AMERICAN INTERNATIONAL UNIVERSITY- BANGLADESH
Faculty of Engineering
Laboratory Report Cover Sheet
Students must complete all details except the faculty use part.

Please submit all reports to your subject supervisor or the office of the concerned faculty.

Experiment Title: Implementation of a weather forecast system using the ADC modules of an Arduino.

Experiment Number: **06**	Due Date: **02/05/2024**	Semester: **Spring 23-24**
Subject Code: **COE310**	Subject Name: **Microprocessor and Embedded Systems**	Section : **E**
Course Instructor: **Protik Parvez Sheikh**		

Declaration and Statement of Authorship:

1. I/we hold a copy of this report, which can be produced if the original is lost/ damaged.
2. This report is my/our original work and no part of it has been copied from any other student's work or fromany other source except where due acknowledgement is made.
3. No part of this report has been written for me/us by any other person except where such collaboration hasbeen authorized by the lecturer/teacher concerned and is clearly acknowledged in the report.
4. I/we have not previously submitted or currently submitting this work for any other course/unit.
5. This work may be reproduced, communicated, compared and archived for the purpose of detecting plagiarism.
6. I/we give permission for a copy of my/our marked work to be retained by the School for review andcomparison, including review by external examiners.

I/we understand that

7. Plagiarism is the presentation of the work, idea or creation of another person as though it is your own. It is a form of cheating and is a very serious academic offence that may lead to expulsion from the University. Plagiarized material can be drawn from, and presented in, written, graphic and visual form, includingelectronic data, and oral presentations. Plagiarism occurs when the origin of the material used is not appropriately cited.
8. Enabling plagiarism is the act of assisting or allowing another person to plagiarize or to copy your work

Group Number (if applicable): 1 ☐ Individual Submission ☑ Group Submission

No.	Student Name	Student Id	Program	Signature
Submitted by:				
1	SHAYAN ABRAR	22-47156-1	CSE	
Group Members:				
2	SAMIA SHARMIN DOLA	22-47126-1	CSE	
3	MD SAMIN YEASAR	22-47139-1	CSE	
4	RIFAH SANZIDA	22-47154-1	CSE	
5	MD. JOBAER HOSSAIN	22-47116-1	CSE	

For faculty use only: **Total Marks:** **Marks Obtained:**

Faculty comments

Theory and Methodology:

Weather Prediction

The BMP180 or MPL115A is an absolute device that can be used to predict and measure barometric pressure to deduce weather patterns. Weather prediction requires a static location for the sensor and 2-3 hours to analyze a full weather pattern. Typically, the pressure changes due to weather are slow, requiring a few hours to determine the sloping of the pressure change. Vertical movement or a significant airflow can interfere with results due to only weather patterns in barometric pressure. The sensor should be kept in a relatively protected area from any strong air flows and kept at that static location during analysis. Temperature effects can change the results of a normal pressure sensor especially if the measurement is done over several hours at varying temperatures. Due to the nature of the calibration and temperature compensation, BMP180 meets these requirements, compensating for temperature swings over a large 0 to 85°C operating range. It will not require auto-zeroing for shifts in offset or span-over temperature.

How Pressure Increases and Decreases with Weather

For weather pattern prediction, the BMP180 or MPL115A is a well-suited device with its pressure range and resolution. Barometric pressure changes can directly correlate to changes in the weather. Low pressure is typically seen as the precursor to worsening weather. High-pressure increases can be interpreted as improving or clearing weather. The typical reasoning can be seen in a comparison of molecular weights. If air is approximately 21% $O_2(g)$, 78% $N_2(g)$, $O_2(g)$ has a molecular mass of 32, and $N_2(g)$ has a mass of 28. Water vapor, $H_2O(g)$ has a molecular mass of 18. So, if there is a large amount of water vapor present in the air, this air is going to be lighter than just regular dry air by itself. It is an interesting fact that explains how weather patterns lead to high or low pressure.

If bad weather originates in an area in the formation of water-vapor clouds, this is falling pressure on a barometer. The vapor will reduce the barometric pressure as the H_2O reduces the mass above that point on the earth. High pressure will signal the clearing of the water vapor as the air dries.

Another quandary is how weather during severe hurricanes/cyclones with high 150 mph winds be defined as low pressure because hurricanes are low-pressure conditions surrounded by higher pressure. The rush of air from higher to lower-pressure regions creates fast-moving winds. The lower the pressure in the center, the greater the differential pressure between high and low areas. This leads to a stronger cyclone or hurricane.

In some areas, it is harder to predict weather patterns. An example is cities located at the base of mountainous regions where condensation and fog are a daily occurrence. An area like Hawaii where high colder mountains meet low warm sea regions can have harder to predict results. A network of sensors can give a more exact trend, but for a single sensor in a static location, there are a few ways to have a simple standalone weather station.

Local Weather Stations

When implementing a weather station, it is best to will check results with a local forecast. When researching local weather pressures, such as barometric pressure at the closest airport, remember that the weather is normalized for altitude. Normalization takes local barometric pressure and shifts it to reflect sea level altitude. **Sea Level is 101.3 kPa**, and by normalizing various points on a map, a meteorologist can see the weather pattern over a region. Without normalization, the effect of altitude on the pressure reported by collection points will lead to useless data. A mountain data point will have pressure affected by altitude and as it leads to the valleys, the pressure point there will be higher, telling nothing about the weather without the normalization.

Airports are typical reporting stations to check barometric pressure. Some display only normalized pressure during a web search. This is such that a pilot landing at any airport can deduce the weather conditions by knowing the barometric pressure. If the airport is located at the beach, or in a mountainous region, normalization of this value removes the barometric variation due to the altitude. It standardizes pressure so that weather patterns can be mapped.

Example:
An airport located at 600 m elevation would have a pressure of 93.97 kPa according to our pressure to altitude equation. If the weather was sunny and mostly clear, it would most probably have a published pressure of 101.3 kPa for weather conditions. It may not be extremely clear skies as this would be a high-pressure weather system. It would be a stable pressure with neither extreme low nor high pressure.

Remember to discern this information when trying to see if the MPL115A value matches the local weather barometric pressure. Sometimes a disparity in the value occurs due to normalization.

Algorithms for weather Simple Approach
How is weather predicted using the barometric sensor? There is a simple approach looking at increasing or decreasing pressure. Simply an increase over time is a trend that approaches “sunny” or “clear” days. Dropping pressure can signal a worsening “cloudy” or “rainy” day. This can be seen typically as a rising or falling bar on many simple solution weather stations. It can be interpreted as an increase/decrease gradient for the user to interpret, but the time interval is not used extensively to reach weather predictions. The user can look at the results for a 12-hour time frame to predict the weather trend.

This table is typically used:

Analysis	**Output**
dP > +0.25 kPa	Sun Symbol
-0.25 kPa < dP < 0.25 kPa	Sun/Cloud Symbol
dP < -0.25 kPa	Rain Symbol

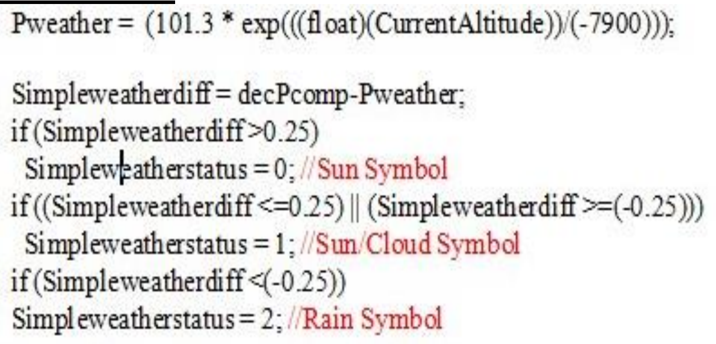
Another approach that is more direct and quicker in calculating the weather in the simple approach is to know the current altitude. This cuts the need to wait and see a “trend”.
By using the equation below:

$$ph = p0 \cdot e^{\frac{-h}{7990m}}$$

Where p0 = 101.3 kPa, and h is the current altitude, the pressure for the local barometric can be calculated. This is the pressure for good sunny weather at the current altitude location.
By using the pressure equation and knowing the normalized good weather pressure for the current location (best for a static weather station), the weather can be deduced by the difference. As in the table for the weather symbols, the ideal pressure is compared to the value from the MPL115A and the appropriate symbol of Sun/Cloud/Rain is selected.
Below simple C code from the DEMOAPEXSENSOR demo kit that calculates which weather symbol to display on the LCD screen.

- CurrentAltitude - (m) Altitude in meters that is entered into the system by the user for that current static location.
- Pweather - (kPa) Pressure at the current altitude. It is calculated using the Height (m) to Pressure (kPa) exponential equation, inputting CurrentAltitude in meters. This is the ideal pressure for the current location on a stable relatively sunny day.
- decPcomp - (kPa) Value of compensated pressure from BMP180/MPL115A.

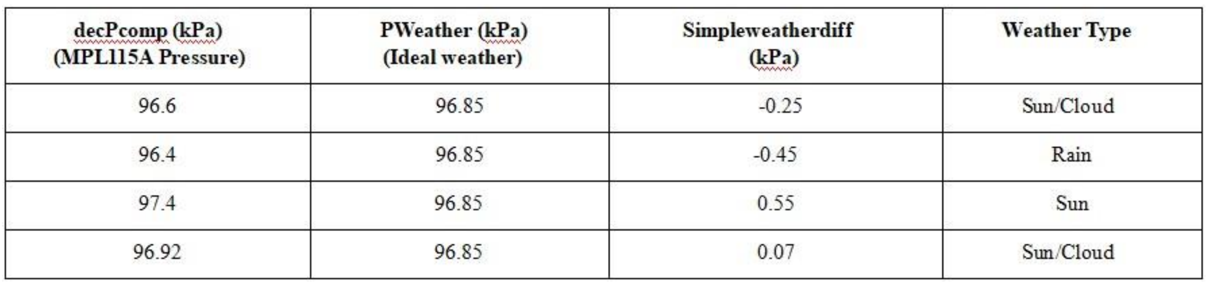
Simple Weather Station Code



```
Pweather = (101.3 * exp(((float)(CurrentAltitude))/(-7900)));

Simpleweatherdiff = decPcomp-Pweather;
if (Simpleweatherdiff>0.25)
  Simpleweatherstatus = 0; //Sun Symbol
if ((Simpleweatherdiff <=0.25) || (Simpleweatherdiff >=(-0.25)))
  Simpleweatherstatus = 1; //Sun/Cloud Symbol
if (Simpleweatherdiff <(-0.25))
Simpleweatherstatus = 2; //Rain Symbol
```

Let us look at some data:



decPcomp (kPa) (MPL115A Pressure)	PWeather (kPa) (Ideal weather)	Simpleweatherdiff (kPa)	Weather Type
96.6	96.85	-0.25	Sun/Cloud
96.4	96.85	-0.45	Rain
97.4	96.85	0.55	Sun
96.92	96.85	0.07	Sun/Cloud

For example, Altitude for Tempe, AZ = 359 m, thus PWeather = 101.3 x $e^{-359/7990m}$ = 96.85 kPa.
Observing Pressure over time will yield similar results over 12 hours. In this case, the changes in the pressure that takes place over an extended time will be enough for the simple method to figure out the weather patterns. This negates the need for user input of the approximate location/altitude. The weather algorithm will be more accurate at the end of the 12-hour interval as the trend is visible versus at initialization.

Knowing the altitude can also be useful for a dynamically changing location to predict simple weather. Take for example a GPS unit: a GPS unit can give an approximate altitude measurement. Measuring the difference from the MPL115A pressure sensor and calculated pressure from the GPS altitude gives a close approximation of weather patterns quickly at that dynamically changing point. Weather approximation can be deduced in the symbol style as above.

Advanced Version of Weather Station

A more complex approach is to measure the P/t and see how the gradient is changing over time. As in the simple approach, this does need to be kept in a static location during measurement. Essentially as time progresses, the weather can be broken into more exact categories than the simple approach of basic symbols.
This can also use less time than waiting for a full 12 hours to see the pattern of pressure change. In Table 3, the ranges of pressure change over time leading to the definition of the weather patterns shown. It is a change in the pressure per hour. 2-3 hours are needed to deduce how the pressure is migrating.

Table 3. Advanced Weather Determination

Analysis	Output
dP/dt > 0.25 kPa/h	Quickly rising High-Pressure System, not stable
0.05 kPa/h < dP/dt < 0.25 kPa/h	Slowly rising High-Pressure System, stable good
-0.05 kPa/h < dP/dt < 0.05 kPa/h	Stable weather condition
-0.25 kPa/h < dP/dt < -0.05 kPa/h	Slowly falling Low-Pressure System, stable rainy
dP/dt < -0.25 kPa/h	Quickly falling Low Pressure, Thunderstorm, not

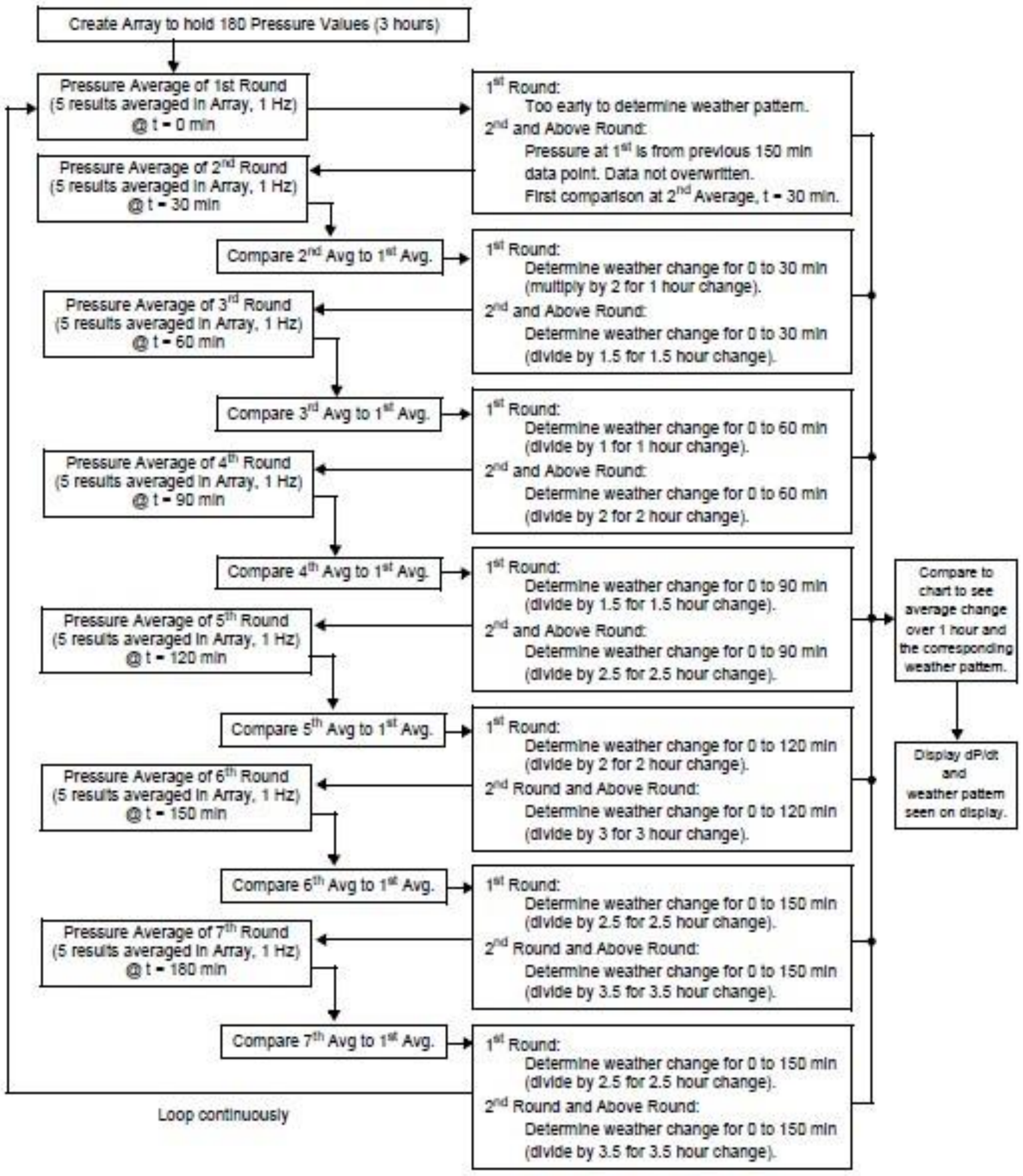

Create Array to hold 180 Pressure Values (3 hours)
Pressure Average of 1st Round (5 results averaged in Array, 1 Hz) @ t = 0 min
1st Round: Too early to determine weather pattern. 2nd and Above Round: Pressure at 1st is from previous 150 min data point. Data not overwritten. First comparison at 2nd Average, t = 30 min.
Pressure Average of 2nd Round (5 results averaged in Array, 1 Hz) @ t = 30 min
Compare 2nd Avg to 1st Avg.
1st Round: Determine weather change for 0 to 30 min (multiply by 2 for 1 hour change). 2nd and Above Round: Determine weather change for 0 to 30 min (divide by 1.5 for 1.5 hour change).
Pressure Average of 3rd Round (5 results averaged in Array, 1 Hz) @ t = 60 min
Compare 3rd Avg to 1st Avg.
1st Round: Determine weather change for 0 to 60 min (divide by 1 for 1 hour change). 2nd and Above Round: Determine weather change for 0 to 60 min (divide by 2 for 2 hour change).
Pressure Average of 4th Round (5 results averaged in Array, 1 Hz) @ t = 90 min
Compare 4th Avg to 1st Avg.
1st Round: Determine weather change for 0 to 90 min (divide by 1.5 for 1.5 hour change). 2nd and Above Round: Determine weather change for 0 to 90 min (divide by 2.5 for 2.5 hour change).
Pressure Average of 5th Round (5 results averaged in Array, 1 Hz) @ t = 120 min
Compare 5th Avg to 1st Avg.
1st Round: Determine weather change for 0 to 120 min (divide by 2 for 2 hour change). 2nd Round and Above Round: Determine weather change for 0 to 120 min (divide by 3 for 3 hour change).
Pressure Average of 6th Round (5 results averaged in Array, 1 Hz) @ t = 150 min
Compare 6th Avg to 1st Avg.
1st Round: Determine weather change for 0 to 150 min (divide by 2.5 for 2.5 hour change). 2nd Round and Above Round: Determine weather change for 0 to 150 min (divide by 3.5 for 3.5 hour change).
Pressure Average of 7th Round (5 results averaged in Array, 1 Hz) @ t = 180 min
Compare 7th Avg to 1st Avg.
1st Round: Determine weather change for 0 to 150 min (divide by 2.5 for 2.5 hour change). 2nd Round and Above Round: Determine weather change for 0 to 150 min (divide by 3.5 for 3.5 hour change).
Loop continuously
Compare to chart to see average change over 1 hour and the corresponding weather pattern.
Display dP/dt and weather pattern seen on display.

Figure 1. Advanced Weather Flowchart

In the provided flowchart, the pressure is sampled every minute for 3 hours/180 minutes into a data array. The first 5 minutes are averaged, followed by 5 minutes near the first ½ hour point. Consecutive ½ hour marks have 5-minute averaged data points stored. This leads to 7 averaged results over the 180 minutes depicting the pressure every ½ hour. Once the data- points are collected, the patterns are deduced. A flowchart provides the method used in deducing the weather pattern. The initial starting point is the reference from which every ½ hour data point is compared to. As the pressure falls, the value is compared and divided so that the change in pressure per 1 hour is compared every half an hour.

Apparatus:

1. Arduino Uno Board
2. BMP180 / MPL115A
3. inches96 inch OLED 128X64
4. Breadboard and Jump Wires

Program:

```
#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_PA (101500)
float simpleweatherdifference, currentpressure, predictedweather, currentaltitude;
void setup() {

  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 BME180 pressure in Hectopascal Pressure Unit*/
  display.setCursor(0,30);
  display.print("P=");
  display.print(bmp.readPressure()/100.0,1);
  display.println("hPa");

    /*prints BME180 altitude in meters*/
  display.setCursor(0,40);
  display.print("A=");
  display.print(bmp.readAltitude(SEALEVELPRESSURE_PA),1);
  display.println("m");
  delay(6000);
  display.display();

 currentpressure=bmp.readPressure()/100.0;
 currentaltitude=bmp.readAltitude(SEALEVELPRESSURE_PA);
 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)
    display.print("SUNNY/CLOUDY");

   if (simpleweatherdifference<-0.25)
    display.print("RAINY");
    display.display();
 delay(2000);
 }
```

Experimental Outcome:

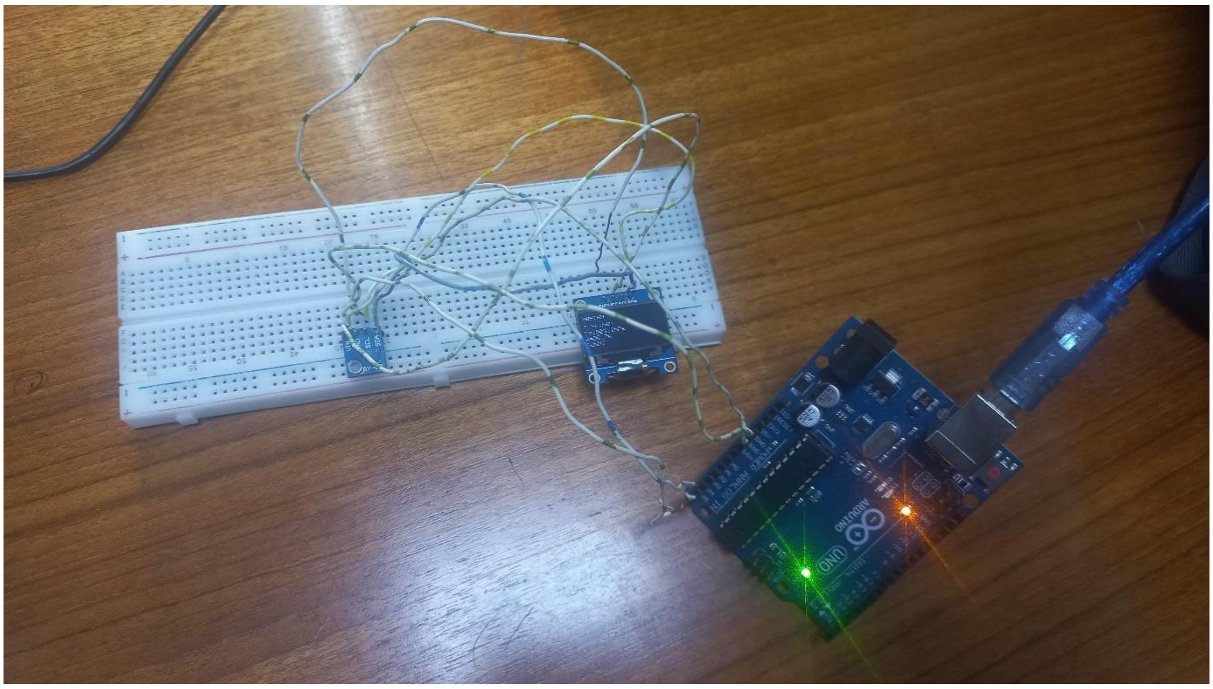


Fig-1: Arduino Uno with BMP180 and OLED

Simulation Outcome:

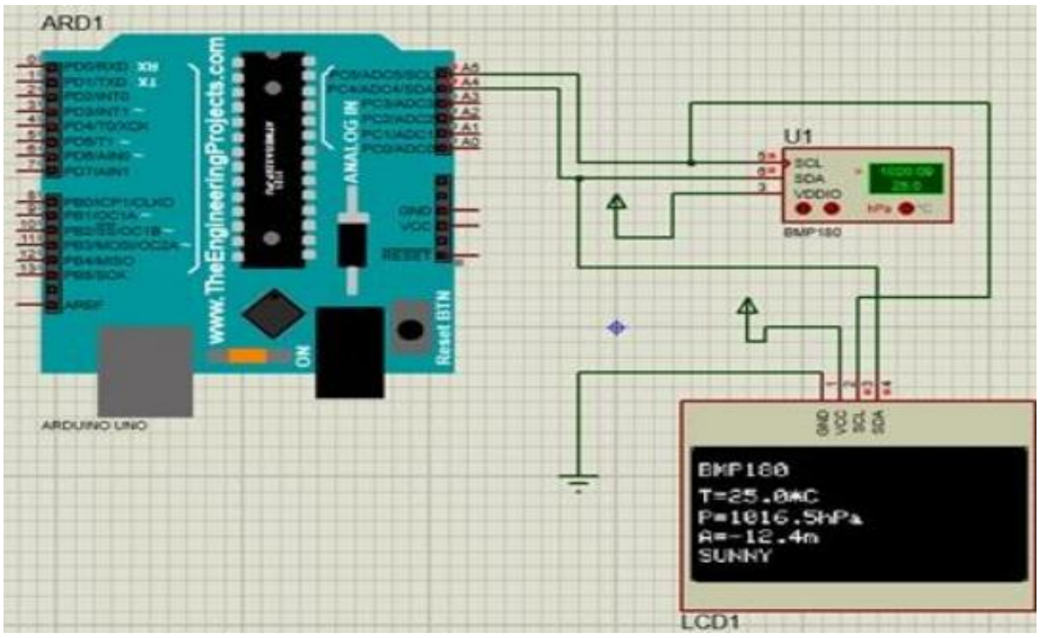


Fig-1: First Simulation where Temperature was 25.0°C, Pressure was 1016.5 hPA, Altitude was 12.4m and SUNNY condition

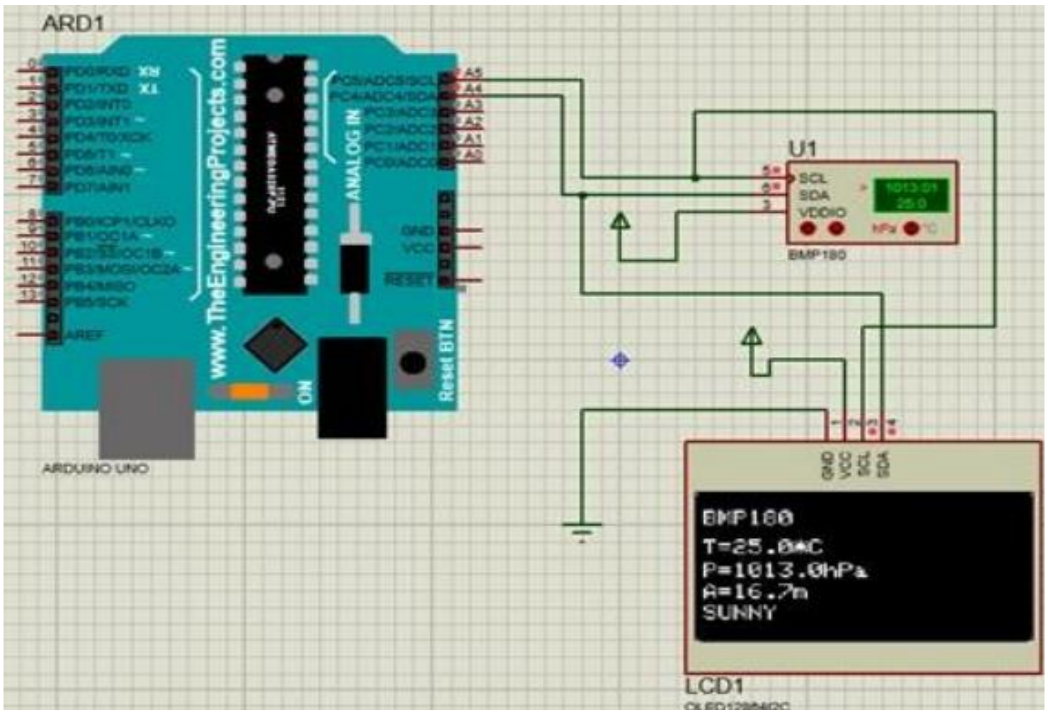
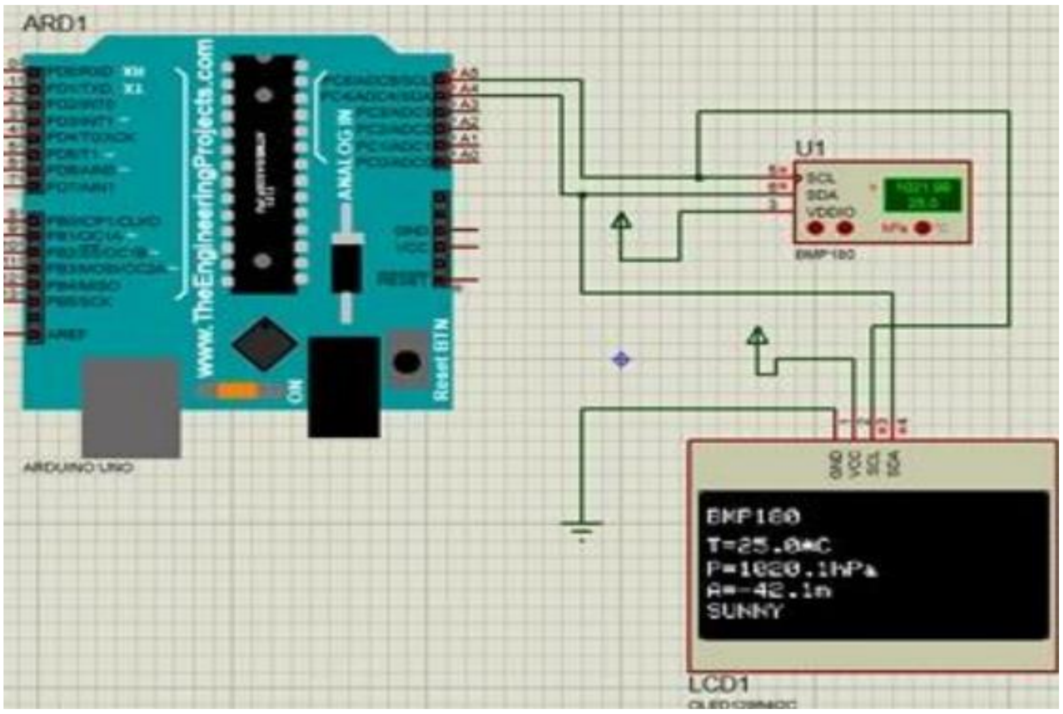


Fig-2: Second Simulation where Temperature was x°C, Pressure was xx hPA, Altitude was ym and SUNNY/CLOUDY condition



```
BMP180
T=25.0*C
P=1020.1hPa
A=-42.1m
SUNNY
```

Fig-3: Third Simulation where Temperature was x°C, Pressure was xx hPA, Altitude was ym and RAINY condition

Discussion:

The experiment focused on establishing communication between two Arduino boards using the Serial Peripheral Interface (SPI) protocol. SPI was chosen due to its full-duplex nature, synchronous communication, and ability to connect one master with multiple peripherals. The experiment aimed to provide a hands-on understanding of SPI communication by implementing a master-slave configuration with LEDs and push buttons connected to each Arduino board. Participants learned about the essential SPI lines—MISO, MOSI, SCK, and SS—and their roles in facilitating communication between master and slave devices. By exploring the SPI protocol within the Arduino environment, participants gained practical insights into hardware interfacing, SPI library usage, and programming techniques for SPI-based applications.

Conclusion:

In conclusion, the experiment successfully demonstrated the implementation of SPI communication between two Arduino boards for controlling LEDs and push buttons. By following the provided circuit diagrams and Arduino code, participants gained proficiency in configuring SPI communication, initializing SPI peripherals, and exchanging data between master and slave devices. The experiment highlighted the versatility and efficiency of SPI in facilitating bidirectional communication and real-time interaction between microcontrollers. Moving forward, the knowledge gained from this experiment can be applied to a wide range of projects, including sensor networks, data acquisition systems, and IoT applications, where reliable and synchronized communication between multiple devices is essential. Overall, the experiment served as a valuable learning experience in exploring the capabilities of SPI protocol within the Arduino ecosystem.

References:

[1]. Arduino. [Online]. Available: https://www.arduino.cc/.

[2]. Atmel Corporation, "ATmega328/P - 8-bit AVR Microcontrollers," Datasheet.

[3]. AVRFreaks, "TUT C Newbies Guide to AVR Timers." [Online]. Available: https://www.avrfreaks.net/forum/tut-c-newbies-guide-avr-timers.

[4]. MaxEmbedded, "AVR Timers - TIMER0," June 2011. [Online]. Available: http://maxembedded.com/2011/06/avr-timers-timer0/.

[5]. Autodesk, "Tinkercad | From mind to design in

minutes." [Online]. Available:

https://www.tinkercad.com/.