UNIVERSITY OF NAIROBI

FACULTY OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND INFORMATION ENGINEERING

DESIGN OF A PC DATA ACQUISITION SYSTEM

PROJECT INDEX:PRJ 33.

BY

WAFULA MICHAEL JUMA

F17/2386/2009

SUPERVISOR: MR.COLLINS OMBURA

EXAMINER: DR. H .OUMA.

Project report submitted in partial fulfillment of the requirement for the award of the degree of Bachelor of Science in Electrical & Electronic Engineering of the University of Nairobi.

Submitted on :28th April 2014.

DECLARATION OF ORIGINALITY

DEDICATION

I dedicate this project to my mother. To me she is my rock.She is the strongest woman I have ever known. Her support, encouragement and constant love have sustained me throughout my life.

ACKNOWLEDGEMENTS

First and foremost, I wish to thank the Almighty God for the strength throughout my studies.

This Project would not have been possible without the support of many people.I would like to express my sincere gratitude to my supervisor Mr.Collins Ombura who was abundantly helpful and who offered invaluable assistance,support and guidance.

A special thanks goes to my family.Words can not express how grateful I am for the sacrifices they have made on my behalf.Your prayers for me is what sustained me this far.My appreciation also goes to my uncle Joseph and cousin Wanjala for their support.

I would also like to thank my classmates such as Elphas Kiptoo,Sally Musonye,Keter,Mark,Erick,Danson,Paul,Mureithi and all my friends for their advice and encouragement.

To all those that i have not been able to give a particular mention here i do express my appreciation to you too.

DECLARATION AND CERTIFICATION

This is my original work and has not been presented for a degree award in this or any other university.

.....

WAFULA MICHAEL JUMA

F17/2386/2009

This report has been submitted to the Department of Electrical and Electronic Engineering, The University of Nairobi with my approval as supervisor:

…….....

MR. C .OMBURA

Date.....

CONTENTS

LIST OF FIGURES

ABBREVIATIONS

PC	Personal Computer
RTD	Resistance Temperature Detector
USB	Universal Serial Bus
EIA-RS	Electronic Industry Association Recommended Standard.
ADC	Analogue to Digital Converter
DAC	Digital to Analogue Converter
DSP	Digital Signal Processor
IC	Integrated Circuit
MFC	Microsoft Foundation Classes

ABSTRACT

This report presents the design and implementation of a PC based data acquisition system.

The aim of the project is to design a system that takes four analogue inputs and display them in real time on a program running on a PC.

The designed system is analyzed where the displayed results are compared with the input analogue signals. The system can find applications in industrial processes such as monitoring and automation.

Indexing terms: Analogue to digital conversion, serial communication, Microcontroller, Sensors.

CHAPTER ONE: INTRODUCTION

1.1 General Background

PC data acquisition systems have evolved over a period of time. The measurement and control are automated by exchanging data between the instruments and computers. There are several ways of exchanging measurement and control data between computer and instruments for example EIA-RS-232 and USB. In general the design of a PC-based measuring system involves the following two steps:

1. Design of a data acquisition hardware, and
2. Design of appropriate application software.

1.2 Problem Statement

Data acquisition is widely used in many applications.The problem hence is to design and implement a system that is capable of data acquisition e.g ambient temperature and display it on a computer.

1.3 Project Justification

Recent technological advancements have led to automation in many industrial processes,military and civilian applications.All these processes are controlled based upon certain physical quantities such as temperature hence the need for an accurate and cheaper data acquisition system such as the PC data acquisition system.

1.4 Objectives

The project aims to attain the following objectives:

1.Design an interface to read four analogue inputs.
2.Transmit the acquired data and display it on a program written on a PC.

1.5 Scope of Project

The project will be limited to data acquisition and display only.It will not involve the production of the hardware e.g microcontrollers used in the project

CHAPTER TWO: LITERATURE REVIEW

The block diagram of the PC based data acquisition system is shown below

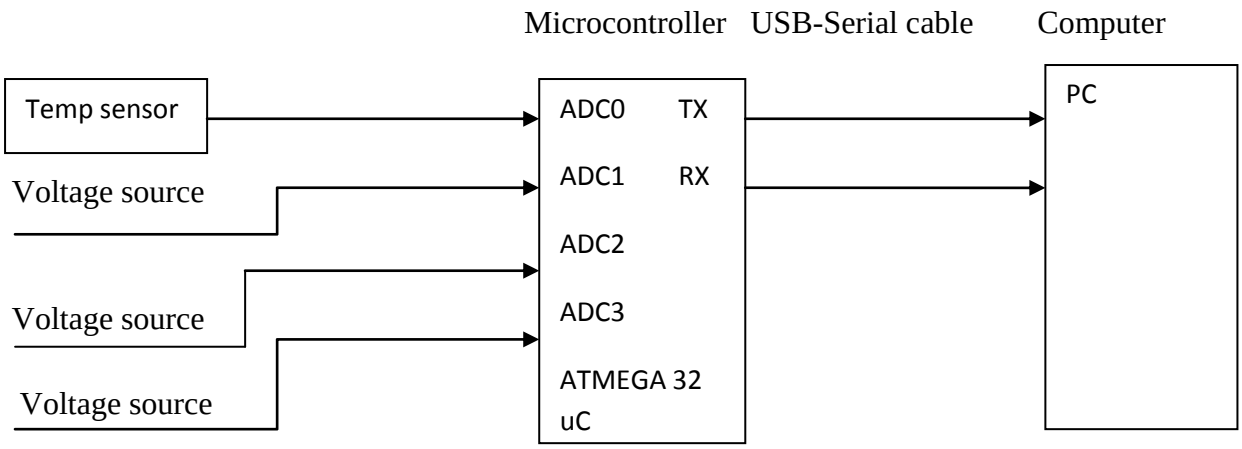


Figure 2.1 PC data acquisition block diagram

It consists of the following :

- Sensors for measuring the input signals.
- Three analogue voltage sources
- Atmega32 microcontroller that performs analogue to digital conversion and also does serial transmission.
- A USB to Serial adaptor cable
- A PC with a display program

2.1 SENSORS

A sensor is a device that measures a physical quantity and converts it into a signal which can be read by an observer or by an electronic instrument like a computer.The output of a sensor could be an electrical or an optical signal.

A sensor's sensitivity is defined as the ratio of the output signal to the measured qunantity.It indicates how much the sensor's output changes when the measured quantity changes for example if mercury in a thermometer moves by 1cm when the temperature rises by one degree then the sensitivity is 1cm/^{o}C.Sensors that measure very small quantities must be very sensitive.

Characteristics of a good Sensor include:

- It is sensitive to the measured quantity only.
- Is insensitive to any other quantity likely to be encountered in its application.
- Does not influence the measured property.
- Have a wide range of values it can read.
- It should be easy to interface

Ideal sensors are linear or linear to some mathematical functions of the measured quantity for example the logarithmic function.The output of a sensor is analogue and if it has to be processed then it has to be converted to a digital form using an ADC.

2.1.1. Temperature Sensor

This is a device that gives a voltage output that varies with temperature.There are a wide variety of temperature sensors used namely:

- ❖ Thermocouples
- ❖ Resistance Temperature Detectors(RTD)
- ❖ Thermistors
- ❖ Infrared and
- ❖ Semiconductor sensors

In this paper RTDs and Semiconductor sensors are discussed.

2.1.1.1 Resistance Temperature Detector

The RTD is a temperature sensing device whose resistance changes with temperature. Typically built from platinum. To measure the resistance
Across an RTD, apply a constant current, measure the resulting voltage, and determine the RTD resistance. The resistance at temperature‘t’ R_t is given by the equation below.

$$R_t = R_0[1+ \alpha (t-t_0)] \quad (2.1)$$

Where:
R_t = resistance at temperature 't'

R_0 = resistance at a reference temperature (Generally 0 degree C)

α = temperature coefficient of resistance ($°C^{-1}$)

$$V_t=IR_t =IR_0[1+ \alpha (t-t_0)] . \ [1] \quad (2.2)$$

From the temperature t can be found as follows

$$t = \frac{Vt-IR0}{IR0\alpha} + t_0 \quad (2.3)$$

RTDs requires external current excitation, as well as signal conditioning to account for Lead wire effects and self-heating. An example of RTD is ADT70, which provides both excitation and signal conditioning for platinum RTD. The output of this device 5 mV/°C [2]. The graph of resistance against temperature for the RTD is shown below

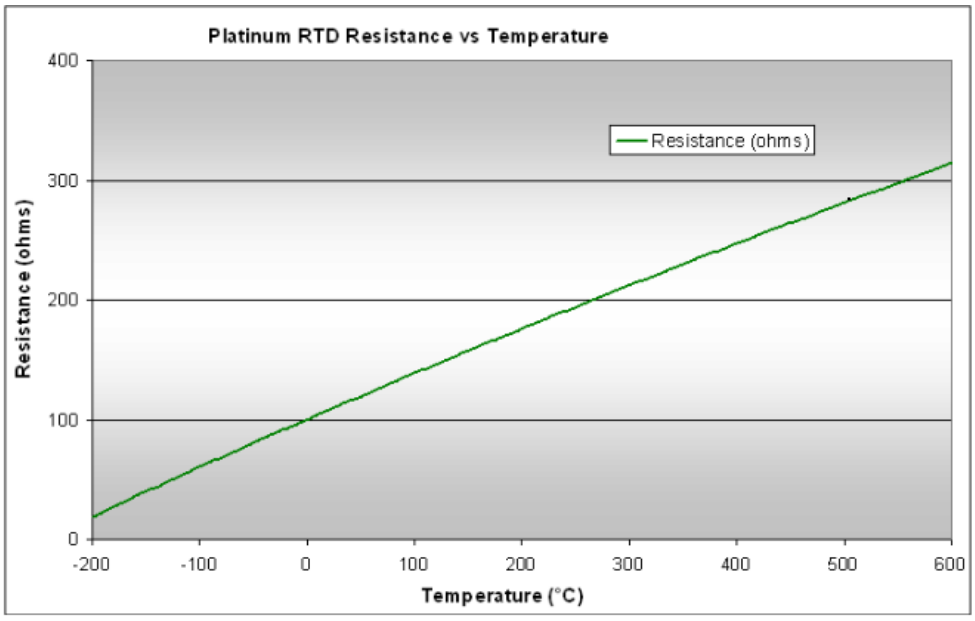


Figure 2.2 Graph of resistance against temperature of platinum RTD

2.1.1.2 Semiconductor Temperature Sensor

These temperature sensors employ the principle that a bipolar junction transistor's (BJT) base emitter voltage to collector current varies with temperature:[2].

$$V_{be} = \frac{kT}{q} \ln\left(\frac{I_C}{I_S}\right) \qquad (2.4)$$

The temperature can be calculated from a measured V_{be} almost without regard to the initial forward voltage, physical size of the junction, leakage, or other junction characteristics.
They provide an output voltage that is linearly proportional to the centigrade temperature and do not require external calibration.[3]
The excessive leakage currents characteristic of silicon PN junctions limits the temperature range for IC-based sensors to about 200°C
These currents double with every 10°C rise in temperature, causing malfunctions in band gap references and signal-conditioning circuitry.
Two main types of IC temperature sensors:

- Analog

This produces a voltage or current proportional to temperature

- Digital

It includes an integrated A/D converter and produces a digital output.
These sensors generates a higher output voltage than RTDs and may not require that the output voltage be amplified.
Semiconductor temperature sensors have the following advantages

- Small and simple

- ❖ Accurate
- ❖ Inexpensive
- ❖ No linearization or cold-junction compensation is required.
- ❖ External or internal hot spots can be monitored.
- ❖ Generally provide better noise immunity through higher-level output signals.
- ❖ Easy to interface with other devices such as amplifiers, regulators, DSPs, and micro-controllers.

A limitation of IC temperature sensors is that the temperature range is within a –55° to 150°C range.

An example of semiconductor temperature sensor is the LM 35 with a temperature range of -550mV at -55^0C to 1500mV at 150^0C

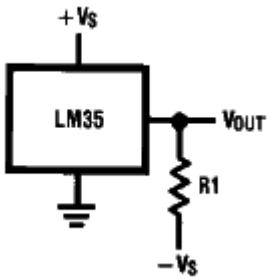


Figure 2.3. LM 35 Temperature sensor

2.2 ANALOGUE TO DIGITAL CONVERSION

Most modern electronic circuits are designed to process digital signals. However most real world signal sources are analogue.This necessitates the need for analogue to digital conversion.

An analog-to-digital convertor (ADC) is an electronic circuit whose digital output is proportional to its analog input; effectively it measures the input voltage, and gives a binary output number proportional to its size.

Advantages of processing digital signals include:

- The characteristics of a digital signal processor do not drift with time or temperature.
- Digital signal processors can be reprogrammed to modify their operation without changing the hardware.
- Several forms of data e.g. speech, images are represented similarly hence they can share transmission, and processing resources.

Several important parameters of ADCs that determine their performance include:

- Reference voltage
- Resolution
- Sampling frequency

2.2.1.Reference Voltage

This is the maximum analogue value that the ADC can convert. For example n bit ADC can convert values from zero to the reference voltage. This voltage range is divided into 2^n steps, where n is the digital word size.

2.2.2 Resolution

Resolution is the smallest analogue change that results when the digital word changes by one bit. For an n bit ADC the resolution is defined as:

$$\text{Resolution} = \frac{V_{ref}}{2^n} \qquad (2.5)$$

From equation (2.5) more accurate conversion is obtained when n is large and the resolution is smaller.

2.2.3 Sampling Frequency

When converting an analog signal to digital, we repeatedly take a sample and quantize this to the accuracy defined by the resolution of our ADC.

The sample frequency needs to be chosen with respect to the rate of which the sampled data is changing. If the sample frequency is too low then rapid changes in the analog signal may not be obvious in the resulting digital data.

The Nyquist sampling criterion states that the sampling frequency must be at least double that of the highest frequency of sampled data i.e. if the sampled data is band limited to B_t Hz then the sampling frequency F_s is:

$$F_s = 2B_t \qquad (2.6)$$

This guarantees that the original signal can be reconstructed from the knowledge of the samples.

If x(t) is the continuous signal then the sampled signal is x(kT) represented as

$$x(kT) = \sum_{k=0}^{k=\infty} \delta(t - kT)x(t) \quad (2.7)$$

2.2.4 Types of Analogue to Digital Converters

There are several types of A/D converters in common use in instruments today, described here as integrating (Dual slop), parallel (Flash), and Successive Approximation ADCs.

2.2.4.1 Integrating (Dual slop)

In this technique the time needed to charge or discharge a capacitor is used in order to determine the input voltage.

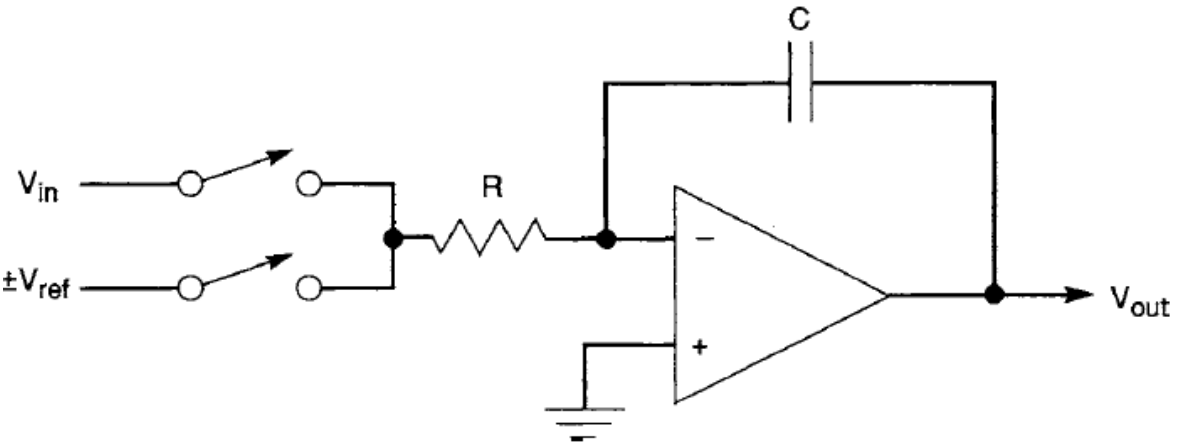


Figure 2.4 Dual slop ADC

There are two half cycles, referred to here as the up slope and the down slope. The input signal is integrated during the up slope for a fixed time (Fig.2.13). Then a reference of opposite sign is integrated during the down slope to return the integrator output to zero. The time required for the down slope is proportional to the value of the input and is the output of the ADC.[7]
The up slope cycle can be described mathematically as follows:

$$V_p = -\frac{T_{up}V_{in}}{RC} \quad (2.8)$$

Where *Vp* is the peak value reached at the integrator output during the up slope, Tup is the known up slope integration time, Vin is the input signal, and *R* and C are the integrator component values.The down slope can be similarly described by

$$Vp = \frac{T_{dn}V_{ref}}{RC} \quad (2.9)$$

Where T_{dn} is the unknown time for the down slope, and *V*ref is the known reference. Equating 2.7 and 2.8 and solving for *T*dn, the output of the ADC:

$$T_{dn} = -\frac{T_{up} V_{in}}{V_{ref}} \quad (2.10)$$

*V*in and *V*ref will always be of opposite sign (to assure a return to zero in the integrator), so that *T*dn will always be positive. Values of *R* and *C* do not appear in *T*dn, so that their values are not critical.[7] This is a result of the same components having been used for both the up and down slopes. Similarly, if the times *T*up and *T*dn are defined by counting periods of a single clock, the exact period of that clock will not affect the accuracy of the ADC. Restating the output in terms of the number of periods of the clock:

$$N_{dn} = -\frac{N_{up} V_{in}}{V_{ref}} \quad (2.11)$$

where N_{up} is the fixed number of clock periods used in the up slope and N_{dn} is the number of clock periods required to return the integrator output to zero.

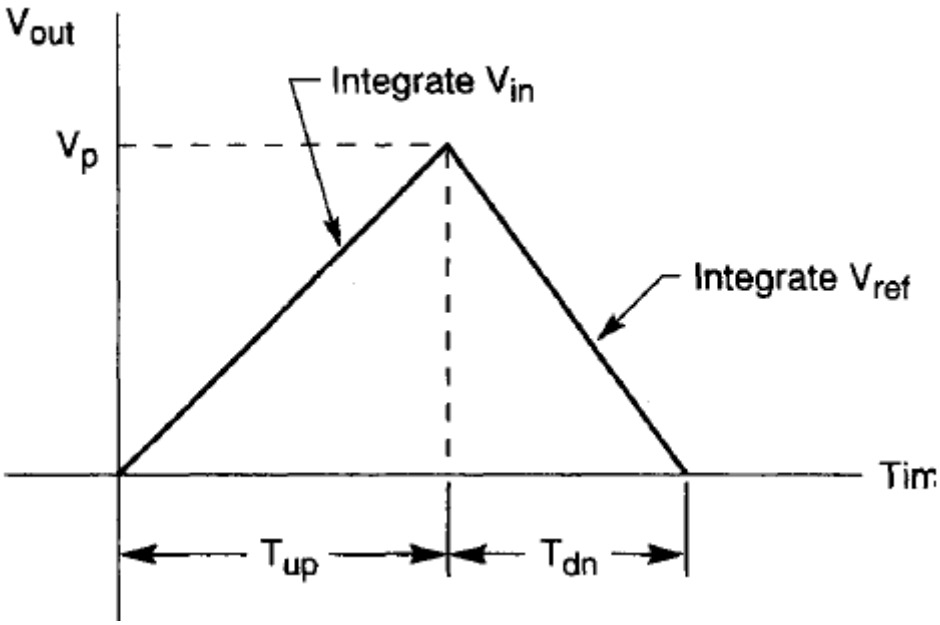


Figure 2.5 Dual slope waveform

Although 20-bit accuracy is common, it has a relatively slow conversion rate, such as 60 Hz maximum.

2.2.4.2 Successive-Approximation ADCs

A successive-approximation converter is composed of a digital-to-analog converter (DAC), a single comparator, and some control logic and registers, When the analog voltage to be measured is present at the input to the comparator, the system control logic initially sets all bits to zero. Then the DAC's most significant bit (MSB) is set to 1, which forces the DAC output to 1/2 of full scale.The comparator then compares the analog output of the DAC to the input signal, and if the DAC output is lower than the input signal, (the signal is greater than 1/2 full scale), the MSB remains set at 1. If the DAC output is higher than the input signal, the MSB resets to zero.[8]

Next, the second MSB with a weight of 1/4 of full scale turns on (sets to 1) and forces the output of the DAC to either 3/4 full scale (if the MSB remained at 1) or 1/4 full scale (if the MSB was reset to zero). The comparator once more compares the DAC output to the input signal and the second bit either remains on (sets to 1) if the DAC output is lower than the input signal or resets to zero if the DAC output is higher than the input signal. The third MSB is then compared the same way and the process continues in order of descending bit weight until the LSB is compared.

At the end of the process, the output register contains the digital code representing the analog input signal. Successive approximation ADCs are relatively slow
because the comparisons run serially, and the ADC must pause at each step to set the DAC and wait for its output to settle. However, conversion rates easily can reach over 1 MHz [8]. Also, 12 and 16-bit successive-approximation ADCs are relatively inexpensive, which accounts for their wide use in many PC-based data acquisition systems.

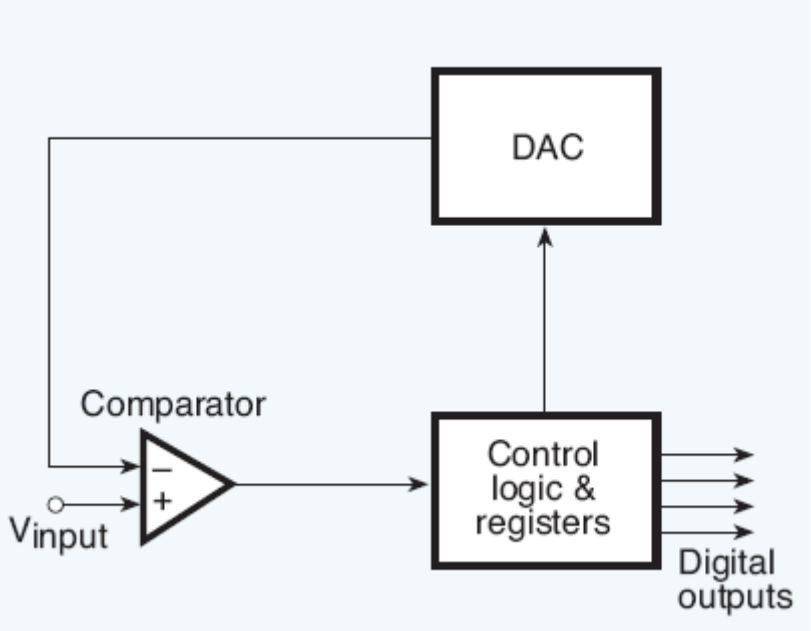


Figure 2.6 .Successive approximation ADC.

2.2.4.3 Flash ADCs

They are used in applications where very high bandwidth and sample rates are required, and moderate resolution is acceptable. For an N bit converter 2N-1 comparators and 2N resistors are required. The 2N resistors provide a reference voltage. For each comparator the reference voltage is one least significant bit greater than the reference voltage for the comparator immediately below it.

Each comparator produces a 1 when its analogue input voltage is higher than the reference voltage applied to it; otherwise the comparator output is zero. The point where the code

changes from ones to zeros is where the input signal becomes smaller than the respective comparator reference voltage. The output bits from these comparators form a thermometer code, so called because it can be displayed as a column of continuous Ones below a similar sequence of zeros. This design is similar to mercury thermometer where mercury column always rises to appropriate temperature and no mercury is present above that the temperature.[8]

The thermometer code is then decoded to an appropriate digital output code. The comparators are wideband and low gain because at high frequency it is hard to obtain both high bandwidth and gain. The comparators are designed for low offset so that the input offset for each comparator is less than the LSB of the ADC.This ensures that the offset cannot falsely trip the ADC resulting in a digital output that is not a representative of the thermometer code.[8]

Flash converters are very fast, since the speed of clocked comparators and logic can be quite high. This makes them well suited to real-time oscilloscope applications. However, numerous disadvantages also exist. The complexity of the circuits grows rapidly as resolution is increased, since there are 2*n* clocked comparators. Furthermore, the power, input capacitance, clock capacitance, and physical extent of the comparator array on the integrated circuit all increase directly with the number of comparators.

The size of the comparator array is important since the flash converter typically samples rapidly changing input signals. If all comparators do not sample the input at the same point on the waveform, errors can result. Furthermore, propagation delays of the signal to all comparators are difficult to match as the array size increases. This is one reason that flash converters are typically used in conjunction with a sample and hold circuit which samples the input and ideally provides an unchanging signal to all comparators at the time of clocking. Modifications to the flash architecture can be used to reduce the cost of higher resolution. These techniques, which include analog encoding, folding, and interpolating, can reduce input capacitance and the size of the comparator array considerably.[8]

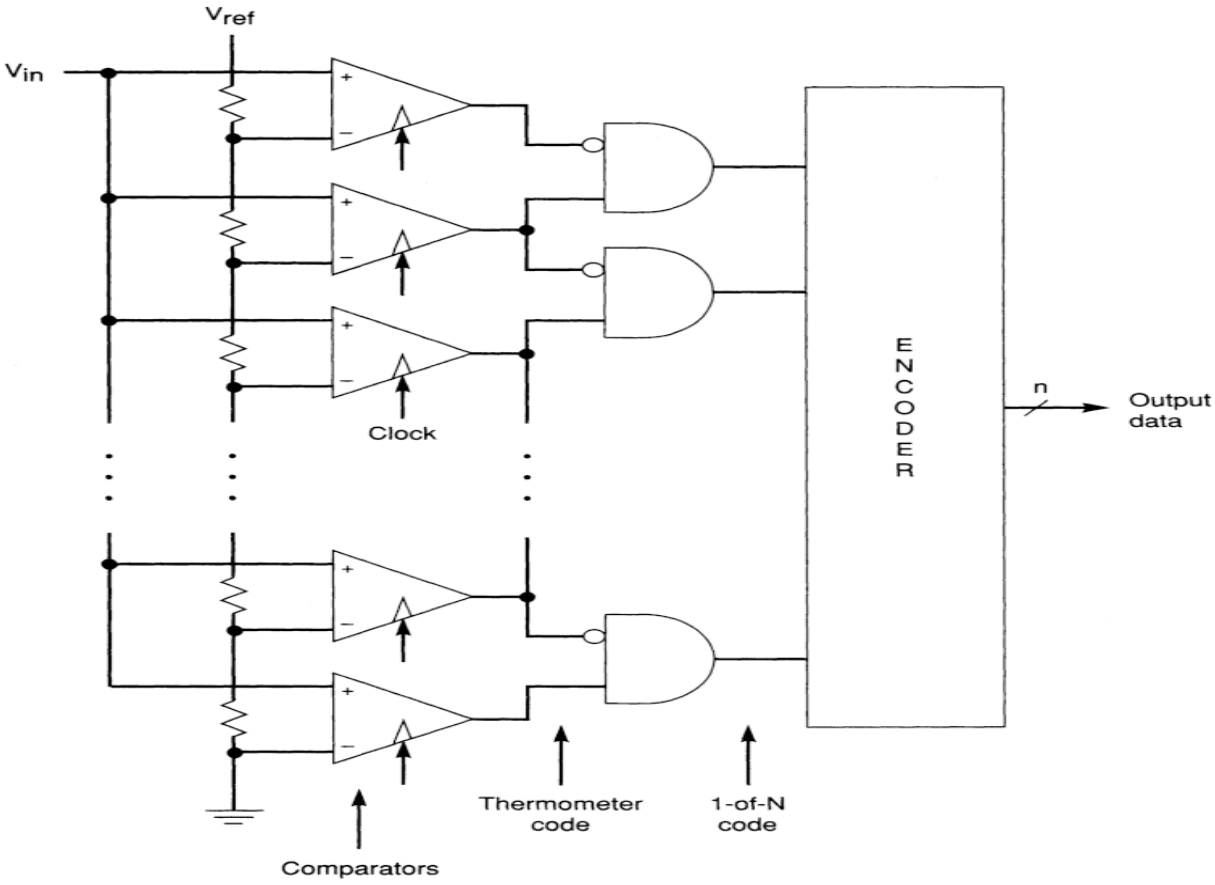
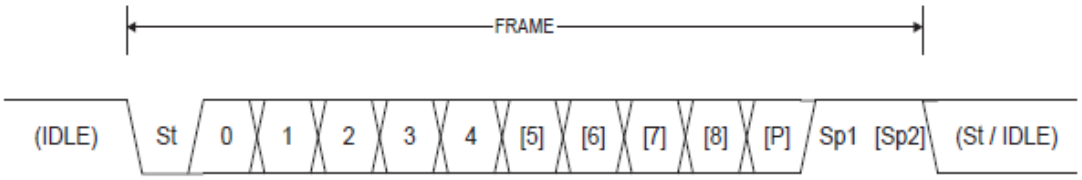


Figure 2.7 Block diagram of a flash A/D converter. The clocked comparators sample the input and form a thermometer code, which is converted to binary.

2.3 SERIAL COMMUNICATION

Serial communication is one of the simplest ways of communication between devices such as a microcontroller and PC or vice versa. It requires only single wire for transmission of a data and another for receiving.The transmitted sequence of data bits are encapsulated between start and stop bits.



St	Start bit, always low.
(n)	Data bits (0 to 8).
P	Parity bit. Can be odd or even.
Sp	Stop bit, always high.

Figure 2.8.Illustration of serial communication

The RS232 standard is used for serial communication. There are three ways in which serial communication can be done

- **Simplex:** Transmission is done in one direction.
- **Half duplex:** Transmission can be done in both the direction but one side at a time.
- **Full duplex:** Transmission can be done in both the direction simultaneously

While doing serial communication with a computer running on windows it is necessary to transmit the data as ascii characters since windows may not understand other forms of data types[9].During processing the ascii characters are then converted to appropriate data types e.g. integers, double etc.[10],[11].This is also necessary on the microcontroller side[12]. There are two methods of serial communication, synchronous and asynchronous.

2.3.1 Synchronous Serial Communication

In Synchronous communication method complete block (characters),byte or codeword is sent at a time. It doesn't require any additional bits (start, stop or parity) to be added for the synchronization of frame. The devices are synchronized by clock.This is much more efficient in bandwidth usage.

2.3.2 Asynchronous Serial Communication

In asynchronous serial communication protocol a start signal is sent prior to each byte, character or codeword and a stop bit is sent after after each codeword. The start signal serves to prepare the receiving mechanism for the reception and registration of a symbol and the stop signal serves to bring the receiving mechanism to rest in preparation for the reception of the next symbol. This is relatively simple and therefore inexpensive. However it has a high overhead in that each byte carries at least two extra bits hence a significant loss in line bandwidth.

2.3.3 Parity Check Bit.

It is possible that an error can occur during transmission of the data. Parity checking is used to detect occurrence of an error during transmission. The transmitter calculates the parity of the digital word and transmits. At the receiver if the received parity does not match with the received parity bit a parity error flag is set to indicate a transmission error and can request for re –transmission. Most common used parity bits are odd, even and no parity.
However these have the ability to detect one erroneous bit in each byte. To detect and correct more than one error other methods of error control coding could be used for example: BCH codes, Reed Solomon codes etc.

CHAPTER THREE: DESIGN AND IMPLEMENTATION

3.1 Design Tools.

Microsoft Visual C++

This design tool was chosen because it allows the use of Microsoft Foundation Classes(MFC) which makes it easier to design a Graphical User Interface to display the input values.

WinAvr

This was used as the Integrated Development Environment for programming the atmega 32 microcontroller

3.2 Algorithm

This describes the general steps undertaken in the project

1.Obtain the four analogue inputs.

2.Perform analogue to digital conversion of the inputs.

3.Transmit the digital values to pc.

4.Display and save the data.

The flow chart below illustrates the implementation of the above algorithm.

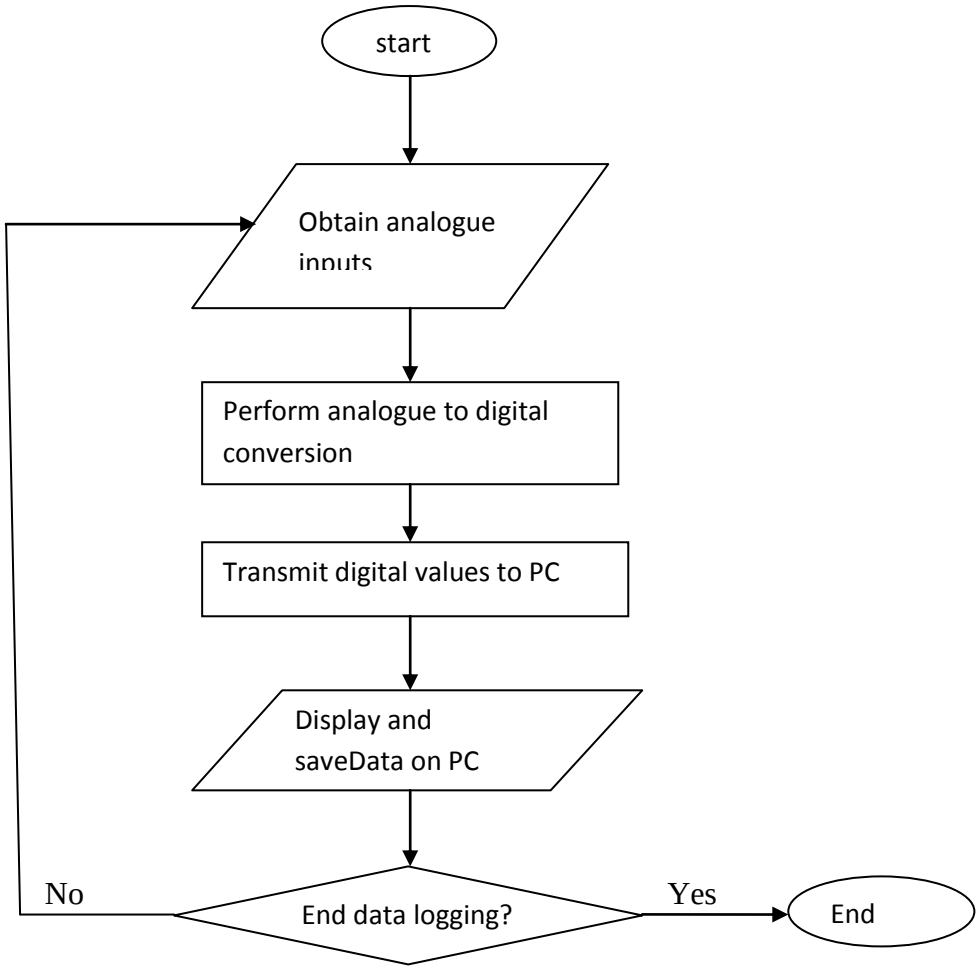


Figure 3.1 PC data acquisition flow chart.

3.2.1 Analogue Data Acquisition

The four analogue inputs used are the temperature sensor,humidity sensor and two voltage sources.They are fed into PORTA of atmega32 microcontroller.The voltages from the sources can be varied using potentiometers.Zener diodes have been used to limit the voltage below 5V so as to avoid destroying the microcontroller.

3.2.2 Analogue to Digital Conversion

Analogue to digital conversion was done using the inbuilt ADC of the atmega32 microcontroller with the reference voltage set at 5V.The four ADC channels are read sequentially i.e from ADC0 to ADC1 to ADC2 toADC3 then back to ADC0.This process continuous as long the microcontroller is powered.

3.2.3 Serial Communication

Data exchange between the microcontroller and the PC was done serially.Asynchronous mode was used since there is no common clock between PC and microcontroller.
On the microcontroller side the values returned by the ADC are converted to ASCII characters for transmission.They are then transmitted one character at a time.Since the four channels are read sequentially a channel identification number is also included in the transmission.This enables the program on the PC to distinguish between the four inputs.Channel 0 is represented by 0 channel 1 by 1,channel 2 by 2 and channel 3 by 3.In addition to the channel identity also a symbol is transmitted to ensure the correct sequence of data is read at the PC.In this case S was used.In general a correct sequence of data from a given channel is SD1D2D3N where D1D2D3 are the values from the ADC and N is the channel identity.
The FTDI usb-serial cable was used to connect the microcontroller to the PC.The baud rate was chosen as 19200 bits per second.

3.2.4 GUI Program on PC

This program controls the data acquisition process.This includes displaying current and past data,saving and setting the serial port parameters like the Baud rate and parity.It can also start and stop the data logging process.This was written in C++ using Microsoft Foundation Classes.This program decodes the sequence from the microcontroller and displays each input in its respective edit control and column basing on the channel number transmitted.

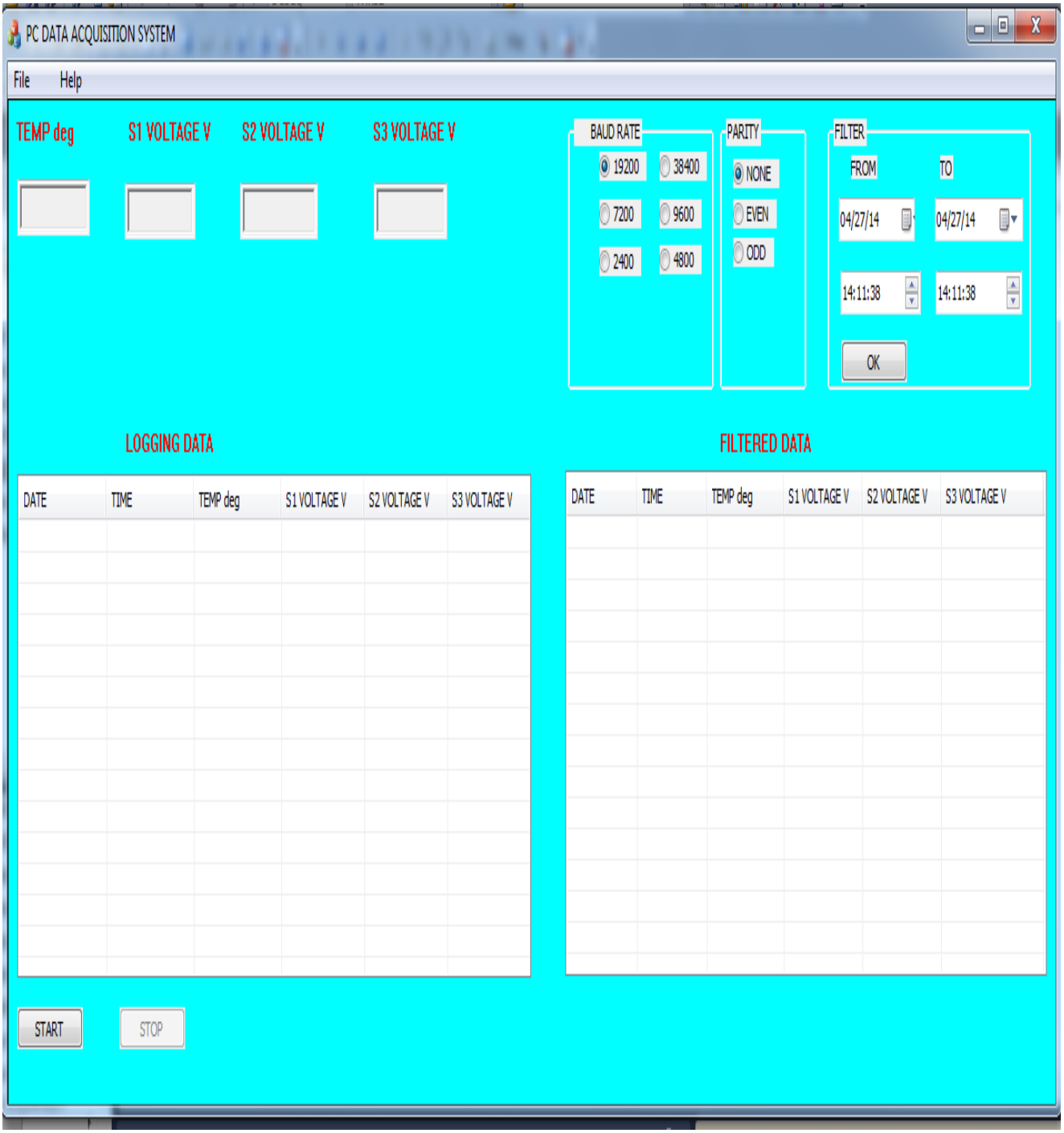


Figure 3.2 GUI for PC data acqusition

3.2.5 Circuit Diagram

The circuit diagram for the PC data acquisition system is shown below.

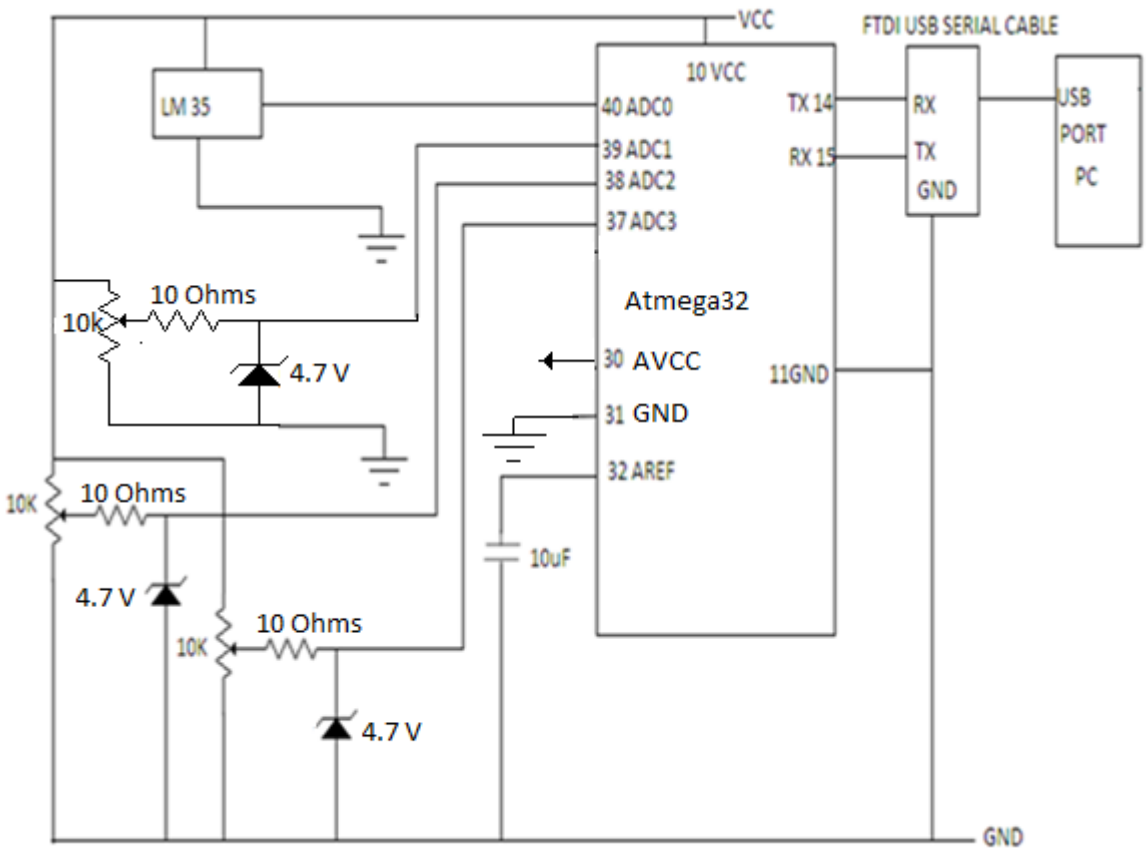


Figure 3.3 PC data acquisition system Circuit diagram.

Zener diodes were used to limit the input voltages to below 4.7 to avoid destroying the microcontroller with higher voltages.

Resistors to limit the current in the zener diodes were calculated as below:

$$R = \frac{Voltage\ at\ potentiometer\ input - voltage\ at\ adc\ input}{load\ current} \qquad (3.1)$$

= (5.5 – 4.7) /100mA

=8 Ohms.

Resistors of 10 Ohms were available and they were used.Here a maximum allowed voltage input of 5.5 V was assumed.

3.2.6 Practical design

The following were the practical circuits used for the project.

The final circuit on the PCB board(Figure 3.5) is well labeled.

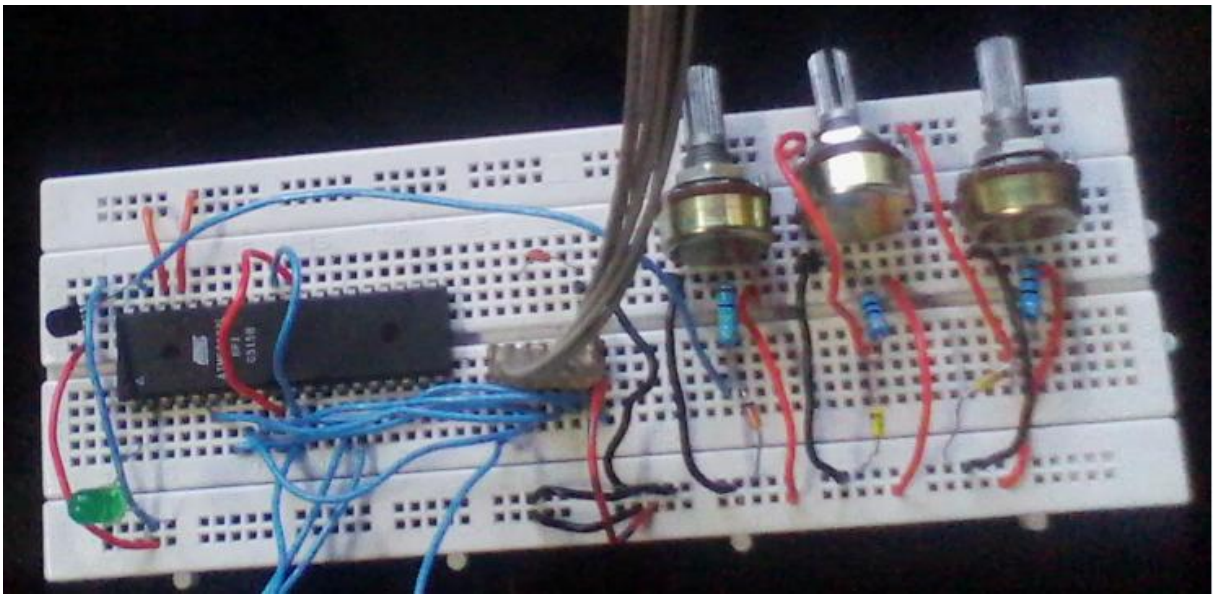


Figure 3.4 Practical circuit on Breadboard

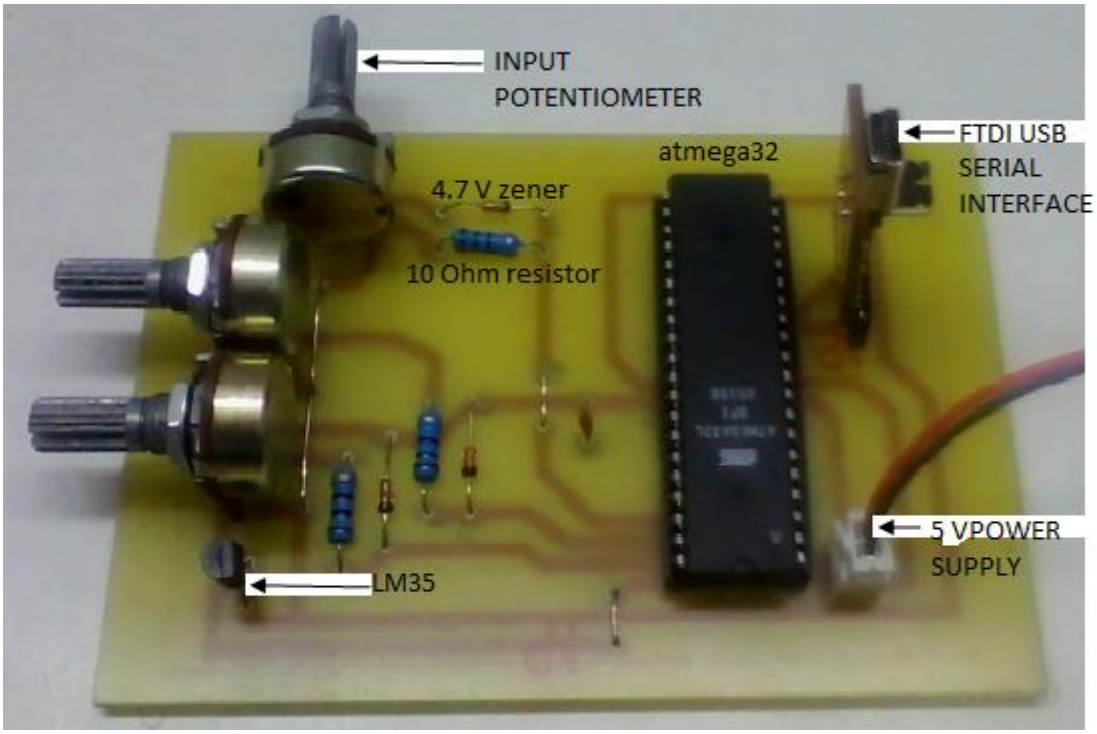


Figure 3.5 Practical circuit on fabricated PCB board

CHAPTER 4: RESULTS AND DISCUSSION

4.1 Results

The following results were obtained from the designed system:

4.1.1 Simulated Results

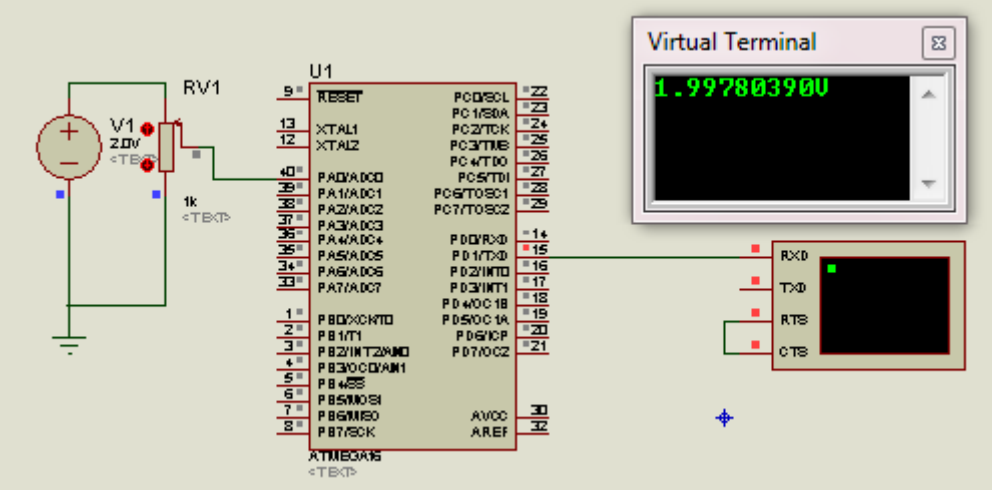
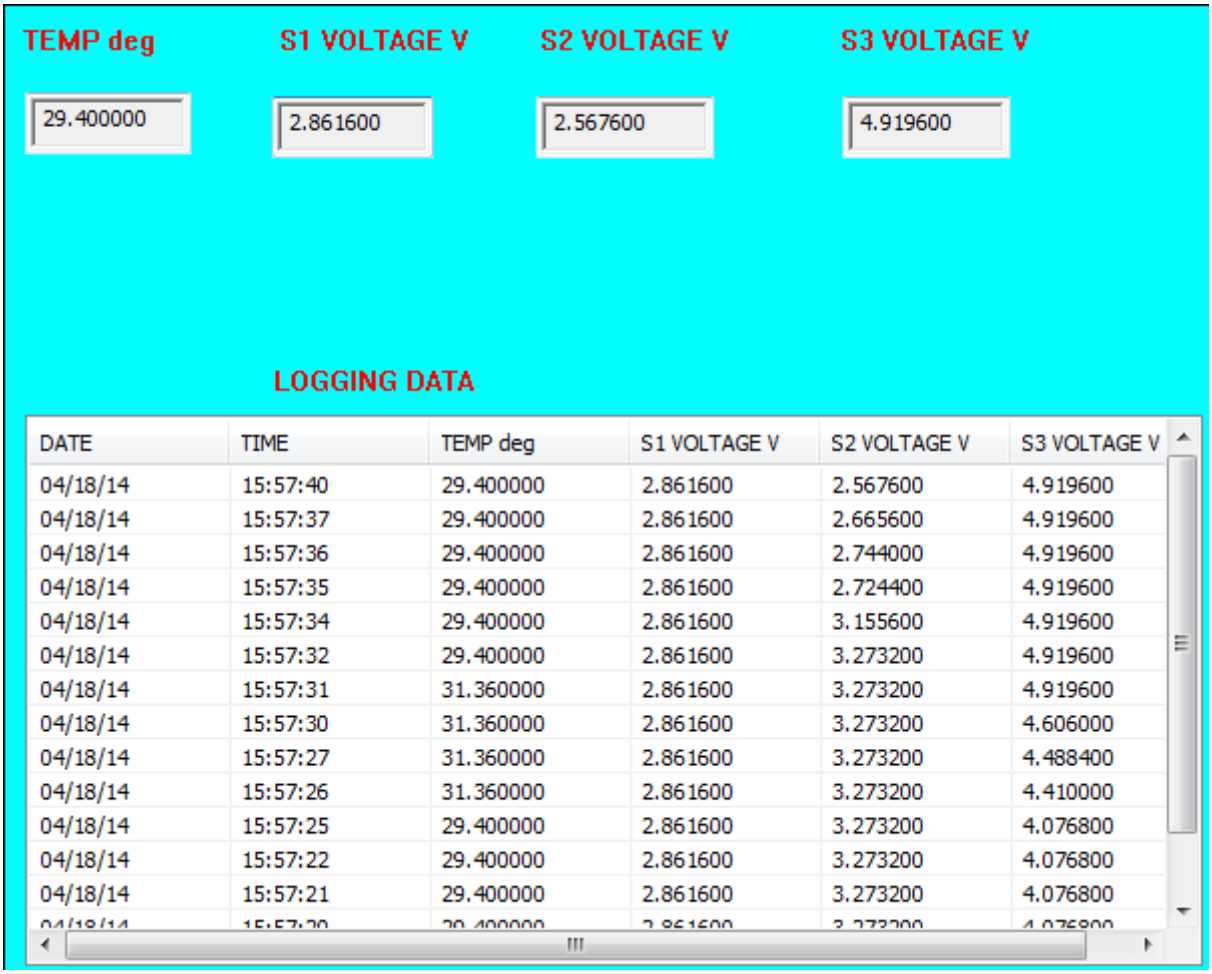


Figure 4.1 Simulated results for one input using proteus.It was not possible to multiplex the virtual terminal in proteus to show the four inputs.

4.1.2 Results for logging data

This displays the logging data continuously

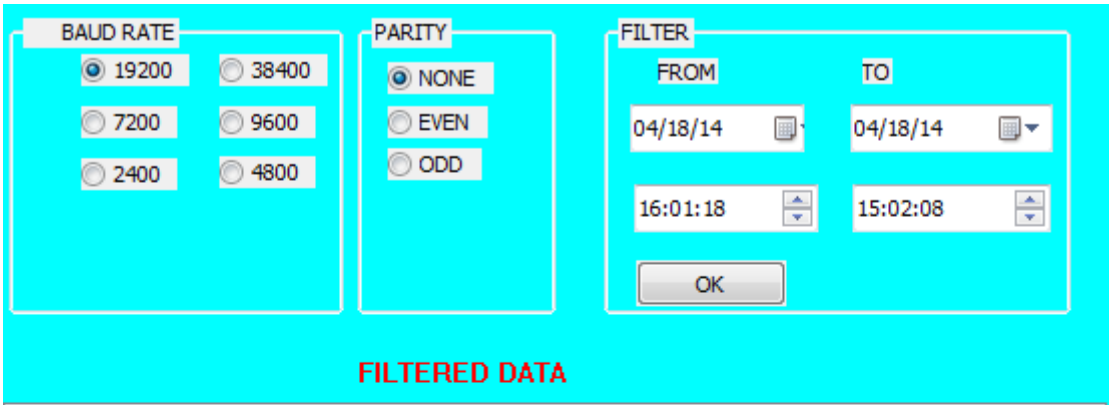


DATE	TIME	TEMP deg	S1 VOLTAGE V	S2 VOLTAGE V	S3 VOLTAGE V
04/18/14	15:57:40	29.400000	2.861600	2.567600	4.919600
04/18/14	15:57:37	29.400000	2.861600	2.665600	4.919600
04/18/14	15:57:36	29.400000	2.861600	2.744000	4.919600
04/18/14	15:57:35	29.400000	2.861600	2.724400	4.919600
04/18/14	15:57:34	29.400000	2.861600	3.155600	4.919600
04/18/14	15:57:32	29.400000	2.861600	3.273200	4.919600
04/18/14	15:57:31	31.360000	2.861600	3.273200	4.919600
04/18/14	15:57:30	31.360000	2.861600	3.273200	4.606000
04/18/14	15:57:27	31.360000	2.861600	3.273200	4.488400
04/18/14	15:57:26	31.360000	2.861600	3.273200	4.410000
04/18/14	15:57:25	29.400000	2.861600	3.273200	4.076800
04/18/14	15:57:22	29.400000	2.861600	3.273200	4.076800
04/18/14	15:57:21	29.400000	2.861600	3.273200	4.076800

Figure 4.2 PC data acquisition Logging data

4.1.2 Results for filtered data

This displays logged data filtered over a given interval of time, in this case on 18/04/2014 from 16:01:18 to 16:02:08.



FILTERED DATA

DATE	TIME	TEMP deg	S1 VOLTAGE V	S2 VOLTAGE V	S3 VOLTAGE V
04/18/14	16:01:18	29.400000	2.861600	2.567600	4.919600
04/18/14	16:01:23	29.400000	2.861600	2.567600	4.919600
04/18/14	16:01:28	29.400000	2.861600	2.567600	4.919600
04/18/14	16:01:33	29.400000	2.861600	2.567600	4.919600
04/18/14	16:01:38	29.400000	2.861600	2.567600	4.919600
04/18/14	16:01:43	29.400000	2.861600	2.567600	4.919600
04/18/14	16:01:48	29.400000	2.861600	2.567600	4.919600
04/18/14	16:01:53	29.400000	2.861600	2.567600	4.919600
04/18/14	16:01:58	29.400000	2.861600	2.567600	4.919600
04/18/14	16:02:03	29.400000	2.861600	2.567600	4.919600
04/18/14	16:02:08	29.400000	2.861600	2.567600	4.919600

Figure 4.3 Filtered data according to the time and date selected

4.1.3 Discussion

The displayed data varied as the inputs.However the zener diode did not limit the voltage to exactly 4.7 volts as required. This could be attributed to the zener tolerance of about +/-5% which brings the voltage to about 4.935.However this is still below 5V that was set as the maximum value of the ADC and the microcontroller would still not be destroyed.

CHAPTER 5: CONCLUSION AND RECOMMENDATIONS

5.1 Conclusion

In this paper the design of a PC based data acqusition system has been presented.An interface was designed to obtain four analogue inputs and transmit them to the PC.The simulated and experimental results show that a microcontroller can be used for data acqusition,analogue to digital conversion and serial transmission to a PC with a program to process the data.

5.2 Recommendations

Although this project was specifically designed for four analogue inputs it can be improved and made more robust.Thus the recommendations for future work are as follows:

1. Expanding the number of inputs to more than four and enable users to select the number of inputs they want to use at any given time using the control program on the PC.
2. To save on memory subsequent values should be saved only if they differ from previous recorded values since in that time interval the input values will be constant.
3. Other forms of transmission to be incorporated to ensure the data logged can be viewed at distances further from the logging station.This include the internet.
4. Other control functionalities can be written so as to allow the system to execute certain control operations automatically such as switching the inputs on and off.
5. Incorporating error control coding techniques to correct any errors that could have occurred during transmission.

APPENDIX A MC.C

This is the microcontroller code.It does analogue to digital conversion and serial communication to PC.

```
#include<avr/io.h>
#include<util/delay.h>
#include<stdlib.h>
int data;
char channel;
void Usartinit(unsigned int baud)
{
    UBRRH = (unsigned char)(baud>>8);
    UBRRL = (unsigned char)baud;            //Enable tra
    UCSRB = (1<<TXEN)|(1<<RXEN);
    UCSRC = (1<<URSEL)|(1<<USBS)|(3<<UCSZ0);

}
void USART_Receive(void)
{
    char ReceivedByte;
    while (( UCSRA & (1 << RXC )) == 0) {};
    ReceivedByte = UDR ;
}

void USART_Transmit_char(char txdata )
{

    while ( !( UCSRA & (1<<UDRE)) )         //Wait for
    ;

    UDR = txdata;                           // Put data
}
void adc_init(void)
{
   ADMUX=0b01100000;
   ADCSRA=(1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);
}
```

```
void ReadADC(void)
{
    switch(ADMUX)
    {
      case 0b01100000:
      ADCSRA|=(1<<ADSC);
      while(ADCSRA&(1<<ADSC));
      data=ADCH;
      channel='0';
      ADMUX=0b01100001;break;

      case 0b01100001:
      ADCSRA|=(1<<ADSC);
      while(ADCSRA&(1<<ADSC));
      data=ADCH;
      channel='1';
      ADMUX=0b01100010;break;
      case 0b01100010:
      ADCSRA|=(1<<ADSC);
      while(ADCSRA&(1<<ADSC));
      data=ADCH;
      channel='2';
      ADMUX=0b01100011;break;
```

```
        case 0b01100011:
        ADCSRA|=(1<<ADSC);
        while(ADCSRA&(1<<ADSC));
        data=ADCH;
        channel='3';
        ADMUX=0b01100000;break;
      }
}

int main(void)
{
   char dat[3];
   adc_init();
   Usartinit(12);
   while(1)
   {
        USART_Receive();
        ReadADC();
        itoa(data,dat,10);
        USART_Transmit_char('s');
        for(int i=0;i<3;i++)
        {
          USART_Transmit_char(dat[i]);
        }
        USART_Transmit_char(channel);
   }
}
```

APPENDIX B: PROJECT CODEDlg.h

This is the header file for the dialog based application to read,save and display the input data.

It contains variable declaration and function definitions.

```
// PROJECT CODEDlg.h : header file
//

#pragma once
#include "afxwin.h"
#include "afxdtctl.h"


// CPROJECTCODEDlg dialog
class CPROJECTCODEDlg : public CDialogEx
{
// Construction
public:
	CPROJECTCODEDlg(CWnd* pParent = NULL);	// standard constructor

// Dialog Data
	enum { IDD = IDD_PROJECTCODE_DIALOG };

	protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV support


// Implementation
protected:
	HICON m_hIcon;

	// Generated message map functions
	virtual BOOL OnInitDialog();
	afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
	afx_msg void OnPaint();
	afx_msg HCURSOR OnQueryDragIcon();
	DECLARE_MESSAGE_MAP()
public:
	// Variable name for the 1st input display edit box
	CEdit m_input1;
	CEdit m_input2;
	CEdit m_input3;
	CEdit m_input4;
	void OpenConfigPort(void);
	char ReadPortData(void);
	afx_msg void OnBnClickedShow();
	afx_msg void OnFileAbout();
	void DecodeSerialData(void);
	CButton m_start;
	afx_msg void OnTimer(UINT_PTR nIDEvent);
	afx_msg void OnBnClickedStop();
	CString GetCurrentTime(void);
	CString GetCurrentDate(void);
	void FillTable(HWND hwnd, CString value, int Row , int Col);
	int CheckEditboxData();
	void FillDataSheet(int k);
	CButton m_parity;
	CButton m_baud;
	void SaveDataLogged(int k);
	CDateTimeCtrl m_date2;
```

```
        CDateTimeCtrl m_date1;
        CDateTimeCtrl m_date3;
        CDateTimeCtrl m_date4;
        afx_msg void OnBnClickedFilter();
        void FillFilterSheet(int StartRow,int EndRow);
};
```

APPENDIX C: PROJECT CODE .h

This code contains the main header file for PC data acquisition application

```
//main header file for the PROJECT_NAME application


#pragma once

#ifndef __AFXWIN_H__
	#error "include 'stdafx.h' before including this file for PCH"
#endif

#include "resource.h"		// main symbols


// CPROJECTCODEApp:
// See PROJECT CODE.cpp for the implementation of this class
//

class CPROJECTCODEApp : public CWinApp
{
public:
	CPROJECTCODEApp();

// Overrides
public:
	virtual BOOL InitInstance();

// Implementation

	DECLARE_MESSAGE_MAP()
};

extern CPROJECTCODEApp theApp;
```

APPENDIX D: RESOURCE.h

This code contains IDs of the controls on the Application Dialog.

```
#define IDM_ABOUTBOX                    0x0010
#define IDD_ABOUTBOX                    100
#define IDS_ABOUTBOX                    101
#define IDD_PROJECTCODE_DIALOG          102
#define IDR_MAINFRAME                   128
#define IDR_MENU1                       129
#define IDC_EDIT1                       1000
#define IDC_EDIT2                       1001
#define IDC_EDIT3                       1002
#define IDC_EDIT4                       1003
#define IDC_BUTTON1                     1004
#define IDC_BUTTON2                     1005
#define IDC_RADIO1                      1006
#define IDC_RADIO2                      1007
#define IDC_RADIO3                      1008
#define IDC_RADIO4                      1009
#define IDC_RADIO5                      1010
#define IDC_RADIO6                      1011
#define IDC_RADIO7                      1012
#define IDC_RADIO8                      1013
#define IDC_RADIO9                      1014
#define IDC_LIST1                       1015
#define IDC_LIST2                       1020
#define IDC_DATETIMEPICKER1             1021
#define IDC_DATETIMEPICKER2             1022
#define IDC_BUTTON3                     1023
#define IDC_DATETIMEPICKER3             1024
#define IDC_DATETIMEPICKER4             1025
#define ID_FILE_ABOUT                   32771
#define ID_FILE_EXIT                    32772
```

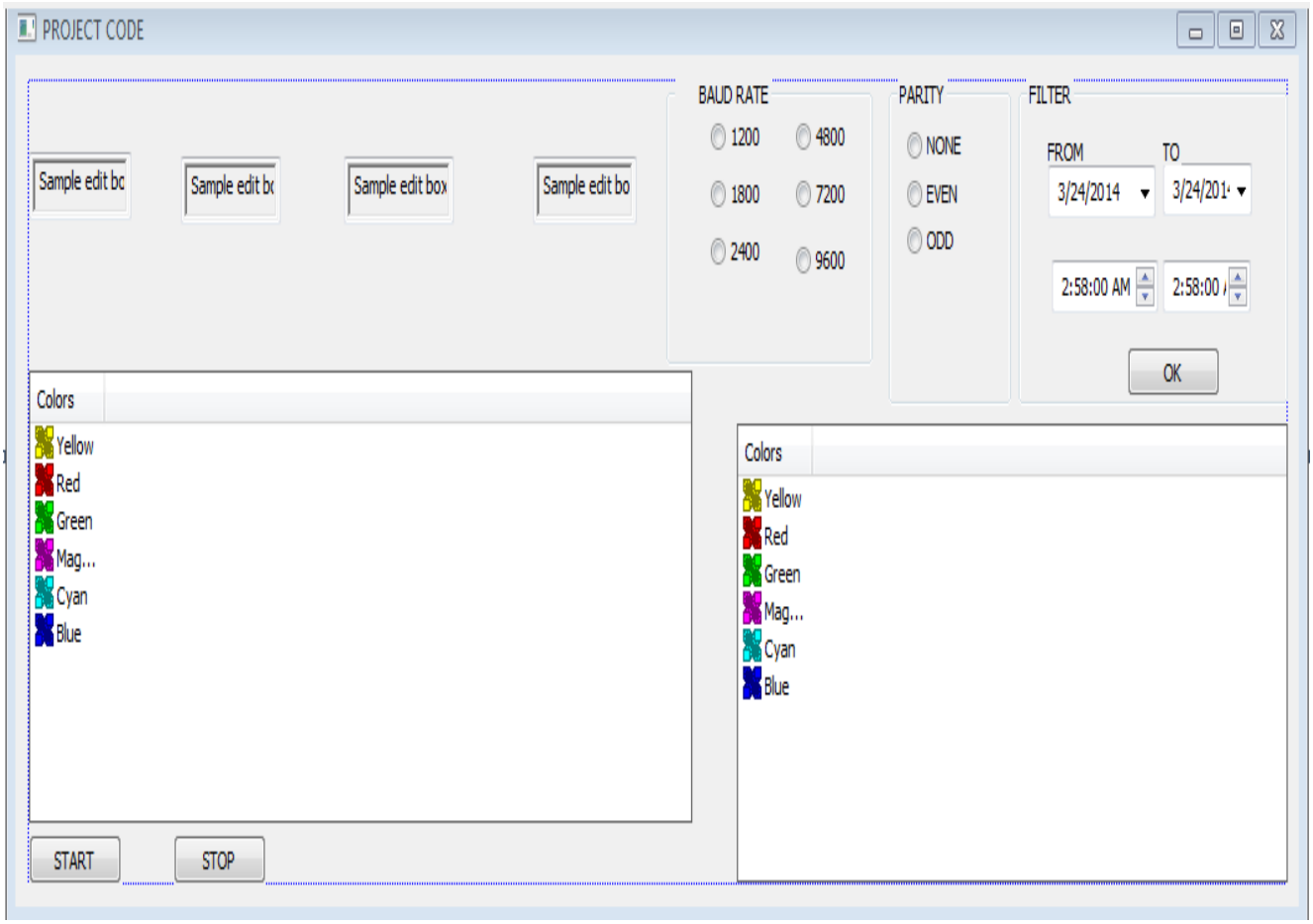
APPENDIX E: SAVEDATA.h

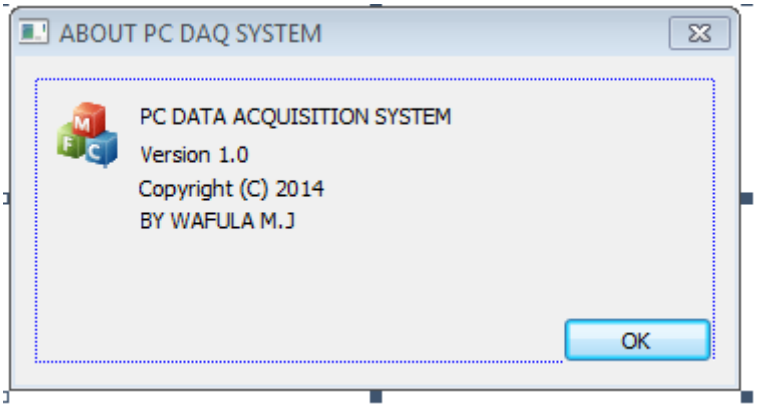
This is the header file for the class definition to save data

```
#pragma once

// SaveData command target


class SaveData : public CObject
{
public:
	SaveData();
	virtual ~SaveData();
	virtual void Serialize(CArchive&ar);
	CArray<CString,CString>data1;
	CArray<CString,CString>data2;
	CArray<CString,CString>data3;
	CArray<CString,CString>data4;
	CArray<CString,CString>data5;
	CArray<CString,CString>data6;
};
```

APPENDIX F: RESOURCE.RC

This is the main dialog and the About Box resource file.It shows the graphical design of the dialog window and AboutBox dialog.





APPENDIX H: PROJECT CODEDlg.cpp

This is the implementation file for the dialog based application to read,save and display the input data.

```
// PROJECT CODEDlg.cpp : implementation file
//

#include "stdafx.h"
#include "PROJECT CODE.h"
#include "PROJECT CODEDlg.h"
#include "afxdialogex.h"
#include<iostream>
#include<Windows.h>
#include "daqabout.h"
#include "SaveData.h"
#include<conio.h>
#ifdef _DEBUG
#define new DEBUG_NEW
#endif
HANDLE serial;
bool CommFlag;
DCB serialparams;
CString strdat;
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialogEx
{
public:
	CAboutDlg();

// Dialog Data
	enum { IDD = IDD_ABOUTBOX };

	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support

// Implementation
protected:
	DECLARE_MESSAGE_MAP()
};
```

```
CAboutDlg::CAboutDlg() : CDialogEx(CAboutDlg::IDD)
{
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialogEx::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialogEx)
END_MESSAGE_MAP()
CPROJECTCODEDlg::CPROJECTCODEDlg(CWnd* pParent /*=NULL*/)
	: CDialogEx(CPROJECTCODEDlg::IDD, pParent)

{
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}
void CPROJECTCODEDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialogEx::DoDataExchange(pDX);
	DDX_Control(pDX, IDC_EDIT3, m_input1);
	DDX_Control(pDX, IDC_EDIT2, m_input2);
	DDX_Control(pDX, IDC_EDIT1, m_input3);
	DDX_Control(pDX, IDC_EDIT4, m_input4);
	DDX_Control(pDX, IDC_BUTTON1, m_start);
	DDX_Control(pDX, IDC_RADIO7, m_parity);
	DDX_Control(pDX, IDC_RADIO1, m_baud);
	DDX_Control(pDX, IDC_DATETIMEPICKER2, m_date2);
	DDX_Control(pDX, IDC_DATETIMEPICKER1, m_date1);
	DDX_Control(pDX, IDC_DATETIMEPICKER3, m_date3);
	DDX_Control(pDX, IDC_DATETIMEPICKER4, m_date4);
}

BEGIN_MESSAGE_MAP(CPROJECTCODEDlg, CDialogEx)
	ON_WM_SYSCOMMAND()
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	ON_BN_CLICKED(IDC_BUTTON1, &CPROJECTCODEDlg::OnBnClickedShow)
	ON_COMMAND(ID_FILE_ABOUT, &CPROJECTCODEDlg::OnFileAbout)
	ON_WM_TIMER()
	ON_BN_CLICKED(IDC_BUTTON2, &CPROJECTCODEDlg::OnBnClickedStop)
	ON_BN_CLICKED(IDC_BUTTON3, &CPROJECTCODEDlg::OnBnClickedFilter)
	ON_WM_CLOSE()
END_MESSAGE_MAP()


// CPROJECTCODEDlg message handlers

BOOL CPROJECTCODEDlg::OnInitDialog()
{
	CDialogEx::OnInitDialog();
	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);

	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
		BOOL bNameValid;
		CString strAboutMenu;
		bNameValid = strAboutMenu.LoadString(IDS_ABOUTBOX);
		ASSERT(bNameValid);
		if (!strAboutMenu.IsEmpty())
```

```
		{
			pSysMenu->AppendMenu(MF_SEPARATOR);
			pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
		}
	}
	SetIcon(m_hIcon, TRUE);
	SetIcon(m_hIcon, FALSE);
    SetWindowText(_T("PC DATA ACQUISITION SYSTEM"));
	CListCtrl*lst=new CListCtrl;
	lst=reinterpret_cast<CListCtrl*>(GetDlgItem(IDC_LIST1));
	lst->SetExtendedStyle(LVS_EX_GRIDLINES|LVS_EX_FULLROWSELECT);

	CListCtrl*lst1=new CListCtrl;
	lst1=reinterpret_cast<CListCtrl*>(GetDlgItem(IDC_LIST2));
	lst1->SetExtendedStyle(LVS_EX_GRIDLINES|LVS_EX_FULLROWSELECT);

	lst->InsertColumn(0,_T("DATE"),0,100,-1);
	lst->InsertColumn(1,_T("TIME"),0,100,-1);
	lst->InsertColumn(2,_T("TEMP deg"),0,100,-1);
	lst->InsertColumn(3,_T("S1 VOLTAGE V"),0,95,-1);
	lst->InsertColumn(4,_T("S2 VOLTAGE V"),0,95,-1);
	lst->InsertColumn(5,_T("S3 VOLTAGE V"),0,100,-1);

	lst1->InsertColumn(0,_T("DATE"),0,80,-1);
	lst1->InsertColumn(1,_T("TIME"),0,80,-1);
	lst1->InsertColumn(2,_T("TEMP deg"),0,88,-1);
	lst1->InsertColumn(3,_T("S1 VOLTAGE V"),0,90,-1);
	lst1->InsertColumn(4,_T("S2 VOLTAGE V"),0,90,-1);
	lst1->InsertColumn(5,_T("S3 VOLTAGE V"),0,117,-1);

	CWnd*start=(GetDlgItem(IDC_BUTTON1));
	start->EnableWindow(TRUE);
	CWnd*stop=(GetDlgItem(IDC_BUTTON2));
	stop->EnableWindow(FALSE);

	m_baud.SetCheck(1);     //select 9600 baud rate
	m_parity.SetCheck(1);   //select no parity

	m_date1.SetFormat(_T("MM/dd/yy"));//set time and date formats
	m_date2.SetFormat(_T("HH:mm:ss"));
	m_date3.SetFormat(_T("MM/dd/yy"));
	m_date4.SetFormat(_T("HH:mm:ss"));

   return TRUE;  // return TRUE  unless you set the focus to a control
}

void CPROJECTCODEDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
	if ((nID & 0xFFF0) == IDM_ABOUTBOX)
	{
		CAboutDlg dlgAbout;
		dlgAbout.DoModal();
	}
	else
	{
		CDialogEx::OnSysCommand(nID, lParam);
	}
}

void CPROJECTCODEDlg::OnPaint()
{
	CPaintDC dc(this);
```

```
	if (IsIconic())
	{

	 SendMessage(WM_ICONERASEBKGND, reinterpret_cast<WPARAM>(dc.GetSafeHdc()), 0);
		int cxIcon = GetSystemMetrics(SM_CXICON);
		int cyIcon = GetSystemMetrics(SM_CYICON);
		CRect rect;
		GetClientRect(&rect);
		int x = (rect.Width() - cxIcon + 1) / 2;
		int y = (rect.Height() - cyIcon + 1) / 2;
		dc.DrawIcon(x, y, m_hIcon);
	}
	else
	{
		CDialogEx::OnPaint();

	}
	   CBrush brush(RGB(0,255,255));
	   dc.SelectObject(brush);
	   dc.Rectangle(0,0,3000,3000);
	   dc.SetTextColor(RGB(255,0,0));
	   dc.SetBkColor(RGB(0,255,255));
	   dc.TextOutW(10,10,_T("TEMP deg"));
	   dc.TextOutW(138,10,_T("S1 VOLTAGE V"));
	   dc.TextOutW(268,10,_T("S2 VOLTAGE V"));
	   dc.TextOutW(418,10,_T("S3 VOLTAGE V"));
	   dc.TextOutW(815,180,_T("FILTERED DATA"));
	   dc.TextOutW(135,180,_T("LOGGING DATA"));




}
HCURSOR CPROJECTCODEDlg::OnQueryDragIcon()
{
	return static_cast<HCURSOR>(m_hIcon);
}

void CPROJECTCODEDlg::OpenConfigPort(void)//opens and configures pc serial port
{
	serial=CreateFile(_T("COM1"),GENERIC_READ|GENERIC_WRITE,0,0,OPEN_EXISTING,FILE_A
TTRIBUTE_NORMAL,0);

	if(serial==INVALID_HANDLE_VALUE)
	{
		if(GetLastError()==ERROR_FILE_NOT_FOUND)
		MessageBox(_T("PORT NOT FOUND"),_T("ERROR MESSAGE"),MB_ICONEXCLAMATION);
	}

	COMMTIMEOUTS time={0};
	if(!GetCommState(serial,&serialparams))
	{
		MessageBox(_T("ERROR GETTING PORT STATE"),_T("ERROR"),MB_ICONERROR);
	}

	serialparams.BaudRate=CBR_19200;
	serialparams.ByteSize=8;
	serialparams.Parity=NOPARITY;
	serialparams.StopBits=ONESTOPBIT;
	if(!SetCommState(serial,&serialparams))//Setting comm timeouts
	{
```

```
		MessageBox(_T("ERROR SETTING PORT STATE"),_T("ERROR"),MB_ICONERROR);
	}
	time.ReadIntervalTimeout=0;
	time.ReadTotalTimeoutConstant=0;
	time.ReadTotalTimeoutMultiplier=0;
	time.WriteTotalTimeoutConstant=0;
	time.WriteTotalTimeoutMultiplier=0;
	if(!SetCommTimeouts(serial,&time))
	{
		MessageBox(_T("TIMEOUTS COULD NOT BE SET"),_T("ERROR"),MB_ICONERROR);
	}


}




void CPROJECTCODEDlg::OnBnClickedShow()//Starts the timer
{
	SetTimer(0,1,0);
	SetTimer(1,1000,0);
	//SetTimer(2,5000,0);
	CWnd*stop=(GetDlgItem(IDC_BUTTON2));
	stop->EnableWindow(TRUE);
	CWnd*start=(GetDlgItem(IDC_BUTTON1));
	start->EnableWindow(FALSE);
}

void CPROJECTCODEDlg::OnFileAbout()//About dialog
{
	// TODO: Add your command handler code here
	KillTimer(0);KillTimer(1);KillTimer(2);
	daqabout dlg;dlg.DoModal();
	SetTimer(0,1,0);
	SetTimer(1,1000,0);
	SetTimer(2,2000,0);
}


void CPROJECTCODEDlg::DecodeSerialData(void)//reads serial data and determines which
channel was read
{
	CString str1,str2,str3,str4,str5;
	str1=ReadPortData();double temp;

 if(str1.Left(1)=='S')
	{
		str2=str1.Right(4);
		str3=str2.Left(3);
		int channel=_ttoi(str2.Right(1));
		int digvalue=_ttoi(str3);double actualvalue=0.0196*digvalue;
		str4.Format(_T("%f"),actualvalue);
		temp=actualvalue*100;
		str5.Format(_T("%f"),temp);
		switch(channel)
		{
		   case 0:m_input1.SetWindowTextW(str5);break;
		   case 1:m_input2.SetWindowTextW(str4);break;
```

```
                case 2:m_input3.SetWindowTextW(str4);break;
                case 3:m_input4.SetWindowTextW(str4);break;
            }



        }

}


void CPROJECTCODEDlg::OnTimer(UINT_PTR nIDEvent)//Manages the timer event,i.e updating
edit boxes and logging sheet
{


        int j=0;
        switch(nIDEvent)
        {

         case 0:DecodeSerialData();break;
               case 1:FillDataSheet(j);j++;UpdateDataBase();break;
               //case 2:SaveDataLogged();break;


        }

}


void CPROJECTCODEDlg::OnBnClickedStop()//stops the counter
{
        // TODO: Add your control notification handler code here
        KillTimer(0);KillTimer(1);
        CWnd*start=(GetDlgItem(IDC_BUTTON1));
        start->EnableWindow(TRUE);
        CWnd*stop=(GetDlgItem(IDC_BUTTON2));
        stop->EnableWindow(FALSE);

}


CString CPROJECTCODEDlg::GetCurrentTime(void)//Gets current time
{
        CTime t=CTime::GetCurrentTime();
        CString s=t.Format("%X");
        return s;

}
CString CPROJECTCODEDlg::GetCurrentDate(void)//Gets current date
{
        CTime t=CTime::GetCurrentTime();
        CString s=t.Format(TEXT("%x"));
        return s;

}
void CPROJECTCODEDlg::FillTable(HWND hwnd, CString value, int Row , int Col) //Fills
columns of the logging sheet with data
{
        TCHAR szString[256];
        wsprintf(szString,value,0);
        LVITEM lv;
```

```
	lv.mask=LVIF_TEXT;
	lv.iItem=Row;
	lv.iSubItem=Col;
	lv.pszText=szString;
	if(Col>0)
	::SendMessage(hwnd,LVM_SETITEM,(WPARAM)0,(WPARAM)&lv);
	else
		ListView_InsertItem(hwnd,&lv);

}

void CPROJECTCODEDlg::FillDataSheet(int k)//function gets data from edit controls and
fills into the sheet
{
	CString str1,str2,str3,str4;
	m_input1.GetWindowTextW(str1);
	m_input2.GetWindowTextW(str2);
	m_input3.GetWindowTextW(str3);
	m_input4.GetWindowTextW(str4);
	FillTable(::GetDlgItem(m_hWnd,IDC_LIST1),GetCurrentDate(),k,0);
	FillTable(::GetDlgItem(m_hWnd,IDC_LIST1),GetCurrentTime(),k,1);
	FillTable(::GetDlgItem(m_hWnd,IDC_LIST1),str1,k,2);
	FillTable(::GetDlgItem(m_hWnd,IDC_LIST1),str2,k,3);
	FillTable(::GetDlgItem(m_hWnd,IDC_LIST1),str3,k,4);
	FillTable(::GetDlgItem(m_hWnd,IDC_LIST1),str4,k,5);
}


void CPROJECTCODEDlg::SaveDataLogged()
{

	   SaveData s;
	      CFile
f(_T("D:\\DATAFILE\\DATA"),CFile::modeCreate|CFile::modeWrite|CFile::shareDenyNone);
	    CArchive ar(&f,CArchive::store);
	      s.Serialize(ar);
	         ar.Close();
	         f.Close();
}


void CPROJECTCODEDlg::OnBnClickedFilter()
{
	KillTimer(0);KillTimer(1);
	CString str1,str2,str3,str4;
	int startpos=-1;int endpos=-1;
	CListCtrl*lst=new CListCtrl;
	lst=reinterpret_cast<CListCtrl*>(GetDlgItem(IDC_LIST2));
	SaveData s;
	CFileFind ff;BOOL findfile=ff.FindFile(_T("D:\\DATAFILE\\DATA"));
	if(findfile==TRUE)
	{
		CFile f(_T("D:\\DATAFILE\\DATA"),CFile::modeRead|CFile::shareDenyNone);
	 CArchive ar(&f,CArchive::load);
		s.Serialize(ar);
	for(int i=0;i<data1.GetCount();i++)
	{
	    s.data1.Add(data1[i]);
	      s.data2.Add(data2[i]);
	      s.data3.Add(data3[i]);
	      s.data4.Add(data4[i]);
	      s.data5.Add(data5[i]);
```

```
			s.data6.Add(data6[i]);
		}
		}
		else
		{
			for(int i=0;i<data1.GetCount();i++)
			{
			s.data1.Add(data1[i]);
			s.data2.Add(data2[i]);
			s.data3.Add(data3[i]);
			s.data4.Add(data4[i]);
			s.data5.Add(data5[i]);
			s.data6.Add(data6[i]);
			}
		}
		MessageBox(s.data1[0]);

}


void CPROJECTCODEDlg::FillFilterSheet(int StartRow, int EndRow)
{
		HWND hwnd=::GetDlgItem(m_hWnd,IDC_LIST2);
		SaveData s;
		CListCtrl*lst=new CListCtrl;
		lst=reinterpret_cast<CListCtrl*>(GetDlgItem(IDC_LIST2));
		CFile f(_T("D:\\DATAFILE\\DATA"),CFile::modeRead|CFile::shareDenyNone);
		 CArchive ar(&f,CArchive::load);
			s.Serialize(ar);

		for(int i=0;i<EndRow-StartRow;i++)
		{
			lst->DeleteItem(i);
		 FillTable(hwnd,s.data5[i+StartRow],i,0);
			FillTable(hwnd,s.data6[i+StartRow],i,1);
			FillTable(hwnd,s.data1[i+StartRow],i,2);
			FillTable(hwnd,s.data2[i+StartRow],i,3);
			FillTable(hwnd,s.data3[i+StartRow],i,4);
		   FillTable(hwnd,s.data4[i+StartRow],i,5);

		}

}
CString CPROJECTCODEDlg::ReadPortData(void)
{

		DWORD dword;BYTE byte;char d1,d2,d3,d4,d5;CString
str1,str2,str3,str4,str5,str6,str7;
		char data='R';
		OpenConfigPort();
		WriteFile(serial,&data,1,&dword,NULL);
		ReadFile(serial,&byte,1,&dword,0);
		d1=(char)byte;

		ReadFile(serial,&byte,1,&dword,0);
		d2=(char)byte;

		ReadFile(serial,&byte,1,&dword,0);
		d3=(char)byte;

	   ReadFile(serial,&byte,1,&dword,0);
		d4=(char)byte;
```

```
        ReadFile(serial,&byte,1,&dword,0);
        d5=(char)byte;

        str1.Format(_T("%c"),d1);
        str2.Format(_T("%c"),d2);
        str3.Format(_T("%c"),d3);
        str4.Format(_T("%c"),d4);
        str5.Format(_T("%c"),d5);
        str6=str1+str2+str3+str4+str5;
        CloseHandle(serial);

        return str6;

}



void CPROJECTCODEDlg::UpdateDataBase(void)
{
   CString str1,str2,str3,str4;
   m_input1.GetWindowTextW(str1);
   m_input2.GetWindowTextW(str2);
   m_input3.GetWindowTextW(str3);
   m_input4.GetWindowTextW(str4);
   data1.Add(str1);
   data2.Add(str2);
   data3.Add(str3);
   data4.Add(str4);
   data5.Add(GetCurrentDate());
   data6.Add(GetCurrentTime());
}


void CPROJECTCODEDlg::OnClose()
{
        // TODO: Add your message handler code here and/or call default
        //SaveDataLogged();
        CDialogEx::OnClose();
}
```

APPENDIX I: PROJECT CODE.cpp

```
PROJECT CODE.cpp : Defines the class behaviors for the application.
#include "stdafx.h"
#include "PROJECT CODE.h"
#include "PROJECT CODEDlg.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#endif
BEGIN_MESSAGE_MAP(CPROJECTCODEApp, CWinApp)
	ON_COMMAND(ID_HELP, &CWinApp::OnHelp)
END_MESSAGE_MAP()

CPROJECTCODEApp::CPROJECTCODEApp()
{

	m_dwRestartManagerSupportFlags = AFX_RESTART_MANAGER_SUPPORT_RESTART;
}

CPROJECTCODEApp theApp;


// CPROJECTCODEApp initialization

BOOL CPROJECTCODEApp::InitInstance()
{

	INITCOMMONCONTROLSEX InitCtrls;
	InitCtrls.dwSize = sizeof(InitCtrls);

	InitCtrls.dwICC = ICC_WIN95_CLASSES;
	InitCommonControlsEx(&InitCtrls);

	CWinApp::InitInstance();


	AfxEnableControlContainer();
	CShellManager *pShellManager = new CShellManager;

	// Standard initialization

	SetRegistryKey(_T("Local AppWizard-Generated Applications"));

	CPROJECTCODEDlg dlg;
	m_pMainWnd = &dlg;
	INT_PTR nResponse = dlg.DoModal();
	if (nResponse == IDOK)
	{

	}
	else if (nResponse == IDCANCEL)

	}
```

```
	if (pShellManager != NULL)
	{
		delete pShellManager;
	}

return FALSE;
}
```

APPENDIX J: SAVEDATA.cpp

This contains SaveData.cpp : implementation file

```
#include "stdafx.h"
#include "PROJECT CODE.h"
#include "SaveData.h"


// SaveData

SaveData::SaveData()
{
}

SaveData::~SaveData()
{
}


void SaveData::Serialize(CArchive&ar)
{
	CObject::Serialize(ar);
	data1.Serialize(ar);
	data2.Serialize(ar);
	data3.Serialize(ar);
	data4.Serialize(ar);
	data5.Serialize(ar);
	data6.Serialize(ar);
}
```

REFERENCES

[1]. Sakshat Virtual Labs,Characterize the temperature sensor(RTD).

[2]. Drew Gilliam,Temperature Sensors,Introduction to Mechatronics 19/03/2003

[3]. National Semiconductor, LM35/LM35A/LM35C/LM35CA/LM35D Precision Centigrade Temperature Sensors, 1994.

[4].Dictionary.com, humidity, in the American Heritage New Dictionary of Cultural Literacy, 3rd edition.Source location: Houghton Mifflin Company, 2005. http://dictionary.reference.com/browse/humidity.Accessed: March 05, 2014

[5].Summary Likoglu, Humidity Sensors.Internet:www.fatih.edu.tr/~hsagkol/…/Humidity% 20Sensors.pptx.Accessed: March 05, 2014

[6] Zhi Chen and Chi Lu Humidity Sensors: A Review of Materials and Mechanisms America Scientific Publishers Vol. 3, 274–295, 2005

[7] John J.Corcoran,Analog-to-Digital Converters Agilent Technologies, Palo Alto, California.

[8] Digital Engineering Library,ADC, McGrawHill, 2004. Internet: www.digitalengineeringlibrary.com,Accessed:March 15, 2014.

[9] Denver, Allen. “Serial Communications in Win32.” MSDN Online Library. 11 December 1995. <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwbgen/html/msdn_serial.asp>3 March 2014.

[10] Petzold, Charles. Programming Windows,5th edition,Redmond, WA: Microsoft Press, 1999.

[11] Rob Bayer ,Win32 Serial Communications.

[12] Dean Camera Using the USART in AVR-GCC , May 5, 2013