

PROJECT REPORT - III on

Real- Time air quality monitoring using microprocessor

For

CSE2006: Microprocessor and Interfacing

Slot: A2

Satvarsh Gondala

18BCE2098

Faculty:

Mr. Anthony Xavier Glittas X



VIT[®]
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Introduction:

Real-time data is of great importance in today's day and age. I am creating more and more data, and more data generation does not necessarily mean more information or more insight. I also need to analyze this fast enough in order to create the meaning of this information. This is where I have to look at the acquisition of data in real-time. It cannot be done with just 'fast' since 'fast' in this age is not enough; I require real-time interaction. One second delay can mean a world of difference. Consumers, as well as industries, need to be able to collect, analyze, and act on the data in real-time. For example, this is helpful for autonomous vehicles, where even a millisecond in delay can cause accidents, emergency response from firefighters, in case if any fire is reported, hospitals using multiple devices to monitor the health conditions of patients. I have the topic of air quality monitoring in real-time for this project. Air quality in our country is getting worse day by day, especially in the northern area of the country, with nine cities in India getting international recognition, but not in the right way. Nine of our cities are ranked in the "Top 10 Most Polluted Cities in the World" in a list given by the World Health Organization (WHO) in 2018. The air quality is monitored. Good air quality is vital for a healthy life. It is estimated that air pollution in Delhi is equivalent to passive smoking, with the amount equating to around ten cigarettes per day. This is harmful as air pollution is responsible for most of the respiratory problems and diseases such as emphysema. Moreover, it also causes long term damage to nerve cells, the brain, kidneys, and other organs.

My aim is to harness the power of microprocessors along with internet for easy and reliable access to real time data of following parameters:

- CO₂
- NO_x
- NH₃
- Benzene
- H₂O

I can use sensors to gather the data, while using microprocessors to analyze and send the data to an external broker or store the data locally; I plan to make this as modular and accessible as possible.

General Method:

First, I use MQ135 Sensor that is connected to either ARDUINO or ESP32 Dev Board to get the current amount of pollutants in the air using Node MCU. Then, get the data about the pollutants and current temperature from the same sensor. The data is then sent to EMU 8086, the EMU 8086 can do the following:

- Temperature calculation
- Greenhouse Effect simulation
- Get the amount of humidity in air
- Get the amount of gases in air

These are the main data I can draw, my goal after this is to make this data accessible and find applications.

MQ135 sensor is the sensor used since it can sense multiple gases. It gives output in the form of voltage levels, and I need to convert it into PPM (Parts Per Million). So, for converting the output in PPM, I have used a library for the MQ135 sensor. The sensor will give us a specific value when there is no gas near it, and the safe level of air quality is 350 PPM, and it should not exceed the threshold limit. When it exceeds the threshold limit, then it starts causing Headaches, sleepiness, and if it exceeds 2000 PPM, then it can cause increased heart rate and many other diseases.

The other sensor used is temperature and humidity sensor; it works similarly except that it senses the temperature and humidity of the area, and based on the given set-points, it functions to alert the user. The sensor calculates relative humidity by measuring the electrical resistance between two electrodes of its capacitor. To measure the surrounding temperature, it uses a thermistor and spits out the digital signal on the data pin.

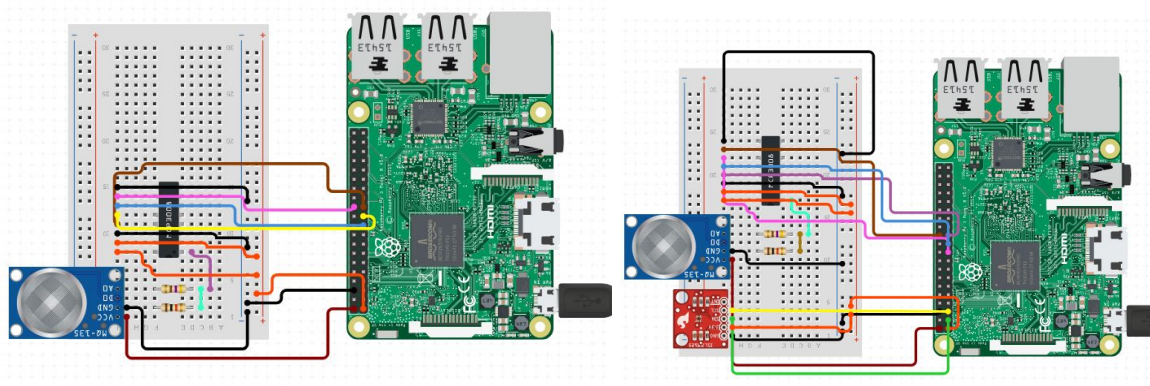
The sent data is then used as Input for EMU8086 where I can perform more computations and functions, you can view them in much detail as we proceed towards output.

Justification of duration:

The main thing I am looking for in this project is to expand the boundaries of EMU8086. While most of the projects were only within EMU8086, I wanted to combine EMU8086 along with hardware. So this is clearly out of my comfort zone especially given that the EMU8086 software is very outdated and does not feature traditional integration. You can look at my objectives to see why we took a lot of time for this project:

- Mastering Arduino, Raspberry Pi, EMU8086
- Ensuring proper connectivity (Online/Offline) between all components
- Creating complex Functions in EMU8086
- Making sure everything is customizable - Learning about various publishing models
- Time computation to analyze data faster
- Mastering Mosquito, Aeres (Broker)
- Adding various features for different parameters
- Enhancing existing raspberry pi model
- Completing the code for raspberry pi model
- Increasing the compatibility of node MCU
- Adding IO feature directly within the temperature calculator device
- Learning RGB functionality for Raspberry Pi code
- Adding more features of RGB color based on the amount of pollutants in air

I even went through a lot of revisions to get this project done, to understand those changes let's look at an example of raspberry pi. My original model did not have RGB integration, so I had to write a lot of code for review III to integrate it into the project. A similar case happened with NODE MCU and also the EMU8086 Device. Below you can see my circuits for both review 2 and review 3.



Planning & Methodology:

MQ135 sensor is the sensor used since it can sense multiple gases. It gives output in the form of voltage levels, and I need to convert it into PPM (Parts Per Million). So, for converting the output in PPM, I have used a library for the MQ135 sensor. The sensor will give us a specific value when there is no gas near it, and the safe level of air quality is 350 PPM, and it should not exceed the threshold limit. When it exceeds the threshold limit, then it starts causing Headaches, sleepiness, and if it exceeds 2000 PPM, then it can cause increased heart rate and many other diseases.

The other sensor used is temperature and humidity sensor; it works similarly except that it senses the temperature and humidity of the area, and based on the given set-points, it functions to alert the user. The sensor calculates relative humidity by measuring the electrical resistance between two electrodes of its capacitor. To measure the surrounding temperature, it uses a thermistor and spits out the digital signal on the data pin.

The final output is then sent to EMU8086 (a separate device called temperature calculator), which can be used to calculate greenhouse effect. It can also send temperature and dew to calculate cloud base for that specific area

Setup setup the circuit where a sensor is connected to a microprocessor and microcontroller

Establish Broker establish a broker and use it as a bridge between microcontroller and client

Gather Data establish an external client that is connected to gather data via sensor, controller, broker

Analyze Data perform functions in microprocessors to analyze data and create triggers, etc...

Compare Data Add any modifiers to existing triggers, compute the requirements, compare to other datasets, etc...

Generate Report All the final data is sent to an external client where you can view data and make changes necessary

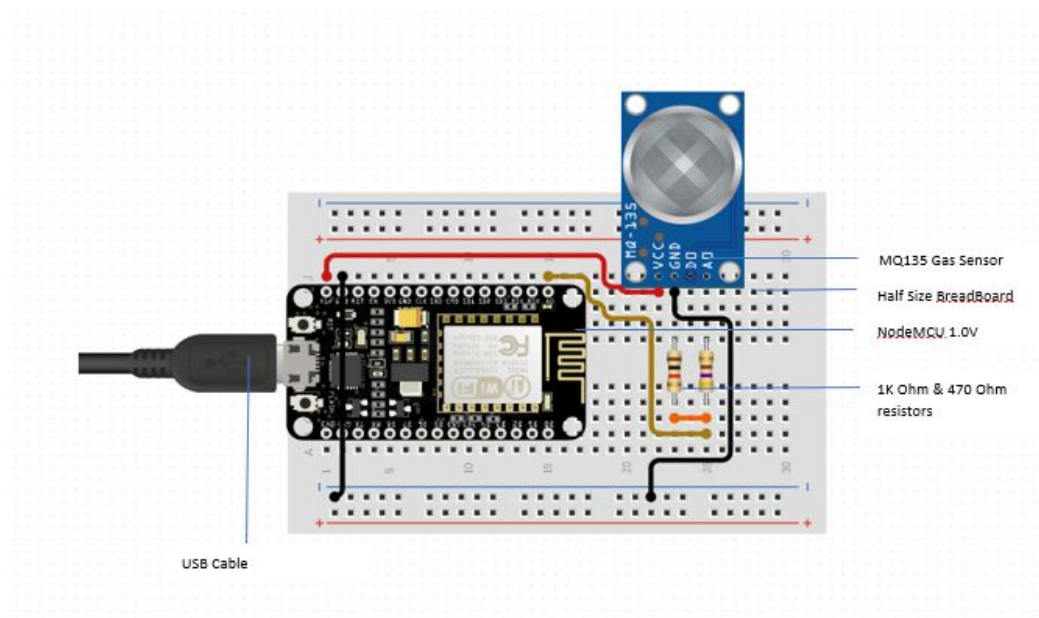
Components and Workflow :

The project from now will be divided into 3 parts:

Part - I: Setting up Arduino and Node MCU

The components required for this are:

- Node MCU
- Mosquito Broker
- Breadboard, resistors
- USB Cable



Setup:

1. Establish a Node MCU network, MQ135 Sensor
2. Establish the circuit design and configure the circuit around (1)
3. Create a test server/Instance in Mosquitto
4. Set up the client in Mosquitto
5. Setup publish/subscribe client via PubSub client library
6. Configure sensor to MQTT Connection
7. Configure the frequency
8. Establish basic authentication by logging in Mosquitto via "mosquitto.conf"
9. Visualize the output via "ny-power.org."

Setting up the Mosquitto Server

1. Install mosquitto, OpenSSL_Light,
2. Install mosquitto as a service in PowerShell via `"./mosquitto install."`
3. Validating the installation via `"sc query mosquitto."`

```
C:\Windows\system32>sc query mosquitto

SERVICE_NAME: mosquitto
        TYPE: 10  WIN64_OWN_PROC
        STATE: 1  STOPPED
        WIN64_EXIT_CODE  : 0 (0x0)
        SERVICE_EXIT_CODE: 0 (0x0)
        CHECKPOINT: 0x0
```

4. Running the mosquitto broker (1883 is MY system port)

```
C:\Windows\system32>netstat -a

proto  Local Address      Foreign Address    State
TCP    0.0.0.0:80         ws6:0              LISTENING
TCP    0.0.0.0:135        ws6:0              LISTENING
TCP    0.0.0.0:445        ws6:0              LISTENING
TCP    0.0.0.0:1801       ws6:0              LISTENING
TCP    0.0.0.0:1883       ws6:0              LISTENING
```

5. Using Syslog to create a login session in config files by
 `log_dest_syslog log`
 `log_IOTproject"`
6. Subscribing to different clients for each sensor via `"subscribe(topic,qos=0)"` (For each gas sensor)

```
C:\Windows\system32>client1.subscribe([("CO2/bulb2",),[("NH3/bulb3",2),("NOx/bulb4",1),("Benzene/bulb5",0)])

connecting to broker
subscribing QOS=0
subscribed result (0,1)
on on_substibe callback mid 1
```

7. Setting Quality of service

Either 1

Either 0

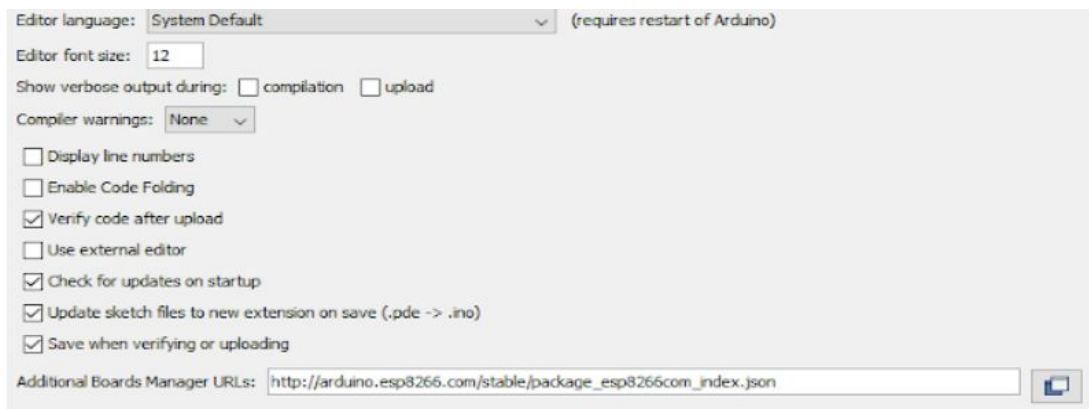
8. Establish subscribe wait loops by (Must do this for reach sensor type)

```
def wait_for(client,msgType,period=0.25):
    if msgType=="SUBACK":
        if client.on_subscribe:
            while not client.suback_flag:
                logging.info("waiting suback for CO2"); client.loop()
                time.sleep(period) #Depends on gas sensor cooldown"
```

Setting up NODE MCU

1. Enter this code in additional board managers in Arduino IDE

["http://arduino.esp8266.com/stable/package_esp8266com_index.json"](http://arduino.esp8266.com/stable/package_esp8266com_index.json)



2. Select the correct COM port and run this program

```
<p>void setup()
{
    // initialize digital pin 13 as an output.
    pinMode(13, OUTPUT);
}

// the loop function as many times as we require
// sensors to run
void loop() {
    digitalWrite(13, HIGH);
    delay(1000); //wait for required time eg; period 0.25 for NOx
    digitalWrite(13, LOW);
    delay(1000);
}
```


Setting up Audrino

Using PubSubClient to determine Publisher, Subscribers

2. Get the port value of your server (1883 by default).

3. Create a user, password in "Mosquitto.config" file

4. Import the basic libraries like ESPWifi, MQTT

```
#Import <ESP8266.Wifi.h>
```

```
#Import<PubSubClient.h>
```

5. Define the constants needed for the MQTT and Wi-Fi configurations

6. Import MQ135 sensor's library

```
#include <dht.h>
```

```
#define dht_apin D5
```

```
dht DHT;
```

```
WiFiClient espClient;
```

```
PubSubClient client(espClient);
```

```
long lastMsg = 0;
```

```
char msg[50];
```

```
int value = 0;
```

7. Define functions to set up Wi-Fi, MQTT, and callback method for MQTT Message

8. Use the functions in set up to configure settings and connect to server

```
void setup() {
```

```
pinMode(BUILTIN_LED, OUTPUT);
```

```
Serial.begin(9600);
```

```
Setup_wifi();
```

```
Client.setServer(mqtt_server, 17389);
```

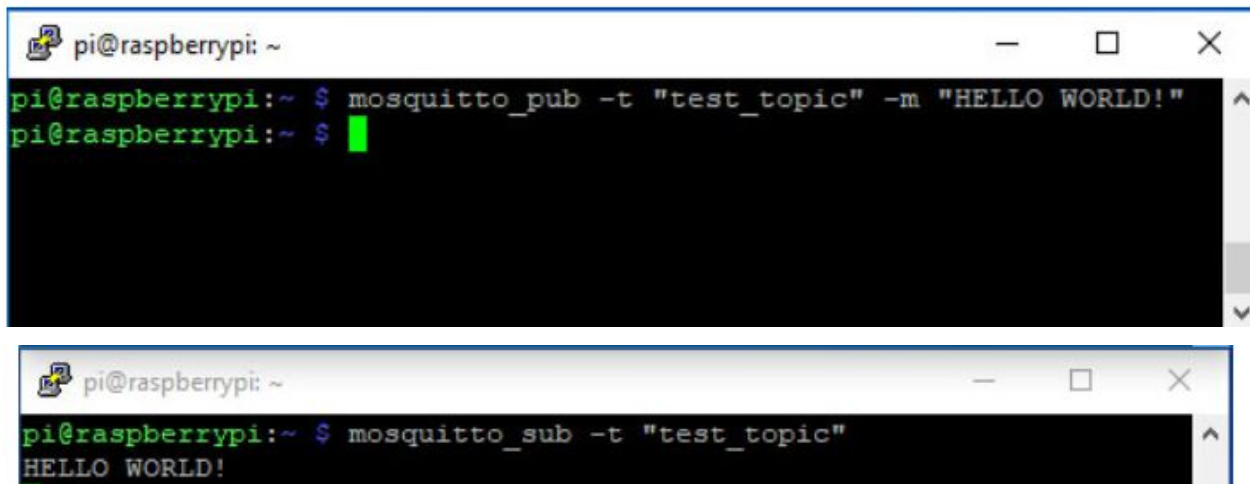
```
Client.setCallback(callback);
```

```
reconnect();
```

```
}
```

Working:

Here the Broker can be named as the central entity of this system. Clients can send messages to the Broker only. They cannot send messages to each other. Here sending messages is known as "publishing." These messages typically have a Topic and a Message Body. The Broker is the entity that filters the messages which are sent by the clients and forwards them. To receive a particular message. We have to subscribe to a specific topic. Moreover, I will receive all the messages published under that specific topic irrespective of the client who sent them.

Part - II : Setting up Raspberry Pi**1. Testing/Establishing Broker**


```

pi@raspberrypi: ~
pi@raspberrypi:~ $ mosquitto_pub -t "test_topic" -m "HELLO WORLD!"
pi@raspberrypi:~ $

pi@raspberrypi: ~
pi@raspberrypi:~ $ mosquitto_sub -t "test_topic"
HELLO WORLD!

```

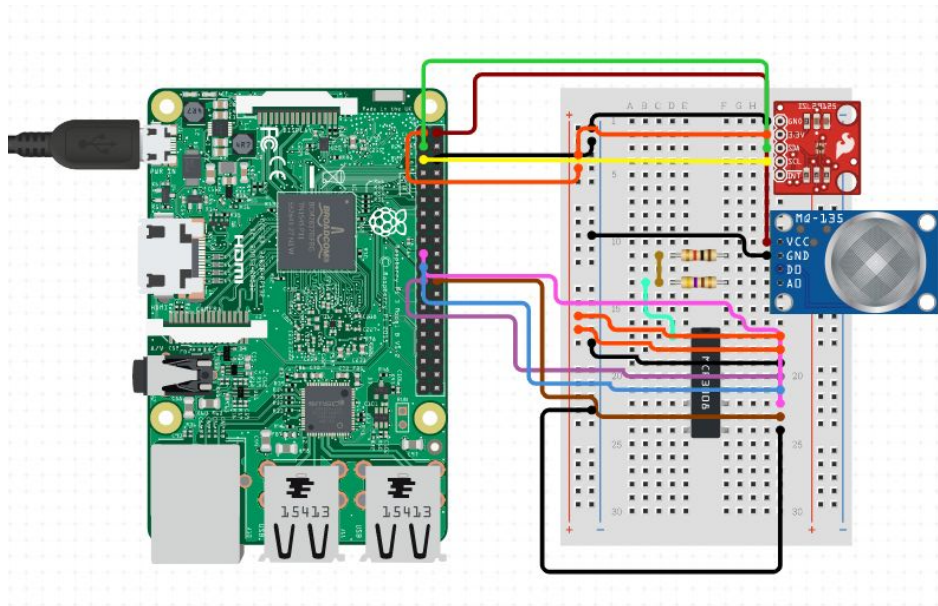
2. Set SWITCH as the publish client
3. Set LIGHT as subscriber client
4. Connect to the broker
5. Establish a frequency
6. Connect to port "1883" in mosquito
7. Get outputs from "ny-power.org."

Working:

Topics are arranged in a directory like structure. A topic might be **"MQ135"**, or **"MQ135/C02"** if you have multiple clients within that parent topic. The subscriber client will listen for incoming messages from the subscribed topic and react to what was published to that topic, such as **"on"** or **"off."** Clients can subscribe to one topic and publish it to another as well. If the client subscribes to **"MQ135/NOx"**, it might also want to publish to another topic like **"MQ135/NH3/Result"** so that other clients can monitor the state of that pollutant

Components Required for raspberry pi:

- Sparkfun RGB light
- MCP3008
- Aedes, Mqtt.js Broker
- Raspberry Pi 3
- USB cable



Setup Code:

```
# set GPIO pin numbering method to BCM
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
```

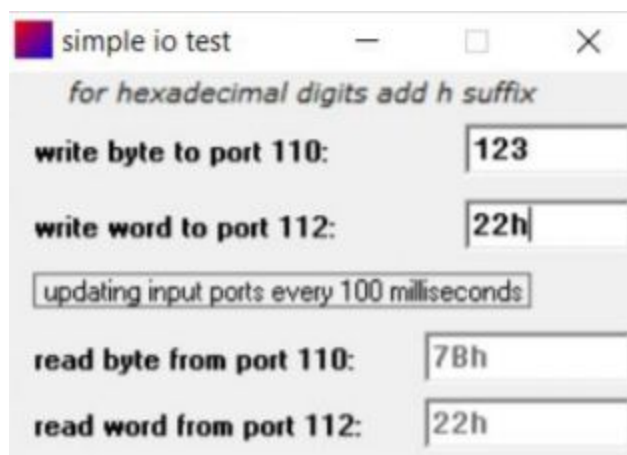
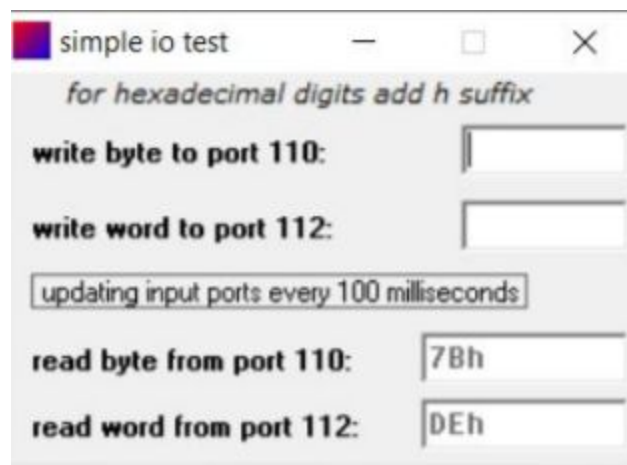
```
# define pins
#define MCP3008_PIN_CS      8
#define MQ135_3V3_PIN_AOUT 0
```

Part - III : Emu 8086 and its Outcome

NOTE : Outcomes for the previous projects are already defined within the project itself, look at complete code in the next section for more info. The current part is showing the main outcome, that being EMU8086

Software used : EMU8086 4.08 - Master Version

- The main thing in EMU8086 we want to do is, to automate the process of getting the input within its code directly from the output of the previous Arduino/Raspberry Pi.
- I made a separate program (Device) which can accept and read the input externally. For the current iteration the program is already included as a part of the new project

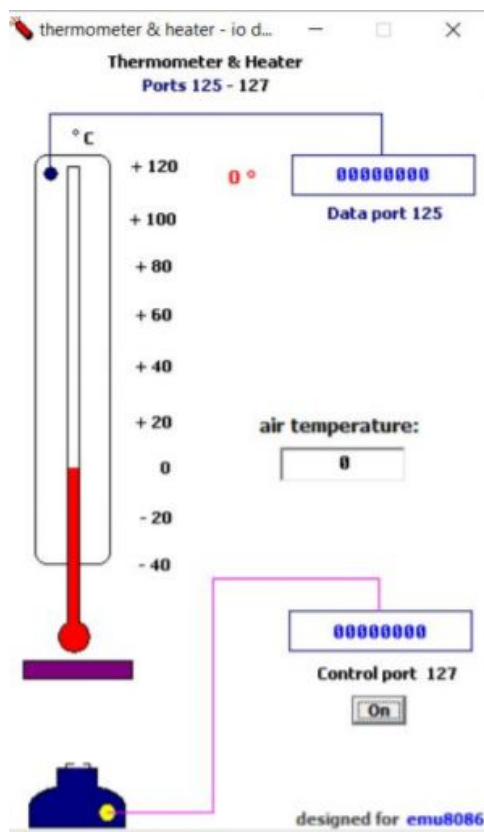


Application - 1: Greenhouse effect calculator

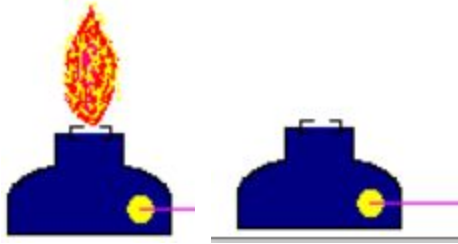
- Once I get the current air temperature, it is entered as input in the previous application. (for review - 1), but now all we need to do is enter within the program itself

air temperature:

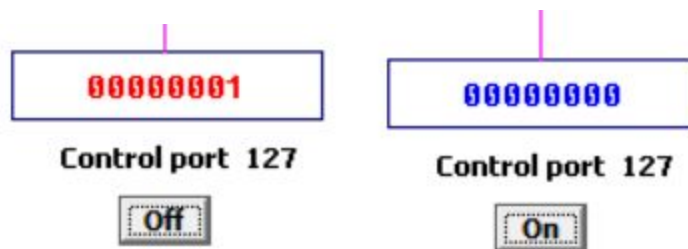
- The air temperature is then inputted in our temperature program, which is then subjected to increase based on the current growth rate. The current picture is showing the device in stop state:



- Once you turn on the calculator you get the variation in diagram as there is increase in heat below you can see turned on vs off



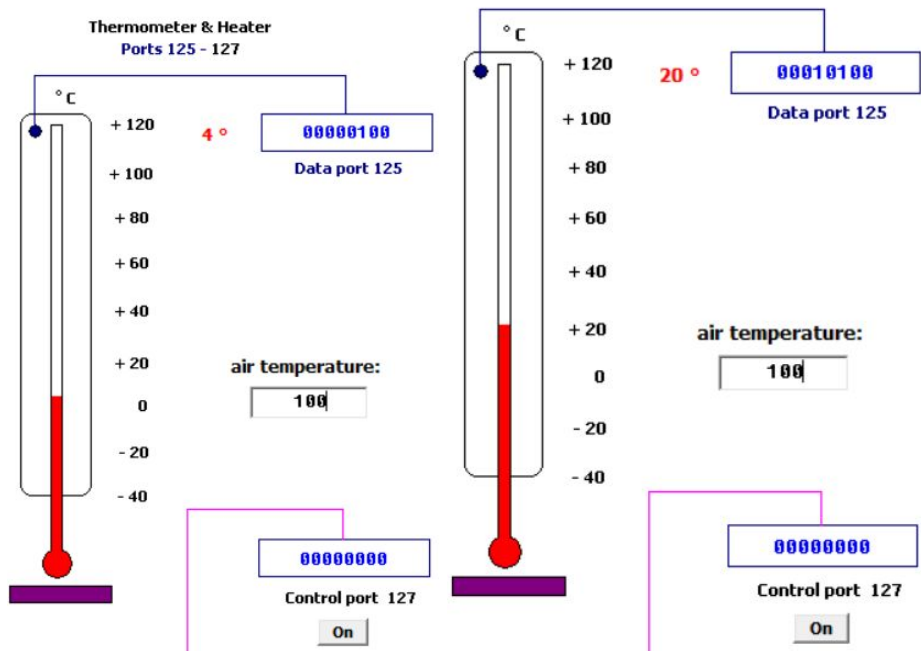
- Currently the rate of calculation is displayed in the control port, the value is set based on the inputs you get as well, by default it's 1. Here you can see the difference with it being turned on and off



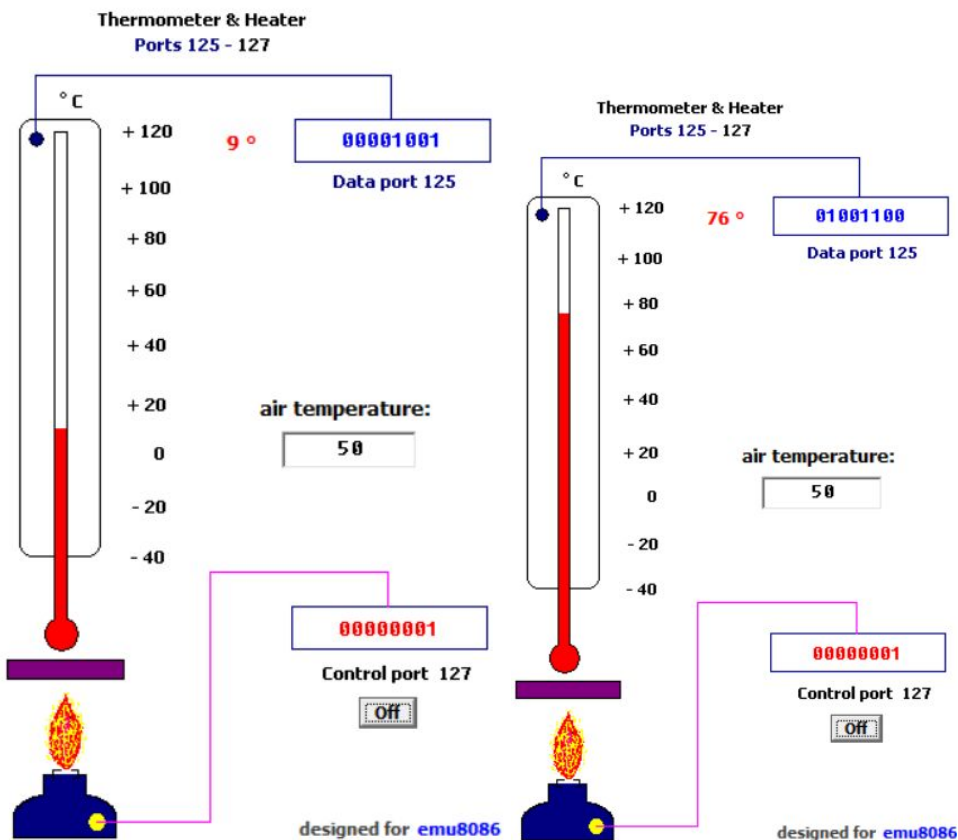
- The output is always updated in realtime in the data port, along with the current temperature of that area



- Added a function where if the air temperature is high, the temperature naturally increases without the input being turned on



- Here we can also see the increase when combined with both air temperature and input even more rapidly (the time frame is same for both above and below)

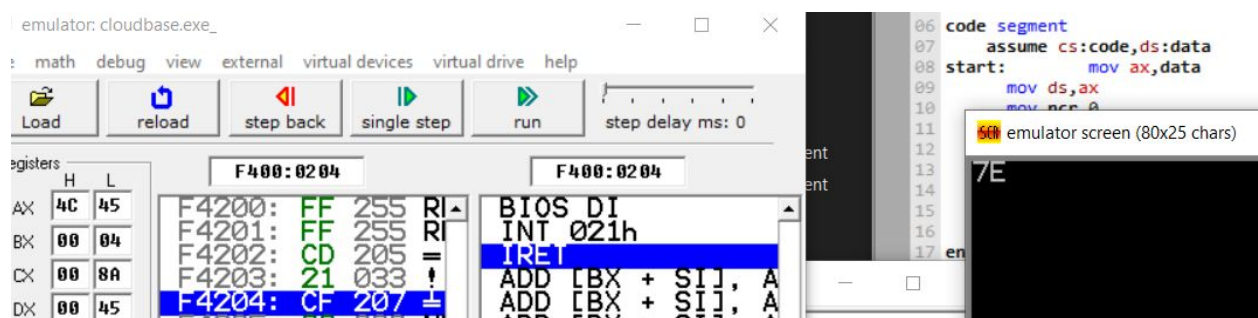


Application - 2: Simple cloud base calculation

Inputs: Temperature (From the control port)

Dew (From Node MCU or Raspberry Pi)

Output:



So 7E = 126

126,000 is the resultant cloud dew.

See complete code for more info

Support Documents (Complete codes):**Raspberry Pi: Includes RGB functionality**

```

#include <WiFi.h>
#include <PubSubClient.h>

const char *ssid = "-----";
const char *password = "-----";

const byte SWITCH_PIN = 0;           // Pin to control the Gas Sensor with
const char *ID = "IOT_Project";
const char *TOPIC = "MQ135/CO2";

IPAddress broker(192,168,1,-);
WiFiClient wclient;

PubSubClient client(wclient); // Setup MQTT client
bool state=0;

// Connect to WiFi network
void setup_wifi() {
  Serial.print("\nConnecting to ");
  Serial.println(ssid);

  WiFi.begin(ssid, password); // Connect to network

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

  Serial.println();
  Serial.println("WiFi connected");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
}

void reconnect() {
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    if (client.connect(ID)) {
      Serial.println("connected");
      Serial.print("Publishing to: ");
      Serial.println(TOPIC);
      Serial.println('\n');
    } else {
      Serial.println(" try again in 5 seconds");
      delay(5000);
    }
  }
}

void setup() {
  Serial.begin(115200);

```

```

pinMode(SWITCH_PIN, INPUT);
digitalWrite(SWITCH_PIN, HIGH);
delay(100);
setup_wifi();
client.setServer(broker, 1883);
}

void loop() {
    if (!client.connected())
    {
        reconnect();
    }
    client.loop();

    // if the switch is being pressed
    if(digitalRead(SWITCH_PIN) == 0)
    {
        state = !state; //toggle state
        if(state == 1) // ON
        {
            client.publish(TOPIC, "on");
            Serial.println((String)TOPIC + " => on");
        }
        else // OFF
        {
            client.publish(TOPIC, "off");
            Serial.println((String)TOPIC + " => off");
        }
    }

    while(digitalRead(SWITCH_PIN) == 0)
    {
        yield();
        delay(20);
    }
}
}

```

Paho MQTT Model:

Main Loop:

```

public void connect(){

    String clientId = MqttClient.generateClientId();
    final MqttAndroidClient client =
        new MqttAndroidClient(this.getContext(),
"tcp://m12.cloudmqtt.com:17389",
        clientId);

    MqttConnectOptions options = new MqttConnectOptions();
    options.setMqttVersion(MqttConnectOptions.MQTT_VERSION_3_1);
    options.setCleanSession(false);
    options.setUserName("Enter Your MQTT UserName Here");
    options.setPassword("Enter Your MQTT Password
Here".toCharArray());
    try {

```

```

IMqttToken token = client.connect(options);
//IMqttToken token = client.connect();
token.setActionCallback(new IMqttActionListener() {
    @Override
    public void onSuccess(IMqttToken asyncActionToken) {
        // We are connected
        Log.d("file", "onSuccess");
        //publish(client,"payloadd");
        subscribe(client,"dht");
        subscribe(client,"bmp");
        client.setCallback(new MqttCallback() {
            TextView tt = (TextView) findViewById(R.id.tt);
            TextView th = (TextView) findViewById(R.id.th);
            @Override
            public void connectionLost(Throwable cause) {

            }
            @Override
            public void messageArrived(String topic,
MqttMessage message) throws Exception {
                Log.d("file", message.toString());

                if (topic.equals("dht")){
                    tt.setText(message.toString());
                }

                if (topic.equals("bmp")){
                    th.setText(message.toString());
                }
            }
            @Override
            public void deliveryComplete(IMqttDeliveryToken
token) {

            }
        });
    }

    @Override
    public void onFailure(IMqttToken asyncActionToken,
Throwable exception) {
        // Something went wrong e.g. connection timeout or
        firewall problems
        Log.d("file", "onFailure");
    }
});
} catch (MqttException e) {
    e.printStackTrace();}
}

```

Node MCU Setup Code: Working code is already shown in Part - I

```
#include "Arduino.h"
#include "SparkFunISL29125.h"

// Pin Definitions
#define MQ135_3V3_PIN_AOUT A0

// Global variables and defines
uint16_t rgbSensorR,rgbSensorG,rgbSensorB;
// object initialization
SFE_ISL29125 rgbSensor;

// define vars for testing menu
const int timeout = 10000;    //define timeout of 10 sec
char menuOption = 0;
long time0;

// Setup the essentials for your circuit to work. It runs first every time your circuit is
powered with electricity.
void setup()
{
    // Setup Serial which is useful for debugging
    // Use the Serial Monitor to view printed messages
    Serial.begin(9600);
    while (!Serial) ; // wait for serial port to connect. Needed for native USB
    Serial.println("start");
    // Initialize the rgbSensor
    if (rgbSensor.init())
        Serial.println("rgbSensor Init OK!");
    else
```

```

        Serial.println("rgbSensor Init Failed! Check your wiring.");
        menuOption = menu();

    }

    // Main logic of your circuit. It defines the interaction between the components you
    // selected. After setup, it runs over and over again, in an eternal loop.
    void loop()
    {

        if(menuOption == '1')
        {
            // Disclaimer: The Hazardous Gas Sensor - MQ-135 is in testing and/or doesn't
            // have code, therefore it may be buggy.
        }
        else if(menuOption == '2') {
            // SparkFun ISL29125 - RGB Light Sensor - Test Code
            // Compare red value and blue value in rgbSensor
            rgbSensorR = rgbSensor.readRed();
            rgbSensorG = rgbSensor.readGreen();
            rgbSensorB = rgbSensor.readBlue();
            Serial.print(F("R: ")); Serial.print(rgbSensorR);
            Serial.print(F("\tG: ")); Serial.print(rgbSensorG);
            Serial.print(F("\tB: ")); Serial.println(rgbSensorB);

        }
        if (millis() - time0 > timeout)
        {
            menuOption = menu();
        }
    }
}

```

```
// Menu function for selecting the components to be tested
// Follow serial monitor for instructions
char menu()
{

    Serial.println(F("\nWhich component would you like to test?"));
    Serial.println(F("(1) Hazardous Gas Sensor - MQ-135"));
    Serial.println(F("(2) SparkFun ISL29125 - RGB Light Sensor"));
    Serial.println(F("(menu) send anything else or press on board reset button\n"));
    while (!Serial.available());

    // Read data from serial monitor if received
    while (Serial.available())
    {
        char c = Serial.read();
        if (isAlphaNumeric(c))
        {

            if(c == '1')
                Serial.println(F("Now Testing Hazardous Gas Sensor - MQ-135
- note that this component doesn't have a test code"));
            else if(c == '2')
                Serial.println(F("Now Testing SparkFun ISL29125 - RGB Light
Sensor"));
            else
            {
                Serial.println(F("illegal input!"));
                return 0;
            }
        }
    }
}
```

```
        time0 = millis();  
        return c;  
    }  
}  
}
```

EMU8086 - Cloudbase

DATA SEGMENT

n db 9

r db 5

ncr db 0

DATA ENDS

code segment

assume cs:code,ds:data

start: mov ax,data

mov ds,ax

mov ncr,0

mov al,n

mov bl,r

call encr

call display

mov ah,4ch

int 21h

encr proc

cmp al,bl

je ncr1

cmp bl,0

```
    je ncr1
    cmp bl,1
    je ncrn
    dec al
    cmp bl,al
    je ncrn1
    push ax
    push bx
    call encr
    pop bx
    pop ax
    dec bl
    push ax
    push bx
    call encr
    pop bx
    pop ax
    ret
ncr1:    inc ncr
        ret
ncrn1:   inc al
ncrn:    add ncr,al
        ret
encr endp

display proc
    push cx
```



```
    mov al,ncr
    mov ch,al
    and al,0f0h
    mov cl,04
    shr al,cl
    cmp al,09h
    jbe next
    add al,07h
next:add al,30h
    mov dl,al
    mov ah,02h
    int 21h
    mov al,ch
    and al,0fh
    cmp al,09h
    jbe next2
    add al,07h
next2:add al,30h
    mov dl,al
    mov ah,02h
    int 21h
    pop cx
ret
display endp
code ends
end start
```

EMU8086 - GreenHouse Effect Calculator (FRM CODE)(80% OF THIS IS OPEN SOURCE CODE NOT MINE)

Begin VB.Form frmThermometer

AutoRedraw = -1 'True

BackColor = &H00FFFFFF&

BorderStyle = 1 'Fixed Single

Caption = "thermometer & heater - io device"

ClientHeight = 7995

ClientLeft = 90

ClientTop = 375

ClientWidth = 4935

BeginProperty Font

Name = "Tahoma"

Size = 9.75

Charset = 0

Weight = 700

Underline = 0 'False

Italic = 0 'False

Strikethrough = 0 'False

EndProperty

Icon = "frmThermometer.frx":0000

LinkTopic = "Form1"

MaxButton = 0 'False

Picture = "frmThermometer.frx":0442

ScaleHeight = 533

ScaleMode = 3 'Pixel

ScaleWidth = 329

Begin VB.Timer TimerPort127Check

Interval = 20

Left = 4380

Top = 6585

End

Begin VB.Timer TimerAnimation

```
Interval    = 500
Left        = 2865
Top         = 4755
```

End

Begin VB.TextBox txtAirTemperature

```
Alignment    = 2 'Center
```

BeginProperty Font

```
Name        = "Fixedsys"
```

```
Size         = 9
```

```
Charset      = 0
```

```
Weight       = 400
```

```
Underline    = 0 'False
```

```
Italic       = 0 'False
```

```
Strikethrough = 0 'False
```

EndProperty

```
Height       = 360
```

```
Left         = 2805
```

```
MaxLength    = 4
```

```
TabIndex     = 5
```

```
Text         = "0"
```

```
Top          = 4065
```

```
Width        = 1290
```

End

Begin VB.CommandButton cmdOnOff

```
Caption      = "On"
```

BeginProperty Font

```
Name        = "Tahoma"
```

```
Size         = 8.25
```

```
Charset      = 0
```

```
Weight       = 700
```

```
Underline    = 0 'False
```

```
Italic       = 0 'False
```

```
Strikethrough = 0 'False
```

EndProperty

Height = 300

Left = 3540

TabIndex = 3

Top = 6600

Width = 555

End

Begin VB.Timer TimerTime

Interval = 200

Left = 3780

Top = 4740

End

Begin VB.Label lblURL

Alignment = 2 'Center

AutoSize = -1 'True

BackColor = &H00FFFFFF&

Caption = "emu8086"

BeginProperty Font

Name = "Tahoma"

Size = 8.25

Charset = 0

Weight = 700

Underline = 0 'False

Italic = 0 'False

Strikethrough = 0 'False

EndProperty

ForeColor = &H00FF0000&

Height = 195

Left = 4080

Mouselcon = "frmThermometer.frx":14C0

MousePointer = 99 'Custom

TabIndex = 7

Top = 7755

```
Width      = 825
End
Begin VB.Label Label2
    Alignment   = 2 'Center
    AutoSize    = -1 'True
    BackColor   = &H00FFFFFF&
    Caption     = "designed for"
    BeginProperty Font
        Name      = "Tahoma"
        Size      = 8.25
        Charset    = 0
        Weight     = 700
        Underline  = 0 'False
        Italic     = 0 'False
        Strikethrough = 0 'False
    EndProperty
    ForeColor   = &H00404040&
    Height      = 195
    Left        = 2970
    TabIndex    = 6
    Top         = 7755
    Width       = 1065
End
Begin VB.Label Label1
    AutoSize    = -1 'True
    BackColor   = &H00FFFFFF&
    Caption     = "air temperature:"
    ForeColor   = &H00000000&
    Height      = 240
    Left        = 2595
    TabIndex    = 4
    Top         = 3720
    Width       = 1635
```

End

Begin VB.Label lblTemperature

Alignment = 2 'Center
AutoSize = -1 'True
BackColor = &H00FFFFFF&
Caption = "20 °"
ForeColor = &H000000FF&
Height = 240
Left = 2220
TabIndex = 2
Top = 1185
Width = 420

End

Begin VB.Shape Column

BackColor = &H000000FF&
BackStyle = 1 'Opaque
BorderColor = &H000000FF&
FillColor = &H000000FF&
Height = 4725
Left = 645
Top = 1215
Width = 105

End

Begin VB.Shape FlameCover

BackStyle = 1 'Opaque
BorderColor = &H00FFFFFF&
Height = 795
Left = 315
Top = 6495
Width = 840

End

Begin VB.Label lblPort127

Alignment = 2 'Center

BackColor = &H00FFFFFF&

Caption = "00000000"

BeginProperty Font

Name = "Fixedsys"

Size = 9

Charset = 0

Weight = 400

Underline = 0 'False

Italic = 0 'False

Strikethrough = 0 'False

EndProperty

ForeColor = &H00FF0000&

Height = 225

Left = 3330

TabIndex = 1

Top = 5835

Width = 975

End

Begin VB.Label lblPort125

Alignment = 2 'Center

AutoSize = -1 'True

BackColor = &H00FFFFFF&

Caption = "00000000"

BeginProperty Font

Name = "Fixedsys"

Size = 9

Charset = 0

Weight = 400

Underline = 0 'False

Italic = 0 'False

Strikethrough = 0 'False

EndProperty

ForeColor = &H00FF0000&

Height = 225

Left = 3360

TabIndex = 0

Top = 1170

Width = 975

End

Begin VB.Image imgFire

Height = 795

Left = 315

Picture = "frmThermometer.frx":17CA

Top = 6495

Width = 795

End

End

Attribute VB_Name = "frmThermometer"

Attribute VB_GlobalNameSpace = False

Attribute VB_Creatable = False

Attribute VB_PredeclaredId = True

Attribute VB_Exposed = False

Option Explicit

Dim iTemperature As Integer ' Thermometer temperature.

Dim isFlame As Boolean ' "True" when fire is burning.

Dim bOLD_VALUE As Byte ' #1122d

Private Sub Form_Load()

' do not allow more than one copy of this program to run simuateniously

If App.PrevInstance Then

ShowPrevInstance

End ' terminate this instance!

End If

SetFlame True

Set_Temperature Val(txtAirTemperature.Text)

GetWindowPos Me

If allow_on_top() Then set_on_top Me

End Sub

Private Sub SetFlame(b As Boolean)

FlameCover.Visible = Not b

isFlame = b

If b Then

cmdOnOff.Caption = "Off"

TimerTime.Interval = 200

Else

cmdOnOff.Caption = "On"

TimerTime.Interval = 1000

End If

End Sub

Private Sub Form_Unload(Cancel As Integer)

SaveWindowState Me

End Sub

Private Sub TimerTime_Timer()

If isFlame Then

Set_Temperature iTemperature + 1

Else

If iTemperature > Val(txtAirTemperature.Text) Then

Set_Temperature iTemperature - 1

End If

If iTemperature < Val(txtAirTemperature.Text) Then

Set_Temperature iTemperature + 1

End If

End If

End Sub

' makes an integer to be a BYTE (unsigned):

Function to_unsigned_byte(i As Integer) As Byte

If i >= -128 And i <= 255 Then

If i >= 0 Then

to_unsigned_byte = i

Else

to_unsigned_byte = 256 + i

End If

```
Else
    to_unsigned_byte = 0
    Debug.Print "Wrong param calling to_unsigned_byte(): " & i
End If
End Function

Sub Set_Temperature(iVal As Integer)

    ' Our thermometer works in -65 to 120 °C range only!
    If iVal > 120 Or iVal < -65 Then Exit Sub

    iTemperature = iVal

    lblPort125.Caption = toBIN_BYTE(to_unsigned_byte(iTemperature))

    lblTemperature.Caption = iTemperature & " °"

    Dim iColumnValue As Integer
    iColumnValue = TemperatureToColumn(iTemperature)
    Column.Top = 81 + (315 - iColumnValue)
    Column.Height = iColumnValue

    ' Write to Emu8086 port #125:
    WRITE_IO_BYTE 125, to_unsigned_byte(iTemperature)

    ' HELPED US TO DRAW VALUES ON TERMOMETER
    ' If iTemperature Mod 20 = 0 Then
    '     PSet (130, Column.Top)
    ' End If

End Sub

Function Make_Min_Len(s As String, minLen As Integer, sAddWhat As String) As String
```

```
Dim i As Integer
```

```
Dim sRes As String
```

```
i = 0
```

```
sRes = s
```

```
While Len(sRes) < minLen
```

```
    sRes = sAddWhat & sRes
```

```
Wend
```

```
Make_Min_Len = sRes
```

```
End Function
```

```
' returns BINARY presentation of a number,
```

```
' return value has 8 bits (zeros & ones)
```

```
Function toBIN_BYTE(ByRef bNum As Byte) As String
```

```
Dim sHEX As String
```

```
Dim sResult As String
```

```
Dim i As Integer
```

```
Dim Size As Integer
```

```
sHEX = Hex(bNum)
```

```
Size = Len(sHEX)
```

```
sResult = ""
```

```
For i = Size To 1 Step -1
```

```
    sResult = HEX_2_BIN(Mid(sHEX, i, 1)) & sResult
```

```
Next i
```

```
toBIN_BYTE = Make_Min_Len(sResult, 8, "0")
```

End Function

' converts single HEX digit to BINARY:

```
Function HEX_2_BIN(ByRef sHEX_DIGIT As String) As String
```

```
    Select Case UCase(sHEX_DIGIT)
```

```
        Case "0"
```

```
            HEX_2_BIN = "0000"
```

```
        Case "1"
```

```
            HEX_2_BIN = "0001"
```

```
        Case "2"
```

```
            HEX_2_BIN = "0010"
```

```
        Case "3"
```

```
            HEX_2_BIN = "0011"
```

```
        Case "4"
```

```
            HEX_2_BIN = "0100"
```

```
        Case "5"
```

```
            HEX_2_BIN = "0101"
```

```
        Case "6"
```

```
            HEX_2_BIN = "0110"
```

```
        Case "7"
```

```
            HEX_2_BIN = "0111"
```

Case "8"

HEX_2_BIN = "1000"

Case "9"

HEX_2_BIN = "1001"

Case "A"

HEX_2_BIN = "1010"

Case "B"

HEX_2_BIN = "1011"

Case "C"

HEX_2_BIN = "1100"

Case "D"

HEX_2_BIN = "1101"

Case "E"

HEX_2_BIN = "1110"

Case "F"

HEX_2_BIN = "1111"

Case "h", "H" ' ignore (suffix).

HEX_2_BIN = ""

Case Else

Debug.Print "wrong argument in HEX_2_BIN(" & sHEX_DIGIT & ")"

End Select

End Function

Function TemperatureToColumn(iTemp As Integer) As Integer

Dim i As Integer

Dim a As Variant

```

a = Array(-65, -64.413, -63.826, -63.239, -62.652, -62.06499, -61.47799, -60.89099, -60.30399,
-59.71699, -59.12999, -58.54298, -57.95598, -57.36898, -56.78198, -56.19498, -55.60798,
-55.02097, -54.43397, -53.84697, -53.25997, -52.67297, -52.08596, -51.49896, -50.91196,
-50.32496, -49.73796, -49.15096, -48.56395, -47.97695, -47.38995, -46.80295, -46.21595,
-45.62894, -45.04194, -44.45494, -43.86794, -43.28094, -42.69394, -42.10693, -41.51993,
-40.93293, -40.34593, -39.75893, -39.17192, -38.58492, -37.99792, -37.41092, -36.82392,
-36.23692, -35.64991, -35.06291, -34.47591, -33.88891, -33.30191, -32.7149, -32.1279, -31.5409,
-30.9539, -30.3669, -29.7799, -29.1929, -28.6059, -28.0189, -27.4319, -26.8449, -26.2579,
-25.6709, -25.0839, -24.4969, -23.9099, -23.3229, -22.7359, -22.1489, -21.5619, -20.97491,
-20.38791, -19.80091, -19.21391, -18.62691, -18.03991, -17.45291, -16.86591, -16.27891,
-15.69191, -15.10491, -14.51791, -13.93091, -13.34391, -12.75691, _
-12.16991, -11.58291, -10.99591, -10.40891, -9.821907, -9.234907, _
-8.647907, -8.060907, -7.473907, -6.886908, -6.299908, -5.712908, -5.125908, -4.538908,
-3.951908, -3.364908, -2.777908, -2.190908, _
-1.603908, -1.016908, -0.4299084, 0.1570916, 0.7440916, 1.331092, 1.918092,
2.505092, 3.092092, 3.679091, 4.266091, 4.853091, 5.440091, 6.027091, _
6.614091, 7.201091, 7.788091, 8.375091, 8.96209, 9.54909, 10.13609, 10.72309,
11.31009, 11.89709, 12.48409, 13.07109, 13.65809, 14.24509, 14.83209, 15.41909, 16.00609,
16.59309, 17.18009, 17.76709, 18.35409, 18.94109, 19.52809, 20.11509, 20.70209, 21.28909,
21.87609, 22.46309, 23.05009, 23.63709, 24.22409, 24.81109, 25.39809, 25.98509, 26.57209,
27.15909, 27.74609, 28.33309, 28.92009, 29.50709, 30.09409, 30.68109, 31.26809, 31.85509,
32.44209, 33.02909, 33.61609, 34.20309, 34.7901, 35.3771, 35.9641, 36.5511, 37.1381, 37.72511,
38.31211, 38.89911, 39.48611, 40.07311, 40.66011, 41.24712, 41.83412, 42.42112, 43.00812,
43.59512, 44.18213, 44.76913, 45.35613, 45.94313, 46.53013, 47.11713, 47.70414, 48.29114,
48.87814, 49.46514, 50.05214, _
50.63914, 51.22615, 51.81315, 52.40015, 52.98715, 53.57415, 54.16116, 54.74816, _
55.33516, 55.92216, 56.50916, 57.09616, 57.68317, 58.27017, 58.85717, 59.44417,
60.03117, 60.61818, 61.20518, 61.79218, 62.37918, 62.96618, 63.55318, 64.14018, 64.72718,
65.31418, 65.90118, 66.48817, 67.07517, 67.66217, 68.24917, 68.83617, 69.42316, 70.01016,
70.59716, 71.18416, 71.77116, 72.35815, 72.94515, 73.53215, 74.11915, 74.70615, 75.29314,
75.88014, 76.46714, 77.05414, 77.64114, 78.22813, 78.81513, 79.40213, 79.98913, 80.57613,
81.16312, 81.75012, 82.33712, 82.92412, 83.51112, 84.09811, 84.68511, 85.27211, 85.85911,
86.44611, 87.0331, 87.6201, 88.2071, 88.7941, 89.3811, 89.96809, 90.55509, 91.14209, 91.72909,
92.31609, 92.90308, 93.49008, 94.07708, 94.66408, 95.25108, 95.83807, 96.42507, 97.01207,
97.59907, 98.18607, 98.77306, 99.36006, 99.94706, 100.5341, 101.1211, 101.7081, 102.2951,
102.882, 103.469, 104.056, 104.643, 105.23, 105.817, 106.404, 106.991, 107.578, 108.165,
108.752, 109.339, 109.926, 110.513, 111.1, _
111.687, 112.274, 112.861, 113.448, 114.035, 114.622, 115.209, 115.796, 116.383, _
116.97, 117.557, 118.144, 118.731, 119.318, 119.905, 120)

```

```
For i = LBound(a) To UBound(a)
    If a(i) >= iTemp Then
        TemperatureToColumn = i
        Exit Function
    End If
Next i
"
' ' HELPED US TO CREATE TEMPARATURE TABLE -65 TO 120
' ' VISIBLE VALUES (-40 to 120)
' ' (120 + Abs(-65)) / 315 = 0.587
' ' 315 - maximum value for the column.
' Dim k As Single
' k = -65
' For i = 0 To 315
'     Debug.Print k & ", ";
'     k = k + 0.587
' Next i
' End

End Function
```

```
Private Sub cmdOnOff_Click()
```

```
    ' Write to Emu8086 port #127:

    If isFlame Then
        WRITE_IO_BYTE 127, 0
    Else
        WRITE_IO_BYTE 127, 1
    End If

End Sub
```



```
Private Sub TimerAnimation_Timer()  
    If isFlame Then  
        imgFire.Visible = Not imgFire.Visible  
    Else  
        imgFire.Visible = False  
    End If  
End Sub
```

```
Private Sub TimerPort127Check_Timer()  
On Error GoTo err1
```

```
    Dim m As Byte
```

```
    m = READ_IO_BYTE(127)
```

```
    If bOLD_VALUE <> m Then ' #1122d  
        If shall_activate(Me) Then Me.Show  
        bOLD_VALUE = m  
    End If
```

```
    ' Checking lower bit:
```

```
    If (m And 1) Then  
        SetFlame True  
        lblPort127.ForeColor = vbRed  
    Else  
        SetFlame False  
        lblPort127.ForeColor = vbBlue  
    End If
```

```
    lblPort127.Caption = toBIN_BYTE(m)
```

```
Exit Sub
```

err1:

 Debug.Print "err1"

 Resume Next

End Sub

\

EMU8086 - Greenhouse Calculator (vbp code)

Type=Exe

Reference=*\G{00020430-0000-0000-C000-000000000046}#2.0#0#C:\WINDOWS\system32\stdole2
.tlb#OLE Automation

Form=frmThermometer.frm

Module=io; io.bas

Module=modSINGLE_INSTANCE; modSINGLE_INSTANCE.bas

Module=mWinState; mWinState.bas

Module=modAlwaysOnTop; modAlwaysOnTop.bas

IconForm="frmThermometer"

Startup="frmThermometer"

HelpFile=""

Title="Thermometer"

ExeName32="Thermometer.exe"

Command32=""

Name="Thermometer"

HelpContextID="0"

CompatibleMode="0"

MajorVer=3

MinorVer=0

RevisionVer=7

AutoIncrementVer=0

ServerSupportFiles=0

VersionCompanyName="www.emu8086.com"

CompilationType=0

OptimizationType=0

FavorPentiumPro(tm)=0

CodeViewDebugInfo=0

NoAliasing=0

BoundsCheck=0

OverflowCheck=0

FIPointCheck=0

FDIVCheck=0

UnroundedFP=0

StartMode=0

Unattended=0

Retained=0

ThreadPerObject=0

MaxNumberOfThreads=1

[MS Transaction Server]

AutoRefresh=0

EMU8086 - Greenhouse Calculator (vbw code)

frmThermometer = 66, 66, 596, 403, C, 88, 88, 618, 425, C

io = 22, 22, 552, 359, C

modSINGLE_INSTANCE = 0, 0, 0, 0, C

mWinState = 0, 0, 0, 0, C

modAlwaysOnTop = 44, 44, 574, 381, C

EMU8086 - Greenhouse Calculator (asm code)(NOT actual code, only ideology and main calculaton)

mov ax, cs

mov ds, ax

start:

in al, 125

cmp al, 60

jl low

cmp al, 80

jle ok

jb high

low:

mov al, 1

out 127, al ; turn heater "on".

jmp ok

high:

mov al, 0

out 127, al ; turn heater "off".

ok:

jmp start ; endless loop.

Output of sensors/ EMU 8086 input : Set - A, Set - B respectively

Trigger(s)	C02	Nox	H2O
No	1.2	2.1	13
No	1.5	2.2	14
No	1.6	2.1	14

Time	H2O	Trigger
11:50	22	0
12:00	25	0
12:05	23	1