VISVESVARAYA TECHNOLOGICAL UNIVERSITY
BELAGAVI - 590018

A Practical Assessment Report

on

"INTERNET OF THINGS" (18CS81)

Submitted in partial fulfillment of the requirements for the Internet Of Things (18CS81)

BACHELOR OF ENGINEERING
IN
COMPUTER SCIENCE AND ENGINEERING

Submitted by

MAHANTESH M S **1JT19CS047**

Under the Guidance of

Mrs. Nayana Bhat
Assistant Professor
Department of Computer Science and Engineering

Department of Computer Science and Engineering
Accredited By NBA, New Delhi

Jyothy Institute of Technology
Tataguni, Off Kanakapura Road, Bangalore-560 082
Academic Year 2022-2023

JYOTHY INSTITUTE OF TECHNOLOGY

Department of Computer Science and Engineering
Accredited by NBA, New Delhi
Tataguni, off. Kanakapura Road,
Bengaluru – 560082

Institution Vision & Mission

Vision of the Institution

To be an institution of excellence in Engineering education, Innovation and Research and work towards evolving great leaders for the country's future and meeting global needs.

Mission of the Institution

The Institution aims at providing a vibrant, intellectually and emotionally rich teaching learning environment with state of art infrastructure and recognizing and nurturing the potential of each individual to evolve into one's own self and contribute to the welfare of all.

Department of CS&E Vision & Mission

Vision of the Department

To be a center of excellence in Computer Science and Engineering education, focus on research, innovation and entrepreneurial skill development with professional competency.

Mission of the Department

M1: To provide state of the art ICT infrastructure and innovative, research-oriented teaching learning environment and motivation for self-learning & problem-solving abilities by recruiting committed faculty.
M2: To Encourage Industry Institute Interaction & multi-disciplinary approach to problem solving and adapt to the ever-changing global IT trends.
M3: To Imbibe awareness on societal responsibility and leadership qualities with professional competency and ethics.

JYOTHY INSTITUTE OF TECHNOLOGY

Department of Computer Science and Engineering

Accredited by NBA, New Delhi

Tataguni, Off. Kanakapura Road,

Bengaluru - 560 082

C E R T I F I C A T E

This is to certify that the **Practical Assessment** on '**INTERNET OF THINGS (18CS81)**' carried out by **MAHANTESH M S [1JT19CS047]** is bonafede student of Department of Computer Science and Engineering, Jyothy Institute of Technology, in partial fulfilment for the award of **Bachelor of Engineering**, in **Computer Science and Engineering** under Visvesvaraya Technological University, Belagavi, during the year **2022-2023**. It is certified that all corrections/suggestions indicated have been incorporated in the report. The practical assessment report has been approved as it satisfies the academic requirements in respect of Internet of Things prescribed for the course Internet of Things (18CS81).

Signature of Guide	Signature of HOD	Signature of Principal
Mrs. Nayana Bhat	Dr. S Prabhanjan	Dr. Gopalakrishna K
Assistant Professor	Prof. & Head	Principal
Dept. of CSE	Dept. of CSE	JIT, Bangalore
JIT, Bangalore	JIT, Bangalore	

Marks :

Signature with Date

ACKNOWLEDGEMENT

This Practical Assessment Report is a result of accumulated guidance, direction and support of several important persons. We take this opportunity to express our gratitude to all who have helped us to complete the Practical Assessment of Internet of Things(18CS81).

We would like to thank our parents and friends without whose constant help, the completion of Practical Assessment would not been possible.

We would like to thank our Principal, **Dr. Gopalakrishna K**, for providing us adequate facilities to undertake this Practical Assessment of Internet of Things.

We would like to thank **Dr. Prabhanjan S**, Professor and Head, Department of Computer science and Engineering for providing us an opportunity and for his valuable support.

We express our deep and profound gratitude to our Guide **Mrs. Nayana Bhat**, Assistant Prof. Computer Science Department, JIT for her encouragement, support, and guidance during the process of Practical Assessment of Internet of Things.

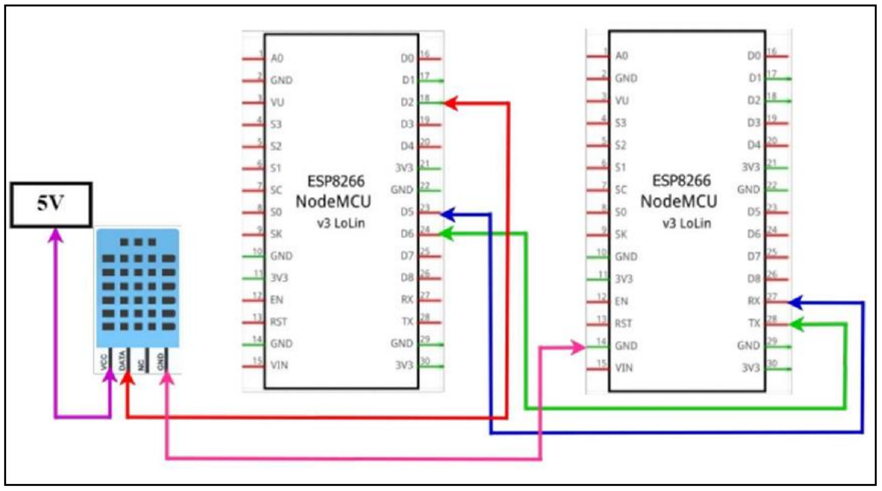
We would like to take this opportunity to express our gratitude for the support and guidance extended to us by the faculty members of the Computer Science and Engineering Department.

We would also like to take this opportunity to express our gratitude for the support and guidance extended to us by the non-teaching faculty members of the Computer Science and Engineering Department.

MAHANTESH M S **[1JT19CS047]**

TABLE OF CONTENTS

Experiment 1: Transmit a string using UART

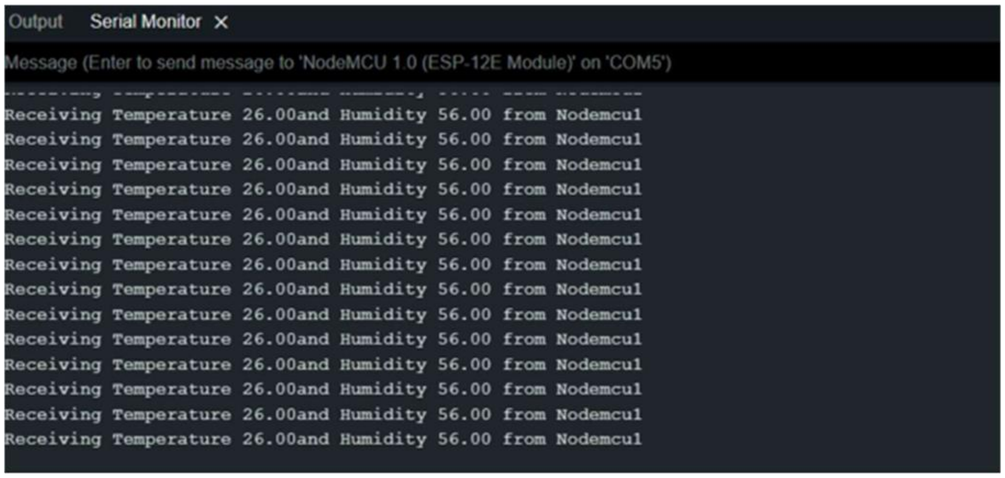


```
#include <DFRobot_DHT11.h>
#include <SoftwareSerial.h>
DFRobot_DHT11 DHT;
#define DHT11_PIN D2
// Rx to Rx and Tx to Tx
// D5 to Rx of Nodemcu and D6 to Tx of Nodemcu
SoftwareSerial espSerial(D5, D6);
String  serial_data;
float temp, humid;
void setup()
{
 // Baud rate is important for uart communication
 Serial.begin(115200);
 // Set the baud rate for custom serial pins and transmit serial data to
//another nodemcu
 espSerial.begin(115200);
}
void loop()
```

```
{
 // Read DHT11_PIN to access temperature and humidity values
 DHT.read(DHT11_PIN);
 // Store temperature value to temp variable
 temp = DHT.temperature;
 // Store humidity value to temp variable
 humid = DHT.humidity;
 // Print temp and humidity data and display it on serial monitor
 Serial.print("Temp:");
 Serial.print(temp);
 Serial.print(" humi:");
 Serial.println(humid);

 // Send temp and humidity data serially to another nodemcu
 serial_data = "Receiving Temperature " + String(temp) + "and Humidity "
+ String(humid) + " from Nodemcu1";
 espSerial.println(serial_data);
 delay(1000);
}
```

Output:


Output
Serial Monitor
Message (Enter to send message to 'NodeMCU 1.0 (ESP-12E Module)' on 'COM5')

```
Receiving Temperature 26.00and Humidity 56.00 from Nodemcu1
Receiving Temperature 26.00and Humidity 56.00 from Nodemcu1
Receiving Temperature 26.00and Humidity 56.00 from Nodemcu1
Receiving Temperature 26.00and Humidity 56.00 from Nodemcu1
Receiving Temperature 26.00and Humidity 56.00 from Nodemcu1
Receiving Temperature 26.00and Humidity 56.00 from Nodemcu1
Receiving Temperature 26.00and Humidity 56.00 from Nodemcu1
Receiving Temperature 26.00and Humidity 56.00 from Nodemcu1
Receiving Temperature 26.00and Humidity 56.00 from Nodemcu1
Receiving Temperature 26.00and Humidity 56.00 from Nodemcu1
Receiving Temperature 26.00and Humidity 56.00 from Nodemcu1
Receiving Temperature 26.00and Humidity 56.00 from Nodemcu1
Receiving Temperature 26.00and Humidity 56.00 from Nodemcu1
Receiving Temperature 26.00and Humidity 56.00 from Nodemcu1
```

<u>Experiment 2: Point-to-Point communication of two Motes over the radio frequency.</u>

NodeMCU Transmitter Code:

```
#include "painlessMesh.h"
// MESH_PREFIX and MESH_PASSWORD can be set to anything you like
#define MESH_PREFIX "nodemcup2p"
#define MESH_PASSWORD "123456789"
// The MESH_PORT refers to the the TCP port that you want the mesh server to run on. The
default
is 5555.
#define MESH_PORT 5555
//Instead of delay it is recommended to use TaskScheduler to run your tasks which is used in
painlessMesh itself.
Scheduler userScheduler;
painlessMesh mesh;
void sendMessage() ;
//Create a task called taskSendMessage responsible for calling the sendMessage() function
every
second as long as the program is running.
Task taskSendMessage( TASK_SECOND * 1 , TASK_FOREVER, &sendMessage );
// The sendMessage() function sends a message to all nodes in the message network
(broadcast).
void sendMessage()
{
 String msg = "LED ON";
 msg += mesh.getNodeId(); // Chip ID
 // To broadcast a message, simply use the sendBroadcast() method on the mesh object and
pass as
argument the message (msg) you want to send.
 mesh.sendBroadcast( msg );
 // Every time a new message is sent, the code changes the interval between messages (1 to 5
seconds).
 taskSendMessage.setInterval( random( TASK_SECOND * 1, TASK_SECOND * 5 ));
}
// The receivedCallback() function prints the message sender (from) and the content of the
message
(msg.c_str()).
void receivedCallback( uint32_t from, String &msg )
{
 Serial.printf("Received from %u msg=%s\n", from, msg.c_str());
}
//The newConnectionCallback() function runs whenever a new node joins the network. This
function simply prints the chip ID of the new node. You can modify //the function to do any
other
task.
void newConnectionCallback(uint32_t nodeId)
{
 Serial.printf("--> startHere: New Connection, nodeId = %u\n", nodeId);
}
// The changedConnectionCallback() function runs whenever a connection changes on the
network
(when a node joins or leaves the network).
void changedConnectionCallback()
{
 Serial.printf("Changed connections\n");
}
// The nodeTimeAdjustedCallback() function runs when the network adjusts the time, so that
all
nodes are synchronized. It prints the offset.
void nodeTimeAdjustedCallback(int32_t offset)
{
 Serial.printf("Adjusted time %u. Offset = %d\n", mesh.getNodeTime(),offset);
}
void setup()
{
 Serial.begin(115200);
 //mesh.setDebugMsgTypes( ERROR | MESH_STATUS | CONNECTION | SYNC |
COMMUNICATION | GENERAL | MSG_TYPES | REMOTE ); // all types on
 // Debug message types
 mesh.setDebugMsgTypes( ERROR | STARTUP ); // set before init() so that you can see
startup
messages
 // Initialize painlessMesh network
 mesh.init( MESH_PREFIX, MESH_PASSWORD, &userScheduler, MESH_PORT );
 // messages received from painless mesh network
 mesh.onReceive(&receivedCallback);
 // The newConnectionCallback() function runs whenever a new node joins the network. This
function simply prints the chip ID of the new node. You can
 //modify the function to do any other task.
 mesh.onNewConnection(&newConnectionCallback);
 // The changedConnectionCallback() function runs whenever a connection changes on the
network
(when a node joins or leaves the network).
 mesh.onChangedConnections(&changedConnectionCallback);
 // The nodeTimeAdjustedCallback() function runs when the network adjusts the time, so that
all
nodes are synchronized. It prints the offset.
 mesh.onNodeTimeAdjusted(&nodeTimeAdjustedCallback);
 // This is what triggers task to broadcast message at regular interval
 userScheduler.addTask( taskSendMessage );
 // Finally, enable the taskSendMessage, so that the program starts sending the messages to
the
```

```
mesh.
 taskSendMessage.enable();
}
void loop()
{
 // To keep the mesh running, add mesh.update() to the loop().
 mesh.update();
}
```

NodeMCU Receiver Code:

```
#include "painlessMesh.h"
// MESH_PREFIX and MESH_PASSWORD can be set to anything you like
#define MESH_PREFIX "nodemcup2p"
#define MESH_PASSWORD "123456789"
// The MESH_PORT refers to the the TCP port that you want the mesh server to run on. The default
is 5555.
#define MESH_PORT 5555
const int led = D0;
//Instead of delay it is recommended to use TaskScheduler to run your tasks which is used in
painlessMesh itself.
Scheduler userScheduler;
painlessMesh mesh;
void sendMessage() ;
//Create a task called taskSendMessage responsible for calling the sendMessage() function every
second as long as the program is running.
Task taskSendMessage( TASK_SECOND * 1 , TASK_FOREVER, &sendMessage );
// The sendMessage() function sends a message to all nodes in the message network (broadcast).
void sendMessage()
```

```
{
 String msg = "Hello from Nodemcu2 ";
 msg += mesh.getNodeId(); // Chip ID
 // To broadcast a message, simply use the sendBroadcast() method on the mesh object and
pass as
argument the message (msg) you want to send.
 mesh.sendBroadcast( msg );
 // Every time a new message is sent, the code changes the interval between messages (1 to 5
seconds).
 taskSendMessage.setInterval( random( TASK_SECOND * 1, TASK_SECOND * 5 ));
}
// The receivedCallback() function prints the message sender (from) and the content of the
message
(msg.c_str()).
void receivedCallback( uint32_t from, String &msg )
{
 if(msg.indexOf("LED ON")!=-1)
  {
 // Goes to deep sleep for 2 seconds(2uS)
 ESP.deepSleep(2e6);
 // Wakes up after 2 seconds deep sleep
 digitalWrite(led, HIGH);
  }
 Serial.printf("Received from %u msg=%s\n", from, msg.c_str());
}
//The newConnectionCallback() function runs whenever a new node joins the network. This
function simply prints the chip ID of the new node. You can modify //the function to do any
other
task.
void newConnectionCallback(uint32_t nodeId)
{
 Serial.printf("--> startHere: New Connection, nodeId = %u\n", nodeId);
}
// The changedConnectionCallback() function runs whenever a connection changes on the
network
(when a node joins or leaves the network).
void changedConnectionCallback()
{
 Serial.printf("Changed connections\n");
}
// The nodeTimeAdjustedCallback() function runs when the network adjusts the time, so that
all
nodes are synchronized. It prints the offset.
void nodeTimeAdjustedCallback(int32_t offset)
{
 Serial.printf("Adjusted time %u. Offset = %d\n", mesh.getNodeTime(),offset);
}
void setup()
{
 Serial.begin(115200);
 pinMode(led, OUTPUT);

 //mesh.setDebugMsgTypes( ERROR | MESH_STATUS | CONNECTION | SYNC |
COMMUNICATION | GENERAL | MSG_TYPES | REMOTE ); // all types on
 // Debug message types
 mesh.setDebugMsgTypes( ERROR | STARTUP );
 // Initialize painlessMesh network
 mesh.init( MESH_PREFIX, MESH_PASSWORD, &userScheduler, MESH_PORT );
 // messages received from painless mesh network
 mesh.onReceive(&receivedCallback);
 // The newConnectionCallback() function runs whenever a new node joins the network. This
```

function simply prints the chip ID of the new node. You can

//modify the function to do any other task.

mesh.onNewConnection(&newConnectionCallback);

// The changedConnectionCallback() function runs whenever a connection changes on the network

(when a node joins or leaves the network).

mesh.onChangedConnections(&changedConnectionCallback);

// The nodeTimeAdjustedCallback() function runs when the network adjusts the time, so that all

nodes are synchronized. It prints the offset.

mesh.onNodeTimeAdjusted(&nodeTimeAdjustedCallback);

// This is what triggers task to broadcast message at regular interval

userScheduler.addTask(taskSendMessage);

// Finally, enable the taskSendMessage, so that the program starts sending the messages to the

mesh.

taskSendMessage.enable();

}

void loop()

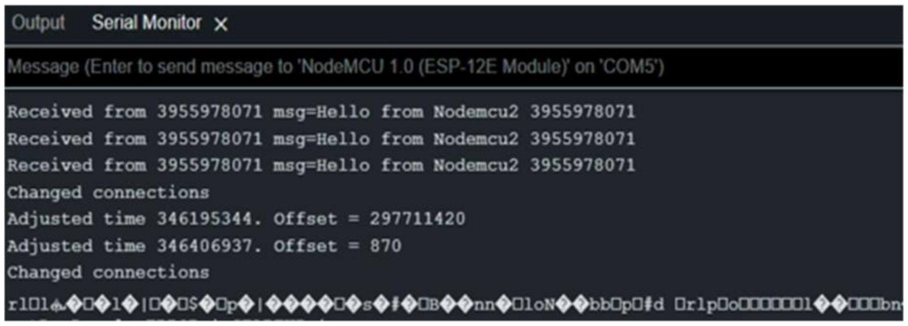
{

// To keep the mesh running, add mesh.update() to the loop().

mesh.update();

}

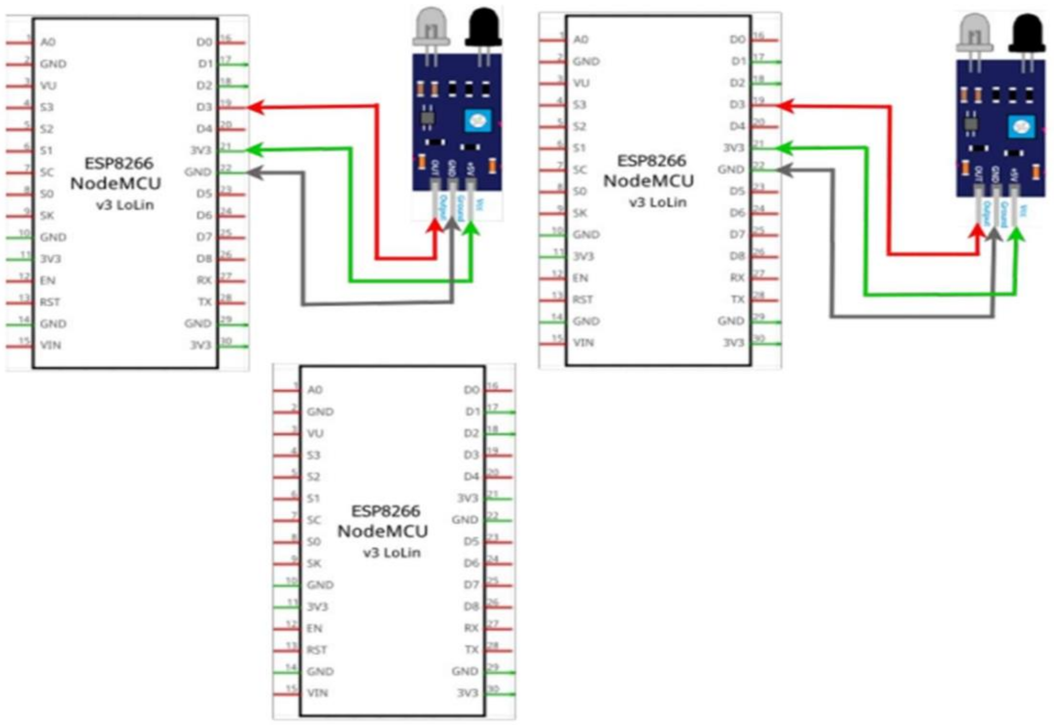
Output:



Output
Serial Monitor
Message (Enter to send message to 'NodeMCU 1.0 (ESP-12E Module)' on 'COM5')

```
Received from 3955978071 msg=Hello from Nodemcu2 3955978071
Received from 3955978071 msg=Hello from Nodemcu2 3955978071
Received from 3955978071 msg=Hello from Nodemcu2 3955978071
Changed connections
Adjusted time 346195344. Offset = 297711420
Adjusted time 346406937. Offset = 870
Changed connections
rl□l�□�l�|□�□$�□p�|�����□�s�#�□B��nn�□loN��bb□p□#d □rlp□o□□□□□□l��□□□bn
```

Experiment 3: Multi-point to single point communication of Motes over the radio frequency.LAN (Subnetting).



NodeMCU Transmitter 1 Code:

```
#include "painlessMesh.h"

#define MESH_PREFIX "nodemcump2p"

#define MESH_PASSWORD "123456789"

#define MESH_PORT 5555

Scheduler userScheduler;

painlessMesh mesh;

const int ir_pin = D3;

int ir_state = 0;

void sendMessage() ;

Task taskSendMessage( TASK_SECOND * 1 , TASK_FOREVER, &sendMessage );

void sendMessage()

{

if (ir_state == RISING)
```

```
{
String msg = "Hi, This is a message from nodemcu1";
mesh.sendBroadcast( msg );
taskSendMessage.setInterval( random( TASK_SECOND * 1, TASK_SECOND
* 5 ));
}
}
void receivedCallback( uint32_t from, String &msg )
{
Serial.printf("startHere: Received from %u msg=%s\n", from,
msg.c_str());
}
void newConnectionCallback(uint32_t nodeId)
{
Serial.printf("--> startHere: New Connection, nodeId = %u\n",
nodeId);
}
void changedConnectionCallback()
{
Serial.printf("Changed connections\n");
}
void nodeTimeAdjustedCallback(int32_t offset)
{
Serial.printf("Adjusted time %u. Offset = %d\n", mesh.getNodeTime(),
offset);
}
void setup()
{
Serial.begin(115200);
pinMode(ir_pin, INPUT);
```

```
//mesh.setDebugMsgTypes( ERROR | MESH_STATUS | CONNECTION | SYNC |
COMMUNICATION | GENERAL | MSG_TYPES | REMOTE ); // all types on
mesh.setDebugMsgTypes( ERROR | STARTUP ); // set before init() so
that you can see startup messages
mesh.init( MESH_PREFIX, MESH_PASSWORD, &userScheduler, MESH_PORT );
mesh.onReceive(&receivedCallback);
mesh.onNewConnection(&newConnectionCallback);
mesh.onChangedConnections(&changedConnectionCallback);
mesh.onNodeTimeAdjusted(&nodeTimeAdjustedCallback);
userScheduler.addTask( taskSendMessage );
taskSendMessage.enable();
}
void loop()
{
ir_state = digitalRead(ir_pin);
// it will run the user scheduler as well
mesh.update();
}
```

NodeMCU Transmitter 2 Code:

```
#include "painlessMesh.h"
#define MESH_PREFIX "nodemcump2p"
#define MESH_PASSWORD "123456789"
#define MESH_PORT 5555
Scheduler userScheduler;
painlessMesh mesh;
const int ir_pin = D3;
int ir_state = 0;
void sendMessage() ;
Task taskSendMessage( TASK_SECOND * 1 , TASK_FOREVER, &sendMessage );
void sendMessage()
```

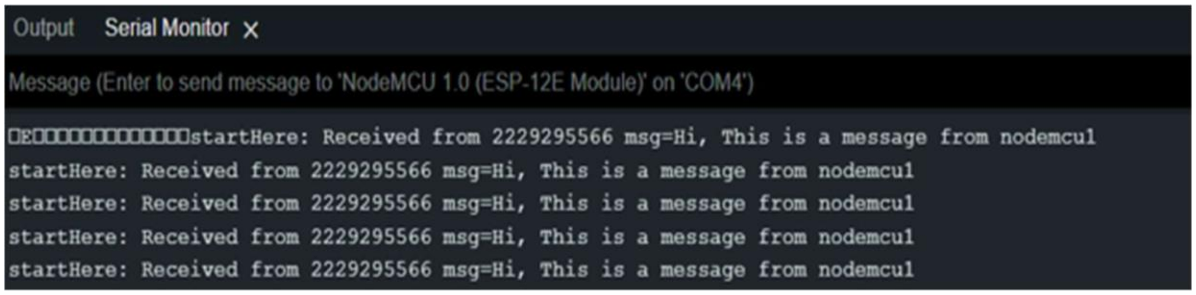
```
{
if (ir_state == RISING)
{
String msg = "Hi, This is a message from nodemcu2";
mesh.sendBroadcast( msg );
taskSendMessage.setInterval( random( TASK_SECOND * 1, TASK_SECOND
* 5 ));
}
}
void receivedCallback( uint32_t from, String &msg )
{
Serial.printf("startHere: Received from %u msg=%s\n", from,
msg.c_str());
}
void newConnectionCallback(uint32_t nodeId)
{
Serial.printf("--> startHere: New Connection, nodeId = %u\n",
nodeId);
}
void changedConnectionCallback()
{
Serial.printf("Changed connections\n");
}
void nodeTimeAdjustedCallback(int32_t offset)
{
Serial.printf("Adjusted time %u. Offset = %d\n", mesh.getNodeTime(),
offset);
}
void setup()
{
```

```
Serial.begin(115200);
pinMode(ir_pin, INPUT);
//mesh.setDebugMsgTypes( ERROR | MESH_STATUS | CONNECTION | SYNC |
COMMUNICATION | GENERAL | MSG_TYPES | REMOTE ); // all types on
mesh.setDebugMsgTypes( ERROR | STARTUP ); // set before init() so
that you can see startup messages
mesh.init( MESH_PREFIX, MESH_PASSWORD, &userScheduler, MESH_PORT );
mesh.onReceive(&receivedCallback);
mesh.onNewConnection(&newConnectionCallback);
mesh.onChangedConnections(&changedConnectionCallback);
mesh.onNodeTimeAdjusted(&nodeTimeAdjustedCallback);
userScheduler.addTask( taskSendMessage );
taskSendMessage.enable();
}
void loop()
{
ir_state = digitalRead(ir_pin);
// it will run the user scheduler as well
mesh.update();
}
```

NodeMCU Receiver Code:

```
#include "painlessMesh.h"
#define MESH_PREFIX "nodemcup2p"
#define MESH_PASSWORD "123456789"
#define MESH_PORT 5555
void receivedCallback( uint32_t from, String &msg );
painlessMesh mesh;
void setup()
{
Serial.begin(115200);
mesh.setDebugMsgTypes( ERROR | STARTUP | CONNECTION );
// set before init() so that you can see startup messages
mesh.init( MESH_PREFIX, MESH_PASSWORD, MESH_PORT );
mesh.onReceive(&receivedCallback);
}
void loop()
{
mesh.update();
}
void receivedCallback( uint32_t from, String &msg )
{
Serial.printf("echoNode: Received from %u msg=%s\n", from,
msg.c_str());
mesh.sendSingle(from, msg);
}
```

Output:


Output
Serial Monitor ×
Message (Enter to send message to 'NodeMCU 1.0 (ESP-12E Module)' on 'COM4')

```
□E□□□□□□□□□□□□startHere: Received from 2229295566 msg=Hi, This is a message from nodemcu1
startHere: Received from 2229295566 msg=Hi, This is a message from nodemcu1
startHere: Received from 2229295566 msg=Hi, This is a message from nodemcu1
startHere: Received from 2229295566 msg=Hi, This is a message from nodemcu1
startHere: Received from 2229295566 msg=Hi, This is a message from nodemcu1
```

Experiment 4: I2C Protocol Study

I2C communication is the short form for inter-integrated circuits. It is a communication protocol developed by Philips Semiconductors for the transfer of data between a central processor and multiple ICs on the same circuit board using just two common wires.

Owing to its simplicity, it is widely adopted for communication between microcontrollers and sensor arrays, displays, IoT devices, EEPROMs etc.

This is a type of synchronous serial communication protocol. It means that data bits are transferred one by one at regular intervals of time set by a reference clock line.

The following are some of the important features of I2C communication protocol:

Only two common bus lines (wires) are required to control any device/IC on the I2C network.

No need of prior agreement on data transfer rate like in UART communication. So the data transfer speed can be adjusted whenever required.

Simple mechanism for validation of data transferred.

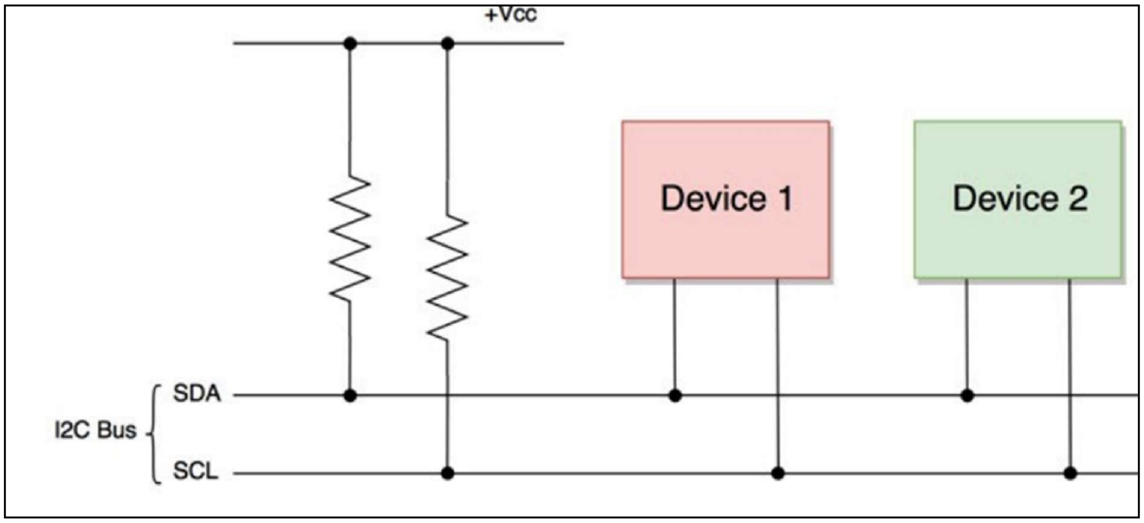
Uses 7-bit addressing system to target a specific device/IC on the I2C bus.

I2C networks are easy to scale. New devices can simply be connected to the two common I2C bus lines.

Hardware

The physical I2C Bus

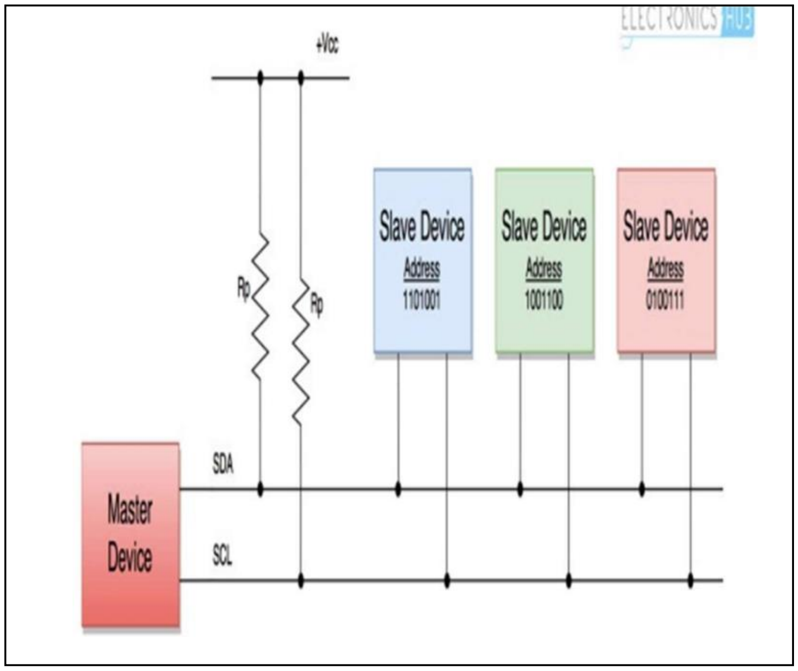
I2C Bus (Interface wires) consists of just two wires and are named as Serial Clock Line (SCL) and Serial Data Line (SDA). The data to be transferred is sent through the SDA wire and is synchronized with the clock signal from SCL. All the devices/ICs on the I2C network are connected to the same SCL and SDA lines as shown below



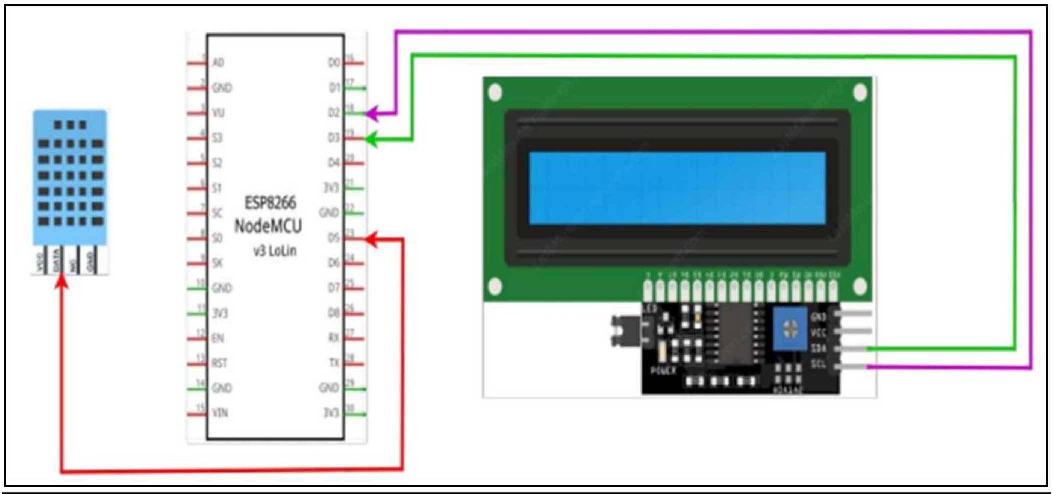
Both the I2C bus lines (SDA, SCL) are operated as open drain drivers. It means that any device/IC on the I2C network can drive SDA and SCL low, but they cannot drive them high. So, a pull up resistor is used for each bus line, to keep them high (at positive voltage) by default. The reason for using an open-drain system is that there will be no chances of shorting, which might happen when one device tries to pull the line high and some other device tries to pull the line low.

Master and Slave Devices

The devices connected to the I2C bus are categorized as either masters or slaves. At any instant of time only a single master stays active on the I2C bus. It controls the SCL clock line and decides what operation is to be done on the SDA data line. All the devices that respond to instructions from this master device are slaves. For differentiating between multiple slave devices connected to the same I2C bus, each slave device is physically assigned a permanent 7-bit address. When a master device wants to transfer data to or from a slave device, it specifies this particular slave device address on the SDA line and then proceeds with the transfer. So effectively communication takes place between the master device and a particular slave device. All the other slave devices doesn't respond unless their address is specified by the master device on the SDA line.

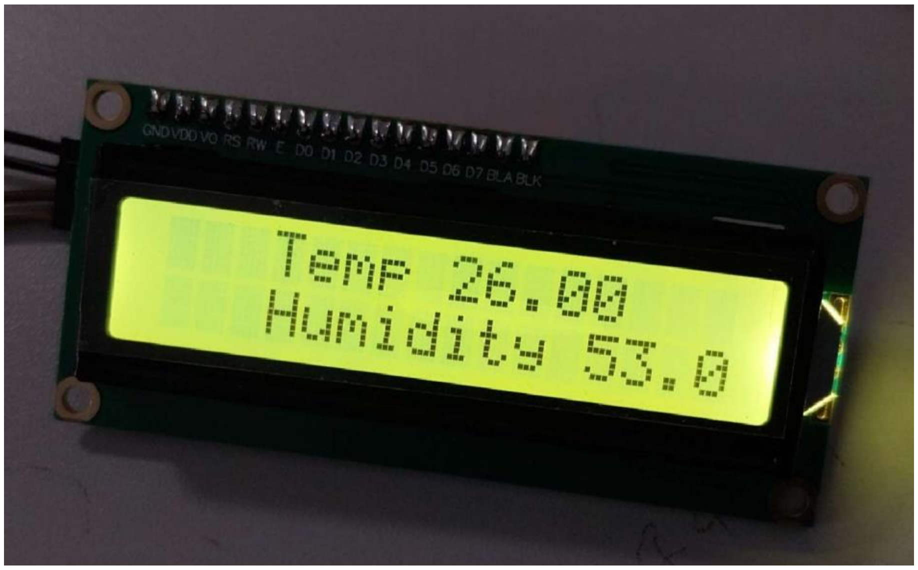


Experiment 5: Reading Temperature and Relative Humidity value from the sensor



```
#include#include#include#include<DFRobot_DHT11.h>
<SoftwareSerial.h>
<Wire.h>
<LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 16, 2);
DFRobot_DHT11 DHT;
#define DHT11_PIN D5
float temp, humid;
void setup()
{
Serial.begin(115200);
lcd.begin();
}
void loop()
{
DHT.read(DHT11_PIN);
temp = DHT.temperature;
humid = DHT.humidity;
Serial.print("Temp:");
Serial.print(temp);
Serial.print(" humi:");
Serial.println(humid);
lcd.clear();
lcd.setCursor(3,0);
lcd.print("Temp ");
lcd.print(temp);
lcd.setCursor(3,1);
lcd.print("Humidity ");
lcd.print(humid);
delay(1000);
}
```

Output:



```
Temp 26.00
Humidity 53.0
```