Raspberry Pi
Wireless Networks

Agus Kurniawan

Raspberry Pi
Wireless Networks

Agus Kurniawan

Copyright

Table of Contents

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:

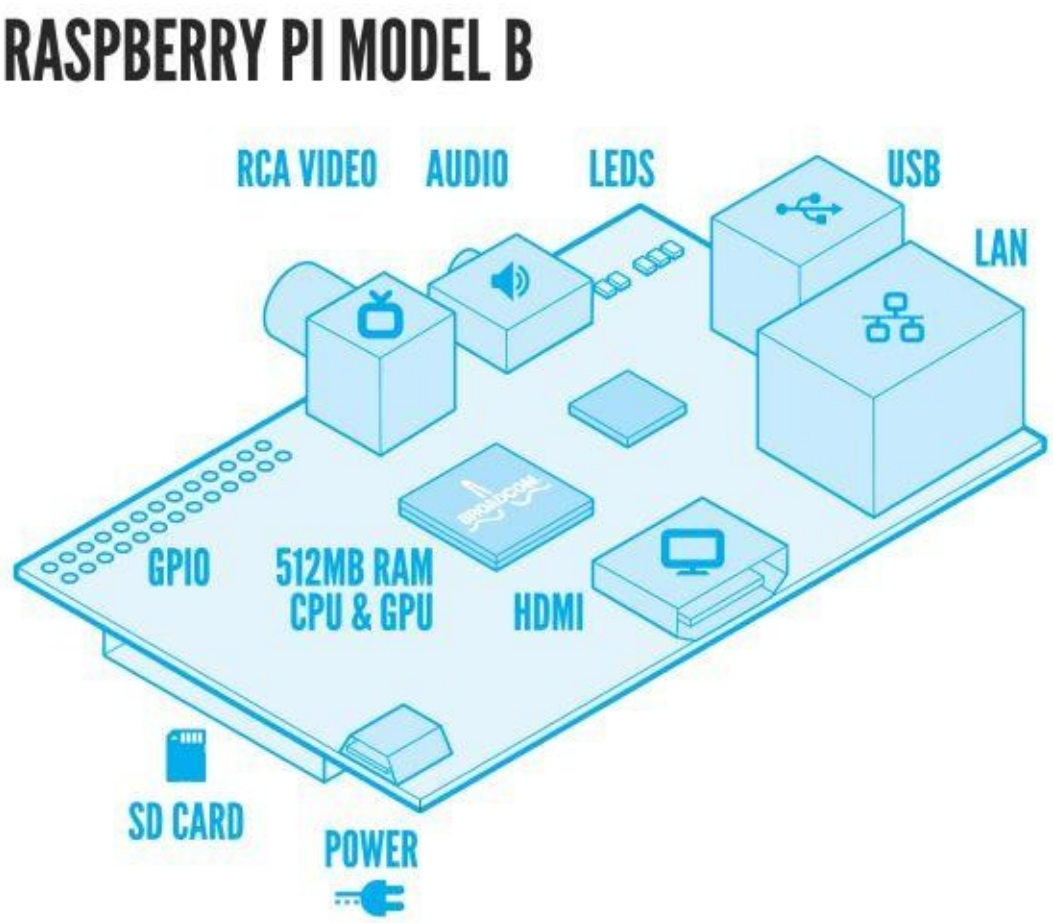
- Model A with 256 Mb RAM
- Model A+ with 256 Mb RAM
- Model B with 512 Mb RAM
- Model B+ with 512MB RAM
- Raspberry Pi Compute Development

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

The following is technical specification of Raspberry Pi device:

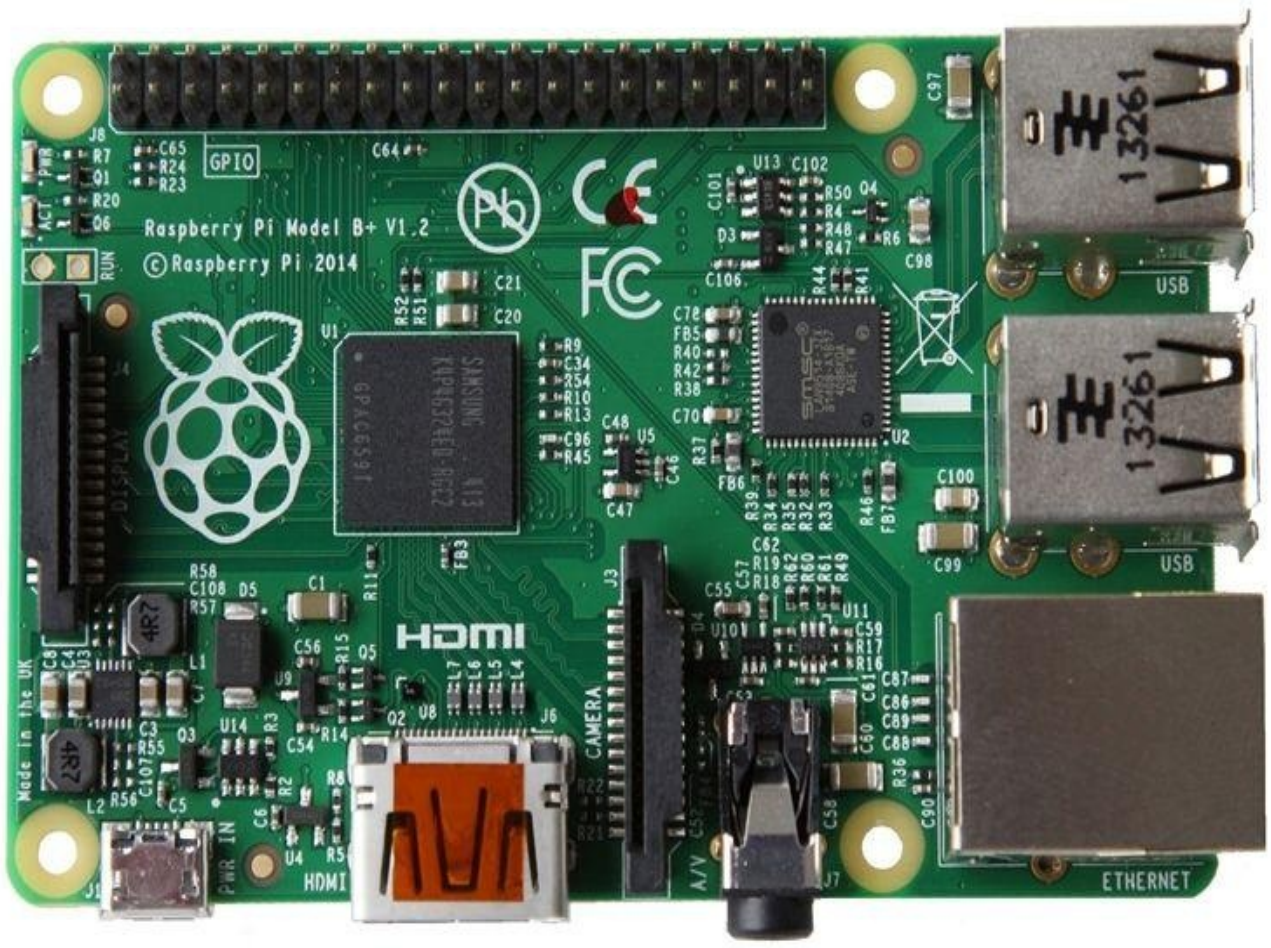
- SoC Broadcom BCM2835 (CPU, GPU, DSP, SDRAM)
- CPU: 700 MHz ARM1176JZF-S core (ARM11 family)
- GPU: Broadcom VideoCore IV, OpenGL ES 2.0, 1080p30 h.264/MPEG-4 AVC high-profile decoder
- Memory (SDRAM): 512 Megabytes (MiB)
- Video outputs: Composite RCA, HDMI
- Audio outputs: 3.5 mm jack, HDMI
- Onboard 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.



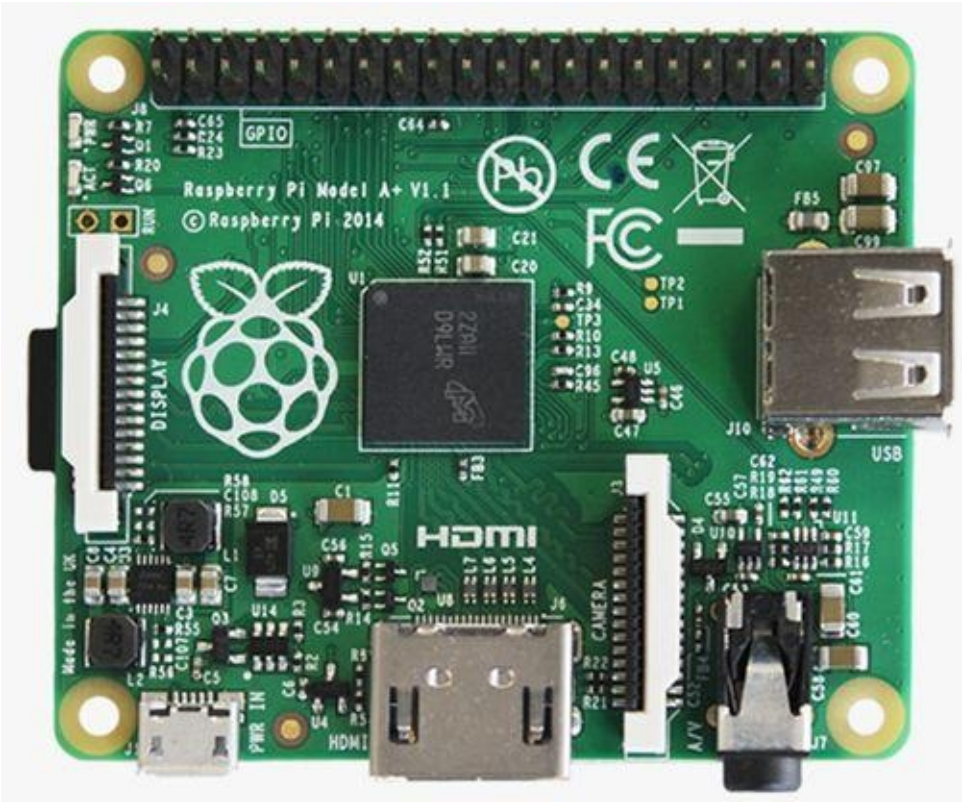
(source: http://www.raspberrypi.org/wp-content/uploads/2011/07/RaspiModelB.png)

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



(source: http://www.exp-tech.de/Mainboards/RPi/raspberry-pi-b-plus.html)

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



(source: http://www.raspberrypi.org/raspberry-pi-model-a-plus-on-sale/)

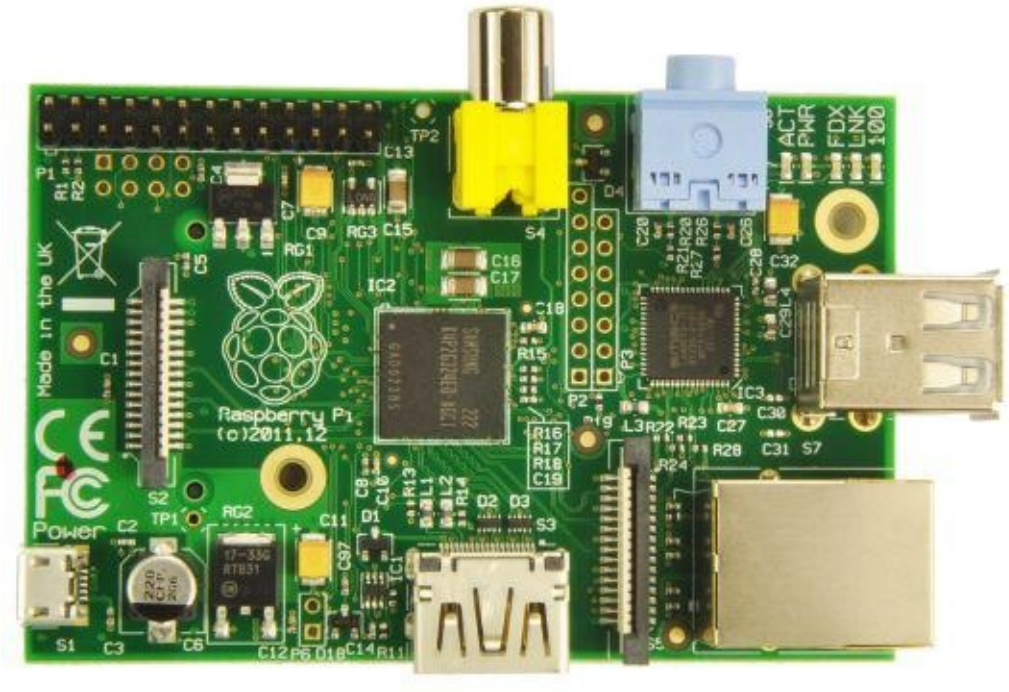
For further information, you can visit on Raspberry Pi website, http://www.raspberrypi.org.

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.

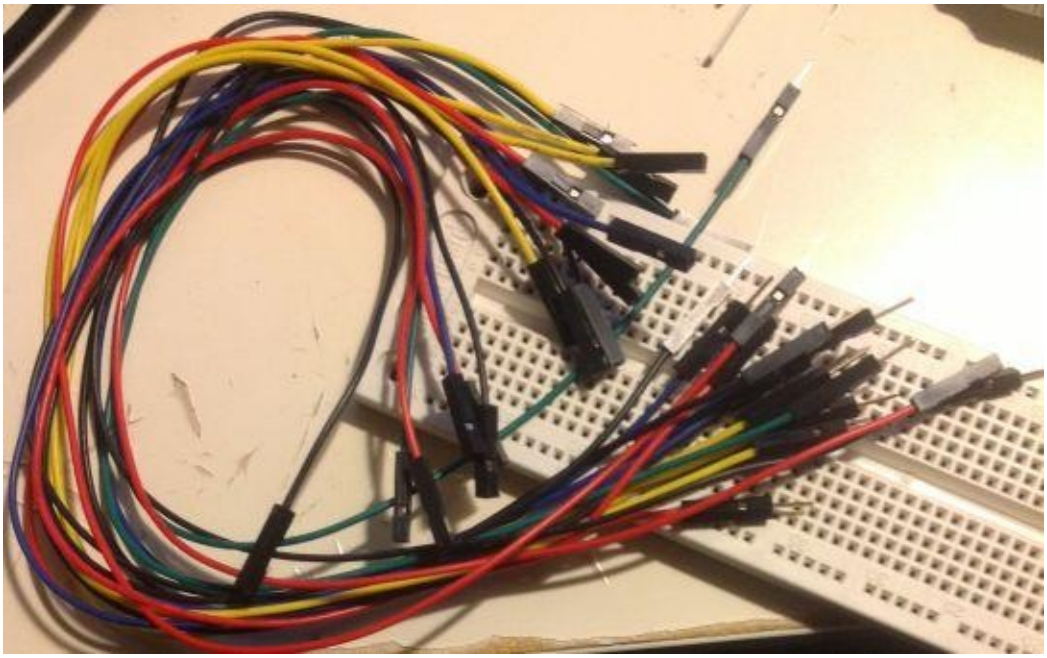
1.3 Development Tools

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

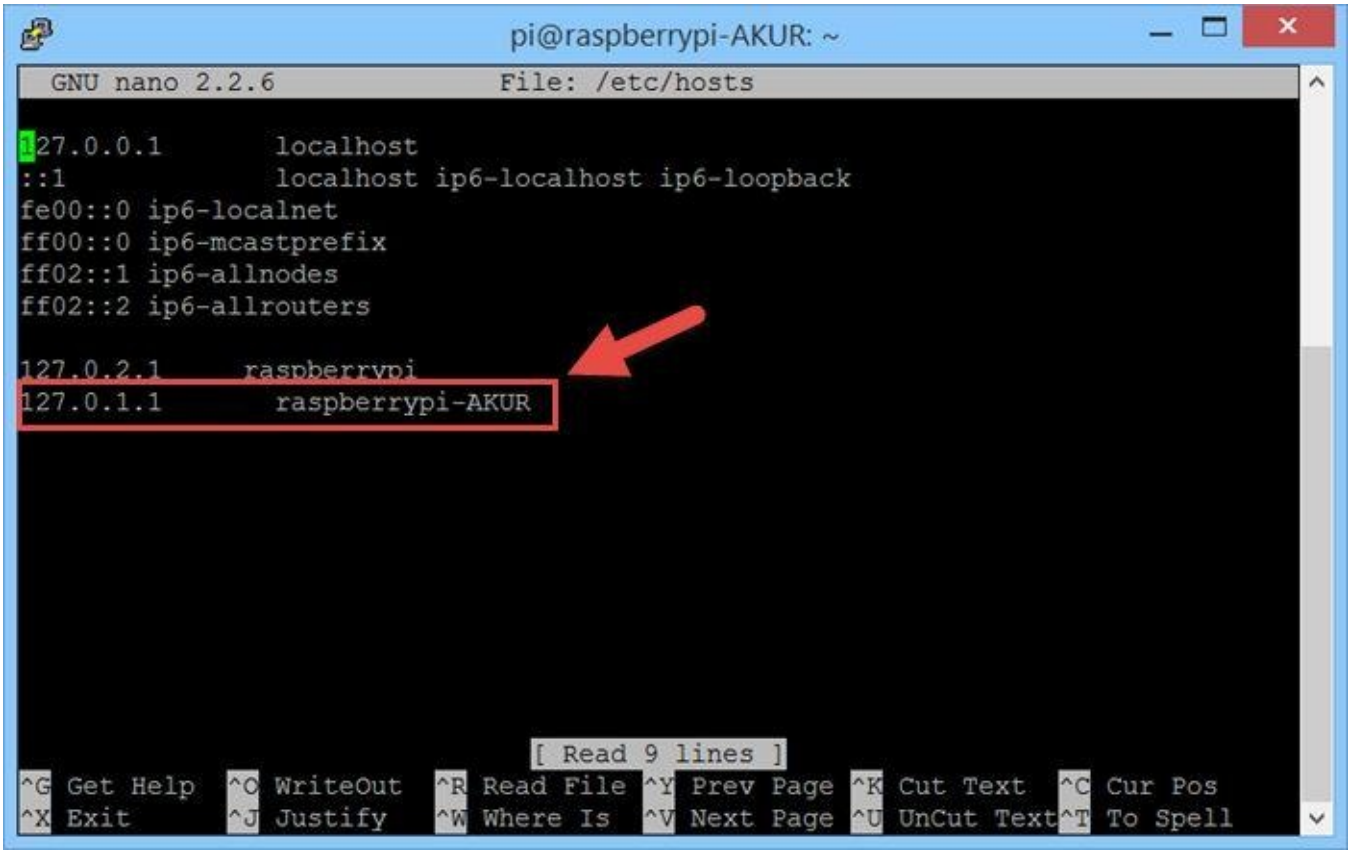
The default hostname of Raspberry Pi is raspberrypi. Pi's hostname should be changed if you have many Raspberry Pi devices to be connected to a network. Let's start to change our Raspberry Pi hostname.

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.

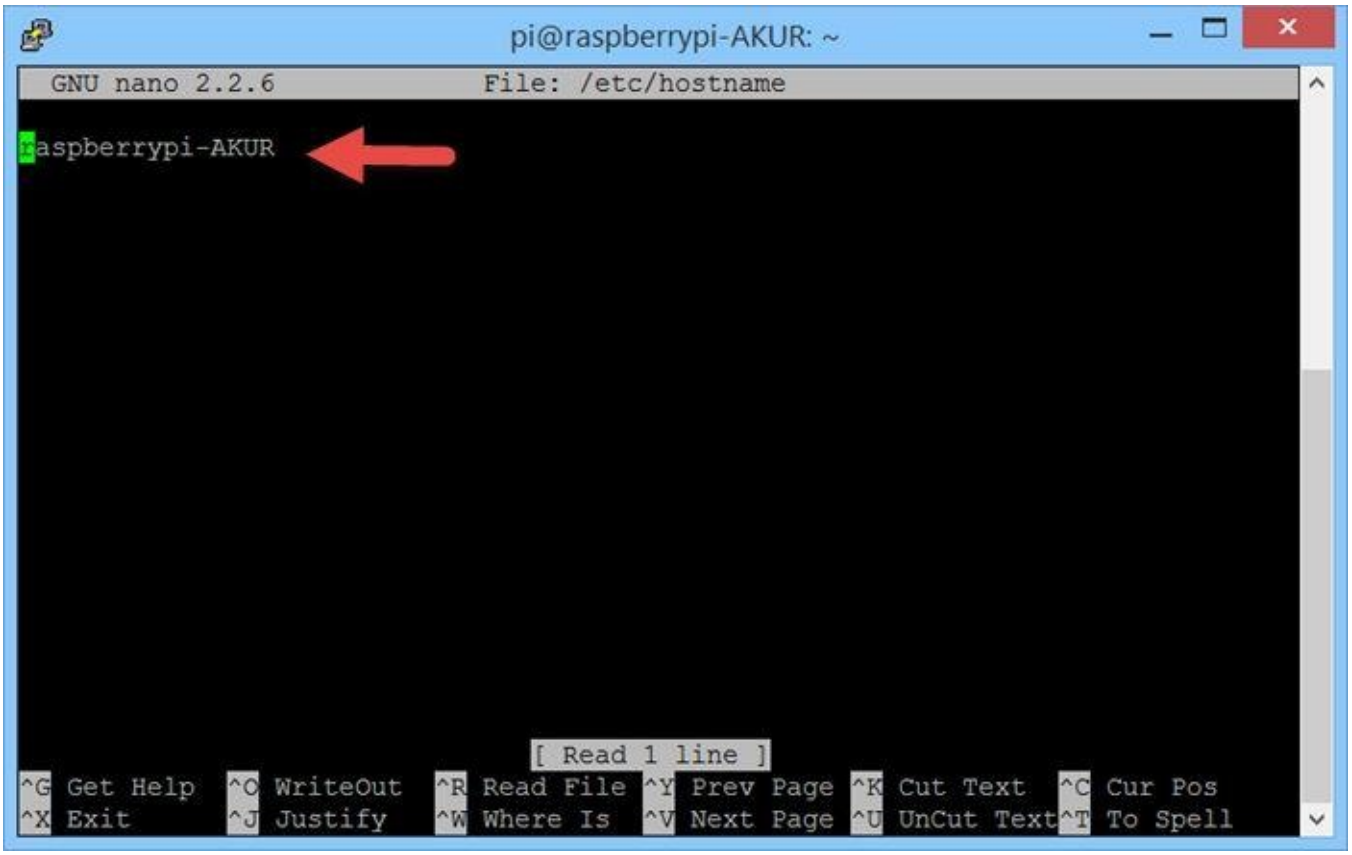


Save your changed.

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

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

Write your hostname.



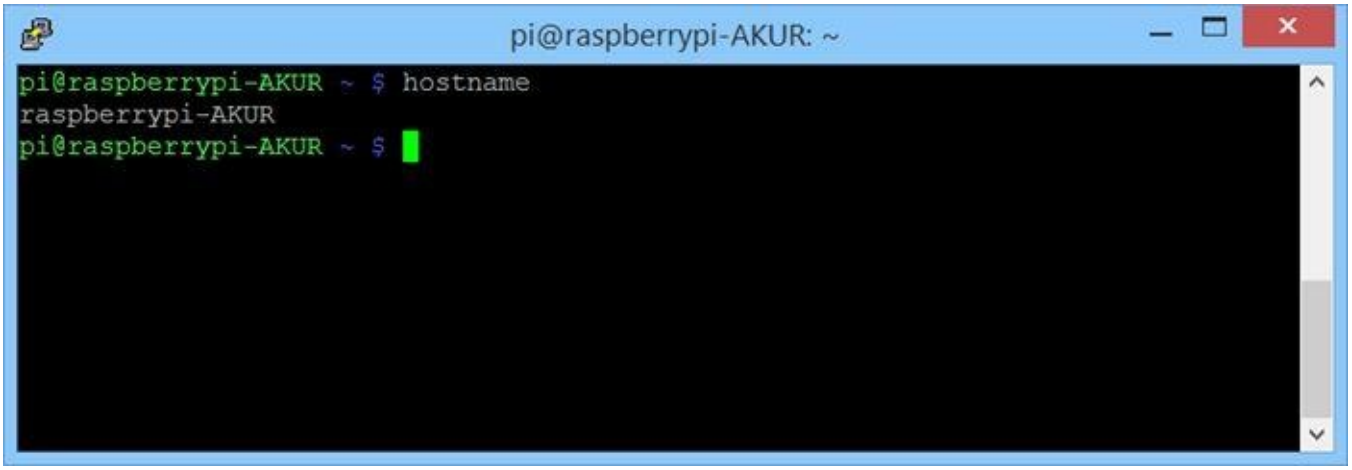
Save all your changed data.

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
```



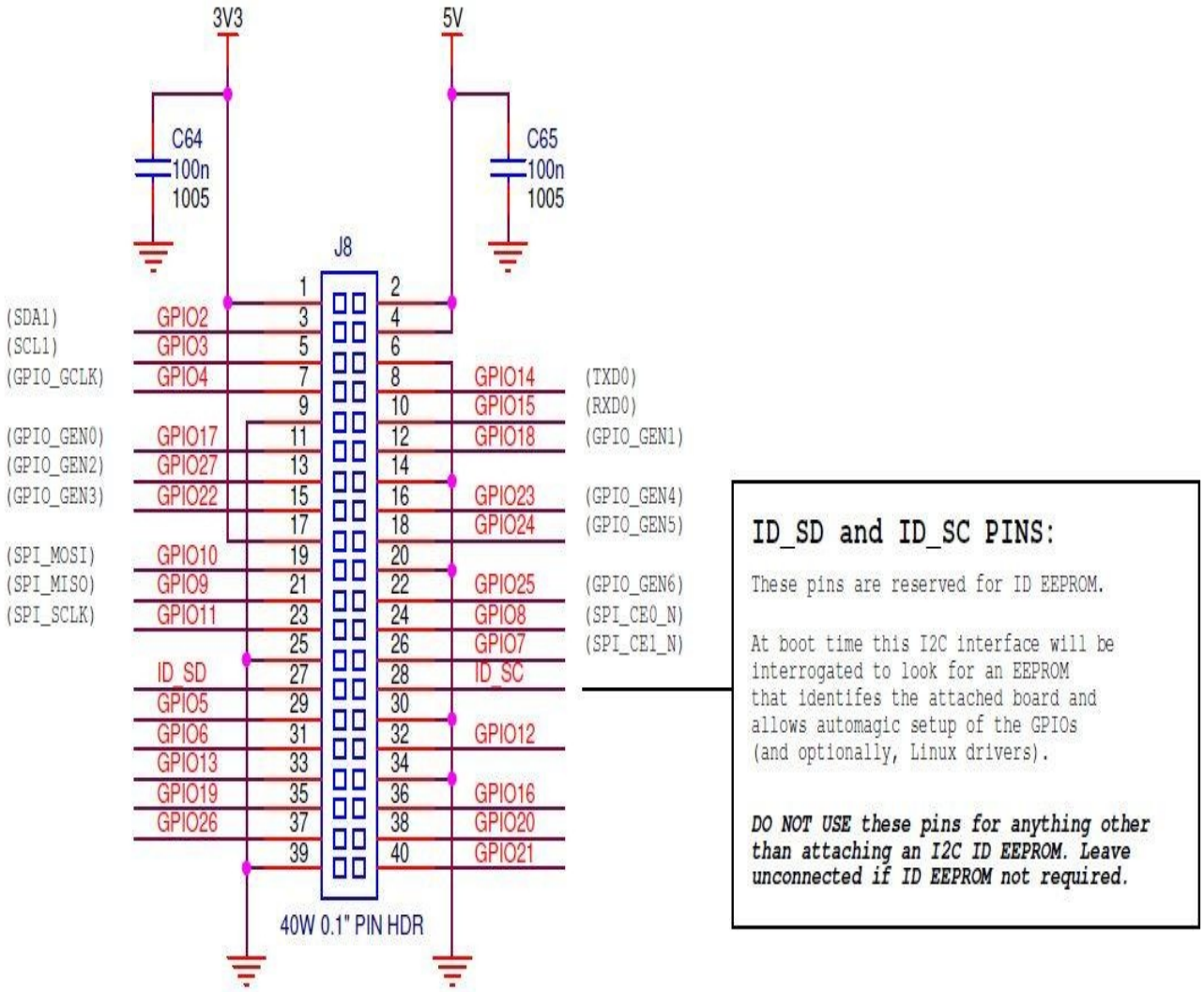
1.6 Raspberry Pi GPIO

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 . 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.



(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?

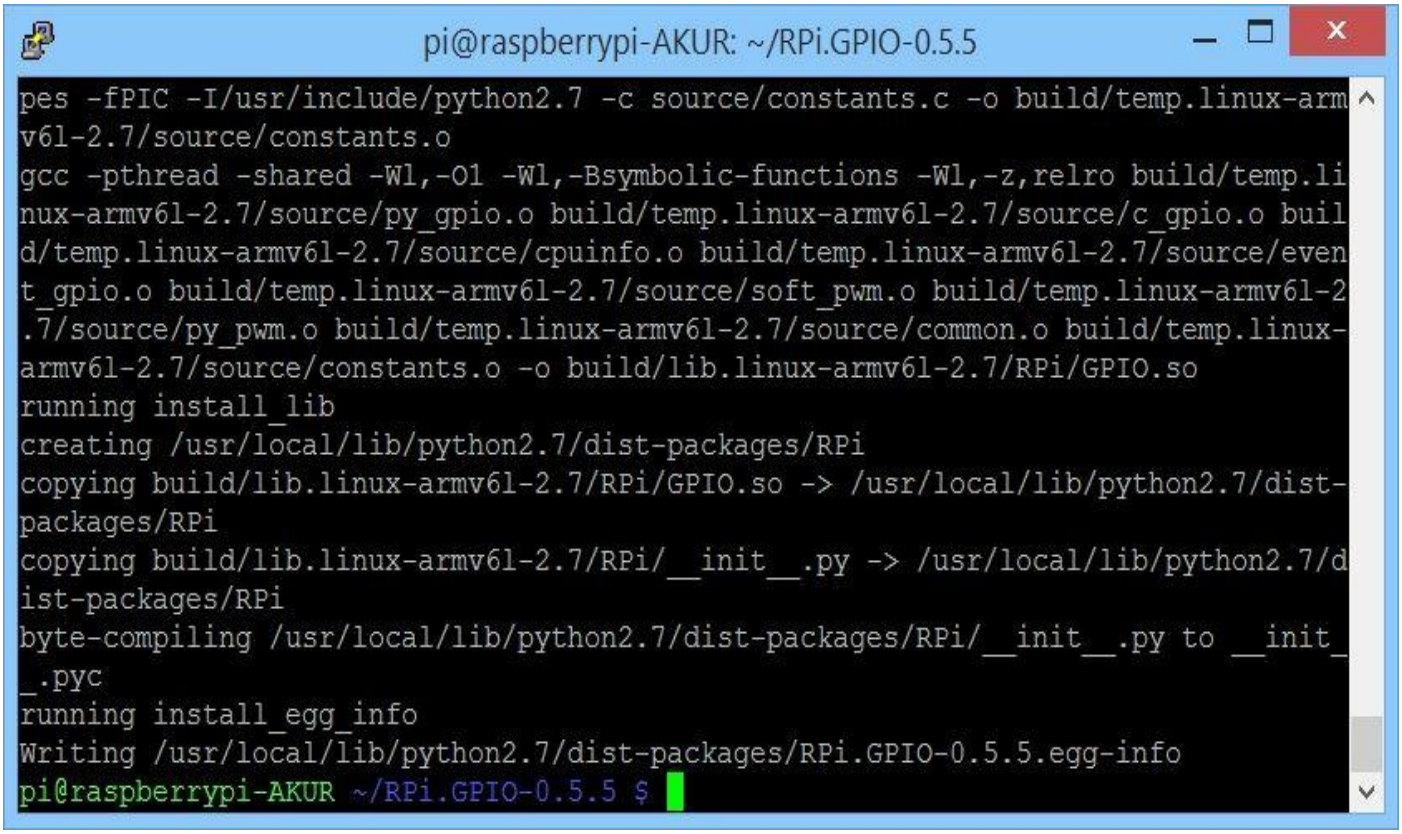
For illustration, we will install RPi.GPIO 0.5.5. Firstly, we need Python development library. Type the following command.

```
$ 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
```

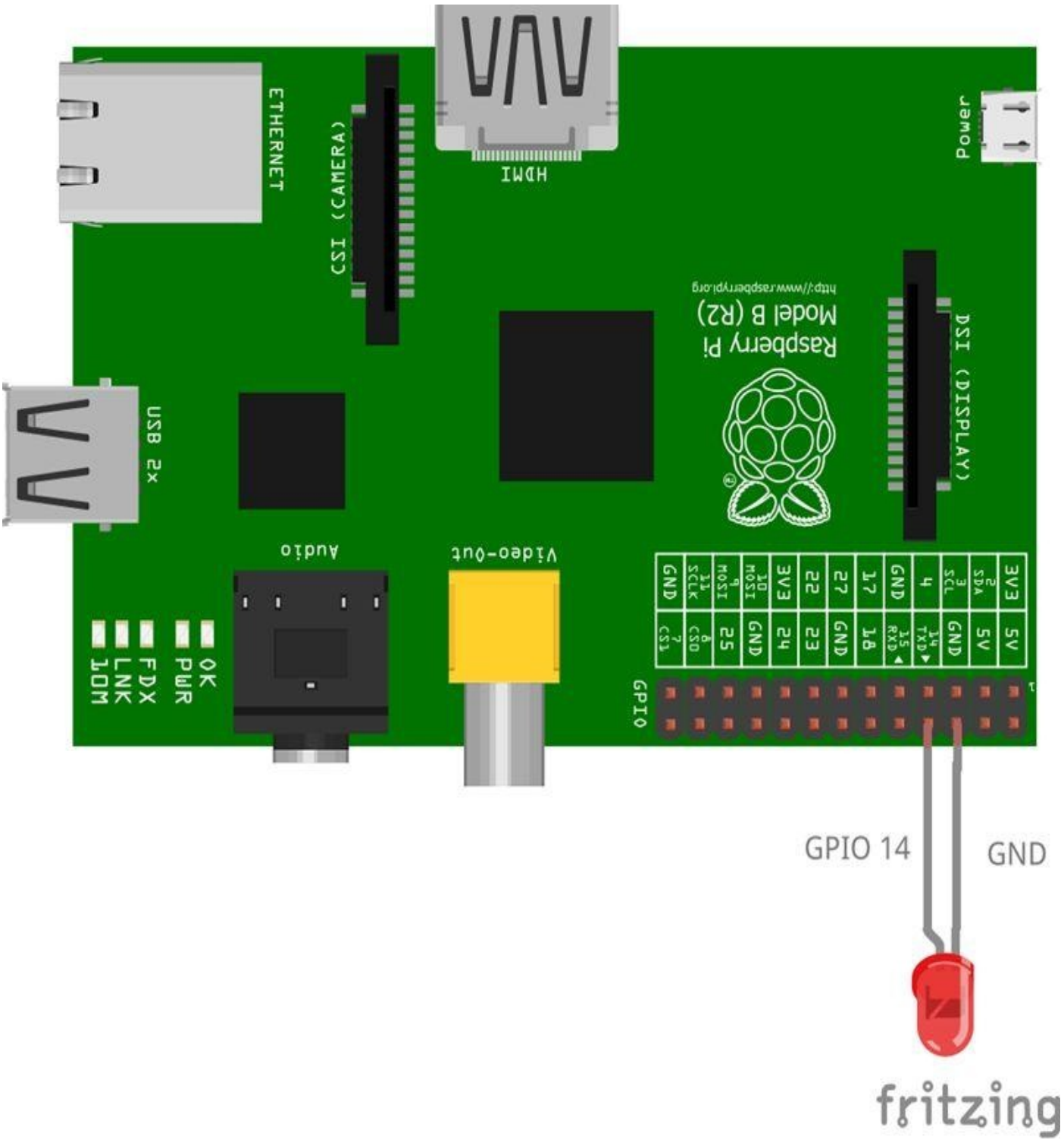
Make sure your Raspberry Pi already connected to Internet network.



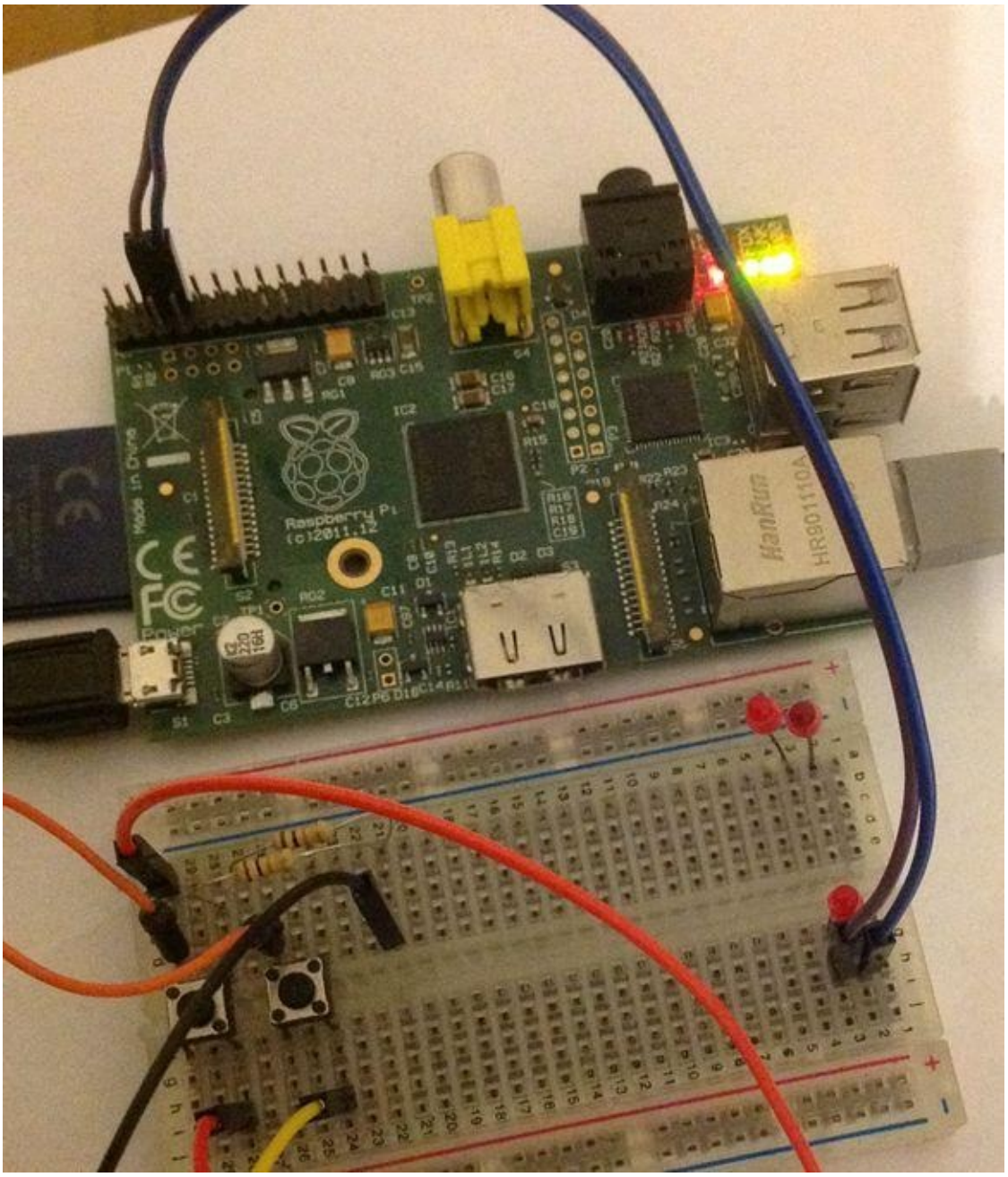
1.8 Hello World

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)
```

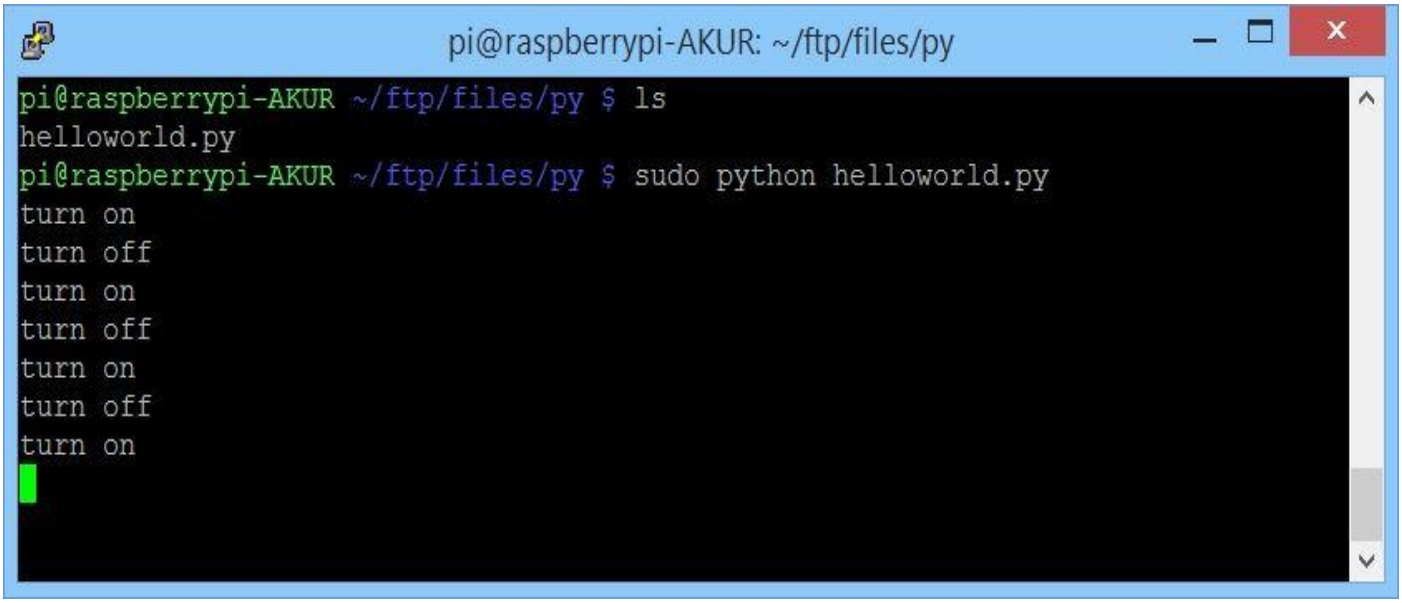
```
            print("turn on")
            GPIO.output(led_pin, True)
            time.sleep(2)
            print("turn off")
            GPIO.output(led_pin, False)


    except KeyboardInterrupt:
        GPIO.cleanup()

    print("done")
```

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

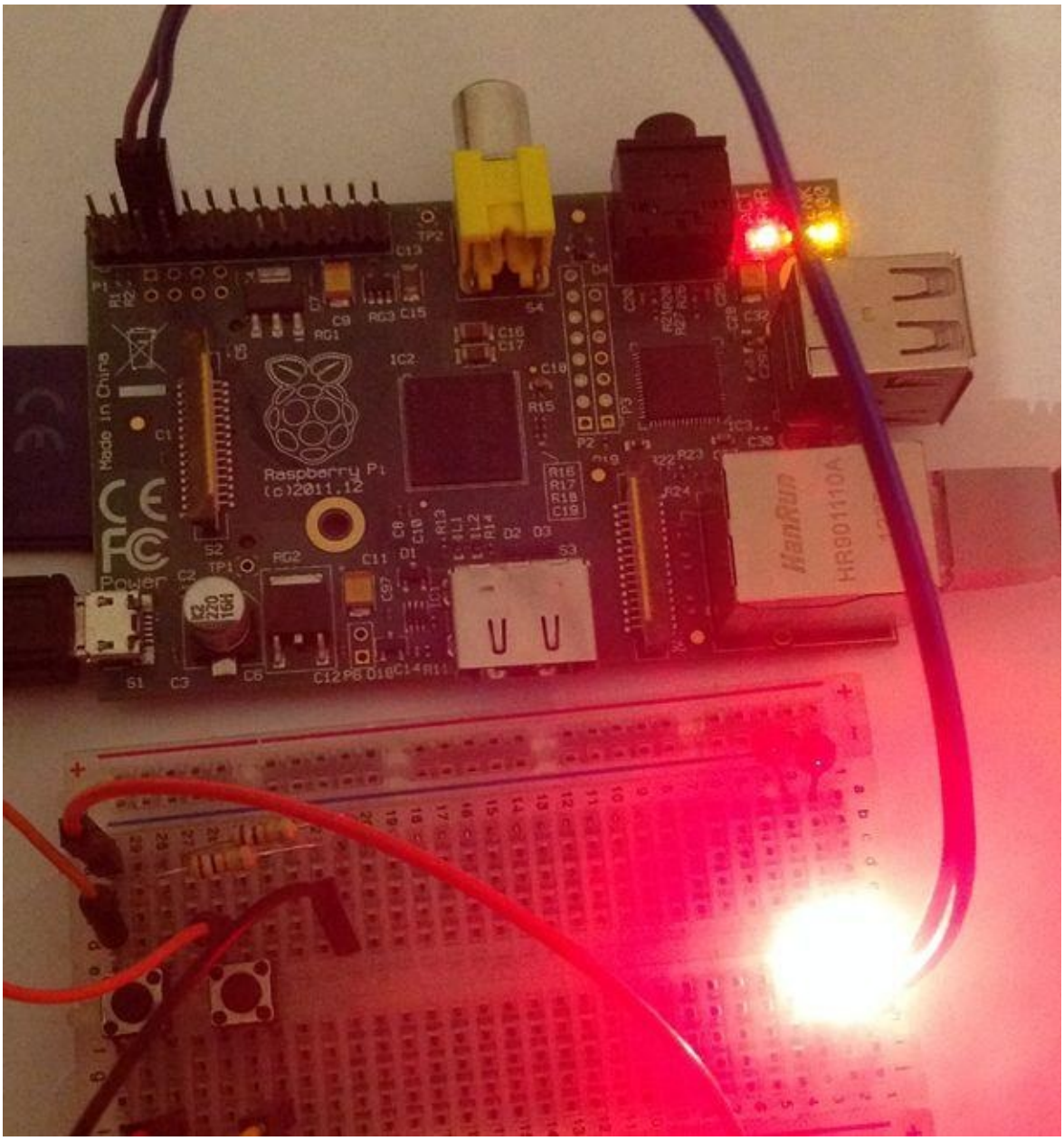
You can run this file by writing the following command.

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

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

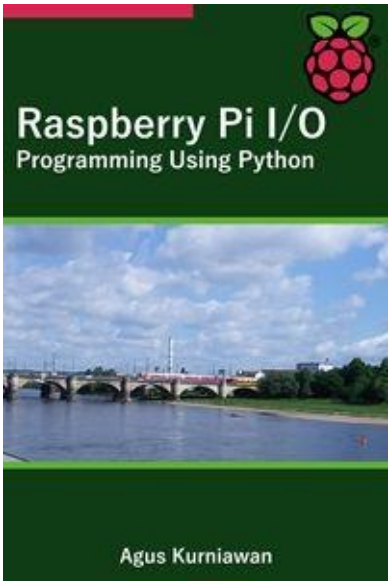


On your Raspberry Pi, you can see the LED is blinking.


Raspberry Pi
(c)2011.12
Made in China
Power
ManRun
HR901110A

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.

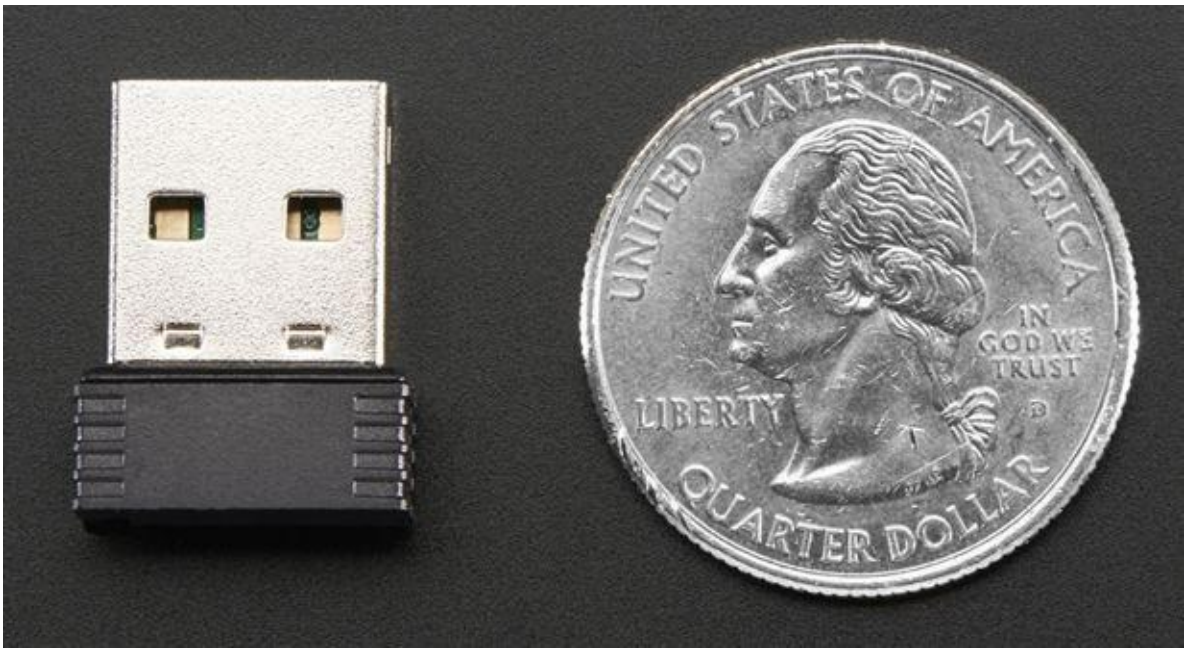
2.2.1 Wi-Fi USB Hardware

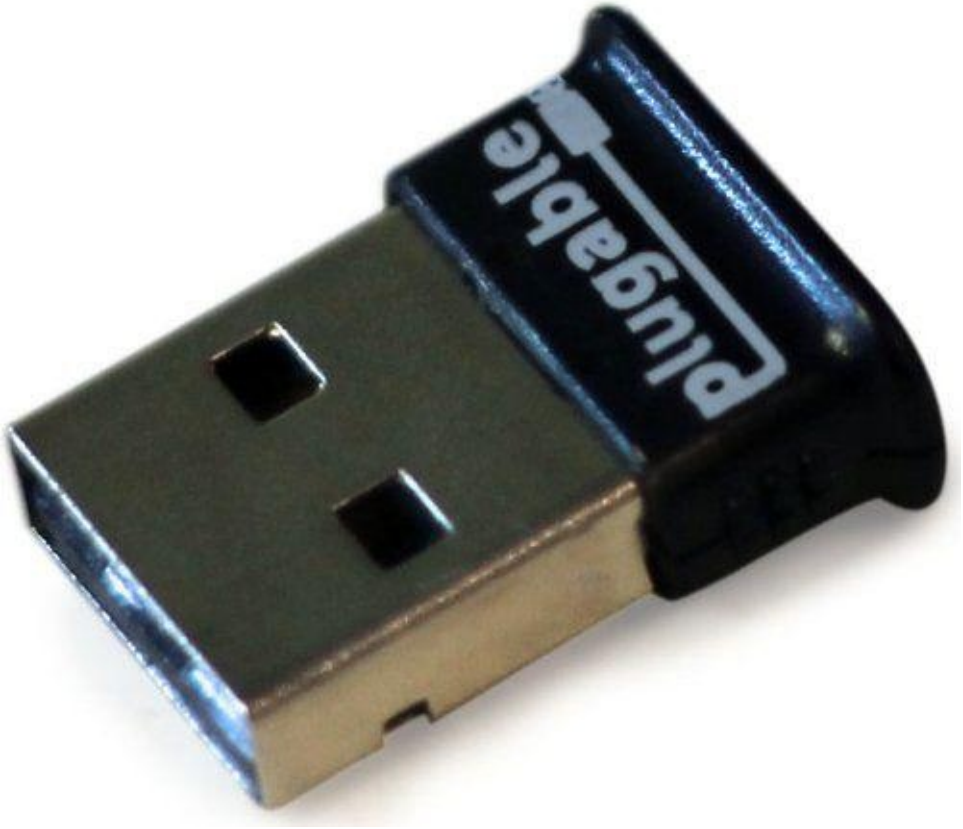
You can get Wi-Fi USB based on your interest, model and budget. You can find it on http://elinux.org/RPi_USB_Wi-Fi_Adapters . The following is a list of Wi-Fi USB samples:

- 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 .

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

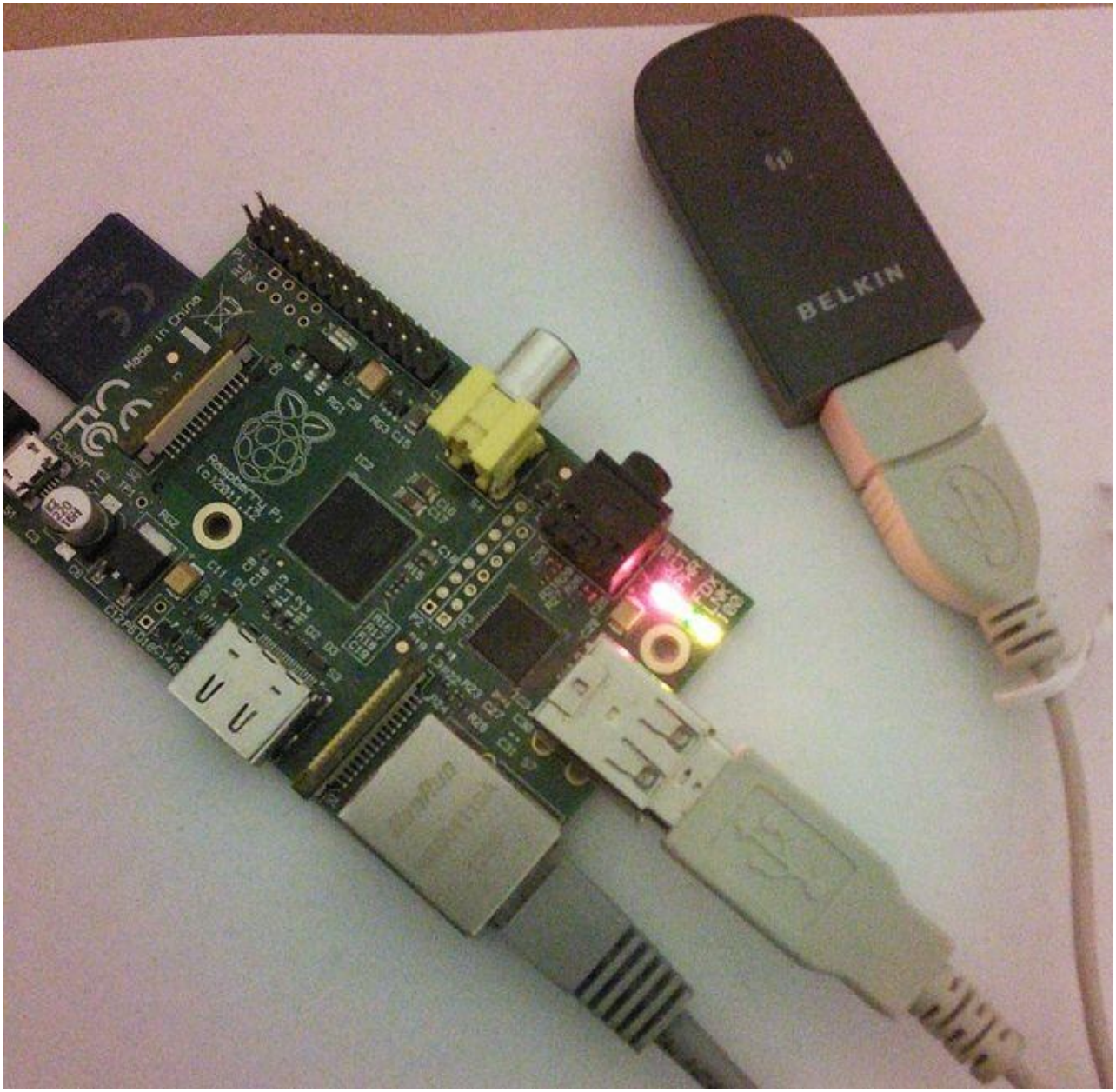




I use WIFI dongle from Belkin. My Raspberry Pi detected this WIFI adapter.

2.2.2 Demo: Connecting Wi-Fi Network

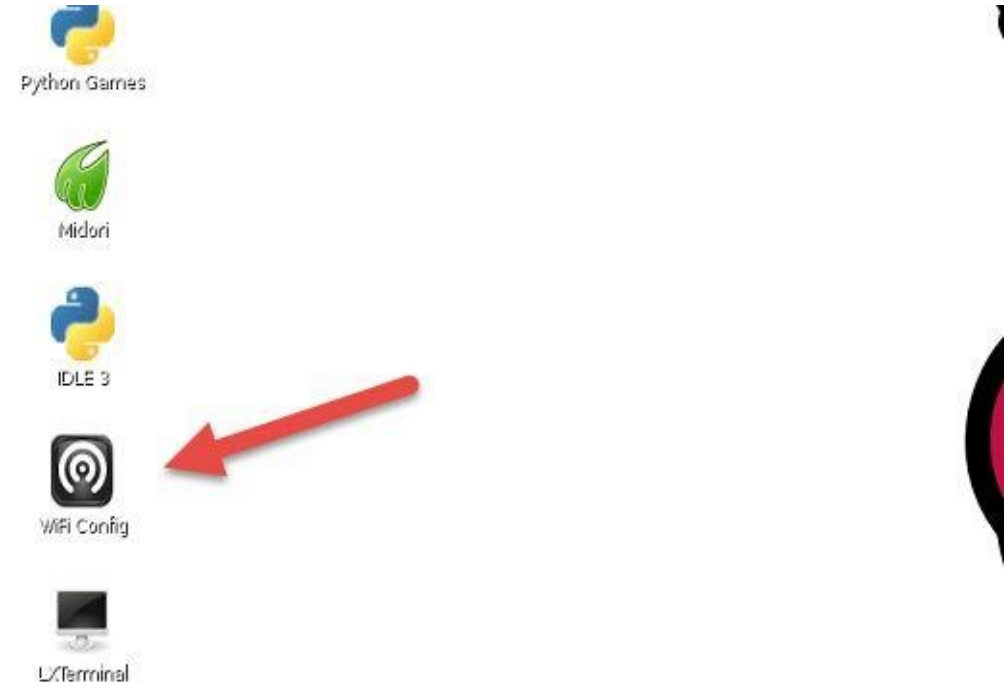
In this section, I want to show you how to connect your Wi-Fi USB via Raspberry Pi desktop. Firstly, you connect your Wi-Fi USB to Raspberry Pi.



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

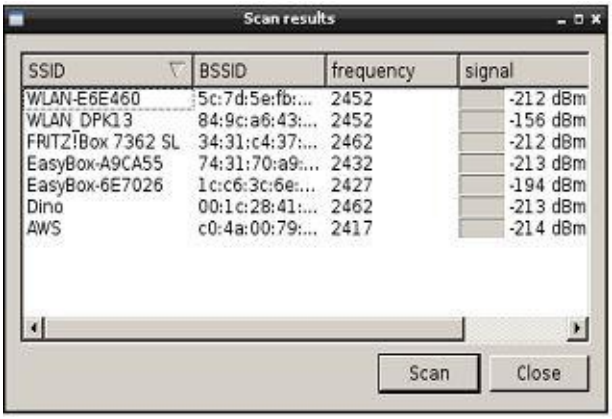
```
$ sudo apt-get install wireless-tools
```

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



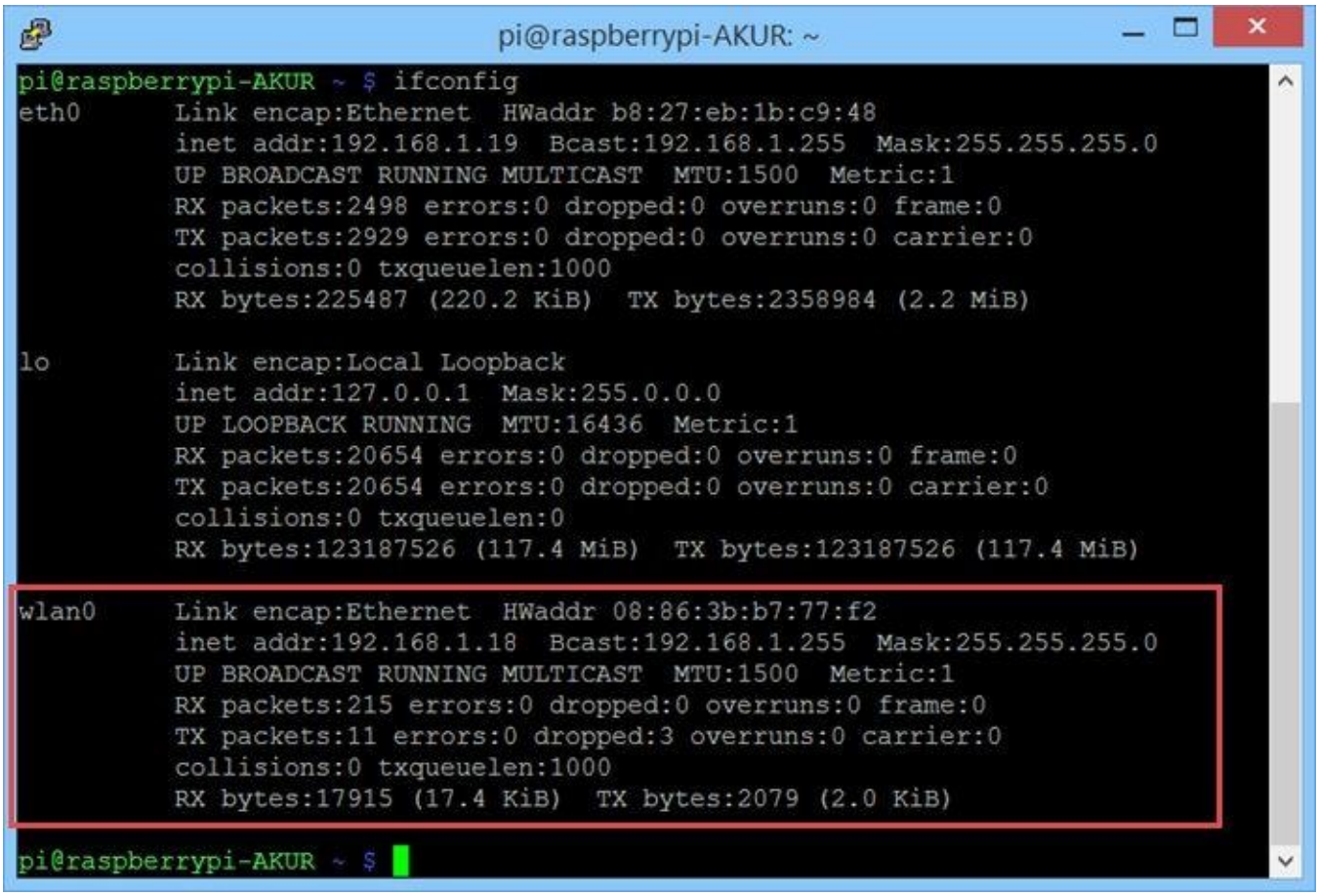
You will get a dialog as follows.

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



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 Termina. You see wlan0 and its IP Address.



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

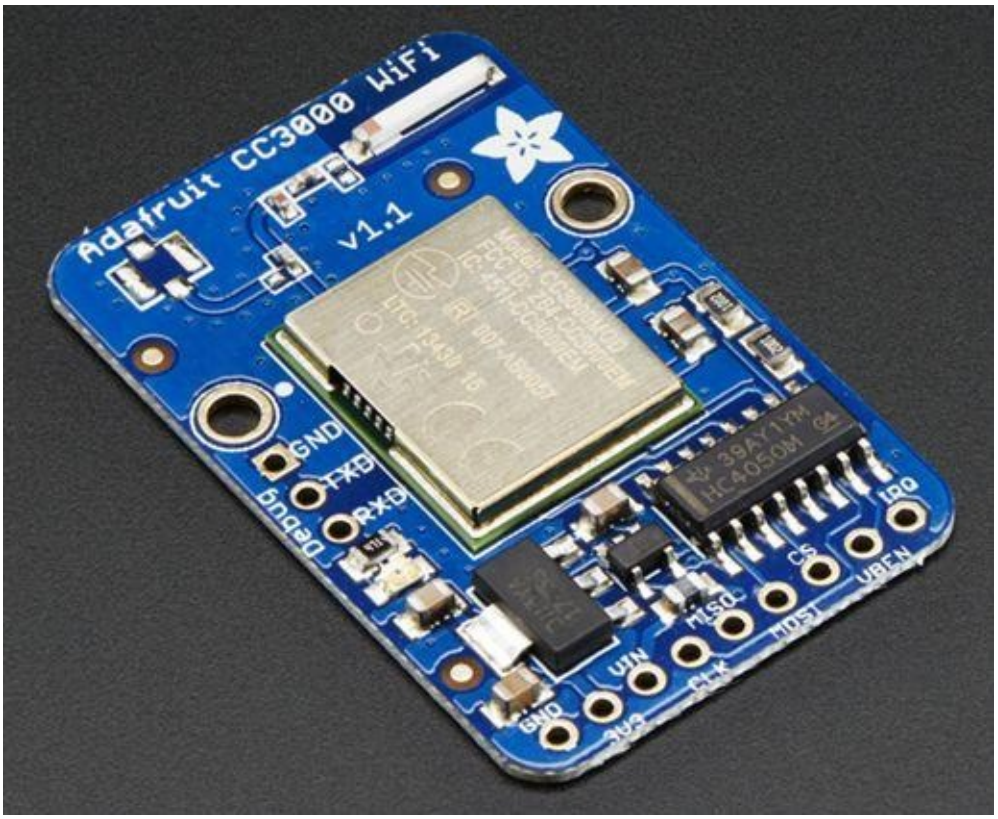
2.3 Wi-Fi IEEE 802.11 Wireless Module

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.

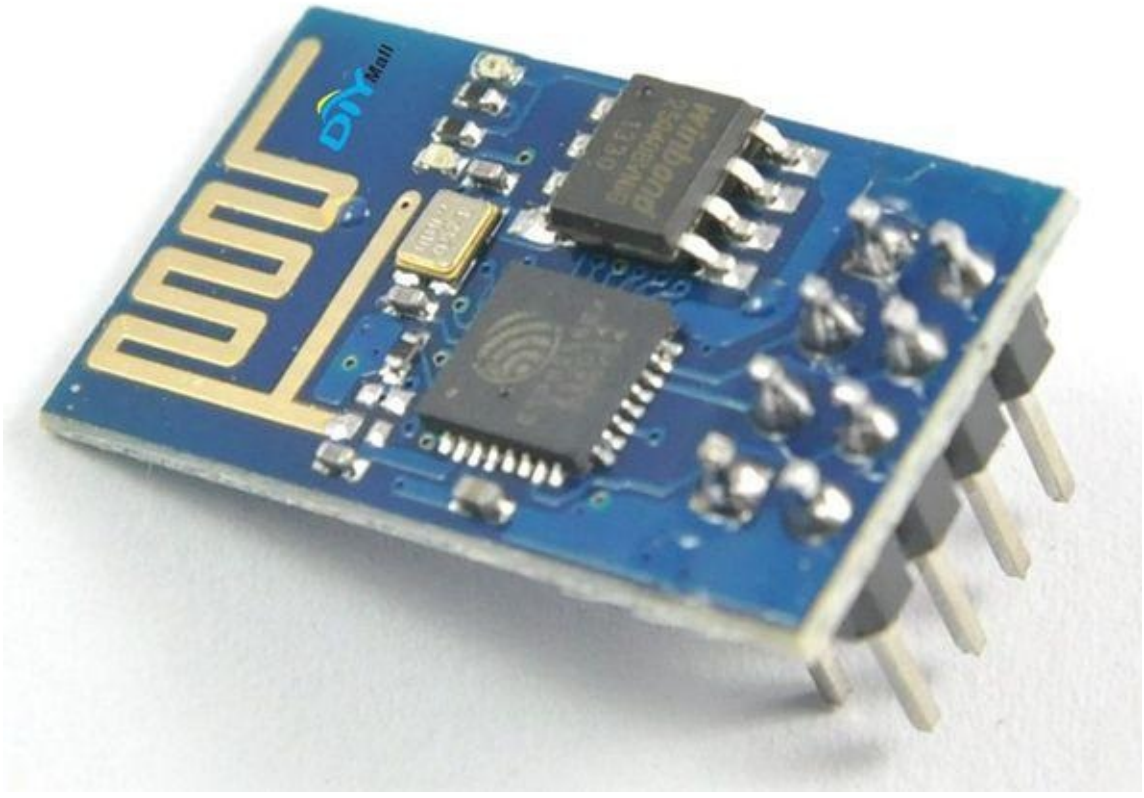
2.3.1 Wi-Fi IEEE 802.11 Wireless Module Hardware

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



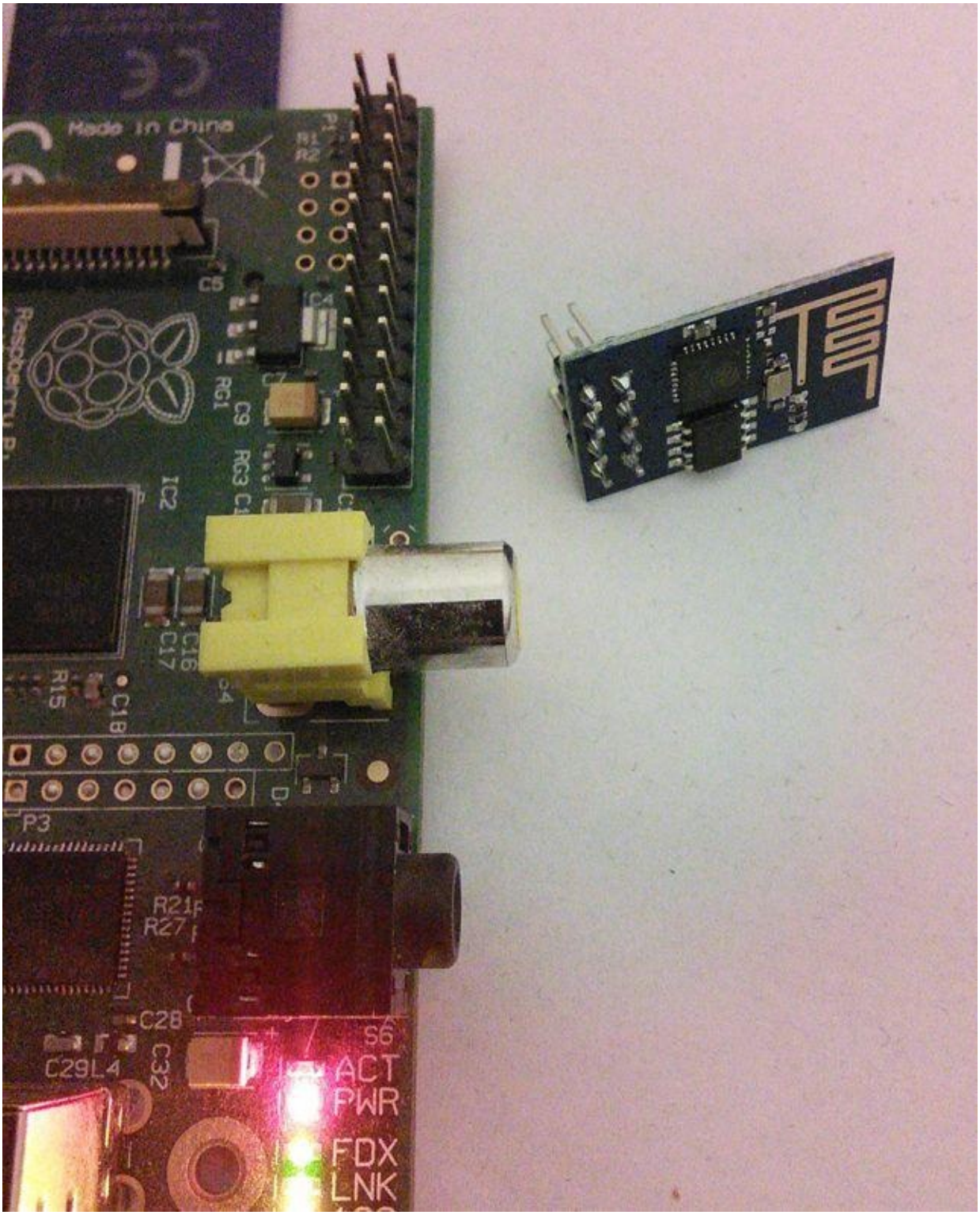
For testing, I use **ESP8266EX Serial WIFI Wireless**. It is a cheap Wi-Fi module about US$ 3.50. You can find it on Amazon.com, http://www.amazon.com/Diymall-Esp8266-Serial-Wireless-Transceiver/dp/B00O34AGSU/ , ebay or Seeedstudio http://seeedstudio.com/depot/WiFi-Serial-Transceiver-Module-w-ESP8266-p-1994.html.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



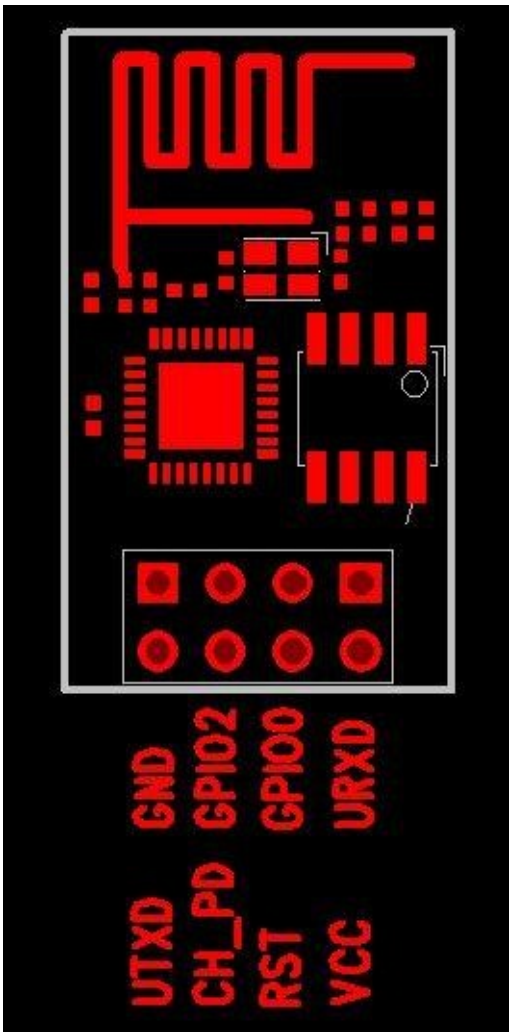
Further information about , you can read datasheet document on http://www.electrodragon.com/w/Wi07c and https://nurdspace.nl/images/e/e0/ESP8266_Specifications_English.pdf .

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

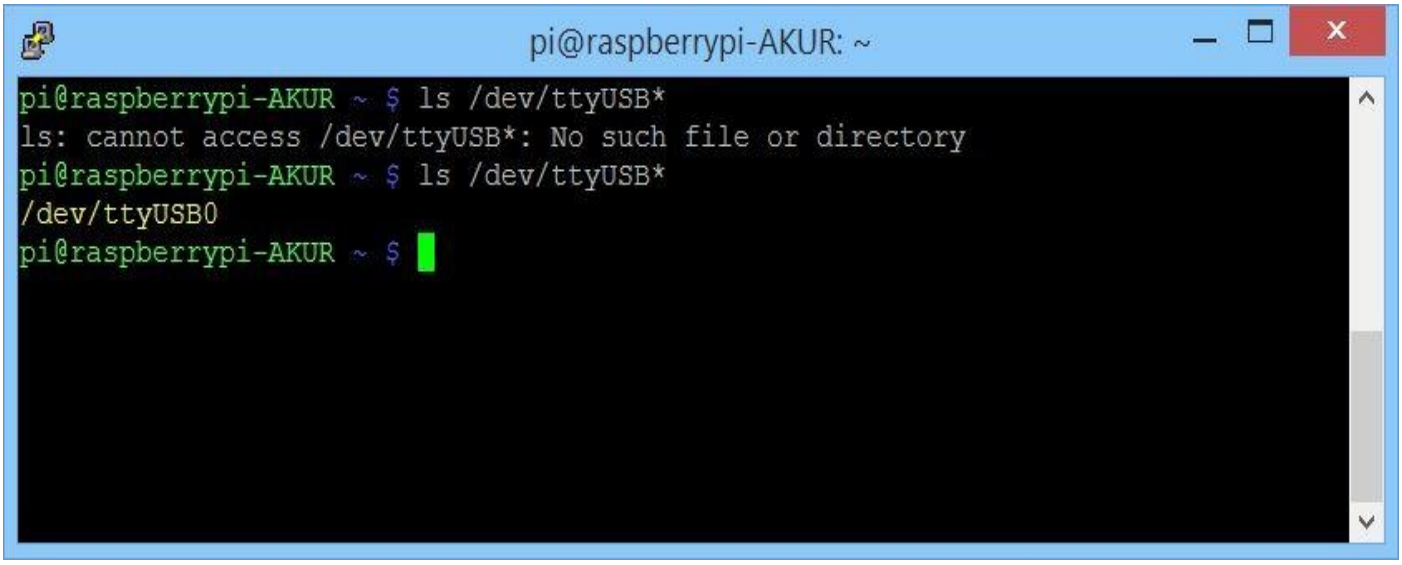
If you want to connect ESP8266 module to Raspberry Pi via USB, you need TTL USB module. The following is wiring

- 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*
```



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 . In this demo, we try to connect a hostspot 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.

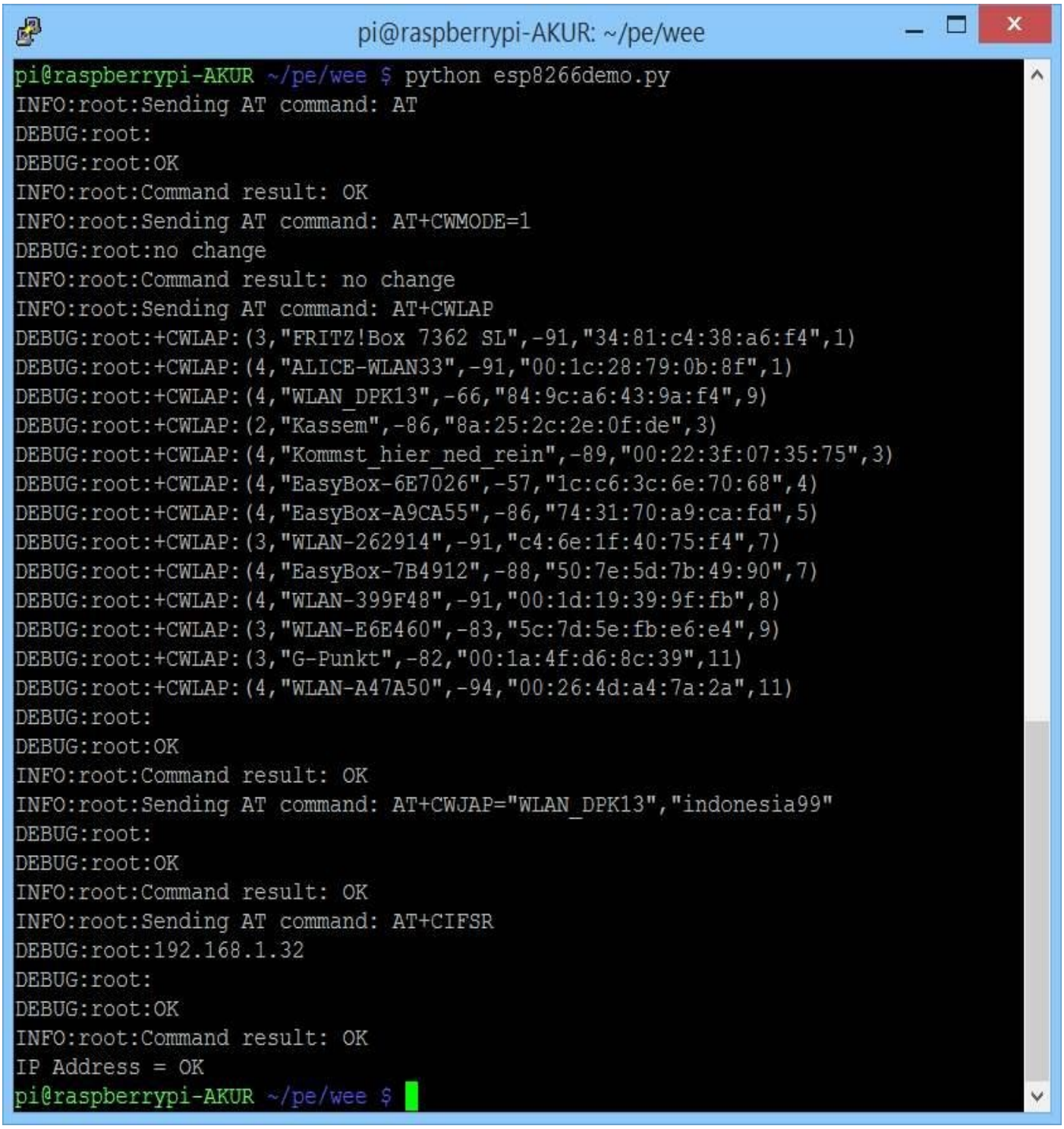
```
import logging
import sys, serial
from time import *


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

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



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.

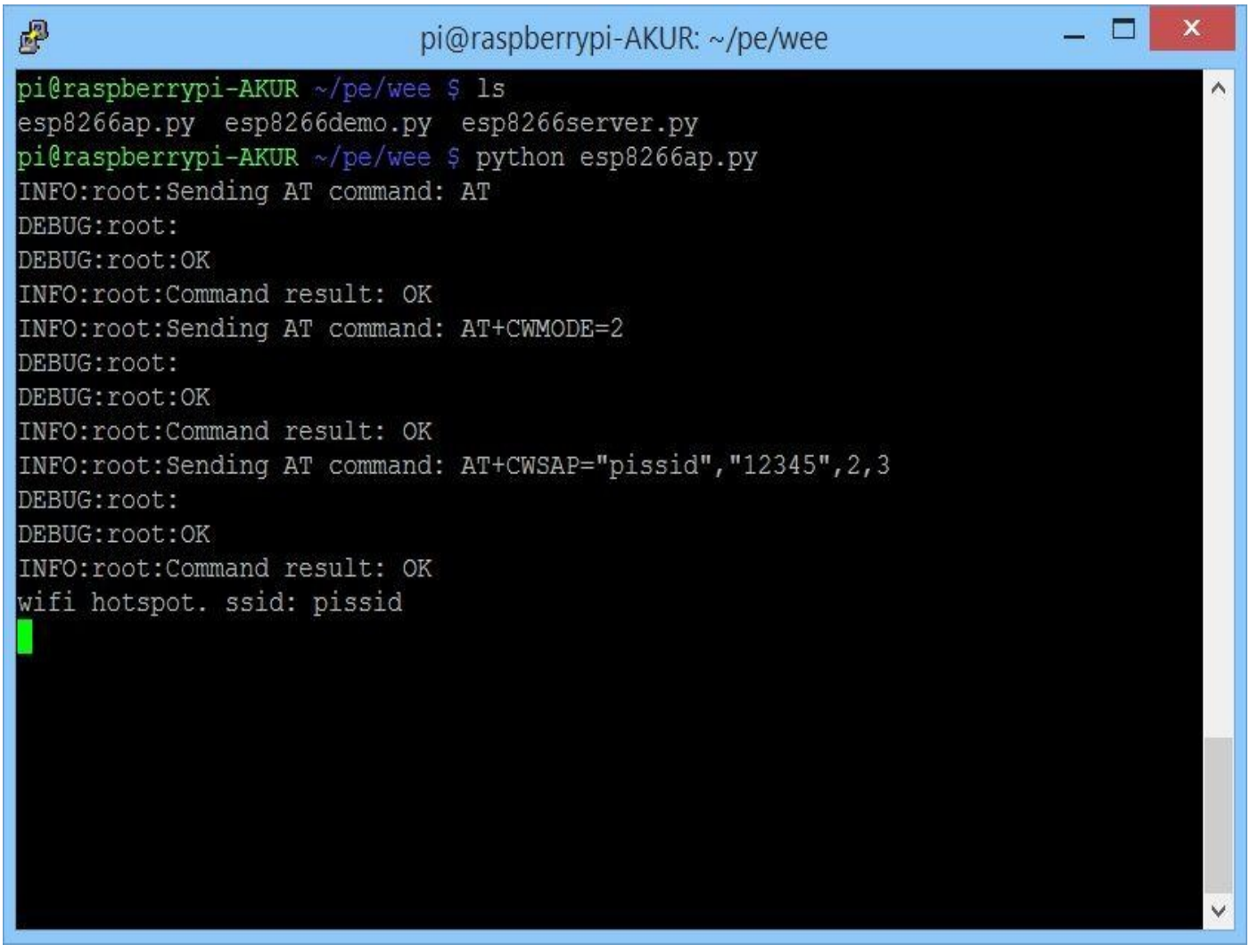
```
import logging
import sys, serial
from time import *


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

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



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 follws:

- 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
```

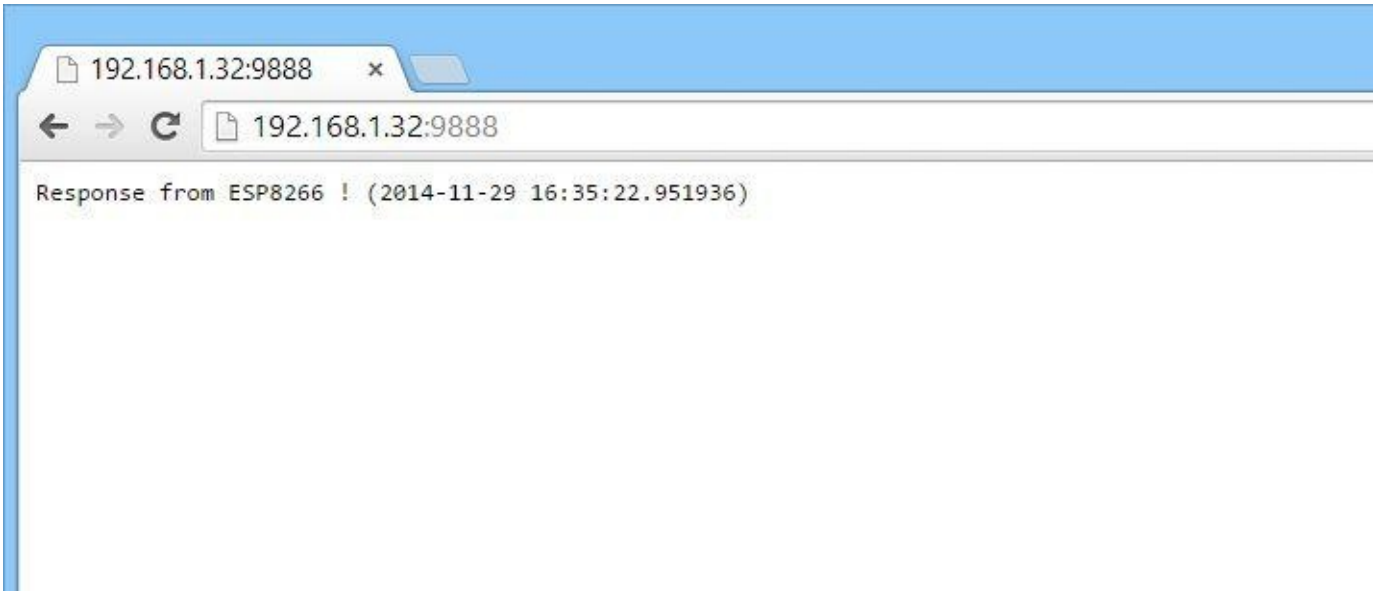
```
def enum(**enums):
    return type('Enum', (), enums)

Status         =         enum(ERR=['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.
        sleep(0.3)

    ser.close()
```

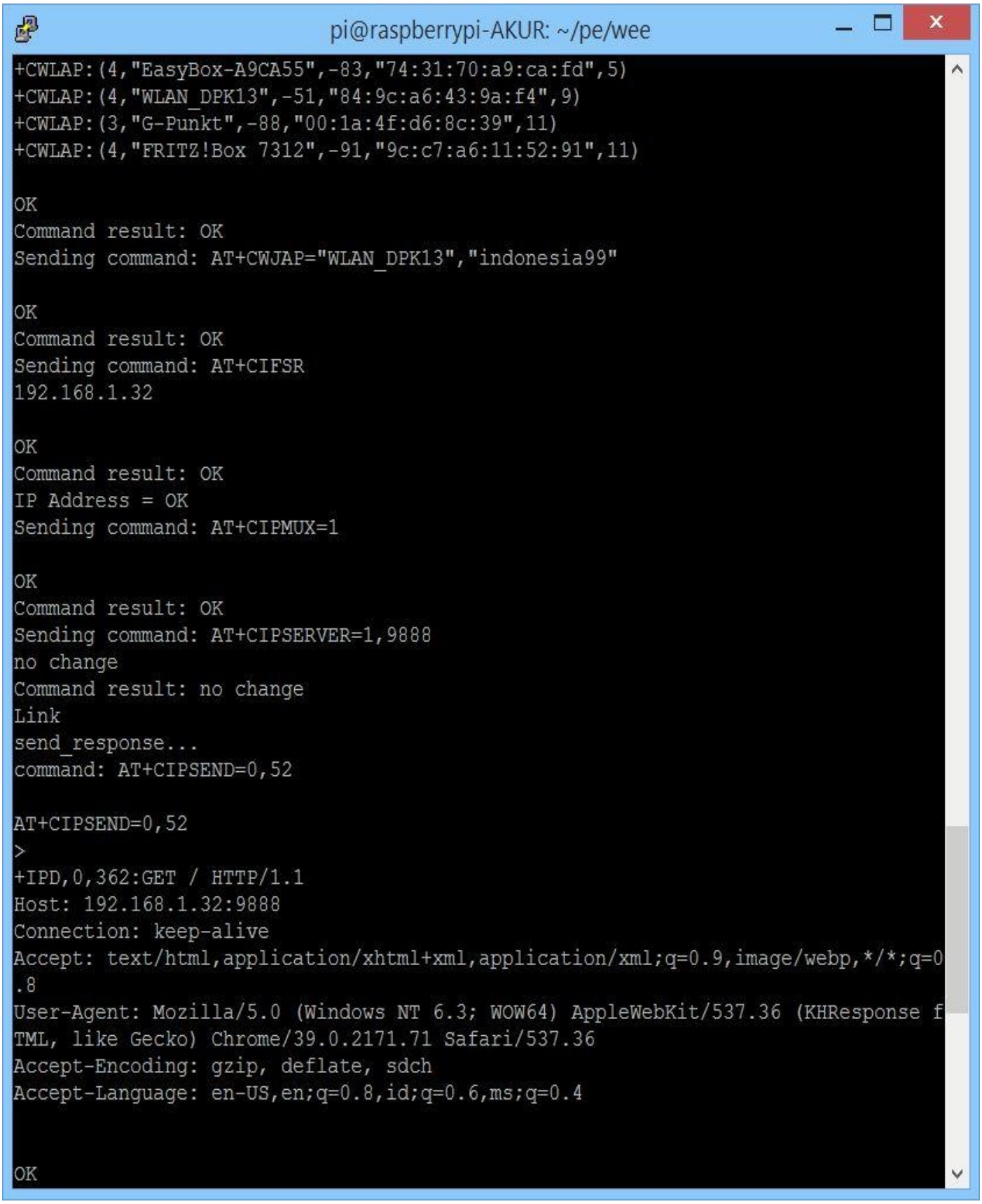
Save this file and try to run.

```
$ 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 (KHResponse f
TML, 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

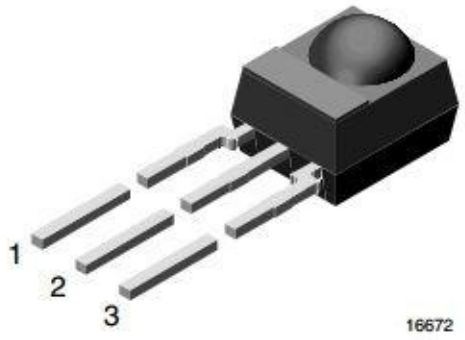
We will explore theses hardware to implement IR Remote application.

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_S

Pinning for TSOP22.., TSOP24..:

1 = OUT, 2 = V_S, 3 = GND

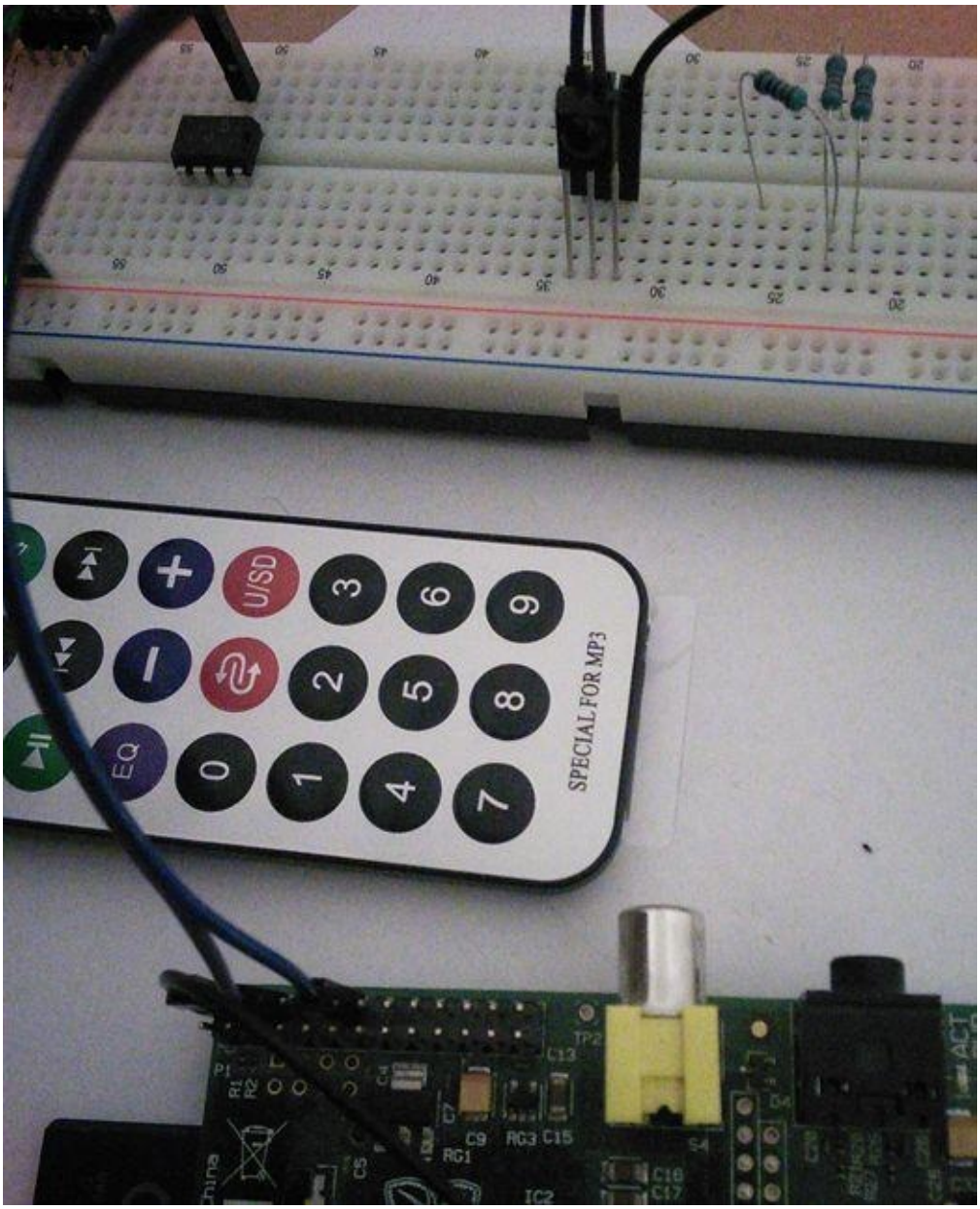
You also need IR remote which is used to IR transmitter. You can get it from this store:

- 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.

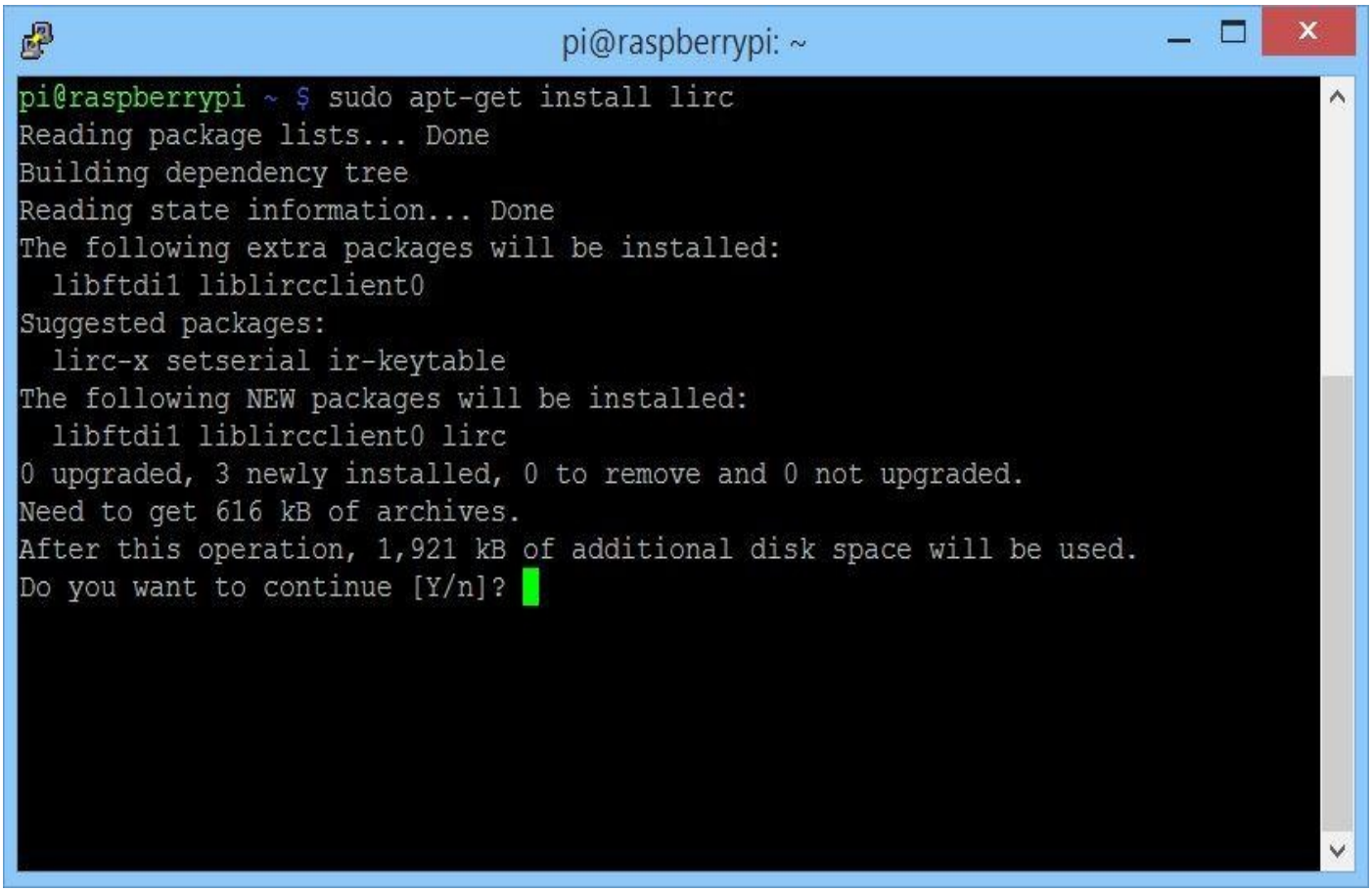


3.2.2 Configuring and Testing

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
```

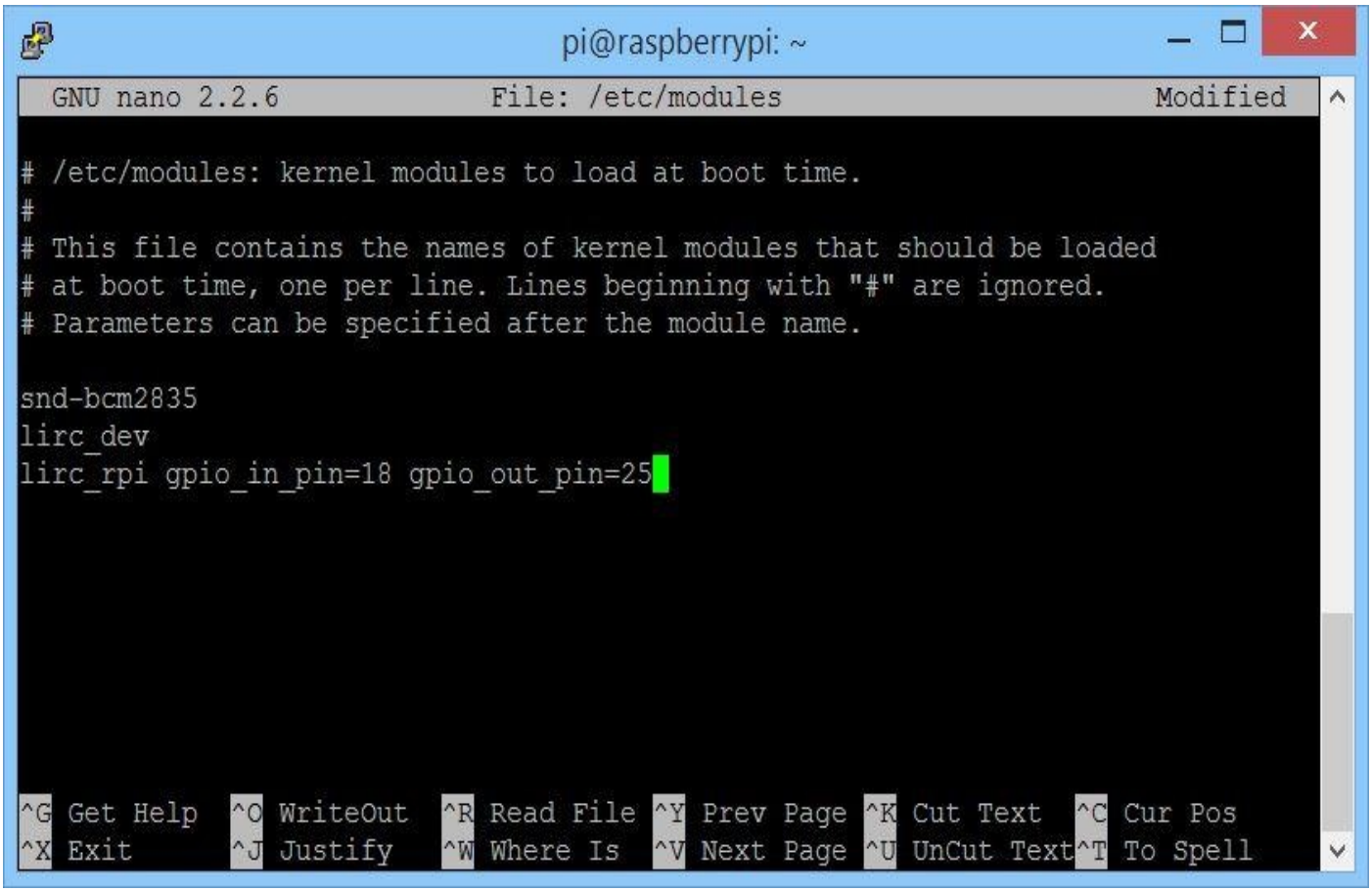


After installed, we modify **/etc/modules** file to load lirc into OS.

```
$ 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
```



Save this file and close nano.

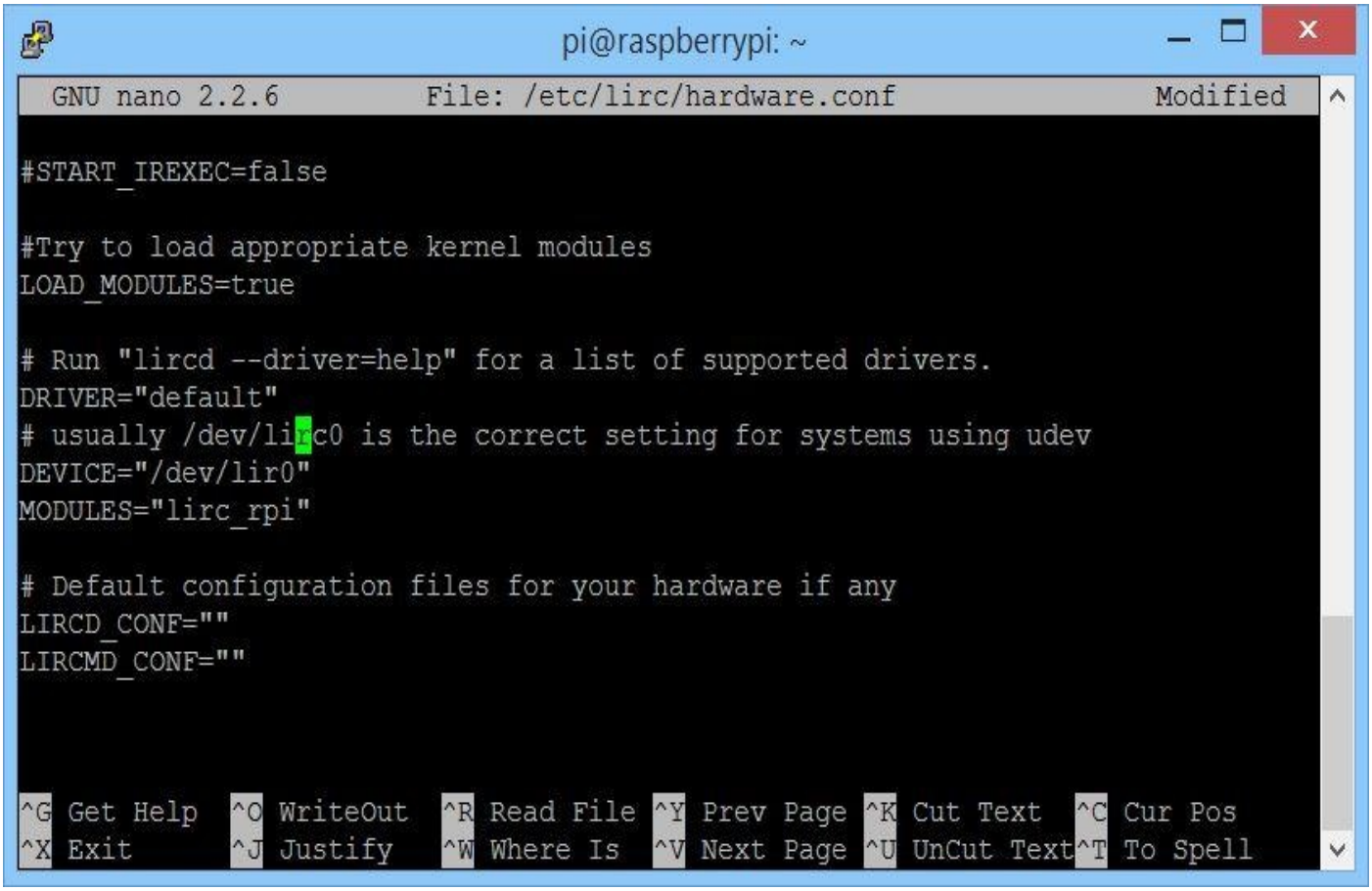
Now you edi **/etc/lirc/hardware.conf** file.

```
$ 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=""
############################################################
```



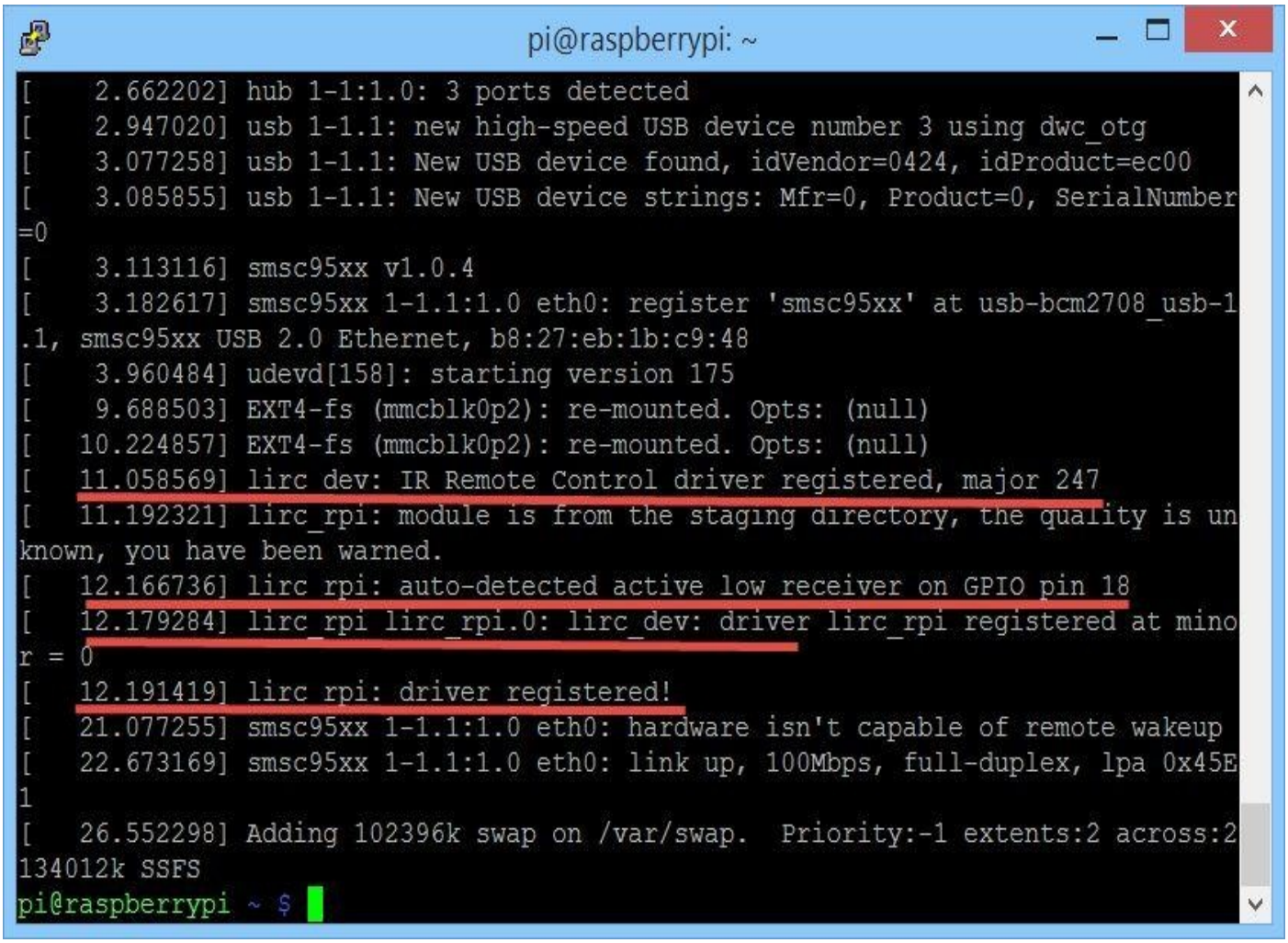
Save this modified file. Now you can restart your board.

```
$ sudo reboot
```

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

```
$ dmesg
```

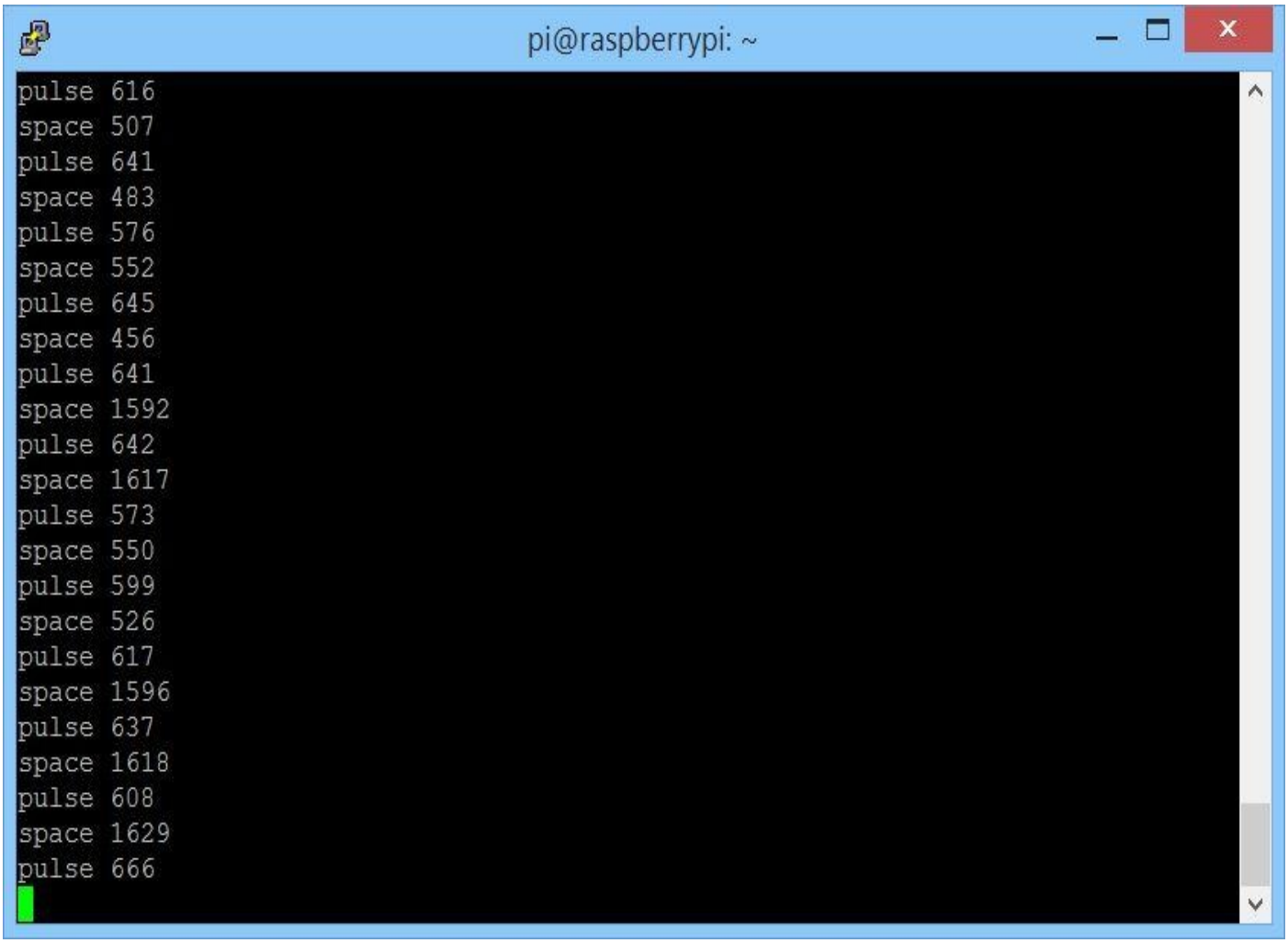
If success, you can see “lirc_rpi: driver registered” on dmesg messages.



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.



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.

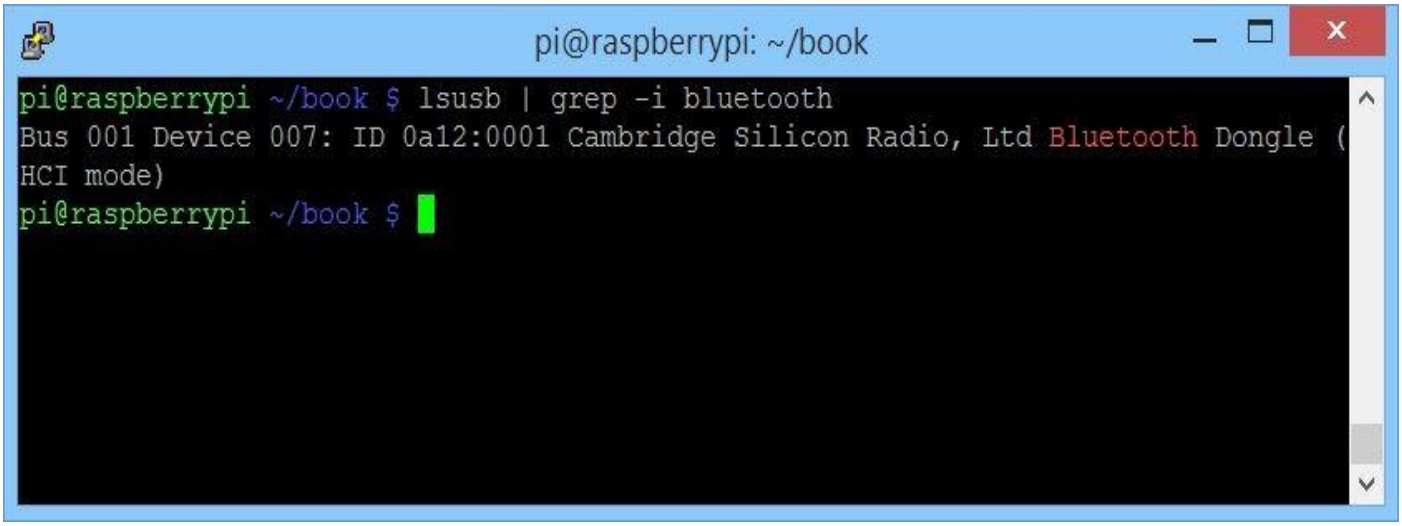
```
$ sudo reboot
```

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

You can verify this bluetooth device using grep.

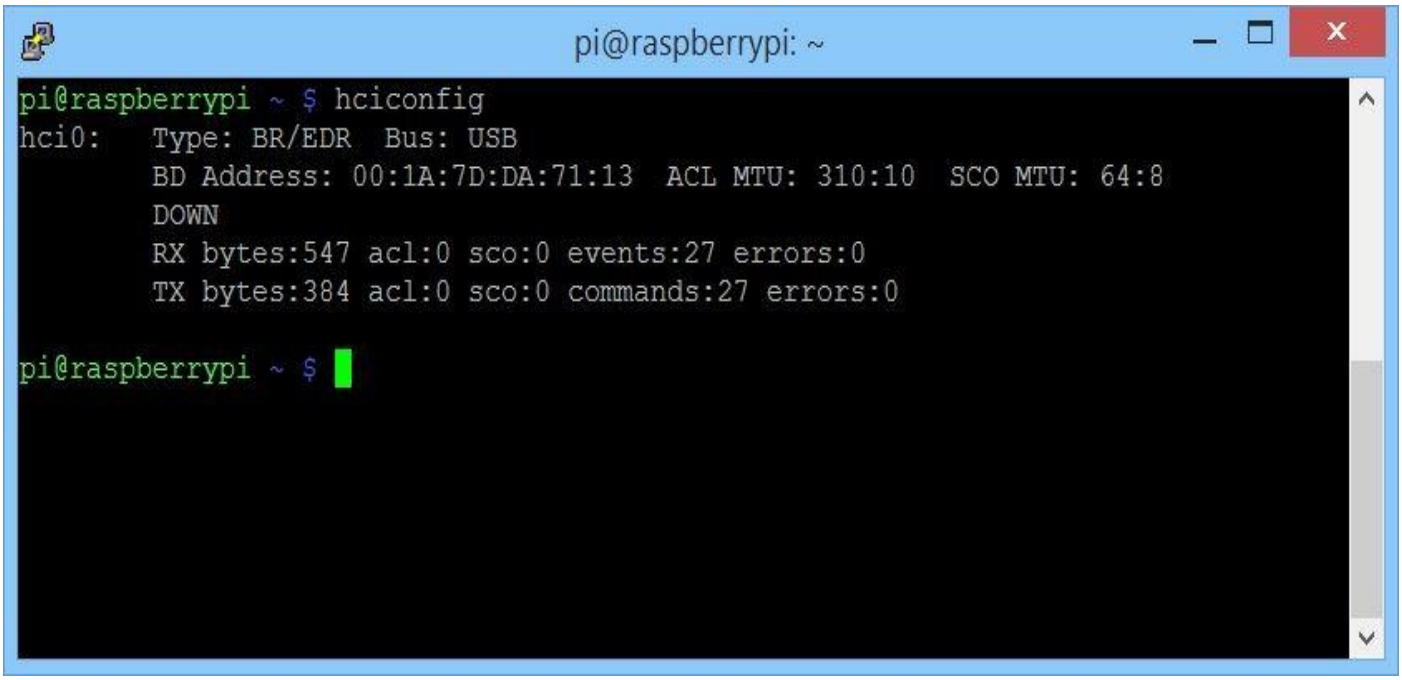
```
$ lsbusb | grep -i bluetooth
```

If success, you can see your Bluetooth USB.



Now you can check your bluetooth config using hciconfig.

```
$ hciconfig
```

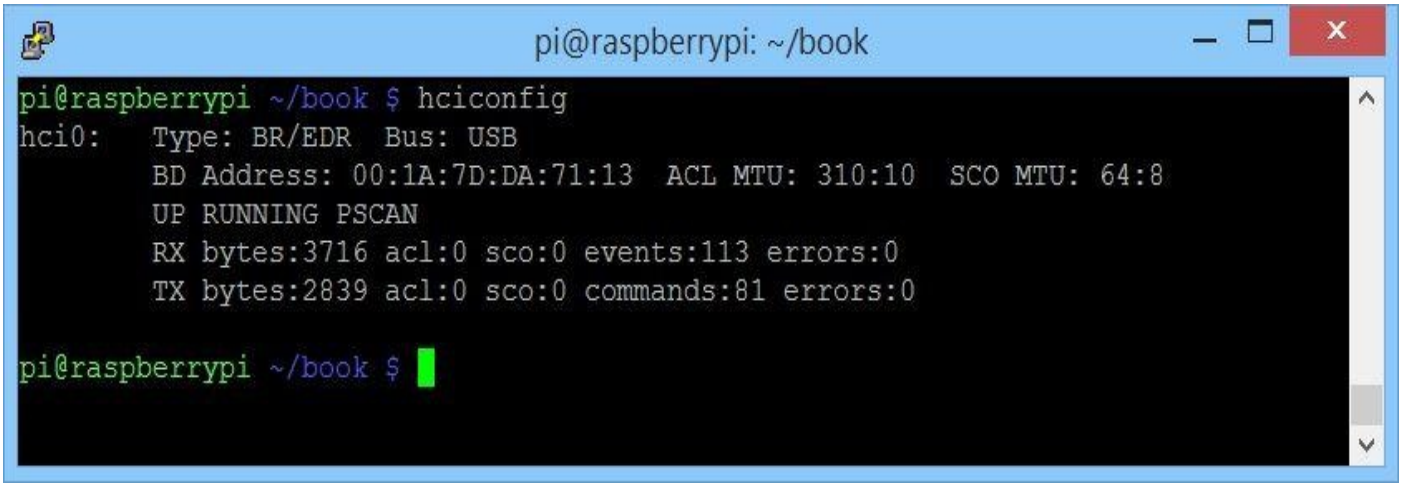


If your bluetooth device is down, you can turn on it using hciconfig up command.

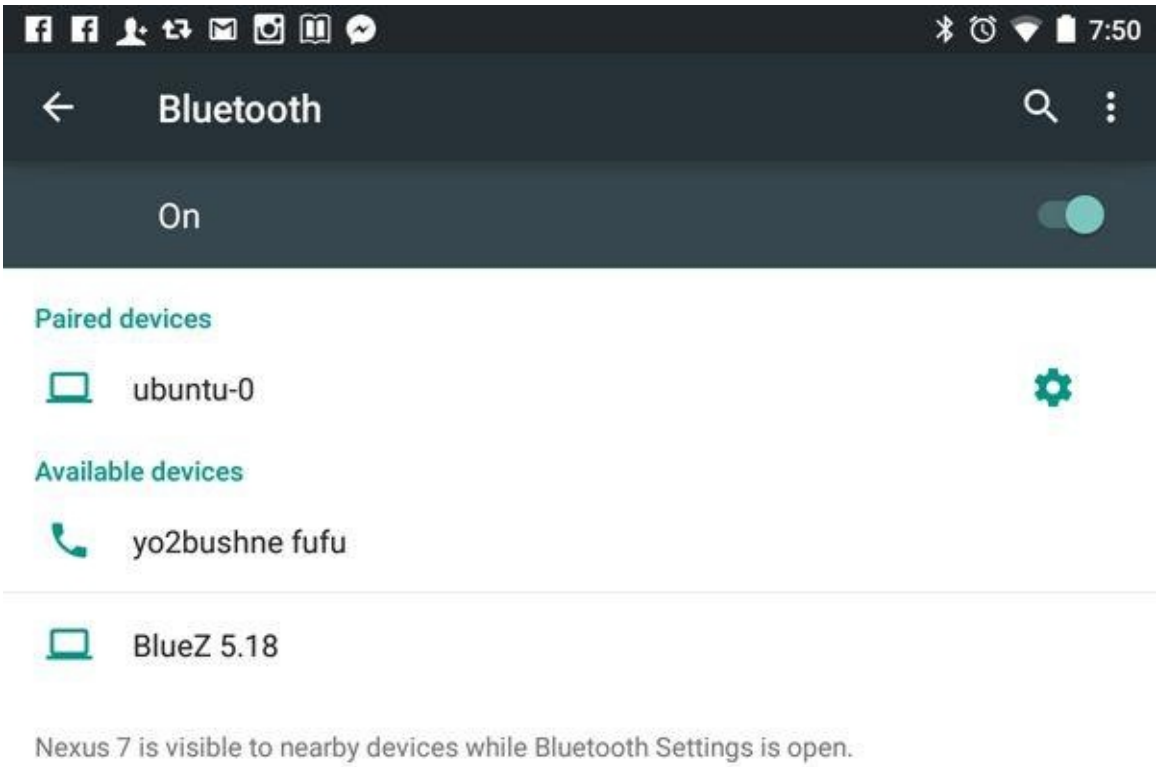
```
$ sudo hciconfig hci0 up
```

After that, you can verify it again.

```
$ hciconfig
```



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.

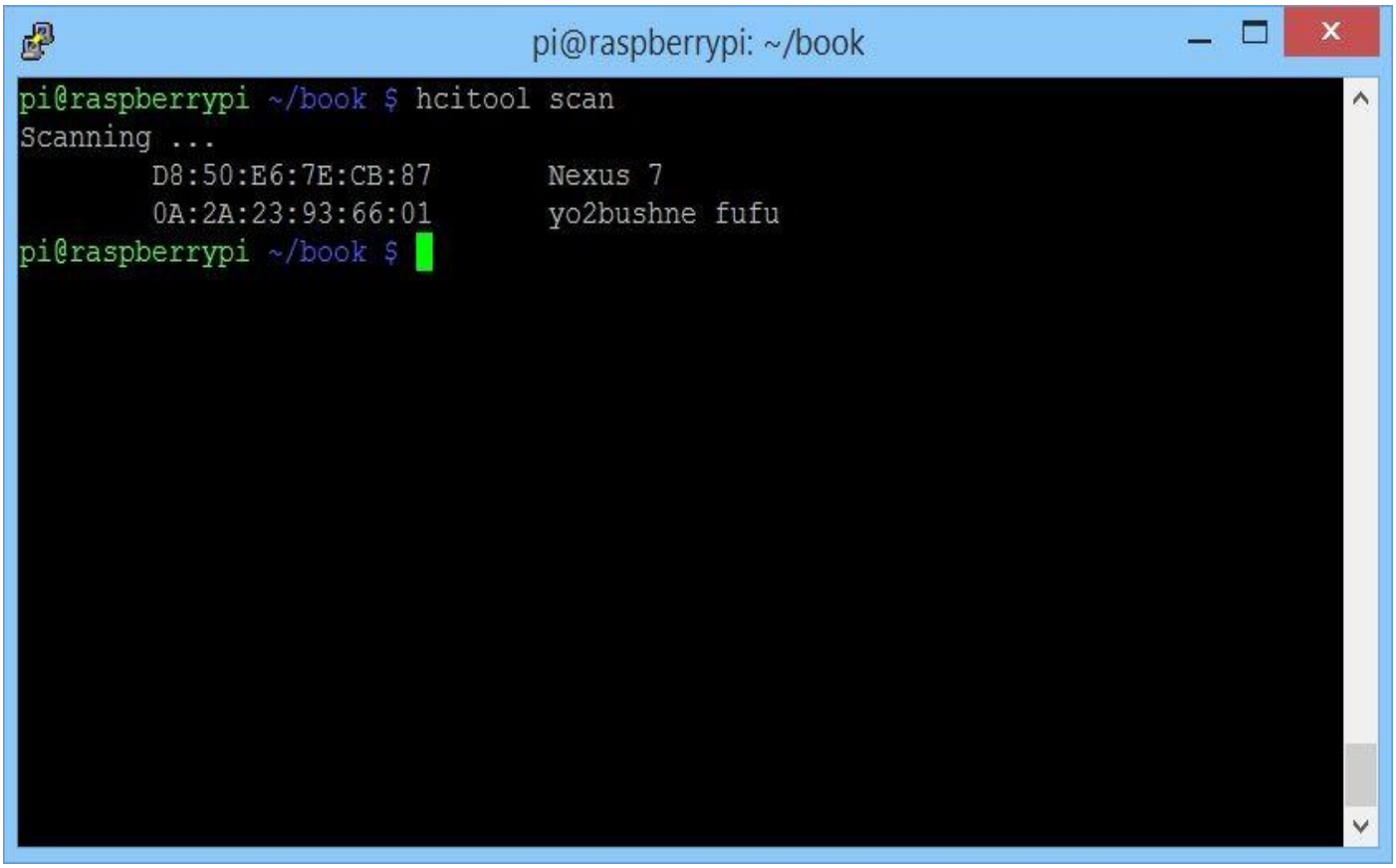
4.3 Pairing Bluetooth Devices

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
```

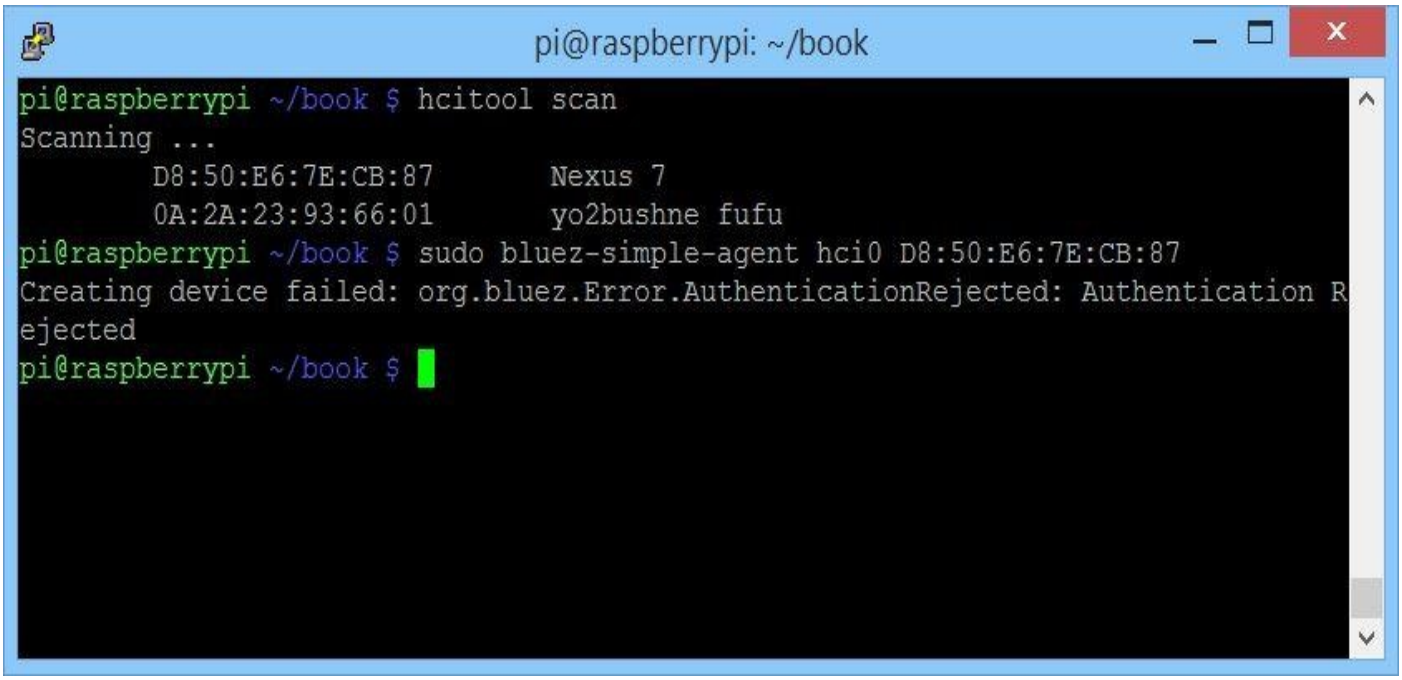
If success, you can see bluetooth devices and their address.



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>
```

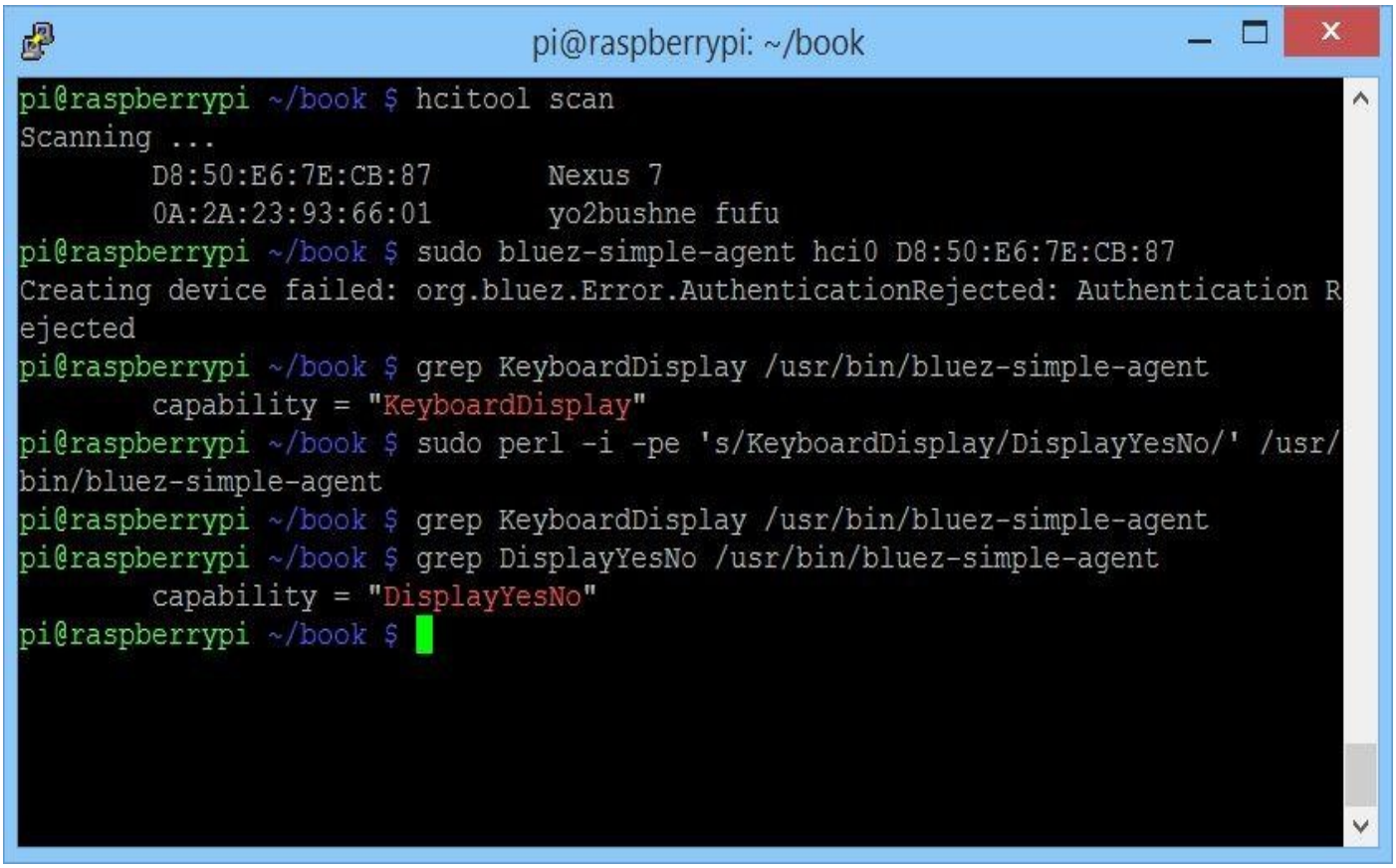
Change <target_address> value for bluetooth target address.



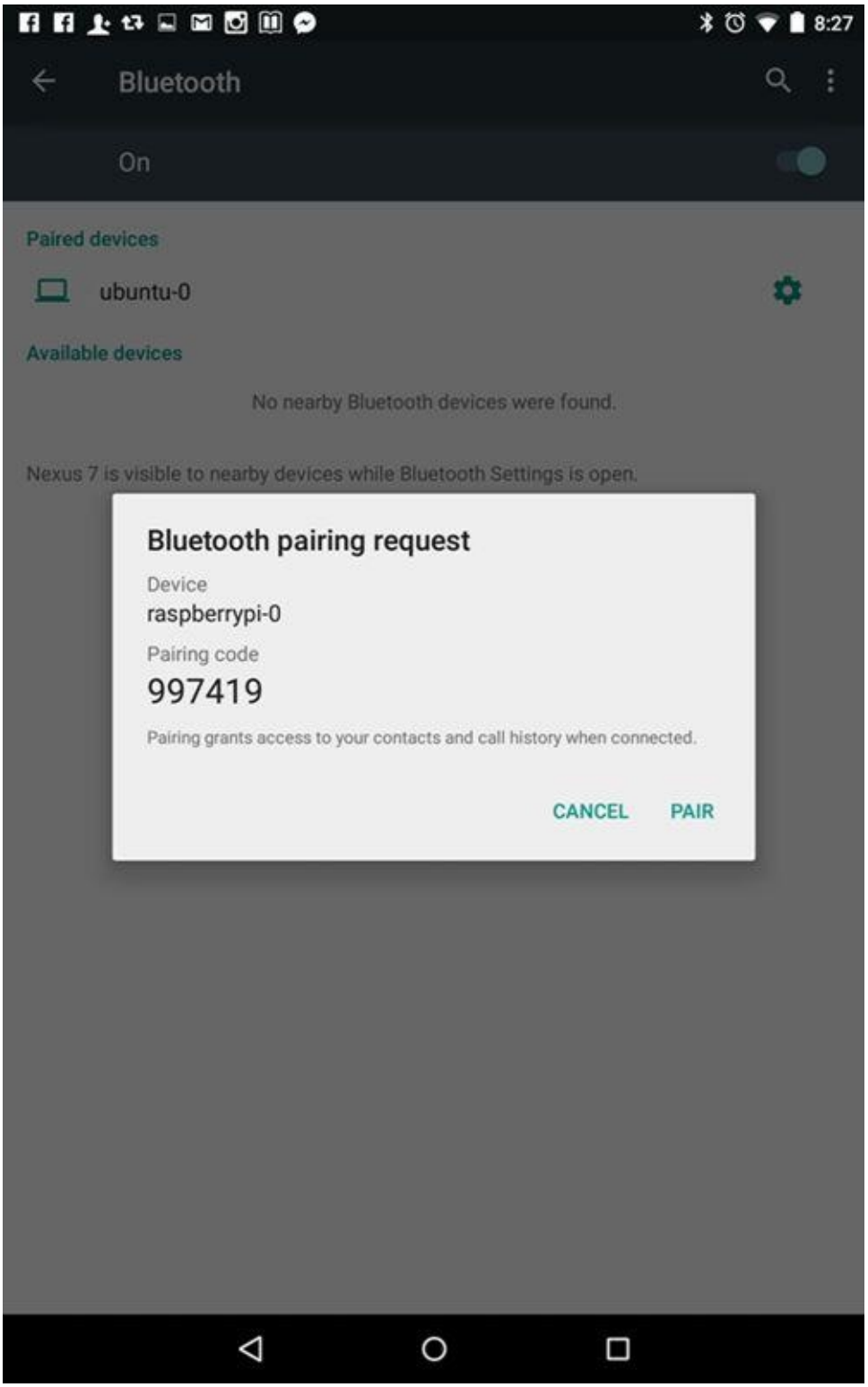
If you can get error message, shown in Figure above. It happens because we cannot handle authentication dialog. We must enable it. I found an article which solved this issue. You can read it on http://www.wolfteck.com/projects/raspi/iphone/ .

Type these commands to enable authentication dialog.

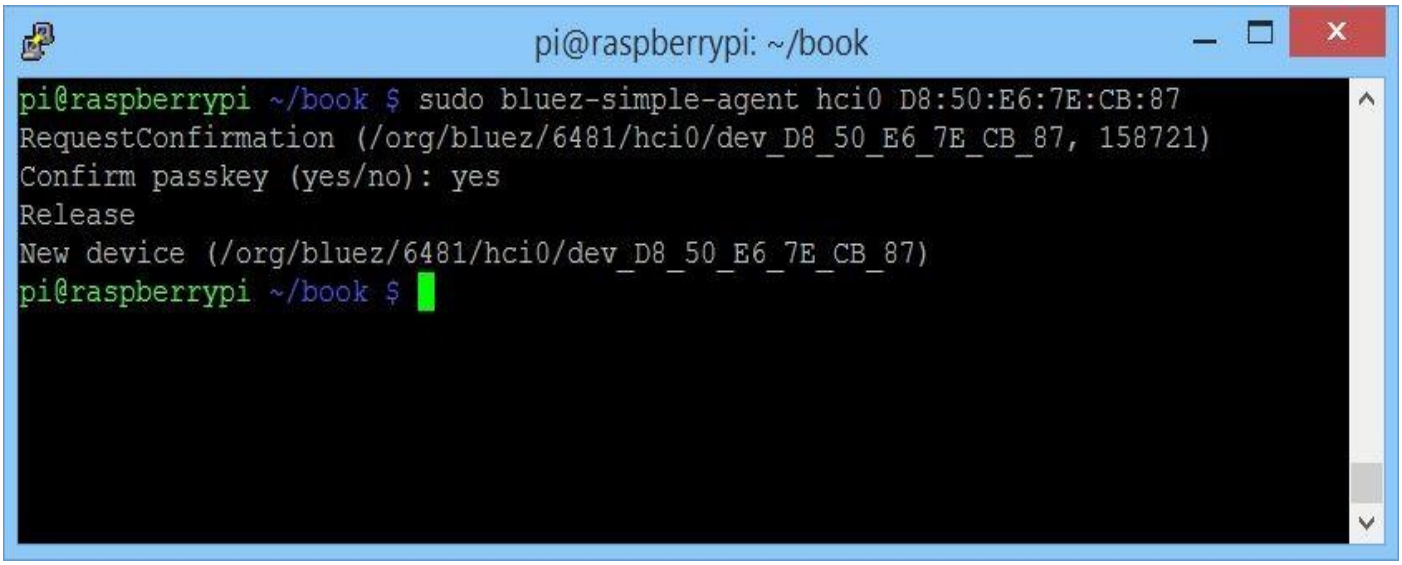
```
$ grep KeyboardDisplay /usr/bin/bluez-simple-agent
$ sudo perl -i -pe 's/KeyboardDisplay/DisplayYesNo/' /usr/bin/bluez-simple-agent
$ grep DisplayYesNo /usr/bin/bluez-simple-agent
```



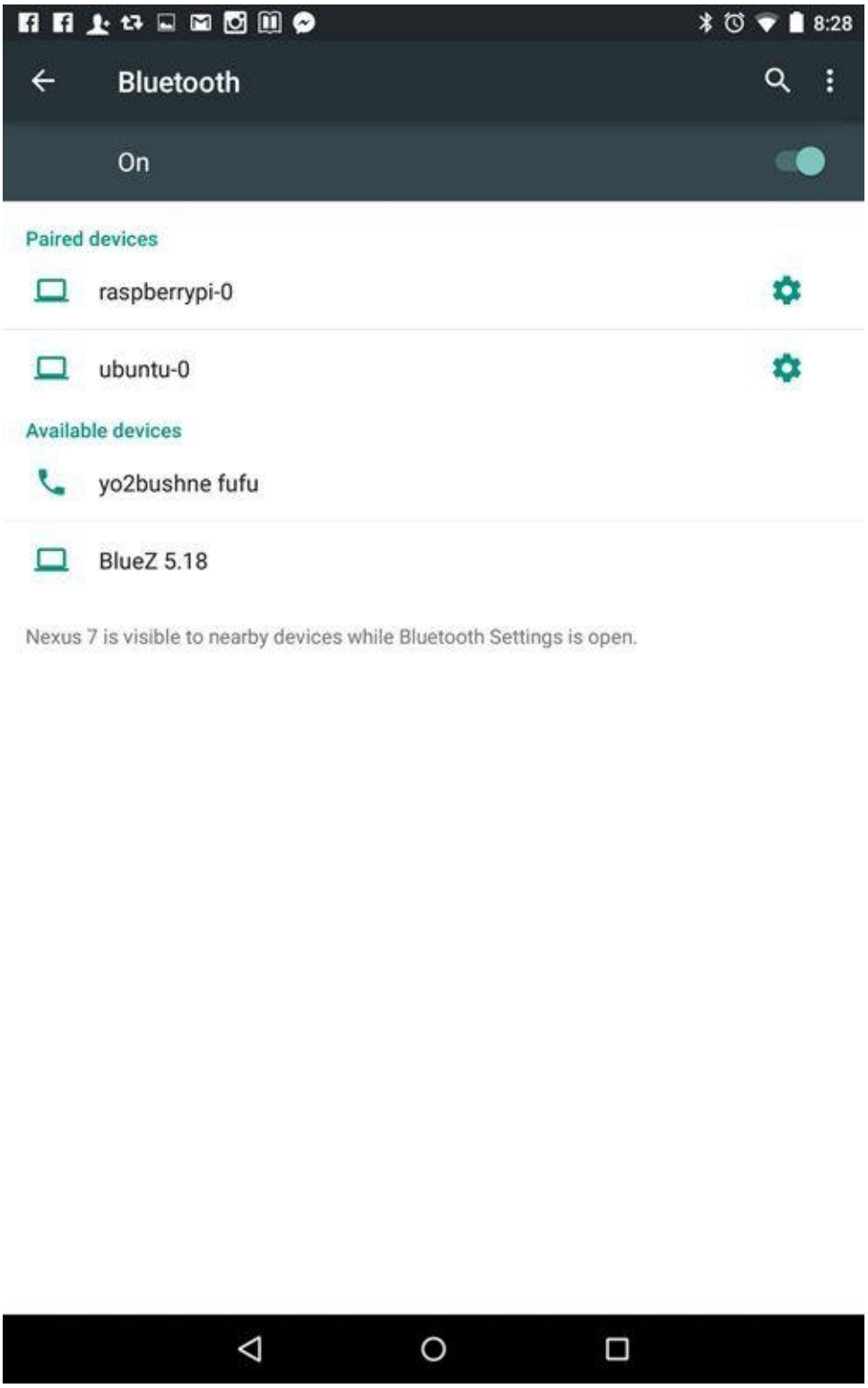
Now you can try to pair your bluetooth. For instance, I connect my bluetooth to Nexus 7 2013. In Nexus, I got confirmation, shown in Figure below. Press PAIR.



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



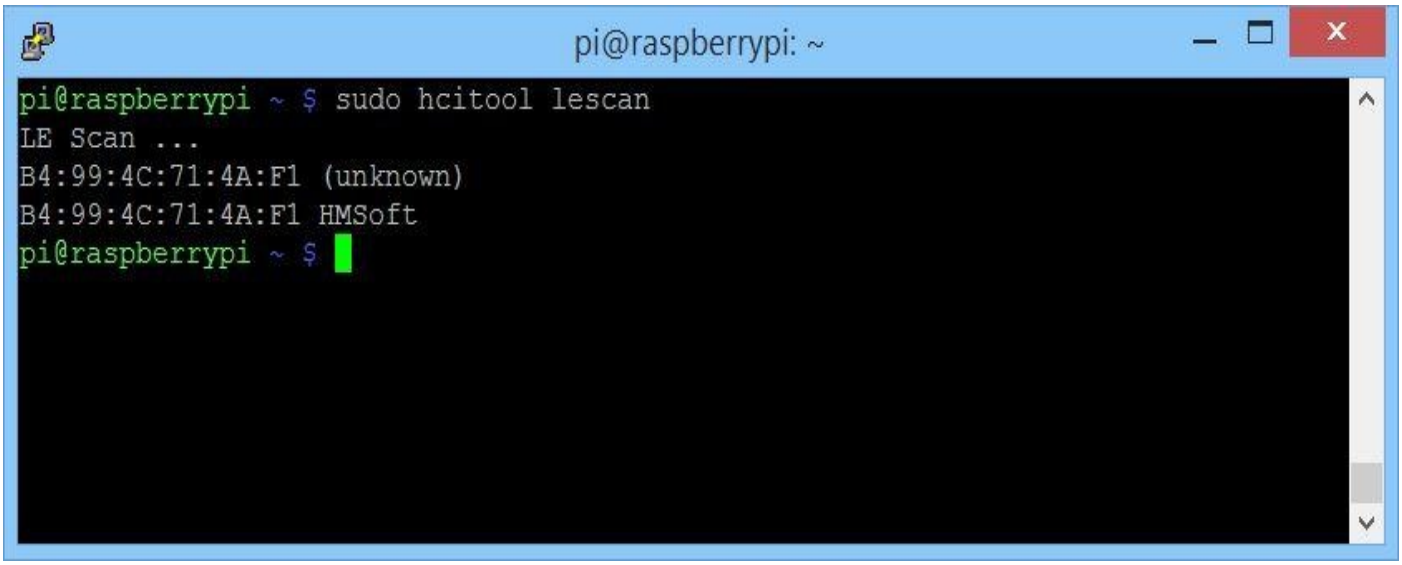
If success, your Raspberry Pi bluetooth will be paired to bluetooth target device.



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.



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
```

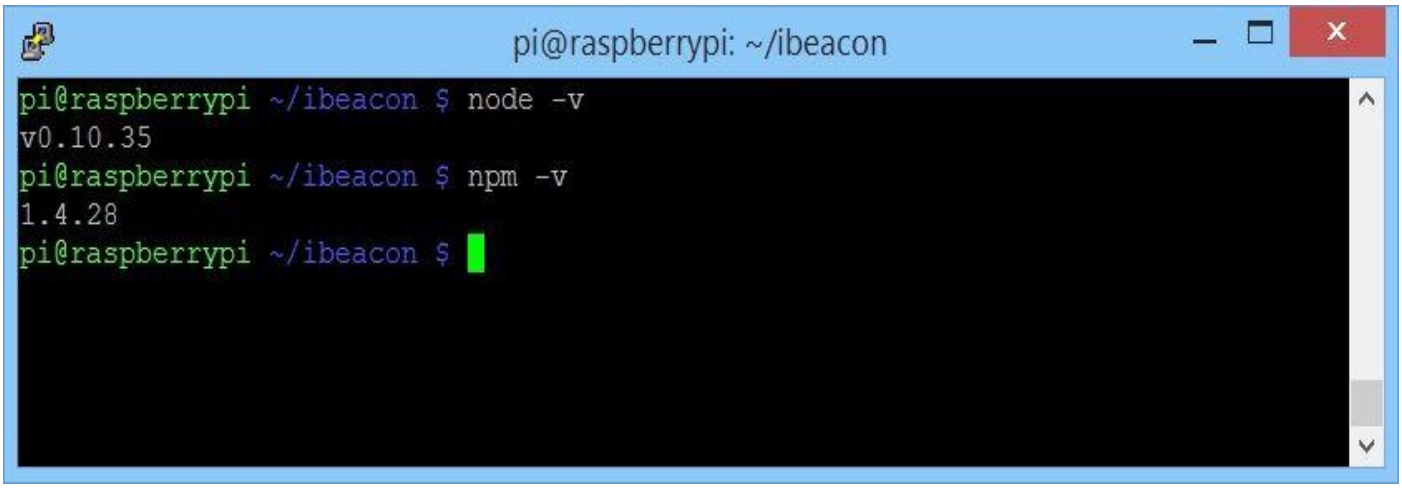
You can install Node.js from compiled binary for ARM. You can type these commands.

```
$ 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.



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.

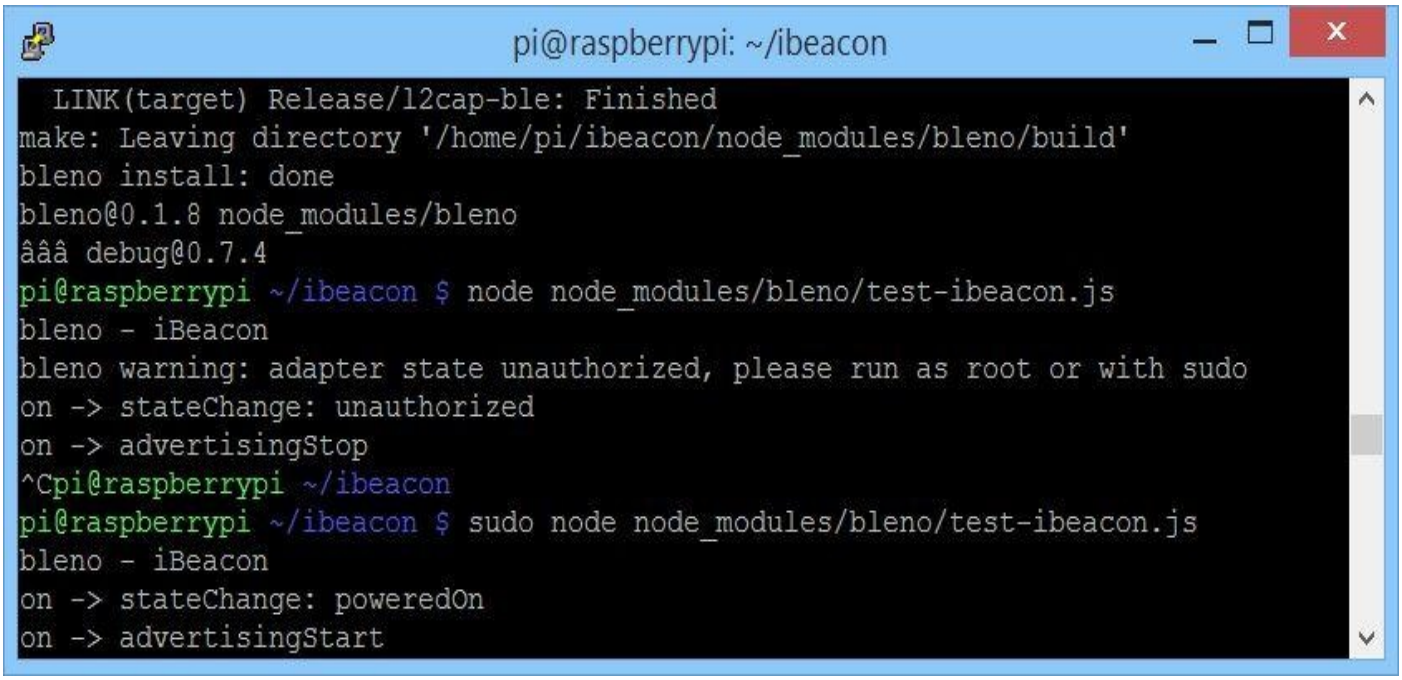
```
$ sudo apt-get install bluetooth bluez-utils libbluetooth-dev
```

Now you can install Noble and Bleno libraries.

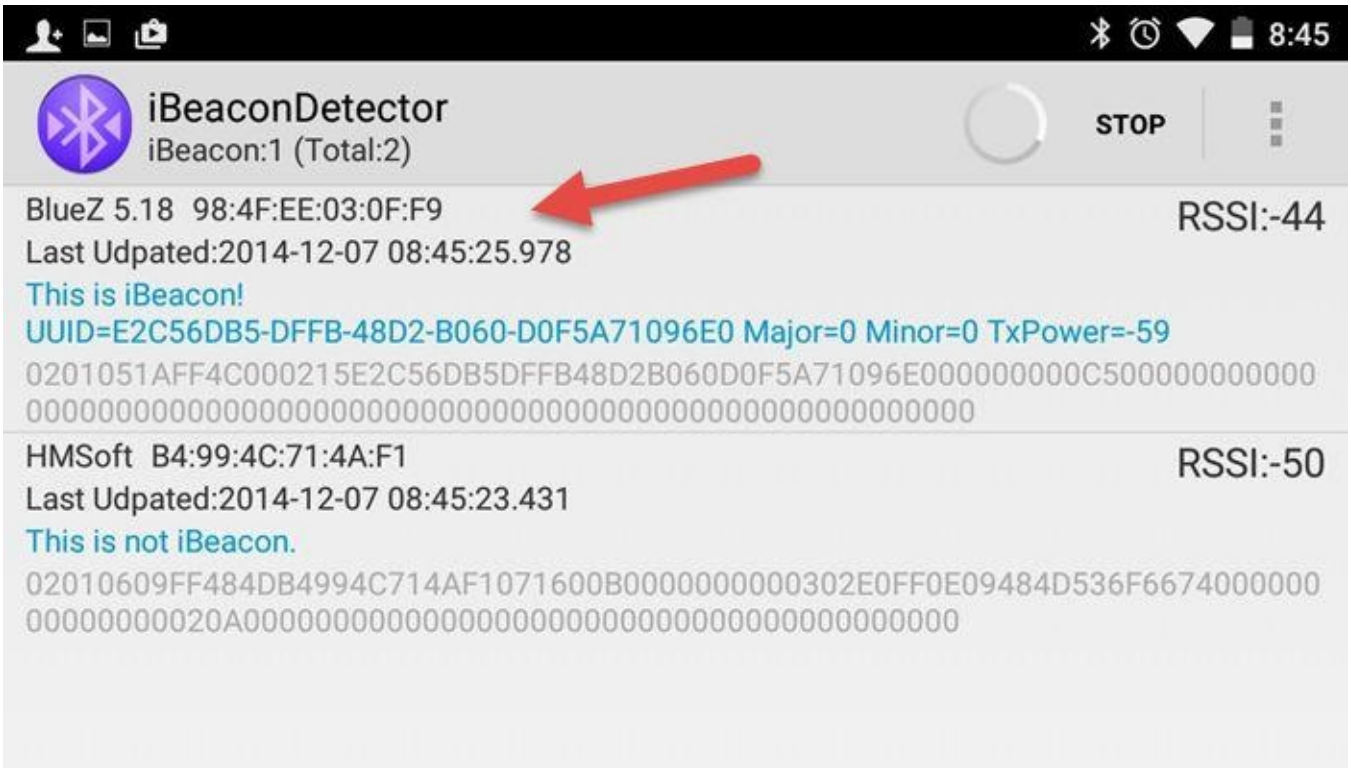
```
$ mkdir ibeacon
$ cd ibeacon/
$ sudo npm install -g async
$ sudo npm install noble
$ sudo npm install bleno
```

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

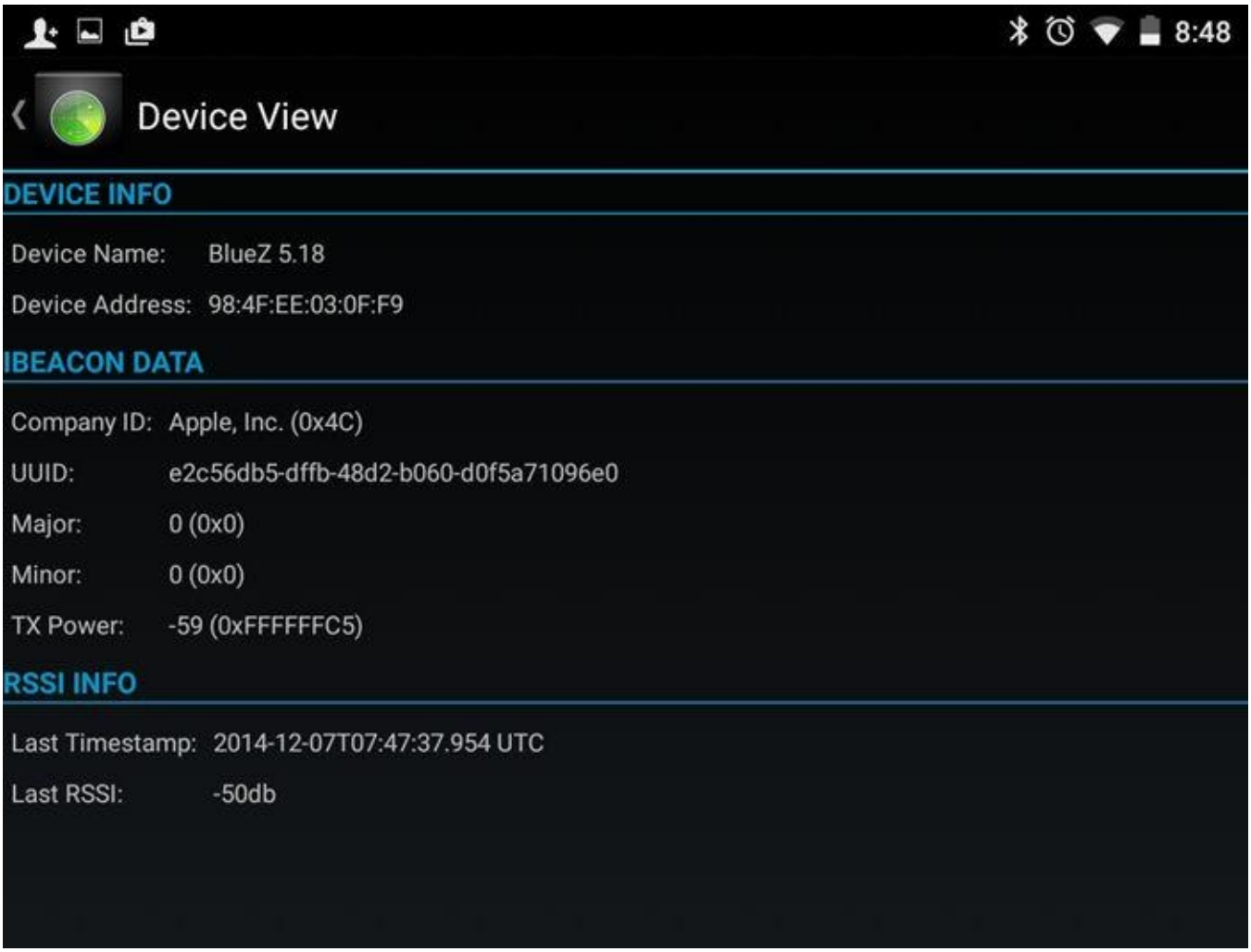
```
$ sudo node node_modules/bleno/test-ibeacon.js
```



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-dffb-48d2-b060-d0f5a71096e0
Major: 0 (0x0)
Minor: 0 (0x0)
TX Power: -59 (0xFFFFFFC5)
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 . 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 fromSeeedstudio, 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:

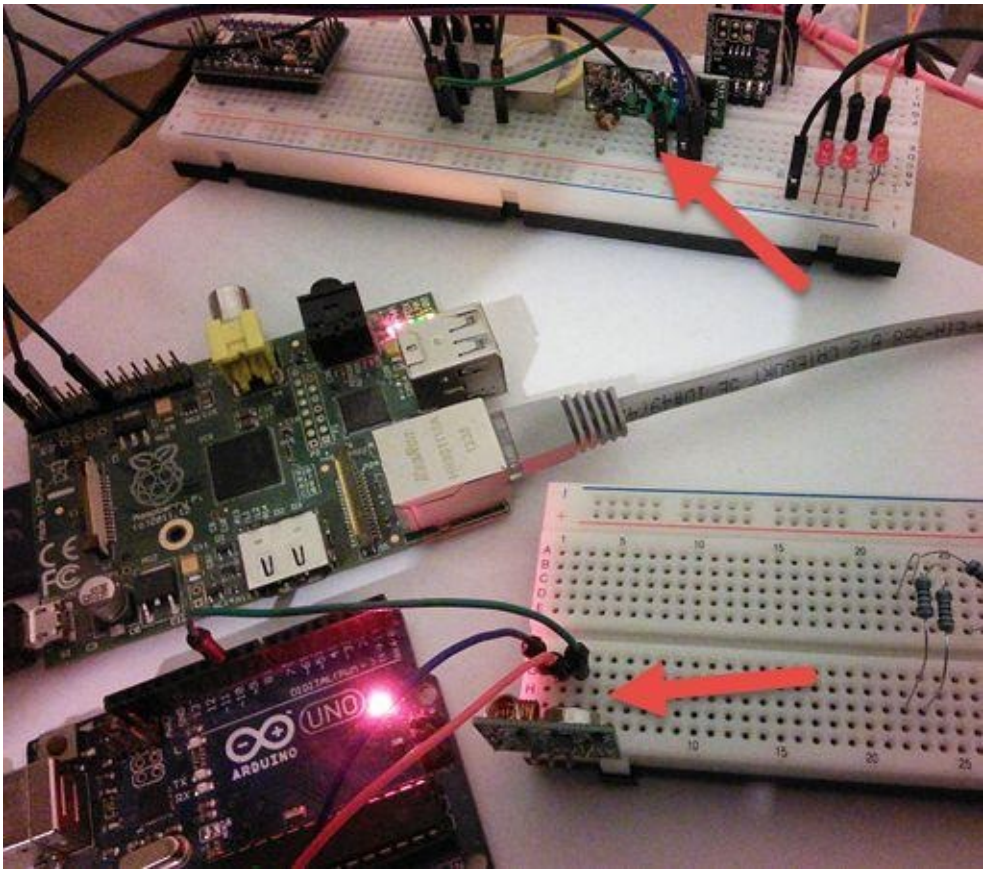
- GND Arduino is connected to GND RF transmitter
- VCC +5V Arduino is connected to +VCC RF transmitter
- Digital 10 (PWM) Arduino is connected to Data RF transmitter

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

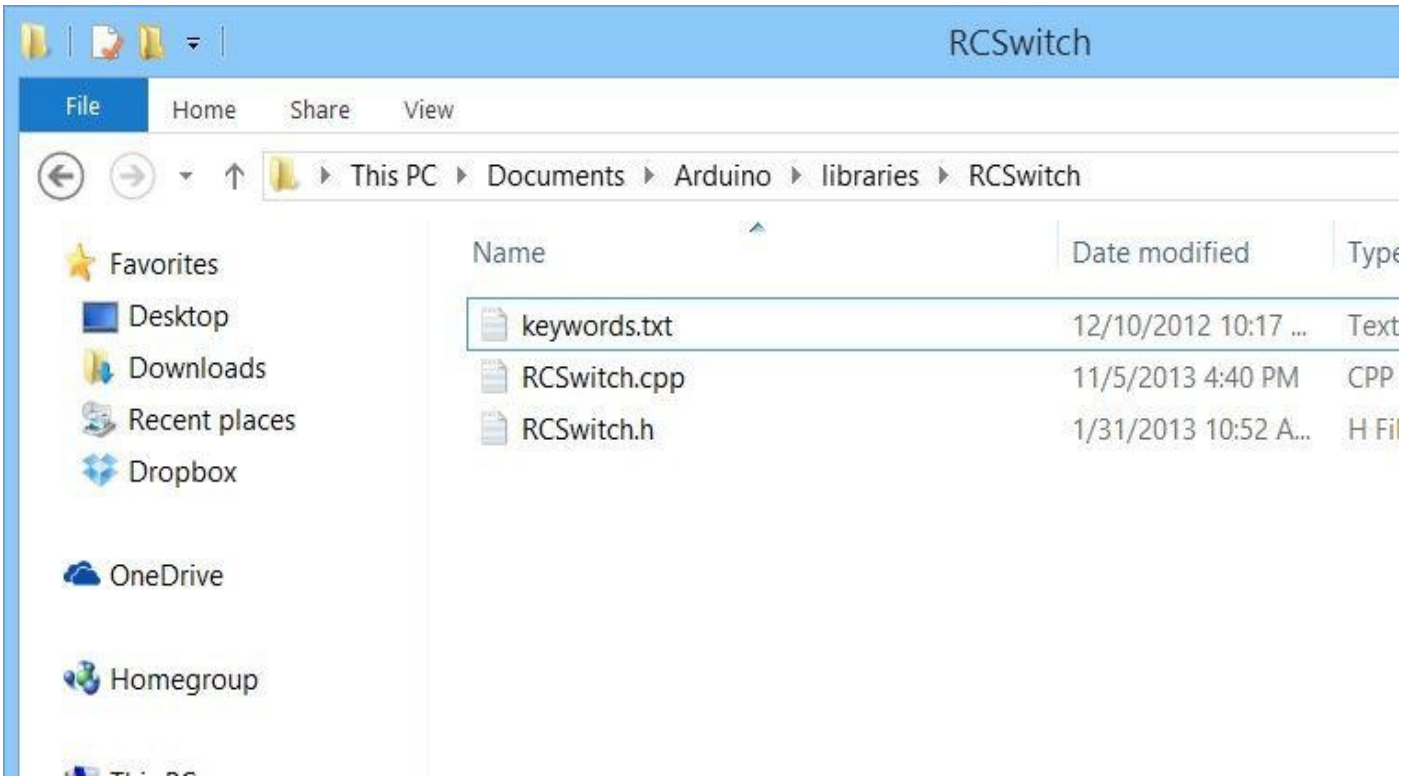
GPIO2 Raspberry Pi is based on WiringPi mapping (see section 5.2.3). If you see Raspberry Pi GPIO (section 1.6), GPIO2 is located on GPIO21/GPIO27 Raspberry Pi.

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.

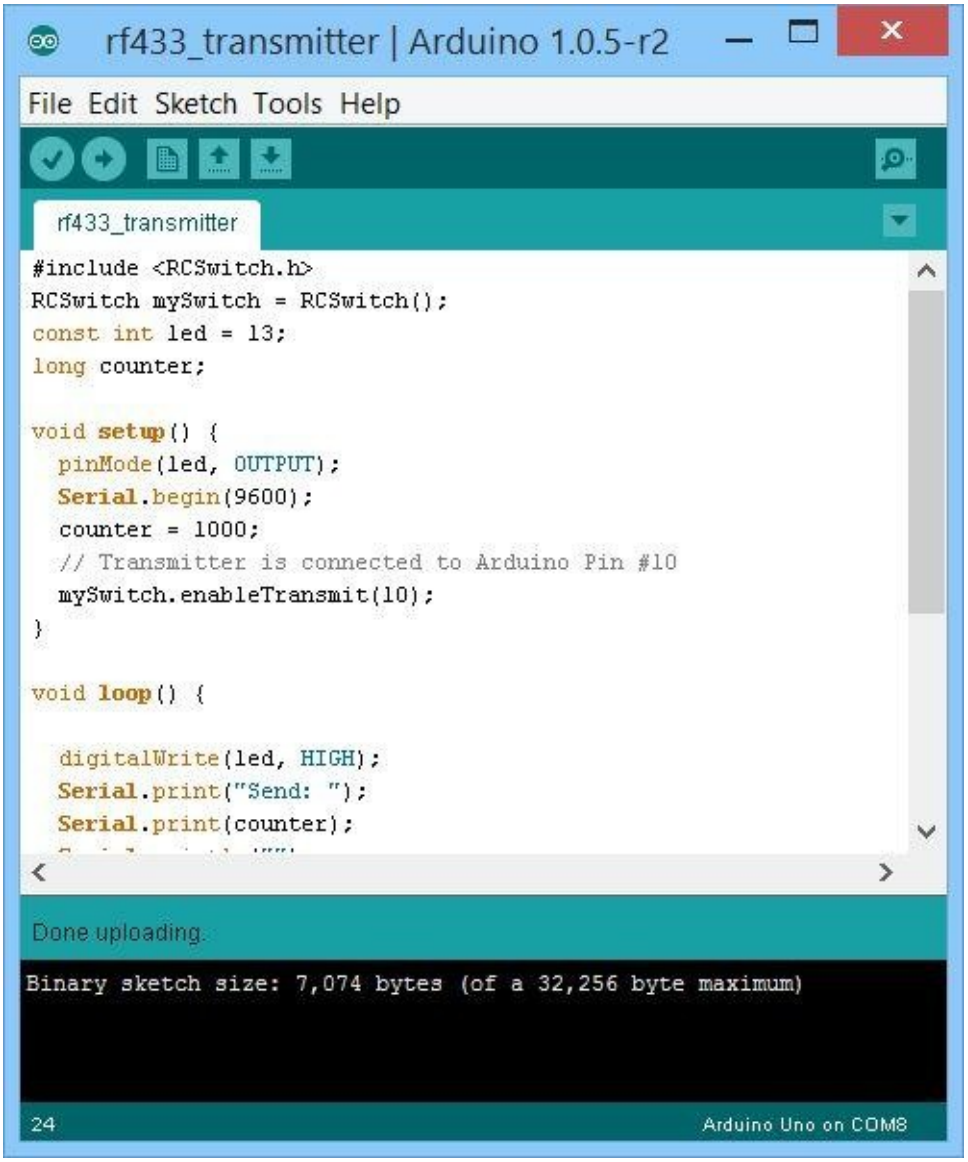


You also can import **RCSwitch.zip** from menu Sketch-> Import library.

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;
```

```
}
```



Explanation:

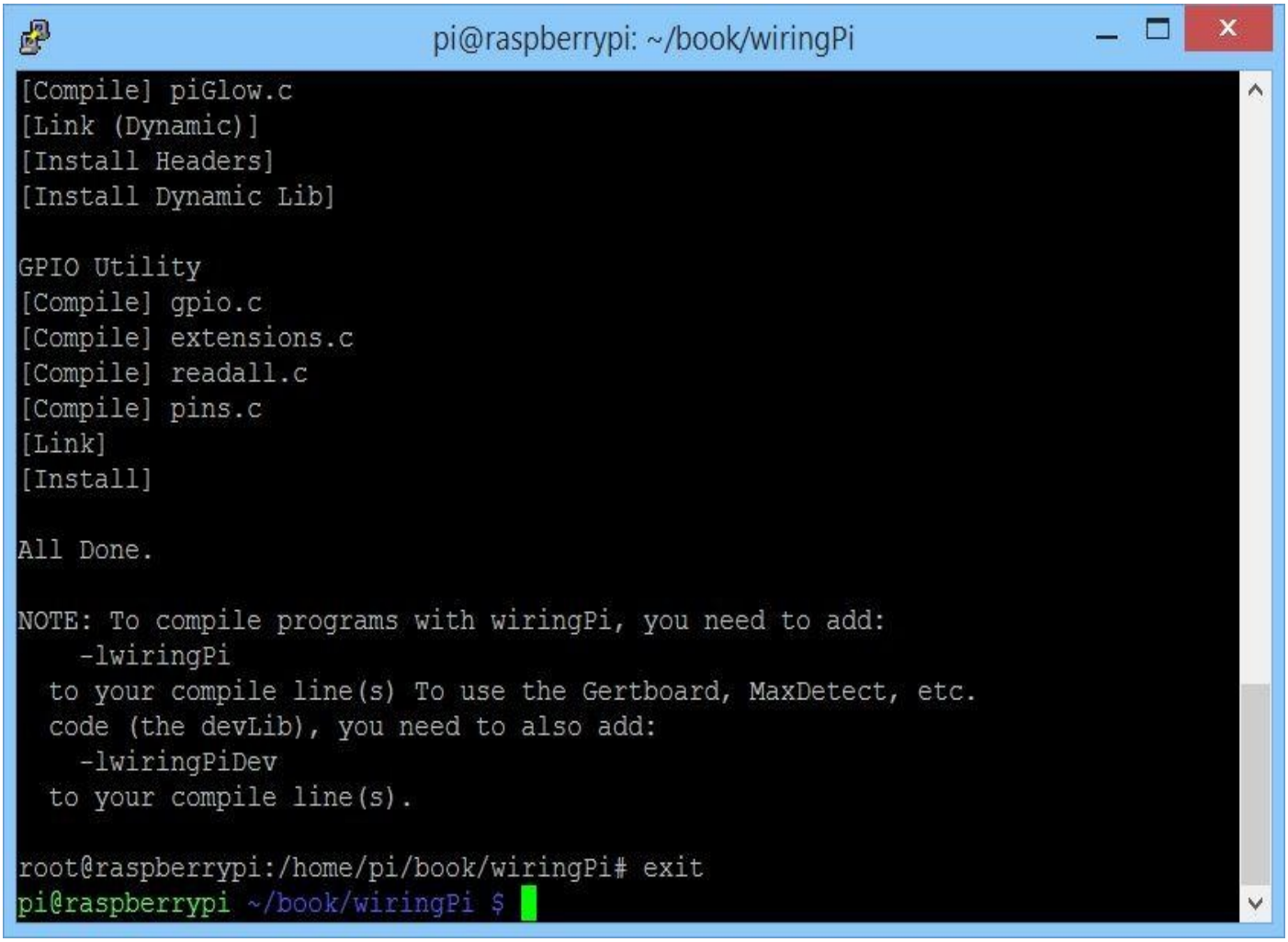
- In setup(), we activate Serial port and LED 13
- Initialize counter = 1000
- Enabling rc-switch to enable to transmitt counter data
- In loop(), turn on LED 13 and then send data using send() from rc-switch library
- counter value is be increased. If counter value is more than 1100, we set counter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

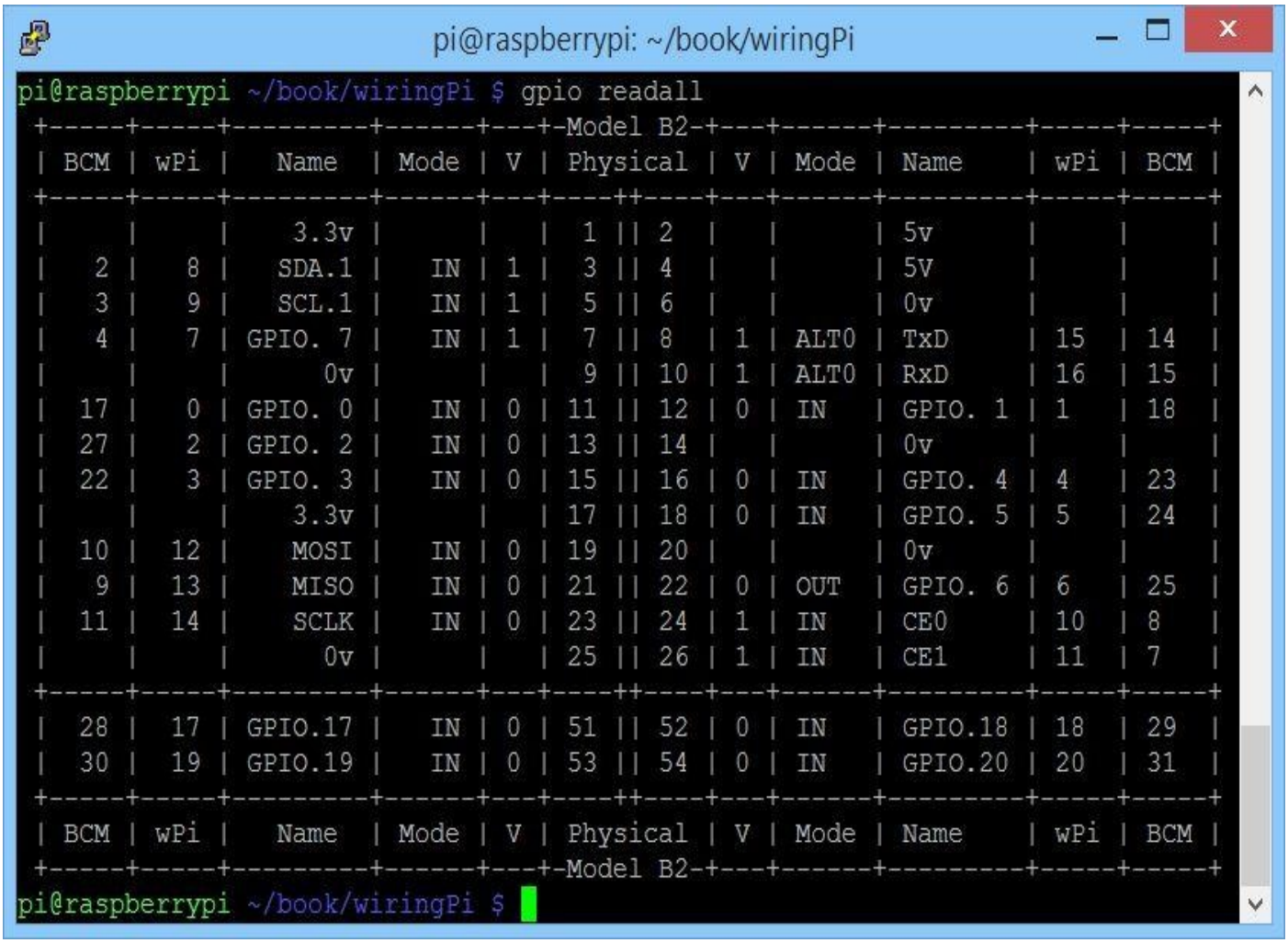
```
$ git clone git://git.drogon.net/wiringPi
$ cd wiringPi/
$ sudo su
$ ./build
```



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.



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);
    }
```

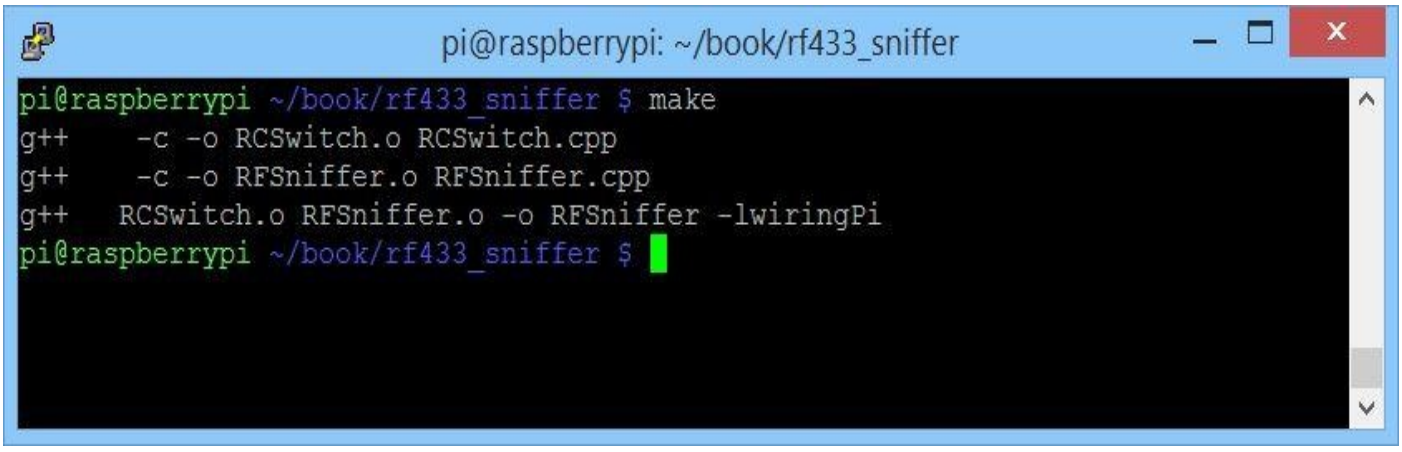
Save this code.

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
```



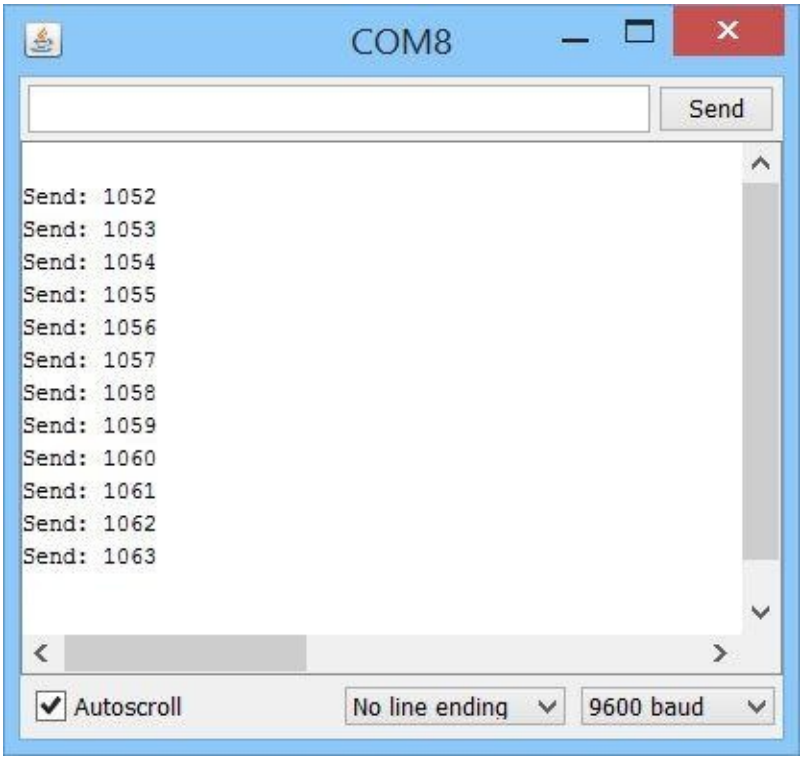
5.2.4 Testing

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.

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.

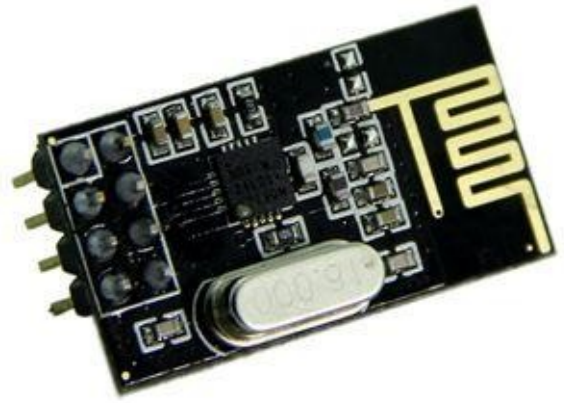
6.1 Getting Started

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

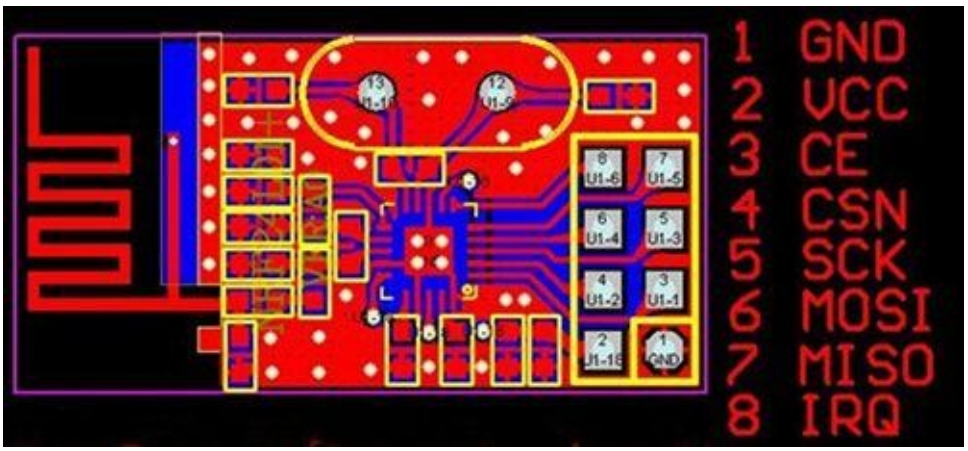
- Seeedstudio, http://www.seeedstudio.com/depot/nRF24L01Module-p-1394.html
- Sparkfun, https://www.sparkfun.com/search/results?term=nRF24L01
- eBay.com
- Amazon

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 . You can see pinout for nRF24L01/nRF24L01+ module.

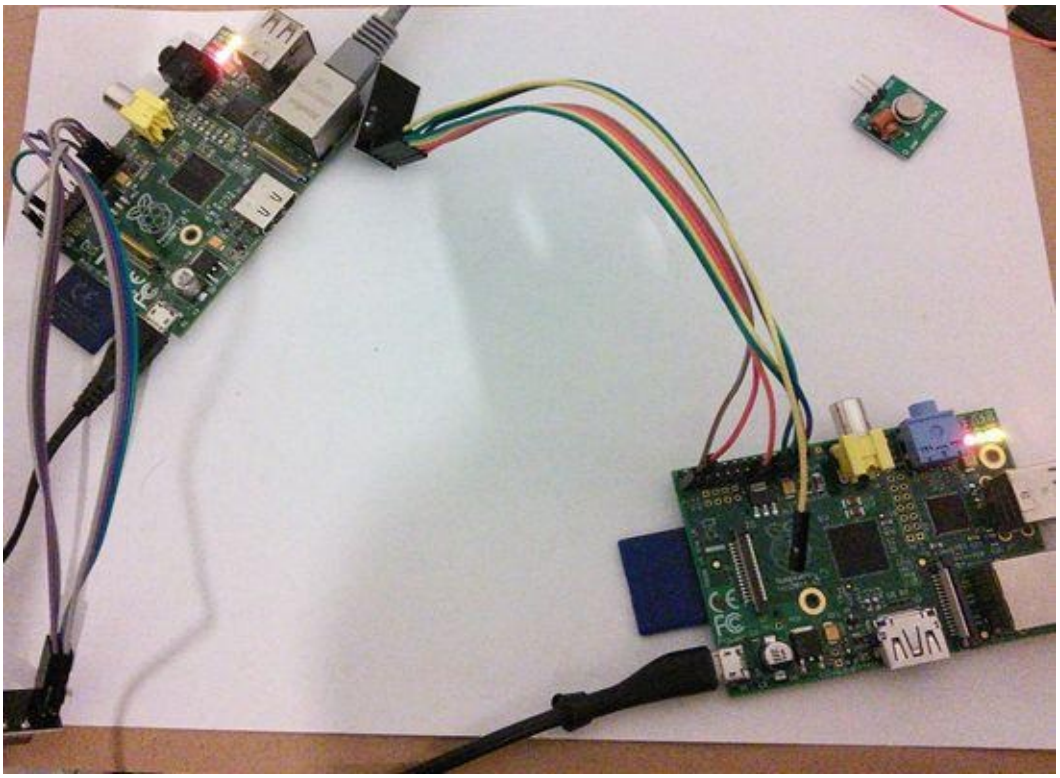
Based nRF24L01/nRF24L01+ datasheet, you can see the pinout on the following Figure. Note: It's top view!!.



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

NRF24L01	Rpi	
	Name	Physical
VCC	+3.3V	
GND	GND	
CE	GPIO3	15
CSN	CE0	24
SCK	SCLK	23
MOSI	MOSI	19
MISO	MISO	21
IRQ	Not connected	

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

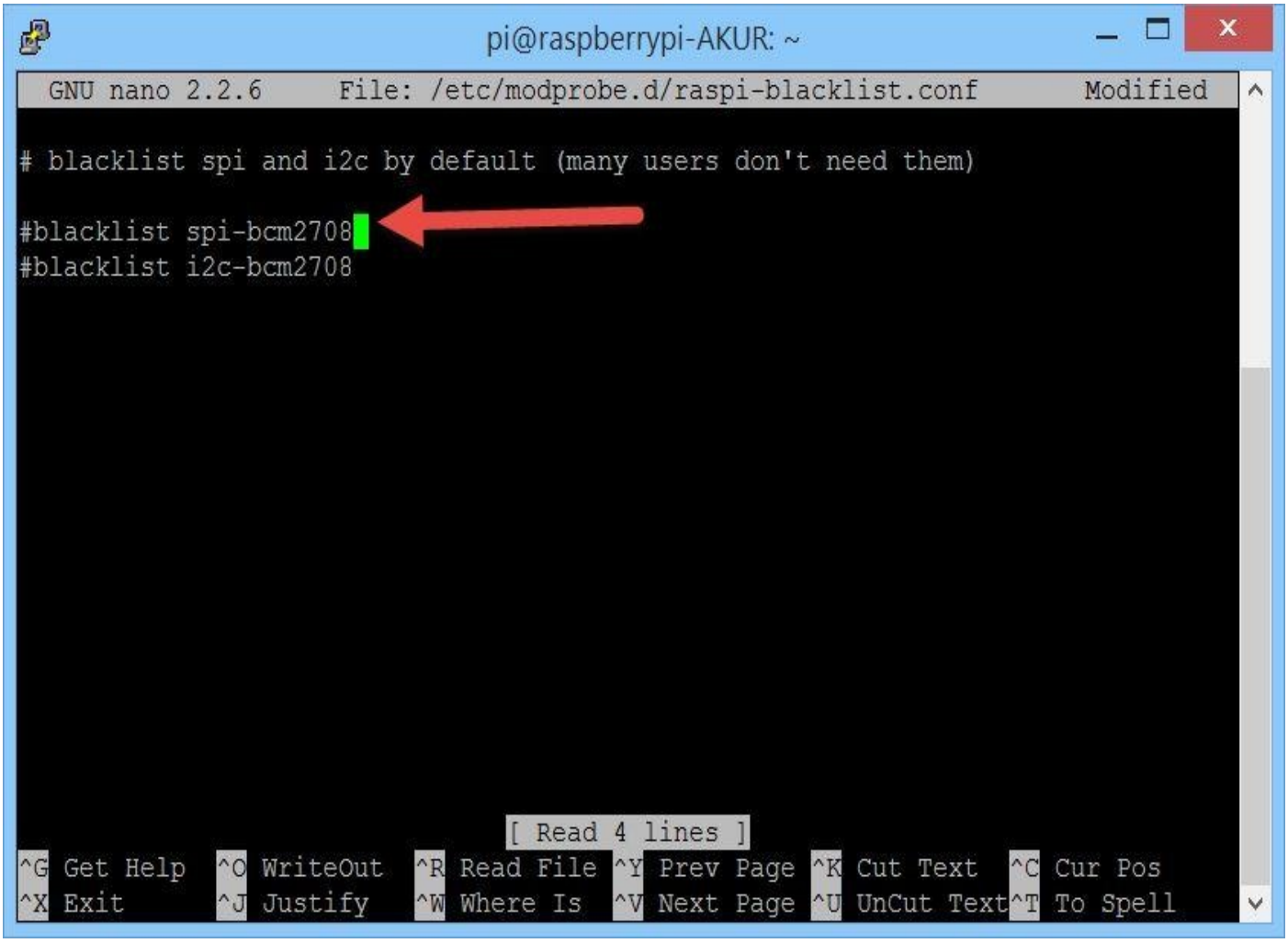


6.4 Configuring SPI on Raspberry Pi

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.



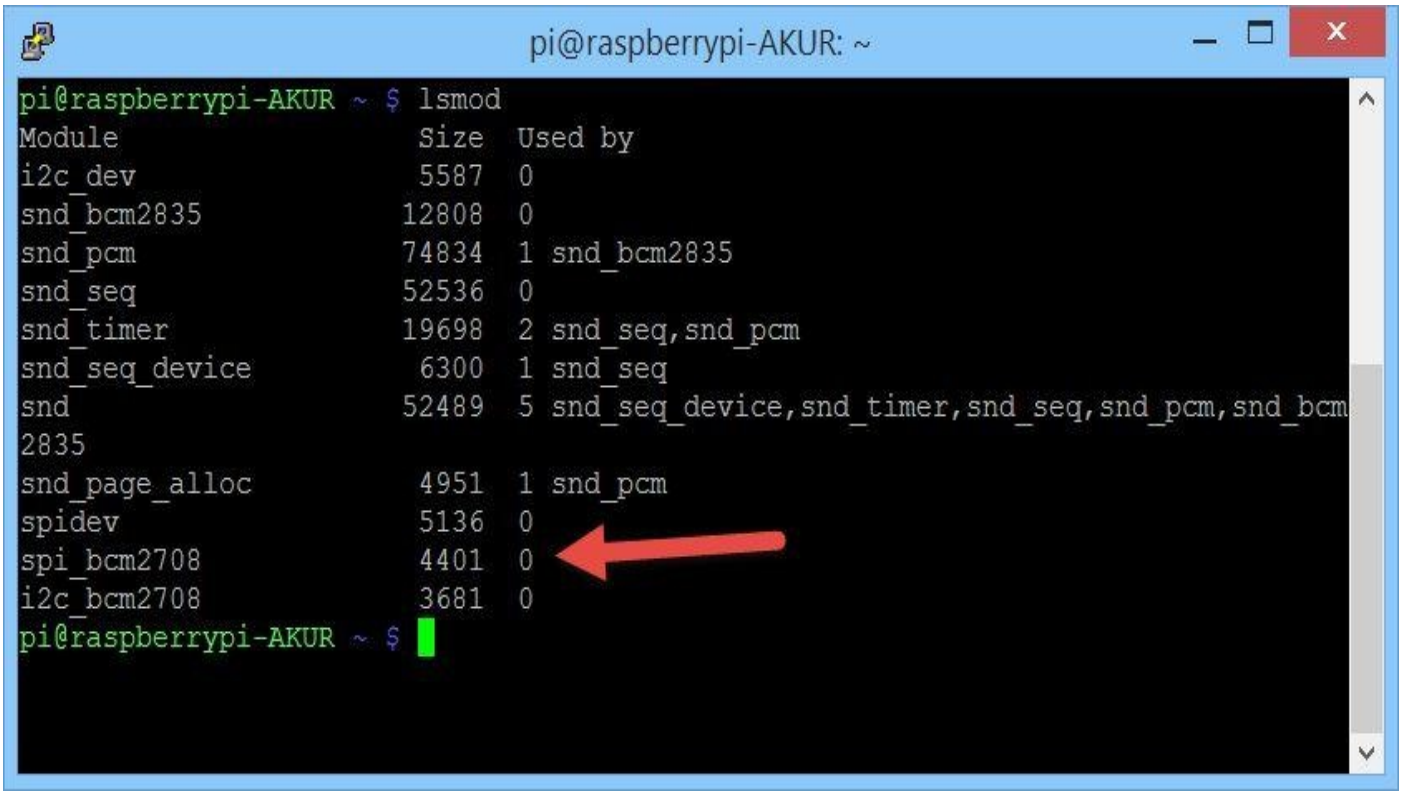
Save this changed code and close this file.

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
```

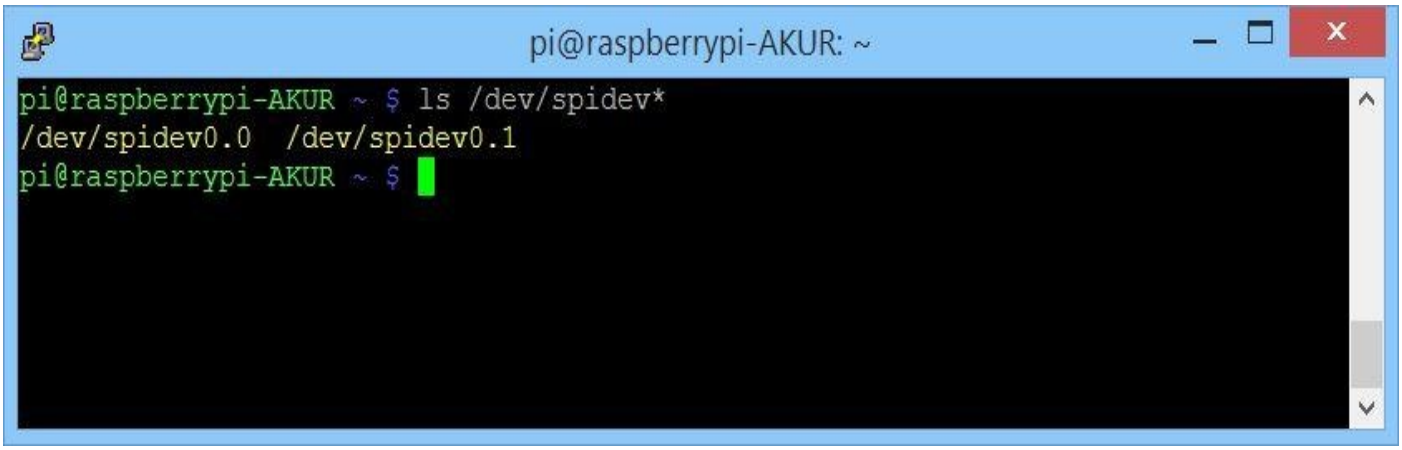
The following is response output.



You can see spi_bcm2708.

You also can verify SPI bus using the following command.

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



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
```

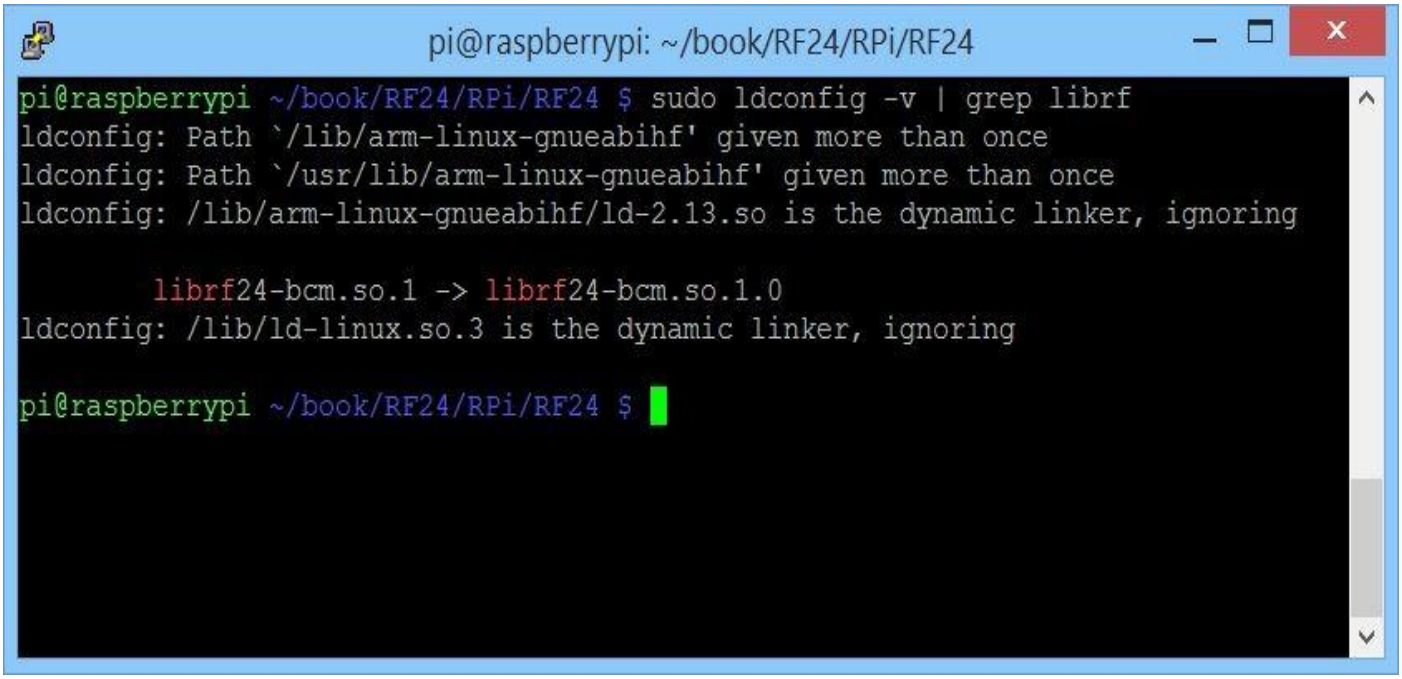
If you don't install git yet, you can install git.

```
$ 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.



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

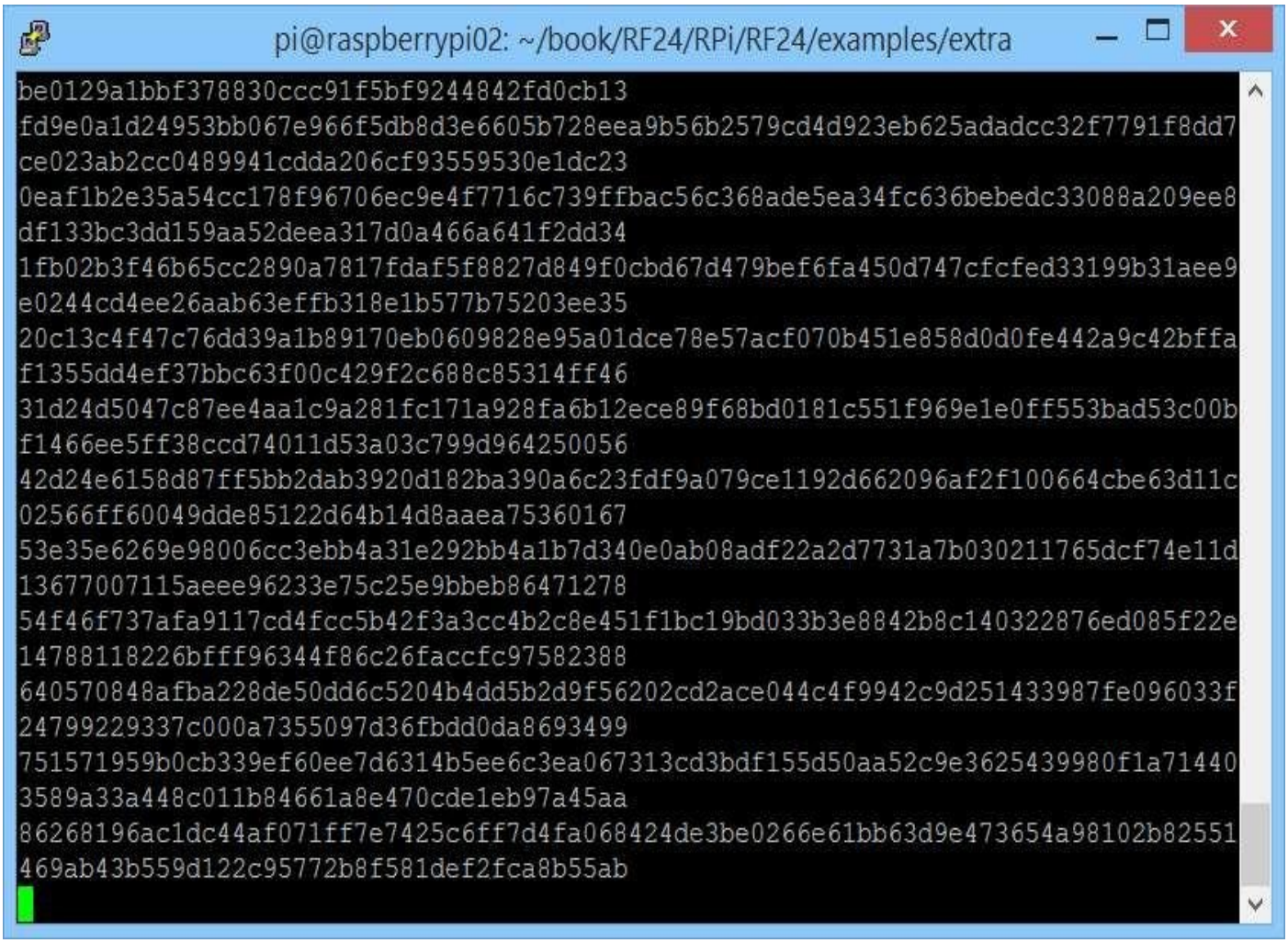
The first demo is scanner application. This app will sniff data from nRF24L01 module.

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.



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.

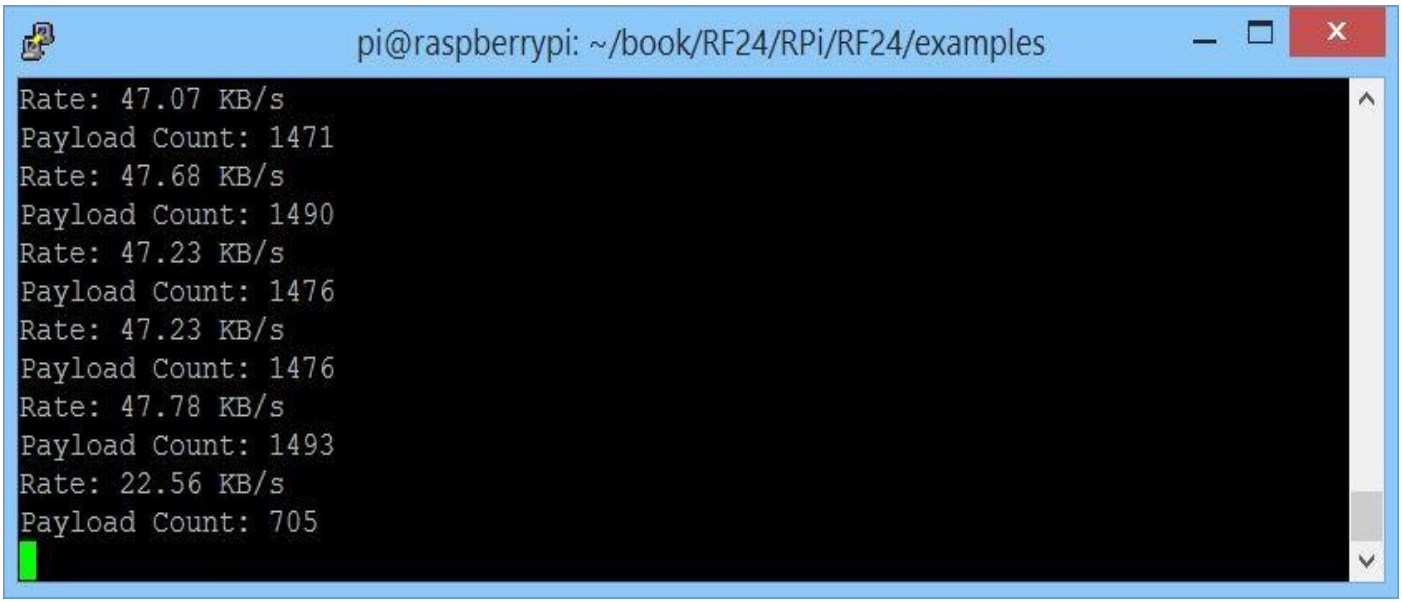
Each program can be run by typing this command.

```
$ sudo ./transfer
```

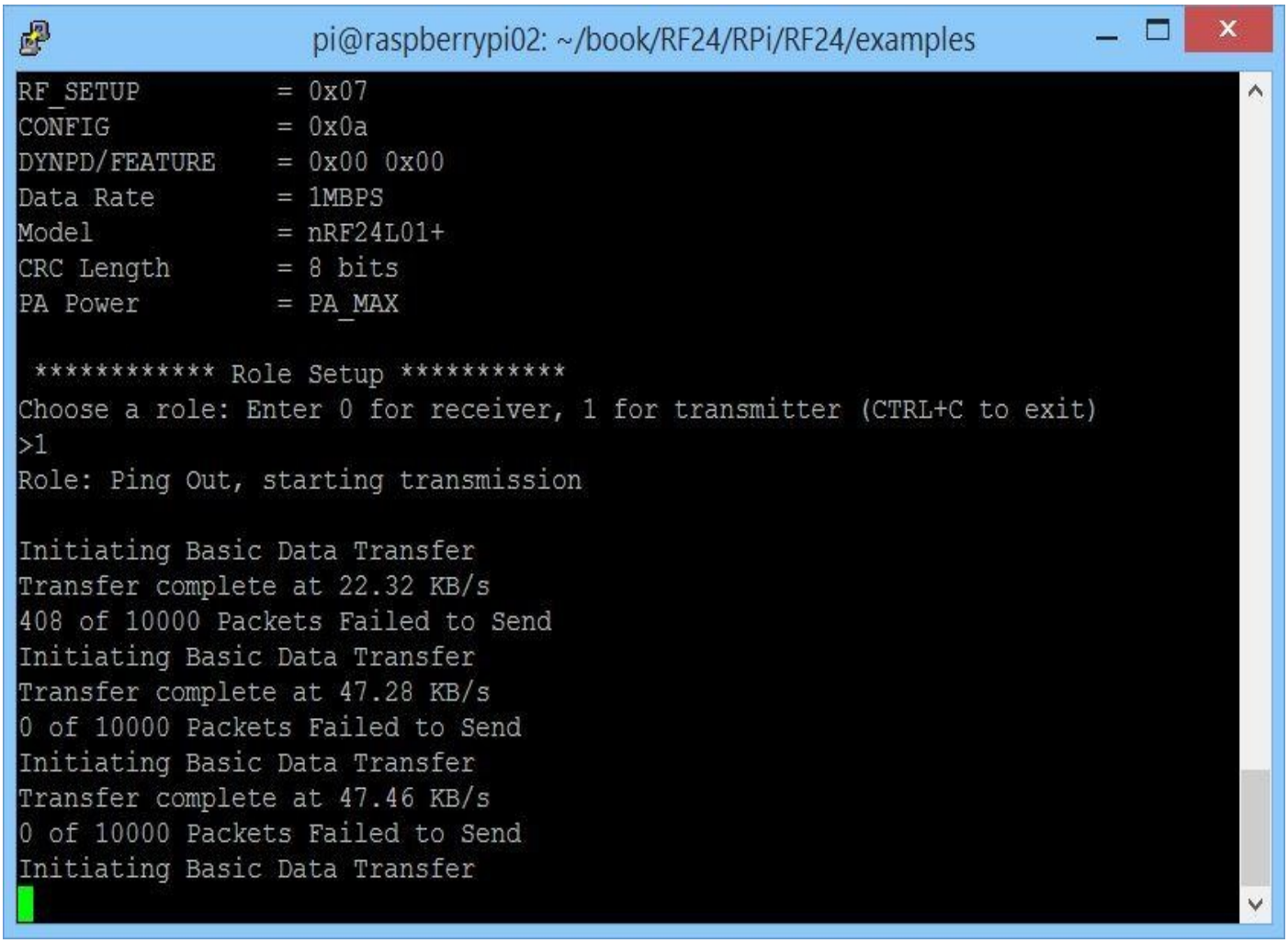
Note: This demo is located on /examples/extra/ folder.

Fill 0 for receiver role and 1 for transmitter role.

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



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



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 onhttp://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-Area-Networks/dp/073816285X/ .

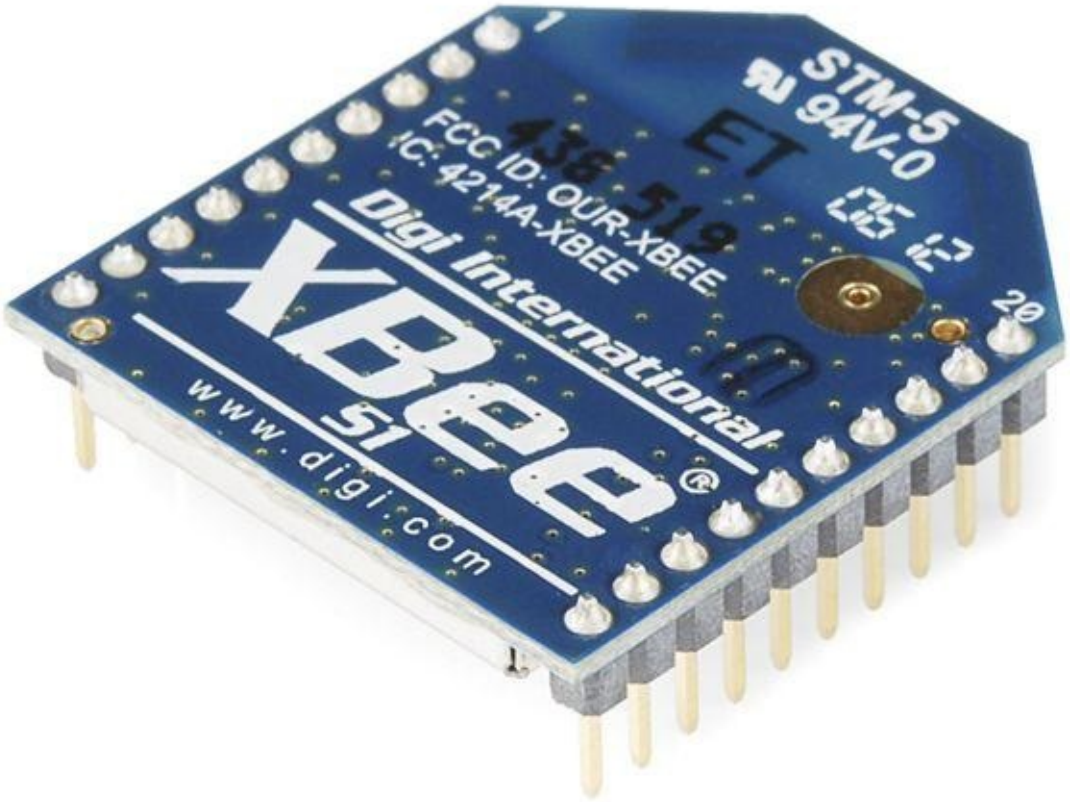
On the next section, we will implement IEEE 802.15.4 using XBee IEEE 802.15.4.

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 .

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
- 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

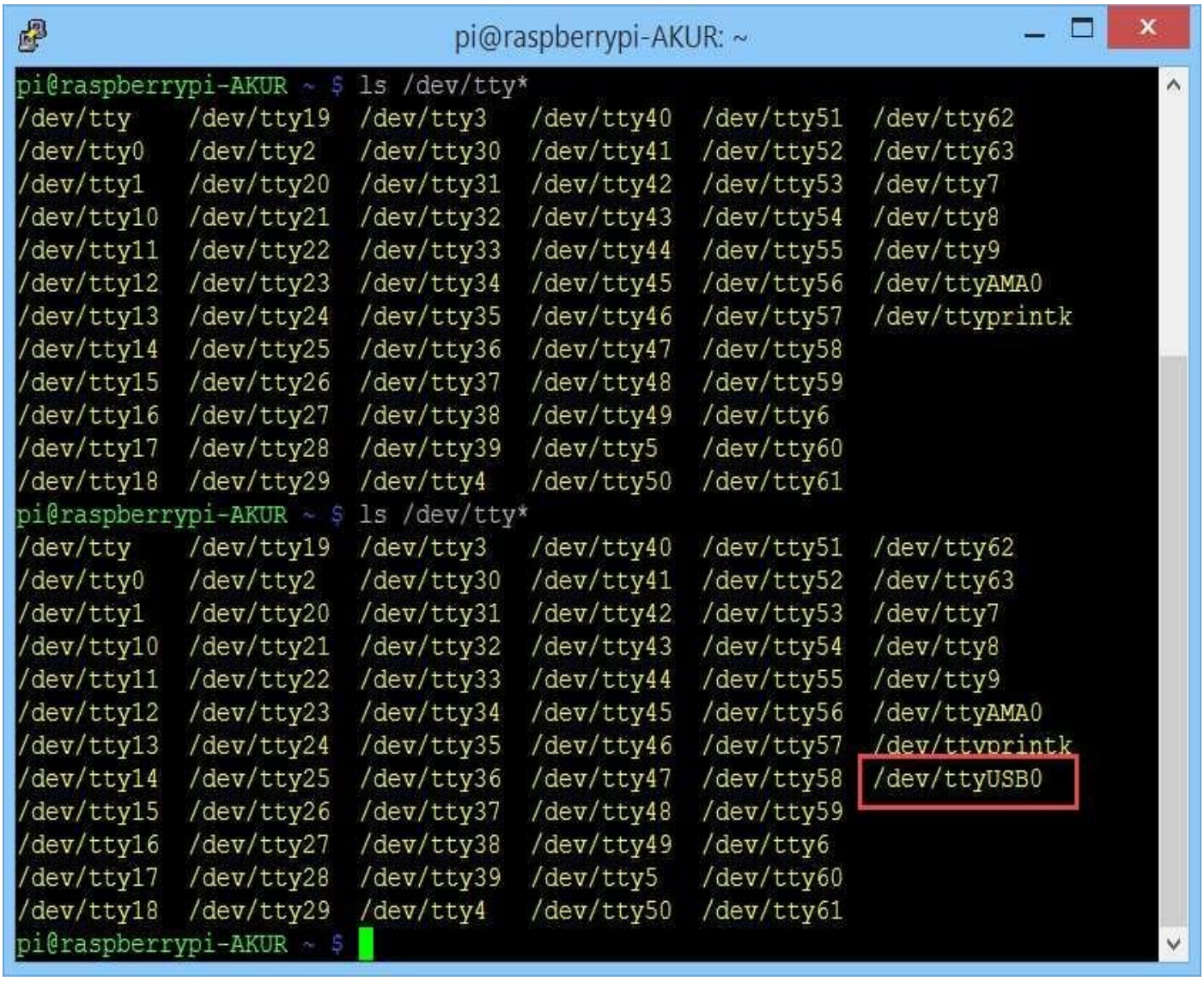
You can connect your XBee to Raspberry Pi through XBee explorer USB. The following is a sample of hardware configuration for XBee and Raspberry Pi.

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*.

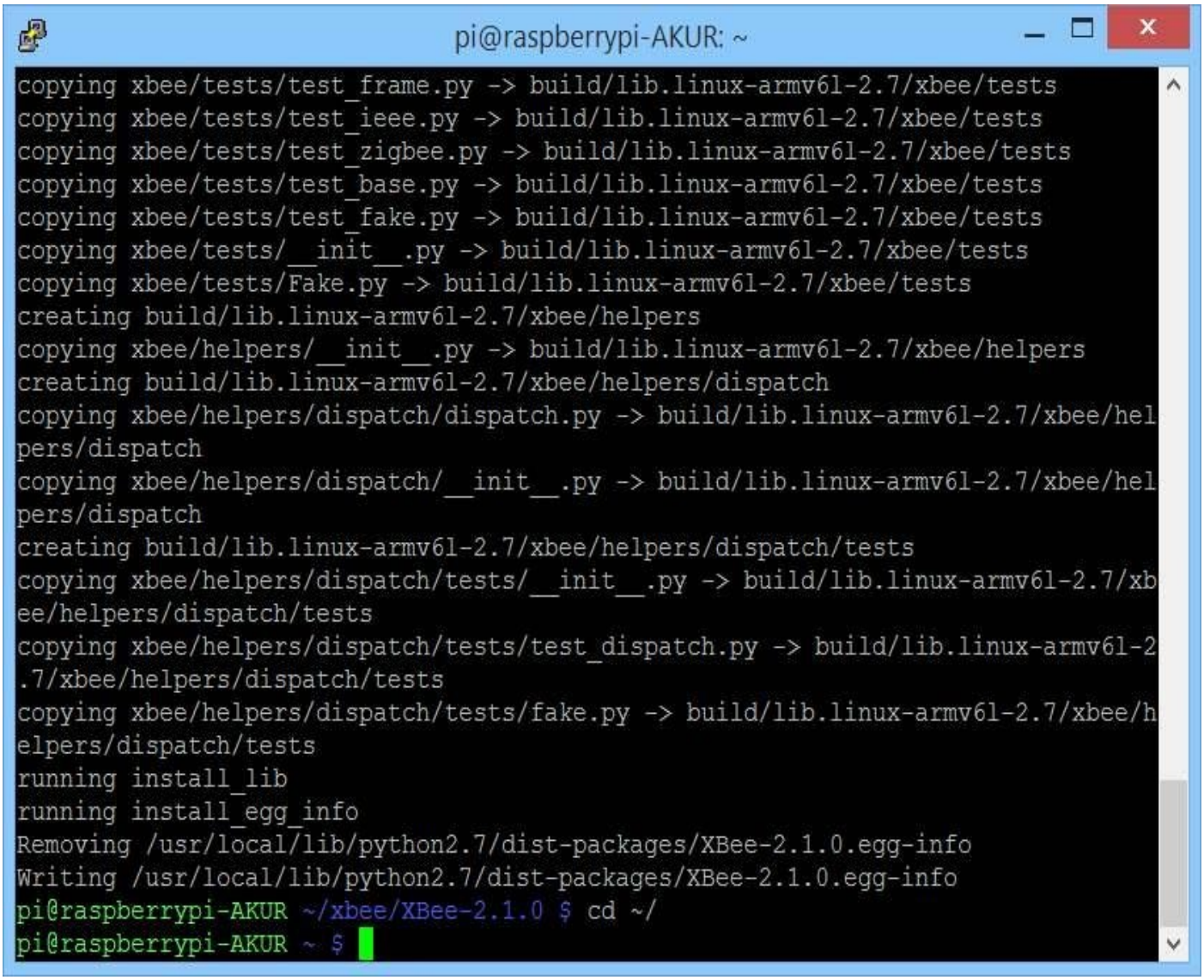


7.2.3 XBee Programming for Raspberry Pi

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/hel
pers/dispatch
copying xbee/helpers/dispatch/__init__.py -> build/lib.linux-armv6l-2.7/xbee/hel
pers/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/xb
ee/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/h
elpers/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]
```

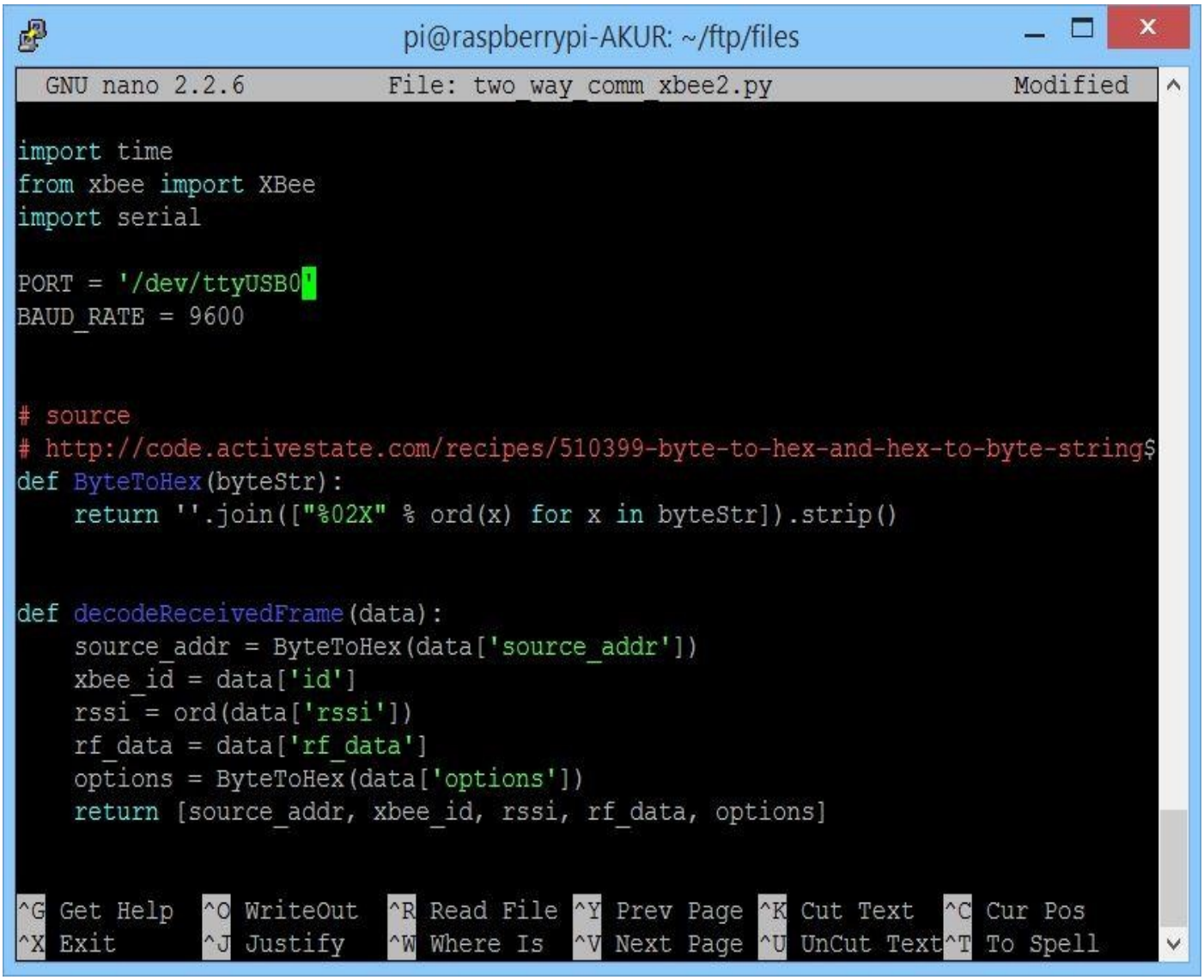
The following is XBee 2 configuration on X-CTU.

```
+++
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

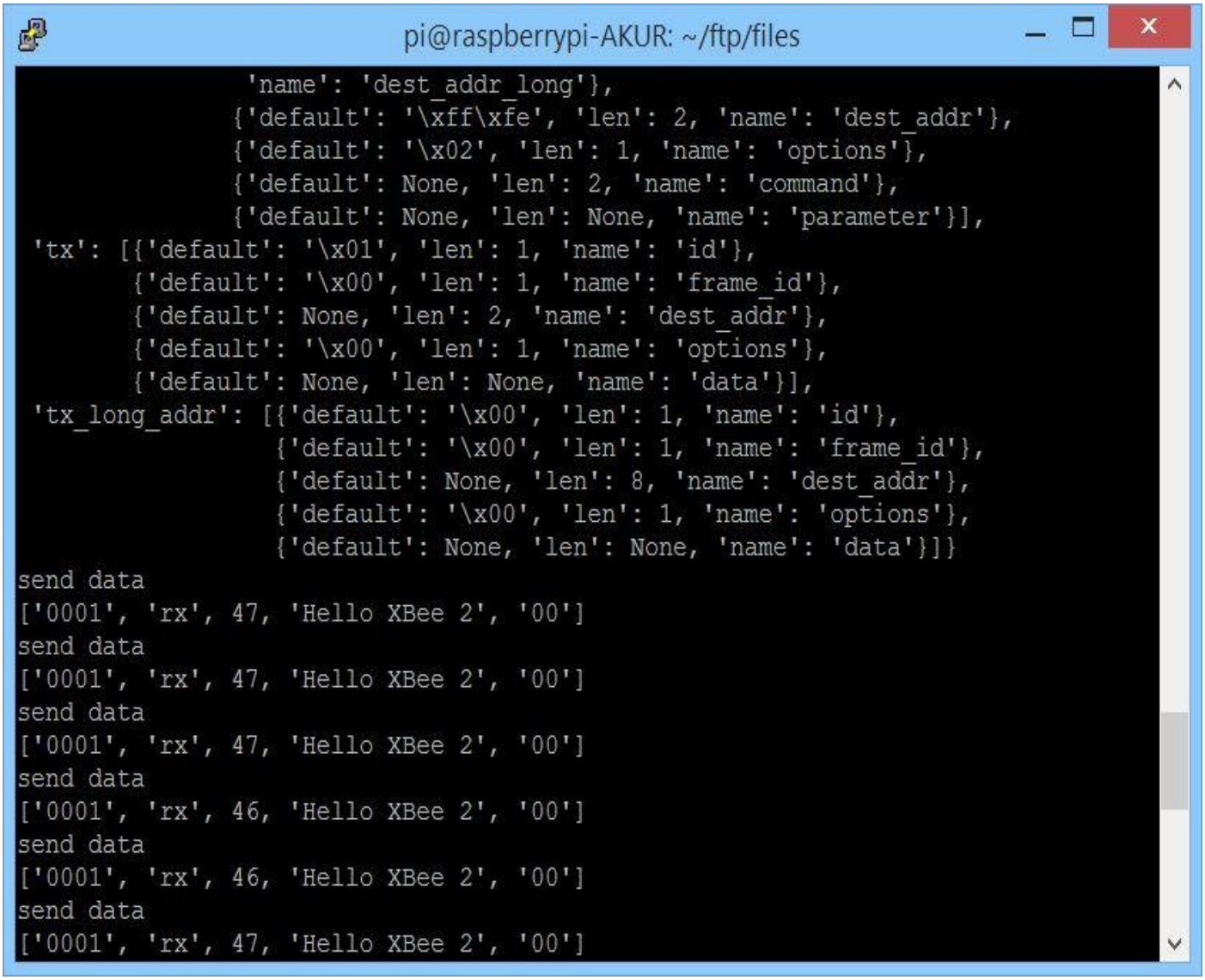
As I explained, we use the same file on chapter 7, **two_way_comm_xbee1.py** and **two_way_comm_xbee2.py** files. Copy **two_way_comm_xbee2.py** to Raspberry Pi and change serial port. The following is a sample code on Raspberry Pi



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.

The following is a sample output on Raspberry Pi.



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/ .

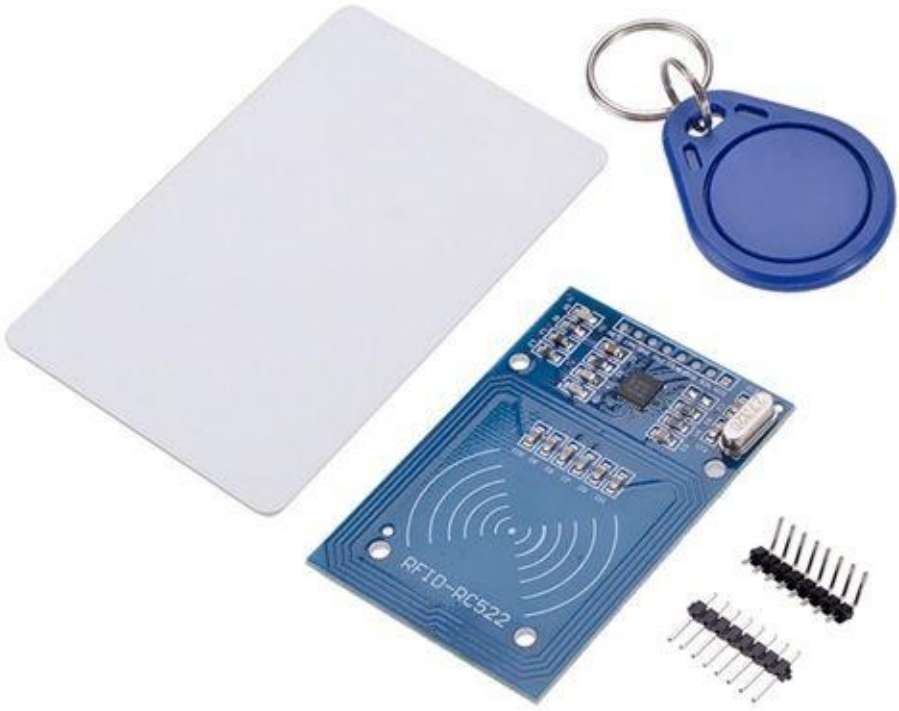
In this chapter, we explorer how to work with RFID/NFC modules on Raspberry Pi. There are two lab scenarios we will build:

- 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=MFRC52 . 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=
.

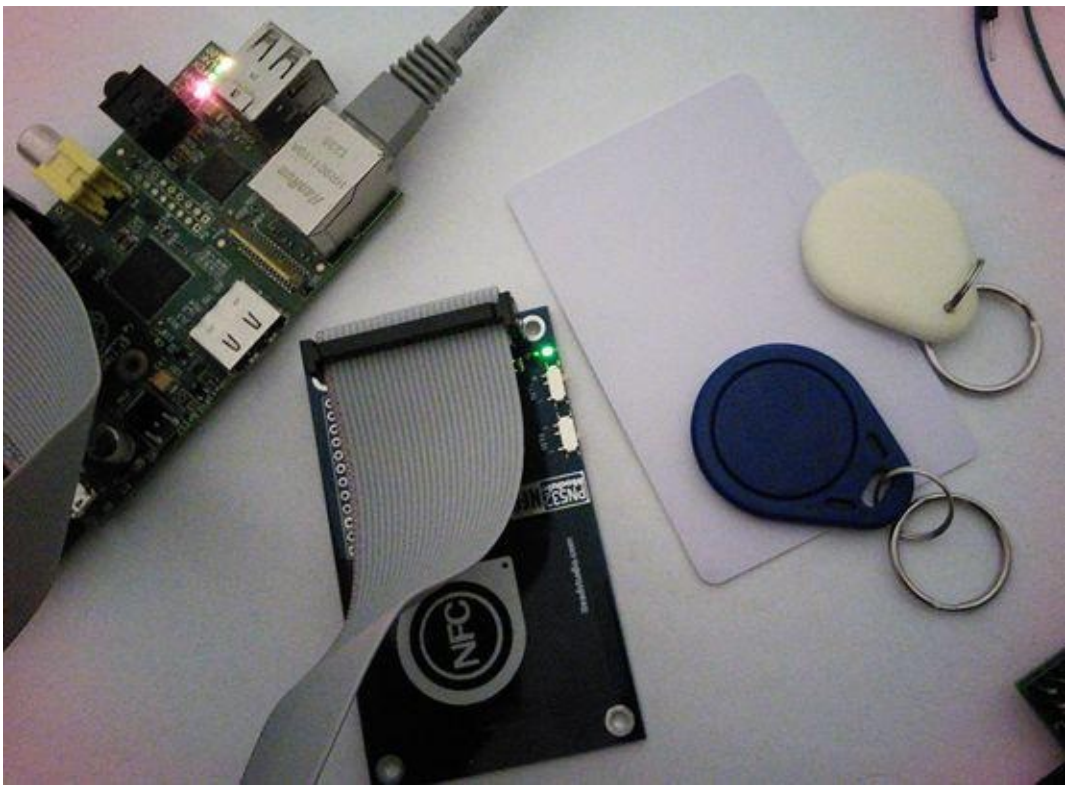
You can buy RFID/NFC modules on the following store:

- Seeedstudio, http://www.seeedstudio.com/depot/s/nfc.html?search_in_description=0
- 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 .

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_ISO14443A);
                if (id != 0)
                {
                    printf("Read card #%d\n",id);
                }
        }
}
```

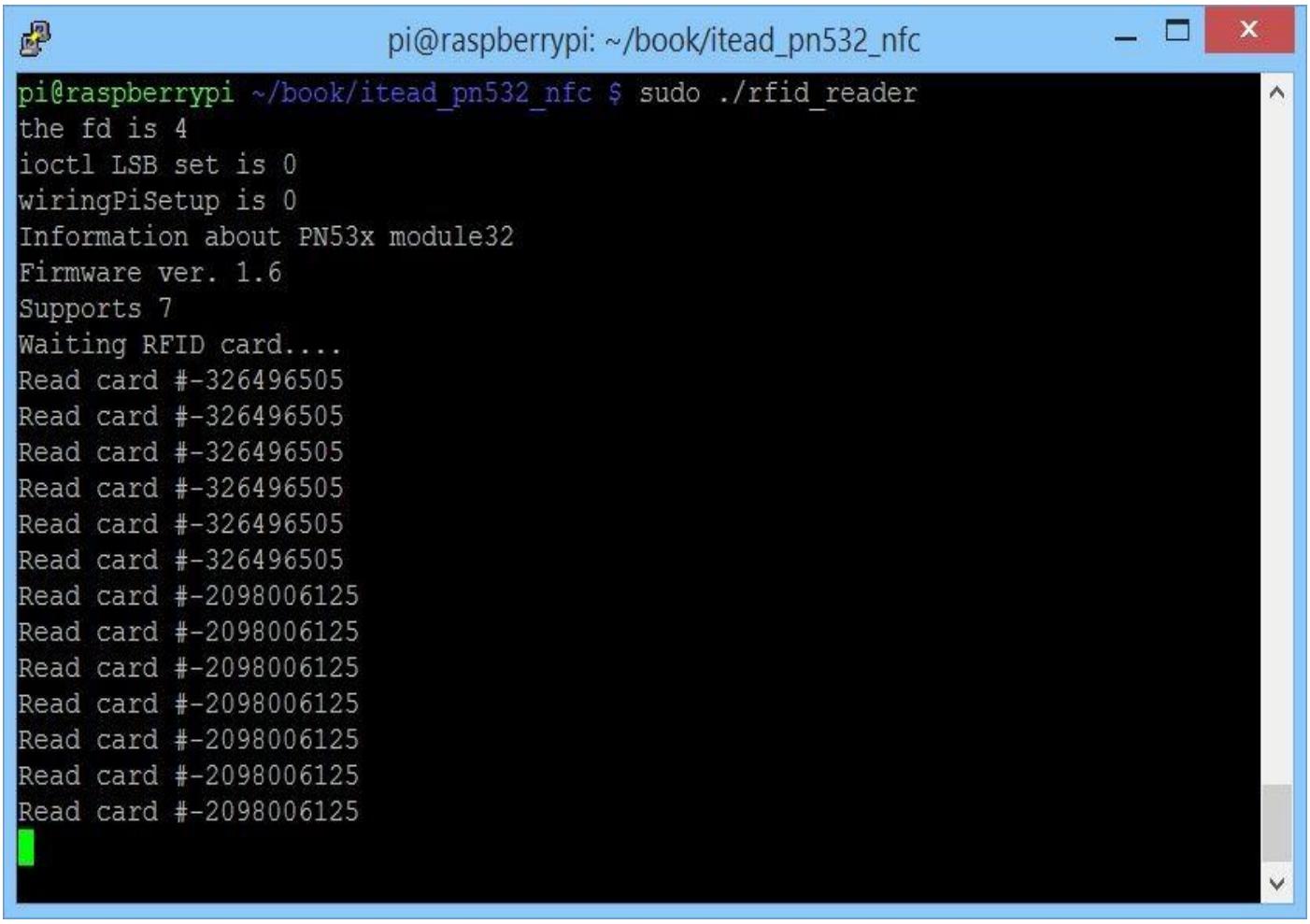
Save this code. You can compile this file using this command.

```
$ 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 #-326496505
Read card #-326496505
Read card #-326496505
Read card #-326496505
Read card #-326496505
Read card #-2098006125
Read card #-2098006125
Read card #-2098006125
Read card #-2098006125
Read card #-2098006125
Read card #-2098006125
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_ISO14443A);
                if (id != 0)
                {
                    printf(“Read card # %d\n”,id);
            for(blockn=0;blockn<64;blockn++)
            {
                if(authenticateBlock(1, id ,blockn, KEY_A,keys)) //authe
                {
                    //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);

            }
    }
```

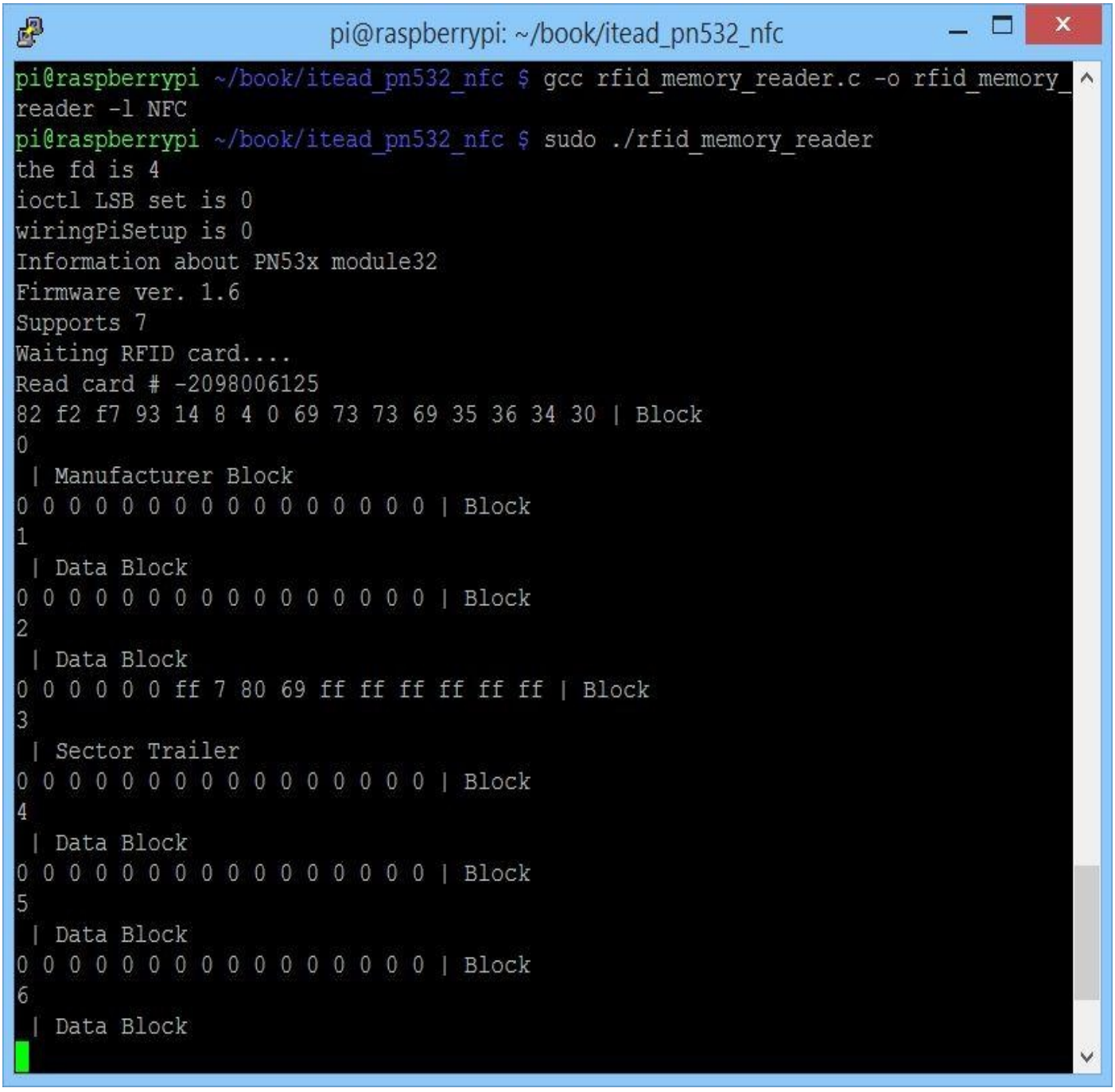
Save this code and try to compile.

```
$ 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.



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.

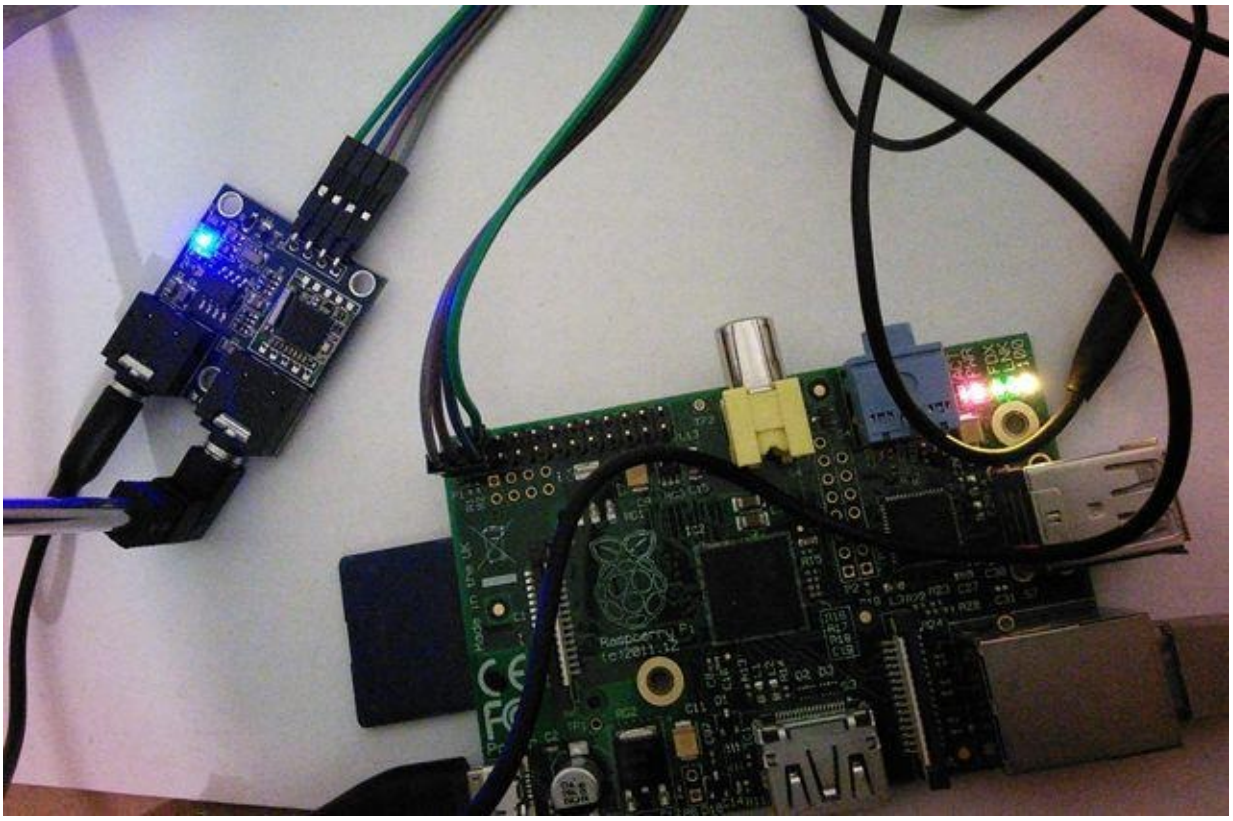
9.3 Wiring

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.

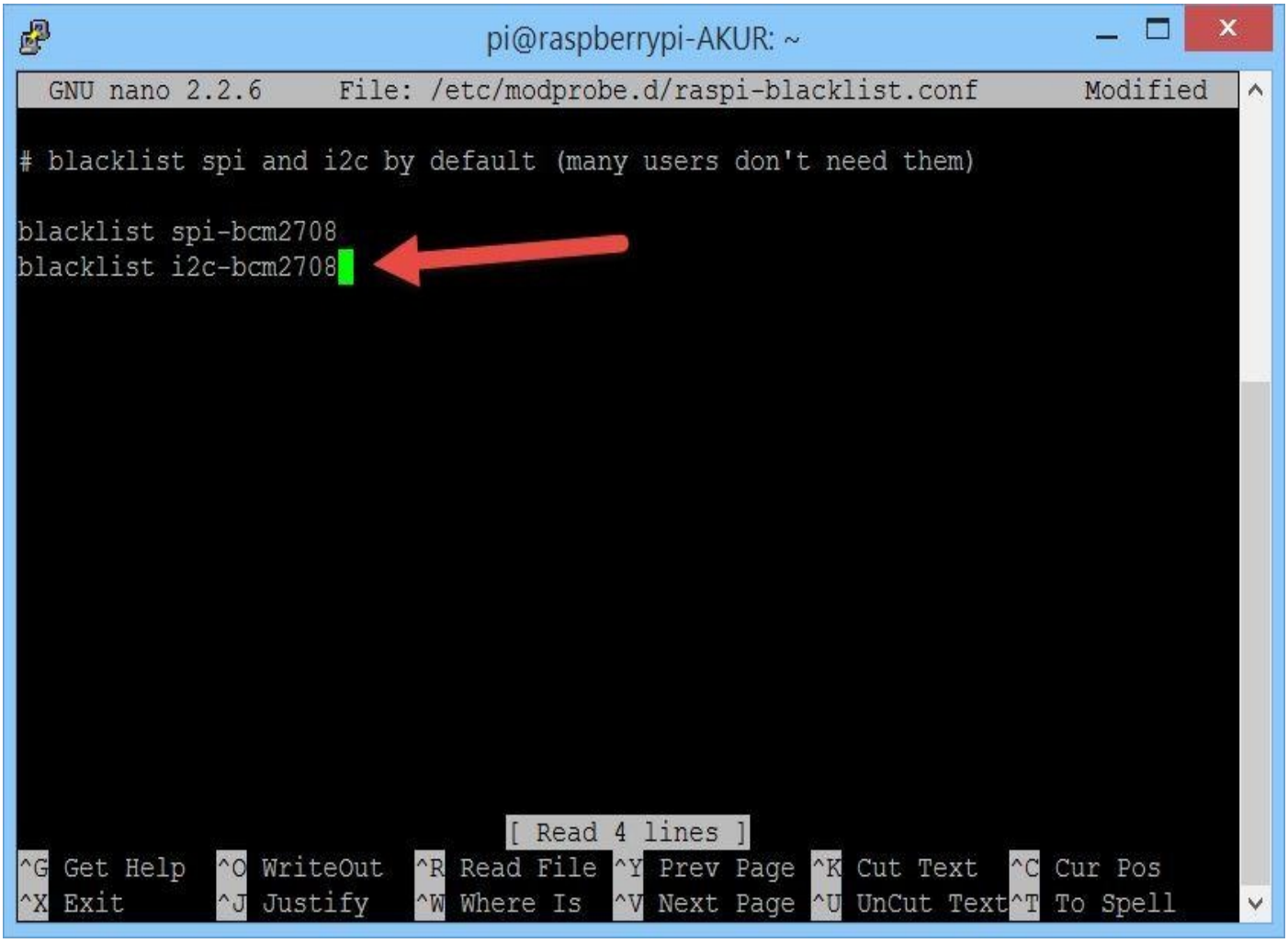


9.4 Configuring I2C for Raspberry Pi

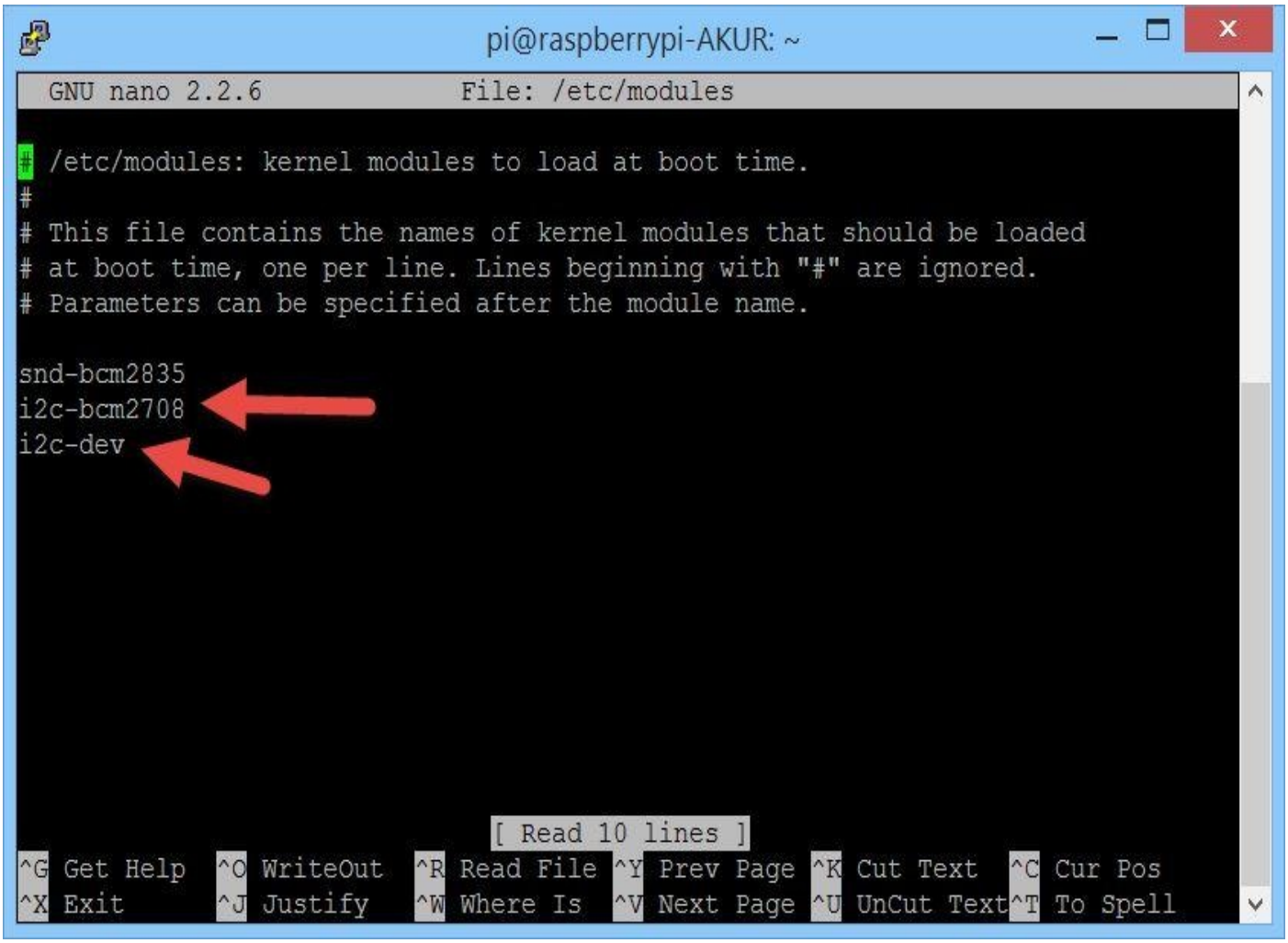
To access I2C, we must configure Raspberry Pi to enable this communication. Firstly, we remove I2C from module blacklist. Open file **/etc/modprobe.d/raspi-blacklist.conf**, for instance I use nano editor.

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

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



The final configuration should be as follows.

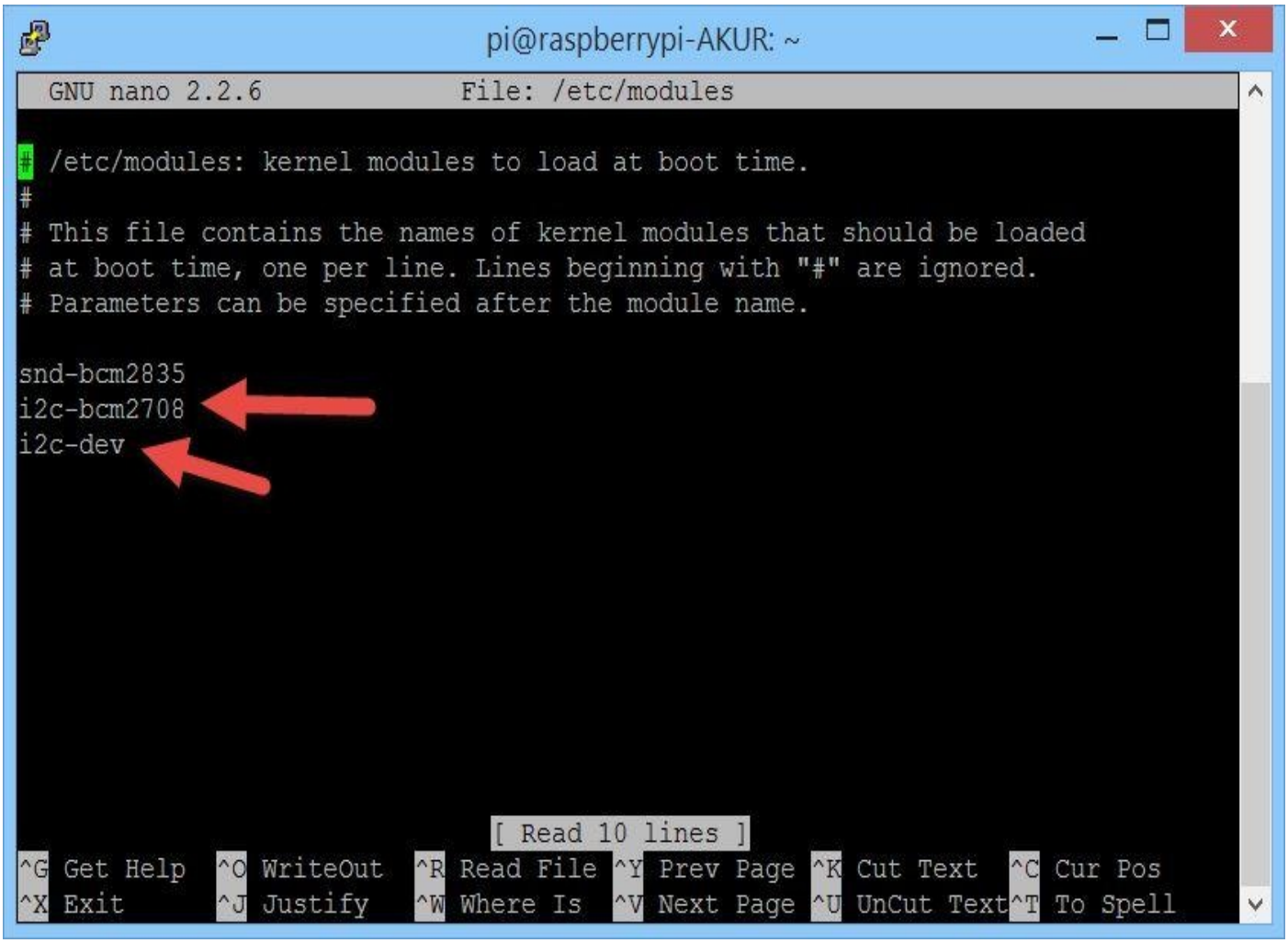


Save this file and close it.

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.



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
```

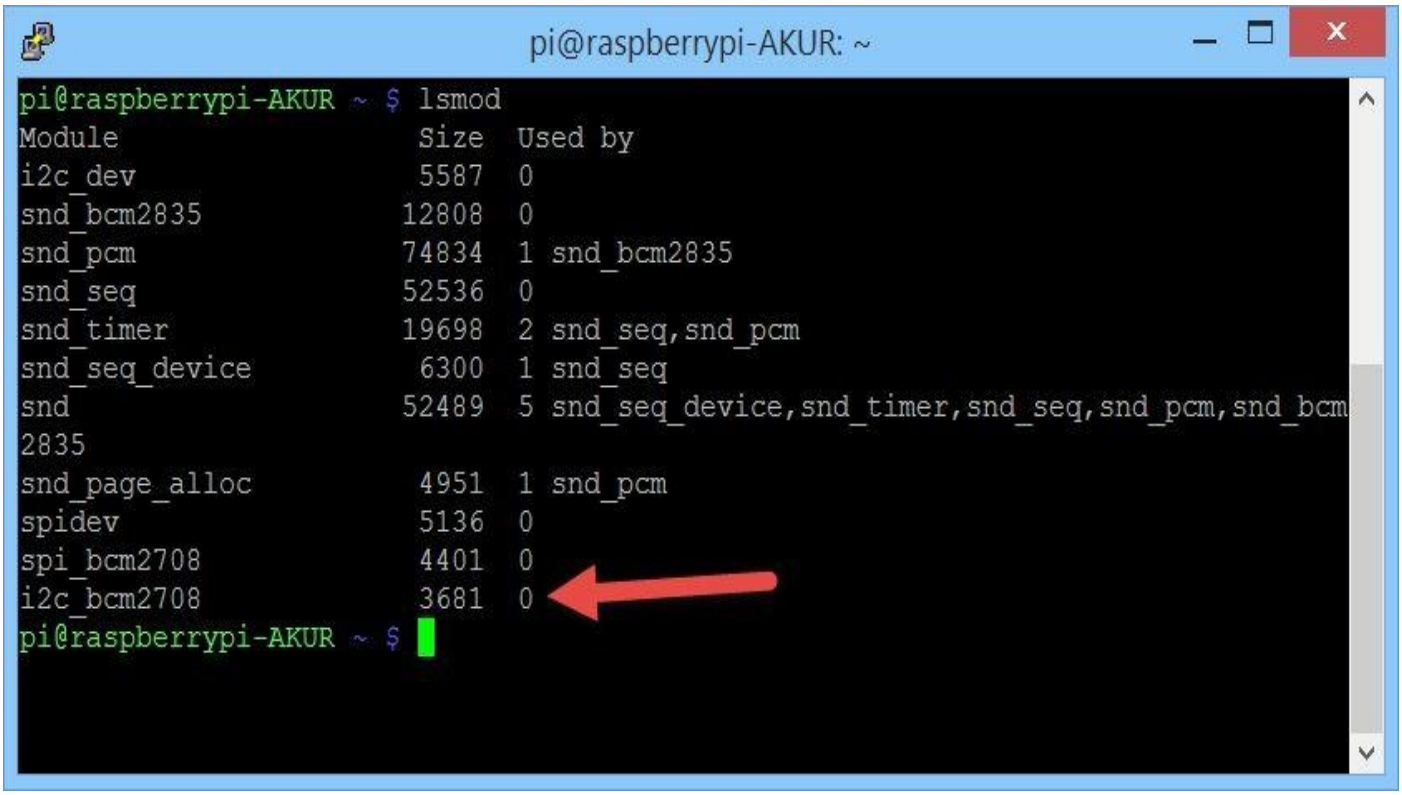
If done, you must reboot your Raspberry Pi by typing the following command.

```
$ 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.



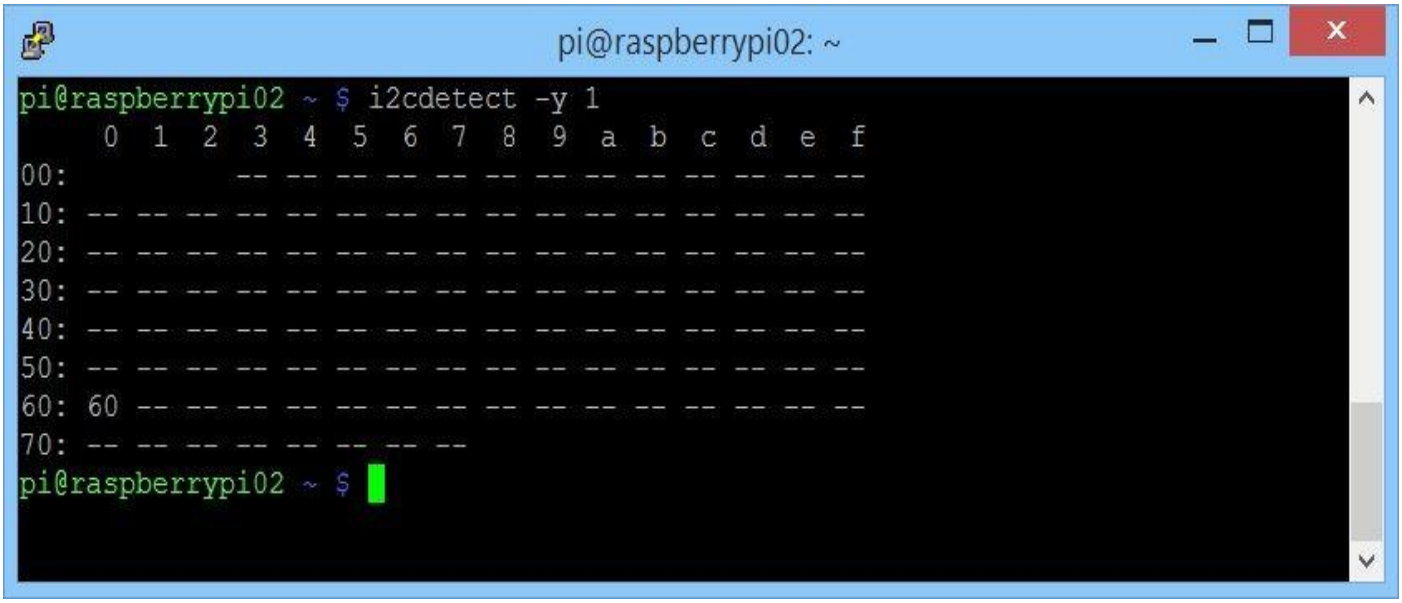
You can see i2c_bcm2708 on the output.

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.



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 LO 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.

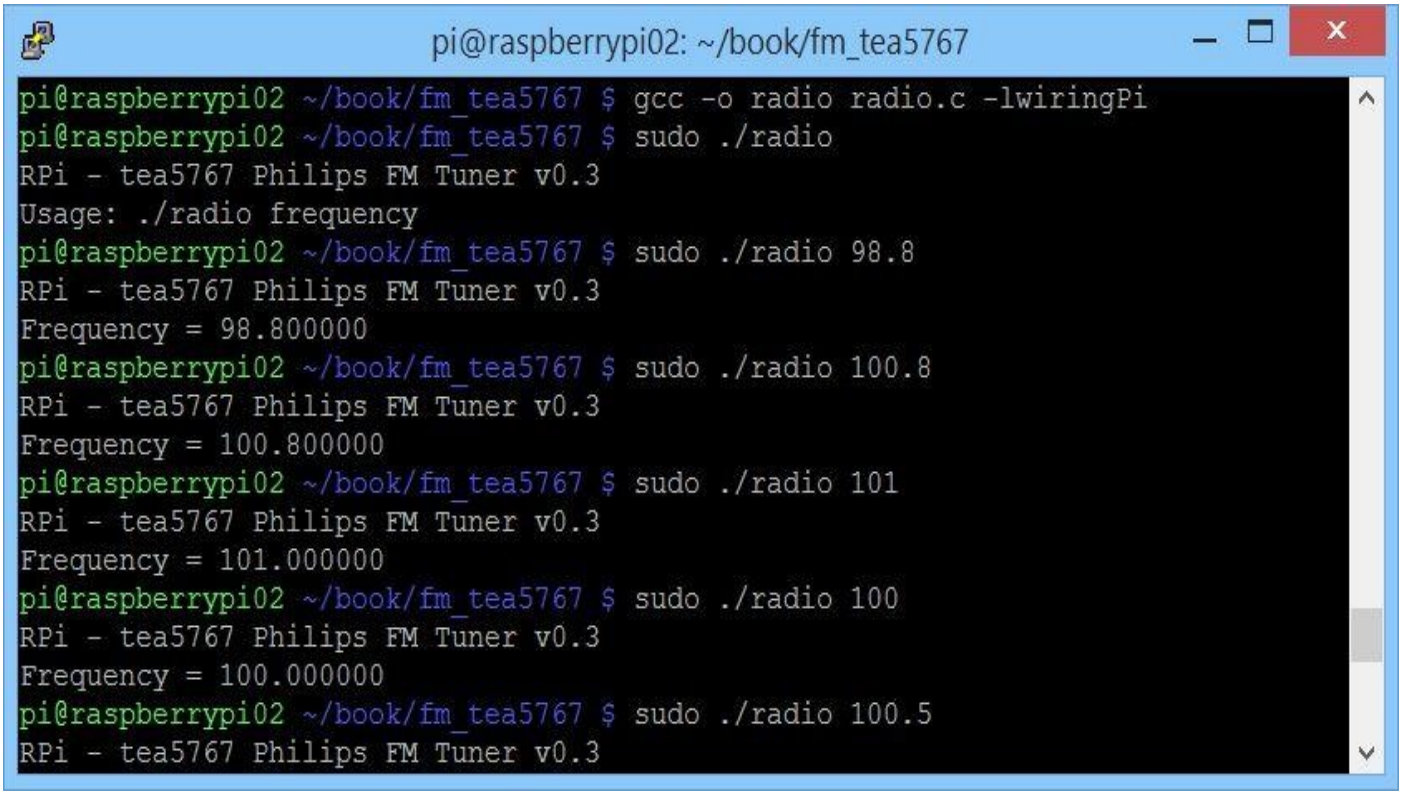
```
$ gcc -o radio radio.c -lwiringPi
```

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.



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.