SUN ROBOTICS

RASPBERRY PI COMPLETE STARTER KIT (1GB)

About the Raspberry Pi

The Raspberry Pi is a low cost, credit-card sized computer that plugs into a computer monitor or TV, and uses a standard keyboard and mouse. It is a capable little device that enables people of all ages to explore computing, and to learn how to program in languages like Scratch and Python. It's capable of doing everything you'd expect a desktop computer to do, from browsing the internet and playing high-definition video, to making spreadsheets, word-processing, and playing games.

What's more, the Raspberry Pi has the ability to interact with the outside world, and has been used in a wide array of digital maker projects, from music machines and parent detectors to weather stations and tweeting birdhouses with infra-red cameras. We want to see the Raspberry Pi being used by kids all over the world to learn to program and understand how computers work.

Link*:*

https://www.raspberrypi.org/help/what-is-a-raspberry-pi/

Getting Started with Raspberry Pi

Introduction:

Welcome to our first lesson on Raspberry Pi. Unlike popular control board such as Arduino, Raspberry Pi is a REAL personal computer with Operation System (OS) like Apple iOS in MacBook and Windows in PC. This allows Raspberry Pi to do much more complicated jobs than Arduino can do.

In this lesson, we will tell you how to install and config Raspbian OS (a Debian Linux distor) in your lovely Raspberry Pi.
First, please learn more about 4 model of raspberry Pi board's differences.

Software Preparation:

Imager utility: Win32DiskImager utility
OS: Raspbian (Use OS Raspbian 2016-11-25 in the subsequent tutorials)
Format Tool: SDFormatter (Optional)
SSH Tool: PuTTY (for Windows users)
Installation Raspbian Operation System (OS):

Download the image for the Raspbian system onto your computer. You can select any version of RASPBIAN system on the official website:

https://downloads.raspberrypi.org/Raspbian/Images/. Write the image via **Win32DiskImager utility** into your microSD/TF card (minimum 8G), then plug the card into the slot on your Raspberry Pi.

Access to Raspberry Pi's Console:

In the subsequent tutorials, the console will be used from time to time. Use use console to interact with Raspbian OS by command line. Therefore, you may need to know how to access to Raspberry Pi's console.

Using Console in GUI (terminal, recommended for beginners):

Connect a monitor to Raspberry Pi via HDMI cable. Connect mouse and keyboard to USB ports. Make sure your Raspbian OS MicroSD card is plug in the SD card slot. At last, connect a 5V 2A DC power to the RPi3. There is a little different from Pi Zero to Pi 3, please refer to the installation guide as followed:

Raspberry Pi Zero	Raspberry Pi Zero W
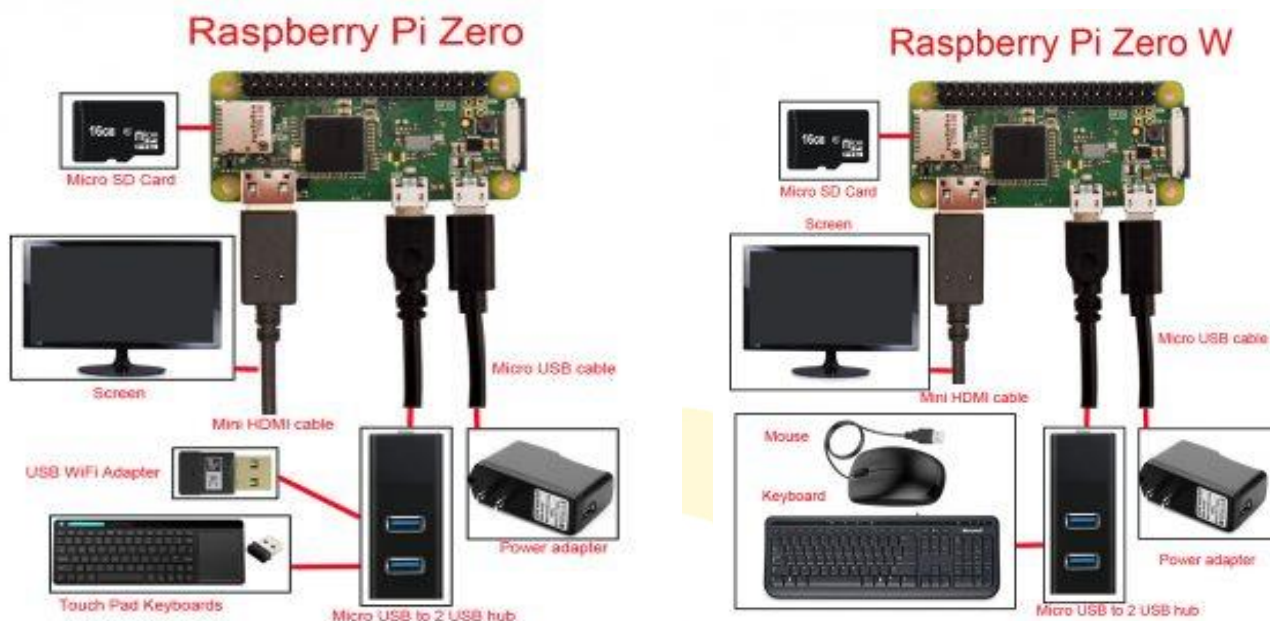
Raspberry Pi 2	Raspberry Pi 3

Figure: 2

Power on the screen and connect wifi hotspot if needed. Then you can see the display showing the Raspberry Pi icon as shown below:

Figure: 3

If the monitor displays colored texts with a black background after booting, and that is in console. You can just use this as a terminal (but not recommend for beginners), or change the option for automatically loading a graphic user interface (GUI). To activate GUI, you can type in startx with the keyboard and press **Enter**, and to always boot up to GUI, type in *sudo raspi-config* and go through **Boot Options** > **Desktop**/**Desk Autologin**, and reboot. Wait for a while and the GUI display will show up as below.

Using Console
There are several ways to use the console only and they can be divided into mainly two ways: using directly and remotely.

Directly (A screen monitor is needed when you use the console directly.)

1. Connect the monitor to power. Then connect it with the Raspberry Pi via the HDMI cable. Connect the keyboard and mouse. Plug the MicroSD card into the Raspberry Pi. At last, connect a 5V 2.5A DC power to the RPi3. Power on the screen and connect wifi hotspot if needed. Then you can see the console full screen.

2. Click the icon of **Terminal** on the screen, or press **CTRL+ALT+T** simultaneously.
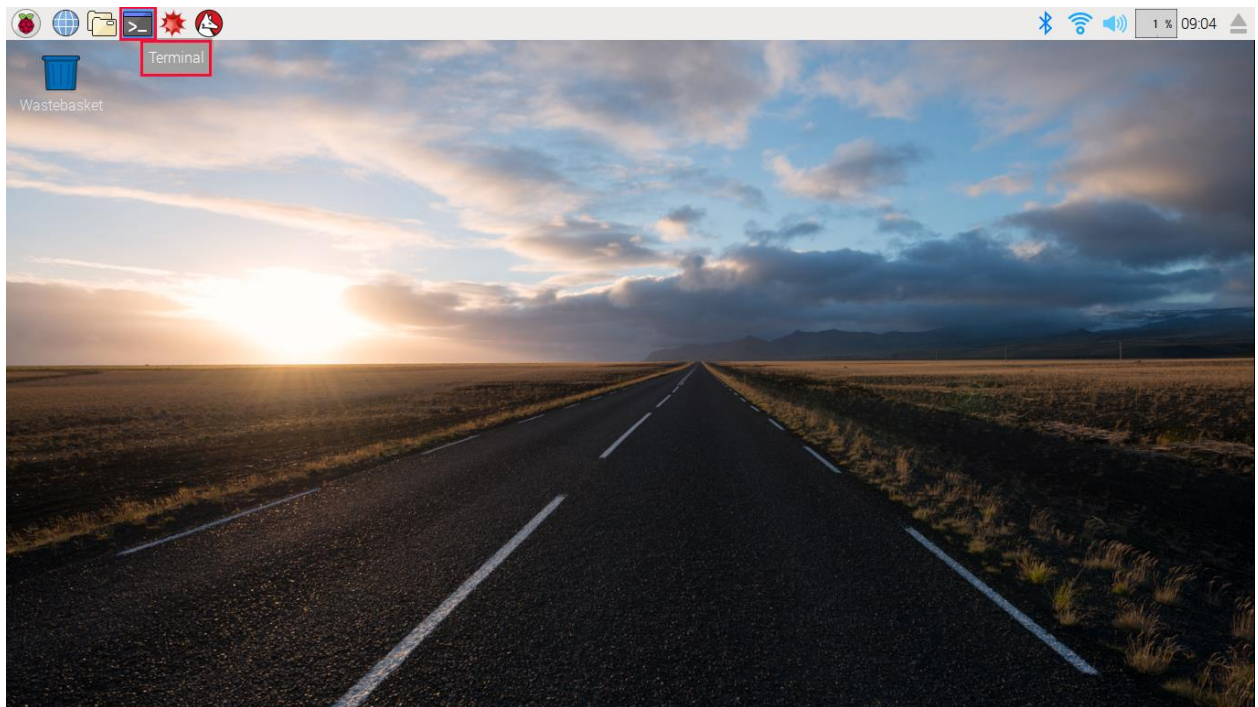
Figure: 4

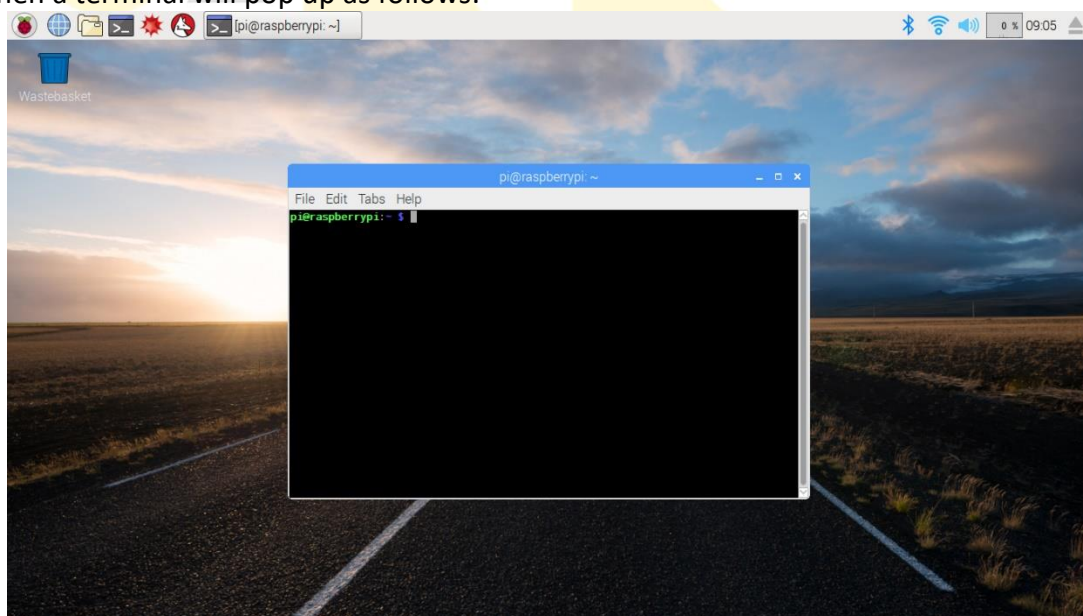3. Then a terminal will pop up as follows:

Figure: 5

If you boot into GUI instead, Open a terminal, type in sudo raspi-config and press enter, go through **Boot Options** > **Console/Console Autologin**, and reboot.

 B. Remotely

Note: when you need to log in remotely, you need to be enable the SSH server. Following the following way can be enable the SSH server via command. Press the command sudo raspi-config in terminal, and then select Advanced Options and click enter; and then select SSH and Click enter.
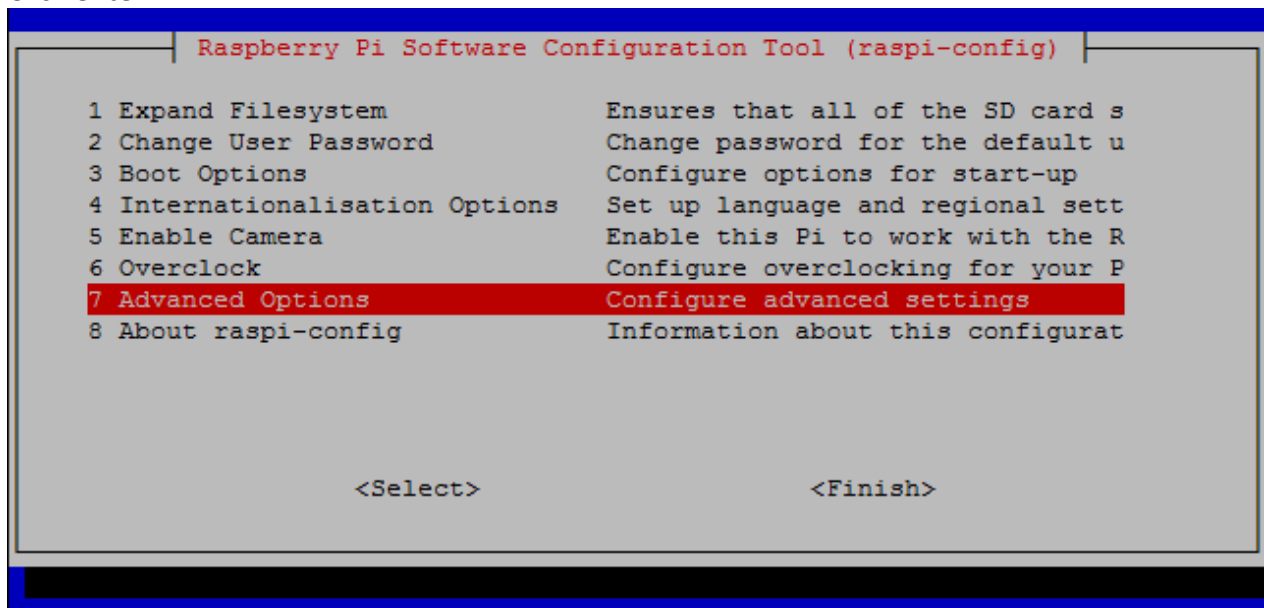
```
┌─────┤ Raspberry Pi Software Configuration Tool (raspi-config) ├─────┐
│                                                                     │
│    1 Expand Filesystem          Ensures that all of the SD card s   │
│    2 Change User Password        Change password for the default u  │
│    3 Boot Options               Configure options for start-up      │
│    4 Internationalisation Options  Set up language and regional sett│
│    5 Enable Camera              Enable this Pi to work with the R    │
│    6 Overclock                  Configure overclocking for your P    │
│    7 Advanced Options           Configure advanced settings          │
│    8 About raspi-config         Information about this configurat    │
│                                                                     │
│                                                                     │
│             <Select>                     <Finish>                    │
│                                                                     │
└─────────────────────────────────────────────────────────────────────┘
```

Figure: 6

```
┌─────┤ Raspberry Pi Software Configuration Tool (raspi-config) ├─────┐
│                                                                     │
│    A1 Overscan          You may need to configure oversca    ↑      │
│    A2 Hostname          Set the visible name for this Pi             │
│    A3 Memory Split      Change the amount of memory made            │
│    A4 SSH               Enable/Disable remote command lin    □      │
│    A5 VNC               Enable/Disable graphical remote a            │
│    A6 SPI               Enable/Disable automatic loading            │
│    A7 I2C               Enable/Disable automatic loading            │
│    A8 Serial            Enable/Disable shell and kernel m            │
│    A9 Audio             Force audio out through HDMI or 3            │
│    AA 1-Wire            Enable/Disable one-wire interface    ↓      │
│                                                                     │
│             <Select>                     <Back>                      │
│                                                                     │
└─────────────────────────────────────────────────────────────────────┘
```
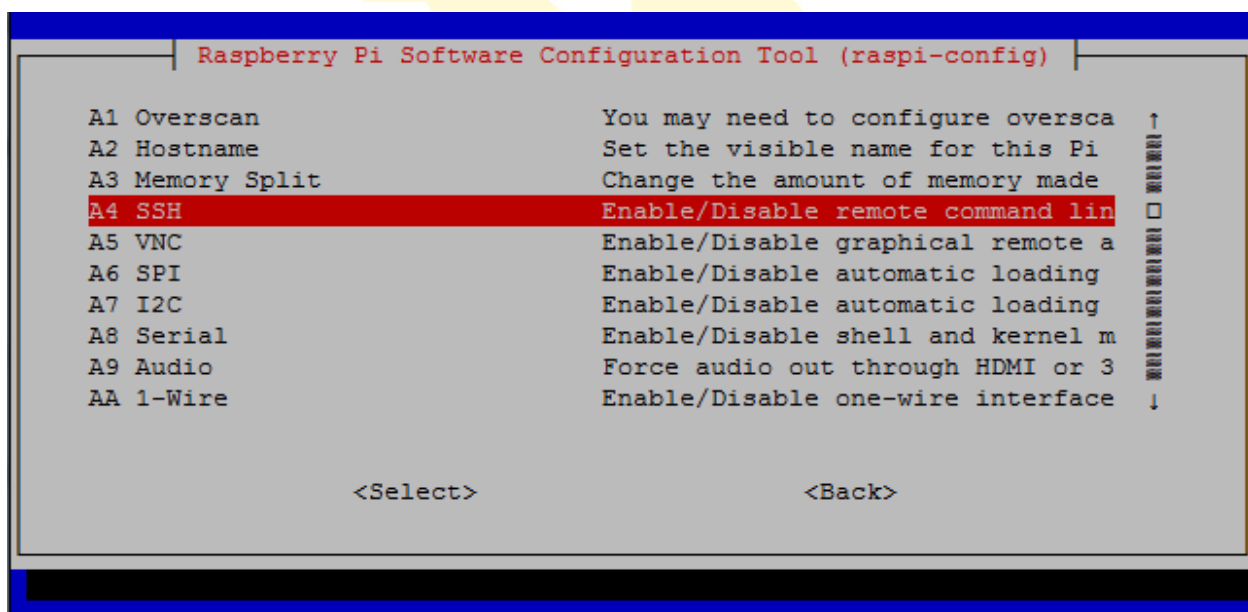
Figure: 7

For three platforms: Windows, Mac and Linux, it might be a little bit different to do this.

(1) Linux and Mac users can easily log into the Raspberry Pi via SSH. On Linux or Mac, find Terminal and open it.
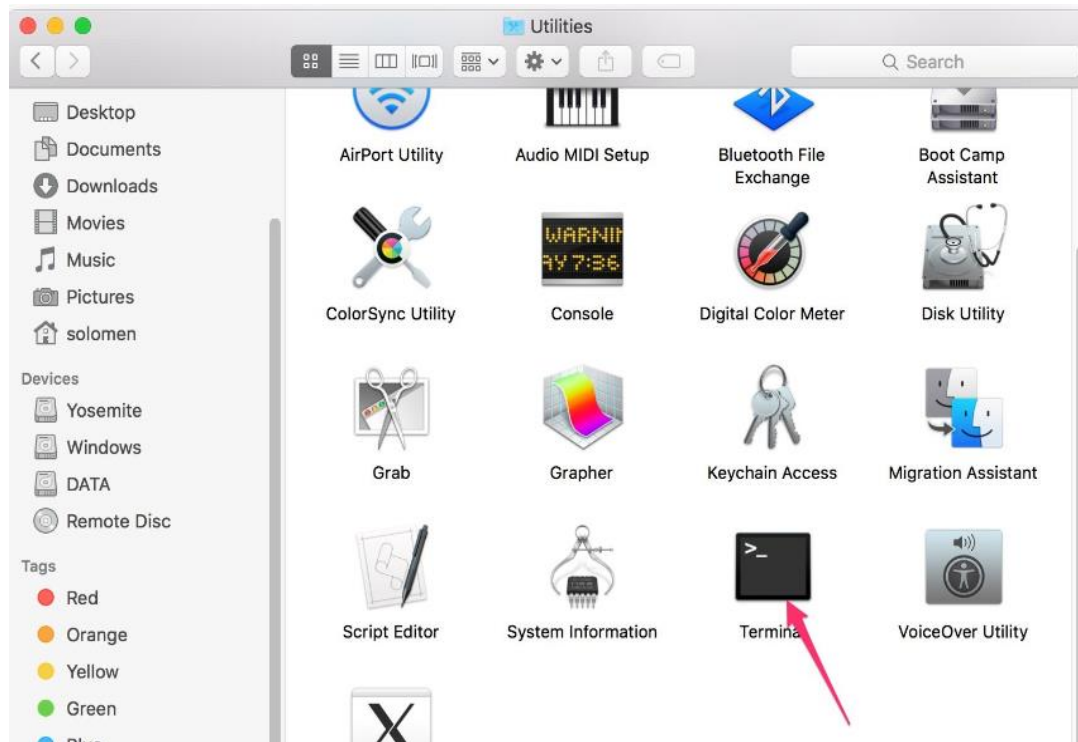
7

Figure: 8

Type in ssh pi@IP address (ssh is the tool for remote login; pi is the user name, and as the name suggests, your RPi's IP address) and then press **Enter** to confirm. For example:

ssh pi@192.168.0.114
If you get a prompt that no ssh is found, you need to install a ssh tool like Ubuntu and Debian by yourself:

sudo apt-get install ssh

(2) For Windows users, you may use a ssh tool to log into Raspberry Pi remotely, like PuTTY.

Step 1. Plug the TF card into the Raspberry Pi, power the RPi with a 5V 2.5A DC power and connect wifi hotspot (or the Ethernet cable). Now the Raspberry Pi is ready.
Note: when connect wifi hotspot, you need to connect with screen monitor, keyboard and mouse

Step 2. Find out the IP address of the RPi.

Method A: Connect your Pi to monitor and mouse, click LAN or Wifi icon to get the IP address as following photo
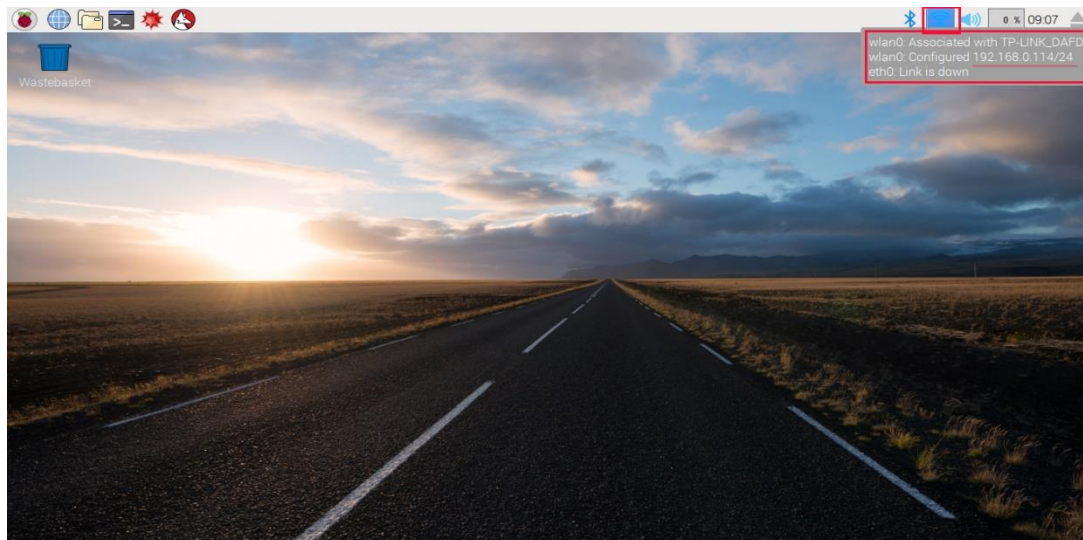
Figure: 9

Method B: you can also find the IP address by typing terminal command: ***ifconfig***

Step 3. Open PuTTY and click **Session** on the left tree-alike structure (generally it's collapsed upon PuTTY startup):
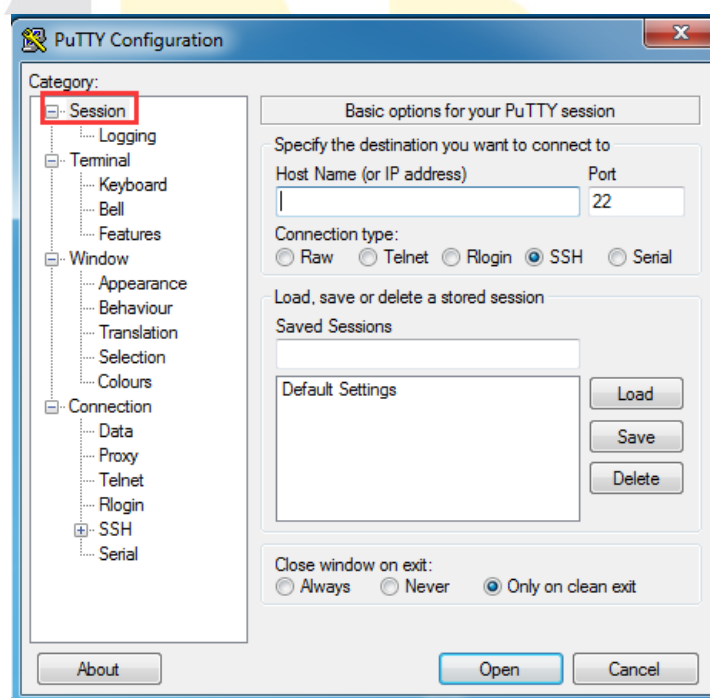
Figure: 10

Step 4. Enter the IP address you got from **Step 2** into the textbox under Host Name (or IP address) and 22 under Port (by default it is 22) /span>

9
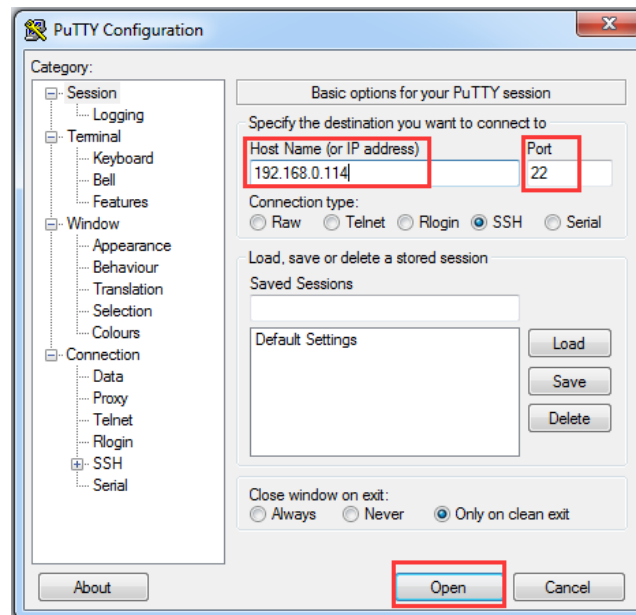
Figure: 11

Step 5. Click Open. Note that when you first log in to the Raspberry Pi with the IP address, you'll be prompted with a security reminder. Just click **yes**. When the PuTTY window prompts login as: type in the user name: **pi**, and password: **raspberry** (the default one, if you haven't changed it).

Note:
when you're typing the password in, the window shows nothing just null, but you're in fact is typing things in. So just focus on typing it right and press Enter. After you log in the RPi successfully, the window will display as follows:
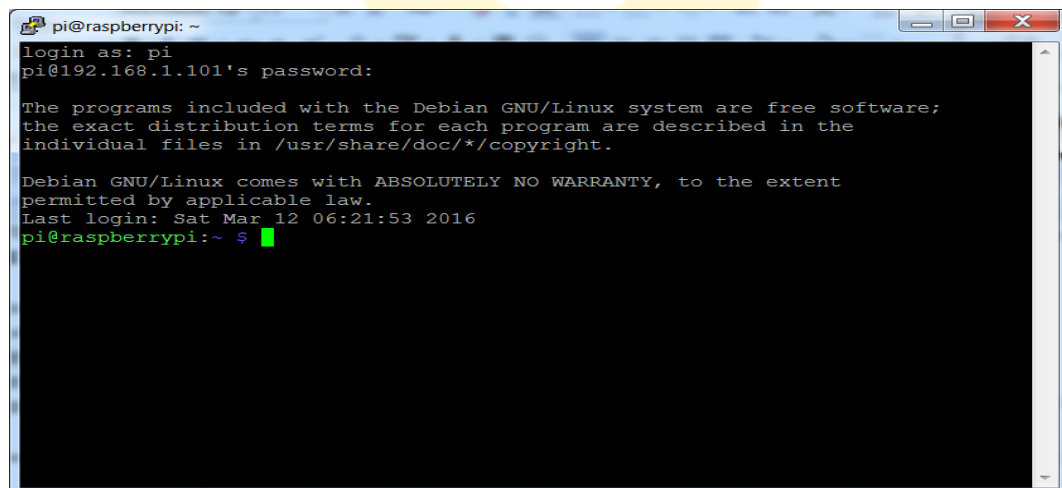
Figure: 12

For other platforms, please contact your supplier.
Now, no matter what method you take, you can get in the console of your Raspberry Pi 3.

10

Introduction of Raspberry Pi GPIO

The Raspberry Pi is a credit card-sized minicomputer with a pre-installed Linux system. As for ports, the Raspberry Pi provides ports for mouse and keyboard. In addition, there are also ports for SD card and HDMI display or TV's connection. Being low cost and low consumption, the Raspberry Pi is very suitable to do very complicated embedded projects through its powerful GPIO pins.

GPIO (General-purpose input/output) are hardware pins rows which locate in the top of RPi board. Raspberry Pi use GPIO pins to interact with other hardware including sensors, motors, and many other peripheral devices.

Through this kit, you will learn how to use the GPIOs to make simple experiments and how to program GPIO.

The Raspberry Pi evolves through many versions including the latest (so far) Raspberry Pi Zero W, Pi 3 Model B, Pi 2 model B, Pi 1 Model B+, Pi Zero, and Pi 1 Model A+. Different version might different GPIO layout.

In this tutorial, our default Pi board is Raspberry Pi 3 Model B which supports Bluetooth and Wi-Fi. You can choose according to actual needs. The following is a photo for RPI3 Model B and Pi Zero:
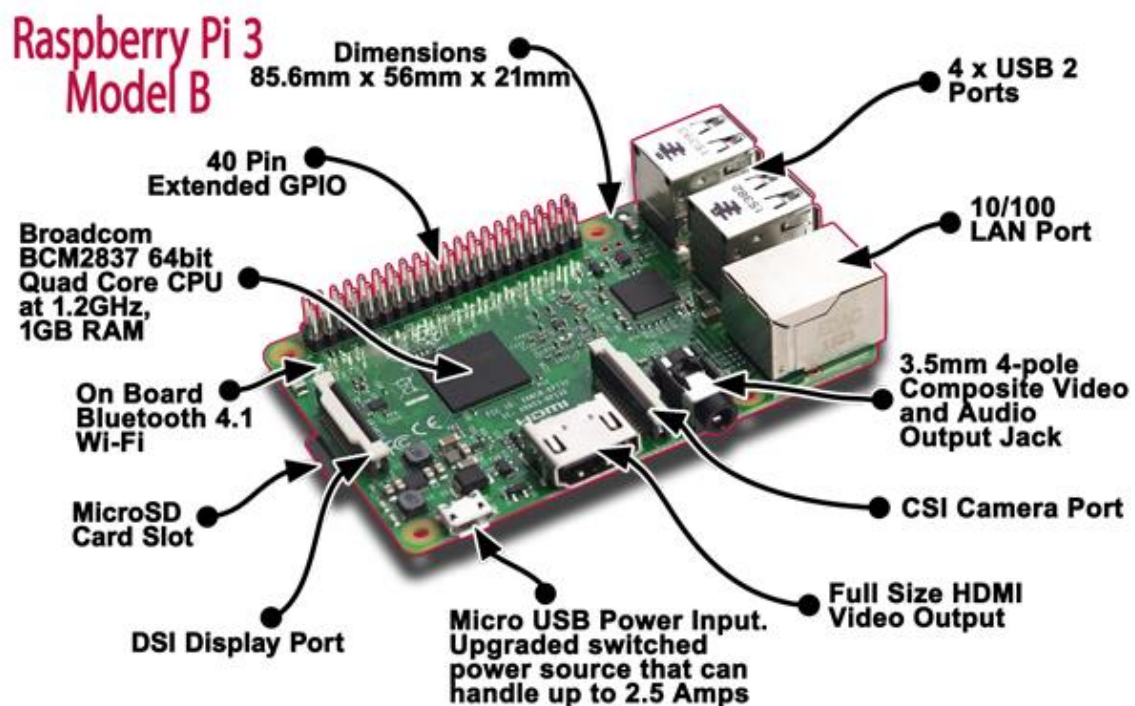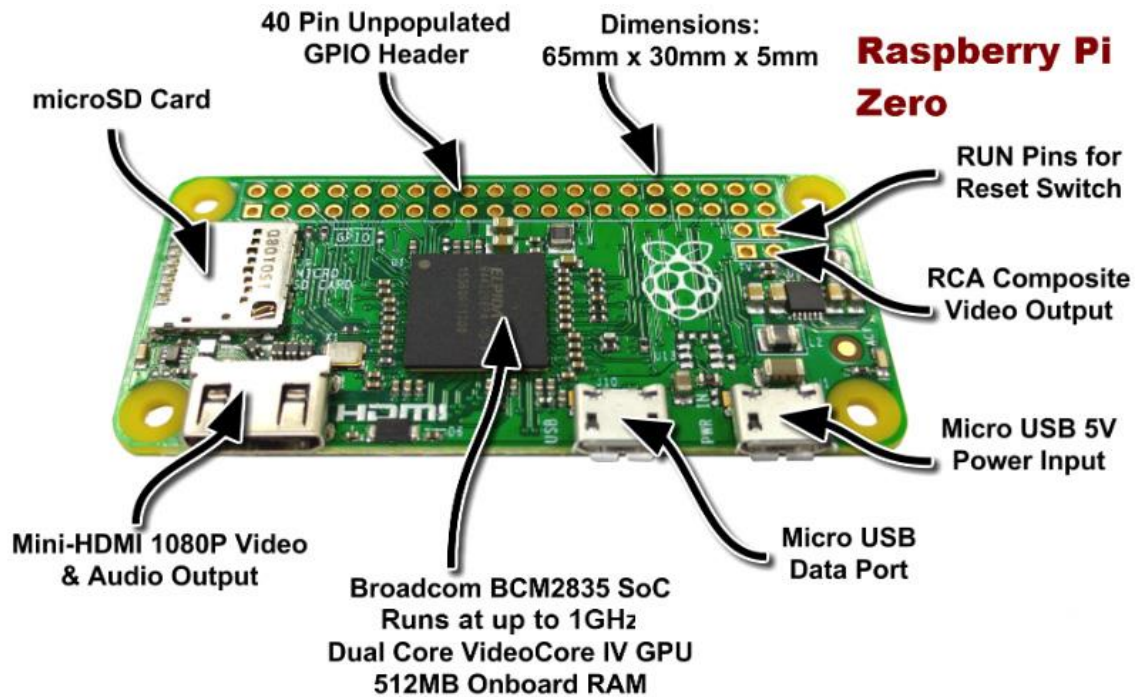

Figure: 1

Figure: 2

GPIO Pin Name System BCM/Wiring pi and their relation with physical pin location:

There are no pins printed on the latest Raspberry Pi, which may cause difficulties for beginners.

To make thing worse, there are actually two major GPIO naming systems: **BCM** which is often used in python language, and **wiringPi** which is often used in C language. Same physical pin with different number has confused many novice learners. So let's start to figure it out.

The relation between Raspberry Pi official GPIO and physical location: In following graph, the number in the circle is the physical pin location , the word beside the pin is the **official** name of the GPIO pin. For example, The Official name of the pin in location 12 (in blue circle of following picture) is "GPIO 18".
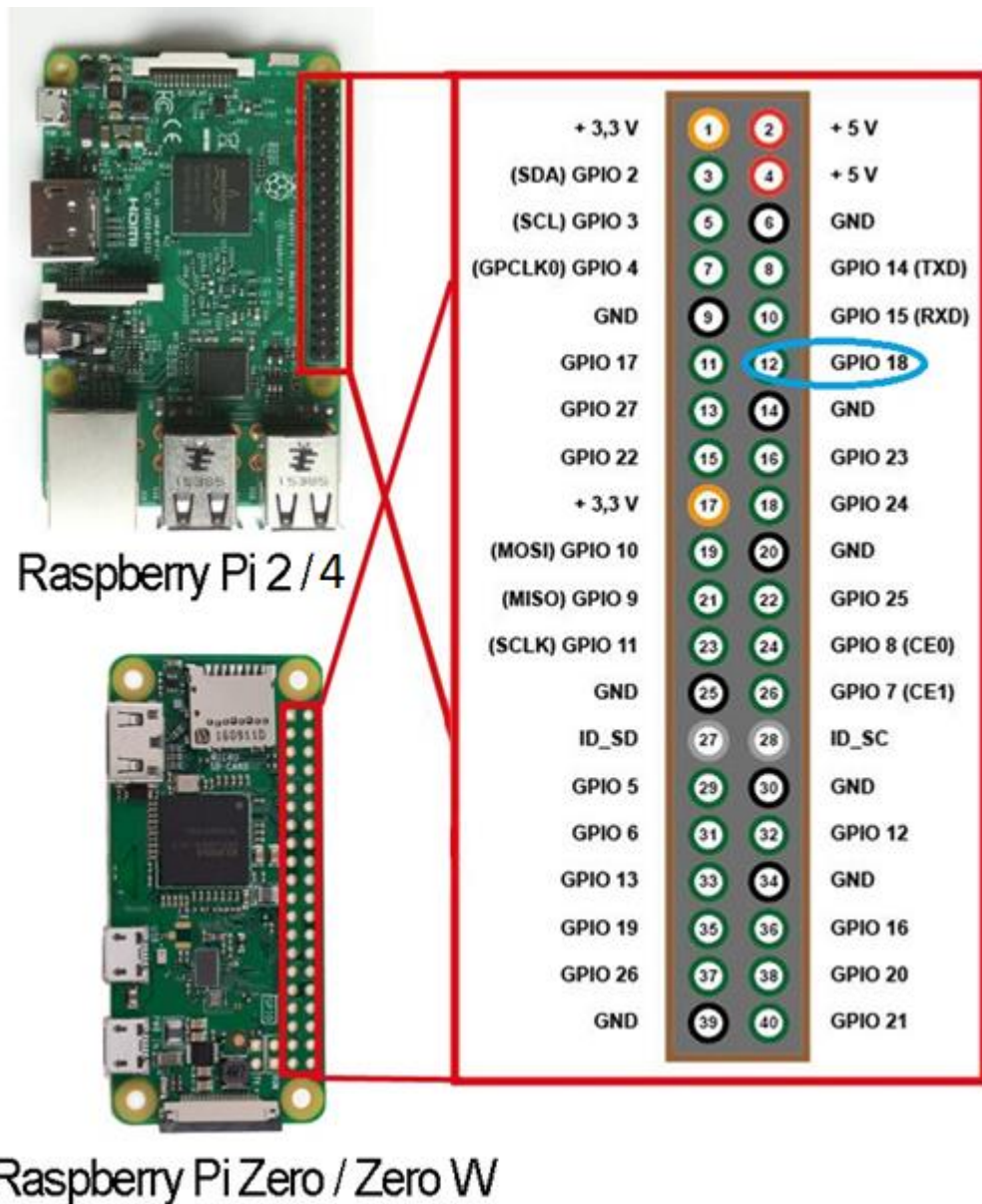
Figure: 3

BCM Pin name system:

The Official name of a GPIO pin normally matches BCM pin number. For example, Physical Pin 12 (inside blue circle of above picture) official name is GPIO 18, in BCM system is pin# 18. If you use Python to program this pin, the sample code should be like following:

```
import RPi.GPIO as GPIO
LedPin = 18
GPIO.output(LEDPIN,GPIO.HIGH)
```

wiringPi GPIO Naming system:

Wiring Pi library is often used in C programming. Pin name/number in wiring Pi is quite different from that of BCM (Raspberry official naming system).

For example, pin in physical location 12(in blue circle of above picture) is called GPIO# **18**(or **GPIO 18**) in BCM, however, in wiringPi, its name is **GPIO. 1** or pin# **1**. If you use C to program this pin, the sample code should be like following:

```
#include < wiringPi.h>
#define LedPin 1
pinMode(LedPin, OUTPUT);
……
```

By running console command **gpio readall**, you can get wiring Pi name, number and its relations to BCM (GPIO official name) numbers as following:

```
pi@raspberrypi:- $ gpio readall
+-----+-----+---------+------+---+---Pi 3---+---+------+---------+-----+-----+
| BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
+-----+-----+---------+------+---+---++---+---+------+---------+-----+-----+
|     |     |    3.3v |      |   | 1  || 2  |   |      | 5v      |     |     |
|   2 |   8 |   SDA.1 |  IN  | 1 | 3  || 4  |   |      | 5v      |     |     |
|   3 |   9 |   SCL.1 |  IN  | 1 | 5  || 6  |   |      | 0v      |     |     |
|   4 |   7 | GPIO. 7 |  IN  | 1 | 7  || 8  | 1 |  IN  | TxD     | 15  | 14  |
|     |     |     0v  |      |   | 9  || 10 | 1 |  IN  | RxD     | 16  | 15  |
|  17 |   0 | GPIO. 0 |  IN  | 0 | 11 || 12 | 0 |  IN  | GPIO. 1 | 1   | 18  |
|  27 |   2 | GPIO. 2 |  IN  | 0 | 13 || 14 |   |      | 0v      |     |     |
|  22 |   3 | GPIO. 3 |  IN  | 0 | 15 || 16 | 0 |  IN  | GPIO. 4 | 4   | 23  |
|     |     |    3.3v |      |   | 17 || 18 | 0 |  IN  | GPIO. 5 | 5   | 24  |
|  10 |  12 |    MOSI |  IN  | 0 | 19 || 20 |   |      | 0v      |     |     |
|   9 |  13 |    MISO |  IN  | 0 | 21 || 22 | 0 |  IN  | GPIO. 6 | 6   | 25  |
|  11 |  14 |    SCLK |  IN  | 0 | 23 || 24 | 1 |  IN  | CE0     | 10  | 8   |
|     |     |     0v  |      |   | 25 || 26 | 1 |  IN  | CE1     | 11  | 7   |
|   0 |  30 |   SDA.0 |  IN  | 1 | 27 || 28 | 1 |  IN  | SCL.0   | 31  | 1   |
|   5 |  21 | GPIO.21 |  IN  | 1 | 29 || 30 |   |      | 0v      |     |     |
|   6 |  22 | GPIO.22 |  IN  | 0 | 31 || 32 | 0 |  IN  | GPIO.26 | 26  | 12  |
|  13 |  23 | GPIO.23 |  IN  | 0 | 33 || 34 |   |      | 0v      |     |     |
|  19 |  24 | GPIO.24 |  IN  | 0 | 35 || 36 | 0 |  IN  | GPIO.27 | 27  | 16  |
|  26 |  25 | GPIO.25 |  IN  | 0 | 37 || 38 | 0 |  IN  | GPIO.28 | 28  | 20  |
|     |     |     0v  |      |   | 39 || 40 | 0 |  IN  | GPIO.29 | 29  | 21  |
+-----+-----+---------+------+---+---++---+---+------+---------+-----+-----+
| BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
+-----+-----+---------+------+---+---Pi 3---+---+------+---------+-----+-----+
```

Figure: 4

Enlarged BCM pin map relation with Physical pin location no.
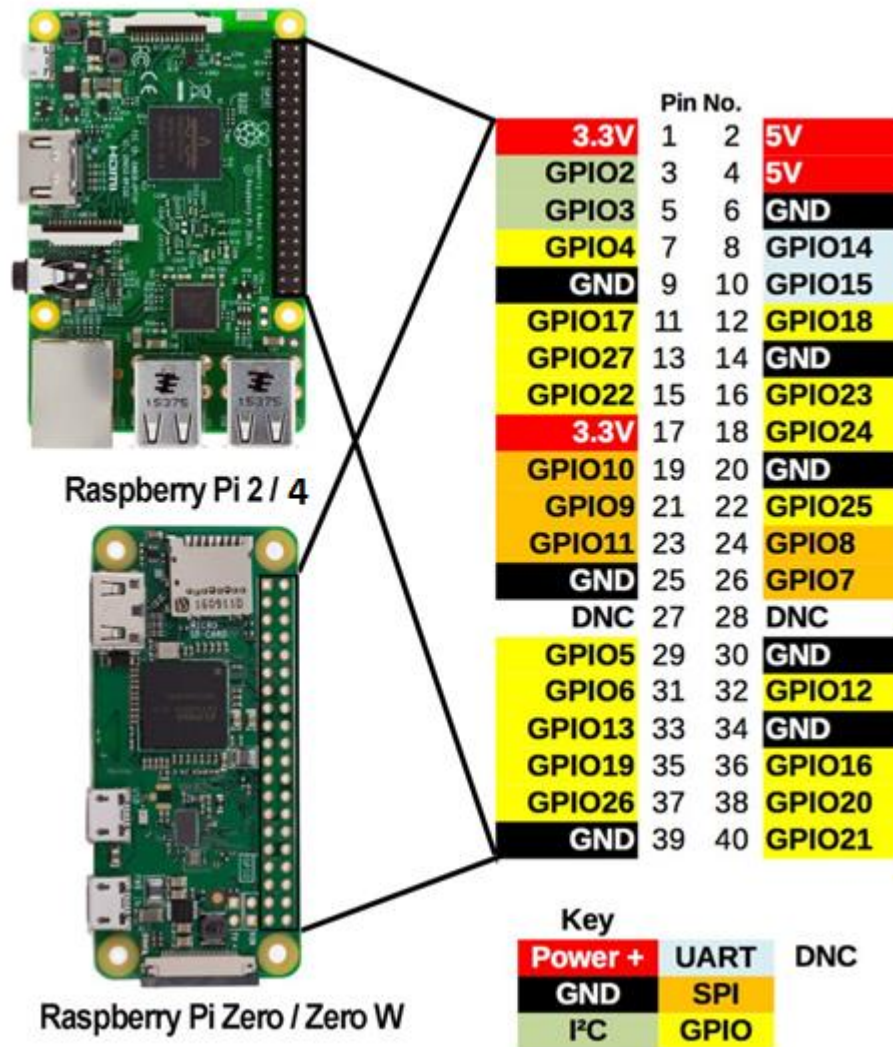
Figure: 5

GPIO are your standard pins that simply be used to turn devices on and off. For example, a LED.

I2C (Inter-Integrated Circuit) pins allow you to connect and talk to hardware modules that support this protocol (I2C Protocol). This will typically take up 2 pins.

SPI (Serial Peripheral Interface Bus) pins can be used to connect and talk to SPI devices. Pretty much the same as I2C but makes use of a different protocol.

UART (Universal asynchronous receiver/transmitter) are the serial pins used to communicate with other devices.

DNC stands for do not connect, this is pretty self-explanatory.

The **power pins** pull power directly from the Raspberry Pi.

GND are the pins you use to ground your devices. It doesn't matter which pin you use as they are all connected to the same line.

T Type 40-pin GPIO Extension Board

15

The function of the extension board is to lead out pins of the Raspberry Pi to breadboard by GPIO Extension Board to avoid GPIO damage caused by frequent plugging in or out. For plugging convenience, the kit includes T-Shape GPIO Extension Board with 40 GPIO flat cable

Figure: 6

This 40-pin GPIO Extension Board and GPIO cable can be plugged into Raspberry Pi 2 model B and 3 model B. as per following graph:

Figure: 7

This 40-pin GPIO Extension Board and GPIO cable can be plugged into Raspberry Pi Zero and Zero W. as per following graph:

16
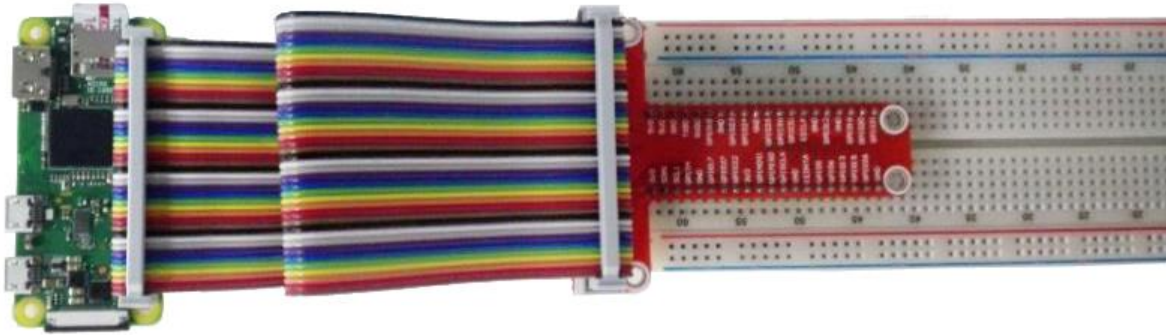
Figure: 8

For your better understanding of every pins, we have drawn a table for you to understand relation between the Name printed in T board, BCM# and wiringPi#

Name	wiringPi	BCM			BCM	wiringPi	Name
			T Extension				
3V3	3.3V	3.3V	3V3	5V0	5V	5V	5V0
SDA1	8	2	SDA1	5V0	5V	5V	5V0
SCL1	9	3	SCL1	GND	GND	GND	GND
GPIO4	7	4	GPIO4	TXD0	14	15	TXD0
GND	GND	GND	GND	RXD0	15	16	RXD0
GPIO17	0	17	GPIO17	GPIO18	18	1	GPIO18
GPIO27	2	27	GPIO27	GND	GND	GND	GND
GPIO22	3	22	GPIO22	GPIO23	23	4	GPIO23
3V3	3.3V	3.3V	3V3	GPIO24	24	5	GPIO24
SPIMOSI	12	10	SPIMOSI	GND	GND	GND	GND
SPIMISO	13	9	SPIMISO	GPIO25	25	6	GPIO25
SPISCLK	14	11	SPISCLK	SPICSO	8	10	SPICSO
GND	GND	GND	GND	SPICSI	7	11	SPICSI
EEDATA	30	0	EEDATA	EECLK	0	31	EECLK
GPIO5	21	5	GPIO5	GND	GND	GND	GND
GPIO6	22	6	GPIO6	GPIO12	12	26	GPIO12
GPIO13	23	13	GPIO13	GND	GND	GND	GND
GPIO19	24	19	GPIO19	GPIO16	16	27	GPIO16
GPIO26	25	26	GPIO26	GPIO20	20	28	GPIO20
GND	GND	GND	GND	GPIO21	21	29	GPIO21

Figure: 9

Raspberry Pi Pin Numbering

Introduction:

WiringPi Pin	BCM GPIO	Name	Header	Name	BCM GPIO	WiringPi Pin
—	—	3.3v	1 | 2	5v	—	—
8	R1:0/R2:2	SDA1	3 | 4	5v	—	—
9	R1:1/R2:3	SCL1	5 | 6	0V	—	—
7	4	GPIO7	7 | 8	TXD	14	15
—	—	0V	9 | 10	RXD	15	16
0	17	GPIO0	11 | 12	GPIO1	18	1
2	R1:21/R2:27	GPIO2	13 | 14	0V	—	—
3	22	GPIO3	15 | 16	GPIO4	23	4
—	—	3.3v	17 | 18	GPIO5	24	5
12	10	MOSI	19 | 20	0V	—	—
13	9	MISO	21 | 22	GPIO6	25	6
14	11	SCLK	23 | 24	CE0	8	10
—	—	0V	25 | 26	CE1	7	11
30	0	SDA0	27 | 28	SCL0	1	31
21	5	GPIO21	29 | 30	0V	—	—
22	6	GPIO22	31 | 32	GPIO26	12	26
23	13	GPIO23	33 | 34	0V	—	—
24	19	GPIO24	35 | 36	GPIO27	16	27
25	26	GPIO25	37 | 38	GPIO28	20	28
		0V	39 | 40	GPIO29	21	29
WiringPi Pin	BCM GPIO	Name	Header	Name	BCM GPIO	WiringPi Pin

Figure: 1

There are three methods for numbering Raspberry Pi's GPIO:

1. Numbering according to the physical location of the pins, from left to right, top to bottom, the left is odd, the right is even.
2. Numbering according the GPIO registers of BCM2835/2836 SOC.
3. Numbering according the GPIO library wiringPi.
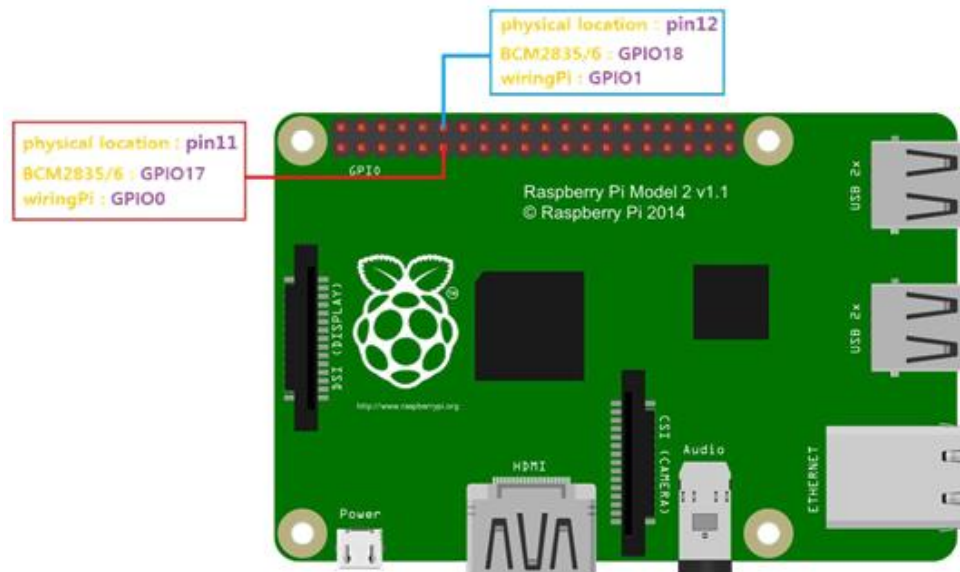
Figure: 2

Raspberry Pi GPIO Library Introduction:

Currently, there are two major GPIO libraries for Raspberry Pi, RPi.GPIO and wiringPi.

RPi.GPIO:

RPi.GPIO is a python module to control Raspberry Pi GPIO channels. For more information about RPi.GPIO, please visit:

https://pypi.python.org/pypi/RPi.GPIO/

For examples and documentation, please visit:
http://sourceforge.net/p/raspberry-gpio-python/wiki/Home/

The RPi.GPIO module is pre-installed in the official Raspbian operating system, you can use it directly.

wiringPi:

The wiringPi is a GPIO access library written in C for the BCM2835/6 SOC used in the Raspberry Pi. It's released under the GNU LGPLv3 license and is usable from C and C++ and many other languages with suitable wrappers. It's designed to be familiar to people who have used the Arduino "wiring" system.

For more information about wiringPi, please visit: http://wiringpi.com/

Install wiringPi:

19

Step 1: Get the source code
 $ git clone git://git.drogon.net/wiringPi
 Step 2: Compile and install

 $ cd wiringPi

 $ git pull origin

 $ sudo ./build

Press Enter, the script "build" will automatically compile wiringPi source code and then install it to the Raspberry Pi.

Next, verify whether the wiringPi is installed successfully or not:

wiringPi includes a command-line utility gpio which can be used to program and setup the GPIO pins. You can use this to read and write the pins and even use it to control them from shell scripts.

You can verify whether the wiringPi is installed successfully or not by the following commands:
 $ sudo gpio -v

```
pi@raspberrypi ~ $ sudo gpio -v
gpio version: 2.26
Copyright (c) 2012-2015 Gordon Henderson
This is free software with ABSOLUTELY NO WARRANTY.
For details type: gpio -warranty

Raspberry Pi Details:
  Type: Model 2, Revision: 1.1, Memory: 1024MB, Maker: Sony
pi@raspberrypi ~ $
```

$ sudo gpio readall

```
pi@raspberrypi ~ $ sudo gpio readall
 +-----+-----+---------+------+---+---Pi 2---+---+------+---------+-----+-----+
 | BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
 +-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
 |     |     |    3.3v |      |   |  1 || 2  |   |      | 5v      |     |     |
 |   2 |   8 |   SDA.1 | ALT0 | 1 |  3 || 4  |   |      | 5V      |     |     |
 |   3 |   9 |   SCL.1 | ALT0 | 1 |  5 || 6  |   |      | 0v      |     |     |
 |   4 |   7 |  GPIO. 7 |  IN | 1 |  7 || 8  | 1 | ALT0 | TxD     | 15  | 14  |
 |     |     |     0v  |      |   |  9 || 10 | 1 | ALT0 | RxD     | 16  | 15  |
 |  17 |   0 | GPIO. 0 |  IN  | 0 | 11 || 12 | 0 | IN   | GPIO. 1 | 1   | 18  |
 |  27 |   2 | GPIO. 2 |  IN  | 0 | 13 || 14 |   |      | 0v      |     |     |
 |  22 |   3 | GPIO. 3 |  IN  | 0 | 15 || 16 | 0 | IN   | GPIO. 4 | 4   | 23  |
 |     |     |    3.3v |      |   | 17 || 18 | 0 | IN   | GPIO. 5 | 5   | 24  |
 |  10 |  12 |    MOSI | ALT0 | 0 | 19 || 20 |   |      | 0v      |     |     |
 |   9 |  13 |    MISO | ALT0 | 0 | 21 || 22 | 0 | IN   | GPIO. 6 | 6   | 25  |
 |  11 |  14 |    SCLK | ALT0 | 0 | 23 || 24 | 1 | ALT0 | CE0     | 10  | 8   |
 |     |     |     0v  |      |   | 25 || 26 | 1 | ALT0 | CE1     | 11  | 7   |
 |   0 |  30 |   SDA.0 |  IN  | 1 | 27 || 28 | 1 | IN   | SCL.0   | 31  | 1   |
 |   5 |  21 | GPIO.21 |  IN  | 1 | 29 || 30 |   |      | 0v      |     |     |
 |   6 |  22 | GPIO.22 |  IN  | 1 | 31 || 32 | 0 | IN   | GPIO.26 | 26  | 12  |
 |  13 |  23 | GPIO.23 |  IN  | 0 | 33 || 34 |   |      | 0v      |     |     |
 |  19 |  24 | GPIO.24 |  IN  | 0 | 35 || 36 | 0 | IN   | GPIO.27 | 27  | 16  |
 |  26 |  25 | GPIO.25 |  IN  | 0 | 37 || 38 | 0 | IN   | GPIO.28 | 28  | 20  |
 |     |     |     0v  |      |   | 39 || 40 | 0 | IN   | GPIO.29 | 29  | 21  |
 +-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
 | BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
 +-----+-----+---------+------+---+---Pi 2---+---+------+---------+-----+-----+
pi@raspberrypi ~ $
```
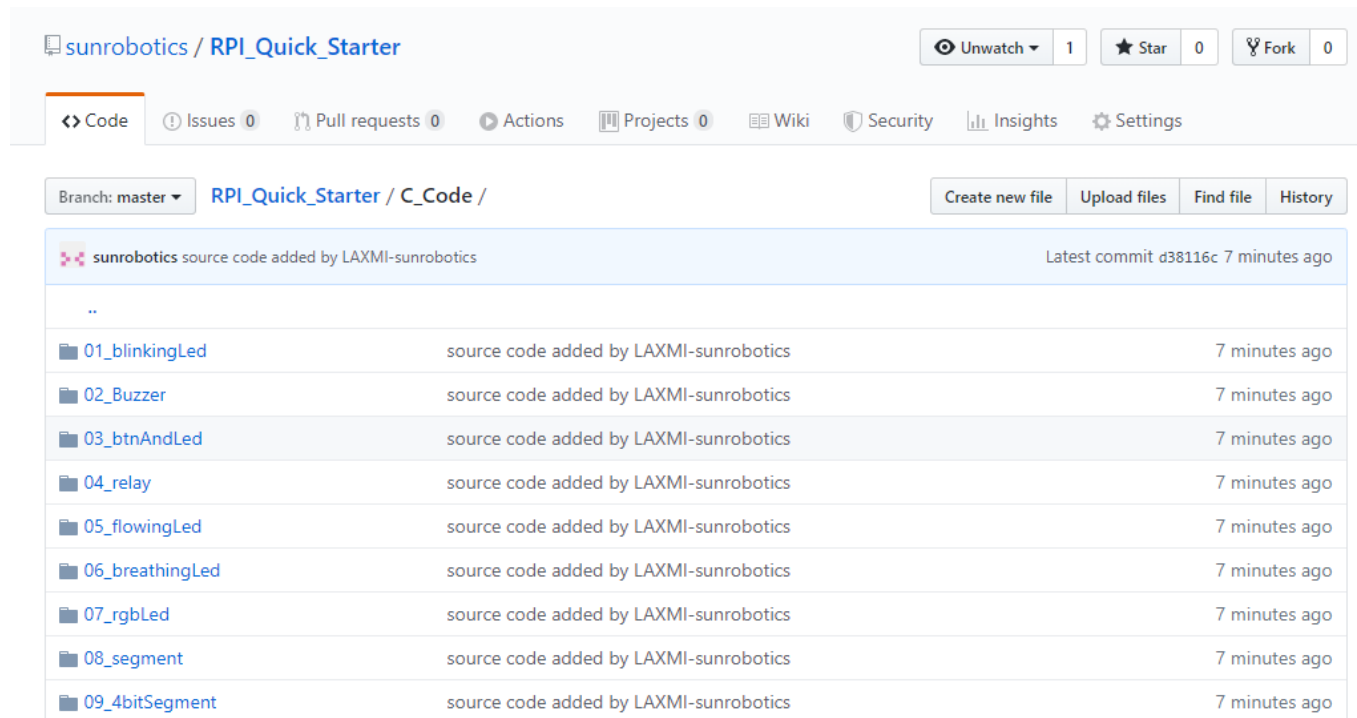
If you can see the information shown above, it indicates that the wiringPi has been installed successfully.

Download the practical's source code: **git clone**
https://github.com/sunrobotics/RPI_Quick_Starter.git
[Make sure when you download the source code, make desktop as your working directory]

C_source_code:

Python_source_code:

Web: www.sunrobotics.co.in
Email: support@sunrobotics.co.in

How to use the wiringPi and the RPi.GPIO

Here we take a blinking LED for example to illustrate how to use the wiringPi C library and the RPi.GPIO Python module.

Step 1: Build the circuit according to the following schematic diagram

Note: Resistance=220Ω
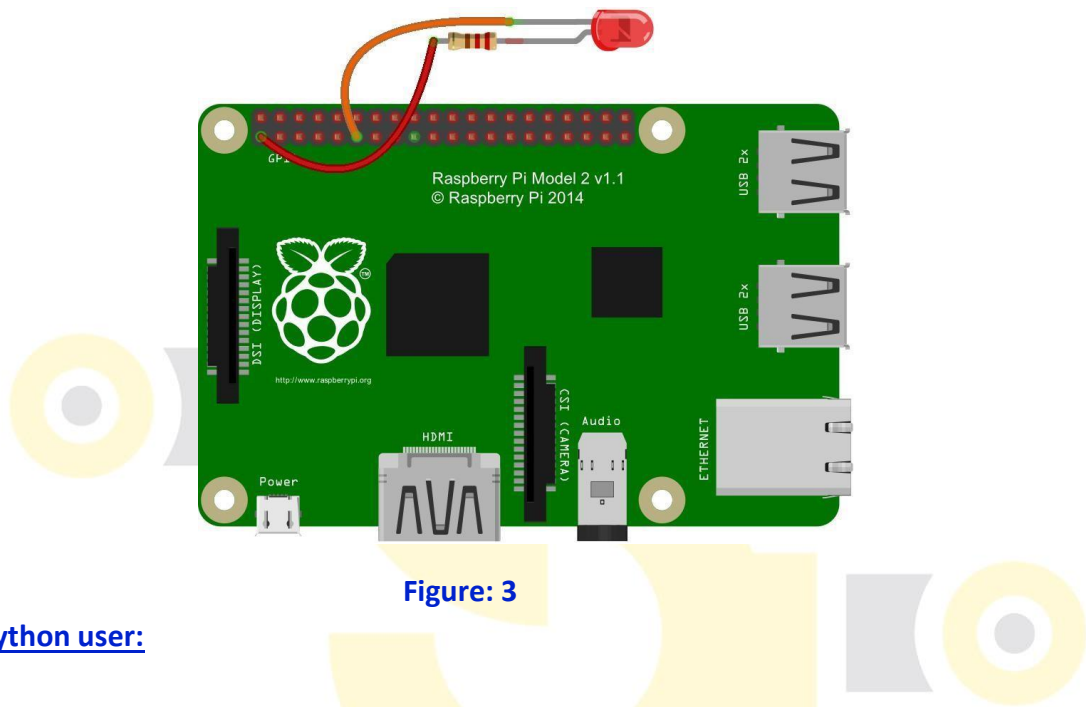
For Python user:

Step 2: Create a file named led.py
$ sudo touch led.py

Step 3: Open the file led.py with vim or nano

$ sudo vim led.py
Write down the following source code, then save and exit.

```python
#!/usr/bin/env python
import RPi.GPIO as GPIO
import time

Led = 11     # pin11

def setup():
    GPIO.setmode(GPIO.BOARD)      # Numbering according to the physical location
    GPIO.setup(Led, GPIO.OUT)     # Set pin mode as output
    GPIO.output(Led, GPIO.HIGH)   # Output high level(+3.3V) to off the led

def loop():
    while True:
        print '...led on'
        GPIO.output(Led, GPIO.LOW)   # led on
        time.sleep(0.5)
        print 'led off...'
        GPIO.output(Led, GPIO.HIGH)  # led off
        time.sleep(0.5)

def destroy():
    GPIO.output(Led, GPIO.HIGH)     # led off
    GPIO.cleanup()                  # Release resource

if __name__ == '__main__':     # Program start from here
    setup()
    try:
        loop()
    except KeyboardInterrupt:   # Press 'Ctrl+C' to end the program
        destroy()
```

Step 4: Run the program

$ sudo python led.py

```
pi@raspberrypi /home $ ls
led.py  pi
pi@raspberrypi /home $ sudo python led.py
...led on
led off...
...led on
led off...
...led on
led off...
```

Press Enter, you should see that the LED is blinking. Press 'Ctrl+C', the program execution will be terminated.

For C language user:
Step 2: Create a file named led.c
$ sudo touch *blinkingLed*.c

```
pi@raspberrypi /home $ ls
led.py  pi
pi@raspberrypi /home $ sudo touch led.c
pi@raspberrypi /home $
```

Step 3: Open the file led.c with vim or nano

23

Write down the following source code, then save and exit.

```c
#include <wiringPi.h>
#include <stdio.h>

#define  Led     0  //wiringPi GPIO0, pin11

int main(void)
{
    if(wiringPiSetup() == -1){
        printf("setup wiringPi failed !\n");
        return -1;
    }

    pinMode(Led, OUTPUT);

    while(1){
        digitalWrite(Led, LOW);    //led on
        printf("led on...\n");
        delay(500);
        digitalWrite(Led, HIGH);   //led off
        printf("...led off\n");
        delay(500);
    }

    return 0;
}
```

Step 4: Compile the code

$ sudo gcc led.c -lwiringPi

```
pi@raspberrypi /home $ ls
led.c  led.py  pi
pi@raspberrypi /home $ sudo gcc led.c -lwiringPi
pi@raspberrypi /home $
```

After executing this command, you'll find a file named a.out appear in the current directory.
It is an executable program.

```
pi@raspberrypi /home $ ls
a.out  led.c  led.py  pi
pi@raspberrypi /home $
```

Step 5: Run the program

$ sudo ./a.out

```
pi@raspberrypi /home $ sudo ./a.out
led on...
...led off
led on...
...led off
```

Press Enter, you should see that the LED is blinking. Press 'Ctrl+C', the program execution
will be terminated.

Resources:

24

Web: www.sunrobotics.co.in
Email: support@sunrobotics.co.in

http://sourceforge.net/p/raspberry-gpio
python/wiki/Examples/ http://wiringpi.com/reference/

NOTE:

Before learning the next courses, please copy the source code we provided in the DVD which came with the kit to your Raspberry Pi's /home/ directory.

C Language Source Code:
Python Source Code:

Lesson -1 Blinking LED

1.1 Overview:

In this tutorial, we will start the journey of learning Raspberry Pi in the first lesson, we will learn how to control an LED.

1.2 Components List:

- ➢ 1 x Raspberry Pi 4
- ➢ 1 x 220Ω Resistor
- ➢ 1 x LED
- ➢ 1 x Breadboard
- ➢ 2 x Jumper Wires

1.3 Principle:

In this lesson, we will program the Raspberry Pi to output high (+3.3V) and low level (0V), and then make an LED which is connected to the Raspberry Pi GPIO flicker with a certain frequency.

1.3.1 What is LED?

The LED is the abbreviation of light emitting diode. It is usually made of gallium arsenide, gallium phosphate semiconductor materials. The LED has two electrodes, a positive electrode and a negative electrode, it will light only when a forward current passes, and it can be red, blue, green or yellow light, etc. The color of light depends on the materials it was made.

In general, the drive current for LED is 5-20mA. Therefore, in reality it usually needs an extra resistor for current limitation so as to protect the LED.

1.3.2. What is the resistor?

The main function of the resistor is to limit current. In the circuit, the character 'R' represents resistor, and the unit of resistor is ohm (Ω).

The band resistor is used in this experiment. A band resistor is one whose surface is coated with some particular color through which the resistance can be identified directly.

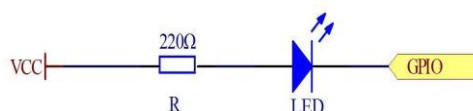There are two methods for connecting an LED with Raspberry Pi GPIO:
①

Figure: 1

26

As shown in the schematic diagram above, the anode of LED is connected to VCC (+3.3V), and the cathode of LED is connected to the Raspberry Pi GPIO. When the GPIO output low level, the LED is on; when the GPIO output high level, the LED is off.

②

Figure: 1.2

As shown in the schematic diagram above, the anode of LED is connected to Raspberry Pi GPIO via a resistor, and the cathode of LED is connected to the ground (GND). When the GPIO output high level, the LED is on; when the GPIO output low level, the LED is off.

The size of the current-limiting resistor is calculated as follows: 5~20mA current is required to make an LED on, and the output voltage of the Raspberry Pi GPIO is 3.3V, so we can get the resistance：

$$R = U / I = 3.3V / (5\text{~}20mA) = 165Ω\text{~}660Ω$$

In this experiment, we choose a 220ohm resistor.

The experiment is based on method ①, we select pin 11 of Raspberry Pi to control an LED. When the pin 11 of Raspberry Pi is programmed to output low level, then the LED will be lit, next delay for the amount of time, and then programmed the pin 11 to high level to make the LED off. Continue to perform the above process, you can get a blinking LED.

1.3.3. Key functions

1.3.3.1 C language user:

- **int wiringPiSetup (void)**

The function must be called at the start of your program or your program will fail to work correctly. You may experience symptoms from it simply not working to segfaults and timing issues.
 Note*:* This function needs to be called with root privileges.
- **void pinMode (int pin, int mode)**

This sets the mode of a pin to either **INPUT**, **OUTPUT**, **PWM_OUTPUT** or **GPIO_CLOCK**. Note that only wiringPi pin 1 (BCM_GPIO 18) supports PWM output and only wiringPi pin 7 (BCM_GPIO 4) supports CLOCK output modes.

This function has no effect when in Sys mode. If you need to change the pin mode, then you can do it with the gpio program in a script before you start your program.

- **void digitalWrite (int pin, int value)**

Writes the value **HIGH** or **LOW** (1 or 0) to the given pin which must have been previously set as an output. *WiringPi* treats any non-zero number as HIGH, however 0 is the only representation of LOW.

- **void delay (unsigned int howLong)**

This causes program execution to pause for at least how Long milliseconds. Due to the multi-tasking nature of Linux it could be longer. Note that the maximum delay is an unsigned 32-bit integer or approximately 49 days.

1.3.3.2 Python user:

- **GPIO.setmode(GPIO.BOARD)**

There are two ways of numbering the IO pins on a Raspberry Pi within RPi.GPIO. The first is using the **BOARD** numbering system. This refers to the pin numbers on the P1 header of the Raspberry Pi board. The advantage of using this numbering system is that your hardware will always work, regardless of the board revision of the RPi. You will not need to rewire your connector or change your code.

The second numbering system is the **BCM** (GPIO.BCM) numbers. This is a lower level way of working - it refers to the channel numbers on the Broadcom SOC. You have to always work with a diagram of which channel number goes to which pin on the RPi board. Your script could break between revisions of Raspberry Pi boards.

- **GPIO.setup(channel, mode)**

This sets every channel you are using as an input (GPIO.IN) or an output (GPIO.OUT).
- **GPIO.output(channel, state)**

This sets the output state of a GPIO pin. Where channel is the channel number based on the numbering system you have specified (BOARD or BCM). **State** can be 0 / GPIO.LOW / False or 1 / GPIO.HIGH / True.

- **GPIO.cleanup( )**
At the end any program, it is good practice to clean up any resources you might have used. This is no different with RPi.GPIO. By returning all channels you have used back to inputs with no pull up/down, you can avoid accidental damage to your RPi by shorting out the pins. Note that this will only clean up GPIO channels that your script has used. Note that GPIO.cleanup() also clears the pin numbering system in use.

28

1.4 Procedures:
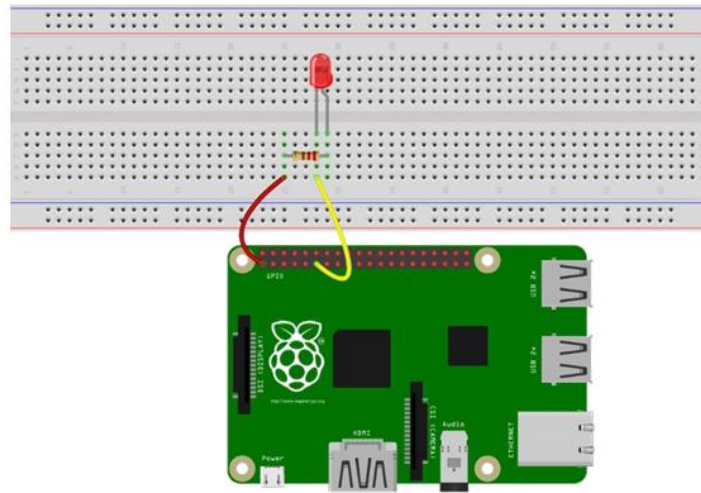
Step 1. Build the circuit

Figure: 1.3

Step 2. Program

1.5 C user:

1.1 Edit and save the code with vim or nano.

C Source Code:

/home/user/DesktopRPI_Quick_Starter/C_Code/01_blinkingLed/blinkingLed.c

```c
#include <wiringPi.h>
#include <stdio.h>


#define  LedPin    11
 int  main(void)
{
  if(wiringPiSetup() == -1){ //when initialize wiringPi failed, print message to screen printf("setup wiringPi failed
        !\n");     return -1;
     }
 pinMode(LedPin, OUTPUT);
        while(1){
        digitalWrite(LedPin, LOW); //led on printf("led on...\n");
        delay(500);
        digitalWrite(LedPin, HIGH); //led off printf("...led  off\n");
         delay(500);
     }
        return 0;
}
```

1.2 Compile the program

$ gcc blinkingLed.c -o led -lwiringPi

Note*:* The parameter '-o' is to specify a file name for the compiled executable program. If you
do not use this parameter, the default file name is *a.out*.

29

1.3 Run the program
$ sudo ./led

1.6 Python user:

1.1 Edit and save the code with vim or nano.

Python Source Code:

/home/user/Desktop/RPI_Quick_Starter/Python_Code/01_blinkingLed.py

```python
#!/usr/bin/env python import
RPi.GPIO as GPIO import time

LedPin = 11       # pin11

def     setup():
    GPIO.setmode(GPIO.BOARD) # Numbers GPIOs by physical location
    GPIO.setup(LedPin, GPIO.OUT) # Set LedPin's mode is output
    GPIO.output(LedPin, GPIO.HIGH) # Set LedPin high(+3.3V) to off led

def loop(): while
        True:
        print '...led on'
        GPIO.output(LedPin, GPIO.LOW) # led on

time.sleep(0.5)   print
'led off...'
        GPIO.output(LedPin, GPIO.HIGH) # led off
        time.sleep(0.5)
 de     f
destroy():
    GPIO.output(LedPin, GPIO.HIGH)      # led off
    GPIO.cleanup()      # Release resource

if __name__ == '__main__': # Program start from here setup()      try: loop()
except KeyboardInterrupt: # When 'Ctrl+C' is pressed, the child program destroy()
will be executed.      destroy()
```

1.2 Run the program

$ sudo python 01_blinkingLed.py

Press Enter, and then you can see the LED is blinking.
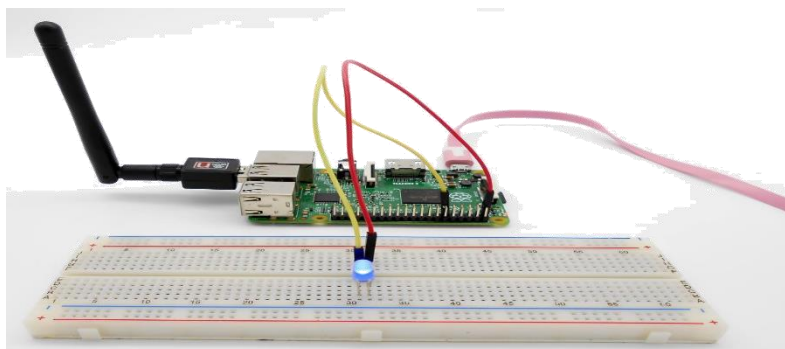
Figure: 1.4

Lesson -2 controlling an LED with a button

2.1 Overview:

In this lesson, we will learn how to detect the status of a button, and then toggle the status of LED based on the status of the button.

2.2 Components List:

- ➢ 1 x Raspberry Pi 4
- ➢ 1 x Button
- ➢ 1 x LED
- ➢ 1 x 220Ω Resistor
- ➢ 1 x Breadboard
- ➢ Several Jumper wires

2.3 Principle:
2.2.1 Button

Buttons are a common component used to control electronic devices. They are usually used as switches to connect or disconnect circuits. Although buttons come in a variety of sizes and shapes, the one used in this experiment will be a 12mm button as shown in the following pictures. Pins pointed out by the arrows of same color are meant to be connected.
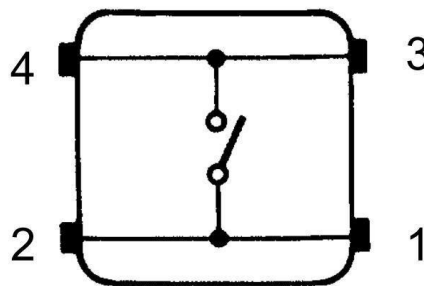
Figure: 2.1

The button we used is a normally open type button. The two contacts of a button is in the off state under the normal conditions, only when the button is pressed they are closed.

The schematic diagram we used is as follows:

Figure: 2.2

31
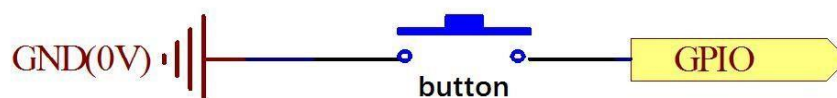
The button jitter must be happen in the process of using. The jitter waveform is as the flowing:
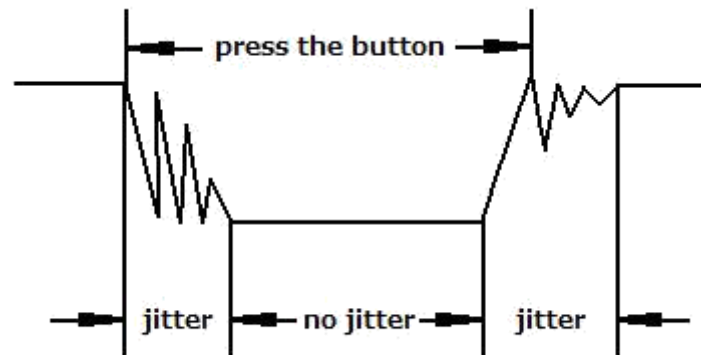

Figure: 2.3

Each time you press the button, the Raspberry Pi will think you have pressed the button many times due to the jitter of the button. We must to deal with the jitter of buttons before we use the button. We can remove the jitter of buttons through the software programming, besides, we can use a capacitance to remove the jitter of buttons. Here we introduce the software method. First, we detect whether the level of button interface is low level or high level. When the level we detected is low level, 5~10 MS delay is needed, and then detect whether the level of button interface is low or high. If the signal is low, we can confirm that the button is pressed once. You can also use a 0.1uF capacitance to clean up the jitter of buttons. The schematic diagram is shown in below:
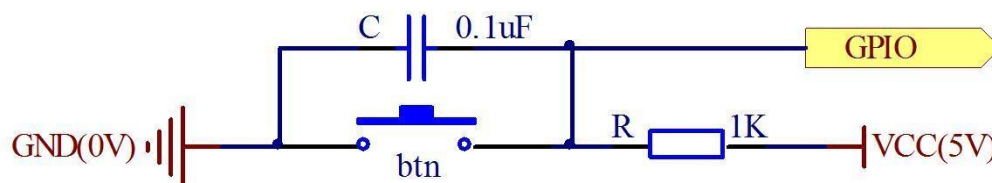

Figure: 2.4

2.2.2. Interrupt

Hardware interrupts were introduced as a way to reduce wasting the processor's valuable time in polling loops, waiting for external events. They may be implemented in hardware as a distinct system with control lines, or they may be integrated into the memory subsystem.

2.2.3 Key functions:
2.2.4.1 C user:

● **void pullUpDnControl (int pin, int pud)**

This sets the pull-up or pull-down resistor mode on the given pin, which should be set as an input. Unlike the Arduino, the BCM2835 has both pull-up a down internal resistors. The parameter pud should be; PUD_OFF, (no pull up/down), PUD_DOWN (pull to ground) or PUD_UP (pull to 3.3v). The internal pull up/down resistors have a value of approximately 50KΩ on the Raspberry Pi.

This function has no effect on the Raspberry Pi's GPIO pins when in Sys mode. If you need to activate a pull-up/pull-down, then you can do it with the gpio program in a script before you start your program.

- **int digitalRead (int pin)**

This function returns the value read at the given pin. It will be HIGH or LOW (1 or 0) depending on the logic level at the pin.

- **int wiringPiISR (int pin, int edgeType, void (*function)(void))**

This function registers a function to received interrupts on the specified pin. The edgeType parameter is either **INT_EDGE_FALLING**, **INT_EDGE_RISING**, **INT_EDGE_BOTH** or **INT_EDGE_SETUP**. If it is **INT_EDGE_SETUP** then no initialisation of the pin will happen – it's assumed that you have already setup the pin elsewhere (e.g. with the gpio program), but if you specify one of the other types, then the pin will be exported and initialised as specified. This is accomplished via a suitable call to the gpio utility program, so it need to be available.

The pin number is supplied in the current mode – native wiringPi, BCM_GPIO, physical or Sys modes.

This function will work in any mode, and does not need root privileges to work.

The function will be called when the interrupt triggers. When it is triggered, it's cleared in the dispatcher before calling your function, so if a subsequent interrupt fires before you finish your handler, then it won't be missed. (However it can only track one more interrupt, if more than one interrupt fires while one is being handled then they will be ignored)

This function is run at a high priority (if the program is run using sudo, or as root) and executes concurrently with the main program. It has full access to all the global variables, open file handles and so on.

2.2.5.2 Python user:

- **GPIO.input(channel)**

This is used for reading the value of a GPIO pin. Where channel is the channel number based on the numbering system you have specified (BOARD or BCM)). This will return either 0 / GPIO.LOW / False or 1 / GPIO.HIGH / True.

- **GPIO.add_event_detect(channel, mode)**

The event_detected( ) function is designed to be used in a loop with other things, but unlike polling it is not going to miss the change in state of an input while the CPU is busy Working on other things. This could be useful when using something like Pygame or PyQt where there is a main loop listening and responding to GUI events in a timely basis.
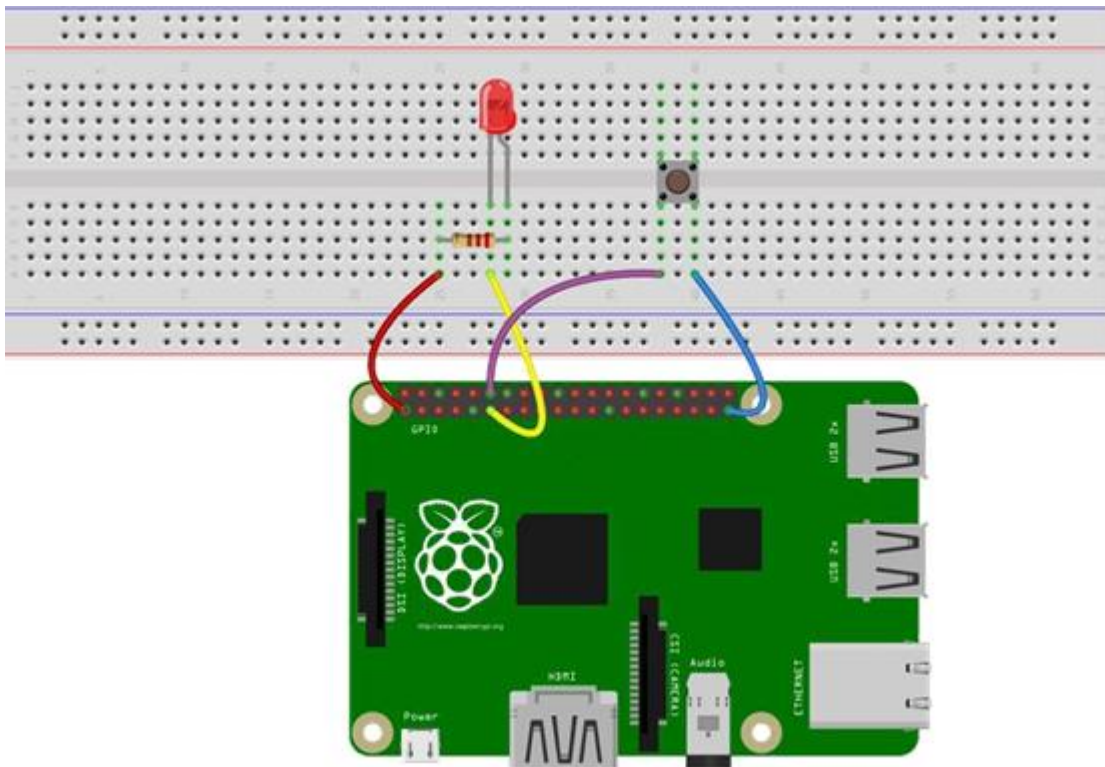
2.4 Procedures:

Step1: Build the circuit

Figure: 2.5

Step2: Program
2.5 C user:
2.1 Edit and save the code with vim or nano.

C Source Code:
/home/user/Desktop/RPI_Quick_Starter/C_Code/02_btnAndLed/btnAndLed.c

2.2 Compile the program

   $ gcc btnAndLed.c -o btnAndLed -lwiringPi

2.3 Run the program

34

$ sudo ./btnAndLed

2.6 Python user:
2.1 Edit and save the code with vim or nano.

Python Source Code:
/home/user/Desktop/RPI_Quick_Starter/Python_Code/02_btnAndLed.py

2.2 Run the program
    $ sudo python 02_btnAndLed.py

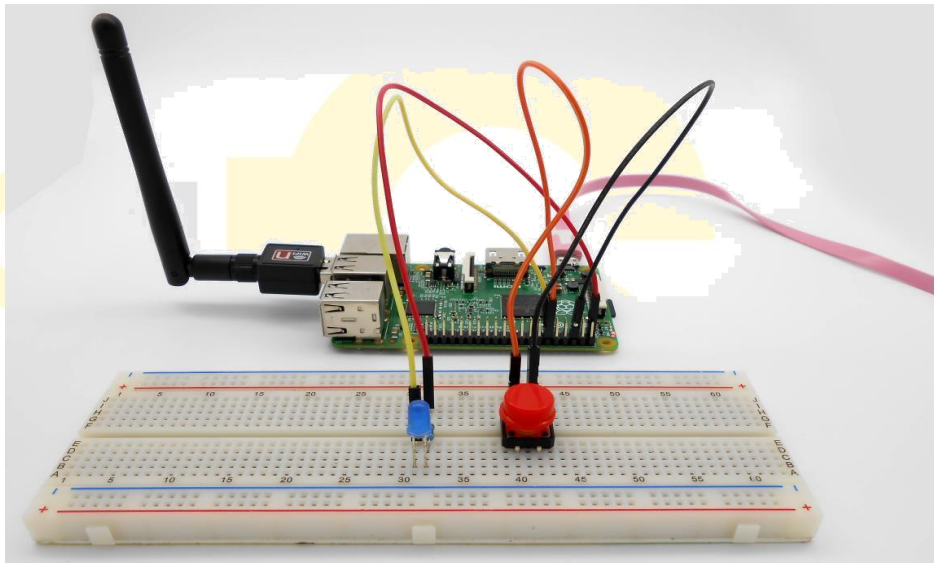Now, when you press the button, you can see the state of the LED will be toggled. (ON>OFF, OFF->ON).

Figure: 2.6

2.7 Summary:
Through this lesson, you should have learned how to use the Raspberry Pi detect an external button status, and then toggle the status of LED relying on the status of the button detected before.

Lesson-3 Controlling a RGB LED with PWM

3.1 Overview:

In this lesson, we will program the Raspberry Pi for RGB LED control, and make RGB LED emits a variety of colors of light.

3.2 Components List:

- ➢ 1 x Raspberry Pi 4
- ➢ 1 x RGB LED
- ➢ 3 x220Ω Resistor
- ➢ 1 x Breadboard
- ➢ Several Jumper wires

3.3 Principle:

RGB LEDs consist of three LEDs, one red, one green and one blue. These three colored LEDs are capable of producing any color.Tri-color LEDs with red, green, and blue emitters, in general using a four-wire connection with one common lead (anode or cathode). These LEDs can have either common anode or common cathode leads.
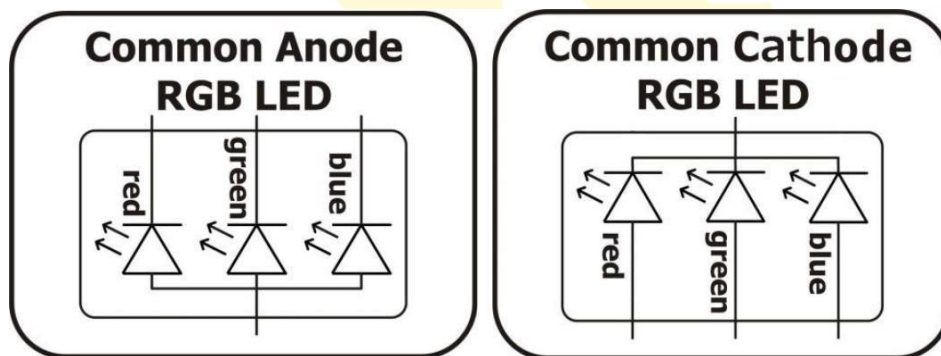

Figure: 3.1

What we used in this experiment is the common anode RGB LED. The longest pin is the common anode of three LEDs. The pin is connected to the +3.3V pin of the Raspberry Pi, and the three remaining pins are connected to the Raspberry Pi's pin11, pin12, pin13 through a current limiting resistor.

In this way, we can control the color of RGB LED by 3-channel PWM signal.

3.3.1 Key function:

- ● **int softPwmCreate (int pin, int initialValue, int pwmRange)**

This creates a software controlled PWM pin. You can use any GPIO pin and the pin numbering will be that of the wiringPiSetup() function you used. Use 100 for the pwmRange, then the value can be anything from 0 (off) to 100 (fully on) for the given pin.
The return value is 0 for success. Anything else and you should check the global error variable to see what went wrong.

- **void softPwmWrite (int pin, int value)**

This updates the PWM value on the given pin. The value is checked to be in-range and pins that haven't previously been initialised via soft PwmCreate will be silently ignored.
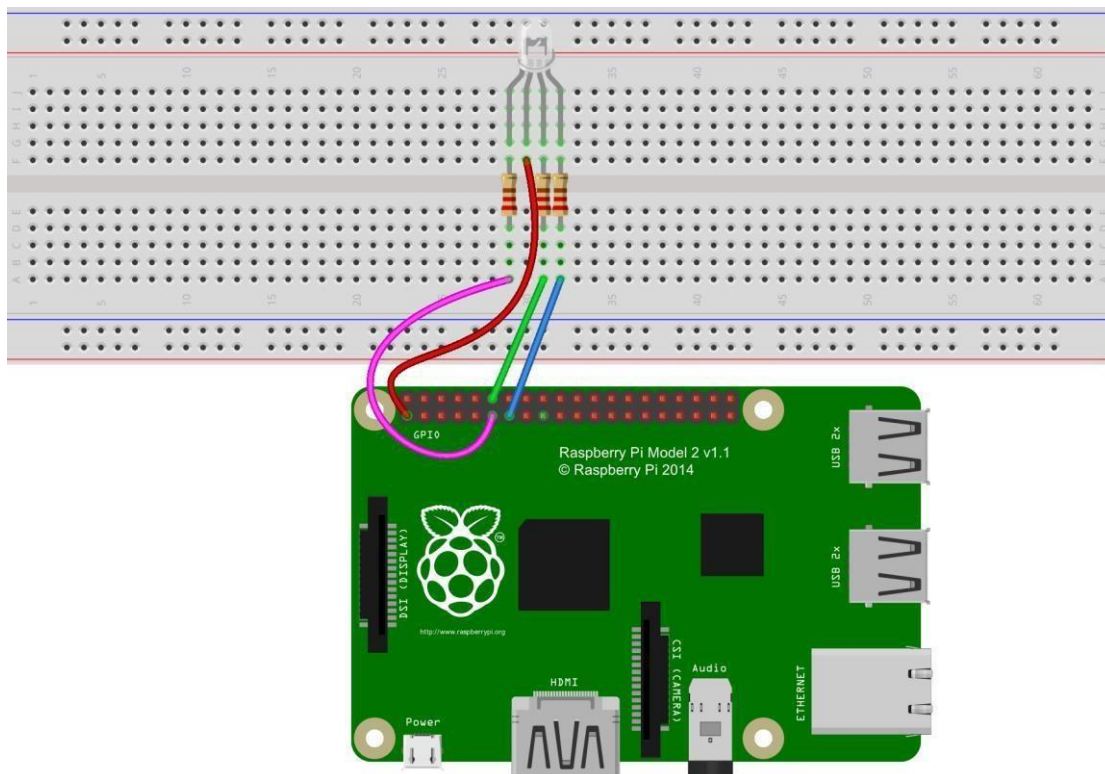
3.4 Procedures:
Step1: Build the circuit

Figure: 3.2

Step2: Program
3.5 C user:

3.1 Edit and save the code with vim or nano.

C Source Code: /home/user/Desktop/RPI_Quick_Starter/C_Code/03_rgbLed/rgbLed.c

3.2 Compile the program

```
$ gcc rgbLed.c -o rgbLed -lwiringPi -lpthread
```

37

NOTE: The compiler option '-lpthread' is essential, because the implementation of softPwm is based on linux multithreading.

3.3 Run the program

$ sudo ./rgbLed

3.6 Python user:

3.1 Edit and save the code with vim or nano.

Python Source Code: /home/user/Desktop/RPI_Quick_Starter/Python_Code/ 03_rgbLed.py

3.2 Run the program

$ sudo python 03_rgbLedLed.py

Now, you can see that the RGB LED emitting red, green, blue, yellow, white and purple light, then the RGB LED will be off, each state continues 1s, after repeating the above procedure.
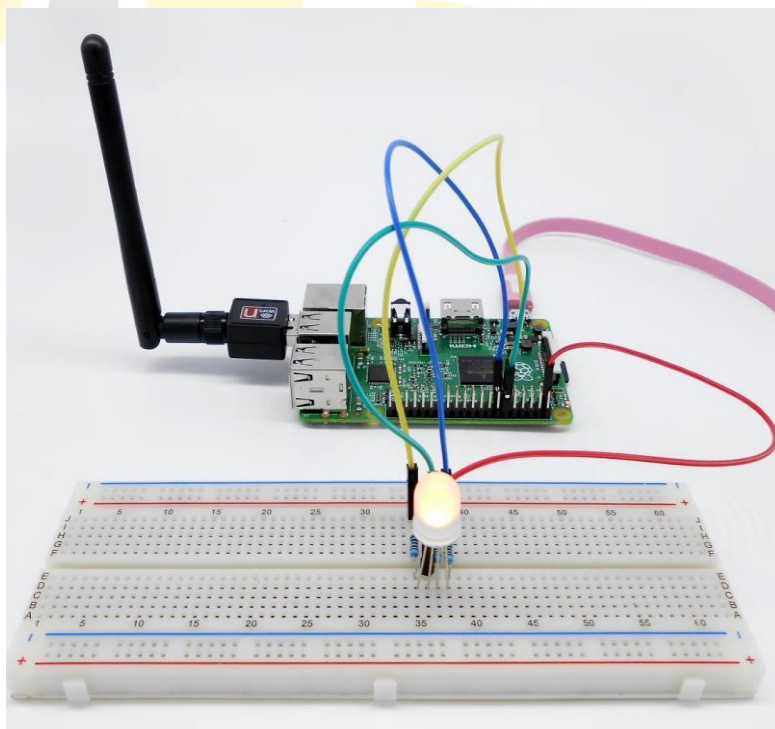

Figure: 3.2

3.7 Summary:

By learning this lesson, I believe that you have already known the principle and the programming of RGB LED. I hope you can use your imagination to achieve even more cool ideas based on this lesson.

Lesson- 4 LED Flowing Lights

4.1 Overview:

In the first class, we have learned how to make an LED blink by programming the Raspberry Pi. Today, we will use the Raspberry Pi to control 8 LEDs, so that 8 LEDs showing the result of flowing.

4.2 Components List:
- ➢ 1 x Raspberry Pi 4
- ➢ 8 x LED
- ➢ 8 x 220Ω Resistor
- ➢ 1 x Breadboard
- ➢ Several Jumper wires

4.3 Principle:
The principle of this experiment is very simple. It is very similar with the first class.
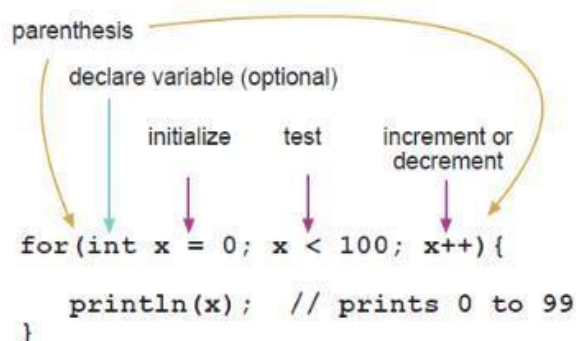
4.3.1 Key function:

● for statements

The for statement is used to repeat a block of statements enclosed in curly braces. An increment counter is usually used to increment and terminate the loop. The for statement is useful for any repetitive operation, and is often used in combination with arrays to operate on collections of data/pins.

There are three parts to the for loop header:

for (initialization; condition; increment) {

    //statement(s);

}


```
parenthesis
    declare variable (optional)
        initialize    test    increment or
                                decrement
for(int x = 0; x < 100; x++){
    println(x);   // prints 0 to 99
}
```

The initialization happens first and exactly once. Each time through the loop, the condition is tested; if it's true, the statement block, and the increment is executed, then the condition is tested again. When the condition becomes false, the loop ends.

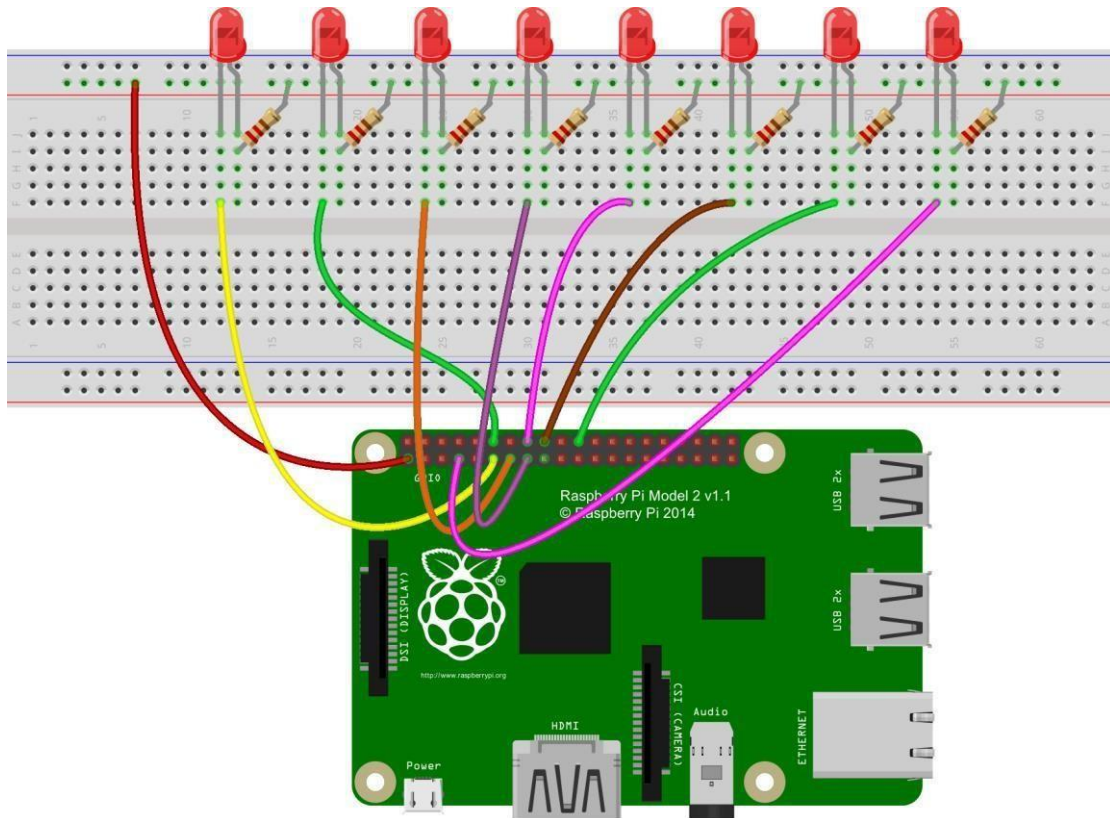## 4.4 Procedures:

Step1: Build the circuit

Figure: 4.1

Step2: Program

4.5 C user:

4.1 Edit and save the code with vim or nano.

C Source Code:
/home/user/Desktop/RPI_Quick_Starter/C_Code/04_flowingLed/flowingLed.c

4.2 Compile the program

```
$ gcc flowingLed.c -o flowingLed -lwiringPi
```

Run the program

```
$ sudo ./flowingLed
```

4.6 Python user:

4.1 Edit and save the code with vim or nano.

Python Source Code: /home/user/Desktop/RPI_Quick_Starter/Python_Code/
04_flowingLed.py

4.2 Run the program

$ sudo python 04_flowingLed.py

Now, you should see 8 LEDs are lit in sequence from the right red one to the left, next from the left to the right one. And then repeat the above phenomenon.
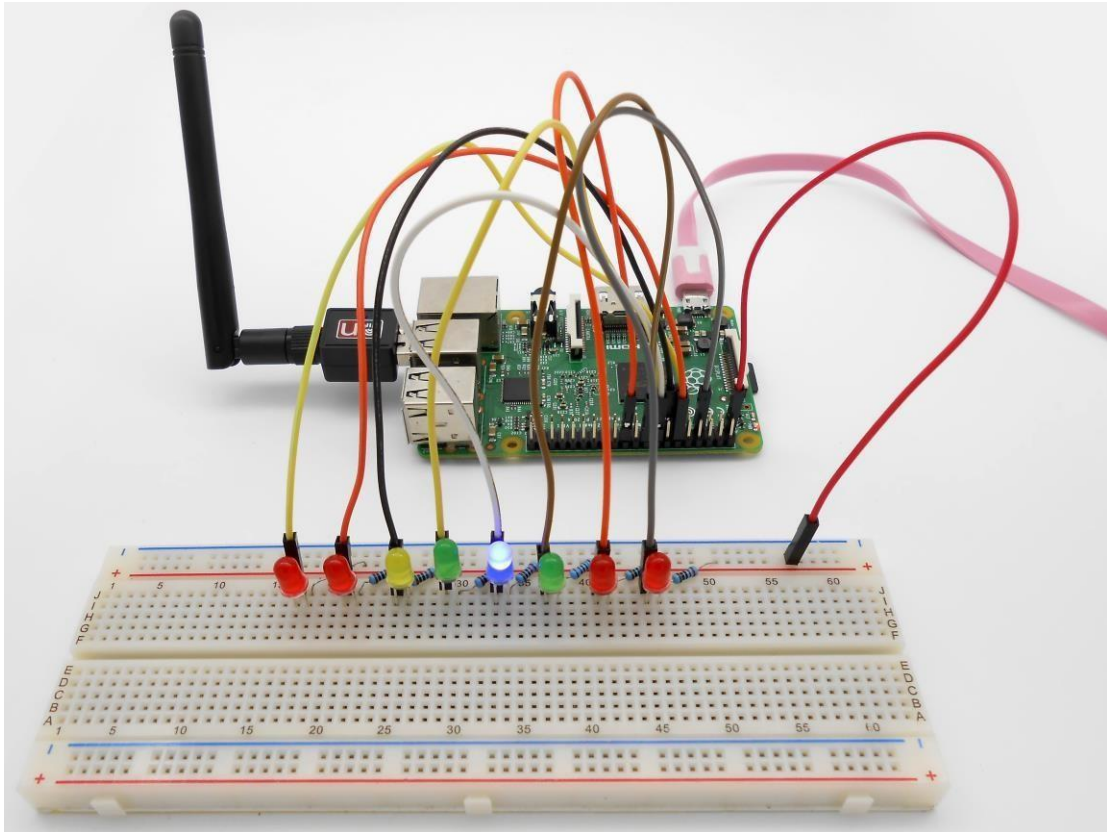

Figure: 4.2

4.7 Summary:

Through this simple and fun experiment, we have learned more skilled programming about the Raspberry Pi. In addition, you can also modify the circuit and code we provided to achieve even more dazzling effect.

THANK YOU

SUN ROBOTICS

www.sunrobotics.co.in