Sardar Patel Institute of Technology
(Computer Engineering Department)

Internet of Things Lab (OE4)

Lab Experiments

Name: Vishal Salvi
Class: TE (Comps)
UID: 2019230069
IOT Lab Batch: B

EXPERIMENT 1

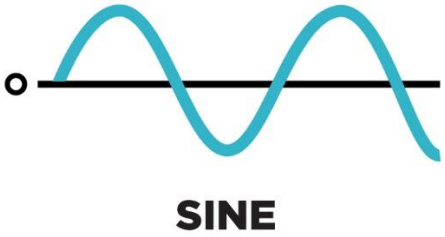
NAME: Vishal Shashikant Salvi **UID: 2019230069**

CLASS: TE COMPS **BATCH: B**

Aim: To analyze the waveform of various signals.

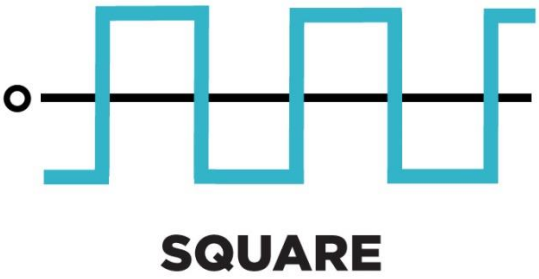
Theory:

Sine Wave



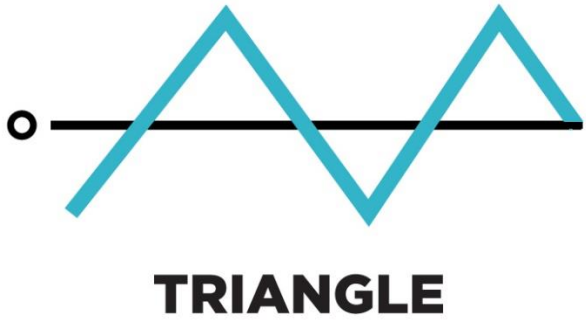
DEFINITION: A sine wave sounds like it looks: smooth and clean. It is sound at its most basic. The sound of a sine wave is only made up of one thing, something known as the fundamental. No partials to be seen! Try whistling one note or imagine the sound of a tuning fork. Those are both approximations of what sine waves sound like, though real-life sine waves are rare.

Square Wave



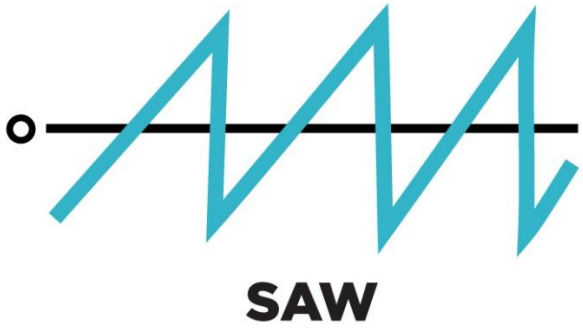
DEFINITION: Remember how a sine wave is only made up of one thing, the fundamental? Not the square wave. A square wave sounds richer and buzzier. It also looks different. These are both because in addition to the fundamental, the square wave also contains **harmonics**. A harmonic is a kind of partial tone which is a whole multiple of a fundamental frequency. In a square wave, these harmonics occur in whole odd-number multiples of the fundamental frequency. The harmonics, combined with the fundamental, give this wave a square shape.

Triangle Wave



DEFINITION: A triangle wave contains the same odd harmonics as a square wave. Unlike a square wave, they taper off as they get further away from the fundamental, giving it its shape. It looks like an angular sine wave, and it sounds somewhere in between a square wave and a sine wave. It's not as buzzy as a square but not as smooth as a sine wave. It sounds clearer, maybe even brighter than a sine wave. Think of a recorder, or a breathily-played flute—that sounds similar to a triangle wave.

Sawtooth Wave



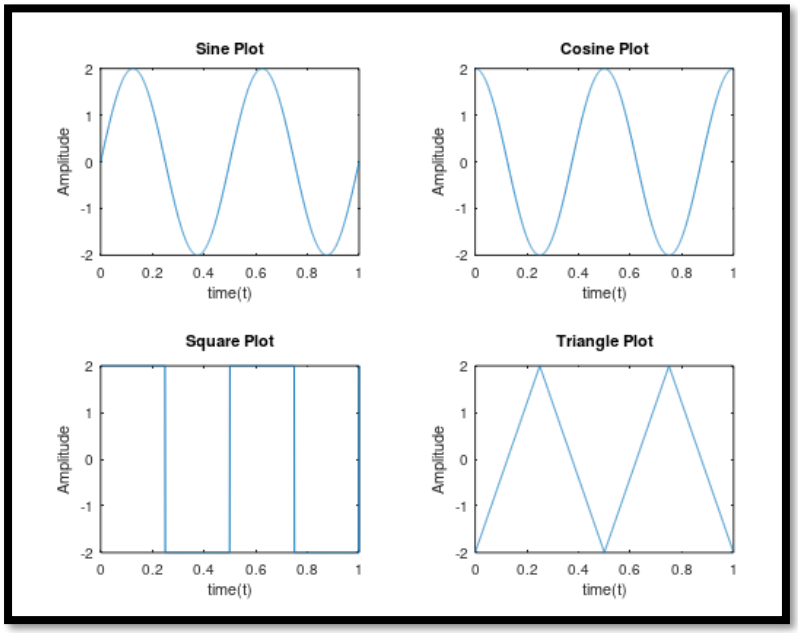
DEFINITION: Also called a saw wave, a sawtooth wave is much more jagged and, well, looks like a saw. It is the buzziest sounding of them all, sounding even harsher than a square wave, and that's because it's the richest in terms of harmonics. This means it can be a really great choice for when you're working with subtractive synthesis, which is when you construct a sound by filtering out frequencies, rather than adding them on.

Input:

```
clc
close all
clear
A =input ('Enter Amplitude:')
F =input ('Enter Frequency:')
t = 0:0.001:1
#sine Plot
y1 = A*sin(2*pi*F*t);
```

```
subplot(2,2,1)
plot(t,y1)
title('Sine Plot')
xlabel('time(t)')
ylabel('Amplitude')
#Cosine Plot
y2 = A*cos(2*pi*F*t);
subplot(2,2,2)
plot(t,y2)
title('Cosine Plot')
xlabel('time(t)')
ylabel('Amplitude')
#Square Plot
y3 = A*square(2*pi*F*t);
subplot(2,2,3)
plot(t,y3)
title('Square Plot')
xlabel('time(t)')
ylabel('Amplitude')
#Triangle Plot
y4 = A*sawtooth(2*pi*F*t,0.5);
subplot(2,2,4)
plot(t,y4)
title('Triangle Plot')
xlabel('time(t)')
ylabel('Amplitude')
```

Output:



Operations on signal:

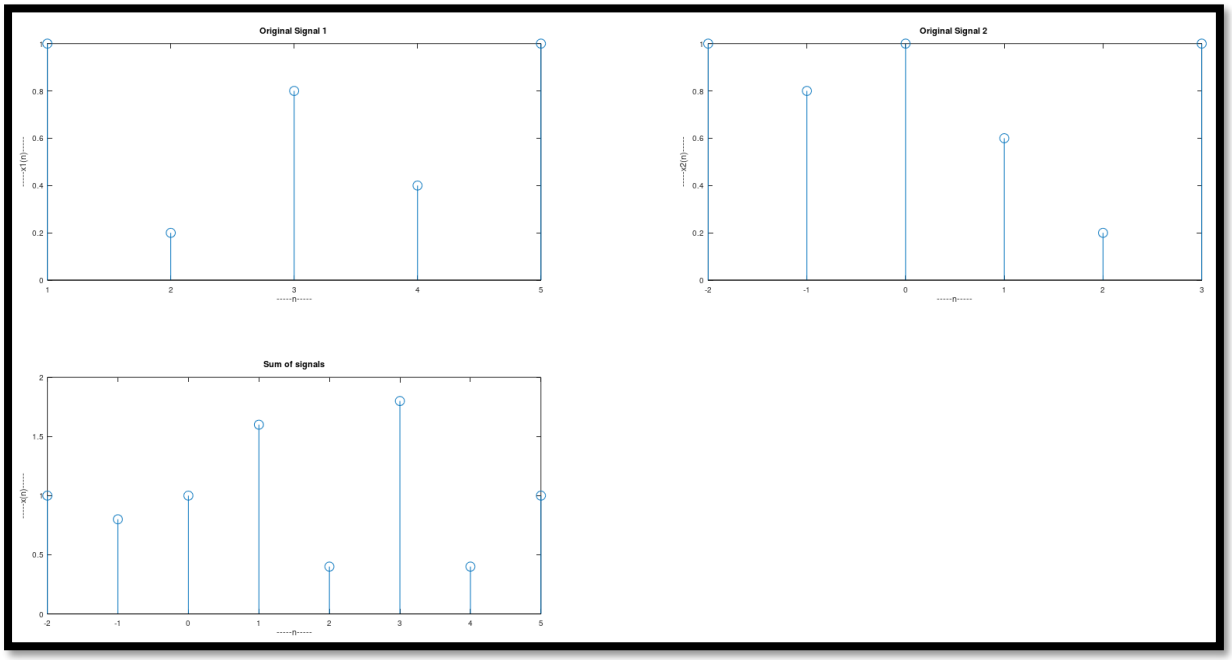
Addition:

```
clc;clear;
x1=[1 0.2 0.8 0.4 1];
n1=1:5

x2=[1 0.8 1 0.6 0.2 1];
n2=-2:3

subplot(2,2,1);
stem(n1,x1);
title('Original Signal 1 ');
xlabel('-----n-----');
ylabel('-----x1(n)-----');

subplot(2,2,2);
stem(n2,x2);
title('Original Signal 2 ');
xlabel('-----n-----');
ylabel('-----x2(n)-----');

[r,n]=signaladdition(x1,x2,n1,n2);

subplot(2,2,3);
stem(n,r);
title('Sum of signals');
xlabel('-----n-----');
ylabel('-----x(n)-----');
```

Function:

```
function[result,n]=signaladdition(x1,x2,n1,n2);
n=min(min(n1),min(n2)):max(max(n1),max(n2));
a=zeros(size(n));
b=a;
a(find(n==min(n1)):find(n==max(n1)))=x1;
b(find(n==min(n2)):find(n==max(n2)))=x2;
result = (a+b);
end
```

Output:



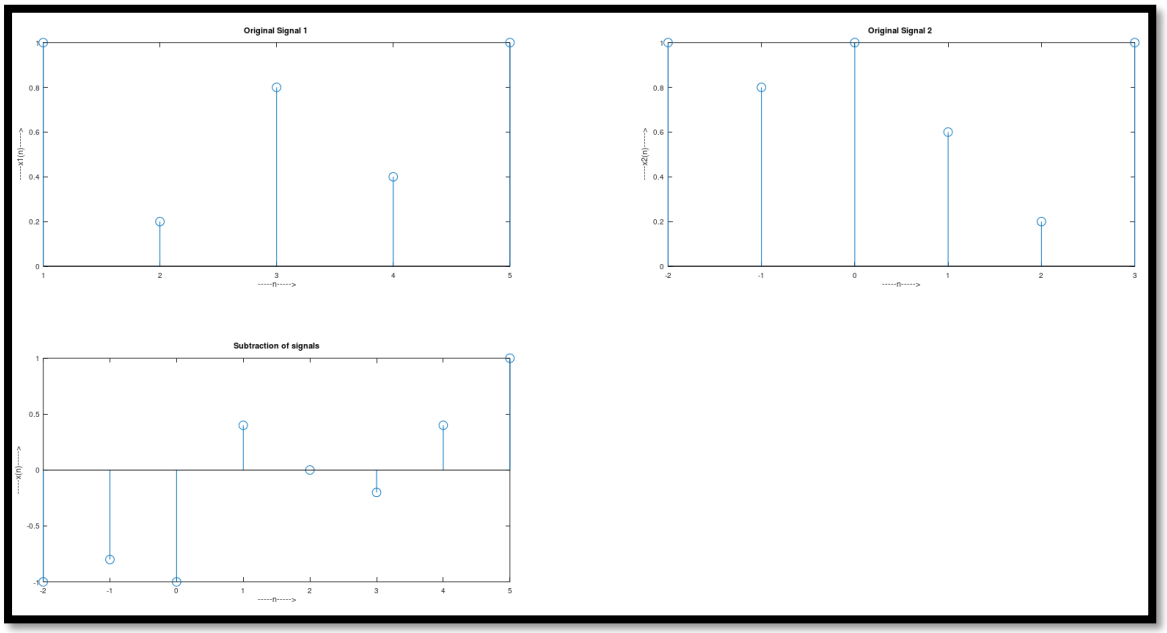
Subtraction:

```
clc;clear;
x1=[1 0.2 0.8 0.4 1];
n1=1:5

x2=[1 0.8 1 0.6 0.2 1];
n2=-2:3

subplot(2,2,1);
stem(n1,x1);
title('Original Signal 1 ');
xlabel('-----n----->');
ylabel('-----x1(n)----->');

subplot(2,2,2);
stem(n2,x2);
title('Original Signal 2 ');
xlabel('-----n----->');
ylabel('-----x2(n)----->');

[r,n]=signaladdition(x1,x2,n1,n2);

subplot(2,2,3);
stem(n,r);
title('Subtraction of signals');
xlabel('-----n----->');
ylabel('-----x(n)----->');
```

Function:

```
function[result,n]=signaladdition(x1,x2,n1,n2);
n=min(min(n1),min(n2)):max(max(n1),max(n2));
a=zeros(size(n));
b=a;
a(find(n==min(n1)):find(n==max(n1)))=x1;
b(find(n==min(n2)):find(n==max(n2)))=x2;
result = (a-b);
end
```

Output:



Multiplication:

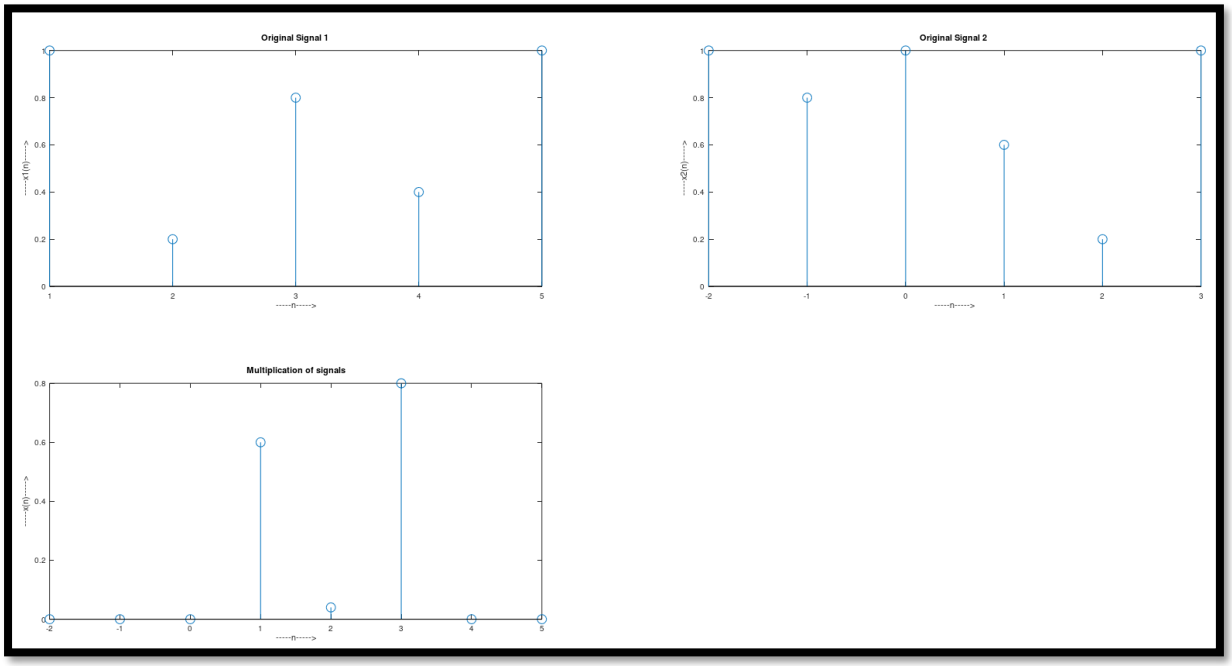
```
clc;clear;
x1=[1 0.2 0.8 0.4 1];
n1=1:5

x2=[1 0.8 1 0.6 0.2 1];
n2=-2:3

subplot(2,2,1);
stem(n1,x1);
title('Original Signal 1 ');
xlabel('-----n----->');
ylabel('-----x1(n)----->');

subplot(2,2,2);
stem(n2,x2);
```

```
title('Original Signal 2 ');
xlabel('-----n----->');
ylabel('-----x2(n)----->');

[r,n]=signaladdition(x1,x2,n1,n2);

subplot(2,2,3);
stem(n,r);
title('Multiplication of signals');
xlabel('-----n----->');
ylabel('-----x(n)----->');
```

Function:

```
function[result,n]=signaladdition(x1,x2,n1,n2);
n=min(min(n1),min(n2)):max(max(n1),max(n2));
a=zeros(size(n));
b=a;
a(find(n==min(n1)):find(n==max(n1)))=x1;
b(find(n==min(n2)):find(n==max(n2)))=x2;
result = (a.*b);
end
```

Output:



Conclusion:

Thus, from this experiment I analyze the waveform of various signals and also studied operations on signal.

EXPERIMENT 2

NAME: Vishal Shashikant Salvi **UID: 2019230069**

CLASS: TE COMPS **BATCH: B**

Aim: To analyse analog signal and convert it to digital signal

Theory:

A signal is an electromagnetic or electrical current that carries data from one system or network to another. In electronics, a signal is often a time-varying voltage that is also an electromagnetic wave carrying information, though it can take on other forms, such as current. There are two main types of signals used in electronics: analog and digital signals.

Analog Signal

An analog signal is time-varying and generally bound to a range (e.g. +12V to -12V), but there is an infinite number of values within that continuous range. An analog signal uses a given property of the medium to convey the signal's information, such as electricity moving through a wire. In an electrical signal, the voltage, current, or frequency of the signal may be varied to represent the information. Analog signals are often calculated responses to changes in light, sound, temperature, position, pressure, or other physical phenomena.

When plotted on a voltage vs. time graph, an analog signal should produce a smooth and continuous curve. There should not be any discrete value changes.

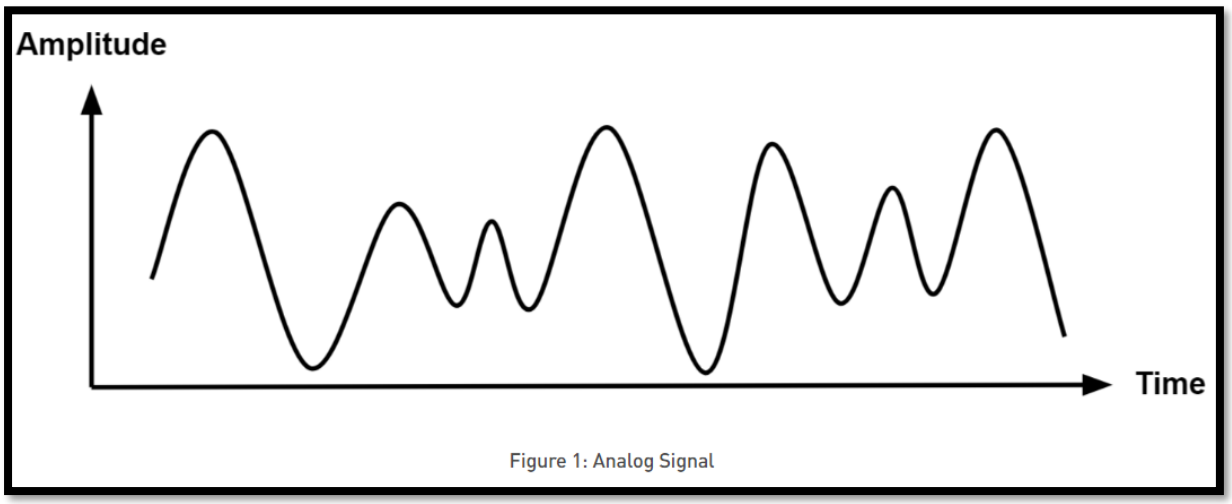


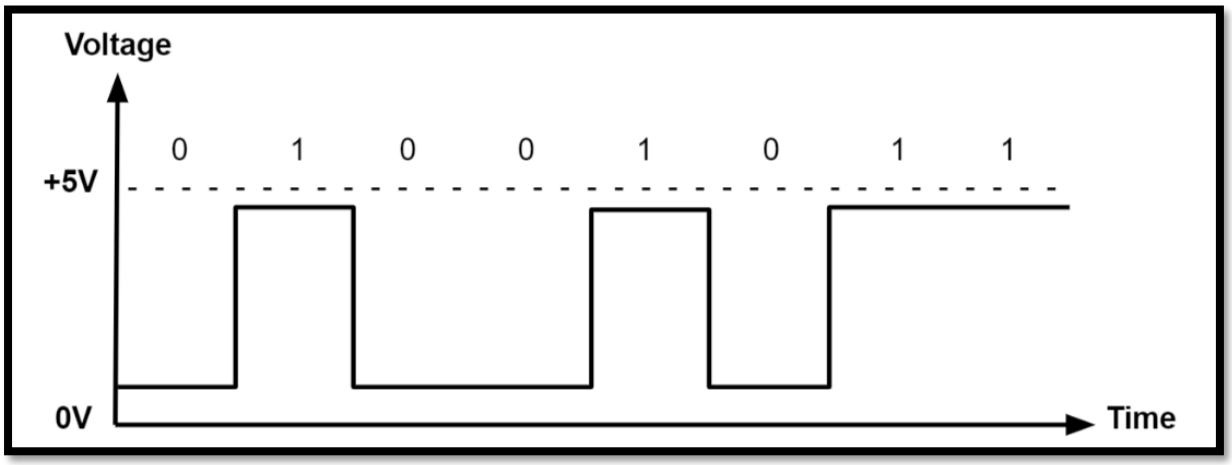
Figure 1: Analog Signal

Digital Signal

A digital signal is a signal that represents data as a sequence of discrete values. A digital signal can only take on one value from a finite set of possible values at a given time. With digital signals, the physical quantity representing the information can be many things:

- Variable electric current or voltage
- Phase or polarization of an electromagnetic field
- Acoustic pressure
- The magnetization of a magnetic storage media

Digital signals are used in all digital electronics, including computing equipment and data transmission devices. When plotted on a voltage vs. time graph, digital signals are one of two values, and are usually between 0V and VCC (usually 1.8V, 3.3V, or 5V).

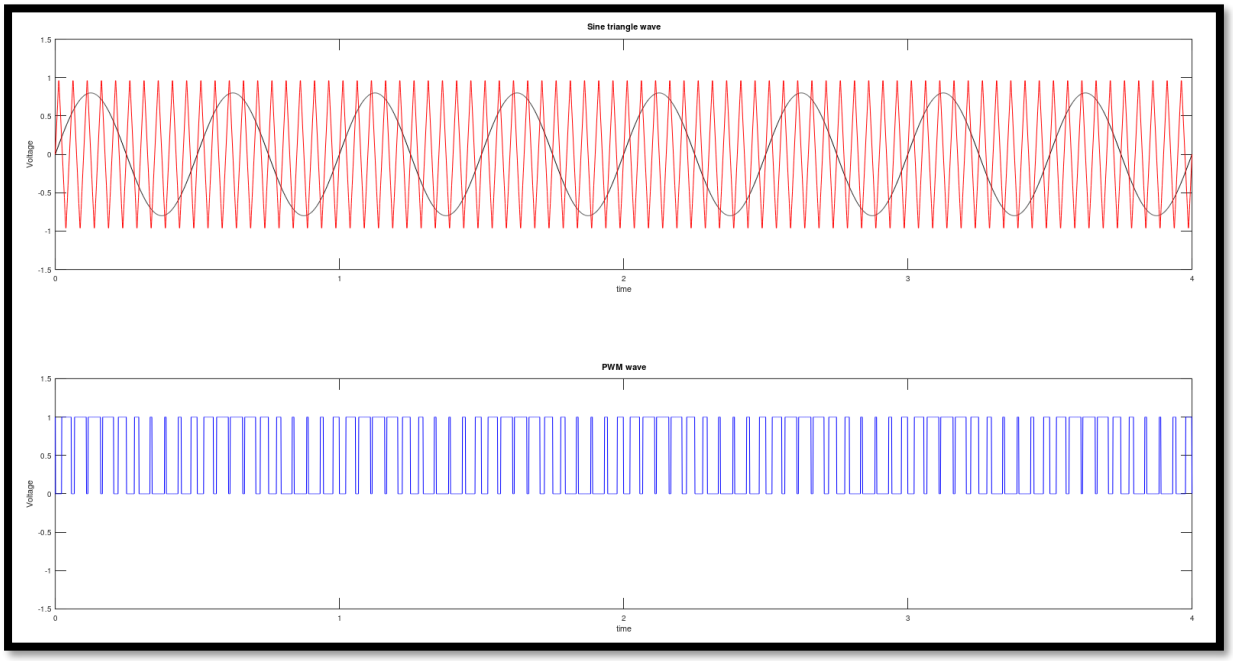


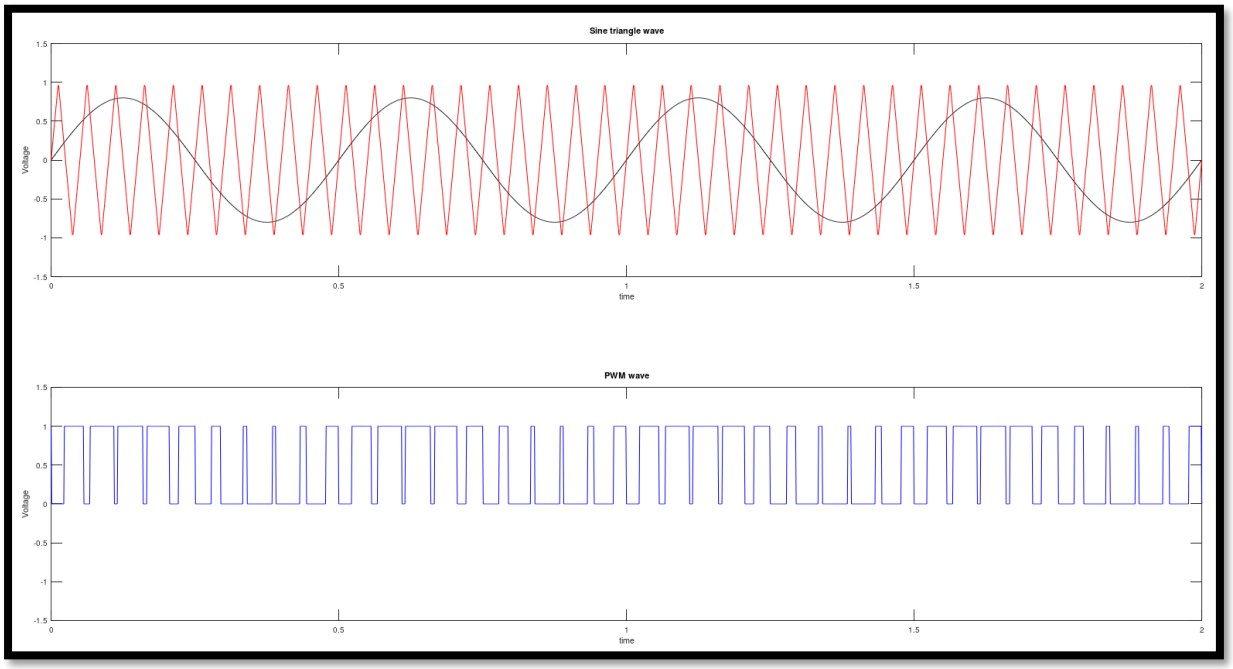
Code:

```
clc
close all
clear
t=0:0.001:4;
s=sawtooth(4*pi*10*t+(pi/2),0.5);
m=0.80*sin(4*pi*1*t);
n=length(s);
for i=1:n
  if(m(i)>=s(i))
  pwm(i)=1;
  elseif(m(i)<=s(i))
  pwm(i)=0;
end
end

#Sine-Triangle wave
subplot(2,1,1)
plot(t,m,'-k',t,s,'-r');
axis([0 4 -1.5 1.5]);
title('Sine triangle wave');
xlabel('time');
ylabel('Voltage');

#PWM Wave
subplot(2,1,2)
plot(t,pwm,'-b');
axis([0 4 -1.5 1.5]);
title('PWM wave');
```

```
xlabel('time');
ylabel('Voltage');
```

Output:





Conclusion:

After completing the above experiment, I have understood the process of converting the given analog signal into digital signal and also understood how to serialize and deserialize the given signal and plotted them using Octave software.

EXPERIMENT 3

NAME: Vishal Shashikant Salvi **UID: 2019230069**

CLASS: TE COMPS **BATCH: B**

Aim: To modulate and demodulate the message signal.

Theory:

What is Signal Modulation?

A message carrying signal has to get transmitted over a distance and for it to establish a reliable communication, it needs to take the help of a high frequency signal which should not affect the original characteristics of the message signal.

The characteristics of the message signal, if changed, the message contained in it also alters. Hence it is a must to take care of the message signal. A high frequency signal can travel up to a longer distance, without getting affected by external disturbances. We take the help of such high frequency signal which is called as a carrier signal to transmit our message signal. Such a process is simply called as Modulation.

Modulation is the process of changing the parameters of the carrier signal, in accordance with the instantaneous values of the modulating signal.

Need for Modulation

The baseband signals are incompatible for direct transmission. For such a signal, to travel longer distances, its strength has to be increased by modulating with a high frequency carrier wave, which doesn't affect the parameters of the modulating signal.

What is Demodulation?

Demodulation is defined as extracting the original information-carrying signal from a modulated carrier wave. A demodulator is an electronic circuit that is mainly used to recover the information content from the modulated carrier wave. There are different types of modulation and so are demodulators. The output signal via a demodulator may describe the sound, images, or binary data.

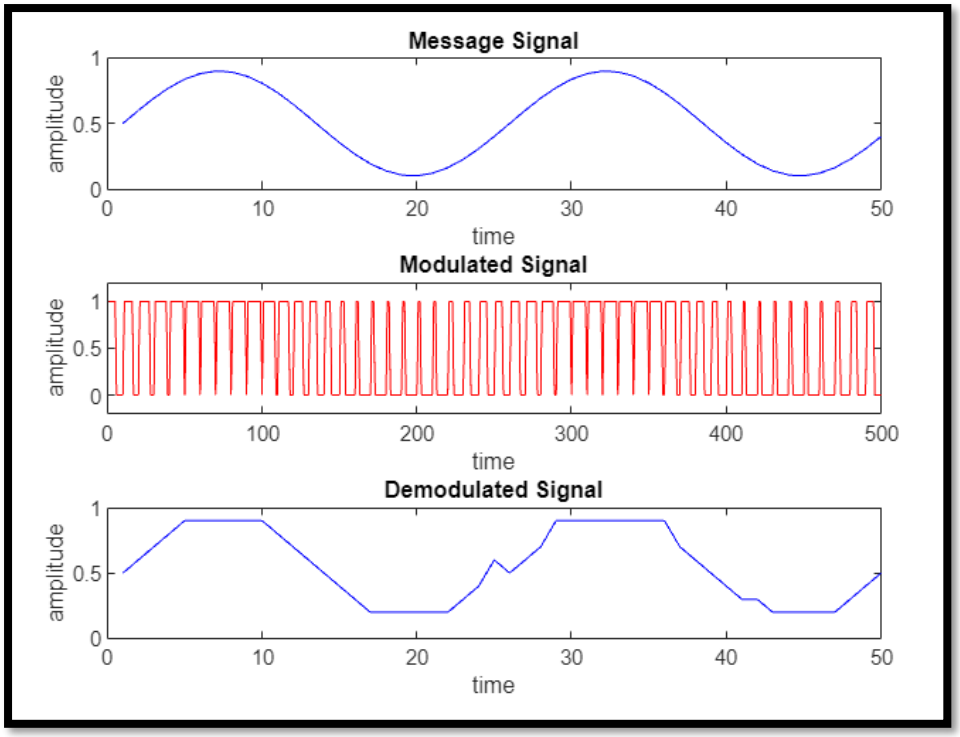
Difference between Modulation and Demodulation

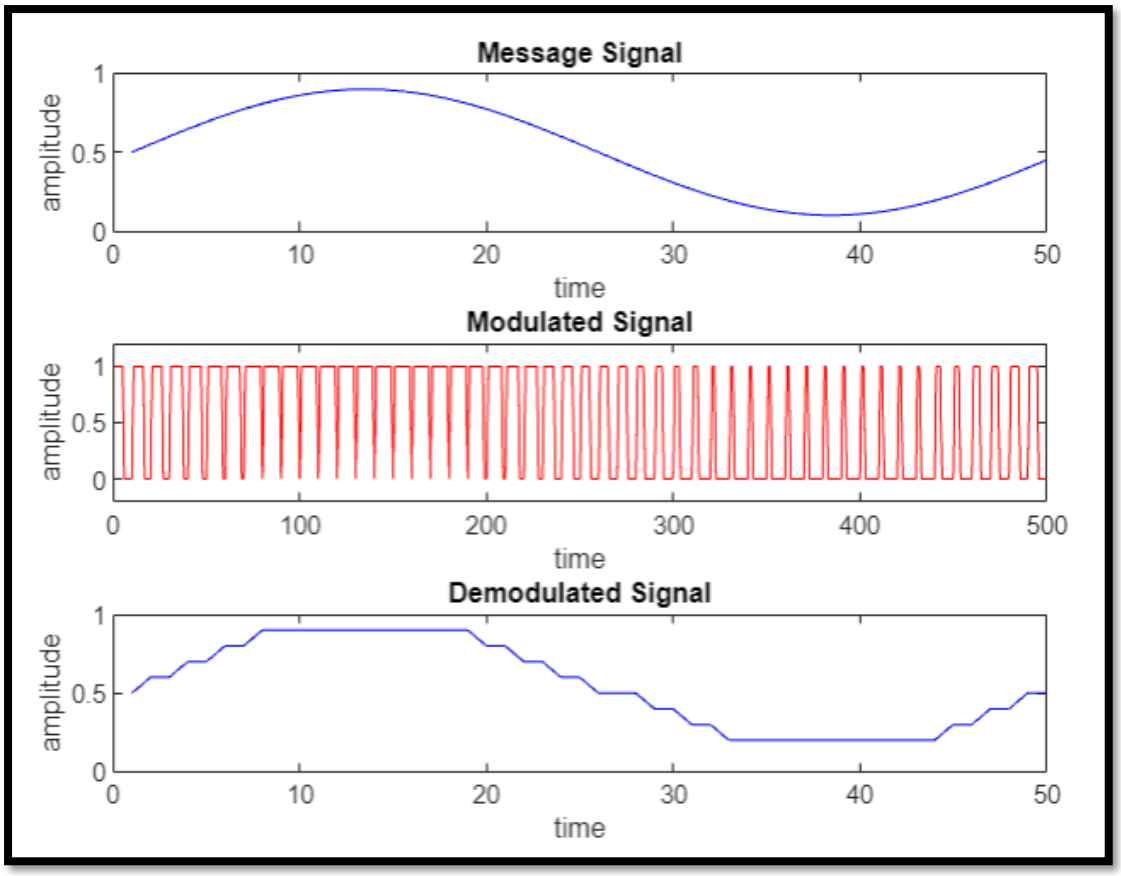
- Modulation is the process of influencing data information on the carrier, while demodulation is the recovery of original information at the distant end from the carrier.
- A modem is an equipment that performs both modulation and demodulation.
- Both processes aim to achieve transfer information with the minimum distortion, minimum loss, and efficient utilisation of spectrum.

Code:

```
carrier_freq=1000;
sampling_freq=10000;
message_freq=200;

t=0:1/sampling_freq:(2/message_freq-1/sampling_freq);
mt=0.4*sin(2*pi*message_freq*t)+0.5;
st= modulate(mt,carrier_freq,sampling_freq,'PWM');
dt= demod(st,carrier_freq,sampling_freq,'PWM');
figure
%message signal
subplot(3,1,1);
plot(mt,'b');
title('Message Signal');
xlabel('time');
ylabel('amplitude');
axis([0 50 0 1])

%modulated signal
subplot(3,1,2);
plot(st,'r');
title('Modulated Signal');
xlabel('time');
ylabel('amplitude');
axis([0 500 -0.2 1.2])

%demodulated signal
subplot(3,1,3);
plot(dt,'b');
title('Demodulated Signal');
xlabel('time');
ylabel('amplitude');
axis([0 50 0 1])
```

Output:





Conclusion:

We learned to use Matlab and performed the experiment of converting the message signal to its modulated and demodulated signal.

EXPERIMENT 4

NAME: Vishal Shashikant Salvi **UID: 2019230069**

CLASS: TE COMPS **BATCH: B**

Aim: To Create a client-server environment where the NodeMCU acts as a client which sends its data to the cloud server here in this case ThingSpeak is used as a server-side platform.

Theory:

What is NodeMCU?

The NodeMCU (**N**ode **M**icro**C**ontroller **U**nit) is an open-source software and hardware development environment built around an inexpensive System-on-a-Chip (SoC) called the ESP8266. The ESP8266, designed and manufactured by Espressif Systems, contains the crucial elements of a computer: CPU, RAM, networking (WiFi), and even a modern operating system and SDK. That makes it an excellent choice for the Internet of Things (IoT) projects of all kinds.

However, as a chip, the ESP8266 is also hard to access and use. You must solder wires, with the appropriate analog voltage, to its pins for the simplest tasks such as powering it on or sending a keystroke to the "computer" on the chip. You also have to program it in low-level machine instructions that can be interpreted by the chip hardware. This level of integration is not a problem using the ESP8266 as an embedded controller chip in mass-produced electronics. It is a huge burden for hobbyists, hackers, or students who want to experiment with it in their own IoT projects.

About ThingSpeak

ThingSpeak is an IoT analytics platform service that allows you to aggregate, visualize, and analyze live data streams in the cloud. You can send data to ThingSpeak from your devices, create instant visualization of live data, and send alerts.

Role of Data Analysis in IOT

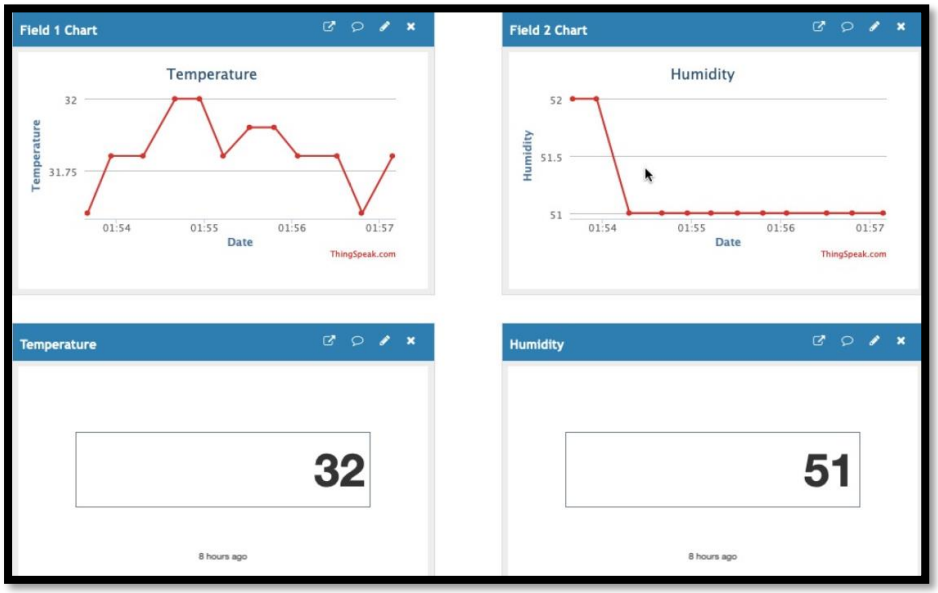
1. IoT and data remain intrinsically linked together. Data consumed and produced keeps growing at an ever-expanding rate. This influx of data is fuelling widespread IoT adoption as there will be nearly 30.73 billion IoT connected devices by 2020.
2. The Internet of Things (IoT) is an interconnection of several devices, networks, technologies, and human resources to achieve a common goal. There are a variety of IoT-based applications being used in different sectors and have succeeded in providing huge benefits to the users.
3. The data generated from IoT devices turns out to be of value only if it gets subjected to analysis, which brings data analytics into the picture. Data Analytics (DA) is defined as a process, which is used to examine big and small data sets with varying data properties to extract meaningful conclusions and actionable insights.

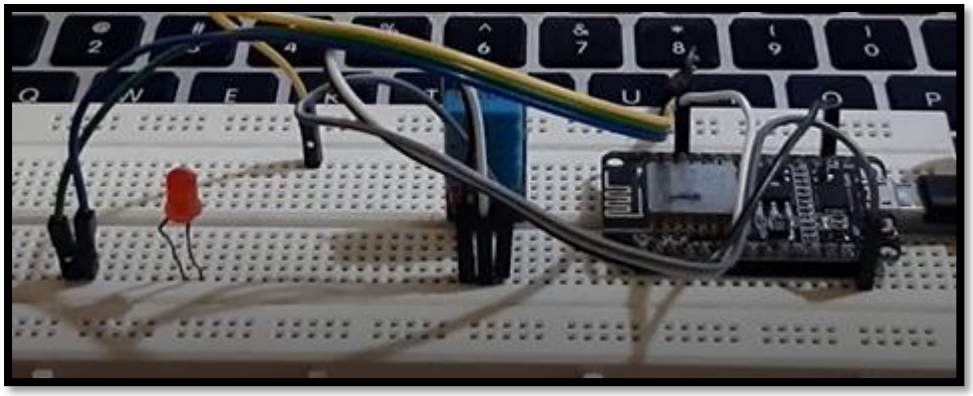
Code:

```
#include <DHT.h>
#include <ESP8266WiFi.h>

String apiKey = "KN0D67T6T7S4A2NO";     //  Enter your Write API key from ThingSpeak

const char *ssid =  "Redmi";     // replace with your wifi ssid and wpa2 key
const char *pass =  "12345678";
const char* server = "api.thingspeak.com";

#define DHTPIN 0          //pin where the dht11 is connected

DHT dht(DHTPIN, DHT11);

WiFiClient client;

void setup()
{
      Serial.begin(115200);
      delay(10);
      dht.begin();

      Serial.println("Connecting to ");
      Serial.println(ssid);


      WiFi.begin(ssid, pass);

     while (WiFi.status() != WL_CONNECTED)
     {
           delay(500);
           Serial.print(".");
     }
      Serial.println("");
      Serial.println("WiFi connected");

}

void loop()
{

     float h = dht.readHumidity();
     float t = dht.readTemperature();

           if (isnan(h) || isnan(t))
              {
```

```
            Serial.println("Failed to read from DHT sensor!");
             return;
          }

             if (client.connect(server,80))   //   "184.106.153.149" or api.thingspeak.com
            {

                String postStr = apiKey;
                postStr +="&field1=";
                postStr += String(t);
                postStr +="&field2=";
                postStr += String(h);
                postStr += "\r\n\r\n";

                client.print("POST /update HTTP/1.1\n");
                client.print("Host: api.thingspeak.com\n");
                client.print("Connection: close\n");
                client.print("X-THINGSPEAKAPIKEY: "+apiKey+"\n");
                client.print("Content-Type: application/x-www-form-urlencoded\n");
                client.print("Content-Length: ");
                client.print(postStr.length());
                client.print("\n\n");
                client.print(postStr);

                Serial.print("Temperature: ");
                Serial.print(t);
                Serial.print(" degrees Celcius, Humidity: ");
                Serial.print(h);
                Serial.println("%. Send to Thingspeak.");
             }
    client.stop();

    Serial.println("Waiting...");

  // thingspeak needs minimum 15 sec delay between updates
  delay(1000);
}
```

Output:





Conclusion:

Hence, the client-server environment was established which uploads data on the server ThingSpeak.

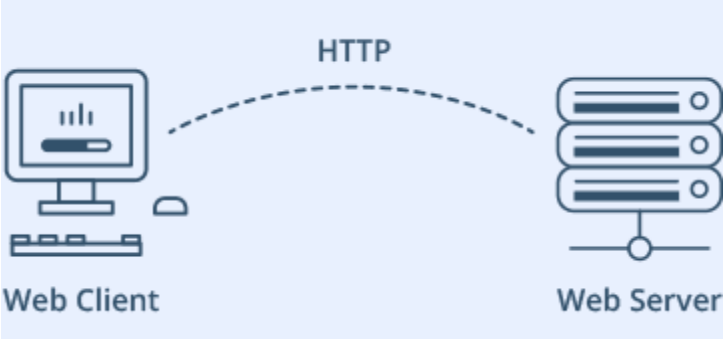
EXPERIMENT 5

NAME: Vishal Shashikant Salvi **UID: 2019230069**
CLASS: TE COMPS **BATCH: B**

Aim: -To implement NodeMCU as web server and mobile phone as a client to switch LED ON/OFF
Theory:

What is a Web server and how it works?

Web server is a place which stores, processes and delivers web pages to Web clients. Web client is nothing but a web browser on our laptops and smartphones. The communication between client and server takes place using a special protocol called Hypertext Transfer Protocol (HTTP).



In this protocol, a client initiates communication by making a request for a specific web page using HTTP and the server responds with the content of that web page or an error message if unable to do so (like famous 404 Error). Pages delivered by a server are mostly HTML documents.

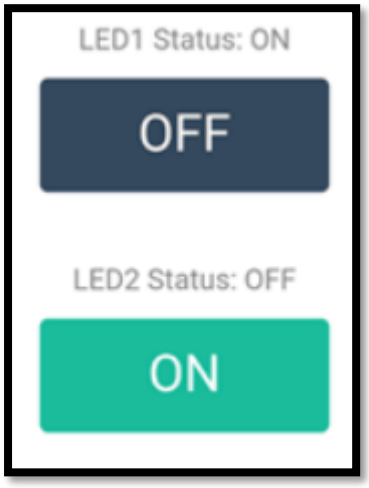
ESP8266 Operating Modes

One of the greatest features ESP8266 provides is that it cannot only connect to an existing WiFi network and act as a Web Server, but it can also set up a network of its own, allowing other devices to connect directly to it and access web pages. This is possible because ESP8266 can operate in three different modes: Station mode, Soft Access Point mode, and both at the same time. This provides possibility of building mesh networks.

Method:

When you type a URL in a web browser and hit ENTER, the browser sends a HTTP request (a.k.a. GET request) to a web server. It's a job of web server to handle this request by doing something. You might have figured it out by now that we are going to control things by

accessing a specific URL. For example, suppose we entered a URL like http://192.168.1.1/ledon in a browser. The browser then sends a HTTP request to ESP8266 to handle this request. When ESP8266 reads this request, it knows that user wants to turn the LED ON. So, it turns the LED ON and sends a dynamic webpage to a browser showing LED status : ON As easy as Pie!

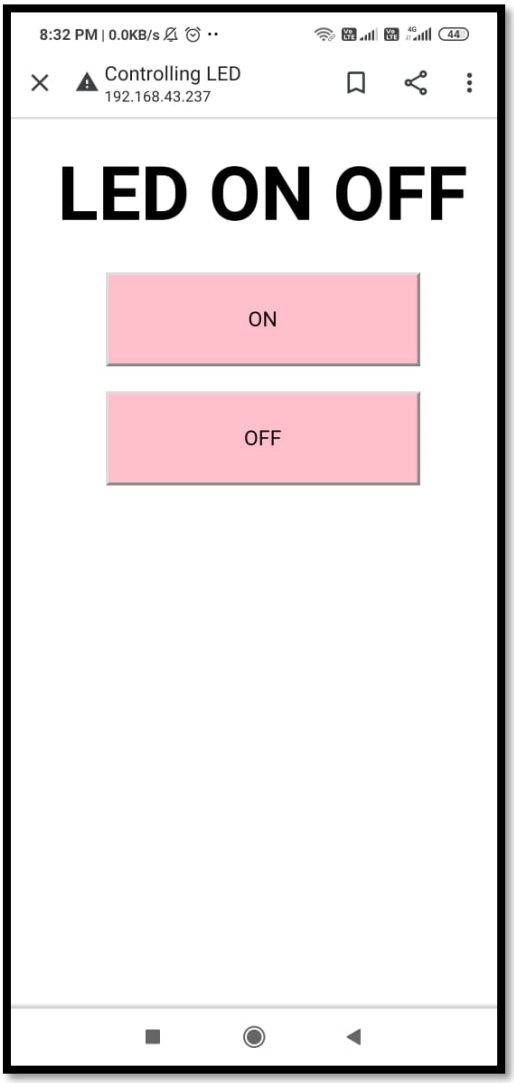


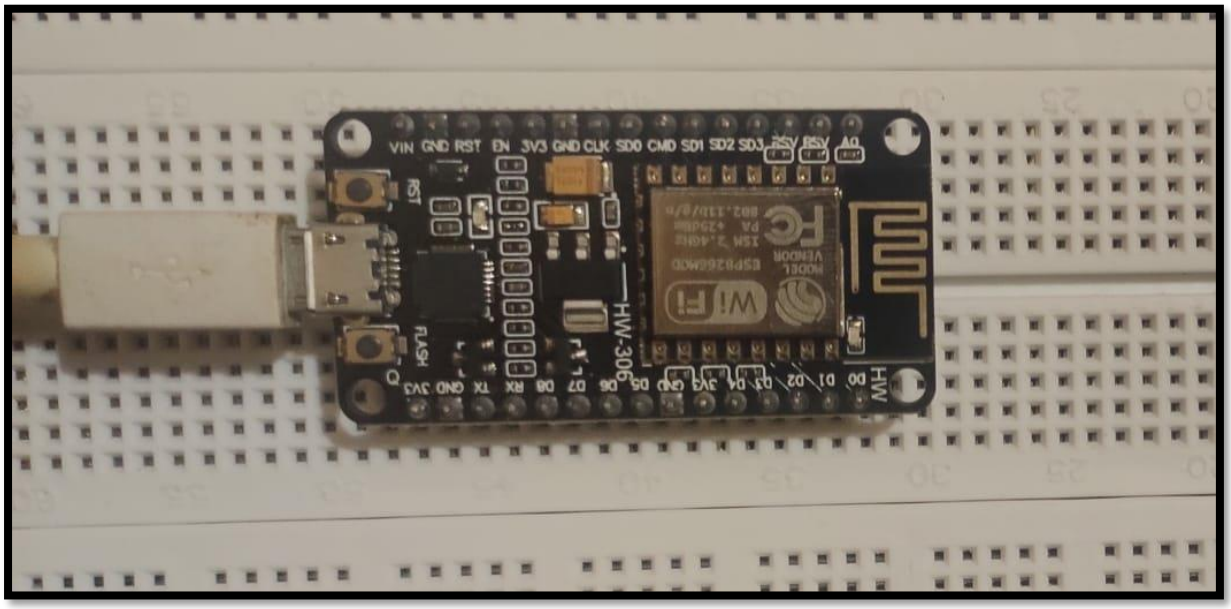
Input:

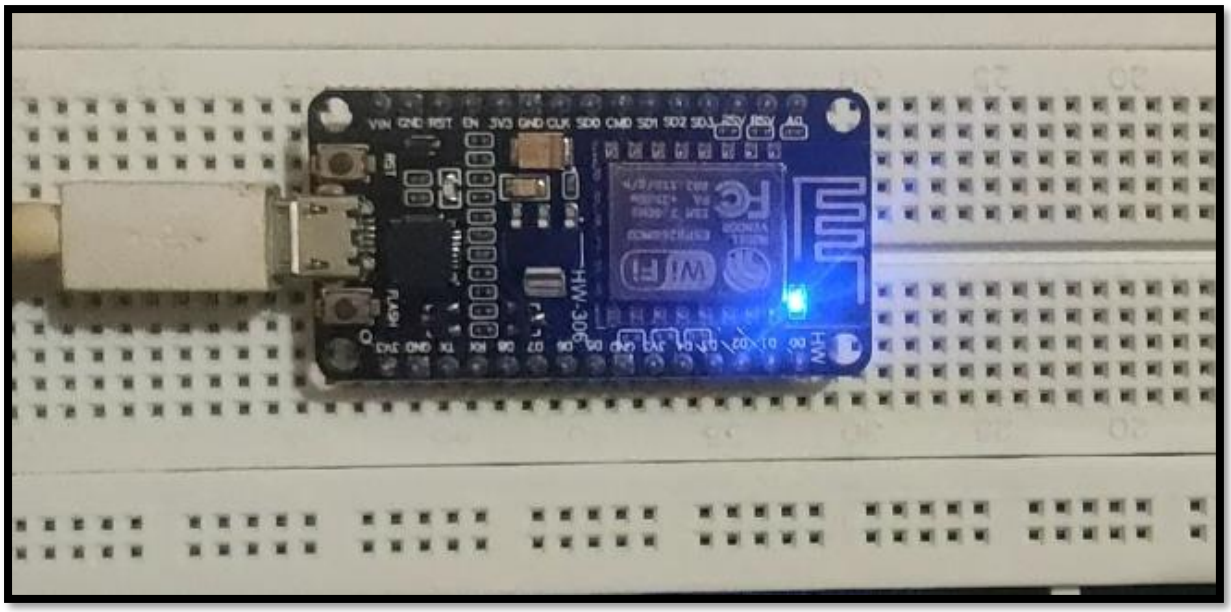
```
#include <ESP8266WiFi.h>

const char* ssid = "Redmi";
const char* password = "vishal@17";

WiFiServer server(80);

void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(LED_BUILTIN, HIGH);
  Serial.begin(9600);
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("NodeMCU is connected to WiFi");
  Serial.print("IP address: ");
```

```
  Serial.println(WiFi.localIP());
  server.begin();
  delay(3000);
}

void loop() {
  WiFiClient client;
  client = server.available();

 if (client == 1) {
    String request = client.readStringUntil('\n');
    client.flush();
    Serial.println(request);
    if (request.indexOf("ledon") != -1)
    { digitalWrite(LED_BUILTIN, LOW);
      Serial.println("LED is ON");
    }

    if (request.indexOf("ledoff") != -1)
    { digitalWrite(LED_BUILTIN, HIGH);
      Serial.println("LED is OFF");
    }

    client.println("HTTP/1.1 200 OK");
    client.println("Content-Type: text/html");
    client.println("");

    client.println("<!DOCTYPE HTML>");
    client.println("<html>");
    client.println("<head>");
    client.println("<title> Controlling LED </title>");
    client.println("</head>");
    client.println("<body align= \"center\" >");
    client.println("<h1> <font size = \"25\" face = \"Cooper\" >  LED ON OFF  </font> </h1> ");
    client.println("<p> <a href=\"/ledon\">  <button style=\"height:60px; background-color: pink;
width:200px; cursor: pointer\"> ON  </button> <a> </font></p> ");
    client.println("<p> <a href=\"/ledoff\"> <button style=\"height:60px; background-color: pink;
width:200px; cursor: pointer\" > OFF </button> <a> </font></p> ");
    client.println("</body>");
    client.println("</html>");

    Serial.println("Client disonnected");
    Serial.println("------------------");
    Serial.println("                  ");

  }
}
```

Output:


8:32 PM | 0.0KB/s
Controlling LED
192.168.43.237
LED ON OFF
ON
OFF


VIN GND RST EN 3V3 GND CLK SD0 CMD SD1 SD2 SD3 RSV RSV A0
RST
FLASH
HW-306
Wi Fi
D0 D1 D2 D3 D4 3V3 GND D5 D6 D7 D8 RX TX GND 3V3



Conclusion:

Thus, a communication between nodeMCU as a web server and the mobile phone as a client was established.

EXPERIMENT 6

NAME: Vishal Shashikant Salvi **UID: 2019230069**
CLASS: TE COMPS **BATCH: B**

Aim: To establish a connection between two Arduinos
Requirements: Tinker
Hardware Required

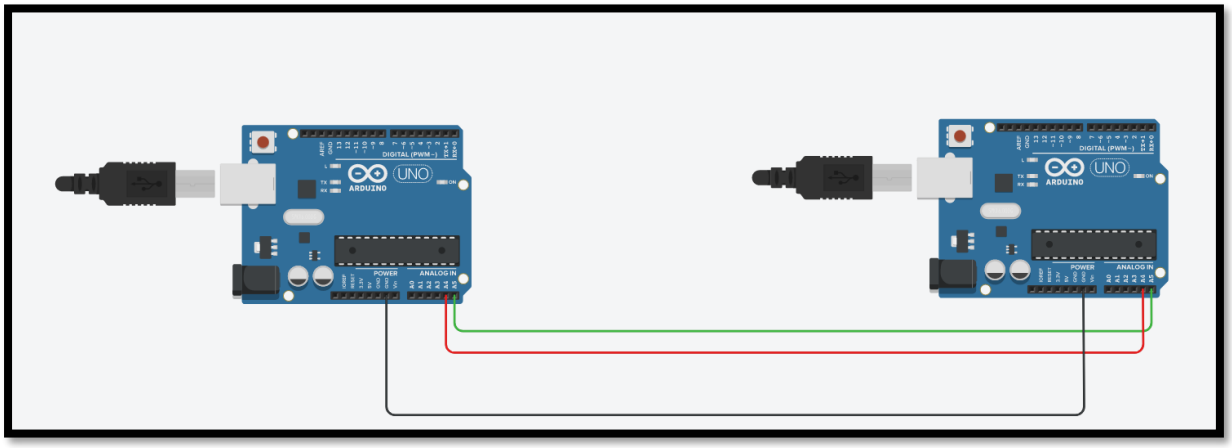
- 2 Arduino Boards
- hook-up wires
- Register
- Switch
- LED

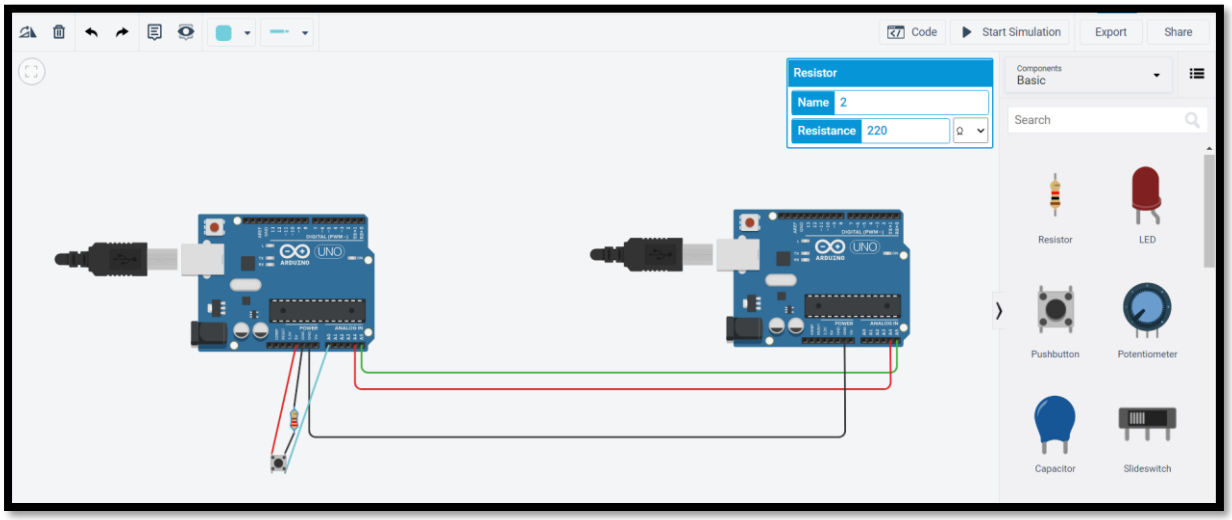
Theory:

In this example, two boards are programmed to communicate with one another in a Master Writer/Slave Receiver configuration via the I2C synchronous serial protocol. Several functions of Arduino's Wire Library are used to accomplish this. Arduino 1, the Master, is programmed to send 6 bytes of data every half second to a uniquely addressed Slave. Once that message is received, it can then be viewed in the Slave board's serial monitor window opened on the USB connected computer running the Arduino Software (IDE).

The I2C protocol involves using two lines to send and receive data: a serial clock pin (SCL) that the Arduino Master board pulses at a regular interval, and a serial data pin (SDA) over which data is sent between the two devices. As the clock line changes from low to high (known as the rising edge of the clock pulse), a single bit of information - that will form in sequence the address of a specific device and a a command or data - is transferred from the board to the I2C device over the SDA line. When this information is sent - bit after bit -, the called upon device executes the request and transmits it's data back - if required - to the board over the same line using the clock signal still generated by the Master on SCL as timing. The initial eight bits (i.e. eight clock pulses) from the Master to Slaves contain the address of the device the Master wants data from. The bits after contain the memory address on the Slave that the Master wants to read data from or write data to, and the data to be written, if any.

Each Slave device has to have its own unique address and both master and slave devices need to take turns communicating over a the same data line line. In this way, it's possible for your Arduino boards to communicate with many device or other boards using just two pins of your microcontroller, using each device's unique address.





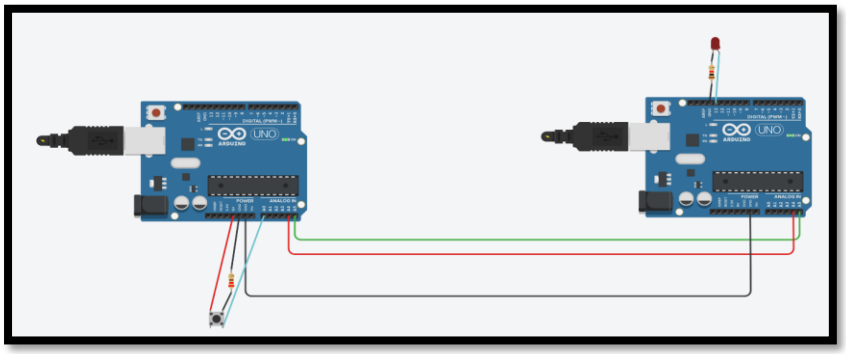
Master Code:

```
#include <Wire.h>
int pushButton = A0;
int x = 0;
void setup()
{
  Wire.begin();
  pinMode(pushButton, INPUT);
}

void loop()
```

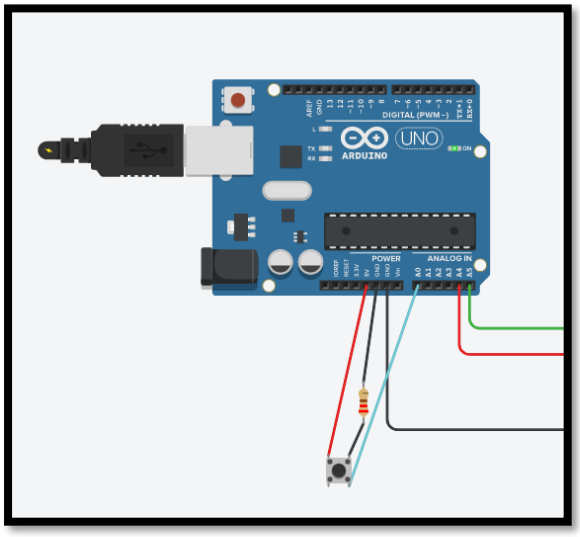
```
{
  Wire.beginTransmission(1);
  x = digitalRead(pushButton);
  Wire.write(x);
  Wire.endTransmission();
  delay(500);
}
```

Slave Code:

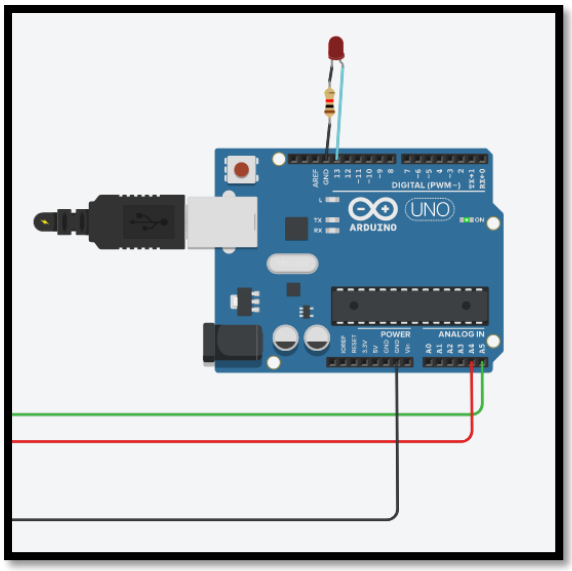
```
#include<Wire.h>
int pinLed=13;
int x=0;

void setup()
{
 Wire.begin(1);
 Wire.onReceive(receiveEvent);
 pinMode(pinLed, OUTPUT);

}

void loop()
{
 delay(100);
}

void receiveEvent(int howMany){
 x = Wire.read();
 if (x == 1){
  digitalWrite(pinLed,HIGH);
 }
 else{
  digitalWrite(pinLed,LOW);
 }
}
```



Master:



Slave:



Conclusion:

Communication was established between two arduino devices i.e one master and other slave, where depending on the output of the former arduino device, the latter arduino device performs various applications.

Link:

https://www.tinkercad.com/things/5KzcXxwZIw0-grand-waasa/editel?sharecode=daYnVGfNUuhkeWqCat3K0JNfR1wehp2yAWhZ8cs0D-M

EXPERIMENT 7

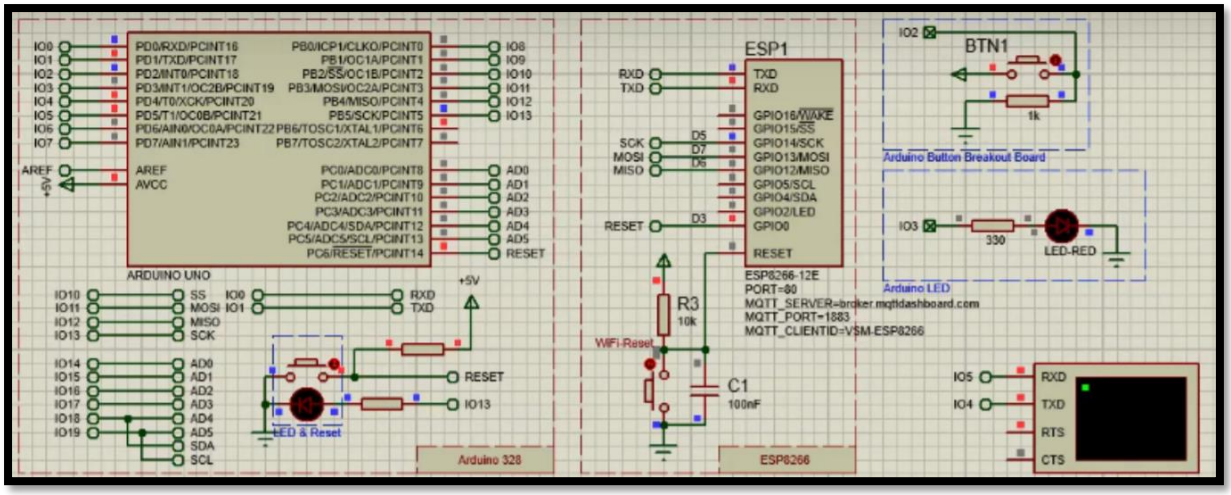
NAME: Vishal Shashikant Salvi **UID: 2019230069**

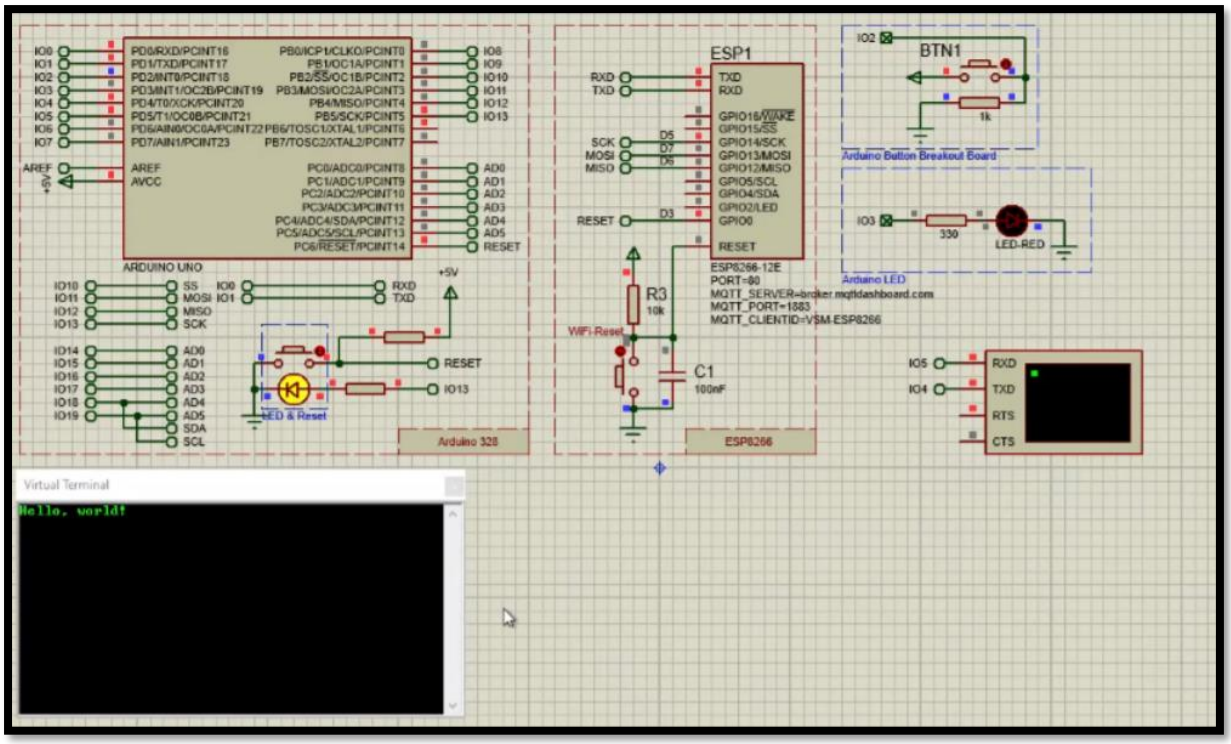
CLASS: TE COMPS **BATCH: B**

Aim: To explore the use of MQTT protocol.

Theory:

1. MQTT clients publish a message to an MQTT broker and other MQTT clients subscribe to messages they want to receive. Implementations of MQTT clients typically require a minimal footprint so are well suited for deployment on small constrained devices and are very efficient in their bandwidth requirements.
2. MQTT brokers receive published messages and dispatch the message to the subscribing MQTT clients. An MQTT message contains a message topic that MQTT clients subscribe to and MQTT brokers use these subscription lists for determining the MQTT clients to receive the message.
3. MQTT implements 3 quality of service levels for agreement between the sender and receiver: 1) At most once (0), 2) At least once (1), and 3) Exactly once (2). These QoS levels allow for more reliable IoT applications since the underlying messaging infrastructure and adapt to unreliable network conditions.
4. MQTT allows for a persistent session between the client and the broker. This allows for sessions to persist even if the network is disconnected. Once the network is reconnected, the information to reconnect the client to the broker still exists. This is one of the key features that makes the MQTT protocol more efficient than HTTP for use over unreliable cellular networks.
5. MQTT clients that subscribe to a new topic have no insight into when to expect the first message they will receive. However, an MQTT broker can store a retained message that can be sent immediately upon a new MQTT subscription. In this case, the MQTT client will receive at least one message upon subscribing to the topic.





Conclusion:

Thus, from this experiment I understood about MQTT Protocol.