RV College of Engineering®

Go, change the world

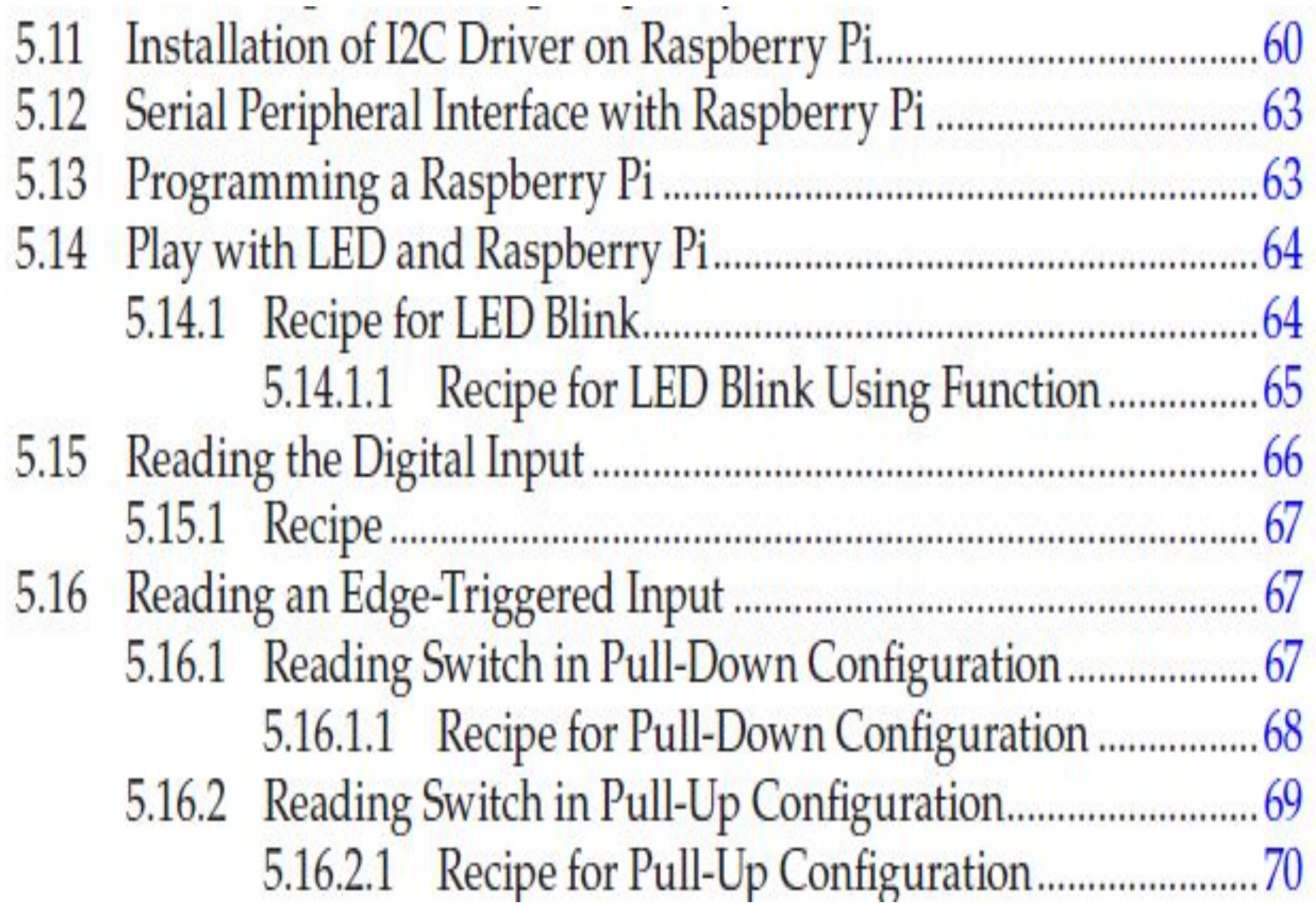
IOT and APPLICATIONS IS224AI-UG 4th sem 2022 scheme

Unit-III

Programming a Raspberry Pi

Reference Book: Internet Of Things With Raspberry Pi And Arduino, Rajesh Singh, Anita Gehlot, Lovi Raj Gupta, Bhupendra Singh, and Mahendra Swain, CRC Press, Taylor & Francis Group, 2020, ISBN: 13: 978-0-367-24821-5

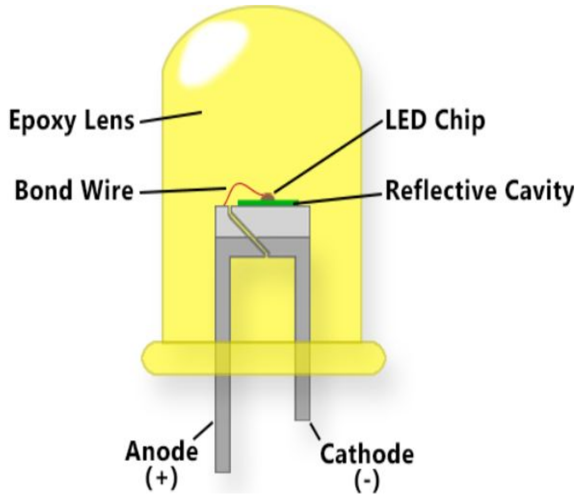
By,
Dr. G S Mamatha
Professor & Associate Dean(PG Studies)
Department of ISE
R V College of Engineering

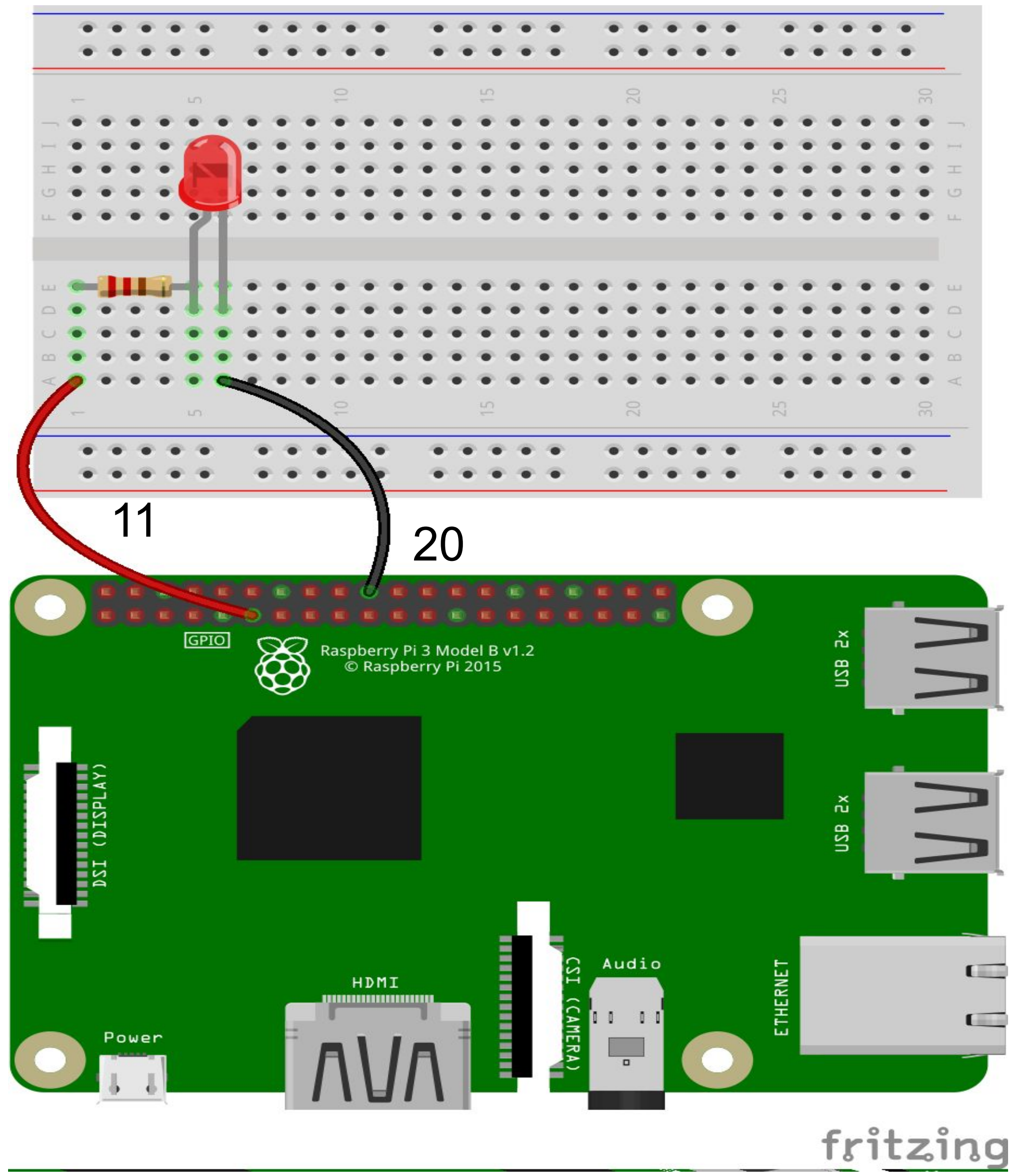


5.11 Installation of I2C Driver on Raspberry Pi 60
5.12 Serial Peripheral Interface with Raspberry Pi 63
5.13 Programming a Raspberry Pi 63
5.14 Play with LED and Raspberry Pi 64
5.14.1 Recipe for LED Blink 64
5.14.1.1 Recipe for LED Blink Using Function 65
5.15 Reading the Digital Input 66
5.15.1 Recipe 67
5.16 Reading an Edge-Triggered Input 67
5.16.1 Reading Switch in Pull-Down Configuration 67
5.16.1.1 Recipe for Pull-Down Configuration 68
5.16.2 Reading Switch in Pull-Up Configuration 69
5.16.2.1 Recipe for Pull-Up Configuration 70

Controlling LED with Raspberry Pi

- You'll need the following components to connect the circuit.
- 1. Raspberry Pi
  2. LED
  3. Resistor - 330 ohm
  4. Breadboard
  5. Jumper Wires





RV College of Engineering®

Controlling LED with Raspberry Pi

Go, change the world

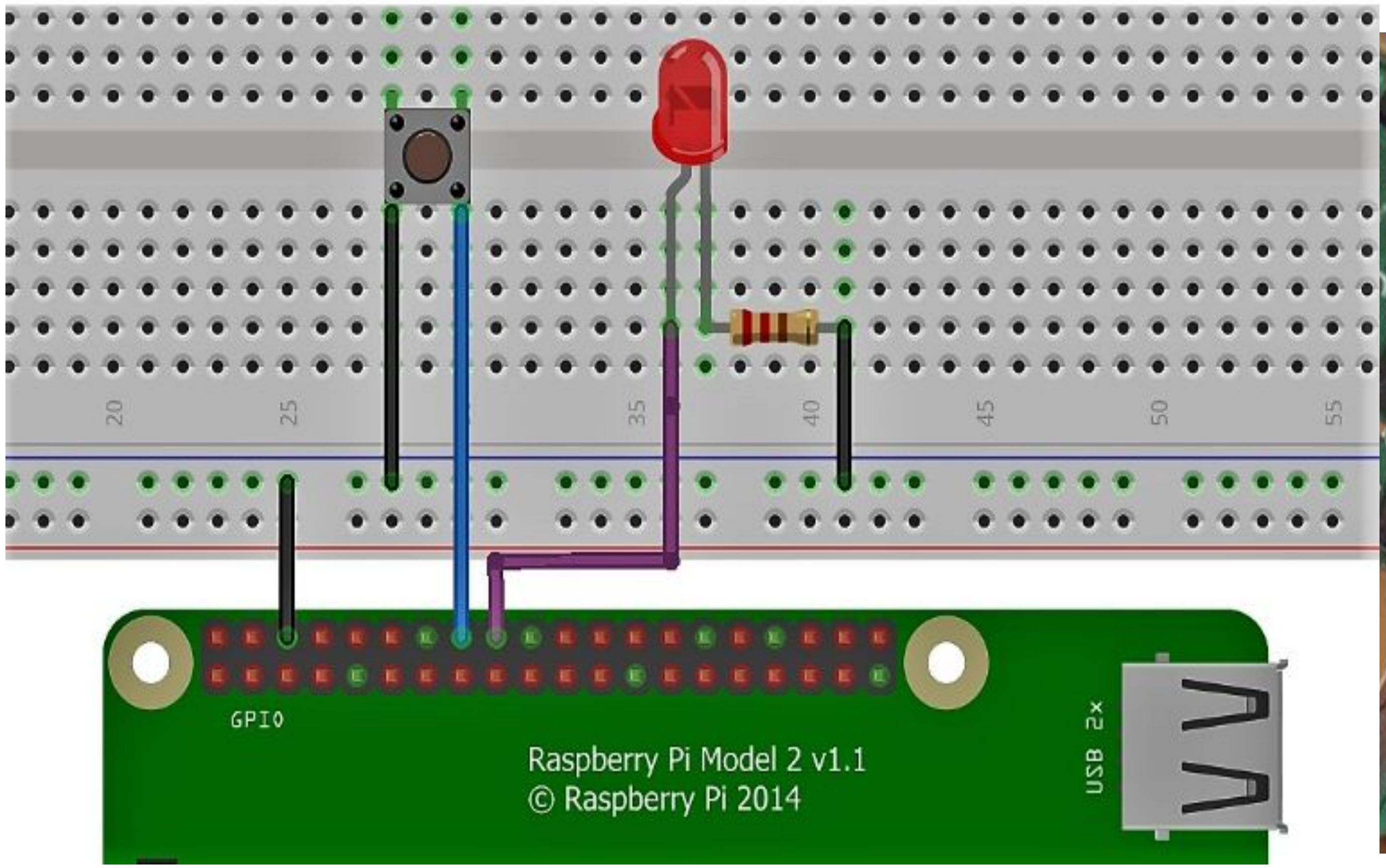
- Connect the circuit:

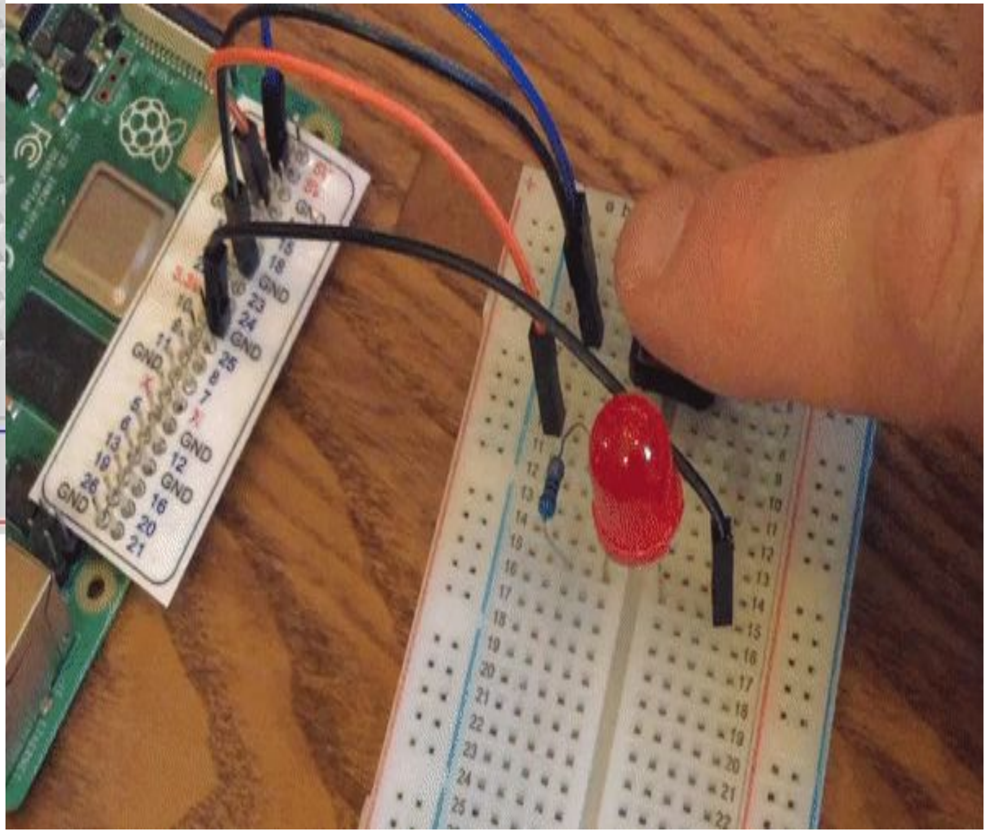
1. Use a jumper wire to connect the ground (Pin 3) of GPIO to rail marked in blue on the breadboard.
2. Connect the resistor from the same row on the breadboard to a column on the breadboard.
3. Connect the LED with the cathode in the same row as the resistor. Insert the anode in the adjacent row.
4. Use another jumper cable to connect the GPIO Pin 21 (3.3 V) in the same row as the anode of LED.

Controlling LED with Raspberry Pi

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(21,GPIO.OUT)
print "LED on"
GPIO.output(21,GPIO.HIGH)
time.sleep(10)
print "LED off"
GPIO.output(21,GPIO.LOW)
```

Controlling LED with button in Raspberry Pi





Reading the Digital Input

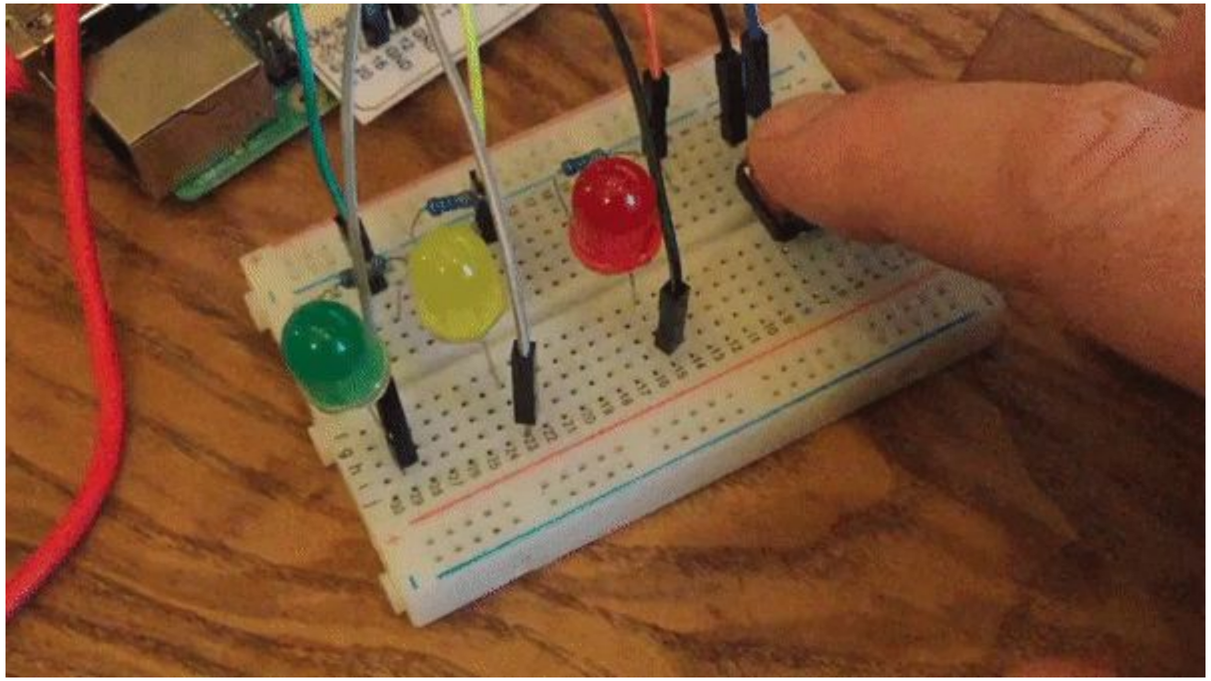
- The digital inputs at Raspberry Pi can be read by two methods named as pull down and pull up, by setting the GPIO pins as active LOW or active HIGH output.
  - These commands enable a pull-down resistor on pin 23 and a pull-up resistor on pin 24.
  - The Pi is looking for a high voltage on pin 23 and a low voltage on pin 24. These are required to define in the loop, so that these can constantly check the pin voltage.
  - To understand the concept, consider a small program for switch.

```
importRPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)                                          # use pi in BCM mode
GPIO.setup(23, GPIO.IN, pull_up_down = GPIO.PUD_DOWN)           # set pin as input
GPIO.setup(24, GPIO.IN, pull_up_down = GPIO.PUD_UP)                 # set pin as input
while True:
if(GPIO.input(23) ==1):
print("pressed button 1")                                       # print string on terminal
if(GPIO.input(24) == 0):
print("pressed button2")                                        # print string on terminal
GPIO.cleanup()                                                  # clean all GPIOs
```

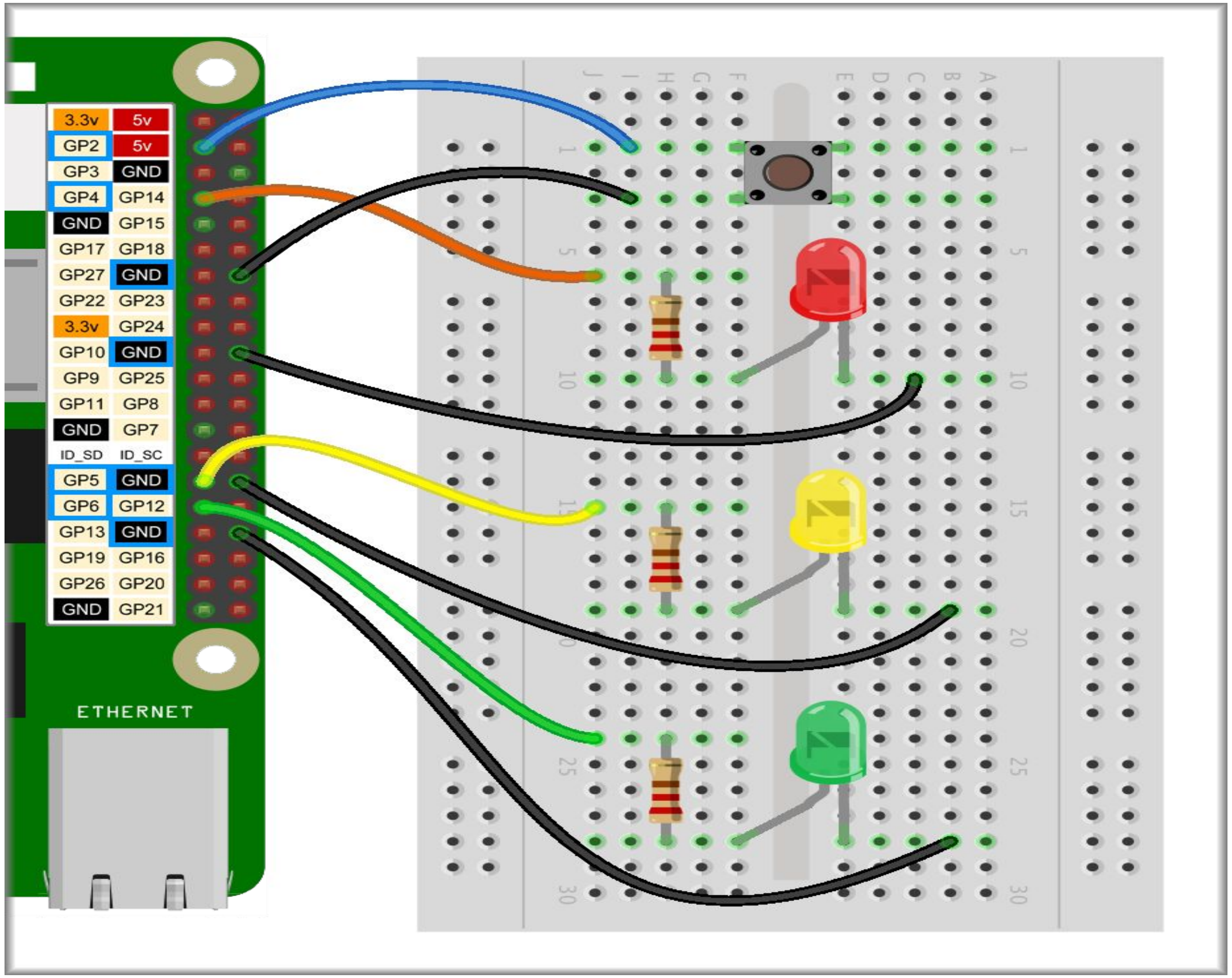
Controlling LED with button in Raspberry Pi

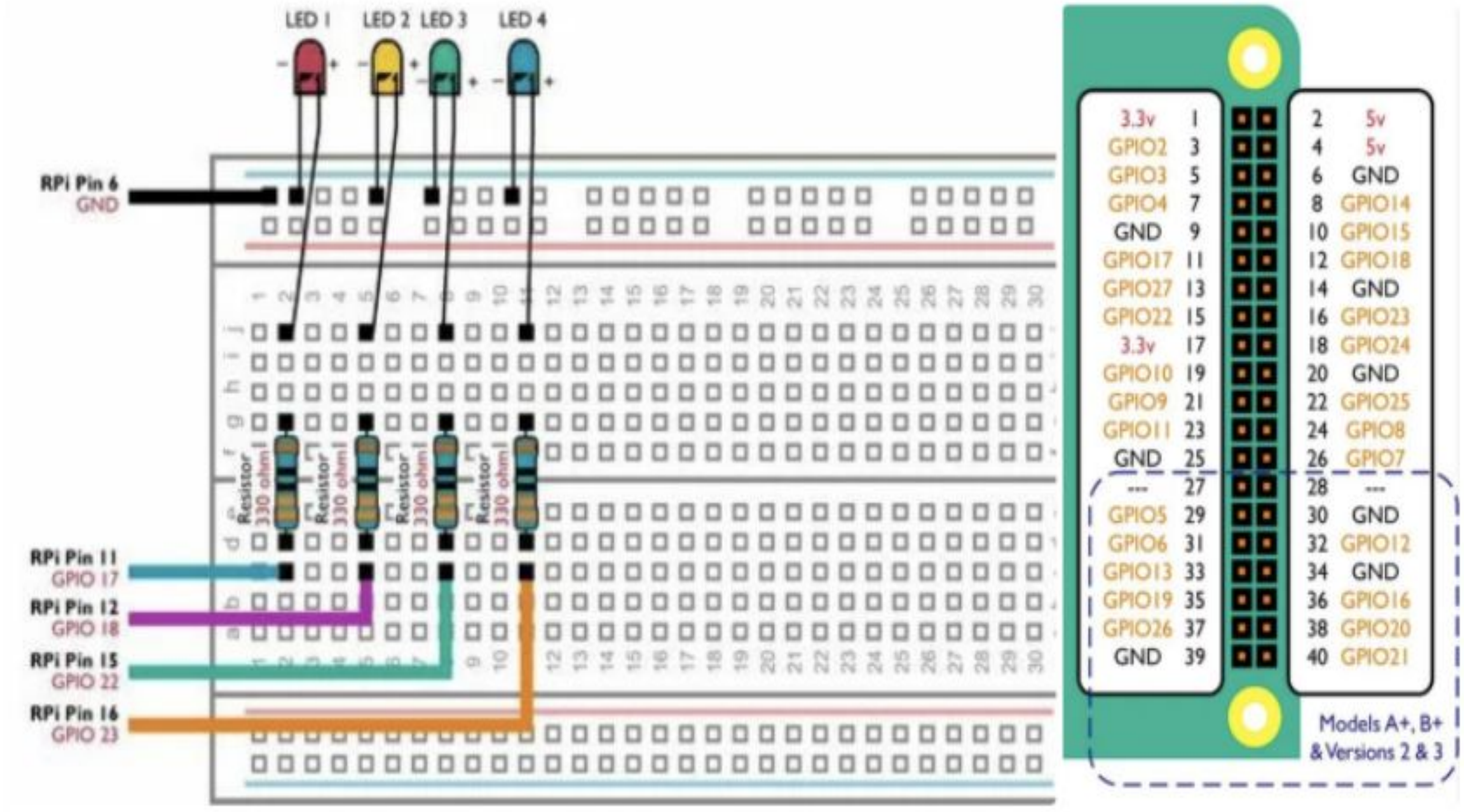
```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
GPIO.setup(23, GPIO.IN, pull_up_down=GPIO.PUD_UP)       #Button to GPIO23, Pull-Up/Down Resistors
GPIO.setup(24, GPIO.OUT)                                #LED to GPIO24

try:
    while True:
         button_state = GPIO.input(23)
         if button_state == False:
             GPIO.output(24, True)
             print('Button Pressed...')
             time.sleep(0.2)
         else:
             GPIO.output(24, False)
except:
    GPIO.cleanup()
```

Challenge which LED???



```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BOARD)
GPIO.setup(11, GPIO.OUT)
GPIO.setup(12, GPIO.OUT)
GPIO.setup(15, GPIO.OUT)
GPIO.output(11, True)
time.sleep(3)
GPIO.output(11, False)
time.sleep(1)
GPIO.output(12, True)
time.sleep(3)
GPIO.output(12, False)
time.sleep(1)
GPIO.output(15, True)
time.sleep(3)
GPIO.output(15, False)
time.sleep(1)
```




LED 1
LED 2
LED 3
LED 4
RPi Pin 6
GND
RPi Pin 11
GPIO 17
RPi Pin 12
GPIO 18
RPi Pin 15
GPIO 22
RPi Pin 16
GPIO 23
Resistor 330 ohm
3.3v 1
GPIO2 3
GPIO3 5
GPIO4 7
GND 9
GPIO17 11
GPIO27 13
GPIO22 15
3.3v 17
GPIO10 19
GPIO9 21
GPIO11 23
GND 25
--- 27
GPIO5 29
GPIO6 31
GPIO13 33
GPIO19 35
GPIO26 37
GND 39
2 5v
4 5v
6 GND
8 GPIO14
10 GPIO15
12 GPIO18
14 GND
16 GPIO23
18 GPIO24
20 GND
22 GPIO25
24 GPIO8
26 GPIO7
28 ---
30 GND
32 GPIO12
34 GND
36 GPIO16
38 GPIO20
40 GPIO21
Models A+, B+ & Versions 2 & 3

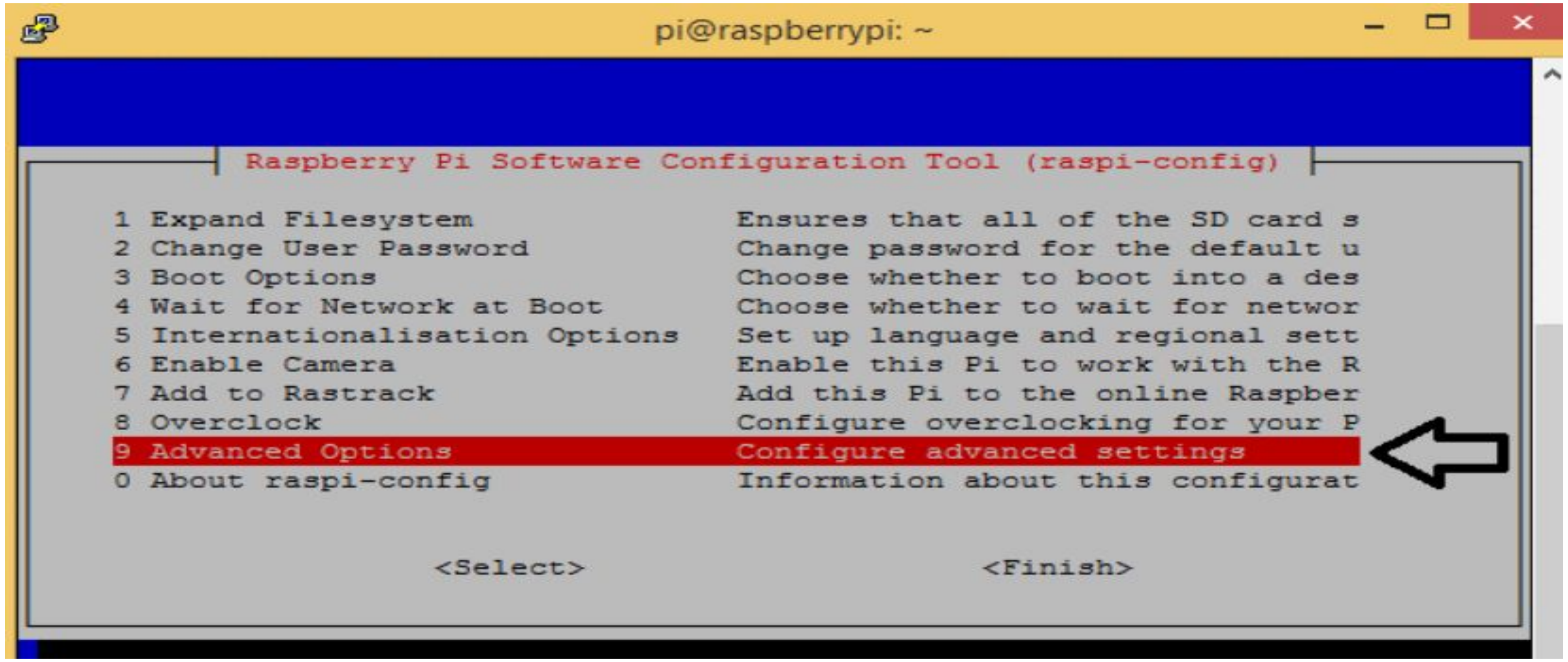
RV College of Engineering®

I2C

Go, change the world

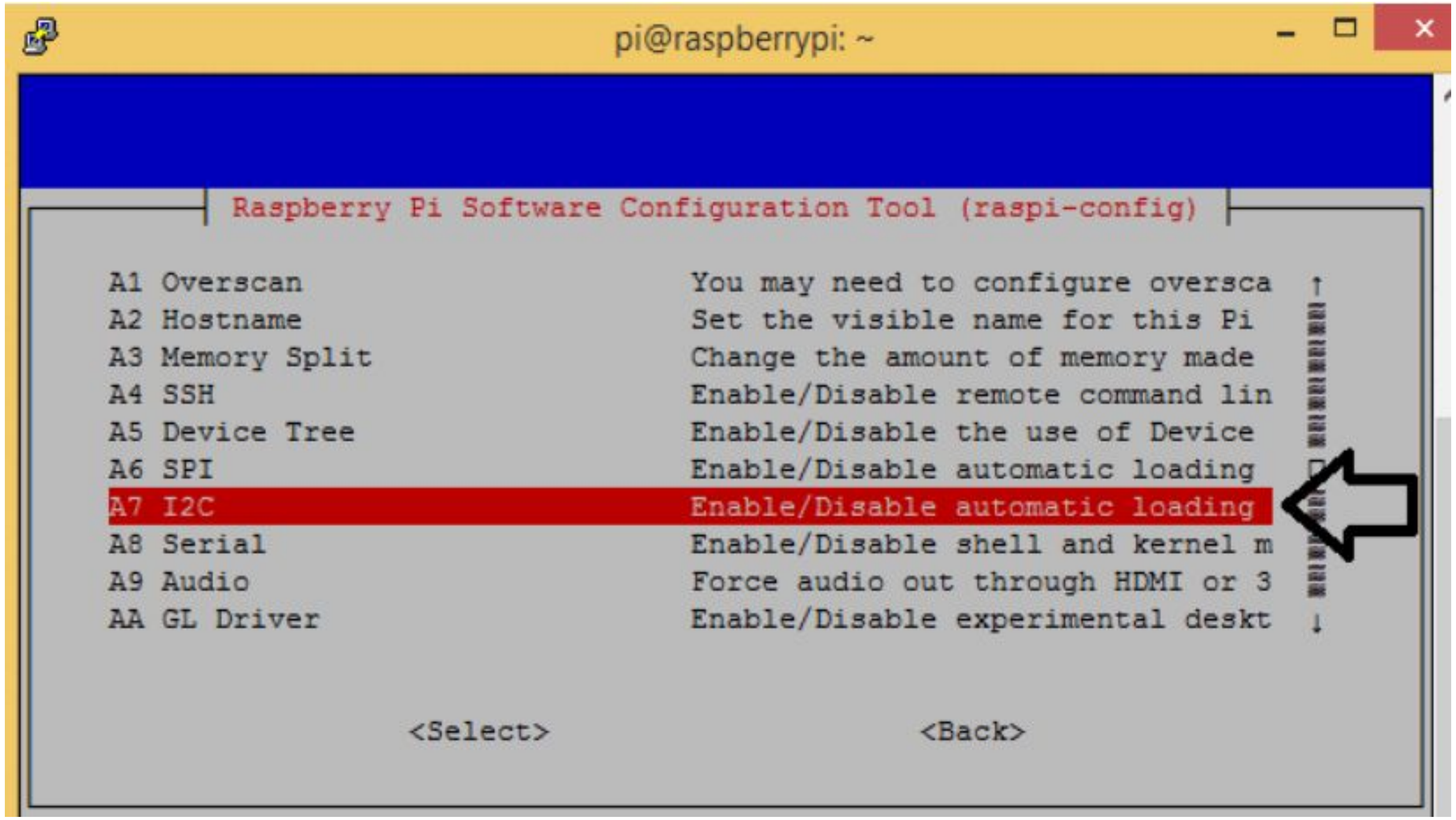
- **ENABLE I2C ON THE PI**
- Before we get into the programming, we need to make sure the I2C module is enabled on the Pi and install a couple tools that will make it easier to use I2C.

- **ENABLE I2C IN RASPI-CONFIG**
- First, log in to your Pi and enter sudo raspi-config to access the configuration menu. Then arrow down and select “Advanced Settings”:



```
pi@raspberrypi: ~

Raspberry Pi Software Configuration Tool (raspi-config)

1 Expand Filesystem             Ensures that all of the SD card s
2 Change User Password          Change password for the default u
3 Boot Options                  Choose whether to boot into a des
4 Wait for Network at Boot      Choose whether to wait for networ
5 Internationalisation Options  Set up language and regional sett
6 Enable Camera                 Enable this Pi to work with the R
7 Add to Rastrack               Add this Pi to the online Raspber
8 Overclock                     Configure overclocking for your P
9 Advanced Options              Configure advanced settings
0 About raspi-config            Information about this configurat

            <Select>                        <Finish>
```

- Choose “Yes” at the next prompt, exit the configuration menu, and reboot the Pi to activate the settings.



```
Raspberry Pi Software Configuration Tool (raspi-config)

A1 Overscan                        You may need to configure oversca
A2 Hostname                        Set the visible name for this Pi
A3 Memory Split                    Change the amount of memory made
A4 SSH                             Enable/Disable remote command lin
A5 Device Tree                     Enable/Disable the use of Device
A6 SPI                             Enable/Disable automatic loading
A7 I2C                             Enable/Disable automatic loading
A8 Serial                          Enable/Disable shell and kernel m
A9 Audio                           Force audio out through HDMI or 3
AA GL Driver                       Enable/Disable experimental deskt

              <Select>                         <Back>
```

Choose “Yes” at the next prompt, exit the configuration menu, and reboot the Pi to activate the settings.

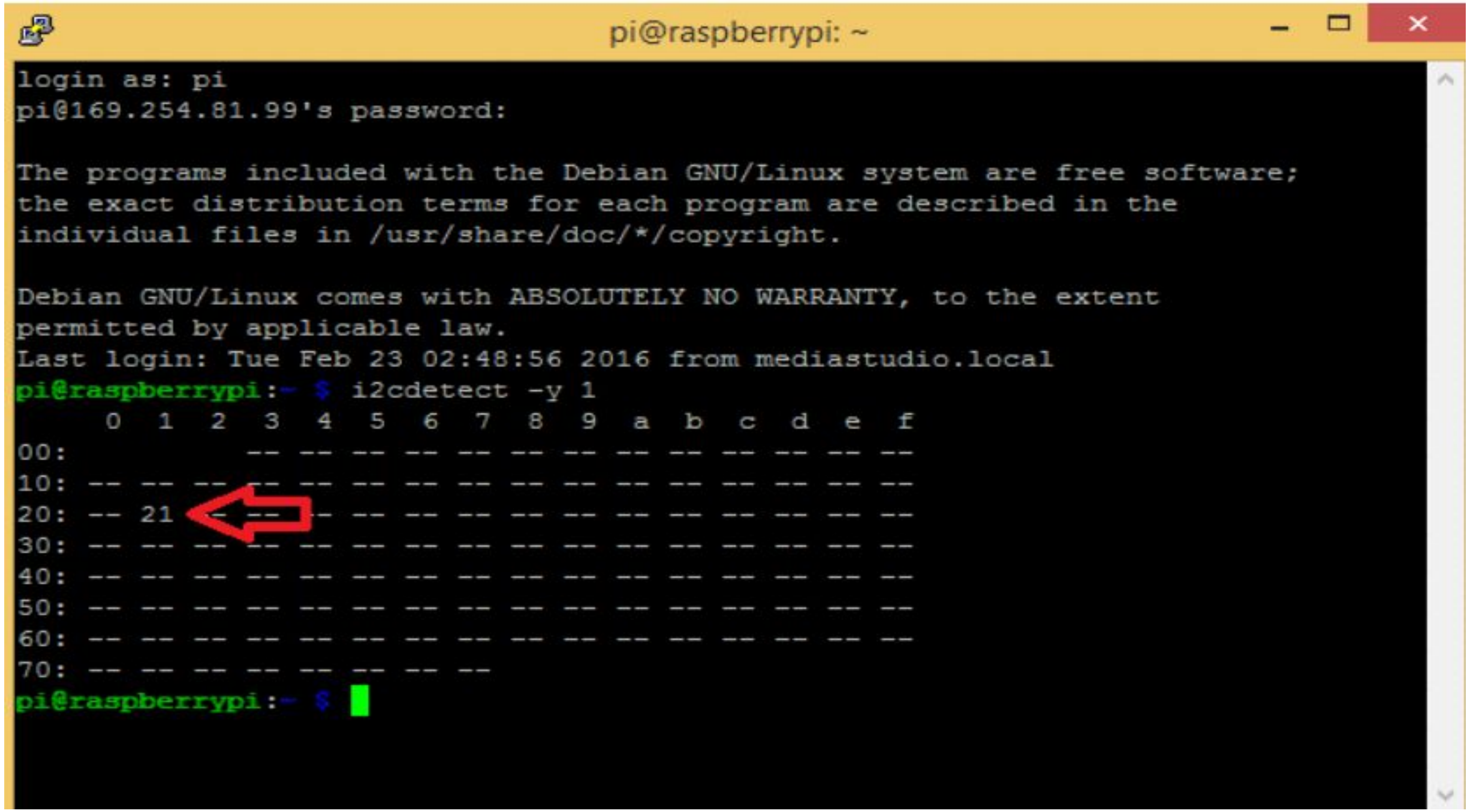
INSTALL I2C-TOOLS AND SMBUS:

- Now we need to install a program called I2C-tools, which will tell us the I2C address of the LCD when it's connected to the Pi. So at the command prompt, enter
- **>sudo apt-get install i2c-tools**

- Next we need to install SMBUS, which gives the Python library we're going to use access to the I2C bus on the Pi. At the command prompt, enter
- **sudo apt-get install python-smbus**

- Now reboot the Pi and log in again. With your LCD connected, enter
- **>i2cdetect -y 1** at the command prompt.
- This will show you a table of addresses for each I2C device connected to your Pi:

- The I2C address of my LCD is 21. Take note of this number, we'll need it later.



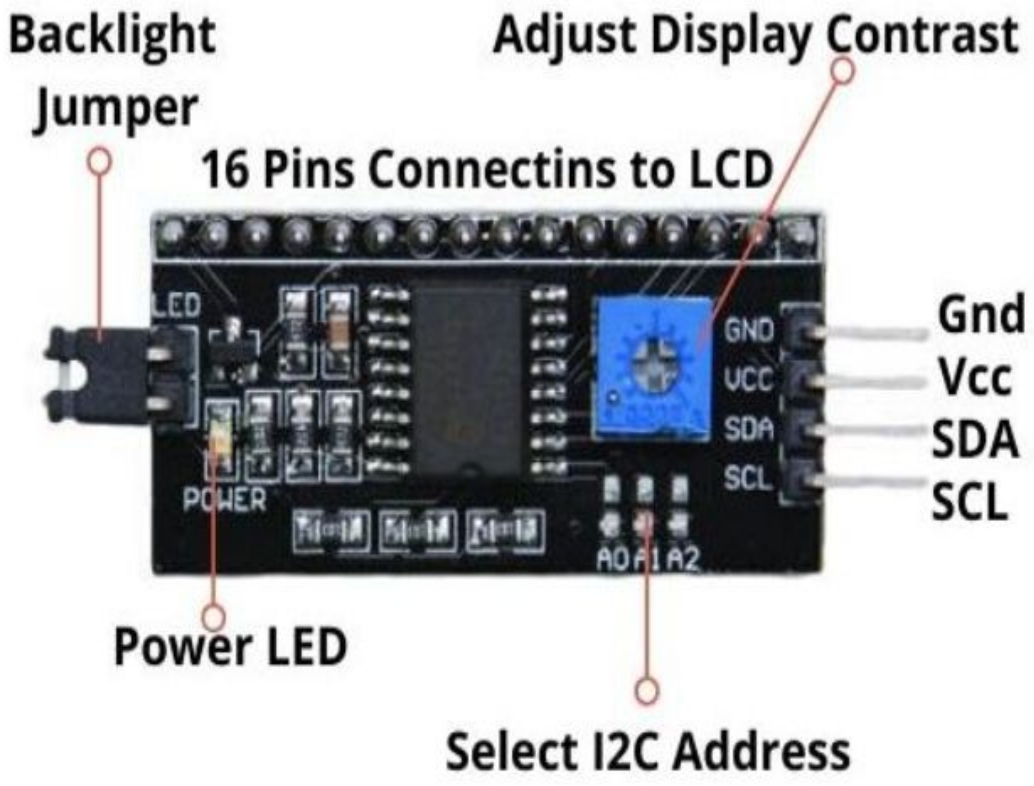
pi@raspberrypi: ~

```
login as: pi
pi@169.254.81.99's password:

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Feb 23 02:48:56 2016 from mediastudio.local
pi@raspberrypi:~ $ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- 21 -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
pi@raspberrypi:~ $
```

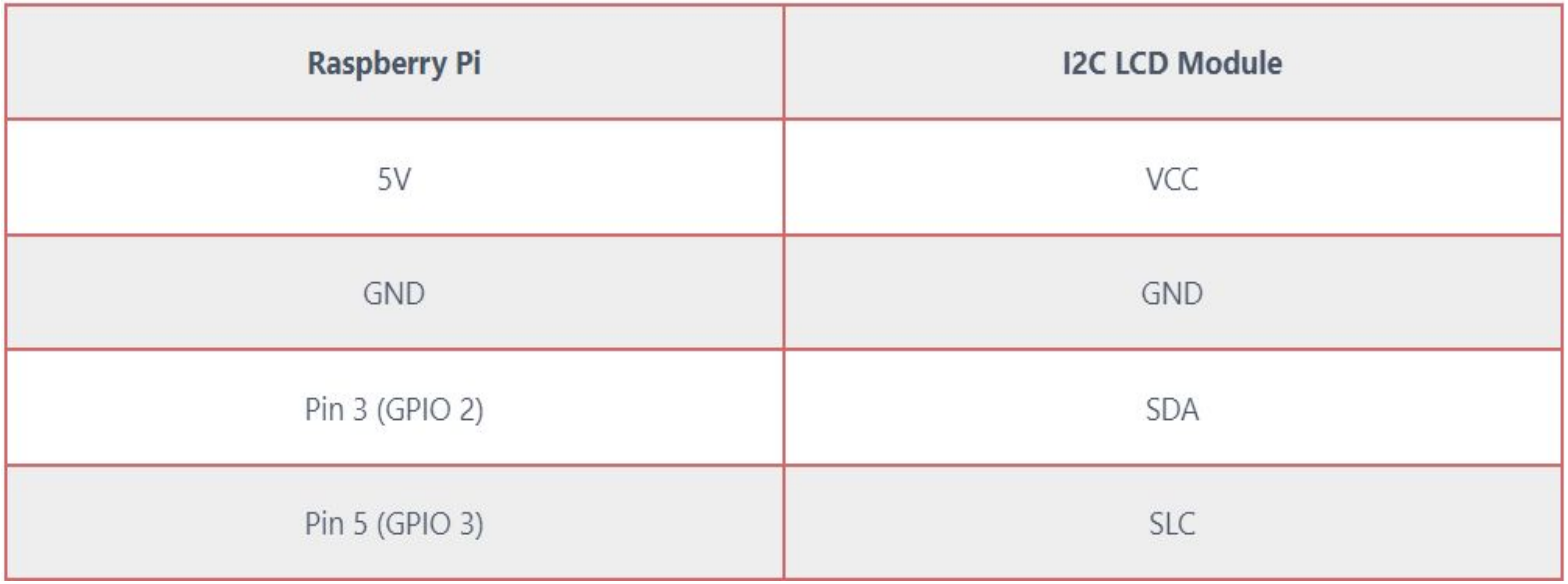
I2C

- I2C stands for inter-integrated circuit and is a method designed to allow one chip to talk to another synchronously.
- The Raspberry Pi features in-built support for the I2C protocol allowing it to connect and talk with a variety of I2C capable circuits.
- **Example: Interfacing 16×2 LCD with Raspberry Pi**
- **Components Required**
- The components you will be required for Raspberry pi LCD display interfacing are as follows
  - Raspberry Pi
  - LCD 16×2
  - Potentiometer 4.7 KΩ
  - Resistor 330 Ω
  - Jumper cables
  - I2C LCD Module

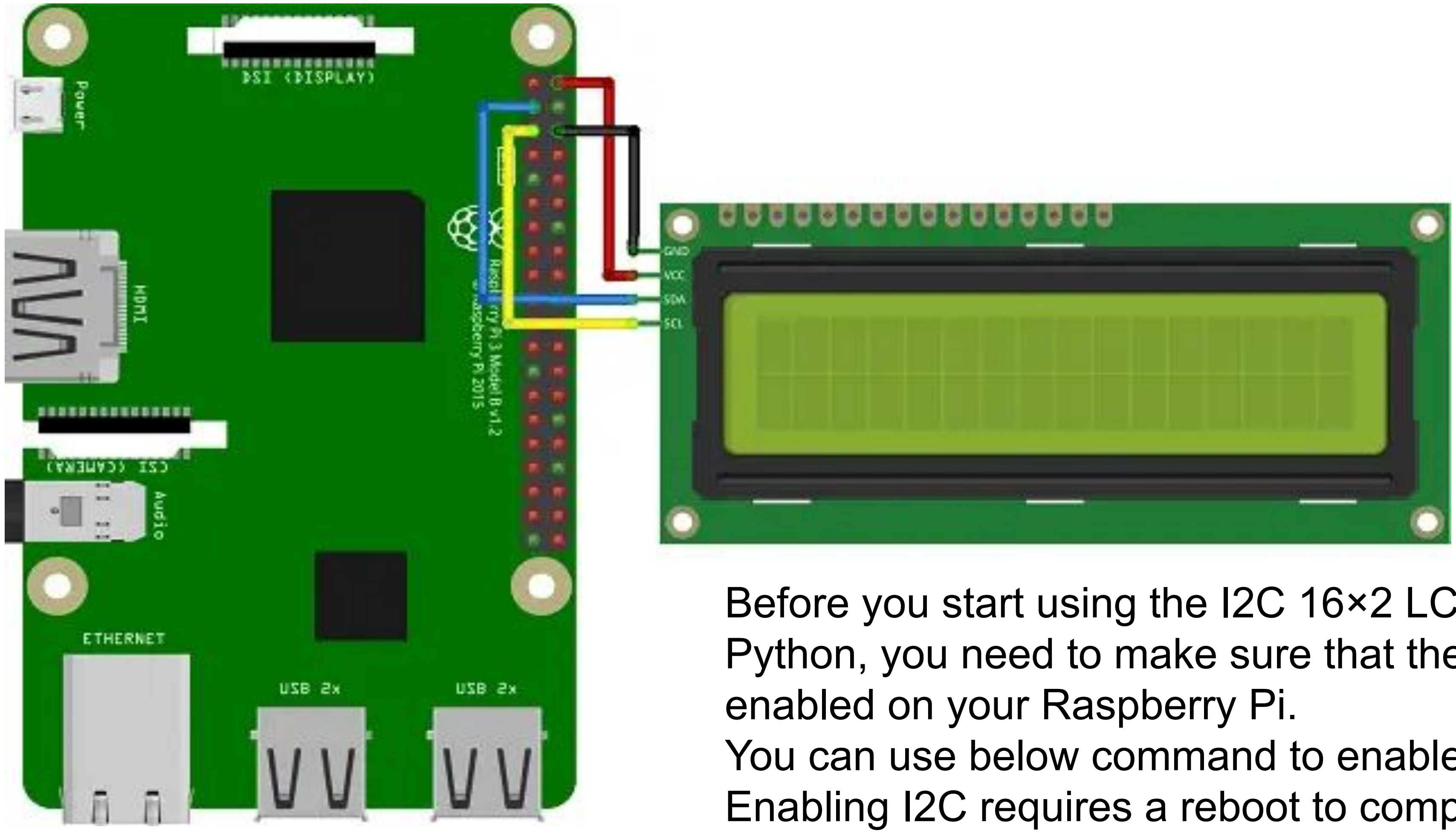
- The backpack module uses the I-squared-C (or **I2C**) protocol to communicate with the Raspberry Pi, which uses **only two wires: SDA and SCL (data and clock).**
- LCD display requires 5V to power and display and it will be powered by Raspberry Pi.
- For sending the data to LCD from Raspberry Pi, I2C protocol will be used.
- It is safe to connect such display to the Raspberry Pi directly.



Connections



Raspberry Pi	I2C LCD Module
5V	VCC
GND	GND
Pin 3 (GPIO 2)	SDA
Pin 5 (GPIO 3)	SLC



Before you start using the I2C 16×2 LCD display with Python, you need to make sure that the I2C protocol is enabled on your Raspberry Pi.
You can use below command to enable this protocol.
Enabling I2C requires a reboot to completely enable it.

```
sudo raspi-config
```

```
from rpi_lcd import LCD
lcd = LCD()
lcd.text(“Hello,”, 1)
    lcd.text(“Raspberry Pi!”, 2)
lcd.clear()
  lcd.text(“Hello,”, 1)
    lcd.text(“Raspberry Pi!”, 2)
```

□

Sample

```
import I2C_LCD_driver
from time import *

mylcd = I2C_LCD_driver.lcd()

mylcd.lcd_display_string("Hello World!", 1)
```

Blinking text

```
import time
import I2C_LCD_driver
mylcd = I2C_LCD_driver.lcd()

while True:
    mylcd.lcd_display_string(u"Hello world!")
    time.sleep(1)
    mylcd.lcd_clear()
    time.sleep(1)
```

Print date and time

```
import I2C_LCD_driver
import time
mylcd = I2C_LCD_driver.lcd()


while True:
    mylcd.lcd_display_string("Time: %s" %time.strftime("%H:%M:%S"), 1)

    mylcd.lcd_display_string("Date: %s" %time.strftime("%m/%d/%Y"), 2)
```

```
import I2C_LCD_driver
import socket
import fcntl
import struct

mylcd = I2C_LCD_driver.lcd()

def get_ip_address(ifname):
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    return socket.inet_ntoa(fcntl.ioctl(
        s.fileno(),
        0x8915,
        struct.pack('256s', ifname[:15])
    )[20:24])
mylcd.lcd_display_string("IP Address:", 1)
mylcd.lcd_display_string(get_ip_address(‘wlan0'), 2)
```

Scrolling text to right continuously

```
import I2C_LCD_driver
from time import *

mylcd = I2C_LCD_driver.lcd()

str_pad = " " * 16
my_long_string = "This is a string that needs to scroll"
my_long_string = str_pad + my_long_string

while True:
    for i in range (0, len(my_long_string)):
        lcd_text = my_long_string[i:(i+16)]
        mylcd.lcd_display_string(lcd_text,1)
        sleep(0.4)
        mylcd.lcd_display_string(str_pad,1)
```

```
import os
import psutil

# Getting loadover15 minutes
load1, load5, load15 = psutil.getloadavg()
cpu_usage = (load15/os.cpu_count()) * 100
print("The CPU usage is : ", cpu_usage)
# Getting % usage of virtual_memory ( 3rd field)
print('RAM memory % used:', psutil.virtual_memory()[2])
# Getting usage of virtual_memory in GB ( 4th field)
print('RAM Used (GB):', psutil.virtual_memory()[3]/1000000000)
```

```
import psutil
import http.client
import json


def main():
    while True:
        print("CPU: \n")
        getCPUUsage()
        print("\n\nMem: \n")
        getMemUsage()
        print("\n\nDisk: \n")
        getDiskUsage()
        print("\n\nNetwork: \n")
        getNetworkUsage()
        time.sleep(10*60) # sleep for 10 minutes
```

```
  def getCPUUsage():
      print("CPU usage %: ", psutil.cpu_percent(), "%")
      print("CPU count: ", psutil.cpu_count(), "cores")
      cpuUsagePercent = psutil.cpu_percent(1)
      print("CPU usage in last 10 secs: ", cpuUsagePercent, "%")

def getMemUsage():
    print("Mem Total:",
int(psutil.virtual_memory().total/(1024*1024)), "MB")
    print("Mem Used:",
int(psutil.virtual_memory().used/(1024*1024)), "MB")
    print("Mem Available:",
int(psutil.virtual_memory().available/(1024*1024)), "MB")
    memUsagePercent = psutil.virtual_memory().percent
    print("Mem Usage %:", memUsagePercent, "%")
    print("Swap Usage %:", psutil.swap_memory().percent,
"%")
```

```
def getDiskUsage():
    for dp in psutil.disk_partitions():
        # print(x)
        print("\nDisk usage of partition ", dp.mountpoint, ": ")
        print("Total: ", int(psutil.disk_usage(dp.mountpoint).total/(1024*1024)), "MB")
        print("Used: ", int(psutil.disk_usage(dp.mountpoint).used/(1024*1024)), "MB")
        print("Free: ", int(psutil.disk_usage(dp.mountpoint).free/(1024*1024)), "MB")
        diskUsagePercent = psutil.disk_usage(dp.mountpoint).percent
        print("Used %: ", diskUsagePercent, "%")
```

```
def getNetworkUsage():
    print("Total bytes sent: ", psutil.net_io_counters().bytes_sent, "Bytes")
    print("Total bytes received: ", psutil.net_io_counters().bytes_recv, "Bytes")
    print("Total packets sent:", psutil.net_io_counters().packets_sent, "Packets")
    print("Total packets received:", psutil.net_io_counters().packets_recv, "Packets")
    print("Total incoming packets dropped:", psutil.net_io_counters().dropin, "Packets")
    print("Total outgoing packets dropped:", psutil.net_io_counters().dropout, "Packets")


if __name__ == "__main__":
    main()
```

RV College of Engineering®

Serial Peripheral Interface with Raspberry Pi

Go, change the world

Adafruit Occidentalis 0.2 or later is preconfigured with serial peripheral interface (SPI) support.

For Raspbian, few configuration changes are required.

What is SPI?

SPI is a synchronous serial communication protocol that allows data exchange between a master device (such as the Raspberry Pi) and one or more slave devices.

It uses four lines:

MOSI (Master Out Slave In): The master sends data to the slave.

MISO (Master In Slave Out): The slave sends data to the master.

SCLK (Serial Clock): Provides the clock signal for synchronization.

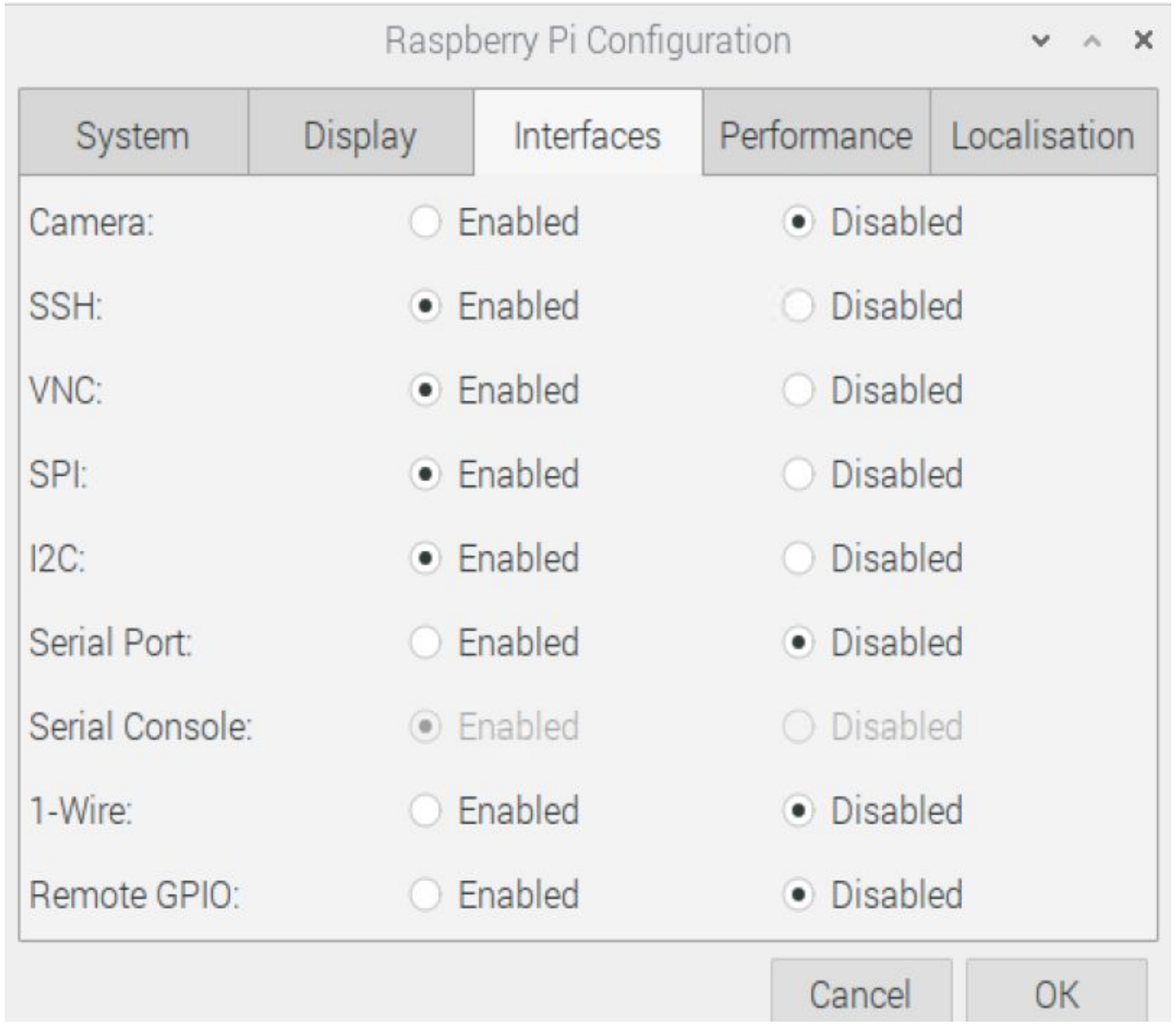
CE (Chip Enable): Selects the specific slave device for communication.

SPI on Raspberry Pi:

The Raspberry Pi has hardware support for SPI.

By default, SPI is turned off, so you need to enable it before using it.

Enabling SPI:



Camera:	Enabled	[✓ Disabled]
SSH:	[✓ Enabled]	Disabled
VNC:	[✓ Enabled]	Disabled
SPI:	[✓ Enabled]	Disabled
I2C:	[✓ Enabled]	Disabled
Serial Port:	Enabled	[✓ Disabled]
Serial Console:	[✓ Enabled]	Disabled
1-Wire:	Enabled	[✓ Disabled]
Remote GPIO:	Enabled	[✓ Disabled]