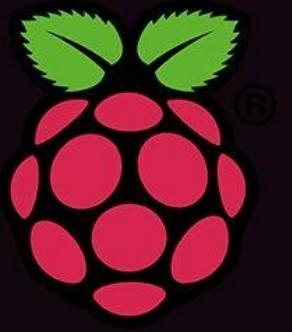


# Raspberry Pi Wireless Networks



Agus Kurniawan



# Raspberry Pi Wireless Networks



Agus Kurniawan

# **Copyright**

Raspberry Pi Wireless Networks

Agus Kurniawan

1st Edition, 2015

Copyright © 2015 Agus Kurniawan

Cover image from <https://www.flickr.com/photos/mrezafaisal/>

# Table of Contents

[Copyright](#)

[Preface](#)

[1. Preparing Development Environment](#)

[1.1 Raspberry Pi](#)

[1.2 Getting Hardware](#)

[1.3 Development Tools](#)

[1.4 Electronics Devices](#)

[1.5 Change Raspberry Pi Hostname](#)

[1.6 Raspberry Pi GPIO](#)

[1.7 Python Library for GPIO](#)

[1.8 Hello World](#)

[1.9 Further Reading](#)

[2. Wi-Fi IEEE 802.11 Networks](#)

[2.1 Getting Started](#)

[2.2 Wi-Fi USB](#)

[2.2.1 Wi-Fi USB Hardware](#)

[2.2.2 Demo: Connecting Wi-Fi Network](#)

[2.3 Wi-Fi IEEE 802.11 Wireless Module](#)

[2.3.1 Wi-Fi IEEE 802.11 Wireless Module Hardware](#)

[2.3.2 Connecting Wi-Fi Module to Raspberry Pi](#)

[2.3.3 Demo 1: Connecting to Wi-Fi Hotspot](#)

[2.3.4 Demo 2: Creating Access Points \(AP\)](#)

[2.3.5 Demo 3: Creating A Simple Web Server](#)

[3. IR Communication](#)

[3.1 Getting Started](#)

[3.2 Building IR Remote](#)

[3.2.1 Hardware Implementation](#)

[3.2.2 Configuring and Testing](#)

[3.2.3 Recording](#)

[4. Bluetooth Low Energy \(BLE\) and iBeacon](#)

[4.1 Bluetooth Low Energy \(BLE\)](#)

[4.2 Installing Bluez5](#)

[4.3 Pairing Bluetooth Devices](#)

[4.4 BLE Development and iBeacon](#)

[4.4.1 Installing Node.js](#)

[4.4.2 Bleno](#)

[4.5 Further Reading](#)

[5. Wireless Communication Using 315/433 Mhz RF Modules](#)

[5.1 Getting Started](#)

[5.2 RF Transmitter and Receiver Link Kit](#)

[5.2.1 Wiring](#)

[5.2.2 Building Transmitter Application](#)

[5.2.3 Building Receiver Application](#)

[5.2.4 Testing](#)

[6. Wireless Communication Using 2.4 GHz RF Modules](#)

[6.1 Getting Started](#)

[6.2 Getting Hardware](#)

[6.3 Wiring](#)

[6.4 Configuring SPI on Raspberry Pi](#)

[6.5 Installing nRF24L01 Module](#)

[6.6 Testing](#)

[6.6.1 Scanner Testing](#)

[6.6.2 Receiver and Transmitter Testing](#)

[7. IEEE 802.15.4 LR-WPAN Networks](#)

[7.1 Getting Started](#)

[7.2 XBee IEEE 802.15.4](#)

[7.2.1 Getting Hardware](#)

[7.2.2 Connecting XBee IEEE 802.15.4 to Raspberry Pi](#)

[7.2.3 XBee Programming for Raspberry Pi](#)

[7.3 Demo: Raspberry Pi and PC Communication Through XBee](#)

[7.3.1 XBee Configuration](#)

[7.3.2 Writing Program for Raspberry Pi and PC](#)

[7.3.3 Testing](#)

[7.4 Further Reading](#)

[8. RFID and NFC Communication](#)

[8.1 Getting Started](#)

[8.2 Getting Hardware](#)

[8.3 Reading RFID Card/Key](#)

[8.4 Reading NFC Memory](#)

[9. FM Radio Receiver](#)

[9.1 Getting Started](#)

[9.2 Getting Hardware](#)

[9.3 Wiring](#)

[9.4 Configuring I2C for Raspberry Pi](#)

[9.5 Writing Program](#)

[9.5 Testing](#)

[Source Code](#)

[Contact](#)

## Preface

This book was written to help anyone who wants to build wireless networks on Raspberry Pi. It describes all the basic elements of building and deploying wireless networks on Raspberry Pi and how to program with step-by-step approach..

Agus Kurniawan

Berlin, January 2015

## **1. Preparing Development Environment**

## 1.1 Raspberry Pi

The Raspberry Pi is a small size computer (85.60mm x 56mm x 21mm) with completed features so that you can play multimedia or run an application. This device is developed by the Raspberry Pi Foundation in UK. There are two type of Raspberry Pi:

- Model A with 256 Mb RAM
- Model A+ with 256 Mb RAM
- Model B with 512 Mb RAM
- Model B+ with 512MB RAM
- Raspberry Pi Compute Development

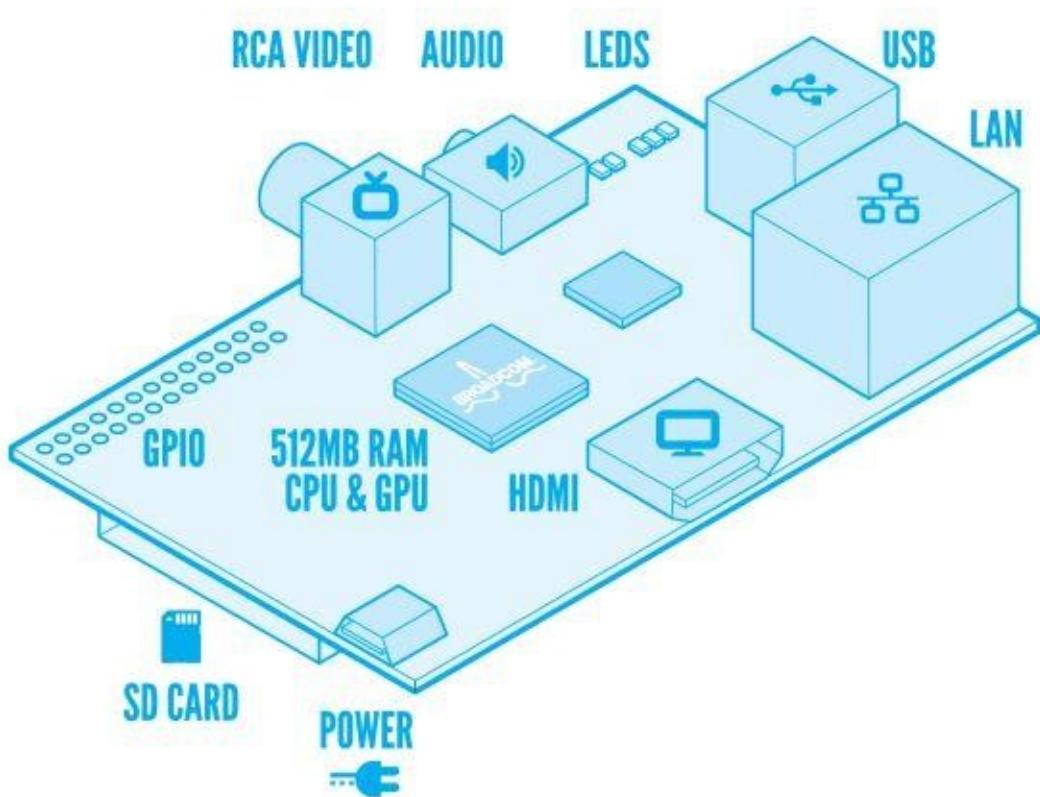
In this book, we will focus on Raspberry Pi model A and B.

The following is technical specification of Raspberry Pi device:

- SoC Broadcom BCM2835 (CPU, GPU, DSP, SDRAM)
- CPU: 700 MHz ARM1176JZF-S core (ARM11 family)
- GPU: Broadcom VideoCore IV, OpenGL ES 2.0, 1080p30 h.264/MPEG-4 AVC high-profile decoder
- Memory (SDRAM): 512 Megabytes (MiB)
- Video outputs: Composite RCA, HDMI
- Audio outputs: 3.5 mm jack, HDMI
- Onboard storage: SD, MMC, SDIO card slot
- 10/100 Ethernet RJ45 onboard network
- Storage via SD/ MMC/ SDIO card slot

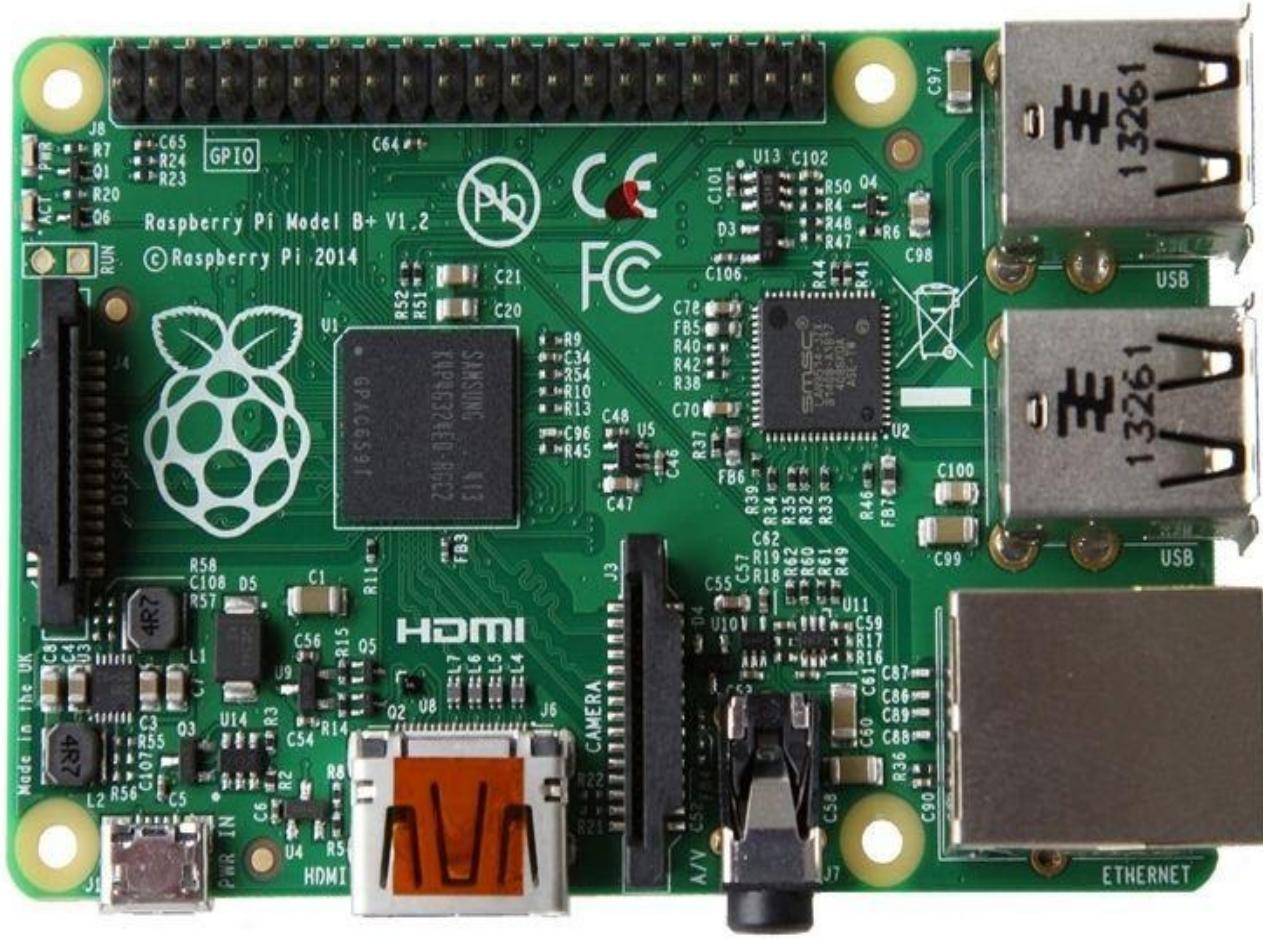
You can see Raspberry Pi device with model B on the Figure below.

# RASPBERRY PI MODEL B



(source: <http://www.raspberrypi.org/wp-content/uploads/2011/07/RaspiModelB.png>)

The following is a sample of Raspberry Pi B+ model.



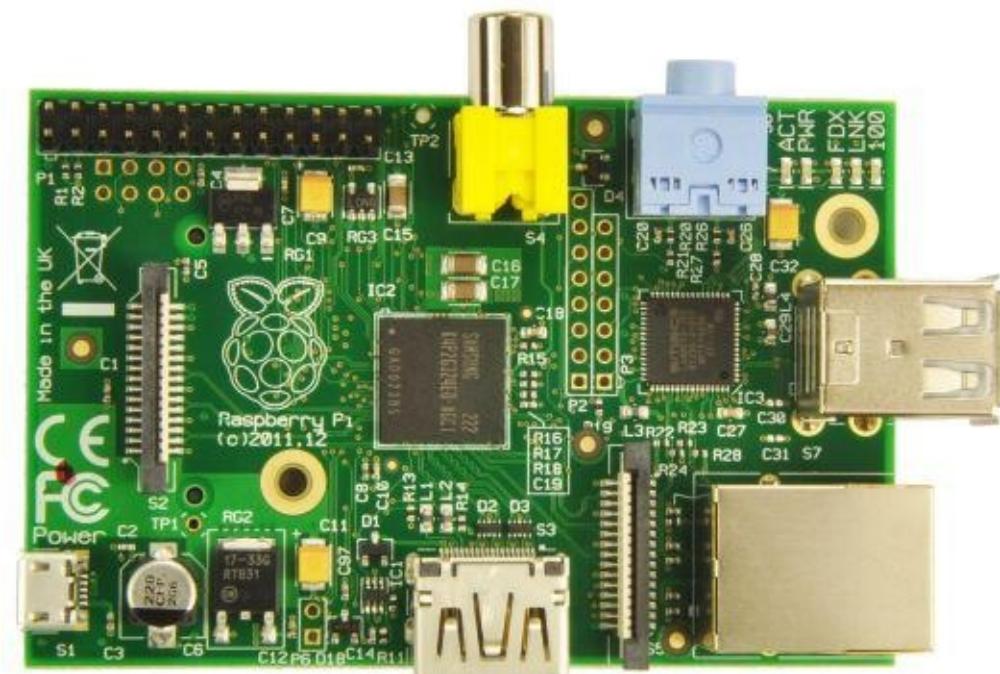
For further information, you can visit on Raspberry Pi website,  
<http://www.raspberrypi.org>.

## 1.2 Getting Hardware

How to get Raspberry Pi device?

Officially you can buy it from the official distributor

- RS, <http://uk.rs-online.com/web/generalDisplay.html?id=raspberrypi>
- Element14, <http://www.element14.com/community/community/raspberry-pi>
- Allied Electronics, <http://www.alliedelec.com/>



You also buy Raspberry Pi peripheral devices for instance, keyboard, mouse, HDMI cable, SD card, USB hub, etc.

I tried to look for buying Raspberry Pi device and found that there are another options to buy

- The Pi Hut, <http://thepihut.com>
- EXP-Tech, <http://www.exp-tech.de/Mainboards/raspberry-pi.html>
- Cooking-hack, <http://www.cooking-hacks.com/index.php/shop/raspberry-pi.html>
- Amazon, <http://www.amazon.com>
- Ebay, <http://www.ebay.com>

For testing, at least you need **two Raspberry Pi devices** to build wireless communication. I have two Raspberry Pi B model boards for testing.

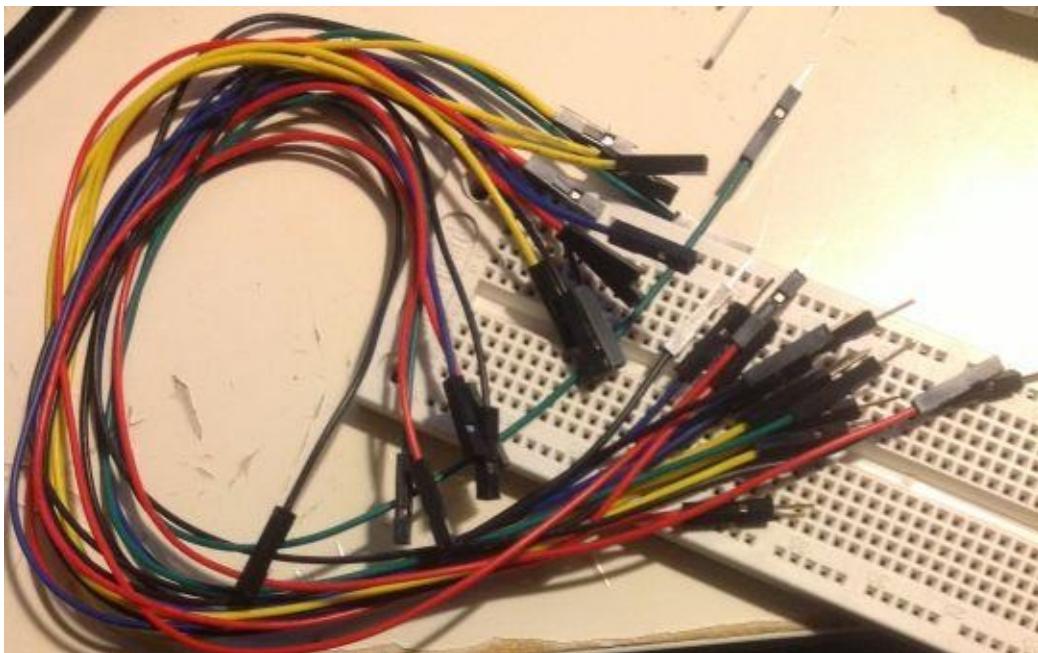
## **1.3 Development Tools**

In this book, we use C and Python to develop several programs based scenario for Wireless Networks programming. C compiler and Python runtime has installed on Raspberry Pi OS. You can use your own C or Python editor tools.

I use **Raspbian OS** for testing. You can download it on <http://www.raspberrypi.org/downloads/> and deploy this OS into your board.

## 1.4 Electronics Devices

We need electronic components to build our testing, for instance, Resistor, LED, cables and etc. I recommend you can buy electronics component kit. You find them on your local electronics shops



In addition, we also prepare wireless module devices for our testing. I will explain these devices later on each chapter in this book.

## 1.5 Change Raspberry Pi Hostname

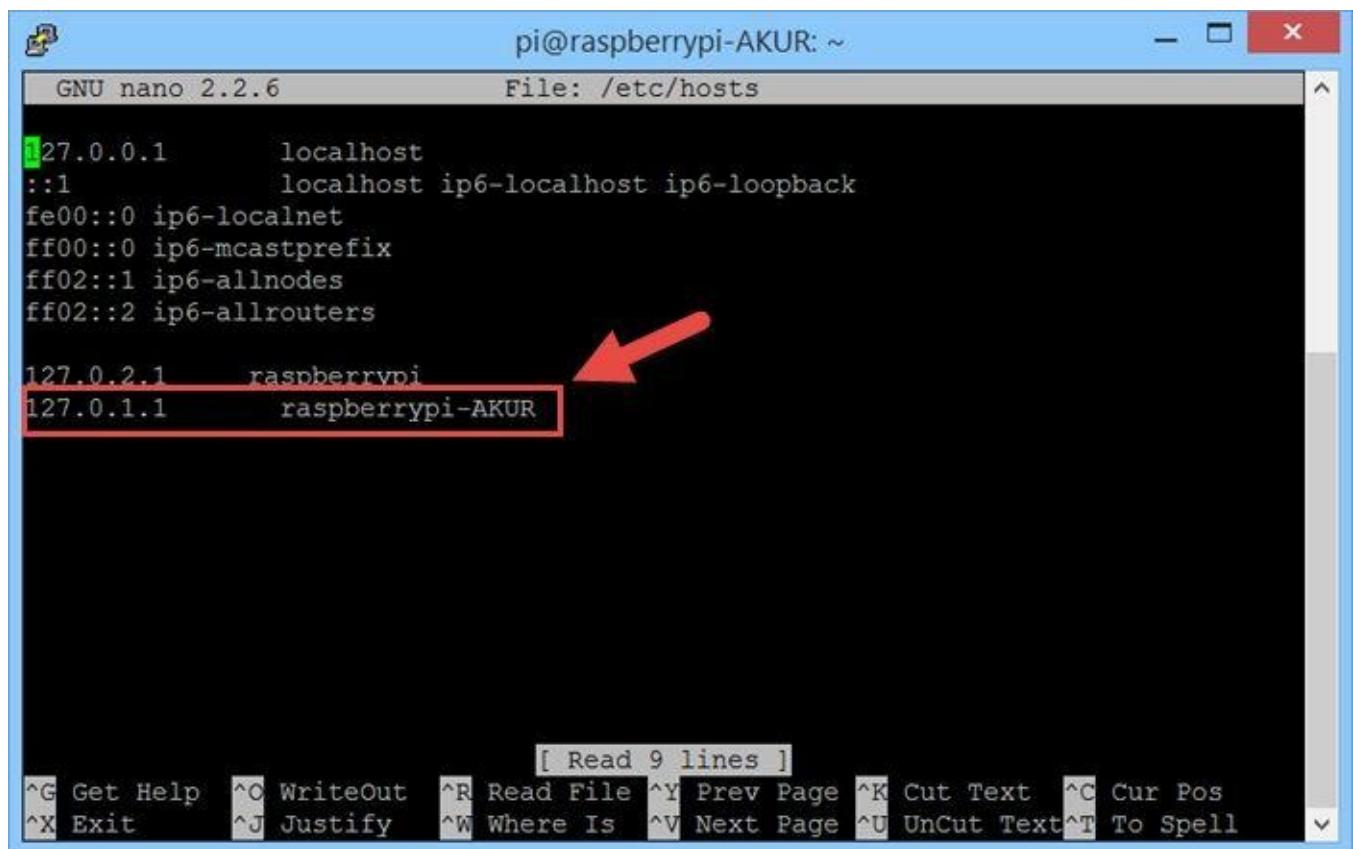
The default hostname of Raspberry Pi is raspberrypi. Pi's hostname should be changed if you have many Raspberry Pi devices to be connected to a network. Let's start to change our Raspberry Pi hostname.

First of all, we change our local IP address to refer to new hostname. Write this command on Terminal.

```
$ sudo nano /etc/hosts
```

Note: you can use vi or vim to open /etc/hosts.

Then, you get a content of /etc/hosts. Change hostname on 127.0.1.1. see Figure below.



```
pi@raspberrypi-AKUR: ~
GNU nano 2.2.6          File: /etc/hosts
127.0.0.1      localhost
::1            localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters

127.0.2.1      raspberrypi
127.0.1.1      raspberrypi-AKUR
```

[ Read 9 lines ]

**^G Get Help** **^C WriteOut** **^R Read File** **^Y Prev Page** **^K Cut Text** **^C Cur Pos**  
**^X Exit** **^J Justify** **^W Where Is** **^V Next Page** **^U UnCut Text** **^T To Spell**

Save your changes.

After that, we change hostname value on /etc/hostname.

```
$ sudo nano /etc/hostname
```

Write your hostname.



```
pi@raspberrypi-AKUR: ~
GNU nano 2.2.6          File: /etc/hostname

raspberrypi-AKUR      ← Red arrow points here

[ Read 1 line ]
^G Get Help  ^O WriteOut  ^R Read File  ^Y Prev Page  ^K Cut Text  ^C Cur Pos
^X Exit        ^J Justify   ^W Where Is   ^V Next Page  ^U UnCut Text ^T To Spell
```

Save all your changed data.

Now you save the changes and reboot your Raspberry Pi.

```
$ sudo /etc/init.d/hostname.sh  
$ sudo reboot
```

After rebooted, you can verify Raspberry Pi hostname.

```
$ hostname
```

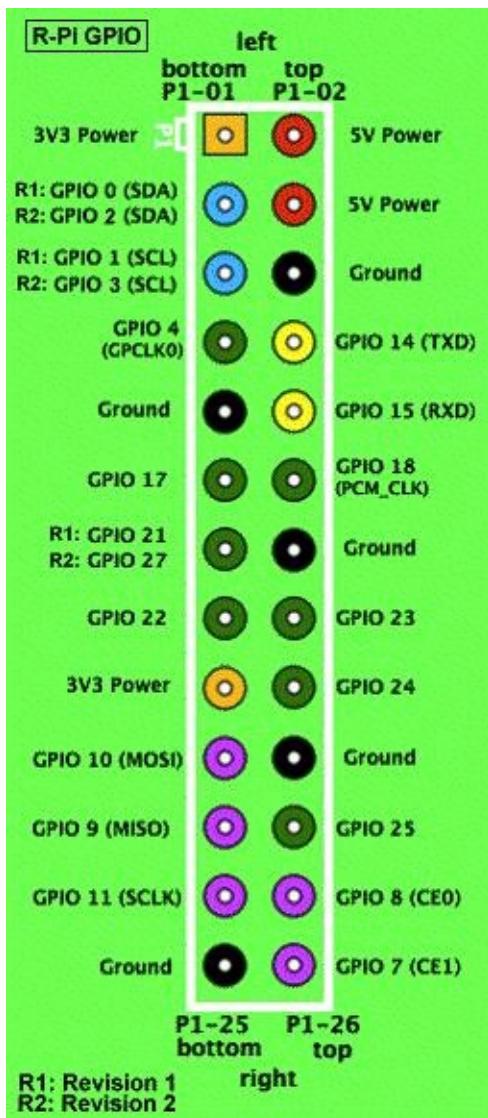
```
pi@raspberrypi-AKUR: ~
pi@raspberrypi-AKUR ~ $ hostname
raspberrypi-AKUR
pi@raspberrypi-AKUR ~ $
```

## 1.6 Raspberry Pi GPIO

The first thing we should know is to understand GPIO on Raspberry Pi. General Purpose In put/Output (GPIO) is a flexible software-controlled digital signal. You can define input/output easily. Further information about GPIO, you can read it on <http://en.wikipedia.org/wiki/GPIO>.

Based on Raspberry Pi specification, you can read GPIO pins on [http://elinux.org/RPi\\_Low-level\\_peripherals#General\\_Purpose\\_Input.2FOutput .28GPIO.29](http://elinux.org/RPi_Low-level_peripherals#General_Purpose_Input.2FOutput .28GPIO.29).

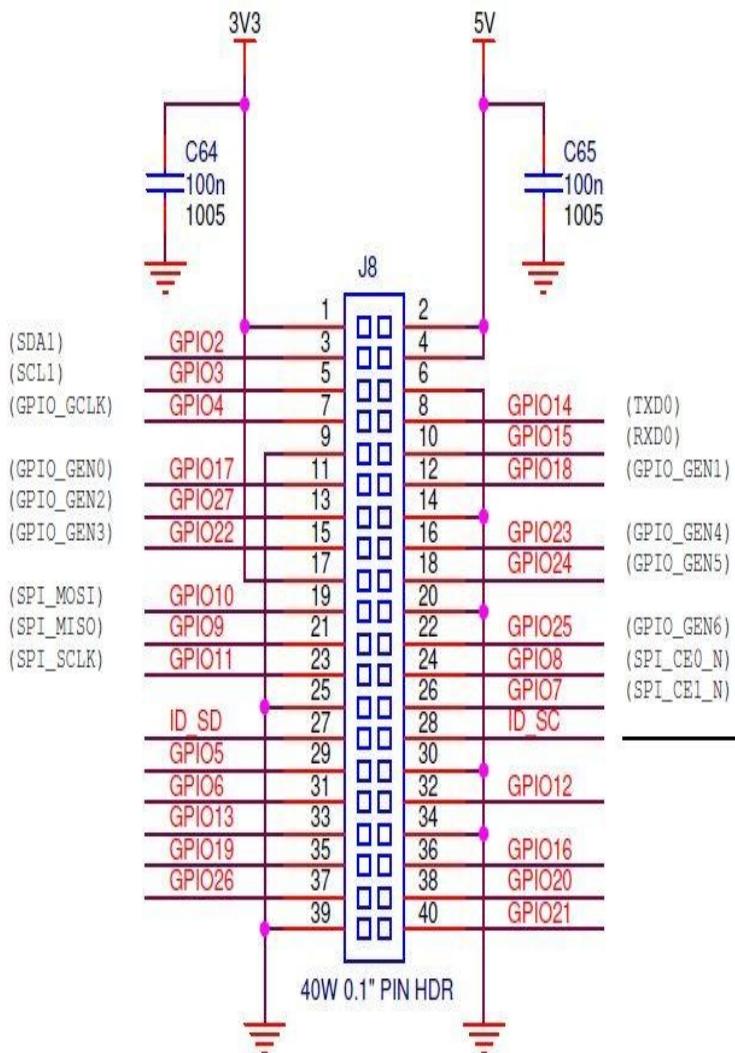
The following GPIO pins on Raspberry Pi revision 1 and 2.





3.3 V	5 V
I2C0_SDA	5 V
I2C0_SCL	GND
GPIO 4	UART0_TXD
GND	UART0_RXD
GPIO 17	GPIO 18
PCM_DIN	GND
GPIO 22	GPIO 23
3.3 V	GPIO 24
SPI0_MOSI	GND
SPI0_MISO	GPIO 25
SPI0_SCLK	SPI0_CE0_N
GND	SPI0_CE1_N

If you have Raspberry Pi B+ model, you can read GPIO schema on the following Figure below.



### ID\_SD and ID\_SC PINS:

These pins are reserved for ID EEPROM.

At boot time this I2C interface will be interrogated to look for an EEPROM that identifies the attached board and allows automagic setup of the GPIOs (and optionally, Linux drivers).

***DO NOT USE these pins for anything other than attaching an I2C ID EEPROM. Leave unconnected if ID EEPROM not required.***

(Source: <http://www.raspberrypi.org/products/model-b-plus/> )

## 1.7 Python Library for GPIO

In this chapter, we will focus on Raspberry Pi GPIO programming. There are many Raspberry Pi GPIO libraries you can choose. I used RPi.GPIO, <http://pypi.python.org/pypi/RPi.GPIO>. How to install?

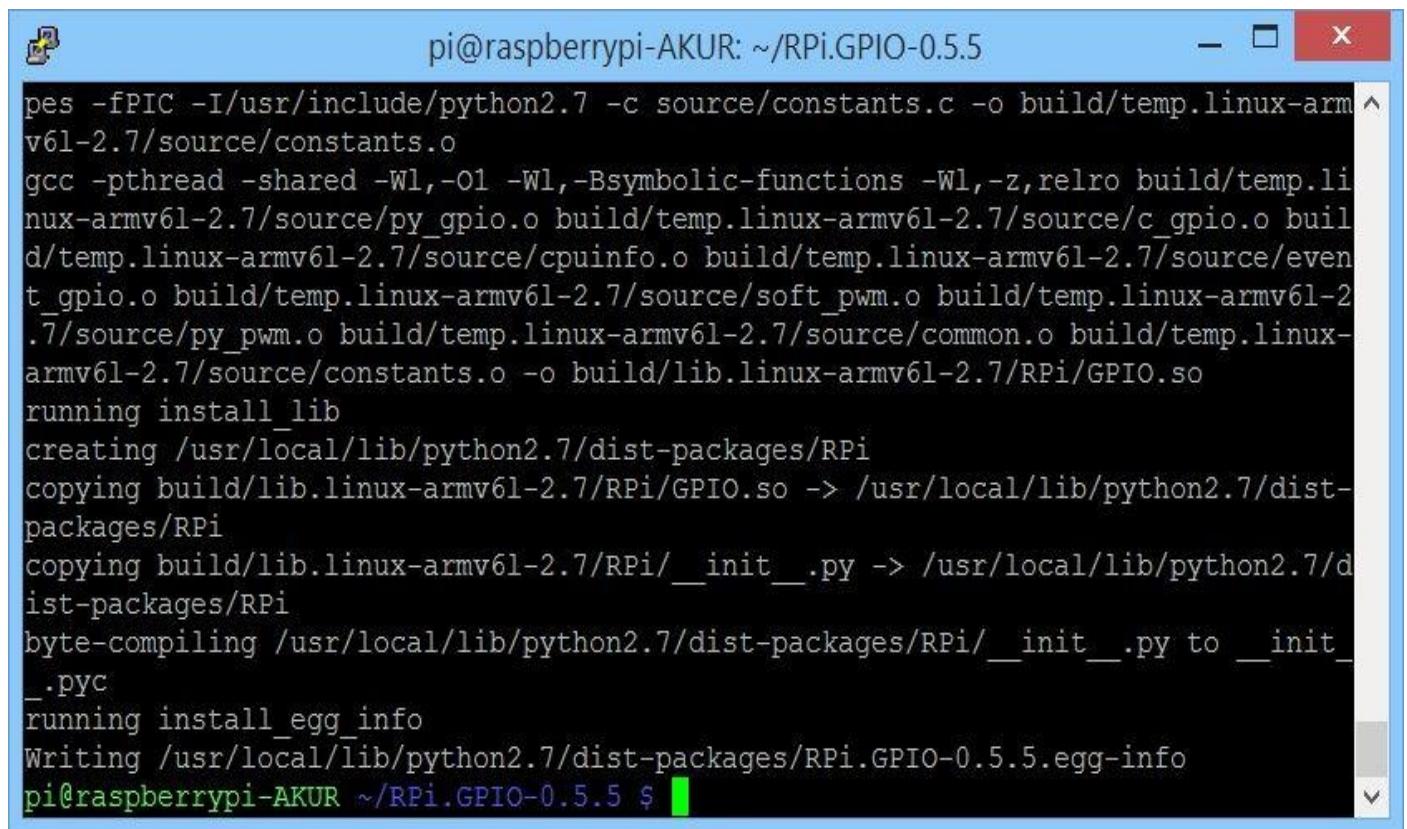
For illustration, we will install RPi.GPIO 0.5.5. Firstly, we need Python development library. Type the following command.

```
$ sudo apt-get install python-dev
```

Now you can download RPi.GPIO and install it.

```
$ wget $ https://pypi.python.org/packages/source/R/RPi.GPIO/RPi.GPIO-0.5.5.tar.gz
$ tar -xvzf RPi.GPIO-0.5.5.tar.gz
$ cd RPi.GPIO-0.5.5/
$ sudo python setup.py install
```

Make sure your Raspberry Pi already connected to Internet network.

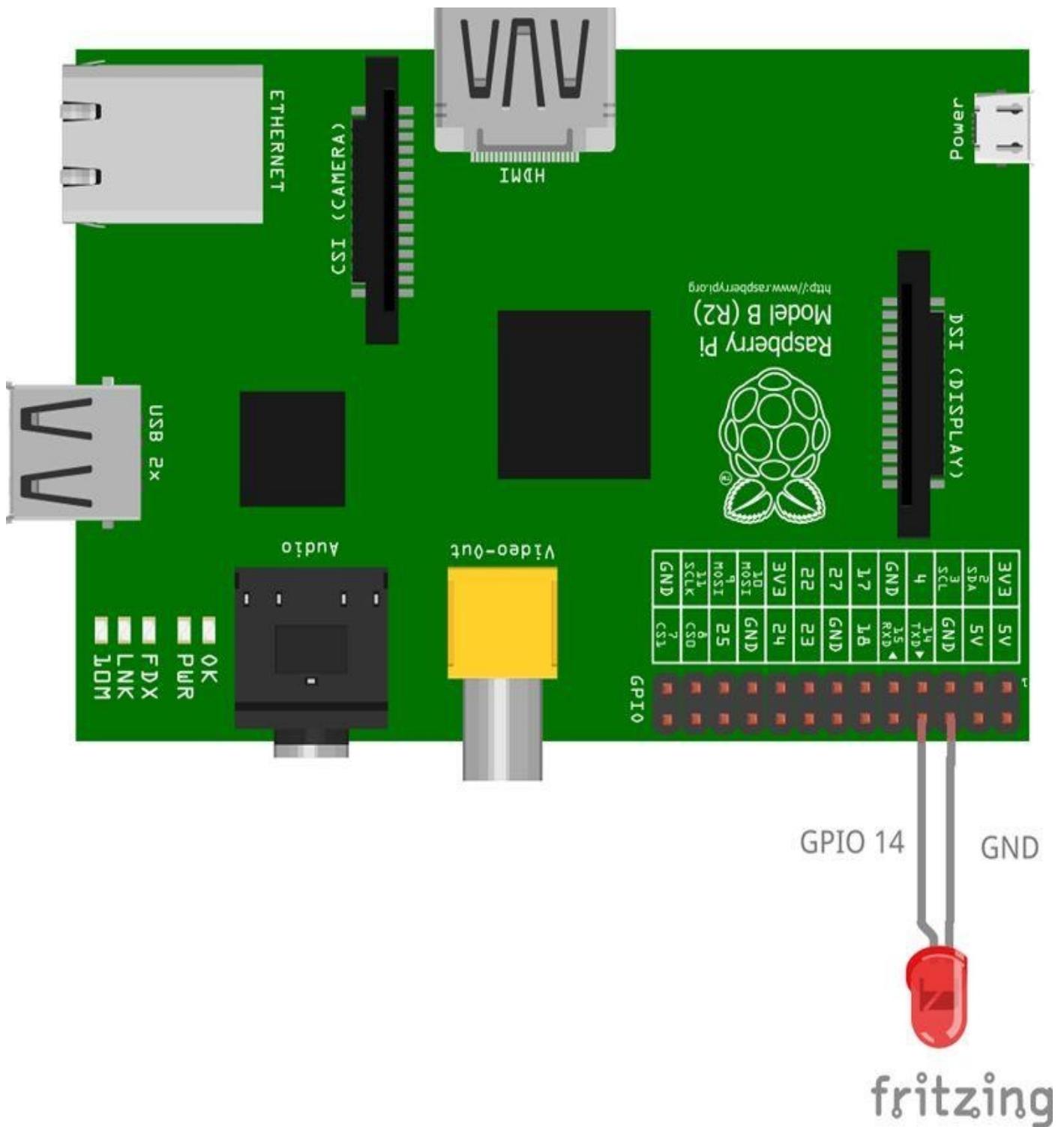


```
pi@raspberrypi-AKUR: ~/RPi.GPIO-0.5.5
pi@raspberrypi-AKUR:~/RPi.GPIO-0.5.5$ gcc -fPIC -I/usr/include/python2.7 -c source/constants.c -o build/temp.linux-armv6l-2.7/source/constants.o
pi@raspberrypi-AKUR:~/RPi.GPIO-0.5.5$ gcc -pthread -shared -Wl,-O1 -Wl,-Bsymbolic-functions -Wl,-z,relro build/temp.linux-armv6l-2.7/source/py_gpio.o build/temp.linux-armv6l-2.7/source/c_gpio.o build/temp.linux-armv6l-2.7/source/cpuinfo.o build/temp.linux-armv6l-2.7/source/even_t_gpio.o build/temp.linux-armv6l-2.7/source/soft_pwm.o build/temp.linux-armv6l-2.7/source/py_pwm.o build/temp.linux-armv6l-2.7/source/common.o build/temp.linux-armv6l-2.7/source/constants.o -o build/lib.linux-armv6l-2.7/RPi.GPIO.so
pi@raspberrypi-AKUR:~/RPi.GPIO-0.5.5$ running install_lib
pi@raspberrypi-AKUR:~/RPi.GPIO-0.5.5$ creating /usr/local/lib/python2.7/dist-packages/RPi
pi@raspberrypi-AKUR:~/RPi.GPIO-0.5.5$ copying build/lib.linux-armv6l-2.7/RPi.GPIO.so -> /usr/local/lib/python2.7/dist-packages/RPi
pi@raspberrypi-AKUR:~/RPi.GPIO-0.5.5$ copying build/lib.linux-armv6l-2.7/RPi/__init__.py -> /usr/local/lib/python2.7/dist-packages/RPi
pi@raspberrypi-AKUR:~/RPi.GPIO-0.5.5$ byte-compiling /usr/local/lib/python2.7/dist-packages/RPi/__init__.py to __init__.pyc
pi@raspberrypi-AKUR:~/RPi.GPIO-0.5.5$ running install_egg_info
pi@raspberrypi-AKUR:~/RPi.GPIO-0.5.5$ Writing /usr/local/lib/python2.7/dist-packages/RPi.GPIO-0.5.5.egg-info
pi@raspberrypi-AKUR:~/RPi.GPIO-0.5.5$
```

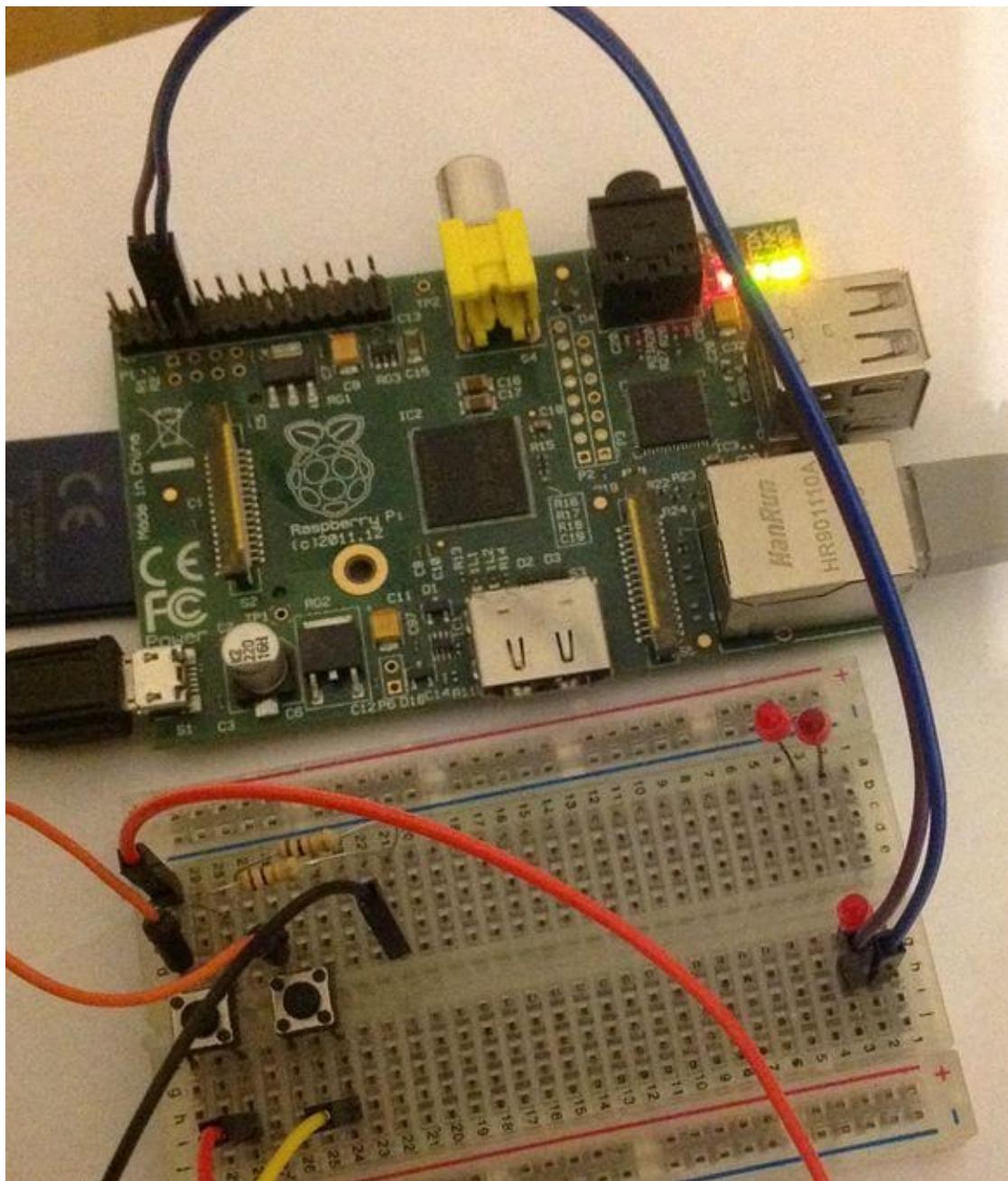
## 1.8 Hello World

In this section, we create a simple Python app to turn on/off a LED or blink app. You need a LED and cables which are connected to GPIO pins. To use Raspberry Pi GPIO, you must install external Python library. You can read how to install on previous section.

Firstly your LED is connected to GPIO pin 14 and GND. You can see this schema on the following figure.



The following is hardware implementation on Raspberry Pi.



Now you can write code to turn on/off a LED. Write the following code:

```
import RPi.GPIO as GPIO
import time

led_pin = 14
GPIO.setmode(GPIO.BCM)
GPIO.setup(led_pin, GPIO.OUT)

try:
    while 1:
        print("turn on")
        GPIO.output(led_pin, True)
        time.sleep(2)
        print("turn off")
        GPIO.output(led_pin, False)
        time.sleep(2)
```

```
print("turn on")
GPIO.output(led_pin, True)
time.sleep(2)
print("turn off")
GPIO.output(led_pin, False)

except KeyboardInterrupt:
    GPIO.cleanup()

print("done")
```

Save this code as file, called **helloworld.py**.

You can run this file by writing the following command.

```
$ sudo python helloworld.py
```

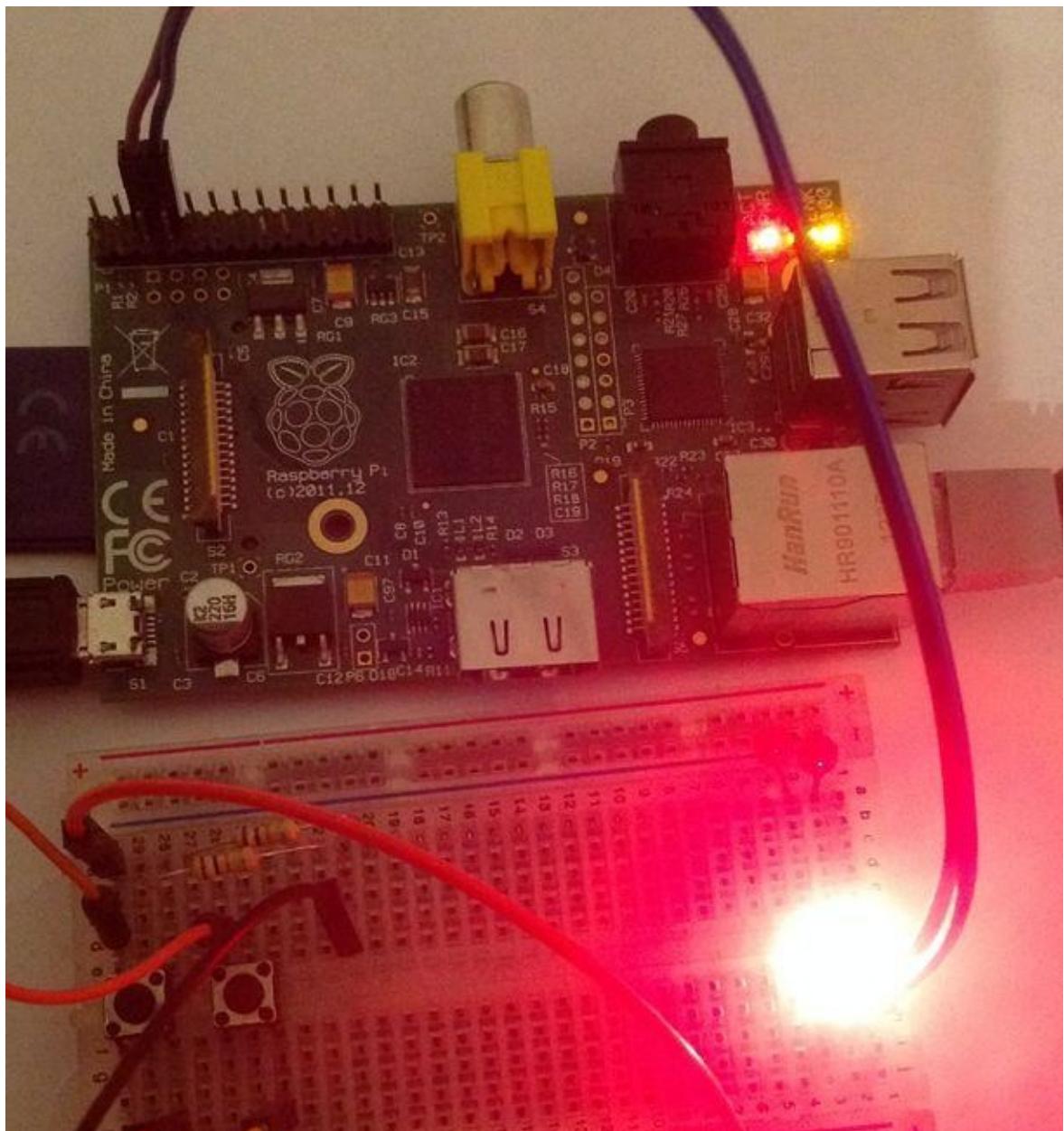
If success, you can see the output, shown in Figure below.



The screenshot shows a terminal window titled "pi@raspberrypi-AKUR: ~/ftp/files/py". The terminal displays the following command and its output:

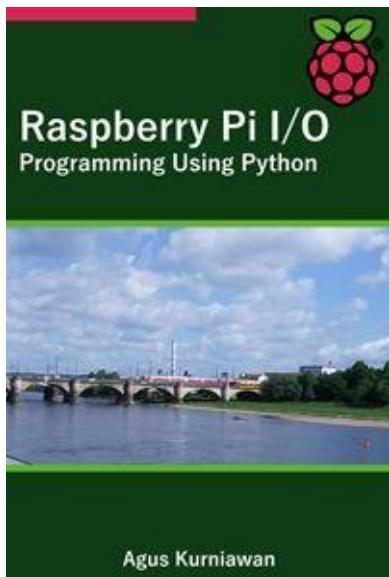
```
pi@raspberrypi-AKUR:~/ftp/files/py $ ls
helloworld.py
pi@raspberrypi-AKUR:~/ftp/files/py $ sudo python helloworld.py
turn on
turn off
turn on
turn off
turn on
turn off
turn on
```

On your Raspberry Pi, you can see the LED is blinking.



## 1.9 Further Reading

In this book, I don't explain about basic I/O programming on Raspberry Pi. You can read this topic from any website or several books related to the topic. I have already written about **Raspberry Pi I/O Programming Using Python**, <http://blog.aguskurniawan.net/post/Raspberry-Pi-IO-Programming-using-Python.aspx>. You can use this book to get started with Raspberry Pi I/O programming.



## **2. Wi-Fi IEEE 802.11 Networks**

This chapter explains how to build a Wi-Fi IEEE 802.11 network and write programs to access this network.

## 2.1 Getting Started

Raspberry Pi B and B+ models only provides wired Ethernet. If you want to connect your Raspberry Pi to Wi-Fi IEEE 802.11 networks (hotspot), you can use Wi-Fi USB or Wifi wireless module. In this chapter, we explore how Raspberry Pi connect to Wi-Fi network (IEEE 802.11).

## 2.2 Wi-Fi USB

In this section, we try to connect our Raspberry pi to Wi-Fi networks via Wi-Fi USB.

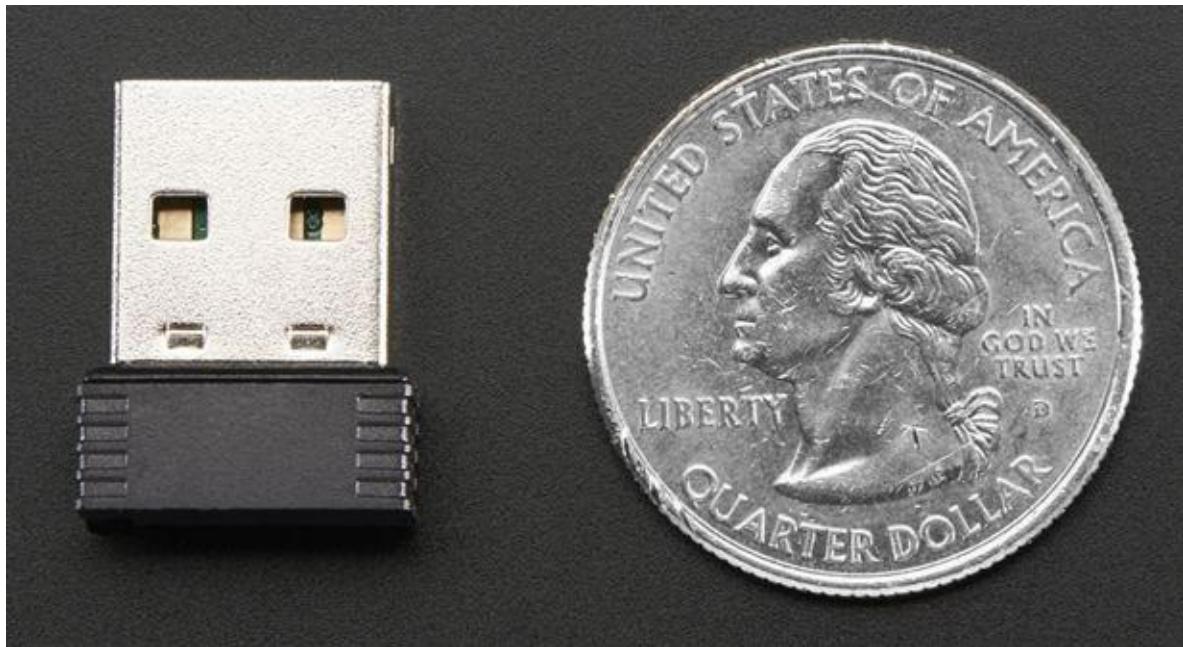
### 2.2.1 Wi-Fi USB Hardware

You can get Wi-Fi USB based on your interest, model and budget. You can find it on [http://elinux.org/RPi\\_USB\\_Wi-Fi\\_Adapters](http://elinux.org/RPi_USB_Wi-Fi_Adapters) . The following is a list of Wi-Fi USB samples:

- Miniature WiFi (802.11b/g/n) Module, <http://www.adafruit.com/product/814>
- WiFi Dongle - Nano USB, <https://www.modmypi.com/wireless-usb-1n-nano-adaptor-802.11N-wifi-dongle>

You can also find it on Amazon.com, search with keyword “wifi usb raspberry pi” , [http://www.amazon.com/s/ref=nb\\_sb\\_ss\\_i\\_0\\_13?url=search-alias%3Daps&field-keywords=wifi+usb+raspberry+pi&sprefix=wifi+usb+rasp%2Caps%2C268](http://www.amazon.com/s/ref=nb_sb_ss_i_0_13?url=search-alias%3Daps&field-keywords=wifi+usb+raspberry+pi&sprefix=wifi+usb+rasp%2Caps%2C268) .

Several Wi-Fi USB hardware model can be seen in Figure below.

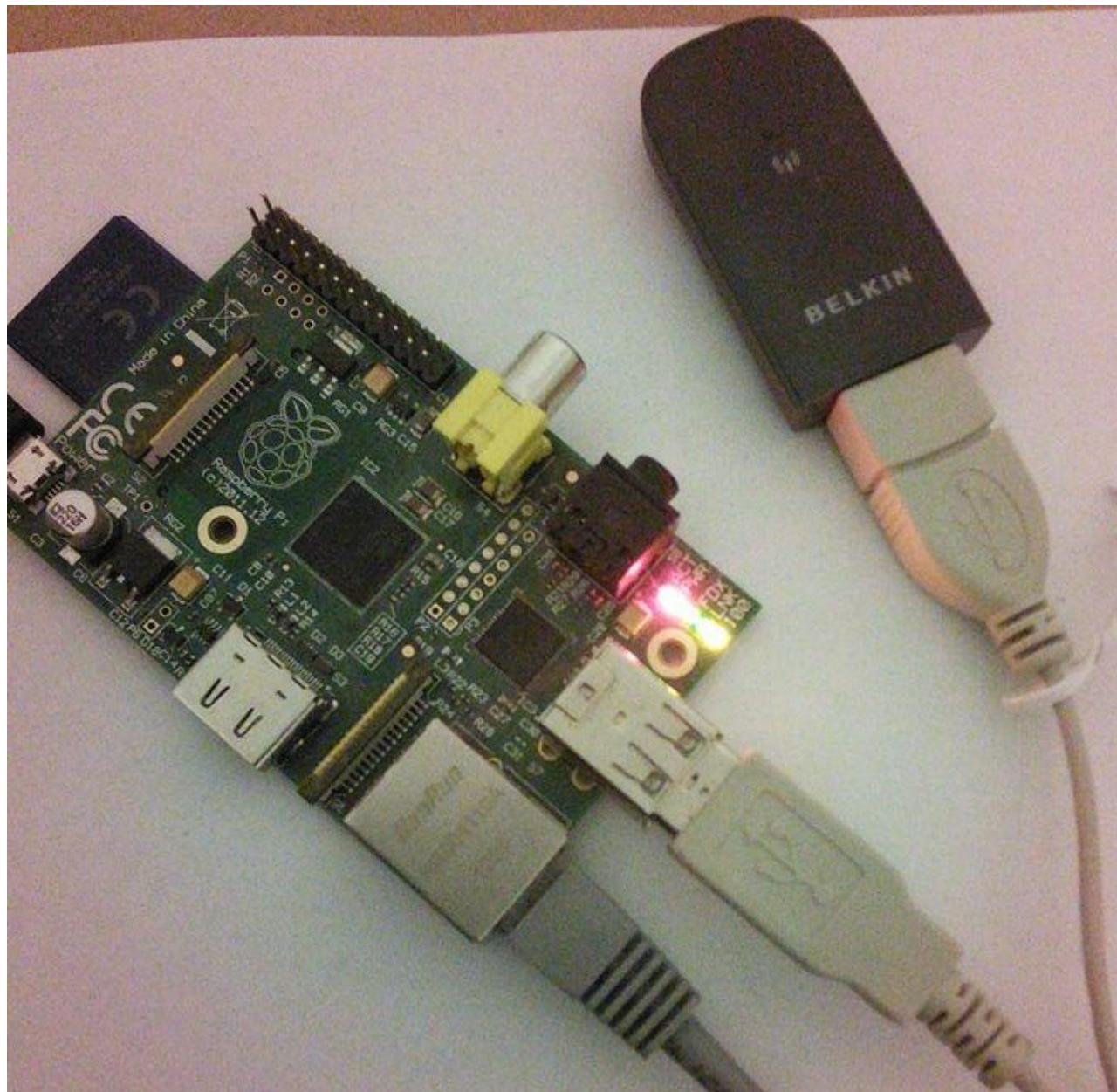




I use WIFI dongle from Belkin. My Raspberry Pi detected this WIFI adapter.

## 2.2.2 Demo: Connecting Wi-Fi Network

In this section, I want to show you how to connect your Wi-Fi USB via Raspberry Pi desktop. Firstly, you connect your Wi-Fi USB to Raspberry Pi.



You must install **wireless-tools** if you don't have it. Install it via Terminal.

```
$ sudo apt-get install wireless-tools
```

Then, click **WiFi Config** icon on desktop.



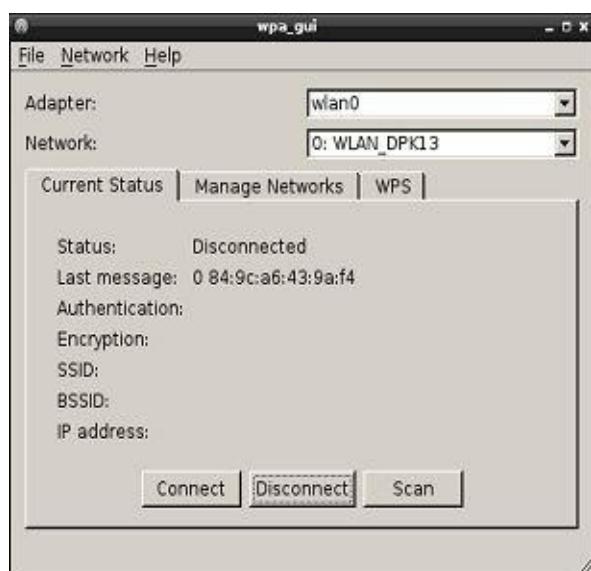
WiFi Config



LXTerminal



You will get a dialog as follows.

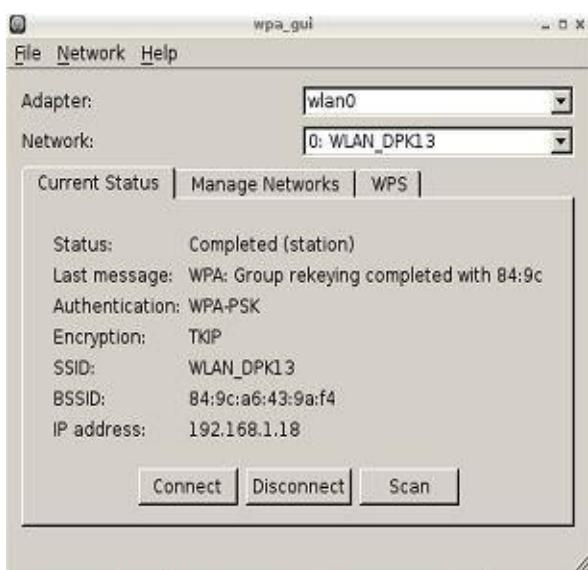


Click **Scan** button to get a list of available hotspot. Select a SSID.

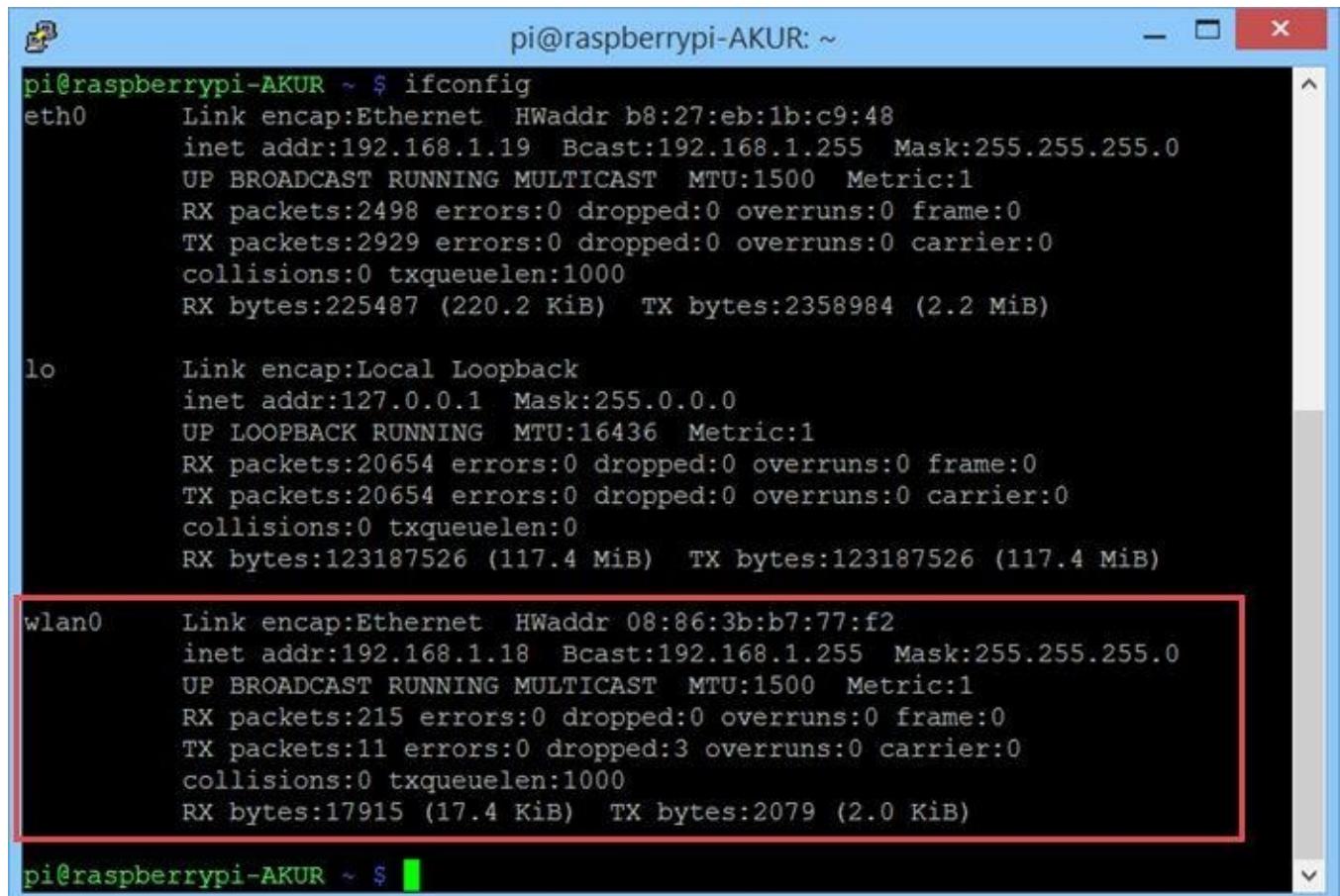
SSID	BSSID	frequency	signal
WLAN-E6E460	5c:7d:5e:fb:... 2452		-212 dBm
WLAN_DPK13	84:9c:a6:43:... 2452		-156 dBm
FRITZ!Box 7362 SL	34:31:c4:37:... 2462		-212 dBm
EasyBox-A9CA55	74:31:70:a9:... 2432		-213 dBm
EasyBox-6E7026	1c:c6:3c:6e:... 2427		-194 dBm
Dino	00:1c:28:41:... 2462		-213 dBm
AWS	c0:4a:00:79:... 2417		-214 dBm

[|] Scan Close

Fill authentication or pin if your hotspot has security input. After that, you click **Connect** button to connect to hotspot.



If success, you verify using command **ifconfig** on Terminal. You see wlan0 and its IP Address.



```
pi@raspberrypi-AKUR: ~ $ ifconfig
eth0      Link encap:Ethernet HWaddr b8:27:eb:1b:c9:48
          inet addr:192.168.1.19 Bcast:192.168.1.255 Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:2498 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2929 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:225487 (220.2 KiB) TX bytes:2358984 (2.2 MiB)

lo       Link encap:Local Loopback
          inet addr:127.0.0.1 Mask:255.0.0.0
          UP LOOPBACK RUNNING MTU:16436 Metric:1
          RX packets:20654 errors:0 dropped:0 overruns:0 frame:0
          TX packets:20654 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:123187526 (117.4 MiB) TX bytes:123187526 (117.4 MiB)

wlan0    Link encap:Ethernet HWaddr 08:86:3b:b7:77:f2
          inet addr:192.168.1.18 Bcast:192.168.1.255 Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:215 errors:0 dropped:0 overruns:0 frame:0
          TX packets:11 errors:0 dropped:3 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:17915 (17.4 KiB) TX bytes:2079 (2.0 KiB)

pi@raspberrypi-AKUR: ~ $
```

Now you can access your Wi-Fi network, for instance, surfing the Internet.

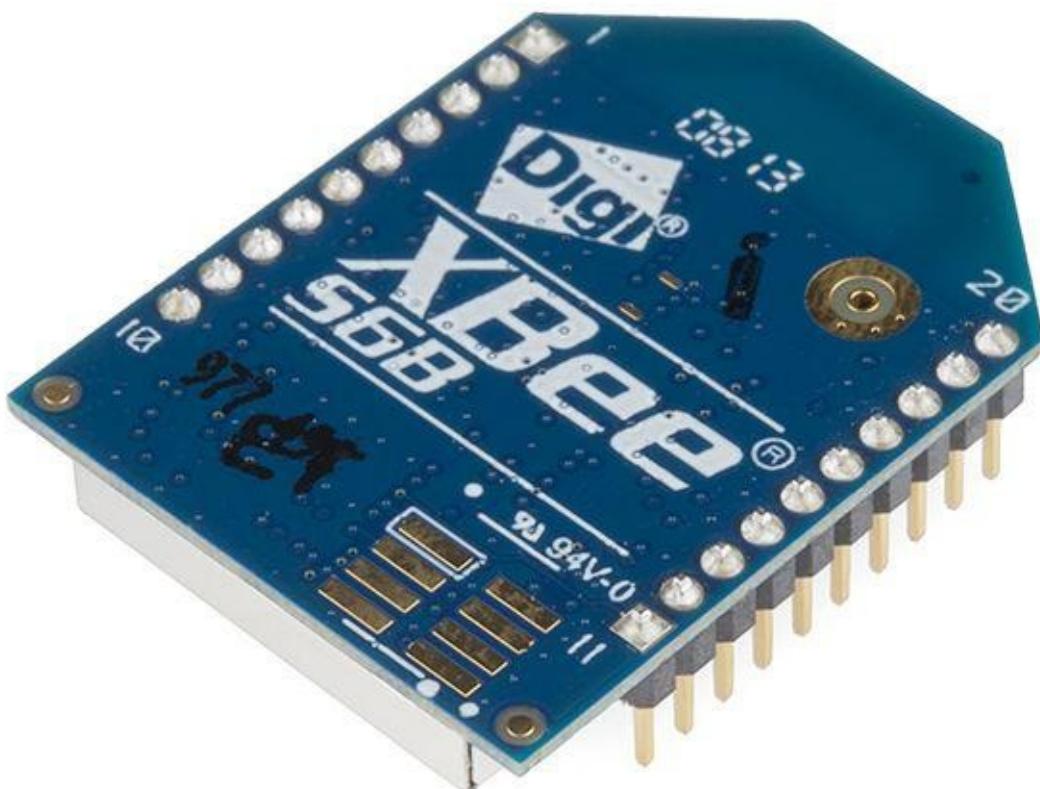
## 2.3 Wi-Fi IEEE 802.11 Wireless Module

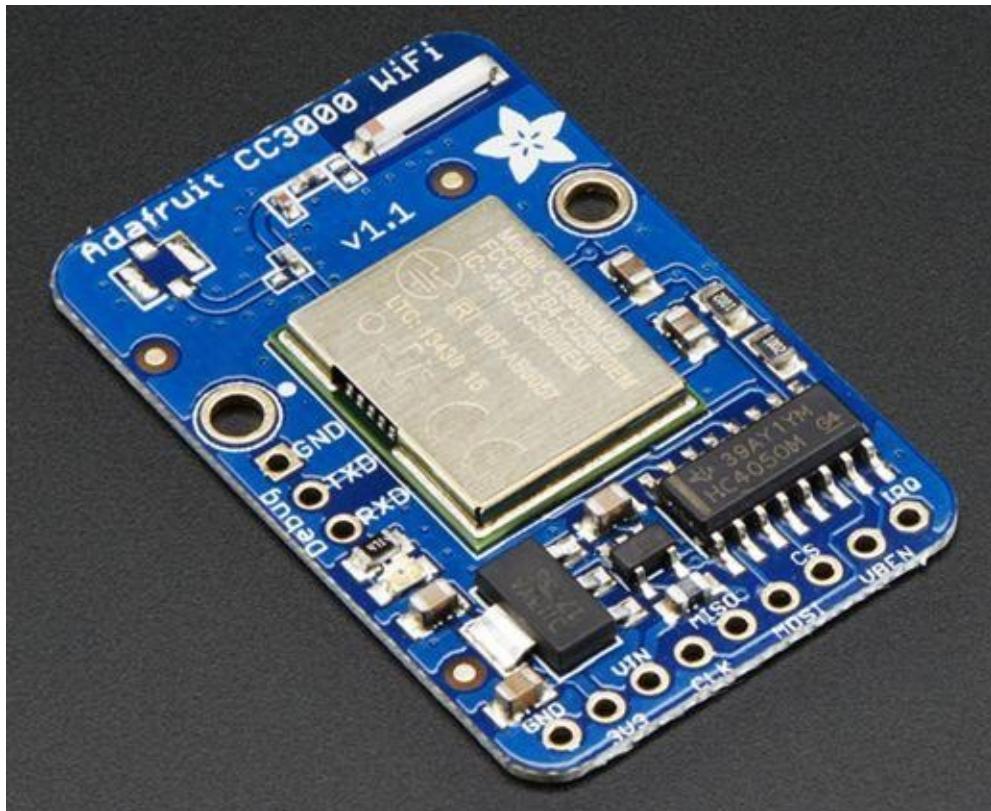
We can connect to Wi-Fi network using wireless module based IEEE 802.11. In this section, we try to use Wi-Fi module which is be attached to Raspberry Pi to connect Wi-Fi network.

### 2.3.1 Wi-Fi IEEE 802.11 Wireless Module Hardware

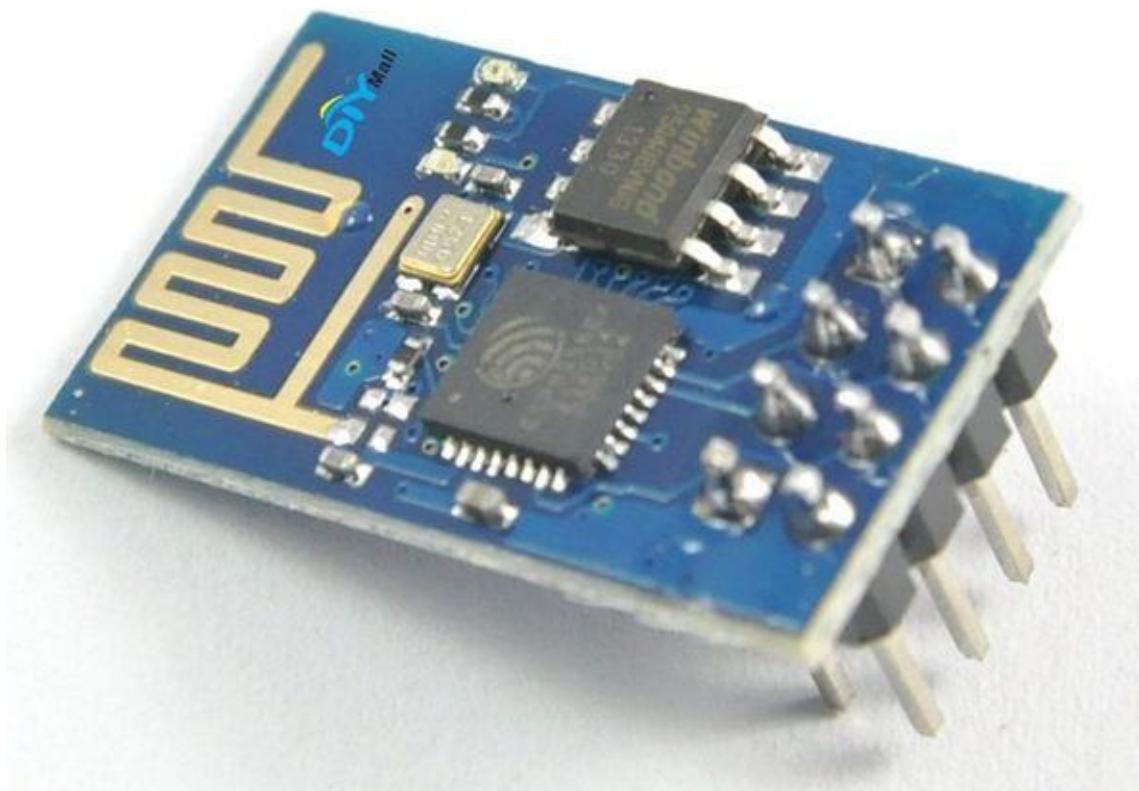
Basically you can use any Wi-Fi module to be connected to Raspberry Pi. The following is a list of Wi-Fi module:

- XBee WiFi Module - PCB Antenna, <https://www.sparkfun.com/products/12568>
- Adafruit CC3000 WiFi Breakout, <http://www.adafruit.com/product/1469>





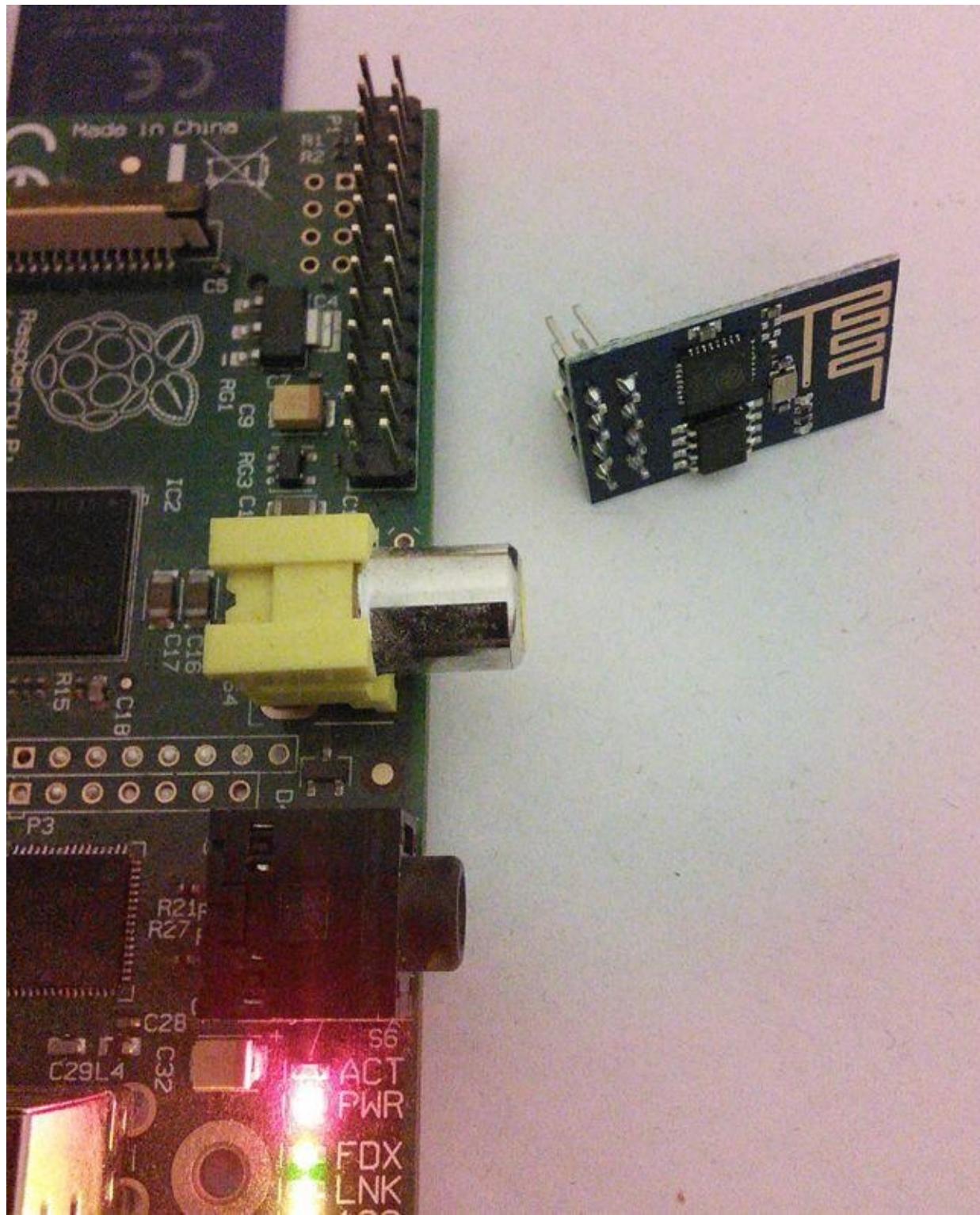
For testing, I use **ESP8266EX Serial WIFI Wireless**. It is a cheap Wi-Fi module about US\$ 3.50. You can find it on Amazon.com, <http://www.amazon.com/Diymall-Esp8266-Serial-Wireless-Transceiver/dp/B00O34AGSU/>, ebay or Seeedstudio <http://seeedstudio.com/depot/WiFi-Serial-Transceiver-Module-w-ESP8266-p-1994.html>. I got this module from ebay.



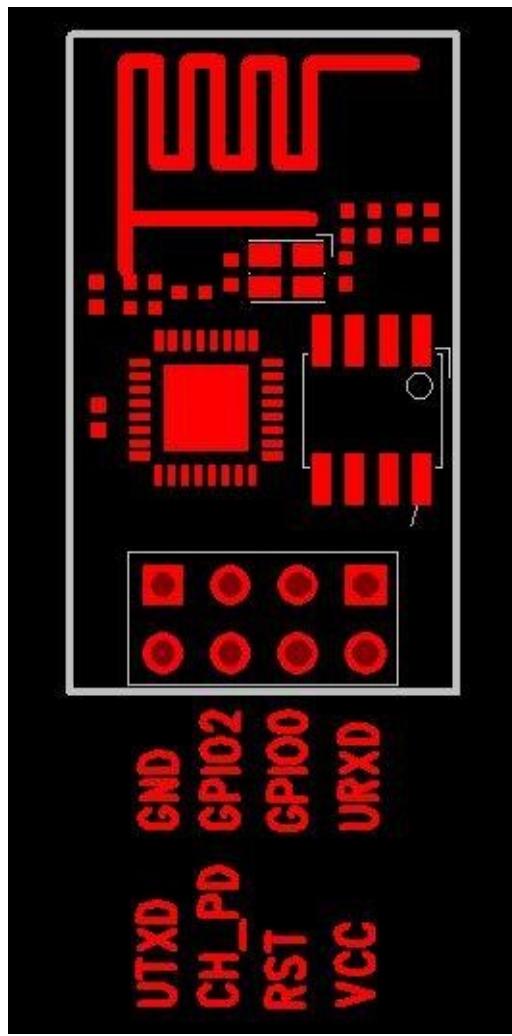
Further information about , you can read datasheet document on  
<http://www.electrodragon.com/w/Wi07c>  
[https://nurdspace.nl/images/e/e0/ESP8266\\_Specifications\\_English.pdf](https://nurdspace.nl/images/e/e0/ESP8266_Specifications_English.pdf) .  
and

### 2.3.2 Connecting Wi-Fi Module to Raspberry Pi

ESP8266 Serial WIFI Wireless is a wireless SoC product from China. We will connect ESP8266 module to Raspberry Pi.



Based on datasheet document, ESP8266 pins can be seen in Figure below.



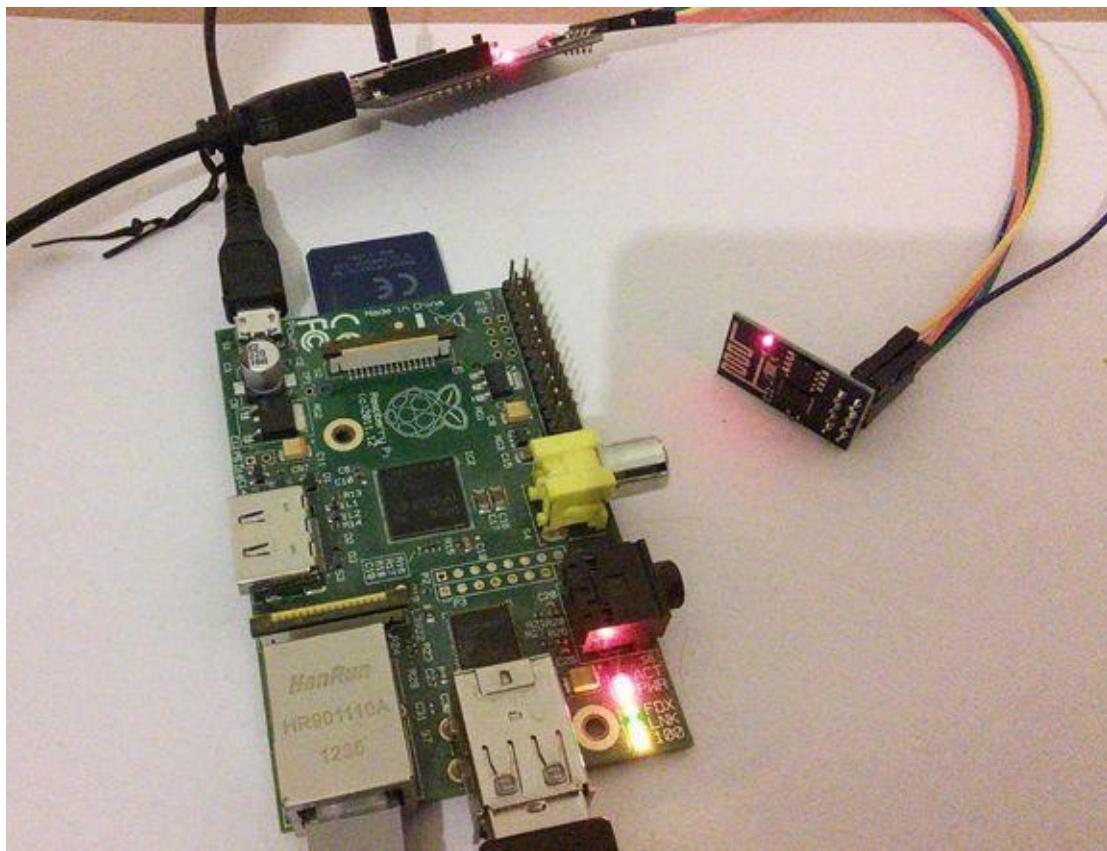
You can connect ESP8266 module to Raspberry Pi GPIO directly or via USB. If you connect it to Raspberry Pi GPIO, following is wiring:

- ESP8266 VCC to Pi VCC +3.3V
- ESP8266 UTXD to Pi UART RX
- ESP8266 URXD to Pi TX
- ESP8266 CH\_PD to Pi VCC +3.3V
- ESP8266 GND to Pi GND

If you want to connect ESP8266 module to Raspberry Pi via USB, you need TTL USB module. The following is wiring

- ESP8266 VCC to TTL USB VCC +3.3V
- ESP8266 UTXD to TTL USB UART RX
- ESP8266 URXD to TTL USB TX
- ESP8266 CH\_PD to TTL USB VCC +3.3V
- ESP8266 GND to TTL USB GND

For illustration, I connect ESP8266 module to TTL USB. Then, TTL USB is be connected to Raspberry Pi USB.



On Raspberry Pi Terminal, you can see TTL USB using this command.

```
$ ls /dev/ttyUSB*
```

```
pi@raspberrypi-AKUR ~ $ ls /dev/ttysUB*
ls: cannot access /dev/ttysUB*: No such file or directory
pi@raspberrypi-AKUR ~ $ ls /dev/ttysUB*
/dev/ttysUB0
pi@raspberrypi-AKUR ~ $
```

### 2.3.3 Demo 1: Connecting to Wi-Fi Hotspot

Firstly, we must understand AT command for ESP8266 module. You can read it on

[https://nurdspace.nl/ESP8266#AT\\_Commands](https://nurdspace.nl/ESP8266#AT_Commands) . In this demo, we try to connect a hotspot via ESP8266 module which uses Python serial library. I use this library, <https://github.com/guyz/pyesp8266> , and modify for our case.

To connect a Wi-Fi hotspot, we can send AT commands as follows:

- send AT for handshaking
- define device mode —> AT+CWMODE=1
- scan WiFi hotspot —> AT+CWLAP
- connect a hotspot with specific SSID and its key —> AT+CWJAP="ssid","key"
- if connected, we can check IP address —> AT+CIFSR

Note: change SSID and key values.

To access UART on Raspberry Pi, we can use pyserial library, <http://pyserial.sourceforge.net/> . To install pyserial on Raspberry Pi, you can follow installation instruction on <http://pyserial.sourceforge.net/pyserial.html#installation> .

Now you can create a file, called **esp8266demo.py** , and write this script.

```
import logging
import sys, serial
from time import *

def enum(**enums):
    return type('Enum', (), enums)

Status = enum(OK=['OK'], Fail=['Fail'],
             ERROR=['ERROR'], BUSY=['busy'],
             LINK=['Link'],
             ready=['ready'], no_change=['no change'],
             SEND_OK=['SEND OK'])

logging.basicConfig(stream=sys.stdout, level=logging.DEBUG)

def esp8266_at_command(sCmd, waitTm=1, retry=5):
    lp = 0
    ret = ""

    logging.info("Sending AT command: %s" % sCmd)

    for i in range(retry):
        ser.flushInput()
        ser.write(sCmd + "\r\n")
        ret = ser.readline() # Eat echo of command.
        sleep(0.2)
        while(lp < waitTm or 'busy' in ret):
            while(ser.inWaiting()):
                ret = ser.readline().strip("\r\n")
                logging.debug(ret)
                lp = 0

            if( ret in Status.OK ): break
            if( ret == Status.ERR ): break
            sleep(1)
```

```

lp += 1

sleep(1)
if( ret == 'Linked' ): break
if( ret == '>' ): break
if( ret == 'SEND OK' ): break
if( ret in Status.OK ): break

logging.info("Command result: %s" % ret)
return ret

port = "/dev/ttyUSB0"
speed = 9600
ssid = "WLAN_DPK13"
pwd = "123456"

ser = serial.Serial(port, speed)
if ser.isOpen():
    ser.close()
ser.open()
ser.isOpen()

esp8266_at_command("AT")
esp8266_at_command("AT+CWMODE=1") # set device mode (1=client, 2=AP, 3=
esp8266_at_command("AT+CWLAP", 10) # scan for WiFi hotspots
esp8266_at_command("AT+CWJAP="" +ssid +pwd "", 5) # connect
addr = esp8266_at_command("AT+CIFSR", 5) # check IP address
print("IP Address = " + addr)

ser.close()

```

Save this file. You can run it.

```
$ python esp8266demo.py
```

A sample output can be seen in Figure below.

```
pi@raspberrypi-AKUR: ~/pe/wee
pi@raspberrypi-AKUR ~/pe/wee $ python esp8266demo.py
INFO:root:Sending AT command: AT
DEBUG:root:
DEBUG:root:OK
INFO:root:Command result: OK
INFO:root:Sending AT command: AT+CWMODE=1
DEBUG:root:no change
INFO:root:Command result: no change
INFO:root:Sending AT command: AT+CWLAP
DEBUG:root:+CWLAP:(3,"FRITZ!Box 7362 SL",-91,"34:81:c4:38:a6:f4",1)
DEBUG:root:+CWLAP:(4,"ALICE-WLAN33",-91,"00:1c:28:79:0b:8f",1)
DEBUG:root:+CWLAP:(4,"WLAN_DPK13",-66,"84:9c:a6:43:9a:f4",9)
DEBUG:root:+CWLAP:(2,"Kassem",-86,"8a:25:2c:2e:0f:de",3)
DEBUG:root:+CWLAP:(4,"Kommst hier ned rein",-89,"00:22:3f:07:35:75",3)
DEBUG:root:+CWLAP:(4,"EasyBox-6E7026",-57,"1c:c6:3c:6e:70:68",4)
DEBUG:root:+CWLAP:(4,"EasyBox-A9CA55",-86,"74:31:70:a9:ca:fd",5)
DEBUG:root:+CWLAP:(3,"WLAN-262914",-91,"c4:6e:1f:40:75:f4",7)
DEBUG:root:+CWLAP:(4,"EasyBox-7B4912",-88,"50:7e:5d:7b:49:90",7)
DEBUG:root:+CWLAP:(4,"WLAN-399F48",-91,"00:1d:19:39:9f:fb",8)
DEBUG:root:+CWLAP:(3,"WLAN-E6E460",-83,"5c:7d:5e:fb:e6:e4",9)
DEBUG:root:+CWLAP:(3,"G-Punkt",-82,"00:1a:4f:d6:8c:39",11)
DEBUG:root:+CWLAP:(4,"WLAN-A47A50",-94,"00:26:4d:a4:7a:2a",11)
DEBUG:root:
DEBUG:root:OK
INFO:root:Command result: OK
INFO:root:Sending AT command: AT+CWJAP="WLAN_DPK13","indonesia99"
DEBUG:root:
DEBUG:root:OK
INFO:root:Command result: OK
INFO:root:Sending AT command: AT+CIFSR
DEBUG:root:192.168.1.32
DEBUG:root:
DEBUG:root:OK
INFO:root:Command result: OK
IP Address = OK
pi@raspberrypi-AKUR ~/pe/wee $
```

### 2.3.4 Demo 2: Creating Access Points (AP)

The second demo is to build a simple app to create an access point (AP). The following is list of AT command to create AP:

- send AT for handshaking
- define device mode as AP —> AT+CWMODE=2
- create SSID and its key —> AT+CWSAP="ssid","key",2,3

To implement it, you can create a file, called **esp8266ap.py**, and write this script.

```
import logging
import sys, serial
from time import *

def enum(**enums):
    return type('Enum', (), enums)

Status      = enum(ERR=['ERROR', 'Fail'],
['OK', 'ready', 'no change', 'SEND OK'], BUSY='busy', LINK='Link') OK=
logging.basicConfig(stream=sys.stdout, level=logging.DEBUG)

def esp8266_at_command(sCmd, waitTm=1, retry=5):
    lp = 0
    ret = ""

    logging.info("Sending AT command: %s" % sCmd)

    for i in range(retry):
        ser.flushInput()
        ser.write(sCmd + "\r\n")
        ret = ser.readline() # Eat echo of command.
        sleep(0.2)
        while(lp < waitTm or 'busy' in ret):
            while(ser.inWaiting()):
                ret = ser.readline().strip("\r\n")
                logging.debug(ret)
            lp = 0

            if( ret in Status.OK ): break
            if( ret == Status.ERR ): break
            sleep(1)
            lp += 1

        sleep(1)
        if( ret == 'Linked' ): break
        if( ret == '>' ): break
        if( ret == 'SEND OK' ): break
        if( ret in Status.OK ): break

    logging.info("Command result: %s" % ret)
    return ret

port = "/dev/ttyUSB0"
speed = 9600
ssid = "pissid"
pwd = "12345"

ser = serial.Serial(port, speed)
if ser.isOpen():
    ser.close()
```

```

ser.open()
ser.isOpen()

esp8266_at_command("AT")
esp8266_at_command("AT+CWMODE=2") # set device mode (1=client, 2=AP, 3=
esp8266_at_command("AT+CWSAP=""ssid"", ""+pwd+"", 2, 3", 5) # create AP
sleep(0.5)
print("wifi hotspot. ssid: " + ssid)
while(1):
    sleep(0.3)

ser.close()

```

Save this code and try to run it.

```
$ python esp8266ap.py
```

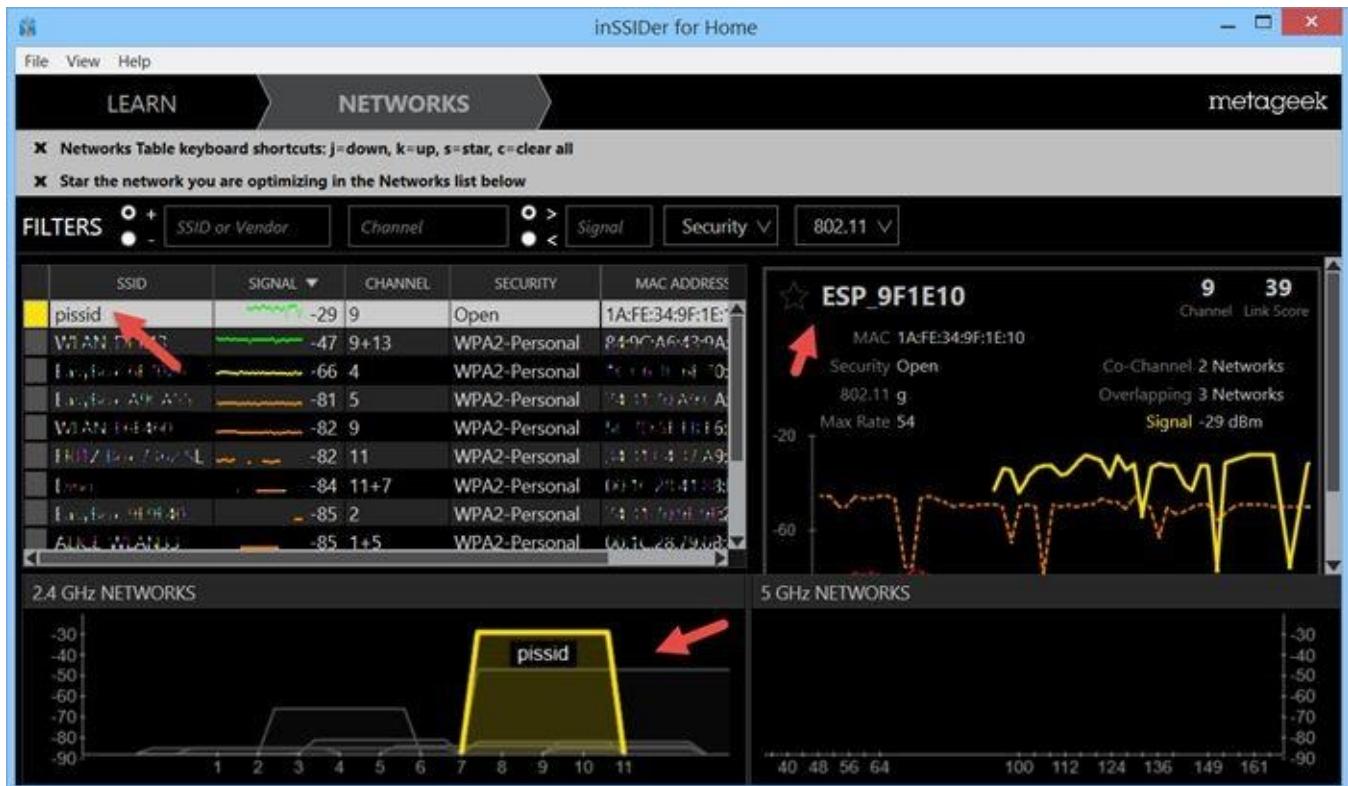
You can see a sample output of program in Figure below.

```

pi@raspberrypi-AKUR: ~/pe/wee
pi@raspberrypi-AKUR ~/pe/wee $ ls
esp8266ap.py  esp8266demo.py  esp8266server.py
pi@raspberrypi-AKUR ~/pe/wee $ python esp8266ap.py
INFO:root:Sending AT command: AT
DEBUG:root:
DEBUG:root:OK
INFO:root:Command result: OK
INFO:root:Sending AT command: AT+CWMODE=2
DEBUG:root:
DEBUG:root:OK
INFO:root:Command result: OK
INFO:root:Sending AT command: AT+CWSAP="ssid", "12345", 2, 3
DEBUG:root:
DEBUG:root:OK
INFO:root:Command result: OK
wifi hotspot. ssid: ssid

```

To verify created AP, You can see WiFi analyzer. I use inSSIDer tool, <http://www.inssider.com/>. You can download and install it. I can find my created AP, shown in Figure below.



### 2.3.5 Demo 3: Creating A Simple Web Server

The last demo is to create a simple web server with port 9888. We will send a response if a client is connecting to Raspberry Pi. After sent a response, Wi-Fi module will close a connection. The algorithm can be defined as follows:

- send AT for handshaking
- define device mode as client —> AT+CWMODE=1
- scan WiFi hotspot —> AT+CWLAP
- connect a hotspot with specific SSID and its key —> AT+CWJAP="ssid","key"
- if connected, we can check IP address —> AT+CIFSR
- set a multiple connection mode —> AT+CIPMUX=1
- set module as server with specific port —> AT+CIPSERVER=1,port

If a client is be connected, we send a response by calling `process_request()` and `send_response()` functions.

For implementation, create a file, called `esp8266server.py`, and write this script.

```
import serial
from time import *
import datetime

# modified code from: https://github.com/guyz/pyesp8266
```

```

def enum(**enums):
    return type('Enum', (), enums)

Status          = enum(ERROR=[ 'ERROR',           'Fail' ],          OK=
[ 'OK',  'ready',  'no change',  'SEND OK' ],  BUSY='busy',  LINK='Link')

def esp8266_at_command( sCmd, waitTm=1, retry=5, delay=1):
    lp = 0
    ret = ""

    print("Sending command: %s" % sCmd)

    for i in range(retry):
        ser.flushInput()
        ser.write(sCmd + "\r\n")
        ret = ser.readline()      # Eat echo of command.
        sleep( 0.2 )
        while(lp < waitTm or 'busy' in ret):
            while( ser.inWaiting() ):
                ret = ser.readline().strip( "\r\n" )
                print(ret)
                lp = 0
            if( ret in Status.OK ): break
            #if( ret == 'ready' ): break
            if( ret in Status.ERR ): break
            sleep(delay)
            lp += 1

        sleep(delay)
        if( ret in Status.OK ): break

    print("Command result: %s" % ret)
    return ret


def send_response(response, cid='0'):
    print("send_response...")
    cmd = "AT+CIPSEND=" + cid + "," + str(len(response)) + "\r\n"
    print("command: " + cmd)
    ser.write(cmd)
    sleep(0.3)
    ser.write(response + "\r\n")
    sleep(0.3)

    send_res = False
    for i in range(100):
        while( ser.inWaiting() ):
            ret = ser.readline().strip("\r\n")
            print ret
            if( ret == Status.OK[3] ):
                print "send ok!"
                send_res = True
        if send_res: break
        sleep(0.1)

```

```

sleep(0.3)
ser.write("AT+CIPCLOSE=" + cid + "\r\n")
sleep(0.3)

def process_request(response):
    has_link = False
    cid = '0'
    while(ser.inWaiting()):
        ret = ser.readline().strip("\r\n")
        print ret
        if (ret in Status.LINK):
            has_link = True
        ipd_str = '+IPD,' 
        if (ipd_str in ret):
            cid = ret[ret.find(ipd_str) + len(ipd_str)]
            print("cid: " + cid)

    if has_link:
        # process response
        send_response(response, cid)

port = "/dev/ttyUSB0"
speed = 9600
ssid = "WLAN_DPK13"
pwd = "123456"
p = 9888

ser = serial.Serial(port, speed)
if ser.isOpen():
    ser.close()
ser.open()
ser.isOpen()

esp8266_at_command("AT")
esp8266_at_command("AT+CWMODE=1") # set device mode (1=client, 2=AP, 3=
esp8266_at_command("AT+CWLAP", 30) # scan for WiFi hotspots
esp8266_at_command("AT+CWJAP=\"" + ssid + "\",\"" + pwd + "\", 5) # connect
addr = esp8266_at_command("AT+CIFSR", 5) # check IP address
print("IP Address = " + addr)

esp8266_at_command("AT+CIPMUX=1") # multiple connection mode
sleep(0.3)
esp8266_at_command("AT+CIPSERVER=1," + str(p))
sleep(0.3)

# process requests
while(1):
    process_request("Response from ESP8266 ! (" + str(datetime.datetime.now())
    sleep(0.3)

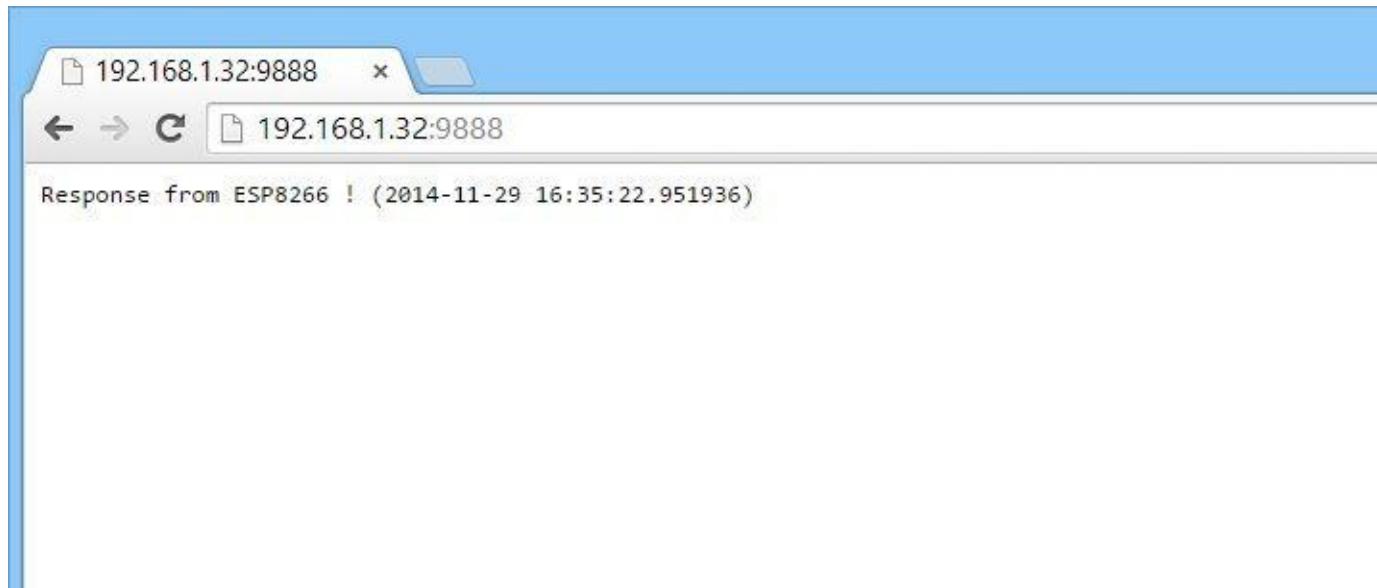
ser.close()

```

Save this file and try to run.

```
$ python esp8266server.py
```

For illustration, I use a browser as client app. Navigate to IP address of Raspberry Pi with port 9888. If success, you will get a response from Raspberry Pi. A sample output can be seen in Figure below.



A sample output for program is shown in Figure below.



pi@raspberrypi-AKUR: ~/pe/wee

```
+CWLAP: (4, "EasyBox-A9CA55", -83, "74:31:70:a9:ca:fd", 5)
+CWLAP: (4, "WLAN_DPK13", -51, "84:9c:a6:43:9a:f4", 9)
+CWLAP: (3, "G-Punkt", -88, "00:1a:4f:d6:8c:39", 11)
+CWLAP: (4, "FRITZ!Box 7312", -91, "9c:c7:a6:11:52:91", 11)

OK
Command result: OK
Sending command: AT+CWJAP="WLAN_DPK13", "indonesia99"

OK
Command result: OK
Sending command: AT+CIFSR
192.168.1.32

OK
Command result: OK
IP Address = OK
Sending command: AT+CIPMUX=1

OK
Command result: OK
Sending command: AT+CIPSERVER=1, 9888
no change
Command result: no change
Link
send_response...
command: AT+CIPSEND=0, 52

AT+CIPSEND=0, 52
>
+IPD,0,362:GET / HTTP/1.1
Host: 192.168.1.32:9888
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0
.8
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/39.0.2171.71 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8,id;q=0.6,ms;q=0.4

OK
```

### **3. IR Communication**

This chapter explains how to work with IR communication on Raspberry Pi.

### **3.1 Getting Started**

Infrared (IR) can be used for line-of-sight communications over low to moderate range. IR is nice because of the lack of interference (except for sun and compact fluroscent lights) and freedom from FCC regulation. In this chapter, we learn how to use IR for communication.

## 3.2 Building IR Remote

In this section, we deploy and setup an IR remote which is integrated to Raspberry Pi. The following is the required hardware:

- IR sensor
- IR remote
- Cables

We will explore thesees hardware to implement IR Remote application.

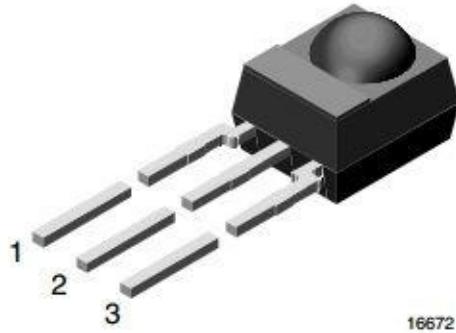
### 3.2.1 Hardware Implementation

For testing, I used TSOP4838 IR Receiver. I bought it from eBay. You can get this stuff from the following online store:

- Farnell, <http://uk.farnell.com/vishay/tsop4838/ir-receiver-38khz/dp/4913190>
- adafruit, <http://www.adafruit.com/products/157>
- sparkfun, <https://www.sparkfun.com/products/10266>
- eBay, <http://www.ebay.com>



The following is its datasheet (<http://www.farnell.com/datasheets/1693301.pdf> ).



## MECHANICAL DATA

Pinning for TSOP44.., TSOP48..:

1 = OUT, 2 = GND, 3 = V<sub>S</sub>

Pinning for TSOP22.., TSOP24..:

1 = OUT, 2 = V<sub>S</sub>, 3 = GND

You also need IR remote which is used to IR transmitter. You can get it from this store:

- adafruit, <http://www.adafruit.com/product/389>
- eBay, <http://www.ebay.com>

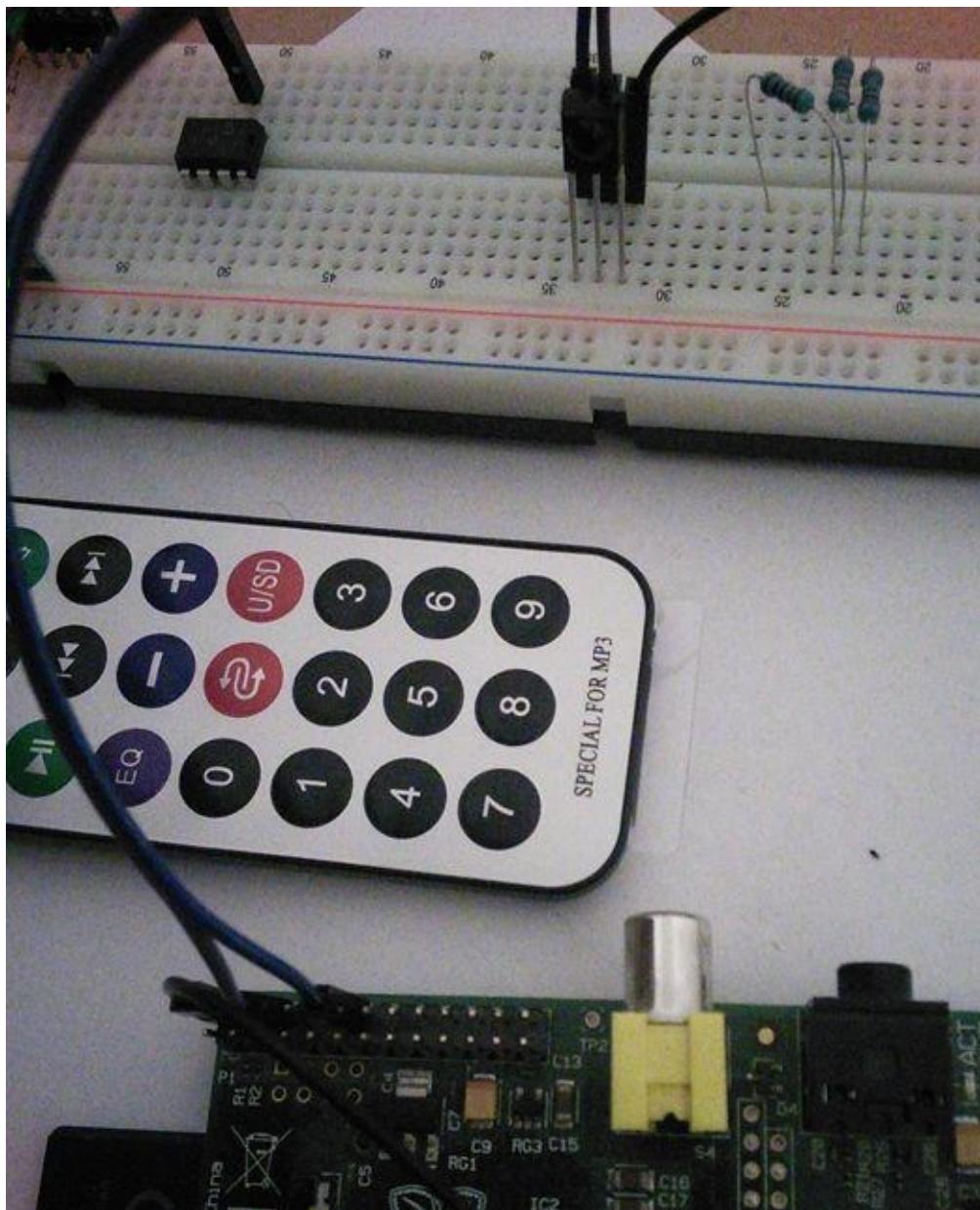


There are many IR kits which provide IR receiver and remote devices. You can find them on eBay or another online store.



Now you can connect IR receiver pin 1 to Raspberry Pi GPIO 18 (PWM). IR receiver pin 2 is connected to Pi GPIO GND and IR receiver pin 3 is connected to Pi GPIO VCC 5V.

The following is my wiring IR receiver to Raspberry Pi.

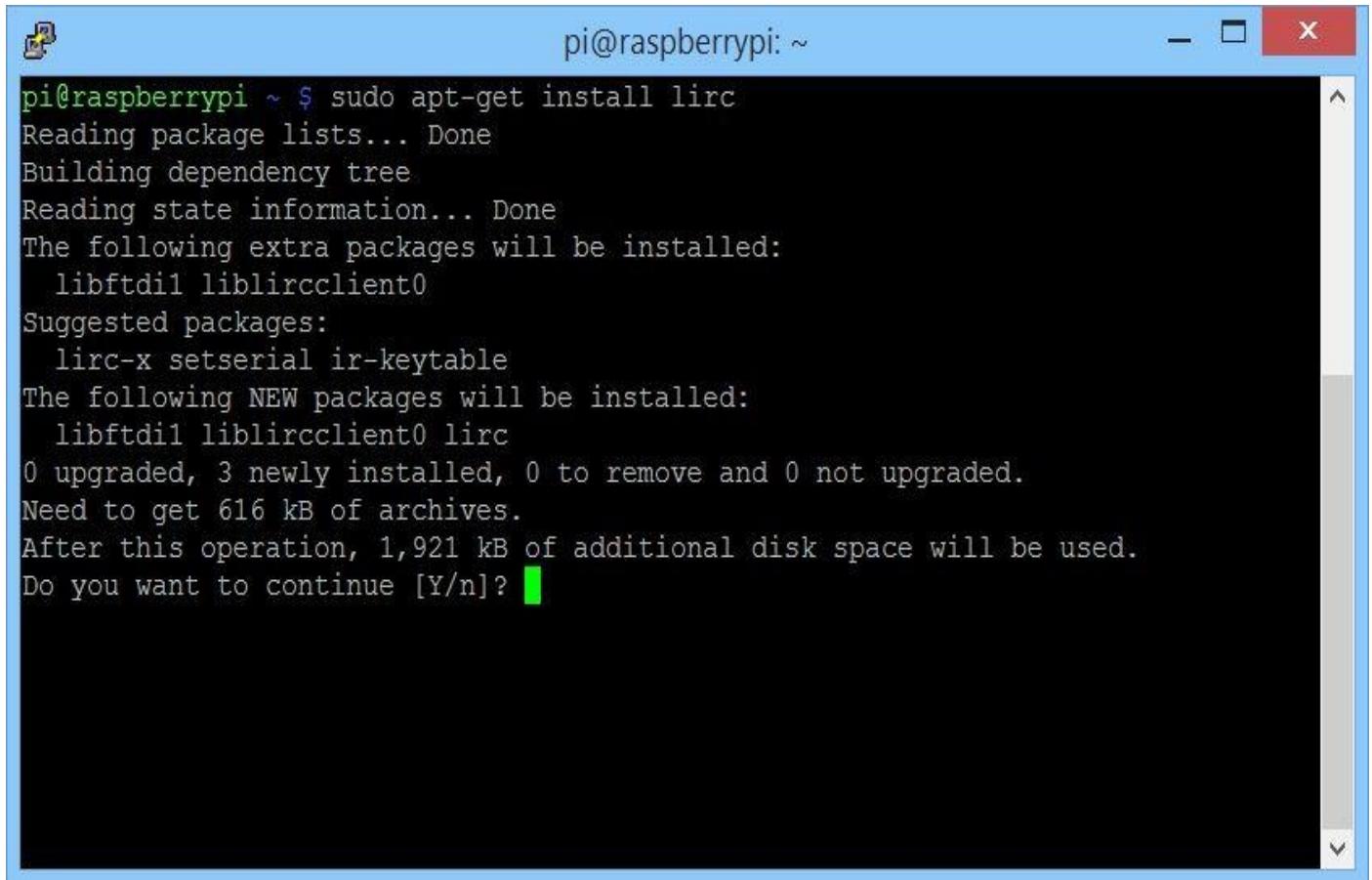


### 3.2.2 Configuring and Testing

I found a good article about how to configure IR on Raspberry Pi, <http://www.jamesrobertson.eu/blog/2013/may/19/installing-lirc-on-raspbian.html>. We use lirc library to build IR Remote application.

Firstly, we install lirc library.

```
$ sudo apt-get update  
$ sudo apt-get upgrade  
$ sudo apt-get install lirc
```



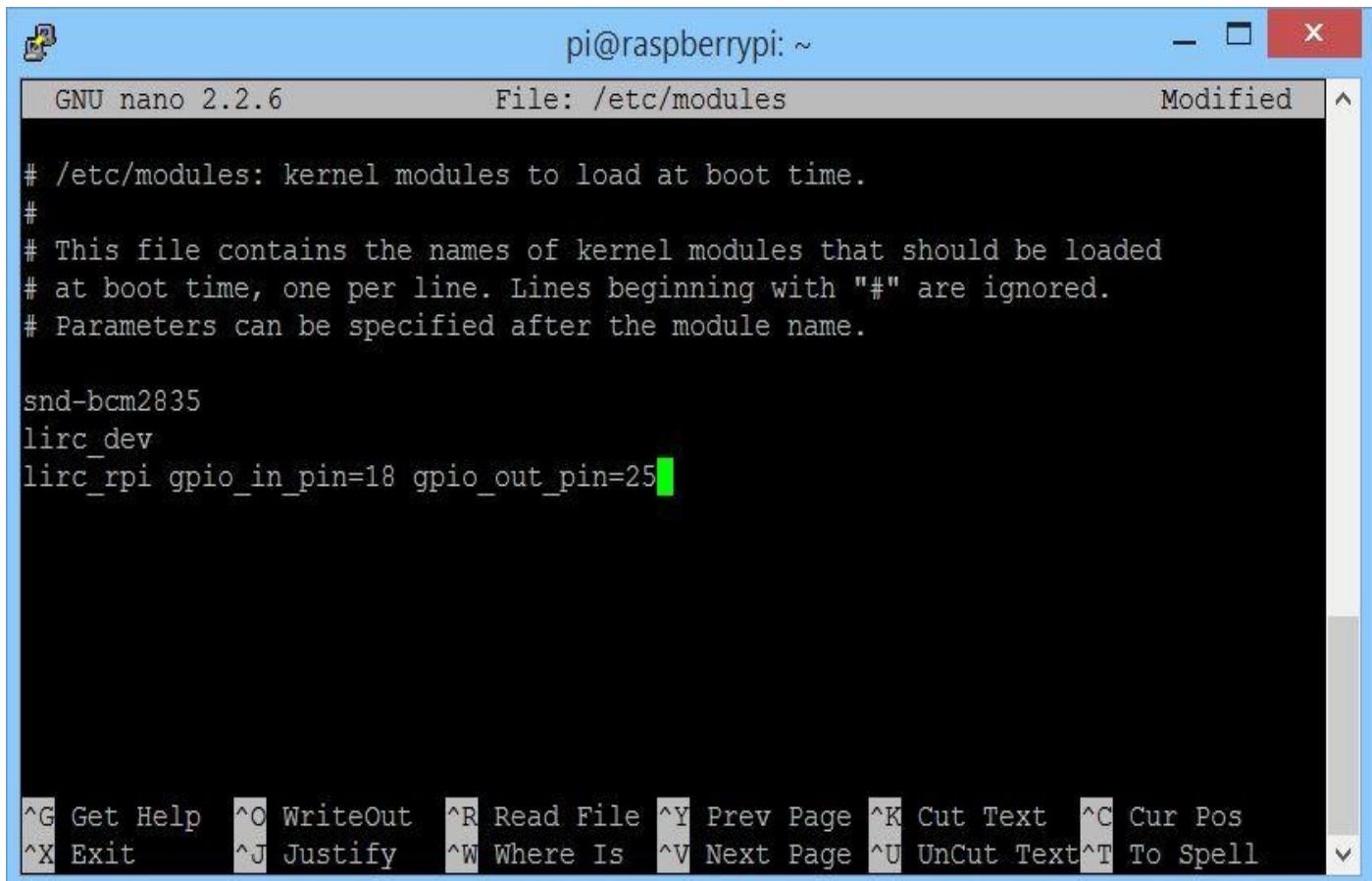
```
pi@raspberrypi ~ $ sudo apt-get install lirc
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  libftdi1 liblircclient0
Suggested packages:
  lirc-x setserial ir-keytable
The following NEW packages will be installed:
  libftdi1 liblircclient0 lirc
0 upgraded, 3 newly installed, 0 to remove and 0 not upgraded.
Need to get 616 kB of archives.
After this operation, 1,921 kB of additional disk space will be used.
Do you want to continue [Y/n]? 
```

After installed, we modify **/etc/modules** file to load lirc into OS.

```
$ sudo nano /etc/modules
```

In this case, I use nano for text editor. You can use your own text editor. Then, add these lines into **/etc/modules** file.

```
lirc_dev
lirc_rpi gpio_in_pin=18 gpio_out_pin=25
```



```
pi@raspberrypi: ~
GNU nano 2.2.6          File: /etc/modules          Modified ^

# /etc/modules: kernel modules to load at boot time.
#
# This file contains the names of kernel modules that should be loaded
# at boot time, one per line. Lines beginning with "#" are ignored.
# Parameters can be specified after the module name.

snd-bcm2835
lirc_dev
lirc_rpi gpio_in_pin=18 gpio_out_pin=25^

^G Get Help  ^O WriteOut  ^R Read File  ^Y Prev Page  ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is   ^V Next Page  ^U UnCut Text ^T To Spell
```

Save this file and close nano.

Now you edi **/etc/lirc/hardware.conf** file.

```
$ sudo nano /etc/lirc/hardware.conf
```

Modify **/etc/lirc/hardware.conf** as follows.

```
#####
# /etc/lirc/hardware.conf
#
# Arguments which will be used when launching lircd
LIRCD_ARGS=""

# Don't start lircmd even if there seems to be a good config file
# START_LIRCMD=false

# Don't start irexec, even if a good config file seems to exist.
# START_IRExec=false

# Try to load appropriate kernel modules
LOAD_MODULES=true

# Run "lircd --driver=help" for a list of supported drivers.
DRIVER="default"
# usually /dev/lirc0 is the correct setting for systems using udev
DEVICE="/dev/lirc0"
MODULES="lirc_rpi"
```

```
# Default configuration files for your hardware if any
LIRCD_CONF=""
LIRCMD_CONF=""
#####
#####
```

The screenshot shows a terminal window titled "pi@raspberrypi: ~" running the "GNU nano 2.2.6" editor. The file being edited is "/etc/lirc/hardware.conf". The content of the file is as follows:

```
#START_IRExec=false
#Try to load appropriate kernel modules
LOAD_MODULES=true

# Run "lircd --driver=help" for a list of supported drivers.
DRIVER="default"
# usually /dev/lirc0 is the correct setting for systems using udev
DEVICE="/dev/lirc0"
MODULES="lirc_rpi"

# Default configuration files for your hardware if any
LIRCD_CONF=""
LIRCMD_CONF=""
```

At the bottom of the terminal window, there is a menu of keyboard shortcuts:

```
^G Get Help  ^O WriteOut  ^R Read File  ^Y Prev Page  ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is   ^V Next Page  ^U UnCut Text ^T To Spell
```

Save this modified file. Now you can restart your board.

```
$ sudo reboot
```

After rebooted, you can verify if lirc already loaded or not using dmesg.

```
$ dmesg
```

If success, you can see “lirc\_rpi: driver registered” on dmesg messages.

```
pi@raspberrypi: ~
[ 2.662202] hub 1-1:1.0: 3 ports detected
[ 2.947020] usb 1-1.1: new high-speed USB device number 3 using dwc_otg
[ 3.077258] usb 1-1.1: New USB device found, idVendor=0424, idProduct=ec00
[ 3.085855] usb 1-1.1: New USB device strings: Mfr=0, Product=0, SerialNumber
=0
[ 3.113116] smsc95xx v1.0.4
[ 3.182617] smsc95xx 1-1.1:1.0 eth0: register 'smsc95xx' at usb-bcm2708_usb-1
.1, smsc95xx USB 2.0 Ethernet, b8:27:eb:1b:c9:48
[ 3.960484] udevd[158]: starting version 175
[ 9.688503] EXT4-fs (mmcblk0p2): re-mounted. Opts: (null)
[ 10.224857] EXT4-fs (mmcblk0p2): re-mounted. Opts: (null)
[ 11.058569] lirc dev: IR Remote Control driver registered, major 247
[ 11.192321] lirc_rpi: module is from the staging directory, the quality is un
known, you have been warned.
[ 12.166736] lirc_rpi: auto-detected active low receiver on GPIO pin 18
[ 12.179284] lirc_rpi lirc_rpi.0: lirc_dev: driver lirc_rpi registered at mino
r = 0
[ 12.191419] lirc_rpi: driver registered!
[ 21.077255] smsc95xx 1-1.1:1.0 eth0: hardware isn't capable of remote wakeup
[ 22.673169] smsc95xx 1-1.1:1.0 eth0: link up, 100Mbps, full-duplex, lpa 0x45E
1
[ 26.552298] Adding 102396k swap on /var/swap. Priority:-1 extents:2 across:2
134012k SSFS
pi@raspberrypi ~ $
```

For testing, you can stop lirc service. Then, run mode2 for /dev/lirc0.

```
$ sudo /etc/init.d/lirc stop
$ mode2 -d /dev/lirc0
```

Use IR remote and shot it on IR receiver of Raspberry Pi. Press any button on IR remote so you can see pulse values on Terminal.

```
pulse 616
space 507
pulse 641
space 483
pulse 576
space 552
pulse 645
space 456
pulse 641
space 1592
pulse 642
space 1617
pulse 573
space 550
pulse 599
space 526
pulse 617
space 1596
pulse 637
space 1618
pulse 608
space 1629
pulse 666
```

### 3.2.3 Recording

If you have new IR remote with unique keys, you can register and record them in Lirc. Follow these steps to record new IR remote device.

Firstly, stop Lirc service.

```
$ sudo /etc/init.d/lirc stop
```

To record IR remote keys, you can use irrecord command. We save this configuration in current directory, `~/lircd.conf`.

```
$ irrecord -d /dev/lirc0 ~/lircd.conf
```

Just follow the instruction how to record. If finished, you copy `~/lircd.conf` to `/etc/lirc/lircd.conf`.

```
$ sudo cp ~/lircd.conf /etc/lirc/lircd.conf
```

After that, you can start Lirc service.

```
$ sudo /etc/init.d/lirc start
```

To test your recording, you can use irw.

```
$ irw
```

Press any key on IR remote device.

## **4. Bluetooth Low Energy (BLE) and iBeacon**

This chapter explains how to work with Bluetooth Low Energy on Raspberry Pi.

## 4.1 Bluetooth Low Energy (BLE)

Bluetooth low energy or Bluetooth LE is a wireless personal area network technology designed and marketed by the Bluetooth Special Interest Group aimed at novel applications in the healthcare, fitness, beacons, security, and home entertainment industries.

In this chapter, I use BLE USB from CSR.



You can get BLE USB from the following store.

- adafruit, <http://www.adafruit.com/product/1327>
- dx, <http://www.dx.com/p/ultra-mini-bluetooth-csr-4-0-usb-dongle-adapter-black-143276>
- eBay, <http://www.ebay.com>
- Amazon, <http://www.amazon.com>

You also can find BLE 4 USB from your local electronics store.

## 4.2 Installing Bluez5

In this section, we try to install Bluez5 library from source code on Raspberry Pi. Firstly, we remove previous bluez library.

```
$ sudo apt-get --purge remove bluez  
$ sudo apt-get update
```

Install required libraries before install bluez.

```
$ sudo apt-get install libusb-dev libdbus-1-dev libglib2.0-dev libudev-dev libical-dev libreadline-dev
```

Download bluez5 source code, for instance Bluez 5.21.

```
$ wget https://www.kernel.org/pub/linux/bluetooth/bluez-5.21.tar.xz  
$ tar -xvf bluez-5.21.tar.gz
```

Now you can compile and install it.

```
$ cd bluez-5.21/  
$ sudo ./configure --disable-systemd  
$ sudo make  
$ sudo make install
```

After finished, try to reboot your board.

```
$ sudo reboot
```

After rebooted, you can connect BLE USB to Raspberry Pi.



You can verify this bluetooth device using grep.

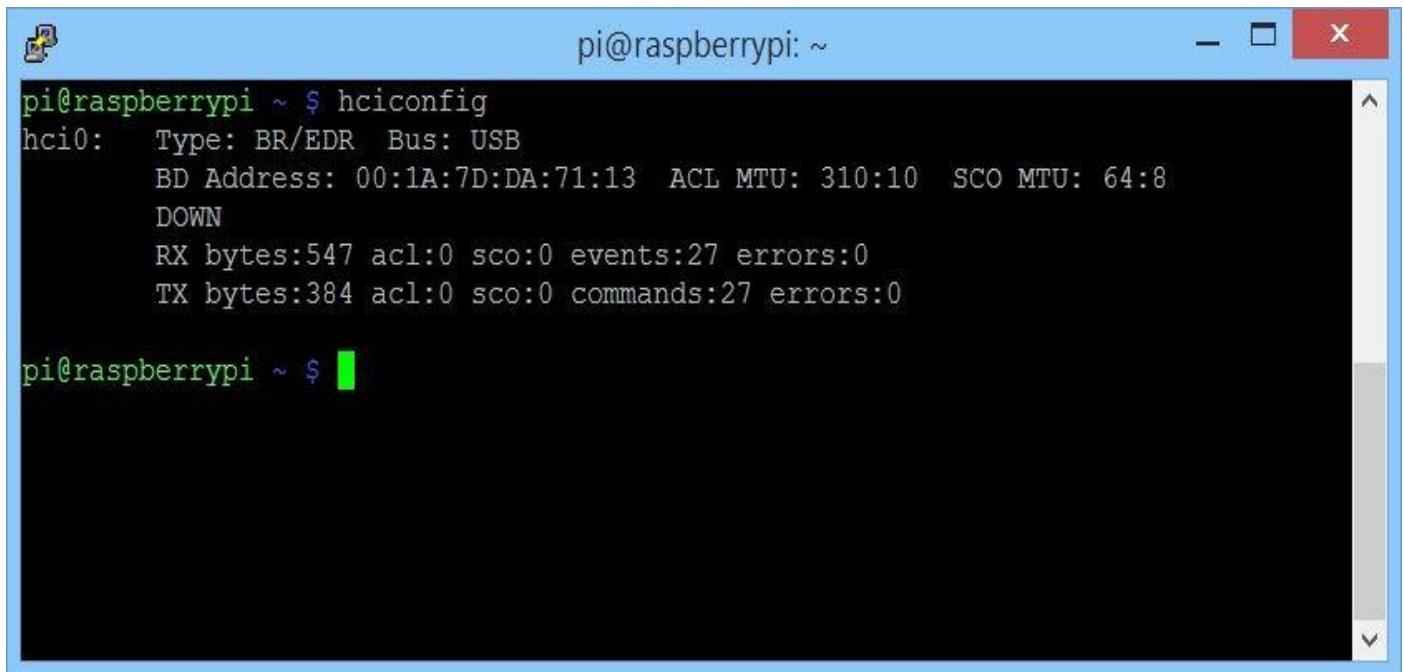
```
$ lsusb | grep -i bluetooth
```

If success, you can see your Bluetooth USB.

```
pi@raspberrypi: ~/book
pi@raspberrypi ~/book $ lsusb | grep -i bluetooth
Bus 001 Device 007: ID 0a12:0001 Cambridge Silicon Radio, Ltd Bluetooth Dongle (HCI mode)
pi@raspberrypi ~/book $
```

Now you can check your bluetooth config using hciconfig.

```
$ hciconfig
```



```
pi@raspberrypi ~ $ hciconfig
hci0:  Type: BR/EDR  Bus: USB
          BD Address: 00:1A:7D:DA:71:13  ACL MTU: 310:10  SCO MTU: 64:8
          DOWN
          RX bytes:547 acl:0 sco:0 events:27 errors:0
          TX bytes:384 acl:0 sco:0 commands:27 errors:0

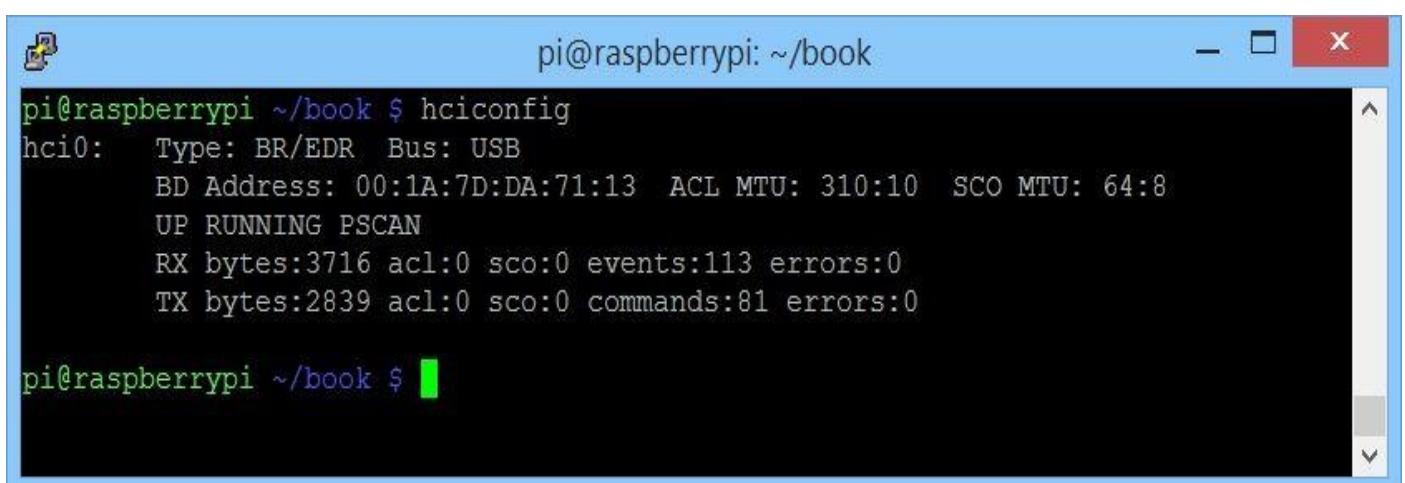
pi@raspberrypi ~ $
```

If your bluetooth device is down, you can turn on it using hciconfig up command.

```
$ sudo hciconfig hci0 up
```

After that, you can verify it again.

```
$ hciconfig
```



```
pi@raspberrypi ~/book $ hciconfig
hci0:  Type: BR/EDR  Bus: USB
          BD Address: 00:1A:7D:DA:71:13  ACL MTU: 310:10  SCO MTU: 64:8
          UP RUNNING PSCAN
          RX bytes:3716 acl:0 sco:0 events:113 errors:0
          TX bytes:2839 acl:0 sco:0 commands:81 errors:0

pi@raspberrypi ~/book $
```

You can also check using your device with BLE support. In my Nexus 7 2013, Raspberry Pi bluetooth is be detected too. It shows as BlueZ 5.18, shown in Figure below.



7:50

## Bluetooth



On



### Paired devices

ubuntu-0

### Available devices

yo2bushne fufu

BlueZ 5.18

Nexus 7 is visible to nearby devices while Bluetooth Settings is open.

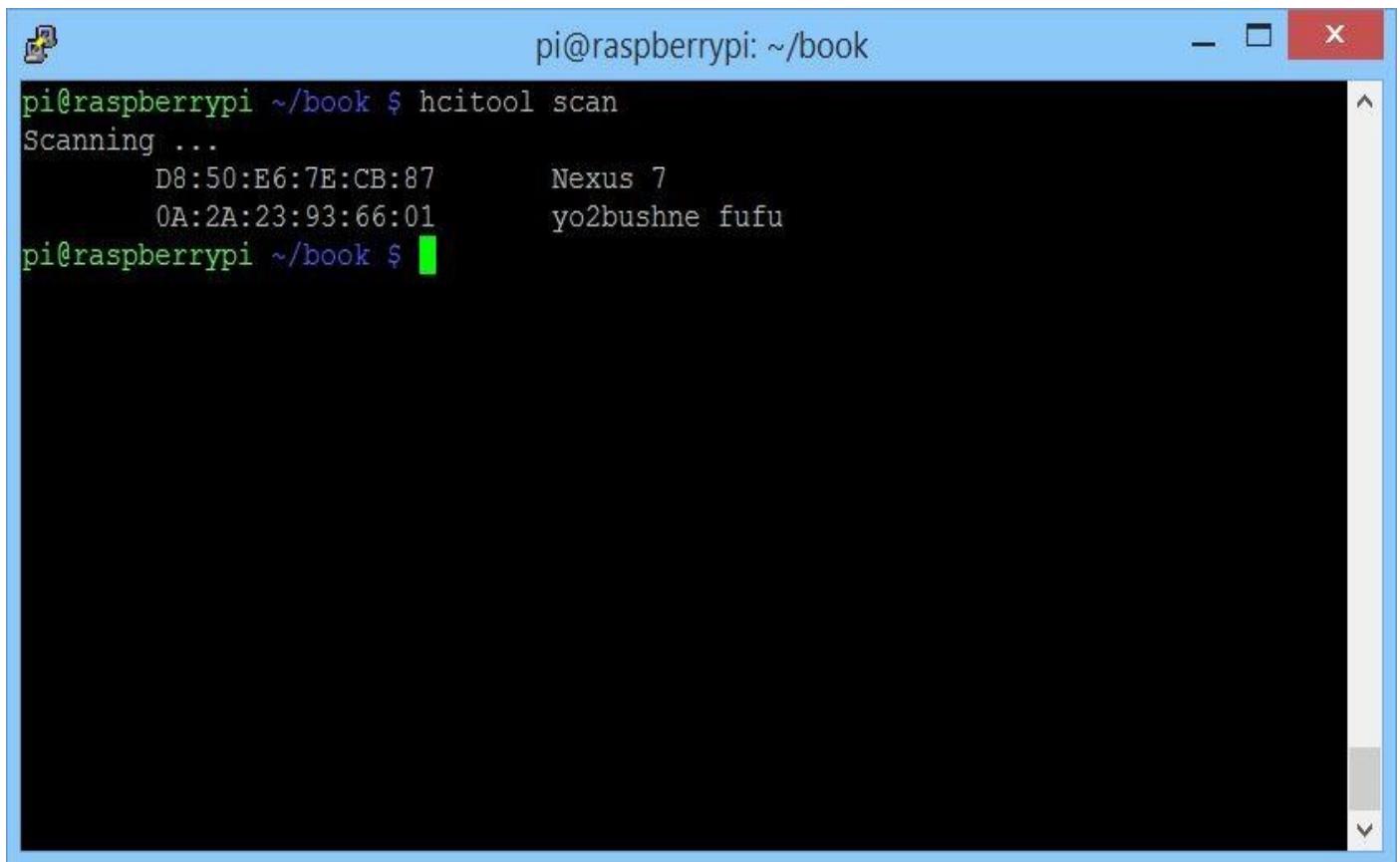
## 4.3 Pairing Bluetooth Devices

For the first demo, we try to connect our Intel Device to another device via BLE. In this case, I use my Nexus 7 2013, running Android 5, which will be connected to Raspberry Pi via Bluetooth 4 USB.

Firstly, we scan bluetooth devices using hcitool tool.

```
$ hcitool scan
```

If success, you can see bluetooth devices and their address.



A screenshot of a terminal window titled "pi@raspberrypi: ~/book". The window shows the command "hcitool scan" being run. The output of the command is displayed, showing two Bluetooth devices found during the scan. The first device is a Nexus 7 with the MAC address D8:50:E6:7E:CB:87 and the name "Nexus 7". The second device is a device named "yo2bushne fufu" with the MAC address 0A:2A:23:93:66:01. The terminal window has a blue header bar and a black body. The scroll bar on the right side of the window is visible.

```
pi@raspberrypi ~$ hcitool scan
Scanning ...
D8:50:E6:7E:CB:87      Nexus 7
0A:2A:23:93:66:01      yo2bushne fufu
pi@raspberrypi ~$
```

To connect to a bluetooth device, we can use **bluez-simple-agent** with parameter: our bluetooth id and bluetooth target address.

```
$ sudo bluez-simple-agent hci0 <target_address>
```

Change <target\_address> value for bluetooth target address.

```
pi@raspberrypi:~/book
pi@raspberrypi ~/book $ hcitool scan
Scanning ...
      D8:50:E6:7E:CB:87      Nexus 7
      0A:2A:23:93:66:01      yo2bushne fufu
pi@raspberrypi ~/book $ sudo bluez-simple-agent hci0 D8:50:E6:7E:CB:87
Creating device failed: org.bluez.Error.AuthenticationRejected: Authentication R
ejected
pi@raspberrypi ~/book $
```

If you can get error message, shown in Figure above. It happens because we cannot handle authentication dialog. We must enable it. I found an article which solved this issue. You can read it on <http://www.wolfteck.com/projects/raspi/iphone/> .

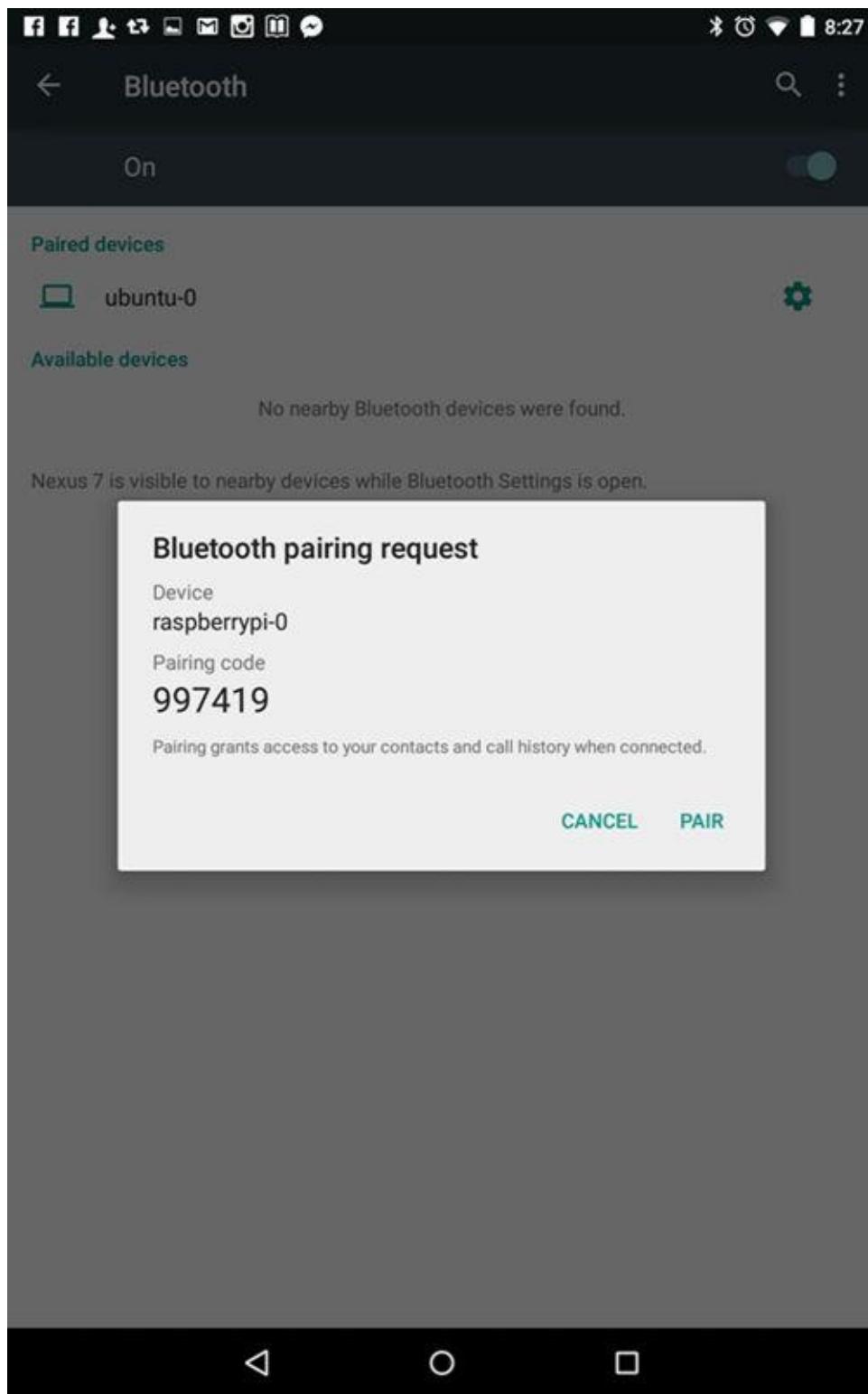
Type these commands to enable authentication dialog.

```
$ grep KeyboardDisplay /usr/bin/bluez-simple-agent
$ sudo perl -i -pe 's/KeyboardDisplay/DisplayYesNo/' /usr/bin/bluez-
simple-agent
$ grep DisplayYesNo /usr/bin/bluez-simple-agent
```

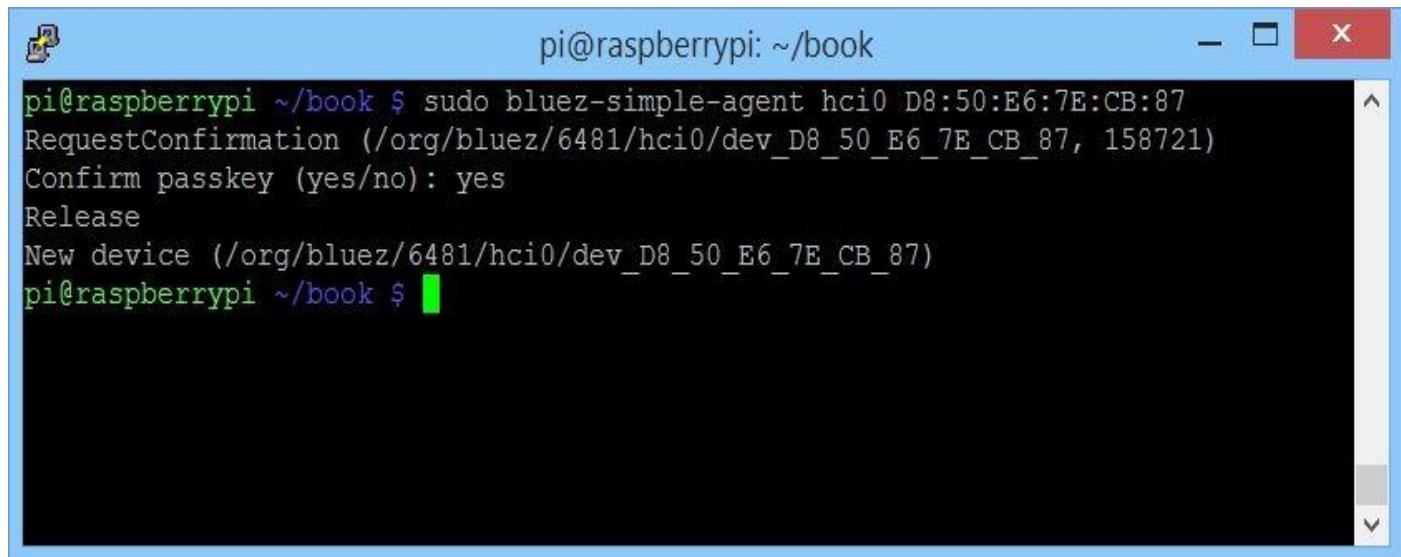
The screenshot shows a terminal window titled "pi@raspberrypi: ~/book". The terminal displays the following command-line session:

```
pi@raspberrypi ~/book $ hcitool scan
Scanning ...
D8:50:E6:7E:CB:87      Nexus 7
0A:2A:23:93:66:01      yo2bushne fufu
pi@raspberrypi ~/book $ sudo bluez-simple-agent hci0 D8:50:E6:7E:CB:87
Creating device failed: org.bluez.Error.AuthenticationRejected: Authentication R
ejected
pi@raspberrypi ~/book $ grep KeyboardDisplay /usr/bin/bluez-simple-agent
    capability = "KeyboardDisplay"
pi@raspberrypi ~/book $ sudo perl -i -pe 's/KeyboardDisplay/DisplayYesNo/' /usr/
bin/bluez-simple-agent
pi@raspberrypi ~/book $ grep KeyboardDisplay /usr/bin/bluez-simple-agent
pi@raspberrypi ~/book $ grep DisplayYesNo /usr/bin/bluez-simple-agent
    capability = "DisplayYesNo"
pi@raspberrypi ~/book $
```

Now you can try to pair your bluetooth. For instance, I connect my bluetooth to Nexus 7 2013. In Nexus, I got confirmation, shown in Figure below. Press PAIR.



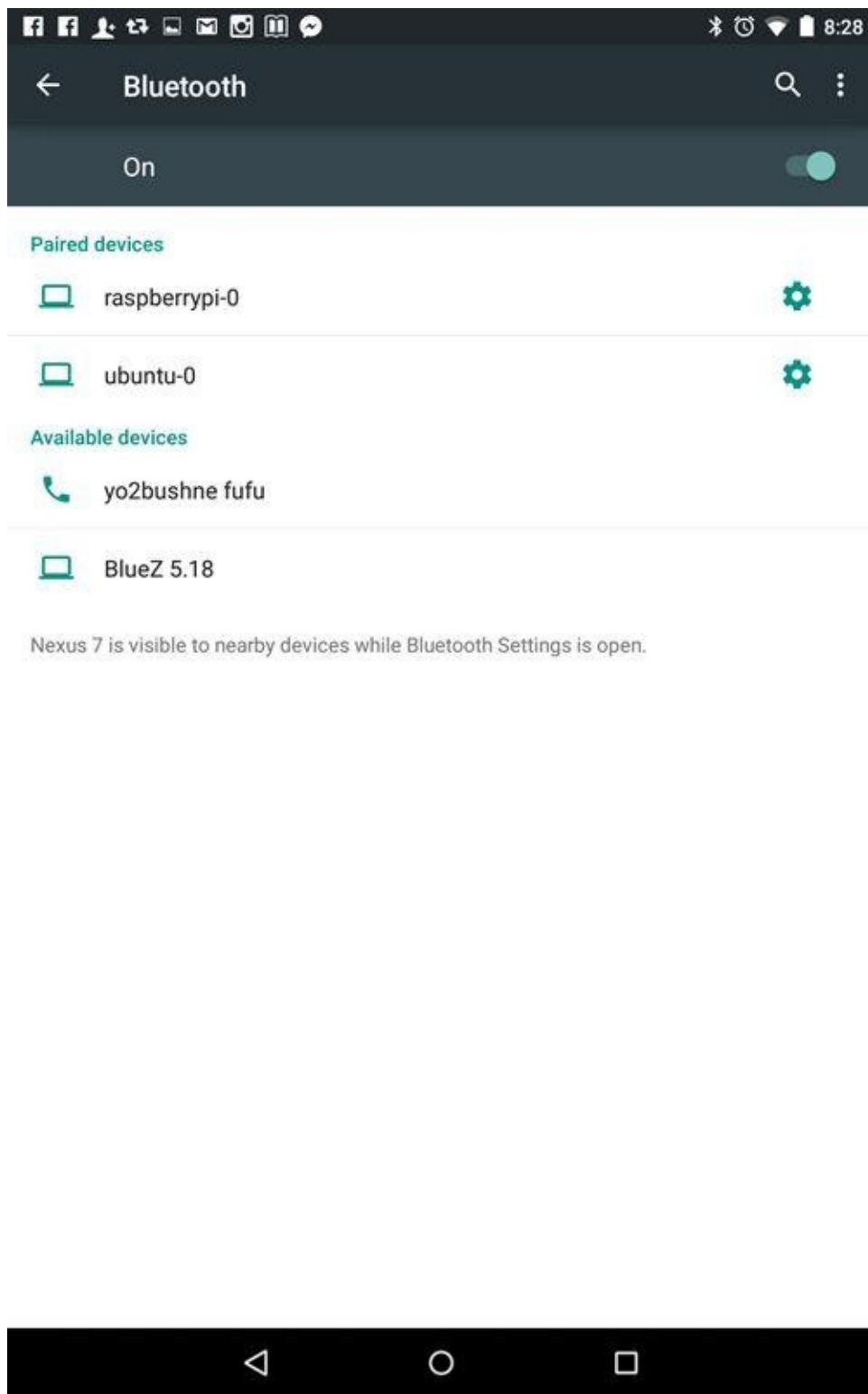
On Raspberry Pi terminal, type yes to confirm passkey by typing yes.



A screenshot of a terminal window titled "pi@raspberrypi: ~/book". The window contains the following text:

```
pi@raspberrypi ~$ sudo bluez-simple-agent hci0 D8:50:E6:7E:CB:87
RequestConfirmation (/org/bluez/6481/hci0/dev_D8_50_E6_7E_CB_87, 158721)
Confirm passkey (yes/no): yes
Release
New device (/org/bluez/6481/hci0/dev_D8_50_E6_7E_CB_87)
```

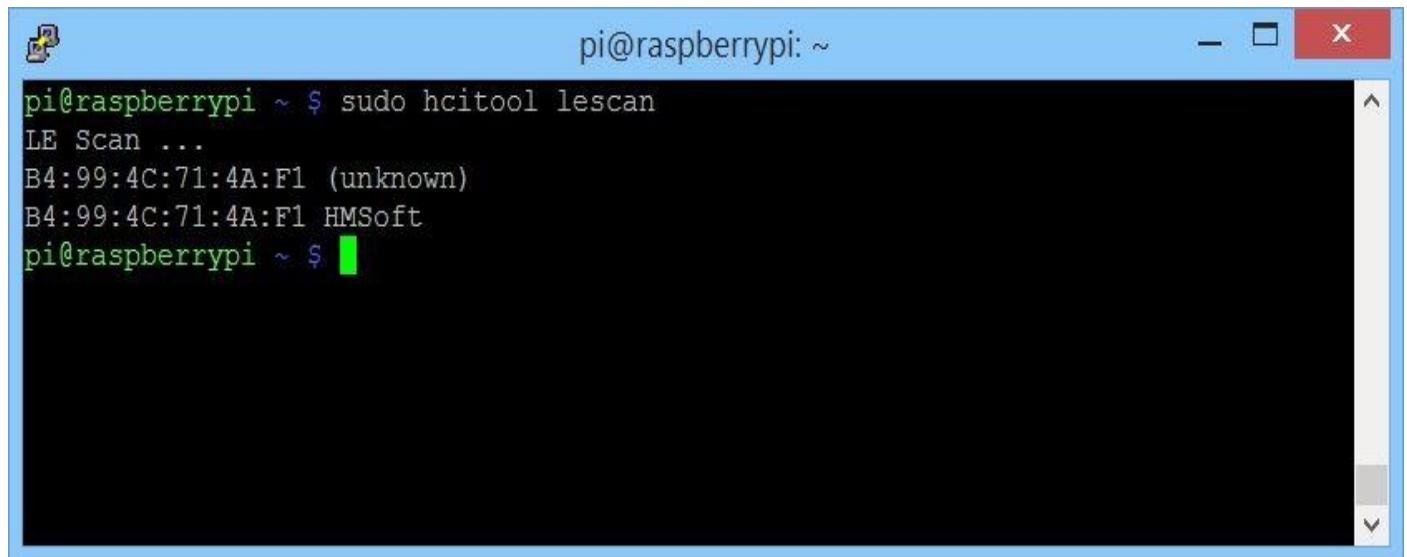
If success, your Raspberry Pi bluetooth will be paired to bluetooth target device.



We can use hcitool to check Bluetooth LE devices. Type the following command.

```
$ sudo hcitool lescan
```

If there is BLE device, **hcitool** tool will detect it.



A screenshot of a terminal window titled "pi@raspberrypi: ~". The window shows the command "sudo hcitool lescan" being run, followed by the output of a Low Energy scan. The output includes the MAC address and name of a nearby device: "B4:99:4C:71:4A:F1 (unknown)" and "B4:99:4C:71:4A:F1 HMSoft". The terminal has a blue header and a black body with white text. There are standard window controls (minimize, maximize, close) in the top right corner.

```
pi@raspberrypi ~ $ sudo hcitool lescan
LE Scan ...
B4:99:4C:71:4A:F1 (unknown)
B4:99:4C:71:4A:F1 HMSoft
pi@raspberrypi ~ $
```

After paired and connected, you can communicate among BLE devices.

## 4.4 BLE Development and iBeacon

iBeacon is Apple's implementation of Bluetooth low-energy (BLE) wireless technology to create a different way of providing location-based information and services to iPhones and other iOS devices.

In this section, we try to build iBeacon on Raspberry Pi. We use bleno, <https://github.com/sandeepmistry/bleno>, and noble, <https://github.com/sandeepmistry/noble> libraries to implement iBeacon. We also need Node.js, <http://nodejs.org/>, as its runtime.

Let's start.

### 4.4.1 Installing Node.js

In this section, we install Node.js from source code. To install Node.js, you can install required libraries. Type the following commands.

```
$ sudo apt-get update  
$ sudo apt-get install git-core build-essential openssl libssl-dev pkg-config
```

Now you can download Node.js source code and install it.

```
$ mkdir nodejs  
$ cd nodejs  
$ git clone git://github.com/joyent/node.git  
$ cd node  
$ sudo ./configure  
$ make  
$ sudo make install
```

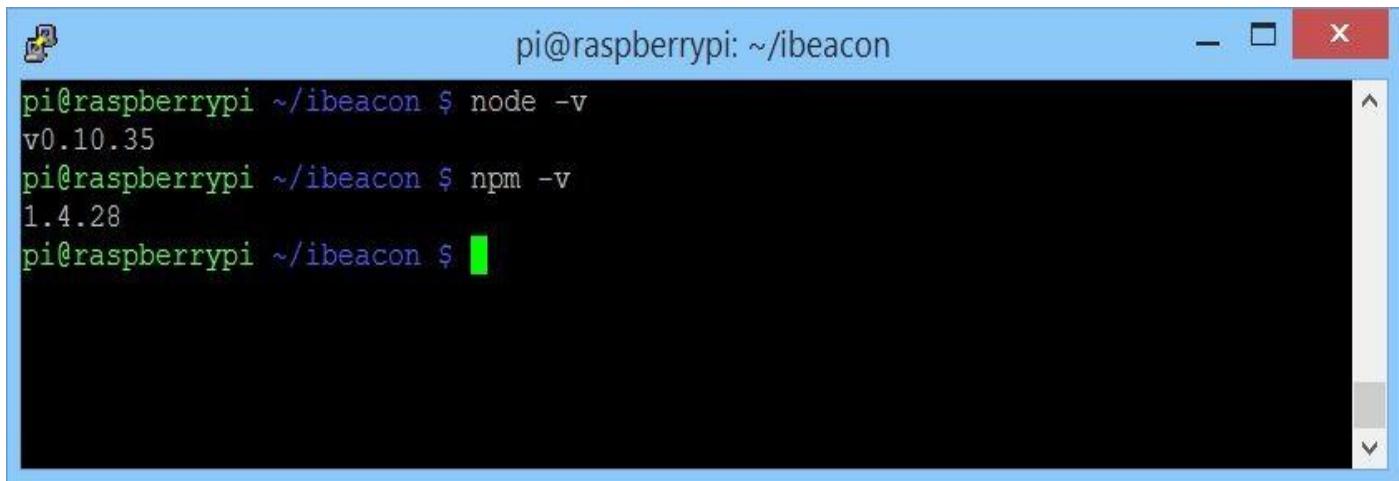
You can install Node.js from compiled binary for ARM. You can type these commands.

```
$ wget http://node-arm.herokuapp.com/node_latest_armhf.deb  
$ sudo dpkg -i node_latest_armhf.deb
```

After installed, you can verify it by checking its version.

```
$ node -v  
$ npm -v
```

A sample output can be seen in Figure below.



A screenshot of a terminal window titled "pi@raspberrypi: ~/ibeacon". The window shows the command "node -v" followed by "v0.10.35", then "npm -v" followed by "1.4.28", and finally a prompt "pi@raspberrypi ~/ibeacon \$". The terminal has a blue header bar with standard window controls.

```
pi@raspberrypi ~/ibeacon $ node -v
v0.10.35
pi@raspberrypi ~/ibeacon $ npm -v
1.4.28
pi@raspberrypi ~/ibeacon $
```

#### 4.4.2 Bleno

After that, we test our Raspberry Pi bluetooth for iBeacon using bleno, <https://github.com/sandeepmistry/bleno>, and noble, <https://github.com/sandeepmistry/noble> libraries.

To install them, you can install required libraries.

```
$ sudo apt-get install bluetooth bluez-utils libbluetooth-dev
```

Now you can install Noble and Bleno libraries.

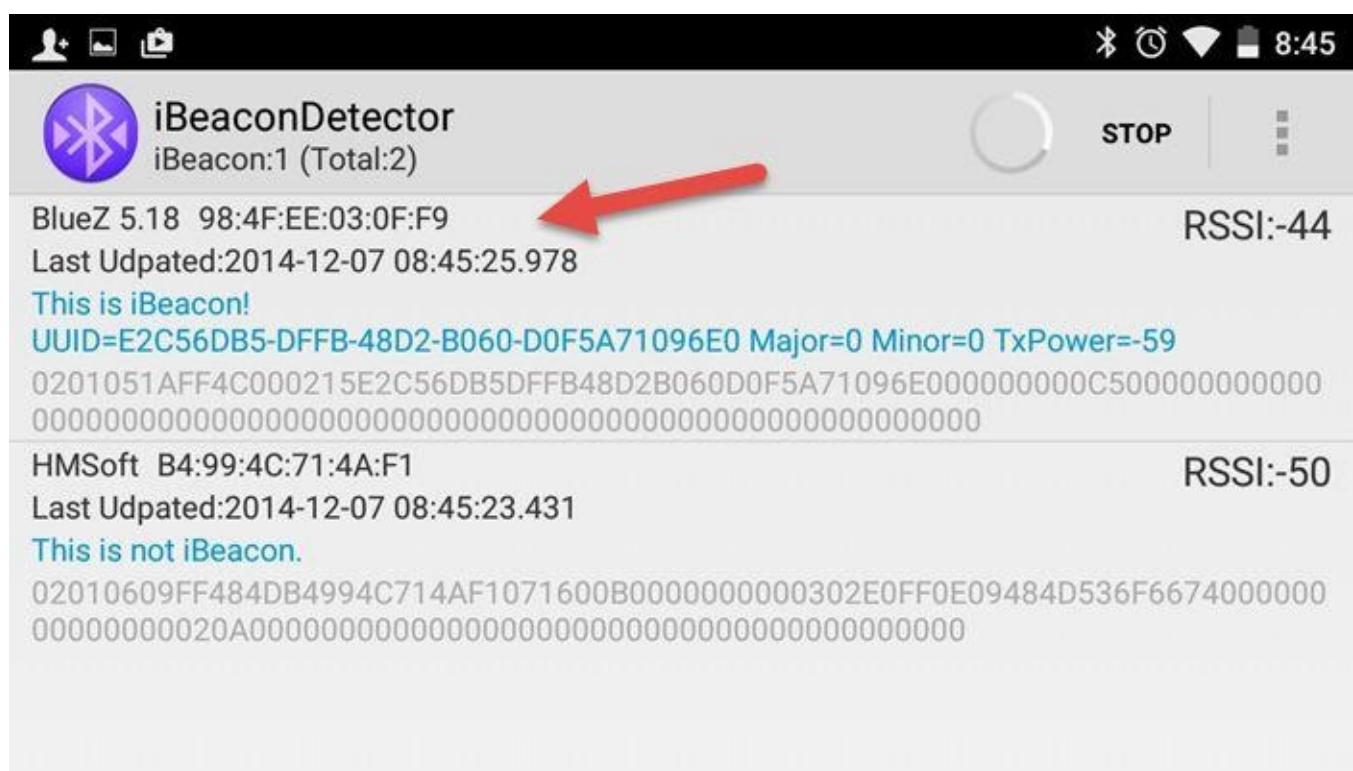
```
$ mkdir ibeacon
$ cd ibeacon/
$ sudo npm install -g async
$ sudo npm install noble
$ sudo npm install bleno
```

We can run test file, for instance, **test-ibeacon.js**, from bleno.

```
$ sudo node node_modules/bleno/test-ibeacon.js
```

```
pi@raspberrypi: ~/ibeacon
LINK(target) Release/l2cap-ble: Finished
make: Leaving directory '/home/pi/ibeacon/node_modules/bleno/build'
bleno install: done
bleno@0.1.8 node_modules/bleno
âââ debug@0.7.4
pi@raspberrypi ~/ibeacon $ node node_modules/bleno/test-ibeacon.js
bleno - iBeacon
bleno warning: adapter state unauthorized, please run as root or with sudo
on -> stateChange: unauthorized
on -> advertisingStop
^Cpi@raspberrypi ~/ibeacon
pi@raspberrypi ~/ibeacon $ sudo node node_modules/bleno/test-ibeacon.js
bleno - iBeacon
on -> stateChange: poweredOn
on -> advertisingStart
```

Now you can test it using iBeacon app, for instance, I use **iBeaconDetector** from Google Play store. It runs on my Nexus 7 2013 and detects Raspberry Pi iBeacon. A sample output is shown in Figure below.



I also test it using iBeacon Scanner on Android app.



8:48



## Device View

### DEVICE INFO

Device Name: BlueZ 5.18

Device Address: 98:4F:EE:03:0F:F9

### IBEACON DATA

Company ID: Apple, Inc. (0x4C)

UUID: e2c56db5-dff8-48d2-b060-d0f5a71096e0

Major: 0 (0x0)

Minor: 0 (0x0)

TX Power: -59 (0xFFFFFC5)

### RSSI INFO

Last Timestamp: 2014-12-07T07:47:37.954 UTC

Last RSSI: -50db

## **4.5 Further Reading**

To learn more BLE programming, you can read BLE resources on books and websites. I'm finishing my new book about Bluetooth Low Energy (BLE) Programming by Example. You can check it on my blog.

## **5. Wireless Communication Using 315/433 Mhz RF Modules**

This chapter explains how to build 315 Mhz and 433 Mhz RF communication on Raspberry Pi.

## **5.1 Getting Started**

In this chapter, we will build RF communication using RF communication with 315 Mhz and 433 Mhz Link Kit.

## 5.2 RF Transmitter and Receiver Link Kit

RF Link Kit modules with 315/433 Mhz are cheap wireless modules which we can build a simple wireless communication. This module uses Amplitude-shift keying (ASK) for modulation, [http://en.wikipedia.org/wiki/Amplitude-shift keying](http://en.wikipedia.org/wiki/Amplitude-shift_keying) . You can find these modules on the following online store.

- RF 315 Mhz from Seeedstudio, <http://www.seeedstudio.com/depot/315Mhz-RF-link-kit-p-76.html>
- RF 433 Mhz from Seeedstudio, <http://www.seeedstudio.com/depot/433Mhz-RF-link-kit-p-127.html>
- Amazon, <http://www.amazon.com>
- eBay, <http://www.ebay.com>

For information about these modules, you can read datasheet document on <http://wiki.jmoon.co/sensors/wireless/rfrxtx/> . Note: Please check your frequency usage regulation on your country.

A sample model of 433 Mhz RF Link kit can be seen in Figure below.



(source: <http://www.seeedstudio.com/>)

A sample model of 315 Mhz RF Link kit can be seen in Figure below.



(source: <http://www.seeedstudio.com/>)

For testing, we need the following hardware:

- RF Link Kit modules with 315/433 Mhz
- Arduino. I use Arduino Uno
- Raspberry Pi

For implementation, we use **rc-switch** library, <https://code.google.com/p/rc-switch/> . In one program, we can run as receiver and transmitter so our scenario is RF transmitter will be attached to Arduino and RF receiver will be attached to Raspberry Pi.



### 5.2.1 Wiring

In general, RF Link Kit module shows pins with description, for instance, VCC, GND and Data.

The following is wiring for Arduino and RF transmitter:

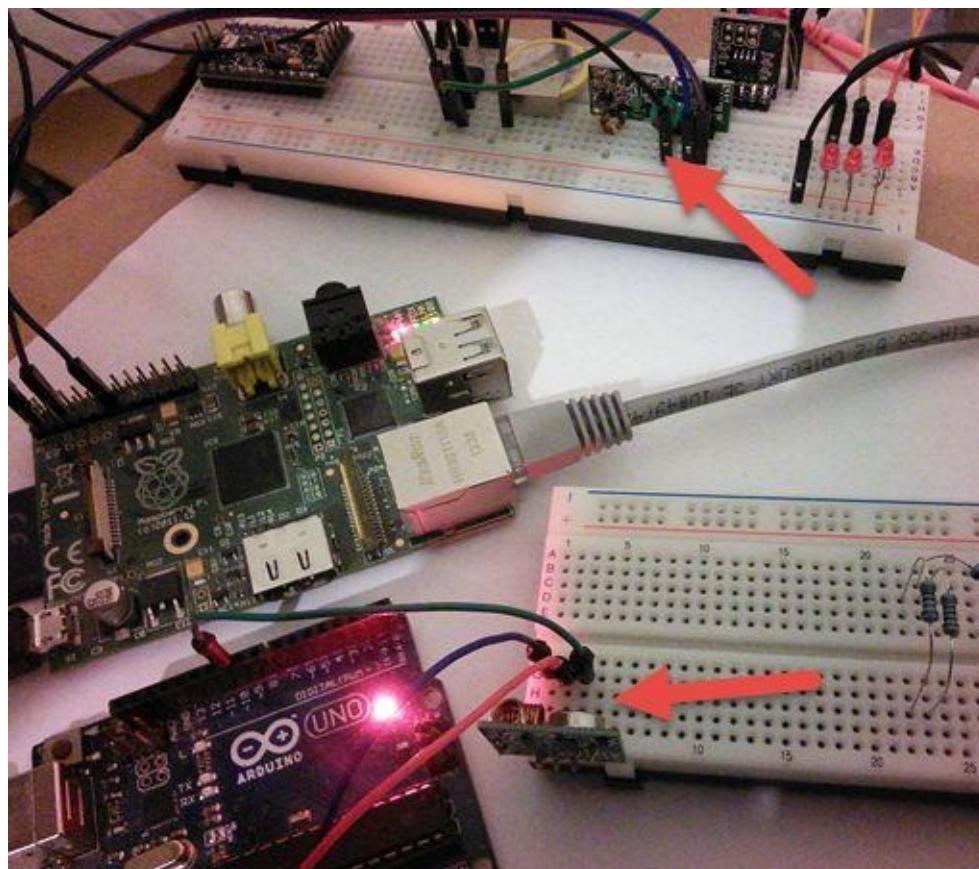
- GND Arduino is connected to GND RF transmitter
- VCC +5V Arduino is connected to +VCC RF transmitter
- Digital 10 (PWM) Arduino is connected to Data RF transmitter

The following is wiring for Raspberry Pi and RF receiver:

- GND Raspberry Pi is connected to GND RF receiver
- VCC +5V Raspberry Pi is connected to +VCC RF receiver
- GPIO2 Raspberry Pi is connected to Data RF receiver

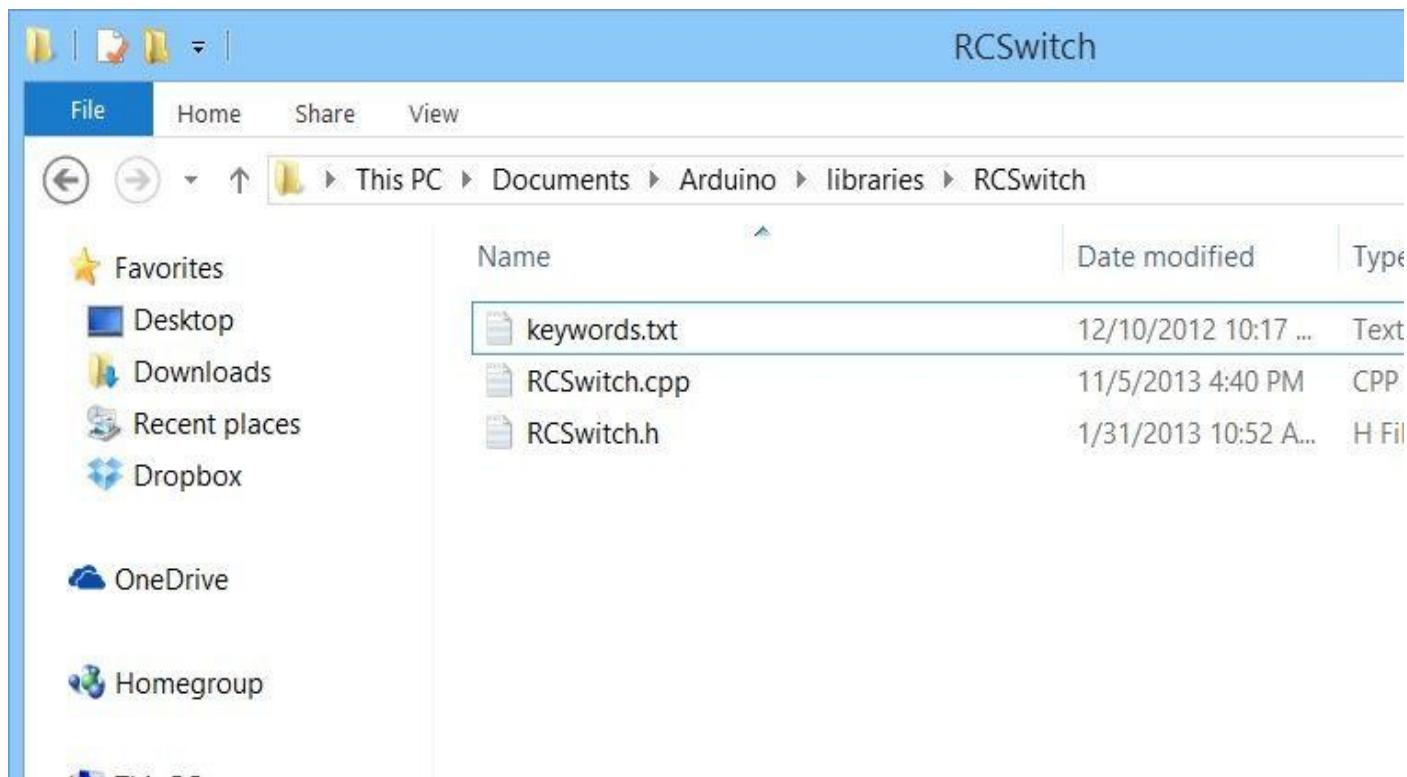
GPIO2 Raspberry Pi is based on WiringPi mapping (see section 5.2.3). If you see Raspberry Pi GPIO (section 1.6), GPIO2 is located on GPIO21/GPIO27 Raspberry Pi.

Hardware implementation can be seen in Figure below.



## 5.2.2 Building Transmitter Application

We use **rc-switch** library, <https://code.google.com/p/rc-switch/>, to build a transmitter application. Download this library, **RCSwitch.zip**. You can extract it to Arduino library folder.



You also can import **RCSwitch.zip** from menu Sketch-> Import library.

Now you can write a program for sending data. In this case, we will send data counter from 1000 until 1100.

```
#include <RCSwitch.h>
RCSwitch mySwitch = RCSwitch();
const int led = 13;
long counter;

void setup() {
    pinMode(led, OUTPUT);
    Serial.begin(9600);
    counter = 1000;
    // Transmitter is connected to Arduino Pin #10
    mySwitch.enableTransmit(10);
}

void loop() {

    digitalWrite(led, HIGH);
    Serial.print("Send: ");
    Serial.print(counter);
    Serial.println("");

    mySwitch.send(counter, 16);
    digitalWrite(led, LOW);
    delay(2000);

    counter++;
    if(counter>1100)
        counter = 1000;
}
```

}

The screenshot shows the Arduino IDE interface with the title bar "rf433\_transmitter | Arduino 1.0.5-r2". The menu bar includes File, Edit, Sketch, Tools, and Help. The toolbar has icons for upload, download, and serial monitor. The code editor contains the following sketch:

```
#include <RCSwitch.h>
RCSwitch mySwitch = RCSwitch();
const int led = 13;
long counter;

void setup() {
  pinMode(led, OUTPUT);
  Serial.begin(9600);
  counter = 1000;
  // Transmitter is connected to Arduino Pin #10
  mySwitch.enableTransmit(10);
}

void loop() {
  digitalWrite(led, HIGH);
  Serial.print("Send: ");
  Serial.print(counter);
  counter++;
}
```

The status bar at the bottom shows "Done uploading." and "Binary sketch size: 7,074 bytes (of a 32,256 byte maximum)".

## Explanation:

- In setup(), we activate Serial port and LED 13
- Initialize counter = 1000
- Enabling rc-switch to enable to transmitt counter data
- In loop(), turn on LED 13 and then send data using send() from rc-switch library
- counter value is be increased. If counter value is more than 1100, we set counter = 1000

Save this program. Try to compile. Make sure there is no error in compiling.

## 5.2.3 Building Receiver Application

Firstly, we install **wiringPi**. Type the following command.

```
$ git clone git://git.drogon.net/wiringPi
$ cd wiringPi/
$ sudo su
$ ./build
```

The screenshot shows a terminal window titled "pi@raspberrypi: ~/book/wiringPi". The terminal output is as follows:

```
[Compile] piGlow.c
[Link (Dynamic)]
[Install Headers]
[Install Dynamic Lib]

GPIO Utility
[Compile] gpio.c
[Compile] extensions.c
[Compile] readall.c
[Compile] pins.c
[Link]
[Install]

All Done.

NOTE: To compile programs with wiringPi, you need to add:
      -lwiringPi
      to your compile line(s) To use the Gertboard, MaxDetect, etc.
      code (the devLib), you need to also add:
      -lwiringPiDev
      to your compile line(s).

root@raspberrypi:/home/pi/book/wiringPi# exit
pi@raspberrypi ~/book/wiringPi $
```

If done, press Ctrl-D to back from **sudo su** command. Now you can verify Raspberry Pi using **wiringPi**. Type this command.

```
$ gpio readall
```

For instance, I got a response, shown in Figure below.

```

pi@raspberrypi ~$ gpio readall
+-----+-----+-----+---+-----+-----+-----+-----+-----+
| 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 | ALTO | TxD | 15 | 14 |
|     |     | 0v |     |   | 9 || 10 | 1 | ALTO | 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 | OUT | GPIO. 6 | 6 | 25 |
| 11 | 14 | SCLK | IN | 0 | 23 || 24 | 1 | IN | CE0 | 10 | 8 |
|     |     | 0v |     |   | 25 || 26 | 1 | IN | CE1 | 11 | 7 |
+-----+-----+-----+---+-----+-----+-----+-----+-----+
| 28 | 17 | GPIO.17 | IN | 0 | 51 || 52 | 0 | IN | GPIO.18 | 18 | 29 |
| 30 | 19 | GPIO.19 | IN | 0 | 53 || 54 | 0 | IN | GPIO.20 | 20 | 31 |
+-----+-----+-----+---+-----+-----+-----+-----+-----+
| BCM | wPi |   Name | Mode | V | Physical | V | Mode | Name | wPi | BCM |
+-----+-----+-----+---+-----+-----+-----+-----+-----+
pi@raspberrypi ~$ 
```

Data pin RF module is connected to GPIO.2 (see Figure above).

To build receiver application, I use **rc-switch** library for Raspberry Pi, <https://github.com/r10r/rcswitch-pi>. Now we build sniffer to listening incoming data from RF transmitter. I modified code from <https://github.com/ninjablocks/433Utils>.

Create a file, called **RFSniffer.cpp**, and write this code.

```

#include "RCSwitch.h"
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>

RCSwitch mySwitch;

int main(int argc, char *argv[]) {

    // This pin is not the first pin on the RPi GPIO header!
    // Consult https://projects.drogon.net/raspberry-pi/wiringpi/pins/
    // for more information.
    int PIN = 2;

    if(wiringPiSetup() == -1)
        return 0;

    mySwitch = RCSwitch(); 
```

```

mySwitch.enableReceive(PIN);

while(1) {
    if (mySwitch.available()) {

        int value = mySwitch.getReceivedValue();

        if (value == 0) {
            printf("Unknown encoding");
        } else {

            printf("Received %i\n", mySwitch.getReceivedValue() );
        }

        mySwitch.resetAvailable();
    }
    sleep(1);

}

exit(0);
}

```

Save this code.

The next step is to create **Makefile** file. Write this script.

```

all: RFSniffer

RFSniffer: RCSwitch.o RFSniffer.o
    $(CXX) $(CXXFLAGS) $(LDFLAGS) $+ -o $@ -lwiringPi

clean:
    $(RM) *.o RFSniffer

```

Now you can compile using **make** command.

```
$ make
```

```

pi@raspberrypi:~/book/rf433_sniffer
pi@raspberrypi ~/book/rf433_sniffer $ make
g++ -c -o RCSwitch.o RCSwitch.cpp
g++ -c -o RFSniffer.o RFSniffer.cpp
g++ RCSwitch.o RFSniffer.o -o RFSniffer -lwiringPi
pi@raspberrypi ~/book/rf433_sniffer $

```

## 5.2.4 Testing

On Arduino software, you can compile and deploy the program to Arduino board. For a Raspberry Pi program, you can compile and run it.

Don't forget to point Transmitter module to receiver module. It addresses propagation issues.

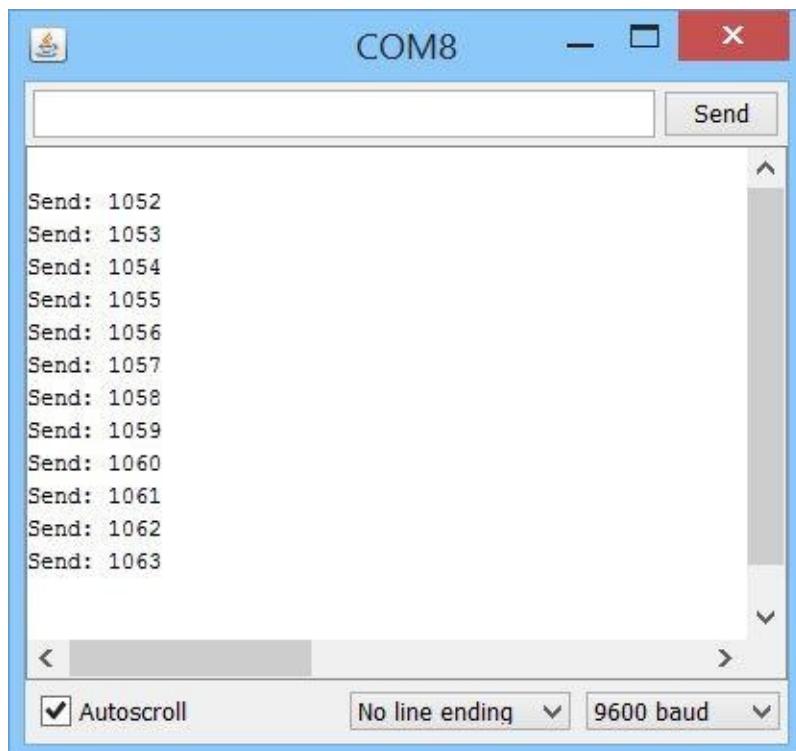
A sample output of **RFSniffer** is shown in Figure below.



A screenshot of a terminal window titled "pi@raspberrypi: ~/book/rf433\_sniffer". The window shows the command "sudo ./RFSniffer" being run, followed by a series of "Received" messages with values 1055 through 1061. The terminal has a blue header bar and a black body with white text. A vertical scroll bar is visible on the right side.

```
pi@raspberrypi: ~/book/rf433_sniffer $ sudo ./RFSniffer
Received 1055
Received 1056
Received 1057
Received 1058
Received 1059
Received 1060
Received 1061
pi@raspberrypi: ~/book/rf433_sniffer $
```

If you already open Serial Monitor of Arduino, you can see data which is sent from RF transmitter.



## **6. Wireless Communication Using 2.4 GHz RF Modules**

This chapter explains how to build wireless communication using 2.4 GHz RF modules.

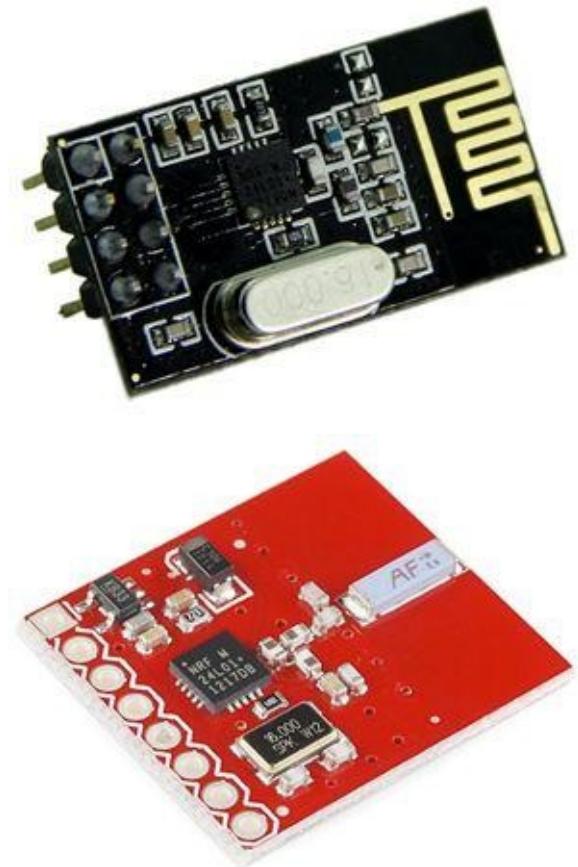
## 6.1 Getting Started

We can build wireless communication with 2.4 GHz using any 2.4 GHz module. In this chapter, I use nRF24L01 or nRF24L01+ to implement wireless communication.

The nRF24L01 is a single chip 2.4GHz transceiver with an embedded baseband protocol engine (Enhanced ShockBurst™), designed for ultra low power wireless applications. The nRF24L01 is designed for operation in the world wide ISM frequency band at 2.400 - 2.4835GHz. An MCU (microcontroller) and very few external passive components are needed to design a radio system with the nRF24L01.

The nRF24L01+. The ‘+’ version of the IC has improved range, sensitivity, and data rates. The command set is backward compatible with the original nRF24L01.

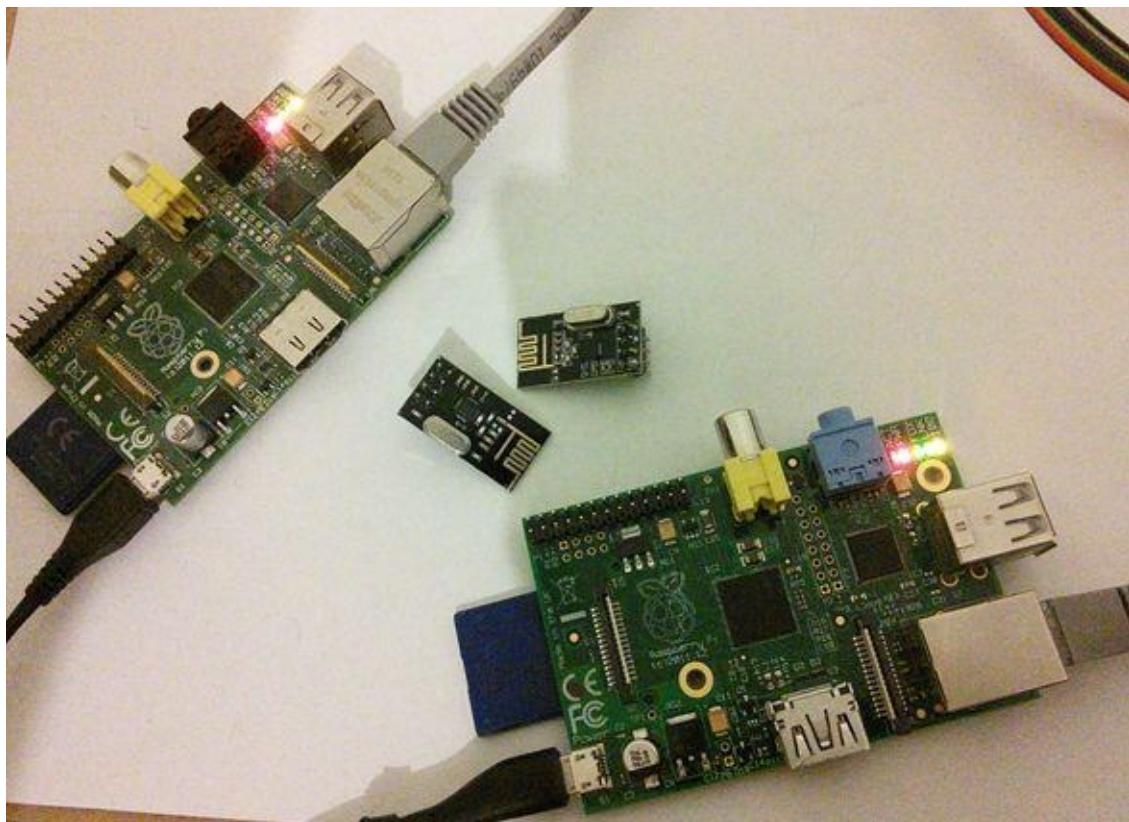
A sample of nRF24L01 module can be seen in Figure below.



For testing, we need the following required hardware:

- Two Raspberry Pi devices
- Two nRF24L01 modules
- Cables

You can see my Raspberry Pi and nRF24L01 modules as below.



To build RF communication using nRF24L01, we need nRF24L01 library/module. In this section, I use RF24, <https://github.com/stanleyseow/RF24/tree/master/RPi/RF24> . To see this library, you must enable SPI on Raspberry Pi. Read section 6.4.

## 6.2 Getting Hardware

You can get nRF24L01 module on the following online store

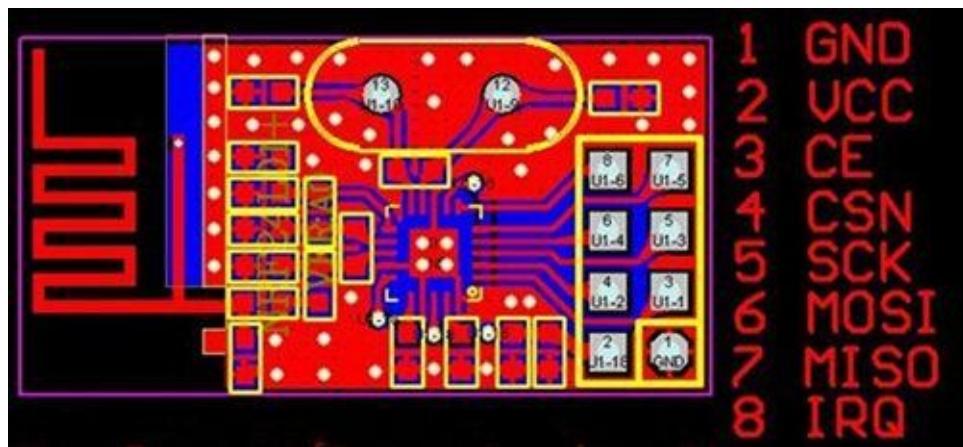
- Seeedstudio, <http://www.seeedstudio.com/depot/nRF24L01Module-p-1394.html>
- Sparkfun, <https://www.sparkfun.com/search/results?term=nRF24L01>
- eBay.com
- Amazon

You also can find this module on your local electronic store.

## 6.3 Wiring

You can read nRF24L01/nRF24L01+ datasheet on this site, [https://www.sparkfun.com/datasheets/Wireless/Nordic/nRF24L01P\\_Product\\_Specification](https://www.sparkfun.com/datasheets/Wireless/Nordic/nRF24L01P_Product_Specification). You can see pinout for nRF24L01/nRF24L01+ module.

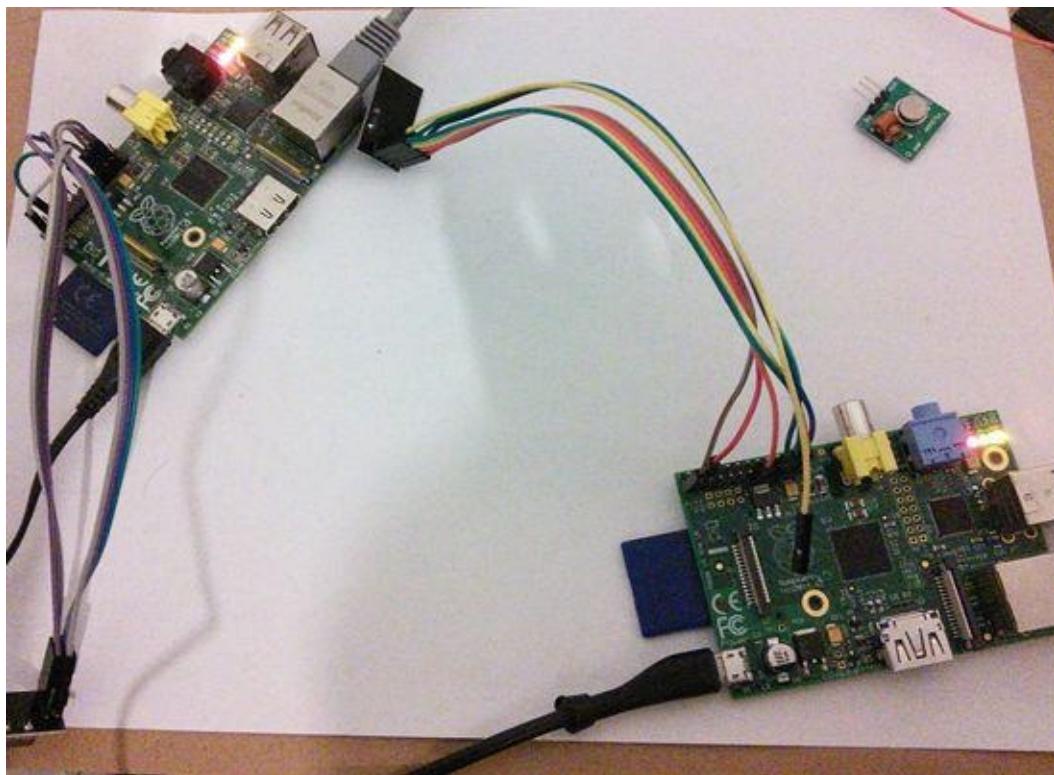
Based nRF24L01/nRF24L01+ datasheet, you can see the pinout on the following Figure. Note: It's top view!!.



To build our lab, you do wiring, shown in Table below.

NRF24L01	Rpi	
	Name	Physical
VCC	+3.3V	
GND	GND	
CE	GPIO3	15
CSN	CEO	24
SCK	SCLK	23
MOSI	MOSI	19
MISO	MISO	21
IRQ	Not connected	

A sample of hardware implementation can be seen in Figure below.

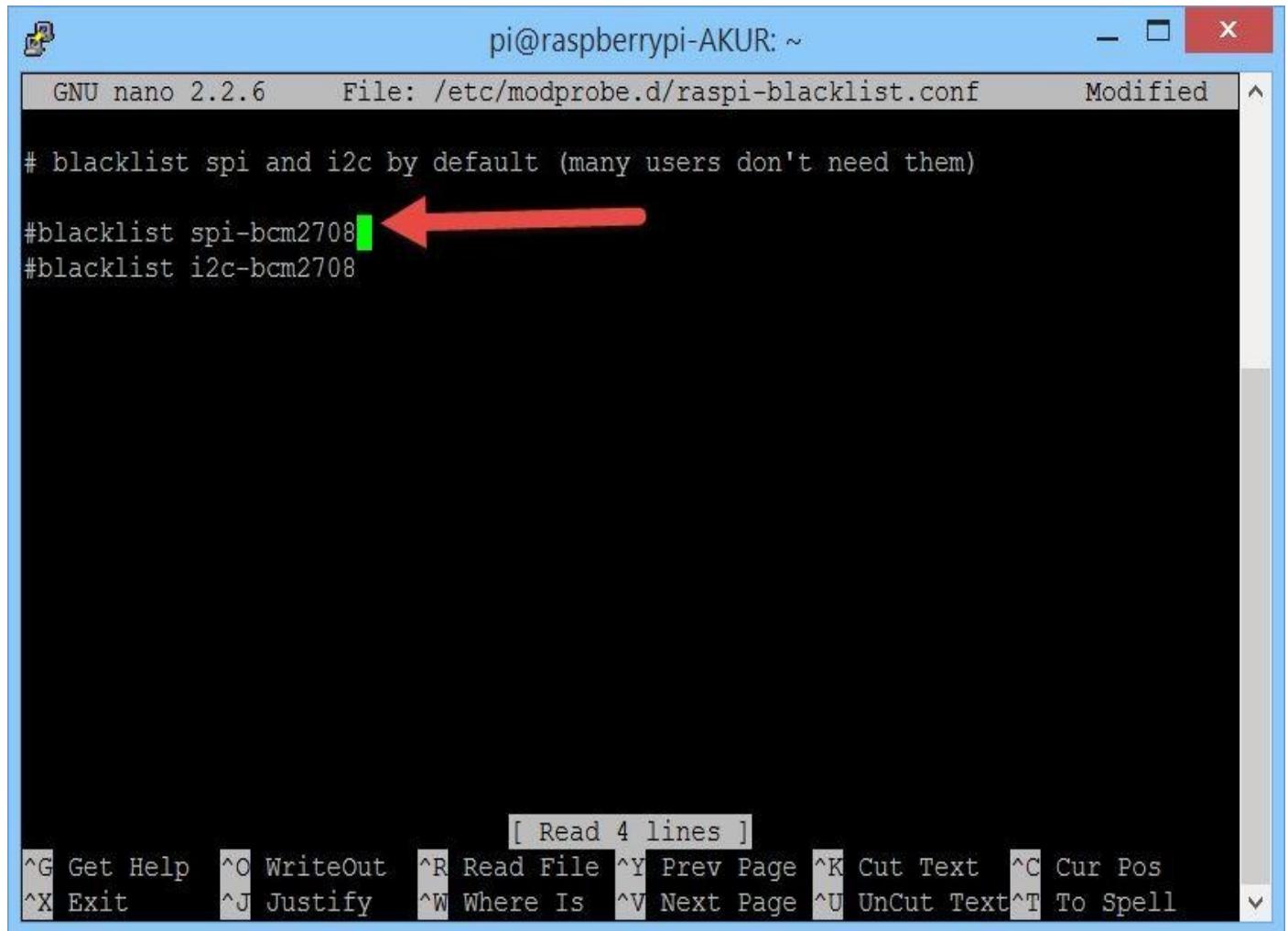


## 6.4 Configuring SPI on Raspberry Pi

To access SPI, we must configure Raspberry Pi to enable this communication. Firstly, we remove SPI from module blacklist. Open file **/etc/modprobe.d/raspi-blacklist.conf**, for instance I use nano editor.

```
$ sudo nano /etc/modprobe.d/raspi-blacklist.conf
```

Then add “#” on **blacklist spi-bcm2708**. The following is final configuration.



```
pi@raspberrypi-AKUR: ~
GNU nano 2.2.6      File: /etc/modprobe.d/raspi-blacklist.conf      Modified
^A Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit      ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
[ Read 4 lines ]
# blacklist spi and i2c by default (many users don't need them)
#blacklist spi-bcm2708
#blacklist i2c-bcm2708
```

Save this changed code and close this file.

Now you can reboot your Raspberry Pi.

After that, you can verify if **spi\_bcm2708** is not be blocked by typing the following command.

```
$ lsmod
```

The following is response output.

```
pi@raspberrypi-AKUR: ~ $ lsmod
Module           Size  Used by
i2c_dev          5587  0
snd_bcm2835     12808  0
snd_pcm          74834  1 snd_bcm2835
snd_seq          52536  0
snd_timer        19698  2 snd_seq,snd_pcm
snd_seq_device   6300   1 snd_seq
snd              52489  5 snd_seq_device,snd_timer,snd_seq,snd_pcm,snd_bcm
2835
snd_page_alloc   4951   1 snd_pcm
spidev           5136   0
spi_bcm2708     4401   0 ←
i2c_bcm2708     3681   0
pi@raspberrypi-AKUR: ~ $
```

You can see spi\_bcm2708.

You also can verify SPI bus using the following command.

```
$ ls /dev/spidev*
```

```
pi@raspberrypi-AKUR: ~ $ ls /dev/spidev*
/dev/spidev0.0  /dev/spidev0.1
pi@raspberrypi-AKUR: ~ $
```

## 6.5 Installing nRF24L01 Module

We use RF24 library, <https://github.com/stanleyseow/RF24/tree/master/RPi/RF24> , to access nRF24L01 module. You can install it by typing the following command.

```
$ git clone https://github.com/stanleyseow/RF24.git  
$ cd RF24/  
$ cd RPi/RF24/  
$ make  
$ sudo make install
```

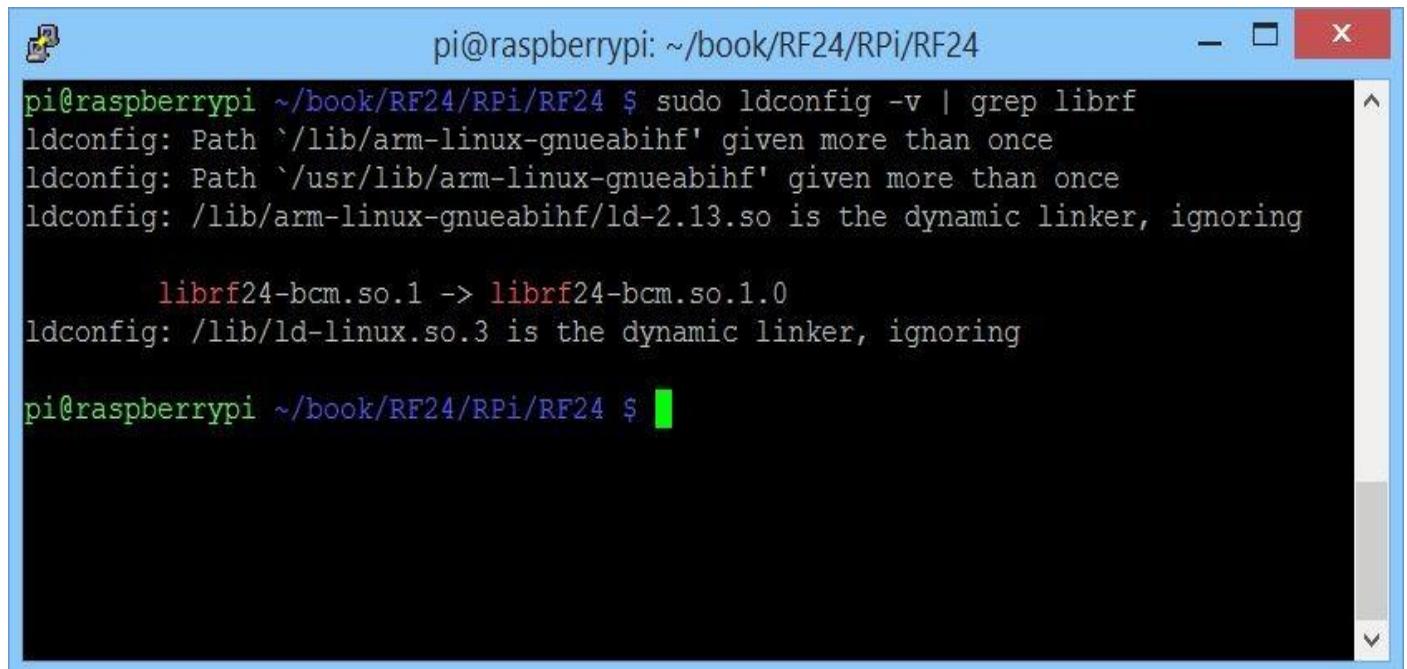
If you don't install git yet, you can install git.

```
$ sudo apt-get update  
$ sudo apt-get install git
```

After that, you can verify this library by typing this command.

```
$ sudo ldconfig -v | grep librf
```

You will see a path of library.



A screenshot of a terminal window titled "pi@raspberrypi: ~ /book/RF24/RPi/RF24". The window shows the command \$ sudo ldconfig -v | grep librf being run, followed by its output. The output indicates that the dynamic linker is ignoring multiple paths for the librf library, specifically /lib/arm-linux-gnueabihf and /usr/lib/arm-linux-gnueabihf, and instead using /lib/arm-linux-gnueabihf/ld-2.13.so as the dynamic linker.

```
pi@raspberrypi ~ /book/RF24/RPi/RF24 $ sudo ldconfig -v | grep librf  
ldconfig: Path `/lib/arm-linux-gnueabihf' given more than once  
ldconfig: Path `/usr/lib/arm-linux-gnueabihf' given more than once  
ldconfig: /lib/arm-linux-gnueabihf/ld-2.13.so is the dynamic linker, ignoring  
  
    librf24-bcm.so.1 -> librf24-bcm.so.1.0  
ldconfig: /lib/ld-linux.so.3 is the dynamic linker, ignoring  
pi@raspberrypi ~ /book/RF24/RPi/RF24 $
```

## 6.6 Testing

For testing, we use code examples from RF24 library. The first demo is scanner. The last demo is to build a sample RF communication.

Firstly, we compile our demo on examples/ folder.

```
$ cd examples/  
$ make
```

We also compile our demo on /examples/extra/

```
$ cd examples/extra/  
$ make
```

### 6.6.1 Scanner Testing

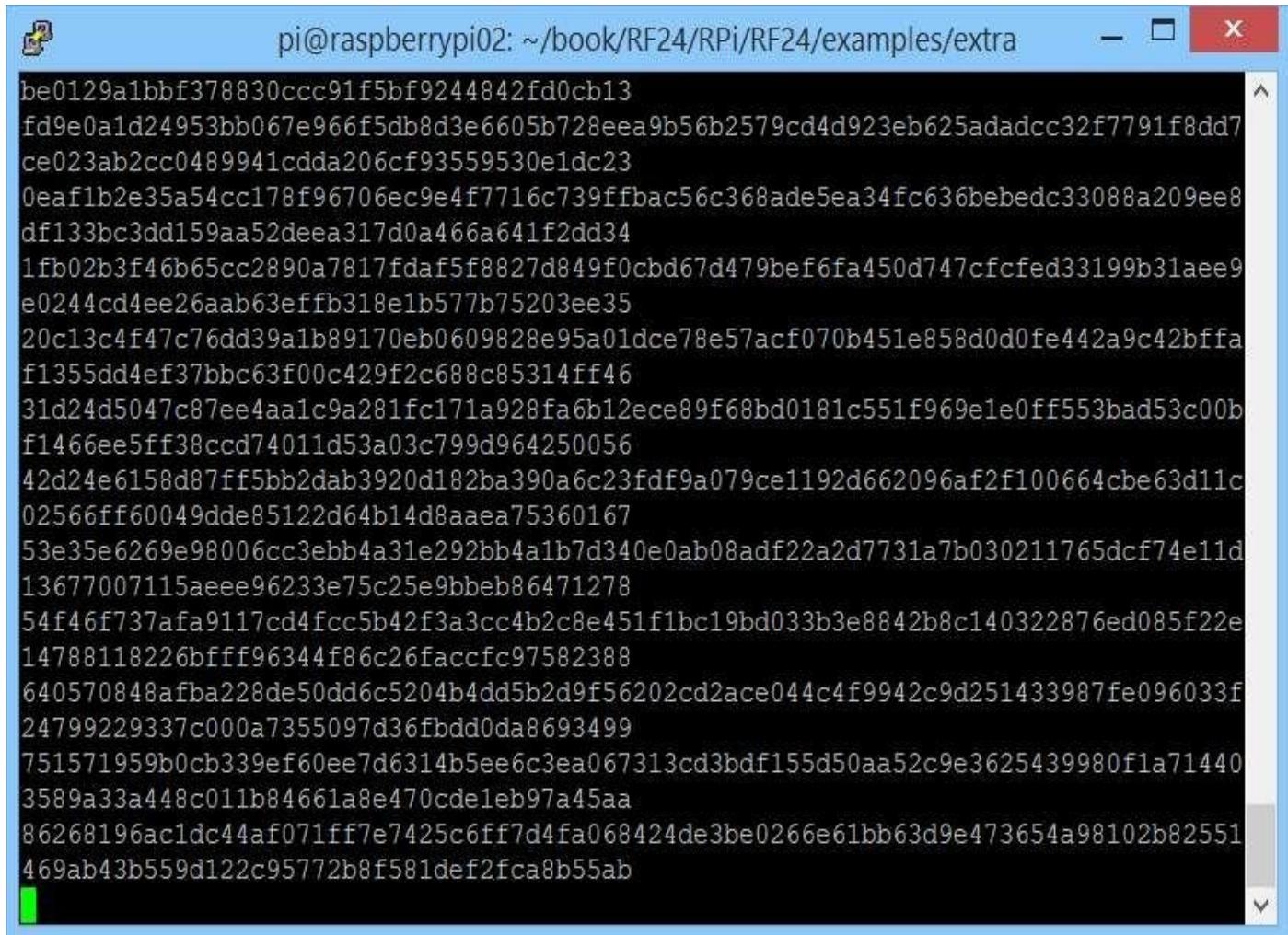
The first demo is scanner application. This app will sniff data from nRF24L01 module.

You can run it by typing this command.

```
$ sudo ./scanner
```

Note: This demo is located on /examples/extra/ folder.

A sample output can be seen in Figure below.



```
be0129a1bbf378830ccc91f5bf9244842fd0cb13  
fd9e0a1d24953bb067e966f5db8d3e6605b728eea9b56b2579cd4d923eb625adadcc32f7791f8dd7  
ce023ab2cc0489941cdda206cf93559530e1dc23  
0eaf1b2e35a54cc178f96706ec9e4f7716c739ffbac56c368ade5ea34fc636bebedc33088a209ee8  
df133bc3dd159aa52deea317d0a466a641f2dd34  
1fb02b3f46b65cc2890a7817fdaf5f8827d849f0cbd67d479bef6fa450d747cfcfed33199b31aee9  
e0244cd4ee26aab63effb318e1b577b75203ee35  
20c13c4f47c76dd39a1b89170eb0609828e95a01dce78e57acf070b451e858d0d0fe442a9c42bffa  
f1355dd4ef37bbc63f00c429f2c688c85314ff46  
31d24d5047c87ee4aa1c9a281fc171a928fa6b12ece89f68bd0181c551f969e1e0ff553bad53c00b  
f1466ee5ff38ccd74011d53a03c799d964250056  
42d24e6158d87ff5bb2dab3920d182ba390a6c23fdf9a079ce1192d662096af2f100664cbe63d11c  
02566ff60049dde85122d64b14d8aaea75360167  
53e35e6269e98006cc3ebb4a31e292bb4a1b7d340e0ab08adf22a2d7731a7b030211765dcf74e11d  
13677007115aehee96233e75c25e9bbeb86471278  
54f46f737afa9117cd4fcc5b42f3a3cc4b2c8e451f1bc19bd033b3e8842b8c140322876ed085f22e  
14788118226bfff96344f86c26faccfc97582388  
640570848afba228de50dd6c5204b4dd5b2d9f56202cd2ace044c4f9942c9d251433987fe096033f  
24799229337c000a7355097d36fbdd0da8693499  
751571959b0cb339ef60ee7d6314b5ee6c3ea067313cd3bdf155d50aa52c9e3625439980f1a71440  
3589a33a448c011b84661a8e470cde1eb97a45aa  
86268196ac1dc44af071ff7e7425c6ff7d4fa068424de3be0266e61bb63d9e473654a98102b82551  
469ab43b559d122c95772b8f581def2fca8b55ab
```

## 6.6.2 Receiver and Transmitter Testing

We also can test receiver and transmitter on nRF24L01 modules. We need two nRF24L01 modules and two Raspberry Pi. Attach the module to Raspberry Pi.

Each program can be run by typing this command.

```
$ sudo ./transfer
```

Note: This demo is located on /examples/extr/ folder.

Fill 0 for receiver role and 1 for transmitter role.

A sample output of receiver app can be seen in Figure below.

```
pi@raspberrypi: ~/book/RF24/RPi/RF24/examples
Rate: 47.07 KB/s
Payload Count: 1471
Rate: 47.68 KB/s
Payload Count: 1490
Rate: 47.23 KB/s
Payload Count: 1476
Rate: 47.23 KB/s
Payload Count: 1476
Rate: 47.78 KB/s
Payload Count: 1493
Rate: 22.56 KB/s
Payload Count: 705
```

A sample output of transmitter app can be seen in Figure below.

```
pi@raspberrypi02: ~/book/RF24/RPi/RF24/examples
RF_SETUP      = 0x07
CONFIG        = 0x0a
DYNPD/FEATURE = 0x00 0x00
Data Rate     = 1MBPS
Model         = nRF24L01+
CRC Length   = 8 bits
PA Power     = PA_MAX

***** Role Setup *****
Choose a role: Enter 0 for receiver, 1 for transmitter (CTRL+C to exit)
>1
Role: Ping Out, starting transmission

Initiating Basic Data Transfer
Transfer complete at 22.32 KB/s
408 of 10000 Packets Failed to Send
Initiating Basic Data Transfer
Transfer complete at 47.28 KB/s
0 of 10000 Packets Failed to Send
Initiating Basic Data Transfer
Transfer complete at 47.46 KB/s
0 of 10000 Packets Failed to Send
Initiating Basic Data Transfer
```

## **7. IEEE 802.15.4 LR-WPAN Networks**

This chapter explains how to work with IEEE 802.15.4 networks on Raspberry Pi.

## 7.1 Getting Started

IEEE 802.15.4 is a standard which specifies the physical layer and media access control for low-rate wireless personal area networks (LR-WPANs). It is maintained by the IEEE 802.15 working group, which has defined it in 2003. You can read IEEE 802.15.4 document on <http://standards.ieee.org/about/get/802/802.15.html>.

Depending on the application requirements, an IEEE 802.15.4 LR-WPAN operates in either of two topologies: the star topology or the peer-to-peer topology. In IEEE 802.15.4 standard, a node can be Full Function Device (FFD) or Reduced Function Device (RFD). A FFD also can be PAN/Network Coordinator.

If you want to know about IEEE 802.15.4, you can read IEEE 802.15.4 standard document or you can read my recommended book, **Low-Rate Wireless Personal Area Networks: Enabling Wireless Sensors With IEEE 802.15.4** by Jose A. Gutierrez, Ludwig Winkel, Edgar H. Callaway Jr., Raymond L. Barrett Jr. You buy this book on <http://www.amazon.com/Low-Rate-Wireless-Personal-Networks/dp/073816285X/>.

On the next section, we will implement IEEE 802.15.4 using XBee IEEE 802.15.4.

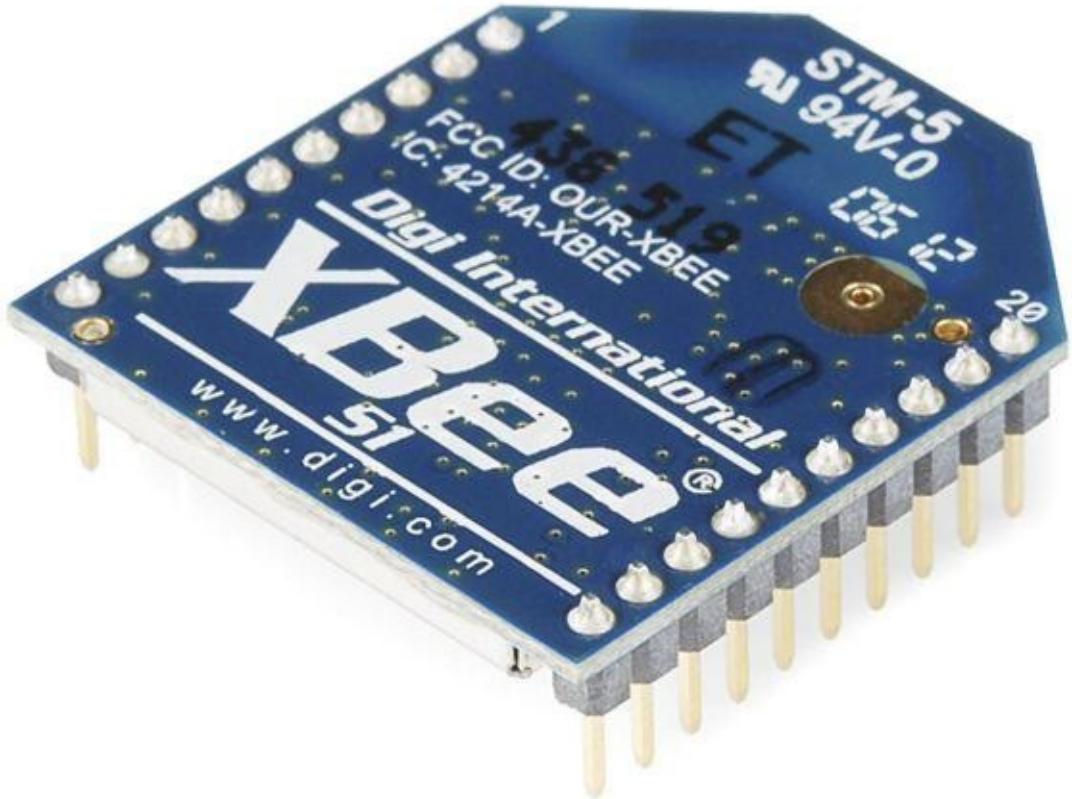
## 7.2 XBee IEEE 802.15.4

Digi International provides XBee products with specific features. You can see product comparison for XBee on this document, [http://www.digi.com/pdf/chart\\_xbee\\_rf\\_features.pdf](http://www.digi.com/pdf/chart_xbee_rf_features.pdf).

In this book, we will focus on XBee IEEE 802.15.4. XBee-PRO 802.15.4 modules are embedded solutions providing wireless end-point connectivity to devices. These modules use the IEEE 802.15.4 networking protocol for fast point-to-multipoint or peer-to-peer networking. They are designed for high-throughput applications requiring low latency and predictable communication timing.

You can read XBee product with IEEE 802.15.4 protocol on this website, <http://www.digi.com/products/wireless-wired-embedded-solutions/zigbee-rf-modules/point-multipoint-rfmodules/xbee-series1-module>. The following is a sample picture of XBee 802.15.4.





### 7.2.1 Getting Hardware

How to get XBee IEEE 802.15.4 device?

Officially you can buy it from Digi International, <http://www.digi.com/howtobuy/> . You

can buy this product from official distributor, <http://www.digi.com/howtobuy/find-a-distributor> .

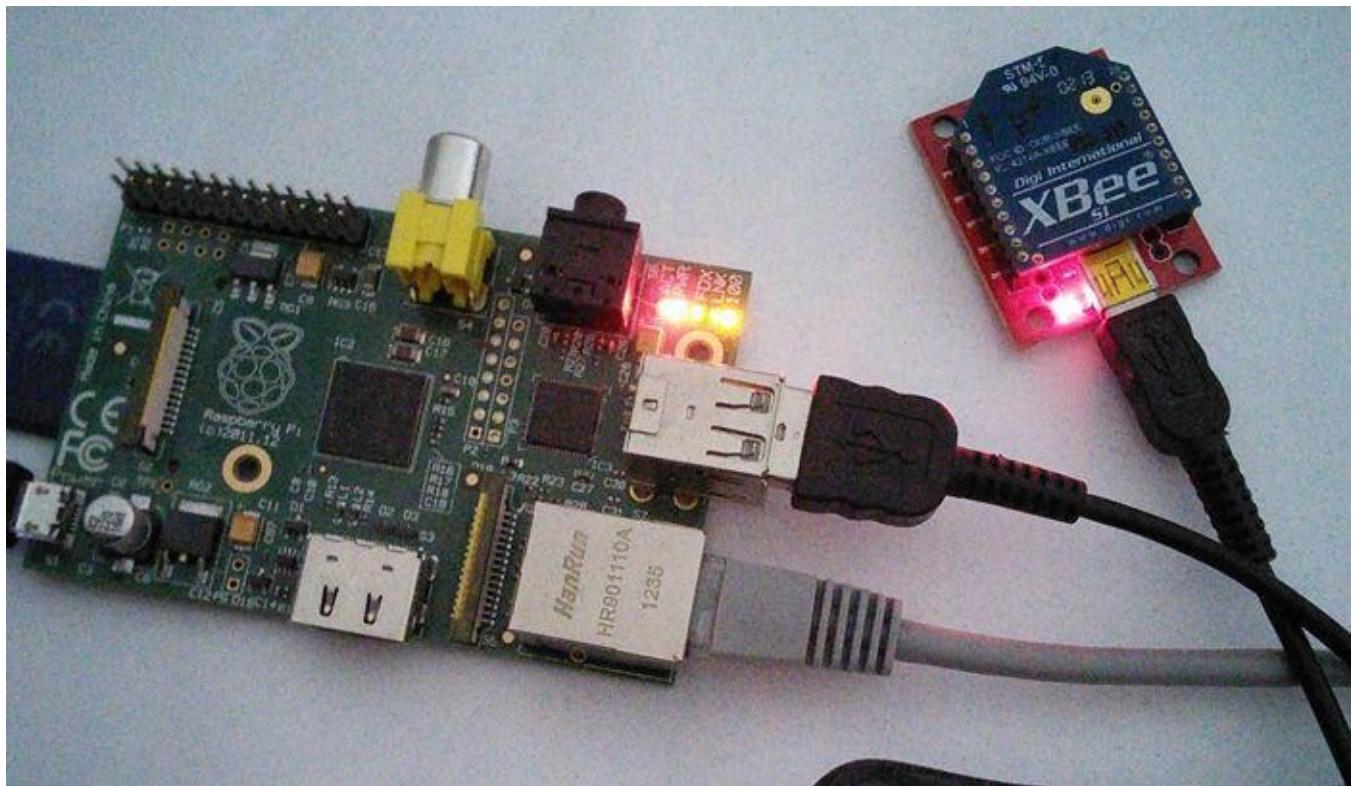
You also can buy these products on the following electronics online store:

- Sparkfun, <https://www.sparkfun.com/> . You can see a list of XBee IEEE 802.15.4 product on <https://www.sparkfun.com/categories/223>
- Amazon, <http://www.amazon.com/> or your local Amazon website
- Cooking-hacks, <http://www.cooking-hacks.com/>
- Watterott electronic, <http://www.watterott.com/index.php?page=manufacturers&mnf=22>
- Exp-tech, [http://www.exp-tech.de/index.php?manu=m35\\_Digi.html](http://www.exp-tech.de/index.php?manu=m35_Digi.html)
- Ebay, <http://www.ebay.com/>

I suggest you to find local electronics store which sells XBee IEEE 802.15.4.

### 7.2.2 Connecting XBee IEEE 802.15.4 to Raspberry Pi

You can connect your XBee to Raspberry Pi through XBee explorer USB. The following is a sample of hardware configuration for XBee and Raspberry Pi.

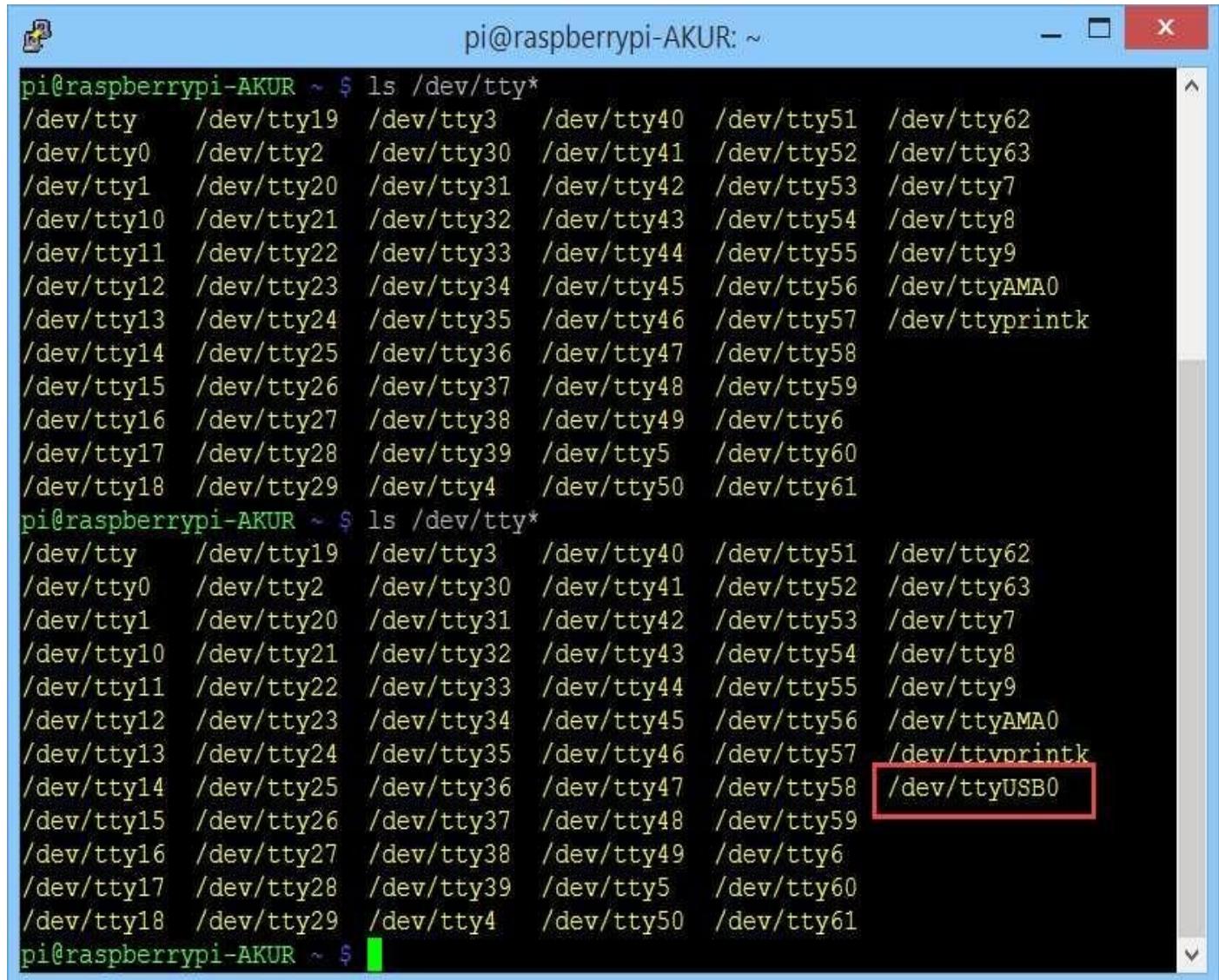


You also can connect XBee to Raspberry Pi directly through UART pins. Make sure you use voltage conversion module for 5V and 3.3V if you have Rx & Tx pins 5V. I have measured that Raspberry Pi UART has 3.3V voltage level so XBee can connect to Raspberry Pi UART pins directly

After you connected your XBee to Raspberry Pi, you can check XBee USB using the following command.

```
$ ls /dev/ttyUSB*
```

The following is a sample output for tty\*.



```
pi@raspberrypi-AKUR: ~ $ ls /dev/tty*
/dev/tty   /dev/tty19  /dev/tty3   /dev/tty40  /dev/tty51  /dev/tty62
/dev/tty0  /dev/tty2   /dev/tty30  /dev/tty41  /dev/tty52  /dev/tty63
/dev/tty1  /dev/tty20  /dev/tty31  /dev/tty42  /dev/tty53  /dev/tty7
/dev/tty10 /dev/tty21  /dev/tty32  /dev/tty43  /dev/tty54  /dev/tty8
/dev/tty11 /dev/tty22  /dev/tty33  /dev/tty44  /dev/tty55  /dev/tty9
/dev/tty12 /dev/tty23  /dev/tty34  /dev/tty45  /dev/tty56  /dev/ttyAMA0
/dev/tty13 /dev/tty24  /dev/tty35  /dev/tty46  /dev/tty57  /dev/ttprintk
/dev/tty14 /dev/tty25  /dev/tty36  /dev/tty47  /dev/tty58
/dev/tty15 /dev/tty26  /dev/tty37  /dev/tty48  /dev/tty59
/dev/tty16 /dev/tty27  /dev/tty38  /dev/tty49  /dev/tty6
/dev/tty17 /dev/tty28  /dev/tty39  /dev/tty5  /dev/tty60
/dev/tty18 /dev/tty29  /dev/tty4  /dev/tty50  /dev/tty61
pi@raspberrypi-AKUR ~ $ ls /dev/tty*
/dev/tty   /dev/tty19  /dev/tty3   /dev/tty40  /dev/tty51  /dev/tty62
/dev/tty0  /dev/tty2   /dev/tty30  /dev/tty41  /dev/tty52  /dev/tty63
/dev/tty1  /dev/tty20  /dev/tty31  /dev/tty42  /dev/tty53  /dev/tty7
/dev/tty10 /dev/tty21  /dev/tty32  /dev/tty43  /dev/tty54  /dev/tty8
/dev/tty11 /dev/tty22  /dev/tty33  /dev/tty44  /dev/tty55  /dev/tty9
/dev/tty12 /dev/tty23  /dev/tty34  /dev/tty45  /dev/tty56  /dev/ttyAMA0
/dev/tty13 /dev/tty24  /dev/tty35  /dev/tty46  /dev/tty57  /dev/ttprintk
/dev/tty14 /dev/tty25  /dev/tty36  /dev/tty47  /dev/tty58
/dev/tty15 /dev/tty26  /dev/tty37  /dev/tty48  /dev/tty59
/dev/tty16 /dev/tty27  /dev/tty38  /dev/tty49  /dev/tty6
/dev/tty17 /dev/tty28  /dev/tty39  /dev/tty5  /dev/tty60
/dev/tty18 /dev/tty29  /dev/tty4  /dev/tty50  /dev/tty61
pi@raspberrypi-AKUR ~ $
```

### 7.2.3 XBee Programming for Raspberry Pi

Basically you do XBee programming as you do on PC. We can use all Python sample codes on previous chapters. Make sure you have installed python-xbee, <https://pypi.python.org/pypi/XBee>. To install python-xbee, you need install pyserial module first. You can download it on <http://pyserial.sourceforge.net/>.

The following is a list of scripting to install python-xbee on Raspberry Pi.

```
$ wget https://pypi.python.org/packages/source/X/XBee/XBee-2.1.0.tar.gz
$ tar -xzf XBee-2.1.0.tar.gz
$ cd XBee-2.1.0
$ sudo python setup.py install
```



pi@raspberrypi-AKUR: ~

```
copying xbee/tests/test_frame.py -> build/lib.linux-armv6l-2.7/xbee/tests
copying xbee/tests/test_ieee.py -> build/lib.linux-armv6l-2.7/xbee/tests
copying xbee/tests/test_zigbee.py -> build/lib.linux-armv6l-2.7/xbee/tests
copying xbee/tests/test_base.py -> build/lib.linux-armv6l-2.7/xbee/tests
copying xbee/tests/test_fake.py -> build/lib.linux-armv6l-2.7/xbee/tests
copying xbee/tests/__init__.py -> build/lib.linux-armv6l-2.7/xbee/tests
copying xbee/tests/Fake.py -> build/lib.linux-armv6l-2.7/xbee/tests
creating build/lib.linux-armv6l-2.7/xbee/helpers
copying xbee/helpers/__init__.py -> build/lib.linux-armv6l-2.7/xbee/helpers
creating build/lib.linux-armv6l-2.7/xbee/helpers/dispatch
copying xbee/helpers/dispatch/dispatch.py -> build/lib.linux-armv6l-2.7/xbee/helpers/dispatch
copying xbee/helpers/dispatch/__init__.py -> build/lib.linux-armv6l-2.7/xbee/helpers/dispatch
creating build/lib.linux-armv6l-2.7/xbee/helpers/dispatch/tests
copying xbee/helpers/dispatch/tests/__init__.py -> build/lib.linux-armv6l-2.7/xbee/helpers/dispatch/tests
copying xbee/helpers/dispatch/tests/test_dispatch.py -> build/lib.linux-armv6l-2.7/xbee/helpers/dispatch/tests
copying xbee/helpers/dispatch/tests/fake.py -> build/lib.linux-armv6l-2.7/xbee/helpers/dispatch/tests
running install_lib
running install_egg_info
Removing /usr/local/lib/python2.7/dist-packages/XBee-2.1.0.egg-info
Writing /usr/local/lib/python2.7/dist-packages/XBee-2.1.0.egg-info
pi@raspberrypi-AKUR ~/xbee/XBee-2.1.0 $ cd ~/
pi@raspberrypi-AKUR ~ $
```

## 7.3 Demo: Raspberry Pi and PC Communication Through XBee

To illustrate how to communicate between Raspberry Pi and PC through XBee, we use the same code from chapter 7, **two\_way\_comm\_xbee1.py** and **two\_way\_comm\_xbee2.py** files. We use the following items:

- Two XBee devices
- Two XBee explorer USB

### 7.3.1 XBee Configuration

We configure AT mode = 2 to both XBee. The following is XBee 1 configuration on X-CTU.

```
+++  
ATID 3001 [ENTER]  
ATMY 1 [ENTER]  
ATAP 2 [ENTER]  
ATWR [ENTER]  
ATCN [ENTER]
```

The following is XBee 2 configuration on X-CTU.

```
+++  
ATID 3001 [ENTER]  
ATMY 2 [ENTER]  
ATAP 2 [ENTER]  
ATWR [ENTER]  
ATCN [ENTER]
```

After you wrote these configuration, you attach your XBees to PC and Raspberry Pi.

### 7.3.2 Writing Program for Raspberry Pi and PC

As I explained, we use the same file on chapter 7, **two\_way\_comm\_xbee1.py** and **two\_way\_comm\_xbee2.py** files. Copy **two\_way\_comm\_xbee2.py** to Raspberry Pi and change serial port. The following is a sample code on Raspberry Pi

The screenshot shows a terminal window titled "pi@raspberrypi-AKUR: ~/ftp/files". The window contains a Python script named "two way comm xbee2.py" using the "GNU nano 2.2.6" editor. The script imports time, xbee, and serial modules, and defines constants for PORT ('/dev/ttyUSB0') and BAUD RATE (9600). It includes a function to convert byte strings to hex and another to decode received XBee frames. The terminal also displays a set of keyboard shortcuts at the bottom.

```
pi@raspberrypi-AKUR: ~/ftp/files
GNU nano 2.2.6          File: two way comm xbee2.py          Modified
import time
from xbee import XBee
import serial

PORT = '/dev/ttyUSB0'
BAUD_RATE = 9600

# source
# http://code.activestate.com/recipes/510399-byte-to-hex-and-hex-to-byte-string$
def ByteToHex(byteStr):
    return ''.join(["%02X" % ord(x) for x in byteStr]).strip()

def decodeReceivedFrame(data):
    source_addr = ByteToHex(data['source_addr'])
    xbee_id = data['id']
    rssi = ord(data['rssi'])
    rf_data = data['rf_data']
    options = ByteToHex(data['options'])
    return [source_addr, xbee_id, rssi, rf_data, options]

^G Get Help  ^O WriteOut  ^R Read File  ^Y Prev Page  ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is   ^V Next Page  ^U UnCut Text ^T To Spell
```

### 7.3.3 Testing

Now you can test. Firstly, run a program on Raspberry Pi. Then, you run program on PC. The following is a sample output on PC.

```
Run two_way_comm_xbee1
send data
['0003', 'rx', 48, 'Hello XBee 1', '00']
send data
['0003', 'rx', 48, 'Hello XBee 1', '00']
send data
['0003', 'rx', 47, 'Hello XBee 1', '00']
send data
['0003', 'rx', 47, 'Hello XBee 1', '00']
send data
['0003', 'rx', 47, 'Hello XBee 1', '00']
send data
['0003', 'rx', 47, 'Hello XBee 1', '00']

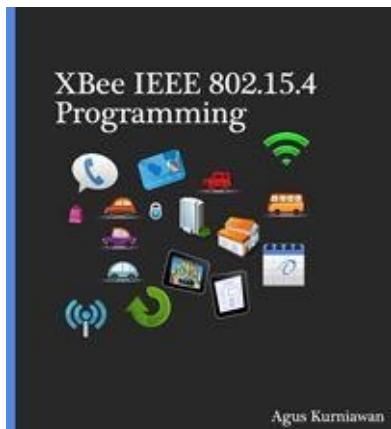
4: Run 6: TODO
```

The following is a sample output on Raspberry Pi.

```
pi@raspberrypi-AKUR: ~/ftp/files
'default': '\xff\xfe', 'len': 2, 'name': 'dest_addr'},
{'default': '\x02', 'len': 1, 'name': 'options'},
{'default': None, 'len': 2, 'name': 'command'},
{'default': None, 'len': None, 'name': 'parameter']},
'tx': [{'default': '\x01', 'len': 1, 'name': 'id'},
 {'default': '\x00', 'len': 1, 'name': 'frame id'},
 {'default': None, 'len': 2, 'name': 'dest_addr'},
 {'default': '\x00', 'len': 1, 'name': 'options'},
 {'default': None, 'len': None, 'name': 'data'}],
'tx_long_addr': [{"default": "\x00", "len": 1, "name": "id"}, {"default": "\x00", "len": 1, "name": "frame id"}, {"default": None, "len": 8, "name": "dest_addr"}, {"default": "\x00", "len": 1, "name": "options"}, {"default": None, "len": None, "name": "data"}]}
send data
['0001', 'rx', 47, 'Hello XBee 2', '00']
send data
['0001', 'rx', 47, 'Hello XBee 2', '00']
send data
['0001', 'rx', 47, 'Hello XBee 2', '00']
send data
['0001', 'rx', 46, 'Hello XBee 2', '00']
send data
['0001', 'rx', 46, 'Hello XBee 2', '00']
send data
['0001', 'rx', 47, 'Hello XBee 2', '00']
```

## 7.4 Further Reading

I write a book about XBee IEEE 802.15.4 Programming. This book helps you how to get started with IEEE 802.15.4 programming through XBee device. All sample codes were written in Python to illustrate how to work with IEEE 802.15.4. If you're interest with this book, you can visit on my blog, <http://blog.aguskurniawan.net/post/XBee-IEEE-802154-Programming.aspx>.



## **8. RFID and NFC Communication**

This chapter explains how to build a communication with RFID and NFC modules on Raspberry Pi.

## 8.1 Getting Started

NFC stands for near field communication, while RFID means radio frequency identification. Both employ radio signals for all sorts of tagging and tracking purposes, sometimes replacing bar codes. NFC is still an emerging technology; RFID, however, is currently in widespread use all over the world. You can read what' the difference between RFID and NFC on this site, <http://blog.atlasrfidstore.com/near-field-communication-infographic/> .

In this chapter, we explorer how to work with RFID/NFC modules on Raspberry Pi. There are two lab scenarios we will build:

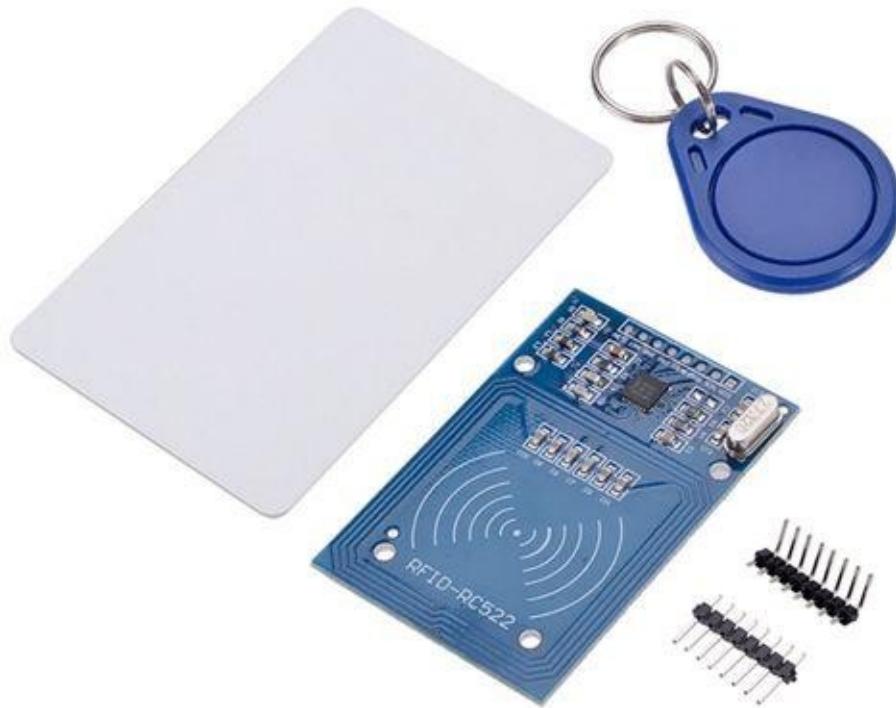
- Reading RFID Card/Tag
- Reading memory Data via NFC

## 8.2 Getting Hardware

There are many options to choose RFID/NFC modules for Raspberry Pi. The following is a sample of RFID/NFC modules.

RFID reader (MFRC-522) module is used in highly integrated 13.56MHz contactless communication card chip to read and write. The MF RC522 use of advanced modulation and demodulation concept completely integrated in all communication methods and protocols at 13.56MHz. I obtained RFID-RC522 from eBay, [http://www.ebay.com/sch/i.html?from=R40&\\_trksid=p2050601.m570.l1313.TR0.TRC0.H0.XMFRC522&\\_nkw=MFRC522](http://www.ebay.com/sch/i.html?from=R40&_trksid=p2050601.m570.l1313.TR0.TRC0.H0.XMFRC522&_nkw=MFRC522).

The price is cheap. You can see this module in Figure below.



ITEAD PN532 NFC module, as its name implies, is based on PN532 chip and used for 13.56MHz near field communication. The module is equipped with onboard antenna, thus no external antenna coil is needed. It is compatible with SPI, IIC and UART interface for communication. You can find and buy this hardware on <http://imall.iteadstudio.com/im130625002.html>.



Another model can be found on eBay, <http://www.ebay.com/sch/i.html?from=R40&trksid=p2047675.m570.l1313.TR0.TRC0.H0.XPN532+NFC+RFID&nkw=>



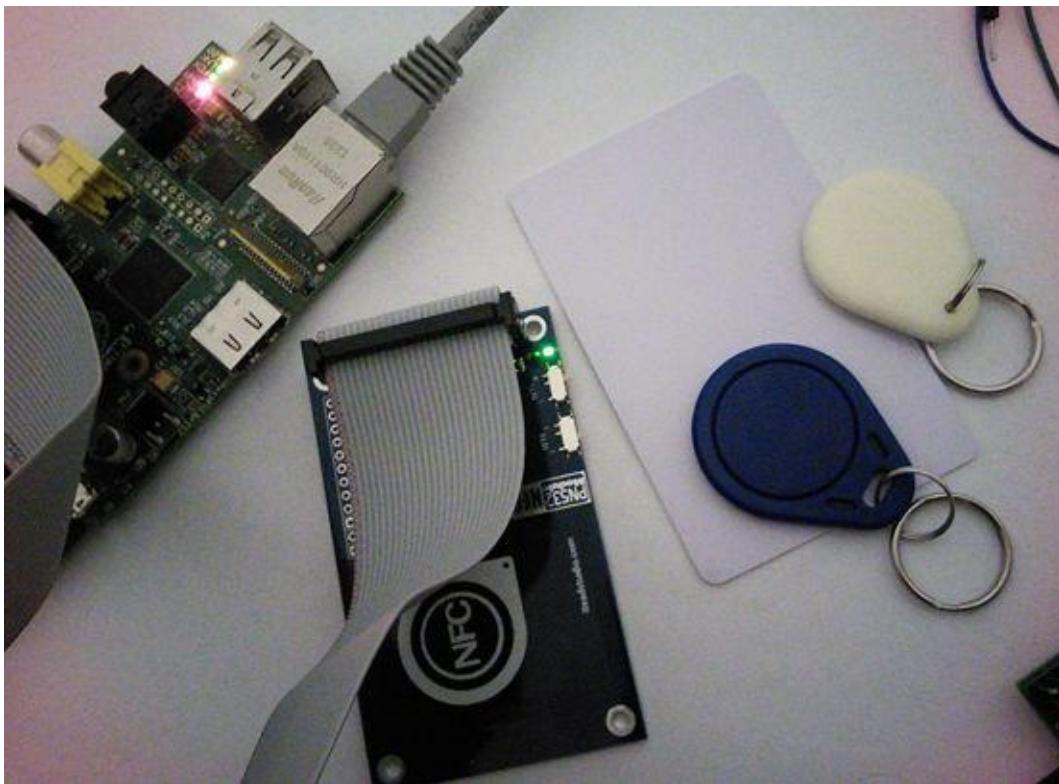
You can buy RFID/NFC modules on the following store:

- Seeedstudio, [http://www.seeedstudio.com/depot/s/nfc.html?search\\_in\\_description=0](http://www.seeedstudio.com/depot/s/nfc.html?search_in_description=0)
- Exp-tech, <http://www.exp-tech.de/modules/nfc-modules>
- Sparkfun, <https://www.sparkfun.com/search/results?term=nfc>
- Amazon.com
- eBay.com

## 8.3 Reading RFID Card/Key

In this section, we build a simple app to read RFID card/key. For implementation, I use ITEAD PN532 NFC module for RFID reader module. You plug ITEAD PN532 NFC to Raspberry Pi via GPIO cable directly. Read the instruction on [http://wiki.iteadstudio.com/ITEAD\\_PN532\\_NFC\\_MODULE](http://wiki.iteadstudio.com/ITEAD_PN532_NFC_MODULE).

The following is hardware implementation.



Firstly, you configure SPI on Raspberry Pi. Please read section 6.4 to enable SPI on Raspberry Pi. Then, you can download ITEAD PN532 NFC library for Raspberry Pi, <http://blog.iteadstudio.com/to-drive-itead-pn532-nfc-module-with-raspberry-pi/> or you can download source code of this book on **itead\_pn532\_nfc** folder.

After that, you can install this library.

```
$ make install
```

It will generate a library, NFC.

Now you can create a file, called **rfid\_reader.c**, and write this code.

```
#include "nfc.h"

#define PN532DEBUG 0

void main(void)
{
    uint32_t versiondata;
    uint32_t id;
```

```

begin();
versiondata = getFirmwareVersion();
if (! versiondata)
{
    printf("Didn't find PN53x board\n");
    return;
}
printf("Information about PN53x module");
printf("%x\n",((versiondata>>24) & 0xFF));
printf("Firmware ver. ");
printf("%d",((versiondata>>16) & 0xFF));
printf(".");
printf("%d\n",((versiondata>>8) & 0xFF));
printf("Supports ");
printf("%x\n", (versiondata & 0xFF));

// configure board to read RFID tags and cards
SAMConfig();
printf("Waiting RFID card....\n");
while(1)
{
    // look for MiFare type cards
    id = readPassiveTargetID(PN532_MIFARE_IS014443A);
    if (id != 0)
    {
        printf("Read card #%-d\n", id);
    }
}
}

```

Save this code. You can compile this file using this command.

```
$ gcc rfid_reader.c -o rfid_reader -l NFC
```

You can run it by typing this command.

```
$ sudo ./rfid_reader
```

After that, you can tap your RFID card/key. A sample output for rfid\_reader application can be seen in Figure below.



pi@raspberrypi: ~/book/itead\_pn532\_nfc

```
pi@raspberrypi ~/book/itead_pn532_nfc $ sudo ./rfid_reader
the fd is 4
ioctl LSB set is 0
wiringPiSetup is 0
Information about PN53x module32
Firmware ver. 1.6
Supports 7
Waiting RFID card....
Read card #-326496505
Read card #-2098006125
```

## 8.4 Reading NFC Memory

In this section, we try to read NFC card/key memory. We still use ITEAD PN532 NFC module. To read data on NFC, we usually need a key. By default, key value is {0xFF,0xFF,0xFF,0xFF,0xFF,0xFF}.

Now you can create a file, called **rfid\_memory\_reader.c** , and write this code.

```
#include "nfc.h"

#define      PN532_CS      10
#define      NFC_DEMO_DEBUG  0
#define      PN532DEBUG 0

void main(void)
{
    uint32_t versiondata;
    uint32_t id;
    // default card key. you can change it
    uint8_t keys[10]={0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF};
    uint8_t blockn;
    uint8_t i;

    begin();
    versiondata = getFirmwareVersion();
    if (!versiondata)
    {
        printf("Didn't find PN53x board\n");
        return;
    }

    printf("Information about PN53x module");
    printf("%x\n",((versiondata>>24) & 0xFF));
    printf("Firmware ver. ");
    printf("%d",((versiondata>>16) & 0xFF));
    printf(".");
    printf("%d\n",((versiondata>>8) & 0xFF));
    printf("Supports ");
    printf("%x\n", (versiondata & 0xFF));

    // configure board to read RFID tags and cards
    SAMConfig();
    printf("Waiting RFID card....\n");

    while(1)
    {
        id = readPassiveTargetID(PN532_MIFARE_IS014443A);
        if (id != 0)
        {
            printf("Read card # %d\n",id);
            for(blockn=0;blockn<64;blockn++)
            {
                if(authenticateBlock(1, id ,blockn, KEY_A,keys)) //authen
                {
                    //if authentication successful
                }
            }
        }
    }
}
```

```

        uint8_t block[16];
        //read memory block blockn
        if(readMemoryBlock(1,blockn,block))
        {
            //if read operation is successful
            for(i=0;i<16;i++)
            {
                printf("%x ",block[i]);
            }
            printf("\n");
            for(i=0;i<16;i++)
            {
                printf("%c ",block[i]);
            }

            printf(" | Block \n");

            printf("%d\n",blockn);
            printf(" | ");

            if(blockn == 0)
            {
                printf("Manufacturer Block");
            }
            else
            {
                if(((blockn + 1) % 4) == 0)
                {
                    printf("Sector Trailer");
                }
                else
                {
                    printf("Data Block");
                }
            }
            printf("\n");
        }

    }
}
delay(2000);

}
}

```

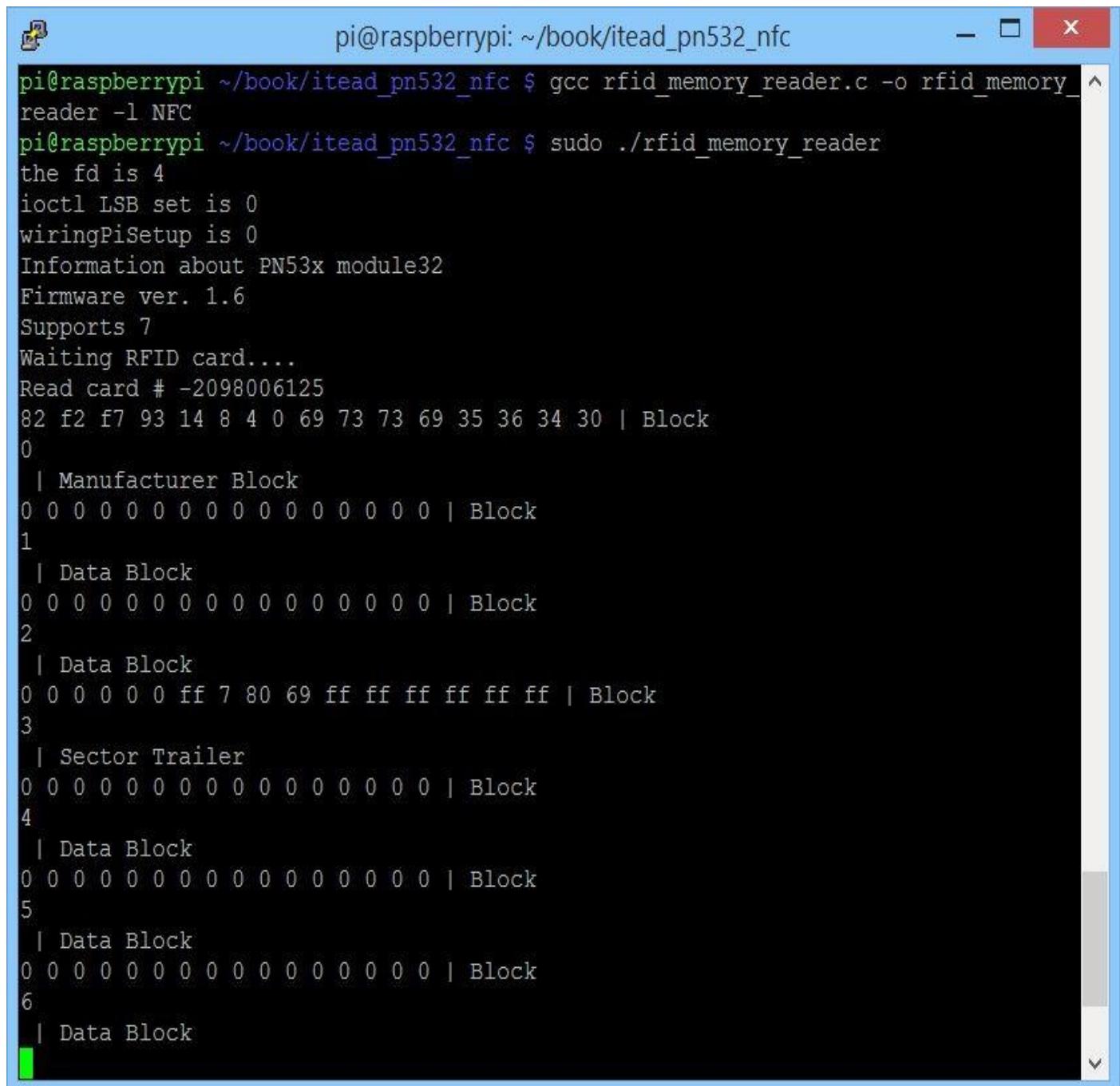
Save this code and try to compile.

```
$ gcc rfid_memory_reader.c -o rfid_memory_reader -l NFC
```

If success, you can run it.

```
$ sudo ./rfid_memory_reader
```

Tap your RFID card/key to the module. The program will read RFID memory. A sample output of program can be seen in Figure below.



The screenshot shows a terminal window titled "pi@raspberrypi: ~/book/itead\_pn532\_nfc". The window displays the execution of a C program named "rfid\_memory\_reader" which reads an RFID card. The output shows the card's identifier (-2098006125) and its memory sectors. Sector 0 contains a manufacturer block (0000000000000000). Sectors 1 and 2 contain data blocks (0000000000000000). Sector 3 is a sector trailer (0000000000000000). Sectors 4 through 6 also contain data blocks (0000000000000000).

```
pi@raspberrypi: ~/book/itead_pn532_nfc $ gcc rfid_memory_reader.c -o rfid_memory_reader -l NFC
pi@raspberrypi: ~/book/itead_pn532_nfc $ sudo ./rfid_memory_reader
the fd is 4
ioctl LSB set is 0
wiringPiSetup is 0
Information about PN53x module32
Firmware ver. 1.6
Supports 7
Waiting RFID card....
Read card # -2098006125
82 f2 f7 93 14 8 4 0 69 73 73 69 35 36 34 30 | Block
0
| Manufacturer Block
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Block
1
| Data Block
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Block
2
| Data Block
0 0 0 0 0 ff 7 80 69 ff ff ff ff ff | Block
3
| Sector Trailer
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Block
4
| Data Block
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Block
5
| Data Block
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Block
6
| Data Block
```

## **9. FM Radio Receiver**

This chapter explains how to build FM Radio receiver on Raspberry Pi.

## **9.1 Getting Started**

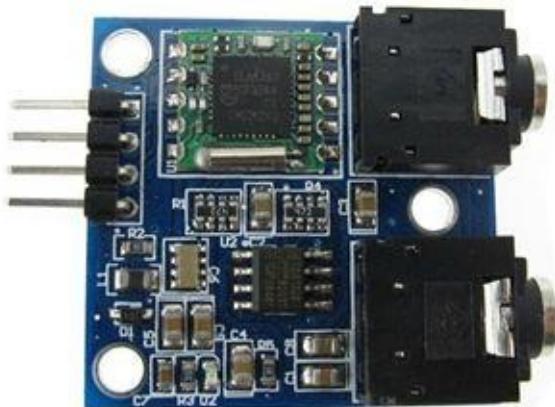
We build FM radio receiver on Raspberry Pi. For implementation, I use TEA5767 module.

## 9.2 Getting Hardware

To get TEA5767 module, you can get it on the following online store:

- eBay, <http://www.ebay.com/>
- Amazon.com

You also can find it on local electronics store. I obtained this module from eBay about US\$ 6.28.



## 9.3 Wiring

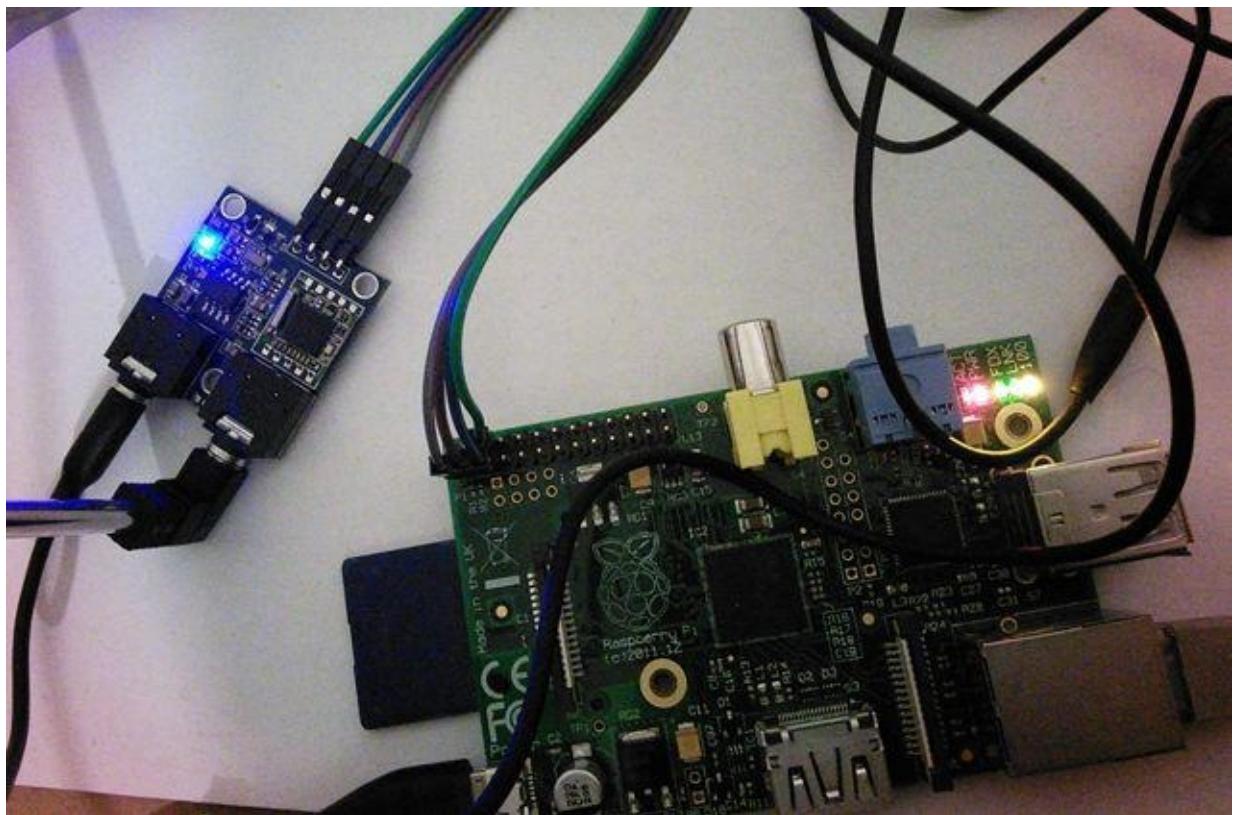
TEA5767 module uses I2C protocol. You can see the module pinout.



The following is our wiring:

- VCC to +3.3V Raspberry Pi
- SDA to SDA Raspberry Pi
- SCL to SCL Raspberry Pi
- GND to GND Raspberry Pi

A sample of hardware implementation can be seen in Figure below.

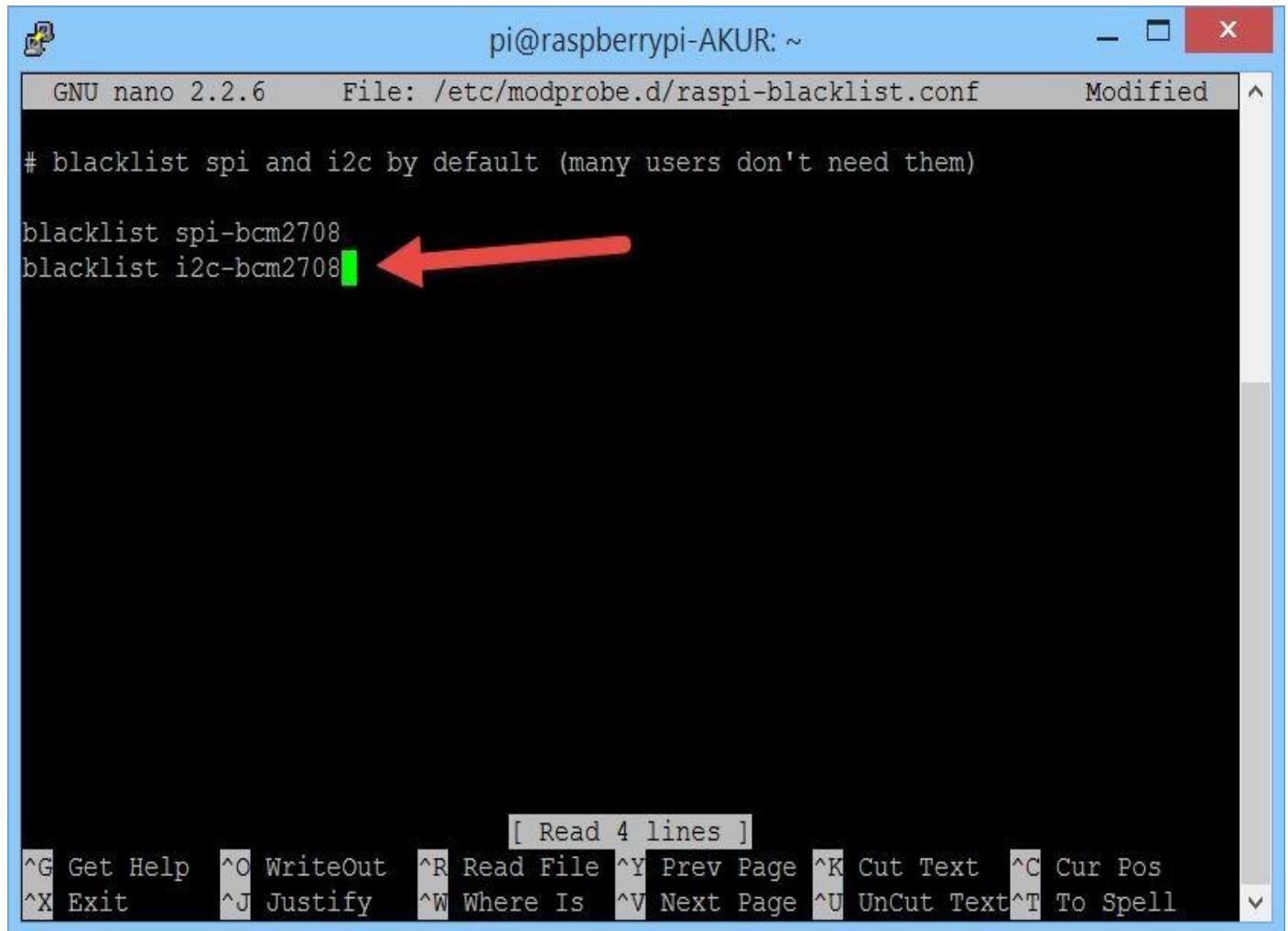


## 9.4 Configuring I2C for Raspberry Pi

To access I2C, we must configure Raspberry Pi to enable this communication. Firstly, we remove I2C from module blacklist. Open file **/etc/modprobe.d/raspi-blacklist.conf**, for instance I use nano editor.

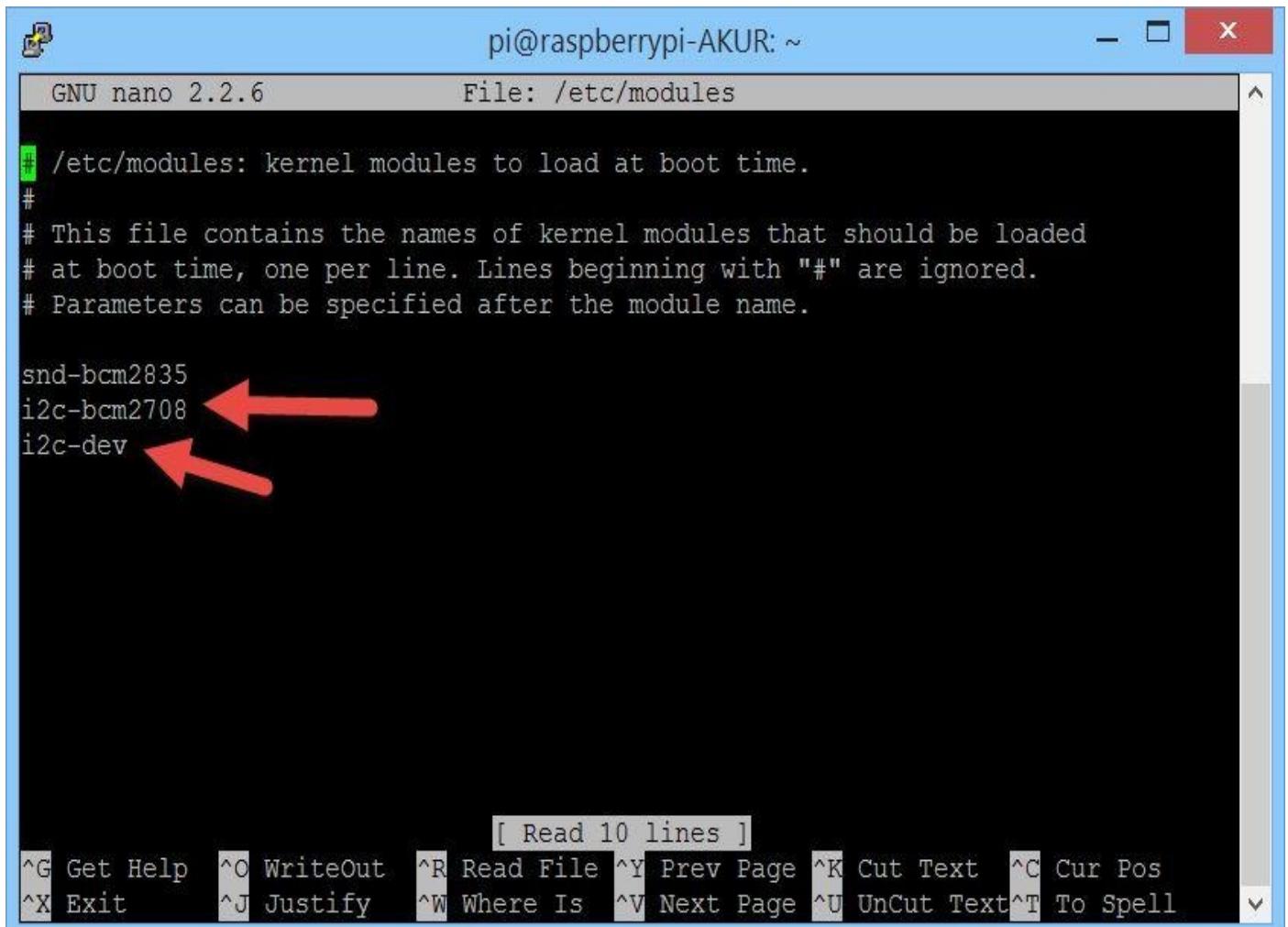
```
$ sudo nano /etc/modprobe.d/raspi-blacklist.conf
```

Then add "#" on **blacklist i2c-bcm2708**.



```
pi@raspberrypi-AKUR: ~
GNU nano 2.2.6      File: /etc/modprobe.d/raspi-blacklist.conf      Modified
^A ^B ^D ^F ^L ^P ^R ^T ^W ^X ^Y ^Z ^G Get Help   ^O WriteOut   ^R Read File   ^Y Prev Page   ^K Cut Text   ^C Cur Pos
^X Exit      ^J Justify    ^W Where Is    ^V Next Page   ^U UnCut Text ^T To Spell
# blacklist spi and i2c by default (many users don't need them)
blacklist spi-bcm2708
blacklist i2c-bcm2708
```

The final configuration should be as follows.



```
pi@raspberrypi-AKUR: ~
GNU nano 2.2.6          File: /etc/modules

# /etc/modules: kernel modules to load at boot time.
#
# This file contains the names of kernel modules that should be loaded
# at boot time, one per line. Lines beginning with "#" are ignored.
# Parameters can be specified after the module name.

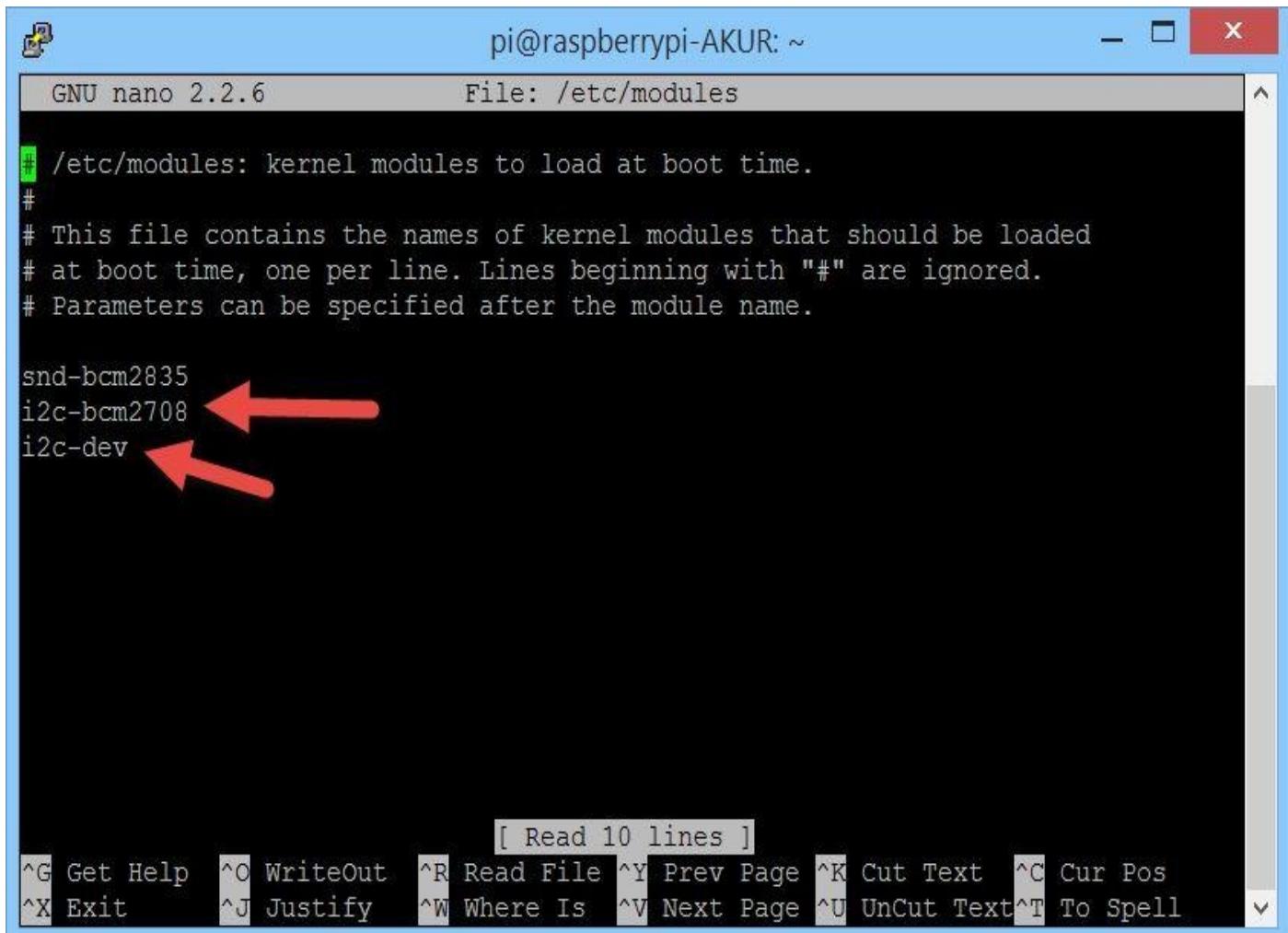
snd-bcm2835
i2c-bcm2708
i2c-dev
```

Save this file and close it.

Next step, we add **i2c-dev** to the **/etc/modules** file so it is loaded on boot.

```
$ sudo nano /etc/modules
```

Add **i2c-bcm2708** and **i2c-dev** by typing these modules.



```
pi@raspberrypi-AKUR: ~
GNU nano 2.2.6          File: /etc/modules
/etc/modules: kernel modules to load at boot time.
#
# This file contains the names of kernel modules that should be loaded
# at boot time, one per line. Lines beginning with "#" are ignored.
# Parameters can be specified after the module name.

snd-bcm2835
i2c-bcm2708
i2c-dev
```

Save this file.

Now we install **i2c-tools** and add user pi to member **i2c**. If you use another account, please add it too.

```
$ sudo apt-get install i2c-tools
$ sudo adduser pi i2c
```

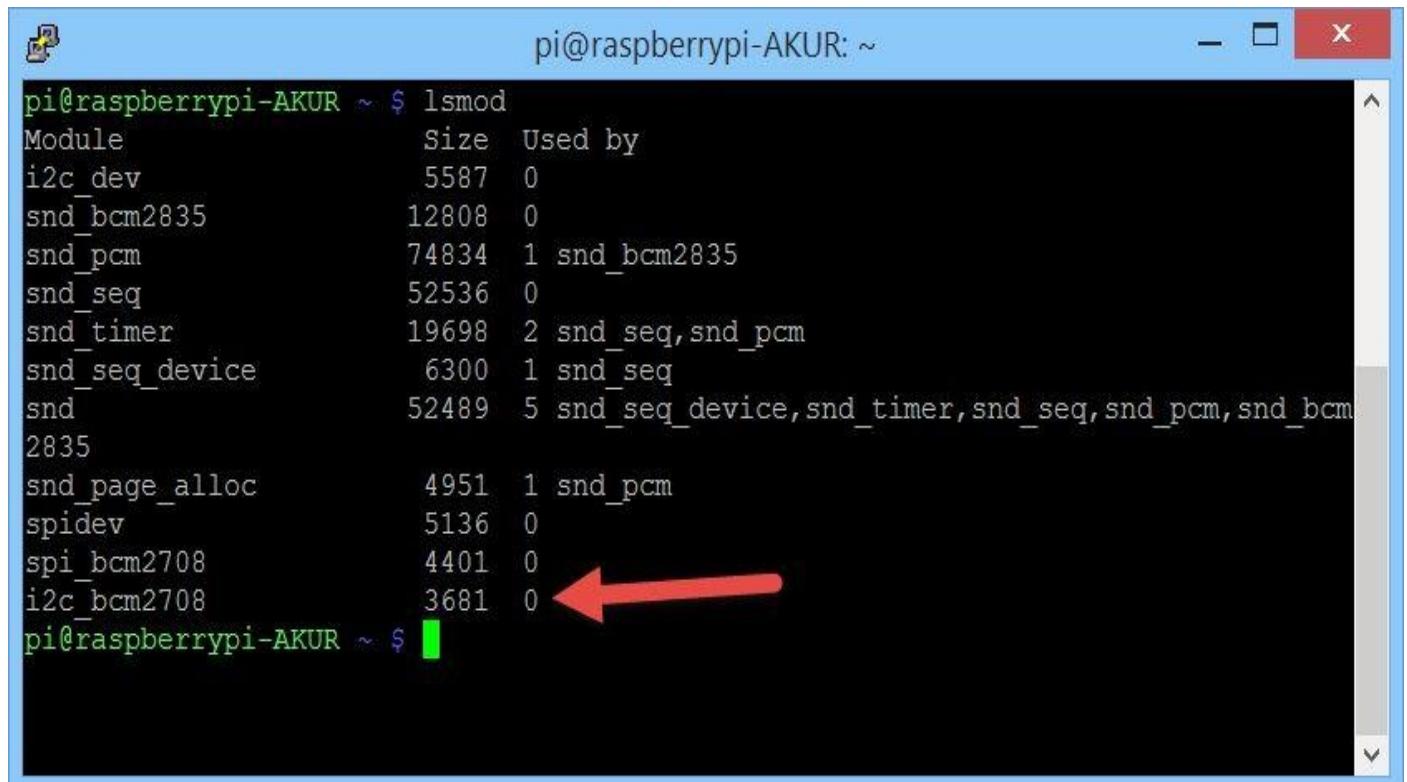
If done, you must reboot your Raspberry Pi by typing the following command.

```
$ sudo reboot
```

After that, you can verify if `i2c_bcm2708` is not be blocked by typing the following command.

```
$ lsmod
```

The following is response output.



```
pi@raspberrypi-AKUR: ~
pi@raspberrypi-AKUR ~ $ lsmod
Module           Size  Used by
i2c_dev          5587  0
snd_bcm2835     12808  0
snd_pcm          74834  1 snd_bcm2835
snd_seq          52536  0
snd_timer        19698  2 snd_seq,snd_pcm
snd_seq_device   6300   1 snd_seq
snd              52489  5 snd_seq_device,snd_timer,snd_seq,snd_pcm,snd_bcm
2835
snd_page_alloc   4951   1 snd_pcm
spidev           5136   0
spi_bcm2708     4401   0
i2c_bcm2708     3681   0
pi@raspberrypi-AKUR ~ $
```

A red arrow points to the line "i2c\_bcm2708 3681 0" in the terminal output.

You can see i2c\_bcm2708 on the output.

## 9.5 Writing Program

After connected your module to Raspberry Pi, you can verify by typing this command

```
$ i2cdetect -y 1
```

If success, the module will show on address 0x60, shown in Figure below.

```
pi@raspberrypi02 ~ $ i2cdetect -y 1
 0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00: --
10: --
20: --
30: --
40: --
50: --
60: - -
70: --
pi@raspberrypi02 ~ $
```

To testing the module, we can write program to play FM radio with specific frequency. This code is taken from <http://www.raspberrypi.org/forums/viewtopic.php?t=53680&p=665022>.

Create a file, called radio.c, and write this code.

```
#include <wiringPi.h>
#include <wiringPiI2C.h>
#include <stdio.h>
#include <stdlib.h>

int main( int argc, char *argv[] )
{
    printf("RPi - tea5767 Philips FM Tuner v0.3 \n");
    if(argc<2)
    {
        printf("Usage: ./radio frequency\n");
        return 0;
    }

    unsigned char radio[5] = {0};
    int fd;
    int dID = 0x60; // i2c Channel the device is on

    unsigned char frequencyH = 0;
    unsigned char frequencyL = 0;

    unsigned int frequencyB;
```

```

double frequency = strtod(argv[1], NULL);
frequencyB=4*

(frequency*1000000+225000)/32768; //calculating PLL word
    frequencyH=frequencyB>>8;
    frequencyL=frequencyB&0XFF;

printf ("Frequency = ");
printf("%f", frequency);
printf("\n");

// data to be sent
radio[0]=frequencyH; //FREQUENCY H
radio[1]=frequencyL; //FREQUENCY L
radio[2]=0xB0; //3 byte (0xB0): high side L0 injection is on, .
radio[3]=0x10; //4 byte (0x10) : Xtal is 32.768 kHz
radio[4]=0x00; //5 byte0x00)

if((fd=wiringPiI2CSetup(dID))<0)
{
    printf("error opening i2c channel\n\r");
}

write (fd, (unsigned int)radio, 5) ;

return 0;
}

```

Save this code.

Now you can compile this code by typing this command.

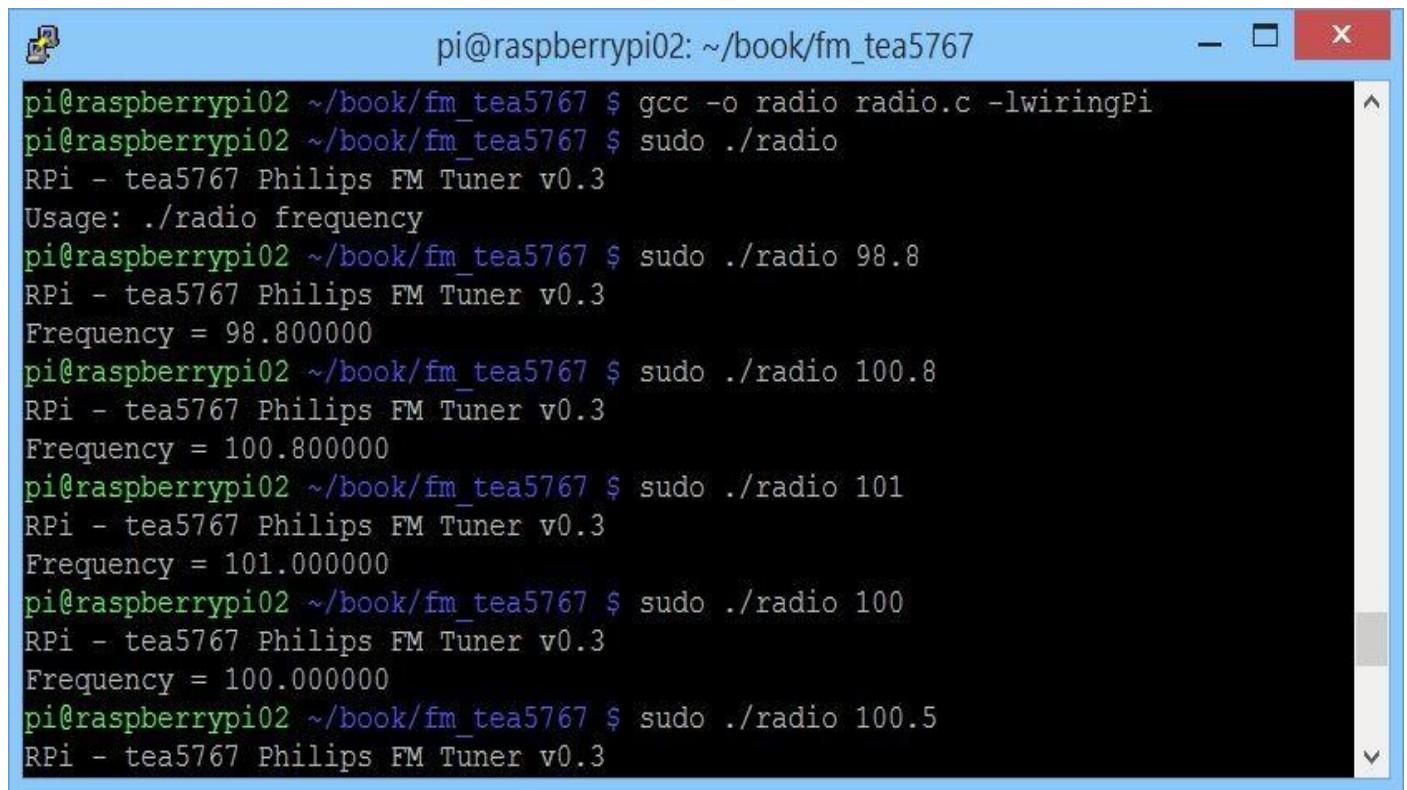
```
$ gcc -o radio radio.c -lwiringPi
```

## 9.5 Testing

To run the program, you can pass frequency value to the program.

```
$ sudo ./radio <frequency>
```

A sample output can be seen in Figure below.



The screenshot shows a terminal window titled "pi@raspberrypi02: ~/book/fm\_tea5767". The window contains the following text:

```
pi@raspberrypi02 ~/book/fm_tea5767 $ gcc -o radio radio.c -lwiringPi
pi@raspberrypi02 ~/book/fm_tea5767 $ sudo ./radio
RPi - tea5767 Philips FM Tuner v0.3
Usage: ./radio frequency
pi@raspberrypi02 ~/book/fm_tea5767 $ sudo ./radio 98.8
RPi - tea5767 Philips FM Tuner v0.3
Frequency = 98.800000
pi@raspberrypi02 ~/book/fm_tea5767 $ sudo ./radio 100.8
RPi - tea5767 Philips FM Tuner v0.3
Frequency = 100.800000
pi@raspberrypi02 ~/book/fm_tea5767 $ sudo ./radio 101
RPi - tea5767 Philips FM Tuner v0.3
Frequency = 101.000000
pi@raspberrypi02 ~/book/fm_tea5767 $ sudo ./radio 100
RPi - tea5767 Philips FM Tuner v0.3
Frequency = 100.000000
pi@raspberrypi02 ~/book/fm_tea5767 $ sudo ./radio 100.5
RPi - tea5767 Philips FM Tuner v0.3
```

Plug-in earphone to the module. Change the frequency value based your local FM radio.

## **Source Code**

Source code can be downloaded on <http://www.aguskurniawan.net/book/piwi.zip>.

## Contact

If you have question related to this book, please contact me at aguskur@hotmail.com . My blog: <http://blog.aguskurniawan.net>.