**LABORATORY MANUAL**

**INTERNET OF THINGS LAB**



**DEPARTMENT OF**

**COMPUTER SCIENCE & ENGINEERING**

**VASAVI COLLEGE OF ENGINEERING (AUTONOMOUS)**

**2019-2020**

**VASAVI COLLEGE OF ENGINEERING (AUTONOMOUS)**

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**B.E. 4/4, VII-Semester**

**INTERNET OF THINGS LAB**

**Syllabus**

|  |  |
| --- | --- |
| **COURSE OBJECTIVE** | **COURSE OUTCOMES** |
| At the end of the Course students should be able to: | At the end of the Course students will be able to: |
| * Learn the basics of Internet of Things(IoT) * Study the overall working of ETS IoT Trainer Kit * Understand the building blocks of Embedded Systems * Learn to interface the various sensors * Learn the important network protocols * Know the characteristics of network topologies * Write programs to interface various devices of IoT systems |  Write programs in python for a specific application   Interface the various devices related to wireless communication   Analyze the performance of Internet of Things(IoT)   Write programs for wireless application & real time monitoring   Formulate mini project using sensors, network protocols, ETS IoT Trainer Kit & Data analyzing |

1. Program to blink LED using Arduino Uno Board.
2. Programming Raspberry PI to read data from Temperature, Pressure & Humidity sensor (BME280).
3. Program to operate buzzer using push buttons.
4. Interfacing ultrasonic, IR sensors to Raspberry PI
5. Interfacing Soil Moisture sensor for Agriculture based Application
6. Developing Control applications to interface actuators.
7. Demonstrate communication protocol Bluetooth
8. Application of Zigbee in IoT systems.
9. Demonstrate communication protocol LoRa.
10. Publishing data on to Cloud using MQTT Protocol.
11. Read the data from the cloud and display them using MQTT Protocol
12. Demonstration of following RTOS concepts Timing, Multitasking, Round Robin & Semaphores

**VASAVI COLLEGE OF ENGINEERING (AUTONOMOUS)**

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**B.E. 4/4, VII-Semester**

**INTERNET OF THINGS LAB**

**CONTENTS**

|  |  |  |
| --- | --- | --- |
| **S.No** | **Name of the Experiment** | **P.No** |
| **Experiment-I: Program to blink LED using Arduino Uno Board** | | |
| 1.(a) | Program to blink LED using Arduino Uno board | 1 |
| 1.(b) | Program to interface Ultrasonic sensor to Arduino Uno board | 3 |
| **Experiment-II: Programming Raspberry PI to read data from Temperature, Pressure & Humidity sensor (BME280).** | | |
| 2.a) | Program to read data from BME-280 Sensor | 5 |
| 2.(b) | Uploading BME-280 Sensor data to Cloud | 7 |
| **Experiment-III: Program to operate buzzer** | | |
| 3.(a) | Program to operate buzzer | 9 |
| 3.(b) | Program to glow LED’s in deserved pattern using push buttons | 11 |
| **Experiment-IV:  Interfacing ultrasonic, PIR sensors to Raspberry PI** | | |
| 4.(a) | Program to Interface ultrasonic sensor to Raspberry PI | 14 |
| 4.(b) | Program to Interface PIR sensor to Raspberry PI | 17 |
| **Experiment-V: Interfacing Soil Moisture sensor for Agriculture based Application** | | |
| 5(a) | Program to Interface Soil Moisture sensor to Raspberry PI | 19 |
| **Experiment-VI: Developing Control applications to interface actuators** | | |
| 6.(a) | Program to Interface Servo motor to Raspberry PI | 21 |
| 6.(b) | Program to Interface Stepper motor to Raspberry PI | 23 |
| **Experiment-VII: Demonstrate communication protocol Bluetooth** | | |
| 7.(a) | Program to demonstrate Mobile to PI communication using Bluetooth protocol | 25 |
| 7.(b) | Program to demonstrate PI to PI communication using Bluetooth protocol | 27 |
| **Experiment-VIII: Application of  ZIGBEE in IoT systems** | | |
| 8.(a) | Program to communicate Zigbee Coordinator & Router using XCTU | 29 |
| **Experiment-IX: Demonstrate communication protocol LoRa** | | |
| 9.(a) | Program to blink LED using LoRa protocol | 32 |
| **Experiment-X: Publishing data on to Cloud using MQTT Protocol** | | |
| 10.(a) | Write a python program to publish message using MQTT  protocol | 34 |
| 10.(b) | **Write a python program to publish sensor data (PIR – Passive Infrared)   using MQTT protocol** | 35 |
| **Experiment-XI: Read the data from the cloud and display them using MQTT Protocol** | | |
| 11.(a) | Write a python program to subscribe  message using MQTT protocol | 36 |
| 11.(b) | Write a python program to subscribe sensor data (PIR – Passive Infrared)   using MQTT protocol | 37 |
| **Experiment-XII:** **Demonstration of following RTOS concepts Timing, Multitasking, Round Robin & Semaphores** | | |
| 12.(a) | Program to Demonstrate RTOS concept Timing | 38 |
| 12.(b) | Program to Demonstrate RTOS concept Multitasking | 39 |
| 12.(c) | Program to Demonstrate RTOS concept Round Robin | 40 |
| 12.(d) | Program to Demonstrate RTOS concept Semaphores | 43 |

**Experiment-I: Program to blink LED using Arduino Uno Board**

**1.(a) Aim:** To blink LED using Arduino Uno Board

**Description:** The Arduino Uno is an [open-source](https://en.wikipedia.org/wiki/Open-source) [microcontroller board](https://en.wikipedia.org/wiki/Microcontroller_board) based on the [Microchip](https://en.wikipedia.org/wiki/Microchip_Technology) [ATmega328P](https://en.wikipedia.org/wiki/ATmega328P) microcontroller and developed by Arduino. The board is equipped with sets of digital and analog [input/output](https://en.wikipedia.org/wiki/Input/output) (I/O) pins that may be interfaced to various [expansion boards](https://en.wikipedia.org/wiki/Expansion_board) (shields) and other circuits. The board has 14 Digital pins, 6 Analog pins, and programmable with the [Arduino IDE](https://en.wikipedia.org/wiki/Arduino#Software) (Integrated Development Environment) via a type B [USB cable](https://en.wikipedia.org/wiki/USB_cable).[[4]](https://en.wikipedia.org/wiki/Arduino_Uno#cite_note-priceton-4) It can be powered by the USB cable or by an external [9-volt battery](https://en.wikipedia.org/wiki/9-volt_battery), though it accepts voltages between 7 and 20 volts.



**Source Code :**

/\* Blink Turns on an LED on for one second, then off for one second, repeatedly.

This example code is in the public domain. \*/

// Pin 13 has an LED connected on most Arduino boards.

int led = 13;

void setup() {

// initialize the digital pin as an output.

pinMode(led, OUTPUT);

}

 // the loop routine runs over and over again forever:

void loop() {

digitalWrite(led, HIGH); // turn the LED on (HIGH is the voltage level)

delay(1000); // wait for a second

digitalWrite(led, LOW); // turn the LED off by making the voltage LOW

delay(1000); // wait for a second

}

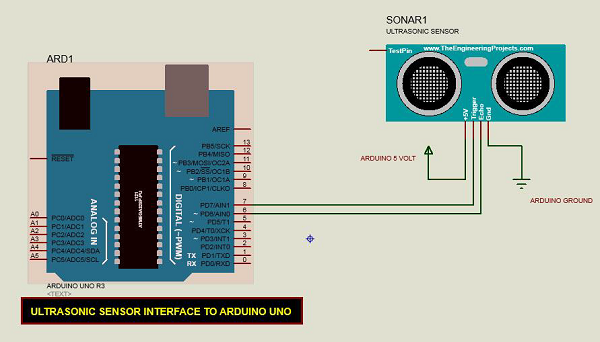
**Input:** Connecting LED to Pin 13

**Output:** LED blink for every 1sec

**1.(b) Aim:** Program to Interface Ultrasonic Sensor to Arduino Uno Board

**Description:** The HC-SR04 ultrasonic sensor uses SONAR to determine the distance of an object just like the bats do. It offers excellent non-contact range detection with high accuracy and stable readings in an easy-to-use package from 2 cm to 400 cm or 1” to 13 feet.The operation is not affected by sunlight or black material, although acoustically, soft materials like cloth can be difficult to detect. It comes complete with ultrasonic transmitter and receiver module.

`



**Source Code:**

const int pingPin = 7; // Trigger Pin of Ultrasonic Sensor

const int echoPin = 6; // Echo Pin of Ultrasonic Sensor

void setup() {

Serial.begin(9600); // Starting Serial Terminal

}

void loop() {

long duration, inches, cm;

pinMode(pingPin, OUTPUT);

digitalWrite(pingPin, LOW);

delayMicroseconds(2);

digitalWrite(pingPin, HIGH);

delayMicroseconds(10);

digitalWrite(pingPin, LOW);

pinMode(echoPin, INPUT);

duration = pulseIn(echoPin, HIGH);

inches = microsecondsToInches(duration);

cm = microsecondsToCentimeters(duration);

Serial.print(inches);

Serial.print("in, ");

Serial.print(cm);

Serial.print("cm");

Serial.println();

delay(100);

}

long microsecondsToInches(long microseconds) {

return microseconds / 74 / 2;

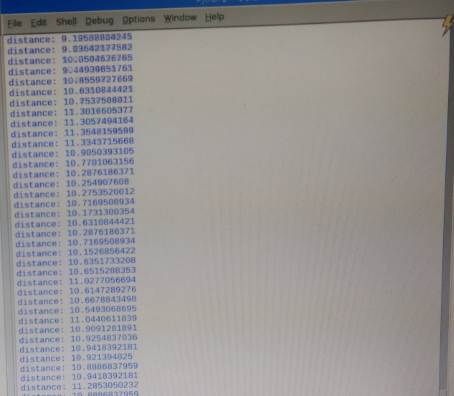
}

long microsecondsToCentimeters(long microseconds) {

return microseconds / 29 / 2;

}

**Output**:



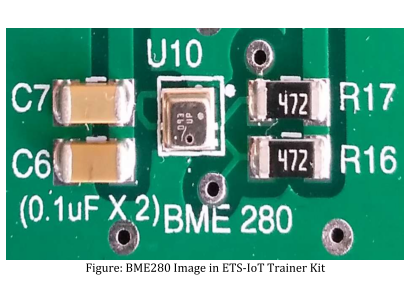
**Experiment-II: Programming Raspberry PI to read data from Temperature, Pressure & Humidity sensor (BME280).**

**2.(a) Aim:** Program to read data from BME-280 Sensor

**Description**: The BME280 is as combined digital humidity, pressure and temperature sensor based on proven sensing principles. The sensor module is housed in an extremely compact metal-lid LGA package with a footprint of only 2.5 × 2.5 mm² with a height of 0.93 mm. Its small dimensions and its low power consumption allow the implementation in battery driven devices such as handsets, GPS modules or watches. The BME280 is register and performance compatible to the Bosch Sensortec BMP280 digital pressure sensor (see chapter 5.2 for details). The BME280 achieves high performance in all applications requiring humidity and pressure measurement. These emerging applications of home automation control, in-door navigation, health care as well as GPS refinement require a high accuracy and a low TCO at the same time.

BME280 can be operated in three power modes

* sleep mode
* normal mode
* forced mode



**Source Code:**

import sys

import urllib

sys.path.append('/home/pi/ETS\_IoT KIT demo/DemoCode/BME280')

import BME280lib as BME

import time

while True:

(chip\_id, chip\_version) = BME.readBME280ID()

print "Chip ID :", chip\_id

print "Version :", chip\_version

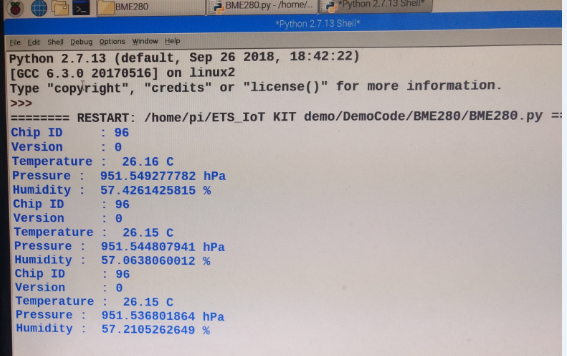
temperature,pressure,humidity = BME.readBME280All()

print "Temperature : ", temperature, "C"

print "Pressure : ", pressure, "hPa"

print "Humidity : ", humidity, "%"

**Output:**



**2.(b) Aim:** Uploading BME-280 Sensor data to Cloud using Thingspeak.com website

**Description:** The BME280 is as combined digital humidity, pressure and temperature sensor based on proven sensing principles. The sensor module is housed in an extremely compact metal-lid LGA package with a footprint of only 2.5 × 2.5 mm² with a height of 0.93 mm. Its small dimensions and its low power consumption allow the implementation in battery driven devices such as handsets, GPS modules or watches. The BME280 is register and performance compatible to the Bosch Sensortec BMP280 digital pressure sensor (see chapter 5.2 for details). The BME280 achieves high performance in all applications requiring humidity and pressure measurement. These emerging applications of home automation control, in-door navigation, health care as well as GPS refinement require a high accuracy and a low TCO at the same time.

BME280 can be operated in three power modes

* sleep mode
* normal mode
* forced mode

**Source Code:**

import sys

import urllib

sys.path.append('/home/pi/ETS\_IoT KIT demo/DemoCode/BME280')

import BME280lib as BME

import time

while True:

(chip\_id, chip\_version) = BME.readBME280ID()

print "Chip ID :", chip\_id

print "Version :", chip\_version

temperature,pressure,humidity = BME.readBME280All()

print "Temperature : ", temperature, "C"

print "Pressure : ", pressure, "hPa"

print "Humidity : ", humidity, "%"

data=urllib.urlopen("https://api.thingspeak.com/update?api\_key=96374EFUFA3QGXUL&field1="+str(temperature)+"&field2="+str(pressure)+"&field3="+str(humidity))

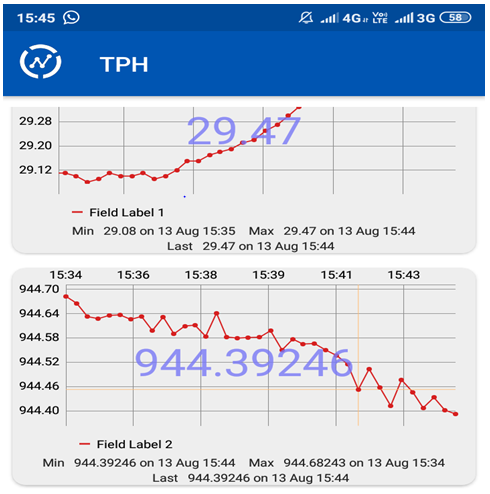
print "data"+str(data)

for i in range(5):

print "hi"

time.sleep(5)

**Output:**

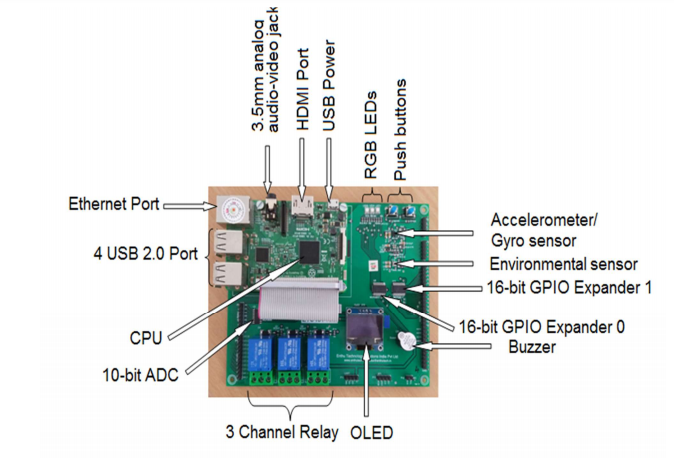
****

****

**Experiment III: Program to operate buzzer**

**3.(a) Aim:** program to operate buzzer

**Description:** ETS IoT kit is an all-in-one prototyping platform for sensor based IoT projects. It’s packed with state of the art sensor technology and ready-to-use software applications, capable of fulfilling all your IoT application needs. It contains inbuilt raspberry pi3 with all in one sensor platform to enable infinite IoT application prototype possibilities.

****

**SourceCode:**

#mcp23017 library path

import sys

sys.path.append('/home/pi/Adafruit-Raspberry-Pi-Python-Code-legacy/Adafruit\_MCP230xx')

from Adafruit\_MCP230XX import Adafruit\_MCP230XX

import time

#mcp IC configuration

mcp = Adafruit\_MCP230XX(busnum = 1, address = 0x20, num\_gpios = 16) # MCP23017

#mcp input/output configuration

mcp.config(8, mcp.OUTPUT)

mcp.config(11,mcp.OUTPUT)

try:

while True:

mcp.output(11, 1)

time.sleep(1)

mcp.output(0, 0)

mcp.output(4, 0)

mcp.output(8, 0)

#mcp.output(3, 0)

#mcp.output(4, 0)

#mcp.output(5, 0)

#mcp.output(6, 0)

#mcp.output(7, 0)

#mcp.output(8, 0)

time.sleep(1)

except KeyboardInterrupt:

mcp.output(0, 0)

mcp.output(1, 0)

mcp.output(2, 0)

mcp.output(3, 0)

mcp.output(4, 0)

mcp.output(5, 0)

mcp.output(6, 0)

mcp.output(7, 0)

mcp.output(8, 0)

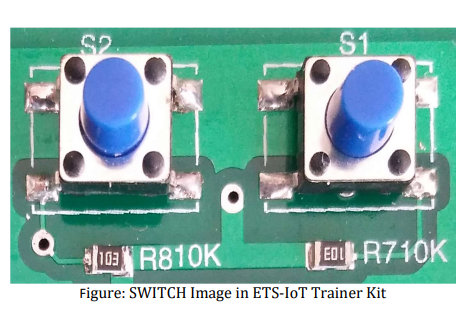
mcp.output(11, 0)

**Output:**



**3.(b) Aim:** Program to glow LED’s in deserved pattern using push buttons

**Description:** ETS IoT kit is an all-in-one prototyping platform for sensor based IoT projects. It’s packed with state of the art sensor technology and ready-to-use software applications, capable of fulfilling all your IoT application needs. It contains inbuilt raspberry pi3 with all in one sensor platform to enable infinite IoT application prototype possibilities.



**Source Code:**

#mcp23017 library path

import sys

sys.path.append('/home/pi/Adafruit-Raspberry-Pi-Python-Code-legacy/Adafruit\_MCP230xx')

from Adafruit\_MCP230XX import Adafruit\_MCP230XX

import time

#mcp IC configuration

mcp = Adafruit\_MCP230XX(busnum = 1, address = 0x20, num\_gpios = 16) # MCP23017

#mcp input/output configuration

mcp.cofig(0, mcp.OUTPUT)

mcp.cofig(1, mcp.OUTPUT)

mcp.cofig(2, mcp.OUTPUT)

mcp.cofig(3, mcp.OUTPUT)

mcp.cofig(4, mcp.OUTPUT)

mcp.cofig(5, mcp.OUTPUT)

mcp.cofig(6, mcp.OUTPUT)

mcp.cofig(7, mcp.OUTPUT)

mcp.config(8, mcp.OUTPUT)

mcp.config(11,mcp.OUTPUT)

mcp.config(9, mcp.INPUT)

mcp.pullup(9,1)

mcp.config(10, mcp.INPUT)

mcp.pullup(10,1)

try:

while True:

print “pin 9=%d” % (mcp.input(9))

print “pin 10=%d” %(mcp.input(10))

x=mcp.input(9)

y=mcp.input(10)

if x==512 and y==1024:

mcp.output(2, 1) # LED1 in Blue

time.sleep(1)

mcp.output(0, 0)

mcp.output(4, 0)

mcp.output(11, 0)

time.sleep(1)

elif x==0 and y==1024:

mcp.ouput(4, 1) # LED2 in Green

time.sleep(1)

mcp.output(0,0)

mcp.output(2,0)

mcp.output(11,0)

time.sleep(1)

elif x==512 and y==0:

mcp.output(0,1) # LED3 in Red

time.sleep(1)

mcp.output(2,0)

mcp.output(4,0)

mcp.output(11,0)

time.sleep(1)

else:

mcp.output(11, 1) # Buzzer

time.sleep(1)

mcp.output(0,0)

mcp.output(2,0)

mcp.output(4,0)

time.sleep(1)

except KeyboardInterrupt:

mcp.output(0, 0)

mcp.output(1, 0)

mcp.output(2, 0)

mcp.output(3, 0)

mcp.output(4, 0)

mcp.output(5, 0)

mcp.output(6, 0)

mcp.output(7, 0)

mcp.output(8, 0)

mcp.output(11, 0)

time.sleep(1)

**Output:**

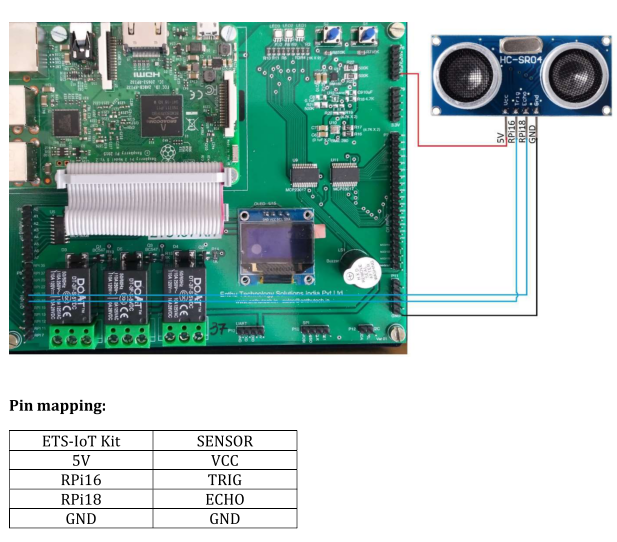
|  |  |  |
| --- | --- | --- |
| **X** | **Y** | **Output** |
| 512 | 1024 | LED 1 glows in Blue |
| 0 | 1024 | LED2 glows in Green |
| 512 | 0 | LED3 glows in Red |
| 0 | 0 | Buzzer |

**Experiment-IV: Interfacing ultrasonic, PIR sensors to Raspberry PI**

**4.(a) AIM:** Program to Interface ultrasonic sensor to Raspberry PI

**Description:** The HC-SR04 ultrasonic sensor uses SONAR to determine the distance of an object just like the bats do. It offers excellent non-contact range detection with high accuracy and stable readings in an easy-to-use package from 2 cm to 400 cm or 1” to 13 feet.

The operation is not affected by sunlight or black material, although acoustically, soft materials like cloth can be difficult to detect. It comes complete with ultrasonic transmitter and receiver module.

****

**Source Code:**

import time

import RPi.GPIO as GPIO

# Use BCM GPIO references

GPIO.setmode(GPIO.BOARD)

GPIO.setwarnings(False)

# Define GPIO to use on Pi

GPIO\_TRIGGER = 16 ##connect with RPI16

GPIO\_ECHO = 18 ##connect with RPI18

print "Ultrasonic Measurement"

# Set pins as output and input

GPIO.setup(GPIO\_TRIGGER,GPIO.OUT) # Trigger

GPIO.setup(GPIO\_ECHO,GPIO.IN) # Echo

# Set trigger to False (Low)

GPIO.output(GPIO\_TRIGGER, False)

# Allow module to settle

time.sleep(0.5)

while True:

# Send 10us pulse to trigger

GPIO.output(GPIO\_TRIGGER, True)

time.sleep(0.00001)

GPIO.output(GPIO\_TRIGGER, False)

while GPIO.input(GPIO\_ECHO)==0:

start = time.time()

while GPIO.input(GPIO\_ECHO)==1:

stop = time.time()

# Calculate pulse length

elapsed = stop-start

# Distance pulse travelled in that time is time

# multiplied by the speed of sound (cm/s)

distance = elapsed \* 34300

# That was the distance there and back so halve the value

distance = distance / 2

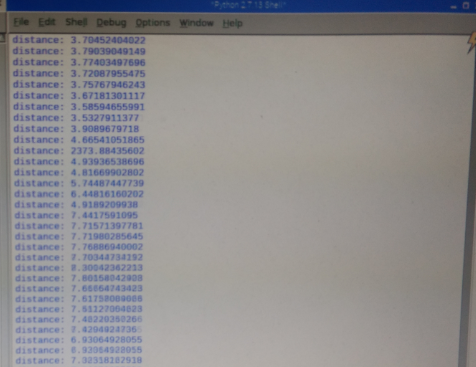
print "Distance : %.1f" % distance "cm"

time.sleep(1)

# Reset GPIO settings

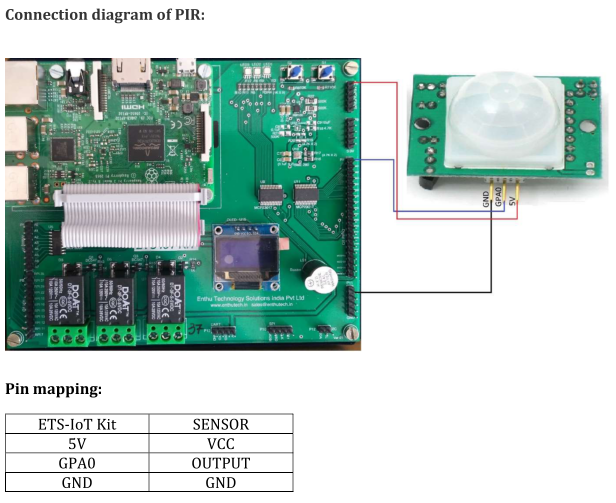
#GPIO.cleanup()

**Output:**

****

**4.(b) Aim:** Program to Interface PIR sensor to Raspberry PI

**Description:** A passive infrared sensor (PIR sensor) is an electronic [sensor](https://en.wikipedia.org/wiki/Sensor) that measures [infrared](https://en.wikipedia.org/wiki/Infrared) (IR) light radiating from objects in its field of view. They are most often used in PIR-based [motion detectors](https://en.wikipedia.org/wiki/Motion_detector). PIR sensors are commonly used in security alarms and automatic lighting applications. PIR sensors detect general movement, but do not give information on who or what moved. For that purpose, an [active IR sensor](https://en.wikipedia.org/wiki/Active_IR_sensor) is required.

****

**SourceCode:**

import RPi.GPIO as GPIO

import time

GPIO.setwarnings(False)

GPIO.setmode(GPIO.BOARD)

GPIO.setup(16, GPIO.IN) #Read output from PIR motion sensor

GPIO.setup(18, GPIO.OUT) #LED output pin

while True:

i=GPIO.input(16)

if i==0: #When output from motion sensor is LOW

print "No intruders",i

GPIO.output(18, 0) #Turn OFF LED/Buzzzer

time.sleep(0.1)

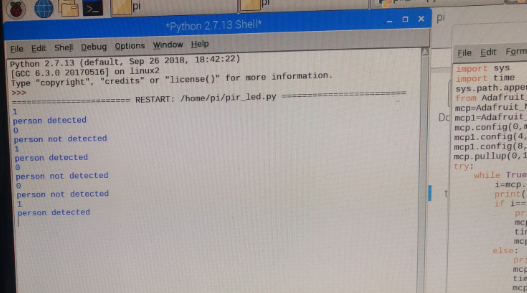
elif i==1: #When output from motion sensor is HIGH

print "Intruder detected",i

GPIO.output(18, 1) #Turn ON LED/Buzzer

time.sleep(0.1)

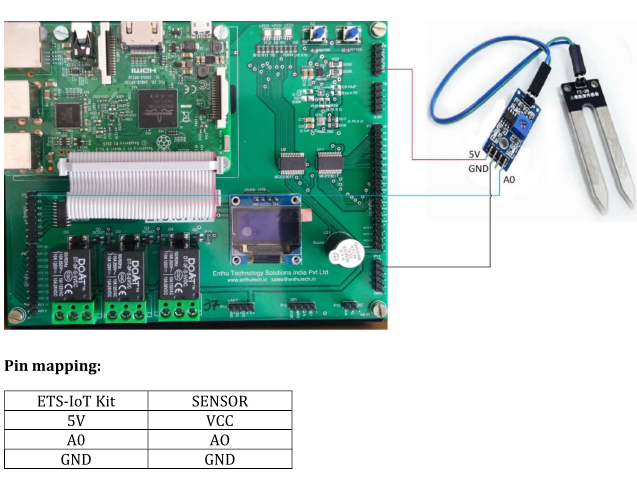
**Output:**

****

**Experiment-V: Interfacing Soil Moisture sensor for Agriculture based Application**

**5.(a) Aim:** Program to Interface Soil Moisture sensor toRaspberry PI

**Description:** Soil moisture sensors measure the volumetric [water content](https://en.wikipedia.org/wiki/Water_content) in [soil](https://en.wikipedia.org/wiki/Soil). Since the direct [gravimetric measurement](https://en.wikipedia.org/wiki/Gravimetric_analysis) of free soil moisture requires removing, drying, and weighing of a sample, soil moisture sensors measure the volumetric water content indirectly by using some other property of the soil, such as electrical resistance, dielectric constant, or interaction with [neutrons](https://en.wikipedia.org/wiki/Neutron), as a proxy for the moisture content.



**SourceCode:**

import spidev

import time

spi = spidev.SpiDev()

spi.open(0,0)

def ReadChannel(channel):

adc = spi.xfer2([1,(8+channel)<<4,0])

data= ((adc[1]&3) << 8) + adc[2]

return data

def ConvertVolts(data,places):

volts = (data \* 3.3) / float(1023)

volts = round(volts,places)

return volts

moisture\_channel = 0 ##connect with A0

delay = 3

while True:

moisture\_level = ReadChannel(moisture\_channel)

moisture\_volts = ConvertVolts(moisture\_level,2)

print "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"

print ("Moisture: {} ({}V)".format(moisture\_level,moisture\_volts))

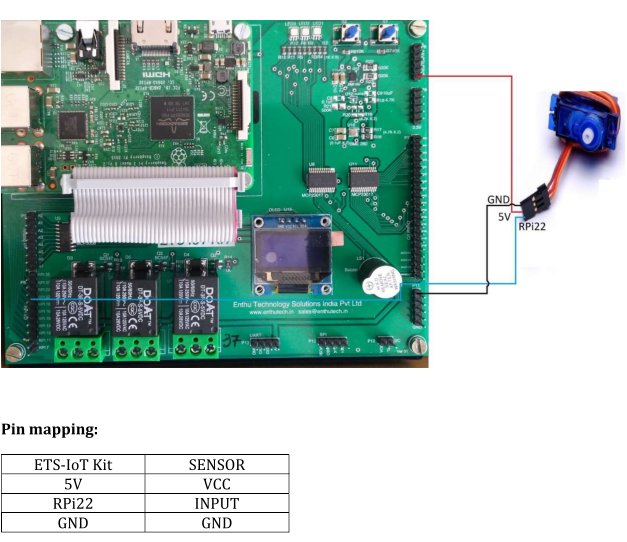
time.sleep(delay)

**Output:**

**Experiment-VI: Developing Control applications to interface actuators**

**6.(a) Aim:** Program to Interface Servo motor toRaspberry PI

**Description:** A **servo motor** is an electrical device which can push or rotate an object with great precision. If you want to rotate and object at some specific angles or distance, then you use servo motor. It is just made up of simple motor which run through **servo mechanism**. If motor is used is DC powered then it is called DC servo motor, and if it is AC powered motor then it is called AC servo motor. We can get a very high torque servo motor in a small and light weight packages. Doe to these features they are being used in many applications like toy car, RC helicopters and planes, Robotics, Machine etc.



**SourceCode:**

import RPi.GPIO as GPIO

import time

GPIO.setmode(GPIO.BOARD)

GPIO.setup(12, GPIO.OUT)

p = GPIO.PWM(12, 50)

p.start(7.5)

try:

while True:

p.ChangeDutyCycle(2.5) # turn towards 0 degree

time.sleep(1) # sleep 1 second

p.ChangeDutyCycle(7.5) # turn towards 90 degree

time.sleep(1) # sleep 1 second

p.ChangeDutyCycle(12.5) # turn towards 180 degree

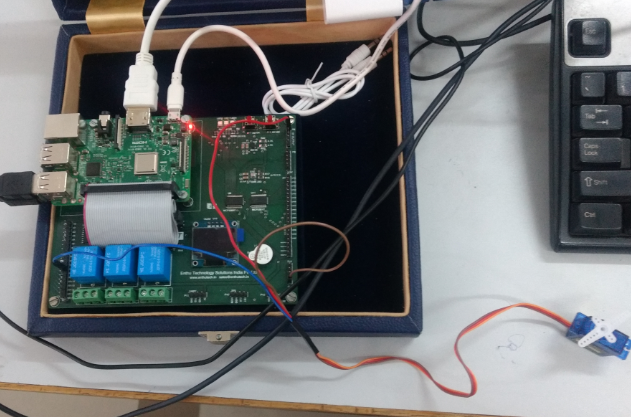
time.sleep(1) # sleep 1 second

except KeyboardInterrupt:

p.stop()

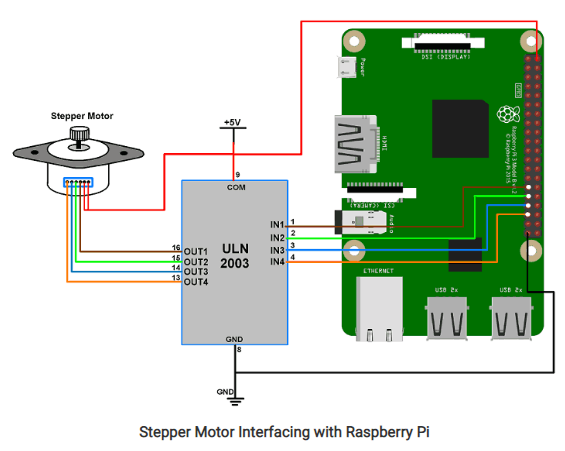
GPIO.cleanup()

**Output:**

****

**6.(b) Aim:** Program to Interface Stepper motor toRaspberry PI

**Description:** A stepper motor, also known as step motor or stepping motor, is a [brushless DC electric motor](https://en.wikipedia.org/wiki/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 [position sensor](https://en.wikipedia.org/wiki/Rotary_encoder) for [feedback](https://en.wikipedia.org/wiki/Feedback) (an [open-loop controller](https://en.wikipedia.org/wiki/Open-loop_controller)), as long as the motor is carefully sized to the application in respect to [torque](https://en.wikipedia.org/wiki/Torque) and speed.



**SourceCode:**

import RPi.GPIO as GPIO

from time import sleep

import sys

#assign GPIO pins for motor

motor\_channel = (29,31,33,35)

GPIO.setwarnings(False)

GPIO.setmode(GPIO.BOARD)

GPIO.setup(motor\_channel, GPIO.OUT)

motor\_direction = input('select motor direction a=anticlockwise, c=clockwise: ')

while True:

try:

if(motor\_direction == 'c'):

print('motor running clockwise\n')

GPIO.output(motor\_channel, (GPIO.HIGH,GPIO.LOW,GPIO.LOW,GPIO.HIGH))

sleep(0.02)

GPIO.output(motor\_channel, (GPIO.HIGH,GPIO.HIGH,GPIO.LOW,GPIO.LOW))

sleep(0.02)

GPIO.output(motor\_channel, (GPIO.LOW,GPIO.HIGH,GPIO.HIGH,GPIO.LOW))

sleep(0.02)

GPIO.output(motor\_channel, (GPIO.LOW,GPIO.LOW,GPIO.HIGH,GPIO.HIGH))

sleep(0.02)

elif(motor\_direction == 'a'):

print('motor running anti-clockwise\n')

GPIO.output(motor\_channel, (GPIO.HIGH,GPIO.LOW,GPIO.LOW,GPIO.HIGH))

sleep(0.02)

GPIO.output(motor\_channel, (GPIO.LOW,GPIO.LOW,GPIO.HIGH,GPIO.HIGH))

sleep(0.02)

GPIO.output(motor\_channel, (GPIO.LOW,GPIO.HIGH,GPIO.HIGH,GPIO.LOW))

sleep(0.02)

GPIO.output(motor\_channel, (GPIO.HIGH,GPIO.HIGH,GPIO.LOW,GPIO.LOW))

sleep(0.02)

#press ctrl+c for keyboard interrupt

except KeyboardInterrupt:

#query for setting motor direction or exit

motor\_direction = input('select motor direction a=anticlockwise, c=clockwise or q=exit: ')

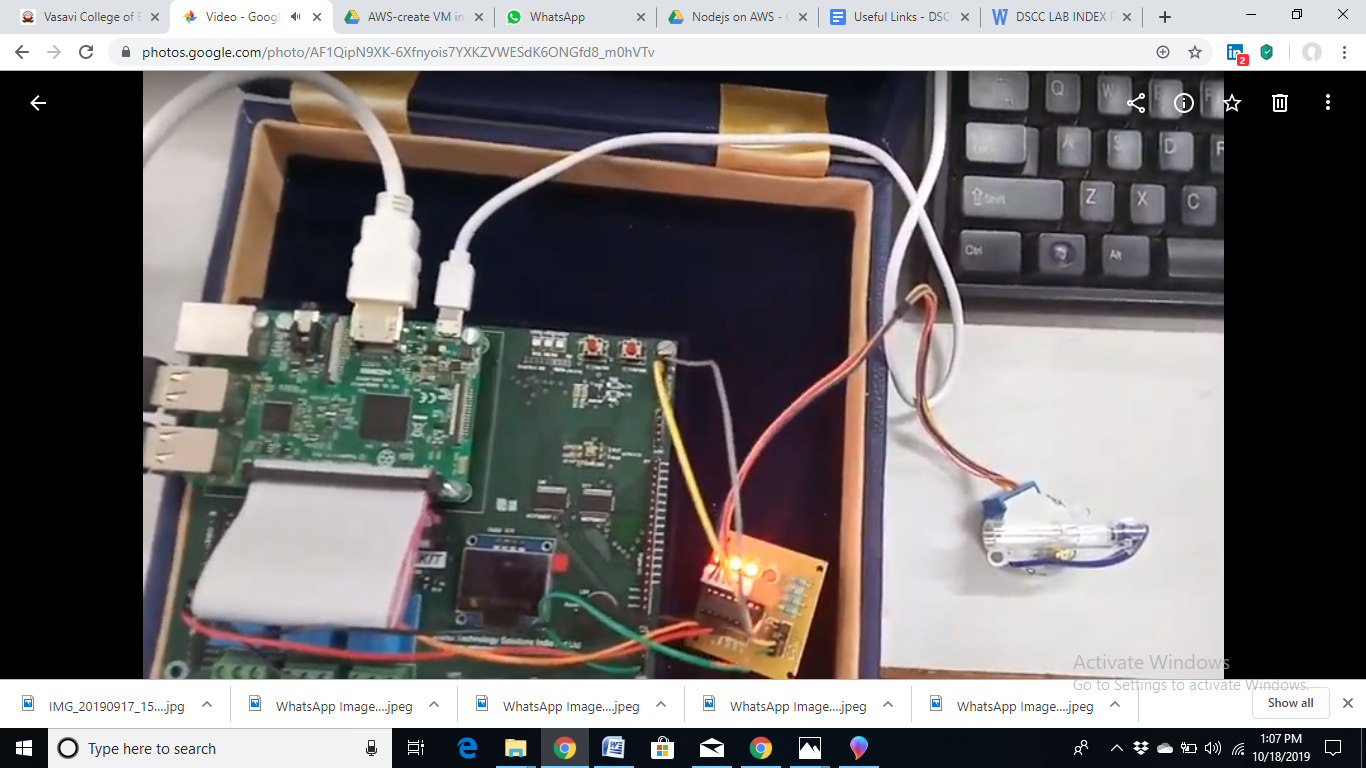
#check for exit

if(motor\_direction == 'q'):

print('motor stopped')

sys.exit(0)

**Output:**

****

**Experiment-VII: Demonstrate communication protocol Bluetooth**

**7.(a) Aim:** Program to demonstrate Mobile to PI communication using Bluetooth protocol

**Description:** Bluetooth is a wireless technology used to transfer data between different electronic devices. It is mainly designed for communicating over short distances less than about 10m or 30ft.A computer network in which one centralized, powerful computer (called the server) is a hub to which many less powerful personal computers or workstations (called clients) are connected Radio frequency communication (RFCOMM) The Bluetooth protocol RFCOMM is a simple set of transport protocols, made on top of the L2CAP protocol, providing emulated RS-232 serial ports. RFCOMM is sometimes called serial port emulation.

**Procedure:**

**Package Installation Codes**

* sudo apt-get install bluetooth
* sudo apt-get install bluez
* sudo apt-get install python-bluez

**Step 1**: First you have to install Pi3 Bluetooth terminal in your Android device through playstore

**Step 2:** Open the LX-Terminal in Raspberry PI and enter the below commands one by one

* sudo bluetoothctl
* power on
* agent on
* default-agent
* discoverable on
* scan on

After the scan command it will show the nearby Bluetooth devices

Step3: Pair the mobile device with Raspberry PI. Once devices got connected you can list out the paired devices enter the below command

* paired-devices

Step 4: Go to Menu🡪Programming🡪Python 2.7 IDLE🡪File🡪New

Step 5: Write your server program in Raspberry PI

Step 6: Save & Run the program (Run🡪Run Module)

Step 7: First open and run the python script from Server device.

**SourceCode:**

**#Server**

import bluetooth

import time

server\_sock=bluetooth.BluetoothSocket( bluetooth.RFCOMM )

port = 1

server\_sock.bind(("",port))

server\_sock.listen(1)

client\_sock,address = server\_sock.accept()

print"accepted connection from",address

while True:

data=client\_sock.recv(1024)

print"received[%s]" % data

time.sleep(0.5)

text=raw\_input("enter your msg")

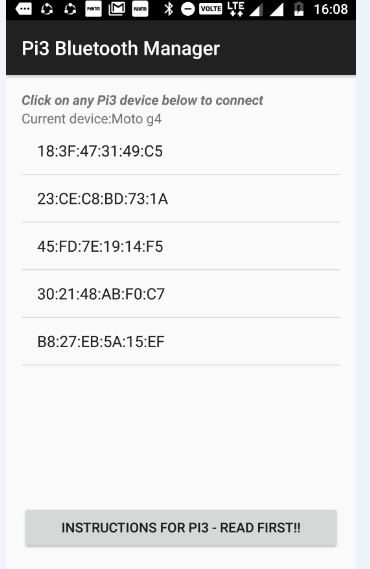
client\_sock.send(text)

time.sleep(0.5)

client\_sock.close()

server\_sock.close()

**Output:**



**7.(b) AIM:** Program to demonstrate PI to PI communication using Bluetooth protocol

**Description:** Bluetooth is a wireless technology used to transfer data between different electronic devices. It is mainly designed for communicating over short distances less than about 10m or 30ft.A computer network in which one centralized, powerful computer (called the server) is a hub to which many less powerful personal computers or workstations (called clients) are connected Radio frequency communication (RFCOMM) The Bluetooth protocol RFCOMM is a simple set of transport protocols, made on top of the L2CAP protocol, providing emulated RS-232 serial ports. RFCOMM is sometimes called serial port emulation.

**Procedure:**

**Package Installation Codes**

* sudo apt-get install bluetooth
* sudo apt-get install bluez
* sudo apt-get install python-bluez

**Step 1**: First you have to pair the two devices

Open the LX-Terminal in two Raspberry pi and enter the below commands one by one

* sudo bluetoothctl
* power on
* agent on
* default-agent
* discoverable on
* scan on

After the scan command it will show the nearby Bluetooth devices

**Step 2:** Then pair the device with your client using the below command

* pair <MAC address of client>

It will ask pairing request and click OK to connect

**Step 3:** Once devices got connected you canlist out the paired devices enter the below command

* paired-devices

**Step 5:** Go to Menu🡪Programming🡪Python 2.7 IDLE🡪File🡪New

**Step 6:** Write your program in separate file for server and client

**Step 7:** Save & Run the program (Run🡪Run Module)

**Step 8:** First open and run the python script from Server device. Then open and run the python script from client device

**Source Code:**

import bluetooth

import time

server\_sock=bluetooth.BluetoothSocket( bluetooth.RFCOMM )

port = 1

server\_sock.bind(("",port))

server\_sock.listen(1)

client\_sock,address = server\_sock.accept()

print"accepted connection from",address

while True:

data=client\_sock.recv(1024)

print"received[%s]" % data

time.sleep(0.5)

text=raw\_input("enter your msg")

client\_sock.send(text)

time.sleep(0.5)

client\_sock.close()

server\_sock.close()

**#Client**

import bluetooth

import time

bd\_addr = "E4:46:DA:A0:E2:50" # MAC address of server

port = 1

sock=bluetooth.BluetoothSocket( bluetooth.RFCOMM )

sock.connect((bd\_addr,port))

while True:

text=raw\_input("enter your msg:")

sock.send(text)

time.sleep(0.5)

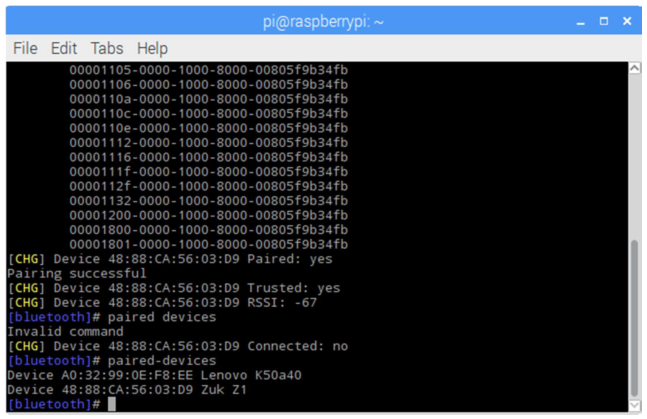
data=sock.recv(1024)

print"received[%s]" % data

time.sleep(0.5)

sock.close()

**Output:**



**Experiment-VIII: Application of ZIGBEE in IoT systems**

**8.(a) AIM:** To establish a communication between Zigbee Devices using XCTU platform.

**Description:** ZigBee is a new standard developed by the ZigBee Alliance for personal-area networks (PANs). The ZigBee Alliance is a consortium that promotes the ZigBee standard for a low-rate/low-power wireless sensor and control network.

The ZigBee protocol stack is built on top of IEEE 802.15.4, which defines the Media Access Control (MAC) and physical layers for low-rate wireless personal-area network (LR-WPAN).

The ZigBee standard offers a stack profile that defines the network, security, and application layers. Zigbee is a low-cost, low-power, wireless mesh network standard targeted at battery-powered devices in wireless control and monitoring applications. Zigbee delivers low-latency communication.

# **Procedure:**

**Step 1**: Click the Discover radio nodes in the same network button of the first radio module.

**Step 2:** Select the scanned ports and click NEXT ->FINISH.

**Step 3:** Select all the listed devices and click add selected devices.

**Step 4:** Click the device that needs to be configured and change the following parameters

|  |  |  |
| --- | --- | --- |
| **PARMETERS** | **CO-ORDINATOR** | **ROUTER** |
| ID | 2015 | 2015 |
| JV | - | Enabled[1] |
| CE | Enabled[1] | - |
| NI | COORD | ROUTER |
| AP | API Enabled [1] | API Enabled [1] |

**Step 5:** Open the XCTU console

1. Switch to the Consoles working mode.
2. Open the serial connection with the radio module.
3. Change to the console of the other Zigbee module.
4. Open the serial connection with the radio module.

**Step 6:** Generate the Transmit Request Frame

1. In the SENDER console, click Add new packet to the list.
2. Open the Frames Generator tool.
3. In the Protocol control, select Zigbee.
4. In the Frame type control, select 0x10 - Transmit Request.
5. In the 64-bit dest. address box, type the 64-bit address of the RECEIVER module.
6. In the RF data box, click the ASCII tab and type the message "Hello, this is SENDER!"
7. Click OK.
8. Click ADD FRAME.

**Step 7:** Send the Transmit Request frame

After you have created a Transmit Request frame, you must send it.

1. Select the frame in the XCTU Send frames section.
2. Click Send selected packet.
3. Select the received packet and set of information is displayed where you can see the received packet.

**Source Code:**

**#Coordinator code:**

import time

import serial

import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BCM)

GPIO.setwarnings(False)

GPIO.setup(23,GPIO.OUT)

ser = serial.Serial(

port='/dev/ttyUSB0',

baudrate = 9600,

parity=serial.PARITY\_NONE,

stopbits=serial.STOPBITS\_ONE,

bytesize=serial.EIGHTBITS,

timeout=1

)

counter=0

while 1:

ser.write(str.encode('Write counter: %d \n'%(counter)))

time.sleep(1)

counter += 1

**#Router code**:

import time

import serial

import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BCM)

GPIO.setwarnings(False)

GPIO.setup(23,GPIO.OUT)

ser = serial.Serial(

port='/dev/ttyUSB0',

baudrate = 9600,

parity=serial.PARITY\_NONE,

stopbits=serial.STOPBITS\_ONE,

bytesize=serial.EIGHTBITS,

timeout=1

)

counter=0

while 1:

x=ser.readline().strip()

print(x)

if x == 'a':

GPIO.output(23,GPIO.HIGH)

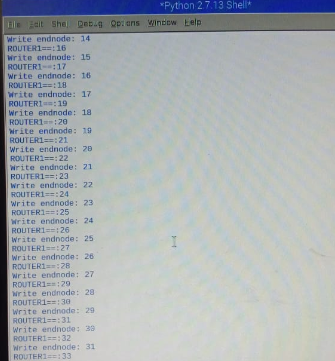
print "23 High"

time.sleep(3)

else:

GPIO.output(23,GPIO.LOW)

**Output:**



**Experiment-IX: Demonstrate communication protocol LoRa**

**9.(a) Aim:** To write a C program to blink the onboard LED

# **Description:** LoRa technology was developed by a company called Semtech and it is a new wireless protocol designed specifically for long-range, low-power communications. LoRa stands for Long Range Radio and is mainly targeted for M2M and IoT networks. This technology will enable public or multi-tenant networks to connect a number of applications running on the same network.

|  |  |
| --- | --- |
| **Specification** | **LoRa Feature** |
| Range | 2-5Km Urban (1.24-3.1 mi), 15Km suburban (9.3 mi) |
| Frequency | ISM 868/915 MHz |
| Standard | IEEE 802.15.4g |
| Modulation | Spread spectrum modulation type based on FM pulses which vary. |
| Capacity | One LoRa gateway takes thousands of nodes |
| Battery | Long battery life |
| LoRa Physical layer | Frequency, power, modulation and signalling between nodes and gateways |

**Procedure:**

**Step 1:** Download the latest Arduino Software(IDE)

**Step 2:** Install the IDE in the PC, open it and click **File** -->**Preference**

**Step 3:** Add below URL

<http://www.dragino.com/downloads/downloads/YunShield/package_dragino_yun_test_index.json> in the Additional Board Manager URLs

**Step 4:** Go to tools --> Boards --> Boards Manager, find Dragino board info and install it

**Step 5:** After install Dragino’s board info in the IDE, we can see the boards info from the IDE

**Step 6:** For LG01, we should choose Dragino Yun-UNO or LG01/OLG01

**Step 7:** Make sure your computer and the LG01 is in the same network

**Step 8**: In the IDE, select the correct network port

# **Source Code:**

int HEART\_LED=A2;

void setup() {

// initialize digital pin as an output.

pinMode(HEART\_LED, OUTPUT);

}

// the loop function runs over and over again forever

void loop() {

digitalWrite(HEART\_LED, HIGH); // turn the HEART\_LED on (HIGH is the voltage level)

delay(1000); // wait for a second

digitalWrite(HEART\_LED, LOW); // turn the HEART\_LED off by making the voltage LOW

delay(1000); // wait for a second

}

**Output:**

**Experiment-X: Publishing data on to Cloud using MQTT Protocol**

**10.(a) Aim:** Write a python program to publish message using MQTT protocol

**Description:** MQTT is a lightweight TCP/IP based protocol used in machine to machine communication. MQTT provides clients with a simple way to distribute telemetry information through a broker by using a publish/subscribe communication

**Source Code:**

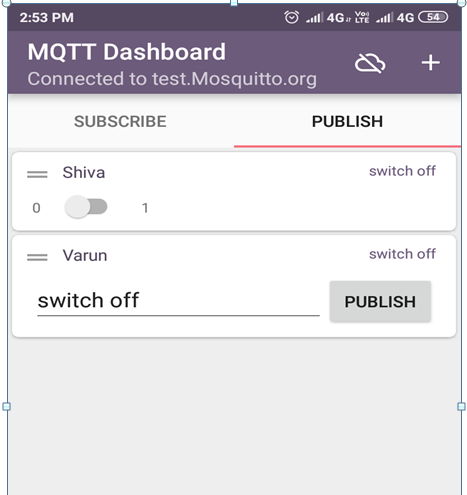
**# Publish Code:**

import paho.mqtt.publish as publish

publish.single("TOPIC/test", "Hello", hostname="test.mosquitto.org")

print("Done")

**Output:**

****

**10.(b) Aim:** Writea python program to publish sensor data (PIR – Passive Infrared) using MQTT protocol

**Description:** MQTT is a lightweight TCP/IP based protocol used in machine to machine communication. MQTT provides clients with a simple way to distribute telemetry information through a broker by using a publish/subscribe communication

**Package Installation Codes**

* sudo pip install paho-mqtt

**Source Code:**

**# Publish Code:**

import sys

sys.path.append('/home/pi/Adafruit-Raspberry-Pi-Python-Code-legacy/Adafruit\_MCP230xx')

sys.path.append('/usr/local/lib/python2.7/dist-packages')

from Adafruit\_MCP230XX import Adafruit\_MCP230XX

import time

import paho.mqtt.publish as publish

mcp = Adafruit\_MCP230XX(busnum = 1, address = 0x21, num\_gpios = 16)

# Set pins 0, 1 and 2 to output (you can set pins 0..15 this way)

mcp.config(0, mcp.INPUT) # PIR

mcp.pullup(0, 1)

def motion():

print(" --------------------MOTION DETECTION PROCESS--------------------")

i = mcp.input(0)

if i == 1:

print ("person Available \n")

publish.single("SMART/PIR","Person available", hostname="test.mosquitto.org")

if i == 0:

print ("person NOT Available \n")

publish.single("SMART/PIR1","person NOT Available", hostname="test.mosquitto.org")

while (True):

motion()

time.sleep(1)

**Output:**

**Experiment-XI: Read the data from the cloud and display them using MQTT Protocol**

**11.(a) Aim:** Write a python program to Subscribe message using MQTT protocol

**Description:** MQTT is a lightweight TCP/IP based protocol used in machine to machine communication. MQTT provides clients with a simple way to distribute telemetry information through a broker by using a publish/subscribe communication

**Package Installation Codes**

* sudo pip install paho-mqtt

**Source Code:**

**#Subscribe Code:**

import paho.mqtt.client as mqtt

def on\_connect(client, userdata, flags, rc):

print("Connected with result code "+str(rc))

client.subscribe("TOPIC/test")

def on\_message(client, userdata, msg):

print(msg.topic+" "+str(msg.payload))

if msg.payload == "Hello":

print("Received message #1, do something")

client = mqtt.Client()

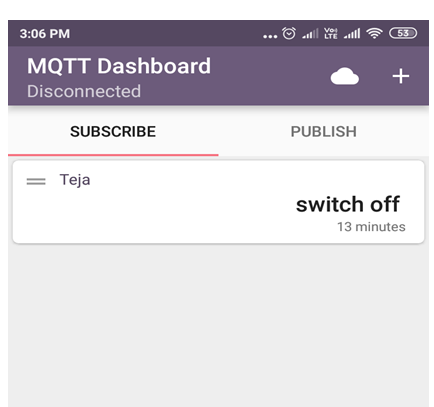
client.on\_connect = on\_connect

client.on\_message = on\_message

client.connect("test.mosquitto.org", 1883, 60)

client.loop\_forever()

**Output:**

****

**11.(b) Aim:** Writea python program to subscribe sensor data (PIR – Passive Infrared) using MQTT protocol

**Description:** MQTT is a lightweight TCP/IP based protocol used in machine to machine communication. MQTT provides clients with a simple way to distribute telemetry information through a broker by using a publish/subscribe communication

**Package Installation Codes**

* sudo pip install paho-mqtt

**Source Code:**

**# Subscribe Code:**

import paho.mqtt.client as mqtt

import RPi.GPIO as GPIO

import time

GPIO.setwarnings(False)

GPIO.setmode(GPIO.BOARD)

GPIO.setwarnings(False)

def on\_connect(client, userdata, flags, rc):

print("Connected with result code "+str(rc))

client.subscribe("SMART/PIR")

client.subscribe("SMART/PIR1")

def on\_message(client, userdata, msg):

print(msg.topic+" "+str(msg.payload))

if msg.payload =="Person available":

print "PERSON AVAILABLE"

print "----------------------------------------------------"

if msg.payload =="person NOT Available":

print "PERSON NOT AVAILABLE"

print "----------------------------------------------------"

client = mqtt.Client()

client.on\_connect = on\_connect

client.on\_message = on\_message

client.connect("test.mosquitto.org", 1883, 60)

client.loop\_forever()

**Output:**

**Experiment-XII:** **Demonstration of following RTOS concepts Timing, Multitasking, Round Robin & Semaphores**

**12.(a) Aim:** Program to Demonstrate RTOS concept Timing

**Description:** The timex() routine times a single execution of a specified function with up to eight integer arguments to be passed to the function. When execution us complete, timex() routine displays the execution time and a margin of error in miliseconds. If the execution was so fast relative to the clock rate that the time is meaningless(error > 50%), a warning message will appear. In such cases, use timexN() which will repeatedly execute the function until the time of a single iteration is known with reasonable certainty.

**Source Code:**

#include "vxWorks.h"

#include "timexLib.h"

#include "stdio.h"

#define ITERATIONS 200

int printit(void);

void timing() /\* Function to perform the timing \*/

{

FUNCPTR function\_ptr = printit; /\* a pointer to the function "printit" \*/

timex(function\_ptr,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL);

/\* Timing the "print" function \*/

}

int printit(void) /\* Function being timed \*/

{

int i;

for(i=0; i < ITERATIONS; i++)

/\* Printing the task id number and the increment variable "i" \*/

printf("Hello, I am task %d and is i = %d\n",taskIdSelf(),i);

return 0;

}

**Output:**

Hello, I am task 1323024 and is i =0

Hello, I am task 1311136 and is i =1

Hello, I am task 1395992 and is i =2

Hello, I am task 1308084 and is i =3

Hello, I am task 1306578 and is i =4

**12.(b) Aim:** Program to Demonstrate RTOS concept Multi Tasking

**Description**: Multitasking creates the appearance of many threads of execution running concurrently when, in fact, the kernel interleaves their execution on a basis of a scheduling algorithm. Each apparently independent program is called a task. Each task has its own context, which is the CPU environment and system resources that the task sees each time it is scheduled to run by the kernel.

The routine taskSpawn creates the new task context, which includes allocating and setting up the task environment to call the main routine (an ordinary subroutine) with the specified arguments. The new task begins at the entry to the specified routine. The arguments to taskSpawn() are the new task's name(an ASCII string), priority, an "options" word(also hex value), stack size(int), main routine address(also main routine name), and 10 integer arguments to be passed to the main routine as startup parameters.

**Source Code:**

#include "vxWorks.h"

#define ITERATIONS 10

void print(void);

spawn() /\* Subroutine to perform the spawning \*/

{

int i, taskId;

for(i=0; i < ITERATIONS; i++) /\* Creates ten tasks \*/

taskId = taskSpawn("tprint",90,0x100,2000,print,0,0,0,0,0,0,0,0,0,0);

}

void print(void) /\* Subroutine to be spawned \*/

{

printf("Hello, I am task %d\n",taskIdSelf()); /\* Print task Id \*/

}

**Output:**

Hello, I am task 1325024

Hello, I am task 1321136

Hello, I am task 1395992

Hello, I am task 1307084

Hello, I am task 1306568

**12.(c) Aim:** Program to Demonstrate RTOS concept Round Robin Task Scheduling

**Description:** The three tasks with the same priority print their task ids and task names on the console. Without round-robin scheduling, "taskOne" would usurp the processor until it was finished, and then "taskTwo" and "taskThree" would do likewise. In the event that "taskOne" was looping indefinitely, the other tasks would never get a chance to run.

To insure that the tasks get an equal share of the CPU time, a call is made to kernelTimeSlice(). This sets the time slice interval value to Time Slice. The Time Slice value is the time slice interval in terms of the number of clock ticks(which in the example and the M68040 is 60 ticks which is equivalent to one second). The sysClkRateGet() can be used to determine the number of clock ticks per second. Having setup the time slice in the manner above, the three tasks are spawned.

**Source Code:**

#include "vxWorks.h"

#include "taskLib.h"

#include "kernelLib.h"

#include "sysLib.h"

#include "logLib.h"

/\* function prototypes \*/

void taskOne(void);

void taskTwo(void);

void taskThree(void);

/\* globals \*/

#define ITER1 100

#define ITER2 10

#define PRIORITY 101

#define TIMESLICE sysClkRateGet()

#define LONG\_TIME 1000000

void sched(void) /\* function to create the three tasks \*/

{

int taskIdOne, taskIdTwo, taskIdThree;

if(kernelTimeSlice(TIMESLICE) == OK) /\* turn round-robin on \*/

printf("\n\n\n\n\t\t\tTIMESLICE = %d seconds\n\n\n", TIMESLICE/60);

/\* spawn the three tasks \*/

if((taskIdOne = taskSpawn("task1",PRIORITY,0x100,20000,(FUNCPTR)taskOne,0,0,0,0,0,0,0,

0,0,0)) == ERROR)

printf("taskSpawn taskOne failed\n");

if((taskIdTwo = taskSpawn("task2",PRIORITY,0x100,20000,(FUNCPTR)taskTwo,0,0,0,0,0,0,0,

0,0,0)) == ERROR)

printf("taskSpawn taskTwo failed\n");

if((taskIdThree = taskSpawn("task3",PRIORITY,0x100,20000,(FUNCPTR)taskThree,0,0,0,0,0,0,0

,0,0,0)) == ERROR)

printf("taskSpawn taskThree failed\n");

}

void taskOne(void)

{

int i,j;

for (i=0; i < ITER1; i++)

{

for (j=0; j < ITER2; j++)

logMsg("\n",0,0,0,0,0,0); /\* log messages \*/

for (j=0; j < LONG\_TIME; j++); /\* allow time for context switch\*/

}

}

void taskTwo(void)

{

int i,j;

for (i=0; i < ITER1; i++)

{

for (j=0; j < ITER2; j++)

logMsg("\n",0,0,0,0,0,0); /\* log messages \*/

for (j=0; j < LONG\_TIME; j++); /\* allow time for context switch\*/

}

}

void taskThree(void)

{

int i,j;

for (i=0; i < ITER1; i++)

{

for (j=0; j < ITER2; j++)

logMsg("\n",0,0,0,0,0,0); /\* log messages \*/

for (j=0; j < LONG\_TIME; j++); /\* allow time for context switch \*/

}

}

**Output:**

CPU: VxSim for Windows

Runtime Name: VxWorks

Runtime Version: 5.5

BSP version: 1.2/1

Created: Oct 2 2002, 10:16:06

WDB Comm Type: WDB\_COMM\_PIPE

WDB: Ready.

TIMESLICE = 1 seconds

0xc81140 (task1):

0xc81140 (task1):

0xc81140 (task1):

0xc81140 (task1):

0xc81140 (task1):

0xc81140 (task1):

0xc81140 (task1):

0xc81140 (task1):

0xc78fc8 (task2):

0xc78fc8 (task2):

0xc78fc8 (task2):

0xc78fc8 (task2):

0xc78fc8 (task2):

0xc78fc8 (task2):

0xc78fc8 (task2):

0xc78fc8 (task2):

0xc70e50 (task3):

0xc70e50 (task3):

0xc70e50 (task3):

0xc70e50 (task3):

0xc70e50 (task3):

0xc70e50 (task3):

0xc70e50 (task3):

0xc70e50 (task3):

**12.(d) Aim:** Program to Demonstrate RTOS concept Semaphores

**Description:** VxWorks semaphores are highly optimized and provide the fastest intertask communication mechanisms in VxWorks. Semaphores are the primary means for addressing the requirements of both mutual exclusion and task synchronization.

A binary semaphore can be viewed as a flag that is available or unavailable. When a task takes a binary semaphore, using semTake(), the outcome depends on whether the semaphore is available or unavailable at the time of the call. If the semaphore is available, then the semaphore becomes unavailable and then the task continues executing immediately. If the semaphore is unavailable, the task is put on a queue of blocked tasks and enters a state of pending on the availability of the semaphore.

When a task gives a binary semaphore, using semGive(), the outcome depends on whether the semaphore is available or unavailable at the time of the call.If the semaphore is already available, giving the semaphore has no effect at all. If the semaphore is unavailable and no task is waiting to take it, then the semaphore becomes available. If the semaphore is unavailable and one or more tasks are pending on its availablity, then the first task in the queue of pending tasks is unblocked, and the semaphore is left unavailable.

The two tasks(taskOne and taskTwo), are competing to update the value of a global variable, called "global." The objective of the program is to toggle the value of the global variable(1s and 0s). taskOne changes the value of "global" to 1 and taskTwo changes the value back to 0. Without the use of the semaphore, the value of "global" would be random and the value of "global" would be corrupted.

**Source Code:**

#include "vxWorks.h"

#include "taskLib.h"

#include "semLib.h"

#include "stdio.h"

/\* function prototypes \*/

void taskOne(void);

void taskTwo(void);

/\* globals \*/

#define ITER 10

SEM\_ID semBinary;

int global = 0;

void binary(void)

{

int taskIdOne, taskIdTwo;

/\* create semaphore with semaphore available and queue tasks on FIFO basis \*/

semBinary = semBCreate(SEM\_Q\_FIFO, SEM\_FULL);

/\* Note 1: lock the semaphore for scheduling purposes \*/

semTake(semBinary,WAIT\_FOREVER);

/\* spawn the two tasks \*/

taskIdOne = taskSpawn("t1",90,0x100,2000,(FUNCPTR)taskOne,0,0,0,0,0,0,0,0,0,0);

taskIdTwo = taskSpawn("t2",90,0x100,2000,(FUNCPTR)taskTwo,0,0,0,0,0,0,0,0,0,0);

}

void taskOne(void)

{

int i;

for (i=0; i < ITER; i++)

{

semTake(semBinary,WAIT\_FOREVER); /\* wait indefinitely for semaphore \*/

printf("I am taskOne and global = %d......................\n", ++global);

semGive(semBinary); /\* give up semaphore \*/

}

}

void taskTwo(void)

{

int i;

semGive(semBinary); /\* Note 2: give up semaphore(a scheduling fix) \*/

for (i=0; i < ITER; i++)

{

semTake(semBinary,WAIT\_FOREVER); /\* wait indefinitely for semaphore \*/

printf("I am taskTwo and global = %d---------------\n", --global);

semGive(semBinary); /\* give up semaphore \*/

}

}

**Output:**

CPU: VxSim for Windows

Runtime Name: VxWorks

Runtime Version: 5.5

BSP version: 1.2/1

Created: Oct 2 2002, 10:16:06

WDB Comm Type: WDB\_COMM\_PIPE

WDB: Ready.

hello ,i am task 13231680

hello ,i am task 13111136

hello ,i am task 13095992

hello ,i am task 13080848

hello ,i am task 13065704

hello ,i am task 13050560

hello ,i am task 13035416

hello ,i am task 13020272

hello ,i am task 13005128

hello ,i am task 12989984