American International University- Bangladesh (AIUB)
Faculty of Engineering (EEE)

Course Name :	MICROPROCESSOR AND EMBEDDED SYSTEMS	**Course Code :**	4103
Semester :	Summer 2023	**Sec :**	F
Lab Instructor :	Md Sajid Hossain		

Experiment No :	07
Experiment Name :	Part-1: Implementation of a weather forecast system using Arduino

Submitted by (NAME):	**Rifat Hossain**	**Student ID:**	**20-42461-1**

Group Members	**ID**	**Name**
1.	20-42461-1	Rifat Hossain
2.	20-42831-1	Ishrat Jahan
3.	21-45019-2	Ahnaf Abdullah Zayad
4.	21-45038-2	Srabone Raxit
5.	21-45206-2	Kazi Ramisa Samiha
6.	21-45263-2	Shakibul Hasan

Performance Date :	**25.07.2023**	**Due Date :**	**31.07.2023**

Marking Rubrics (to be filled by Lab Instructor)

Category	Proficient [6]	Good [4]	Acceptable [2]	Unacceptable [1]	Secured Marks
Theoretical Background, Methods & procedures sections	All information, measures and variables are provided and explained.	All Information provided that is sufficient, but more explanation is needed.	Most information correct, but some information may be missing or inaccurate.	Much information missing and/or inaccurate.	
Results	All of the criteria are met; results are described clearly and accurately;	Most criteria are met, but there may be some lack of clarity and/or incorrect information.	Experimental results don't match exactly with the theoretical values and/or analysis is unclear.	Experimental results are missing or incorrect;	
Discussion	Demonstrates thorough and sophisticated understanding. Conclusions drawn are appropriate for analyses;	Hypotheses are clearly stated, but some concluding statements not supported by data or data not well integrated.	Some hypotheses missing or misstated; conclusions not supported by data.	Conclusions don't match hypotheses, not supported by data; no integration of data from different sources.	
General formatting	Title page, placement of figures and figure captions, and other formatting issues all correct.	Minor errors in formatting.	Major errors and/or missing information.	Not proper style in text.	

Writing & organization	Writing is strong and easy to understand; ideas are fully elaborated and connected; effective transitions between sentences; no typographic, spelling, or grammatical errors.	Writing is clear and easy to understand; ideas are connected; effective transitions between sentences; minor typographic, spelling, or grammatical errors.	Most of the required criteria are met, but some lack of clarity, typographic, spelling, or grammatical errors are present.	Very unclear, many errors.	
Comments:				Total Marks (Out of):	

Implementation of a weather forecast system using Arduino

I. Abstract: This lab manual describes the implementation of a weather forecast system using Arduino and microcontroller-based environmental sensors to measure temperature, pressure, and humidity. The primary objective is to familiarize participants with the weather prediction capabilities of the BMP180 or MPL115A absolute pressure sensor. The sensor is employed to deduce weather patterns based on barometric pressure changes. The manual explores the theory and methodology behind weather prediction using pressure sensors and explains how pressure fluctuations are directly correlated with changes in weather conditions. It also discusses the challenges of predicting weather in specific regions and emphasizes the importance of normalizing barometric pressure data for accurate analysis. The manual presents simple algorithms for weather prediction based on the pressure sensor readings, allowing users to determine weather trends and conditions.

II. Introduction: The implementation of a weather forecast system using Arduino and microcontroller-based environmental sensors offers an excellent opportunity to explore weather prediction. Weather patterns are influenced by various factors, with atmospheric pressure being a critical parameter. The BMP180 or MPL115A absolute pressure sensor is well-suited for measuring barometric pressure accurately and efficiently. Weather prediction using pressure sensors involves monitoring changes in barometric pressure over time to infer weather patterns. Low pressure often precedes deteriorating weather conditions, while high pressure can signal improving or clear weather. This experiment aims to demonstrate how the BMP180 or MPL115A sensor can predict weather conditions based on barometric pressure variations.

It is crucial to understand the impact of altitude on barometric pressure measurements. Local weather stations often normalise pressure data to reflect sea-level altitude (101.3 kPa). Without normalisation, altitude can significantly affect pressure readings, leading to inaccurate weather predictions. Additionally, certain regions, such as mountainous areas or places with frequent condensation and fog, may pose challenges for weather prediction. The manual introduces two approaches for weather prediction using pressure sensor readings. The first approach involves observing pressure trends over time, where a gradual increase may indicate sunny weather, stable pressure suggests a mix of sun and clouds, and a decrease in pressure could imply rainy conditions. The second approach is more direct, relying on knowing the current altitude and using an equation to calculate the expected pressure for sunny weather at that altitude. By

comparing the actual pressure readings with the calculated ideal pressure, the weather condition can be deduced.

To facilitate practical implementation, a simple C code from the DEMOAPEXSENSOR demo kit is provided. This code calculates the appropriate weather symbol to display on an LCD screen based on the pressure sensor readings and user-input altitude.

Overall, this experiment offers valuable insights into weather forecasting using Arduino and microcontroller-based sensors, helping participants better comprehend the interplay between barometric pressure and weather patterns.

III. Theory and Methodology:

Theory:

Weather Prediction using Barometric Pressure: Weather forecasting involves predicting atmospheric conditions over a specific period in a particular location. Barometric pressure, also known as atmospheric pressure, plays a crucial role in determining weather patterns. The BMP180 or MPL115A absolute pressure sensors are well-suited devices for this purpose, as they can accurately measure barometric pressure.

Barometric pressure is the force exerted by the Earth's atmosphere on a unit area. It varies with altitude and weather conditions. When analysing weather patterns, it is essential to consider the trends in barometric pressure over time. Slow changes in pressure often correlate with shifts in weather, while rapid fluctuations are associated with more immediate weather changes.

The relationship between barometric pressure and weather conditions can be understood by examining molecular weights. Air primarily consists of oxygen (O2) and nitrogen (N2) gases. Oxygen has a molecular mass of 32, nitrogen has a molecular mass of 28, and water vapour (H2O) has a molecular mass of 18. Air with a higher water vapour content is lighter than dry air, leading to lower barometric pressure. Consequently, an increase in water vapour, such as during the formation of clouds, leads to falling barometric pressure, indicating the potential for bad weather. On the other hand, clearing water vapour from the atmosphere results in higher pressure, suggesting improved weather conditions.

Methodology:

1. Sensor Placement and Calibration: To implement the weather forecast system, position the BMP180 or MPL115A sensor in a static location protected from strong airflow and other environmental disturbances. Calibration of the sensor is essential for accurate readings. The BMP180 sensor's temperature compensation capabilities enable it to handle varying temperatures over a wide operating range (0 to 85°C) without requiring auto-zeroing.

2. Normalisation of Barometric Pressure: To compare barometric pressure data from different locations, perform normalisation to standardise pressure values to sea level altitude (101.3 kPa). Normalisation helps meteorologists map weather patterns over a

region without the influence of altitude on pressure readings. For instance, normalise the reported pressure from an airport at a specific elevation to reflect the pressure at sea level.

3. Algorithms for Weather Prediction: Two approaches are used for weather prediction based on the pressure sensor readings:

   a. Trend-based Approach: Observe the pressure trend over time. An increasing pressure trend indicates sunny weather, while stable pressure suggests a mix of sun and clouds. A decreasing pressure trend can be indicative of rainy conditions. Examine the pressure trend over a 12-hour time frame to predict the weather.

   b. Direct Approach: Rely on knowing the current altitude and using an equation to calculate the expected pressure for sunny weather at that altitude. Compare the actual pressure readings with the calculated ideal pressure to deduce the weather condition. The difference between the actual pressure and the ideal pressure determines the weather symbol to display (sun, cloud, or rain).

4. Implementation using Arduino: Use Arduino and microcontroller programming to read data from the BMP180 or MPL115A sensor. Apply the normalisation process to compare the pressure readings accurately. Use the provided C code from the DEMOAPEXSENSOR demo kit as a reference to calculate the appropriate weather symbol for display on an LCD screen.

By following this theory and methodology, participants can gain a comprehensive understanding of weather prediction using Arduino and microcontroller-based pressure sensors, enhancing their knowledge of weather forecasting principles and technology.

IV. Apparatus:

1. Arduino Board
2. BMP180 or MPL115A Absolute Pressure Sensor.
3. Temperature Sensor
4. Breadboard and jumper wires.
5. LCD Screen: A display to visualise the weather symbols or pressure readings.
6. Computer: To program the Arduino board and analyse the sensor data.

7. USB Cable

V. Precautions:

1. It was ensured that the Arduino board was properly connected to the computers.
2. The connections on the board that corresponded with the code implementation were thoroughly verified.
3. Before making any connections or modifications to the circuit, it is essential to ensure that the Arduino board and its components are not connected to any power source. This precaution helps prevent accidental short circuits and damage to components.
4. Establish a connection between the ground (GND) pin of the Arduino board and the ground of the external power supply or the circuit's ground reference.
5. Exercise caution and handle the components gently, avoiding excessive force or rough handling, especially when inserting or removing jumper wires. Mishandling can lead to component damage or loose connections, affecting the reliability of the experiment.
6. Prior to uploading the code to the Arduino board, thoroughly inspect it for errors and mistakes. Incorrect code can result in unexpected outcomes or behaviours. If possible, seek guidance from a mentor or colleague to verify the accuracy of the code.

VI. Experimental Procedure:

1. Gather all the required components listed in the Apparatus section.

2. Set up the Arduino board and connect it to the computer using a USB cable. Ensure that the Arduino IDE software is installed on the computer.

3. Open the Arduino IDE and create a new sketch for the weather forecast system.

4. Connect the BMP180 or MPL115A absolute pressure sensor to the Arduino board:

a. Connect the sensor's VCC pin to the Arduino's 5V pin or the appropriate power supply pin.
b. Connect the sensor's GND pin to the Arduino's GND pin for the common ground reference.
c. Connect the sensor's SDA pin to the Arduino's SDA pin (A4 for Arduino Uno) for I2C communication.
d. Connect the sensor's SCL pin to the Arduino's SCL pin (A5 for Arduino Uno) for I2C communication.

5. If using a humidity sensor, connect it to the Arduino board following the sensor's specifications.

6. Optionally, connect an LCD screen to the Arduino board to display weather symbols or pressure readings.

7. Double-check all the wiring connections to ensure they are secure and correctly made according to the circuit diagram and pin assignments.

8. Upload the provided Arduino sketch to the board:

a. Verify that the code does not contain any errors or typos.
b. Select the correct Arduino board model and port from the Tools menu in the Arduino IDE.
c. Click on the "Upload" button to compile and upload the code to the Arduino board.

9. Once the code is successfully uploaded, the weather forecast system is ready for operation.

10. Place the weather forecast system in a suitable location with minimal airflow and environmental disturbances.

11. Allow the system to collect data for at least 2–3 hours to analyse a complete weather pattern.

12. Observe and record the pressure readings displayed on the LCD screen or through the serial monitor in the Arduino IDE.

13. Compare the pressure readings over time to identify trends in barometric pressure and potential weather pattern changes.

14. Additionally, if using a humidity sensor, monitor the humidity readings for any correlation with pressure changes.

15. Document your observations and interpretations of the weather patterns based on the pressure readings and humidity data (if applicable).

16. If desired, repeat the experiment in different locations to observe variations in weather patterns.

17. After completing the experiment, power off the Arduino board and disconnect it from the power source.

18. Document the entire experimental procedure, observations, results, and conclusions in the lab report.

Code for Weather System:

cloud print**********

```
#include <Wire.h>
#include <SPI.h>
#include <Adafruit_BMP085.h>
#include <Adafruit_Sensor.h>
//for Pin defined for SPI Connections
```

```
#include "U8glib.h"

Adafruit_BMP085 bmp;

U8GLIB_SSD1306_128X64 u8g(U8G_I2C_OPT_NONE|U8G_I2C_OPT_DEV_0); // I2C / TWI
float pressure;
float temp;
const uint8_t storm[] PROGMEM= {
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFE, 0x00, 0x00, 0x00, 0x00, 0x01, 0xFF,
0x80, 0x00, 0x00, 0x00, 0x07, 0x01, 0xC0, 0x00, 0x00, 0x00, 0x0E, 0x00, 0x60, 0x00, 0x00, 0x00,
0x0C, 0x00, 0x60, 0x00, 0x00, 0x00, 0x18, 0x00, 0x30, 0x00, 0x00, 0x00, 0x18, 0x00, 0x18, 0x00,
0x00, 0x00, 0x10, 0x00, 0x18, 0x00, 0x00, 0x03, 0xF0, 0x00, 0x1E, 0x00, 0x00, 0x07, 0xE0, 0x00,
0x0F, 0x00, 0x00, 0x0C, 0x00, 0x04, 0x01, 0x80, 0x00, 0x18, 0x00, 0x0C, 0x00, 0xC0, 0x00, 0x18,
0x00, 0x1C, 0x00, 0x40, 0x00, 0x10, 0x00, 0x1C, 0x00, 0x60, 0x00, 0x30, 0x00, 0x3C, 0x00, 0x60,
0x00, 0x30, 0x00, 0x6C, 0x00, 0x60, 0x00, 0x30, 0x00, 0xCC, 0x00, 0x60, 0x00, 0x10, 0x00, 0xCC,
0x00, 0x60, 0x00, 0x10, 0x01, 0x88, 0x00, 0x40, 0x00, 0x18, 0x03, 0x08, 0x00, 0xC0, 0x00, 0x0C,
0x06, 0x08, 0x01, 0x80, 0x00, 0x07, 0x06, 0x1F, 0xC3, 0x80, 0x00, 0x03, 0xFC, 0x0F, 0xFE, 0x00,
0x00, 0x00, 0x18, 0x01, 0x80, 0x00, 0x00, 0x00, 0x1F, 0x83, 0x00, 0x00, 0x00, 0x00, 0x01, 0x86,
0x00, 0x00, 0x00, 0x00, 0x01, 0x86, 0x00, 0x00, 0x00, 0x00, 0xC1, 0x8C, 0x30, 0x00, 0x00, 0x03,
0xC1, 0x18, 0x78, 0x00, 0x00, 0x07, 0xC1, 0x30, 0xF8, 0x00, 0x00, 0x0C, 0xC3, 0x31, 0x98, 0x00,
0x00, 0x08, 0x83, 0x63, 0x18, 0x00, 0x00, 0x18, 0x83, 0xC3, 0x10, 0x00, 0x00, 0x19, 0x83, 0x83,
0x30, 0x00, 0x00, 0x0F, 0x83, 0x01, 0xF0, 0x00, 0x00, 0x07, 0x07, 0x00, 0xE0, 0x00, 0x00, 0x00,
0x06, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
```

```
};
const uint8_t presure[] PROGMEM = {
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x0F, 0xF0, 0x00, 0x00, 0x00, 0x00, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x03, 0xFF, 0xFF, 0xC0,
0x00,
0x00, 0x0F, 0xFF, 0xFF, 0xF0, 0x00, 0x00, 0x3F, 0xC0, 0x03, 0xFC, 0x00, 0x00, 0x7E, 0x03,
0xC0,
0x7E, 0x00, 0x00, 0xF8, 0x03, 0xC0, 0x1F, 0x00, 0x01, 0xF0, 0x03, 0xC0, 0x0F, 0x80, 0x03,
0xE6,
0x03, 0xC0, 0x67, 0xC0, 0x07, 0xCE, 0x03, 0xC0, 0x73, 0xE0, 0x0F, 0x8F, 0x03, 0xC0, 0xF1,
0xF0,
0x0F, 0x07, 0x83, 0xC1, 0xE0, 0xF0, 0x1E, 0x07, 0xC0, 0x03, 0xE0, 0x78, 0x1C, 0x03, 0xC0,
0x03,
0xC0, 0x38, 0x3C, 0x01, 0x80, 0x01, 0x80, 0x3C, 0x38, 0x00, 0x00, 0x00, 0x00, 0x1C, 0x78,
0x00,
0x00, 0x00, 0x00, 0x1E, 0x7B, 0x80, 0x01, 0x80, 0x01, 0xDE, 0x73, 0xF0, 0x01, 0x80, 0x0F,
0xCE,
0x73, 0xF8, 0x01, 0x80, 0x1F, 0xCE, 0xF1, 0xF8, 0x01, 0x80, 0x1F, 0x8F, 0xF0, 0x30, 0x01,
0x80,
0x0C, 0x0F, 0xE0, 0x00, 0x01, 0x80, 0x00, 0x07, 0xE0, 0x00, 0x03, 0xC0, 0x00, 0x07, 0xE0,
0x00,
0x03, 0xC0, 0x00, 0x07, 0xE0, 0x00, 0x03, 0xC0, 0x00, 0x07, 0xF0, 0x00, 0x03, 0xC0, 0x00,
0x0F,
0xF0, 0xF0, 0x03, 0xC0, 0x0F, 0x0F, 0xF3, 0xF8, 0x07, 0xE0, 0x1F, 0xCF, 0x73, 0xF8, 0x0F,
0xF0,
0x1F, 0xCE, 0x73, 0xC0, 0x1F, 0xF8, 0x03, 0xCE, 0x78, 0x00, 0x1F, 0xF8, 0x00, 0x1E, 0x38,
0x00,
0x1F, 0xF8, 0x00, 0x1C, 0x38, 0x00, 0x1F, 0xF8, 0x00, 0x1C, 0x00, 0x00, 0x1F, 0xF8, 0x00,
0x00,
0x00, 0x00, 0x0F, 0xF0, 0x00, 0x00, 0x00, 0x00, 0x07, 0xE0, 0x00, 0x00, 0x00, 0x00, 0x03,
0xC0,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00
};
const uint8_t rain[] PROGMEM = {
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x07, 0xF0, 0x00, 0x00, 0x00, 0x00, 0x1C, 0x18, 0x00,
0x00,
```

```
 0x00, 0x00, 0x30, 0x04, 0x00, 0x00, 0x00, 0x00, 0x60, 0x02, 0x00, 0x00, 0x00, 0x00, 0x40,
0x03,
 0x00, 0x00, 0x00, 0x00, 0x80, 0x01, 0x00, 0x00, 0x00, 0x07, 0x80, 0x01, 0x80, 0x00, 0x00,
0x1E,
 0x00, 0x00, 0xE0, 0x00, 0x00, 0x30, 0x00, 0x00, 0x18, 0x00, 0x00, 0x60, 0x00, 0x00, 0x08,
0x00,
 0x00, 0x40, 0x00, 0x00, 0x0C, 0x00, 0x00, 0xC0, 0x00, 0x00, 0x04, 0x00, 0x00, 0xC0, 0x00,
0x00,
 0x04, 0x00, 0x00, 0xC0, 0x00, 0x00, 0x04, 0x00, 0x00, 0x40, 0x00, 0x00, 0x04, 0x00, 0x00,
0x40,
 0x00, 0x00, 0x0C, 0x00, 0x00, 0x60, 0x00, 0x00, 0x18, 0x00, 0x00, 0x38, 0x00, 0x00, 0x30,
0x00,
 0x00, 0x0F, 0xFF, 0xFF, 0xE0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x06, 0x03,
0x01,
 0x80, 0x00, 0x00, 0x0F, 0x07, 0x03, 0xC0, 0x00, 0x00, 0x1B, 0x09, 0x04, 0xC0, 0x00, 0x00,
0x32,
 0x19, 0x0C, 0x80, 0x00, 0x00, 0x32, 0x19, 0x0C, 0x80, 0x00, 0x00, 0x1E, 0x0F, 0x07, 0x00,
0x00,
 0x00, 0x00, 0x00, 0x20, 0x00, 0x00, 0x00, 0x00, 0xE0, 0x70, 0x00, 0x00, 0x00, 0x00, 0xA0,
0xD0,
 0x00, 0x00, 0x00, 0x03, 0xA1, 0x10, 0x00, 0x00, 0x00, 0x02, 0x21, 0x30, 0x00, 0x00, 0x00,
0x03,
 0x61, 0xB0, 0x00, 0x00, 0x00, 0x01, 0xC0, 0xE0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00};
 const uint8_t temperature[] PROGMEM = {
 0x00, 0x00, 0x1F, 0x80, 0x00, 0x00, 0x00, 0x00, 0x7F, 0xC0, 0x00, 0x00, 0x00, 0x00, 0xF9,
0xE0,
 0x00, 0x00, 0x00, 0x01, 0xE0, 0x70, 0x00, 0x00, 0x00, 0x01, 0xC0, 0x70, 0x00, 0x00, 0x00,
0x03,
 0x80, 0x39, 0xFC, 0x00, 0x00, 0x03, 0x80, 0x39, 0xFC, 0x00, 0x00, 0x03, 0x80, 0x38, 0x00,
0x00,
 0x00, 0x03, 0x80, 0x38, 0x00, 0x00, 0x00, 0x03, 0x80, 0x38, 0x00, 0x00, 0x00, 0x03, 0x80,
0x39,
 0xFC, 0x00, 0x00, 0x03, 0x80, 0x39, 0xFC, 0x00, 0x00, 0x03, 0x8E, 0x38, 0x00, 0x00, 0x00,
0x03,
 0x9F, 0x38, 0x00, 0x00, 0x00, 0x03, 0x9F, 0x39, 0xFC, 0x00, 0x00, 0x03, 0x9F, 0x39, 0xFC,
0x00,
 0x00, 0x03, 0x9F, 0x39, 0xFC, 0x00, 0x00, 0x03, 0x9F, 0x38, 0x00, 0x00, 0x00, 0x03, 0x9F,
0x38,
```

```
 0x00, 0x00, 0x00, 0x03, 0x9F, 0x39, 0xFC, 0x00, 0x00, 0x03, 0x9F, 0x39, 0xFC, 0x00, 0x00,
0x03,
 0x9F, 0x39, 0xFC, 0x00, 0x00, 0x03, 0x9F, 0x38, 0x00, 0x00, 0x00, 0x03, 0x9F, 0x38, 0x00,
0x00,
 0x00, 0x03, 0x9F, 0x39, 0xFC, 0x00, 0x00, 0x03, 0x9F, 0x39, 0xFC, 0x00, 0x00, 0x03, 0x9F,
0x38,
 0x00, 0x00, 0x00, 0x07, 0x9F, 0x3C, 0x00, 0x00, 0x00, 0x0F, 0x1F, 0x1E, 0x00, 0x00, 0x00,
0x1E,
 0x3F, 0x87, 0x00, 0x00, 0x00, 0x1C, 0x7F, 0xC7, 0x00, 0x00, 0x00, 0x38, 0xFF, 0xE3, 0x80,
0x00,
 0x00, 0x39, 0xFF, 0xF3, 0x80, 0x00, 0x00, 0x31, 0xFF, 0xF1, 0x80, 0x00, 0x00, 0x73, 0xFF,
0xF9,
 0xC0, 0x00, 0x00, 0x73, 0xFF, 0xF9, 0xC0, 0x00, 0x00, 0x73, 0xFF, 0xF9, 0xC0, 0x00, 0x00,
0x73,
 0xFF, 0xF9, 0xC0, 0x00, 0x00, 0x31, 0xFF, 0xF1, 0x80, 0x00, 0x00, 0x39, 0xFF, 0xF1, 0x80,
0x00,
 0x00, 0x38, 0xFF, 0xE3, 0x80, 0x00, 0x00, 0x1C, 0x7F, 0xC7, 0x00, 0x00, 0x00, 0x1E, 0x3F,
0x87,
 0x00, 0x00, 0x00, 0x0F, 0x00, 0x0E, 0x00, 0x00, 0x00, 0x07, 0x80, 0x3C, 0x00, 0x00, 0x00,
0x03,
 0xF1, 0xF8, 0x00, 0x00, 0x00, 0x00, 0xFF, 0xE0, 0x00, 0x00, 0x00, 0x00, 0x3F, 0x80, 0x00,
0x00
 };
 const uint8_t sunny[] PROGMEM = {
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
 0x00, 0x00, 0x00, 0x00, 0x03, 0x00, 0x00, 0x00, 0x00, 0x00, 0x83, 0x04, 0x00, 0x00, 0x00,
0x00,
 0x83, 0x04, 0x00, 0x00, 0x00, 0x00, 0xC3, 0x0C, 0x00, 0x00, 0x00, 0x00, 0x40, 0x08, 0x00,
0x00,
 0x00, 0x00, 0x1F, 0xE0, 0x00, 0x00, 0x00, 0x10, 0x30, 0x38, 0x20, 0x00, 0x00, 0x18, 0xC0,
0x0C,
 0x60, 0x00, 0x00, 0x0C, 0x80, 0x04, 0xC0, 0x00, 0x00, 0x01, 0x80, 0x02, 0x00, 0x00, 0x00,
0x01,
 0x00, 0x03, 0x00, 0x00, 0x00, 0x03, 0x00, 0x01, 0x00, 0x00, 0x00, 0x02, 0x00, 0x01, 0x00,
0x00,
 0x00, 0x72, 0x00, 0x01, 0x38, 0x00, 0x00, 0x72, 0x00, 0x01, 0x38, 0x00, 0x00, 0x02, 0x00,
0x01,
 0x00, 0x00, 0x00, 0x03, 0x00, 0x01, 0x00, 0x00, 0x00, 0x01, 0x00, 0x03, 0x00, 0x00, 0x00,
0x01,
```

```
 0x00, 0x02, 0x00, 0x00, 0x00, 0x04, 0x80, 0x06, 0x80, 0x00, 0x00, 0x1C, 0xC0, 0x0C, 0xE0,
0x00,
 0x00, 0x10, 0x70, 0x38, 0x20, 0x00, 0x00, 0x00, 0x1F, 0xE0, 0x00, 0x00, 0x00, 0x00, 0x40,
0x08,
 0x00, 0x00, 0x00, 0x00, 0x43, 0x0C, 0x00, 0x00, 0x00, 0x00, 0x83, 0x04, 0x00, 0x00, 0x00,
0x00,
 0x83, 0x04, 0x00, 0x00, 0x00, 0x00, 0x03, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00
 };
 const uint8_t cloudy[] PROGMEM= {
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x08, 0x00, 0x00, 0x00, 0x00, 0x02, 0x08, 0x40,
0x00,
 0x00, 0x00, 0x03, 0x08, 0x40, 0x00, 0x00, 0x00, 0x01, 0x00, 0xC0, 0x00, 0x00, 0x00, 0x00,
0x3C,
 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x31, 0x81, 0x8C, 0x00, 0x00,
0x00,
 0x1B, 0x00, 0xD8, 0x00, 0x00, 0x00, 0x02, 0x00, 0x60, 0x00, 0x00, 0x00, 0x06, 0x00, 0x20,
0x00,
 0x00, 0x00, 0x3F, 0x80, 0x20, 0x00, 0x00, 0x00, 0xE0, 0xC0, 0x10, 0x00, 0x00, 0x00, 0x80,
0x60,
 0x17, 0x00, 0x00, 0x01, 0x80, 0x30, 0x10, 0x00, 0x00, 0x03, 0x00, 0x10, 0x10, 0x00, 0x00,
0x02,
 0x00, 0x10, 0x20, 0x00, 0x00, 0x1E, 0x00, 0x1C, 0x20, 0x00, 0x00, 0x7C, 0x00, 0x0E, 0x48,
0x00,
 0x00, 0xC0, 0x00, 0x01, 0xCC, 0x00, 0x00, 0x80, 0x00, 0x01, 0x84, 0x00, 0x01, 0x80, 0x00,
0x00,
 0x80, 0x00, 0x01, 0x00, 0x00, 0x00, 0xC0, 0x00, 0x01, 0x00, 0x00, 0x00, 0xC0, 0x00, 0x01,
0x00,
 0x00, 0x00, 0xC0, 0x00, 0x01, 0x00, 0x00, 0x00, 0x80, 0x00, 0x01, 0x80, 0x00, 0x00, 0x80,
0x00,
 0x00, 0xC0, 0x00, 0x01, 0x80, 0x00, 0x00, 0x60, 0x00, 0x03, 0x00, 0x00, 0x00, 0x3F, 0xFF,
0xFE,
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
```

```
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00
 };
 void draw(void) {
 // graphic commands to redraw the complete screen should be placed here
 u8g.setFont(u8g_font_8x13);
 u8g.setPrintPos(5,10);
 u8g.print("Weather Station");
 u8g.setPrintPos(55,30);
 u8g.print("By");
 u8g.setPrintPos(2,50);
 u8g.print("The IoT Projects");
 }
 void draw_4(void){
 pressure=bmp.readPressure()/100;
 delay(1);
 u8g.setPrintPos(32,10);
 u8g.print("Pressure");
 u8g.setPrintPos(65,45);
 u8g.print(pressure);
 u8g.setPrintPos(70,60);
 u8g.print("hPa");
 u8g.drawBitmapP( 0, 15, 6, 48,presure);
 }
 void draw_2(void){
 u8g.setFont(u8g_font_8x13);
 u8g.setPrintPos(0,10);
 u8g.print(" Temperature");
 u8g.setPrintPos(55,40);
 u8g.print(temp);
 u8g.setPrintPos(97,40);
 u8g.print("C");
 u8g.drawBitmapP( 0, 17, 6,48,temperature);
 }
 void draw3(void){
 u8g.setPrintPos(3,10);
 u8g.print("Weather Forecast");
 if ((pressure<=1000.59)&&(pressure>998.5)){
 u8g.drawBitmapP( 10, 15, 6, 48,cloudy);
 u8g.setPrintPos(60,45);
 u8g.print("Cloudy");
 }
```

```
 else if((pressure<998.5)&&(pressure>996.5))
 {
 u8g.drawBitmapP( 10, 15, 6, 48,rain);
 u8g.setPrintPos(66,38);
 u8g.print("Rain");}
 else if((pressure>1000.6)&&(pressure<1050)) { u8g.drawBitmapP( 5, 15, 6, 48,sunny);
u8g.setPrintPos(50,45); u8g.print("Clear Sky");} else if((pressure>990)&&(pressure<=996.4))
 {
 u8g.drawBitmapP( 5, 15, 6, 48,storm);
 u8g.setPrintPos(60,45);
 u8g.print("Storm");}
 else{
 u8g.setPrintPos(40,40);
 u8g.print("Error!");
 }
 }
 void setup() {
 Serial.begin(9600);
 Serial.println(F("BMP280 test"));
 if (!bmp.begin()) {
 Serial.println(F("Could not find a valid BMP280 sensor, check wiring!"));
 while (1);
 }
 u8g.firstPage(); // first page
 do {
 draw();
 }
 while( u8g.nextPage() );
 delay(2000);
 }
 void loop() {
 temp=bmp.readTemperature();
 u8g.firstPage(); // Next page
 do {
 draw_2();
 }
 while( u8g.nextPage() );
 delay(2000);
 u8g.firstPage(); // Next page
 do {
 draw_4();
 }
 while( u8g.nextPage() );
 delay(2000);
 u8g.firstPage(); // Next page
 do {
```

```
 draw3();
 }
 while( u8g.nextPage() );
 delay(2000);
 }
**********Scrolling
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Adafruit_BMP085.h>
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT);
Adafruit_BMP085 bmp;
#define SEALEVELPRESSURE_HPA (101500)
float simpleweatherdifference, currentpressure, predictedweather, currentaltitude;
void setup() {
 // put your setup code here, to run once:
 display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
if (!bmp.begin()) {
 Serial.println("Could not find a valid BMP085 sensor, check wiring!");
 while (1) {}
 }
}
void loop() {
 // put your main code here, to run repeatedly:
 display.clearDisplay();
 display.setTextSize(1);
 display.setTextColor(SSD1306_WHITE);

 display.setCursor(0,5);
 display.print("BMP180");
 display.setCursor(0,19);
 display.print("T=");
 display.print(bmp.readTemperature(),1);
 display.println("*C");
 /*prints BMP180 pressure in Hectopascal Pressure Unit*/
 display.setCursor(0,30);
 display.print("P=");
 display.print(bmp.readPressure()/100.0F,1);
 display.println("hPa");

 /*prints BMP180 altitude in meters*/
 display.setCursor(0,40);
 display.print("A=");
```

```
 display.print(bmp.readAltitude(SEALEVELPRESSURE_HPA),1);
 display.println("m");
 delay(6000);
 display.display();
currentpressure=bmp.readPressure()/100.0;
predictedweather=(101.3*exp(((float)(currentaltitude))/(-7900)));
simpleweatherdifference=currentpressure-predictedweather;
//display.clearDisplay();
display.setCursor(0,50);
if (simpleweatherdifference>0.25)
 display.print("SUNNY");
 if (simpleweatherdifference<=0.25 || simpleweatherdifference>=-0.25)
 display.print("CLOUDY");

 if (simpleweatherdifference<-0.25)
 display.print("RAINY");
display.display();
delay(2000);
display.startscrollright(0,7);
delay(6000)  //increasing delay will scroll for longer time
display.stopscroll();
}
```

VII. Simulation and Measurement:

Simulation was done after implementation of the hardware part to match with the result.

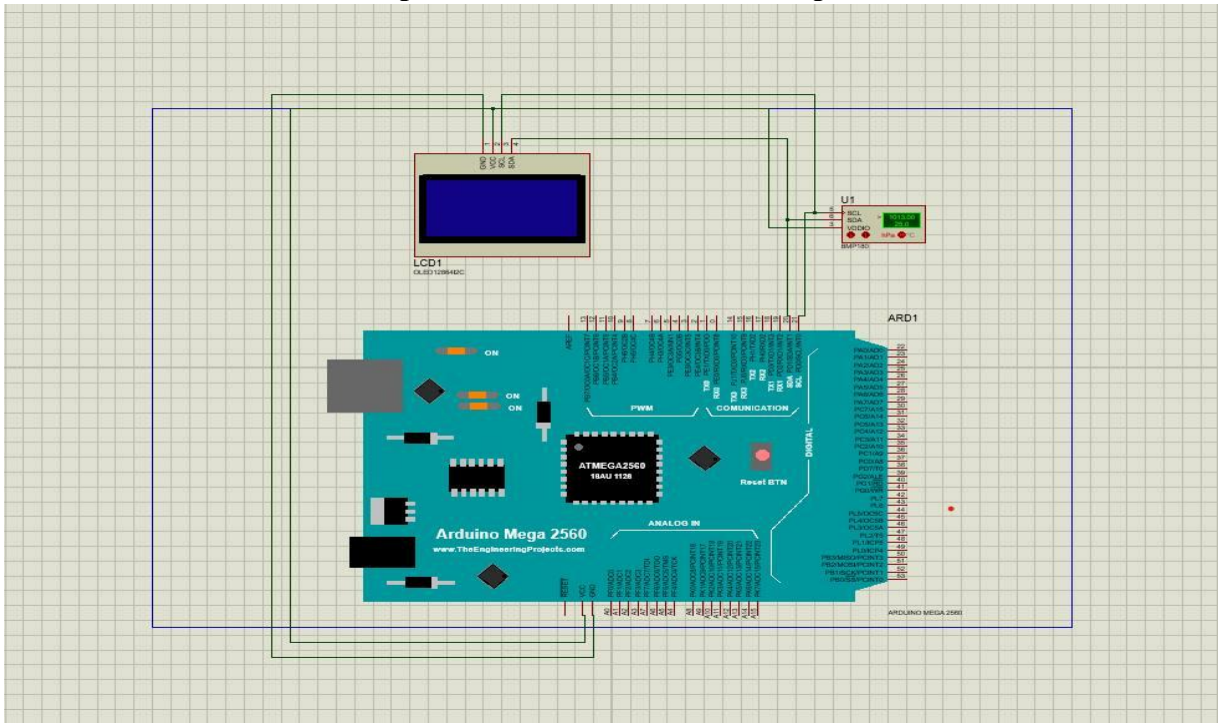

Figure 7.1: Whole Weather System using protious.

VIII. Results:

The Algorithm was uploaded to the Arduino.

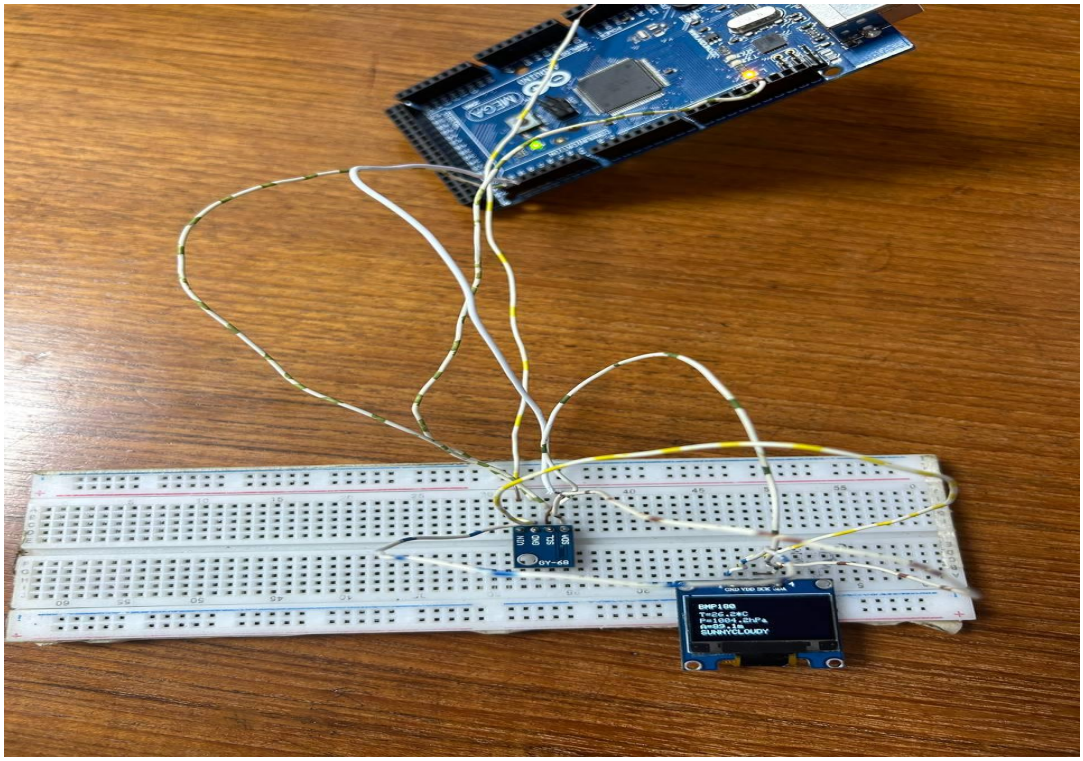

Figure 8.1: Display text showing.

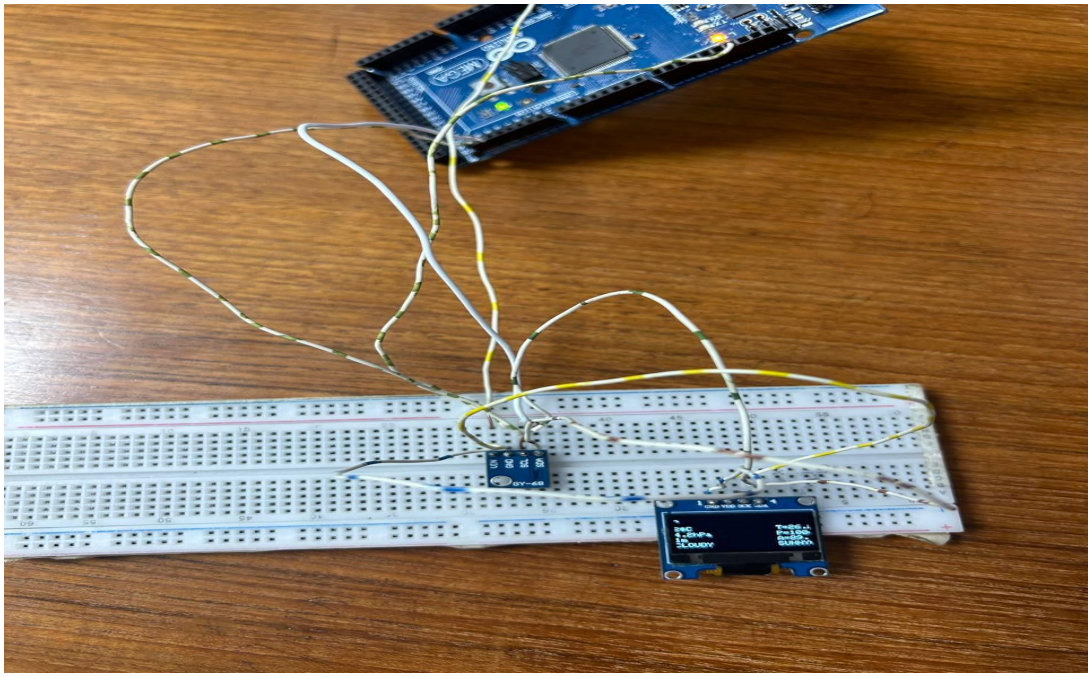

Figure 8.2: Display Text rotating

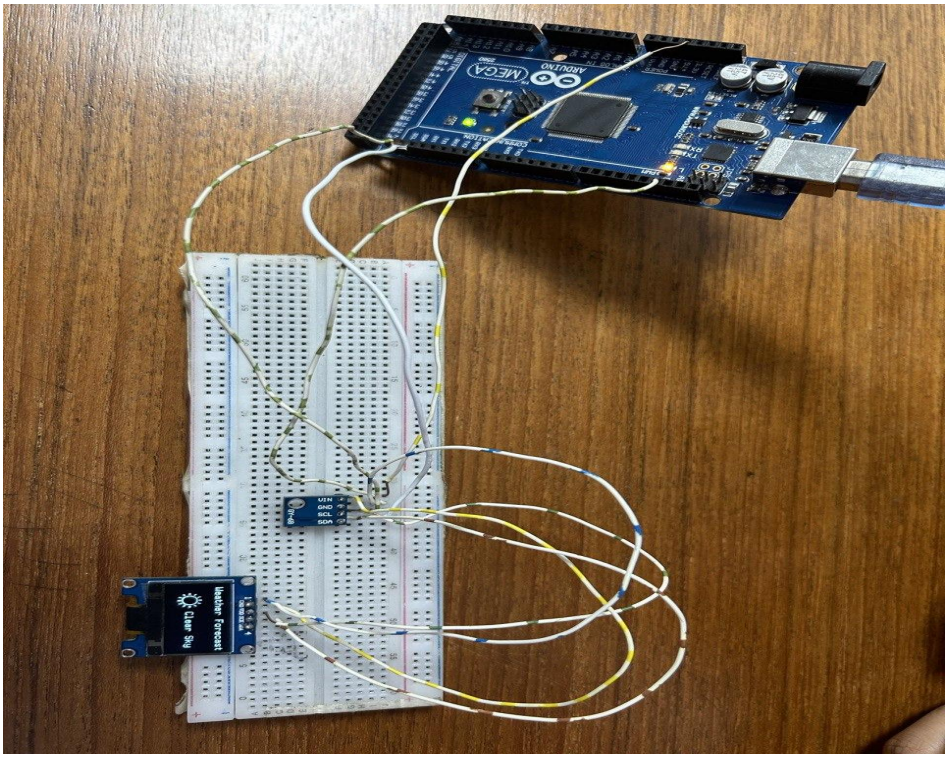

Figure 8.3: Weather Signs Showing on Display .

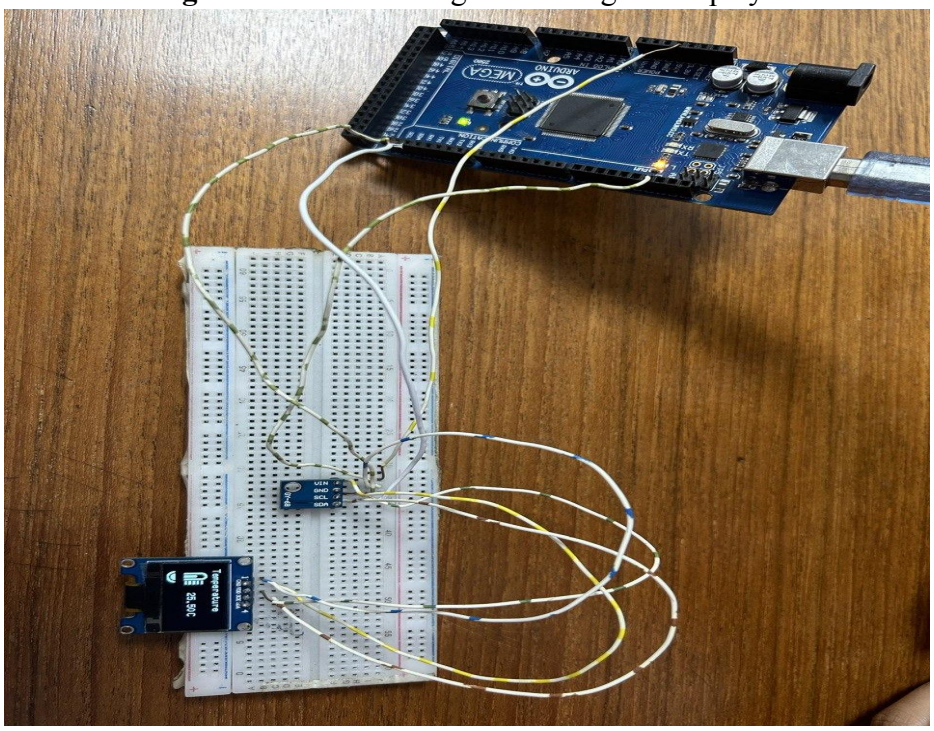

Figure 8.4: Weather Signs Showing on Display .

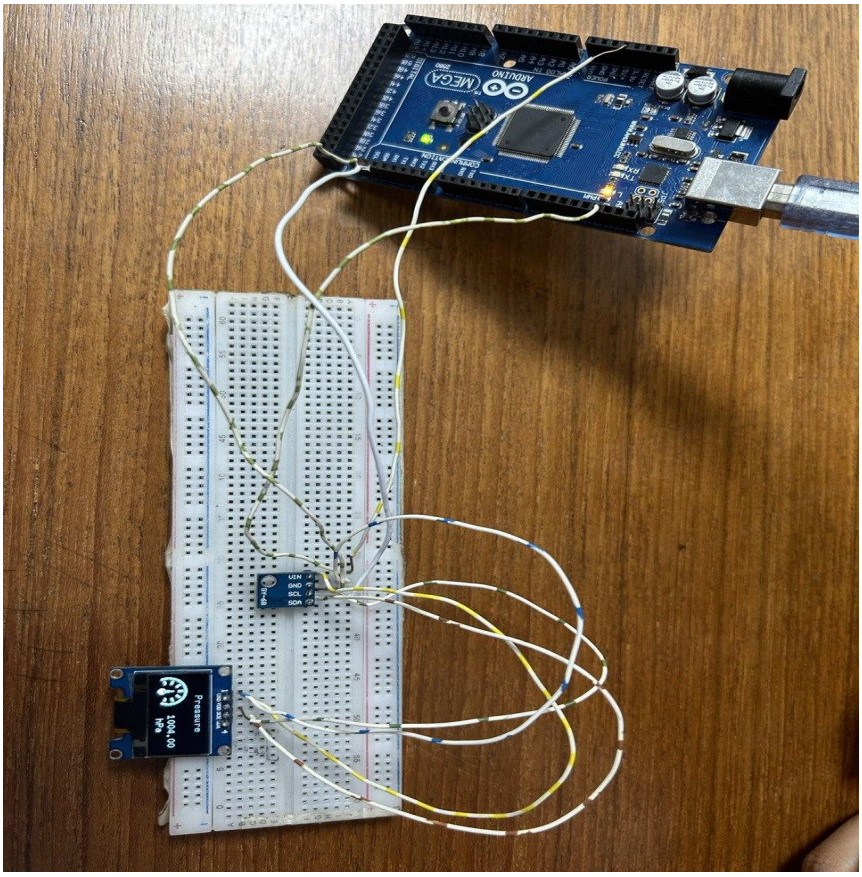

Figure 8.5: Weather Signs Showing on Display .

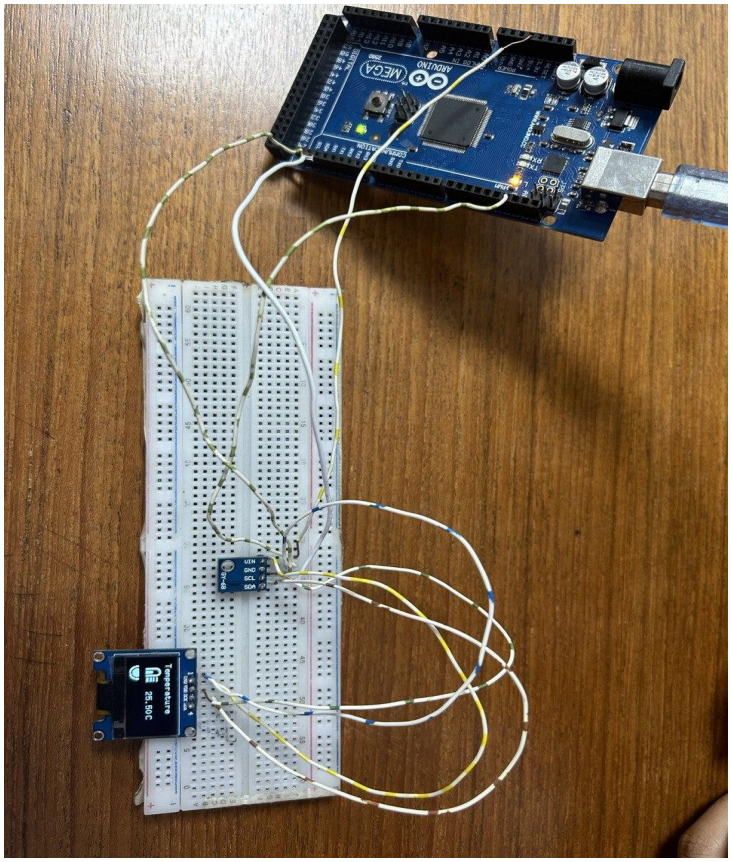

Figure 8.3: Display Text rotating

IX. Discussion:

Using Arduino and microcontroller-based sensors, the implementation of the weather forecast system provided valuable insights into weather prediction based on barometric pressure and humidity data (if applicable). The purpose of the experiment was to investigate the relationship between pressure changes and weather patterns, as well as the predictability of the system.

During the experiment, we successfully programmed the Arduino board and connected the BMP180 or MPL115A absolute pressure sensor in order to accurately measure barometric pressure. We also incorporated a humidity sensor, if applicable, to collect additional environmental data. The sensor connections were double-checked for precision and consistency.

The Arduino design for the weather forecast system was uploaded error-free, allowing the system to operate as intended. The code implemented weather prediction algorithms, including a trend-based approach and a direct approach based on pressure calculations adjusted for altitude. If utilised, the LCD screen displayed weather symbols or pressure measurements, making data interpretation simple.

Throughout the phase of data capture, we observed fluctuations in barometric pressure and relative humidity measurements. By monitoring pressure trends, we were able to identify potential shifts in weather patterns, such as rising pressure signifying sunny weather and falling pressure indicating rainy conditions. If applicable, the combination of barometric pressure and humidity data enhanced the comprehension of weather changes.

The experiment highlighted the significance of normalising barometric pressure data for accurate analysis and comparison between locations. Normalisation allowed us to eliminate altitude-related effects and provide a standard pressure value for accurate mapping of weather patterns.

We discovered that the weather forecast system was effective at predicting weather patterns based on variations in atmospheric pressure. However, the accuracy of the forecasts may be affected by factors such as the positioning of the sensor and local environmental conditions. For example, cities located near mountainous regions or regions prone to condensation and fog may exhibit more complex weather patterns that necessitate a network of sensors for accurate forecasts.

X. Conclusion:

The weather forecast system using Arduino and microcontroller-based sensors proved to be a valuable tool for weather prediction based on barometric pressure and humidity data (if applicable). The experiment provided hands-on experience with data collection, analysis, and the interpretation of weather patterns. By understanding the relationship between pressure changes and weather conditions, we gained insights into the practical applications of such systems for forecasting local weather phenomena. Further improvements and adjustments, along with a broader network of sensors, could enhance the system's accuracy and make it more effective for predicting weather in diverse geographical locations. Overall, this experiment highlighted the significance of technology in weather forecasting and the potential for innovative solutions to tackle real-world challenges.

XI. Reference (s):

[1] ATmega328 Arduino Uno Board Working and Its Applications-Elprocus. Available at: https://www.elprocus.com/atmega328-arduino-uno-board-working-and-itsapplications/?fbclid=IwAR31Al9NcljIcVp0TRvwr7XA7Pu6Vq9yaaJXyTs7kpuPHvqNjOXkS2dFUAk [Accessed: Jul. 3, 2023].

[2] J. Riyaz, "How to Make Simple Weather Station Using Arduino," Instructables. [Online]. Available: https://www.instructables.com/How-to-Make-Simple-Weather-Station-Using-Arduino/. [Accessed: 31-Jul-2023].