Channabasaveshwara Institute of Technology
(Affiliated to VTU, Belgaum & Approved by AICTE, New Delhi)
(NAAC Accredited & ISO 9001:2015 Certified Institution)
NH 206 (B.H. Road), Gubbi, Tumkur – 572 216. Karnataka.

IOT LABORATORY MANUAL- [22SCSL17]

Department of Computer Science & Engineering

MTech - I Semester

Academic Year: 2022-23

Prepared by,

Mr. Dharaneshkumar M L
Asst. Professor,
Dept of CSE,
CIT, Gubbi

Internet of Things Laboratory

Course Code	**22SCSL17**	CIE Marks	50
Teaching Hours/Week (L:T:P: S)	1:2:0	SEE Marks	50
Credits	02	Exam Hours	03

Course objectives:

- Describe what IoT is and how it works today
- Design and program IoT devices
- Use real IoT protocols for communication

	Experiments
1	Transmit a string using UART
2	Point-to-Point communication of two Motes over the radio frequency
3	Multi-point to single point communication of Motes over the radio frequency. AN (Subnetting).
4	I2C protocol study
5	Reading Temperature and Relative Humidity value from the sensor
6	Study of Connectivity and Configuration of Raspberry-Pi/ Beagle Board circuit with basic peripherals, LEDs, Understanding GPIO and its use in program.
7	Study of different operating systems for Raspberry Pi / Beagle board. Understanding the process of OS installation on Raspberry – Pi/ Beagle board.
8	Familiarization with the concept of IOT, Arduino / Raspberry Pi and perform necessary software installation.

Course outcomes (Course Skill Set):

At the end of the course the student will be able to:

- Apply key Internet applications and their protocols, and ability to develop their own applications (e.g. Client Server applications, Web Services) using the sockets API.
- Design and evaluate application layer protocol
- Analyze the vulnerabilities in any computing system and hence be able to design a security solution.
- Identify the security issues in the network and resolve it.
- Evaluate security mechanisms using rigorous approaches, including theoretical.

Assessment Details (both CIE and SEE)

The weightage of Continuous Internal Evaluation (CIE) is 50% and for Semester End Exam (SEE) is 50%. The minimum passing mark for the CIE is 50% of the maximum marks. A student shall be deemed to have satisfied the academic requirements and earned the credits allotted to each course. The student has to secure not less than 40%of maximum marks in the semester-end examination (SEE). In total of CIE and SEE student has to secure 50% maximum marks of the course.

Continuous Internal Evaluation (CIE):

CIE marks for the practical course is **50 Marks**.

The split-up of CIE marks for record/ journal and test are in the ratio **60:40**.

- Each experiment to be evaluated for conduction with observation sheet and record write- up. Rubrics for the evaluation of the journal/write-up for hardware/software experiments designed by the faculty who is handling the laboratory session and is made known to students at the beginning of the practical session.
- Record should contain all the specified experiments in the syllabus and each experiment write-up will be evaluated for 10 marks.
- Total marks scored by the students are scaled downed to 30 marks (60% of maximum marks).
- Weightage to be given for neatness and submission of record/write-up on time.
- Department shall conduct 02 tests for 100 marks, the first test shall be conducted after the 8th week of the semester and the second test shall be conducted after the 14th week of the semester.
- In each test, test write-up, conduction of experiment, acceptable result, and procedural knowledge will carry a weightage of 60% and the rest 40% for viva-voce.
- The suitable rubrics can be designed to evaluate each student's performance and learning ability.
- The average of 02 tests is scaled down to **20 marks** (40% of the maximum marks).

The Sum of **scaled-down** marks scored in the report write-up/journal and average marks of two tests is the total CIE marks scored by the student.

Semester End Evaluation (SEE):

SEE marks for the practical course is 50 Marks.

SEE shall be conducted jointly by the two examiners of the same institute, examiners are appointed by the University.

All laboratory experiments are to be included for practical examination.

(Rubrics) Breakup of marks and the instructions printed on the cover page of the answer script to be strictly adhered to by the examiners. **OR** based on the course requirement evaluation rubrics shall be decided jointly by examiners.

Students can pick one question (experiment) from the questions lot prepared by the internal /External examiners jointly.

Evaluation of test write-up/ conduction procedure and result/viva will be conducted jointly by examiners.

General rubrics suggested for SEE are mentioned here, writeup-20%, Conduction procedure and result in -60%, Viva-voce 20% of maximum marks. SEE for practical shall be evaluated for 100 marks and scored marks shall be scaled down to 50 marks (however, based on course type, rubrics shall be decided by the examiners)

Change of experiment is allowed only once and 10% Marks allotted to the procedure part to be made zero.

The duration of SEE is 03 hours

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.

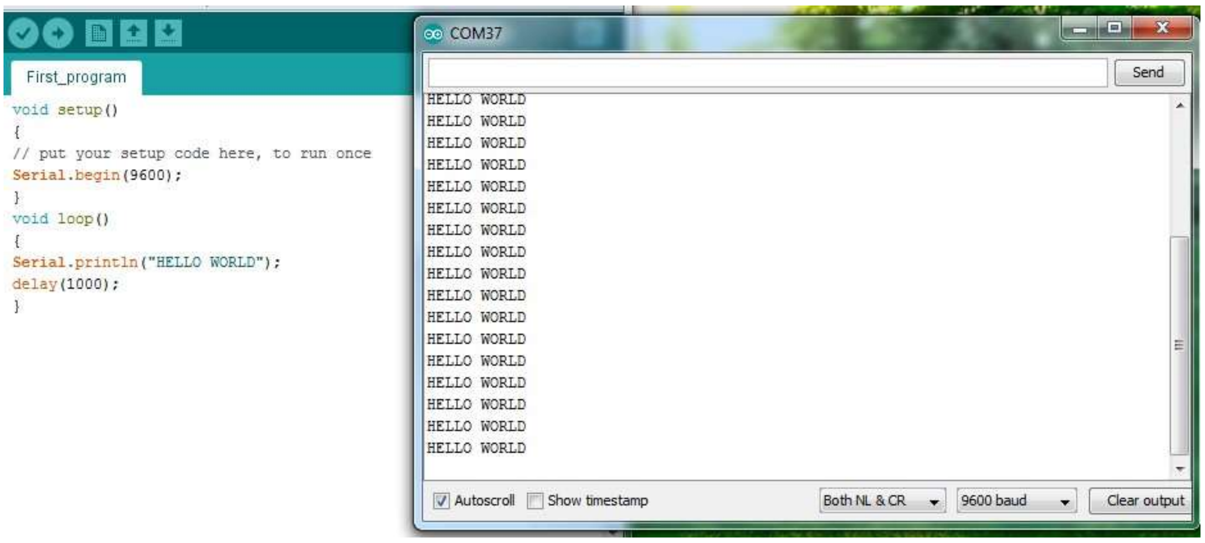
Steps:

1. Type the program in the Arduino IDE.
2. Connect the Arduino board with CPU using the cable
3. Save the program
4. In Arduino IDE, Tools -> Board -> Select Arduino UNO
5. In Arduino IDE, Tools -> Port -> 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()
{
Serial.begin(9600);
}
void loop()
{
Serial.println("HELLO WORLD");
delay(1000);
}
```

Output:



```
void setup()
{
// put your setup code here, to run once
Serial.begin(9600);
}
void loop()
{
Serial.println("HELLO WORLD");
delay(1000);
}
```

```
HELLO WORLD
HELLO WORLD
HELLO WORLD
HELLO WORLD
HELLO WORLD
HELLO WORLD
HELLO WORLD
HELLO WORLD
HELLO WORLD
HELLO WORLD
HELLO WORLD
HELLO WORLD
HELLO WORLD
HELLO WORLD
HELLO WORLD
HELLO WORLD
HELLO WORLD
```

String displayed in serial monitor

PROGRAM 2 - POINT-TO-POINT COMMUNICATION OF TWO MOTES OVER THE RADIO FREQUENCY.

Components required: Two Arduino boards (UNO), data cables, connecting wires, two NRF24L01 modules, one LED.

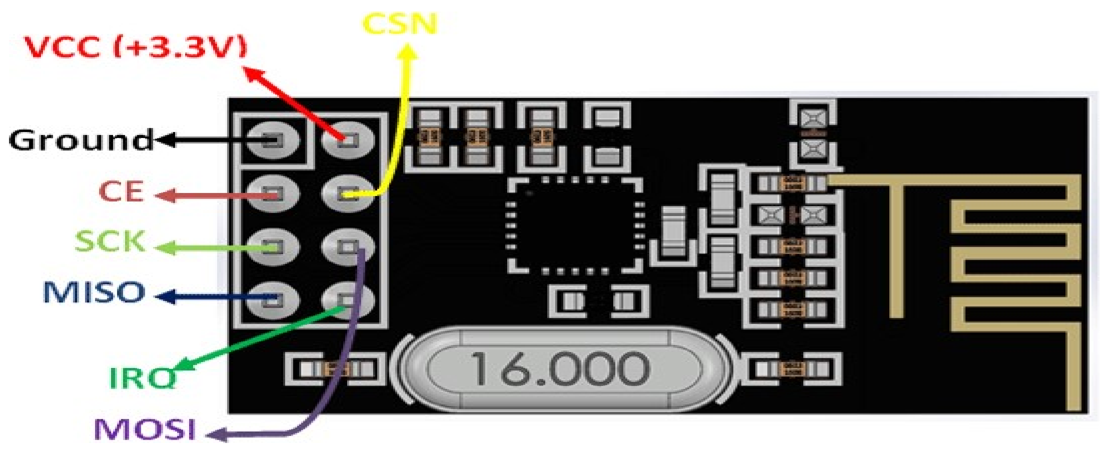
NRF24L01 Transceiver Module (Radio Frequency)

It uses the 2.4 GHz band and it can operate with baud rates from 250 kbps up to 2 Mbps. If used in open space and with lower baud rate its range can reach up to 100 meters.

The module can use 125 different channels which give a possibility to have a network of 125 independently working modems in one place. Each channel can have up to 6 addresses, or each unit can communicate with up to 6 other units at the same time.

The power consumption of this module is just around 12mA during transmission, which is even lower than a single LED. The operating voltage of the module is from 1.9 to 3.6V, but the good thing is that the other pins tolerate 5V logic, so we can easily connect it to an Arduino without using any logic level converters.

Three of these pins are for the SPI communication and they need to be connected to the SPI pins of the Arduino, but note that each Arduino board have different SPI pins. The pins CSN and CE can be connected to any digital pin of the Arduino board and they are used for setting the module in standby or active mode, as well as for switching between transmit or command mode. The last pin is an interrupt pin which doesn't have to be used.



NRF24L01 PINOUT

Arduino Uno and NRF24L01 connection (same connection for transmitter and receiver)

Arduino	**NRF24L01**
VCC	3.3V
GND	GND
Pin 8	CSN
Pin 7	CE
Pin 13	SCK
Pin 11	MOSI
Pin 12	MISO

LED connection: positive (long wire of LED) to pin 2 of Arduino Uno and negative toGND

Steps:

1. Import RF24Network-master.zip to Arduino IDE via Sketch->Include Library-> ADD .Zip library
2. Import RF24-master.zip to Arduino IDE via Sketch->Include Library-> ADD .Zip library
3. Type the transmitter code in Arduino IDE and select board and port for first Arduino uno. Upload the code to transmitter Arduino (first Arduino).
4. Type the receiver code in another Arduino IDE and select board and port for second Arduino uno. Upload the code to receiver Arduino (second Arduino).
5. If data sent in the code is ‘1’ then LED in receiver will be ON otherwise LED will be OFF.

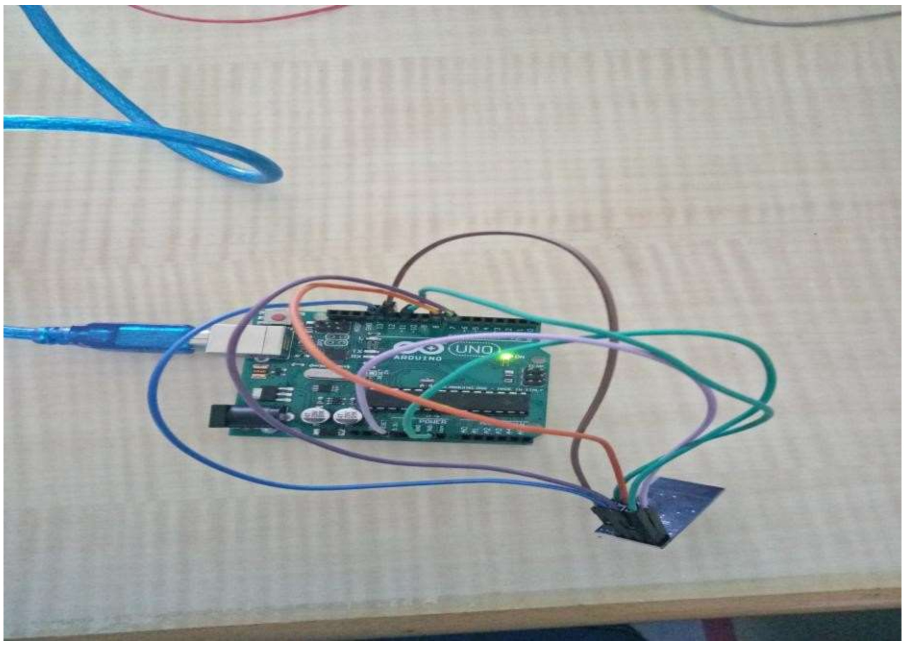
Transmitter code

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
RF24 radio(7, 8); // CE, CSN
const byte address[6] = "00001";
void setup() {
  radio.begin();
  radio.openWritingPipe(address);
  radio.setPALevel(RF24_PA_MIN);
  radio.stopListening();
}
void loop() {
  int text= 1;
  radio.write(&text, sizeof(text));
  delay(1000);
}
```

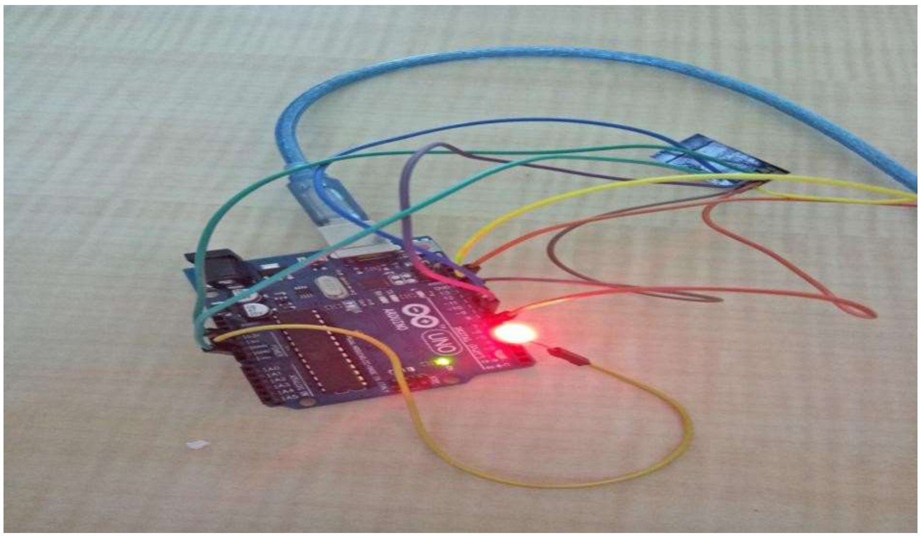
Receiver Code

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
RF24 radio(7, 8); // CE, CSN
const byte address[6] = "00001";
void setup() {
Serial.begin(9600);
  pinMode(2, OUTPUT);
  radio.begin();
  radio.openReadingPipe(0, address);
  radio.setPALevel(RF24_PA_MIN);
  radio.startListening();
}
void loop() {
  if (radio.available()) {
    int text,t;
    radio.read(&text, sizeof(text));
    delay(1000);
    Serial.println(text);
    if(text==1)
    {
      digitalWrite(2, HIGH);
    }
    else
    {
      digitalWrite(2, LOW);
     }
    }
 }
```

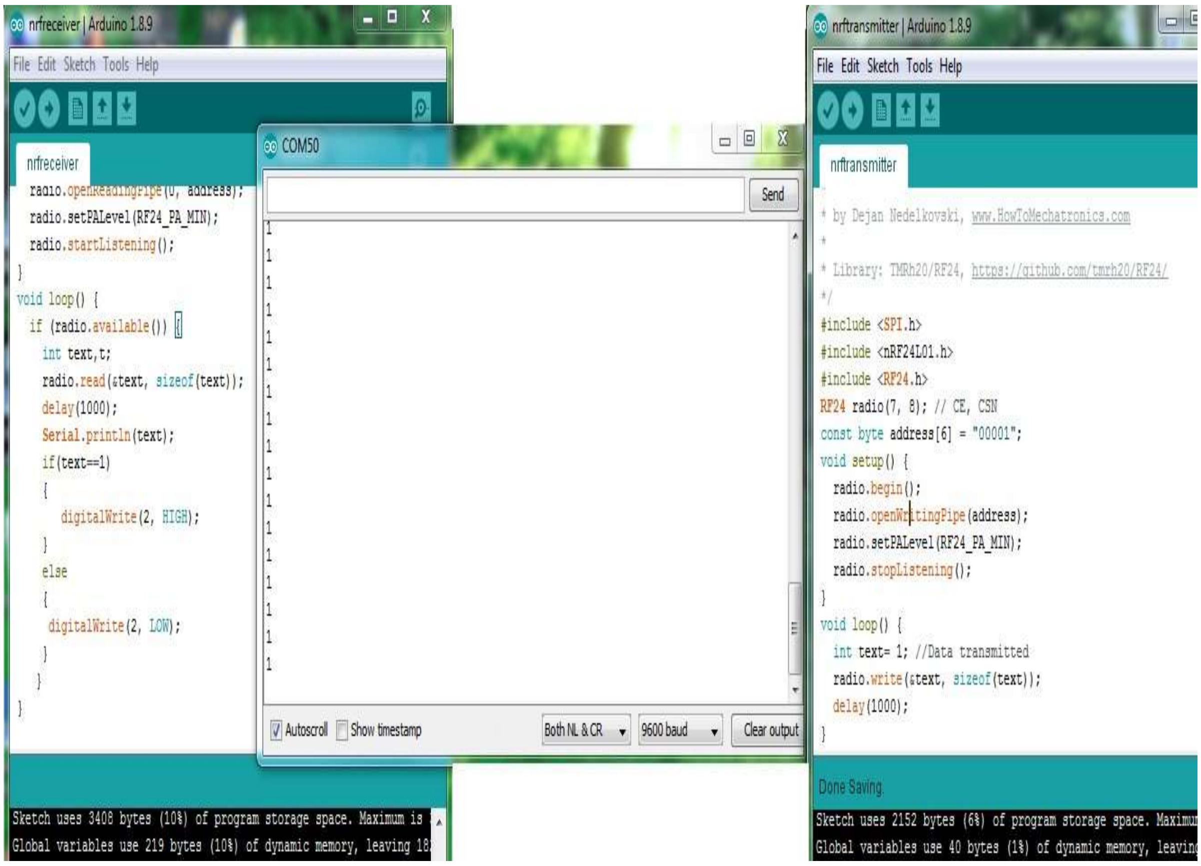
Output



Transmitter



Receiver with data 1 (LED ON)


nrfreceiver | Arduino 1.8.9
File Edit Sketch Tools Help
nrfreceiver
COM50
Send
Autoscroll
Show timestamp
Both NL & CR
9600 baud
Clear output
nrftransmitter | Arduino 1.8.9
File Edit Sketch Tools Help
nrftransmitter
Done Saving

```
  radio.openReadingPipe(0, address);
  radio.setPALevel(RF24_PA_MIN);
  radio.startListening();
}
void loop() {
  if (radio.available()) {
    int text,t;
    radio.read(&text, sizeof(text));
    delay(1000);
    Serial.println(text);
    if(text==1)
    {
       digitalWrite(2, HIGH);
    }
    else
    {
     digitalWrite(2, LOW);
    }
   }
}
```

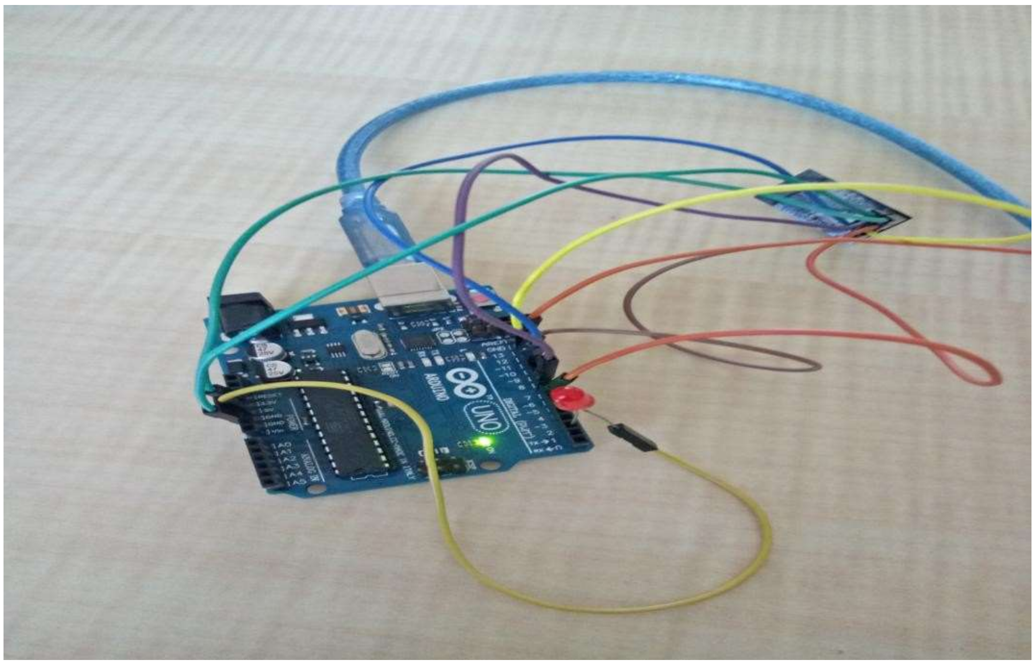
```
Sketch uses 3408 bytes (10%) of program storage space. Maximum is
Global variables use 219 bytes (10%) of dynamic memory, leaving 18
```

```
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
```

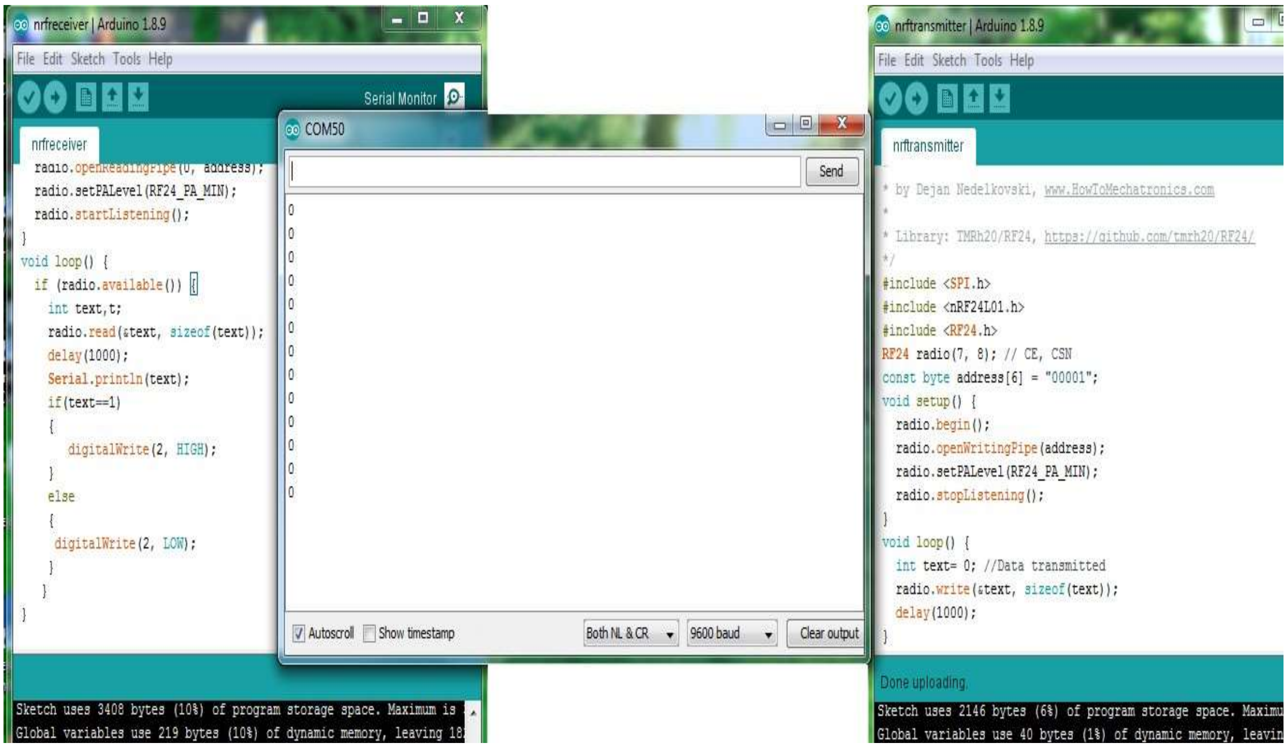
```
* by Dejan Nedelkovski, www.HowToMechatronics.com
*
* Library: TMRh20/RF24, https://github.com/tmrh20/RF24/
*/
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
RF24 radio(7, 8); // CE, CSN
const byte address[6] = "00001";
void setup() {
  radio.begin();
  radio.openWritingPipe(address);
  radio.setPALevel(RF24_PA_MIN);
  radio.stopListening();
}
void loop() {
  int text= 1; //Data transmitted
  radio.write(&text, sizeof(text));
  delay(1000);
}
```

```
Sketch uses 2152 bytes (6%) of program storage space. Maximum
Global variables use 40 bytes (1%) of dynamic memory, leaving
```

Receiver with data 1 in serial monitor



Receiver with data 0 (LED OFF)



```
  radio.openReadingPipe(0, address);
  radio.setPALevel(RF24_PA_MIN);
  radio.startListening();
}
void loop() {
  if (radio.available()) {
    int text,t;
    radio.read(&text, sizeof(text));
    delay(1000);
    Serial.println(text);
    if(text==1)
    {
       digitalWrite(2, HIGH);
    }
    else
    {
     digitalWrite(2, LOW);
    }
   }
}
```

```
Sketch uses 3408 bytes (10%) of program storage space. Maximum is
Global variables use 219 bytes (10%) of dynamic memory, leaving 18
```

```
0
0
0
0
0
0
0
0
0
0
0
0
0
```

```
* by Dejan Nedelkovski, www.HowToMechatronics.com
*
* Library: TMRh20/RF24, https://github.com/tmrh20/RF24/
*/
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
RF24 radio(7, 8); // CE, CSN
const byte address[6] = "00001";
void setup() {
  radio.begin();
  radio.openWritingPipe(address);
  radio.setPALevel(RF24_PA_MIN);
  radio.stopListening();
}
void loop() {
  int text= 0; //Data transmitted
  radio.write(&text, sizeof(text));
  delay(1000);
}
```

```
Sketch uses 2146 bytes (6%) of program storage space. Maximu
Global variables use 40 bytes (1%) of dynamic memory, leavin
```

Receiver with data 0 in serial monitor

PROGRAM 3 - MULTI-TO-POINT COMMUNICATION OF TWO MOTES OVER THE RADIO FREQUENCY.

Components required: Three Arduino boards (UNO), data cables, connecting wires, three NRF24L01 modules, two LED.

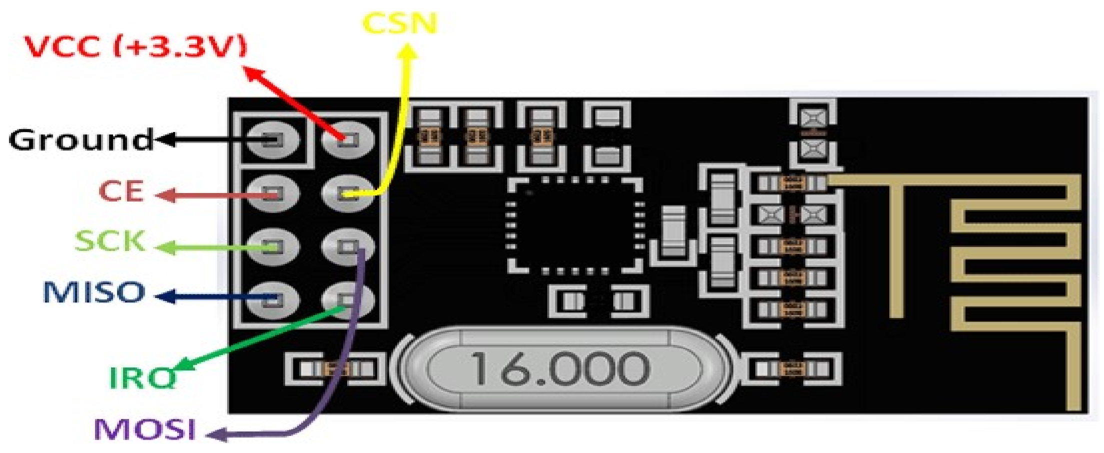
NRF24L01 Transceiver Module (Radio Frequency)

It uses the 2.4 GHz band and it can operate with baud rates from 250 kbps up to 2 Mbps. If used in open space and with lower baud rate its range can reach up to 100 meters.

The module can use 125 different channels which give a possibility to have a network of 125 independently working modems in one place. Each channel can have up to 6 addresses, or each unit can communicate with up to 6 other units at the same time.

The power consumption of this module is just around 12mA during transmission, which is even lower than a single LED. The operating voltage of the module is from 1.9 to 3.6V, but the good thing is that the other pins tolerate 5V logic, so we can easily connect it to an Arduino without using any logic level converters.

Three of these pins are for the SPI communication and they need to be connected to the SPI pins of the Arduino, but note that each Arduino board have different SPI pins. The pins CSN and CE can be connected to any digital pin of the Arduino board and they are used for setting the module in standby or active mode, as well as for switching between transmit or command mode. The last pin is an interrupt pin which doesn't have to be used.



NRF24L01 PINOUT

Three Arduino Uno and three NRF24L01 connections (same connection for two transmitter and single receiver)

Arduino	**NRF24L01**
VCC	3.3V
GND	GND
Pin 8	CSN
Pin 7	CE
Pin 13	SCK
Pin 11	MOSI
Pin 12	MISO

LED1(RED) connection: positive (long wire of LED) to pin 2 of Arduino Uno and negative to GND

LED2(Green) connection: positive (long wire of LED) to pin 4 of Arduino Uno and negative to GND

Steps:

1. Import RF24Network-master.zip to Arduino IDE via Sketch->Include Library-> ADD .Zip library
2. Import RF24-master.zip to Arduino IDE via Sketch->Include Library-> ADD .Zip library
3. Type the transmitter1 code in Arduino IDE and select board and port for first Arduino uno. Upload the transmitter1 code to Arduino (first Arduino).
4. Type the trasmitter2 code in another Arduino IDE(2nd) and select board and port for second Arduino uno. Upload the trasmitter2 code to Arduino (second Arduino).
5. Type the receiver code in another Arduino IDE(3rd) and select board and port for 3rd Arduino uno. Upload the receiver code to Arduino (Third Arduino).
6. If data sent in the transmitter1 code is ‘1’ then LED1(red) in receiver will be ON otherwise LED will be OFF.
7. If data sent in the transmitter2 code is ‘3’ then LED2(green) in receiver will be ON otherwise LED will be OFF.

Transmitter1 code

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
RF24 radio(7, 8); // CE, CSN
const byte address[6] = "00001";
void setup() {
  radio.begin();
  radio.openWritingPipe(address);
  radio.setPALevel(RF24_PA_MIN);
  radio.stopListening();
}
void loop() {
  int text= 1;
  radio.write(&text, sizeof(text));
  delay(1000);
}
```

Transmitter2 code

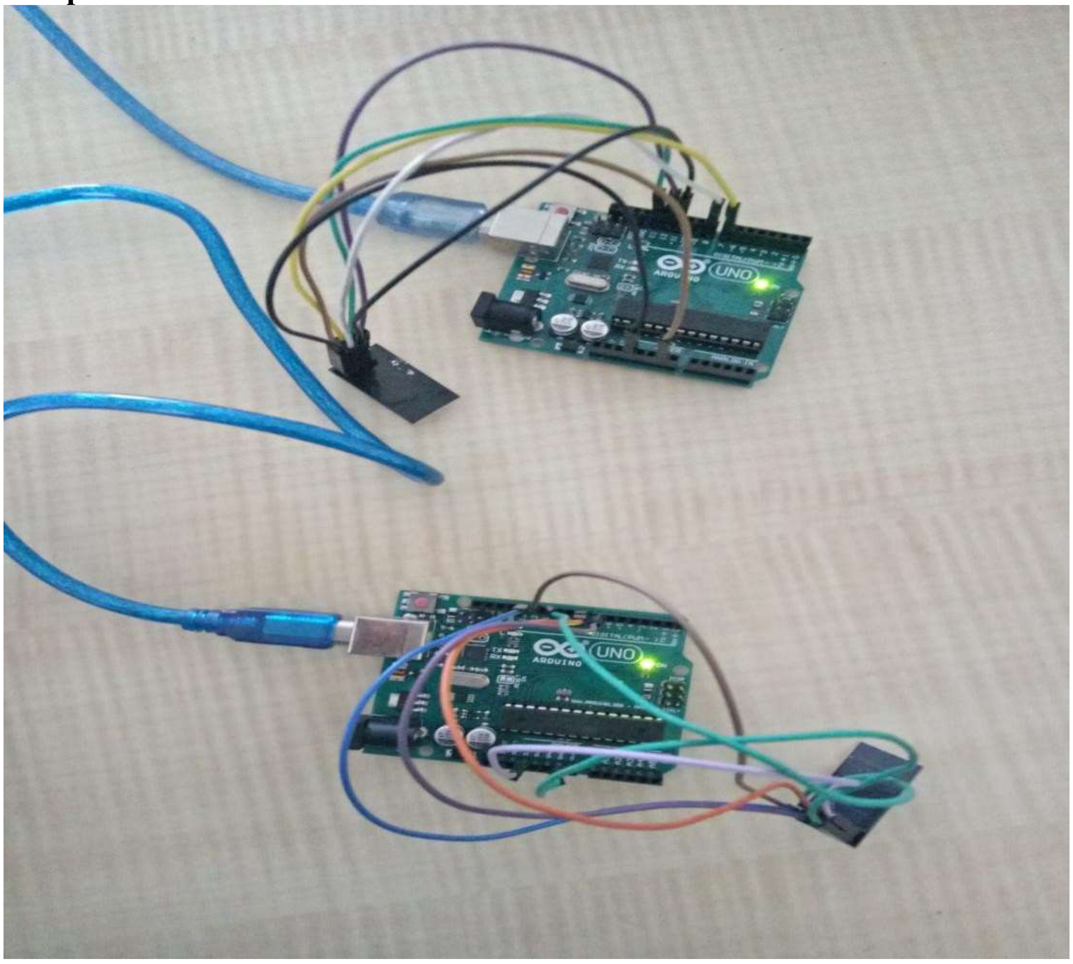
```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
RF24 radio(7, 8); // CE, CSN
const byte address[6] = "00001";
void setup() {
  radio.begin();
  radio.openWritingPipe(address);
  radio.setPALevel(RF24_PA_MIN);
  radio.stopListening();
}
void loop() {
  int t = 3;
  radio.write(&t, sizeof(t));
  delay(1000);
}
```

Receiver code

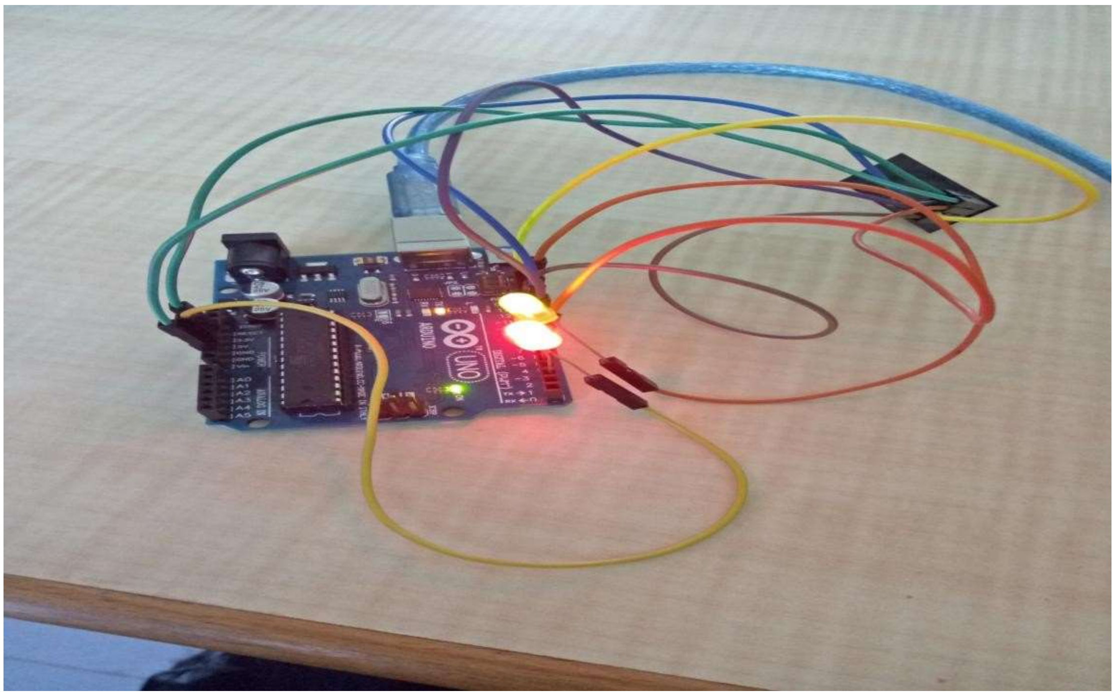
```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
RF24 radio(7, 8); // CE, CSN
const byte address[6] = "00001";
void setup() {
Serial.begin(9600);
  pinMode(2,  OUTPUT);
  pinMode(4, OUTPUT);
  radio.begin();
  radio.openReadingPipe(0, address);
  radio.setPALevel(RF24_PA_MIN);
  radio.startListening();
}
void loop() {
  if (radio.available()) {
    int text,t;
    radio.read(&text, sizeof(text));
    delay(1000);
    radio.read(&t, sizeof(t));
    Serial.println(t);
    Serial.println(text);
    if(text==1)
    {
      digitalWrite(2, HIGH);
    }
    else
    {
     digitalWrite(2, LOW);
    }

    if(t==3)
    {
```

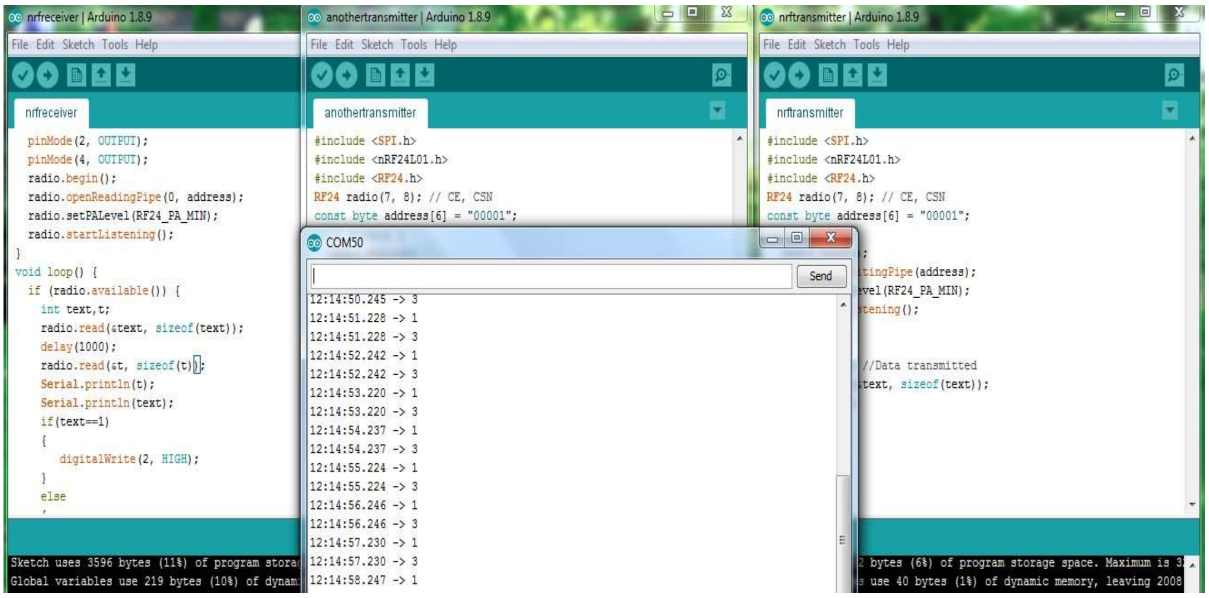
```
    digitalWrite(4, HIGH);
   }
   else
   {
    digitalWrite(4, LOW);
   }
   }
 }
```

Output



Two Transmitters


Receiver Side: Data received from transmitter1 is 1 and transmitter2 is 3 (Both Red and Green LED is ON)



```
  pinMode(2, OUTPUT);
  pinMode(4, OUTPUT);
  radio.begin();
  radio.openReadingPipe(0, address);
  radio.setPALevel(RF24_PA_MIN);
  radio.startListening();
}
void loop() {
  if (radio.available()) {
    int text,t;
    radio.read(&text, sizeof(text));
    delay(1000);
    radio.read(&t, sizeof(t));
    Serial.println(t);
    Serial.println(text);
    if(text==1)
    {
      digitalWrite(2, HIGH);
    }
    else
```

```
Sketch uses 3596 bytes (11%) of program storag
Global variables use 219 bytes (10%) of dynam
```

anothertransmitter | Arduino 1.8.9
File Edit Sketch Tools Help
anothertransmitter

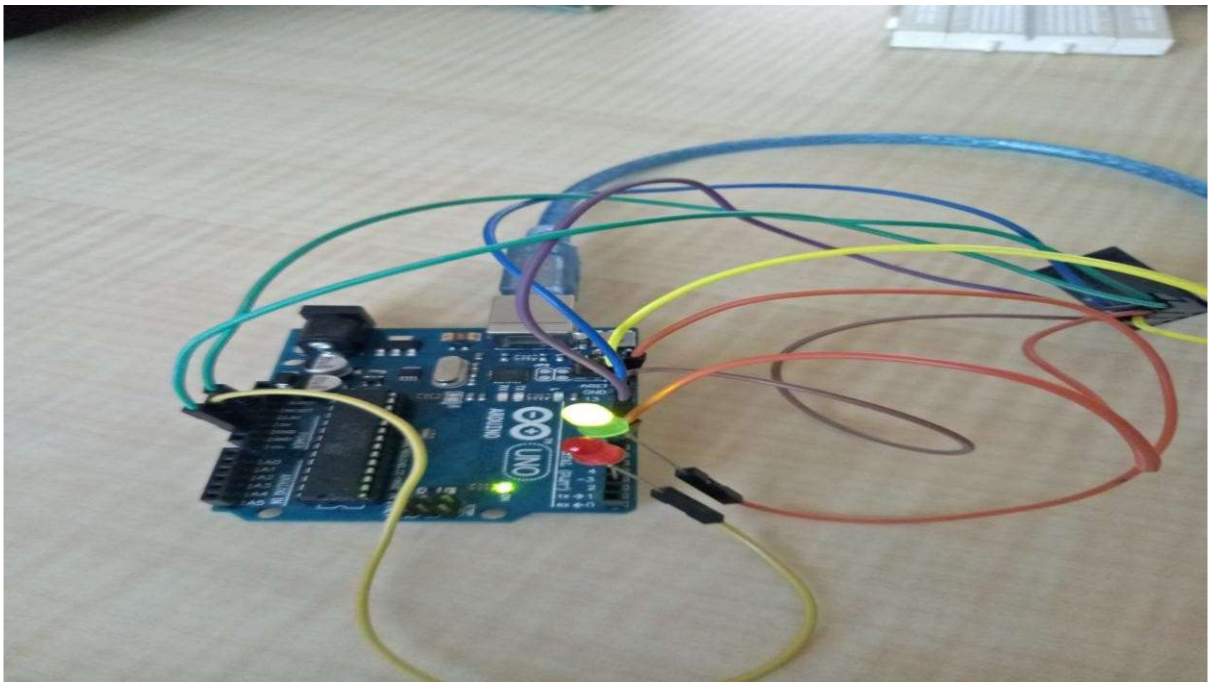
```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
RF24 radio(7, 8); // CE, CSN
const byte address[6] = "00001";
```

```
12:14:50.245 -> 3
12:14:51.228 -> 1
12:14:51.228 -> 3
12:14:52.242 -> 1
12:14:52.242 -> 3
12:14:53.220 -> 1
12:14:53.220 -> 3
12:14:54.237 -> 1
12:14:54.237 -> 3
12:14:55.224 -> 1
12:14:55.224 -> 3
12:14:56.246 -> 1
12:14:56.246 -> 3
12:14:57.230 -> 1
12:14:57.230 -> 3
12:14:58.247 -> 1
```

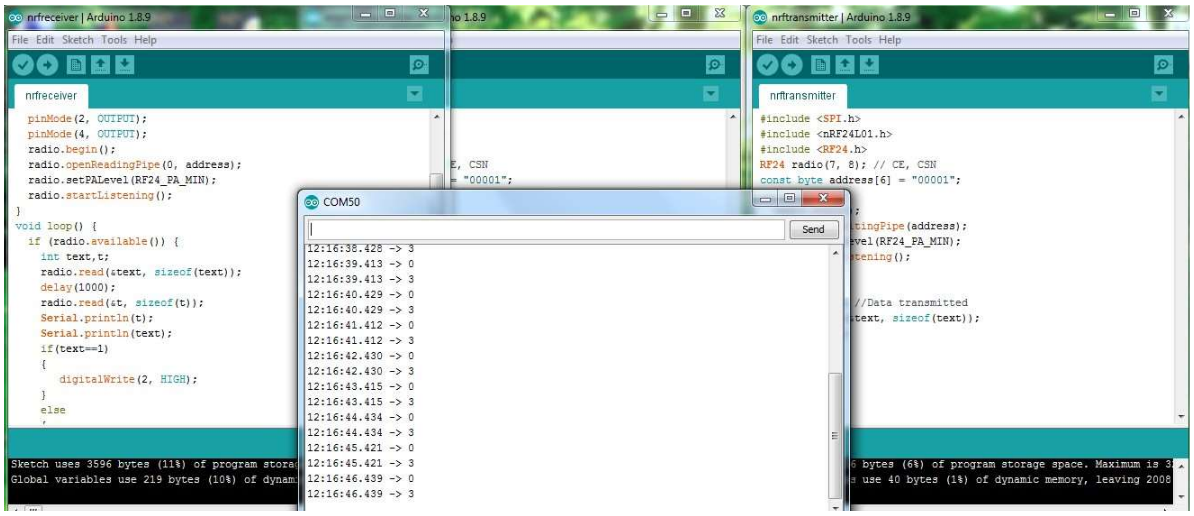
```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
RF24 radio(7, 8); // CE, CSN
const byte address[6] = "00001";

;
tingPipe(address);
evel(RF24_PA_MIN);
tening();


 //Data transmitted
text, sizeof(text));
```

```
2 bytes (6%) of program storage space. Maximum is 3
s use 40 bytes (1%) of dynamic memory, leaving 2008
```

Data received from transmitter1 is 1 and transmitter2 is 3 in serial monitor of receiver



Receiver Side: Data received from transmitter1 is 0 and transmitter2 is 3 (Red is OFF and Green LED is ON)



```
  pinMode(2, OUTPUT);
  pinMode(4, OUTPUT);
  radio.begin();
  radio.openReadingPipe(0, address);
  radio.setPALevel(RF24_PA_MIN);
  radio.startListening();
}
void loop() {
  if (radio.available()) {
    int text,t;
    radio.read(&text, sizeof(text));
    delay(1000);
    radio.read(&t, sizeof(t));
    Serial.println(t);
    Serial.println(text);
    if(text==1)
    {
       digitalWrite(2, HIGH);
    }
    else
```

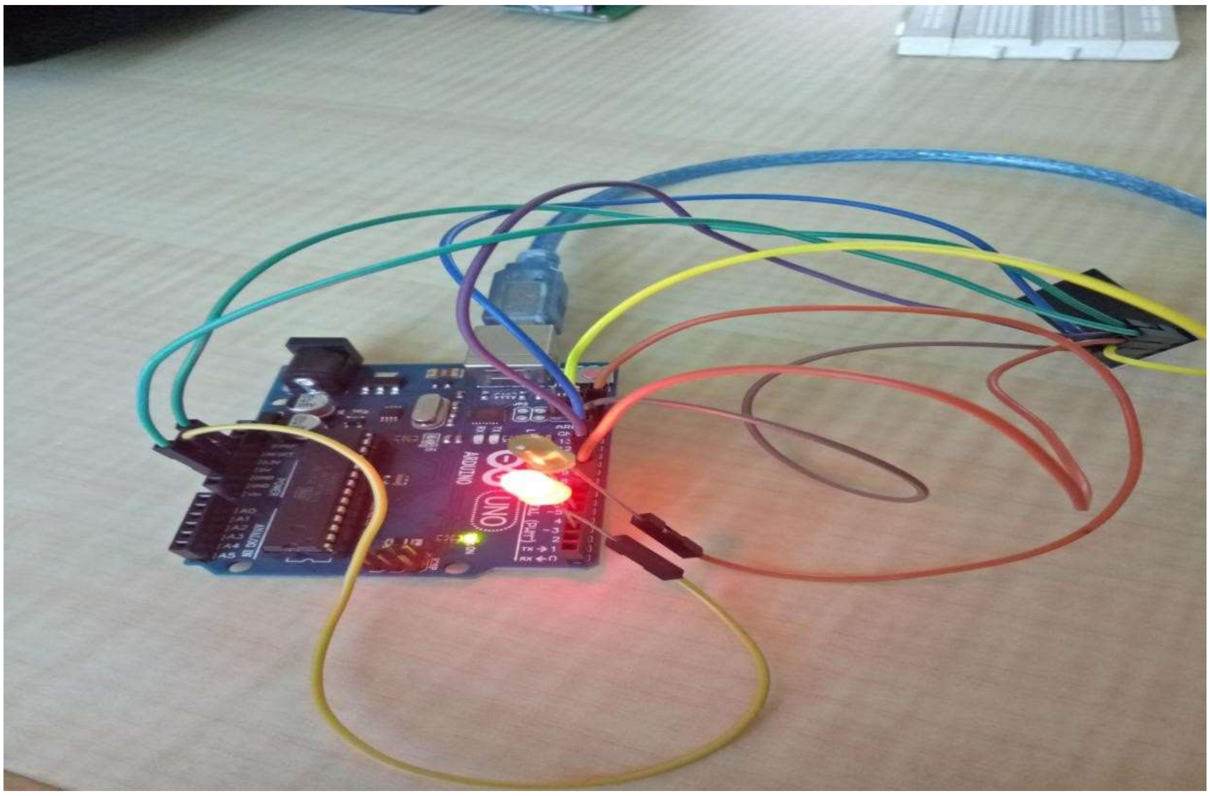
```
Sketch uses 3596 bytes (11%) of program storag
Global variables use 219 bytes (10%) of dynam
```

```
12:16:38.428 -> 3
12:16:39.413 -> 0
12:16:39.413 -> 3
12:16:40.429 -> 0
12:16:40.429 -> 3
12:16:41.412 -> 0
12:16:41.412 -> 3
12:16:42.430 -> 0
12:16:42.430 -> 3
12:16:43.415 -> 0
12:16:43.415 -> 3
12:16:44.434 -> 0
12:16:44.434 -> 3
12:16:45.421 -> 0
12:16:45.421 -> 3
12:16:46.439 -> 0
12:16:46.439 -> 3
```

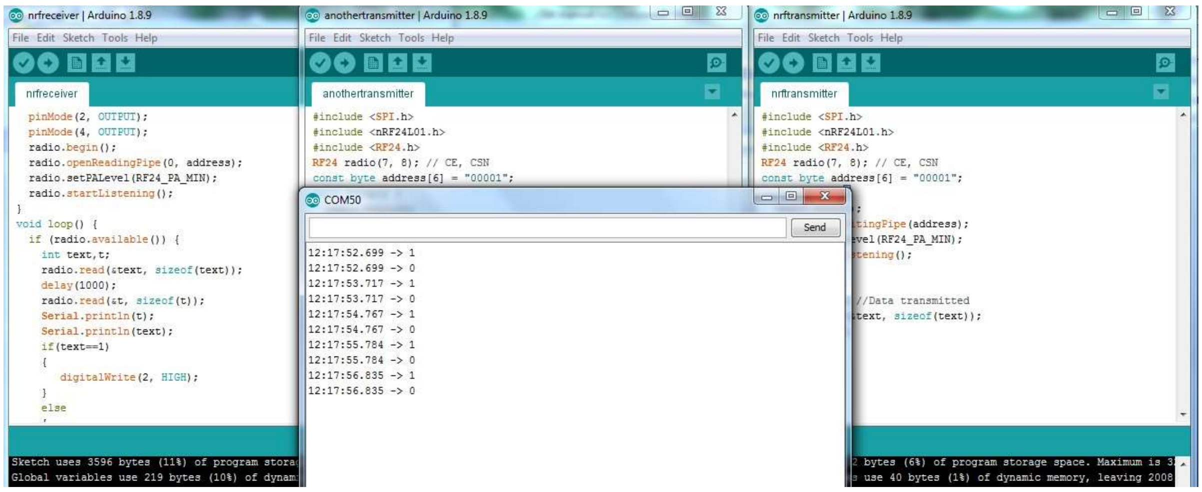
```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
RF24 radio(7, 8); // CE, CSN
const byte address[6] = "00001";

;
tingPipe(address);
vel(RF24_PA_MIN);
tening();


 //Data transmitted
text, sizeof(text));
```

```
6 bytes (6%) of program storage space. Maximum is 3
s use 40 bytes (1%) of dynamic memory, leaving 2008
```

Data received from transmitter1 is 0 and transmitter2 is 3 in serial monitor of receiver



Receiver Side: Data received from transmitter1 is 1 and transmitter2 is 0 (Red is ON and Green LED is OFF)



nrfreceiver | Arduino 1.8.9

```
  pinMode(2, OUTPUT);
  pinMode(4, OUTPUT);
  radio.begin();
  radio.openReadingPipe(0, address);
  radio.setPALevel(RF24_PA_MIN);
  radio.startListening();
}
void loop() {
  if (radio.available()) {
    int text,t;
    radio.read(&text, sizeof(text));
    delay(1000);
    radio.read(&t, sizeof(t));
    Serial.println(t);
    Serial.println(text);
    if(text==1)
    {
      digitalWrite(2, HIGH);
    }
    else
```

```
Sketch uses 3596 bytes (11%) of program stora
Global variables use 219 bytes (10%) of dynam
```

anothertransmitter | Arduino 1.8.9

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
RF24 radio(7, 8); // CE, CSN
const byte address[6] = "00001";
```

COM50

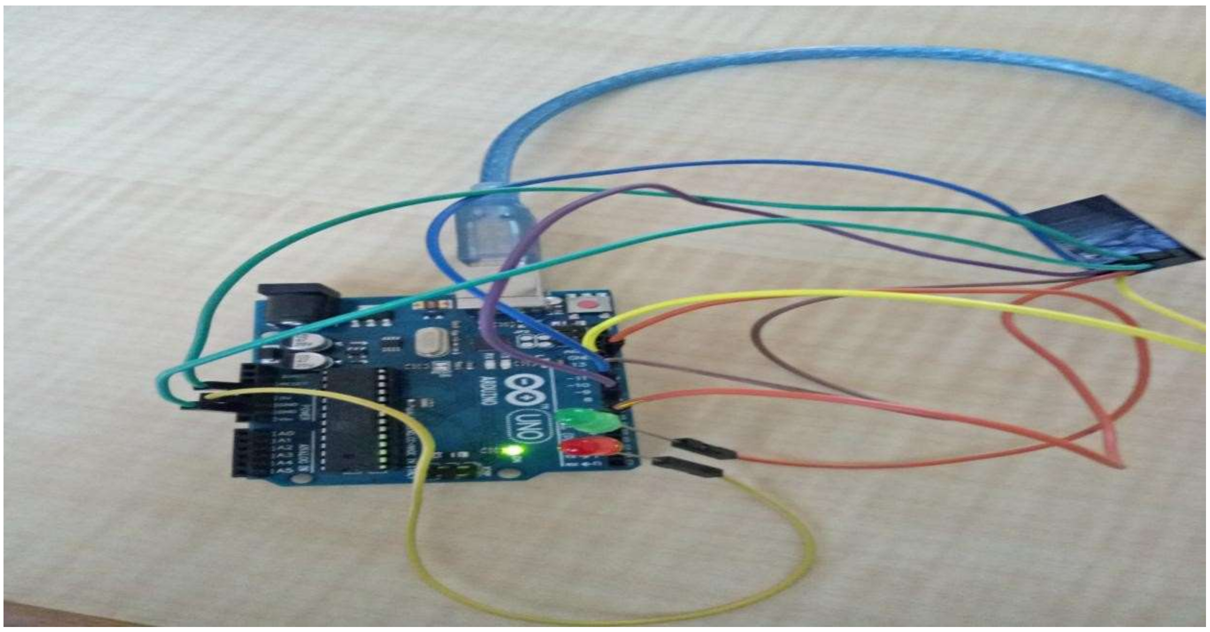
```
12:17:52.699 -> 1
12:17:52.699 -> 0
12:17:53.717 -> 1
12:17:53.717 -> 0
12:17:54.767 -> 1
12:17:54.767 -> 0
12:17:55.784 -> 1
12:17:55.784 -> 0
12:17:56.835 -> 1
12:17:56.835 -> 0
```

nrftransmitter | Arduino 1.8.9

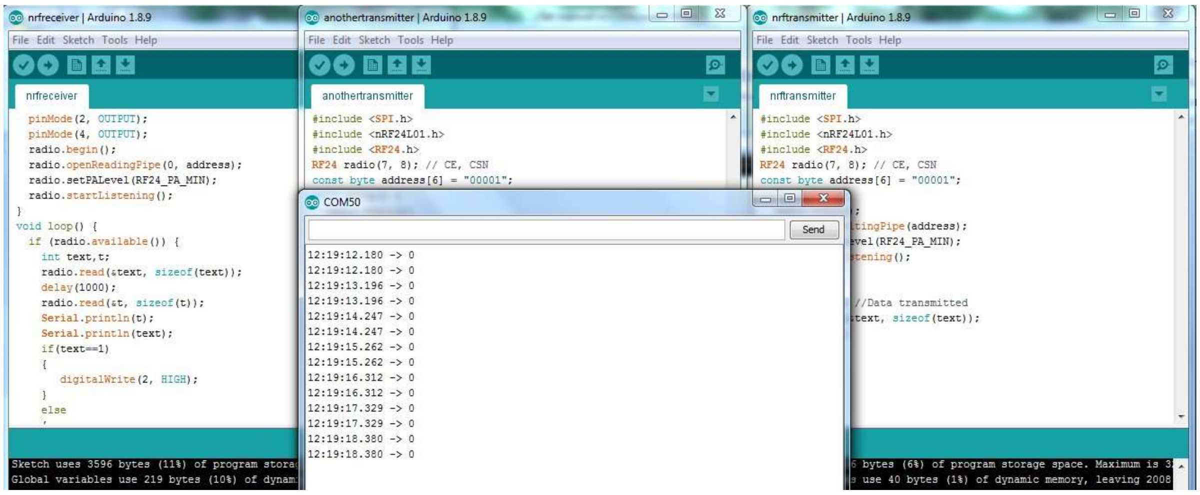
```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
RF24 radio(7, 8); // CE, CSN
const byte address[6] = "00001";

;
tingPipe(address);
vel(RF24_PA_MIN);
tening();


 //Data transmitted
text, sizeof(text));
```

```
2 bytes (6%) of program storage space. Maximum is 3
s use 40 bytes (1%) of dynamic memory, leaving 2008
```

Data received from transmitter1 is 1 and transmitter2 is 0 in serial monitor of receiver



Receiver Side: Data received from transmitter1 is 0 and transmitter2 is 0 (Both Red and Green LED is OFF)


nrfreceiver | Arduino 1.8.9

```
  pinMode(2, OUTPUT);
  pinMode(4, OUTPUT);
  radio.begin();
  radio.openReadingPipe(0, address);
  radio.setPALevel(RF24_PA_MIN);
  radio.startListening();
}
void loop() {
  if (radio.available()) {
    int text,t;
    radio.read(&text, sizeof(text));
    delay(1000);
    radio.read(&t, sizeof(t));
    Serial.println(t);
    Serial.println(text);
    if(text==1)
    {
      digitalWrite(2, HIGH);
    }
    else
```

```
Sketch uses 3596 bytes (11%) of program storag
Global variables use 219 bytes (10%) of dynam
```

anothertransmitter | Arduino 1.8.9

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
RF24 radio(7, 8); // CE, CSN
const byte address[6] = "00001";
```

COM50

```
12:19:12.180 -> 0
12:19:12.180 -> 0
12:19:13.196 -> 0
12:19:13.196 -> 0
12:19:14.247 -> 0
12:19:14.247 -> 0
12:19:15.262 -> 0
12:19:15.262 -> 0
12:19:16.312 -> 0
12:19:16.312 -> 0
12:19:17.329 -> 0
12:19:17.329 -> 0
12:19:18.380 -> 0
12:19:18.380 -> 0
```

nrftransmitter | Arduino 1.8.9

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
RF24 radio(7, 8); // CE, CSN
const byte address[6] = "00001";
;
tingPipe(address);
evel(RF24_PA_MIN);
tening();

//Data transmitted
text, sizeof(text));
```

```
6 bytes (6%) of program storage space. Maximum is 3
s use 40 bytes (1%) of dynamic memory, leaving 2008
```

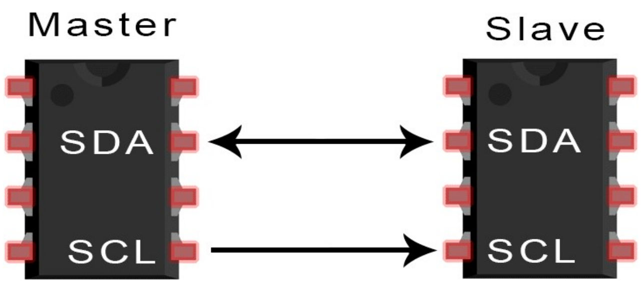
Data received from transmitter1 is 0 and transmitter2 is 0 in serial monitor of receiver

PROGRAM 4: I2C PROTOCOL STUDY

I2C Protocol:

I2C is a serial **protocol** for two-wire interface to connect low-speed devices like microcontrollers, EEPROMs, A/D and D/A converters, I/O interfaces and other similar peripherals in embedded systems.

I2C combines the best features of SPI and UARTs. With I2C, you can connect multiple slaves to a single master (like SPI) and you can have multiple masters controlling single, or multiple slaves. This is really useful when you want to have more than one microcontroller logging data to a single memory card or displaying text to a single LCD. Like UART communication, I2C only uses two wires to transmit data between devices:

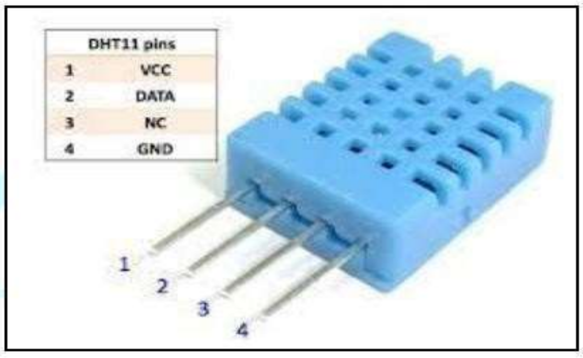


SDA (Serial Data) – The line for the master and slave to send and receive data.
SCL (Serial Clock) – The line that carries the clock signal.

I2C is a serial communication protocol, so data is transferred bit by bit along a single wire (the SDA line). Like SPI, I2C is synchronous, so the output of bits is synchronized to the sampling of bits by a clock signal shared between the master and the slave. The clock signal is always controlled by the master.

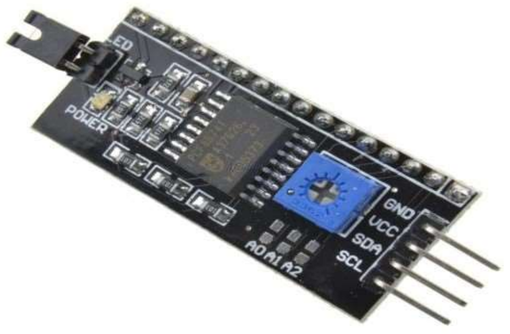
Components required: Arduino boards (UNO), data cable, connecting wires, DHT11 Sensor, one LCD with IIC/I2C Module.

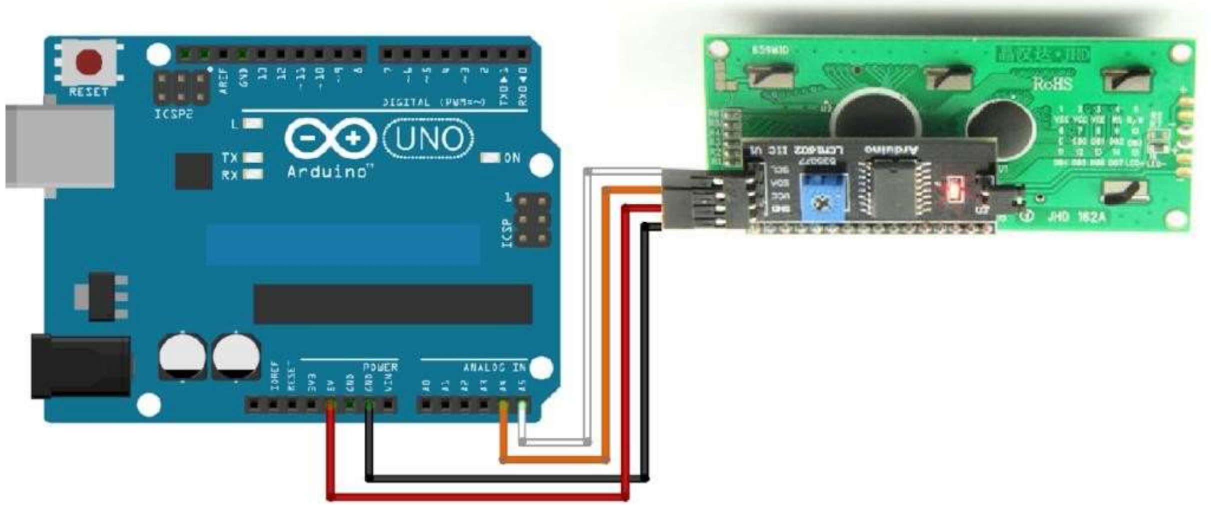
DHT11 sensor: Measures temperature and Humidity. Operates between 3.3-5V



IIC/I2C Module

This is an I2C interface for 16*2 LCD Display.





LCD with I2C module interfaced with Arduino uno

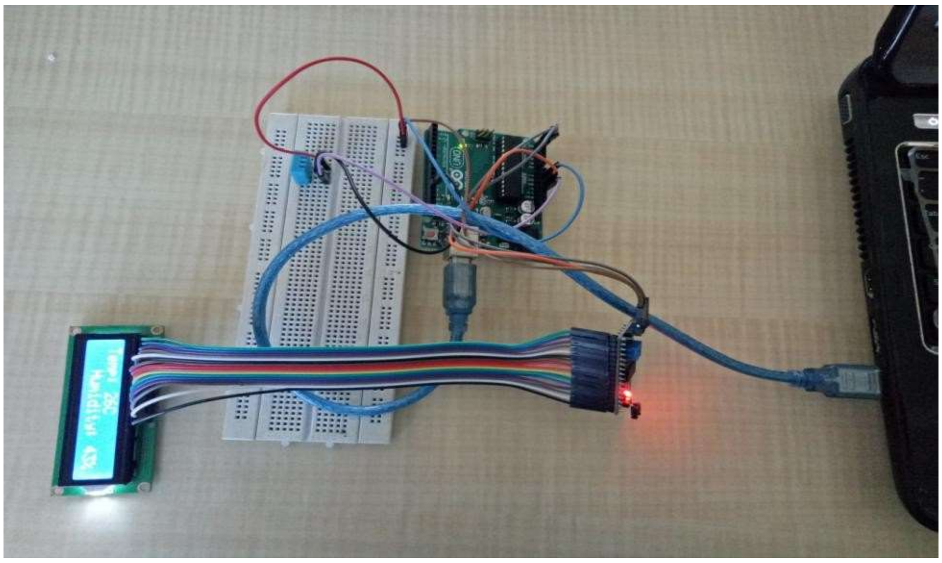
Connections:

DHT11	**Arduino Uno**
Data	Pin 12
VCC	3.3V
NC (no Connection)	
GND	GND
IIC/I2C	**Arduino Uno**
VCC	5V
GND	GND
SDA	A4
SCL	A5

I2C Module to LCD (16 pins of I2C module directly connected to 16 pins of LCD)

Steps:

1. Import DHL-sensor-library.zip to Arduino IDE via Sketch->Include Library-> ADD .Zip library
2. Import Newliquidcrystal_1.3.5.zip to Arduino IDE via Sketch->Include Library-> ADD .Zip library
3. Import LiquidCrystal_I2C.zip to Arduino IDE via Sketch->Include Library-> ADD .Zip library
4. Import Adafruit-sensor-master.zip to Arduino IDE via Sketch->Include Library-> ADD .Zip library
5. Type the code and connect to components. Select the board and port and upload the code.
6. Verify the output in serial monitor and also on LCD screen.



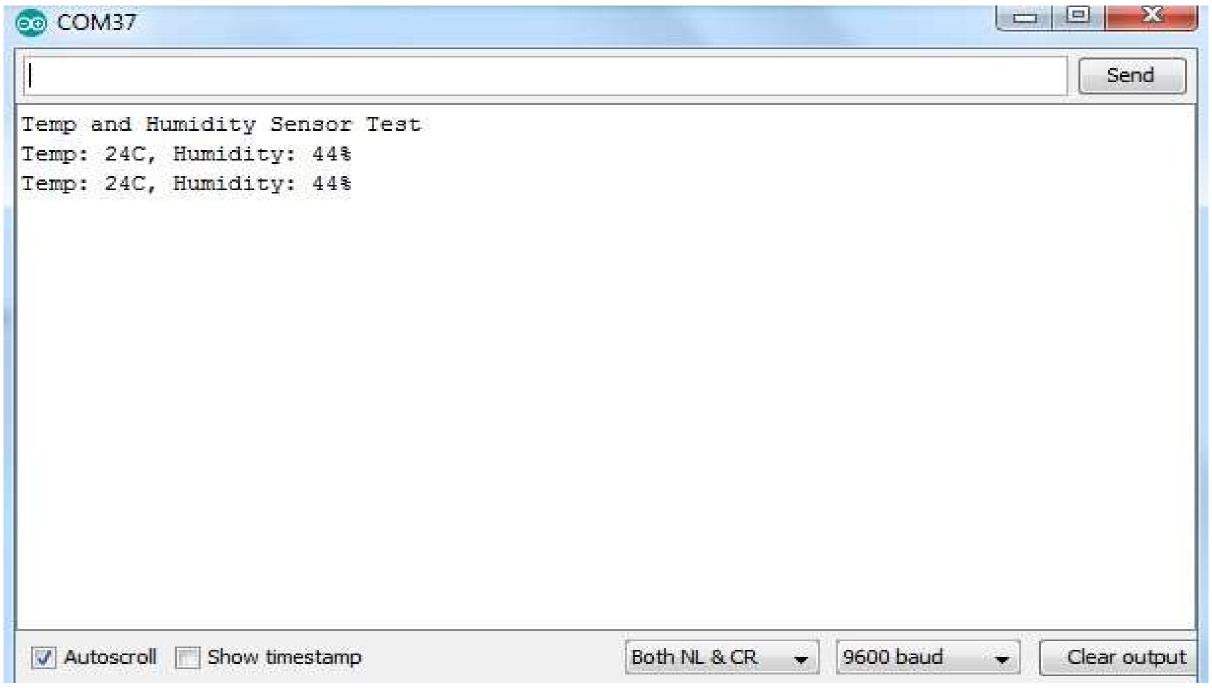
Overall Setup

Code

```
#include "DHT.h"
#define DHTPIN 12     // what digital pin we're connected to
#define DHTTYPE DHT11 // DHT 11
DHT dht(DHTPIN, DHTTYPE);
#include <Wire.h> // Comes with Arduino IDE
#include <LiquidCrystal_I2C.h>
// Set the LCD I2C address
LiquidCrystal_I2C lcd(0x27, 16,2);
```

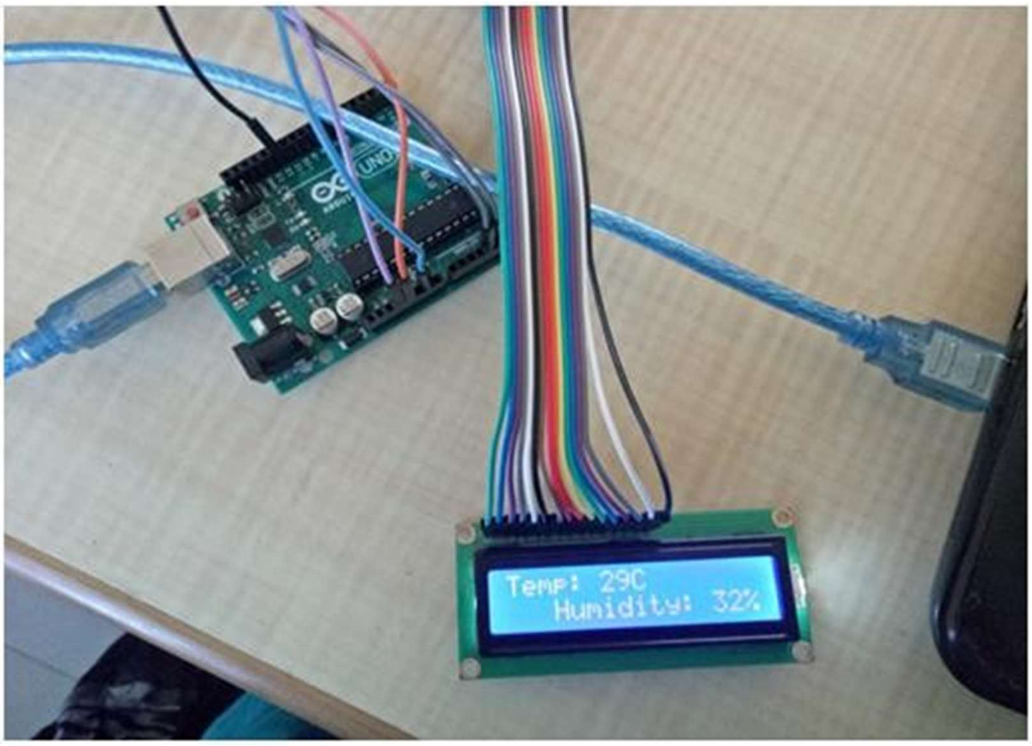
```
void setup( )
{
  Serial.begin(9600);
  lcd.init( );
  lcd.backlight( );
  Serial.println("Temp and Humidity Sensor Test");
  dht.begin( );
}
void loop( ) {
int h = dht.readHumidity();
  int t = dht.readTemperature();
lcd.clear( );
    // set the cursor to (0,0):
 lcd.setCursor(0, 0);
 lcd.print("Temp: ");
 lcd.print(t);
 lcd.print("C");
 lcd.setCursor(3,1);
 lcd.print("Humidity: ");
 lcd.print(h);
 lcd.print("%");
 Serial.print("Temp: ");
  Serial.print(t);
  Serial.print("C, Humidity: ");
  Serial.print(h);
  Serial.println("%");
  delay(2000);
}
```

Output



```
Temp and Humidity Sensor Test
Temp: 24C, Humidity: 44%
Temp: 24C, Humidity: 44%
```

Temperature and humidity display in Serial Monitor

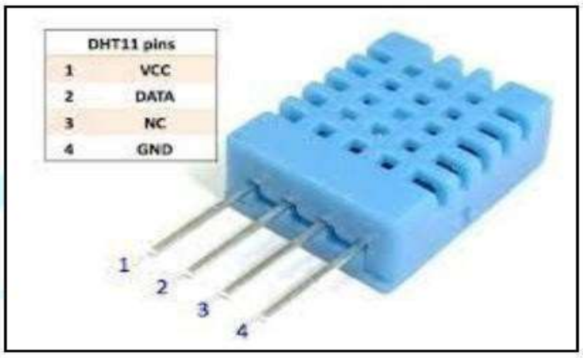


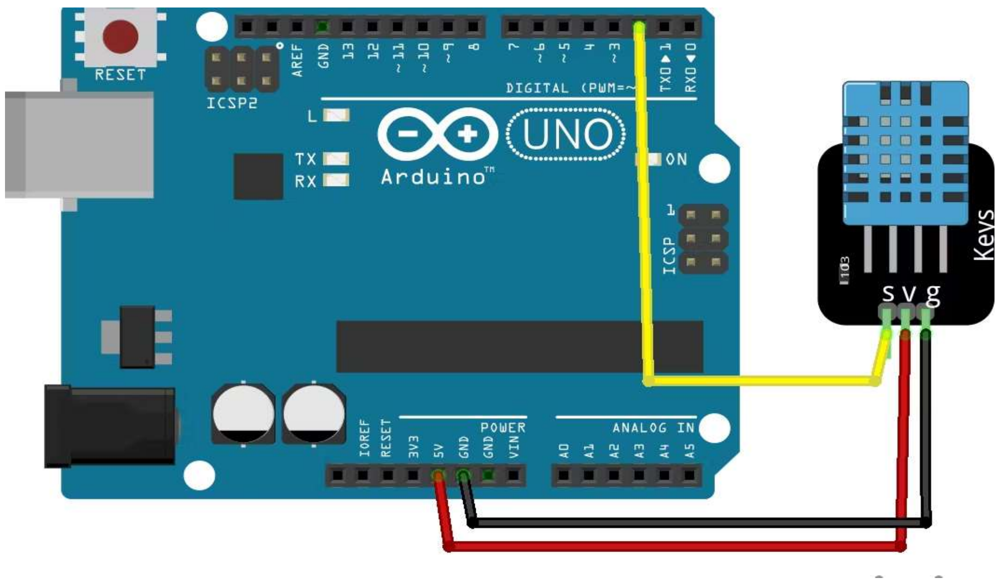
Temperature and humidity display on LCD screen

PROGRAM 5: READING TEMPERATURE AND RELATIVE HUMIDITY VALUE FROM THE SENSOR

Components required: Arduino boards (UNO), data cable, connecting wires, DHT11 Sensor.

DHT11 sensor: Measures temperature and Humidity. Operates between 3.3-5V

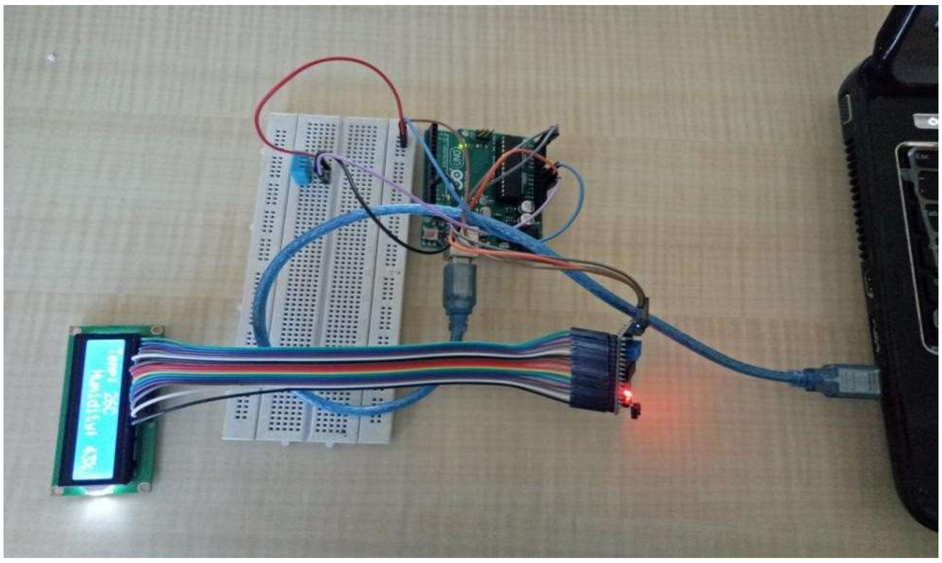




How to Set Up the DHT11 on an Arduino

Steps:

1. Import DHL-sensor-library.zip to Arduino IDE via Sketch->Include Library-> ADD .Zip library
2. Import Newliquidcrystal_1.3.5.zip to Arduino IDE via Sketch->Include Library-> ADD .Zip library
3. Import LiquidCrystal_I2C.zip to Arduino IDE via Sketch->Include Library-> ADD .Zip library
4. Import Adafruit-sensor-master.zip to Arduino IDE via Sketch->Include Library-> ADD .Zip library
5. Type the code and connect to components. Select the board and port and upload the code.
6. Verify the output in serial monitor and also on LCD screen.



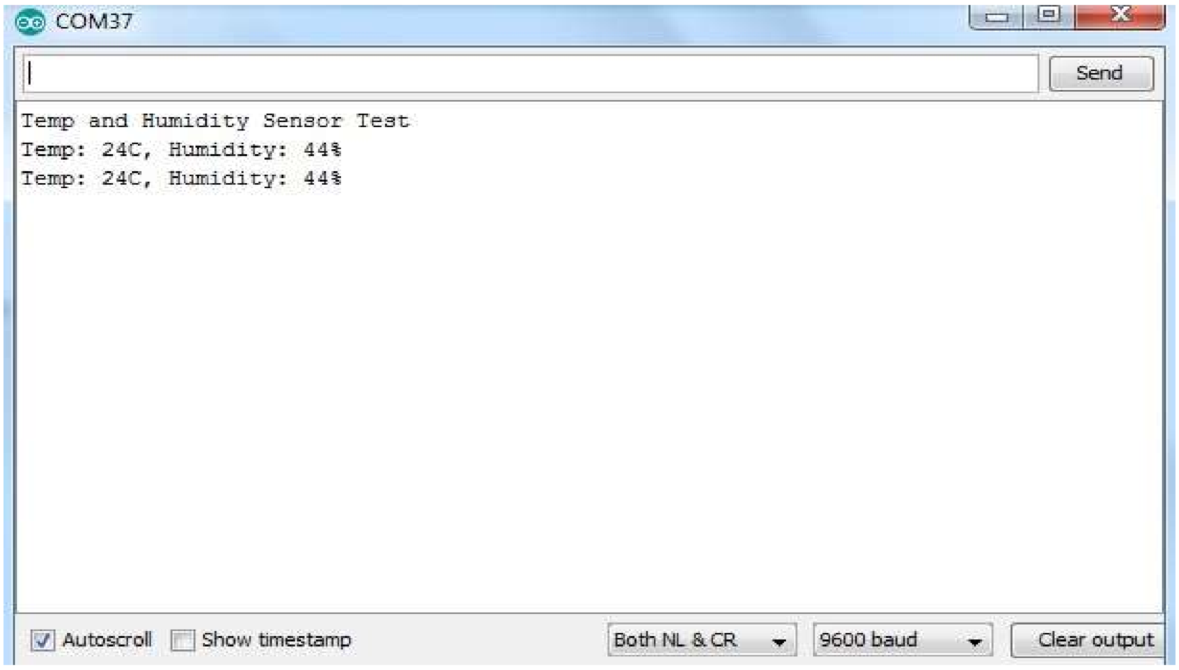
Overall Setup

Code

```
#include "DHT.h"
#define DHTPIN 12     // what digital pin we're connected to
#define DHTTYPE DHT11 // DHT 11
DHT dht(DHTPIN, DHTTYPE);
#include <Wire.h> // Comes with Arduino IDE
#include <LiquidCrystal_I2C.h>
// Set the LCD I2C address
LiquidCrystal_I2C lcd(0x27, 16,2);
```

```
void setup( )
{
  Serial.begin(9600);
  lcd.init( );
  lcd.backlight( );
  Serial.println("Temp and Humidity Sensor Test");
  dht.begin( );
}
void loop( ) {
int h = dht.readHumidity();
  int t = dht.readTemperature();
lcd.clear( );
    // set the cursor to (0,0):
 lcd.setCursor(0, 0);
 lcd.print("Temp: ");
 lcd.print(t);
 lcd.print("C");
 lcd.setCursor(3,1);
 lcd.print("Humidity: ");
 lcd.print(h);
 lcd.print("%");
 Serial.print("Temp: ");
  Serial.print(t);
  Serial.print("C, Humidity: ");
  Serial.print(h);
  Serial.println("%");
  delay(2000);
}
```

Output



```
Temp and Humidity Sensor Test
Temp: 24C, Humidity: 44%
Temp: 24C, Humidity: 44%
```

Temperature and humidity display in Serial Monitor

PROGRAM 6: STUDY OF CONNECTIVITY AND CONFIGURATION OF RASPBERRY-PI/ BEAGLE BOARD CIRCUIT WITH BASIC PERIPHERALS, LEDS, UNDERSTANDING GPIO AND ITS USE IN PROGRAM.

A) **Raspberry-Pi:-** The Rasberry Pi is a series of small single-board computers developed in the United Kingdom by the Raspberry Pi Foundation_to promote teaching of basic computer science in schools and in developing countries. It does not include peripherals (such as keyboards and mice). The Raspberry Pi is a low cost, credit-card sized computer that plugs into a computer monitor or TV, and uses a standard keyboard and mouse. It is a capable little device that enables people of all ages to explore computing, and to learn how to program in languages like Scratch and Python. The Raspberry Pi is a credit-card-sized computer that costs between $5 and $35. It's available anywhere in the world, and can function as a proper desktop computer or be used to build smart devices. A Raspberry Pi is a general-purpose computer, usually with a Linux operating system, and the ability to run multiple programs. Raspberry Pi is like the brain. Its primary advantage comes in processing higher level processing capability. It's a single board computer.

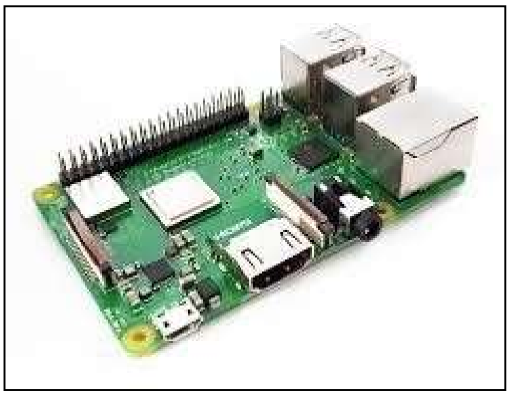


Fig.A.1: - **Raspberry-Pi**

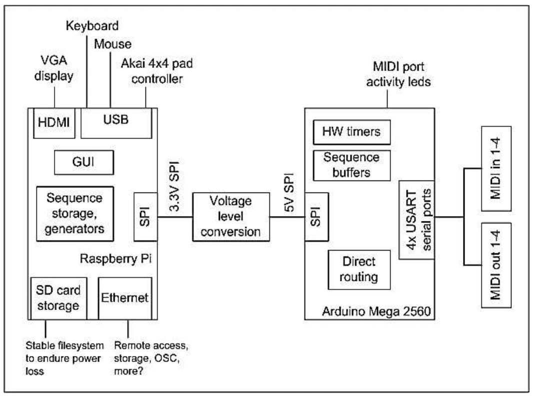


Fig. A.2: -**Raspberry-Pi Architecture**

Here are the various components on the Raspberry Pi board:

- **ARM CPU/GPU** -- This is a Broadcom BCM2835 System on a Chip (SoC) that's made up of an ARM central processing unit (CPU) and a Video core 4 graphics processing unit (GPU). The CPU handles all the computations that make a computer work (taking input, doing calculations and producing output), and the GPU handles graphics output.
- **GPIO** -- These are exposed general-purpose input/output connection points that will allow the real hardware hobbyists the opportunity to tinker.
- **RCA** -- An RCA jack allows connection of analog TVs and other similar output devices.
- **Audio out** -- This is a standard 3.55-millimeter jack for connection of audio output devices such as headphones or speakers. There is no audio in.
- **LEDs** -- Light-emitting diodes, for your entire indicator light needs.
- **USB** -- This is a common connection port for peripheral devices of all types (including your

mouse and keyboard). Model A has one, and Model B has two. You can use a hub to expand the number of ports or plug your mouse into your keyboard if it has its own port.

- **HDMI** -- This connector allows you to hook up a high-definition television or other compatible device using an HDMI cable.
- **Power** -- This is a 5v Micro power connector into which you can plug your compatible power supply.
- **SD card slot** -- This is a full-sized SD card slot. An SD card with an operating system (OS) installed is required for booting the device. They are available for purchase from the manufacturers, but you can also download an OS and save it to the card yourself if you have a Linux machine and the wherewithal.
- **Ethernet** -- This connector allows for wired network access and is only available on the Model B.

B) Beagle board:- The Beagle Board is a low-power open-source single-board computer produced by Texas Instruments in association with Digi-Key and Newark element14. The Beagle Board was also designed with open source software development in mind, and as a way of demonstrating the Texas Instrument's OMAP3530 system-on-a-chip.]The board was developed by a small team of engineers as an educational board that could be used in colleges around the world to teach open source hardware and software capabilities. It is also sold to the public under the Creative Commons share-alike license. The board was designed using Cadence OrCAD for schematics and Cadence Allegro for PCB manufacturing; no simulation software was used. Beagle Bone Black is a low-cost, open source, community-supported development platform for ARM® Cortex™-A8 processor developers and hobbyists. Boot Linux in under 10-seconds and get started on Sitara™ AM335x ARM Cortex-A8 processor development in less than 5 minutes with just a single cable.

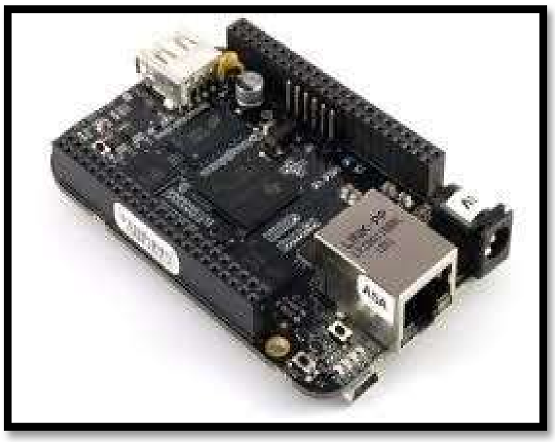


Fig.B.1: -Beagle Board Black

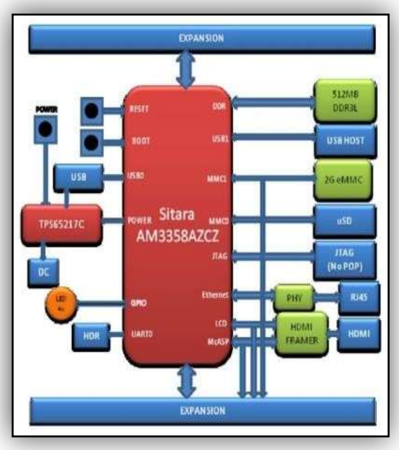


Fig.B.1: - Beagle Board Black architecture

Here are the various components on the Beagle board:

Processor: AM335x 1GHz ARM® Cortex-A8

- 512MB DDR3 RAM
- 4GB 8-bit eMMC on-board flash storage
- 3D graphics accelerator
- NEON floating-point accelerator
- 2x PRU 32-bit microcontrollers

Connectivity

- client for power & communications
- host
- Ethernet
- HDMI
- 2x 46 pin headers

Software Compatibility

- Debian
- Android
- Ubuntu
- Cloud9 IDE on Node.js w/ BoneScript library
- plus, much more

- **Arduino:- Arduino** is an open-source hardware and software company, project and user community that designs and manufactures single-board microcontrollers and microcontroller kits for building digital devices and interactive objects that can sense and control objects in the physical and digital world. Arduino board designs use a variety of microprocessors and controllers. The boards are equipped with sets of digital and analog input/output (I/O) pins that may be interfaced to various expansion boards or breadboards (shields) and other circuits. The boards feature serial communications interfaces, including Universal Serial Bus () on some models, which are also used for loading programs from personal computers. The microcontrollers are typically programmed using a dialect of features from the programming languages C and C++. In addition to using traditional compiler tool chains, the Arduino project provides an integrated development environment (IDE) based on the Processing language project. Arduino is open-source hardware. The hardware reference designs are distributed under a Creative Commons Attribution Share-Alike

    - 2.5 license and are available on the Arduino website. Layout and production files for some versions of the hardware are also available.

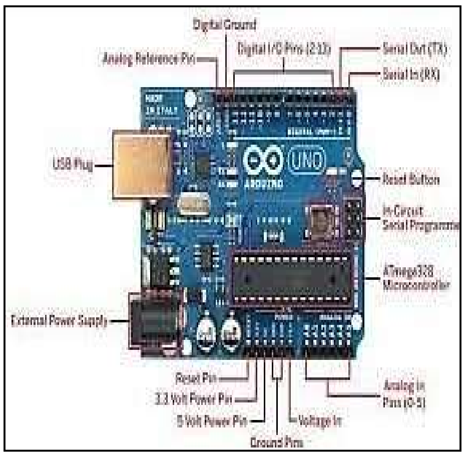


Fig.C.1: - Arduino Board

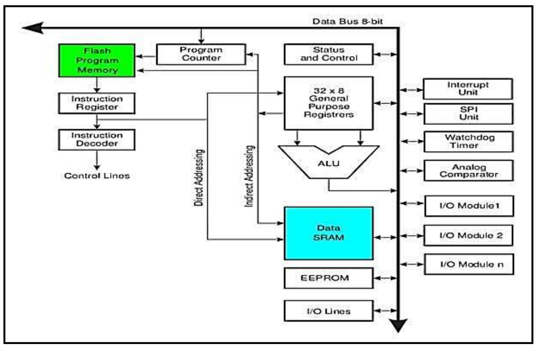


Fig.C.1: - Arduino Board Architecture.

Here are the various components on the Arduino board:

Microcontrollers`

- ATmega328P (used on most recent boards)

- ATmega168 (used on most Arduino Diecimila and early Duemilanove) ATmega8 (used on some older board)

Digital Pins

- In addition to the specific functions listed below, the digital pins on an Arduino board can be used for general purpose input and output via the pinMode(), digitalRead(), and digitalWrite() commands. Each pin has an internal pull-up resistor which can be turned on and off using digitalWrite() (w/ a value of HIGH or LOW, respectively) when the pin is configured as an input. The maximum current per pin is 40 mA.

Analog Pins

- In addition to the specific functions listed below, the analog input pins support 10-bit analog-to-digital conversion (ADC) using the analogRead() function. Most of the analog inputs can also be used as digital pins: analog input 0 as digital pin 14 through analog input 5 as digital pin 19. Analog inputs 6 and 7 (present on the Mini and BT) cannot be used as digital pins.

Power Pins

- **VIN** (sometimes labelled "9V"). The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin. Note that different boards accept different input voltages ranges, please see the documentation for your board. Also note that the LilyPad has no VIN pin and accepts only a regulated input.

Other Pins

- **AREF.** Reference voltage for the analog inputs. Used with analogReference().
- **Reset.** Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.
- Analog Reference pin (orange)
- Digital Ground (light green)
- Digital Pins 2-13 (green)
- Digital Pins 0-1/Serial In/Out - TX/RX (dark green) - These pins cannot be used for digital i/o (digitalRead and digitalWrite) if you are also using serial communication (e.g. Serial.begin).
- Reset Button - S1 (dark blue)
- In-circuit Serial Programmer (blue-green)
- Analog In Pins 0-5 (light blue)
- Power and Ground Pins (power: orange, grounds: light orange)
- External Power Supply In (9-12VDC) - X1 (pink)
- Toggles External Power and Power (place jumper on two pins closest to desired supply) - SV1 (purple)
- USB (used for uploading sketches to the board and for serial communication between the board and the computer; can be used to power the board) (yellow)

Code:

```
import RPi.GPIO as GPIO
import time
led1Pin = 7
led2Pin = 11
led3Pin = 13

led4Pin = 15

GPIO.setmode(GPIO.BOARD)
GPIO.setwarnings(False)
GPIO.setup(led1Pin, GPIO.OUT)
```

```
    GPIO.setup(led2Pin, GPIO.OUT)
    GPIO.setup(led3Pin, GPIO.OUT)
    GPIO.setup(led4Pin, GPIO.OUT)

    while True:
      print "on"
      GPIO.output(led1Pin, True)
      time.sleep(1)
      GPIO.output(led2Pin, True)
      time.sleep(1)
      GPIO.output(led3Pin, True)
      time.sleep(1)
      GPIO.output(led4Pin, True)
      time.sleep(1)

      print "off"
      GPIO.output(led1Pin, False)
      time.sleep(1)
      GPIO.output(led2Pin, False)
      time.sleep(1)
      GPIO.output(led3Pin, False)
      time.sleep(1)
      GPIO.output(led4Pin, False)
 time.sleep(1)
```

PROGRAM 7: STUDY OF DIFFERENT OPERATING SYSTEMS FOR RASPBERRY-PI/BEAGLE BOARD. UNDERSTANDING THE PROCESS OF OS INSTALLATION ON RASPBERRY-PI/BEAGLE BOARD

1) **Raspberry-Pi: -** The Pi can run the official Raspbian OS, Ubuntu Mate, Snappy Ubuntu Core, the Kodi- based media centers OSMC and LibreElec, the non-Linux based Risc OS (one for fans of 1990s Acorn computers). It can also run Windows 10 IoT Core, which is very different to the desktop version of Windows, as mentioned below.

- **OS which installs on Raspberry-Pi:** Raspbian, Ubuntu MATE, Snappy Ubuntu, Pidora, Linutop, SARPi, Arch Linux ARM, Gentoo Linux, etc**.**

How to install Raspbian on Raspberry-Pi:

Step 1: Download Raspbian

Step 2: Unzip the file. The Raspbian disc image is compressed, so you'll need to unzip it. The file uses the ZIP64 format, so depending on how current your built-in utilities are, you need to use certain programs to unzip it.

Step 3: Write the disc image to your microSD card. Next, pop your microSD card into our computer and write the disc image to it. The process of actually writing the image will be slightly different across these programs, but it's pretty self-explanatory no matter what you're using. Each of these programs will have you select the destination (make sure you've picked your microSD card!) and the disc image (the unzipped Raspbian file). Choose, double-check, and then hit the button to write.

Step 4: Put the microSD card in your Pi and boot up. Once the disc image has been written to the microSD card, you're ready to go! Put that sucker into your Raspberry Pi, plug in the peripherals and power source, and enjoy. The current edition to Raspbian will boot directly to the desktop. Your default credentials are username pi and password raspberry.

2) **BeagleBone Black: -** The BeagleBone Black includes a 2GB or 4GB on-board eMMC flash memory chip. It comes with the Debian distribution factory pre-installed. You can flash new operating systems including Angstrom, Ubuntu, Android, and others.

Os which install on BeagleBone Black: Angstrom, Android, Debian, Fedora, Buildroot, Gentoo, Nerves Erlang/OTP, Sabayon, Ubuntu, Yocto, MINIX 3

- **How to install Debian on BeagleBone Black:**

Step 1: Download Debian img.xz file.

Step 2: Unzip the file.

Step 3: Insert your MicroSD (µSD) card into the proper slot. Most µSD cards come with a full-sized SD card that is really just an adapter. If this is what you have then insert the µSD into the adapter, then into your card reader.

Step 4: Now open Win32 Disk imager, click the blue folder icon, navigate to the Debian img location, and double click the file. Now click Write and let the process complete. Depending on your processor and available RAM it should be done in around 5 minutes.

Step 5: Once that's done, you'll get a notification pop-up. Now we're ready to get going. Remove the SD adapter from the card slot, remove the µSD card from the adapter. With the cable disconnected insert the µSD into the BBB.

Step 6: Now, this next part is pretty straight forward. Plug the cable in and wait some more. If everything is going right you will notice that the four (4) leds just above the cable are doing the KIT impression. This could take up to 45 minutes, I just did it again in around 5 minutes. Your mileage will vary. Go back and surf reddit some more**.**

Step 7: If you are not seeing the leds swing back and forth you will need to unplug the cable, press and hold down the user button above the µSD card slot (next to the 2 little 10 pin ICs) then plug in the cable. Release the button and wait. You should see the LEDs swinging back and forth after a few seconds. Once this happens it's waiting time. When all 4 LEDs next to the slot stay lit at the same time the flash process has been completed.

Step 8: Remove the card and reboot your BBB. You can reboot the BBB by removing and reconnecting the cable, or hitting the reset button above the cable near the edge of the board.

Step 9: Now using putty, or your SSH flavor of choice, connect to the BBB using the IP address 192.168.7.2. You'll be prompted for a username. Type root and press Enter. By default, there is no root password. I recommend changing this ASAP if you plan on putting your BBB on the network. To do this type password, hit enter, then enter your desired password. You will be prompted to enter it again to verify.

3) **Arduino: -** The Arduino itself has no real operating system. You develop code for the Arduino using the Arduino IDE which you can download from Arduino - Home. Versions are available for Windows, Mac and Linux. The Arduino is a constrained microcontroller.

Arduino consists of both a physical programmable circuit board (often referred to as a microcontroller) and a piece of software, or IDE (Integrated Development Environment) that runs on your computer, used to write and upload computer code to the physical board. You are literally writing the "firmware" when you write the code and upload it. It's both good and its bad**.**

PROGRAM 8: FAMILIARIZATION WITH THE CONCEPT OF IOT, ARDUINO / RASPBERRY PI AND PERFORM NECESSARY SOFTWARE INSTALLATION.

Internet of Things (IoT): The Internet of Things (IoT) describes the network of physical objects — “things”—that are embedded with sensors, software, and other technologies for the purpose of connecting and exchanging data with other devices and systems over the internet. These devices range from ordinary household objects to sophisticated industrial tools

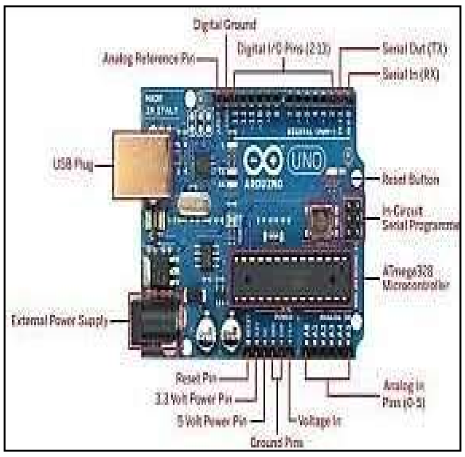
IOT is the core principle in applications of ‘smart devices. Over the past few years, IoT has become one of the most important technologies of the 21st century. Now that we can connect everyday objects—kitchen appliances, cars, thermostats, baby monitors—to the internet via embedded devices, seamless communication is possible between people, processes, and things.

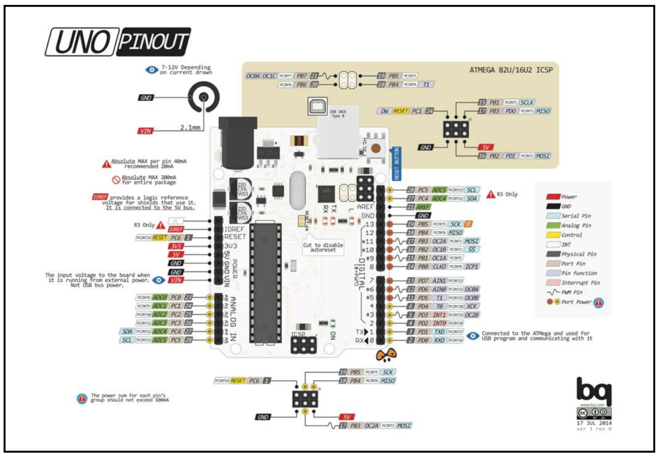
By means of low-cost computing, the cloud, big data, analytics, and mobile technologies, physical things can share and collect data with minimal human intervention. In this hyperconnected world, digital systems can record, monitor, and adjust each interaction between connected things. The physical world meets the digital world—and they cooperate.

Industrial IoT (IIoT) refers to the application of IoT technology in industrial settings, especially with respect to instrumentation and control of sensors and devices that engage cloud technologies. Refer to this Titan use case PDF for a good example of IIoT. Recently, industries have used machine-to-machine communication (M2M) to achieve wireless automation and control. But with the emergence of cloud and allied technologies (such as analytics and machine learning), industries can achieve a new automation layer and with it create new revenue and business models. IIoT is sometimes called the fourth wave of the industrial revolution, or Industry 4.0. The following are some common uses for IIoT:

- Smart manufacturing
- Connected assets and preventive and predictive maintenance
- Smart power grids
- Smart cities
- Connected logistics
- Smart digital supply chains

Arduino UNO: The **Arduino Uno** is an open-source microcontroller board based on the Microchip ATmega328P microcontroller (MCU) and developed by Arduino.cc and initially released in 2010. The board is equipped with sets of digital and analog input/output (I/O) pins that may be interfaced to various expansion boards (shields) and other circuits. The board has 14 digital I/O pins (six capable of PWM output), 6 analog I/O pins, and is programmable with the Arduino IDE (Integrated Development Environment), via a type B USB cable. It can be powered by a USB cable or a barrel connector that accepts voltages between 7 and 20 volts, such as a rectangular 9-volt battery.





General pin functions

- **LED**: There is a built-in LED driven by digital pin 13. When the pin is high value, the LED is on, when the pin is low, it is off.
- **VIN**: The input voltage to the Arduino/Genuino board when it is using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.
- **5V**: This pin outputs a regulated 5V from the regulator on the board. The board can be supplied with power either from the DC power jack (7 - 20V), the USB connector (5V), or the VIN pin of the board (7-20V). Supplying voltage via the 5V or 3.3V pins bypasses the regulator, and can damage the board.

- **3V3**: A 3.3-volt supply generated by the on-board regulator. Maximum current draw is 50 mA.
- **GND**: Ground pins.
- **IOREF**: This pin on the Arduino/Genuino board provides the voltage reference with which the microcontroller operates. A properly configured shield can read the IOREF pin voltage and select the appropriate power source, or enable voltage translators on the outputs to work with the 5V or 3.3V.
- **Reset**: Typically used to add a reset button to shields that block the one on the board.

Special pin functions

Each of the 14 digital pins and 6 analog pins on the Uno can be used as an input or output, under software control (using pinMode(), digitalWrite(), and digitalRead() functions). They operate at 5 volts. Each pin can provide or receive 20 mA as the recommended operating condition and has an internal pull-up resistor (disconnected by default) of 20-50K ohm. A maximum of 40mA must not be exceeded on any I/O pin to avoid permanent damage to the microcontroller. The Uno has 6 analog inputs, labeled A0 through A5; each provides 10 bits of resolution (i.e. 1024 different values). By default, they measure from ground to 5 volts, though it is possible to change the upper end of the range using the AREF pin and the analog Reference () function.

In addition, some pins have specialized functions:

- **Serial** / UART: pins 0 (RX) and 1 (TX). Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the ATmega8U2 USB-to-TTL serial chip.
- **External interrupts**: pins 2 and 3. These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value.
- **PWM** (pulse-width modulation): pins 3, 5, 6, 9, 10, and 11. Can provide 8-bit PWM output with the analog Write () function.
- **SPI** (Serial Peripheral Interface): pins 10 (SS), 11 (MOSI), 12 (MISO), and 13 (SCK). These pins support SPI communication using the SPI library.
- **TWI** (two-wire interface) / I²C: pin SDA (A4) and pin SCL (A5). Support TWI communication using the Wire library.
- **AREF** (analog reference): Reference voltage for the analog inputs.

Steps to Install Arduino IDE on Windows:

1. Download the latest version of Arduino IDE for Windows from Arduino.cc website
2. Double click the downloaded exe file, latest version: **arduino_ide_2.1.0.windows_64bit.exe**
3. Click on ‘I Agree’ to accept terms and conditions
4. Select the user and click Next
5. Click on ‘I Agree’ for License Agreement
6. Select the destination folder for installation and click Install
7. Once installation is complete, click Finish and Run IDE
8. Click on ‘Allow Access’ to the Firewall Features.
9. Once the IDE is open, for Windows, few pop ups for Drive Installs come up, install all

Now IDE is ready to be used by plugging in the Arduino Uno and programs written in IDE.