DATA PROCESSING THROUGH BIOSENSORS AND DEVELOPMENT OF SIMULATION SOFTWARE IN WINDOWS & RT-LINUX

Kazi Mohammed Razin; ID: 09221067
Jonayet Hossain; ID: 09221060
Mahmudul Hasan Oyon; ID: 09221214

Department of Electrical & Electronic Engineering

December, 2011

BRAC University, Dhaka, Bangladesh

DECLARATION

We hereby declare that the thesis titled "**Data Processing through Biosensors & Development of Simulation Software in Windows & RT-LINUX**" submitted to the Department of Electrical & Electronic Engineering (EEE) BRAC University, Dhaka, in partial fulfillment of the Bachelor of Science in Electrical &Electronic Engineering, is our original work and was not submitted elsewhere for the award of any other Degree or Diploma.

Dhaka, Date 15th December, 2011

SUPERVISOR

_____

Dr. AKM ABDUL MALEK AZAD
ASSOCIATE PROFESSOR
DEPARTMENT of
ELECTRICAL & ELECTRONIC ENGINEERING,
BRAC UNIVERSITY

KAZI MOHAMMED RAZIN

Student ID: 09221067

JONAYET HOSSAIN

Student ID: 09221060

MAHMUDUL HASAN OYON

Student ID: 09221214

Acknowledgement

We would like to express our sincere and firm gratitude and pay a lot thanks to our honorable thesis supervisor Dr. AKM Abdul Malek Azad, Associate Professor, Department of Electrical & Electronics Engineering for his constant supervision to carry out the thesis. He extended his helping hand by providing us encouragement, inspiration, facilities and valuable feedback throughout the course of this thesis. We would also like to thank Marzia Alam, Lecturer, and Department of Electrical & Electronics Engineering for giving us the direction of our work.

Abstract

Solving real life problems are often very time demanding due to the high sensitivity of the system such as biomedical devices which are actually used to get data to represent the critical state of a human or animal body. There are some important indicators of human health those include ECG, SPO_2, Temperature and Blood pressure. By monitoring those indicators the current state of any human health is possible to determine and hence necessary steps can be taken. Therefore the Real Time monitoring of the health indicators signals is very important in Biomedical Engineering. In our approach we have experimentally proved and did the signal processing of important biomedical signals using Windows and Linux OS. For Linux we have used Red hat enterprise Linux 4 and RT Linux which are soft real time and hard real time operating system respectively. The data acquisition system of the signals has been done through a data acquisition Card and interfacing of the system to computer has been done both in Windows and Linux. We did it to make a comparison between the time dependencies of signals in different operating system under multitasking environment. So our objectives are to make a Real time signal processing systems and software for biomedical devices as well as to compare the time delay of the measurement system under 2 operating systems. We approached it by choosing three health indicators signals which are ECG, SPO_2 and Temperature.

C O N T E N T S

Chapter 6 Electrocardiogram Signal monitoring system

Chapter 7 Pulse Oximetry Signal Monitoring system

Chapter 8 Experiment Techniques & Result

Chapter 9 Conclusion

Chapter 1

Introduction

1.1 Project overview

The goal of the project is to develop a 3 channel data acquisition system which receives the analog biomedical signals from human body/animal and sends these signals to computer through a data acquisition system. For data acquisition purpose we have used USB-4716 data acquisition card which is an updated product of Advantech Company where USB Hid communication has been used. The signal processing circuits filter, amplify, and cancel the noise of the analog signal (ASP) coming from the transducers of the bio sensors and send it to data acquisition system (DAQ) for digital signal processing. The signal processing circuit of each bio sensor is different due to its variety of data. For example ECG needs only to show the graphics of heart electrical activity where SPO_2 or oxygen saturation has a complex algorithm to determine. DAQ digitizes the signals and converts it into digital numeric value so that it can be manipulated by the computer. Firstly when the DAQ get analog signals it makes the signal to digital for computer manipulation and then make it analog again for software manipulation. The DAQ card we have used is a 16 bit system and 5V or 10V reference can be set so the resolution of the card is .07mV which is enough accurate for our development. After the DAQ card processing is done it comes to the computer. To make the DAQ card works with the computer system firstly we had to interface the DAQ card with the system. We did interfacing both in windows and Linux. We made comparison between windows and Linux signal processing. By understanding the sensitivity of the bio medical data we decided to use RT Linux which coexists with Linux and has a real time kernel that can give nanosecond precision

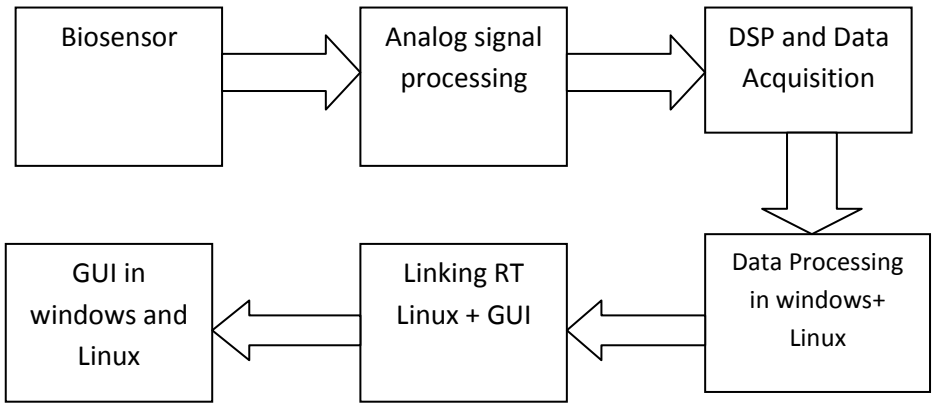


Figure 1: The flow of the project's I/O system.

The analysis of the system was necessary to gain the prime objective properly. By analyzing the system we get three major parts. The major parts of the systems and also the components of the parts have been briefly described below in the diagram.

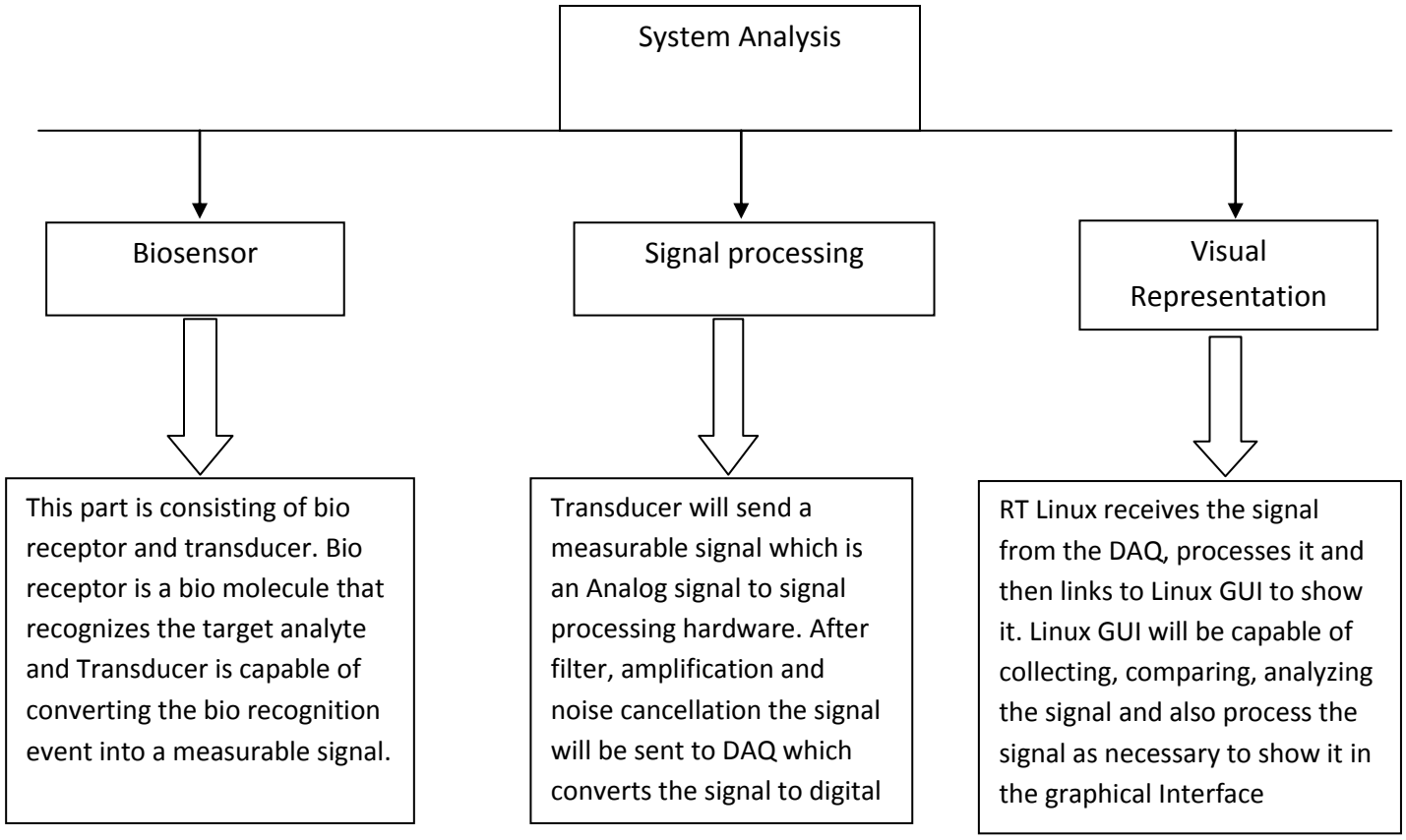


Figure 2: Analysis of the total system

1.2 Background and Motivation

Medical physics is an area of increasing importance in hospitals and in health-related occupations. Physics is used in medicine to diagnose illness and disorder and to design appropriate treatment and solutions. Biomedical engineering is a field of research of high importance in any sense around the world. Biomedical engineering is also a very sensitive field of engineering measurement where delay of a second can cause someone life's to death. So Real time computing has great importance in the field of biomedical engineering. There are always sensitive cases where it needs to follow up the pulse rate, blood pressure etc for every single moment. If one data is missed or cannot be processed in due time by the machine (biomedical instrument) it may cause serious impact on the patient's body. So real time patient monitoring or medicine testing equipment has great importance in healthcare. As we know Bangladesh is very good in pharmaceuticals research and development. But due to lack of medicine testing equipment they cannot prosper as expected. Cost effective patient monitoring system has also great importance for giving proper treatment to every

people of the country. A real time patient monitoring and medicine testing equipment will fulfill all the needs.

1.3 Selection of Biosensors

The biosensors we have selected for our project are mainly 3 sensors those are very necessary for health indication of any human body. Those are

- Temperature sensor
- Oxygen saturation
- Electrocardiogram

1.4 Comparison of different operating system

The part of the OS which handles all of the details of sharing and device handling is called the kernel or the core. The kernel is not something that can be used directly but its services can be accessed by system calls. What is needed is a user interface or command line interface that can be used by the user to make the use of the kernel. Since this is a layer of software it is called the shell around the kernel. So the kernel is the most important thing that actually decides the time sharing or priority sharing of any task that is given to the processor. If we compare windows with Linux we will get huge differences in this time or task sharing system. Windows is a single user, single tasking environment where Linux is multi user, multi tasking environment. There are different versions of windows and Linux which are definitely not have the same kind of kernel means their time sharing or task sharing capacity is not the same but generally we can call any version of Linux must have more multitasking ability than any version of windows. If we go a little detail we can see Linux has total real time kernel which is RT Linux or Real time Linux which is hard real time. Other than that any other Linux OS is known as soft real [6] time OS where windows are a general OS. If we think about the time dependency Linux has nanosecond precision where windows have millisecond. In our project we have tried to make signal processing both in windows and Linux to see the time dependence of the each system as well as to collect data about how important the time dependence when it comes to any sensitive issues like biomedical signals.

Windows	Linux
Non real time kernel	Soft and hard real time kernel
Single user single tasking environment	Multi user multi tasking environment
Priority or time sharing system	Round robin or real time system
Millisecond signal processing precision	Nanosecond signal processing precision
Not much time sensitive	Much more time sensitive
Suitable for general tasking and non sensitive signal processing	Suitable for multi tasking or real time signal processing

1.5 Objectives of the Present Work

The objectives of the present work is to Research about the time dependence of different OS in real life signal processing and its impact on biomedical signal processing which is very much sensitive in any sense. The objectives also include the development of software in windows and Linux environment for biomedical signal processing purpose. The objectives are given below,

- Interfacing the real life biomedical signals with windows and Linux OS and make a comparison of the signals processed by different OS.
- Development of signal processing software both in windows and Linux so that comparison can be precise and the software can be used for further signal processing research and development projects.
- Development of Data acquisition techniques for biomedical sensors.
- Establishing the technique of getting different biomedical signal such as Temparatue, ECG, and SPO_2 by making filtering, data acquisition and data representation.

1.6 Outline of the thesis paper

This paper describes the research technique and current development of the project in detail as possible. Chapter 2 describes the Data acquisition system. Chapter 3 and 4 describe the software development phases in windows and Linux. These chapters also describe the interfacing technique with data acquisition system and the software. Chapter 5, 6,7 describe three biomedical sensors measuring techniques which are temperature, ECG and Pulse oximeter. Chapter 8 provides the experiment techniques and result. Chapter 9 provides the achievements and limitations of the project as well as the future development plans.

Chapter 2

Data Acquisition System

2.1 Overview of the embedded system

The system is totally embedded. It consists of hardware and software part and both part are equally important in this project. Hardware part includes the Data Acquisition system and primary signal processing circuit. Biomedical sensors are also part of the hardware system [4]. Software system consists of operating system and developed software in windows and Linux. The contract between the hardware and software system is given below,

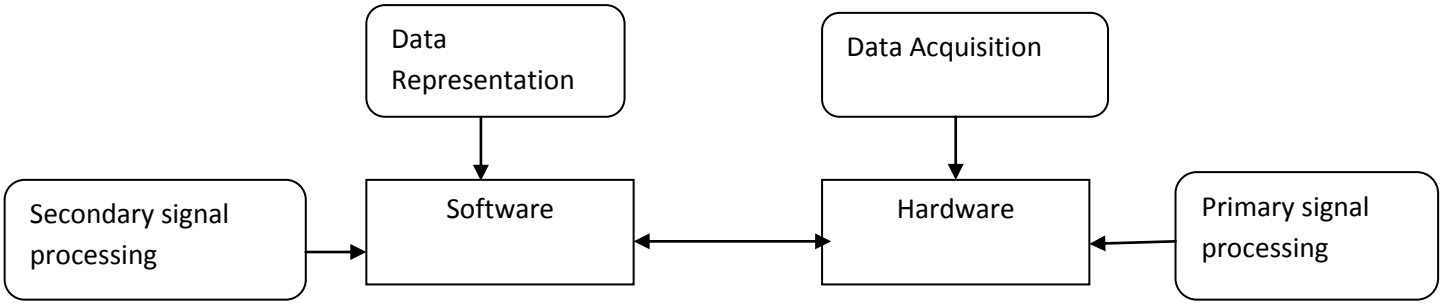


Figure 3: The Contract between hardware and software [17]

2.2 Role of DAQ card in the system

Traditionally, measurements are done on standalone instruments of various types- oscilloscopes, multi meters, counters etc. However, the need to record the measurements and process the collected data for visualization has become increasingly important. There are several ways in which the data can be exchanged between instruments and a computer. Many instruments have a serial port which can exchange data to and from a computer or another instrument. Another way to measure signals and transfer the data into a computer is by using a Data Acquisition board. A typical commercial DAQ card contains ADC and DAC that allows input and output of analog and digital signals in addition to digital input/output channels [7].

- Sampling:

The data is acquired by an ADC using a process called sampling. Sampling an analog signal involves taking a sample of the signal at discrete times. This rate at which the signal is sampled is known as sampling frequency. The sampling frequency determines the quality of the analog signal that is converted. Higher sampling frequency achieves better conversion of the analog signals. The minimum sampling frequency required to represent the signal should at least be twice the maximum frequency of the analog signal under test (this is called the Nyquist rate).

- ADC:

Once the signal has been sampled, we need to convert the analog samples into a digital code. This process is called analog to digital conversion. This is shown in Most boards also have a multiplexer that acts a like a switch between different channels and the ADC. Therefore with 1 ADC, it is possible to have a multichannel input DAQ board. The DAQ board we are using has 16 channel analog inputs. This makes it possible to acquire up to 16 analog signals in parallel (however, the sampling frequency will be divided by the number of parallel channels).

- Resolution:

Precision of the analog input signal converted into digital format is dependent upon the number of bits the ADC uses. The resolution of the converted signal is a function of the number of bits the ADC uses to represents the digital data. The higher the resolution, the higher the number of divisions the voltage range is broken into, and therefore, the smaller the detectable voltage changes. An 8 bit ADC gives 256 levels (2^8) compared to a 12 bit ADC that has 4096 levels (2^12). Hence, 12 bit ADC will be able to detect smaller increments of the input signals then a 8 bit ADC. If the full scale of the input signal is 10V than the LSB for a 3-bit ADC corresponds to 10/2^3=1.25V. That is not very good! However, for a 12 bit ADC the least significant bit will be 10/2^12=10/4096=2.44mV. If we need to detect smaller changes, one has to use a higher resolution ADC. Clearly, the resolution is an important characteristic of the DAQ board.

2.3 USB-4716 Specifications

The data acquisition board we are using for signal generation and communicating with computer is a DAQ from Advantech Company model no USB-4716. The USB-4716 is a true Plug & Play data acquisition device. No need to opening up the computer chassis just need to use the USB port for data acquisition. USB-4716 has 16 single-ended/ 8 differential inputs with 16-bit resolution, up to 200 kS/s throughput, 16 digital I/O lines and 1 user counter, add two 16-bit analog outputs [12]. It obtains all required power from the USB port, so no external power connection is ever required. The features of the USB-4716 are given below,

Main Features:

- Supports USB 2.0
- Portable
- Bus-powered
- 16 analog input channels
- 16-bit resolution AI
- Sampling rate up to 200 kS/s
- 8-ch DI/8-ch DO, 2-ch AO and one 32-bit counter
- Detachable screw terminal on modules
- Suitable for DIN-rail mounting

- One lockable USB cable for secure connection included

Analog Input (AI)

- 16-channel Single-Ended or 8-channel Differential A/D Input
- 16-bit A/D conversion
- Sampling rate form 1 Hz to 200 kHz
- Input Range (V): +/-10, +/-5, +/-2.5, +/-1.25, +/-0.625, 0~10V, 0~5V, 0~2.5V, 0~1.25V
- Automatic Channel/Gain Scanning

Analog Output (AO)

- 2-channel D/A Output
- Output Range with internal reference (V): 0~5, 0~10, +/-5, +/-10

Digital Input (DI)

- 8-channel Digital Input

Digital Output (DO)

- 8-channel Digital Output

Counter (C)

- Channel 0: 32 bits event counter with max to 1k input rate or 0.1HZ~10KHZ frequency measure.
    - Channel 1: 24 bits counter with 24M base clock provides pulse out function.

Figure 4: USB-4716

Chapter 3

Hardware Software Interface (Windows)

3.1 Role of Software in the system

The system consists of both hardware and software. Software is responsible for secondary signal processing and representation of the received data in a Graphical User Interface. Software does the critical part of the system by receiving data through the operating system and representing the data in a user friendly way. Software is also needed to compare data in different OS such as windows or Linux.

3.2 Interfacing Technique

3.2.1 Device driver

The interfacing is done between the DAQ system and the computer software. The data acquisition card USB-4716 provides us a device driver that can give different functionality of system. The device driver software named ActiveDAQ Pro gives us different function to use the DAQ system and represent the data. The functions primarily classified as two categories which are ActiveDAQ Pro device control and ActiveDAQ Pro GUI control. We have used the device control functions to manipulate the data coming through the DAQ card .We integrated the device control function to ourGraphical User Interface to control the data coming from the DAQ card [12]. The device control functions categories are given below

.dll functions	Descriptions of the functions
ADvAI	Analog Input Control
ADvAO	Analog Output Control
ADvDIO	Digital Input/output Control
ADvThermo	Thermocouple measurement Control
ADvCounter	Counter Input Control
ADvPulse	Pulse output control

Table 1: USB-4716 .dll functions

These are the .dll functions which are consist of several function that can control the device for specific purpose for example we need to receive analog signal from the sensors so we needed to use the ADvAI means analog input control .dll function to be integrated with our software. This AdvAI.dll consists of number of functions those are giving us the ability of controlling the analog input coming from the sensors.

3.2.2 Function and properties used

As we knew earlier we had to use AdvAI.dll which is actually a bunch of analog control based functions from which we have used few for our software purpose.

Type	Function +Properties	Description
Function	SelectDevice	Selects a device that supports AI (analog input) functions from the *installed Advantech* device list in the system.
Properties	DeviceNumber	Sets the device number for opening the specified AI device, or retrieves the device number of the current opened AI device.
Properties	DeviceName	Retrieves the device name corresponding to the DeviceName.
Properties	DataAnalog	Retrieves the sampling data (float) from the current AI channel ChannelNow on the DAS card.

Table 2 : Function and properties used for windows interfacing

3.3 Steps of interfacing DAQ and the software

Interfacing between the DAQ card and the software is most important part in the project. The card takes the data from the sensors and sends the data to the computer. The computer gets a digital data and software takes the responsibility for further processing of the data and shows it in specific manner. So at first the communication between the software and the card is very necessary. The steps of establishment of communication process are given below:

- Active DAQPro is the device driver software given by the manufacturer company Advantech. By installing the software we made sure that the windows will recognize the hardware. It just recognizes the hardware and does not make any communication with the developed software.
- Then we had to choose the necessary .dll functions needed to process the signal. We needed the analog signal processing function AdvAI. As we used C# language for the graphical user interface we added the specified functions in C# development environment as reference.
- Then we select a device by calling **selectdevice** function. It makes sure we are using the correct version of the product which is for us USB-4716.
- After that we selected device name and device number by using **devicename** and **devicenumber** properties

- Then Using **Dataanalog** properties to control analog input data coming from the sensors.

After getting the analog input data software processes it as needed then shows.

3.4 Software Development (Windows)

3.4.1 .Net framework

The .net framework is a great platform creating by Microsoft for developing applications for windows environment. An application developed by .net framework can be run in any version of windows. The advantages of using .net framework are lot. First of all we can develop the software in any language that supports .net such as visual basic++, c# etc. .Net contains CLR (common language runtime) that is capable of running any code supports by .net framework [23]. Another reason of using it is its drag and drop design option that is definitely an easier choice of any kind of software development. It has a gigantic library of code that we can use for different development. It also has a huge collection of device driver function. We have used C# language for our application development because it supports the object oriented programming module.

3.4.2 Options of the software

Now the software has the minimum options those are strongly required for preview the specific sensors data. The options are discussed below,

- Device select: this option is for selecting the right device for the software.
- Showing graph: It is necessary to show the graph of the data because graph can represent any critical state very easily. So we have an option of showing graph of the data in our software. In our software we have shown data in y axis and voltage is x axis.
- Respective voltage: Respective voltage is necessary because of the research purpose use. So in our software we have a option of showing the respective voltage of the processed data.
- Data: Main information those are shown. In our case it is temperature, ECG, SPO_2.
- Control: we have given a control option for controlling the data coming. It controls the flow of information and can stop or restart it.

3.4.3 Graphical user interface

Graphical user interface (GUI) is very much necessary to show our data in friendly and specific manners. GUI has been developed in visual studio 2010 by using C# language. The GUI represents the data those are obtained by the software. GUI has different options which were discussed earlier section. We can use those options by using the buttons given in the GUI. The code for the GUI software has been given in a appendix in the report.

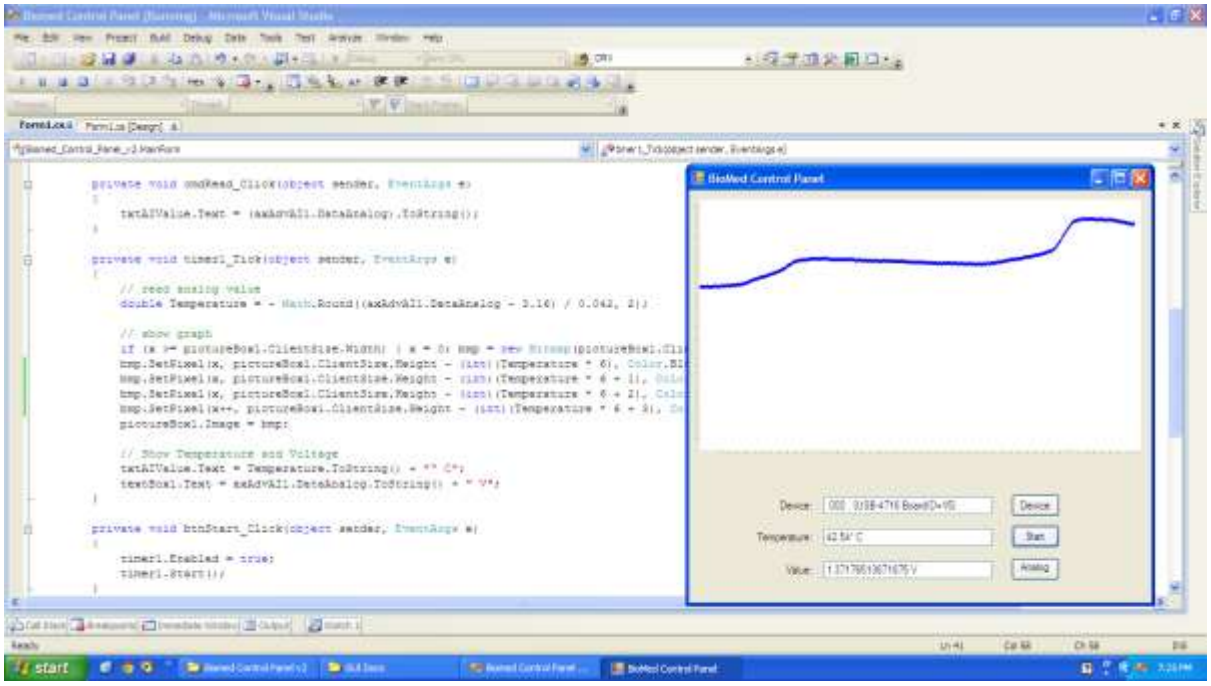


Figure 5: The Graphical user interface

3.4.4 Graph generating technique

Graph generation is necessary to show the data in friendly way. For graph generation we have used picture box class in visual studio 2010 [23]. Graph can be generated by using the class bmp which is within the picture box. The class named bmp (stand for bit map) works for setting different pixel level in the GUI. The function is bmp.setpixel(x, y, color). The parameters are x, y and color. Color is for show the graph in different color. Y is for showing the data and x is for showing the voltage. It is just similar to a traditional 2-dimensional system.

Figure 6: Graph generation of signal

Chapter 4

Hardware software Interface (Linux)

4.1 Why Linux

Linux have both soft real time and hard time kernel. General Linux is soft real time but there is hard real time Linux operating system which is called RT Linux. We used Linux because of its less time latency or delay than windows. Any Linux operating system is faster than windows and it has also some time latency where RT Linux time latency is nanosecond level which is negligible. Our data acquisition card gives us 3 Linux drivers which are

- Red hat enterprise Linux4

- Debian4
- Fedora core6

We have used Red Hat Linux4 (RHEL4) which is a soft real time operating system. RHEL is a commercial version of Linux which packaging format is rpm.

4.2 Interfacing Techniques for Linux

To interface the Linux with the hardware we needed to compile the program or driver named advdaq-1.09.0001-el4.i386 In RHEL4 operating system. By compiling the driver file we actually inserted a module for our OS RHEL4. The module can be inserted in code as below

- Insmod/usr/src/adddrv_core.ko
- Insmod/usb4716.ko

We have got 2 .ko type file. After this the process becomes a part of the Linux0S. Obtaining analog data from USB-4716 is consisting of two parts or steps in Linux.

Step 1 : binding the hardware with the software

Step 2 : obtaining analog data from hardware

Step 1 makes a hardware-hardware contract between our DAQ system and LinuxOS and step2 makes the hardware-software contract with the DAQ system with the Linux software. Both the steps are discussed below.

Step 1: to complete the step 1 successfully we needed to follow some Linux conventional coding and filing system. As we said earlier the first step is to insert the correct module to the OS so that there is a process for the hardware attached with the Linux OS [5].

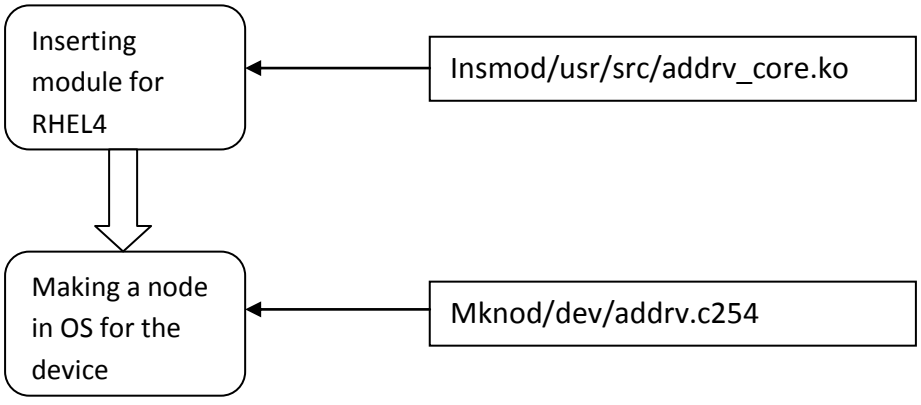
- Insmod/usr/src/addrv_core.ko
- Insmod/usb4716.ko

After inserting the module we will get a file in the process folder of Linux which is addrv file.

- /proc/device/addrev

Then we found out the major type of the file. The major type is 254 for our specified file. Using the major type of the file we can make a node with the USB hardware with the OS by using [13]

Mknod/dev/addrv.c254 . After making the node we bind the node of the OS to the hardware by using binding command advdevice_bind. After binding the device with the OS hardware-hardware contact part is done. Then the Linux OS is ready for getting analog data through the channels of the DAQ card. The binding procedures can be shown in flow chart in following way,



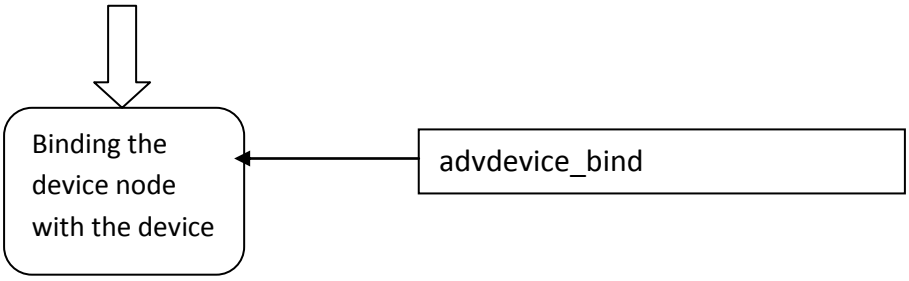


Figure 7: Procedures of Linux-DAQ communication

Step 2: After making the communication between Linux and DAQ we need to make communication with the process of Linux to obtain analog data through the hardware channels. Then actually we need to communicate with the device node that is made within the Linux OS. For obtaining analog data as input we used several functions those are given by the DAQ card manufacturer [4].

First we need to open the device to make it ready for starting data obtaining process. There is a function named DRV_DeviceOpen which actually makes the device ready for work. This function has two parameters where one is a utility of advdevice_bind and another is a pointer. Calling up the function makes sure if the device is successfully opened or not. It opens successfully it will return o otherwise will generate an error. After opening up the device we need to configure it for obtaining the analog data correctly. For obtaining the analog data we need to set up a specific channel and a gain code. This function also has two parameters where both of them are pointers. One is retrieved from DRV_DeviceOpen another is from PT_AIConfig. Last thing is to read analog data from the sensors via the DAQ. For this purpose we have used DRV_AIvoltageIn function which has two parameters where both are pointer. One is retrieved from DRV_DeviceOpen and another is from PT_AIConfig. List of functions those are used in process are given in the next page,

- DRV_DeviceOpen(filename, &fd)
- DRV_GetErrorMessage(ret, err_msg)
- DRV_DeviceSetProperty(fd, CFG_AiChanConfig, &buffer, sizeof(unsigned int))
- DRV_DeviceClose(&fd)

- DRV_AIConfig(fd, &AIConfig)
- DRV_AIVoltageIn(fd, &AIVoltageIn)

So the software-Hardware communication is based on 3 procedures those are given below,

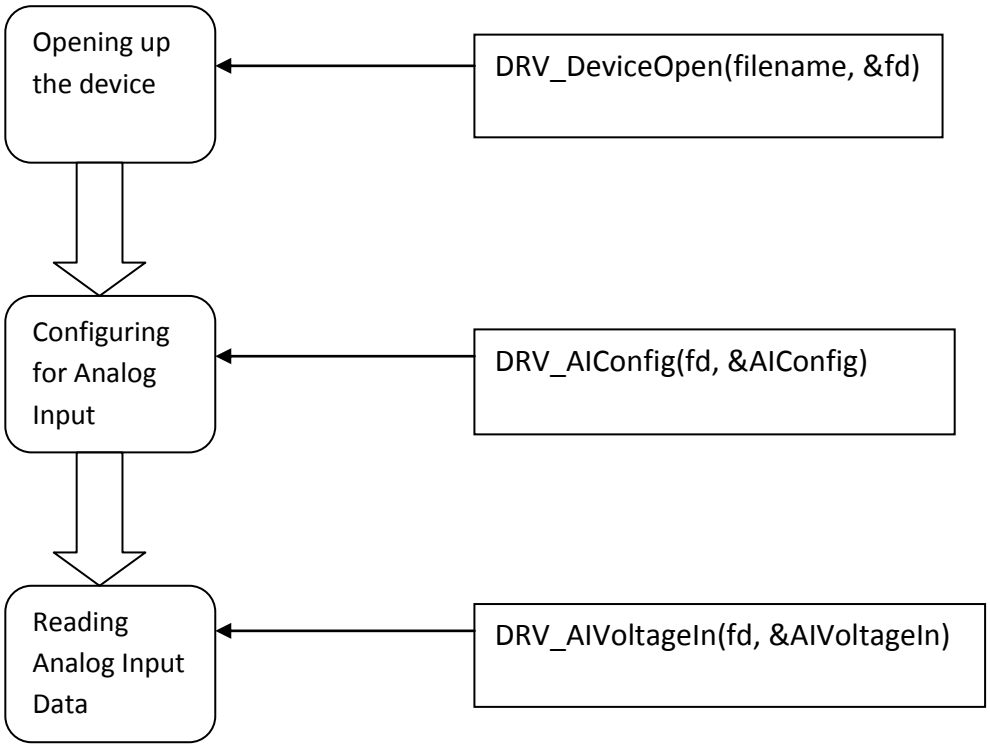


Figure 8: Procedures of Software-DAQ Communication

Chapter 5

Temperature Monitoring System

5.1 Importance of Temperature monitoring

Metabolic activities such as muscle action require energy. The energy expended by the human body to do work varies according activity. The average human muscle is no more than 25% efficient. Thus a person who is working at a rate of 1000w must be using energy at a rate of 4000w. The energy wasted in metabolic activities is converted to

thermal energy in the body and this energy must be dissipated as heat to the surroundings otherwise the body would overheat. the specific heat capacity of a body is 4200 J/kg*K (Same as water) and prove for yourself that the temperature of a 60 Kg person producing a thermal energy at a rate of 3000w would rise by almost 8 K in 10 minutes [20]. Such a temperature rise would be fatal. In fact the temperature of a healthy human body remains remarkably constant at between 36^0centigrade to 37^0centigrade. Clearly the body must automatically lose heat when undertaking vigorous activities. Equally heat loss must be prevented when the external temperature is very low otherwise the body temperature might become dangerously low. The blood vessels near the surface automatically become wider when the body is too hot, and become narrower when the body is too cold. In this way the flow of heat to the surface is adjusted to keep the temperature of the body is constant [21].

So it is clear that any human body always need to maintain a optimum level of temperature in the body. If the body is overheated or loss heat at a rapid rate it can be fatal for both a healthy and weak people. So temperature measurement is necessary when someone is taken under intensive care [19].

5.2 Determining biomedical temperature table

5.2.1 Determining technique

Temperature measurement can be done in some specific ways. These ways include the system of clinical thermometer and thermocouple thermometer. Measuring the temperature noninvasively is easier and better to accommodate with any situation. This is why we are using a noninvasive temperature sensor which will give some p.d (potential difference) according to the body temperature. After the digitization of the taken p.d the temperature of the human body will be shown in a workstation which will be in windows Linux environment. The measurement of temperature will be accommodated with three other measuring parameters of human body through the graphical user interface.

We used a biomedical temperature sensor which is used in vital sign monitor for giving precise temperature of human body. As biomedical temperature sensors are not used everywhere other than the biomedical sensitive purpose. It is tough to get a datasheet for the sensor. So before using this sensor in our signal processing system we needed to determine the data shit of it. As we tested we got it is a negative coefficient temperature sensor. The steps of using the sensor are given below sequentially:

- Design of the circuit for using the biomedical temperature sensor.
- Obtaining the value of the temperature versus voltage.
- Using the value finding an equation by using MATLAB.
- Using the equation in developed software.
- Making a comparison between the manual and the developed software data.

The circuit that was designed is consisting of a 22K resistor, 5V DC power, the temperature sensor and the data acquisition card.

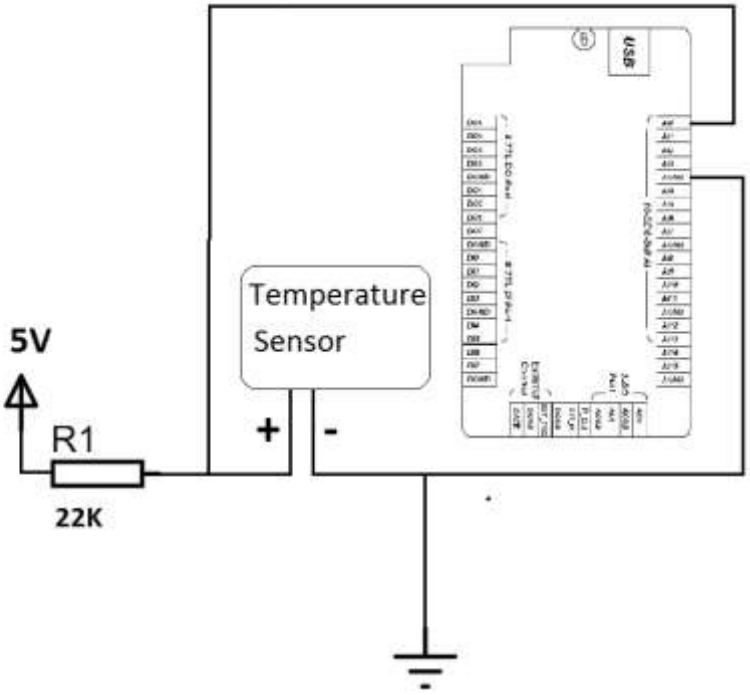


Figure 9: The circuit of temperature sensor

5.2.2 Obtained data table

The data table is obtained manually by changing the temperature and measuring the changing voltage with respect to the temperature.

Temperature (degree)	Respective Voltages
25-46	3.66,3.62,3.58,3.54,3.50,3.45,3.38,3.31,3.29,3.24,3.18,3.14,3.10,3.07,2.98,2.93, 2.87,2.83,2.81,2.78,2.74,2.70

By obtaining the temperature versus voltage table we used the values in the MatLab and used curve fitting method to obtain an equation for this. We need the equation to use in our developed software. We got the equation y=0.042x+2.2 where y is voltage and x is temperature.

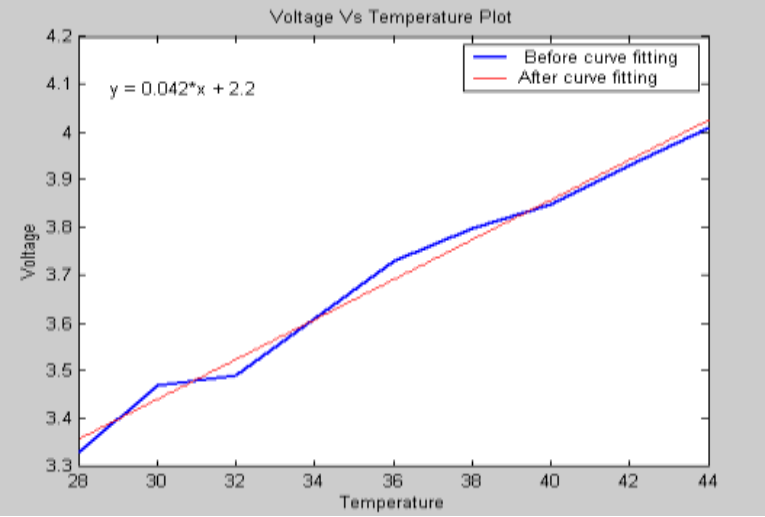


Figure 10: Curve fitting for temperature equation

We obtain the temperature x= y-2.2/0.042. We used this equation in the software to show temperature getting from the sensor.

5.3 Data Deployment on developed Software

After getting the equation x= y-2.2/0.042 where x= temperature and y=voltage we used the equation in our developed software to get voltage and temperature shown in Graphical user interface. Voltage and temperature is also shown in a graph. The temperature graph shown in the GUI has been given below

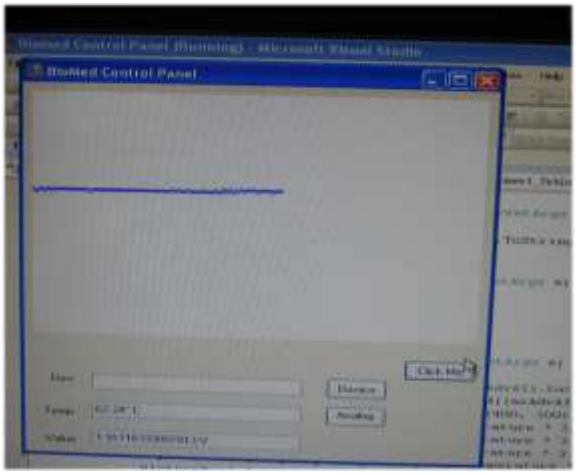


Figure 11: Temperature shown in GUI

By using the equation x= y-2.2/0.042 in our software we get some deviation. The deviation table is given below,

Temperature from	Temperature from

Thermometer	pc software
23	17.72
24	19.94
25	21.61
26	23.54
27	24.04
28	24.66
29	27.67
30	28.84
31	29.71
32	31.21
33	33

Table 3: temperature deviation

The deviation is not much and we got the equation which most perfectly suites with the temperature sensor.

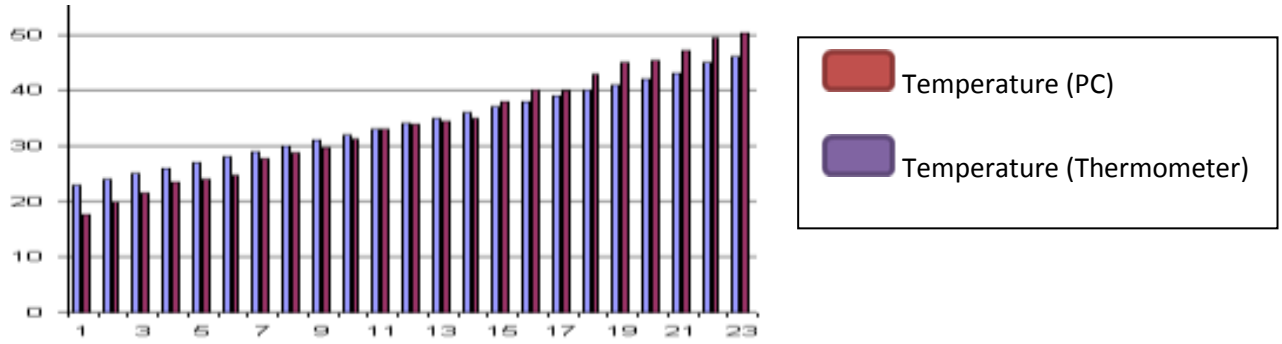


Figure 12: Deviation of manual versus software data

Chapter 6

Electrocardiogram Signal monitoring system

6.1 Working principle of ECG

The human heart beats at a normal rate of 70 times per minute without stopping from birth to death. It pumps about 5 liters of blood round the body every minute, using energy at a rate of just a few watts.Each time the heart beats, its electrical potential changes by more than 100mV. As the muscles of the heart contract and relax in a sequence that forces blood from the two atrial chambers of the heart into the corresponding ventricles and out into the artery. The sequence is due to nerve cells that conduct electrical signals generated at the sinu-atrial (SA) node in the right atrium. The nerve cells spread from the SA node across the surface of the heart, making the muscles of the atrial chambers contract and stimulating the atrio-ventricular (AV) node. This relays the electrical signals to further nerve cells which make the ventricles contract. Valves ensure the blood passes one way only through the heart from each atrial chamber [9].

The change of electrical potential at the heart is conducted through body fluids and tissue to the skin causing changes of potential of the order of 1mV which can be detected through

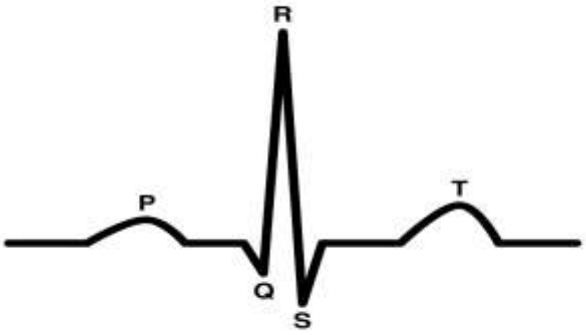


Figure 13: ECG Wave [10]

electrodes connected to an amplifier. An electrocardiogram is designed to measure and record the potential difference two points on the surface of the body. An ECG trace provides important information about the condition of the heart [1].
Figure shows how the potential of the heart changes each time the heart beats. The atria, making the atrial muscles contract then relax. The potential differences between any two points at the body surface due to conductivity of the body fluids. The electrodes are normally connected to two limbs with a third limb earthed.
A recording of such a p.d with time is called the electrocardiogram or ECG. The features of the above curve is given below,

1. The peak p.d is about 1mV, lasing about 0.1s [1].
2. The main features of the trace are labeled as P,Q,R,S and T according to convention
    - The wave at P is due to the atria depolarizing and contracting
    - QRS is due to the depolarization and contraction of the ventricles
    - The wave at T is due to repolarization and relaxation of the ventricles.

6.2 ECG Measuring technique

To measure and display and ECG waveform, the p.d between two points on the body surface must amplified from 1mV to about 1V enables it to be displayed on an oscilloscope. The steps of measuring an ECG signal from gaining the voltage by electrode to digitally display in a system are given below.

- Measurement: The electrical signals which command cardiac musculature can be detected on the surface of the skin. In theory one could grab the two leads of a standard volt meter, one with each hand, and see the voltage change as their heart beats, but the fluctuations are rapid and by the time these signals reach the skin they are extremely weak (a few millionths of a volt) [11] and difficult to detect with simple devices. Therefore, amplification is needed.
- Amplification**:** A simple way to amplify the electrical difference between two points is to use an operational amplifier, otherwise known as an op-amp. The gain (multiplication factor) of an op-amp is controlled by varying the resistors attached to it, and an op-amp with a gain of 1000 will take a 1 mV signal and amplify it to 1 V [9].
- Digitization**:** Unfortunately, the heart is not the only source of voltage on the skin. Radiation from a variety of things (computers, cell phones, lights, and especially the wiring in your walls) is absorbed by your skin and is measured with your ECG, in many cases masking your ECG in a sea of electrical noise. The traditional method of eliminating this noise is to use complicated analog circuitry, but since this noise has a characteristic, repeating, high-frequency wave pattern, it can be separated from the ECG (which is much slower in comparison) using digital signal processing system. Once amplified, the ECG signal along with a bunch of noise is in analog form. We now can display the output with an oscilloscope, but we need to load it into the Linux environment which will be linked with RT Linux for real time signal processing. The noise cancellation part will be done through the Data acquisition system.

Chapter 7

Pulse Oximetry Signal Monitoring system

7.1 Principle of Pulse oximetry

Oxygen saturation is defined as the ratio of Oxyhemoglobin to the total concentration of hemoglobin present in the blood (i.e. Oxyhemoglobin + reduced hemoglobin).

Hemoglobin is an iron-containing protein bound to red blood cells and makes up nearly all the oxygen presence (there is a minute amount dissolved in the plasma). Hemoglobin is responsible for transporting oxygen from lungs to other parts of the body, where the oxygen can be used by other cells. Oxyhemoglobin (HbO2) is the bright red hemoglobin that is a combination of hemoglobin and oxygen from the lungs.A hemoglobin molecule can carry a maximum of four oxygen molecules. 1000 hemoglobin molecules can carry a maximum of 4000 oxygen molecules; if they together were carrying 3600 oxygen molecules, then the oxygen saturation level would be (3600/4000)*100 or 90% [22].

When arterial Oxyhemoglobin saturation is measured by an arterial blood gas it is called SaO2. When arterial Oxyhemoglobin saturation is measured non-invasively by pulse oximetry, it is called SPO_2.

Saturation of peripheral oxygen (SPO_2) is an estimation of the oxygen saturation level usually measured with a pulse oximeter device. It can be calculated with the pulse oximetry according to the following formula:

$$S_{\mathrm{p}}O_2 = \frac{HbO_2}{HbO_2 + Hb}$$

7.2 Pulse oximeter device

A pulse oximeter is a device intended for the non-invasive measurement of arterial blood oxygen saturation and pulse rate. Typically it uses two LEDs (light-emitting diodes) generating red and infrared lights through a translucent part of the body. Bone, tissue, pigmentation, and venous vessels normally absorb a constant amount of light over time. Oxyhemoglobin and its deoxygenated form have significantly different absorption pattern. The arteriolar bed normally pulsates and absorbs variable amounts of light during systole and diastole, as blood volume increases and decreases. The ratio of light absorbed at systole and diastole is translated into an oxygen saturation measurement.

Chapter 8

Experiment Techniques & Result

8.1 Experiment overview

In this chapter we will explain the experiments we have done in the project, techniques used for the experiments and the result or significance of the experiments. Other than developing the data processing unit for biomedical device we have experiment some very important aspects of signal processing in different operating system environment. For our experiments we have used windows XP and Red hat enterprise Linux4 where one in a general operating system and another is soft real time operating system. We found out the latency of signal processing in both environments. In our signal processing purpose we have used one channel biomedical data which is biomedical temperature sensor data.

Latency (OS): Latency can generally be described as time delay experienced in a system. Every electronic system has a latency or time delay. Latencies may have different meaning in different system. One kind of latency is OS latency. OS latency differs from OS to OS. One kind of OS may have small time latency one may have a big time latency. For sensitive signal processing purpose the measurement of latency is very important because the delay of 1 micro second can bring significant changes in output level. So in research latency is very important. Biomedical engineering is a very sensitive engineering field because it is directly connected with life of human and animals. So the signal processing of biomedical device should have a minimum scale of time latency. The latency can be explained as below
Let T be a task belonging to a time sensitive application that requires execution at time t, let t' is the time at which T is actually scheduled. We define the OS latency experienced by T as L=t'-t.
There are several reasons behind the generation of Latency. The reasons include timer resolution, scheduling jitter etc.
Timer is generally implemented by using a periodic tick interrupt. A task that sleeps for an arbitrary amount of time can experience some timer resolution latency if its expected activation time is not in a tick boundary. Scheduling jitter happens when the task is not highest in the scheduling queue.
Latency cannot be removed totally due to some constraint on electronic and software part but it can be reduced. It really needs to be reduced when we go for sensitive biomedical signal processing. In our experiment we have found out the latency in windows and Linux for biomedical signal processing. The techniques of experiments and the result will be discussed in the later section.

8.2 Time measurement techniques

8.2.1 Windows technique

Measuring time in windows is necessary to find out the time latency of any specified process. Windows has a performance counter within it which counts the clock frequency of windows. If we want to find out the latency of any process we need to count the clock frequency in start time and count the clock frequency in the end time. Subtracting start time clock frequency from end time clock frequency we will get the number of frequency needed for the process. After getting the number of frequency we can easily calculate the time needed for the process by using the formula time =1/frequency. For this purpose we have

used the function QueryPerformanceCounter(&Value). Windows latency measurement tecnique flow chart for temparature signal is given below

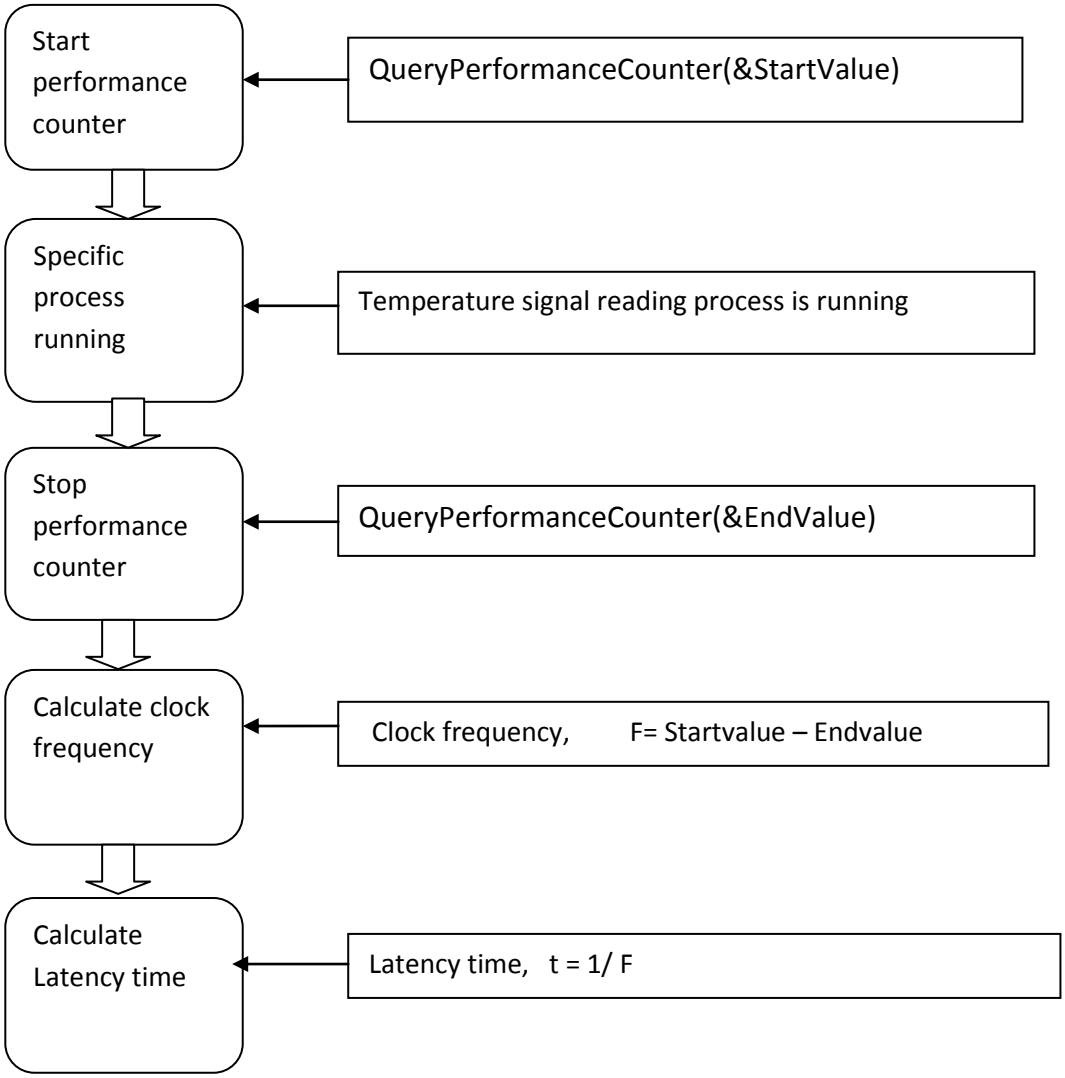


Figure 14: Time latency measurement steps in windows

8.2.2 Linux technique

To measure time latency in Linux environment we have used a function given by the Linux developer. The function is gettimeofday which is used to obtain the time of the day. The time data can be set to micro processor level. This process is simpler than windows because we don't need to calculate time from frequency rather we get time directly from the function. To use the function gettimeofday we need to include a header file include <sys/time.h>. The steps are given in the next page

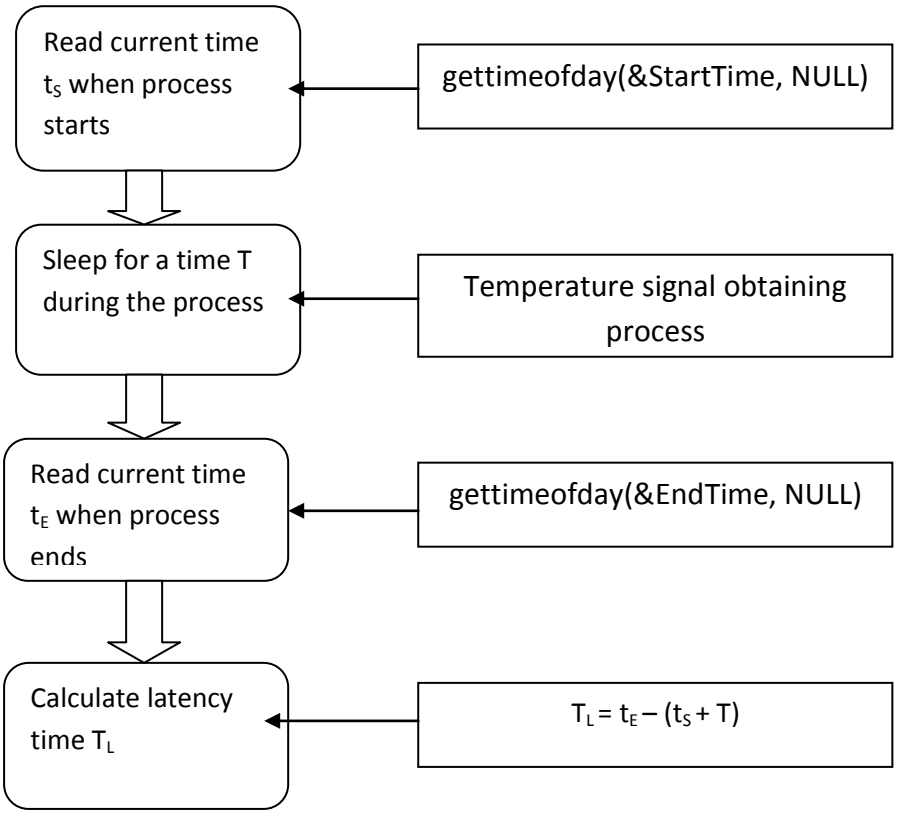


Figure 15: Time latency measurement procedures in Linux

8.3 Test results of comparison

We obtained time latency data for both in windows and Linux. Now we will compare between them. We have used biomedical temperature sensor for the signal processing purpose. We have taken 100 samples for each unit of average data for both in windows and Linux. We run the process for 5 times that means we take 500 data each in windows and Linux. The average time latency for both the systems are given in the next page,

Data: No process running condition

Sample No	Total time taken for 100 samples (windows) Microsecond	Average (windows) Microsecond	Total time taken for 100 samples (Linux) Microsecond	Average (Linux) Microsecond
1	39016	390	33274	332
2	38193	381	33077	330
3	36828	368	33172	331
4	37355	373	33271	332
5	38357	383	32921	329
	Average	**379**	**Average**	**332**

Table 4: Windows and Linux time latency comparison (No process running)

Data: 5 process running background

Sample No	Total time taken for 100 samples (windows) Microsecond	Average (windows) Microsecond	Total time taken for 100 samples (Linux) Microsecond	Average (Linux) Microsecond
1	40861	408	32801	328
2	40346	403	36885	368
3	41286	412	32755	327
4	40719	407	34475	344
5	40257	402	36558	365
	Average	**406**	**Average**	**346**

Table 5 : Windows and Linux time latency comparison (5 processes running)

5 Running processes are given below

No	Windows	Linux
1	Calculator	Calculator
2	Google crome	Internet browser
3	Paint	KPaint
4	Visual studio	Kdeveloper
5	Media player	Media Player

Data: 10 processes running background

Sample No	Total time taken for 100 samples (windows) Microsecond	Average (windows) Microsecond	Total time taken for 100 samples (Linux) Microsecond	Average (Linux) Microsecond
1	44000	440	40464	404
2	43656	436	39061	390
3	42963	429	42851	428
4	43590	435	42061	420
5	43052	430	41076	410
	Average	**434**	**Average**	**410**

Table 6 : Windows and Linux time latency comparison (10 processes running)

10 Running processes are given below

Windows	Linux
Calculator, google crome, paint, visual studio, media player, time and date, sound recorder, foxit reader, task manager,volume controller	Calculator,Internet browser, KPaint, Kdeveloper, Media Player,time and date, sound recorder, KPDF viewer,volume controller, task manager

Data: 15 processes running background

Sample No	Total time taken for 100 samples (windows) Microsecond	Average (windows) Microsecond	Total time taken for 100 samples (Linux) Microsecond	Average (Linux) Microsecond
1	50145	501	41446	414
2	54938	549	43608	436
3	48851	488	49070	490
4	56676	566	51667	516
5	53070	530	49191	491
	Average	**526**	**Average**	**469**

Table 7: Windows and Linux time latency comparison (15 processes running)

15 Running processes are given below

Windows	Linux
Calculator, google crome, paint, visual studio, media player, time and date, sound recorder, foxit reader, task manager,volume controller, User Account, notepad++, printer, display properties, character map	Calculator,Internet browser, KPaint, Kdeveloper, Media Playe,r,time and date, sound recorder, KPDF viewer, volume controller, task manager, character map, desktop properties, printing, emacs text editor, user manager

From the following data we can easily come to a decision about the time latency in windows and Linux OS. We see in the table that each unit which is average of 100 data taken in windows and Linux are in a range. For windows the range is 379-526 microseconds and in Linux 332-469 microseconds in a certain state of the processor. So the variation of signal processing time is greater in a window which is definitely a system drawback. Other than that we observed some time window take too much time which varied from 3 second to 10 second to read a data which is in general at maximum 469 microseconds. That refers to the uncertainty of windows signal processing purpose. From the following data we can make a table that will show the variation of data depending on the number of processes running on the system, table is given below

Running processes	Windows (microsecond)	Linux (microsecond)
No process running	379	332
5 processed running	406	346
10 processed running	434	410
15 processes running	526	469

Table 8 : Dependency on running processes

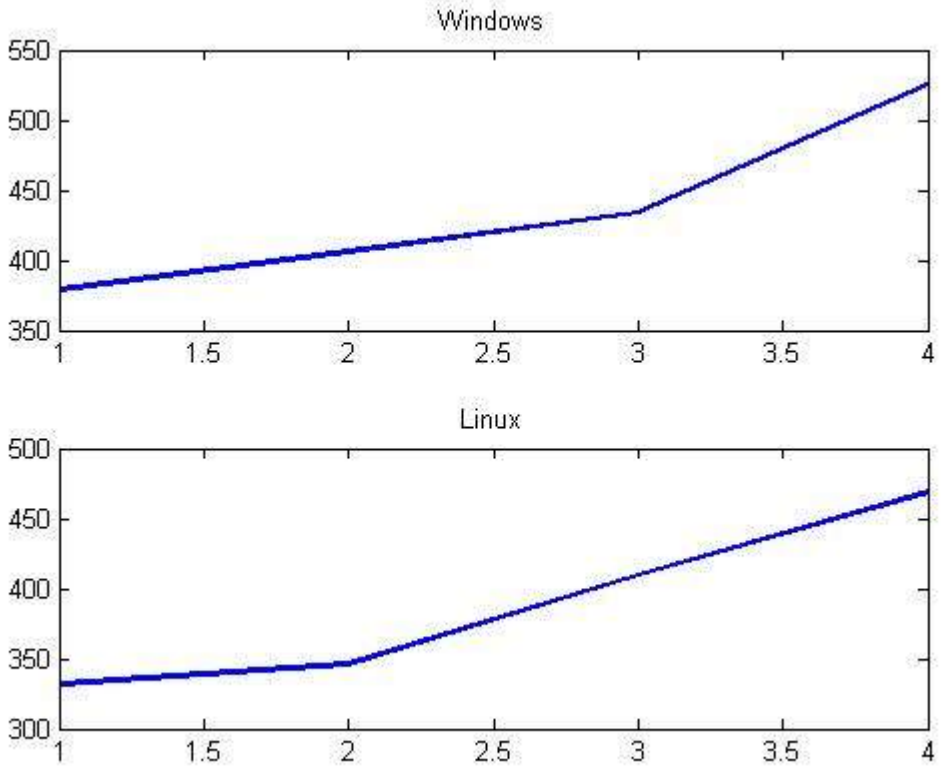


Figure 16: Windows and Linux time latency variation depending on running processes

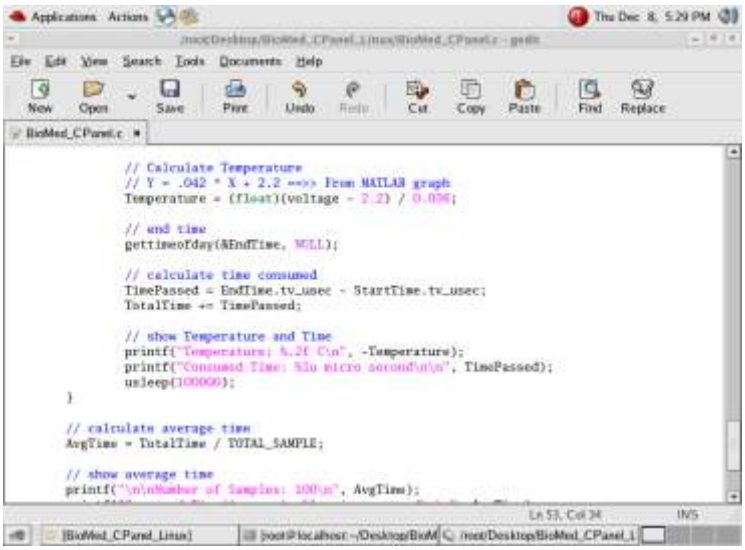


Figure 17: Linux code for time latency measurement

Chapter 9

Conclusion

9.1 Achievements of present work

Our present work has several achievements which we initially targeted. We could not achieve the total target due to time constraint. Our achievements in the present work are given below:

- Development of Data processing in Graphical user interface for biomedical device in windows environment
- Interfacing DAQ system both in windows and Linux
- Development of time measurement system for data reading through the DAQ card both in windows and Linux
- Hardware and software implementation for biomedical temperature monitoring system
- Making the time sensitivity comparison on signal processing between windows and Linux
- Hardware design of ECG and SPO_2 signal processing

9.2 Limitations of present work

The present work has few limitations. Limitations are given below

- We could interface the DAQ system only with one version of Linux which is RED hat enterprise Linux 4. In other versions time latency can be changed.
- The system is largely dependent on USB-4716. Any other data acquisition system can reduce or increase the precise level of data and time.

9.3 Decisions obtained from the research

This research significant due to some decisions obtained. We did signal processing both in windows and Linux. We have been come to a decision that the windows time latency is much higher than the Linux OS. That means windows in not suitable for sensitive signal processing purpose. But soft real time Linux also has latency. The difference of latency level is 100micro second which has great significance in biomedical signal processing. So Linux is definitely better in signal processing purpose. We got some other observations on this issue also. One of them is, due to huge number of process running under windows OS windows unexpectedly takes too much time for processing or reading a single data. That time ranges from 3000 micro second to 10000 micro second means 3-10 seconds. This is a major drawback of the windows in signal processing. Another thing we can see from the statistics is that the variation of signal reading time in windows is much higher than in Linux. For example in a single tasking environment windows has variation from 418-440 microsecond which is 323-327 microsecond in Linux Which is another drawback of windows system. So we can definitely tell Linux is better in signal processing than windows.

9.4 Future work

We will continue our research and development in this area. The future work we have targeted to do are given below

- Development of signal processing system for biomedical devices in Real Time Linux.
- Hardware- software interface implementation for ECG and SPO_2.
- Making time latency comparison between windows, Linux and Real time Linux.
- Development of a real time signal processing system for biomedical sensors which will include a graphical user interface for representation of real time data.
- Further improvement of out developed software in windows by adding more options and database in system.

References

[1] E.G. Zailis, M.H. Conover (1972). "Understanding Electrocardio Graphy" The C.V. Mosby Company, USA.

[2] R.S. Khandpur (2003). "Handbook of Biomedical Instrumentation." Tata McGraw Hill, New Delhi.

[3] S.P. Mohantay, and E. Kugojianas, “Biosensor a Tutorial Review”

[4] M.R. Neuman, “Biomedical sensor”

[5] http://en.wikipedia.org/wiki/RTLinux

[6] http://en.wikipedia.org/wiki/Real_time_computing

[7] http://en.wikipedia.org/wiki/Data_acquisition

[8] C. Saritha, V. Sukanya. “ECG signal analysis using wavelet transform”

[9] L. Sornmo ,P. Laguna “Electrocardiogram signal processing”

[10] G Kaur (2006) “Design and Development of Dual Channel ECG Simulator and Peak Detector.” Thapar Institute of Engineering and Technology, Deemed University, Patalia, June 2006.

[11] D Hussain (2002). “An Elecardiogram Simulator and Amplifier.” HST-6.121 Laboratory Report. November 2002.

[12] Advantech. “Active DAQ pro user guide.”

[13] M. Burgess (2002). A Short Introduction to Operating System.” December 29, 2002.

[14] M. B. Yehuda, (2005). “Instroduction to LINUX Device Drivers.” IBM Haifa Research Labs and Haifux. January, 2005.

[15] A .Rubini, J .Corbet and G.K. Hartman (2005). “LINUX Device Drivers.” O’Reilly USA, 2005.

[16] R.J.M. Theunissen, R.R.H.Schiffeders, D.A.V Beek and J.E. Rooda (2008). “Supervisory Control Snthesis for a patient support system.” Eindhoven University of Technology. December 2008.

[17] W.M. Wonham. Supervisory control of discrete-event systems. Dept. Elect. Comput. Eng., University Toronto, Canada 2007.

[18] C. Li, C. Zheng (1993) Proc. Annual Int. Conf. IEEE in Med. & Biol. Soc., San-Diego, California.

[19] http://www.omega.com/prodinfo/rtd.html

[20] http://www.ladyada.net/learn/sensors.html

[21] http:/www.temperatures.com/csensors.html

[22] H. Deni, D. M. Muratore, and R. A. Malkin, “Development of a Pulse

Oximeter Analyzer for the Developing World," Proc. of the 2005.

[23] http://www.msdn.microsoft.com/en-us/library/system.drawing.bitmap.aspx

Appendices

Appendix A

A.1 Code for the GUI in windows

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
```

```
namespace Biomed_Control_Panel_v2
{

    public partial class MainForm : Form
    {
        int x = 0, y = 0;
        Bitmap bmp = new Bitmap(1024, 768);

        public MainForm()
        {
            InitializeComponent();
        }

        private void cmdSelectDevice_Click(object sender, EventArgs e)
        {
            // selecting between the different model of usb daq devic (imp)
            axAdvAI1.SelectDevice();
            txtDeviceName.Text = axAdvAI1.DeviceName;
        }

        private void cmdRead_Click(object sender, EventArgs e)
        {
            txtAIValue.Text = (axAdvAI1.DataAnalog).ToString();
        }

        private void timer1_Tick(object sender, EventArgs e)
        {
            // read analog value
            double Temperature = - Math.Round((axAdvAI1.DataAnalog - 2.2) /
0.042, 2);

            // show graph
            if (x >= pictureBox1.ClientSize.Width) { x = 0; bmp = new
Bitmap(pictureBox1.ClientSize.Width, pictureBox1.ClientSize.Height); }
            bmp.SetPixel(x, pictureBox1.ClientSize.Height -
(int)(Temperature * 6), Color.Blue);
            bmp.SetPixel(x, pictureBox1.ClientSize.Height -
(int)(Temperature * 6 + 1), Color.Blue);
            bmp.SetPixel(x, pictureBox1.ClientSize.Height -
(int)(Temperature * 6 + 2), Color.Blue);
            bmp.SetPixel(x++, pictureBox1.ClientSize.Height -
(int)(Temperature * 6 + 3), Color.Blue);
            pictureBox1.Image = bmp;

            // Show Temperature and Voltage
            txtAIValue.Text = Temperature.ToString() + "° C";
            textBox1.Text = axAdvAI1.DataAnalog.ToString() + " V";
        }

        private void btnStart_Click(object sender, EventArgs e)
        {
            timer1.Enabled = true;
            timer1.Start();
        }
    }
 }
```

A.2 Code for time measurement in windows console

```
#include <windows.h>
#include <windef.h>
#include <stdio.h>
#include <conio.h>
#include "include\driver.h"

// define total number of sample
const int TOTAL_SAMPLE = 100;

/*****************************
 * Local function declaration *
 *****************************/
void ErrorHandler(DWORD dwErrCde);
void ErrorStop(long*, DWORD);

// time stamp
LARGE_INTEGER StartValue;
LARGE_INTEGER EndValue;
LARGE_INTEGER Frequency;
LARGE_INTEGER Interval;
double TempTime;
unsigned long TotalTime;
unsigned long ConsumedTime;

int main(int argc, char *argv[])
{
    DWORD  dwErrCde;
    ULONG  lDevNum;
    long   lDriverHandle;
    USHORT usChan;
    float  fVoltage;
      PT_AIVoltageIn ptAIVoltageIn;
    PT_AIConfig ptAIConfig;
      int i;
      float Temperature;
      long TotalTime;
      long AvgTime;

      //Step 1: Show Message
      printf("\n\n\nStart: BioMed Control Panel\n");
      printf("File: /dev/advdaq0\n");
      printf("Channel: 0\n");
      printf("Range: -+10 V\n\n");
      Sleep(1);

    //Step 2: Input parameters
      lDevNum = 0;
      usChan = 0;

    //Step 3: Open device
    dwErrCde = DRV_DeviceOpen(lDevNum, &lDriverHandle);
    if (dwErrCde != SUCCESS){return 0;}

    //Step 4: Config device
```

```
    ptAIConfig.DasChan = usChan;
    ptAIConfig.DasGain = 4;
    dwErrCde = DRV_AIConfig(lDriverHandle, &ptAIConfig);
    if (dwErrCde != SUCCESS){DRV_DeviceClose(&lDriverHandle); return 0;}

      // reset TotalTime
      TotalTime = 0;

      for(i = 0; i < TOTAL_SAMPLE; i++)
      {
            // start Time
            QueryPerformanceCounter(&StartValue);

            // Step 5: Read one data
            ptAIVoltageIn.chan = usChan;              // input channel
            ptAIVoltageIn.gain = ptAIConfig.DasGain;  // gain code: refer
to menual for voltage range
            ptAIVoltageIn.TrigMode = 0;               // 0: internal
trigger, 1: external trigger
            ptAIVoltageIn.voltage = &fVoltage;        // Voltage retrieved

            dwErrCde = DRV_AIVoltageIn(lDriverHandle, &ptAIVoltageIn);
            if (dwErrCde != SUCCESS){DRV_DeviceClose(&lDriverHandle);
return 0;}

            // Calculate Temperature
            // Y = .042 * X + 2.2 ==>> From MATLAB graph
            Temperature = (fVoltage - 2.2) / 0.042;

            // end time
            QueryPerformanceCounter(&EndValue);
            QueryPerformanceFrequency(&Frequency);


            // calculate time consumed
            Interval.QuadPart = EndValue.QuadPart - StartValue.QuadPart;
            TempTime = (double)Interval.QuadPart /
(double)Frequency.QuadPart;
            ConsumedTime = TempTime * 1000000;
            TotalTime += ConsumedTime;


            // show Temperature and Time
            printf("Temperature: %.2f C\n", -Temperature);
            printf("Voltage: %f V\n", fVoltage);
            printf("Consumed Time: %lu micro second\n\n", ConsumedTime);
            Sleep(100);
      }

    // Step 7: Close device
    dwErrCde = DRV_DeviceClose(&lDriverHandle);
      if (dwErrCde != SUCCESS){return 0;}

    // calculate average time
      AvgTime = TotalTime / TOTAL_SAMPLE;

      // show average time
      printf("\n\nNumber of Samples: %i\n", TOTAL_SAMPLE);
      printf("Consumed Time(Average): %lu micro second\n\n", AvgTime);
```

```
	return 0;
}//main
```

A.3 Code for time measurement in Linux Terminal

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <string.h>
#include <sys/mman.h>
#include <termios.h>
#include <signal.h>
#include <Advantech/advdevice.h>
#include <sys/time.h>

// define total number of sample
int TOTAL_SAMPLE = 100;

// time stamp
struct timeval StartTime;
struct timeval EndTime;


int main(int argc, char *argv[])
{
	PT_AIConfig AIConfig;
	PT_AIBinaryIn AIBinaryIn;
	PT_AIVoltageIn AIVoltageIn;
	PT_AIScale AIScale;
	unsigned short wdata;
	unsigned short channel;
	unsigned short gain;
	unsigned int buffer;
	float voltage = 0;
	char *filename = NULL;
	char err_msg[100];
	int ret;
	int fd;
	int i;
	float Temperature;
	ulong TimePassed;
	ulong TotalTime;
	ulong AvgTime;

	// initial settings
	filename = "/dev/advdaq0";
	channel = 0;
	gain = 4;

	// show message
	printf("\n\n\nStart: BioMed Control Panel\n");
	printf("File: /dev/advdaq0\n");
	printf("Channel: 0\n");
```

```
	printf("Range: -+10 V\n\n");
	sleep(1);


	/* Step 1: Open Device */
	ret = DRV_DeviceOpen(filename, &fd);
	if (ret) {
		DRV_GetErrorMessage(ret, err_msg);
		printf("err msg: %s\n", err_msg);
		return -1;
	}

	memset(&AIConfig, 0, sizeof(PT_AIConfig));
	memset(&AIBinaryIn, 0, sizeof(PT_AIBinaryIn));
	memset(&AIVoltageIn, 0, sizeof(PT_AIVoltageIn));


	/* Step 3: Set Single-end or Differential */
	buffer = 0x0000;  /* 0: single-end */
	ret = DRV_DeviceSetProperty(fd, CFG_AiChanConfig, &buffer,
sizeof(unsigned int));
	if (ret) {
		DRV_GetErrorMessage(ret, err_msg);
		printf("err msg: %s\n", err_msg);

		DRV_DeviceClose(&fd);
		return -1;
	}

	/* Step 2: Config AI Setting */
	AIConfig.DasChan = channel;
	AIConfig.DasGain = gain;

	ret = DRV_AIConfig(fd, &AIConfig);
	if (ret) {
		DRV_GetErrorMessage(ret, err_msg);
		printf("err msg: %s\n", err_msg);

		DRV_DeviceClose(&fd);
		return -1;
	}

	// reset TotalTime
	TotalTime = 0;


	/* Step 3: Start Single-channel AI */
	for(i = 0; i < TOTAL_SAMPLE; i++)
	{

		// massure start Time
		gettimeofday(&StartTime, NULL);

		/* Voltage In*/
		AIVoltageIn.chan = channel;
		AIVoltageIn.gain = gain;
		AIVoltageIn.TrigMode = 0;
		AIVoltageIn.voltage = &voltage;

		ret = DRV_AIVoltageIn(fd, &AIVoltageIn);
```

```
            if (ret) {
                  DRV_GetErrorMessage(ret, err_msg);
                  printf("err msg: %s\n", err_msg);

                  DRV_DeviceClose(&fd);
                  return -1;
            }

            // Calculate Temperature
            // Y = .042 * X + 2.2 ==>> From MATLAB graph
            Temperature = (voltage - 2.2) / 0.042;

            // end time
            gettimeofday(&EndTime, NULL);

            // calculate time consumed
            TimePassed = EndTime.tv_usec - StartTime.tv_usec;
            TotalTime += TimePassed;

            // show Temperature and Time
            printf("Temperature: %.2f C\n", -Temperature);
            printf("Voltage: %f V\n", voltage);
            printf("Consumed Time: %lu micro second\n\n", TimePassed);
            usleep(100000);
      }

      // calculate average time
      AvgTime = TotalTime / TOTAL_SAMPLE;

      // show average time
      printf("\n\nNumber of Samples: %i\n", TOTAL_SAMPLE);
      printf("Consumed Time(Average): %lu micro second\n\n", AvgTime);


      /* Step 4: Close Device */
      DRV_DeviceClose(&fd);   return 0;
}
```

A.4 MatLab code for temperature sensor equation

```
clear all;
 t=[25:1:46];
v=[3.66,3.62,3.58,3.54,3.50,3.45,3.38,3.31,3.29,3.24,3.18,3.14,3.10,3.07,2.98,2.93,2.87,2.83
,2.81,2.78,2.74,2.70];
figure(1)
plot(t,v, 'linewidth', 2)
 %stem(t,v, 'linewidth', 3)
title('Voltage Vs Temperature Plot')
```

A.5 MatLab code for windows-Linux comparison

```
s=[1,2,3,4];
L=[379,406,434,526];
figure(1)
```

```
subplot(2,1,1);
plot(s,L, 'linewidth', 2);
%stem(s,L, 'linewidth', 3)
title('Windows')
S=[1,2,3,4];
L=[332,346,410,469];
figure(1)
subplot(2,1,2);
plot(s,L, 'linewidth', 2)
 %stem(s,L, 'linewidth', 3)
title('Linux')
```

Appendix B

B.1 sample data set for windows time measurement

Voltage: 1.418762 V
Consumed Time: 432 micro second

Temperature: 21.34 C
Voltage: 1.431885 V
Consumed Time: 428 micro second

Temperature: 21.69 C
Voltage: 1.419067 V
Consumed Time: 432 micro second

Temperature: 21.32 C
Voltage: 1.432495 V
Consumed Time: 433 micro second

Temperature: 21.67 C
Voltage: 1.419983 V
Consumed Time: 418 micro second

Temperature: 21.31 C
Voltage: 1.432800 V
Consumed Time: 437 micro second

Temperature: 21.65 C
Voltage: 1.420593 V
Consumed Time: 424 micro second

Temperature: 21.29 C
Voltage: 1.433411 V
Consumed Time: 437 micro second

Temperature: 21.62 C
Voltage: 1.421814 V
Consumed Time: 431 micro second

Temperature: 21.34 C
Voltage: 1.431885 V
Consumed Time: 440 micro second

Temperature: 21.69 C
Voltage: 1.419067 V
Consumed Time: 437 micro second

B.2 sample data set for Linux time measurement

Start: BioMed Control Panel
File: /dev/advdaq0
Channel: 0
Range: -+10 V

Temperature: 26.39 C
Voltage: 1.249962 V
Consumed Time: 371 micro second

Temperature: 26.39 C
Voltage: 1.249962 V
Consumed Time: 332 micro second

Temperature: 26.39 C
Voltage: 1.249962 V
Consumed Time: 333 micro second

Temperature: 26.39 C
Voltage: 1.249962 V
Consumed Time: 334 micro second

Temperature: 26.39 C
Voltage: 1.249962 V
Consumed Time: 335 micro second

Temperature: 26.39 C
Voltage: 1.249962 V
Consumed Time: 337 micro second

Temperature: 26.39 C
Voltage: 1.249962 V
Consumed Time: 338 micro second

Temperature: 26.39 C
Voltage: 1.249962 V
Consumed Time: 340 micro second

Temperature: 26.39 C
Voltage: 1.249962 V
Consumed Time: 342 micro second

Temperature: 26.39
Voltage: 1.249962 VConsumed Time: 343 micro second

List of Figures

List of Tables