UNLV

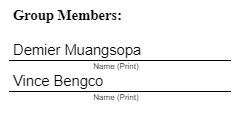
**University of Nevada Las Vegas**

**Department of Electrical and Computer Engineering**

**CPE 301L**

Spring 2024

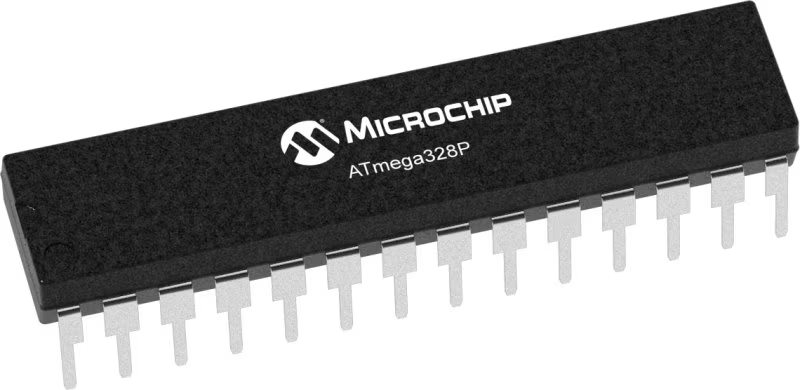
Humidity, Temperature, and Gas Detector



**Introduction**

The project is to create a cost-effective temperature and gas detector that can be utilized in tight enclosed spaces. The key components utilized in the project are the ATmega328P microcontroller, DHT11, MQ-135, capacitors, MAX232N, and a standard USB to RS232 cable for UART.

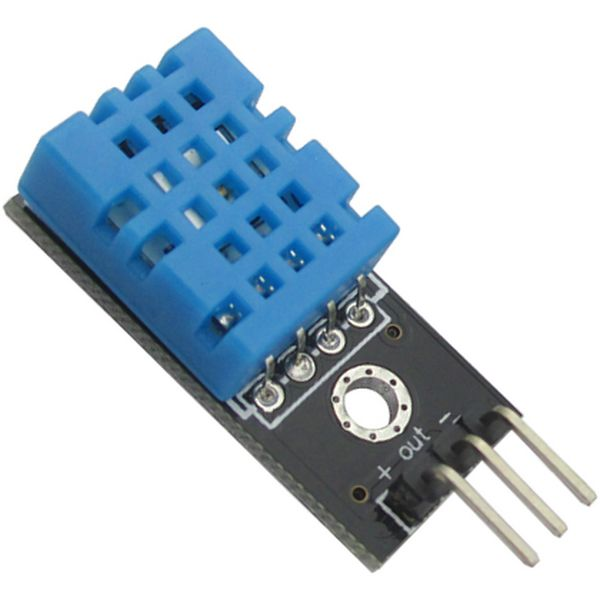
**The ATMEGA328P Microcontroller**

****

First and foremost, we need to be able to utilize a simple CPU design unit that's able to take commands accordingly. This is where the ATMEGA328P microcontroller comes in. The microcontroller can execute powerful instructions in a single cycle. The specifications come down to being able to execute 1MIPS (Million Instructions Per Second) per 1 MHz. Due to the specifications, the designers can optimize the device for power consumption vs processing speed.

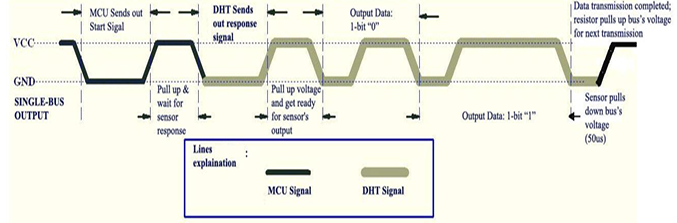
For our design, the microcontroller is used to control essentially all of our circuits, which includes reading the sensors and handling the data read from the sensors to transmit serially to any component. For the case of serial communication, we will utilize a terminal on a PC capable of understanding the transmission.

**The DHT11 Sensor**

****

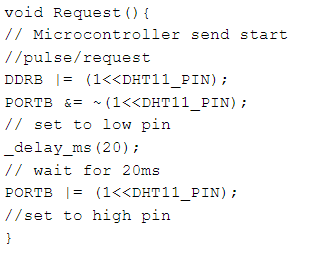
The sensor is a basic, low-cost temperature and humidity sensor. Humidity is given as a percentage ranging from 20 to 90% RH with a tolerance of +/- 5%, whereas the temperature values are given in Celcius ranging from 0 to 50 degrees. To utilize the sensor correctly, we must poll the sensor to get accurate readings from it.

When power is first given to the sensor, we must wait for one full second due to the instability of the power-up state. Once there we can then signal the microcontroller to send a start signal to the sensor.



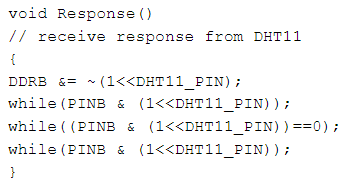
*Figure 1*

Figure 1 shows how the microcontroller interacts with the sensor. First, the microcontroller has to send a start signal, which tells the sensor the microcontroller is requesting the data from the sensor.



*Request Code*

From there the DHT11 Sensor sends a response signal by first pulling up the voltage to signify where the data is starting. After the sensor pulls down the bus voltage, the microcontroller has all the data it needs for that poll.



*Response Code*

The data received from the sensor is given as 40 bits

* Bits 7:0 - 8-bit Integer Humidity Data
* Bits 15:8 - 8-bit Decimal (Fractional Portion) Humidity Data
* Bits 23:16 - 8-bit Integer Temperature Data
* Bits 31:24 - 8-bit Decimal (Fractional Portion) temperature data
* Bits 39:32 - 8-bit CheckSum Data for Error Detection

The datasheet specifies that we should not sample the sensor for less than 1 second, otherwise errors will occur, or output will not be the desired outcome.

**The MQ-135 Sensor**

****

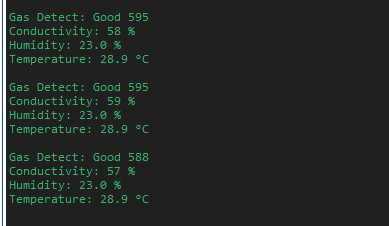
According to the datasheet, the MQ-135 is a sensor that can detect and measure gasses like ammonia, benzene, sulfur, carbon dioxide, smoke, and other harmful gases. The sensor itself has two outputs: analog out and digital out. The analog out pin is the actual analog value given as 10 bits, so we must read ADCH, and ADCL to get all the appropriate values. The digital out pin is utilized as a limitation, where if the gas exceeds the threshold limit, the digital output pin then goes high.

In our design, we had to convert the values to digital to read the data, so we used ADC0 on the microcontroller to read the data. The way we executed the ADC conversion was through an interrupt when the conversion was complete.

**Final Note**: The datasheet says to utilize a load resistor of around 10K ohms to 47K ohms where the higher resistance causes the sensor to become more sensitive to the inputs. However, upon realizing the PCB module of the MQ-135, we realize that the SMD load resistor is 102, making it a 1K ohm load resistor which was probably the culprit for the inaccuracies in our demonstration/results.

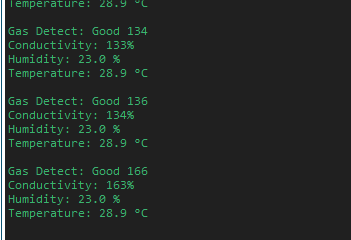
****

**Results**

****

*Figure 2*

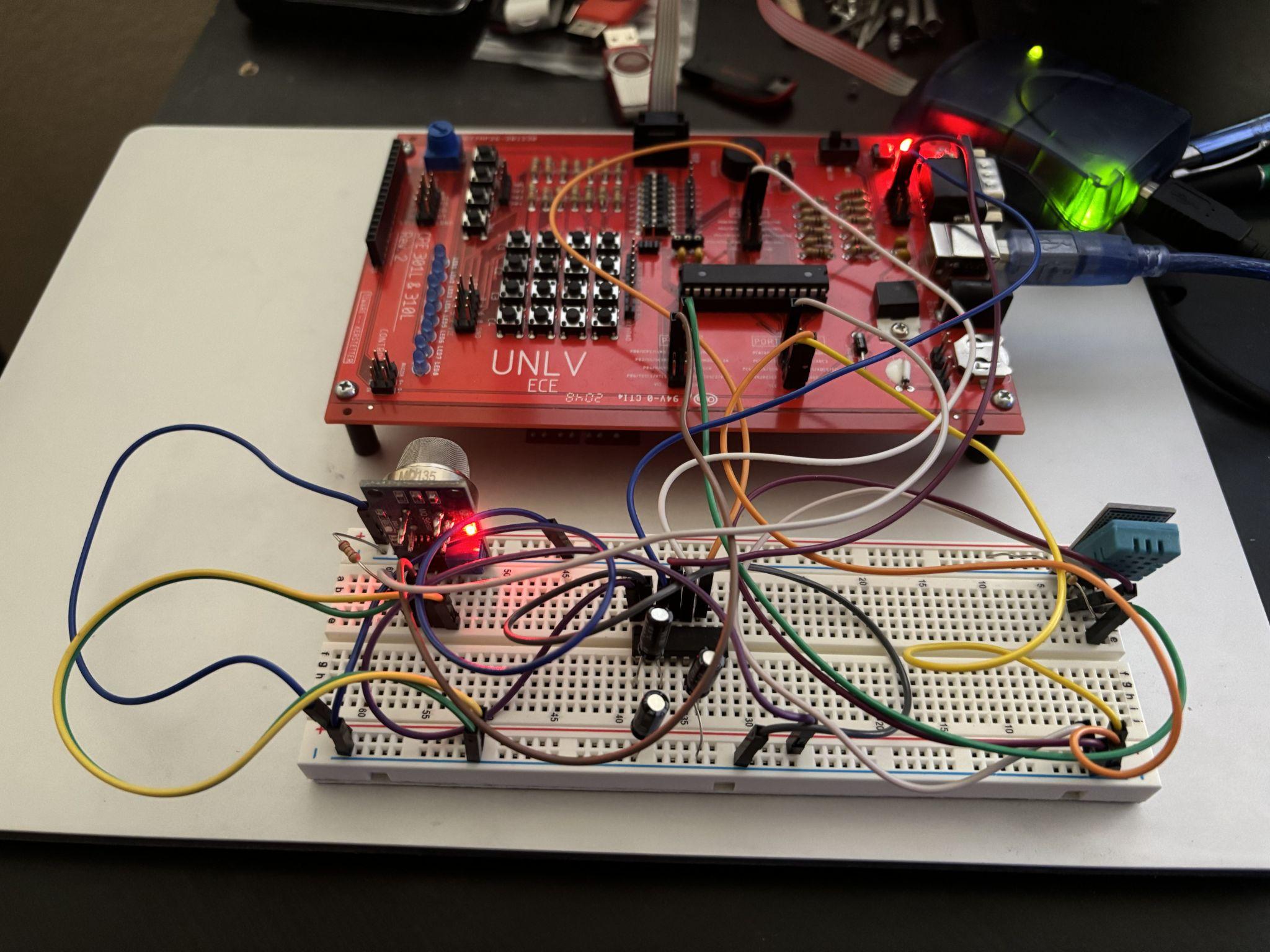
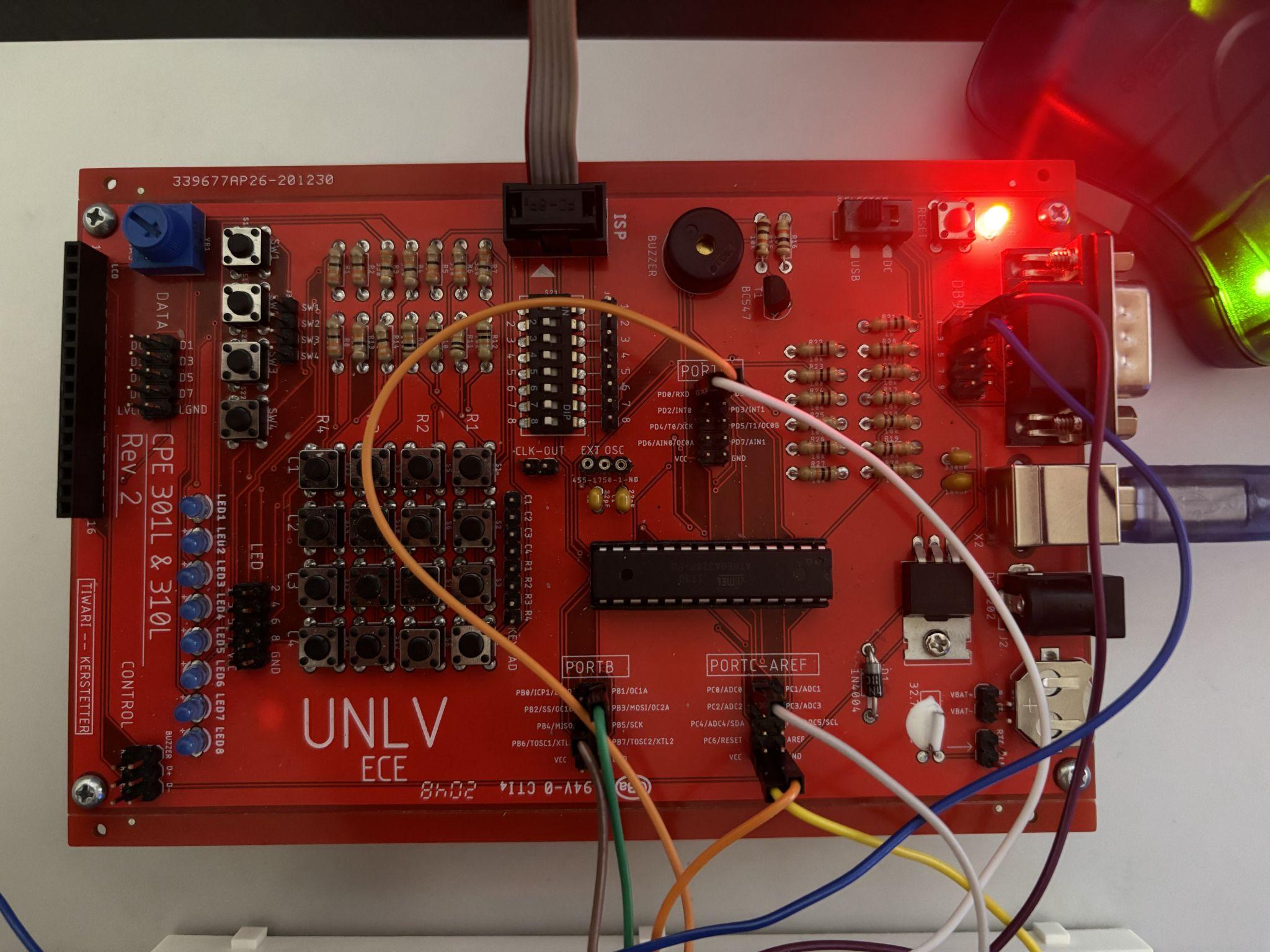
To test the DHT11 sensor, we can utilize our hands and body heat/sweat to detect the temperature/humidity changes. For the MQ-135, we will utilize drinking alcohol to detect the gas fumes in the air.

****

*Figure 3*

From Figures 2 and 3, the numbers beside the gas detection differ because the latter figure is sensing alcohol, whereas in the normal state, the detection sits around 600 ppm. The conductivity shows that the sensor is indeed sensing something, which is the alcohol.

**Board and Wiring**



**Conclusion**

So the biggest obstacle we faced when interfacing with the MQ-135 was the load resistance. We didn’t realize before it was too late that the load resistance was already integrated into the PCB module of the gas sensor itself. And since the load resistor does not match the datasheet specifications, then the resulting data should be inaccurate. Although the data is inaccurate, we can clearly see that the sensor is sensing gas, but not as we intended. Consequently, this affected our air quality detection and determining whether the air was good or bad.

**References**

1. “Dynamic Product Page | Microchip Technology.” Www.microchip.com,  
   [www.microchip.com/en-us/product/ATmega328P#document-table](http://www.microchip.com/en-us/product/ATmega328P#document-table).
2. DHT11 Humidity & Temperature Sensor, [www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf](http://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf).
3. MQ-135 GAS SENSOR, [www.olimex.com/Products/Components/Sensors/Gas/SNS-MQ135/resources/SNS-MQ135.pdf](http://www.olimex.com/Products/Components/Sensors/Gas/SNS-MQ135/resources/SNS-MQ135.pdf).

**Other Resources**

Presentation link: <https://docs.google.com/presentation/d/1NWq3kMcucMOSBF30szlAUnThHF-vEfYoAWHSNy6FHkk/edit?usp=sharing>

Code link (screenshots below for reference):

<https://drive.google.com/file/d/1_fFp0quFbeQyNn5Ol81573QI39Sy020S/view?usp=drive_link>

**Code**

***/\****

***\* Final\_Project.c***

***\****

***\* Created: 4/19/2024 9:24:25 AM***

***\* Author : bengcov***

***\*/***

***#define F\_CPU 8000000UL***

***#include <avr/io.h>***

***#include <stdlib.h>***

***#include <stdio.h>***

***#include <util/delay.h>***

***#include <avr/interrupt.h>***

***#include <stdbool.h>***

***#define DHT11\_PIN 1***

***uint8\_t c=0,I\_RH,D\_RH,I\_Temp,D\_Temp,CheckSum;***

***uint16\_t digital\_data;***

***unsigned int conductivity;***

***unsigned char received\_data;***

***bool flag;***

***void print\_gas();***

***void print\_temp();***

***void print\_humidity();***

***void print\_warning();***

***void print\_poor();***

***void print\_moderate();***

***void print\_good();***

***void print\_load();***

***void print\_conductivity();***

***////////////////////////////////////////////////////////////////////////***

***void UART\_setup()***

***{***

***UCSR0B |= (1<<TXEN0) | (1<<RXEN0); //enable transmitter, receiver function (UCSR0B)***

***UCSR0C |= (1<<UCSZ01) | (1<<UCSZ00); //set 8 bit, 1stop bit, disabled parity, asynchronous USART (UCSR0C)***

***UBRR0H = 0x00;***

***UBRR0L = 0x33; //select UBRR0L/H for 8Mhz 9600 baud, check data sheet***

***}***

***////////////////////////////////////////////////////////////////////////***

***void send(int data)***

***{***

***//check if buffer is empty so that data can be written to transmit***

***while ( !(UCSR0A & (1<<UDRE0)));***

***UDR0 = data; //copy “data” to be sent to UDR0***

***}***

***////////////////////////////////////////////////////////////////////////***

***void ADC\_Init() {***

***//Set channel to take input for ADC0, set ALDAR, AVcc with external cap at AREF***

***ADMUX |= (1<<REFS0) | (1<<ADLAR);***

***//0x0110 0000***

***//Set pre-scaler to 128, enable ADC interrupt, enable ADC, start conversion***

***ADCSRA |= (1<<ADEN) | (1<<ADSC) |(1<<ADIE) | (1<<ADPS2) | (1<<ADPS1) | (1<<ADPS0);***

***//1100 1111***

***sei();***

***}***

***////////////////////////////////////////////////////////////////////////***

***ISR (ADC\_vect)***

***{***

***digital\_data = ADC; // stores data from ADC***

***ADCSRA |=(1<<ADSC); // start conversion***

***}***

***////////////////////////////////////////////////////////////////////////***

***void Request() // Micro-controller send start pulse/request***

***{***

***DDRB |= (1<<DHT11\_PIN);***

***PORTB &= ~(1<<DHT11\_PIN); // set to low pin***

***\_delay\_ms(20); // wait for 20ms***

***PORTB |= (1<<DHT11\_PIN); // set to high pin***

***}***

***////////////////////////////////////////////////////////////////////////***

***void Response() // receive response from DHT11***

***{***

***DDRB &= ~(1<<DHT11\_PIN);***

***while(PINB & (1<<DHT11\_PIN));***

***while((PINB & (1<<DHT11\_PIN))==0);***

***while(PINB & (1<<DHT11\_PIN));***

***}***

***////////////////////////////////////////////////////////////////////////***

***uint8\_t Receive\_data() // receive data***

***{***

***for (int q=0; q<8; q++)***

***{***

***while((PINB & (1<<DHT11\_PIN)) == 0); // check received bit 0 or 1***

***\_delay\_us(30);***

***if(PINB & (1<<DHT11\_PIN)) // if high pulse is greater than 30ms***

***c = (c<<1)|(0x01); // then its logic HIGH***

***else // otherwise its logic LOW***

***c = (c<<1);***

***while(PINB & (1<<DHT11\_PIN));***

***}***

***return c;***

***}***

***////////////////////////////////////////////////////////////////////////***

***int main(void)***

***{***

***UART\_setup();***

***ADC\_Init();***

***uint16\_t Ttemp = 0xFFFF; // Temperature temporary variable***

***uint16\_t Htemp = 0xFFFF; // humidity temporary variable***

***uint16\_t Data\_temp = 0;***

***flag = false;***

***while(1)***

***{***

***char storage[8];***

***Request(); // send start pulse***

***Response(); // receive response***

***I\_RH=Receive\_data(); // store first eight bit in I\_RH***

***D\_RH=Receive\_data(); // store next eight bit in D\_RH***

***I\_Temp=Receive\_data(); // store next eight bit in I\_Temp***

***D\_Temp=Receive\_data(); // store next eight bit in D\_Temp***

***CheckSum=Receive\_data();// store next eight bit in CheckSum***

***// parity/data check***

***if ((I\_RH + D\_RH + I\_Temp + D\_Temp) != CheckSum)***

***{***

***send('E');***

***send('r');***

***send('r');***

***send('o');***

***send('r');***

***}***

***// module for printing data via UART***

***else***

***{***

***Data\_temp = (digital\_data & 0xFF);***

***// check if air quality is poor***

***if (Data\_temp >= 400)***

***{***

***// print warning message***

***print\_gas();***

***send('\n');***

***print\_warning();***

***print\_poor();***

***flag = true;***

***}***

***// check if humidity or temperature changes or bad air***

***if ( (Htemp != ((8 << I\_RH) | D\_RH)) | (Ttemp != ((8 << I\_Temp) | D\_Temp)) | flag )***

***{***

***// if gas is good/moderate/loading***

***if (!flag)***

***{***

***// print gas detect info***

***print\_gas();***

***itoa(digital\_data,storage,10);***

***if (Data\_temp >= 300)***

***{***

***print\_moderate();***

***send(storage[0]);***

***send(storage[1]);***

***send(storage[2]);***

***}***

***else***

***{***

***print\_good();***

***send(storage[0]);***

***send(storage[1]);***

***send(storage[2]);***

***}***

***send('\n');***

***}***

***/////////////////////***

***// print conductivity (overall air quality)***

***flag = false;***

***print\_conductivity();***

***conductivity = round(((float)digital\_data/1023\*10));***

***itoa(conductivity,storage,10);***

***send(storage[0]);***

***send(storage[1]);***

***send(storage[2]);***

***send('%');***

***send (' ');***

***send('\n');***

***/////////////////////***

***// print humidity***

***Htemp = (8 << I\_RH) | (D\_RH); // update Htemp***

***print\_humidity();***

***itoa(I\_RH,storage,10);***

***// send integral value***

***send(storage[0]);***

***send(storage[1]);***

***// send decimal value***

***itoa(D\_RH,storage,10);***

***send('.');***

***send(storage[0]);***

***send(storage[1]);***

***send('%');***

***send('\n');***

***/////////////////////***

***// print temperature***

***Ttemp = (8 << I\_Temp) | (D\_Temp); // update Ttemp***

***print\_temp();***

***// send integral value***

***itoa(I\_Temp,storage,10);***

***send(storage[0]);***

***send(storage[1]);***

***// send decimal value***

***itoa(D\_Temp,storage,10);***

***send('.');***

***send(storage[0]);***

***send(storage[1]);***

***send('°');***

***send('C');***

***send('\n');***

***/////////////////////***

***send('\n');***

***}***

***}***

***\_delay\_ms(1200); //Minimum 1 sec for sampling***

***}***

***}***

***////////////////////////////////////////////////////////////////////////***

***// print display functions***

***void print\_gas()***

***{***

***send('G');***

***send('a');***

***send('s');***

***send(' ');***

***send('D');***

***send('e');***

***send('t');***

***send('e');***

***send('c');***

***send('t');***

***send(':');***

***send(' ');***

***}***

***void print\_temp()***

***{***

***send('T');***

***send('e');***

***send('m');***

***send('p');***

***send('e');***

***send('r');***

***send('a');***

***send('t');***

***send('u');***

***send('r');***

***send('e');***

***send(':');***

***send(' ');***

***}***

***void print\_humidity()***

***{***

***send('H');***

***send('u');***

***send('m');***

***send('i');***

***send('d');***

***send('i');***

***send('t');***

***send('y');***

***send(':');***

***send(' ');***

***}***

***void print\_warning()***

***{***

***send('W');***

***send('A');***

***send('R');***

***send('N');***

***send('I');***

***send('N');***

***send('G');***

***send(':');***

***send(' ');***

***}***

***void print\_poor()***

***{***

***send('A');***

***send('I');***

***send('R');***

***send(' ');***

***send('Q');***

***send('U');***

***send('A');***

***send('L');***

***send('I');***

***send('T');***

***send('Y');***

***send(' ');***

***send('P');***

***send('O');***

***send('O');***

***send('R');***

***send('\n');***

***}***

***void print\_moderate()***

***{***

***send('M');***

***send('o');***

***send('d');***

***send('e');***

***send('r');***

***send('a');***

***send('t');***

***send('e');***

***send(' ');***

***}***

***void print\_good()***

***{***

***send('G');***

***send('o');***

***send('o');***

***send('d');***

***send(' ');***

***}***

***void print\_load()***

***{***

***send('L');***

***send('o');***

***send('a');***

***send('d');***

***send('i');***

***send('n');***

***send('g');***

***send('.');***

***send('.');***

***send('.');***

***}***

***void print\_conductivity()***

***{***

***send('C');***

***send('o');***

***send('n');***

***send('d');***

***send('u');***

***send('c');***

***send('t');***

***send('i');***

***send('v');***

***send('i');***

***send('t');***

***send('y');***

***send(':');***

***send(' ');***

***}***

***////////////////////////////////////////////////////////////////////////***