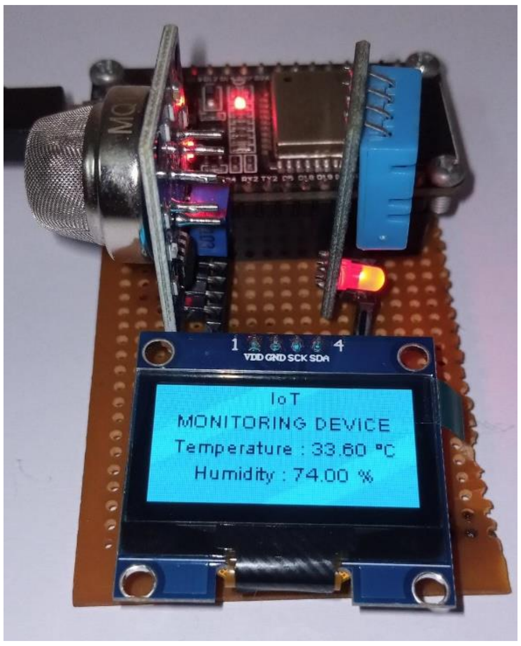
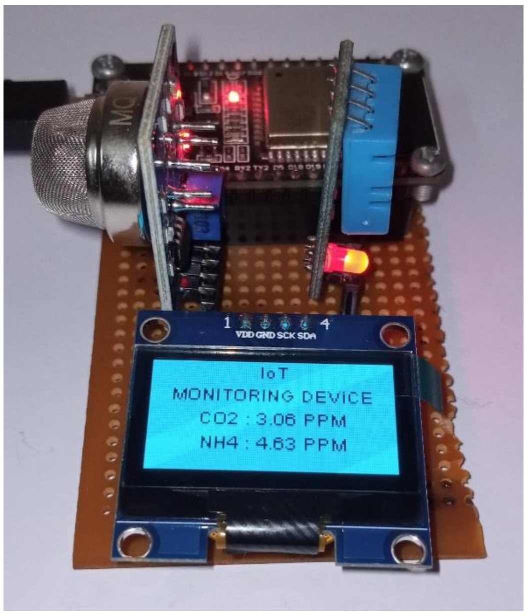
Semester VI (SPRING-2022)

IoT Monitoring Device

EE-332 Integrated Electronics





Instructor: Dr. Babar Khan

UIT-NED

Abstract

A smart city enables the effective utilization of resources and better quality of services to the citizens. To provide services such as air quality management, weather monitoring and automation of homes and buildings in a smart city, the basic parameters are temperature, humidity, and CO2.

Introduction

In this modern era, a new disaster has been discovered named as climatic change. The affect produced by this can produce massive destruction to human lives as well as other living things if not considered seriously. It includes some basic factors such as temperature, humidity, and gas quality. If these factors are not monitored correctly, it can cost huge loss.

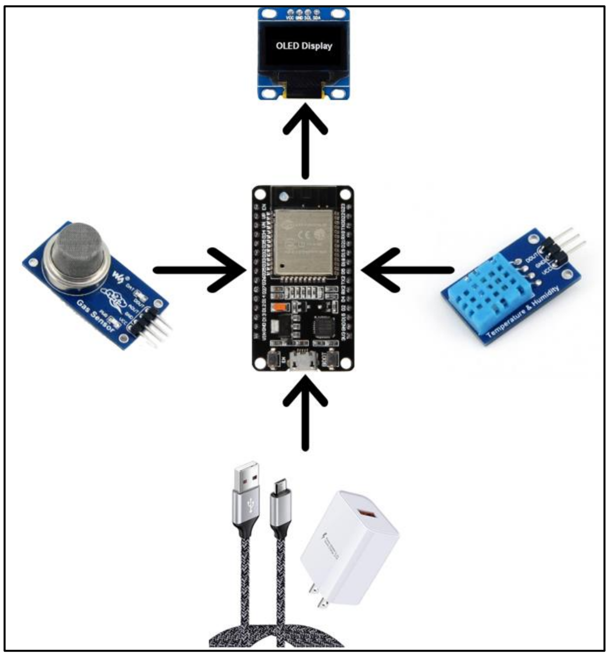
Description

To overcome the above-mentioned problem, an IoT based monitoring device is going to be developed which would not only be cost-efficient, but it would also be user friendly having utmost durability. The reason of these betterments is that a complete product would be provided with modules embedded in the system.

Components

- ESP32 Microcontroller
- DHT11 Temperature and Humidity Sensor
- MQ135 Air Quality Detector
- OLED Display 1.3 Inch SH1106
- Veroboard
- Data Cable USB 2.0 – Micro USB
- USB 2.0 adapter with plug
- Veroboard
- Header Pin

Block Diagram



Connections

- Connect VCC pin of OLED, DHT11, and MQ135 with 3V3 pin of ESP32.
- Connect GND pin of OLED, DHT11, and MQ135 with GND pin of ESP32.
- Connect DHT11's DATA pin with GPIO15 pin of ESP32.
- Connect MQ135's Aout pin with GPIO17 pin of ESP32.
- Connect SDA (Serial Data) pin of OLED with ESP32's GPIO21 pin.
- Connect SCL (Serial Clock) pin of OLED with ESP32's GPIO22 pin.
- Insert the micro-USB port of data cable into micro-USB adapter of ESP32.
- Insert the USB 2.0 port of data cable into USB 2.0 adapter with plug.

Code

```
//Include the libraries
#include "DHTesp.h"          // DHT Enable
#include <Ticker.h>          // Temperature Sync
#include <MQUnifiedsensor.h>  // MQ135 Enable
#include "SH1106Wire.h"       // OLED SH1106 1.3 Inch Enable
#include <Wire.h>            // SPI Communication Protocol Enable

#ifndef ESP32
#pragma message(THIS EXAMPLE IS FOR ESP32 ONLY!)
#error Select ESP32 board.
#endif

//Definitions
#define board ("ESP-32")         // Development Board
#define Voltage_Resolution 3.3    // VCC
#define pin 15                   // Analog Input 0 of ESP32
#define type "MQ-135"            // MQ135
#define ADC_Bit_Resolution 12     // For ESP32
#define RatioMQ135CleanAir 3.6    // RS / R0 = 3.6 ppm

//Declare Sensor
MQUnifiedsensor MQ135(board, Voltage_Resolution, ADC_Bit_Resolution, pin, type);
SH1106Wire display(0x3c, 21, 22); // OLED instance with address and SDA and SCL pins
DHTesp dht;
void tempTask(void *pvParameters);
bool getTemperature();
void triggerGetTemp();
bool initTemp();

/** Task handle for the light value read task */
TaskHandle_t tempTaskHandle = NULL;
/** Ticker for temperature reading */
Ticker tempTicker;
/** Flag if task should run */
bool tasksEnabled = false;
/** Pin number for DHT11 data pin */
int dhtPin = 17;

/**
 * initTemp
 * Setup DHT library
 * Setup task and timer for repeated measurement
 * @return bool
 *    true if task and timer are started
 *    false if task or timer couldn't be started
 **/
bool initTemp()
{
  byte resultValue = 0;
  // Initialize temperature sensor
  dht.setup(dhtPin, DHTesp::DHT11);
  Serial.println("DHT initiated");

  // Start task to get temperature
  xTaskCreatePinnedToCore
  (
    tempTask,                        /* Function to implement the task */
    "tempTask ",                     /* Name of the task */
    4000,                            /* Stack size in words */
    NULL,                            /* Task input parameter */
    5,                               /* Priority of the task */
    &tempTaskHandle,                 /* Task handle. */
    1);                              /* Core where the task should run */

  if (tempTaskHandle == NULL)
  {
    Serial.println("Failed to start task for temperature update");
    return false;
  }
  else
  {
    // Start update of environment data every 5 seconds
    tempTicker.attach(5, triggerGetTemp);
  }
  return true;
}

/**
 * triggerGetTemp
 * Sets flag dhtUpdated to true for handling in loop()
 * called by Ticker getTempTimer
 **/
void triggerGetTemp()
{
  if (tempTaskHandle != NULL)
  {
     xTaskResumeFromISR(tempTaskHandle);
  }
}

/**
 * Task to reads temperature from DHT11 sensor
 * @param pvParameters
 *    pointer to task parameters
 **/
void tempTask(void *pvParameters)
{
  Serial.println("tempTask loop started");
  while (1) // tempTask loop
  {
    if (tasksEnabled)
    {
      // Get temperature values
      getTemperature();
    }
    // Got sleep again
    vTaskSuspend(NULL);
  }
}

/**
 * getTemperature
 * Reads temperature from DHT11 sensor
 * @return bool
 *    true if temperature could be aquired
 *    false if aquisition failed
**/
bool getTemperature() {
  // Reading temperature for humidity takes about 250 milliseconds!
  // Sensor readings may also be up to 2 seconds 'old' (it's a very slow sensor)
  TempAndHumidity newValues = dht.getTempAndHumidity();
  // Check if any reads failed and exit early (to try again).
  if (dht.getStatus() != 0) {
    Serial.println("DHT11 error status: " + String(dht.getStatusString()));
    return false;
  }

  display.clear();
  display.setTextAlignment(TEXT_ALIGN_CENTER);
  display.drawString(64, 0, "IoT" );
  display.setTextAlignment(TEXT_ALIGN_CENTER);
  display.drawString(64, 14, "MONITORING DEVICE" );
  display.setTextAlignment(TEXT_ALIGN_CENTER);
  display.drawString(64, 28, "Temperature : " + String(newValues.temperature) + " °C" );
  display.setTextAlignment(TEXT_ALIGN_CENTER);
  display.drawString(64, 42, "Humidity : " + String(newValues.humidity) + " %" );
  display.display();
  return true;
}

void setup()
{
  //Init the serial port communication - to debug the library
  Serial.begin(115200);
  Wire.begin();
  display.init();
  display.setFont(ArialMT_Plain_10);
  display.invertDisplay();
  display.flipScreenVertically();
  //Set math model to calculate the PPM concentration and the value of constants
  MQ135.setRegressionMethod(1); //_PPM =  a*ratio^b
  MQ135.setA(110.47); MQ135.setB(-2.862); // Configure the equation to to calculate C02 concentration

  /*
    Exponential regression:
  GAS      | a      | b
  CO       | 605.18 | -3.937
  Alcohol  | 77.255 | -3.18
  CO2      | 110.47 | -2.862
  Toluen   | 44.947 | -3.445
  NH4      | 102.2  | -2.473
  Aceton   | 34.668 | -3.369
  */
  MQ135.init();
  Serial.print("Calibrating please wait.");
  float calcR0 = 0;
  for(int i = 1; i<=10; i ++)
  {
    MQ135.update(); // Update data, the arduino will read the voltage from the analog pin
    calcR0 += MQ135.calibrate(RatioMQ135CleanAir);
    Serial.print(".");
  }
  MQ135.setR0(calcR0/10);
  Serial.println("  done!.");

  if(isinf(calcR0)) {Serial.println("Warning: Conection issue, R0 is infinite (Open circuit detected) please check
your wiring and supply"); while(1);}
  if(calcR0 == 0){Serial.println("Warning: Conection issue found, R0 is zero (Analog pin shorts to ground)
please check your wiring and supply"); while(1);}
  MQ135.serialDebug(true);

  Serial.println();
  Serial.println("DHT ESP32 example with tasks");
  initTemp();
  // Signal end of setup() to tasks
  tasksEnabled = true;
}

void loop()
{
  if (!tasksEnabled)
  {
    // Wait 2 seconds to let system settle down
    // Enable task that will read values from the DHT sensor
    tasksEnabled = true;
    if (tempTaskHandle != NULL)
    {
      vTaskResume(tempTaskHandle);
    }
  }

  yield();
  printDisplay();
  delay(5000);
}


void drawProgressBarDemo()
{
  float NH4, CO2, ALC;
  for(int i = 0; i<=3; i++)
  {
    switch (i)
    {
     case 1:
            MQ135.setA(110.47); MQ135.setB(-2.862); // Configure the equation to to calculate C02 concentration
            MQ135.update(); // Update data, the arduino will read the voltage from the analog pin
            MQ135.readSensor(); // Sensor will read PPM concentration using the model, a and b values set
previously or from the setup
            CO2 = MQ135.ppmprint(1);
            Serial.print("CO2 : ");
            break;
    case 2:
          MQ135.setA(102.2); MQ135.setB(-2.473); // Configure the equation to to calculate NH4 concentration
          MQ135.update(); // Update data, the arduino will read the voltage from the analog pin
          MQ135.readSensor(); // Sensor will read PPM concentration using the model, a and b values set
previously or from the setup
          NH4 = MQ135.ppmprint(1);
          Serial.print("NH4 : ");
          break;
    }
  }

  // MQ135.update(); // Update data, the arduino will read the voltage from the analog pin
  // MQ135.readSensor(); // Sensor will read PPM concentration using the model, a and b values set previously
or from the setup

  // float h = lightMeter.readLightLevel();
  display.setTextAlignment(TEXT_ALIGN_CENTER);
  display.drawString(64, 0, "IoT" );
  display.setTextAlignment(TEXT_ALIGN_CENTER);
  display.drawString(64, 14, "MONITORING DEVICE" );
  display.setTextAlignment(TEXT_ALIGN_CENTER);
  display.drawString(64, 28, "CO2 : "+ String(CO2) + " PPM" );
  display.setTextAlignment(TEXT_ALIGN_CENTER);
  display.drawString(64, 42, "NH4 : " + String(NH4) + " PPM" );
}

void printDisplay()
{
  display.clear();
  drawProgressBarDemo();
  display.display();
}
```

Working

When the power supply is given to the microcontroller, it instantly supplies 3.3 voltages to the other three modules. The OLED display could be seen displaying project title along with temperature and humidity and after a delay of five seconds the display of project title along with CO_2 and NH_4 is seen. The OLED is working on the principal of I2C communication protocol of Arduino for which it gets serial data from sensor and detector at every five seconds which is synced along with the serial clock and the displays interchanged after every five seconds. The sensor senses temperature and humidity and convert it into digital signal, on the other hand, the detector detects the CO_2 and NH_4 and convert it into analog signal. Both the signals are transmitted to the microcontrollers that are processed to be displayed on the OLED.

Results

Since, the purpose of the project is to develop a device, so Veroboard is used instead of PCB. The reason behind this is that in Veroboard header pins are used to integrate modules with each other where the connection is established between the header pins not with the module pins which cannot be achieved in PCB that gives the user easy access to detach and attach the modules.

Then, the readings that are obtained from the sensor and detector are slightly different from the actual readings that realizes to use better version of these modules if this project is to be modified in the future.

Finally, the shape of the project cannot be made into device because its packaging is quite complex which can only be done professionally, somehow, the placement of every component is made good enough to be suited with the objective.

Conclusion

It is concluded that, the students learned a lot about the integration of modules and the concepts learned in the course are applied successfully. The environmental factors are sensed, and the air quality is detected easily which is then processed and modified to be display on the screen. The interfacing between different technologies is completed. On the contrary, this project can be used at the residential as well as commercial level if put into casing which is a great achievement.

Cost Estimation

Serial Number	Components	Unit Price (PKR)	Quantity	Total Price (PKR)
1	ESP32	850.00	1	850.00
2	DHT11	230.00	1	230.00
3	MQ135	280.00	1	280.00
4	OLED 1.3 Inch	680.00	1	680.00
5	Veroboard	50.00	1	50.00
6	Header Pins	10.00	1	10.00
7	Data Cable	200.00	1	200.00
8	Miscellaneous	0.00	1	0.00

The total approximate cost of the project is about 2,300 PKR.

Motivation

This motivation has been identified through articles, research papers and it is included in our FYP Project.

https://ieeexplore.ieee.org/abstract/document/7562757
https://ieeexplore.ieee.org/abstract/document/5565120
https://ieeexplore.ieee.org/abstract/document/7218144

Group Members

Name of the Student	Group Members
Ammar Bin Amir	19B-004-EE
Maaz Siddiqui	19B-008-EE
Khallil Rehman	19B-028-EE
Muhammad Farhan Ur Rehman Malik	19B-036-EE