PRACTICAL GUIDE TO Subject: PhysicalComputing and IoT Programming

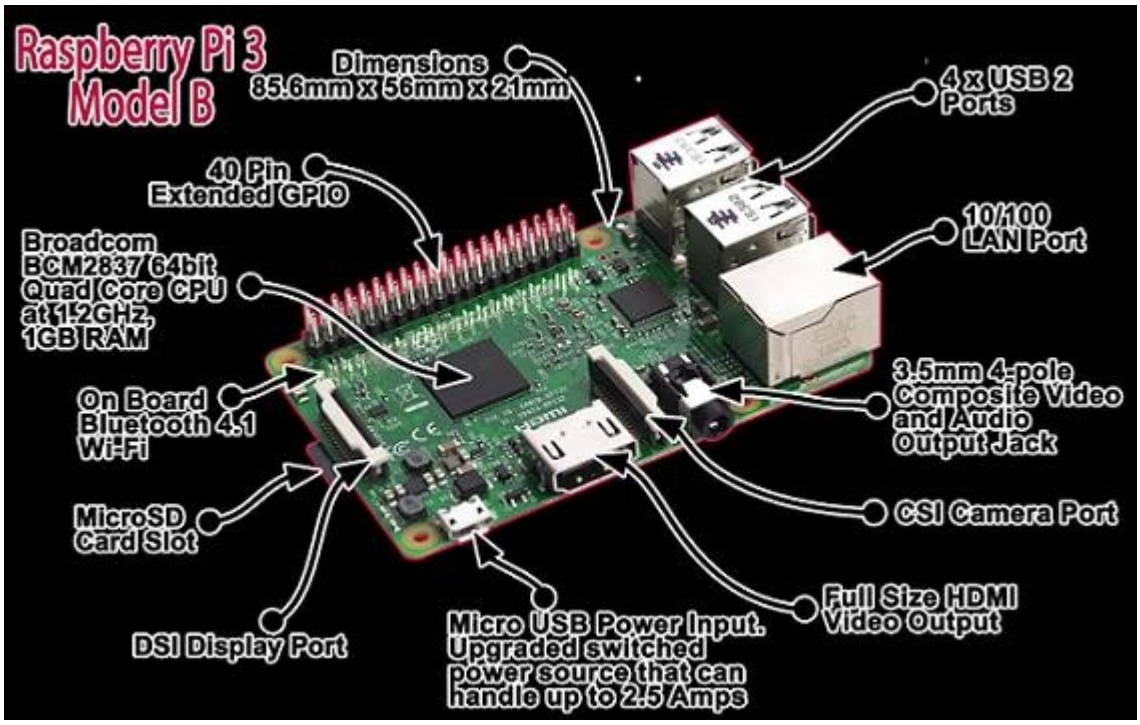
S.Y.Bsc. Computer Science

2017 -18

…Microbyte Solutions

RPI = Raspberry Pi

PRACTICAL DOCUMENTATION:



This is the Broadcom chip used in the Raspberry Pi 3, and in later models of the

Raspberry Pi 2.

The underlying architecture of the BCM2837 is identical to the BCM2836.

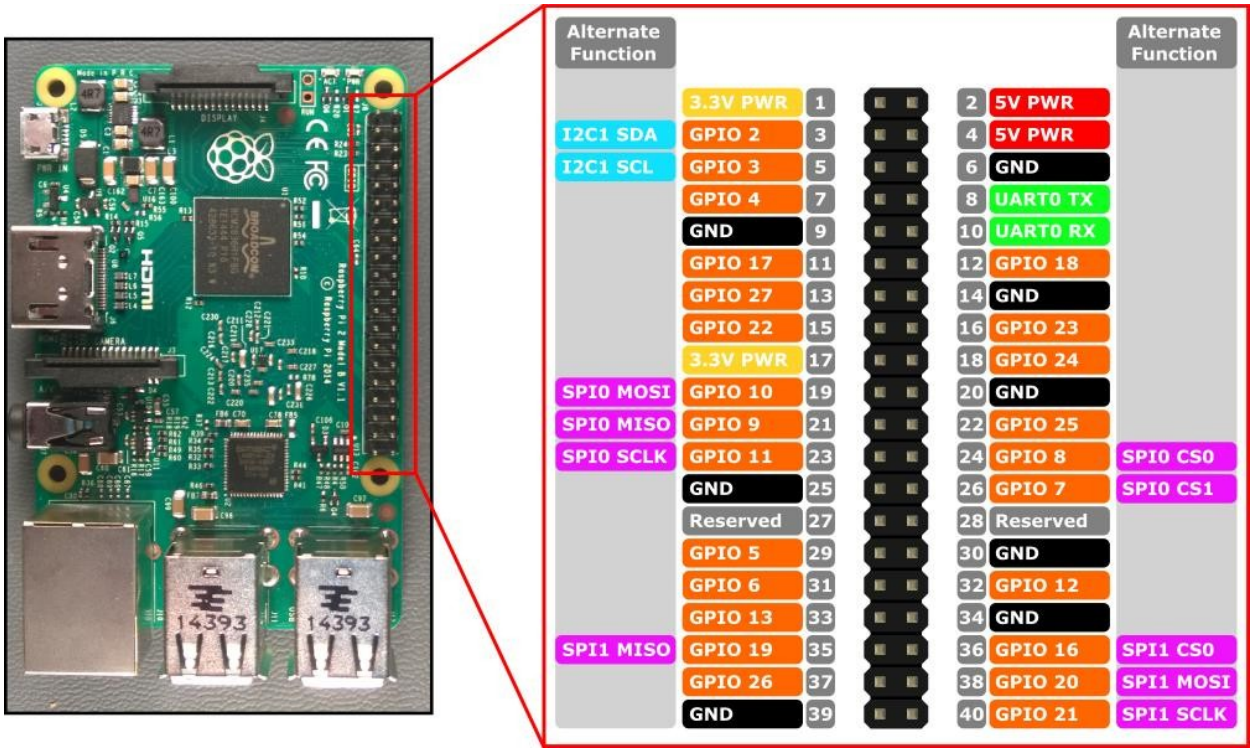
The only significant difference is the replacement of the ARMv7 quad core cluster with a

quad-core ARM Cortex A53 (ARMv8) cluster.

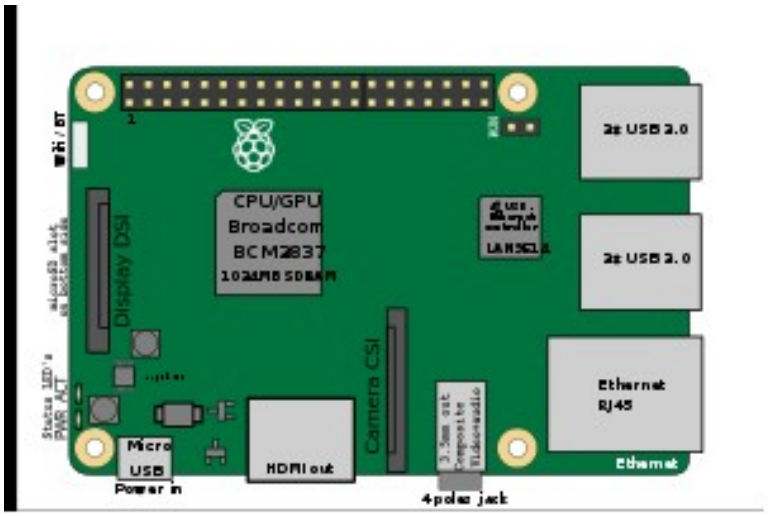
The ARM cores run at 1.2GHz, making the device about 50% faster than the Raspberry

Pi 2. The VideocoreIV runs at 400Mhz.

The Raspberry Pi 2's chip BCM2836 and the Raspberry Pi 1's chip BCM2835



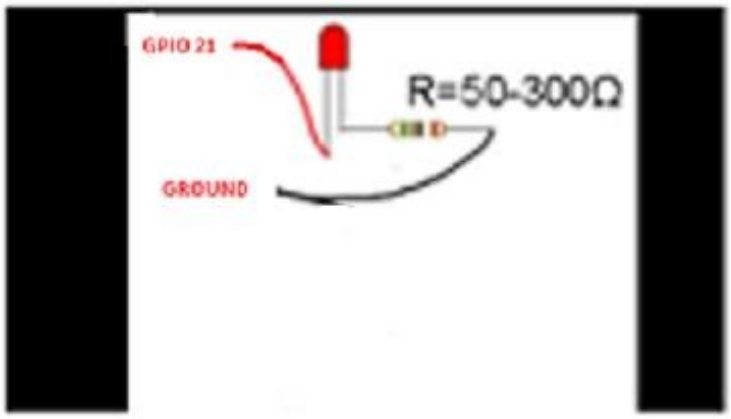
PIN DETAILS OF RPI



Practicals No.#1:

Write a program to blink LED connected to GPIO 21

General Diagram:



SOLUTION:

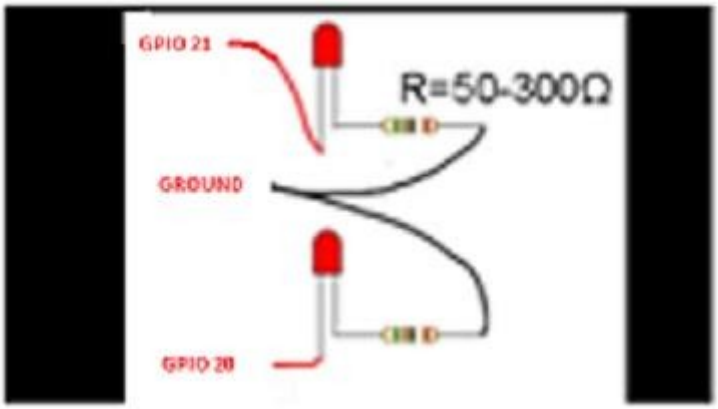
```
import RPi.GPIO as GPIO
import time
```

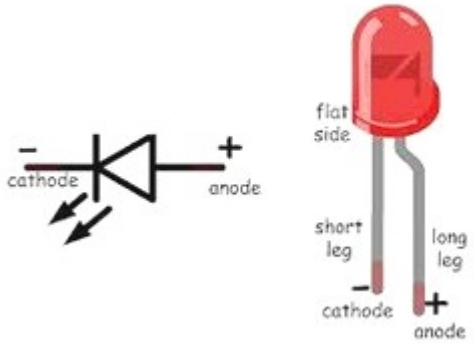
SOLUTION:

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
GPIO.setup(20,GPIO.OUT)
while True:
    GPIO.output(21,1)
    time.sleep(0.2)
    GPIO.output(21,0)
    time.sleep(0.2)
```

Practicals No.#2:

Write a program to blink LED connected to GPIO 20 and GPIO 21.





Use GPIO 20 & GPIO 21

NOTE : IN THE PRACTICAL BOARD LEDS ARE CONNECTED TO GPIO 20 & GPIO 21.

SOLUTION:

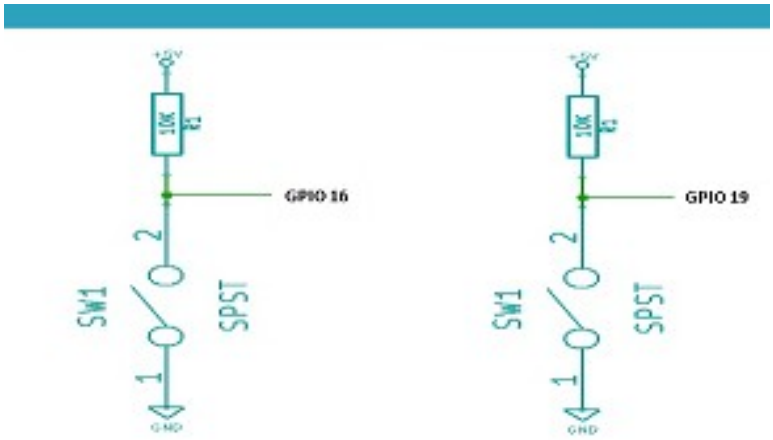
```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
GPIO.setup(20,GPIO.OUT)
```

```
GPIO.setup(21,GPIO.OUT)
while True:
    GPIO.output(20,1)
    time.sleep(0.2)
    GPIO.output(20,0)
    time.sleep(0.2)
    GPIO.output(21,1)
    time.sleep(0.2)
    GPIO.output(21,0)
    time.sleep(0.2)
```

NOTE : IN THE PRACTICAL BOARD LEDS ARE CONNECTED TO GPIO 20 & GPIO 21.

Practicals No.#3:

Write a program to detect switch closure , connected to RPI GPIO 16 & GPIO 19



GPIO 16 GPIO 19

```
import RPi.GPIO as GPIO

import time

GPIO.setmode(GPIO.BCM)

GPIO.setup(19,GPIO.IN)

GPIO.setup(16,GPIO.IN)
```

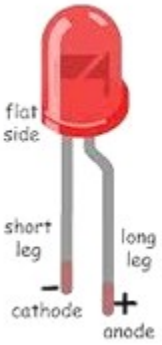
```
while True:
    reading = GPIO.input(19)
    if reading == 0:
        print "FIRST BUTTON Pressed"

    reading1= GPIO.input(16)
    if reading1 == 0:
        print "SECOND BUTTON Pressed"

    time.sleep(0.2)
```

Above Technique is Known as Scanning Keys

Practicals No.#4:

Write a program to detect switches closure ,connected to RPI pins GPIO 16 & GPIO 19

& also BLINK LEDS CONNECTED TO GPIO 20 & GPIO 21.



```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
GPIO.setup(19,GPIO.IN)
GPIO.setup(16,GPIO.IN)
GPIO.setup(20,GPIO.OUT)
GPIO.setup(21,GPIO.OUT)
while True:
    reading = GPIO.input(19)
    if reading == 0:
        print "FIRST BUTTON Pressed"

    reading1= GPIO.input(16)
    if reading1 == 0:
        print "SECOND BUTTON Pressed"

    GPIO.output(20,1)
    time.sleep(0.2)
    GPIO.output(20,0)
    time.sleep(0.2)
    GPIO.output(21,1)
    time.sleep(0.2)
    GPIO.output(21,0)
```

time.sleep(0.2)

Practicals No.#5:

Write a program to generate Sound of different pitch and duration using buzzer

#NOTE: Use GPIO 13 ---- FOR BUZZER

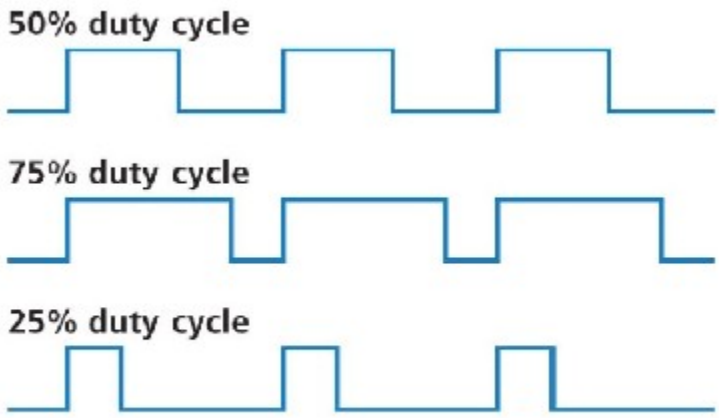
```
import RPi.GPIO as GPIO
import time
buzzer_pin=13
GPIO.setmode(GPIO.BCM)
GPIO.setup(13,GPIO.OUT)
def buzz(pitch,duration):
        period=1.0/pitch
        delay=period/2
        cycles=int(duration*pitch)
        for i in range(cycles):
```

```
            GPIO.output(buzzer_pin,True)
            time.sleep(delay)
            GPIO.output(buzzer_pin,False)
            time.sleep(delay)
while True:
    pitch_s=raw_input("Enter Pitch between (200 to 2000): ")
    pitch=float(pitch_s)
    duration_s=raw_input("Enter Duration in Seconds: ")
    duration=float(duration_s)
    buzz(pitch,duration)
```

Practicals No.#6:

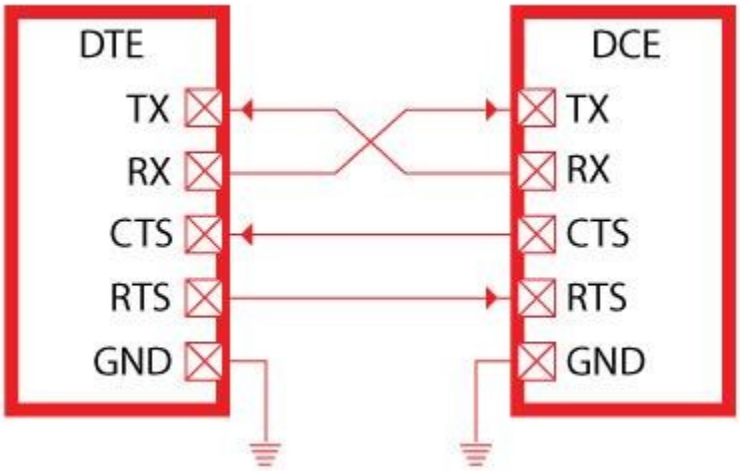
Write a program to change intensity of LED using Pulse Width Modulation (PWM)

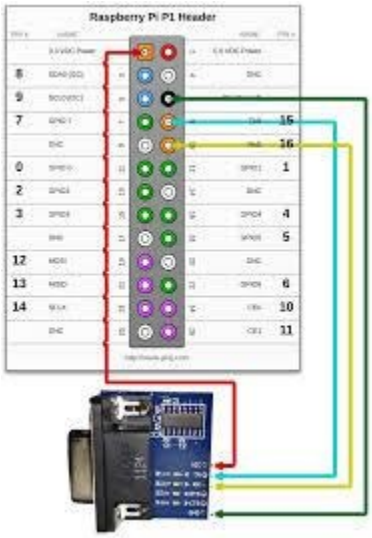
#NOTE: Use GPIO 20 ---- FOR PWM – LED




```
import RPi.GPIO as GPIO
import time
led_pin=20
GPIO.setmode(GPIO.BCM)
 GPIO.setup(20,GPIO.OUT)
pwm_led=GPIO.PWM(led_pin,500)       # freq =500  object is created
pwm_led.start(100)
while True:
    duty_s=raw_input("Enter Brightness between (0 to 100):")
    duty=int(duty_s)
    pwm_led.ChangeDutyCycle(duty)
    time.sleep(0.2)
```

Serial Communication:





Practicals No.#7:

Write a program to transmit “HI STUDENTS I AM XXXX” data to IBM PC USING RPI-3 at 9600 baud rate with 1 start, 1stop and 8 bits of data as framing data

```
import time
import serial
ser=serial.Serial('/dev/serial0',9600)
while True:
        ser.write('HI STUDENTS I AM XXXX ')
        time.sleep(10)
```

Practicals No.#8:

Write a program to transmit “HI “ data to IBM PC USING RPI-3 at 9600 baud rate with 1 start, 1stop and 8 bits of data

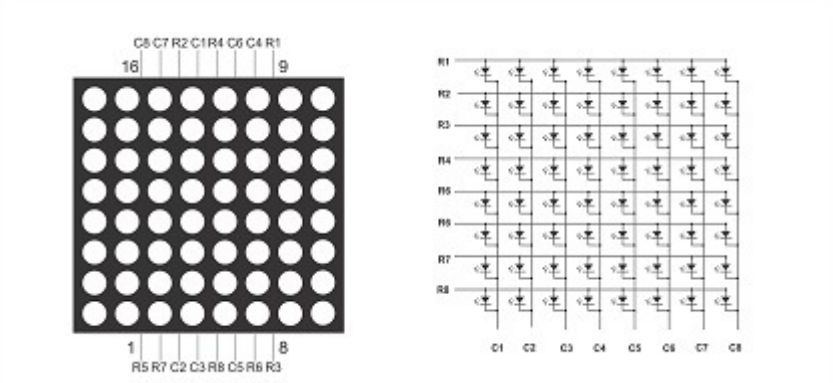
&

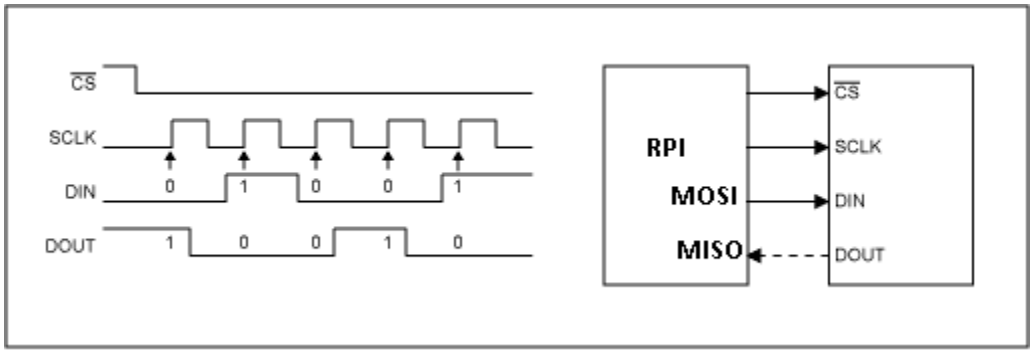
ECHO the data transmitted by IBM PC using RPI

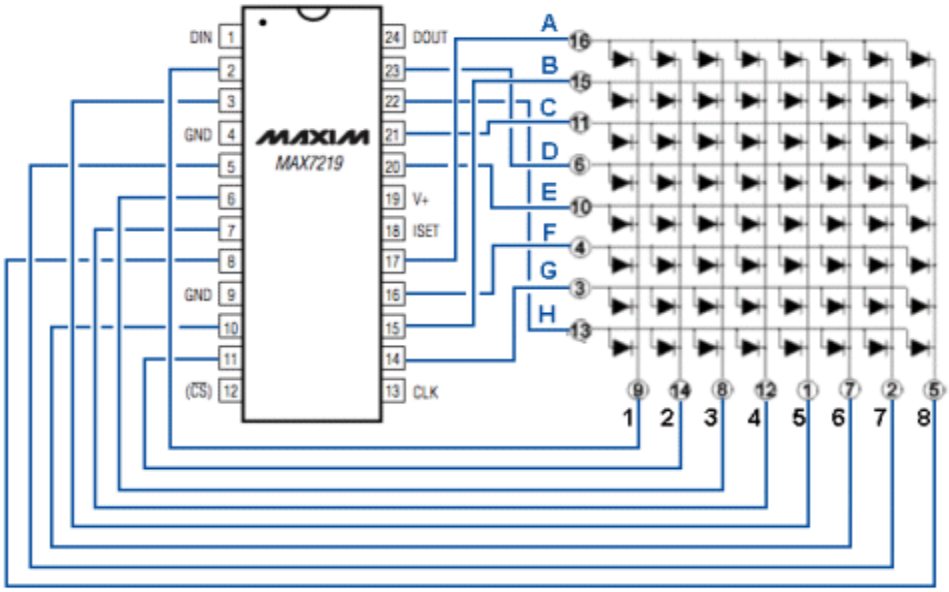
```
import time
import serial
ser=serial.Serial('/dev/serial0',9600)
```

```
while True:
        ser.write('Hi')
        p=ser.read()
        print(p)
        print('\n')
        ser.write(p)
        time.sleep(0.1)
```

DISPLAY USING 8x8 LED MATRIX DISPLAY







Connecting the 8x8 Display to the MAX7219 chip

Practicals No.#9:

Write a program to DISPLAY VARIOUS character information using MAX7219

import time

from random import randrange

Import library

```
import MAX7219array as m7219

# Import fonts

from MAX7219fonts import CP437_FONT, SINCLAIRS_FONT, LCD_FONT,
TINY_FONT

# The following imported variables make it easier to feed parameters to the library
functions

from MAX7219array import DIR_L, DIR_R, DIR_U, DIR_D

from MAX7219array import DIR_LU, DIR_RU, DIR_LD, DIR_RD

from MAX7219array import DISSOLVE, GFX_ON, GFX_OFF, GFX_INVERT


# Initialise the library and the MAX7219/8x8LED array

m7219.init()

try:

    # Display a stationary message

    m7219.static_message("Welcome!")

    time.sleep(2)

    m7219.clear_all()


    # Cycle through the range of brightness levels - up then down

    m7219.brightness(0)

    m7219.static_message("Bright ?")

    for loop in range(2):

        for brightness in range(15*(loop%2), 16-17*(loop%2), 1-2*(loop%2)):

            m7219.brightness(brightness)
```

```
        time.sleep(0.1)
    time.sleep(1)

# Clear the whole display and reset brightness
m7219.clear_all()
m7219.brightness(3)
time.sleep(1)

# Random flashing lights (Hollywood's version of a computer)
for loop in range(16):
    for matrix in range(8):
        for col in range(8):
            m7219.send_matrix_reg_byte(matrix, col+1, randrange(0x100))
            time.sleep(0.001)
m7219.clear_all()
time.sleep(1)

# Display all characters from the font individually
for char in range(0x100):
    m7219.send_matrix_letter(7-(char%8), char)
    time.sleep(0.02)
time.sleep(0.5)
m7219.clear_all()
```

```
    time.sleep(0.5)


    # Scroll characters in each of 4 directions
    for matrix in range(8):
        m7219.send_matrix_letter(matrix, 72 - matrix)
    time.sleep(0.5)
    letter_offset=0
    for dir in (DIR_L, DIR_R, DIR_U, DIR_D):
        for stage in range(8):
            for matrix in range(8):
                m7219.send_matrix_shifted_letter(matrix, 72 - matrix + letter_offset, 73 -
matrix - letter_offset, stage, dir)
            time.sleep(0.1)
        letter_offset = 1 - letter_offset
    for dir in (DIR_R, DIR_L, DIR_D, DIR_U):
        for stage in range(8):
            for matrix in range(8):
                m7219.send_matrix_shifted_letter(matrix, 72 - matrix - letter_offset, 71 -
matrix + letter_offset, stage, dir)
            time.sleep(0.1)
        letter_offset = 1 - letter_offset
    for matrix in range(8):
        m7219.send_matrix_letter(matrix, 72 - matrix)
    time.sleep(1)
```

```
    m7219.clear_all()


    # Scroll only part of a display
    Floors = ["B", "G", "1", "2"]
    m7219.static_message("Floor: " + Floors[0])
    time.sleep(1)
    for floor, display in enumerate(Floors[:-1]):
        for stage in range(8):
            m7219.send_matrix_shifted_letter(0, ord(display), ord(Floors[floor+1]), stage,
DIR_D)
            time.sleep(0.1)
    m7219.static_message("Floor: " + Floors[-1])
    time.sleep(1)
    m7219.clear_all()


    # Horizontally scroll and repeat a long message
    for dir in [DIR_L, DIR_R]:
        for speed in [3,6,9]:
            m7219.scroll_message_horiz("Speed:"+chr(48+speed)+" ", speed/3 , speed, dir)
        time.sleep(1)


    # Vertically transition (scroll) between different lines of a message
    for speed in [3,6,9]:
        m7219.static_message("Speed: "+chr(48+speed))
```

```
    time.sleep(1)
    m7219.scroll_message_vert("Speed: "+chr(48+speed), "Line 2",speed, DIR_U)
    time.sleep(0.25)
    m7219.scroll_message_vert("Line 2", "Line 3", speed, DIR_U)
    time.sleep(0.25)
    m7219.scroll_message_vert("Line 3", "Speed: "+chr(48+speed), speed, DIR_U)
    time.sleep(1)
    m7219.scroll_message_vert("Speed: "+chr(48+speed), "Line 5", speed, DIR_D)
    time.sleep(0.25)
    m7219.scroll_message_vert("Line 5", "Line 6", speed, DIR_D)
    time.sleep(0.25)
    m7219.scroll_message_vert("Line 6", "Speed: "+chr(48+speed), speed, DIR_D)
    time.sleep(1)
m7219.clear_all()
time.sleep(1)

# Wipe/fade effects
m7219.static_message("ABCDEFGH")
time.sleep(1)
for trans in (DIR_U, DIR_RU, DIR_R, DIR_RD, DIR_D, DIR_LD, DIR_L, DIR_LU):
    m7219.wipe_message("ABCDEFGH", "IJKLMNOP" ,4, trans)
    time.sleep(0.5)
    m7219.wipe_message("IJKLMNOP", "ABCDEFGH" ,4, trans)
```

```
        time.sleep(0.5)
    time.sleep(1)
    for repeat in range(2):
        m7219.wipe_message("ABCDEFGH", "Dissolve" ,4, DISSOLVE)
        time.sleep(0.5)
        m7219.wipe_message("Dissolve", "ABCDEFGH" ,4, DISSOLVE)
        time.sleep(0.5)
    time.sleep(1)
    m7219.clear_all()


    # Different fonts available in fonts.py
    m7219.scroll_message_horiz("CP437_FONT : ABCDEFGH abcdefgh 1234567890
+++ ", 2, 7.5, DIR_L, CP437_FONT)
    m7219.scroll_message_horiz("LCD_FONT : ABCDEFGH abcdefgh 1234567890 +++
", 2, 7.5, DIR_L, LCD_FONT)
    m7219.scroll_message_horiz("SINCLAIRS_FONT : ABCDEFGH abcdefgh
1234567890 +++ ", 2, 7.5, DIR_L, SINCLAIRS_FONT)
    m7219.scroll_message_horiz("TINY_FONT : ABCDEFGH abcdefgh 1234567890
+++ ", 2, 7.5, DIR_L, TINY_FONT)


    # Displaying 'graphics' (a simulated ECG) by a low-level method
    heartbeat = [0x10, 0x10, 0x0F, 0xFC, 0x30, 0x08, 0x10, 0x10]
    for loop in range(2):
        for matrix in range(7, -1, -1):
            for col in range(8):
```

```
            m7219.send_matrix_reg_byte((matrix-1)%8, col+1, 0x00)
            m7219.send_matrix_reg_byte(matrix, col+1, heartbeat[col])
            time.sleep(0.15)

# Clear each matrix in turn
for matrix in range(7, -1, -1):
    m7219.clear([matrix])
    time.sleep(0.2)
time.sleep(1)

# Print text characters using gfx_ method
text="MAX 7219"
for letter in range(len(text)):
    m7219.gfx_letter(ord(text[letter]), 8*letter)
m7219.gfx_render()
time.sleep(1)

# Using gfx_ methods allows easy subsequent manipulation eg inverting text
for matrix in range(3,8):
    for col in range(8):
        m7219.gfx_set_col(8*matrix+col, GFX_INVERT)
    m7219.gfx_render()
time.sleep(1)
```

```
    # Draw some line patterns and demonstrate graphics scrolling
    for fill in (GFX_OFF, GFX_ON):
        m7219.gfx_set_all(GFX_OFF)
        m7219.gfx_line(0, 3, 63, 3, GFX_ON)
        m7219.gfx_line(0, 4, 63, 4, GFX_ON)
        for matrix in range(8):
            m7219.gfx_line(8*matrix+3 ,0 ,8*matrix+3 ,7 , GFX_ON)
            m7219.gfx_line(8*matrix+4 ,0 ,8*matrix+4 ,7 , GFX_ON)
        m7219.gfx_render()
        time.sleep(1)
        for index, scroll in enumerate([DIR_LD, DIR_L, DIR_LU, DIR_U, DIR_RU, DIR_R,
DIR_RD, DIR_D]):
            for repeat in range(8):
                m7219.gfx_scroll(scroll, 8*index, 8, 0, 8, fill)
                m7219.gfx_render()
                time.sleep(0.05)
    m7219.gfx_set_all(GFX_OFF)
    m7219.gfx_render()


    # Draw random lines in both 'on' & 'off' modes
    x_new = 32
    y_new = 0
    for ink in [GFX_ON, GFX_OFF]:
```

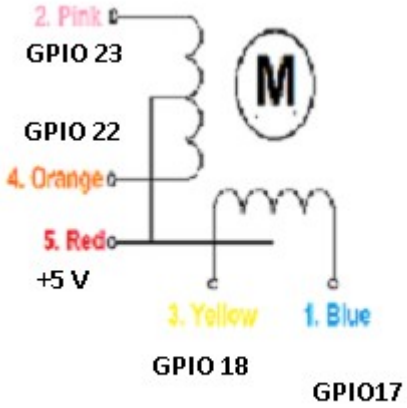
```
        for line in range(128):
            x_old, y_old = x_new, y_new
            x_new, y_new = randrange(64), 7 - y_old
            m7219.gfx_line(x_old, y_old, x_new, y_new, ink)
            m7219.gfx_render()
            time.sleep(0.1)
    time.sleep(1)
    m7219.gfx_set_all(GFX_OFF)
    m7219.gfx_render()


    # Printing text in 'invert' mode allows easy subsequent erasure (eg to scroll it over a
background)
    for x_s in range(16, 41):
        m7219.gfx_line(x_s, 7, x_s + 7, 0)
    text = " Raspberry Pi "
    rasp = [0x03, 0x05, 0x39, 0x46, 0xAA, 0x94, 0xAA, 0x46, 0x39, 0x05, 0x03]
    length = len(text)
    ext = len(rasp)
    for position in range(64, -8*length-2*ext-1, -1):
        m7219.gfx_sprite(rasp, position, GFX_INVERT)
        for letter in range(length):
            m7219.gfx_letter(ord(text[letter]), ext+position+8*letter, GFX_INVERT)
        m7219.gfx_sprite(rasp, ext+position+8*length, GFX_INVERT)
        m7219.gfx_render()
```

```
        time.sleep(0.1)
        m7219.gfx_sprite(rasp, position, GFX_INVERT)
        for letter in range(length):
            m7219.gfx_letter(ord(text[letter]), ext+position+8*letter, GFX_INVERT)
        m7219.gfx_sprite(rasp, ext+position+8*length, GFX_INVERT)


    # Similarly graphics drawn eg with gfx_sprite can be erased and scrolled by using
'invert' mode
    m7219.gfx_set_all(GFX_OFF)
    m7219.gfx_render()
    sinewave = [0x0C, 0x02, 0x01, 0x01, 0x02, 0x0C, 0x30, 0x40, 0x80, 0x80, 0x40,
0x30]
    sine_len=len(sinewave)
    text = "Max 7219"
    for letter in range(len(text)):
        m7219.gfx_letter(ord(text[letter]), 8*letter, GFX_ON)
    m7219.gfx_render()
    for loop in range(16):
        for position in range(sine_len):
            for repeat in range(64//sine_len+2):
                m7219.gfx_sprite(sinewave, repeat*sine_len - position, GFX_INVERT)
            m7219.gfx_render()
            time.sleep(0.02)
            for repeat in range(64//sine_len+2):
```

```
            m7219.gfx_sprite(sinewave, repeat*sine_len - position, GFX_INVERT)
m7219.gfx_render()


# Define & draw a large sprite, and then move it around on the array
Pi = [0x7E, 0x12, 0x12, 0x6C, 0x00, 0x54, 0x54, 0x78, # Ra
      0x00, 0x48, 0x54, 0x24, 0x00, 0xFC, 0x24, 0x18, # sp
      0x00, 0x7E, 0x48, 0x30, 0x00, 0x38, 0x54, 0x58, # be
      0x00, 0x78, 0x04, 0x04, 0x00, 0x78, 0x04, 0x04, # rr
      0x9C, 0xA0, 0x7C, 0x00, 0x00, 0x00, 0x00, 0x00, # y_
      0x00, 0x7E, 0x12, 0x12, 0x0C, 0x00, 0x74, 0x00, # Pi
      0x00, 0x00, 0x00, 0x03, 0x05, 0x39, 0x46, 0xAA, # } logo
      0x94, 0xAA, 0x46, 0x39, 0x05, 0x03]             # }
m7219.gfx_set_all(GFX_OFF)
m7219.gfx_sprite(Pi,1)
m7219.gfx_render()
time.sleep(1)
for repeat in range(2):
    for scroll in (DIR_L, DIR_LU, DIR_U, DIR_RU, DIR_R, DIR_RD, DIR_D, DIR_LD):
        moves = 2*repeat+1
        if scroll in [DIR_R, DIR_RD, DIR_D, DIR_LD]:
            moves += 1
        for loop in range(moves):
            m7219.gfx_scroll(scroll)
```

```
            m7219.gfx_render()

            time.sleep(0.1)


    # Continuous marquee display

    diamonds = chr(4) * 5

    m7219.scroll_message_horiz(" This is the end of the demo " + diamonds + " Press
<Ctrl><C> to end " + diamonds, 0, 5)



except KeyboardInterrupt:

    # reset array

    m7219.scroll_message_horiz("Goodbye!", 1, 8)

    m7219.clear_all()
```

Stepper Motor



A **stepper motor** (or step **motor**) is a brushless DC electric **motor** that divides a full rotation into a number of equal steps. The **motor's** position can then be commanded to move and hold at one of these steps without any feedback sensor (an open-loop controller), as long as the **motor** is carefully sized to the application.

Stepper motor - Wikipedia, the free encyclopedia
https://en.wikipedia.org/wiki/**Stepper_motor** Wikipedia

Practicals No.#10:

Write a program to rotate stepper motor in clockwise direction

Windings are connected to # GPIO17,GPIO18 ,GPIO22 ,GPIO23

```
import time
import RPi.GPIO as GPIO
GPIO.cleanup()
 #cleaning up in case GPIOS have been preactivated
# Use BCM GPIO references
# instead of physical pin numbers
GPIO.setmode(GPIO.BCM)
# be sure you are setting pins accordingly
# GPIO17,GPIO18,GPIO22,GPIO23
StepPins = [17,18,22,23]
# Set all pins as output
for pin in StepPins:
  GPIO.setup(pin,GPIO.OUT)
  GPIO.output(pin, False)

#wait some time to start
time.sleep(0.5)

# Define some settings
StepCounter = 0
WaitTime = 0.0015

# Define simple sequence
```

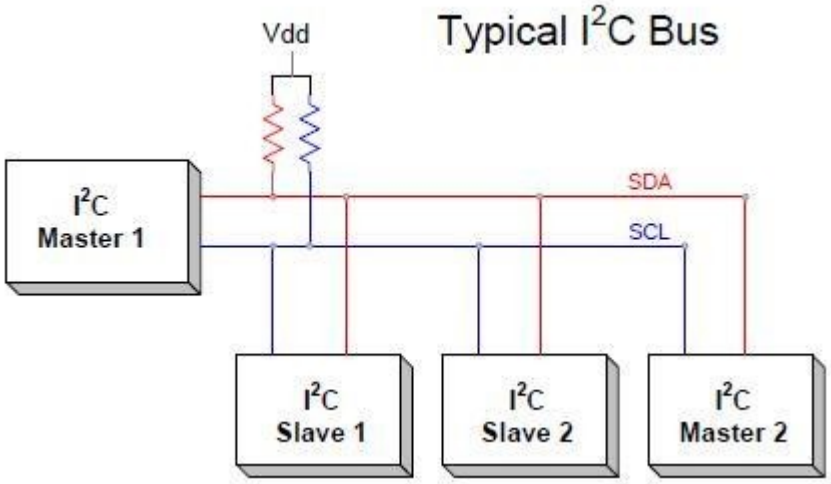
```
StepCount1 = 4

Seq1 = []

Seq1 = range(0, StepCount1)

Seq1[0] = [1,0,0,0]

Seq1[1] = [0,1,0,0]

Seq1[2] = [0,0,1,0]

Seq1[3] = [0,0,0,1]


# Define advanced sequence

# as shown in manufacturers datasheet

StepCount2 = 8

Seq2 = []

Seq2 = range(0, StepCount2)

Seq2[0] = [1,0,0,0]

Seq2[1] = [1,1,0,0]

Seq2[2] = [0,1,0,0]

Seq2[3] = [0,1,1,0]

Seq2[4] = [0,0,1,0]

Seq2[5] = [0,0,1,1]

Seq2[6] = [0,0,0,1]

Seq2[7] = [1,0,0,1]


#Full torque
```

```
StepCount3 = 4
Seq3 = []
Seq3 = [3,2,1,0]
Seq3[0] = [0,0,1,1]
Seq3[1] = [1,0,0,1]
Seq3[2] = [1,1,0,0]
Seq3[3] = [0,1,1,0]

# set
Seq = Seq2
StepCount = StepCount2

# Start main loop
try:
  while 1==1:
    for pin in range(0, 4):
      xpin = StepPins[pin]
      if Seq[StepCounter][pin]!=0:
        #print " Step %i Enable %i" %(StepCounter,xpin)
        GPIO.output(xpin, True)
      else:
        GPIO.output(xpin, False)
    StepCounter += 1
```

```
  # If we reach the end of the sequence
  # start again
    if (StepCounter==StepCount):
      StepCounter = 0
    if (StepCounter<0):
      StepCounter = StepCount


  # Wait before moving on
    time.sleep(WaitTime)
except:
  GPIO.cleanup();
finally: #cleaning up and setting pins to low again (motors can get hot if you wont)
  GPIO.cleanup();
  for pin in StepPins:
    GPIO.setup(pin,GPIO.OUT)
    GPIO.output(pin, False)
```

Practicals No.#11:

Explain i2c protocol and detect address map of i2c devices



Command to detect i2c devices in RPI

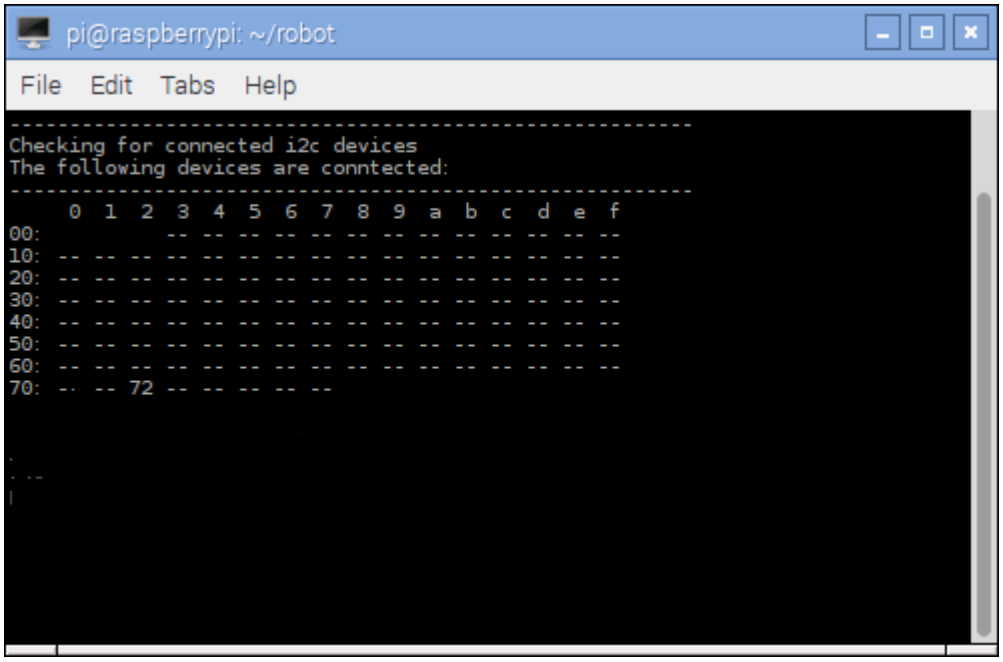
$ sudo apt-get install python-smbus

Reboot your Pi

$ sudo apt-get install i2c-tools

Attach your i2c device and run

$ sudo i2detect –y 1



… ALL THE BEST

HAPPY CODING