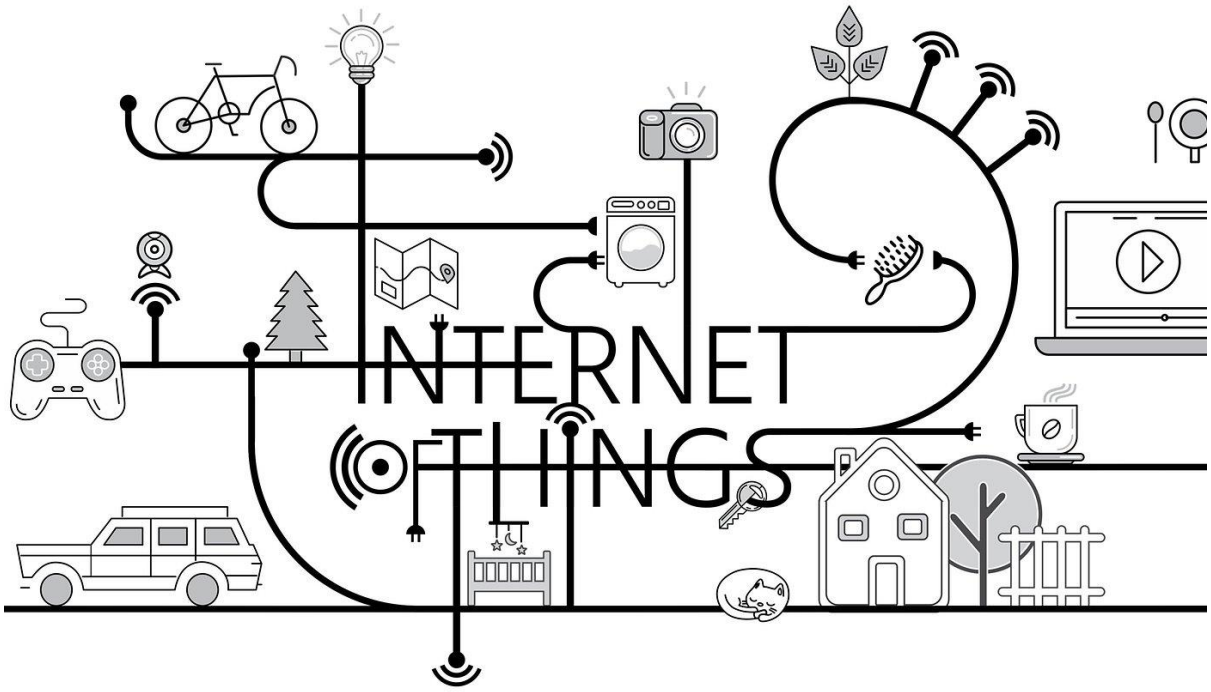
IOT Mini Project



Project Done By:

Sl No	Name	Section	ID
1	Mohammed Umar Raehan	7CSE7	20171CSE0416
2	Nagashree R	7CSE7	20171CSE0434
3	Nikhil R	7CSE7	20171CSE0462
4	Nikith A R	7CSE7	20171CSE0463
5	Umraz Hussain	7CSE7	20181LCS0006

1 Air Quality Monitoring System

1.1 Abstract

Air pollution has become a common phenomenon everywhere. Specially in the urban areas, air pollution is a real-life problem. In the urban areas, the increased number of petrol and diesel vehicles and the presence of industrial areas at the outskirts of the major cities are the main causes of air pollution. The problem is seriously intensified in the metropolitan cities. The governments all around the world are taking every measure in their capacity. Many European countries have aimed to replace petrol and diesel vehicles with the electric vehicles by 2030. Even India has aimed to do so by 2025.

The main aim of this project is to develop a device which can monitor PPM in air in real time, tell the quality of air and log data to a remote server(ThingSpeak).

The air monitoring device developed in this project is based on Arduino Uno. The Arduino board connects with ThingSpeak platform using ESP8266 Wi-Fi module. The sensor used for monitoring the air pollution is MQ-135 gas sensor. The sensor data is also displayed on a character LCD.

1.2 System Design and Architecture

Arduino Uno

Arduino Uno is one of the most popular prototyping boards. It is small in size and packed with rich features. The board comes with built-in Arduino boot loader. It is an Atmega 328 based controller board which has 14 GPIO pins, 6 PWM pins, 6 Analog inputs and on board UART, SPI and TWI interfaces. In this IOT device, 9 pins of the board are utilized. There are six pins used to interface the character LCD. There are two pins utilized to interface the ESP8266 Wi-Fi Module and an analog input pin is used to connect the MQ-135 sensor.

16X2 Character LCD

The 16X2 LCD display is used to monitor the sensor values read by the Arduino board from MQ-135. It is interfaced with the Arduino Uno by connecting its data pins D4 to D7 with pins 6 down to 3 of the controller respectively. The RS and E pins of the LCD are connected to pins 13 and 12 of the controller respectively. The RW pin of the LCD module is connected to the ground.

16x2 LCD

ESP8266 Wi-Fi Module

The ESP8266 WiFi Module is a self contained SOC with integrated TCP/IP protocol stack that can give any microcontroller access to your WiFi network. The ESP8266 is capable of either hosting an application networking functions from another application Each ESP8266 module comes pre-programmed with an AT command.

The ESP8266 supports APSD for VoIP applications and Bluetooth co-existence interfaces, it contains a self-calibrated RF allowing it to work under all operating conditions, and requires no external RF parts.

Features

802.11 b/g/n

Wi-Fi Direct (P2P), soft-AP

Integrated TCP/IP protocol stack

Integrated TR switch, balun, LNA, power amplifier and matching network

Integrated PLLs, regulators, DCXO and power management units

+19.5dBm output power in 802.11b mode

Power down leakage current of <10uA

1MB Flash Memory

Integrated low power 32-bit CPU could be used as application processor

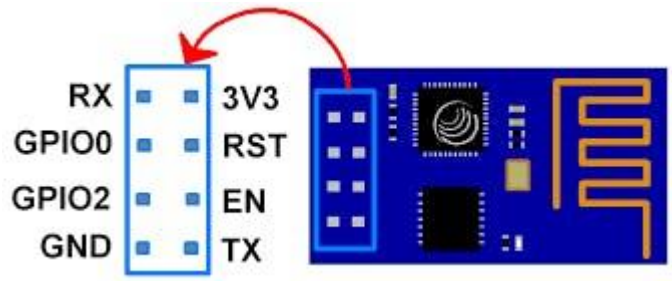
SDIO 1.1 / 2.0, SPI, UART

STBC, 1×1 MIMO, 2×1 MIMO

A-MPDU & A-MSDU aggregation & 0.4ms guard interval

Wake up and transmit packets in < 2ms

Standby power consumption of < 1.0mW (DTIM3)



For connecting ESP8266 Module with Arduino Uno, you need 3.3 voltage regulator because Arduino is not capable of providing 3.3 v to ESP8266.

MQ-135 Gas sensor

The MQ-135 gas sensor senses the gases like ammonia nitrogen, oxygen, alcohols, aromatic compounds, sulfide and smoke. The operating voltage of this gas sensor is from 2.5V to 5.0V. MQ-135 gas sensor can be implementation to detect the smoke, benzene, steam and other harmful gases.

Connections

Arduino ==> LCD

GND ==> GND

5 V ==> Vcc

D13 ==> RS

GND ==> R/W

D12 ==> Enable

D6 ==> DB4

D5 ==> DB5

D4 ==> DB6

D3 ==> DB7

5V ==> LED+

GND ==> LED-

Arduino ==> Gas sensor

GND ==> GND

5V ==> Vcc

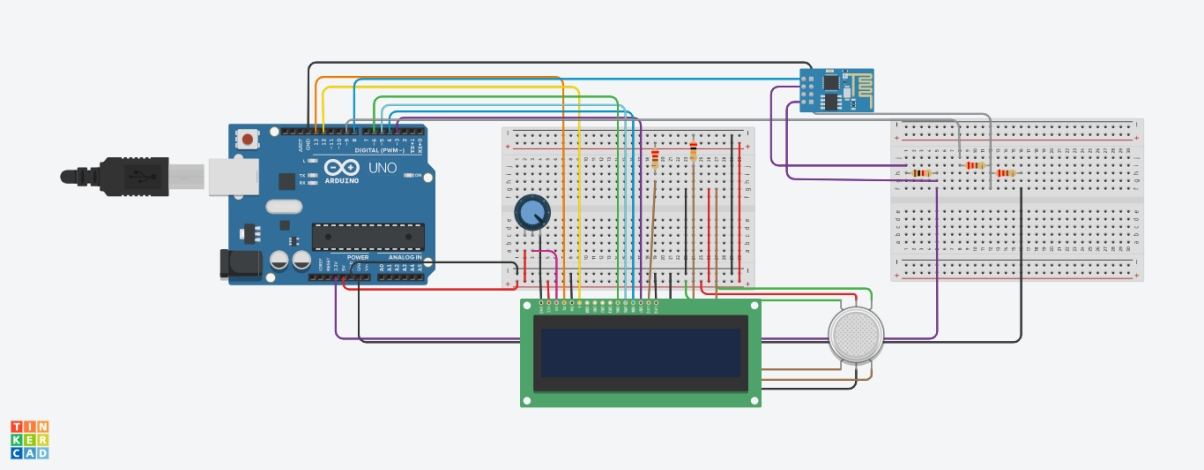
Analog0 ==> A0

Arduino ==> ESP8266

D10 ==> Tx

D11 ==> Rx

System Architecture



1.3 Algortihm

As the device is powered, the Arduino board loads the required libraries, flashes some initial messages on the LCD screen and start sensing data from the MQ-135 sensor. The sensor can be calibrated so that its analog output voltage is proportional to the concentration of polluting gases in PPM. The analog voltage sensed at the pin A0 of the Arduino is converted to a digital value by using the in-built ADC channel of the Arduino. The Arduino board has 10-bit ADC channels, so the digitized value ranges from 0 to 1023. The digitized value can be assumed proportional to the concentration of gases in PPM. The read value is first displayed on LCD screen and passed to the ESP8266 module wrapped in proper string through virtual serial function. The Wi-Fi module is configured to connect with the ThingSpeak IOT platform. ThingSpeak is an IOT analytics platform service that allows to aggregate, visualize and analyze live data streams in the cloud. ThingSpeak provides instant visualizations of data posted by the IOT devices to ThingSpeak server.

The Wi-Fi module can be connected with the ThingSpeak server by sending AT commands from the module. The module first test the AT startup by sending the AT command. Then, command is passed by the controller to the Wi-Fi module using software serial function. In response to the command 'AT', the platform must respond with 'OK' if the cloud service is running.

Then, the AT command to view the version information is passed.

```
AT + GMR
```

In response to this command, the IOT platform must respond by sending back the version information, SDK version and the time bin is compiled.

1. ESP-01 output : it will be 00160901.

2. ESP-12 output : it will be 00180000902-AI03.

Next, the AT command to set the connection to Wi-Fi mode is send.

```
AT + CWMODE = 3
```

By setting the parameter in CWMODE to 3, the Wi-Fi connection is configured to SoftAP as well as station mode. This AT command can take three parameters

1 - set Wi-Fi connection to station mode

2 - set Wi-Fi connection to SoftAP mode

3 - set Wi-Fi connection to SoftAP + station mode

In response to this command, the IOT platform must send back the string indication the Wi-Fi connection mode set.

Next, the AT command to reset the module is send.

```
AT + RST
```

In response to this command, the Wi-Fi module must restart and send back a response of 'OK'. After resetting the module.

Next, command to setup multiple connections is AT+ CIPMUX.

```
AT + CIPMUX=1
```

This AT command can take two parameters - 0 for setting single connection and 1 for setting multiple connections.

Next, the command to connect with the Access Point (AP) is passed which takes two parameters where first parameter is the SSID and the other parameter is the password.

```
AT+CWJAP=\"SSID\",\"Password\"
```

Next, the AT command to get local IP address is passed.

```
AT + CIFSR
```

In response to this command, the local IP address of the Wi-Fi connection is sent back by the module. Now, the module is ready to establish TCP IP connection with the ThingSpeak server. The controller reads the sensor data and store it in a string variable.

The TCP IP connection is established by sending the following AT command

```
AT + CIPSTART = 4, "TCP", "184.106.153.149", 80
```

The AT + CIPSTART command can be used to establish a TCP connection, register an UDP port or establish an SSL connection. Above command is used to establish a TCP IP connection. For establishing a TCP-IP connection, the command takes four parameters where first parameter is link ID which can be a number between 0 to 4, second parameter is connection type which can be TCP or UDP, third parameter is remote IP address or IP address of the cloud service to connect with and last parameter is detection time interval for checking if the connection is live. If the last parameter is set to 0, the TCP keep-alive feature is disabled otherwise a time interval in seconds range from 1 to 7200 can be passed as parameter. In response to this command, the server must respond with 'OK' if connection is successfully established otherwise it should respond with message 'ERROR'.

When the connection with the server is successfully established and the controller has read the sensor value, it can send the data to the cloud using `AT+CIPSEND` command.

```
AT + CIPSEND = 4
```

This command takes four parameters, where first parameter is the link ID which can be a number between 0 to 4, second parameter is data length which can be maximum 2048 bytes long, third parameter is remote IP in case of an UDP connection and remote port number in case of UDP connection. The third and fourth parameter are optional and used only in case of UDP connection with the server. Since, the TCP IP connection is established, these parameters are not used. The command is followed by a string containing the URL having the field names and values passed through the HTTP GET method.

In this project, a string containing the URL having API Key and the sensor value as the field and value is passed. The passed field and its value are logged on the cloud server. It is important to pass the API key in this URL as one of the field value in order to connect with the registered cloud service. The Air quality measured by sensor can now be monitored and recorded through the ThingSpeak IOT platform.

1.4 CODE

```
#include <SoftwareSerial.h>

/*
 LiquidCrystal Library - Hello World

 Demonstrates the use a 16x2 LCD display.  The LiquidCrystal
 library works with all LCD displays that are compatible with the
 Hitachi HD44780 driver. There are many of them out there, and you
 can usually tell them by the 16-pin interface.

 This sketch prints "Hello World!" to the LCD
 and shows the time.

 The circuit:
* LCD RS pin to digital pin 12
* LCD Enable pin to digital pin 11
* LCD D4 pin to digital pin 5
* LCD D5 pin to digital pin 4
* LCD D6 pin to digital pin 3
* LCD D7 pin to digital pin 2
* LCD R/W pin to ground
* LCD VSS pin to ground
* LCD VCC pin to 5V
* 10K resistor:
* ends to +5V and ground
* wiper to LCD VO pin (pin 3)
```

```
 This example code is in the public domain.

 http://www.arduino.cc/en/Tutorial/LiquidCrystal
 */

// include the library code:
#include <LiquidCrystal.h>

// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(13, 12, 6, 5, 4, 3); // LCD connections
/*
 D13 ==> RS
GND  ==>   R/W
D12  ==> Enable
D6   ==> DB4
D5   ==> DB5
D4   ==> DB6
D3   ==> DB7

*/
float t=0;
char data = 0;
String apiKey = "XBQDVORXXGAROWDW";  // Write API key

// connect 8 to TX of ESP
// connect 9 to RX of ESP
SoftwareSerial ser(8,9); // RX, TX
```

```
void setup()
{
// enable debug serial
  Serial.begin(9600); //  serial data transmission at Baudrate of 9600

// enable software serial

  ser.begin(9600);
  lcd.begin(16, 2);  // to intialize LCD

  lcd.setCursor(0,0);

  lcd.print("    Welcome");

  lcd.setCursor(0,1);

  lcd.print("       To       ");

  delay(3000);

  lcd.clear();

  lcd.setCursor(0,0);

  lcd.print("     AIR");

  lcd.setCursor(0,1);

  lcd.print("QUALITY MONITOR");
```

```
  delay(3000);
  ser.println("AT");  // Attenuation

  delay(1000);

  ser.println("AT+GMR"); // To view version info for ESP-01 output: 00160901 and ESP-12 output: 0018000902-AI03

  delay(1000);

  ser.println("AT+CWMODE=3"); // To determine WiFi mode
/*
1 = Station mode (client)
2 = AP mode (host)
3 = AP + Station mode (ESP8266 has a dual mode)
*/

  delay(1000);

  ser.println("AT+RST"); // To restart the module

  delay(5000);

  ser.println("AT+CIPMUX=1"); // Enable multiple connections
  /*

    0: Single connection
    1: Multiple connections (MAX 4)

*/
```

```
 delay(1000);

 String cmd="AT+CWJAP=\"SSID\",\"PASSWORD\""; // connect to Wi-Fi

 ser.println(cmd);

 delay(1000);

 ser.println("AT+CIFSR"); // Return or get the local IP address

 delay(1000);

 lcd.clear();

 lcd.setCursor(0,0);

 lcd.print("      WIFI");

 lcd.setCursor(0,1);

 lcd.print("   CONNECTED");

}

void loop()
{
 delay(1000);

 t = analogRead(A0);  // Read sensor value and stores in a variable t
```

```
Serial.print("Airquality = ");

Serial.println(t);

lcd.clear();
lcd.setCursor (0, 0);
lcd.print ("Air Qual: ");
lcd.print (t);
lcd.print (" PPM ");

lcd.setCursor (0,1);
if (t<=500)
 {
 lcd.print("Fresh Air");
 Serial.print("Fresh Air ");

 }
else if( t>=500 && t<=1000 )
 {
 lcd.print("Poor Air");
 Serial.print("Poor Air");

 }
else if (t>=1000 )
{
lcd.print("Very Poor");
Serial.print("Very Poor");

}
```

```
  //lcd.scrollDisplayLeft();
  delay(10000);

  lcd.clear();

  lcd.setCursor(0,0);

  lcd.print("  SENDING DATA");

  lcd.setCursor(0,1);

  lcd.print("    TO CLOUD");

  esp_8266();
}



void esp_8266()
{

   // TCP connection AT+CIPSTART=4,"TCP","184.106.153.149",80

    String cmd = "\nAT+CIPSTART=4,\"TCP\",\"";  // Establish TCP connection
    /*
     AT+CIPSTART=id,type,addr,port

    id: 0-4, id of connection
    type: String, “TCP” or “UDP”
    addr: String, remote IP
    port: String, remote port
```

```
 */
 cmd += "184.106.153.149"; // api.thingspeak.com
 cmd += "\",80";
 ser.println(cmd);
 Serial.println(cmd);
 if(ser.find("Error"))
 {
  Serial.println("AT+CIPSTART error");
  return;
 }
String getStr = "GET /update?api_key=";  // API key
getStr += apiKey;
//getStr +="&field1=";
//getStr +=String(h);
getStr +="&field1=";
```

```
  getStr +=String(t);

  getStr += "\r\n\r\n";

  // send data length

  cmd = "AT+CIPSEND="; // Send data AT+CIPSEND=id,length

  cmd += String(getStr.length());

  ser.println(cmd);

  Serial.println(cmd);

  delay(1000);

  ser.print(getStr);

  Serial.println(getStr);

  // thingspeak needs 16 sec delay between updates

  delay(17000);

}
```