ISTANBUL TECHNICAL UNIVERSITY

FACULTY OF ELECTRIC AND ELECTRONICS

DEPARTMENT OF CONTROL & AUTOMATION ENGINEERING

MODELLING AND CONTROL OF BIOLOGICAL SYSTEMS

Final Project Report

Team#4

EMRE AY

040100675

Spring 2015

1. Purpose of the Project

It is desired to build an electronic flow meter so that the air flow rate of a human can be measured.

2. Conceptual Design

There are several methods and instruments in order to measure fluid flow rates where necessary. Since cost is a vital parameter, it is desired to keep the cost as low as possible. After several thoughts and research, it is decided to build a differential flow meter due to its simplicity and costs of needed equipment for such instrument.

The logic of differential flow meter is quite simple. Consider a tube with a varying diameter as in *Figure 1.*

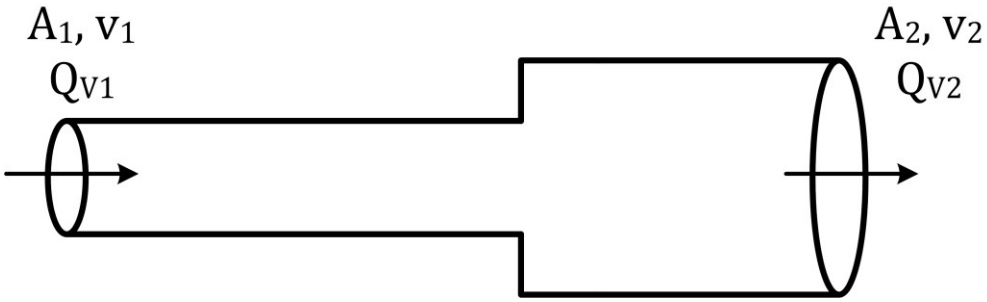

Figure 1 - A tube with varying diameter

Let A_1 and A_2 be the cross sectional areas of the tube, v_1 and v_2 are the speed of the fluid –in this case air–, and Q_{V1} and Q_{V2} are the volumetric flow rates. Under the assumption of an incompressible fluid, the volume that passes would be equal. Hence, along the tube the flow rate should be constant and Q_{V1} and Q_{V2} are equal.

$$Q_{V1} = A_1 \cdot v_1 \quad (1)$$

$$Q_{V2} = A_2 \cdot v_2 \quad (2)$$

So the equation can be found;

$$A_1 \cdot v_1 = A_2 \cdot v_2 \quad (3)$$

From Bernoulli's equations;

$$\frac{1}{2}\rho_1 v_1^2 + P_1 = \frac{1}{2}\rho_2 v_2^2 + P_2 \quad (4)$$

where P_1 and P_2 are the absolute pressure values at two sides. Taking squares of both sides in equation 4 and modifying yields;

$$2(P_1 - P_2) = \rho_2 v_2^2 - \rho_1 v_1^2 \quad (5)$$

Isolating v_2 and taking its square in equation 3,

$$v_2^2 = \left(\frac{A_1 \cdot v_1}{A_2}\right)^2 = \left(\frac{r_1 \cdot v_1}{r_2}\right)^2, r_1, r_2 \; are \; radii \quad (6)$$

Combining equations 5 and 6;

$$2(P_1 - P_2) = \rho_2 \left(\frac{r_1 \cdot v_1}{r_2}\right)^2 - \rho_1 v_1^2 \tag{7}$$

Solving the equation for v_1;

$$v_1 = r_2 \cdot \sqrt{\frac{2(P_1 - P_2)}{r_1^2 \rho_2 - r_2^2 \rho_1}} \tag{8}$$

By putting equation 8 into equation 1;

$$Q_V = Q_{V1} = A_1 \cdot r_2 \cdot \sqrt{\frac{2(P_1 - P_2)}{r_1^2 \rho_2 - r_2^2 \rho_1}} \tag{9}$$

From equation 9, it can be seen that volumetric flow rate can be calculated if the pressure difference, diameters and air density is known. The air densities can be calculated by;

$$\rho = \frac{p}{R_{specific} \cdot T} \tag{10}$$

where p is the absolute pressure in Pa, T is the absolute temperature in K and $R_{specific}$ is the specific gas constant of dry air 287.058 J/(kg K). It is assumed to have dry air.

Two tubes with diameters 14mm and 20mm are chosen. The necessary values are pressure and temperature at the tubes, so this will be the criteria of sensor selection.

3. Hardware Selection

It is needed to measure the absolute pressure values and temperatures in two tubes and calculate the flow rate from their difference. For this task, Bosch BMP180 sensor was a good fit due to its price, specifications, size and reachability. BMP180 is a compact sensor with 3.3V operating voltage. It can sense temperature and absolute air pressure.

BMP180 comes with a 13mm*10mm sized circuit board which would fit into the tubes. BMP180 communicates with I^2C protocol. Normally, many devices can communicate over the same I^2C bus but they should have different addresses. However, BMP180 sensors didn't produced with different address options; they all have the same address. In order to use two separate sensors with the same address on an I^2C bus, an analog multiplexer can be used such as 74HC4052.

74HC4052 is a dual 4-channel analog multiplexer/demultiplexer integrated circuit. It has two select pins for four configurations. In each combination, the IC switches its X and Y output with four X's and Y's available. Since only two sensors are going to be used only two of the 4 dual channels will be used.

For simplicity, an Arduino Mega will be used as Arduino has many core libraries and it is also easy to find external libraries as the one written for BMP180. BMP180 Arduino library has many functions that ease the settling of communication.

It is aimed to use a 16*2 LCD to display the flow rate value, but alternatively the results would be also plotted on MATLAB in real time over serial communication.

Lastly, in order to check and calibrate the system, a low cost simple mechanic respirometer as in *Figure 2* is bought.

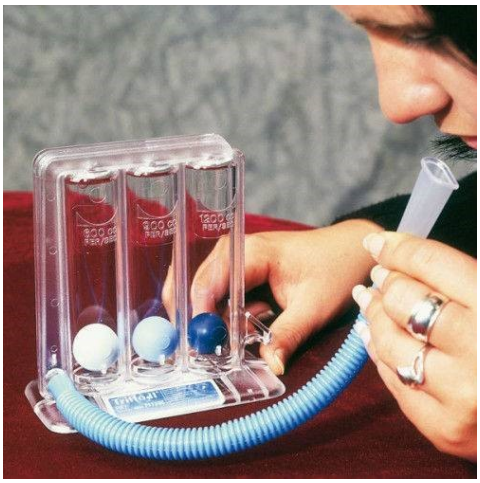

Figure 2 - Low cost respirometer for calibration

This respirometer can measure certain values; 600 cc/s, 900 cc/s and 1200 cc/s depending on the column of the balls. This system will be attached to the tubes so the measurements can be compared.

4. Assembling the System

Two tubes with diameters 14mm and 20mm are connected, two BMP180 sensors are put inside of them and the mechanical flow meter is connected at the tubes. Then the circuit in *Figure 3* is assembled.

Two sensors are powered with 3.3V, their SDA and SCL pins are connected to the analog multiplexer. Since only two combinations will be used, by looking at the truth table of the multiplexer, one of the select pins is grounded and the other one is connected to a GPIO pin of Arduino to do the selection. Output pins of multiplexer are connected to Arduino's SDA and SCL pins. An LCD display is connected with necessary setup for 4bit mode.

After that Arduino is programmed so that is would read the temperature and pressure values of two sensors by switching the channels then calculating the volumetric flow rate using equation 9 and displaying the results on LCD as well as sending them serially over USB so that it can be monitored on the computer. The Arduino program code can be seen in Appendix A.

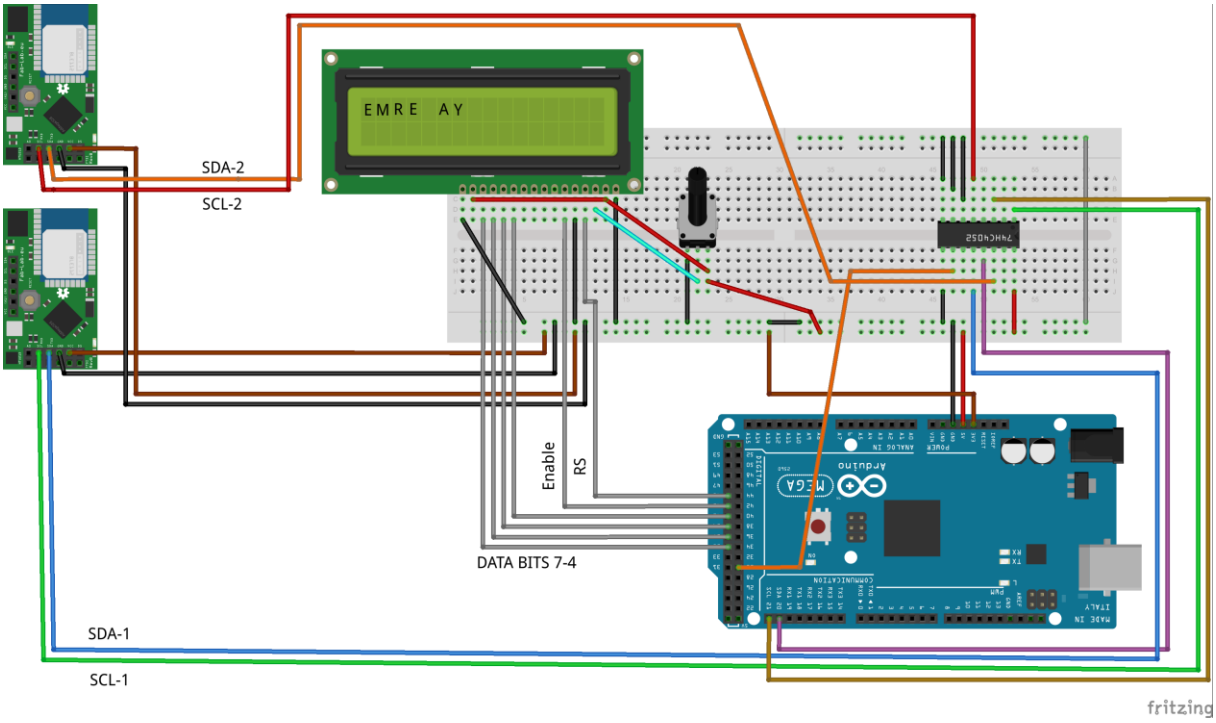

Figure 3 - Assembled circuit drawn in Fritzing

5. First Test of the System

When the calculations are made according to the equations 9 and 10, the output unfortunately wasn't successful. The first reason is calculating the air densities on the tubes. When equation 10 is used and the densities are calculated they appeared to be the same and thus, when put into equation 9, inside of the square root becomes negative since r_1^2 is always less than r_2^2. So when the pressure difference is positive, it calculates an imaginary number. The air densities appear to be the same due to the sensor specifications and incapability of the assembled system. So the direct physical model did not fit.

6. Alternative Linearized Approach with Calibration

Since the result was not successful, an alternative approach is proposed. As it can be seen from the equation 9, we can see that the flow rate is related with the pressure difference. Neglecting other dynamics, let us assume that there is a linear relationship between the flow rate and the pressure difference such that;

$$Q_V \cong K \cdot \Delta P \tag{11}$$

By using the mechanical flow meter and the pressure difference, we can found a gain. Since this is a linearization, 600 cc/s measurements from the mechanical flow meter is selected as the linearization point. In other words, the gain will be calculated for the difference pressure at 600 cc/s.

After measurements, it has simply fit;

$$Q_V = 2 \cdot \Delta P \tag{12}$$

Equation 12 gives rough but enough accuracy for such low cost system. It is important to take ΔP value in Pascal, and the volumetric flow rate will be in unit cm^3/s or cc/s. The code for this linearized approach can be found at Appendix A.

7. Tests and MATLAB Plotting

When the system with linearized calculation is tested, the results are acceptable. A MATLAB script is prepared so that the serial data is read and simultaneously plotted as in *Figure 4*. The script can be found at Appendix B.

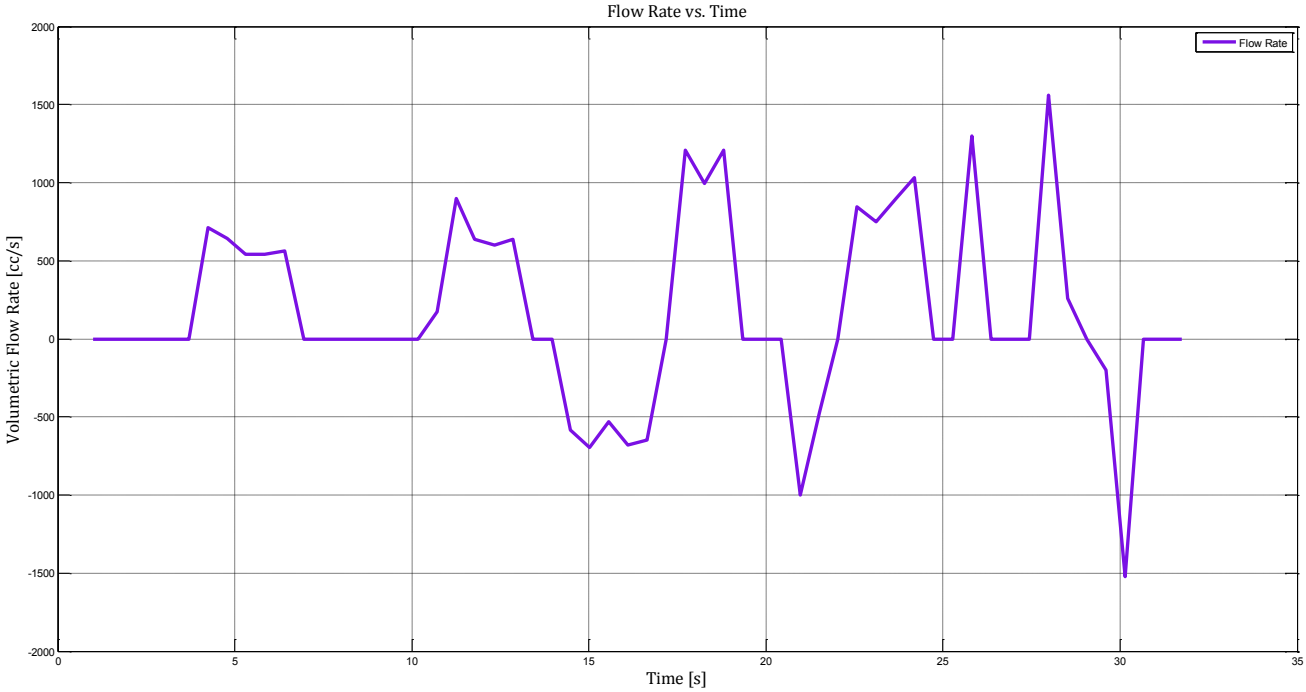

Figure 4 - Simultaneous Plot of Flow Rate

8. Conclusion

A low cost differential flow meter is designed and assembled successfully. The results are displayed on LCD screen and it is also possible to be reached serially. A serial data plotting is also made. The flow meter works acceptable for this cost range.

APPENDIX – A

Arduino Codes

Code of the first try with physical equations.

```
/*
Emre Ay
Differential flow rate measuring project for Modelling & Control of
Biological Systems course, Spring 2015, Istanbul Technical University.

Bosch BMP180 sensors are used with an analog multiplexer.
*/
#include <SFE_BMP180.h>
#include <Wire.h>
#include <Math.h>
#include <LiquidCrystal.h>

#define selectPin 30
#define local_altitude 100.0
#define SELECT_BMP_NARROW digitalWrite(selectPin,LOW); //X0-Y0 is selected
#define SELECT_BMP_WIDE digitalWrite(selectPin,HIGH);  //X1-Y1 is selected
#define MBAR2MMHG 0.7500616827042
#define MBAR2PA 100
#define MCUBE2CC 10000
#define air_density 1.184
#define narrow 0.014
#define wide 0.02
#define gas_constant 287.058

SFE_BMP180 bmp_narrow, bmp_wide;

struct BMP_DATA
{
  String _id;
  double _pressure;
  double _temperature;
};

boolean readSensors(BMP_DATA *bmp1, BMP_DATA *bmp2);
BMP_DATA newBmp(String id);
LiquidCrystal lcd(45,43,41,39,37,35);

void setup()
{
       boolean failed = false ;
       pinMode(selectPin, OUTPUT);

       Serial.begin(9600);
        lcd.begin(16,2);
        //bmp narrow sensor is activated
       SELECT_BMP_NARROW
       if(bmp_narrow.begin()){
              Serial.println("BMP NARROW initialized successfully.");
       }
       else
        {
                failed = true ;
        }

        //bmp wide sensor is activated
       SELECT_BMP_WIDE
       if(bmp_wide.begin()){
                Serial.println("BMP WIDE initialized successfully.");
       }
```

```
        else
                failed = true ;

        if(failed){
                Serial.println("BMP sensor initialization failed");
                while (1);
        }
}

void loop()
{
         BMP_DATA narrow_data = newBmp("bmp_narrow");
         BMP_DATA wide_data = newBmp("bmp_wide");

         boolean sensor_is_read = readSensors(&narrow_data,&wide_data);


         if(sensor_is_read){
          /* print the value to serial terminal */
          Serial.println();
          Serial.print("provided altitude: ");
          Serial.print(local_altitude,0);
          Serial.println(" meters ");
           Serial.println("");

          /* Pressure Sensor #1 value */
          // Print out the measurement:
          Serial.println("BMP Narrow Data:");
           Serial.print("temperature: ");
          Serial.print(narrow_data._temperature,2);
           Serial.print(" deg C ");

          // Print out the measurement:
          Serial.print("absolute pressure: ");
          Serial.print(narrow_data._pressure,2);
          Serial.print(" mb ");
          Serial.print(narrow_data._pressure*MBAR2MMHG,2);
          Serial.println(" mmHg");

          Serial.println();
          /* Pressure Sensor #2 value */
          // Print out the measurement:
          Serial.println("BMP Wide Data:");
          Serial.print("temperature: ");
          Serial.print(wide_data._temperature,2);
           Serial.print(" deg C ");

          // Print out the measurement:
          Serial.print("absolute pressure: ");
          Serial.print(wide_data._pressure,2);
          Serial.print(" mb ");
          Serial.print(wide_data._pressure*MBAR2MMHG,2);
          Serial.println(" mmHg");

          Serial.println();

           double flow, v1,alpha, dummy, delta_pressure,ro1, ro2, temp1, temp2, A1;
           A1 = M_PI*pow(narrow,2);
           temp1 = narrow_data._temperature+273.15;
           temp2 = wide_data._temperature+273.15;
           ro1 = narrow_data._pressure*MBAR2PA/(gas_constant*temp1);
           ro2 = wide_data._pressure*MBAR2PA/(gas_constant*temp2);
           delta_pressure = -(narrow_data._pressure-wide_data._pressure)*MBAR2PA;
           dummy = (ro2*pow(narrow,2)-ro1*pow(wide,2));
           alpha = sqrt(2*delta_pressure/dummy);
           v1 = wide*alpha;
```

```
          flow = A1*v1;

          Serial.print("Flow [cc/s]:");
          Serial.print(flow*MCUBE2CC,3);
          lcd.setCursor(0,0);
          lcd.print(flow*MCUBE2CC,3);
       }
      else
         Serial.println("Error on sensor reading!");

  delay(500);
}
boolean readSensors(BMP_DATA *bmp1, BMP_DATA *bmp2)
{
       char status_narrow, status_wide ;
       double temp_narrow , temp_wide , pr_narrow, pr_wide;

       //activate bmp narrow
       SELECT_BMP_NARROW
        //function returns number of ms to wait for succes
        //and 0 for fail
       status_narrow = bmp_narrow.startTemperature();

        //activate bmp wide
       SELECT_BMP_WIDE
        //function returns number of ms to wait for succes
        //and 0 for fail
       status_wide = bmp_wide.startTemperature();

        //if the status are successful
       if (status_narrow != 0 && status_wide != 0)
        {
               // Wait for the measurement to complete:
              delay(max(status_narrow,status_wide));

                //activate bmp narrow
              SELECT_BMP_NARROW
                //funciton returns 1 for succes, 0 for fail
              status_narrow = bmp_narrow.getTemperature(temp_narrow);

                //activate bmp wide
              SELECT_BMP_WIDE
                //funciton returns 1 for succes, 0 for fail
              status_wide = bmp_wide.getTemperature(temp_wide);

                //if the status are successful
              if(status_wide != 0 && status_narrow != 0)
                {
                      /* save the temparature */
                      bmp1->_temperature = temp_narrow;
                      bmp2->_temperature = temp_wide;

                      //activate bmp narrow
                       SELECT_BMP_NARROW
                         //function returns number of ms to wait for succes
                         //and 0 for fail, the argument (0-3) is for oversampling value
                      status_narrow = bmp_narrow.startPressure(3);

                      //activate bmp wide
                       SELECT_BMP_WIDE
                         //function returns number of ms to wait for succes
                         //and 0 for fail, the argument (0-3) is for oversampling value
                      status_wide = bmp_wide.startPressure(3);

                      if(status_narrow != 0 && status_wide != 0)
                         {
```

```
                            delay(max(status_narrow,status_wide));

                            //activate bmp narrow
                              SELECT_BMP_NARROW
                                //funciton returns 1 for succes, 0 for fail
                            status_narrow = bmp_narrow.getPressure(pr_narrow,temp_narrow);

                            //activate bmp wide
                              SELECT_BMP_WIDE
                            status_wide= bmp_wide.getPressure(pr_wide,temp_wide);


                            if(status_narrow != 0 && status_wide != 0)
                                {
                                    /* save pressure value */
                                    bmp1->_pressure = pr_narrow ;
                                    bmp2->_pressure = pr_wide;

                                    return true ;
                            }
                                    else Serial.println("error reading pressure measurement\n");
                        }
                        else Serial.println("error starting pressure measurement\n");
                }
                else Serial.println("error reading temperature measurement\n");
        }
        else Serial.println("error starting temperature measurement\n");

        return false ;
}

BMP_DATA newBmp(String id)
{
  BMP_DATA _bmp ;
  _bmp._id = id ;
  return _bmp ;
}
```

Code of the alternative linearized approach.

```
/*
Emre Ay
Differential flow rate measuring project for Modelling & Control of
Biological Systems course, Spring 2015, Istanbul Technical University.

Bosch BMP180 sensors are used with an analog multiplexer.

Linearized model.
*/
#include <SFE_BMP180.h>
#include <Wire.h>
#include <Math.h>
#include <LiquidCrystal.h>

#define selectPin 30
#define local_altitude 100.0
#define SELECT_BMP_NARROW digitalWrite(selectPin,LOW); //X0-Y0 is selected
#define SELECT_BMP_WIDE digitalWrite(selectPin,HIGH);  //X1-Y1 is selected
#define MBAR2MMHG 0.7500616827042
#define MBAR2PA 100
#define MCUBE2CC 10000
#define air_density 1.184
#define narrow 0.014
```

```
#define wide 0.021

SFE_BMP180 bmp_narrow, bmp_wide;

struct BMP_DATA
{
  String _id;
  double _pressure;
  double _temperature;
};

boolean readSensors(BMP_DATA *bmp1, BMP_DATA *bmp2);
BMP_DATA newBmp(String id);

LiquidCrystal lcd(45, 43, 41, 39, 37, 35);

void setup()
{
  boolean failed = false ;
  pinMode(selectPin, OUTPUT);

  Serial.begin(9600);
  lcd.begin(16, 2);
  //bmp narrow sensor is activated
  SELECT_BMP_NARROW
  if (bmp_narrow.begin()) {
   // Serial.println("BMP NARROW initialized successfully.");
  }
  else
  {
    failed = true ;
  }

  //bmp wide sensor is activated
  SELECT_BMP_WIDE
  if (bmp_wide.begin()) {
   // Serial.println("BMP WIDE initialized successfully.");
  }
  else
    failed = true ;

  if (failed) {
    Serial.println("BMP sensor initialization failed");
    while (1);
  }
}

void loop()
{
  BMP_DATA narrow_data = newBmp("bmp_narrow");
  BMP_DATA wide_data = newBmp("bmp_wide");

  boolean sensor_is_read = readSensors(&narrow_data, &wide_data);


  if (sensor_is_read) {
    double diff;
    diff = (narrow_data._pressure - wide_data._pressure)*MBAR2PA;
    if (diff <= 100 && diff >= -100) diff = 0;
    Serial.println(diff*2);
    lcd.setCursor(0, 0);
    lcd.print(diff*2, 3);
  }
  else
    Serial.println("Error on sensor reading!");
```

```
  delay(500);
}
boolean readSensors(BMP_DATA *bmp1, BMP_DATA *bmp2)
{
  char status_narrow, status_wide ;
  double temp_narrow , temp_wide , pr_narrow, pr_wide;

  //activate bmp narrow
  SELECT_BMP_NARROW
  //function returns number of ms to wait for succes
  //and 0 for fail
  status_narrow = bmp_narrow.startTemperature();

  //activate bmp wide
  SELECT_BMP_WIDE
  //function returns number of ms to wait for succes
  //and 0 for fail
  status_wide = bmp_wide.startTemperature();

  //if the status are successful
  if (status_narrow != 0 && status_wide != 0)
  {
    // Wait for the measurement to complete:
    delay(max(status_narrow, status_wide));

    //activate bmp narrow
    SELECT_BMP_NARROW
    //funciton returns 1 for succes, 0 for fail
    status_narrow = bmp_narrow.getTemperature(temp_narrow);

    //activate bmp wide
    SELECT_BMP_WIDE
    //funciton returns 1 for succes, 0 for fail
    status_wide = bmp_wide.getTemperature(temp_wide);

    //if the status are successful
    if (status_wide != 0 && status_narrow != 0)
    {
      /* save the temparature */
      bmp1->_temperature = temp_narrow;
      bmp2->_temperature = temp_wide;

      //activate bmp narrow
      SELECT_BMP_NARROW
      //function returns number of ms to wait for succes
      //and 0 for fail, the argument (0-3) is for oversampling value
      status_narrow = bmp_narrow.startPressure(3);

      //activate bmp wide
      SELECT_BMP_WIDE
      //function returns number of ms to wait for succes
      //and 0 for fail, the argument (0-3) is for oversampling value
      status_wide = bmp_wide.startPressure(3);

      if (status_narrow != 0 && status_wide != 0)
      {

        delay(max(status_narrow, status_wide));

        //activate bmp narrow
        SELECT_BMP_NARROW
        //funciton returns 1 for succes, 0 for fail
        status_narrow = bmp_narrow.getPressure(pr_narrow, temp_narrow);

        //activate bmp wide
        SELECT_BMP_WIDE
```

```
        status_wide = bmp_wide.getPressure(pr_wide, temp_wide);


        if (status_narrow != 0 && status_wide != 0)
        {
          /* save pressure value */
          bmp1->_pressure = pr_narrow ;
          bmp2->_pressure = pr_wide;

          return true ;
        }
        else Serial.println("error reading pressure measurement\n");
      }
      else Serial.println("error starting pressure measurement\n");
    }
    else Serial.println("error reading temperature measurement\n");
  }
  else Serial.println("error starting temperature measurement\n");

  return false ;
}

BMP_DATA newBmp(String id)
{
  BMP_DATA _bmp ;
  _bmp._id = id ;
  return _bmp ;
}
```

APPENDIX – B

MATLAB Script

Matlab script for serial data reading and simultaneous plotting.

```
clear
clc

%User Defined Properties
serialPort = 'COM6';           % define COM port #
baudRate = 9600;
plotTitle = 'Volumetric Flow Rate Plot';              % plot title
xLabel = 'Elapsed Time [s]';                          % x-axis label
yLabel = 'Volumetric Flow Rate [cc/s]';               % y-axis label
plotGrid = 'on';                % 'off' to turn off grid
min = -1600;                    % set y-min
max = 1600;                     % set y-max
scrollWidth = 20;
delay = .01;

%Define Function Variables
time = 0;
data = 0;
count = 0;

%Set up Plot
plotGraph = plot(time,data,'LineWidth',3,'Color',[0.2,0,0.5]);

title(plotTitle,'FontSize',25);
xlabel(xLabel,'FontSize',15);
ylabel(yLabel,'FontSize',15);
axis([0 1 min max]);
set(gca,'ytick',[min:200:max])
grid(plotGrid);

%Open Serial COM Port
s = serial(serialPort,'BaudRate',baudRate)
disp('Close Plot to End Session');
fopen(s);

tic

while ishandle(plotGraph) %Loop when Plot is Active

    dat = fscanf(s,'%f'); %Read Data from Serial as Float

    if(~isempty(dat) && isfloat(dat)) %Make sure Data Type is Correct
        count = count + 1;
        time(count) = toc;    %Extract Elapsed Time
        data(count) = dat(1); %Extract 1st Data Element

        %Set Axis according to Scroll Width
        if(scrollWidth > 0)
        set(plotGraph,'XData',time(time > time(count)-
scrollWidth),'YData',data(time > time(count)-scrollWidth));
        axis([time(count)-scrollWidth time(count) min max]);

        else
        set(plotGraph,'XData',time,'YData',data);
        axis([0 time(count) min max]);
        end

        %Allow MATLAB to Update Plot
        pause(delay);
    end
end
```

```
%Close Serial COM Port and Delete useless Variables
fclose(s);
clear count dat delay max min baudRate plotGraph plotGrid plotTitle s ...
        scrollWidth serialPort xLabel yLabel;


disp('Session Terminated...');
plot(time,data)
```