

Experiment - 3.1

1. AIM:

Implement building blocks for Android Application using different layouts (such as linear, relative and absolute).

2. CODE:

MainActivity.java

```
package com.example.myapplication9;
import android.content.Intent;
import android.os.Bundle;
import android.widget.Button;
import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Button linearLayoutButton = findViewById(R.id.linear_layout_button);
        Button relativeLayoutButton =
            findViewById(R.id.relative_layout_button);
        Button absoluteLayoutButton =
            findViewById(R.id.absolute_layout_button);
        linearLayoutButton.setOnClickListener(v -> {
            Intent intent = new Intent(MainActivity.this,
            LinearLayoutActivity.class);
            startActivity(intent);
        });
        relativeLayoutButton.setOnClickListener(v -> {
            Intent intent = new Intent(MainActivity.this,
            RelativeLayoutActivity.class);
            startActivity(intent);
        });

        absoluteLayoutButton.setOnClickListener(v -> {
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
        Intent intent = new
Intent(MainActivity.this, AbsoluteLayoutActivity.class);
startActivity(intent);
    });
}
}
```

activity_main.xml

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button android:text="@string/linear_layout"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/linear_layout_button"
        android:textSize="18sp"
        android:layout_marginTop="40dp"
        android:layout_gravity="center_horizontal"
        />

    <Button android:text="@string/relative_layout"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/relative_layout_button"
        android:textSize="18sp"
        android:layout_gravity="center_horizontal"/>

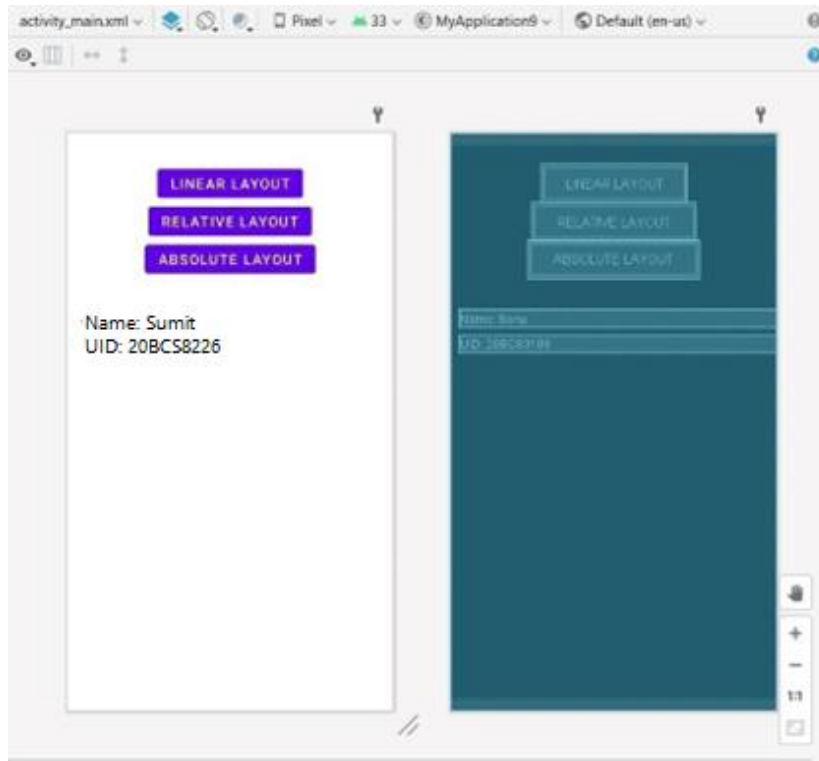
    <Button android:text="@string/absolute_layout"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/absolute_layout_button"
        android:textSize="18sp"
        android:layout_gravity="center_horizontal"/>

    <TextView android:id="@+id/textView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="16sp"
        android:layout_marginTop="40dp"
        android:layout_marginStart="10dp"
        android:text="@string/Simran" />

    <TextView android:id="@+id/textView2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="16sp"
        android:layout_marginTop="10dp"
```

```
android:layout_marginStart="10dp"
android:text="@string/20BCS3400" />
```

```
</LinearLayout>
```



LinearLayoutActivity.java

```
package com.example.myapplication9; import android.os.Bundle;

import androidx.appcompat.app.AppCompatActivity; public class

LinearLayoutActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_linear_layout); }
}
```

activity_linear_layout.xml

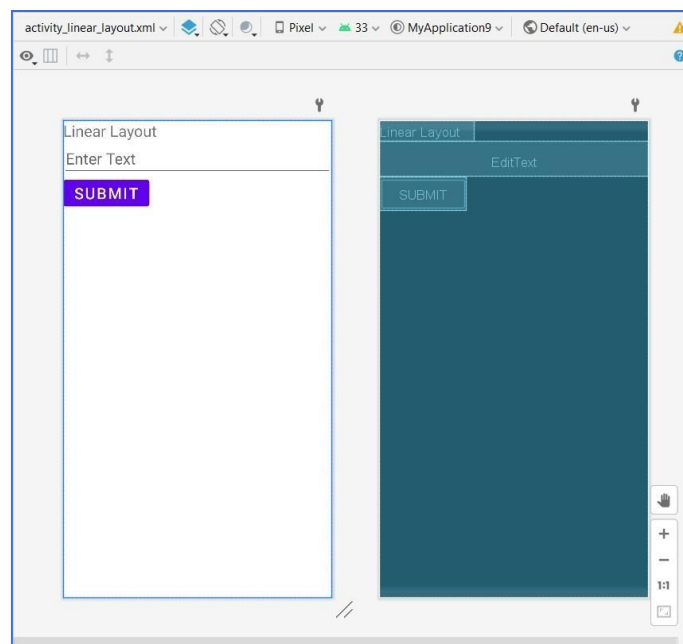
```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
```

```
<TextView
    android:text="@string/element_11"
    android:textSize="24sp"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
/>

<EditText
    android:hint="@string/element_22"
    android:textSize="24sp"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
/>

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="24sp"
    android:text="@string/element_33" />

</LinearLayout>
```



RelativeLayoutActivity.java

```
package com.example.myapplication9; import android.os.Bundle;

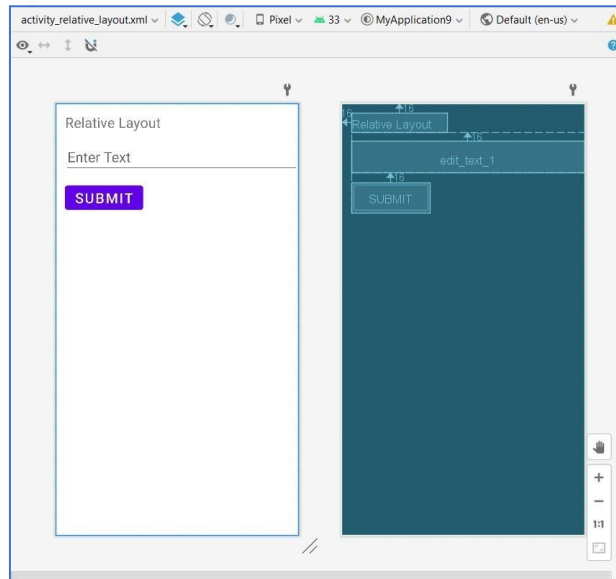
import androidx.appcompat.app.AppCompatActivity; public class
RelativeLayoutActivity extends AppCompatActivity {

    @Override
```

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_relative_layout  
    );  
}  
}
```

activity_relative_layout.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent">  
  
    <TextView  
        android:text="@string/element_1"  
        android:textSize="24sp"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:id="@+id/text_view_1"  
        android:layout_alignParentTop="true"  
        android:layout_alignParentStart="true"  
        android:layout_marginStart="16dp"  
        android:layout_marginTop="16dp"/>  
  
    <EditText android:hint="@string/element_2"  
        android:textSize="24sp"  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content"  
        android:id="@+id/edit_text_1"  
        android:layout_below="@id/text_view_1"  
        android:layout_alignStart="@id/text_view  
_1" android:layout_marginTop="16dp"/>  
  
    <Button android:text="@string/element_3"  
        android:textSize="24sp"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:id="@+id/button_1"  
        android:layout_below="@id/edit_text_1"  
        android:layout_alignStart="@id/edit_text  
_1" android:layout_marginTop="16dp"/>  
  
</RelativeLayout>
```



AbsoluteLayoutActivity.java

```
package com.example.myapplication9; import android.os.Bundle;

import androidx.appcompat.app.AppCompatActivity; public class

AbsoluteLayoutActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_absolute_layout)
    ; }
}
```

activity_absolute_layout.xml

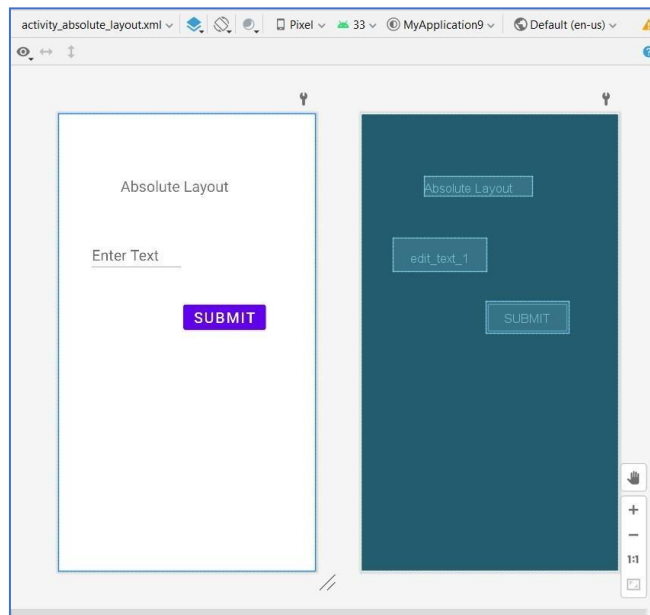
```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:text="@string/element_111"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/text_view_1"
        android:textSize="24sp"
        android:layout_x="100dp"
        android:layout_y="100dp"/>
```

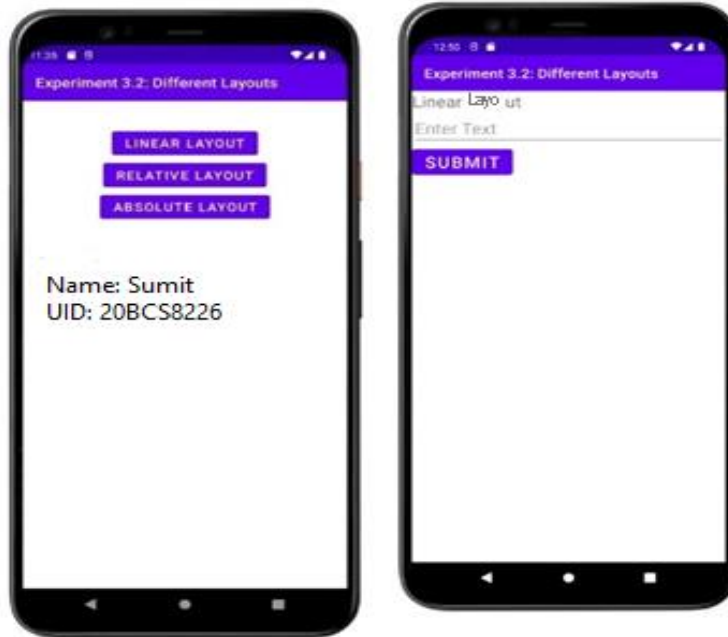
```
<EditText
    android:hint="@string/element_222"
    android:textSize="24sp"
    android:layout_width="150dp"
    android:layout_height="wrap_content"
    android:id="@+id/edit_text_1"
    android:layout_x="50dp"
    android:layout_y="200dp"/>

<Button
    android:text="@string/element_333"
    android:textSize="24sp"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/button_1"
    android:layout_x="200dp"
    android:layout_y="300dp"/>
```

```
</AbsoluteLayout>
```



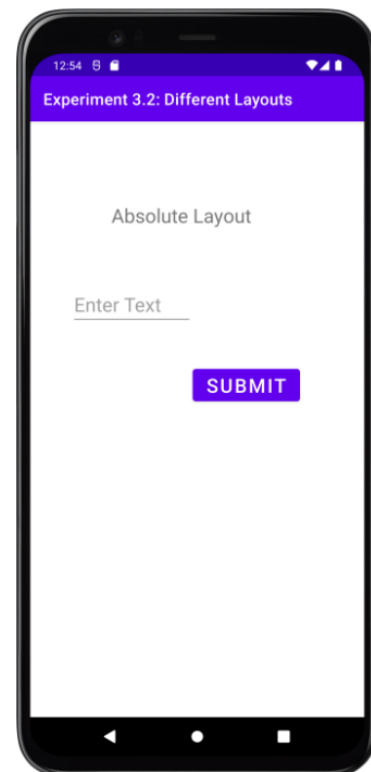
3. OUTPUT:



Output for Linear Layout



Output for Relative Layout



Output for Absolute Layout

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

“Experiment 2.3”

1. Aim:

To display data generated by sensor on LCD using Arduino/Raspberry Pi.

2. Objective:

- Learn about Arduino Uno.
- Learn about IoT programming.

3. Components Required:

You will need the following components –

- Arduino Uno
- LCD
- Breadboard
- Jumper Wires

4. Procedure:

Interfacing 16×2 LCD to Arduino uno:

LCD modules form a very important part in many arduino based embedded system designs. So, the knowledge on interfacing LCD module to arduino is very essential in designing embedded systems. This section of the article is about interfacing an Arduino to 16×2 LCD. JHD162A is the LCD module used here. JHD162A is a 16×2 LCD module based on the HD44780 driver from Hitachi. The JHD162A has 16 pins and can be operated in 4-bit mode (using only 4 data lines) or 8-bit mode (using all 8 data lines). Here we are using the LCD module in 4-bit mode. First, I will show you how to display a plain text messages on the LCD module using arduino and then, I have designed a useful project using LCD and arduino – a digital thermometer. Before going in to the details of the project, let's have a look at the JHD162A LCD module.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

5. Code:

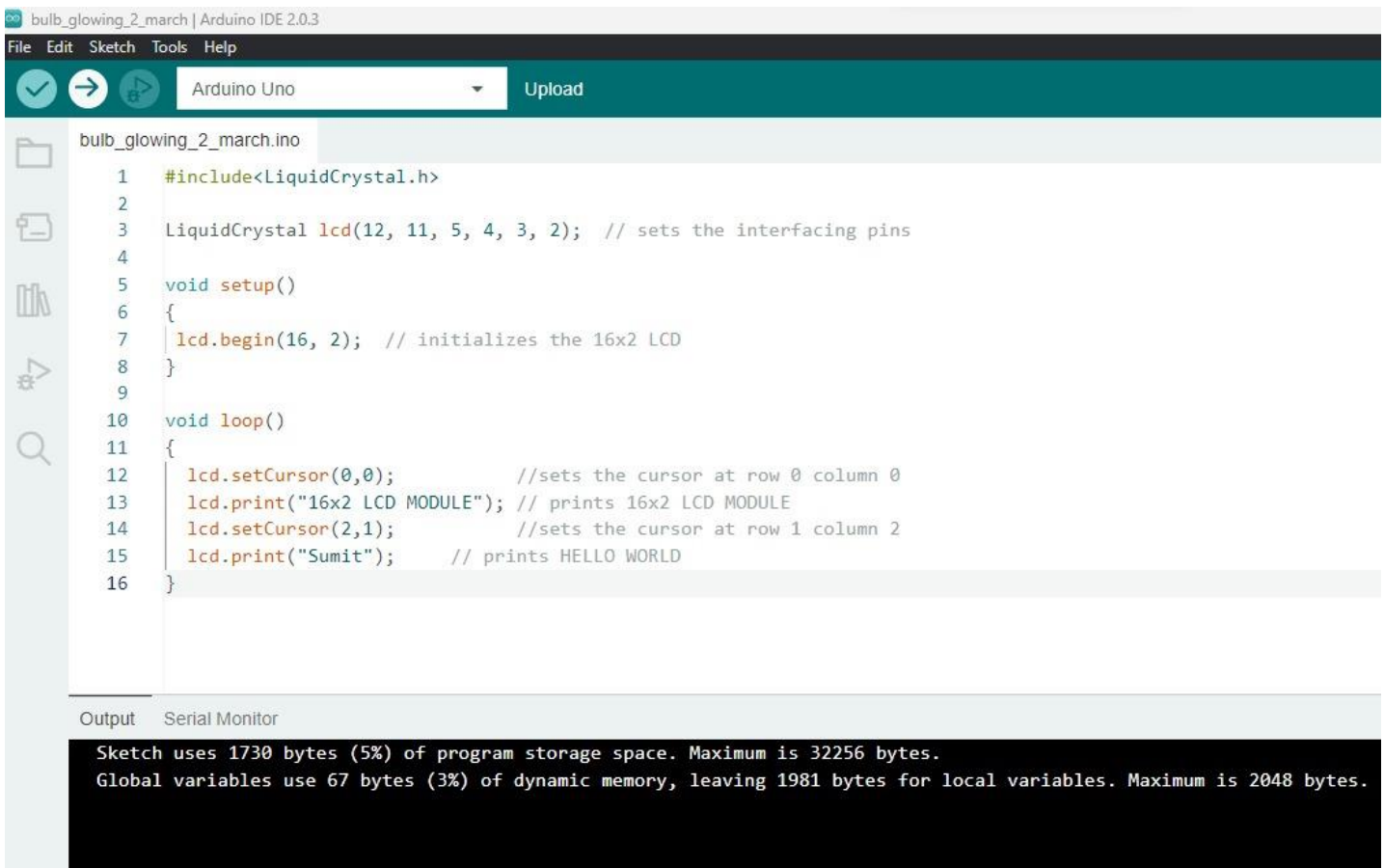
```
#include<LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2); // sets the interfacing pins

void setup()
{
    lcd.begin(16, 2); // initializes the 16x2 LCD
}

void loop()
{
    lcd.setCursor(0,0);           //sets the cursor at row 0 column 0
    lcd.print("16x2 LCD MODULE"); // prints 16x2 LCD MODULE
    lcd.setCursor(2,1);           //sets the cursor at row 1 column 2
    lcd.print("Sumit");           // prints HELLO WORLD
}
```

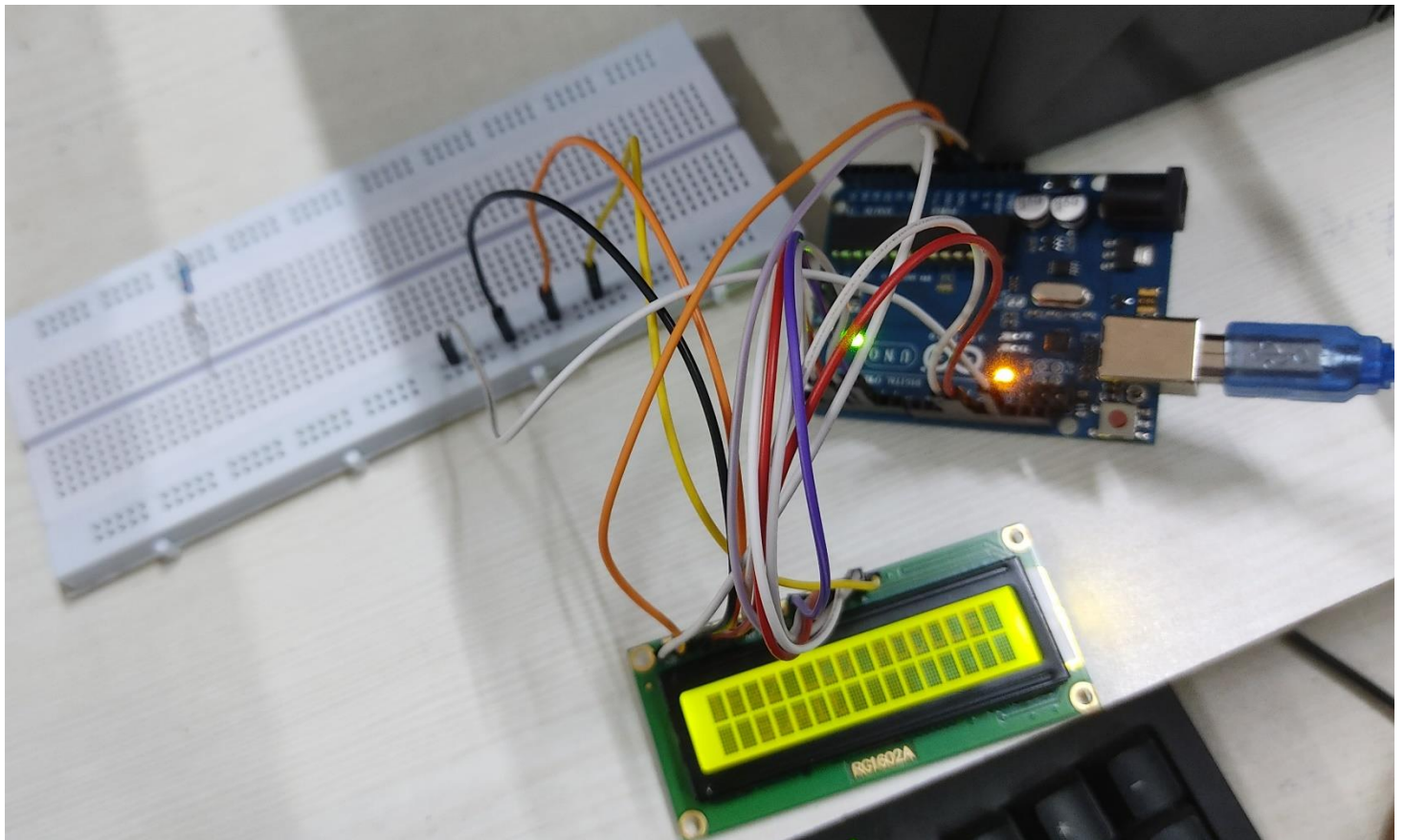
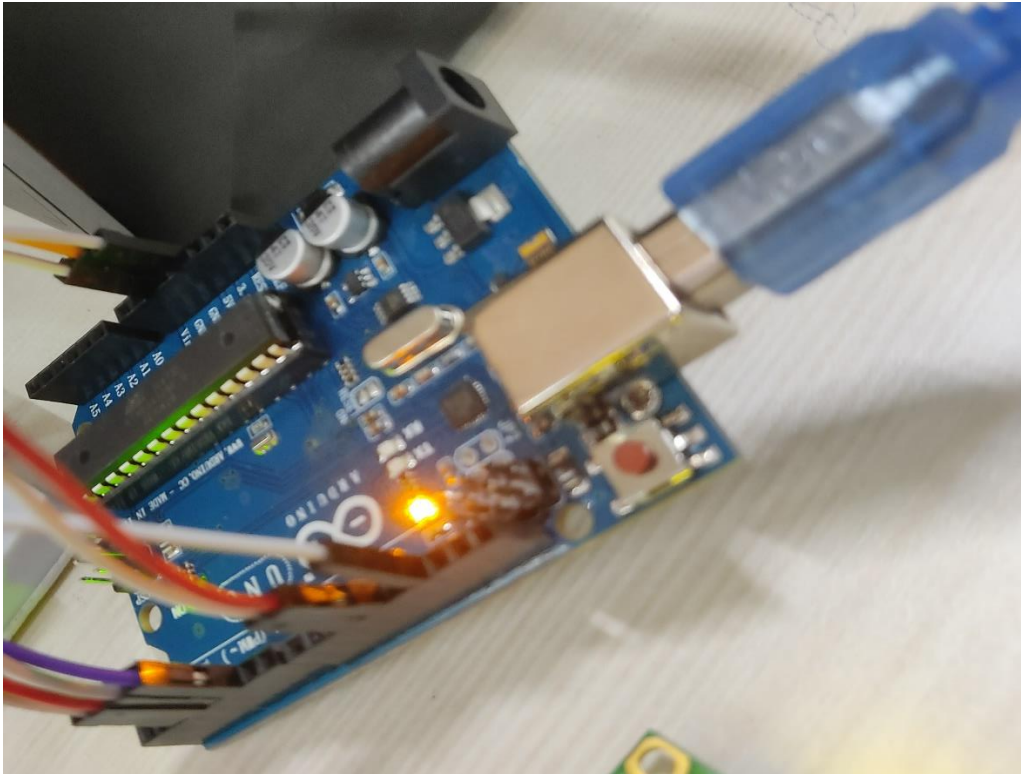
6. Output:



The screenshot displays the Arduino IDE 2.0.3 interface. The main editor window shows the code for 'bulb_glowing_2_march.ino', which is identical to the code provided in the previous block. The IDE's menu bar includes File, Edit, Sketch, Tools, and Help. The toolbar features a checkmark, a right arrow, a play button, a dropdown menu currently set to 'Arduino Uno', and an 'Upload' button. The left sidebar contains icons for file explorer, sketchbook, libraries, and search. The bottom status bar indicates 'Output' and 'Serial Monitor' tabs. The 'Output' window is active, showing the following memory usage information:

```
Sketch uses 1730 bytes (5%) of program storage space. Maximum is 32256 bytes.
Global variables use 67 bytes (3%) of dynamic memory, leaving 1981 bytes for local variables. Maximum is 2048 bytes.
```

DEPARTMENT OF **COMPUTER SCIENCE & ENGINEERING**



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING



Learning outcomes (What I have learnt):

- Learnt about displaying text on LED screen.
- Learnt about IoT programming.
- Learnt about the working of Arduino Uno.

“Experiment 3.1”

Aim:

Interfacing Air Quality Sensor (MQ135), displays data on LCD.

Objective:

1. Learn about interfacing.
2. Learn about IOT programming.

Components Required:

1. Arduino Uno R3
2. MQ 135 AirQuality Sensor Module
3. Male to Female Jumper Wire
4. Software: Arduino IDE

About Air Quality Sensor:

MQ-135 sensor belongs to the MQ series that are used to detect different gasses present in the air. The MQ-135 sensor is used to detect gases such as NH₃, NO_x, alcohol, Benzene, smoke, CO₂, etc. steel exoskeleton houses a sensing device within the gas sensor module.

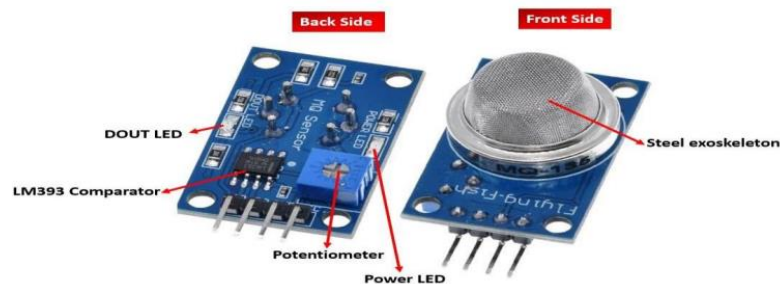


Figure: MQ 135 Sensor

Pin out:

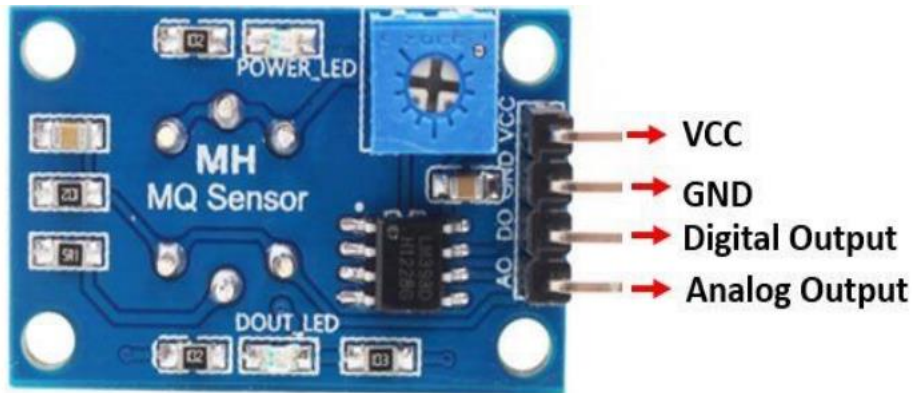


Figure: MQ-135 Sensor Pin out

This sensor has 4 pins:

- 5V: Module power supply – 5 V
- GND: Ground
- DOUT: Digital output
- AOUT: Analog output

Circuit:

Interfacing MQ-135 Gas Sensor with Arduino:

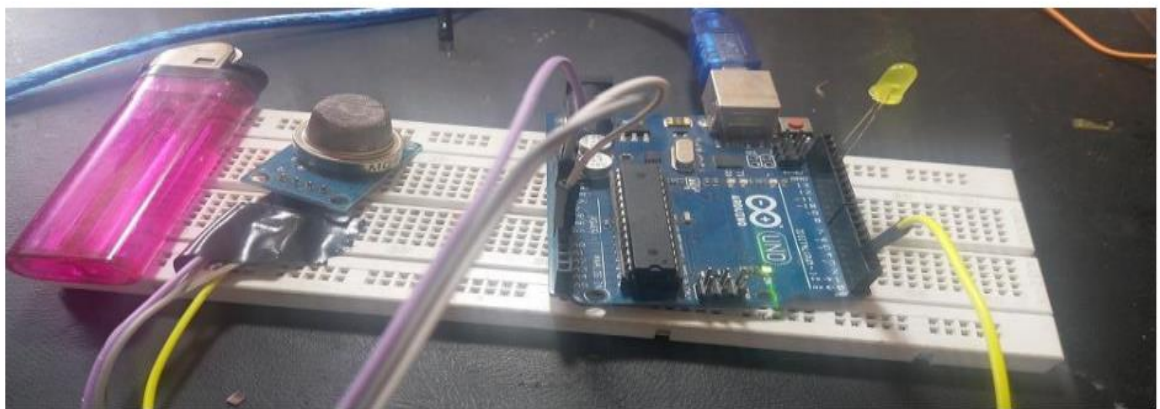


Figure: Interfacing of MQ135 with Arduino

The table below shows the connections you need to make between the MQ3 sensor module and Arduino using both the analog output and the digital output pins of the sensor.

MQ-135 Module	Arduino
VCC	5V
GND	GND
AO	A0
DO	Pin 2

Connect MQ-135 sensor's VCC pin with 5V terminal of Arduino UNO. This will power up the sensor. Additionally, we will connect the analog pin AO with A0 and DO with Pin 2 of Arduino UNO. Both the devices will be commonly grounded. Follow the connection diagram below, to connect your devices accordingly.

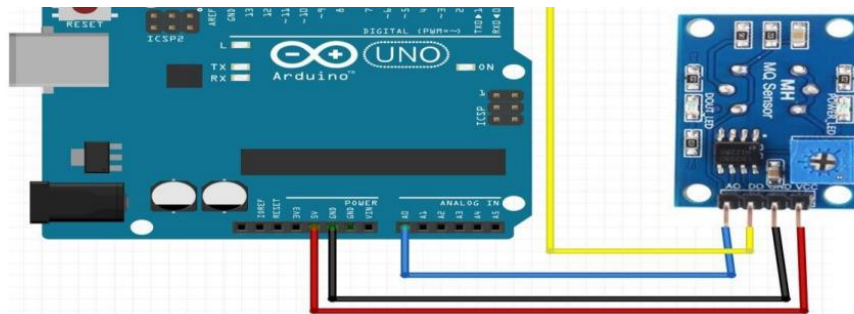


Figure: Arduino UNO with MQ-135 Module using both digital and analog outputs

MQ-135 Gas Detection Arduino Sketch

Open your Arduino IDE and go to File > New. Copy the code below in that file.

This sketch will read both the analog and digital outputs of the sensor. If the analog output is greater than 400 then an LED connected at Arduino pin 2 will turn ON. Otherwise, turn the LED OFF and print both the analog and digital output readings on the serial monitor.

```
int sensorValue;
int digitalValue;
void setup()
{
    Serial.begin(9600); // sets the serial port to 9600
    pinMode(13, OUTPUT);
    pinMode(2, INPUT);
}
void loop()
{
    sensorValue = analogRead(0); // read analog input pin 0
    digitalValue = digitalRead(2);
```

```
        if (sensorValue > 400)
        {
            digitalWrite(13, HIGH);
        }
        else
            digitalWrite(13, LOW);
        Serial.println(sensorValue, DEC); // prints the value read
        Serial.println(digitalValue, DEC);
        delay(1000); // wait 100ms for next reading
    }
}
```

How the Code Works

Create two int variables to hold the analog and digital output readings.

```
int sensorValue;
```

```
int digitalValue;
```

Inside the setup() function, we will open the serial communication at a baud rate of 9600. Then configure pin2 connected with the DO pin of the sensor as an input and pin13 connected with the LED's anode pin as an output.

```
void setup()
{
    Serial.begin(9600); // sets the serial port to 9600
    pinMode(13, OUTPUT);
    pinMode(2, INPUT);
}
```

In the infinite loop(), we will use analogRead() on the A0 pin and save the value in 'sensorValue.' Likewise, we will read the digital output on pin2 using digitalRead() and save the value in 'digitalValue.' Next, using an if-else statement we will check if the analog reading is greater than 400 or not. If it is then turn the LED ON. Otherwise, leave the LED OFF and display the current analog and digital readings in the serial monitor.

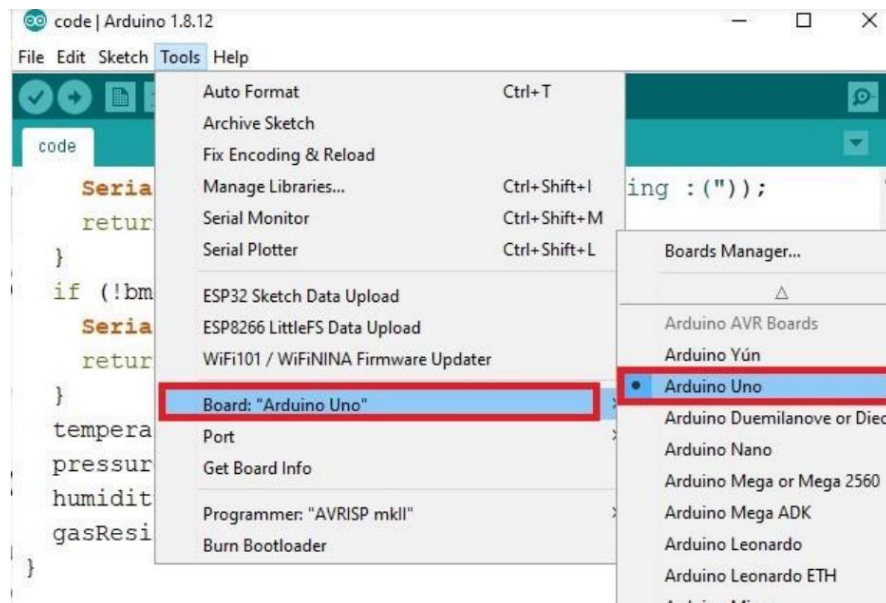
```
void loop()
{
    sensorValue = analogRead(0); // read analog input pin 0
    digitalValue = digitalRead(2);
    if (sensorValue > 400)
```



```
{  
    digitalWrite(13, HIGH);  
}  
  
else digitalWrite(13, LOW);  
  
Serial.println(sensorValue, DEC); // prints the value read  
Serial.println(digitalValue, DEC);  
delay(1000); // wait 100ms for next reading  
}
```

Demonstration

To see the demonstration of the above code, upload the code to Arduino. But, before uploading code, make sure to select the Arduino board from Tools > Board and also select the correct COM port to which the Arduino board is connected from Tools > Port.



On the serial monitor, you can see the values of the analog pin being detected. Currently, in my case, they are around about 150, which indicates normal air.

- Normal air returns approximately 100-150
- Alcohol returns approximately 700
- Lighter gas returns approximately 750

“Experiment 3.2”

Aim:

Real Time application of controlling actuators through Bluetooth application using Arduino.

Objective:

1. Learn about interfacing.
2. Learn about IOT programming.

Components Required:

1. 8 Male/Male Jumper Wires
2. 1 HC-05 Bluetooth Module
3. 1 (5 mm) LED: Red
4. 1 Arduino UNO
5. 1 Resistor
6. 1k ohm

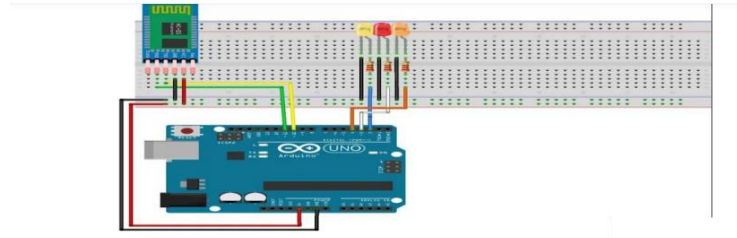
Apps and platforms:

1. Arduino IDE
2. MIT App Inventor

Step 1 Here is what you need to control Led's with Bluetooth:

- Arduino
- HC-05 Bluetooth module
- Solder less breadboard
- 3 Led's
- 3 220Ω resistors
- Wires
- Most importantly your phone and a downloaded Bluetooth app (Arduino Bluetooth Controller, which offers many different features)

Step 2: Circuit



Bluetooth module connection:

- Connect the BT module's Rx pin to pin 11 on the Arduino
- Connect the BT module's Tx pin to pin 10 on the Arduino
- Connect up the Gnd and Vcc (5v) to the Arduino

Led's connection

- Connect all the cathodes (short pin) of the led to Gnd
- Connect each anode to a 220 Ω resistor
- Connect a resistor to Arduino pin 2,3 and 4

If the led on the Bluetooth Module is blinking quickly then it is ready to pair to your phone, if not then check your connections

Code:

```
const int LED = 5;
char switchstate;
void setup()
{
    //Here the code only runs once.
    Serial.begin(9600);
    pinMode(LED, OUTPUT); }
void loop()
{
    //This code repeats. This is our main code.
    while(Serial.available()>0)
    {
        //code to be executed only when Serial.available()>0
        switchstate = Serial.read();
        Serial.print(switchstate);
        Serial.print("\ ");
        delay(15);
        if(switchstate == '1')
```

```
{      //Checking if the value from app is '1'
digitalWrite(5, HIGH);
}
else if(switchstate == '0')
{      //Else, if the value from app is '0',
digitalWrite(5, LOW);      //Write the component on pin 5(LED) low.
}
}
}
```

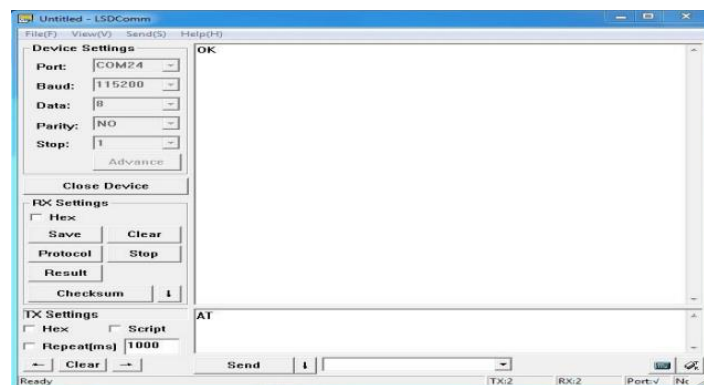
1. How to program on the Arduino Bluetooth Module

After understanding the software configurations, here's how to configure Bluetooth with a PC. For hardware connection, do refer to the "Hardware configurations" section. You'll find that the flashing blue LED on the module illustrates no connection is set up

Step 1: Open a serial terminal and set Baud Rate: 115200, Databits: 8, Stopbits: 1, and no flow control like above

Step 2: Send "AT" to Bluetooth with the serial terminal to check if you receive an "OK"
The Bluetooth only respond AT commands either when: No connection is set up All commands were seen as string and sent out

You can distinguish the above status in step 2 through LED indications



We used two Bluetooth that were connected with the PC, with one set as central while the other as Peripheral. Several seconds later, they find each other, and the LED stops flashing connected!

2. How to pair Arduino Bluetooth Module with iPhone and Android

Since the Grove – Blueseed – Dual model (HM13) have two protocol: Bluetooth EDR (Enhanced Data Rate) and Bluetooth Low Energy (BLE), it can communicate with either Android or iPhones! For this part of the tutorial, we'll use an iPhone to demonstrate how you can interact with Bluetooth!

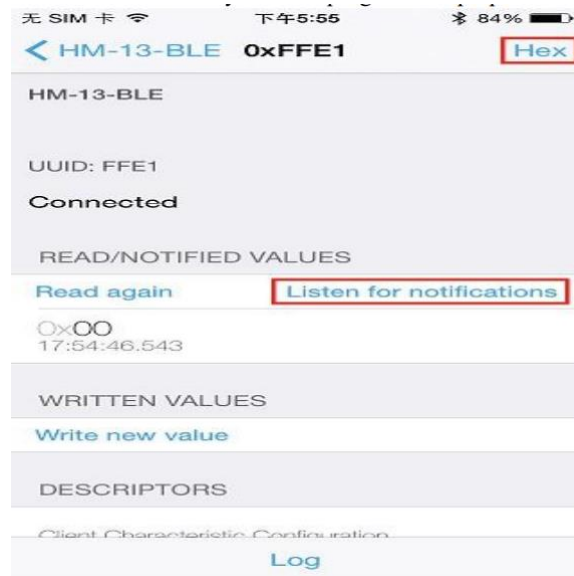
Note: The tutorial below is run on an older version of ios but it should still work the same

Step 1: Power the Bluetooth and configure it as a Peripheral role

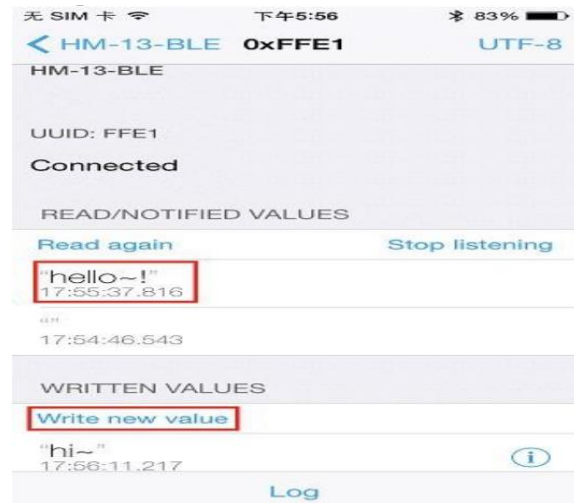
Step 2: Search Light Blue in the App Store and install it

Step 3: Launch the app, and connect to “HM-13-BLE”

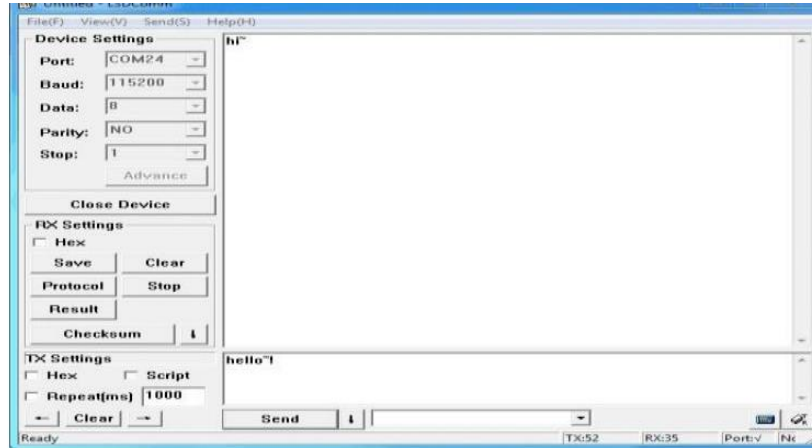
Step 4: Touch on properties and hit “listen for notifications” to enable data receiving
There's a “Hex” key on the top right under properties to change data format as well



Step 5: Hit “Write new value” and write some words to start sending data to the PC



With the serial terminal, you can transfer data from the PC to iPhone as well:



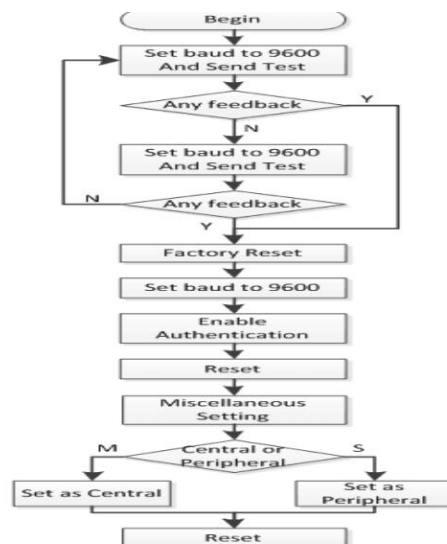
3. Bluetooth Data transmission guide between two Arduino boards

Now after all the above steps, are you ready to code? In this final section, we'll use two Arduino Uno and a pair of Bluetooth modules to get started!

Step 1: Set up the connection mentioned in the hardware configurations section

Step 2: Assign the Bluetooth to the Central role by modifying the text to “#define MASTER 1” The program of Central and Peripheral use the same code but there's a difference in the micro define at the beginning of the program

Step 3: Follow the flow chart below for initialization of the program



After the initialization, the Central and Peripheral will do different things; Central will send a message to the Peripheral interval and print what's received from the Peripheral while the Peripheral only responds to the central

Step 4: Download the test code and open HM-13_SW.ino with Arduino IDE, compile and download to Arduino Uno. Remember to configure the Bluetooth to the different role by modifying the macro at the beginning

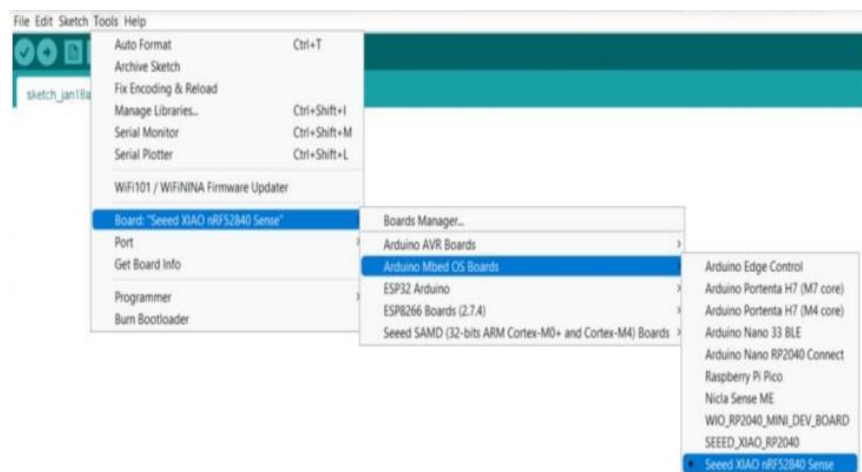
Step 5: After the program is downloaded, open two serial terminal windows and wait for the Bluetooth connection. A connection is indicated by: LEDs on the Bluetooth modules will flash for a few seconds, stop flashing, and kept on. According to the program written, the Central will now send a message to the Peripheral continually and get feedback every time.

Arduino-supported MCU with Bluetooth 5.0

As the first wireless product in the Seeed XIAO family, Seeed XIAO BLE & BLE Sense has equipped a powerful Nordic nRF52840 MCU which is designed in a **Bluetooth 5.0 module**, built around a 32-bit ARM® Cortex™-M4 CPU with Floating-Point Unit(FPU) operating at 64Mhz.

Seeed XIAO BLE nRF52840 -Supports Arduino / MicroPython -Bluetooth5.0 with Onboard Antenna

Seeed XIAO BLE nRF52840 Sense – TinyML/TensorFlow Lite- IMU / Microphone – Bluetooth5.0





“Experiment 3.3”

Aim:

Study the Implementation of Zigbee Protocol using Raspberry Pi/Arduino.

Introduction

XBee wireless transceivers provide a quick, easy way to add wireless communication to any system. This page will outline how to set up two XBee Pro Series 2 transceivers for communication with each other.

Hardware Required:

1. 2 XBee Pro S2 Transceiver
2. 2 UART to USB adapter board
3. 1 USB Cord

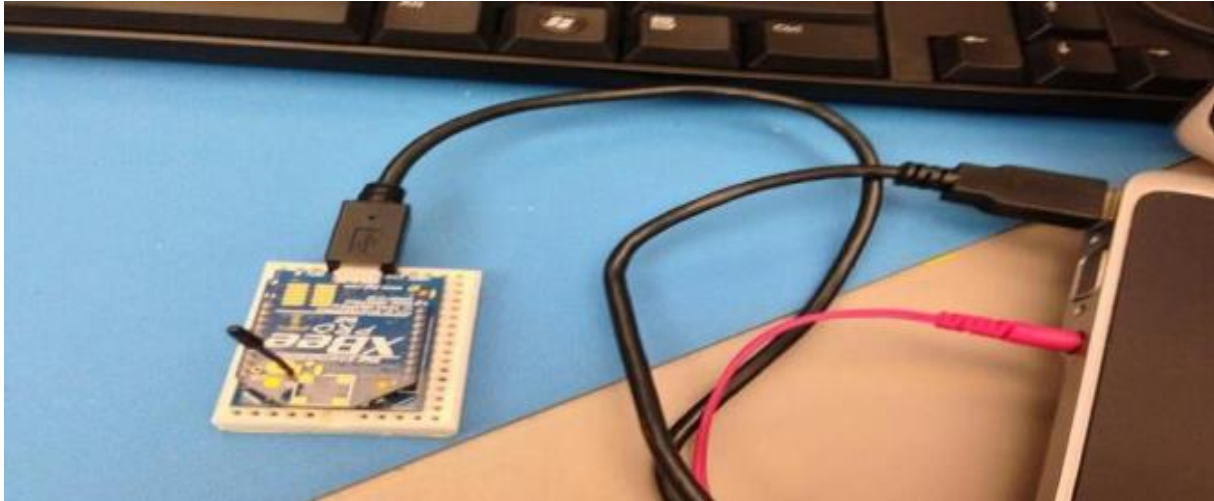
XBee Wireless Communication Setup

Step 1: Download X-CTU Software

The X-CTU software is free to download and provides a simple interface to configure and update your XBee transceivers. With this software firmware updates are a breeze and configuration is simple. The software can be downloaded from Digi's website.

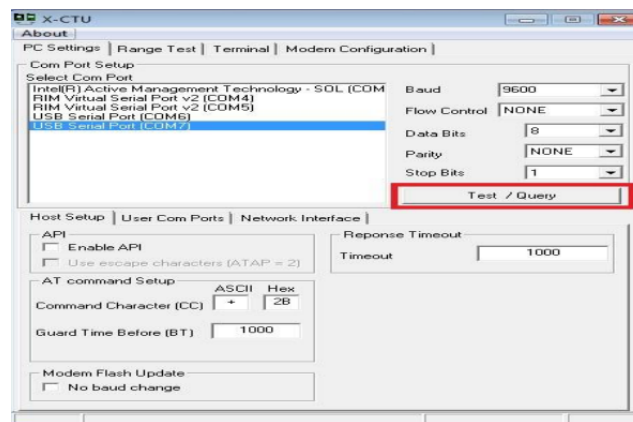
Step 2: Put together your XBee breakout board

The XBee transceivers have a 2mm pin spacing, which does not allow them to be plugged into a standard 0.1-inch breadboard. There are, however, several different breakout boards available that allow the transceiver to be inserted into a breadboard. The various adapter boards also allow for connection through USB or serial to your computer. The breakout board used here has a UART to USB conversion circuit and allows the XBee to be connected to the computer and XCTU software easily.



Step 3: Configure 1st XBee as a coordinator

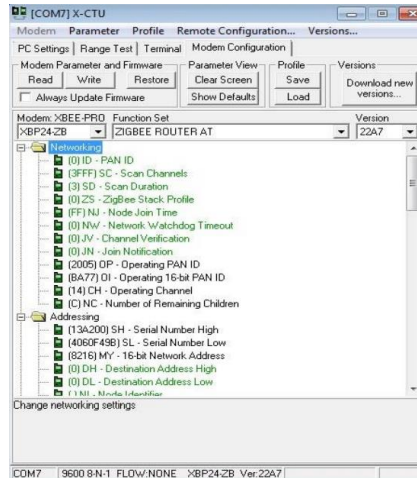
When opening the X-CTU software you should see a window like that shown. After selecting the proper COM port click the Test/Query button.



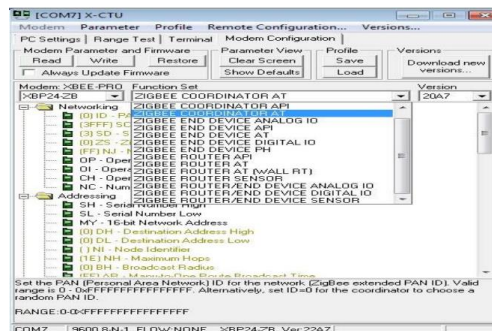
After selecting the Test/Query button, you should see a dialogue box like the one below. You will want to record the serial number shown, as you will need it in a couple minutes.



After recording the serial number you can click OK. Next, select the Modem Configuration tab at the top of the window. Once here, select the read button. This will bring up the current configuration for the connected XBee and will be similar to the following:



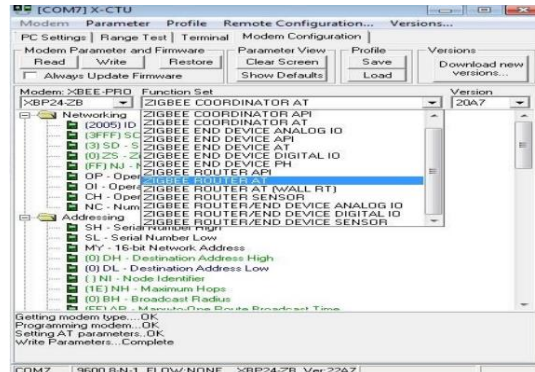
Once here, you want to select Zigbee Coordinator AT in the function set drop down menu.



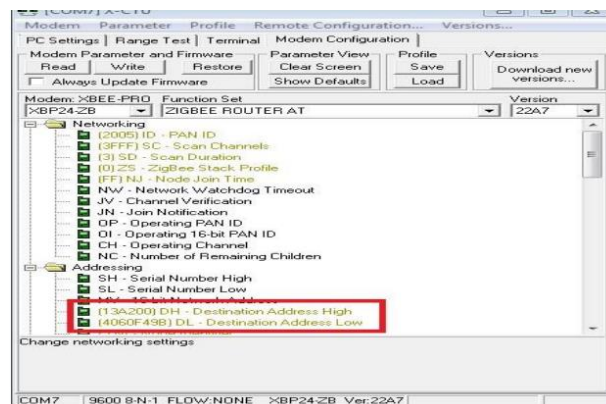
After selecting the coordinator function set, you will need to set the PAN ID. This can be any four-digit number and allows the XBees to distinguish between modules in their network and those from other networks. Once you have added the wanted PAN ID click the “Write” button. This will update and configure the XBee. Once this is done you can disconnect the XBee and plug in the second XBee.

Step 4: Configure 2nd XBee as Router

To configure the 2nd XBee, you will follow the same process as for the coordinator with one difference. In the PC settings tab, again, click the “Test/Query” button and record the serial number. Then in the Modem configuration tab, click the read button to load the current configuration of the XBee and set the PAN ID to the same ID used for the coordinator. The only change will be the function set you choose. For the second XBee we will set this as Zigbee Router AT.



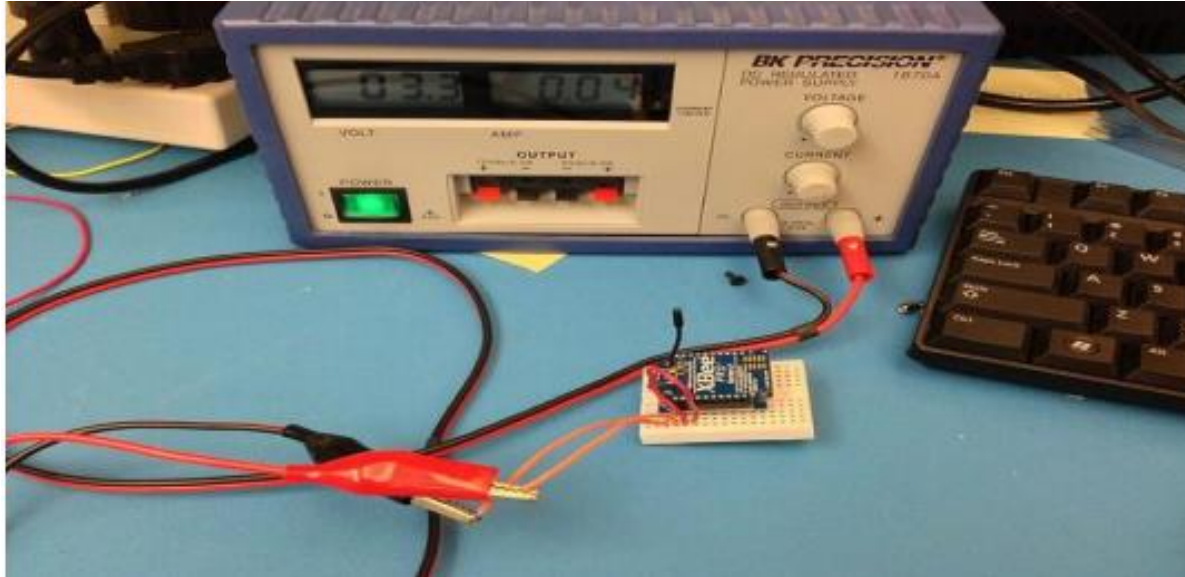
After setting the PAN ID and the function set for the router you will put the recorded serial numbers to use. Enter the first 6 digits of the coordinators serial number into the Destination Address High field and the rest of the serial number into the Destination Address Low field. Then select the “Write” button to update the configuration settings for the router. Once this is done updating you can disconnect the router and reconnect the coordinator to the computer.



After re-connecting the coordinator to the computer. You will again go into the Modem Configuration tab and click the read button. You will then want to set the Destination Address to the serial number of the Router XBee in the same manner as you just did for the Router. Once done you will again click the write button to update the coordinator XBee settings. After the write process is complete you are ready to use your XBees and communicate wirelessly

Step 5: Test the configuration

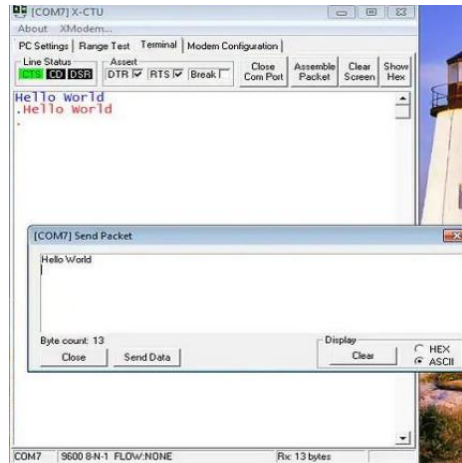
A simple test can be done to be sure the two XBees are communicating properly. You can connect either one of the XBees to the computer. Then, connect the second XBee to 3.3V power and connect the Dout and Din pins together. This will cause the XBee to automatically retransmit any data it receives.



When you have both XBees connected go to the Terminal tab in the X-CTU window. Whatever you type in the terminal window will appear in blue font and whatever is received will appear in red font. If the XBees are configured correctly every character you type should be mirrored in red.



When typing single characters, you should see a screen similar to the one above. To send strings of data you can assemble a data packet. To do this click the “Assemble Packet” button and type the wanted string into the box then click send data. This will send the entire packet before receiving the same packet back.



If everything you type is being reflected back in red, you are successfully transmitting and receiving with your XBees!! You are now ready to use them however you wish. You can connect them to any microcontroller and transmit data through the UART peripheral or you can connect each XBee to a different computer and have a chat application. There are many more possibilities that are now at your fingertips with your working XBees.

Code:

The code is quite straightforward. If you are using a board other than Arduino Uno, all digital pins may not support Software Serial.

On the transmitting side, the code will be –

```
#include<SoftwareSerial.h>
SoftwareSerial xbeeSerial(2,3); //RX, TX
void setup() {
    Serial.begin(9600);
    xbeeSerial.begin(9600);
}
void loop() {
    if(Serial.available() > 0){
        char input = Serial.read();
        xbeeSerial.print(input);
    }
}
```

As you can see, whatever the user on the Serial Monitor sends is sent to the XBee module, and it will be received on the receiving side. The code for the receiving side is –


```
#include<SoftwareSerial.h>
SoftwareSerial xbeeSerial(2,3); //RX, TX
void setup() {
    Serial.begin(9600);
    xbeeSerial.begin(9600);
}
void loop() {
    if(xbeeSerial.available() > 0){
        char input = xbeeSerial.read();
        Serial.print(input);
    }
}
```

Circuit Diagram:

