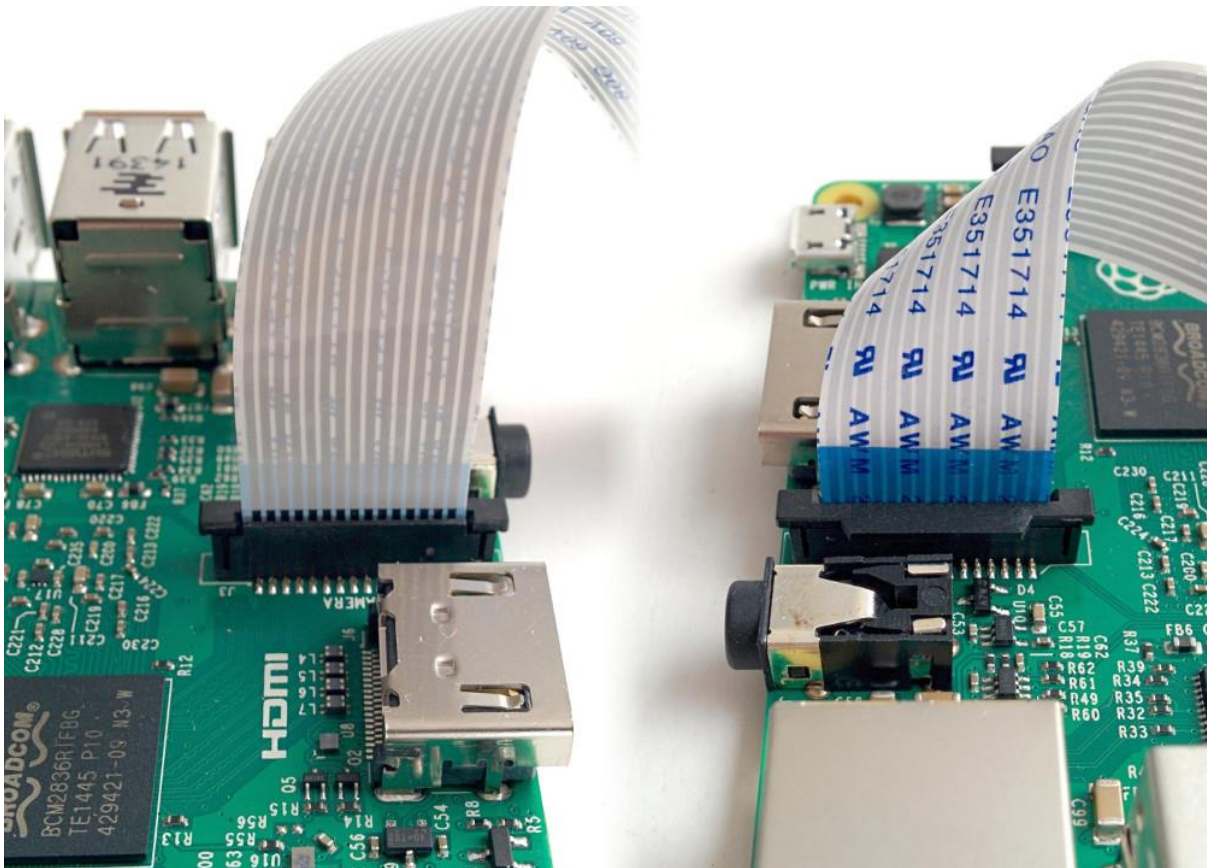
The Raspberry Pi Camera Board Features:

1. Fully Compatible with Both the Model A and Model B Raspberry Pi
2. 5MP Omnivision 5647 Camera Module
3. Still Picture Resolution: 2592 x 1944
4. Video: Supports 1080p @ 30fps, 720p @ 60fps and 640x480p 60/90 Recording
5. 15-pin MIPI Camera Serial Interface – Plugs Directly into the Raspberry Pi Board
6. Size: 20 x 25 x 9mm
7. Weight 3g
8. Fully Compatible with many Raspberry Pi cases

### **Connect the Camera Module:**

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.

1. Locate the camera port and connect the camera:



2. Start up the Pi.
3. Open the Raspberry Pi Configuration Tool from the main menu.
4. Ensure the camera software is enabled. If it's not enabled, enable it and reboot your Pi to begin.

## **Software Guide:**

Now your camera is connected and the software is enabled, you can get started by capturing an image.

You can capture an image by just typing a single line command. Open terminal window and type the command as follows:

```
$ 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').

You can even write a code in Python to capture an image using raspberry pi camera. Open Python3, create a new file and type the code as follows:

**Code:****#Camera Program****# import time and picamera library**

```
from time import sleep
from picamera import PiCamera
```

```
camera = PiCamera()
```

```
camera.resolution = (1280, 720)    # selecting resolution 1280x720 px
```

```
camera.start_preview()
```

**# Camera warm-up time**

```
sleep(2)
```

```
camera.capture('/home/pi/Pictures/newImage.jpg') #capture and save image at specified location
```

```
camera.stop_preview()
```

**#end of code**

Hurray! We have learned how to interface camera with raspberry pi and how to capture image. You can also take videos and do much more things.



# Visitor Monitoring with Raspberry Pi and Pi Camera

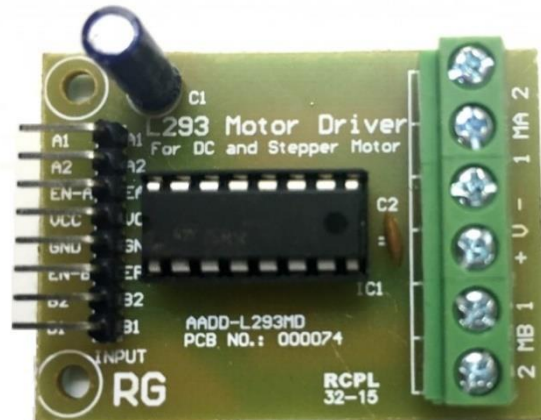
## **Hardware Guide:**

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

1. Camera Module
2. L293D Module
3. DC motor
4. LED
5. Push Button
6. 1K and 10K resistor
7. External 5V power supply

### **L293D Module:**

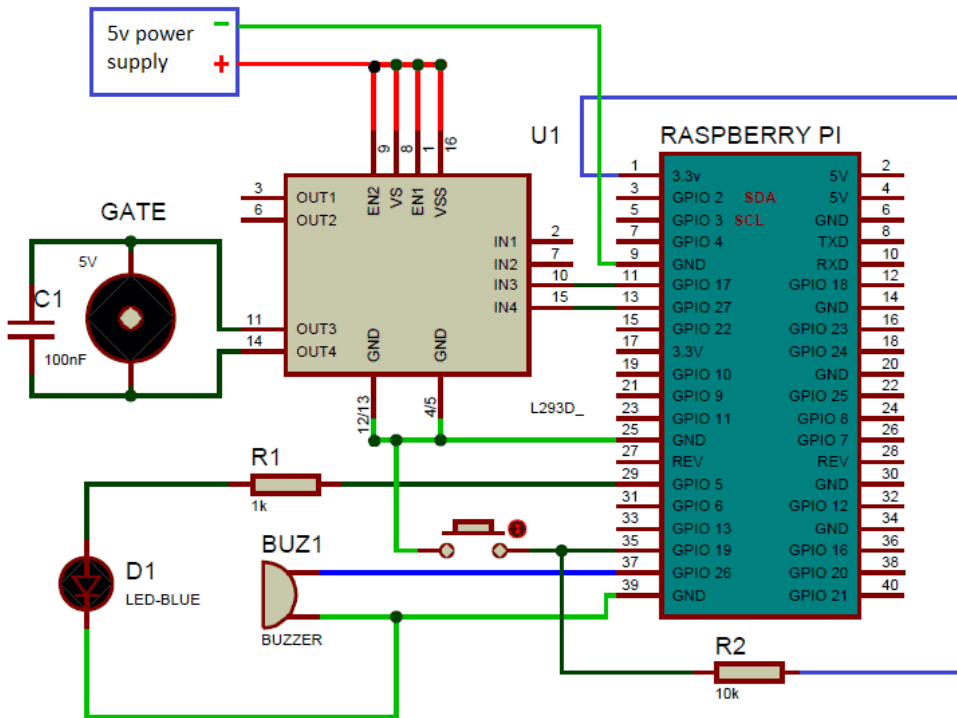
The L293 and L293D devices are quadruple high current half-H drivers. The L293 is designed to provide bidirectional drive currents of up to 1 A at voltages from 4.5 V to 36 V. The L293D is designed to provide bidirectional drive currents of up to 600-mA at voltages from 4.5 V to 36 V. Both devices are designed to drive inductive loads such as relays, solenoids, DC and bipolar stepping motors, as well as other high-current/high-voltage loads in positive supply applications.



## **Wiring up your Circuit:**

1. Connect the Pi camera in the pi camera slot of raspberry pi as explained in the previous lesson
2. Connect the DC motor to raspberry Pi via L293D driver module as shown in the circuit diagram below
3. Connect the Push Button and LED as shown in the circuit diagram
4. You can also interface a buzzer with raspberry pi

## Place PiCamera in Camera slot of Raspberry Pi



## Software Guide:

Don't forget to enable camera as explained in the previous lesson. Now after the whole connection is done write the following code in python 3 IDLE

**Code:**

```
import RPi.GPIO as gpio
```

```
import picamera
```

import time

m11=17

m12=27

led=5

buz=26

button=19

HIGH=1

LOW=0

```
gpio.setwarnings(False)
```

```
gpio.setmode(gpio.BCM)
```

```
gpio.setup(led, gpio.OUT)
```

```
gpio.setup(buz, gpio.OUT)
```

```
gpio.setup(m11, gpio.OUT)
```



```

gpio.setup(m12, gpio.OUT)
gpio.setup(button, gpio.IN)
gpio.output(led , 0)
gpio.output(buz , 0)
gpio.output(m11 , 0)
gpio.output(m12 , 0)
data=""

```

```

def capture_image():
    print("Please Wait..");
    data= time.strftime("%d_%b_%Y\%H:%M:%S")
    camera.start_preview()
    time.sleep(5)
    print (data)
    camera.capture('/home/pi/Desktop/Visitors/%s.jpg'%data)
    camera.stop_preview()
    print("Image Captured Successfully")
    time.sleep(2)

```

```

def gate():
    print("  Welcome ")
    gpio.output(m11, 1)
    gpio.output(m12, 0)
    time.sleep(1.5)
    gpio.output(m11, 0)
    gpio.output(m12, 0)
    time.sleep(3)
    gpio.output(m11, 0)
    gpio.output(m12, 1)
    time.sleep(1.5)
    gpio.output(m11, 0)
    gpio.output(m12, 0)
    print(" Thank You ")
    time.sleep(2)

```

```

print("Visitor Monitoring")
print(" Using RPI ")
time.sleep(3)
camera = picamera.PiCamera()
camera.rotation=180
camera.awb_mode= 'auto'
camera.brightness=55
time.sleep(2)

```

while 1:

```
print(" Please Press Button")
print(" to open the gate ")
gpio.output(led, 1)
if gpio.input(button)==0:
    gpio.output(buz, 1)
    gpio.output(led, 0)
    time.sleep(0.5)
    gpio.output(buz, 0)
    capture_image()
    gate()
time.sleep(0.5)
```

# IOT based Web Controlled Home Automation using Raspberry Pi

## **Hardware Guide:**

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

1. Relay Module
2. An Appliance
3. Connecting wires

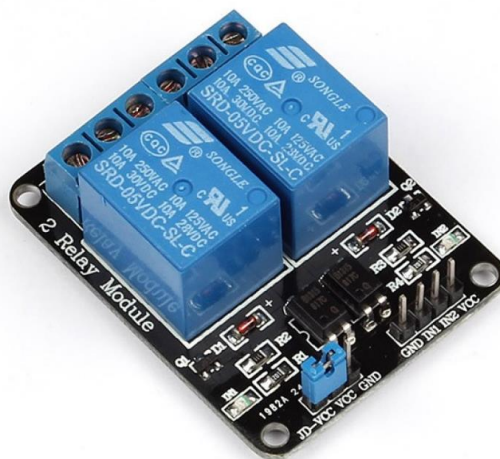
### **Relay Module**

This is ready breakout board with two 5V activated relays, and the signal is opto-isolated, it is protected. Easy to use and suitable for beginner, and of course student.

If you need to control AC or high current, high voltage load, this will be the perfect board.

Features:

1. 5V 2 channel relay interface board
2. Maximum Current Rating: 10A
3. Maximum Voltage Rating: AC 250V / DC 30V
4. Can control various appliances with large current and high voltage
5. Can be controlled directly by microcontroller such as Arduino, 8051, AVR, PIC, DSP and ARM
6. Build in 2 units of Opto-Isolator IC 817C.



## **Software Guide:**

Now before wiring up your circuit, you need to install WebIOPi framework which will help us handle communication from the webpage to the raspberry pi

### **Installation of WebIOPi:**

To update the raspberry Pi below commands and then reboot the RPi:

1. `sudo apt-get update`
2. `sudo apt-get upgrade`
3. `sudo reboot`

Now download the latest version of WebIOPi from the following link on your raspberry pi:

<http://webiopi.trouch.com/DOWNLOADS.html>

Now move to the downloads folder using the command: `cd /home/pi/Downloads/`

Now you need to extract the file and adapt x.y.z with the version you download:

1. `tar xvzf WebIOPi-x.y.z.tar.gz`
2. `cd WebIOPi-x.y.z`

At this point before running the setup, we need to install a patch as this version of the WebIOPi does not work with the raspberry pi 3 which I am using and I couldn't find a version of the WebIOPi that works expressly with the Pi 3.

Below commands are used to install patch while still in the WebIOPi directory, run:

1. `wget https://raw.githubusercontent.com/doublebind/raspi/master/webiopi-pi2bplus.patch`
2. `patch -p1 -i webiopi-pi2bplus.patch`

Then we can run the setup installation for the WebIOPi using:

1. `sudo ./setup.sh`

Keep saying yes if asked to install any dependencies during setup installation. When done, reboot your pi:

1. `sudo reboot`

### **Test WebIOPi Installation:**

For instance, to start with verbose output and the default config file:

1. `sudo webiopi -d -c /etc/webiopi/config`

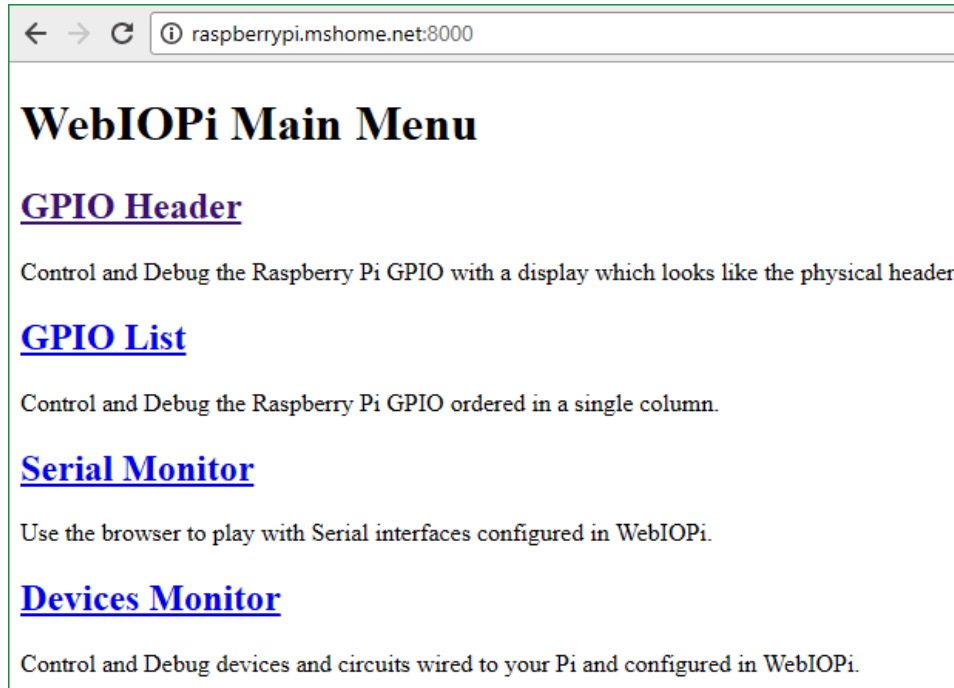
### **NOTE:**

You can also start/stop the background service, the configuration will be loaded from `/etc/webiopi/config`.

1. `sudo /etc/init.d/webiopi start`
2. `sudo /etc/init.d/webiopi stop`

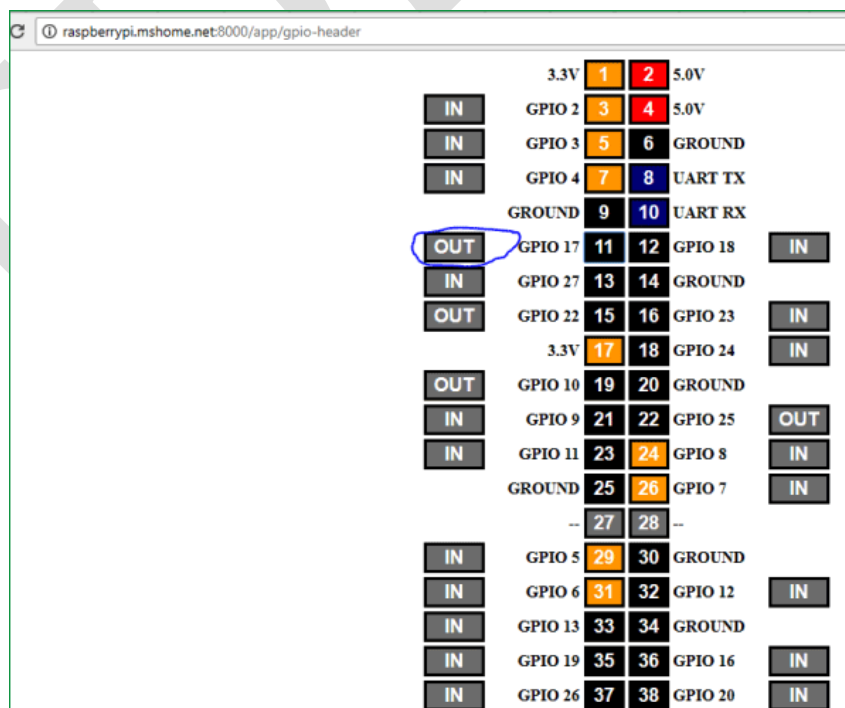
After issuing the command above on the pi, point the web browser of your computer connected to the raspberry pi to `http://theipi'sIPAddress:8000`. The system will prompt you for username and password. Default username is 'webiopi' and password is 'raspberrypi'.

After the login, look around, and then click on the GPIO header link.

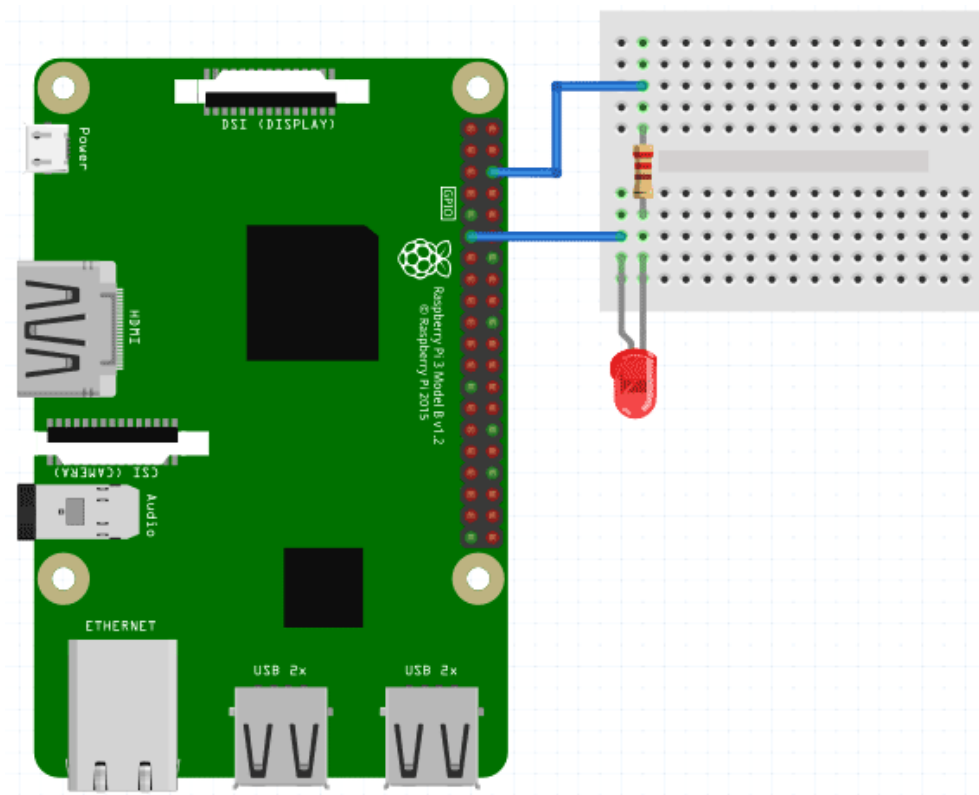


Wiring up your circuit:

For this test, we will be connecting an LED to GPIO 17, so go on and set GPIO 17 as an output.



With this done, connect the led to your raspberry pi as shown in the schematics below.



After the connection, go back to the webpage and click the pin 11 button to turn on or off the LED. This way we can control the Raspberry Pi GPIO using WebIOPi.

After the test, if everything worked as described, then we can go back to the terminal and stop the program with CTRL + C.

Once everything is working properly you can connect the relay module to raspberry pi as follows:

1. Connect the positive of external 5v supply to the VCC pin of relay module.
2. Connect the negative of external 5v supply to the GND pin of relay module.
3. Also connect the GND pin of relay module to any GND pin of raspberry pi
4. Lastly connect IN1 and IN2 pins of relay module to any GPIO pins of raspberry which you need to use.

**Note:**

You can get more information on WebIOPi framework on the following link:

<http://webiopi.trouch.com/>

# Installing Windows 10 IOT Core on Raspberry Pi

Windows 10 IoT is a member of the Windows 10 family that brings enterprise-class power, security and manageability to the Internet of Things. It leverages Windows' embedded experience, ecosystem and cloud connectivity, allowing organizations to create their Internet of Things with secure devices that can be quickly provisioned, easily managed, and seamlessly connected to an overall cloud strategy.

## **Hardware Guide:**

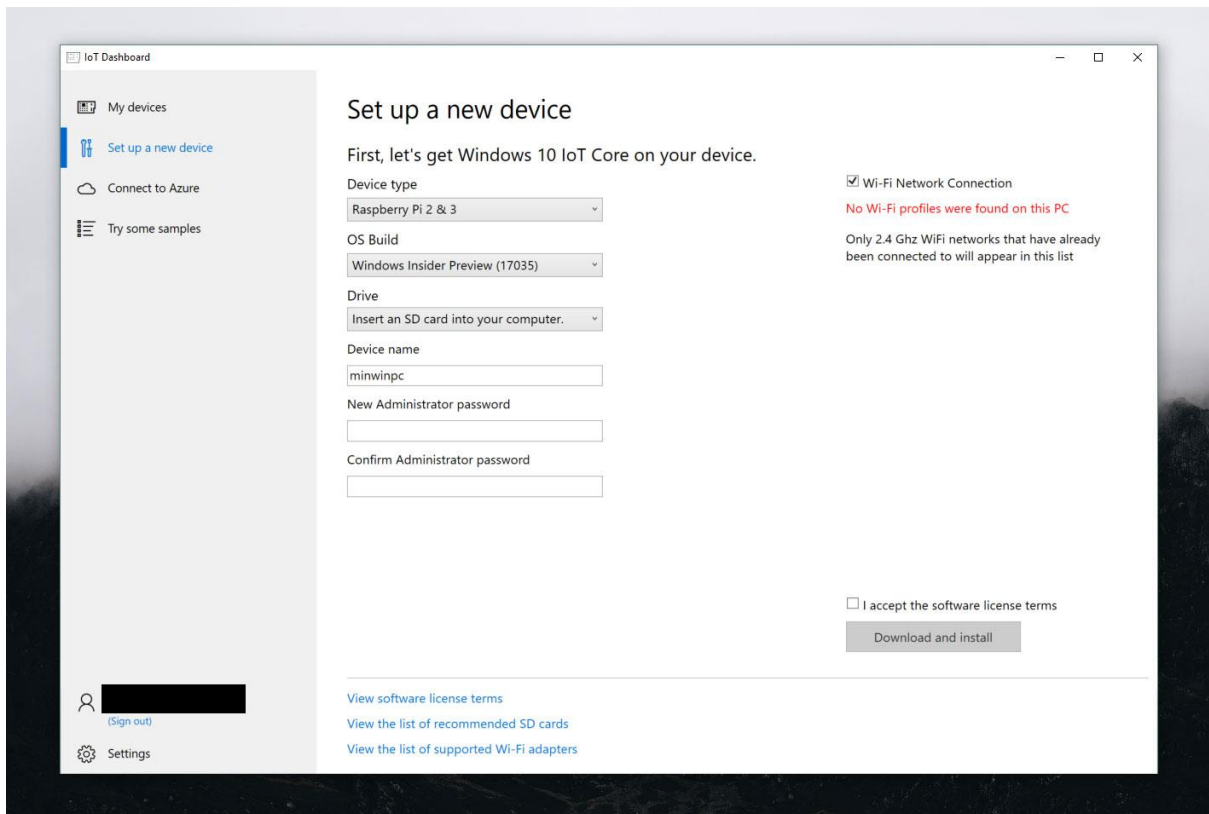
For getting started with windows 10 IOT, you will require the following hardware

1. Raspberry Pi 3
2. 16 GB Micro SD Card – class 10
3. Display
4. Keyboard
5. Mouse
6. Windows 10 PC
7. Card Reader

## **Installation Guide:**

Use this tutorial to get yourself comfortable with Windows 10 IoT quickly. You'll learn how to flash a Windows 10 IoT Core image onto a device and how to deploy an app from your device.

1. Download the Windows 10 IoT Core Dashboard from <https://developer.microsoft.com/en-us/windows/iot/Downloads>
2. Once downloaded, open the Dashboard and click on set up a new device and insert a SD card into your computer.
3. Fill out all of the fields as indicated.
4. Accept the software license terms and click Download and install. You'll see that Windows 10 IoT Core is now flashing onto your device.



## **Connecting to a network**

### **Wired connection**

If your device comes with an Ethernet port or USB Ethernet adapter support to enable a wired connection, attach an Ethernet cable to connect it to your network.

### **Wireless connection**

If your device supports Wi-Fi connectivity and you've connected a display to it, you'll need to:

1. Go into your default application and click the settings button next to the clock.
2. On the settings page, select Network and Wi-Fi.
3. Your device will begin scanning for wireless networks.
4. Once your network appears in this list, select it and click Connect.

If you haven't connected and display and would like to connect via Wi-Fi, you'll need to:

1. Go to the IoT Dashboard and click on My Devices.
2. Find your unconfigured board from the list. Its name will begin with "AJ\_"... (e.g. AJ\_58EA6C68). If you don't see your board appear after a few minutes, try rebooting your board.
3. Click on Configure Device and enter your network credentials. This will connect your board to the network.



# Building Google Assistant with Raspberry Pi

## **Introduction to the Google Assistant Library**

The Google Assistant Library for Python is a turnkey solution for anyone who wants to quickly integrate the Assistant into a prototype device. The library is written in Python and is supported on popular prototyping devices such as the Raspberry Pi 3.

**Note:** Refer to the Compatibility and feature support table to see the differences between the Google Assistant Library and the service.

## **Get started**

Once you have your hardware, read on to learn how to get the Google Assistant running on it!

## **Embed the Google Assistant**

This section gets the Google Assistant Library working on your device:

1. Set Up Hardware and Network Access
2. Configure and Test the Audio
3. Configure a Developer Project and Account Settings
4. Register the Device Model
5. Install the SDK and Sample Code
6. Run the Sample Code

## **Set Up Hardware and Network Access**

Before you begin, you'll need the following components:

1. Raspberry Pi 3 Model B and power supply
2. USB microphone
3. Speaker
4. SD Card with Raspbian OS on it
5. You may also want to have a USB keyboard, USB mouse, and a monitor with an HDMI cable.  
These simplify initial hardware setup

You'll now set up the hardware and configure network access.

## Connect the hardware and configure network access

1. Connect the microphone and speaker to the Raspberry Pi.
2. Insert the SD card into the Raspberry Pi (with NOOBS or Raspbian with Desktop already loaded).
3. Connect a USB keyboard, USB mouse and HDMI monitor to your Raspberry Pi. If you don't have these, you can always connect to the Pi remotely.
4. Plug in an ethernet cable or connect to a Wi-Fi network

After you configure access, you can connect to the Raspberry Pi via SSH (optional).

Check that the date and time are set correctly on the device using command: date

## Configure and Test the Audio

Before running the sample, you must configure the audio system on the Raspberry Pi.

1. Find your recording and playback devices.
  - a. Locate your USB microphone in the list of capture hardware devices. Write down the card number and device number: arecord -l
  - b. Locate your speaker in the list of playback hardware devices. Write down the card number and device number. Note that the 3.5mm-jack is typically labeled Analog or bcm2835 ALSA (not bcm2835 IEC958/HDMI): aplay -l
2. Create a new file named .asoundrc in the home directory (/home/pi). Make sure it has the right slave definitions for microphone and speaker; use the configuration below but replace <card number> and <device number> with the numbers you wrote down in the previous step. Do this for both pcm.mic and pcm.speaker.

```
pcm.!default {
    type asym
    capture.pcm "mic"
    playback.pcm "speaker"
}
pcm.mic {
    type plug
    slave {
        pcm "hw:<card number>,<device number>"
    }
}
pcm.speaker {
    type plug
    slave {
        pcm "hw:<card number>,<device number>"
    }
}
```

3. Verify that recording and playback work:
  - a. Adjust the playback volume: alsamixer  
Press the up arrow key to set the playback volume level to around 70.

- b. Play a test sound (this will be a person speaking). Press Ctrl+C when done. If you don't hear anything when you run this, check your speaker connection.  
`speaker-test -t wav`
- c. Record a short audio clip.  
`arecord --format=S16_LE --duration=5 --rate=16000 --file-type=raw out.raw`
- d. Check the recording by replaying it. If you don't hear anything, you may need to check the recording volume in alsamixer.  
`aplay --format=S16_LE --rate=16000 out.raw`

If recording and playback are working, then you are done configuring audio. If not, check that the microphone and speaker are properly connected. If this is not the issue, then try a different microphone or speaker.

Note that if you have both an HDMI monitor and a 3.5mm jack speaker connected, you can play audio out of either one. Run the following command: `sudo raspi-config`

Go to Advanced options > Audio and select the desired output device.

## **Configure a Developer Project and Account Settings**

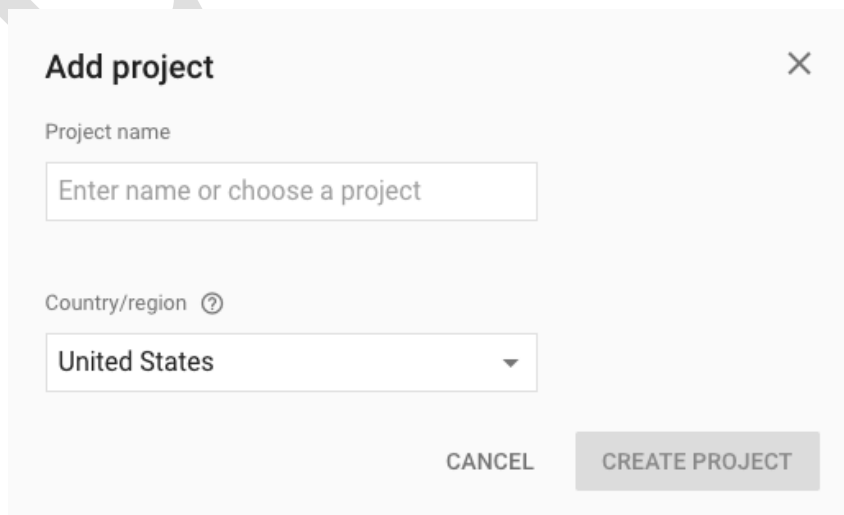
### **Configure an Actions Console project**

A Google Cloud Platform project, managed by the Actions Console, gives your device access to the Google Assistant API. The project tracks quota usage and gives you valuable metrics for the requests made from your device.

To enable access to the Google Assistant API, do the following:

1. Open the Actions Console.
2. Click on Add/import project.
3. To create a new project, type a name in the Project name box and click CREATE PROJECT.

If you already have an existing Google Cloud Platform project, you can select that project and import it instead.



Keep this browser tab open.

4. Enable the Google Assistant API on the project you selected (see the Terms of Service). You need to do this in the Cloud Platform Console.
5. Click Enable.

### **Set activity controls for your account**

In order to use the Google Assistant, you must share certain activity data with Google. The Google Assistant needs this data to function properly; this is not specific to the SDK.

Open the Activity Controls page for the Google account that you want to use with the Assistant. You can use any Google account, it does not need to be your developer account.

Ensure the following toggle switches are enabled (blue):

1. Web & App Activity
2. In addition, be sure to select the Include Chrome browsing history and activity from websites and apps that use Google services checkbox.
3. Device Information
4. Voice & Audio Activity

### **Register the Device Model**

In order for the Google Assistant to respond to commands appropriate to your device and the given context, the Assistant needs information about your particular device. You provide this information, which includes fields like device type and manufacturer, as a device model. You can think of this model as a general class of device - like a light, speaker, or toy robot.

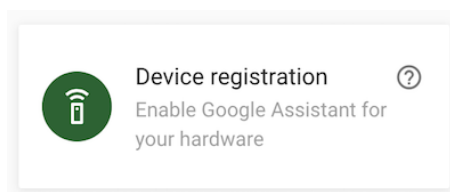
This information is then accessible to the Google Assistant and is associated with your Actions Console project. No other projects have access to your model and device information.

### **Use the registration UI**

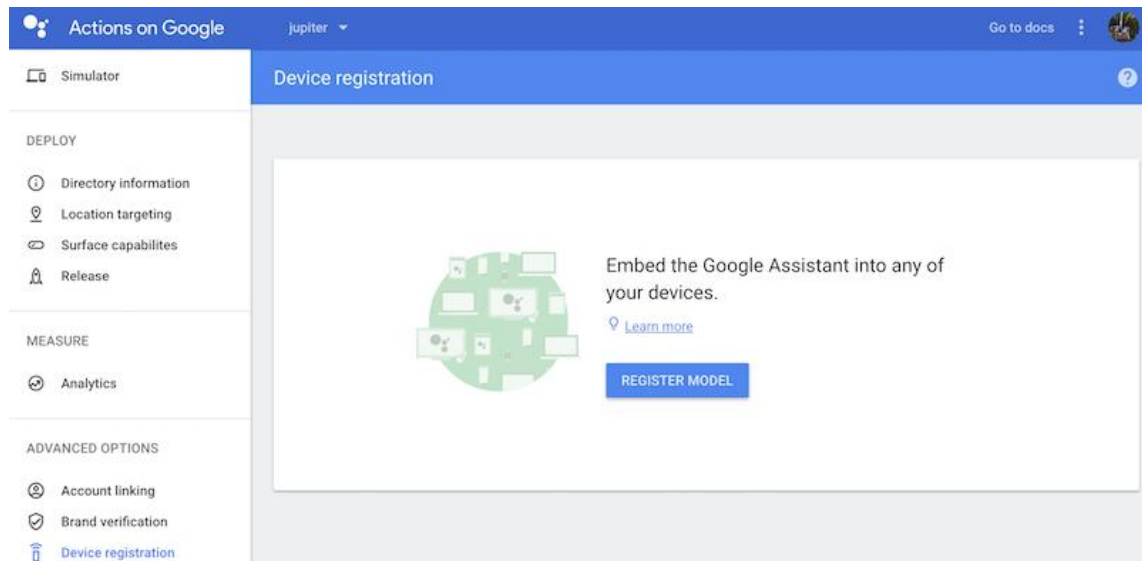
Use the registration UI in the Actions Console to register a device model.

1. Open the Actions Console. You may already have this tab open from a previous step.
2. Select the project you created or imported previously.

If you created a new project, click the Device registration box near the bottom of the page. If you imported a previously-created project, this box will not be displayed; select the Device registration tab (under ADVANCED OPTIONS) from the left navbar.



3. Click the REGISTER MODEL button.



### Create model

1. Fill out all of the fields for your device. Select any device type, such as Light. See the device model JSON reference for more information on these fields.
2. When you are finished, click REGISTER MODEL.

### Download credentials

The `client_secret_<client-id>.json` file must be located on the device. This file contains a client ID and client secret, but no access token. Later, you will run an authorization tool and reference this file in order to authorize the Google Assistant SDK sample to make Google Assistant queries (see the OAuth 2.0 documentation for more information). Do not rename this file.

Download this file and transfer it to the device. Click NEXT.

Make sure this file is located in /home/pi. If you want to upload the file to the device, do the following:

1. Open a new terminal window. Run the following command in this new terminal:  
`scp ~/Downloads/client_secret_client-id.json pi@raspberrypi-ip-address:/home/pi/`
2. Close this terminal window.

The screenshot shows the 'Register model' dialog box with three steps: 'Create model' (completed), 'Download credentials' (current step), and 'Specify traits'. A warning message states: 'Keep this file secure, do not upload it to public repositories like GitHub. It's possession allows client applications to make call to the Google Assistant Service and will consume your project quota (see the OAuth 2.0 documentation for more information)'. Step 1 is 'Download OAuth 2.0 credentials'. Step 2 instructions are: 'Place the client\_secret\_\*.json file in folder where you're running the Assistant SDK. To copy over SSH to a remote device, run the following command from your current computer: scp ~/Downloads/client\_secret\_\*.json <username>@<device-ip-address>: </path/to/assistant-sdk/project> password: password-for-device'. A 'NEXT' button is at the bottom right.

### Specify traits

Later, you will specify the different abilities that your device supports on this screen. But for now, click the SKIP button.

The screenshot shows the 'Register model' dialog box with three steps: 'Create model' (completed), 'Download credentials' (completed), and 'Specify traits' (current step). It features a search bar and a list of traits: 'All 7 traits', 'Brightness' (with a description and a 'View details' link), and 'ColorSpectrum' (with a description). At the bottom, there are 'SKIP' and 'SAVE TRAITS' buttons.

## Edit the model

If you need to edit the model, click its row in the list. Make sure to click SAVE after any edits.

REGISTER MODEL				
Product name	Manufacturer name	Model ID	Device Type	Last updated time
Assistant SDK light	Assistant SDK developer	jupiter-86b58-assistant-sdk-light-xw2x1y	action.devices.types.LIGHT	Jul 13, 2018, 02:20 PM

If you need to download the credentials file again, click the ellipses. You can delete the model from this menu as well.

Linked device models

← Assistant SDK light

SAVE

Download OAuth 2.0 credentials

Remove model

Product name: Assistant SDK light

Manufacturer name: Assistant SDK developer

Model ID: jupiter-86b58-assistant-sdk-light-xw2x1y

Device type: Light

## Alternative ways to register

You can also use the registration tool (included with the Google Assistant SDK samples) or the REST API to register a device model.

You must be an Owner or Editor of a given Actions Console project to register models for it. Add these roles for other users in the Cloud Platform Console—see the IAM documentation.

## Install the SDK and Sample Code

Follow these instructions to install the SDK and sample code on your device. Run all of the commands on this page in a terminal on the device (either directly or via an SSH connection).

### Configure a new Python virtual environment

Use a Python virtual environment to isolate the SDK and its dependencies from the system Python packages.

**Note:** For the Raspberry Pi, run the following commands from the `/home/pi` directory.

For Python 3:

1. `sudo apt-get update`
2. `sudo apt-get install python3-dev python3-venv`
3. `python3 -m venv env`
4. `env/bin/python -m pip install --upgrade pip setuptools wheel`
5. `source env/bin/activate`

For Python 2.7:

1. `sudo apt-get update`
2. `sudo apt-get install python-dev python-virtualenv`
3. `virtualenv env --no-site-packages`
4. `env/bin/python -m pip install --upgrade pip setuptools wheel`
5. `source env/bin/activate`

## Get the package

The Google Assistant SDK package contains all the code required to get the Google Assistant running on the device, including the sample code.

Install the package's system dependencies:

```
sudo apt-get install portaudio19-dev libffi-dev libssl-dev libmpg123-dev
```

Use pip to install the latest version of the Python package in the virtual environment:

1. `python -m pip install --upgrade google-assistant-library`
2. `python -m pip install --upgrade google-assistant-sdk[samples]`

## Generate credentials

1. Install or update the authorization tool:

```
python -m pip install --upgrade google-auth-oauthlib[tool]
```

2. Generate credentials to be able to run the sample code and tools. Reference the JSON file you downloaded in a previous step; you may need to copy it to the device. Do not rename this file.

```
google-oauthlib-tool --scope https://www.googleapis.com/auth/assistant-sdk-prototype \
--scope https://www.googleapis.com/auth/gcm \
--save --headless --client-secrets /path/to/client_secret_client-id.json
```

You should see a URL displayed in the terminal:

Please visit this URL to authorize this application: <https://...>

3. Copy the URL and paste it into a browser (this can be done on any machine). The page will ask you to sign in to your Google account. Sign into the Google account that created the developer project in the previous step.
4. After you approve the permission request from the API, a code will appear in your browser, such as "4/XXXX". Copy and paste this code into the terminal:

Enter the authorization code:

If authorization was successful, you will see a response similar to the following:

```
credentials saved: /path/to/.config/google-oauthlib-tool/credentials.json
```

If instead you see `InvalidGrantError`, then an invalid code was entered. Try again, taking care to copy and paste the entire code.



**Note:** The authorization tool creates a new `credentials.json` file in a hidden `.config` directory on the device. This file contains an access token that is used to call the Google Assistant API.

## **Run the Sample Code**

At this point, you are ready to run the sample and make a query.

In the following command:

1. Replace `my-dev-project` with the Google Cloud Platform project ID for the Actions Console project you created. To find the project ID in the Actions Console, select the project, click the gear icon, and select Project settings.
2. Replace `my-model` with the name of the model you created in the previous step.

(env) \$ `googlesamples-assistant-hotword --project id my-dev-project --device_model id my-model`

Say Ok Google or Hey Google, followed by your query. You can try some of the following:

- Who am I?
- Listen to This American Life podcast.
- What is the weather in San Francisco?

You can find the entire details on the link:

<https://developers.google.com/assistant/sdk/guides/library/python/>

# 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:

2. `sudo systemctl stop dnsmasq`
3. `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 dhcpd configuration file with:

- `sudo nano /etc/dhcpd.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 dhcpd daemon and set up the new wlan0 configuration:

- `sudo service dhcpd 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 be 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/dhcpd.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.

You can refer the following link for details:

<https://www.raspberrypi.org/documentation/configuration/wireless/access-point.md>

# Oscilloscope using Raspberry Pi

The oscilloscope is an electronic test instrument that allows the visualization and observation of varying signal voltages, usually as a two dimensional plot with one or more signals plotted against time. Today's project 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

## **Hardware Guide:**

To build this project, the following components/part are required:

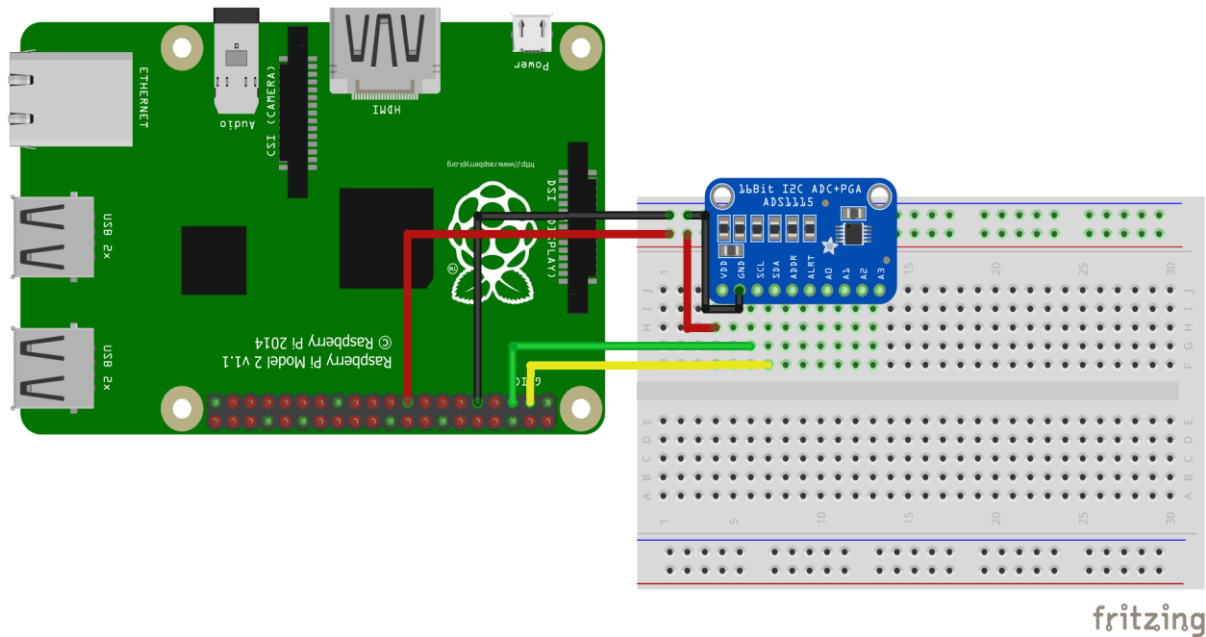
1. Raspberry pi 2 (or any other model)
2. 8 or 16GB SD Card
3. LAN/Ethernet Cable
4. Power Supply or USB cable
5. ADS1115 ADC
6. 10k or 1k resistor
7. Jumper wires
8. Breadboard
9. Monitor or any other way of seeing the pi's Desktop(VNC inclusive)

## **Wiring up your Circuit:**

To convert the analog input signals to digital signals which can be visualized with the Raspberry Pi, we will be using the ADS1115 ADC chip. This chip becomes important because the Raspberry Pi, unlike Arduino and most micro-controllers, does not have an on-board analog to digital converter(ADC). While we could have used any raspberry pi compatible ADC chip, I prefer this chip due to its high resolution(16bits) and its well documented datasheet and use instructions by Adafruit. You can also check our Raspberry Pi ADC tutorial to learn more about it.

The ADC is an I2C based device and should be connected to the Raspberry Pi as shown in the schematics below.

For clarity, the pin connection between the two components is also described below.



ADS1115 and Raspberry Pi Connections:

1. VDD – 3.3v
2. GND – GND
3. SDA – SDA
4. SCL – SCL

## Software Guide:

### **Install Dependencies for Raspberry Pi Oscilloscope:**

Before we start writing the python script to pull data from the ADC and plot it on a live graph, we need to enable the I2C communication interface of the raspberry pi and install the software requirements that were mentioned earlier. This will be done in below steps so its easy to follow:

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

```
sudo raspi-config
```

When the configuration panels open, select interface options, select I2C and click enable.

#### **Step 2: Update the Raspberry pi**

The first thing I do before starting any project is updating the Pi. Through this, I am sure every thing on the OS is up to date and I won't experience compatibility issue with any latest software I choose to install on the Pi. To do this, run below two commands:

```
sudo apt-get update
sudo apt-get upgrade
```



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

With the update done, we are now ready to install the dependencies starting with the Adafruit python module for the ADS115 chip. Ensure you are in the Raspberry Pi home directory by running;

```
cd ~
```

then install the build-essentials by running;

```
sudo apt-get install build-essential python-dev python-smbus git
```

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

```
git clone https://github.com/adafruit/Adafruit_Python_ADS1x15.git
```

Change into the cloned file's directory and run the setup file;

```
cd Adafruit_Python_ADS1x15
sudo python setup.py install
```

### Step 4: Test the library and I2C communication.

Before we proceed with the rest of the project, it is important to test the library and ensure the ADC can communicate with the raspberry pi over I2C. To do this we will use an example script that comes with the library.

While still in the Adafruit\_Python\_ADS1x15 folder, change directory to the examples directory by running;

```
cd examples
```

Next, run the sampletest.py example which displays the value of the four channels on the ADC in a tabular form.

Run the example using:

```
python sampletest.py
```

If the I2C module is enabled and connections good, you should see the data as shown in the image below.

```

pi@raspberrypi:~ $ cd Adafruit_Python_ADS1x15
pi@raspberrypi:~/Adafruit_Python_ADS1x15 $ cd examples
pi@raspberrypi:~/Adafruit_Python_ADS1x15/examples $ python simpletest.py
Reading ADS1x15 values, press Ctrl-C to quit...
| 0 | 1 | 2 | 3 |
|---|
| 4699 | 4584 | 4625 | 4665 |
| 4583 | 4587 | 4601 | 4614 |
| 4563 | 4604 | 4600 | 4612 |
| 4601 | 4630 | 4609 | 4585 |
| 4614 | 4606 | 4577 | 4636 |
| 4616 | 4580 | 4621 | 4630 |
| 4566 | 4630 | 4618 | 4631 |
| 4614 | 4619 | 4615 | 4620 |
| 4577 | 4622 | 4609 | 4625 |
| 4624 | 4615 | 4626 | 4648 |
| 4636 | 4660 | 4656 | 4607 |
| 4609 | 4616 | 4629 | 4651 |

```

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

To visualize the data we need to install the matplotlib module which is used to plot all kind of graphs in python. This can be done by running;

```
sudo apt-get install python-matplotlib
```

### Step6: Install the Drawnow python module

Lastly, we need to install the drawnow python module. This module helps us provide live updates to the data plot.

We will be installing drawnow via the python package installer; pip, so we need to ensure it is installed. This can be done by running;

```
sudo apt-get install python-pip
```

We can then use pip to install the drawnow package by running:

```
sudo pip install drawnow
```

### Python Code for Raspberry Pi Oscilloscope:

The python code for this Pi Oscilloscope is fairly simple especially if you are familiar with the python matplotlib module. Before showing us the whole code, I will try to break it into part and explain what each part of the code is doing so you can have enough knowledge to extend the code to do more stuffs.

At this stage it is important to switch to a monitor or use the VNC viewer, anything through which you can see your Raspberry Pi's desktop, as the graph being plotted won't show on the terminal.

With the monitor as the interface open a new python file. You can call it any name you want, but I will call it scope.py.

```
sudo nano scope.py
```

now copy the code give below in the file scope.py and then run it to get the desired output.

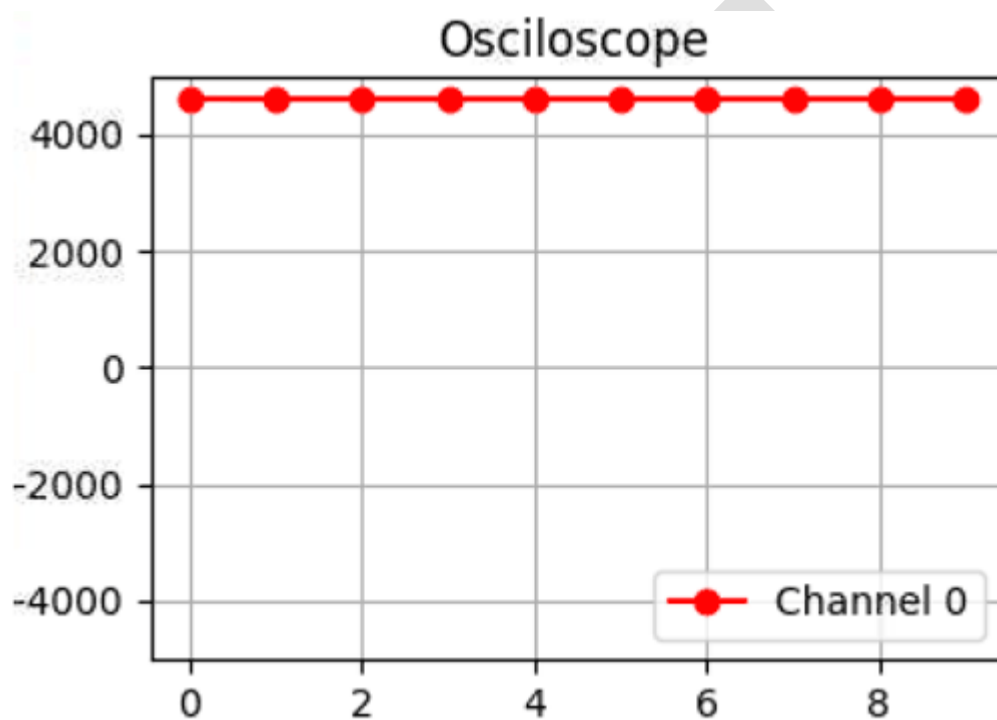
**Code:**

```
import time
import matplotlib.pyplot as plt
#import numpy
from drawnow import *
# Import the ADS1x15 module.
import Adafruit_ADS1x15
# Create an ADS1115 ADC (16-bit) instance.
adc = Adafruit_ADS1x15.ADS1115()

GAIN = 1
val = [ ]
cnt = 0
plt.ion()
# Start continuous ADC conversions on channel 0 using the previous gain value.
adc.start_adc(0, gain=GAIN)
print('Reading ADS1x15 channel 0')
#create the figure function
def makeFig():
    plt.ylim(-5000,5000)
    plt.title('Oscilloscope')
    plt.grid(True)
    plt.ylabel('ADC outputs')
    plt.plot(val, 'ro-', label='Channel 0')
    plt.legend(loc='lower right')
while (True):
    # Read the last ADC conversion value and print it out.
    value = adc.get_last_result()
    print('Channel 0: {}'.format(value))
    # Sleep for half a second.
    time.sleep(0.5)
    val.append(int(value))
    drawnow(makeFig)
    plt.pause(.000001)
    cnt = cnt+1
    if(cnt>50):
        val.pop(0)
```

Output:

```
pi@raspberrypi:~ $ sudo nano scope.py
pi@raspberrypi:~ $ sudo python scope.py
Reading ADS1x15 channel 0
Channel 0: 4618
/usr/lib/python2.7/dist-packages/matplotlib/backend
plotlibDeprecationWarning: Using default event loop
cific to this GUI is implemented
    warnings.warn(str, mplDeprecation)
Channel 0: 4615
Channel 0: 4616
Channel 0: 4615
Channel 0: 4614
Channel 0: 4613
Channel 0: 4614
```



Note: You can find the entire details at <https://circuitdigest.com/microcontroller-projects/raspberry-pi-based-oscilloscope>

# LED Matrix Module interfacing with Raspberry Pi

In this lesson, we will be interfacing 8x8 LED matrix Module with raspberry pi. Since you are now familiar with the GPIO pins, it will be a fun to know more about the python coding and installing libraries to do more advance things.

## **Hardware Guide:**

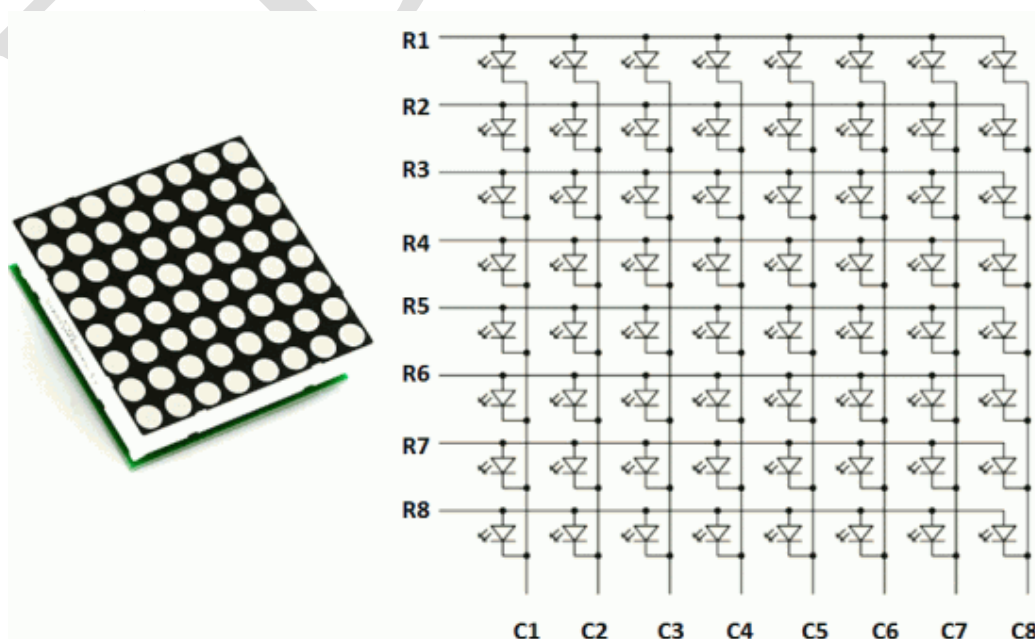
For completing this lesson, you will require the following things along with your initial raspberry pi setup

1. 8x8 LED matrix module
2. 7219 driver board
3. Connecting wires

## **8x8 LED matrix Module:**

A LED-Matrix Display is a display device which contains light emitting diodes aligned in the form of matrix. This LED matrix displays are used in applications where Symbol, Graphic, Characters, Alphabets, Numerals are needed to be displayed together in static as well as Scrolling motion. LED Matrix Display is manufactured in various dimensions like 5x7,8x8,16x8,128x16, 128x32 and 128x64 where the numbers represent LED's in rows and columns, respectively. Also, these displays come in different colours such as Red, Green, Yellow, Blue, Orange, White.

In LED matrix display, multiple LED's are wired together in rows and columns, to minimize the number of pins required to drive them. The matrix pattern is made either in row anode-column cathode or row cathode-column anode pattern. In row anode-column cathode pattern, the entire row is anode while all columns serve as cathode which is shown below and it is vice-versa in row cathode-column anode pattern.



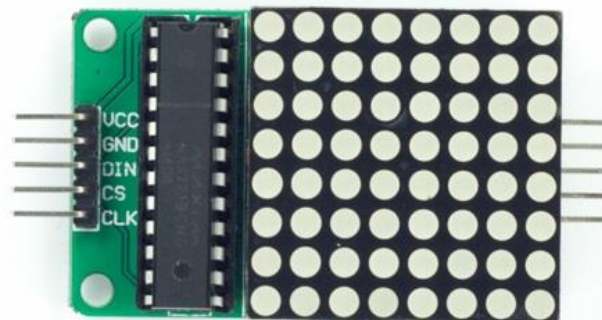
### 7219 Driver board:

Before interfacing LED matrix with raspberry pi, we need to connect the Max7219 IC which is an LED driver to the LED matrix display. The reason behind using this LED driver is that it drives the 64 LEDs simultaneously which in turn reduces the number of wires so that the user will find it easy to connect the display to the raspberry pi.

The MAX7219 has four wire SPI interface (we need only these four wires to interface it to the raspberry pi):

1. Din - MOSI - Master Output Serial Input.
2. Chip select - Load (CS) - active low Chip select.
3. Clock - SCK
4. Ground.

And of course VCC (5V) is required.



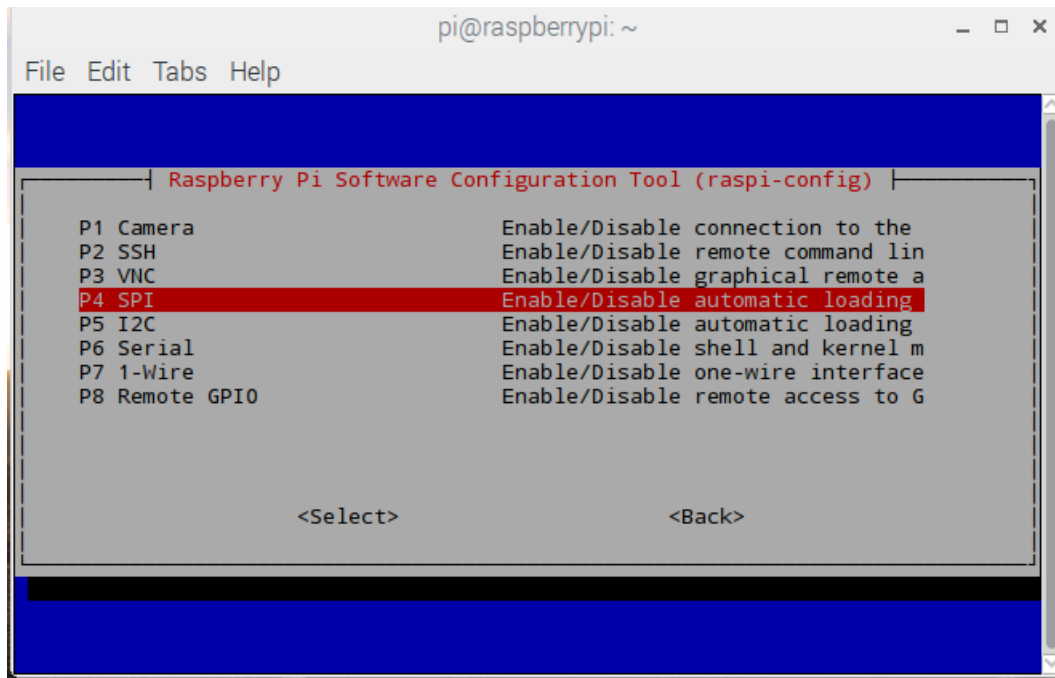
### Software Guide:

Here we will be writing our code in python. But before writing our code we need to enable SPI and we need to install the library for driving the LED Matrix Module using our raspberry pi.

#### **Pre-requisites:**

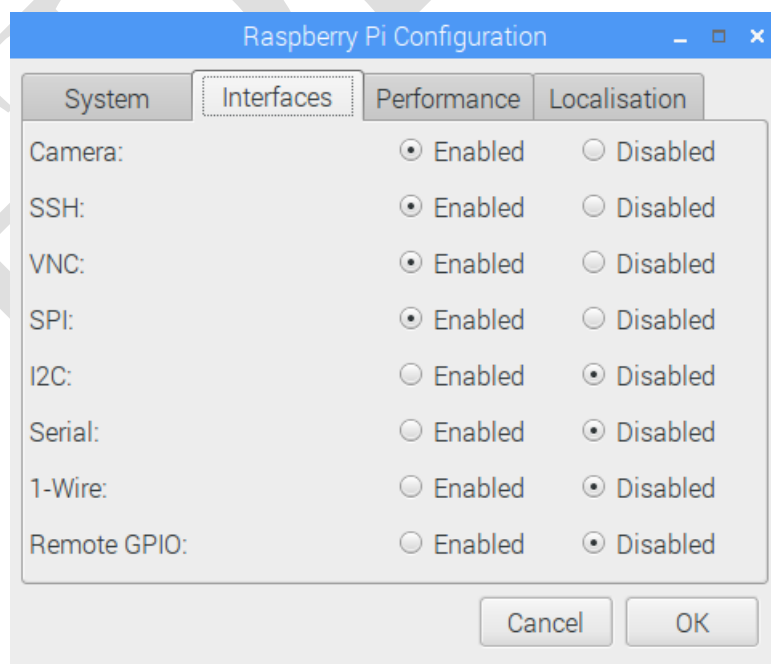
By default, the SPI kernel driver is NOT enabled on the Raspberry Pi Raspian image. Enable the SPI as follows:

1. Open terminal and type `sudo raspi-config` and press Enter.
2. Use the down arrow to select 5 Interfacing options
3. Arrow down to P4 SPI.
4. Select yes when it asks you to enable SPI,
5. Also select yes when it asks about automatically loading the kernel module.
6. Use the right arrow to select the <Finish> button.
7. Select yes when it asks to reboot.



Alternatively using GUI, you can also follow the following steps to enable SPI:

1. Select Preference from the raspberry pi application menu.
2. From Preference select Raspberry Pi Configuration.
3. Now form the Raspberry Pi Configuration window, navigate to Interfaces option.
4. Select the enabled radio button in front of SPI to enable it and click on OK.
5. Finally, do not forget to reboot your raspberry pi after changing this setting.



### Installing the Library:

Open the terminal window. Install the latest version of the library directly from PyPI by typing the following commands:

```
$ sudo apt-get install python3-dev python3-pip  
$ sudo pip3 install max7219
```

Alternatively, clone the code from github:

```
$ git clone https://github.com/rm-hull/max7219.git  
$ cd max7219  
$ sudo pip3 install -e  
$ cd max7219  
$ sudo apt-get install python-dev python-pip  
$ sudo pip install spidev  
$ sudo python setup.py install
```

Now it is the time to write our code. Open Python3, navigate to files and open a new file and write the code given below

#### Code:

```
#Program to Display message on LED matrix module  
import max7219.led as led      #import the max7219 library  
device = led.matrix()  
device.show_message("Hello Edkits")  
#end of code
```

### Wiring up your Circuit:

We are using SPI protocol for wiring LED matrix module to raspberry pi, since it reduces the number pins required for wiring the circuit. You can follow the diagram given below while wiring your circuit.

### GPIO pin-outs

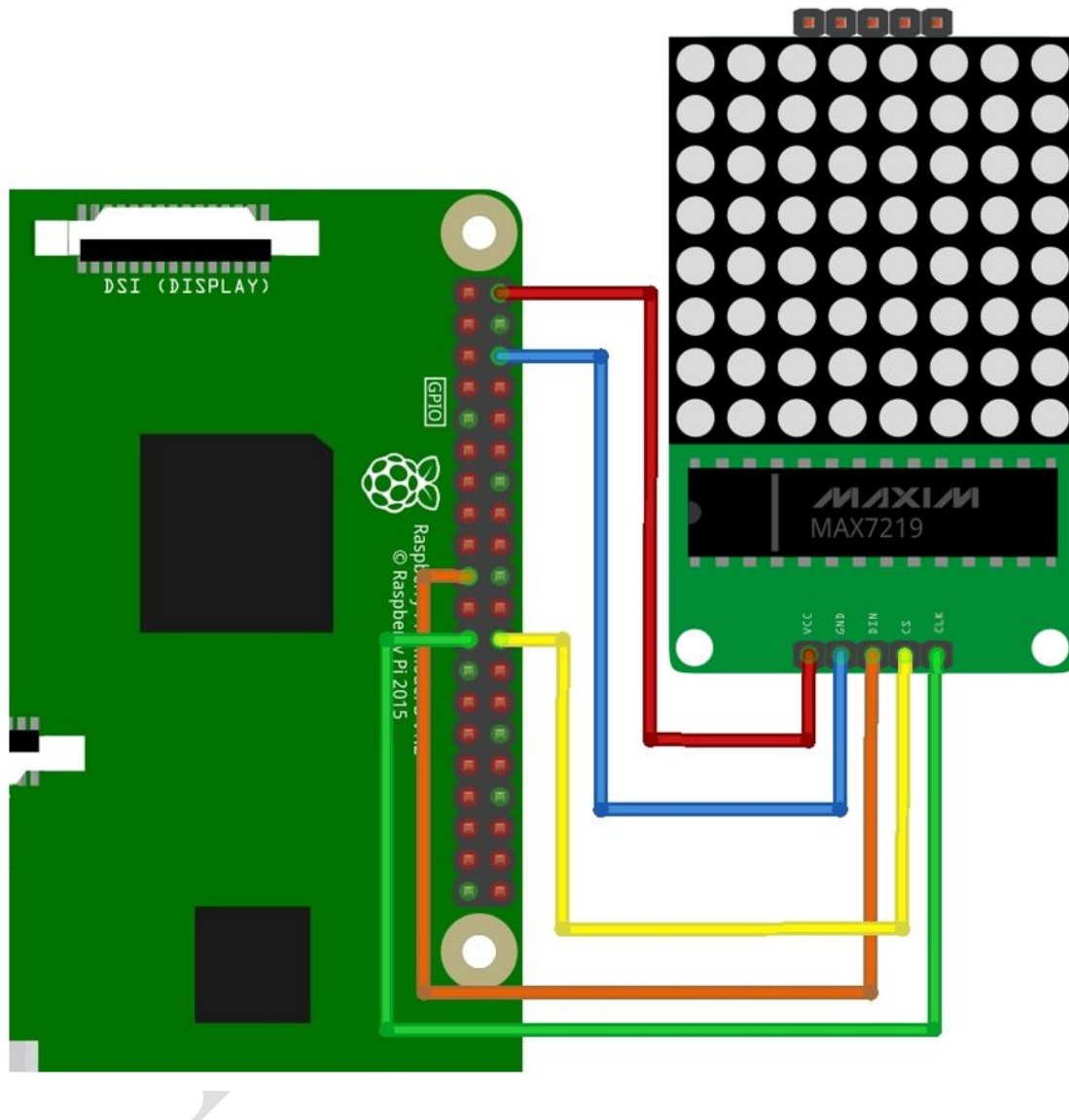
The breakout board has two headers to allow daisy-chaining:

Board Pin	Name	Remarks	RPi Pin	RPi Function
1	VCC	+5V Power	2	5V0
2	GND	Ground	6	GND
3	DIN	Data In	19	GPIO 10 (MOSI)
4	CS	Chip Select	24	GPIO 8 (SPI CE0)
5	CLK	Clock	23	GPIO 11 (SPI CLK)



1. Connect the VCC pin of 7219 driver board to Pin2 of raspberry pi.
2. Connect the Gnd pin of 7219 driver board to Pin6 of raspberry pi.
3. Connect the DIN pin of 7219 driver board to Pin19 of raspberry pi.
4. Connect the CS pin of 7219 driver board to Pin24 of raspberry pi.
5. Lastly, connect the CLK Pin of 7219 driver board to Pin23 of raspberry pi.

**Circuit Diagram:**



After ensuring that the connections are done properly, power on your raspberry pi. Now open Python3 and run the code that you have written for this lesson. So now you have learned how to interface 8x8 LED matrix module with your raspberry pi, how to install libraries and how to enable SPI. Next lesson will bring new things and new fun, so stay tuned!