

ASSIGNMENT – 4

Write code and connections in wokwi for the ultrasonic sensor.

Whenever the distance is less than 100 cms send an "alert" to the IBM cloud and display in the device recent events.

SOURCE CODE:

```
#include <WiFi.h> #include
```

```
<PubSubClient.h> #define
```

```
ORG "486ral"
```

```
#define DEVICE_TYPE "IOT"
```

```
#define DEVICE_ID "id07"
```

```
#define TOKEN "123456789"
```

```
#define trigpin 5
```

```
#define echopin 18
```

```
char server[] = ORG ".messaging.internetofthings.ibmcloud.com";char
```

```
publishTopic[] = "iot-2/evt/data/fmt/json";
```

```
char authMethod[] = "use-token-auth";
```

```
char token[] = TOKEN;
```

```
char clientId[] = "d:" ORG ":" DEVICE_TYPE ":" DEVICE_ID
```

```
WiFiClient wifiClient;
```

```
PubSubClient client(server, 1883, wifiClient);
```

```
long duration;
```

```
float dist;
```

```
void setup()
```

```
{
```

```
  Serial.begin(9900);
```

```
  pinMode(trigpin, OUTPUT);
```

```
  pinMode(echopin, INPUT);
```

```
  wifiConnect(); mqttConnect();
```

```
}
```

```
void loop() {
```

```
  publishData();
```

```
  delay(500);
```

```
  if (!client.loop())
```

```
  {
```

```
    mqttConnect();
```

```
  }
```

```
}
```

```

void wifiConnect()
{
    Serial.print("Connecting to ");
    Serial.print("Wifi"); WiFi.begin("Wokwi-
    GUEST", "", 6);
    while (WiFi.status() != WL_CONNECTED)
    {
        delay(500);
        Serial.print(".");
    }
    Serial.print("WiFi connected, IP address: ");
    Serial.println(WiFi.localIP());
}

```

```

void mqttConnect()
{
    if (!client.connected())
    {
        Serial.print("Reconnecting MQTT client to ");
        Serial.println(server);
        while (!client.connect(clientId, authMethod, token))
        {
            Serial.print(".");
            delay(500);
        }

        Serial.println();
    }
}

```

```

void publishData()
{
    digitalWrite(trigpin,LOW);
    digitalWrite(trigpin,HIGH);
    delayMicroseconds(10);
    digitalWrite(trigpin,LOW);
    duration=pulseIn(echopin,HIGH);
    dist=duration*0.034 /2; if(dist<100)
    {

        String payload = "{\"Distance\".";
        payload += dist;
        payload += ",";
        payload += "\"Status\".";
        payload += "\"Alert\"}";

        Serial.print("\n");
        Serial.print("Sending payload: ");
        Serial.println(payload);
        if (client.publish(publishTopic, (char*) payload.c_str()))
        {
            Serial.println("Publish OK");
        }

    }

    if(dist>100)

```

```

{

String payload = "{ \"Distance\": ";
payload += dist;
payload += " ";
payload += "\"Status\": ";
payload += "\"Normal\"}";

Serial.print("\n");

Serial.print("Sending payload: ");

Serial.println(payload);

if(client.publish(publishTopic, (char*) payload.c_str()))
{
    Serial.println("Publish OK");
}
else
{
    Serial.println("Publish FAILED");
}
}
}

```

diagram.json:

```

{
  "version": 1,
  "author": "Uri Shaked",
  "editor": "wokwi", "parts":
  [
    { "type": "wokwi-esp32-devkit-v1", "id": "esp", "top": 0, "left": 0, "attrs": {} },
    { "type": "wokwi-hc-sr04", "id": "ultrasonic1", "top": -109.38, "left": 180.61, "attrs": {} }
  ]
}

```

```

],

"connections": [

  ["esp:TX0", "$serialMonitor:RX", "", []],

  ["esp:RX0", "$serialMonitor:TX", "", []],

  ["ultrasonic1:ECHO", "esp:D18", "green", [ "v0" ] ],

  ["ultrasonic1:TRIG", "esp:D5", "orange", [ "v0" ] ],[

    "ultrasonic1:VCC",

    "esp:VIN",

    "red",

    [ "v22.14", "h-48.86", "v-27.94", "h-253.24", "v173.77" ]

  ],

  [ "ultrasonic1:GND", "esp:GND.2", "black", [ "v250.04", "h-311.59", "v3.06" ] ]

]

}

```

OUTPUT:

The screenshot shows the Wokwi IDE interface. On the left, the code for `esp32-blink.ino` is displayed, which includes necessary libraries, defines MQTT settings, and configures the ultrasonic sensor pins (D5 for TRIG, D18 for ECHO). The `setup` function initializes the serial port, pin modes, and MQTT connection. The `loop` function (though not explicitly shown in the snippet, the output implies it) triggers the sensor and publishes the distance data.

On the right, the simulation window shows a 3D model of the ESP32 and the HC-SR04 sensor connected by wires. Below the simulation, the terminal output shows the following sequence of events:

```

Sending payload: {"Distance":399.92,"Status":"Normal"}
Publish OK

Sending payload: {"Distance":399.92,"Status":"Normal"}
Publish OK

Sending payload: {"Distance":399.92,"Status":"Normal"}

```

