SUNFEST Technical Report TR-CST01DEC05, Center for Sensor Technologies, Dept of Electrical and Systems Eng, Univ. of Pennsylvania, Philadelphia, PA 2005

University of Pennsylvania

SUNFEST

NSF REU Program
Summer 2005

LIQUID FLOW MEASUREMENTS USING A PYROELECTRIC ANEMOMETER

NSF Summer Undergraduate Fellowship in Sensor Technologies
Robert Callan, Electrical Engineering, University of Pennsylvania
Advisor: Dr. J.N. Zemel

ABSTRACT

The transport of fluids through pipes is an essential part of many industrial processes, requiring an accurate measurement of the flow. The measurement of natural gas flow in pipelines and the precise proportions of reactants flowing into a reaction chamber are but two examples. Many flow meters have been developed based on a wide variety of operating principles. Properties, such as physical size, speed, measurement accuracy, cost, reliability, and difficult recalibration often limit the usefulness of these meters in different applications.

The device investigated in this study is the Pyroelectric Anemometer (PA). Its operation is based on the convective heat loss from the device to the fluid in gas flows and standard heat transfer theory accounted nicely for the experimental data in previous research. This project addresses the pyroelectric anemometer's response to liquid flows. A test system was built and data were collected. These measurements were compared to gas flows and a previously derived fluid flow model.

Table of Contents

1. INTRODUCTION

The movement of fluids and their accurate measurement is important in many industrial processes. As a result, numerous sensors have been developed which rely on a number of different operating principles, including pressure differences, mechanical methods, and heat transfer characteristics to determine flow rates. The purpose of this report is to present the results of a study of the operation of a thermal flow sensor based on the pyroelectric effect. Previous research had established that the Pyroelectric Anemometer (PA) is remarkably precise, and may be used in flows whose Reynolds numbers (Re) range over 5 orders of magnitude. These studies showed that the PA met and/or exceeded the performance of commercially available sensors for a wide range of gas flows. [1]

This paper presents information on the PA's performance and behavior in liquid flows. Two absolute measurement systems were developed and an electronic signal processing system was developed to take measurements of the flow rate. Several differences between the responses to the liquid and gas flows are demonstrated, and their causes explored.

2. THEORY OF OPERATION

Pyroelectricity arises in certain types of crystalline materials as a result of ion motions as temperature changes. The result is a change of the surface charge that is directly proportional to the change in the temperature. In convective heat flow, the net heat flow depends on the difference in the temperature between the fluid and solid. The change in temperature of the solid is then a measure of the convective heat flow, and .a measure of the fluid motion. This is common experience where a breeze is a welcome coolant on a hot day.

The PA is used to determine flow rate by relating changes in convective heat loss from the sensor to the moving liquid. The heat loss is transduced to electrical signals by the pyroelectric effect with a 3x4mm $LiTaO_3$ crystal shown in Figure 1.

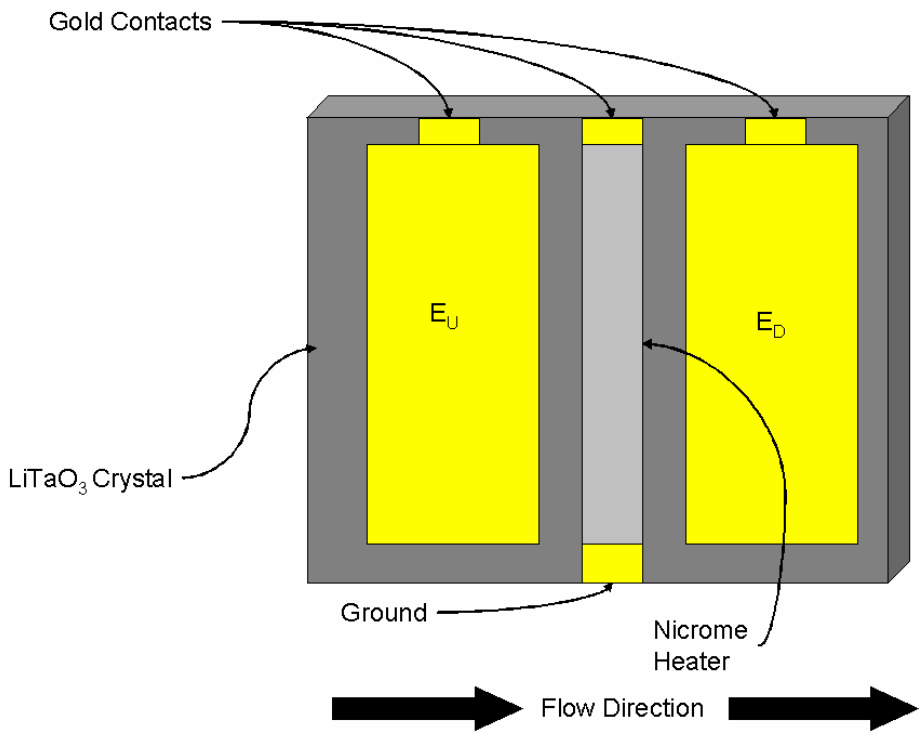

Figure 1: Layout of PA Crystal

The physical properties of the $LiTaO_3$ crystal can be used as in [1] to develop an equivalent circuit of the PA, which consists of a current source, capacitance, and resistance in parallel.

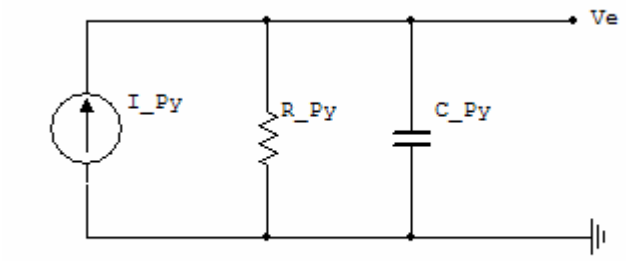

Figure 2: An equivalent circuit for a PA electrode

Because the $LiTaO_3$ crystal is a good dielectric, there is a capacitance C_py between the electrodes and ground. A resistance R_py models the leakage current that flows from the electrodes through the crystal to ground. The current source I_py is used to model the pyroelectric effect, where $I_{py} = \tilde{p} \cdot A_e \cdot \frac{d\Phi}{dt}$ where $\tilde{p}$ is the pyroelectric coefficient, A_e is the area of the electrode, and $\frac{d\Phi}{dt}$ is the change in temperature of the crystal (under the electrode) with respect to time.

The heater of the PA is driven by a sinusoidal voltage, adding an amount of heat proportional to the power dissipated in the heater resistor equal to

$$I^2 R = I_M^2 \cos(wt)^2 R = I_M^2 (1-\cos(2wt)) R/2.$$

This causes Joule heating which then changes the temperature (and heat content) of the device and then produces a varying surface charge on the crystal. This varying charge induces an external detectable current. Any net charge accumulation that arises as a result of the DC change in heat is neutralized over time, and all that remains is the AC component at twice the driving frequency. It is this signal which is processed and is related to the flow rate.

The two electrodes of the PA measure the changing charge on either side of the PA in a one dimensional flow. Heat flows from the heater, both through the crystal and through the fluid around the crystal. A change in the velocity of the fluid flowing over the crystal causes a change in the heat exchange between the crystal and the fluid. This heat exchange provides the connection between the velocity of the fluid flowing over the crystal and the change in the surface charge of the crystal. Previous gas flow research has demonstrated that the difference in the amplitudes of the response signals at the two electrodes is directly proportional to Reynolds numbers ≤ 10 and is proportional to the square root of the flow for a region of flows above Reynolds numbers of ~10 [1].

2.1 Electrical Response of the PA

The characteristics of interest in the system are the frequency, amplitude, and phase of the thermal input to the system called Φ_h (which is given by I^2R as above), as well as the frequency, amplitude, and phase of the thermal content of the crystal below each electrode, given as Φ_d, Φ_u (subscripts meaning upstream and downstream). All three of these signals are of the same frequency. Φ_h is determined by the voltage across the heating element, and its resistance and is the heat dissipated in the resistive element. The charge that develops on each of the electrodes as a result of the pyroelectric effect is proportional to Φ_d, Φ_u. Therefore $\frac{dQ_d}{dt}, \frac{dQ_u}{dt}$ are proportional to $\frac{d\Phi_d}{dt}, \frac{d\Phi_u}{dt}$.

$\frac{dQ_d}{dt}, \frac{dQ_u}{dt}$ are currents, and are proportional to voltages named V_d, V_u which are the quantities actually measured. The frequency of the voltage across the heater was set to 3Hz, resulting in a thermal response at 6Hz. Therefore, the quantities to be measured are the amplitude and phase of the 6Hz component of each of Φ_h, V_d, V_u (called $A_h, A_d, A_u, \theta_h, \theta_d, \theta_u$) as well as the flow rate (given as the Reynolds Number of the flow). These measurements were taken over a wide range of flow rates.

3. MEASUREMENT SYSTEM

3.1 Overview

A diagram of the complete measurement system is given in figure 3 below.

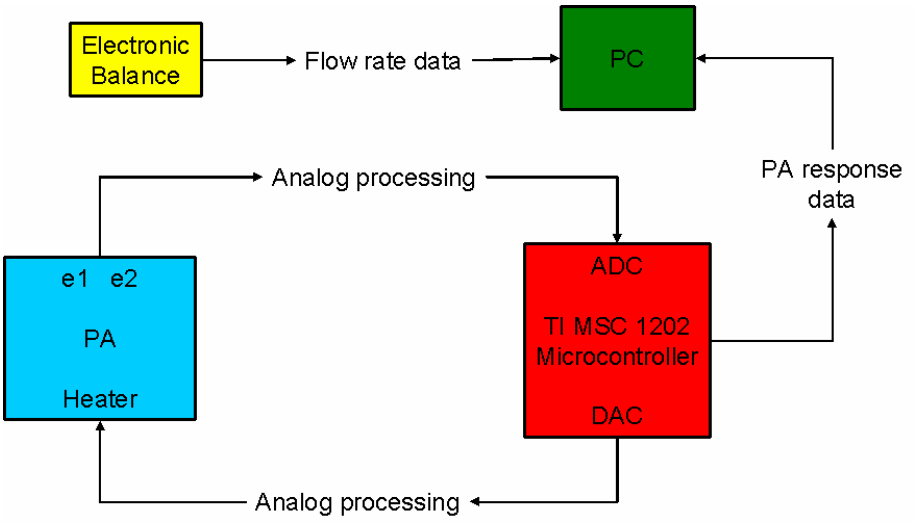

Figure 3: Diagram of the complete measurement system

The measurement system consists of 5 separate components: the PA, an analog processing section, an electronic balance, a PC, and a TI MSC1202 Microcontroller. The microcontroller outputs the driver signal on its DAC, which is filtered and amplified to excite the PA. The signals from each electrode of the PA, E_u, E_d are filtered and then each is sampled by the ADC of the microcontroller. The microcontroller converts these voltages to a digital representation, and sends them via an RS232 line to a PC for processing. Simultaneously, an electronic balance weighs a collection container to determine how rapidly the liquid is being collected. It sends this weight data over a second RS232 line to the PC for processing. The PC extracts the necessary data from these two sources, and determines the values of $A_u, A_d, \theta_u, \theta_d$, $flow$ and outputs them to the PC's screen. The process is repeated continuously as the flow rate is varied.

3.2 Electronics Onboard the PA

The PA sensor has the electronics shown in figure 4, directly connected to the electrodes on the crystal of the PA (at the point Ve in the figure). The raw voltage present at the point Ve is in the range of 2-20mV peak-to-peak. Each electrode has the buffer and amplifier shown below. The second op amp gives an amplification of 410 to bring the signal to usable levels.

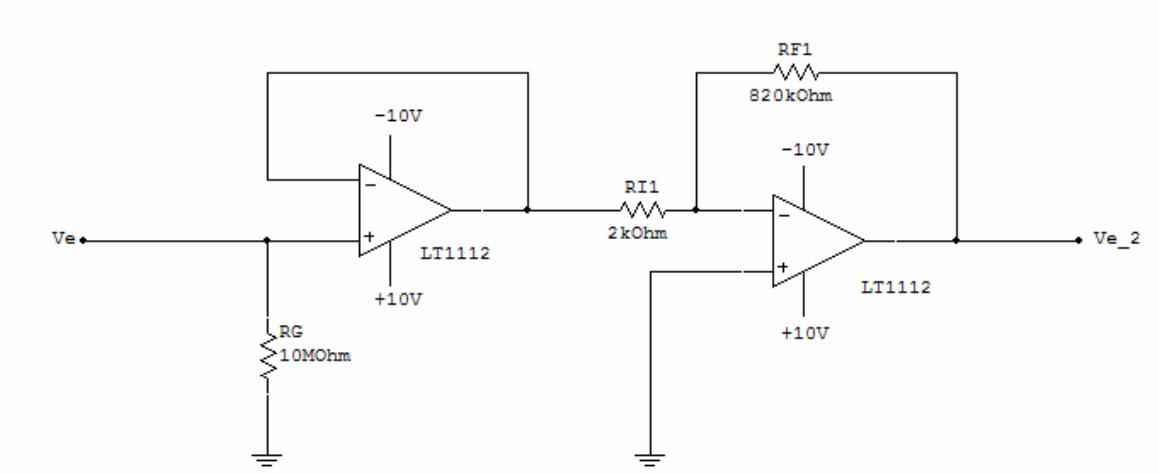

Figure 4: PA Onboard Electronics

3.3 Analog Filtering of the Reponse Signals

The PA is connected to a protoboard containing the electronics shown in figure 5. These electronics allow adjustment of the signal level and the addition of an offset of 1.25V. The ADC of the microcontroller has an input range of 0-2.5V, so it is necessary to adjust the signal level in the first stage, and to add the 1.25 offset.

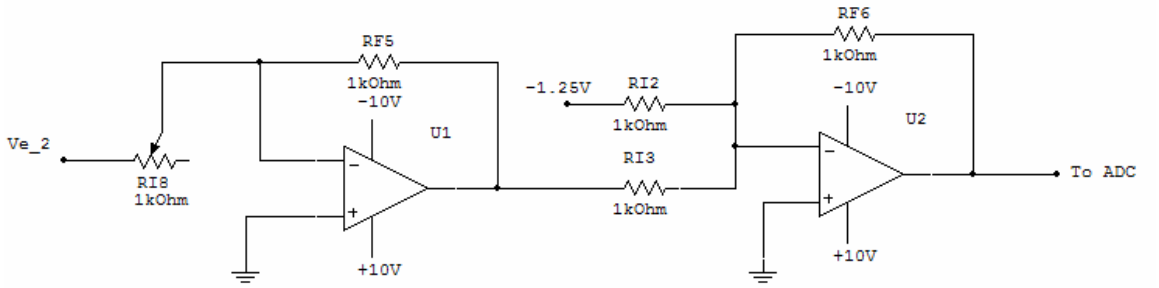

Figure 5: Analog Filtering of Response

3.4 Analog Filtering of the Heater Signal

The output of the DAC is a high impedance signal, and must be conditioned and amplified in order to drive the PA heater at the required energy level (20mW[1]). The resistance of the heating element is 800 Ohms, so an RMS voltage of 4V is needed. In addition, an offset of -0.5 V must be added to the signal because the output of the DAC is positive only. If there is an offset present in the input signal to the heater, this can have a negative effect on the response signals. A low pass filter is also necessary to smooth the heater signal.

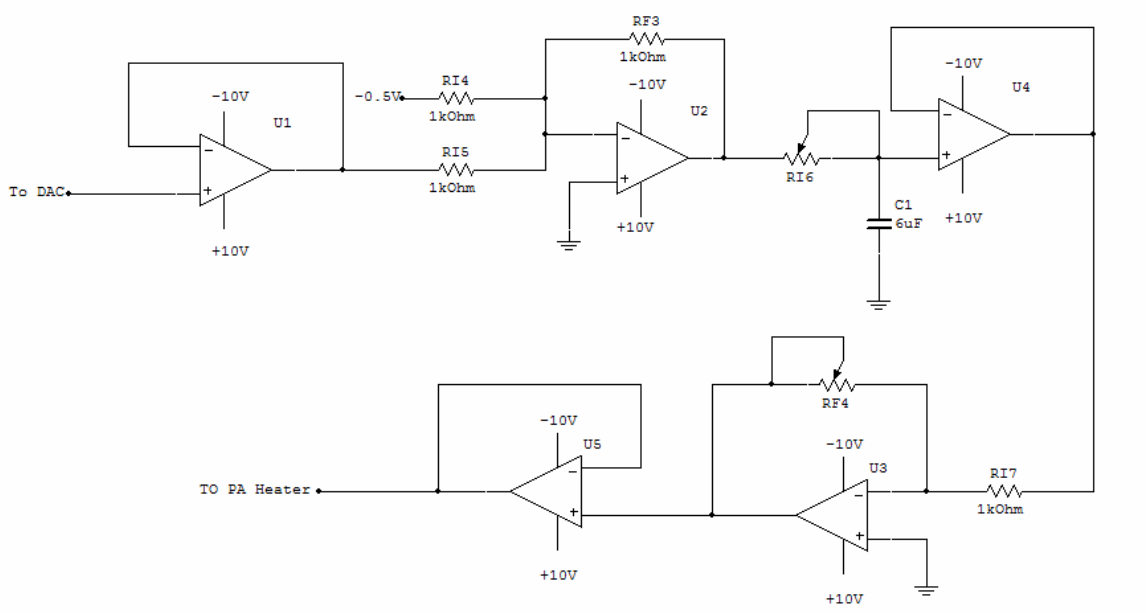

Figure 6: Analog filtering of the heater signal

3.5 TI MSC1202 Microcontroller

As shown in the System Diagram in Figure 3, the microcontroller generates the driver signal, samples the filtered responses of both electrodes, and sends this information to the PC for processing. The microcontroller chosen to do this is the TI MSC, an 8051 class Microcontroller which has a 16 bit Delta Sigma ADC, and an 8 bit IDAC. The ADC offers 16 Bits of resolution with an input voltage range of 0-2.5V, and the IDAC has an output current range of 0-1mA. Driving this current through a 1k resistor, results in an output voltage in the 0-1V range.

The microcontroller's ADC sampling rate controls the timing of the system. The signal of interest is at 6 Hz, and was sampled at 16 points per wave. The ADC is switching between two channels (one per electrode) so it must throw away every other measurement (due to the continuous sampling of the Delta-Sigma architecture). This means that the sampling rate of the ADC must be (2 Channels)*(16*6 Samples Per Second/Channel)*2(for thrown away measurements)= 384 Samples Per Second.

The microcontroller completes the following cycle every ~2.6 mSec (1/384):

1) Output the appropriate voltage on the IDAC to create a 3Hz sine wave
2) Record the voltage of the ADC conversion for this cycle
3) If cycle number is odd:
    a. Switch the Mux between the two input channels
    b. Output the raw ADC data through the RS232 link to the PC

If it were necessary to embed the measurement system such that the microcontroller was the only processor in the system, the microcontroller would also have to perform the amplitude calculations. This however was not desirable, because it was much easier to develop, monitor, and update code running on the PC.

4. FLOW SYSTEM

4.1 Use of the Electric Balance to Measure Flow

An AND electronic balance was used in these experiments to determine the flow rate. This was accomplished by relating a change in mass of the liquid collected to a change on volume of liquid (by knowing the density of the liquid) and then converting this to a velocity by knowing the area of the tubing at the PA element. The density was determined by using a 500mL standard flask to determine the weight in grams of 500mL of liquid. The density of the liquid tested, Wolf's Head Automatic Transmission Fluid, is .857 g/ml (g/cm^3), and the radius of the tubing is .292 cm^2. Therefore the conversion factor from grams/sec to cm/sec is grams/sec*1/.857 cm^3/gram*1/(.292 cm^2)= 3.996cm/sec. The accuracy of the scale is ± .01 gram, so the accuracy of velocity measurement is ~± .04 cm/sec.

4.2 Flow System and Absolute Measurement of the Flow

Two flow systems were used to generate a steady flow. They had in common the PA sensor element, and a collection container resting on a balance. The flow rate was determined by measuring the difference in masses over a period of 2 2/3 seconds. This was related to the velocity of the flow as described in "Use of the Electronic Balance to Measure Flow."

The measurement system designed for low flows is shown in Figure 7. It uses the pressure developed by a column of liquid to create a flow past the sensor. The large tube at the left of the figure is filled with liquid, and measurements of the PA response and flow rate are simultaneously collected as the liquid drains from the tube. This system requires no intervention from the experimenter.

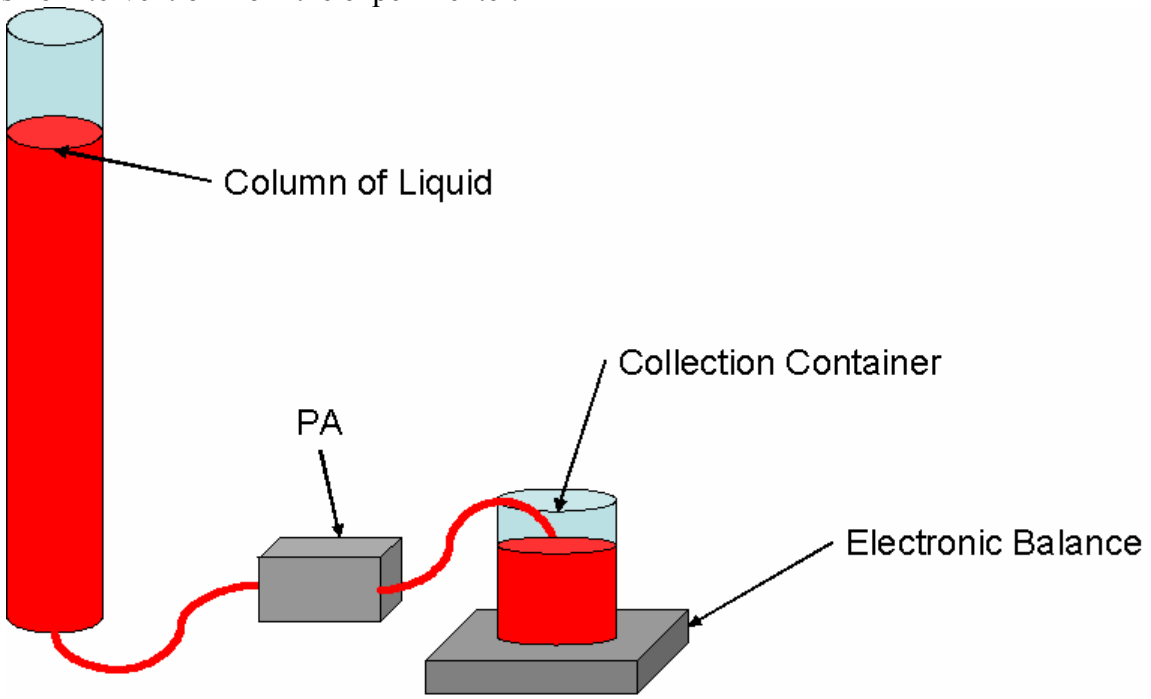

Figure 7: Flow system for lower flow rates

The second measurement system, which is used for higher flow rates, is shown in Figure 8. It uses a pump to develop a flow past the sensor. This allows for higher flow rates, but requires more time to take measurements because the speed of the pump must be adjusted gradually by hand. The higher flow rates also require more frequent emptying of the collection container, which complicates the process of taking accurate measurements.

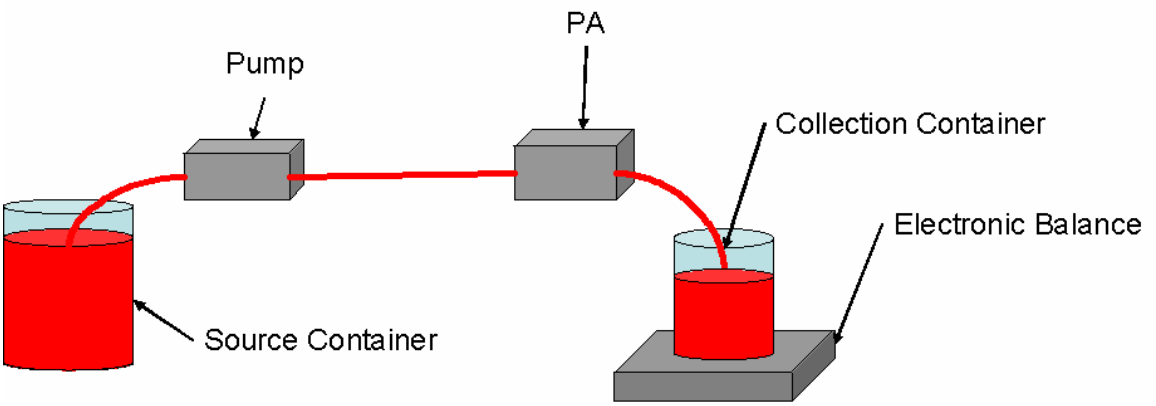

Figure 8: Flow system for higher rates

5. ROLE OF THE PC

The PC combines and analyzes all the sensor data and displays the end result: the PA amplitude and phase data $A_d, A_u, \theta_d, \theta_u$, as well as the flow rate past the sensor in cm/sec. Each of these measurements is averaged over approximately 2.6 seconds (256 sampling points per each channel) and then output and recorded continuously.

All the software used to achieve this task is written in Java. The RS232 ports are controlled using an API from Sun called javacomm. Java was chosen for two reasons: 1) it could perform all the necessary calculations within a single synchronized process, and 2) It provides excellent error reporting and handling.

Figure 9 shows a functional flow diagram of the program used to process the flow data. The yellow boxes are devices linked to the PC through RS232 ports and the blue boxes are java classes.

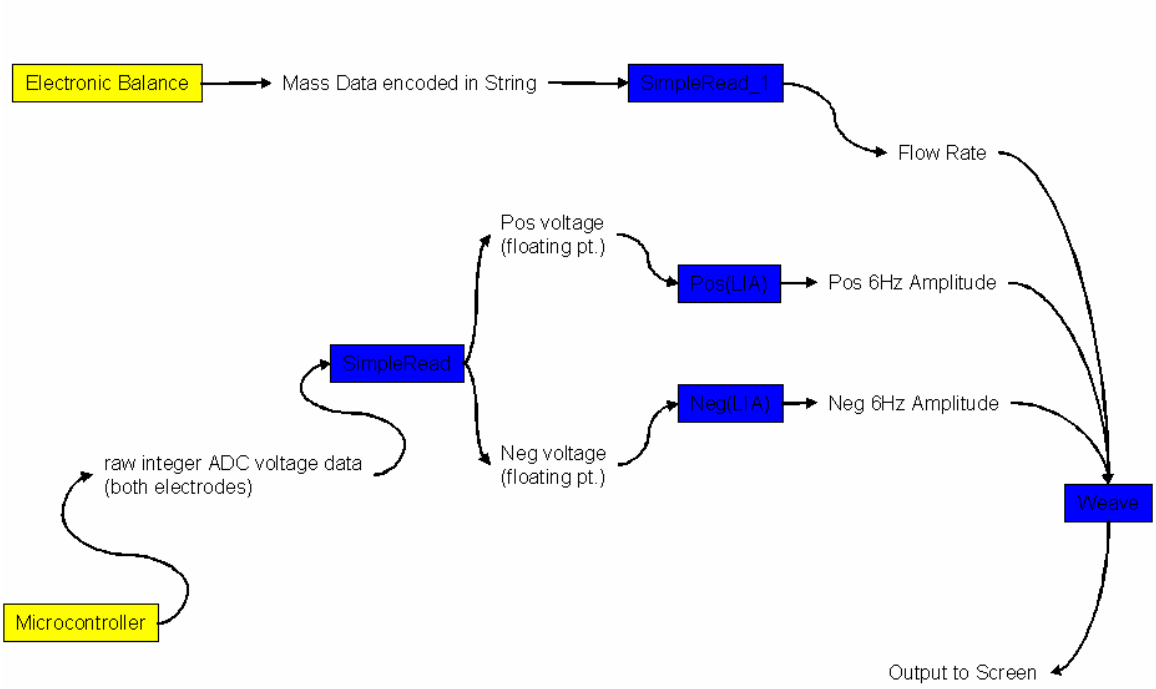

Figure 9: Diagram of the software used to determine PA response amplitudes and flow rate

The data sent from the microcontroller and the balance is converted into a usable form first. Two instances of the LIA class each collect 256 voltage measurements from the ADC, and then perform the LIA procedure described below.

5.1 The Lock-In-Amplification (LIA) Procedure

The purpose of this algorithm is to calculate the amplitude and phase of a sinusoid of a particular frequency. This is accomplished by multiplying the signal by $e^{-12\,j\pi t}$ and then averaging over a period of time. The complex amplitude of the 6Hz component(X) of the input signal s(t) is given by:

$$X = 2\,\frac{\int_0^T s(t)\,\mathbf{e}^{(-12j\,\pi\,t)}\,dt}{T}$$

The result is multiplied times 2 because the integral gives only the amplitude of the positive frequency component $\mathbf{e}^{(j\,12\,\pi\,t)}$ of $\cos(12\,\pi\,t) = \frac{\mathbf{e}^{(j\,12\,\pi\,t)} + \mathbf{e}^{(-j\,12\,\pi\,t)}}{2}$ and divided by T so that all the other frequency components sum to zero. The magnitude of the component X is given by $|X| = \sqrt{\Im(X)^2 + \Re(X)^2}$ and the phase is given by $\text{angle}(X) = \arctan\left(\frac{\Im(X)}{\Re(X)}\right)$. This is implemented by taking

$XReal = \int_0^T s(t)\cos(12\pi t)\,dt$ and $XComplex = \int_0^T s(t)\sin(12\pi t)\,dt$. This gives the real and complex parts of X. These equations hold because $e^{(j\theta)} = \cos(\theta) + j\sin(\theta)$.

The code below (Taken from the class LIA given in appendix A) shows how this process is actually performed. The array pts[] contains the voltage data from the ADC over a period of 2 2/3 seconds. The double[] sines and cosines contain 6Hz sine and cosine waves respectively.

```
public double[] pts= new double[256];
public double[] sines= new double[pts.length],
                cosines= new double[pts.length];
 public double calculateAmplitude(){
     double sineSummation=0, cosineSummation=0;
     for(int i= 0; i<pts.length; i++){
         sineSummation+= pts[i]*sines[i];
         cosineSummation+= pts[i]*cosines[i];
     }
     return Math.sqrt(sineSummation*sineSummation
+cosineSummation*cosineSummation)*2./pts.length;
}
public double calculatePhase(){
    double sineSummation=0, cosineSummation=0;
    for(int i= 0; i<pts.length; i++){
       sineSummation+= pts[i]*sines[i];
       cosineSummation+= pts[i]*cosines[i];
    }
    return Math.atan(cosineSummation/sineSummation)*180/Math.PI;
}
```

6. ANALYSIS OF DESIGN

6.1 Analog vs. Digital Implementation

Research previously done with the PA in gases used an analog *differential* of the two input signals $(V_d - V_u)$ and then sampled the signals, and found the amplitude of this differential signal. This method assumes that the phase difference between the two electrodes is zero, and more importantly, that the two signals remain completely in phase over all measurements. These two assumptions were true in gases. However, it was observed in the oil tested that this was not the case. The phase between the two electrode signals at zero flow differed by ~10 degrees and the difference changed by more than 20 degrees at high flow rates. Therefore it would not be feasible to use an analog differential of the two signals, if a pure amplitude must be known. However, it would be possible to take the differential for practical purposes, ignoring the fact that the resulting signal would be a combination of phase differential and amplitude change. Figure X shows the resulting error surface when the differential of two signals is taken.

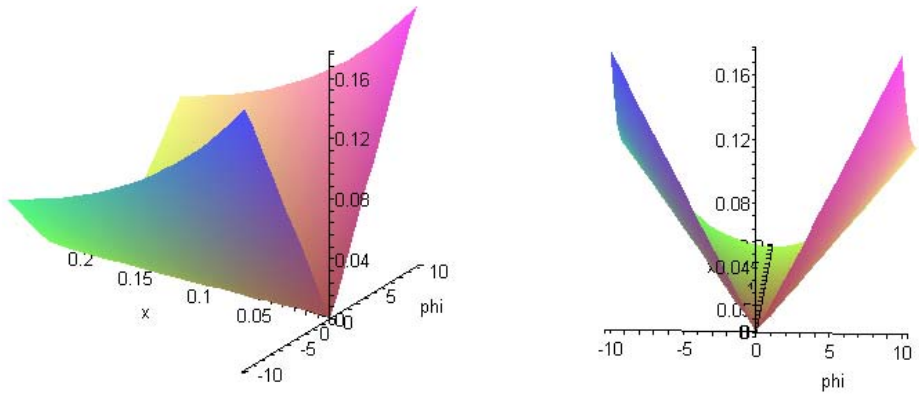

Figure 10: Error Surfaces resulting from Differences in Phase between Electrodes

These two figures show the result of $\frac{amplitude(a(t)-b(t))}{a(t)}$ where $a(t)=(1+x)\sin(wt), b(t)=\sin(wt+phi)$.

In order to eliminate this source of error entirely, a digital difference of the amplitudes is taken. The two signals $V_d(t), V_u(t)$ are each sampled separately and each of their amplitudes and phases determined. Then these two amplitudes are normalized and their difference $\frac{A_u}{A_{u0}}-\frac{A_d}{A_{d0}}$ is related to the flow rate.

6.2 Synchronization

The synchronization between the heater signal and the frequency at which the calculations take place must be as close as possible. Slight differences in frequency between the signal of interest contained in pts[], and signals sines[] and cosines[] can have a dramatic effect on the amplitude and phase data calculated using this algorithm. The results of such a difference are shown in Figure 11.

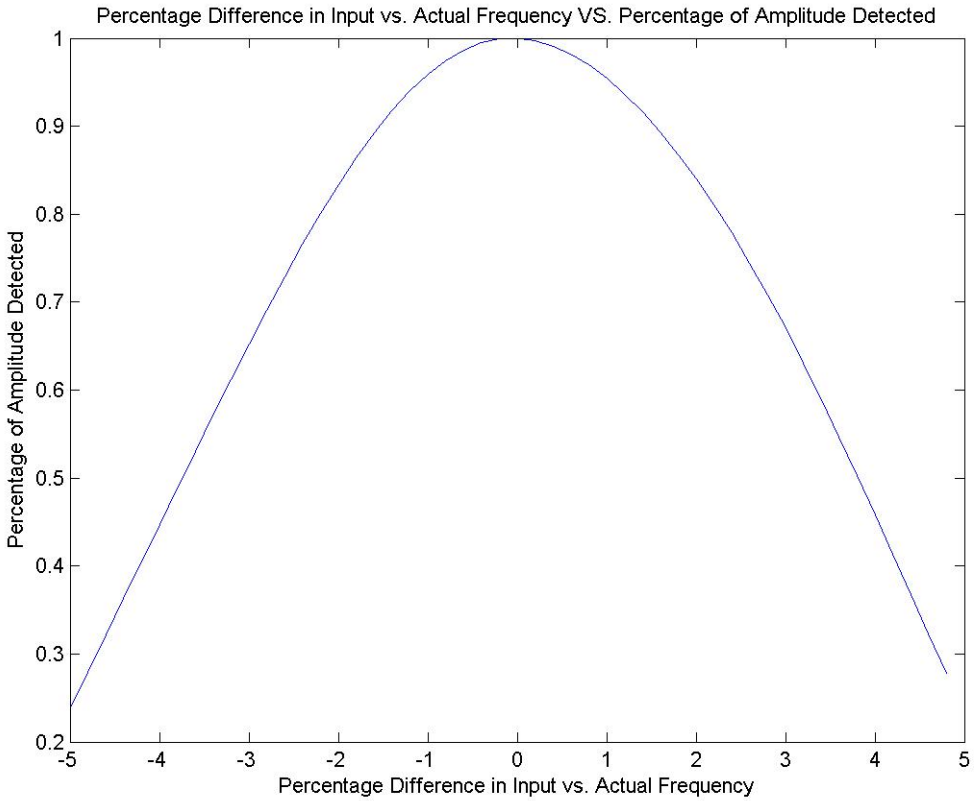

Figure 11: Change in measured amplitude as a result of differences in frequency

The results suggest that even relatively small differences in frequency can result in large errors, especially if the differences in frequency change during the course of the experiment (as was observed when using a low quality function generator). This analysis shows that the heater signal must be synchronized directly with the sampling of the ADC, which was accomplished in this system by using a microcontroller with DAC and ADC driven by the same clock.

7. EXPERIMENTAL RESULTS

7.1 Summary of Previous Research

There are three conclusions from previous research that we were able to compare to our results. The first is the theoretical models developed to model the response of the PA to gas flows. It was found that in flows of lower Reynolds number, theory predicts that the response of the PA will be linearly related to the Reynolds number. This model agreed quite well with experimental data collected in Nitrogen, Helium, and Argon gases. As the flow rate increases sufficiently, the response transitions to the square root of the Reynolds number according to theory. This result also agreed with the experimental data.

The second result is related to the phase of the responses with respect to the heater signal. It was found in the gases that as the Reynolds number increased, the phase of the differential response with respect to the heater signal remained constant up to a point,

then changed quickly for a period, and then saturated again at another value. In the case of the liquid flow measurements made in this study, the phase difference between the two electrodes varied as shown in Figure 17.

The third conclusion relevant to the data collected in this study was that the single electrode response and normalized differential response of the PA were relatively insensitive to changes in temperature and pressure. While thorough testing over a wide range of temperatures was not done, it was noted that small changes in temperature did not affect the response of the device. All these conclusions were reached in [1].

7.2 Summary of Collected Data

Four sets of data were collected. The first set of data to be collected used the flow system employing a column of liquid to generate the flow. No effort was made to monitor the temperature of the oil, the phase of the signals, or the timing between trials. The second set of data was collected from the high flow system, which used a pump to generate a flow. The phase of data was collected along with the amplitudes of the responses. Again, no attempt was made to hold the temperature of the oil constant, or to monitor the amount of time passing between trials. The third set of data collected recorded the response of the PA, holding the flow rate constant, and varying the temperature, as well as the passage of time between trials. The fourth set of data collected was similar to the second, however the temperature of the oil, and the amount of time between trials was held constant. All flow rates are given in cm/sec, because the viscosity of the oil had not been determined.

7.3 Constant Flow Response

After a significant amount of variability was detected in preliminary measurements, several measurements were taken to determine whether the responses would change over time, even as the flow rate was held constant. Figure 12 shows one

such set of measurements.

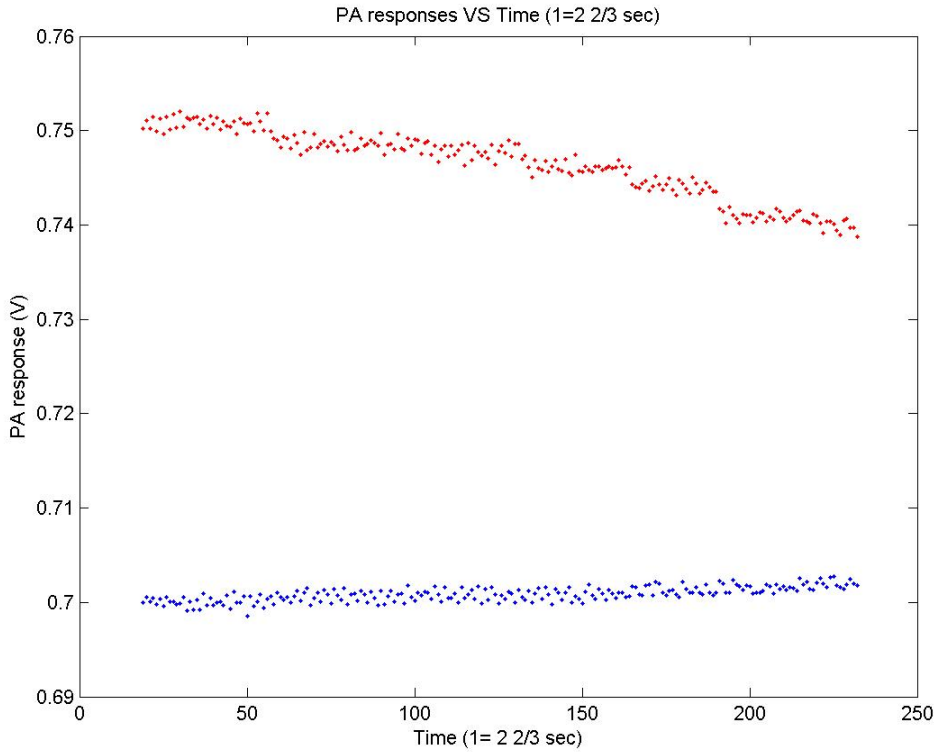

Figure 12: PA responses VS. time with pump current at .35 amps

This data was collected by setting up the pump in a closed loop system and allowing it to run as measurements were taken. Under ideal conditions, the two responses should remain constant because the flow rate should be constant. However, it is believed that neither the flow rate, nor the speed of the pump was constant over the course of this measurement as will be described later. Further measurements were taken, and it was determined that after a period of time the signal levels reached a final level (though this level was not the same between trials).

Another experiment was performed to determine the temperature sensitivity of the PA. At room temperature (20^{o}C), the zero flow responses of the two electrodes are approximately 15mV. However, if the oil is heated to ~40^{o}C, the responses drop to ~11.7mV. When the oil temperature is 0^{o}C, the responses remained at 15mV.

7.4 PA Response to Low Flow

The data collected in these trials was taken by filling the column of oil in the low flow system and allowing the liquid to drain into the collection container on the balance. This was done four times, and the results are presented in figures 13 and 14.

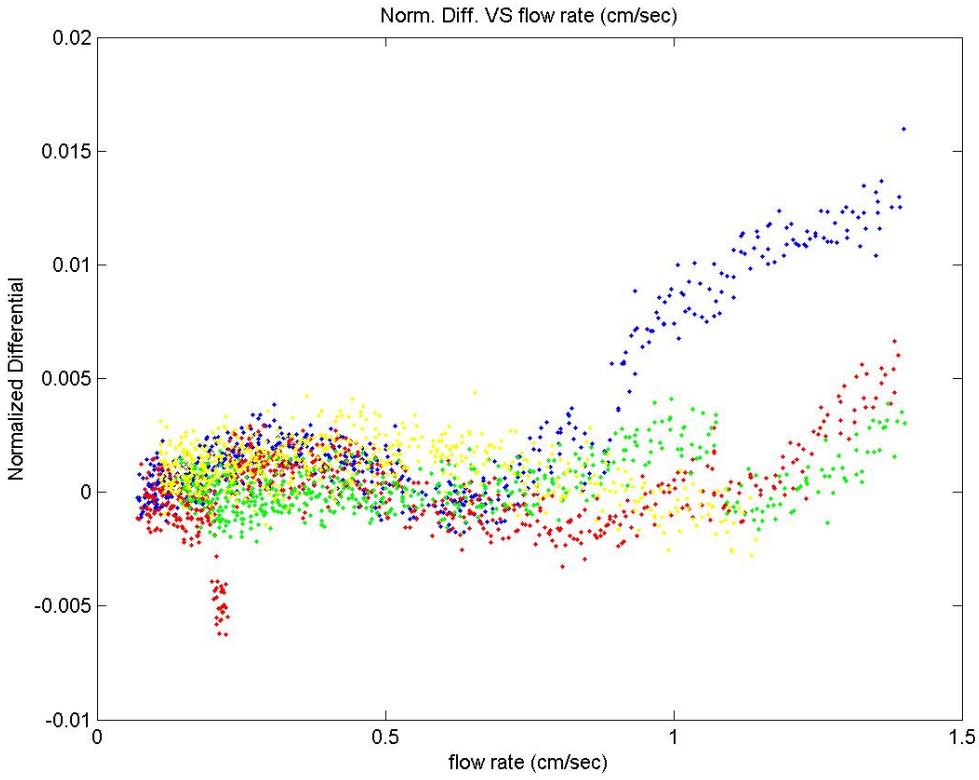

Figure 13: Normalized Differential Response at low flow rates on linear scale

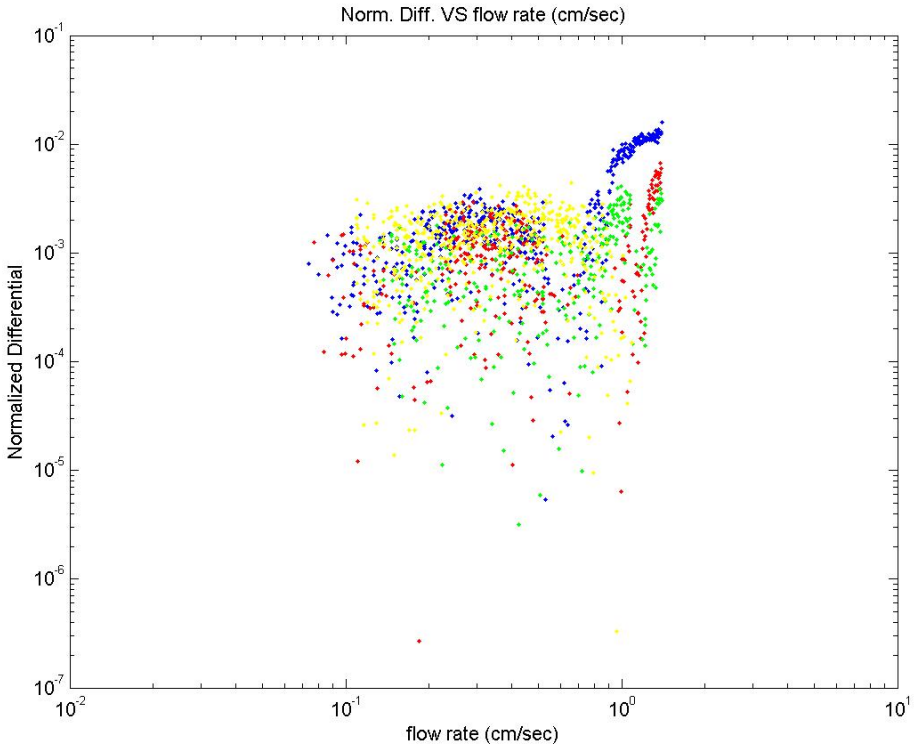

Figure 14: Normalized Differential Response in low flow rates on log log scale

The results are clearly not reproducible across trials. It appears that the response is increasing as the flow rate increases, but this varies between trials. There also appear to be long term fluctuations in the response. This is believed to be caused by a change in temperature, as will be explain later. The temperature of the oil most likely fluctuated over the course of the experiment, as the temperature of the oil and the tubing approached the same temperature. It is believed that either the measurement system is not sensitive enough to detect the response accurately, or that the PA response itself is not significant at these flow rates. The measurements made in the gases had a "noise floor" below which the response could not be measured as a result of thermal noise, as well as noise in caused by the electronics. It is possible that the situation is similar at these rates in the liquids.

The oil was initially at room temperature, as was the pump. As the pump circulates the oil, there are two occurrences which could possibly explain this data. The friction of the moving parts of the pump causes its temperature to rise. This produces two results. First, the speed of the pump, (and therefore the flow rate) decreases slightly because of the increased resistance. Second, the oil begins to heat up as it draws heat away from the pump.

7.5 PA Response to High Flow Rates

Two sets of data were collected at high flow rates. The first set of data consists of 8 separate trials during which the effects of temperature and the time elapsed between trials were disregarded. The speed of the pump was varied gradually to achieve different flow velocities. The normalized differential response vs. flow rate is shown in figure 15.

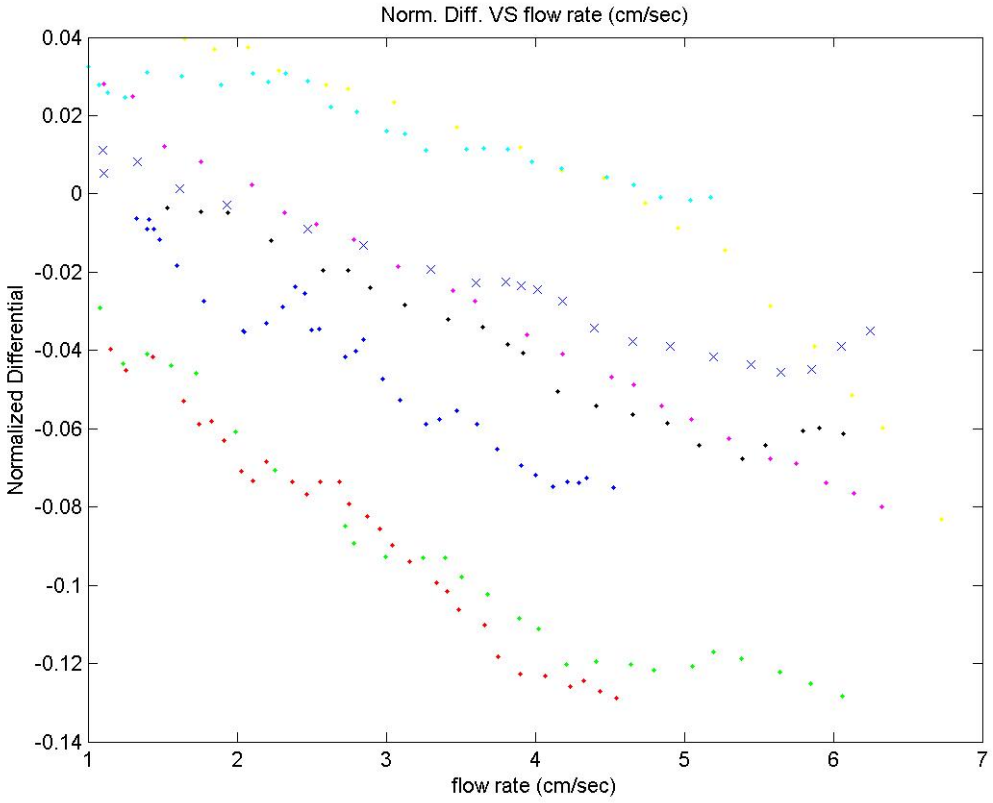

Figure 15: Norm. Diff. vs flow rate, neglecting effects of temperature

The second set of data was collected in a way to minimize the effects of temperature and the passage of time. The pump was run in a closed loop system for 30 minutes with the PA electronics on to allow the temperature of the system to reach an equilibrium. Then eight trials were run, with approximately 30 seconds between trials. The trials appeared to be more repeatable, as shown in figure 16.

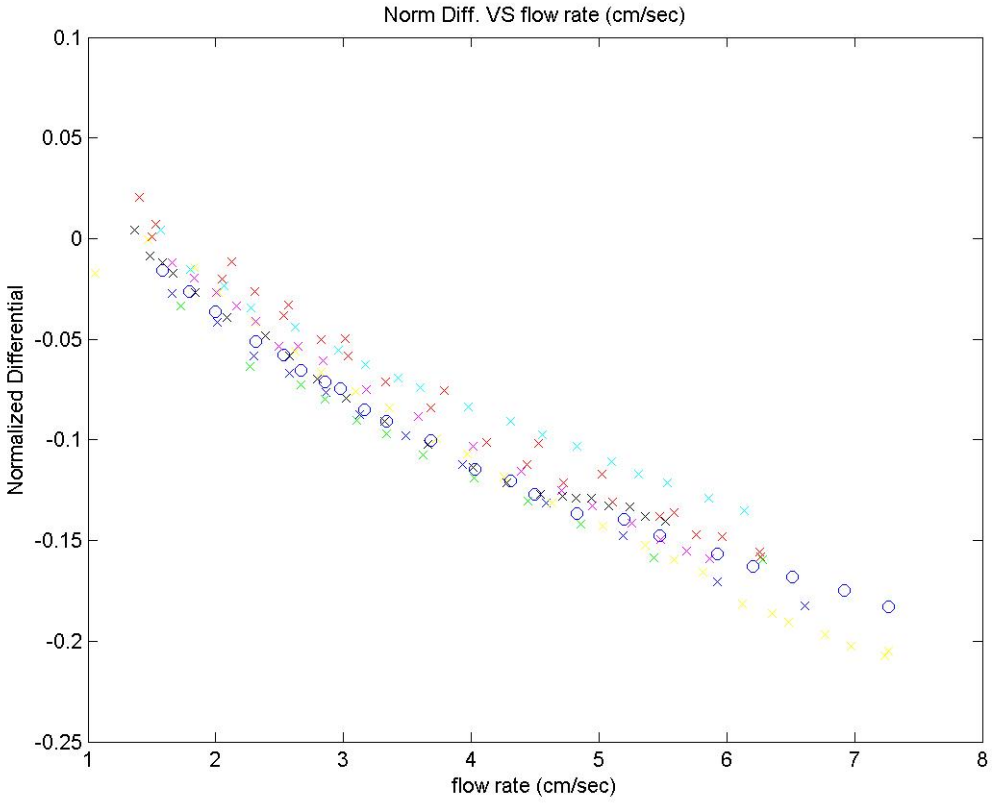

Figure 16. Normalized Differential vs flow rate under steady temperature conditions in linear scale

The phase for the second set of high flow measurements is shown in figure 17. It is clear that a direct comparison to previous research will not be possible because the phase of the two electrodes relative to one another changes with flow rate. This did not happen in measurements taken in gases.

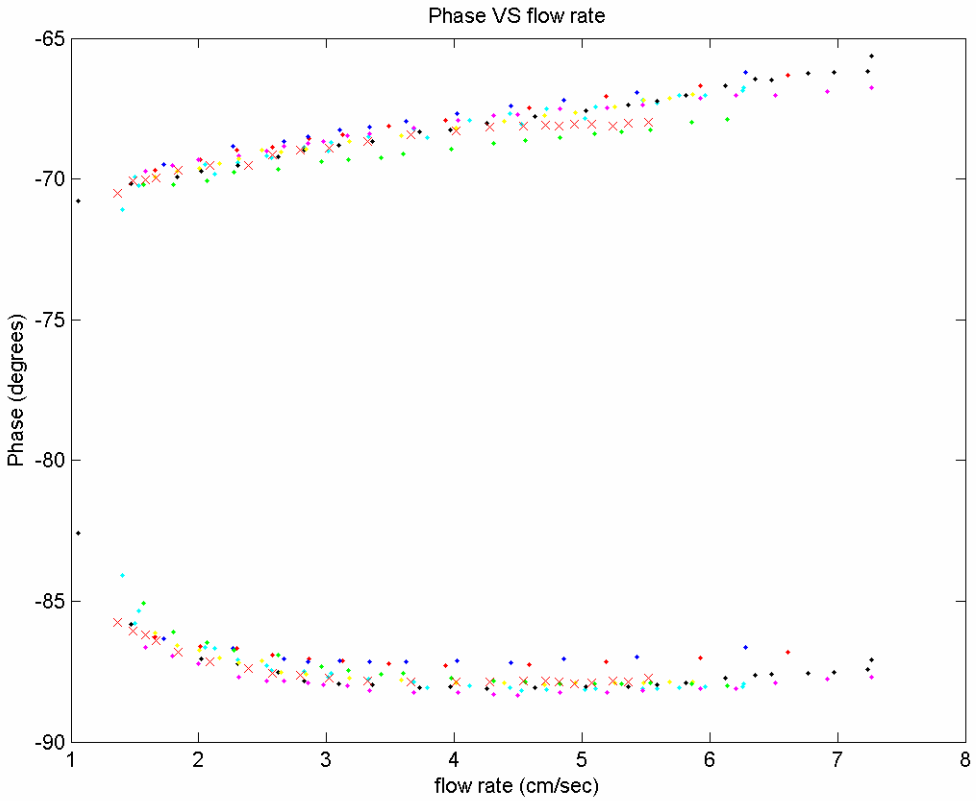

Figure 17: Phases of the two electrodes versus flow rate (phase of upstream electrode greater than -70, downstream electrode less than -80)

8. DISCUSSION AND CONCLUSIONS

8.1 The Effects of Temperature

The experimental results indicate that there are similarities, as well as differences between the results of previous research and this study. There are several major differences between the properties of helium, nitrogen, and argon gases and the properties of the oil tested. First, the thermal conductivity of these gases is much lower than that of the PA and of the oil. Second, the viscosity of the oil was much more sensitive to changes in temperature than that of the gases. This is important because the Reynolds number of the flow is given by $\mathrm{Re} = \frac{V \cdot D}{v}$ where V is the velocity of the flow, D is the diameter of the tubing, and v is the kinematic viscosity of the oil. This means that as the viscosity of the oil changes, the Reynolds number (and therefore the PA response) will change as well, even if the volumetric velocity of the oil remains constant. Since the PA response is related to the Reynolds number, it is important to examine in detail how changes in temperature affect the viscosity, the resulting Reynolds number and the PA response.

Several observations were made of the temperature over the course of the measurements, and it was found that the temperature of the oil varied in the range 20-25^{o}C. Room temperature was ~20^{o}C. It is believed that as the oil circulates through the

pump, it is heated, depending on how fast the pump is running, and how long it has been on. When the oil is collected on the balance, it cools down again, approaching room temperature. A viscosity-temperature curve of the oil that was used (transmission fluid) was not available, but one was available for SAE 10W oil, which is believed to have similar characteristics. Over the observed temperature range, the kinematic viscosity of 10W oil varies from .0001 to .00006 m^2/sec. It is also noted that the specification of 10W oil allows for a 50% variation in these figures "especially at low temperatures".[2] It is not known what the specification for viscosity of the transmission fluid is for these temperatures. This would result in a range of error of ± 30% in the Reynolds number (and therefore PA response, assuming a linear relationship) for a constant volumetric flow.

Further complicating the effects of temperature on the PA's behavior is the observation that the single electrode response of the PA appears to decrease with an increase in temperature. It was observed that the amplitude of the single electrode response at zero flow decreased from 15mV to 11.7mV when the temperature was changed from 20°C to ~40°C. This results in a single electrode response of 14.2mV at 25°C assuming linearity. The exact signal-temperature relationship was not determined. This change in signal level can have a profound effect on the normalized differential response if the temperature change occurs during the trial. If the sensor's zero flow response is measured when the temperature is 20°C, and a normalized difference measurement is taken over temperatures ranging from 20-25°C (as is believed to be the case), a significant amount of variability would be expected.

The exact effect of temperature is difficult to determine because the temperature of the liquid was not recorded during any of the flow measurements. However, these preliminary calculations seem to suggest that the effects of a relatively minor fluctuation in temperature (+5°C) could result in relatively large errors.

8.2 Comparisons to Previous Research

There were three main points to be compared to the previous research. The first was that the PA response is linearly related to the flow rate for lower Reynolds numbers. While the Reynolds numbers of the flows used could not be determined, there appears to be a linear relationship between the volumetric flow rate of the oil and the PA response. Assuming the viscosity of the oil remained relatively constant, the PA response is found to be linearly related to the Reynolds number as had been observed in gases. However, the significant fluctuation in signals between trials, and within the trials themselves, suggest that the most important factor affect the PA response may be the temperature.

The second result of previous research was that the phase between the two electrodes of the PA with respect to each other remained constant regardless of the flow rate. This can not be said for the data collected for flows in oil. The phase of both electrodes clearly changes with respect to the heater signal and with respect to each other. This makes it difficult to make a comparison to the phase changes observed in the gases.

The third result of the research conducted previously was that the PA response (both single electrode and normalized differential) was not significantly affected by small changes in temperature. The effect of temperature was not under study in previous research, but it appears as though the changes in temperature (if they occurred) did not

lead to variability in signal levels of the PA. At room temperature, the viscosity of gases is much less dependent on temperature than oil. [2]

9. RECOMMENDATIONS

This study uncovered several characteristics of liquid flows that could possibly warrant changes in the current design of the measurement and flow systems. The most obvious is the apparent effect changes in temperature have on the PA response. Future research should attempt to keep the temperature of the liquid as constant as possible during and between trials, and also to quantitatively determine the effect temperature has on the response of the PA. The temperature should be monitored during all trials. Also, switching the order of the PA and pump in the tubing (so that any heat generated by the pump would not effect the temperature as it passes the PA) might improve the variability of the measurements. It also might be beneficial to develop a system that does not require a pump, such as a larger version of the low flow system developed, as it is likely the temperature of the oil in this type of system would remain more constant. It might also be worthwhile to consider the temperature vs. viscosity curve when selecting future liquids to be tested, as it would be desirable to keep the viscosity as constant as possible.

It would also be beneficial to develop a system that could handle gas as well as liquid flows. This would allow more direct comparisons to previous research and would ensure that the measurement electronics themselves are not contributing to the uncertainty and error in the measurements.

10. ACKNOWLEDGMENTS

I would like to thank Dr. Zemel for his help, advice, support and encouragement in completing this project. I would also like to thank the National Science Foundation for their support through an NSF-REU grant and Microsoft Corporation for their financial support.

11. REFERENCES

1. Hsieh,Hsin-Yi, . "Pyroelectric Anemometers. A Dissertation in Electrical Engineering" 1993.
2. F.M. White, *Fluid Mechanics*, Mc Graw-Hill, New York, 3rd ed., 1994, p. 700-702.

APPENDIX

JAVA CODE

This code was compiled an run on a PC using the javacomm API available on sun.com. These java classes read the data sent over the rs232 line from the microcontroller and from the electronic balance and determine the amplitude of each electrode as well as the flow rate.

(start LIA.java)

```
/*
 * LIA.java
 *
 * Created on June 16, 2005, 11:22 AM
 */

package commapi.samples.Simple;

/**
 *
 * @author  Rob
 */
public class LIA {
    public Weave weave;
    public String name;
    public int ptsIndex=0,ptsPerWave=16;
    public double[] pts= new double[256];
    public double[] sines= new double[pts.length], cosines= new double[pts.length];
    public double amplitude,phase;
    public boolean full= false;
    public double calculateAmplitude(){
        double sineSummation=0, cosineSummation=0;
        for(int i= 0; i<pts.length; i++){
            sineSummation+= pts[i]*sines[i];
            cosineSummation+= pts[i]*cosines[i];
        }
        return
Math.sqrt(sineSummation*sineSummation+cosineSummation*cosineSummation)*2./pts.length;
    }
    public double calculatePhase(){
        double sineSummation=0, cosineSummation=0;
        for(int i= 0; i<pts.length; i++){
            sineSummation+= pts[i]*sines[i];
            cosineSummation+= pts[i]*cosines[i];
        }
        return Math.atan(cosineSummation/sineSummation)*180/Math.PI;
    }
    public void printAll(){
        System.out.println(name+"pts: ");
        for(int i= 0; i<pts.length; i++){
            System.out.println(pts[i]);
        }System.out.println();

    }
    public void printIDAC(){
```

```
        System.out.println(name+"IDAC: ");
        for(int i= 0; i<pts.length; i++){
            System.out.println(IDAC[i]);
        }System.out.println();

    }
    public final double LSB= 3.8147e-5;
    /** Creates a new instance of LIA */
    public LIA(String name,Weave weave) {
        this.name= name;
        this.weave= weave;
        for(int i= 0; i<pts.length; i++){
            sines[i]= Math.sin((i*2*Math.PI)/ptsPerWave);
            cosines[i]= Math.cos((i*2*Math.PI)/ptsPerWave);
        }
    }
    public int[] IDAC= new int[pts.length];
    public void getNextInt(int next,int IDAC){
        if(full) {System.out.println("ERROR: LIA was full, tried to rewrite."); System.exit(0);}
        this.IDAC[ptsIndex]= IDAC;
        pts[ptsIndex++]= LSB*next;

        if(ptsIndex==pts.length){

            //System.out.println(name+" amp: "+calculateAmplitude());
            weave.LIADone(this);
            ptsIndex=0;
        }

    }
    double phaseX= Math.random(), phase2= Math.random();

    public double signalSource(int i, double percentDiff){
        return Math.sin(i*2*Math.PI/(ptsPerWave)*(1+percentDiff));
    }
    public void fill(int i,double percentDiff){
        phase= Math.random();
        for(int j= 0; j<pts.length; j++){
            pts[j]= signalSource(i+j,percentDiff);
        }
    }
     public static void main(String[] args) {
        LIA l= new LIA("NONE",null);
        for(int i= 0; i<50; i++){
            l.fill(0,(i-25)*10./50./100.);
            System.out.println((i-25)*10./50+"\t"+l.calculateAmplitude()+"\t"+l.calculatePhase()+";");
        }


    }
}
```

(end LIA.java)

(start SimpleRead.java)

```
/*
 * @(#)SimpleRead.java    1.12 98/06/25 SMI
 *
 * Copyright (c) 1998 Sun Microsystems, Inc. All Rights Reserved.
 *
 * Sun grants you ("Licensee") a non-exclusive, royalty free, license
 * to use, modify and redistribute this software in source and binary
 * code form, provided that i) this copyright notice and license appear
 * on all copies of the software; and ii) Licensee does not utilize the
 * software in a manner which is disparaging to Sun.
 *
 * This software is provided "AS IS," without a warranty of any kind.
 * ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES,
 * INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN AND
 * ITS LICENSORS SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY
 * LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THE
 * SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS LICENSORS
 * BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT,
 * INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES,
 * HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING
 * OUT OF THE USE OF OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
 *
 * This software is not designed or intended for use in on-line control
 * of aircraft, air traffic, aircraft navigation or aircraft
 * communications; or in the design, construction, operation or
 * maintenance of any nuclear facility. Licensee represents and
 * warrants that it will not use or redistribute the Software for such
 * purposes.
 */
package commapi.samples.Simple;
import java.io.*;
import java.util.*;
import javax.comm.*;

public class SimpleRead implements Runnable, SerialPortEventListener, CommPortOwnershipListener {
    static CommPortIdentifier portId;
    static Enumeration portList;
    LIA posElectrode,negElectrode;
    InputStream inputStream;
    OutputStream outputStream;
    SerialPort serialPort;
    Thread readThread;



    public static void printBytes(byte[] b){
        for(int i= 0; i<b.length; i++){
            System.out.print(Integer.toBinaryString((b[i]&0x1FF)|0x100)+" ");

        }
        System.out.println();
    }
    public SimpleRead(int num,LIA pos, LIA neg) {
```

```
    try{
       portId = CommPortIdentifier.getPortIdentifier("COM"+num);
       portId.addPortOwnershipListener(this);
    }catch(Exception e){
       System.out.println("There is no such port");
    }
    try {
       serialPort = (SerialPort) portId.open("SimpleReadApp", 2000);
    } catch (PortInUseException e) {}
    try {
       inputStream = serialPort.getInputStream();
       outputStream = serialPort.getOutputStream();
    } catch (IOException e) {}
         try {
       serialPort.addEventListener(this);
         } catch (TooManyListenersException e) {}
    serialPort.notifyOnDataAvailable(true);
    try {
       serialPort.setSerialPortParams(115200,
          SerialPort.DATABITS_8,
          SerialPort.STOPBITS_1_5,
          SerialPort.PARITY_NONE);
    } catch (UnsupportedCommOperationException e) {}
    String s;


    posElectrode= pos;
    negElectrode= neg;
    try{
       s= "\r";outputStream.write(s.getBytes());
       System.out.println("Hit enter for PA electrodes.");
       while(System.in.available()==0)  ;
       s= "\r";outputStream.write(s.getBytes());
    }catch(Exception e){System.out.println("Couldn't write");}

    readThread = new Thread(this);
    readThread.start();
}

public void run() {
    try {
       Thread.sleep(20000);
    } catch (InterruptedException e) {}
}

public void serialEvent(SerialPortEvent event) {
    switch(event.getEventType()) {
    case SerialPortEvent.BI:
    case SerialPortEvent.OE:
    case SerialPortEvent.FE:
    case SerialPortEvent.PE:
    case SerialPortEvent.CD:
    case SerialPortEvent.CTS:
    case SerialPortEvent.DSR:
    case SerialPortEvent.RI:
    case SerialPortEvent.OUTPUT_BUFFER_EMPTY:
```

```
        break;
    case SerialPortEvent.DATA_AVAILABLE:
        byte[] readBuffer = new byte[200];
        int numBytes=0;
        try {
            while (inputStream.available() > 0) {
                numBytes = inputStream.read(readBuffer);

            }
            //displayText(readBuffer,numBytes);
            groupBytes(readBuffer,numBytes);

        } catch (IOException e) {}
        break;
    }
}
public int byteI=0;
public byte[] twoInts= new byte[5];
public void groupBytes(byte[] bytes, int byteCount){
    for(int i= 0; i<byteCount; i++){
        if(byteI==5){
            //printBytes(twoInts);

            posElectrode.getNextInt(((twoInts[1]<<8)&0xFF00)|(twoInts[2]&0xFF),twoInts[0]);
            negElectrode.getNextInt(((twoInts[3]<<8)&0xFF00)|(twoInts[4]&0xFF),twoInts[0]);
            byteI= 0;
        }
        twoInts[byteI]= bytes[i];
        byteI++;
    }
}
private String displayText(byte[] bytes, int                    byteCount)
{
    String          str;
    int  i,
            idx;
    byte[]          nb;



    nb = new byte[byteCount * 4];

    for (i = 0, idx = 0; i < byteCount; i++)
    {
            /*  Wrap any control characters      */



                nb[idx++] = bytes[i];

    }
    str="";
    try{
            byte[] newa= new byte[idx];
            System.arraycopy(nb,0,newa,0,idx);
            System.out.print("REC: ");
```

```
            printBytes(newa);
            str = new String(nb, 0, idx);

            //System.out.println("DISPLAYING: "+str);
            //printBytes(str.getBytes());
            //this.text.append(str);
        }catch(Exception e){System.out.println("OOPS");}
        return(str);
    }
    public void ownershipChange(int param) {
    }



}
```
(end SimpleRead.java)

(start SimpleRead_1.java)

```
/*
 * @(#)SimpleRead_1.java            1.12 98/06/25 SMI
 *
 * Copyright (c) 1998 Sun Microsystems, Inc. All Rights Reserved.
 *
 * Sun grants you ("Licensee") a non-exclusive, royalty free, license
 * to use, modify and redistribute this software in source and binary
 * code form, provided that i) this copyright notice and license appear
 * on all copies of the software; and ii) Licensee does not utilize the
 * software in a manner which is disparaging to Sun.
 *
 * This software is provided "AS IS," without a warranty of any kind.
 * ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES,
 * INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN AND
 * ITS LICENSORS SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY
 * LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THE
 * SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS LICENSORS
 * BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT,
 * INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES,
 * HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING
 * OUT OF THE USE OF OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
 *
 * This software is not designed or intended for use in on-line control
 * of aircraft, air traffic, aircraft navigation or aircraft
 * communications; or in the design, construction, operation or
 * maintenance of any nuclear facility. Licensee represents and
 * warrants that it will not use or redistribute the Software for such
 * purposes.
 */
package commapi.samples.Simple;
import java.io.*;
import java.util.*;
import javax.comm.*;

public class SimpleRead_1 implements Runnable, SerialPortEventListener, CommPortOwnershipListener
{
   static CommPortIdentifier portId;
   static Enumeration portList;
   static LIA posElectrode,negElectrode;
   InputStream inputStream;
   OutputStream outputStream;
   SerialPort serialPort;
   Thread readThread;

   public static void main(String[] args) {
      //portList = CommPortIdentifier.getPortIdentifiers();
      SimpleRead_1 x= new SimpleRead_1(4);


   }
   public static void printBytes(byte[] b){
      for(int i= 0; i<b.length; i++){
         System.out.print(Integer.toBinaryString((b[i]&0x1FF)|0x100)+" ");
```

```
    }
    System.out.println();
}
public SimpleRead_1(int num) {
    try{
        portId = CommPortIdentifier.getPortIdentifier("COM"+num);
        portId.addPortOwnershipListener(this);
    }catch(Exception e){
        System.out.println("There is no such port");
    }
    try {
        serialPort = (SerialPort) portId.open("SimpleReadApp", 2000);
    } catch (PortInUseException e) {}
    try {
        inputStream = serialPort.getInputStream();
        outputStream = serialPort.getOutputStream();
    } catch (IOException e) {}
          try {
        serialPort.addEventListener(this);
          } catch (TooManyListenersException e) {}
    serialPort.notifyOnDataAvailable(true);
    try {
        serialPort.setSerialPortParams(2400,
            SerialPort.DATABITS_7,
            SerialPort.STOPBITS_1,
            SerialPort.PARITY_EVEN);
    } catch (UnsupportedCommOperationException e) {}
    String sir= "SIR\r\n";
    try{outputStream.write(sir.getBytes());}catch(Exception e){}
    readThread = new Thread(this);
    readThread.start();
}

public void run() {
    try {
        Thread.sleep(20000);
    } catch (InterruptedException e) {}
}
double[] masses= new double[600];//5*120secs
int massIndex= 0;

public boolean isSteady(){
    return true;
}
public double getRateAndReset(){
    //for(int i= 1; i<massIndex; i++) System.out.print((masses[i]-masses[i-1])*4+"  ");
    //System.out.println();

    double returnD= (masses[massIndex-1]-masses[0])*4/(massIndex-1);//-1?
    massIndex= 0;
    return returnD;
}
public byte[] line= new byte[17];//the maximum size of a line xxxnnnnnnnnnxxx\n
public int lineIndex=0;
public void serialEvent(SerialPortEvent event) {
```

```
        switch(event.getEventType()) {
        case SerialPortEvent.BI:
        case SerialPortEvent.OE:
        case SerialPortEvent.FE:
        case SerialPortEvent.PE:
        case SerialPortEvent.CD:
        case SerialPortEvent.CTS:
        case SerialPortEvent.DSR:
        case SerialPortEvent.RI:
        case SerialPortEvent.OUTPUT_BUFFER_EMPTY:
           break;
        case SerialPortEvent.DATA_AVAILABLE:
           try {
              while (inputStream.available() > 0) {
                 inputStream.read(line,lineIndex,1);
                 if(line[lineIndex]=='\n'){
                    byte[] number= new byte[9];
                    for(int i= 0; i<9; i++){
                       number[i]= line[(lineIndex+17-13+i)%17];//circular
                    }
                    String s= new String(number);

                    masses[massIndex++]= Double.parseDouble(s);

                 }
                 lineIndex= (lineIndex+17+1)%17;//circular
              }
              //displayText(readBuffer,numBytes);
              //groupBytes(readBuffer,numBytes);

           } catch (Exception e) {System.out.println("Missed a measurement");}
           break;
        }
     }
     public void ownershipChange(int param) {
     }



}
```
(end SimpleRead_1.java)

(start SimpleWrite.java)

```
/*
 * @(#)SimpleWrite.java   1.12 98/06/25 SMI
 *
 * Copyright (c) 1998 Sun Microsystems, Inc. All Rights Reserved.
 *
 * Sun grants you ("Licensee") a non-exclusive, royalty free, license
 * to use, modify and redistribute this software in source and binary
 * code form, provided that i) this copyright notice and license appear
 * on all copies of the software; and ii) Licensee does not utilize the
 * software in a manner which is disparaging to Sun.
 *
 * This software is provided "AS IS," without a warranty of any kind.
 * ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES,
 * INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN AND
 * ITS LICENSORS SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY
 * LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THE
 * SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS LICENSORS
 * BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT,
 * INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES,
 * HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING
 * OUT OF THE USE OF OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
 *
 * This software is not designed or intended for use in on-line control
 * of aircraft, air traffic, aircraft navigation or aircraft
 * communications; or in the design, construction, operation or
 * maintenance of any nuclear facility. Licensee represents and
 * warrants that it will not use or redistribute the Software for such
 * purposes.
 */
package commapi.samples.Simple;
import java.io.*;
import java.util.*;
import javax.comm.*;

public class SimpleWrite {
    static Enumeration portList;
    static CommPortIdentifier portId;
    static String messageString = "\n";
    static SerialPort serialPort;
    static OutputStream outputStream;

    public static void main(String[] args) {
        portList = CommPortIdentifier.getPortIdentifiers();

        //while (portList.hasMoreElements()) {
            try{
            portId = (CommPortIdentifier) CommPortIdentifier.getPortIdentifier("COM4");
            }catch(Exception e){System.out.println("NO");}
            if (portId.getPortType() == CommPortIdentifier.PORT_SERIAL) {
                // if (portId.getName().equals("COM1")) {
                if (true){//portId.getName().equals("/dev/term/a")) {
                    try {
                        serialPort = (SerialPort)
```

```
                portId.open("SimpleWriteApp", 2000);
            } catch (PortInUseException e) {System.out.println("IN USE");}
            try {
               outputStream = serialPort.getOutputStream();
            } catch (IOException e) {}
            try {
               serialPort.setSerialPortParams(9600,
                  SerialPort.DATABITS_8,
                  SerialPort.STOPBITS_1,
                  SerialPort.PARITY_NONE);
            } catch (UnsupportedCommOperationException e) {}
            try {
               outputStream.write(messageString.getBytes());
            } catch (IOException e) {}
         }
      }
    //}
  }
}
```

(end SimpleWrite.java)

(start Weave.java)

```
/*
 * Weave.java
 *
 * Created on July 5, 2005, 4:44 PM
 */

package commapi.samples.Simple;

/**
 *
 * @author  Rob
 */
public class Weave {
    SimpleRead_1 sr1;
    SimpleRead sr;
    LIA pos,neg;
    int cycleCount=0;
    public Weave(){
        pos= new LIA("pos",this);
        neg= new LIA("neg",this);
        sr= new SimpleRead(5,pos,neg);
        sr1= new SimpleRead_1(4);

    }
    public void LIADone(LIA lia){
        if(lia.full) {System.out.println("ERROR: LIA completed twice"); System.exit(0);}
        //System.out.println(lia.name+" is full.");
        lia.amplitude= lia.calculateAmplitude();
        lia.phase= lia.calculatePhase();
        lia.full=true;
        if(pos.full&&neg.full) endCycle();
    }

    public void endCycle(){
        //check if rate is steady enough
        //pos.printIDAC();
        //if(System.in.available()>0){
            //tell SimpleRead_1 to finish, give the flow rate, and restart
            //print all three
            double phaseAdjust= 360./64;

System.out.println(cycleCount+"\t"+pos.amplitude+"\t"+neg.amplitude+"\t"+sr1.getRateAndReset()+"\t\t\t
"+(pos.phase-phaseAdjust)+"\t"+(neg.phase-360/16.-phaseAdjust));
            pos.full= false;
            neg.full= false;
            //set pos and neg to false;
            //else error
            cycleCount++;
        //}

    }
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
```

```
        // TODO code application logic here

        Weave w= new Weave();

    }

}
```
(end Weave.java)

8051 MICROCONTROLLER CODE

This code was compiled using Raisonance IDE which was supplied along with the TI MSC1202 Evaluation module. It creates an executable image that is loaded into the microcontroller over the RS232 line using TI's supplied bootloader.

(Start adc.c)

```
//*********************************************************************
// File name: adc.c
//
// Copyright 2003 Texas Instruments Inc as an  unpublished work.
//
// Version 1.0
//
// Compiler Version (Keil V2.38), (Raisonance V6.10.13)
//
// Module Description:
//  ADC Example Program
//
//*********************************************************************
//#include "legal.c"         //Texas Instruments, Inc. copyright and liability
#include <reg1200.h>       // The header file with the MSC register definitions
#include <stdio.h>         // Standard I/O so we can use the printf function
#include <math.h>
extern signed long bipolar(void);
extern unsigned long unipolar(void);
extern int getSpi();

#define autobaud()                                  ((void (code *) (void)) 0xFBFA) ();// MSC1200
#define sendByte(BYTE)                              ((void (code *) (char)) 0xFBEA) (BYTE);    //
MSC1200

#define LSB (3.8147e-5)
#define SET 1
#define CLEAR 0

sbit RedLed = P3^4;
sbit YellowLed= P3^5;

int ticks= 0;
unsigned int voltage;

sbit SCLK=P3^6;
sbit SDA=P1^2;
sbit SSN=P1^1;
sbit ADC_CS = P1^0;
long tCosSum,tSinSum;
/*void liad(int result){
                        //tCosSum+= (result-(65535/2))*(cos(ticks/64.*2*3.1416)));
                        tSinSum+= (result-(65535/2));//*(sin(ticks/64.*2*3.1416)));
                       if(ticks==0){
                        tSinSum= sqrt(tCosSum*tCosSum+tSinSum*tSinSum);
                        tCosSum= 0; tSinSum=0;
                       }
}*/
```

```
void main(void) {
            int adc2; //the better one
            //IDAC= 0x00;
            char is=0;
   RedLed = !RedLed;
            CKCON = 0x10;// MSC1200 Timer1 div 4
            TCON = 0;                   // MSC1200 Stop TR1
   autobaud();
   /*while(1){
           scanf("%c",&is);
           //printf("ABCDE");
           sendByte(0x55);
           sendByte(0xf0);
           //sendByte(0x55); sendByte(0x0f);

           }*/
   //printf("ADC Test, ACLK2\n");
   //printf("ADC Test, ACLK3\n");
   //printf("ADC Test, ACLK4\n");
   //printf("ADC Test, ACLK5\n");
   //while(1);

   PDCON = 0x75;          // Turn on the A/D
   PDCON&= 0xBF;
   SYSCLK= 0x00;
   ACLK =3;//3             // ACLK freq. = XTAL Freq./(ACLK +1) = 0.9216 MHz
                  // 0.9216 Mhz/64 = 14,400 Hz
   DECIMATION = 150;//150     // Dtrata Rate = 14,400/1,440 = 10 Hz
   ADMUX = 0x10;          // AINP = AIN7, AINN = AIN6
   ADCON0 = 0x30;         // Vref On, 2.5V, Buffer Off, PGA=1
   ADCON1 = 0x51;         // unipolar, auto, self calibration, offset, gain

   //TCON= 0x50;//6 is run 1, 4 is run 0
   //CKCON|= 0x18;//4  is 1 div by (0-12,1-4),3 is 0 div by...
   //TMOD= (TMOD&0xF0)|0x01;//0x1 for correct mode


   //#define time 65536-x

   ADMUX= 0x20;
   //IDAC= 0x00;
   YellowLed= !YellowLed;
   //for(is= 0; is<128; is++){
   //while(1){
   //int i= 0;
   //printf("ENTER DECIMATION\n"); scanf("%i\n",&adc2);
   //DECIMATION= adc2;
   //printf("ENTER ACLK\n"); scanf("%i\n",&adc2);
   //ACLK= adc2&0xFF;
   //printf("ACLK %i,DEC %i\n",ACLK,DECIMATION);

           //for(i= 0; i<50; i++){
           while(1){
             char tock=ticks%4;

             while(!(AIPOL&0x20))  ;
```

```
        //voltage= bipolar();
        IDAC= ((char)(127*sin(ticks/(128.)*2*3.1416)))+127;

        voltage= unipolar();
        //while(!TF0);
        //TH0= 256-(256-179)/2-2;//(65536-5000)>>8;
        //TL0= 0x00;//(65536-5000)&0xFF;

        //TF0=0;
        ticks++;
   //while(ticks>127) ;
   ticks= ticks&(128-1);

   //sendByte(is&0xFF);
   if(ticks%128==1) {
                //printf("%f\n",voltage*LSB);
   }
   if (tock==0){
       ADMUX= 0x10;//
       sendByte(ticks&0xFF);
       sendByte((voltage>>8)&0xFF);
       sendByte(voltage&0xFF);

   }
   else if(tock==2){
                ADMUX= 0x20;
                sendByte((voltage>>8)&0xFF);
        sendByte(voltage&0xFF);

        }









}
/*while(1){
   while(!(AIPOL&0x20))  ;

   voltage= LSB*bipolar();//unipolar();
   //adc2= getSpi();
   if(ticks%4==0){
    ADMUX= 0x10;
   }
   if(ticks%4==2){
                ADMUX= 0x23;
        }


   if(ticks==2);//; //ticks++;
      //printf("%f\n",adc2*2.5/65536);
      //printf("%f\n",voltage);
```

```
    IDAC= ((char)(127*sin(ticks/(2*256.)*2*3.1416)))+127;
    //printf("V=%li\n",sendSpi(1028));
    ticks= ticks%(2*256);
    //nop
    //nop
    //nop
    //IDAC+= 0x80;
    ticks++;
  }*/



        /*while(1){
          TL0= (0xA4); TH0=(0x98); TF0= 0;
          //printf("!");
           i++; j=i;
           while(!(AIPOL&0x20))  ;
           j=j%2;
    //if(j==0){ ADMUX=0x23;}
    //else if(j==1){
    ADMUX =0x10;//}

    result=unipolar();     // Save Results
    voltage= result*LSB;

    if(i%65==0){

    //if(j==1)printf("23-1\t");
    //if(j==1)printf ("%f\t",voltage);
    //if(j==3)printf("12-1\t");
    //if(j==0)
    printf ("%f\n",voltage);

    }
    IDAC= ~IDAC;
        //IDAC= ((char)(127*sin(j/128.*2*3.1416)))+127;
          //while(!(AIPOL&0x20))  ;
          //result=unipolar();     // Save Results

          while(!(TF0));
        }/*
        j=1;
  while(1){
     // Waiting for conversion
    if((j%4)==0||(j%4)==1) ADMUX=0x23;
    else ADMUX =0x10;
    IDAC= ~IDAC;
    result=unipolar();     // Save Results
    lastResult= result;
    j++;

  }
  //long getSysTime(){
  //    long time*/
  while(1);
}
```

(end adc.c)

(start utilities.a51)

```
;*------------------------------------------------------------------*
; File name: utilities.a51
;
; Copyright 2003 Texas Instruments Inc as an unpublished work.
; All Rights Reserved.
;
; Revision History
;           Version 1.0
;
; Assembler Version  (Keil V2.38), (Raisonance V6.10.13)
;
; Module Description:
; ADC routines to read 24-bit ADC and return the value as a long integer.

;*------------------------------------------------------------------*
;$include (legal.a51) ; Texas Instruments, Inc. copyright and liability
$include (reg1200.inc)

;*------------------------------------------------------------------*
PUBLICunipolar, bipolar

adc_sub SEGMENT  CODE
          RSEG  adc_sub

;.....................
;;;;;;;;;;;;;;;;;;;;;
; unsigned long unipolar(void)
; return the 3 byte adres to R4567 (MSB~LSB)
; unsigned long int with R4=0
unipolar:
          mov     r4,#0
          mov     r5,adresh
          mov     r6,adresm
          mov     r7,adresl
          ret

;.....................
;;;;;;;;;;;;;;;;;;;;;
; signed long bipolar(void)
; return the 3 byte adres to R4567 (MSB~LSB)
; return signed long int with sign extendsion on R4
bipolar:
          mov     r4,#0
          mov     a,adresh
          mov     r5,a
          mov     r6,adresm
          mov     r7,adresl
          jnb     acc.7,positive
          mov     r4,#0ffh
positive:
          ret

;.....................
;;;;;;;;;;;;;;;;;;;;;
; signed long read_sum_regs(void)
; return the 4 byte sumr to R4567 (MSB~LSB)
; return signed long int, sign extension done by hardware
read_sum_regs:
```

```
mov       r4, SUMR3;
mov       r5, SUMR2;
mov       r6, SUMR1;
mov r7, SUMR0;
ret

end
```

```
$NOMOD51
$include (REG1200.INC)

;PUBLIC _spim_send_recv_byte
;.org 0x00
PUBLIC getSpi

spim_routines      SEGMENT  CODE
           RSEG    spim_routines
delay equ 32d

waitShort macro insts;( ;must be 6 atleast)
   add A,#insts-4d
   rrc  A
   mov r0,A
   jnc $+2
   nop
   djnz r0,$
endm
   ;mov r,
off macro
anl P3,#10111111b
endm
on macro
orl P3,#01000000b
endm

wait macro insts
   mov A,#((insts)-9d)
   call waitsub
endm

toggle macro      ;starts high
            off
            add A,#0
   wait delay
   on
endm
waitsub:;the number in A+1(so put the number in a and call)
   ;;MUST BE AT LEAST 12
   rrc  A
   mov r0,A
   jnc $+2
   nop
   djnz r0,$
   ret
getSpi:
           orl P1DDRL,#11000000b;set P1.3 as input
           anl P1,#11111110b ; CS to low
           mov r7,#25d
           mov r7,#0x00
           mov r6,#0x00
           mov r1,#22d
           mov r2,#02
lp:
           toggle
```

```
            mov A,r7
            add A,#0
            rlc A   ;;carry?
            mov r7,A
            mov A,r6
            rlc A
            mov r6,A
            mov A,P1;get result
            anl A,#00001000b
            add A,#11111000b
            mov A,#0
            addc A,r7
            mov r7,A
            wait delay-15d;+1d;1 for clearing carry
            djnz r1,lp
            djnz r2,store_result
            orl P1,#01b ;CS to high
            mov A,r4
            mov r7,A
            mov A,r5
            mov r6,A
            ;jmp getSpi
            ret
store_result:
            mov A,r7
            mov r4,A
            mov A,r6
            mov r5,A
            mov r1,#10d
            jmp lp

;garbage
;           call spiSub;
;           mov r2,r6
;           mov r3,r7
;           call spiSub
;           mov A,r3
;           add r7,A
;           mov A,r2
;           addc r6,A
;           add A,#0
;           mov A,r6
;           rr A
;           mov r6,A
;           mov A,r7
;           rrc A
;           mov r7,A
;           ret

;DSEG
;   bit: 1

end
```
(end utilities.a51)

(start reg1200.inc)

```
;*****************************************************************
; File name: reg1200.inc
;
; Copyright 2004 Texas Instruments Inc as an  unpublished work.
; Created By:  Ritu Ghosh - Russell Anderson
;
; Version 1.0 Initial Version 12/03/2003
;
; Compiler Version (Keil V2.38), (Raisonance V6.10.14)
;
; Module Description:
; Header file for TI MSC1200 microcontroller
;
;*****************************************************************

$NOMOD51
$SAVE
$NOLIST
;  BYTE Registers


SP      DATA    081H ;STANDARD 8051
DPL     DATA    082H ;STANDARD 8051
DPL0    DATA    082H ;STANDARD 8051
DPH     DATA    083H ;STANDARD 8051
DPH0    DATA    083H ;STANDARD 8051
DPL1    DATA    084H
DPH1    DATA    085H
DPS     DATA    086H
PCON    DATA    087H ;STANDARD 8051
TCON    DATA    088H ;STANDARD 8051
TMOD    DATA    089H ;STANDARD 8051
TL0     DATA    08AH ;STANDARD 8051
TL1     DATA    08BH ;STANDARD 8051
TH0     DATA    08CH ;STANDARD 8051
TH1     DATA    08DH ;STANDARD 8051
CKCON   DATA    08EH
MWS     DATA    08FH
P1      DATA    090H ;STANDARD 8051
EXIF    DATA    091H
CADDR   DATA    093H
CDATA   DATA    094H
SCON    DATA    098H ;STANDARD 8051
SCON0   DATA    098H ;STANDARD 8051
SBUF    DATA    099H ;STANDARD 8051
SBUF0   DATA    099H ;STANDARD 8051
SPICON DATA 09AH
I2CCON  DATA    09AH
SPIDATA DATA 09BH
I2CDATA DATA    09BH
AIPOL   DATA    0A4H
PAI DATA 0A5H
AIE DATA 0A6H
AISTAT DATA 0A7H
IE      DATA    0A8H ;STANDARD 8051
```

```
P1DDRL DATA 0AEH
P1DDRH DATA 0AFH
P3     DATA   0B0H ;STANDARD 8051
P3DDRL DATA 0B3H
P3DDRH DATA 0B4H
IDAC   DATA   0B5H
IP     DATA   0B8H ;STANDARD 8051
EWU DATA 0C6H
SYSCLK DATA   0C7H
PSW    DATA   0D0H ;STANDARD 8051
OCL DATA 0D1H
OCM DATA 0D2H
OCH DATA 0D3H
GCL DATA 0D4H
GCM DATA 0D5H
GCH DATA 0D6H
ADMUX DATA 0D7H
EICON  DATA   0D8H
ADRESL DATA 0D9H
ADRESM DATA 0DAH
ADRESH DATA 0DBH
ADCON0 DATA 0DCH
ADCON1 DATA 0DDH
ADCON2 DATA 0DEH
ADCON3 DATA 0DFH
ACC    DATA   0E0H ;STANDARD 8051
SSCON  DATA 0E1H
SUMR0 DATA 0E2H
SUMR1 DATA 0E3H
SUMR2 DATA 0E4H
SUMR3 DATA 0E5H
ODAC DATA 0E6H
LVDCON DATA 0E7H
EIE    DATA   0E8H
HWPCO DATA 0E9H
HWPC1 DATA 0EAH
HWVER DATA 0EBH
FMCON DATA 0EEH
FTCON DATA 0EFH
B      DATA   0F0H ;STANDARD 8051
PDCON  DATA   0F1H
PASEL  DATA   0F2H
PLLL   DATA   0F4H
PLLH   DATA   0F5H
ACLK   DATA   0F6H
SRST   DATA   0F7H
EIP    DATA   0F8H
SECINT DATA 0F9H
MSINT DATA 0FAH
USEC DATA 0FBH
MSECL DATA 0FCH
MSECH DATA 0FDH
HMSEC  DATA   0FEH
WDTCON DATA   0FFH
```

```
;  BIT Registers
; *** TCON ***
TF1     BIT     08FH
TR1     BIT     08EH
TF0     BIT     08DH
TR0     BIT     08CH
IE1     BIT     08BH
IT1     BIT     08AH
IE0     BIT     089H
IT0     BIT     088H

; *** P1 ***
INT5    BIT     097H
INT4    BIT     096H
INT3    BIT     095H
INT2    BIT     094H
DIN     BIT     093H
DOUT    BIT     092H

; *** SCON0 ***
SM0_0   BIT     09FH
SM0     BIT     09FH
SM1     BIT     09EH
SM1_0   BIT     09EH
SM2     BIT     09DH
SM2_0   BIT     09DH
REN     BIT     09CH
REN_0   BIT     09CH
TB8     BIT     09BH
TB8_0   BIT     09BH
RB8     BIT     09AH
RB8_0   BIT     09AH
TI      BIT     099H
TI_0    BIT     099H
RI      BIT     098H
RI_0    BIT     098H

; *** IE ***
EA      BIT     0AFH
ES      BIT     0ACH
ES0     BIT     0ACH
ET1     BIT     0ABH
EX1     BIT     0AAH
ET0     BIT     0A9H
EX0     BIT     0A8H

; *** P3 ***
CLKS    BIT     0B6H
T1      BIT     0B5H
T0      BIT     0B4H
INT1    BIT     0B3H
INT0    BIT     0B2H
TXD     BIT     0B1H
TXD0    BIT     0B1H
RXD     BIT     0B0H
RXD0    BIT     0B0H
```

```
; *** IP ***
PS      BIT     0BCH
PS0     BIT     0BCH
PT1     BIT     0BBH
PX1     BIT     0BAH
PT0     BIT     0B9H
PX0     BIT     0B8H

; *** PSW ***
CY      BIT     0D7H
AC      BIT     0D6H
F0      BIT     0D5H
RS1     BIT     0D4H
RS0     BIT     0D3H
OV      BIT     0D2H
F1      BIT     0D1H
P       BIT     0D0H

; *** EICON ***
;SMOD1   BIT     0DFH
EAI     BIT     0DDH
AI      BIT     0DCH
WDTI    BIT     0DBH

; *** EIE ***
EWDI    BIT     0ECH
EX5     BIT     0EBH
EX4     BIT     0EAH
EX3     BIT     0E9H
EX2     BIT     0E8H

; *** EIP ***
PWDI    BIT     0FCH
PX5     BIT     0FBH
PX4     BIT     0FAH
PX3     BIT     0F9H
PX2     BIT     0F8H

; *** Reg ***
Reg0 Data 00H
Reg1 Data 01H
Reg2 Data 02H
Reg3 Data 03H
Reg4 Data 04H
Reg5 Data 05H
Reg6 Data 06H
Reg7 Data 07H
RegB Data 0F0H
$RESTORE
```
(end reg1200.inc)

(start rom.a51)

```
PUBLIC_put_string                ; void put_string(char code *string);
PUBLIC_page_erase                ; char page_erase (int faddr, char fdata, char fdm)
PUBLICwrite_flash                ; char write_flash (int faddr, char fdata, char fdm)
PUBLIC_write_flash_chk ; char write_flash_chk (int faddr, char fdata, char fdm)
PUBLIC_write_flash_byte          ; char write_flash_byte (int faddr, char fdata, char fdm)
PUBLIC_faddr_data_read ; char faddr_data_read(char);
PUBLIC_data_x_c_read             ; char data_x_c_read(int addr);
PUBLIC_tx_byte         ; void tx_byte(char);
PUBLIC_tx_hex                    ; void tx_hex(char);
PUBLIC  putok                    ; void putok(void);
PUBLICrx_byte                    ; char rx_byte(void);
PUBLICrx_byte_echo               ; char rx_byte_echo(void);
PUBLICrx_hex_echo                ; char rx_hex_echo(void);
PUBLICrx_hex_double_echo         ; char rx_double_echo(void);
PUBLICrx_hex_word_echo           ; char rx_word_echo(void);
PUBLIC  autobaud                 ; void autobaud(void);
PUBLICputspace4                  ; void putspace4(void)
PUBLICputspace3                  ; void putspace3(void)
PUBLICputspace2                  ; void putspace2(void)
PUBLICputspace1                  ; void putspace1(void)
PUBLIC  putcr                    ; void putcr(void);


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; Interface to C compatible Boot ROM Build In Routines
;
        CSEG AT 0FFD5H
_put_string:                     ; void put_string(char *string);

        CSEG AT         0FFD7H
_page_erase:                     ; char page_erase (int faddr, char fdata, char fdm)

        CSEG AT 0FFD9H
write_flash:                     ; DPTR = address, acc = data (Not callable by C programs)

        CSEG AT 0FFDBH
_write_flash_chk:                ; char write_flash_chk (int faddr, char fdata, char fdm)

        CSEG AT 0FFDDH
_write_flash_byte:               ; char write_flash_byte (int faddr, char fdata, char fdm)

        CSEG AT 0FFDFH
_faddr_data_read:                ; char faddr_data_read(char);

        CSEG AT 0FFE1H
_data_x_c_read:                  ; char data_x_c_read(int addr);

        CSEG AT 0FFE3H
_tx_byte:                        ; void tx_byte(char);

        CSEG AT 0FFE5H
_tx_hex:               ; void tx_hex(char);

        CSEG AT 0FFE7H
```

```
putok:                          ; void putok(void);

        CSEG AT 0FFE9H
rx_byte:                ; char rx_byte(void);

        CSEG AT 0FFEBH
rx_byte_echo:                   ; char rx_byte_echo(void);

        CSEG AT 0FFEDH
rx_hex_echo:                    ; char rx_hex_echo(void);

        CSEG AT 0FFEFH
rx_hex_double_echo:                     ; char rx_double_echo(void);

        CSEG AT 0FFF1H
rx_hex_word_echo:                       ; char rx_word_echo(void);

        CSEG AT 0FFF3H
autobaud:                       ; void autobaud(void);

        CSEG AT 0FFF5H
putspace4:                      ; void putspace4(void)

        CSEG AT 0FFF7H
putspace3:                      ; void putspace3(void)

        CSEG AT 0FFF9H
putspace2:                      ; void putspace2(void)

        CSEG AT 0FFFBH
putspace1:                      ; void putspace1(void)

        CSEG AT 0FFFDH
putcr:                          ; void putcr(void);

        END
```
(end rom.a51)