Fi-Pi Software Manual

e-Yantra summer internship 2017

July 5,2017

Contents

Chapter 1

Installation of an Operating System on Raspberry Pi

1.1 Objective

This tutorial will help a Windows or a Linux user to install an operating system on Raspberry Pi successfully

1.2 Prerequisites

- An idea about Windows or Linux operating system and their features.

1.3 Hardware Requirement

1. Raspberry Pi (I will be using Version 2 Model-B)
2. Adapter (To power up an R-pi)
3. SD card reader
4. PC (either Linux or Windows machine)

1.4 Software Requirement

1. Winrar or any other zipping software
2. Utorrent or any suitable downloading platform
3. win32DiskManager

1.5 Theory and Description

The Raspberry Pi is a series of small single-board computers developed in the United Kingdom by the Raspberry Pi Foundation.Raspberry Pi is a low cost, credit-card sized computer that plugs into a computer monitor or TV, and uses a standard keyboard and mouse. It is a capable little device that enables people of all ages to explore computing, and to learn how to program in languages like Scratch and Python.The Raspberry Pi 2 uses a Broadcom BCM2836 SoC with a 900 MHz 32-bit quad-core ARM Cortex-A7 processor, with 256 KB shared L2 cache Its capable of doing everything youd expect a desktop computer to do, from browsing the internet and playing high-definition video, to making spreadsheets, word-processing, and playing games.

1.6 A comparison between different models and versions of Raspberry pi

Parameter	Model A	Model A+	Model B	Model B+	Generation 2 Model B
Target price(in US dollars):	25	20	35	25	35
SoC:	Broadcom BCM2835 (CPU, GPU, DSP, SDRAM, one USB port)	Broadcom BCM2835 (CPU, GPU, DSP, SDRAM, one USB port)	Broadcom BCM2835 (CPU, GPU, DSP, SDRAM, one USB port)	Broadcom BCM2835 (CPU, GPU, DSP, SDRAM, one USB port)	Broadcom BCM2836 (CPU, GPU, DSP, SDRAM, one USB port)
CPU:	700 MHz single-core ARM1176JZF-S	700 MHz single-core ARM1176JZF-S	700 MHz single-core ARM1176JZF-S	700 MHz single-core ARM1176JZF-S	900 MHz quad-core ARM Cortex-A7
Memory (SDRAM):	256 MB (shared with GPU) 512 MB	256 MB (shared with GPU)	512 MB (shared with GPU)	512 MB (shared with GPU)	1 GB (shared with GPU)
USB 2.0 ports:	1 (direct from BCM2835 chip)	1 (direct from BCM2835 chip)	2 (via the on-board 3-port USB hub)[38]	4 (via the on-board 5-port USB hub)	4 (via the on-board 5-port USB hub)
On-board storage:	SD / MMC / SDIO card slot (3.3 V with card power only)	MicroSD slot[32]	SD / MMC / SDIO card slot	MicroSD slot	MicroSD slot
On-board network:	None	None	10/100 Mbit/s Ethernet (8P8C) USB adapter on the third/-fifth port of the USB hub (SMSC lan9514-jzx)	10/100 Mbit/s Ethernet (8P8C) USB adapter on the third/-fifth port of the USB hub (SMSC lan9514-jzx)	10/100 Mbit/s Ethernet (8P8C) USB adapter on the third/-fifth port of the USB hub (SMSC lan9514-jzx)
Low-level peripherals:	8 GPIO	17 GPIO plus the same specific functions, and HAT ID bus	8 GPIO	17 GPIO	17 GPIO
Power ratings:	300 mA (1.5 W)	200 mA (1 W)	700 mA (3.5 W)	600 mA (3.0 W)	800 mA(4.0 W)

Ref: [2]

1.7 Experiment

Instructions for installing operating system in Raspberry Pi:
Download Raspbian software from raspberry pi website. It should be available in the form of a zip file. Download it from the following link: https://www.raspberrypi.org/downloads/ Also download win32DiskManager from the following link:http://sourceforge.net/projects/win32diskimager/ and then install it.

A windows user should follow these steps:

- Insert an sd card(4GB or greater size) into the laptop in the memory card slot if available or use a memory card reader. Run win32DiskImager and choose the Raspbian image and select the drive corresponding to your sd card.
- Click "Write" to copy the files of the image on to the sd card.
- Eject SD card and insert it into the sd card slot of the Raspberry Pi.

A Linux user should follow these steps:

- Insert an sd card(4GB or greater size) into the laptop in the memory card slot if available or use a memory card reader.
- Run df -h to see what devices are currently mounted. The new device that has appeared is your SD card. The left column gives the device name of your SD card; it will be listed as something like /dev/mmcblk0p1 or /dev/sdd1. The last part (p1 or 1 respectively) is the partition number but you want to write to the whole SD card, not just one partition. Therefore you need to remove that part from the name (getting, for example, /dev/mmcblk0 or /dev/sdd) as the device for the whole SD card. Note that the SD card can show up more than once in the output of df; it will do this if you have previously written a Raspberry Pi image to this SD card, because the Raspberry Pi SD images have more than one partition.
- Note down your device name. You need to unmount it so that files can't be read or written to the SD card while you are copying over the SD image.
- Run umount /dev/sdd1, replacing sdd1 with whatever your SD card's device name is (including the partition number). If your SD card shows up more than once in the output of df due to having multiple partitions on the SD card, you should unmount all of these partitions.

- In the terminal, write the image to the card with the command below, making sure you replace the input file if= argument with the path to your .img file, and the /dev/sdd in the output file of= argument with the right device name. This is very important, as you will lose all data on the hard drive if you provide the wrong device name. Make sure the device name is the name of the whole SD card as described above, not just a partition of it; for example sdd, not sdds1 or sddp1; or mmcblk0, not mmcblk0p1.

  dd bs=4M if=2015-05-05-raspbian-wheezy.img of=/dev/sdd

  Please note that block size set to 4M will work most of the time; if not, please try 1M, although this will take considerably longer.

  Also note that if you are not logged in as root you will need to prefix this with sudo.

- The dd command does not give any information of its progress and so may appear to have frozen; it could take more than five minutes to finish writing to the card. If your card reader has an LED it may blink during the write process. To see the progress of the copy operation you can run pkill -USR1 -n -x dd in another terminal, prefixed with sudo if you are not logged in as root. The progress will be displayed in the original window and not the window with the pkill command; it may not display immediately, due to buffering.

  Instead of dd you can use dcfldd; it will give a progress report about how much has been written.

  You can check what's written to the SD card by dd-ing from the card back to another image on your hard disk, truncating the new image to the same size as the original, and then running diff (or md5sum) on those two images.

- The SD card might be bigger than the original image, and dd will make a copy of the whole card. We must therefore truncate the new image to the size of the original image. Make sure you replace the input file if= argument with the right device name. diff should report that the files are identical.

  dd bs=4M if=/dev/sdd of=from-sd-card.img truncate –reference 2015-05-05-raspbian-wheezy.img from-sd-card.img diff -s from-sd-card.img 2015-05-05-raspbian-wheezy.img

- Run sync; this will ensure the write cache is flushed and that it is safe to unmount your SD card.

- Remove the SD card from the card reader and insert it into Raspberry Pi's SD card slot. [1]

After inserting the SD card into Raspberry Pi follow these steps:

1. Connect the keyboard and mouse.Use HDMI cable to connect the board to the VGA monitor

2. Power on the board and the monitor. You will notice a set of code running on the monitor.

3. After a while a software configuration tool opens.

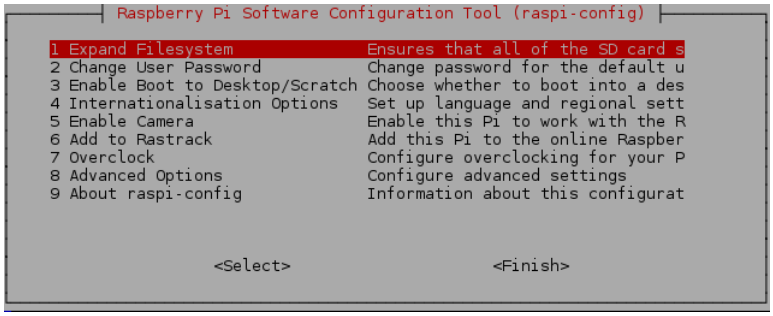

Figure 1.1:

4. In case the size of the SD card is greater than 4GB you can select the 1st option i.e.Expand file system in the configuration tool as shown above.

5. Select the internationalisation options.

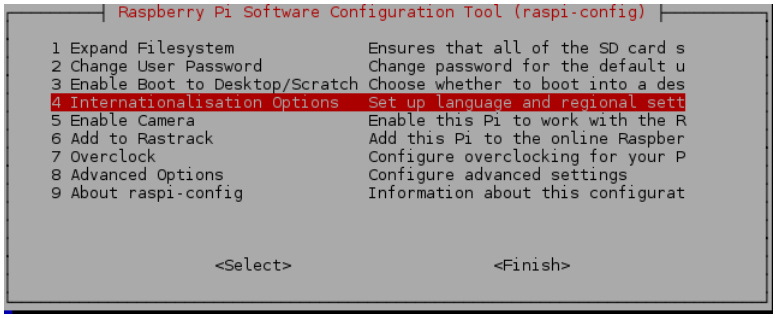

Figure 1.2:

6. Change the keyboard and language settings as per your choice. Default keyboard will be of UK. Step by step change to US keyboard is as shown:

   - First select the option change Keyboard layout and click ok

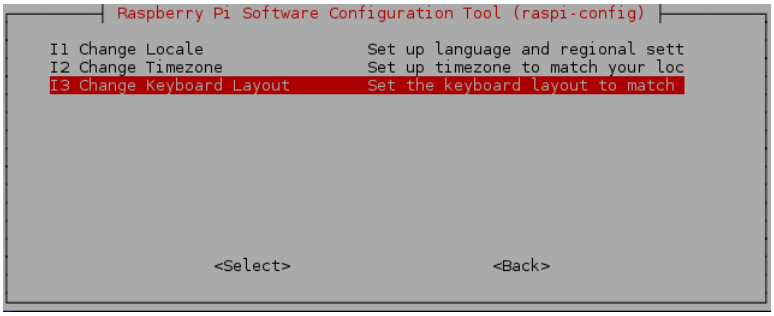

Figure 1.3:

- Then select the respective keyboard you are using and if you dont find your kind then select any of the generic types.

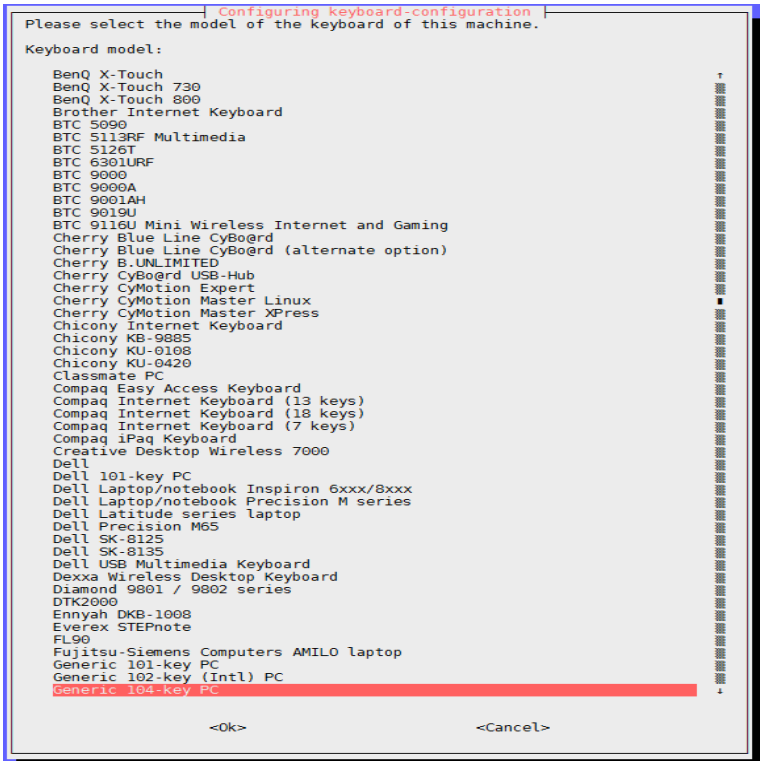

Figure 1.4:

- Select "English US" which will be available at the top. Then for the rest of the process simply click enter till you reach the main configuration menu.

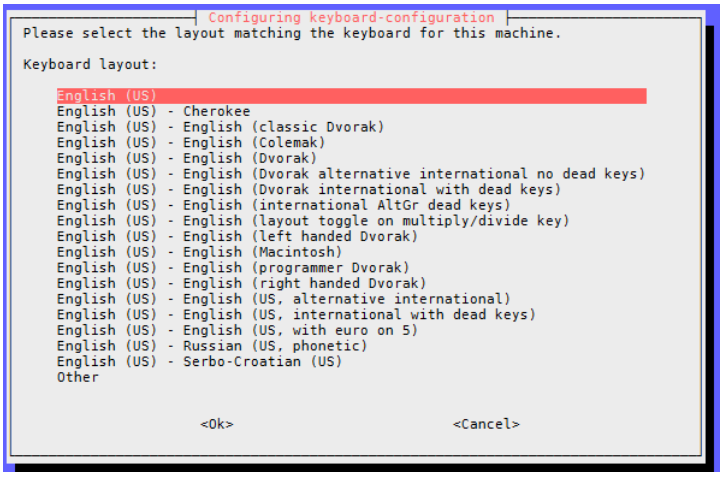

Figure 1.5:

7. Click on finish using the keyboard by using arrow keys and enter key.
8. Click on "ok" when prompted for reboot.
9. Upon reboot, enter 'pi' as user id and 'raspberry' as password. Then type 'startx' to enter the graphical interface.
10. The linux based raspbian interface will be displayed on the monitor. Some of the softwares will be preloaded such as wolfram, python etc.
11. The command prompt will be the lx terminal.

With this we end the installation of Raspbian OS in R-Pi.

Chapter 2

Establishing Network connection and SSH between pc and Raspberry pi

2.1 Objective

In this tutorial we will be learning how to establish a network connection in an R-Pi and how to connect a PC to a Raspberry Pi using SSH connection(i.e. remote control).

2.2 Prerequisites

One should be aware of :

- Basic terminal commands.
- The ssid and password of a wireless network to which an R-Pi has to be connected.

2.3 Hardware Requirement

1. Raspberry Pi (I will be using Version 2 Model B)
2. Monitor,HDMI cable,Keyboard,Mouse (This an optional mode of connecting an R-Pi.In order to learn about different ways to connect an R-Pi kindly refer the previous tutorial)
3. Wireless adapter
4. Power adapter
5. PC(either Windows or Linux)

2.4 Software Requirement

MobaXterm(for Windows user)

2.5 Theory and Description

Secure Shell is a program to log into another computer over a network, to execute commands in a remote machine, and to move files from one machine to another. It provides strong authentication and secure communications over insecure channels. It is a replacement for rlogin, rsh, rcp, and rdist. SSH protects a network from attacks such as IP spoofing, IP source routing, and DNS spoofing. An attacker who has managed to take over a network can only force ssh to disconnect. He or she cannot play back the traffic or hijack the connection when encryption is enabled. When using ssh's slogin (instead of rlogin) the entire login session, including transmission of password, is encrypted; therefore it is almost impossible for an outsider to collect passwords. [2] **Applications of SSH**

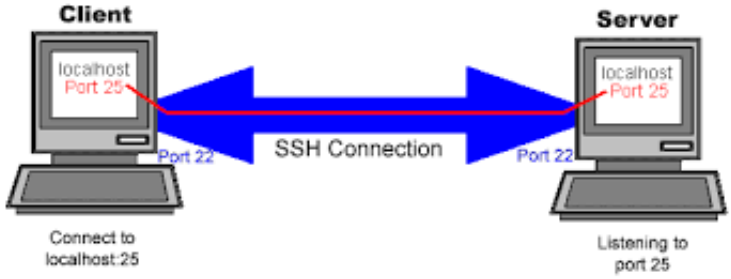

Figure 2.1: [3]

or OpenSSH protocol

- For login to a shell on a remote host (replacing Telnet and rlogin)
- For executing a single command on a remote host (replacing rsh)
- For setting up automatic (passwordless) login to a remote server (for example, using OpenSSH)
- Secure file transfer
- In combination with rsync to back up, copy and mirror files efficiently and securely
- For forwarding or tunneling a port (not to be confused with a VPN, which routes packets between different networks, or bridges two broadcast domains into one).
- For using as a full-fledged encrypted VPN. Note that only OpenSSH server and client supports this feature.

- For forwarding X from a remote host (possible through multiple intermediate hosts)
- For browsing the web through an encrypted proxy connection with SSH clients that support the SOCKS protocol.
- For securely mounting a directory on a remote server as a filesystem on a local computer using SSHFS.
- For automated remote monitoring and management of servers through one or more of the mechanisms discussed above.
- For development on a mobile or embedded device that supports SSH.[1]

MobaXterm

MobaXterm is a tool used for remote computing. In a single Windows application, it provides loads of functions that are tailored for programmers, webmasters, IT administrators and pretty much all users who need to handle their remote jobs in a more simple fashion.It provides all the important remote network tools (SSH, X11, RDP, VNC, FTP, MOSH, ...) and Unix commands (bash, ls, cat, sed, grep, awk, rsync, ...) to Windows desktop, in a single portable exe file which works out of the box.

There are many advantages of having an All-In-One network application for your remote tasks, e.g. when you use SSH to connect to a remote server, a graphical SFTP browser will automatically pop up in order to directly edit your remote files. Your remote applications will also display seamlessly on your Windows desktop using the embedded X server.[4]

2.6 Experiments

2.7 Establishing a network connection

Before we can start using a Raspberry Pi remotely we need to configure its network settings.There are two ways to configure network setting:

- One is using DHCP server (wireless connection)
- The other one is by assigning a static IP

2.7.1 Network settings configuration using DHCP Server(wireless settings)

1. Insert the SD card (with Raspbian OS already written) into the micro SD slot in an R-Pi.
2. Connect the wireless adapter, keyboard , mouse and monitor (using a HDMI cable) to the Raspberry Pi.
3. Power on the board and monitor.You will notice a set of code running on the monitor.
4. Enter the set user name and password and then type *startx* command to launch the GUI.

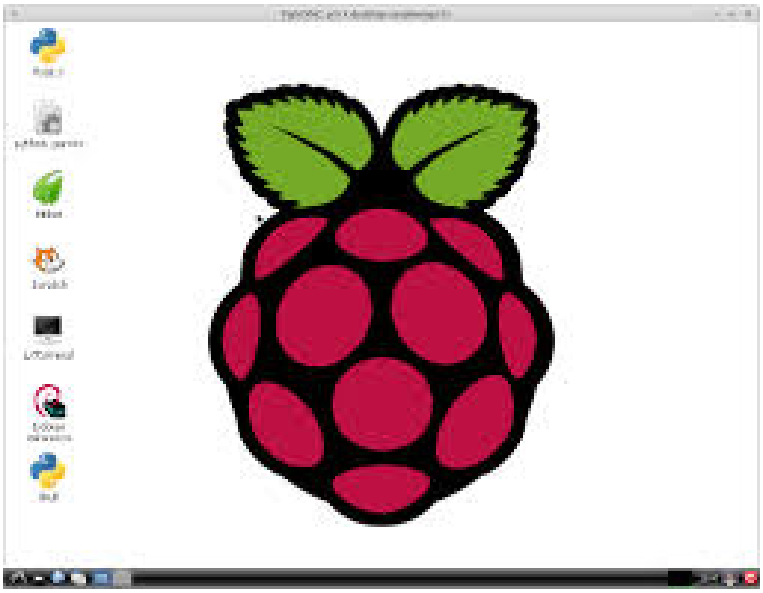

Figure 2.2:

5. Click on the icon at the bottom-left on the screen. This is the start icon.

6. Select LXTerminal option to open a terminal window

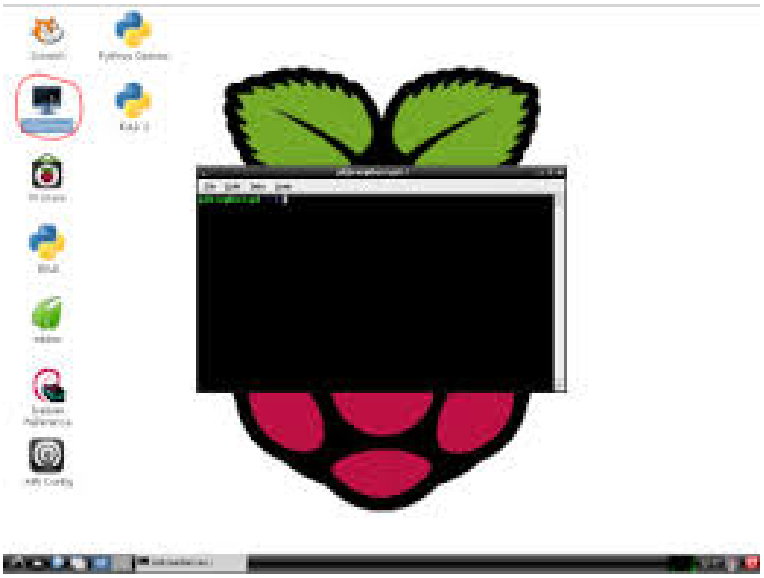

Figure 2.3:

7. Some of the previous versions of Raspbian OS do not support wireless module. To know about the version you are using type *uname -a*. To upgrade to the latest one type "sudo apt-get upgrade".

8. Connect your wireless adapter . To check if its connected type *lsusb*(We used D-link adapter which is highlighted in yellow)

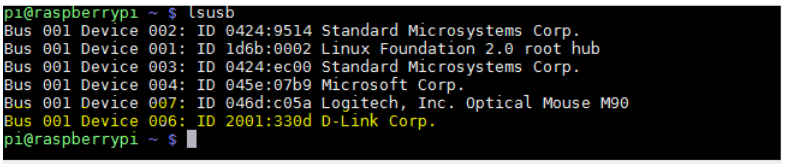

Figure 2.4:

9. Now type *sudo nano /etc/network/interfaces* and press enter. You will see the following window:

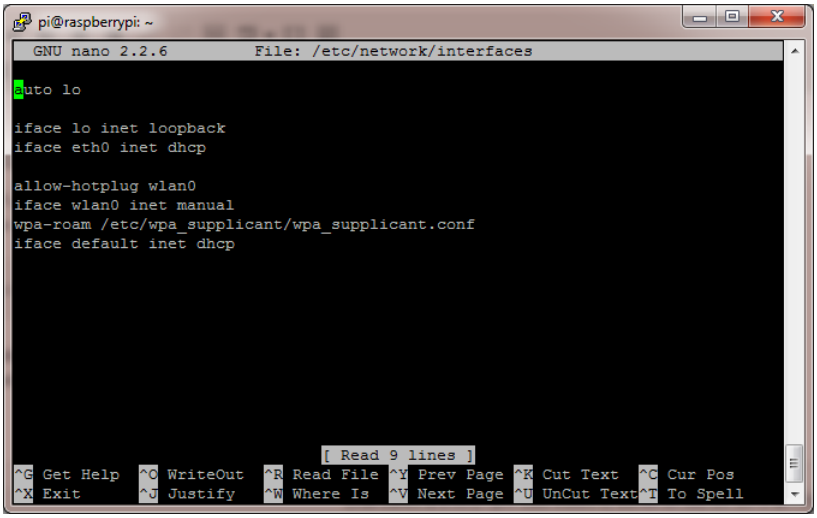

Figure 2.5:

10. Change the code by adding the following lines:

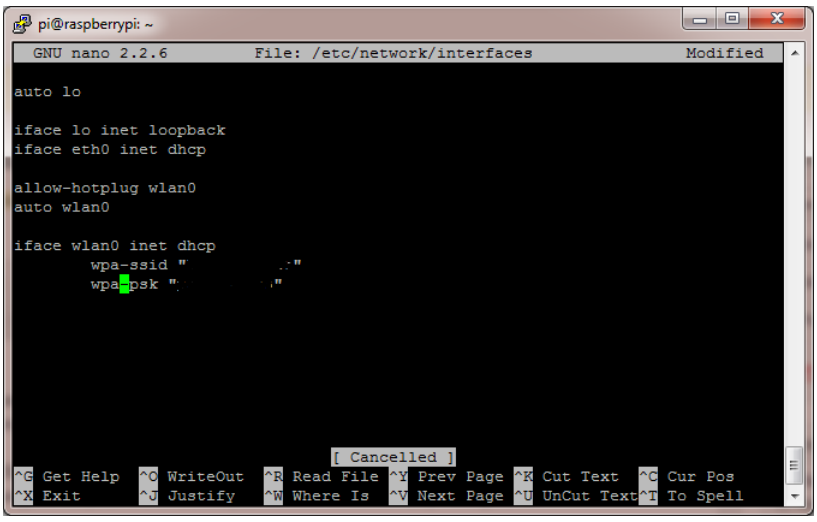

Figure 2.6:

11. Add in the SSID(username) and Password of your wifi. Then for changes to take effect type *sudo /etc/init.d/networking restart*

12. Type *ifconfig* to obtain the IP address of R-Pi etc. It will be under wlan0(written as inet address).

2.7.2 Network settings using static IP method

The router normally distributes the dynamic IP addresses but it isnt guaranteed that it will assign the same IP address every time. This can cause problems if you are trying to connect to your Raspberry Pi remotely. And hence we can assign a static IP. In order to do so follow these steps:

1. Open the LXTerminal and type the following command
   cat /etc/network/interfaces

2. Before you make changes to the document you should be aware of your current IP address, the broadcast IP and the Mask Use the command *ifconfig* to retrieve this information To find your gateway address type the following command: *sudo route nee*

3. In order to edit the interfaces file type *sudo nano /etc/network/interfaces*

4. Remove the following line *iface eth0 inet dhcp* and add the following:
   iface eth0 inet static
   address 192.168.0.x
   netmask 255.255.255.0
   network 192.168.0.0
   broadcast 192.168.0.255
   gateway 192.168.0.y

5. Save the file using Ctrl+X, Y.

6. For changes to take effect type *sudo /etc/init.d/networking restart* and reboot the system.

2.8 Establishing an SSH connection

To start using a Raspberry Pi remotely follow these steps:

1. A windows user should download MobaXterm (latest version) using the following link: http://mobaxterm.mobatek.net/

2. Run the .exe file and open the application.

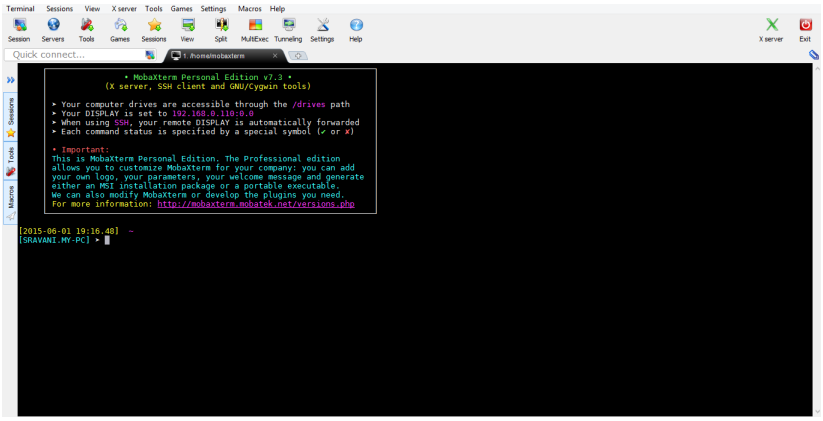

Figure 2.7:

3. Then select the session option on the toolbar.

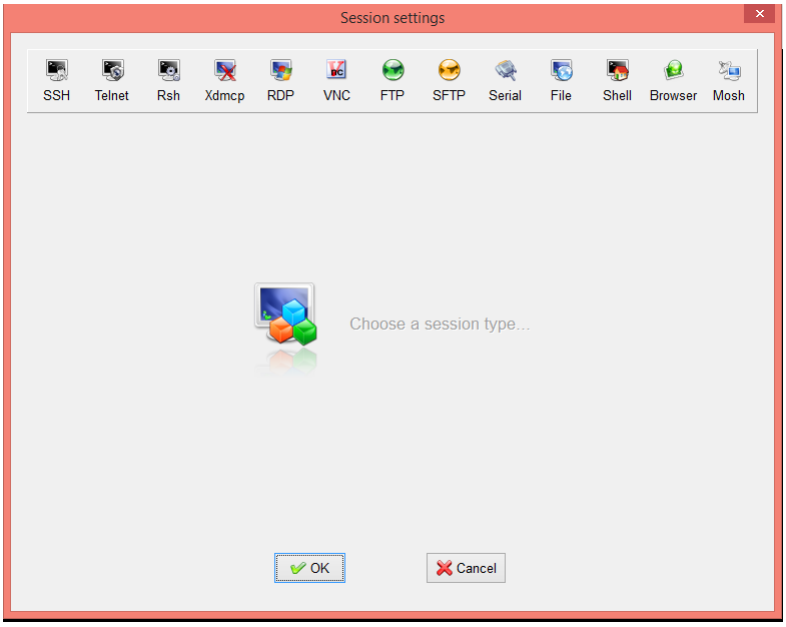

Figure 2.8:

4. Click on the SSH option. A settings window opens as shown

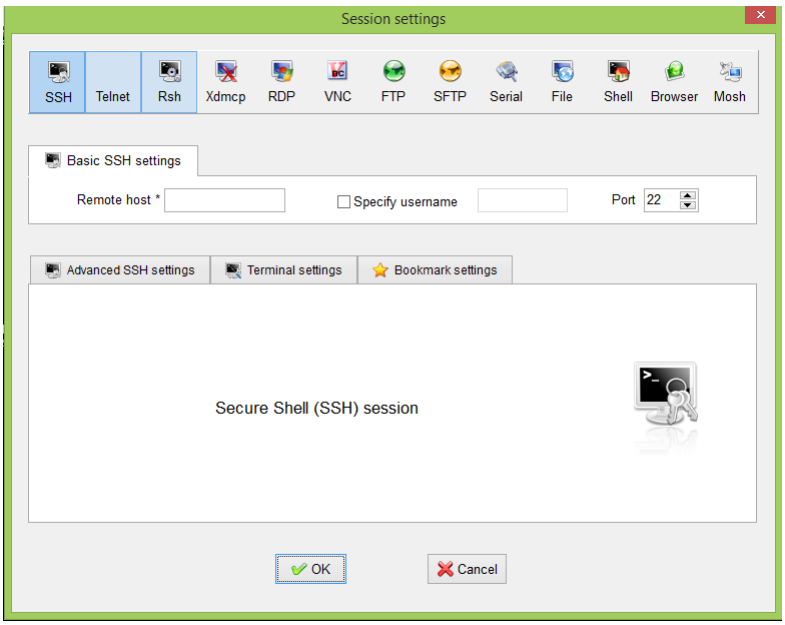

Figure 2.9:

5. Enter the IP address of the R-pi in the Remote host field and in the 'Advanced SSH settings' option configure the remote environment either as 'Interactive shell (terminal based)' or as an 'LXDE desktop(GUI based)'and click OK

6. If you are using the Interactive shell environment login to the R-Pi using the following command: ssh -X pi@192.168.0.4 (IP address of pi) and then enter the login id and password to start using the Pi remotely.

7. If you are using the LXDE desktop i.e. the GUI environment then use LX terminal icon to start programming the Pi.

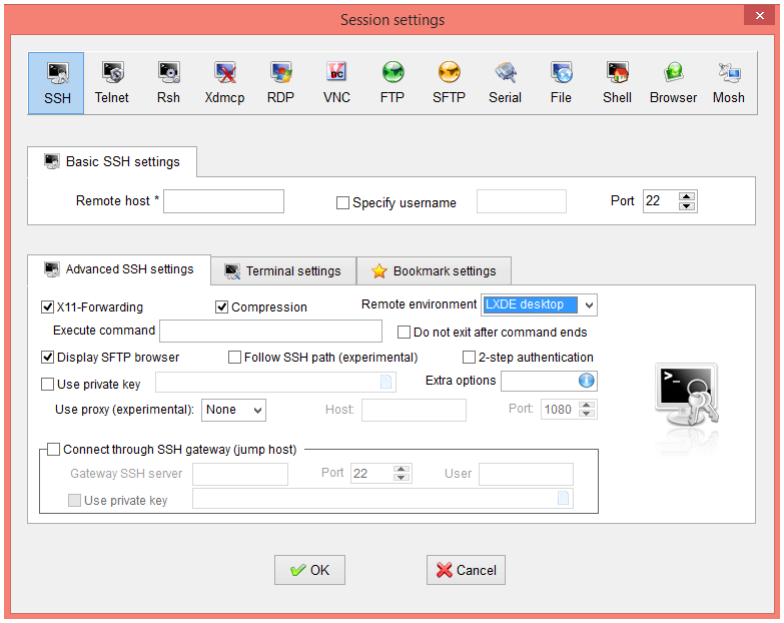

Figure 2.10:

8. Also the established session is saved. So the next time you want to access directly click on the R-Pi's address mentioned in the saved session option to start using the Pi remotely.

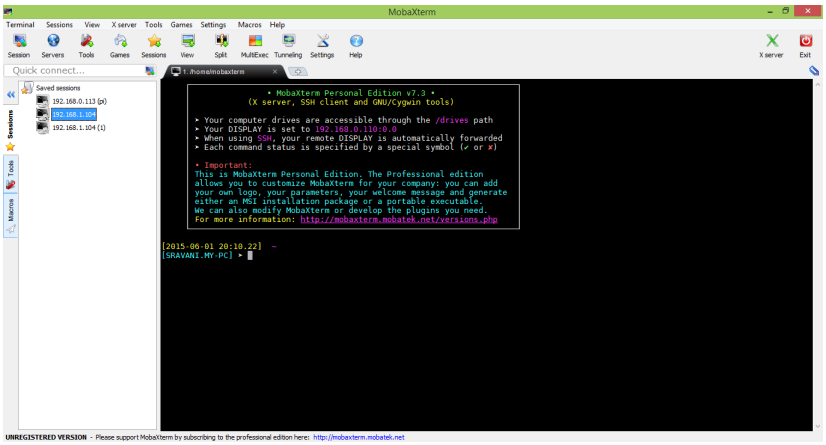

Figure 2.11:

Note: A linux user needn't download any Xterm file. They can directly start accessing the R-Pi using the command: ssh -X pi@192.168.0.4 (IP address of pi) and then entering the login id and password for using the Pi remotely.

Chapter 3

Programming Raspberry-Pi

3.1 Prerequisites

1. PyScripter (version 2.7 or above)
2. Mobaxterm (for windows users)

3.2 Raspberry-pi J8 Header

Expansion Header
The Raspberry Pi 2 Model B board contains a single 40-pin expansion header labelled as 'J8' providing access to 26 GPIO pins. (Pins 1, 2, 39 and 40 are also labelled below.)

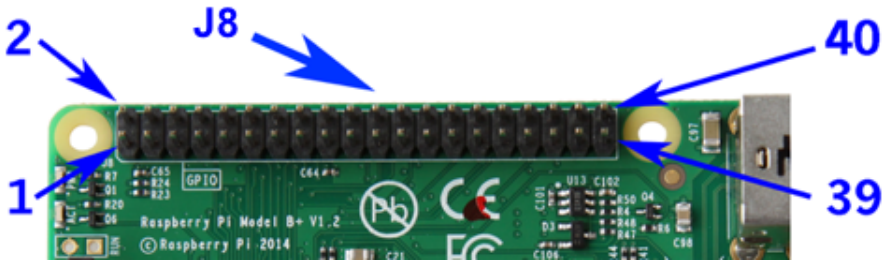

Figure 3.1: [3]

The diagram below illustrates the pin out diagram of Raspberry Pi 2:

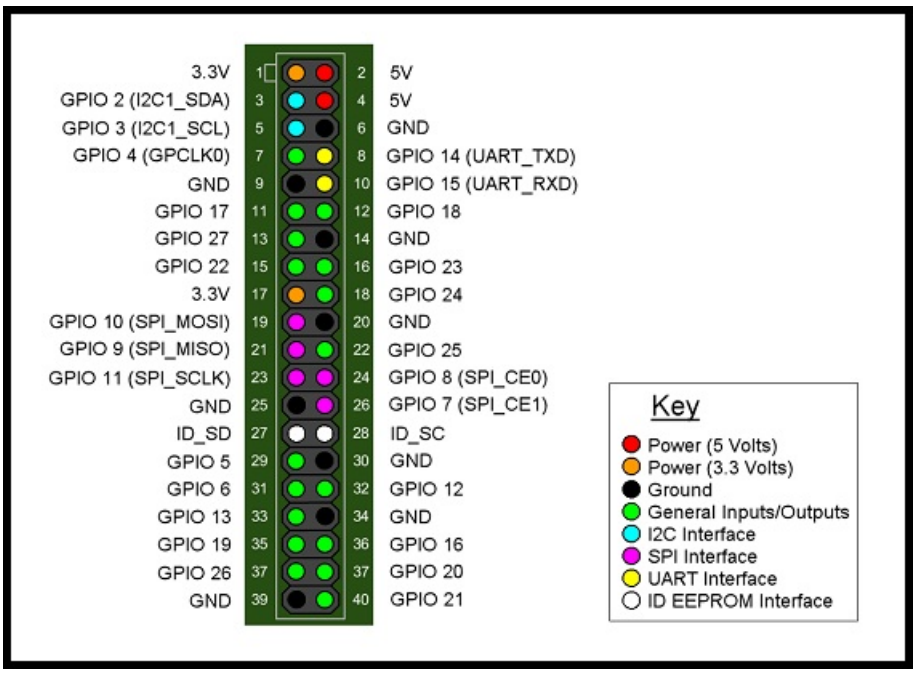

Figure 3.2: [4]

You must have noticed that the board contains pins named as GPIO (that are used for interfacing input and output devices) and hence in order to refer to the R-Pi pins there exists two modes:

1. **BCM mode:** Referring the pins with the GPIO number
2. **Board mode:** Referring the pins using the IC pin numbers.

3.3 Different ways of programming

Before we write a code to access GPIO pins of the Pi let's understand the different ways to program Pi.

1. GUI based programming using IDLE3. In order to do so follow these steps:
    - First, load up IDLE 3 by double-clicking the icon on your LXDE desktop(either on the monitor or using Mobaxterm as explained in the previous tutorial based on establshing a SSH connection).

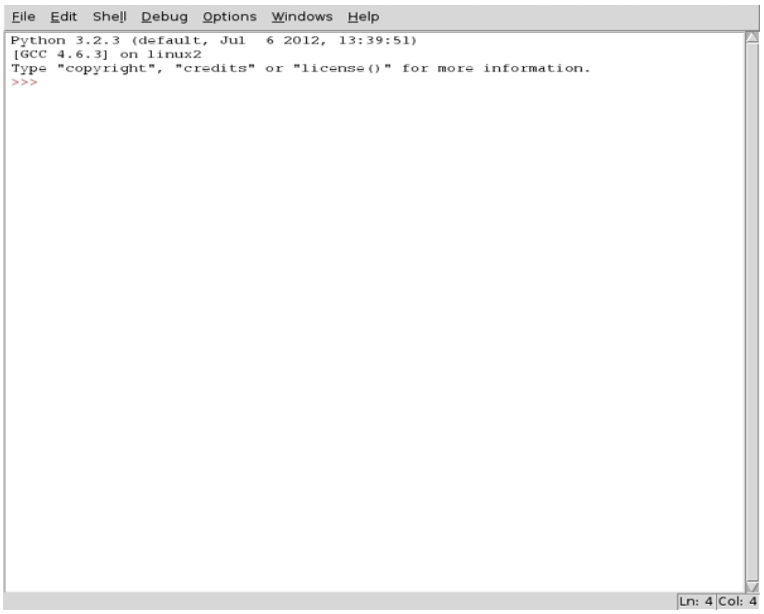

Figure 3.3: [1]

- Click File,New Window, which will then bring up a new blank window which you can type in.
- Now click File,Save As and save your file in the desired folder.
- Click Run, and then Run Module or simply press F5.

2. Terminal based programming:

   - Using PyScripter and MobaXterm: PyScripter is a free and open-source Python IDE used for programming in Python.In order to download PyScripter use the following link https://code.google.com/p/pyscripter/ (I use version 2.5.3).

     (a) Open the application. You will see a window like this:
     (b) Delete the text already present and type your program. Once you finish typing the program goto File ¿ Save As and save the program.
     (c) Now open MobaXterm. On the left side you can see the files in /home/pi/ directory and you also have some small icons on the toolbar. Click on the icon 'Upload to current folder' as shown:

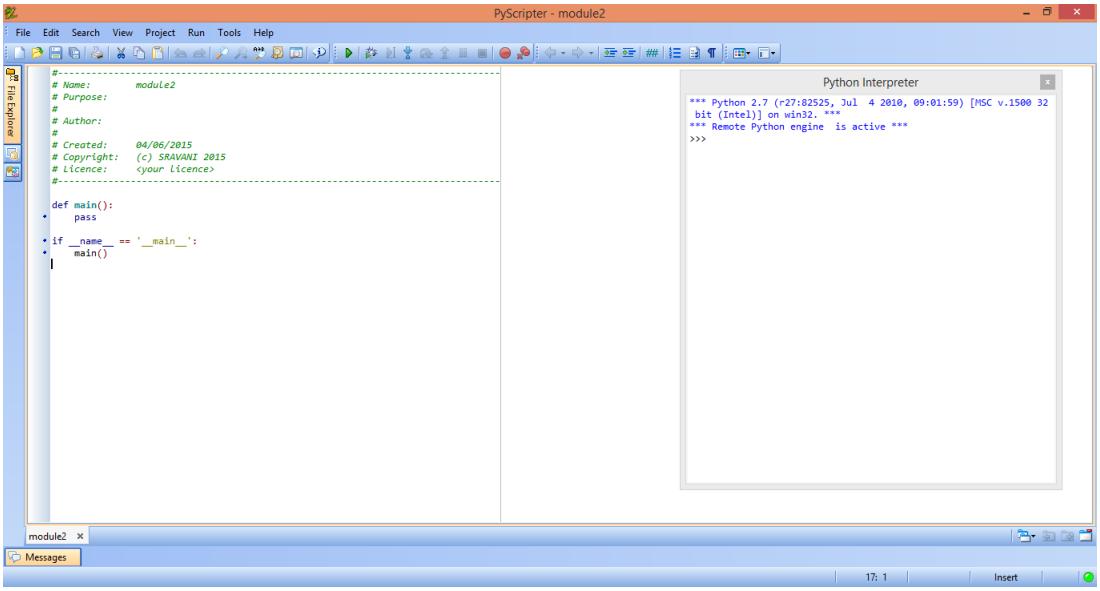

Figure 3.4:

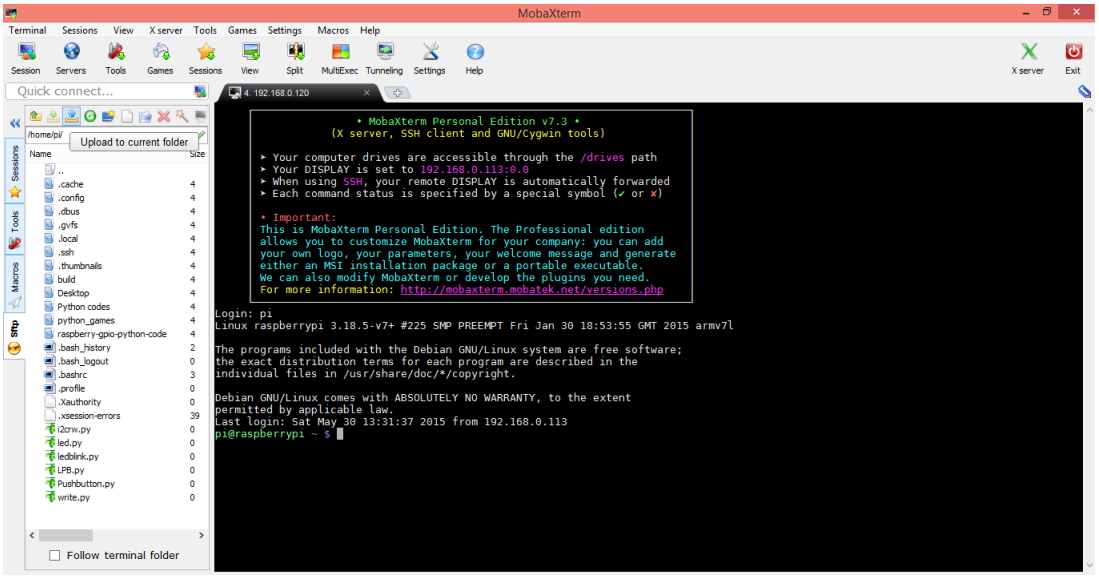

Figure 3.5:

(d) Upload that file wherein you have written your python program. Note: You can either upload your files to the home directory i.e. /home/pi/ or else you can create a new directory using the icons on the toolbar as illustrated before and upload your python file over there.

(e) After you have uploaded the code then type the following command on the terminal to execute the code *python filename.py*. In case you have uploaded the file to a directory other than home directory then change the path by typing *cd directoryname* and then *sudo python filename.py* (Note: In order to execute files form any directory other than home directory dont forget to use *cd* command before your directory name and textitsudo python command before your file name that you want to execute.)

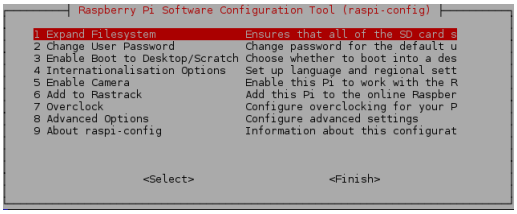

Figure 3.6:

- Using Notepad++ and LXTerminal(or MobaXterm):
  Notepad++ is a source code and a Windows text editor that is widely used by programmers.(It is more efficient than other text editors because it prompts you with indentation related errors that count in python programming)
  (a) Open the application and a create a new file.

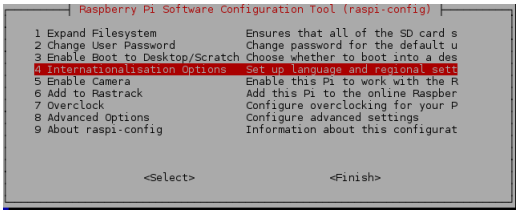

Figure 3.7:

(b) Type your python code and then save it as filename.py

(c) After that follow the steps(3-5) mentioned in the above method i.e using PScripter and MobaXterm.

- Using MobaXterm(for remote operation) or LXTerminal: In this method we can create a python file on the terminal window in the following way:

  (a) Open the MobaXterm or LXTerminal.

  (b) Type the command *touch filename.py* to create a file in a directory say home directory

  (c) Once the file is created in order to edit it type the command *nano filename.py*

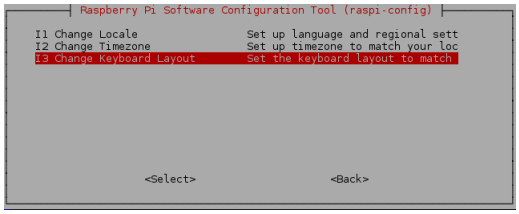

Figure 3.8:

(d) A blank file opens as shown

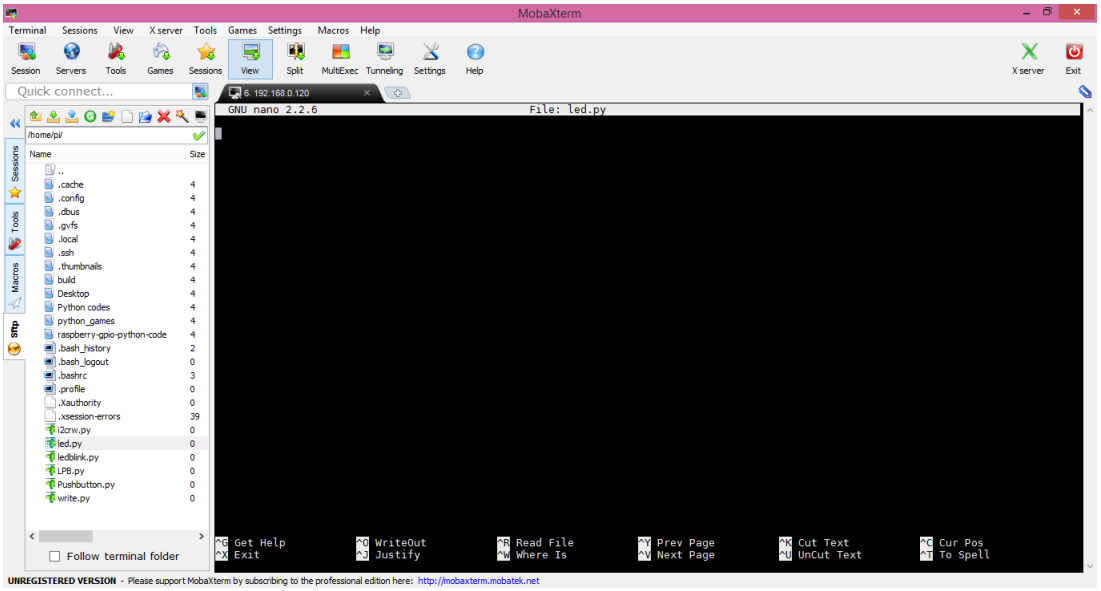

Figure 3.9:

(e) Type the required code and save the contents by typing *Ctrl + X, Y* (to save the file type Y)

(f) Then press enter to exit the editor onto terminal window.

(g) You can now execute the file by using the command *sudo python filename.py*(Don't forget to change the directory if your file is not saved in the home directory.PLease refer the steps mentioned before in the document)

Note: Just in case you want to view the contents of the file on the terminal window type *cat filename.py*

Now we are all set to write basic programs to access GPIO pins in your R-Pi

3.4 Experiments

3.4.1 Program to access GPIO pin of R-Pi for glowing LED by push button

- Anode of the LED is connected to pin no. 19
- Cathode of the LED is connected to a resistor(330 ohms) which is in turn connected to GND pin on R-Pi 2.
- One pin of the push button is connected to Ground
- The other pin of the push button is connected to pin no. 18

Code:-

```
import RPi.GPIO as GPIO #To control Pi GPIO channels
import time # time library to include sleep function

# to use Raspberry Pi GPIO pin numbers
GPIO.setmode(GPIO.BCM)
# the input pin(18) is  normally pulled up to 3.3V therefore we press
#the button a logic low or false value is returned at this pin
GPIO.setup(18, GPIO.IN, pull_up_down=GPIO.PUD_UP)

GPIO.setup(19,GPIO.OUT) #gpio pin 19 to whichled is connected
while True:
    input_state = GPIO.input(18) # variable to measure the
                                 # state of the input pin

    if input_state == False: # means button is pressed
        print('Button Pressed')
        GPIO.output(19,HIGH)
        time.sleep(0.2) # this is the min debouncing delay that
we
                        # give in order to ensure that the
                        # switch is definitely pressed

GPIO.cleanup()#to clear the state of GPIO pin given during program
```

3.4.2 IR sensors interfacing

The objective is to Interface the ADC MCP3008 to 8 IR sensors

Enabling SPI

Enabling using raspi-config
Run the following command:
Sudo raspi-config
This will launch the raspi-config utility. Select option 8 "Advanced Options".

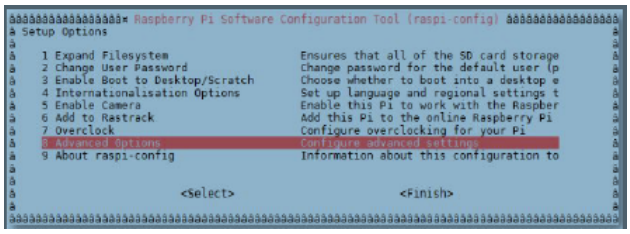

Figure 3.10:

Select the "SPI" option.

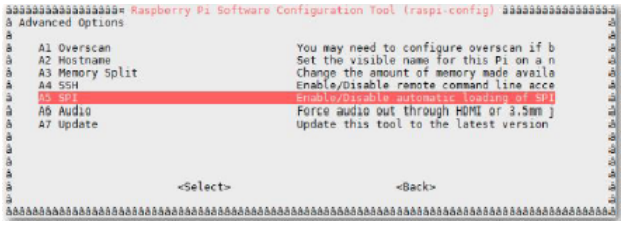

Figure 3.11:

Set the option to "Yes".

Select "OK".

Select "Finish".

Reboot for the changes to take effect :
sudo reboot
SPI is now enabled.

Installing SPI Wraper in python

In order to read data from the SPI bus in Python we can install a library called ‘py-spidev’. To install it we first need to install ‘python-dev’: sudo apt-get install python2.7-dev Then to finish we can download ‘py-spidev’ and compile it ready for use, run the following commands: wget https://github.com/Gadgetoid/py-spidev/archive/master.zip unzip master.zip rm master.zip cd py-spidev-master sudo python setup.py install cd .. You should now be ready to either communicate with add-on boards us- ing their own libraries (e.g. the PiFace) or other SPI devices.

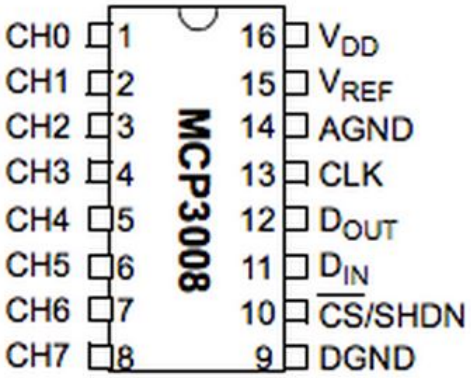

Figure 3.12: MCP3008 IC

IR sensors are connected to ADC 1 to the specific channels as shown :-

IR sensor number	ADC channel
IR 1	0
IR 2	1
IR 3	2
IR 4	3
IR 5	4
IR 6	5
IR 7	6
IR 8	7

Code:-

```
import spidev#spi library to control spi devices
import RPi.GPIO as GPIO #To control Pi GPIO channels
from time import sleep # time library to include sleep function


spi=spidev.SpiDev() # Open SPI bus  to create spi object
spi.open(0,0)# opened in bus 0 device 0

# Function to read SPI data from MCP3008 chip
# Channel must be an integer 0-7
# Function name :getADC
# Input : channel
# Output : analog data converted to digital in 10 bit resolution
# Example call: getADC(channel)
def getADC(channel):
    if(channel>7) or (channel<0):
        return -1
    r=spi.xfer([1,(8+channel) << 4,0])
    adc_out=((r[1]&3) << 8) + r[2]
    #print(adc_out)
    #if(adc_out<600 and adc_out>100):
        #print("1")
    #else:
        #print("0")

    print("adc output:{0:4d}".format(adc_out))


# Loop to get8 simultaneous analog values from 8 analog sensors
#and displaying the corresponding digital
#value using spi communication
while 1:
    for i in range(8):
        print("-----------------------------------------------------------------------")
        print("analogsensor")
        print(i)
        getADC(i)
        print("-----------------------------------------------------------------------")
    sleep(2)
```

3.4.3 LCD interface with port expander

In this tutorial we will be interfacing LCD with R-pi using port expander using port expander

Enabling I2C

Enabling using raspi-config
Run the following command:
Sudo raspi-config
This will launch the raspi-config utility. Select option 8 "Advanced Options".

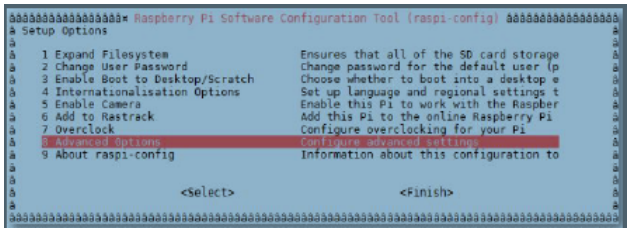

Figure 3.13:

Select the "I2C" option.

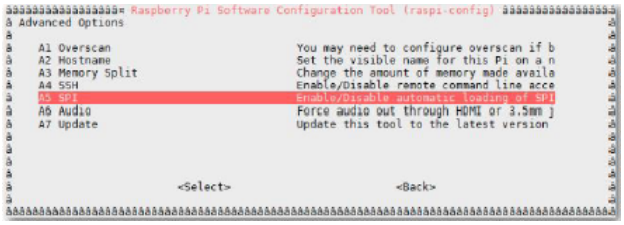

Figure 3.14:

Set the option to "Yes".

Select "OK".

Select "Finish".

Reboot for the changes to take effect :
sudo reboot
I2C is now enabled.

When you power up or reboot your Pi you can check the i2c module is

running by using the following command :
*lsmod — grep i2c_*
That will list all the modules starting with i2c_. If it lists i2c_bcm2708 then the module is running correctly.
Once you have connected your hardware double check the wiring. Make sure 3.3V is going to the correct pins and you have got not short circuits. Power up the Pi and wait for it to boot.
Type the command:
sudo i2cdetect -y 1
You should the output as:
LCD is connected to portA of port expander 1 inthe order :- vspace0.3cm

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

Figure 3.15: [4]

LCD pin	port expander 1 port A pin
RS	pA0
EN pA1	
D5	pA2
D4	pA3
D6	pA4
D7	pA5

Code:-

```
import smbus # python module to access i2c based interfaces
import time

bus = smbus.SMBus(1) # Rev 2 Pi uses 1
DEVICE = 0x20 # Device address (A0-A2)
IODIRA = 0x00 # Pin direction register
IODIRB = 0x01 # Pin direction register
OLATA  = 0x14 # Register for outputs
OLATB  = 0x15 # Register for outputs

# Function name :initialize_mcp
# Input : None
# Output : Initializes MCP23017 IC
# Example call: initialize_mcp()
def initialize_mcp():
    # all bits of IODIRA and IODIRB register are set to 0 meaning GPA
        # and GPB pins are outputs
    bus.write_byte_data(DEVICE,IODIRA,0x00)
    bus.write_byte_data(DEVICE,IODIRB,0x00)

    # Set all 7 output bits of port A and port B to 0
    bus.write_byte_data(DEVICE,OLATA,0)
    bus.write_byte_data(DEVICE,OLATB,0)

    return

# Function name :cmdset4bit
# Input : None
# Output : Sets the 16x2 LCD in 4 bit mode
# Logic : Command 0x30 is sent thrice and command 0x20 is sent
#         once to initialize the LCD
# Example call: cmdset4bit()
def cmdset4bit():
    time.sleep(1.0/1000)
    bus.write_byte_data(DEVICE,OLATB,0b00000000)
    bus.write_byte_data(DEVICE,OLATA,0b00110000)
    bus.write_byte_data(DEVICE,OLATB,0b00000100)
    time.sleep(5.0/1000)
    bus.write_byte_data(DEVICE,OLATB,0b00000000)

    time.sleep(1.0/1000)
    bus.write_byte_data(DEVICE,OLATA,0b00110000)
```

```
    bus.write_byte_data(DEVICE,OLATB,0b00000100)
    time.sleep(5.0/1000)
    bus.write_byte_data(DEVICE,OLATB,0b00000000)

    time.sleep(1.0/1000)
    bus.write_byte_data(DEVICE,OLATA,0b00110000)
    bus.write_byte_data(DEVICE,OLATB,0b00000100)
    time.sleep(5.0/1000)
    bus.write_byte_data(DEVICE,OLATB,0b00000000)

    time.sleep(1.0/1000)
    bus.write_byte_data(DEVICE,OLATA,0b00100000)
    bus.write_byte_data(DEVICE,OLATB,0b00000100)
    time.sleep(5.0/1000)
    bus.write_byte_data(DEVICE,OLATB,0b00000000)

    return

# Function name : convert
# Input : A list of hexadecimal values (commands or data)
# Output : The function returns a list with corresponding decimal
#          equivalent of the hexadecimal values
# Logic : A 2 digit hexadecimal value is separated into 1 digit
#         each and a zero is appended at the end of each digit
#         (eg: '0f' is converted to '00' and 'f0')
# Example call: convert(lst)
def convert(lst):
    fc = []
    l = len(lst)
    for i in range(0,l):
        j = lst[i]
        temp1 = j[0] + '0'
        temp2 = j[1] + '0'
        t = int(temp1,16) # returns the decimal form of a string (temp1)
        u = int(temp2,16)
        fc.append(t)
        fc.append(u)

    return fc # output list

# Function name : stringconvert
# Input : A string i.e. data to be displayed on LCD
# Output : The function returns a list with corresponding hexadecimal
#          equivalent of every character in a string
```

```
# Logic : This function converts a character in a string into hex
#         format and appends the converted hex value into a list for
#         further processing. (eg: 'A' is converted to '41' i.e. the
#         hex value of the character )
# Example call: stringconvert(s)
def stringconvert(s):
    s1 = []
    l1 = len(s)
    for i in range(0,l1):
        j = ord(string[i]) # ord() function returns the decimal
                           # equivalent of an ascii character
        f = hex(j)
        s1.append(f)

    l2 = len(s1)
    newlst = []
    for i in range (0,l2):
        s2 = s1[i]
        s3 = s2[2]
        s4 = s2[3]
        sf = s3 + s4
        newlst.append(sf)

    return newlst # output list

# Function name : conversion
# Input : 2 lists a data list and a command list
# Output : The function returns 2 lists 'data' and 'cmd' that
#         contain data and commands in the form that can be
#         directly given as a pin output of MCP23017 IC
# Example call: conversion(com,s)
def conversion(com,s):
    conv_string = stringconvert(s)
    cmd = convert(com)
    data = convert(conv_string)

    return data,cmd # output lists

# Function name : lcd_start
# Input : None
# Output : MCP23017 is initialized and lcd is set in 4 bit mode.
# Example call: lcd_start()
def lcd_start():
    initialize_mcp()
```

```
    time.sleep(5.0/100)
    cmdset4bit()

# Function name : commandwrt
# Input : A list with commands in converted form (hexadecimal to
#         decimal format)
# Output : Commands are sent to LCD display one by one
# Logic : For sending commands RS pin of an LCD is set to 0 and R/W
#         pin of an LCD is set to 0(for write operation) and a high
#         to low enable pulse is applied every time a command is sent
# Example call: commandwrt(cmd)
def commandwrt(cmd):
    l = len(cmd)
    bus.write_byte_data(DEVICE,OLATB,0b00000000)
    for i in range(0,l):
        time.sleep(1.0/1000)
        bus.write_byte_data(DEVICE,OLATA,cmd[i])
        bus.write_byte_data(DEVICE,OLATB,0b00000100)
        time.sleep(5.0/1000)
        bus.write_byte_data(DEVICE,OLATB,0b00000000)

    return

# Function name : datawrt
# Input : A list with data in converted form (string to decimal format)
# Output : Data (a string) is sent to LCD display one by one
# Logic : For sending data RS pin of an LCD is set to 1 and R/W pin
#         of an LCD is set to 0(for write operation) and a high to
#         low enable pulse is applied every time data is sent
# Example call: datawrt(data)
def datawrt(data):
    l = len(data)
    for i in range(0,l):
        time.sleep(1.0/1000)
        bus.write_byte_data(DEVICE,OLATA,data[i])
        bus.write_byte_data(DEVICE,OLATB,0b00000101)
        time.sleep(5.0/1000)
        bus.write_byte_data(DEVICE,OLATB,0b00000001)

    return

# Function name : lcdclear
# Input : None
# Output : LCD screen gets cleared(command = 0x01)
```

```
# Example call: lcdclear()
def lcdclear():
    time.sleep(1.0/1000)
    bus.write_byte_data(DEVICE,OLATA,0b00000000)
    bus.write_byte_data(DEVICE,OLATB,0b00000100)
    time.sleep(5.0/1000)
    bus.write_byte_data(DEVICE,OLATB,0b00000000)

    bus.write_byte_data(DEVICE,OLATA,0b00010000)
    bus.write_byte_data(DEVICE,OLATB,0b00000100)
    time.sleep(5.0/1000)
    bus.write_byte_data(DEVICE,OLATB,0b00000000)

try:
  lcd_start()
  com = ['28','01','0f','06'] # a list of commands
  s = "ABCD" # data to be displayed

  data,cmd = conversion(com,s)
  commandwrt(cmd)
  datawrt(data)


except KeyboardInterrupt:
        pass
        lcdclear()
```

3.4.4 Xbee Communication

In this experiment we will study to enable the serial communication through XBEE.

Enabling Serial communication in Pi
Refer to link :- Enabling Guide[10]

Code:-

```
import RPi.GPIO as GPIO # import GPIO librery
from time import sleep # time library to include sleep function
import threading # importing threading for parallel programming to avoid

GPIO.setmode(GPIO.BCM)
import serial

GPIO.setwarnings(False)
Motor1A = 13# set GPIO-13 as  motor1 pin 1
Motor1B = 19 # set GPIO-19 as motor1 pin 2
Motor1E = 26 # set GPIO-26 as Enable pin of motor1

GPIO.cleanup()
GPIO.setup(Motor1A,GPIO.OUT)# making motor pin1 as output
GPIO.setup(Motor1B,GPIO.OUT)# making motor pin2 as output
GPIO.setup(Motor1E,GPIO.OUT)# making enable as output


# Function for  executing the threading
# Function name :threadfunc
# Input : a(any variable)
# Example call: t2 = threading.Thread(target = threadfunc, args = (1,))
def thread_func(a):
    while 1:
        ser=serial.Serial('/dev/ttyAMA0')
        ser.baudrate=9600
        ser.write("a".encode())
        sleep(1)
        ser.write("b".encode())
        sleep(1)

pwm=GPIO.PWM(26,100) # configuring Enable pin means GPIO-04 for PWM
pwm.start(50) # starting it with 50% dutycycle
```

```
# calling of thread function
t=threading.Thread(target=thread_func,args=(1,))
t.start()

while 1:
    pwm.ChangeDutyCycle(30)
    GPIO.output(Motor1A,GPIO.HIGH)
    GPIO.output(Motor1B,GPIO.HIGH)
    GPIO.output(Motor1E,GPIO.HIGH)

    # this will run your motor in forward direction for 2 seconds with 50
    sleep(0.001)
```

3.4.5 Turning off pi with press of push button

In order to protect our SD card from getting corrupted we should shut down the Raspberry-Pi properly instead of powering it off directly.
So in order to do that we have put a intterupt switch on the Adaptor Board .When we will press the switch the Raspberry pi will be Switched off or Shut Down properly . We are using a Pull up register for checking the Input,so if button is pressed then output will be 0 as we are using it in pull up configuration
We have connected the interupt switch to GPIO 20 of raspberry pi.

Code:-

```
import RPi.GPIO as GPIO #To control Pi GPIO channels
import time # time library to include sleep function
from subprocess import call
# to use Raspberry Pi GPIO pin numbers
GPIO.setmode(GPIO.BCM)

GPIO.setup(20, GPIO.IN, pull_up_down=GPIO.PUD_UP) # the input pin(20) is
# normally pulled up to 3.3V therefore when we press the button a logic
# low or false value is returned at this pin

while True:
    input_state = GPIO.input(20) # variable to measure the
                                 #state of the input pin
    if input_state == False: # means button is pressed
        print('Button Pressed')
        call("sudo shutdown -h now",shell=True)
```

3.4.6 Servo motors

We have given 3 ports on adaptor board to control the 3 servo motors and their enable pins are connected to the following Raspberry-pi pins:-

Servo motor	Raspberry pin(GPIO)
Servo1	18
Servo2	23
Servo3	24

Code:-

```
import RPi.GPIO as GPIO # module to control Pi GPIO channels
import time

GPIO.setmode(GPIO.BOARD) # to use Raspberry Pi board pin numbers
i1= 18 # servo motor enable
GPIO.setup(i1,GPIO.OUT,initial=GPIO.LOW) # set up GPIO output channel i.e
                                         # on pin 11 and initially output

# Function name : rotation
# Input : Angle in degrees
# Output : On time and off time of a pwm pulse for angle = x degrees
# Example call: rotation(x)
def rotation(x):
    m = (2.2 - 0.6)/(180 - 0) # 0.6ms corresponds to 0 degree and 2.2ms c
    on_time = 0.6 + m*x
        off_time = 20 - on_time # time period of a pwm pulse for a servo
        return on_time,off_time

try:
    while 1:
                angle = 90 # it can range from 0 to 360 degrees
                on_time,off_time = rotation(angle)
                GPIO.output(i1,True)
            time.sleep(on_time/1000) # time in ms
                GPIO.output(i1,False)
                time.sleep(off_time/1000)


except KeyboardInterrupt:
        pass
GPIO.cleanup() # all GPIO pins are cleared
```

Chapter 4

References

1. https://www.raspberrypi.org/
2. http://en.wikipedia.org/wiki/Raspberry_Pi
3. https://www.raspberrypi.org/wp-content/uploads/2014/11/Raspberry_Pi_Family_A-annotated-15001.jpg
4. http://assets.windowsphone.com/3f82dfe6-a179-4ddf-9738-91989190c3fa/IoT-rpi2-board_InvariantCulture_Default.png
5. http://en.wikipedia.org/wiki/Secure_Shell
6. http://www.webopedia.com/TERM/S/SSH.html
7. http://www.codemastershawn.com/library/tutorial/images/ssh.tunnel.overview.gif
8. http://mobaxterm.mobatek.net/
9. http://www.suntimebox.com/raspberry-pi-tutorial-course/week-3/day-5/
10. https://electrosome.com/uart-raspberry-pi-python/