# PROGRAM 1

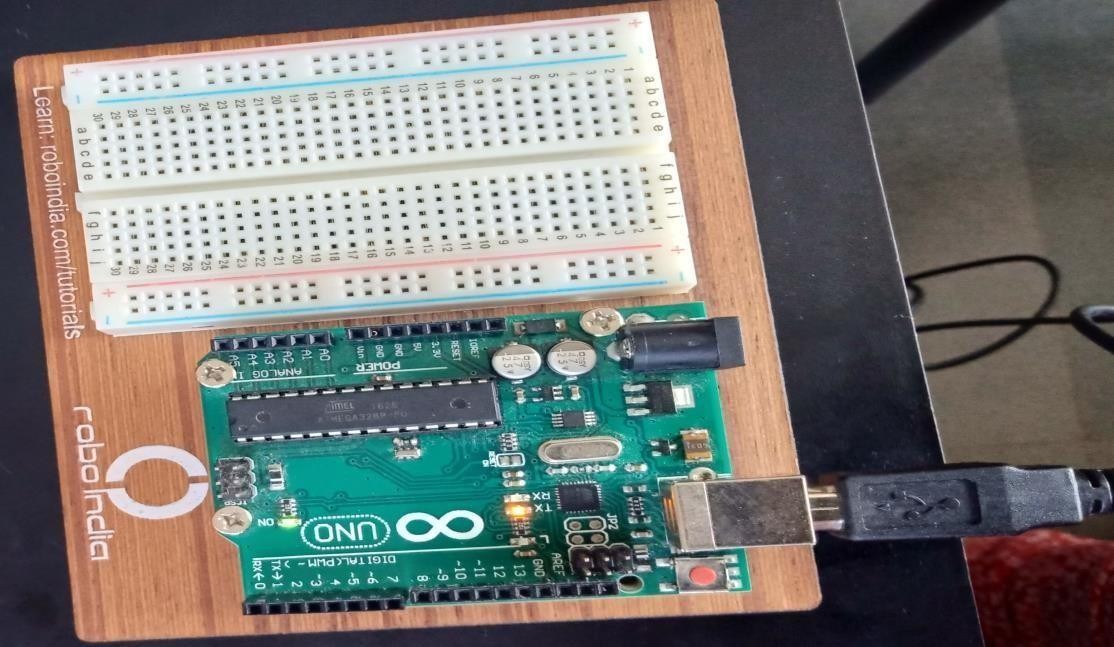
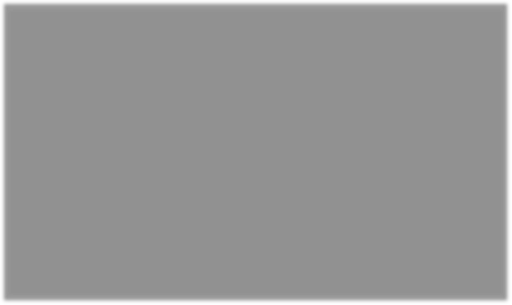
## TRANSMIT A STRING USING UART

**Components Required:** Arduino board (UNO), data cable, connecting wires.

### UART Communication

UART stands for Universal Asynchronous Receiver/Transmitter. It’s not a communication protocol like SPI and I2C, but a physical circuit in a microcontroller, or a stand-alone IC. A UART’s main purpose is to transmit and receive serial data. The UART that is going to transmit data receives the data from a data bus. The data bus is used to send data to the UART by another device like a CPU, memory, or microcontroller. Data is transferred from the data bus to the transmitting UART in parallel form. After the transmitting UART gets the parallel data from the data bus, it adds a start bit, a parity bit, and a stop bit, creating the data packet. Next, the data packet is output serially, bit by bit at the Tx pin. The receiving UART reads the data packet bit by bit at its Rx pin. The receiving UART then converts the data back into parallel form and removes the start bit, parity bit, and stop bits. Finally, the receiving UART transfers the data packet in parallel to the data bus on the receiving end.

### Circuit Connection



**Figure: Circuit Connection**

**Steps:**

1. Type the program in the Arduino IDE.
2. Connect the Arduino board with CPU using the USB cable
3. Save the program
4. Tools -&gt; Board -&gt; Select Arduino UNO
5. Tools -&gt; Port -&gt; Select port for Arduino UNO
6. Verify the program
7. Upload the program
8. Click on serial monitor (right hand side corner) for output.

### Code:

void setup() {

// put your setup code here, to run once Serial.begin(9600);

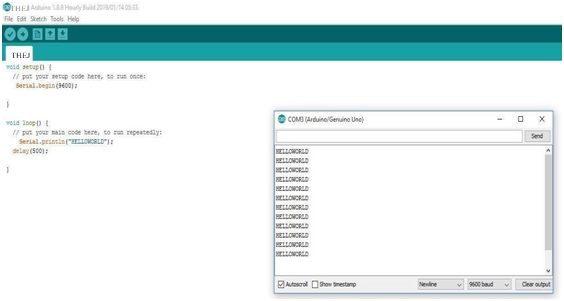
}

void loop() {

// put your main code here, to run repeatedly: Serial.println(&quot;HELLOWORLD&quot;); delay(500);

}

### Output Screenshot:



**PROGRAM 2**

**POINT-TO-POINT COMMUNICATION OF TWO MOTES OVER THE RADIO FREQUENCY**

**Components required:** 2 Arduino boards (UNO or MEGA), 2 data cables, 2 Bread boards, 2HC-12 modules, connecting wire.



### Figure: HC-12

HC-12 wireless serial port communication module is a new-generation multichannel embedded wireless data transmission module. Its wireless working frequency band is 433.4 - 473.0MHz, multiple channels can be set, with the stepping of 400 KHz, and there are totally 100 channels. The maximum transmitting power of module is 100mW (20dBm), the receiving sensitivity is -117dBm at baud rate of 5,000bps in the air, and the communication distance is 1,000m in open space. The module is encapsulated with stamp hole, can adopt patch welding, and its dimension is 27.8mm × 14.4mm × 4mm (including antenna cap, excluding spring antenna), so it is very convenient for customers to go into application system. There is a PCB antenna pedestal ANT1 on the module, and user can use external antenna of 433M frequency band through coaxial cable; there is also an antenna solder eye ANT2 in the module, and it is convenient for user to weld spring antenna. User could select one of these antennas according to use requirements.

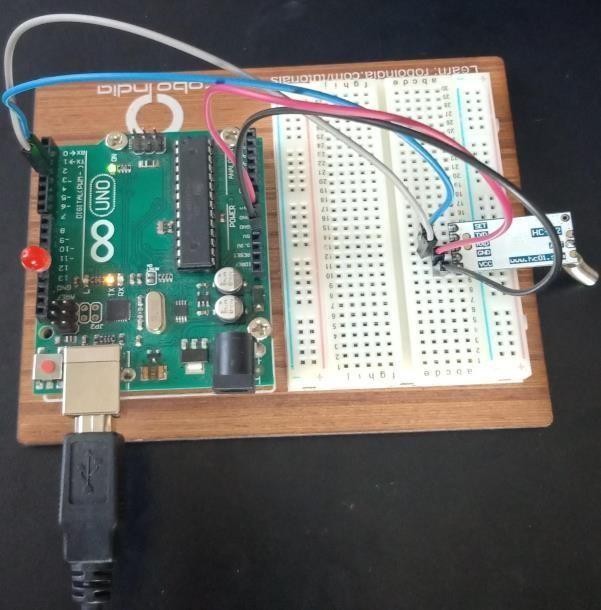
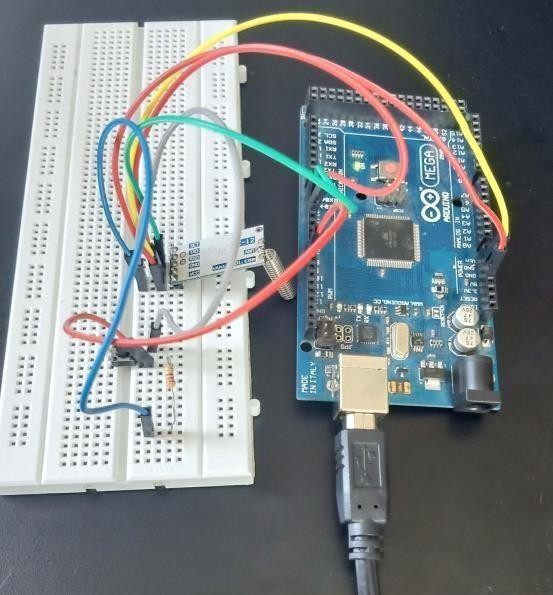
### Product Features

* 1. Long-distance wireless transmission (1,000m in open space/baud rate 5,000bps in the air).
  2. Working frequency range (433.4-473.0MHz, up to 100 communication channels).
  3. Maximum 100mW (20dBm) transmitting power (8 gears of power can be set).
  4. Three working modes, adapting to different application situations.
  5. Built-in MCU, performing communication with external device through serial port.
  6. The number of bytes transmitted unlimited to one time.
  7. Update software version through serial port.

### Product Application

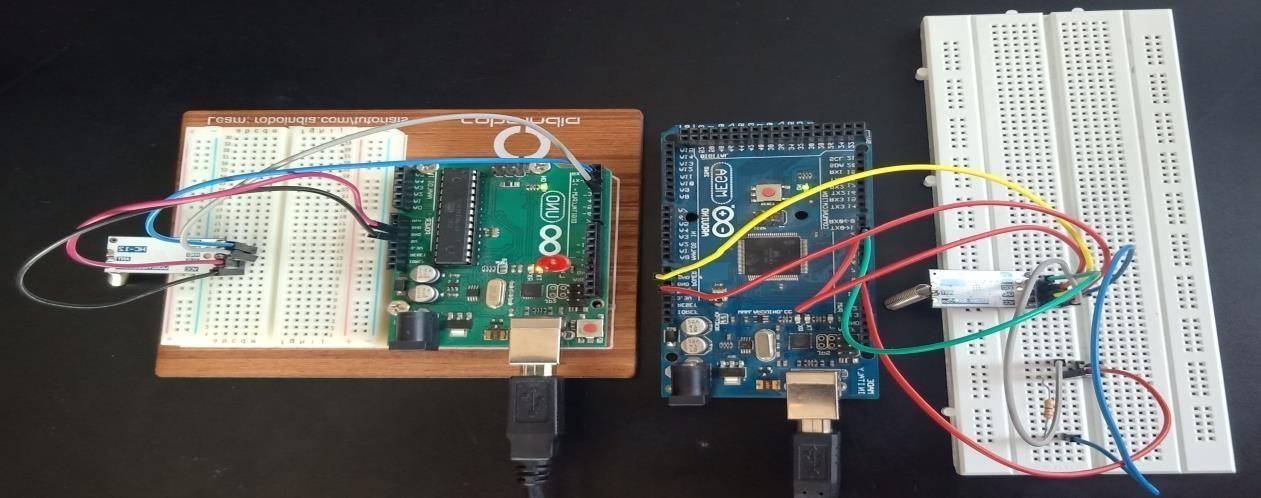
1. Wireless sensor
2. Community building security
3. Robot wireless control
4. Industrial remote control and telemetering
5. Automatic data acquisition
6. Container information management
7. POS system
8. Wireless acquisition of gas meter data
9. Vehicle keyless entry system
10. PC wireless networking

### Circuit Connection



**Sender**

**Receiver**



**Figure: Sender and Receiver**

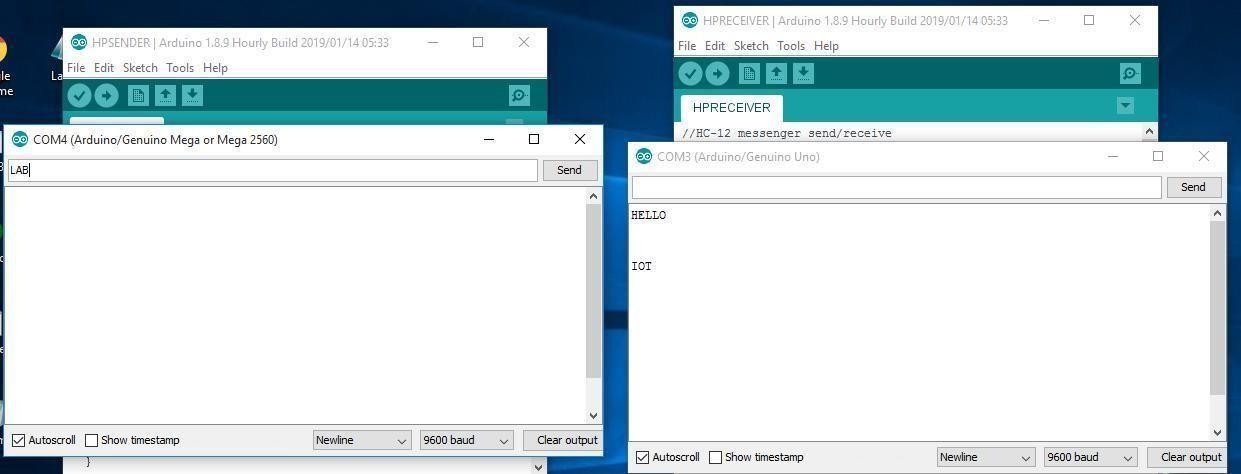
**Steps:**

1. Type the programs in the Arduino IDE for sender and receiver separately in two different tabs.
2. Sender side: ARDUINO MEGA or ARDUINO UNO.
   1. → TX
   2. → GND
3. Receiver side: ARDUINO MEGA or ARDUINO UNO.

Ardunio→ HC-12

1. → TX
2. → GND
3. Connect the Arduino boards with CPU using the USB cable (separately for sender board and receiver board).
4. Save the programs
5. Sender side: Tools → Board → Select Arduino MEGA.
6. Sender side: Tools → Port →Select port for Arduino MEGA.
7. Receiver side: Tools → Board → Select Arduino UNO.
8. Receiver side: Tools → Port → Select port for Arduino UN

**Output Screenshot:**

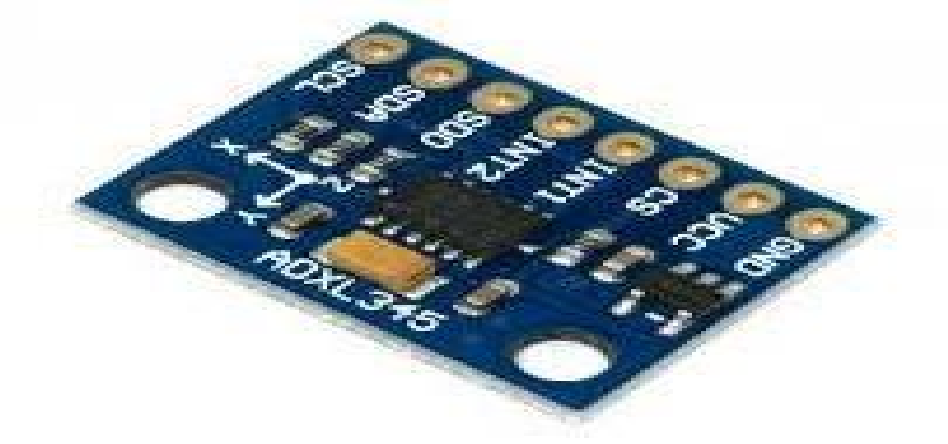
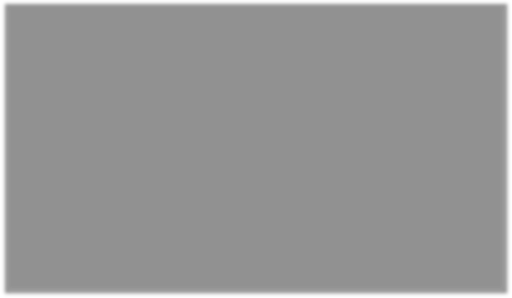


**PROGRAM 3 & 4**

**3 MULTI-POINT TO SINGLE POINT COMMUNICATION OF MOTES OVER THE RADIOFREQUENCY. LAN (SUB-NETTING)**

**4 I2C PROTOCOL STUDY**

**Components required:** Arduino board (UNO), data cable, Sensors GY-521, (GY-9250 GY-6500), I2C pressure sensor (0x3C), connecting wires.



### Figure: ADXL345

ADXL345 is a 3-axis accelerometer with high-resolution (13-bit) measurement at up to ±16 g. Digital output data is formatted as 16-bit twos complement and is accessible through either an SPI (3- or 4-wire) or I2C digital interface.

The ADXL345 is well suited for mobile device applications. It measures the static acceleration of gravity in tilt-sensing applications, as well as dynamic acceleration resulting from motion or shock. Its high resolution (4 mg/LSB) enables measurement of inclination changes less than 1.0°.

Several special sensing functions are provided. Activity and inactivity sensing detect the presence or lack of motion and if the acceleration on any axis exceeds a user-set level. Tap sensing detects single and double taps. Free-fall sensing detects if the device is falling. These functions can be mapped to one of two interrupt output pins.

An integrated, patent pending, 32-level first in, first out (FIFO) buffer can be used to store data to minimize host processor intervention.

Low power modes enable intelligent motion-based power management with threshold sensing and active acceleration measurement at extremely low power dissipation. The ADXL345 is supplied in a small, thin, 3 mm × 5 mm × 1 mm, 14-lead, plastic package.

### Features and Benefits

Ultra-low power: as low as 23 μA in measurement mode and 0.1 μA in standby mode at VS = 2.5 V (typical)

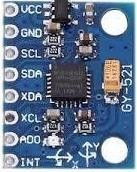
Power consumption scales automatically with bandwidth User-selectable resolution

Fixed 10-bit resolution - Full resolution, where resolution increases with g range, up to 13- bit resolution at ±16g (maintaining 4 mg/LSB scale factor in all g ranges).

### Applications

* Handsets
* Medical instrumentation
* Gaming and pointing devices.
* Industrial instrumentation Personal navigation devices
* Hard disk drive (HDD) protection

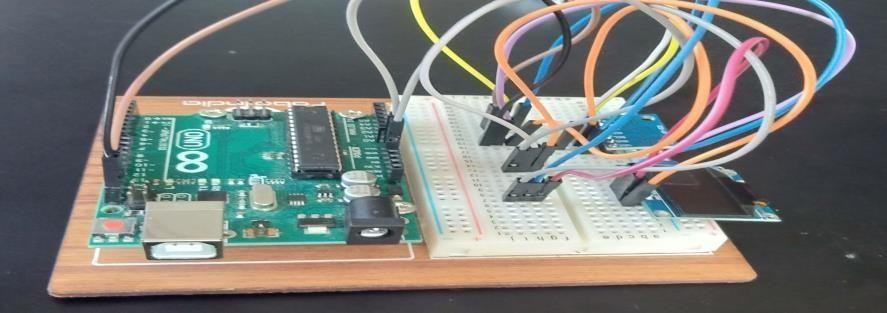
Traditionally the sensor of preference for stabilization is a gyroscope. Now-days gyroscopes are extremely small and very cheap to buy, so they are ideal for amateur electronics projects. Unfortunately, these gyroscopes (both the cheap and the not-so- cheap versions) also come with their own problems. They are good for short-term and quick movements, but tend to drift over time as the error accumulates. They also record a lot of jitter and noise, which needs to be filtered by the micro-controller before the data can be used. The MPU-6050 uses I2C to communicate with the micro-controller, so I started by connecting up the pins as shown in the schematics: the SDA line connects to the Analogue pin 4, the SCL to Analogue pin 5, power input to the 3.3v pin and the ground to the GND pin.



### GY 521-MPU-6050

The accelerometer measures the acceleration along one direction, while the gyroscope measures the angular acceleration on one axis. The analogic pins are not set on INPUT because it's their default setting. The values read by the analogic pins will be sent to the serial port. Open the Serial Monitor*,* move the sensor and try to see how the values change.

## Circuit Connection



### Figure: Circuit Diagram

**Steps:**

|  |  |
| --- | --- |
| 4. Bread board | Arduino board |
| VCC of two sensors to vertical ports in bread board | 5V |
| 2 GNDs to bread board | GND |
| 2 SCL of sensors to vertical point bread board | from bread board take wire and connect to  right top first port |
| 2 SDA to bread board | Right top second port (above pin 13 and  GND) |

1. Type the program in the Arduino IDE – scan and data files separately in two differentfiles.
2. Sensors must have clock (SCL), data (SDA), VCC and GND.
3. Connect the two sensor points vertically to single port in horizontal bread board andfrom that point make a single connection to the Arduino board.
4. Connect the Arduino board with CPU using the USB cable
5. Save the programs
6. Tools -> Board -> Select Arduino UNO (for both scan and data files)
7. Tools -> Port -> Select port for Arduino UNO (for both scan and data files)
8. Compile the program (for both scan and data files)
9. Upload the program (for both scan and data files
10. Click on serial monitor of scan file (right hand side corner) for output (output:no.of devices connected as well as addresses of those sensors), then close the serial monitor.
11. In the code →data file (take the address from Step 11 and change it in data file) #include<wire.h>

Int Adxl address = 0x68;→ sensor

address Int Adxl address = 0x3C; → sensor address Save the changes.

1. Click on serial monitor of data file (right hand side corner) for output.

### Code:

**Scan File:**

// This sketch tests the standard 7-bit addresses

// Devices with higher bit address might not be seen properly. #include <Wire.h>

void setup()

{

Wire.begin();

Serial.begin(9600);

while (!Serial); // Leonardo: wait for serial monitor Serial.println("\nI2C Scanner");}

void loop(){ byte error, address;int nDevices;

Serial.println("Scanning...)

;nDevices = 0;

for(address = 1; address < 127; address++ )

{

// The i2c\_scanner uses the return value of

// the Write.endTransmisstion to see if

// a device did acknowledge to the address. Wire.beginTransmission(address);

error Wire.endTransmission();

if (error == 0){

Serial.print("I2C device found at address 0x"); if (address<16)

Serial.print("0");

Serial.print(address,HEX);Serial.println(" !"); nDevices++;

}

else if (error==4)

{

Serial.print("Unknown error at address 0x"); if (address<16)

Serial.print("0"); Serial.println(address,HEX);

}

}

if (nDevices == 0)

Serial.println("No I2C devices found\n");else Serial.println("done\n");

delay(5000); // wait 5 seconds for next scan

}

### Data File:

#include <Wire.h>

int ADXLAddress = 0x3C; // Device address in which is also included the 8th bit for selecting the mode, read in this case.

//#define X\_Axis\_Register\_DATAX0 0x0D // Hexadecimal address for the DATAX0 internal register.

//#define X\_Axis\_Register\_DATAX1 0x0E // Hexadecimal address for theDATAX1 internal register.

#define X\_Axis\_Register\_DATAX0 0x0D // Hexadecimal address for the DATAX0 internal register.

#define Y\_Axis\_Register\_DATAX1 0x0E

#define Z\_Axis\_Register\_DATAX2 0x0F // Hexadecimal address for the DATAX0 internal register.

//#define X\_Axis\_Register\_DATAX1 0x0E // Hexadecimal address for the DATAX1internal register.// Hexadecimal address for the DATAX1 internal register.

#define Power Register 0x2D // Power Control Register int X0,X1,X2,X\_out;

void setup()

{

Wire.begin(); // Initiate the Wire librarySerial.begin(9600); delay(5000);

// Enable measurement Wire.beginTransmission(ADXLAd dress);Wire.write(Power\_Register);

// Bit D3 High for measuring enable (0000 1000) Wire.write(8);

Wire.endTransmission();} void loop() {

Wire.beginTransmission(ADXLAddress); // Begin transmission to the Sensor

//Ask the particular registers for data Wire.write(X\_Axis\_Register\_DATAX0); Wire.write(Y\_Axis\_Register\_DATAX1); Wire.write(Z\_Axis\_Register\_DATAX2);

Wire.endTransmission(); // Ends the transmission and transmits the data from the two

//registers

Wire.requestFrom(ADXLAddress,3); // Request the transmitted two bytes from the two

//registers if(Wire.available<=3{

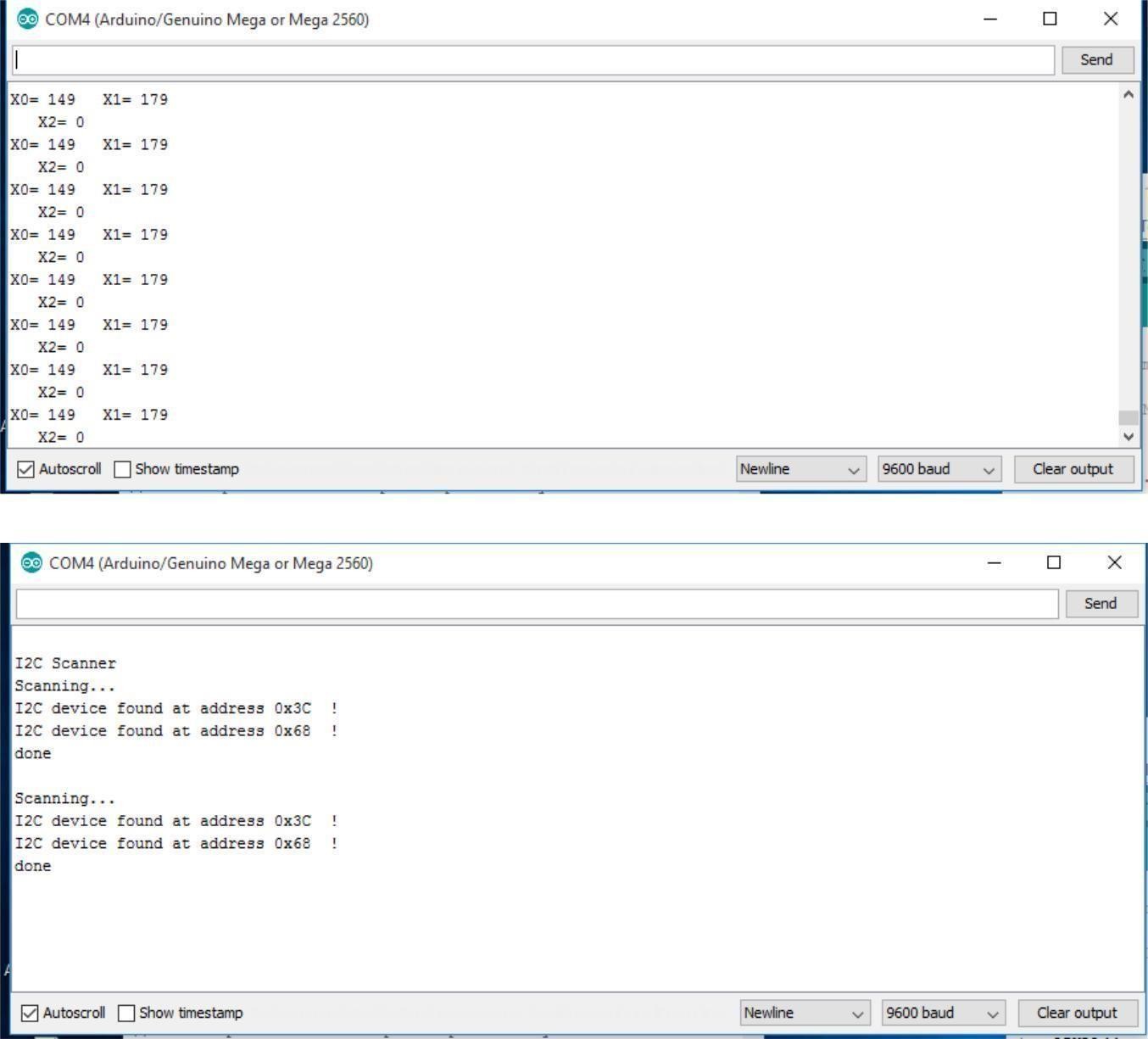
X0 = Wire.read(); // Reads the data from the registerX1 = Wire.read();

X2 = Wire.read();}

Serial.print("X0"); Serial.print(X0); Serial.print(" X1="); Serial.println(X1); Serial.print(" X2="); Serial.println(X2);

### }

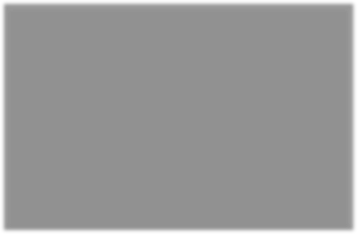
### Output Screenshot:



**PROGRAM 5**

**READING TEMPERATURE AND RELATIVE HUMIDITY VALUE FROM THE SENSOR**

**Components required:** Arduino board (UNO), data cable, DHT11 sensor



### DHT11 sensor

This DHT11 Temperature and Humidity Sensor features a calibrated digital signal output with the temperature and humidity sensor capability. It is integrated with a high- performance 8-bit microcontroller. Its technology ensures the high reliability and excellent long-term stability.This sensor includes a resistive element and a sensor for wet NTC temperature measuring devices. It has excellent quality, fast response, anti-interference ability and high performance. Each DHT11 sensors features extremely accurate calibration of humidity calibration chamber The calibration coefficients stored in the OTP program memory, internal sensors detect signals in the process we should call these calibration coefficients. The single-wire serial interface system is integrated to become quick and easy. Small size, low power, signal transmission distance up to 20 meters enabling a variety of applications and even the most demanding ones.The product is 4-pin single row pin package. Convenient connection, special packages can be provided according to users need.

### Specification

1. Supply Voltage: +5 V
2. Temperature range :0-50 °C error of ± 2 °C
3. Humidity :20-90% RH ± 5% RH error
4. Interface: Digital

### IMG20190117132804Circuit Connection

**Figure: DHT 11 Sensor Circuit**

**Steps:**

* 1. Type the program in the Arduino IDE
  2. Make the following connections:Sensor Board

DATA→7 GND→GND VCC→5V

* 1. Connect the Arduino board with CPU using the USB cable
  2. Save the program
  3. Tools -> Board -> Select Arduino UNO
  4. Tools -> Port -> Select port for Arduino UNO
  5. Compile the program
  6. Upload the program
  7. Click on serial monitor (right hand side corner) for output.
  8. To include Header files/Libraries:

Download DHT library -> In IDE, Sketch -> Include library -> Add zip library -> Search the folders (Downloads) -> DHT library -> open

### Code:

#include <dht.h> dht DHT;

#define DHT11\_PIN

7void setup(){ Serial.begin(9600);

}

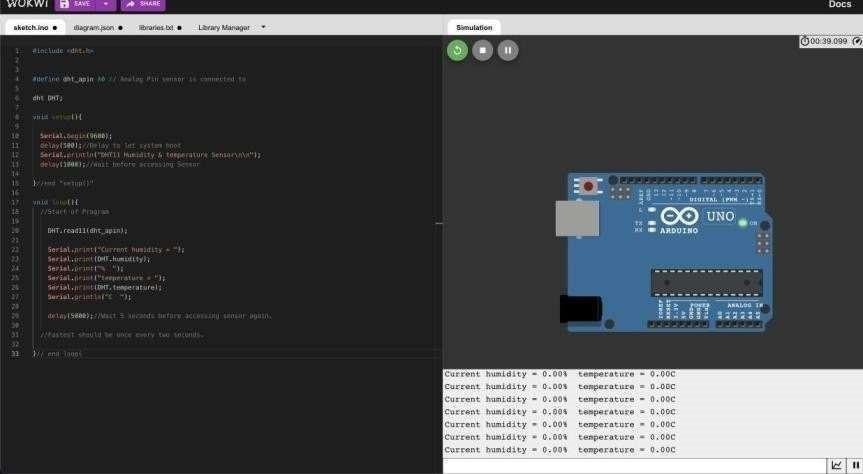
void loop()

{

int chk = DHT.read11(DHT11\_PIN); Serial.print("Temperature = "); Serial.println(DHT.temperature); Serial.print("Humidity = "); Serial.println(DHT.humidity); delay(1000);

}

### Output Screenshot:



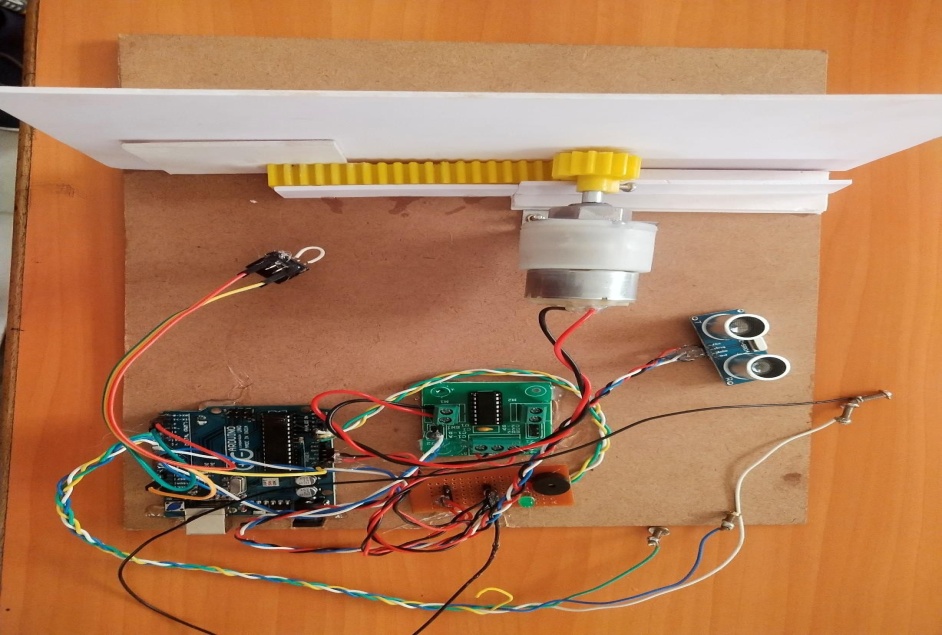
**PROGRAM 6**

# “EARLY FLOOD MONITORING SYSTEM USING IOT”

**Components required:** Arduino Uno, Ultrasonic sensor, Water level sensor, Power supply, Wi-Fi module ,Dc motor, LED, Buzzer

* This project consists of flood detection and avoidance system using the IOT technology.
* The sensors present in this are used to estimate the water levels and send the real-time data via the mobile app.
* This model is widely used to alarm the people before a flood occurs and necessary precautions could be taken and to open the dam gate manually for the flow of water.
* This project will plan a flood checking framework for flood region that will utilize nearby detecting information by means of microcontroller framework for observing and the framework can likewise qualify different flood boundaries, for example water level.
* Whenever the water level rise above the threshold level, the dam gets manually opens by the respective authorities.
* In this project we use IOT based web server blink application to provide real time data for the locality and authorities. This project also send a email notification to the android.
* So, the respective can be educated about the flood and prompt precautionary measures can be taken.

**Circuit connection:**

**Figure : Circuit Connection**

* Overall, system able to send an alert via Blynk application as well as IOT enabled when the water raises to the predetermined level.
* A quick notification through Blynk application as well as buzzer triggers and led glows is vital since the system is aimed at alerting people and audience.
* Total four nodes are deployed at the dam site each prototype will communicate individually with IOT enabled phone through web server.
* The data will be stored in Arduino UNO as per sequence given by water level sensor simultaneously.
* The system provides a real world application of internet of things and offers services like accurate level monitoring directly or indirectly benefitted by the system sensors are important elements in the flood observatory system.

### Code:

#include <SerialESP8266wifi.h>

#include <Blynk.h>

#define BLYNK\_PRINT Serial

#include <ESP8266\_Lib.h>

#include <BlynkSimpleShieldEsp8266.h>

char auth[] = "iCuO5mzjGVyK3ZMFP4W92A5UXbw8YRE3";

char ssid[] = "Guru";

char pass[] = "8971160715";

ESP8266 wifi(&Serial);

#define level1 2

#define level2 3

#define level3 4

#define level4 5

#define trig 6

#define echo 7

int distance;

int cm;

#define red 14

#define green 15

#define buzzer 16

#define in1 8

#define in2 9

int flag=0;

void setup()

{

Serial.begin(115200);

while (!Serial);

Blynk.begin(auth, wifi, ssid, pass);

pinMode(red, OUTPUT);

pinMode(green, OUTPUT);

pinMode(buzzer, OUTPUT);

pinMode(in1, OUTPUT);

pinMode(in2, OUTPUT);

pinMode(trig, OUTPUT);

pinMode(echo, INPUT);

pinMode(level1, INPUT\_PULLUP);

pinMode(level2, INPUT\_PULLUP);

pinMode(level3, INPUT\_PULLUP);

pinMode(level4, INPUT\_PULLUP);

}

void loop()

{

Blynk.run();

ultrasonic();

if((digitalRead(level1) == 1) && (digitalRead(level2) == 1) && (digitalRead(level3) == 1) && (digitalRead(level4) == 0)) //25%

{

Blynk.notify("Water level is 25%");

digitalWrite(green, HIGH);

delay(1000);

}

if((digitalRead(level1) == 1) && (digitalRead(level2) == 1) && (digitalRead(level3) == 0) && (digitalRead(level4) == 0)) //50%

{

Blynk.notify("Water level is 50%");

}

if((digitalRead(level1) == 1) && (digitalRead(level2) == 0) && (digitalRead(level3) == 0) && (digitalRead(level4) == 0)) //75%%

{

Blynk.email("Emergency","More water stored in dam !!!!.... Reached threshold level");

Blynk.notify("Water level is above 75%");

digitalWrite(red, HIGH);

delay(100);

for(int i=0;i<5;i++)

{

digitalWrite(buzzer, HIGH);

delay(1000);

digitalWrite(buzzer, LOW);

delay(100);

}

}

else

{

digitalWrite(red, LOW);

digitalWrite(green, LOW);

}

}

BLYNK\_WRITE(V1)

{

int buttonState=param.asInt();

if((buttonState==1) && (flag==0))

{

digitalWrite(in1,LOW);

digitalWrite(in2,HIGH);

delay(5000);

digitalWrite(in1,HIGH);

digitalWrite(in2,HIGH);

flag=1;

}

if((buttonState==0) && (flag==1))

{

digitalWrite(in2,LOW);

digitalWrite(in1,HIGH);

delay(5000);

digitalWrite(in1,HIGH);

digitalWrite(in2,HIGH);

flag=0;

}

else

{

digitalWrite(in1,HIGH);

digitalWrite(in2,HIGH);

}

}

void ultrasonic()

{

digitalWrite(trig, LOW);

delay(2);

digitalWrite(trig, HIGH);

delay(10);

digitalWrite(trig, LOW);

distance = pulseIn(echo, HIGH);

cm = distance\*0.034/2;

Serial.print(cm);

delay(1000);

if(cm < 20)

{

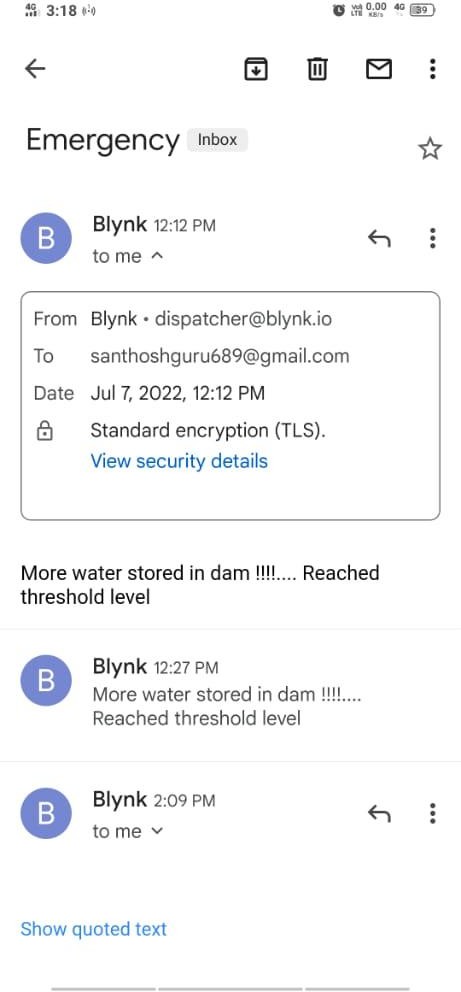
Blynk.notify("Flood");

}

### OUTPUT SCREENSHOT:

### 

###### **Fig 5.2 SMS alert and web-based interface**



###### **Fig 5.2 SMS alert and web-based interface**