Time Delay Analysis of a Bio Signal Analyzer for Designing a Closed Loop Controller

A thesis submitted in partial fulfillment of the requirements for the degree of

M.Sc. in Electrical and Electronic Engineering

By

Marzia Alam

Supervisor

Dr. AKM Abdul Malek Azad
Associate Professor, EEE Department

Department of Electrical and Electronic Engineering,
BRAC University

January, 2013

APPROVAL

Title: Time Delay Analysis of a Bio Signal Analyzer for Designing a Closed Loop Controller

Author: Marzia Alam

Date: 24th January, 2013

Dr. AKM ABDUL MALEK AZAD

Supervisor

[illegible]

Signature

Prof. Dr. S. Shahnawaz Ahmed

Committee Member

[illegible]

Signature

Tareq Ahmed

Committee Member

[illegible]

Signature

Sadia Hamid Kazi

Committee Member

SHKazi

Signature

Md Sayeed Salem

Chairperson

[illegible]

Signature

DECLARATION

We do hereby declare that the thesis titled "Time Delay Analysis of a Bio Signal Analyzer for Designing a Closed Loop Controller", a thesis submitted to the Department of Electrical and Electronics Engineering of BRAC University in partial fulfillment of the Masters of Science in Electrical and Electronics Engineering. This is our original work and was not submitted elsewhere for the award of any other degree or any other publication.

Date: 24th January, 2013

[illegible]

Signature of the Supervisor

Dr. AKM Abdul Malek Azad
Associate Professor
Department of Electrical and Electronic Engineering
BRAC University

Marzia Alam
(Students Name)
Student ID: 10261004

[illegible]
Signature of Student

Abstract

Real time task for which timing must be guaranteed is most important requirement in biomedical measurement systems. If the system fails to meet the deadline it can end up with severe error for the measurement purpose. In this thesis, time delay is observed for an open loop biomedical system under different operating systems. To guarantee the response time of the biomedical data acquisition system hard real-time Linux operating system is proposed. The system is implemented both in Red Hat Linux and in Windows. In case of closed loop system this time delay degrades the performance of the system and cause the stability problem. The research given in this thesis aims to analyze time delay for closed loop Biosignal Analyzer, introduced by hardware and software in real time communication. Total time delay is calculated both theoretically and practically and the results are found to be quite similar. A controller needs to be designed to eliminate the stability problem caused by time delay. The mathematical model of the closed loop control systems in this regard is also developed for future work.

Table of Contents

List of Figures

List of Abbreviations

ASP:	Analog Signal Processor
A/D Converter:	Analog to Digital Converter
DAQ Card:	Data Acquisition Card
DMA:	Direct Memory Access
ECG:	Electrocardiographic
EEG:	Electroencephalographic
FIFO:	First in first out
FD:	Finite Dimensional
GUI:	Graphical User Interface
LTI:	Linear Time Invariant
RT-Linux:	Real-time Linux

List of Tables

Acknowledgements

I would like to thank my supervisor Dr. AKM Abdul Malek Azad for his continuous guidance and support for the completion of my thesis. I would like to acknowledge the EEE Department of BRAC University for funding my work. Special thanks to MD. Arif Khan, Lecturer, CSE Department, BRAC University, for his invaluable co-operation in understanding coding required for my thesis.

List of Publications

M. Alam, A. Azad *"Developing a Bio Signal Analyzer in Hard Real-Time Linux Environment"*, proceedings of International IEEE Conference on Biomedical Engineering, (ICOBE'12), February 27th -28th , 2012, Penang, Malaysia.

M. Alam, A. Azad, *"Time Delay Analaysis of Biomedical Signal Analyzer in Hard Real-Time Linux Environment"*, Proceedings of International IEEE Conference on Intelligent and Advanced System (ICIAS'2012), 12th -14th June, 2012, kuala Lumpur, Malaysia.

M. Alam, A. Azad, *"Implementation of 16 Channels Biosensor Based Animal Testing Data Acquisition System in Hard Real-Time Operating System"*, accepted to be appeared in International IEEE Conference on Sensor Technology (ICST'2012), to be held on 18th -21st December. 2012, Kolkata, India.

Chapter 1

INTRODUCTION

1.1 Motivation and Background

Biomedical engineering is the application of engineering principles and techniques to the medical field. The development of biomedical engineering is responsible for improving healthcare diagnosis, monitoring and therapy. It is 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 has great importance in healthcare.

Complexity arises when random delays arise into the control loop. System performance can be adversely affected by the presence of random access delays in the loop. In fact small or large delays and their associated phase lag provide unwanted oscillation in the system response, which may even lead to instability.

1.2 Literature Review

In order to pursuit real-time requirement for biomedical system queued management is proposed for several processes under multi tasking environment in paper [1]. Here a study of queued system based upon an operating system is presented for soft real time case study. To validate the proposed method two operating systems MAC and Linux is used. System Modeling and timing requirement is met by proper manipulation of two entities: Queue management and timing thread manipulation without modifying the operating scheduling algorithm. Python and Java language platform is used since it offers concurrent queues facility. The behavior of concurrent queue is evaluated in an application to monitor the signal of a Doppler ultrasound instrument for the measurement of the blood flow.

To meet the hard real time constrain a pre run time scheduling algorithm approach is implemented on Linux OS is proposed in this paper[2]. A method is presented for hard real-time system scheduling considering dynamic voltage scaling, precedence and exclusion relation. The system is modeled based on time petri nets in order to find a feasible schedule using a pre run time scheduling approach for time critical biomedical system. The proposed pre-runtime schedule synthesis is performed in two steps: preprocessing based on Yao's approach extended with discrete set of voltages. Proposed method was applied for measuring level of oxygen in human blood and was implemented on Linux platform.

In DJ Christini paper [3], it reports a software driven hard real time control system for electrophysiological optical mapping with a deterministic time delay <1 ms. Paper [4] proposes a time delay estimation approach for a sensor-actuator set up by linearizing the measurement equation in time which leads to an augmented system from which time delays and system states can be jointly estimated. Response delay between the input voltage and the pressure is measured for an automatic compression pump. The approach is based on a linearization in time of the measurement equation that leads to extended Kalman filter for an extended system.

A high resolution low power successive approximation ADC designing technique is proposed for biomedical signal detecting system in paper [5]. In order to detect low amplitude biomedical signal such as EEG, ECG high resolution circuit is designed and simulated using HSPICE. Based on the characteristics of biomedical signal several methods are used for improving ADC accuracy.

The acquisition system for physiologic signals in ICU (Intensive Care Unit) environment has also been reported in recent research. Most of them used a portable device that is connected to analog to digital board in a personal computer. In paper [6], Kevin et al developed the continuous physiologic data acquisition system (PDAS) for clinical research in the ICU environment.

An algorithm is proposed in paper [7] for stability analysis of a discrete time sampled data system. Stability analysis algorithm is proposed by showing robustness of sampled data system against perturbation caused by variation of sampling interval based on the small gain framework. Some direction for reducing conservatism is also discussed. The proposed algorithm measure the stability caused by the difference of sampling interval within the system using Lypnouv function.

In paper [8] propose an approach for analyzing stability of linear system. Conditions are derived to ensure asymptotic stability and to obtain an estimate of the convergence rate of the solutions. Example is given to show the efficiency of the method. Similar work is done to determine the stability criteria of asynchronous system in paper [9].

Many works have been done on biomedical data acquisition system so far. ADI instrument, the biggest biomedical equipment provider developed their data acquisition system on Windows and Mac platform but no such system is built keeping the hard real time (HRT) constrain. Here the work proposes HRT Linux based USB 4716 DAQ system for analyzing biomedical signal. Different model is built for this kind of system such as in paper [10-13]. In this thesis, discrete time based modeling for analyzing biomedical sampled data system is used.

1.3 Objectives

The prime objectives of this work are to analyze the time delay of the biomedical signal processing system. To achieve this goal first the system will be designed as open loop biomedical data acquisition system and it will be shown that computational delay can be reduced by introducing hard real time operating system. Then the effect of time delay in case of closed loop system will be evaluated which involves calculation of time delay both theoretically and practically. As the time delay in case of closed loop cause the system to fail to produce actuation signal and thus corrupts the system, therefore in close loop case only switching to hard real time operating system cannot solve the problem. So the necessity of designing controller arises. In this work the mathematical model for the closed loop system will be developed with an example of automatic heart pump to remove the stability problem caused by time delay.

1.4 System Overview

The system consists of biosensors and 16 channel data acquisition card which will receive the analog biomedical signals from human/animal body through the biosensor and these signals are sent to the computer. The signal processor (ASP) filter, amplify, and cancel the noise of the analog signal processor (ASP) developed from the transducers of the biosensors and send it to the data acquisition system (DAQ) for digital signal processing (DSP). DAQ digitize the signal and convert it into digital numeric value so that it can be manipulated by the computer. . Out of the 16 channels 5 channels data has been considered for the time being. These channels will be used for measuring ECG signal, cardiac output, blood pressure, respiratory rate, oxygen saturation, body temperature, heart rate. Overall system block diagram of the system is shown in Fig.1.

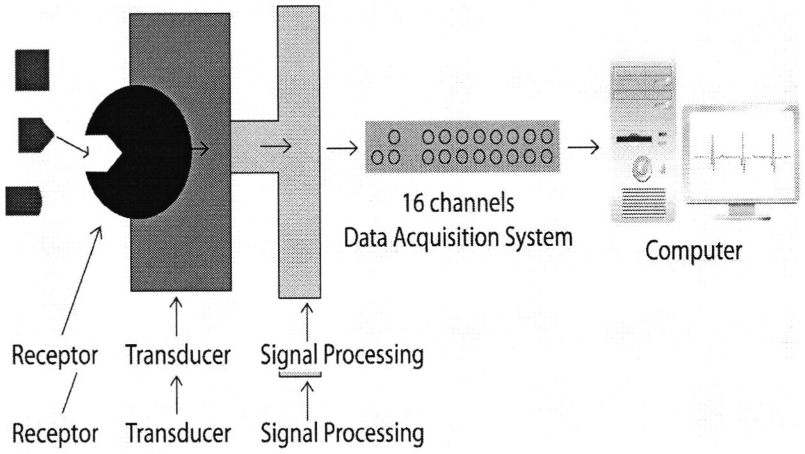


Figure1. 1 System Overview

To understand the source of delay, block diagram representation of the above system is given in Figure 1.2. Here the system is divided into different levels. The I/O part which consists of sensor and actuator is considered as top level L, the controller is the lower level, 0 and the data acquisition part is the higher level, 1.

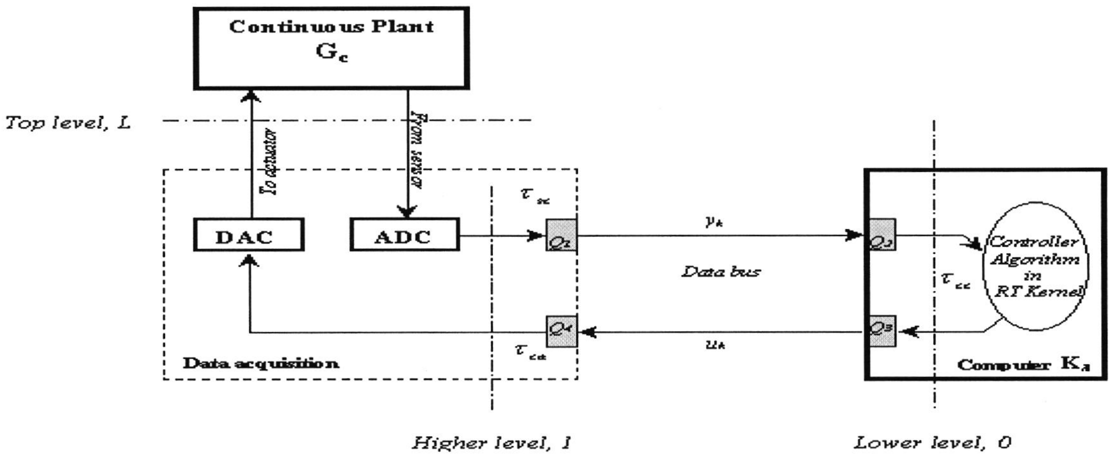


Figure1. 2 System Block Diagram

Now one delay is coming between the sensor and the controller which is τ_{SC} , another delay is coming between controller and actuator which is τ_{ca} . These two delays will be added with the controller delay τ_{cc} and the bus delay τ_{bus}, thus giving the total delay of the system.

$$\tau_{total} = \tau_{SC} + \tau_{CC} + \tau_{bus} + \tau_{ca}$$

Any of these delays can affect the total delay of the system threatening the system stability. In this thesis first the system is analyzed for the open loop case and it was found that time delay in open loop system can be reduced to some extent by using hard real time operating system. But if the closed loop controller for biosignal acquisition system is to be designed, the effect of delay cannot simply handled by the use of hard real time operating system. In close loop case delay can cause the stability problem which needs to be solved by appropriate design of a controller. In chapter two, overview of the whole system along with the experimental results done on open loop system is discussed. Chapter three represents the time delay analysis of the closed loop system and chapter four build up the mathematical modeling of the closed loop control systems required to reduce the time delay effect.

1.5 Overview of Contents

The rest of the dissertation is organized as follows:

Chapter 2: *Overview of Biosignal Processing System*

In this chapter, in section one, different operating systems and their characteristics are presented. The Real-Time RT-Linux solution is proposed. Also the performance characteristics of RT Linux are discussed. Section two presents the source and characteristics of different bio signals. Simulation output of the mentioned bio signals is shown. Section three, Biomedical Signal Acquisition System of the proposed system is discussed. The hardware structure of the data acquisition board, the device driver and the library functions used for the proposed system is discussed. Finally this section concludes by describing the interfacing technique between different operating systems. In experimental Results sections results on temperature sensor and ECG sensor is given. The results on temperature sensor are plotted in MATLAB and from the equation the algorithm was obtained to be used in computer program. A comparative result for windows, Linux and RT-Linux is shown for total delay calculation.

Chapter 3: *Time Delay Analysis of Sampled Data System*

The aim of this chapter is to study the time delay of the system. The considered biomedical data acquisition system is a sampled data system where time delay arises due to sampling. This chapter gives an overview of what are the sources of these delays and the effect of the delay on the system. Time delay analysis of the bio signal processing system is done. Probability of possible time delay is also discussed.

Chapter 4: *Mathematical Model of Sampled Data System*

To minimize the effect of time delay controller is needed. Before giving a complete form of any sampled data system it is important to design the controller such a way that can meet the system requirement. In this chapter the mathematical model of the controller is given with an example of automatic heart pump control system.

Chapter 5: *Conclusions and Future Work*

In this last chapter the main results of this dissertation is summarized, and some concluding remarks and identify potential directions for future research has been given.

Chapter 2

OVERVIEW OF BIO SIGNAL PROCESSING SYSTEM

2.1 Introduction

In this chapter a clear overview of biosignal processing system is discussed. This includes different types of operating systems that can be applied for implementing biosignal processing system, a brief overview of biosignals, biomedical signal acquisition systems and some experimental results done for implementing the system.

2.2 Operating Systems

Biomedical signal acquisition system can be either embedded or PC-based. In both implementations, a piece of software is capable of being utilized, knows as the operating system (OS). An operating system allows for numerous applications to be executed on a single workstation, instead of just one customized application. However, there are a variety of different types of operating systems that are available, each with their own strengths and characteristics. There are three types of operating system available: general purpose OS, soft real-time (SRT) OS and hard real-time (HRT) OS.

2.2.1 General Purpose OS

The most widely known operating system is the Personal Computer Operating System. The main job of this operating system is to provide a good interface to user. Microsoft Windows is the most widely known of this type of general purpose operating system which is non real-time. Moreover, there are operating systems for embedded systems. These operating systems do not provide the simple interface that PC operating system provide but, instead, are concerned with deterministic control over multiple routines [14,15].

Another type of operating system is the real-time system. In the following subsection two types of real-time OS-soft real-time OS (SRTOS) and hard real-time OS (HRTOS) will be described.

2.2.2 Soft Real-Time OS

In these types of system, missing an occasional deadline is acceptable. For instance, an example of a soft real-time system would be a live audio-video system. In this type of system, violation of timing constraints can result in degradation of audio and video quality but the system will still continue to operate. Example: Linux, UNIX.

2.2.3 Hard Real-Time OS

Another type of operating system is the hard real-time system. These systems are distinguished by having time as a key consideration. For instance, in control applications, real-time computers have hard deadlines that must be met. For example, in an assembly line at a car manufacturing plant, if a welding robot welds too late or too early then the car will be ruined. Hence, if an action absolutely must occur at a certain moment within a certain time interval, the system is deemed to be a hard real-time. Example: Real-Time Linux (RT-Linux), QNX etc [16].

2.3 Scheduling

When a computer is multi programmed, it frequently has multiple processes that are competing for the central processing unit (CPU) at the same time. If only one CPU is available in a system, a choice has to be made regarding which process is to be run next. The part of the operating system that makes the choice about which process is to be run is called the scheduler. Furthermore, the algorithm that the scheduler uses is called the scheduling algorithm. Information regarding scheduling procedure in two specific types of operating systems can be seen in the following sections.

2.3.1 Scheduling in Real-Time Systems

Hard real-time systems are characterized by having deadlines that must be made while soft real-time operating systems are characterized by having deadlines that should be met. Moreover, process scheduling in real-time systems must be highly predictable and regular. In real-time systems, priority scheduling is the type of scheduling algorithm that is implemented.

The basic idea of priority scheduling is that each process is assigned a priority. After a process is assigned a priority, the process with the highest priority will be allowed to execute. There are variations regarding how this priority scheduling algorithm is implemented. Most real-time operating systems utilize a non-preemptive priority scheduling algorithm. In this scheduling algorithm, the highest priority process continually runs until it voluntarily releases the CPU. Then, of the available processes, the process with the highest priority executes when the CPU becomes available. In the case when two processes have the same priority, one process is randomly scheduled to execute and other process has to wait for the process to finish execution before it can run.

2.3.2 Scheduling in Windows XP

The Windows operating system is currently the most widely used general purpose operating system. The Windows operating system does not have the same operational criteria as real-time operating system. As a general operating system, Windows is not predictable but instead attempts to allow for each available process to have a fair share of CPU time.

The latest version of Microsoft's operating system uses a preemptive thread based priority scheduling algorithm. While a process is traditionally perceived to only have one thread of execution, Windows allows for multiple threads to exist in the same process. Consequently, Windows XP has 32 priority levels, with 0 being the lowest and 31 being the highest. The highest 16 levels (15-31) are characterized as real-time levels. Although threads at these levels are characterized as real-time, the name is misleading. Threads running on these priority levels are not guaranteed to receive process time. On the contrary, threads running on the highest priority may receive no processors cycles because the processor may be busy handling hardware and software interrupts. Moreover, system level threads are not running on these levels. Consequently, setting an application to a real-time level may block a system task and cause system instability.

During execution, the difference between real-time levels and lower levels (called dynamic levels) is that the scheduler will never change the priority of a thread running on a real-time level. Processes running on lower levels will occasionally have their priorities boosted by the operating system to avoid process starvation. Therefore, before a thread is run on Windows XP, it must first wait for hardware and software interrupts to finish. After the interrupts finished, the waiting thread is finally allowed to execute, whenever an interrupts arrives or a higher priority thread becomes ready, the running process will become preempted and have to wait for the other interrupts or threads to execute. Consequently, the current kernel architecture for Windows XP is not ideal for high precision real-time applications.

2.4 The Linux Operating System

To understand Real-Time Linux, and why and how it is used, it is important to understand a little about the Linux Operating system. After all, RT-Linux is built, on or as an extension to, the Linux OS.

2.4.1 Linux and Real-Time

Linux is a general purpose, fully featured, free operating system based on UNIX (POSIX). Much of the supporting software is borrowed from UNIX, such as the graphical desktop environment, known as X-Windows, compilers, editors, free software GNU tools, etc. The Linux kernel is responsible for maintaining all the important abstractions of the operating systems, including virtual memory and process management. System libraries define standard functions which allow applications to interact with the kernel [17].

An important and useful feature of Linux is the ability to use *Loadable Kernel Modules*. These modules are compiled object code, and can be loaded and unloaded into the kernel on demand. This is an advantage for the reason that the kernel can be modified without the need to be constructed carefully and with security in mind.

Importantly, the Linux Kernel is non preemptable, as interrupts are disabled during its operation. So, kernel code cannot be interrupted while running. One consequence of this is that there is no need to protect critical sections of kernel code, as they cannot be interrupted.

Possible solution to enabling real-time performance in a Linux OS include, changing the Linux kernel to include preempt ability, low interrupt processing latency, and perhaps eliminating some of the functionality of the Linux kernel, that is, to strip it down and make it "lighter". Another alternative is to provide a real-time patch underneath the Linux kernel, where Linux is run as a low-priority process in a small real-time kernel. The real-time kernel takes over the real-time hardware from Linux and replaces it with software emulation. There are two commonly known real-time operating systems (RTOSs) which conform to this approach RT-Linux and Real-Time Application Interface (RTAI).

2.5 The RT-Linux Solution

RT-Linux is a patch for the standard Linux kernel. Due to it being available in a free version, it is particularly useful for teaching exercises, laboratory equipment; PCs used for instrumentation, and embedded systems. There are also commercial versions which include more features, and have, among other things, undergone more extensive treatment.

The following diagram illustrates (Figure 2.1) the structure of the real-time operating system. The diagram shows that Linux itself is treated just as another task to run, but with lowest priority. Linux in turn controls the running of its non real-time processes, such as editors, browsers, consoles, viewers, utilities, etc.

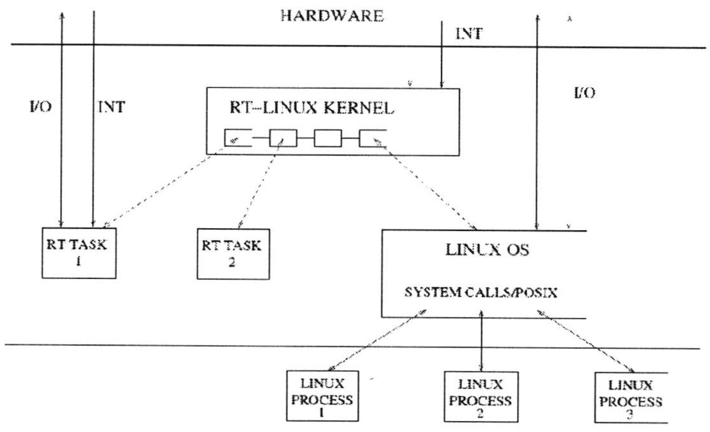


Figure 2.1: RT-Linux Kernel

The real-time requirements are met in RT-Linux by the real-time kernel capturing all hardware interrupts. These are checked for their importance, that is, time-critical (real-time) or non-time-critical (non-real-time). If an interrupt is time-critical and is destined for a real-time service routine, the appropriate routine is launched, otherwise if it is not time-critical, the real-time kernel forwards it to Linux in the form of a virtual interrupt (a software emulation), which is held until such time as there are no further critical real-time task to run.

Real-time tasks run at the kernel privilege level, giving them direct access to the computer resources, such as the CPU, memory, and hardware devices. Running at the kernel privilege level also gives the ability to change task priority, engage inter process communication (IPC), run user defined IPC handlers, and executes user defined scheduling algorithms. With privilege however, responsibility comes, care must be taken when constructing real-time programs so that the program does not make undesirable changes to the system that would otherwise not be possible without the privilege. Not only the real-time tasks run at the privilege level of the kernel, but they all exist and run within the same kernel address space. One consequence of this, aside from the security issue mentioned, is that switching between real-time tasks is made easier and quicker, and hence reducing latency.

2.6 Performance Characteristics

The fundamental criteria for evaluating performance of a real-time operating system are event latency and periodic jitter. An event can be an interrupt generated by hardware external to the CPU or it can be a signal internal to the operating system such as a notification to start the next task on a ready-to-run queue.

2.6.1 Event Latency

Events can be either hardware-generated interrupts or operating system generated software signals. Event latency is shown in Figure 2.2[17].

For an operating system generated event, the latency is the time from when the signal is generated to when the first instruction of the task is executed. This latency can be measured using the CPU's clock counter register. This 64-bit register is incremented at the clock rate, every 2 nanoseconds for a 500MHz CPU clock, and a sequence of its values can be stored in memory for later analysis. This measurement technique is not affected by any delays due to peripheral device bus accesses, because the counter register is internal to the CPU and is always incremented every clock cycle.

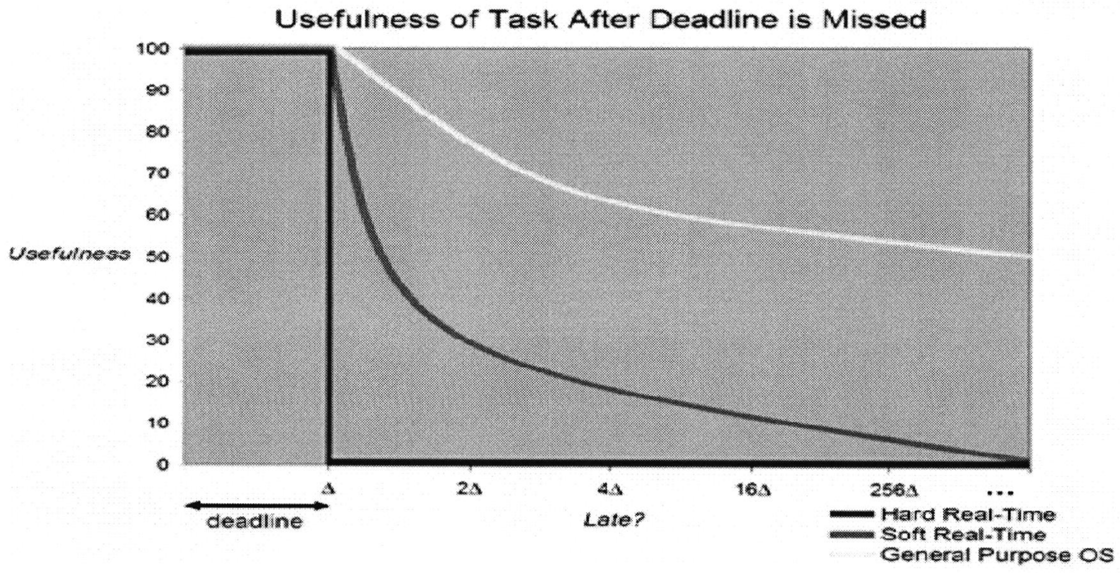


Figure 2.2: Event Latency

2.6.2 Periodic Jitter

Periodic jitter as shown in Figure 2.3 [17] refers to the variations in time that a repetitive task experiences as it executes.

The repetitive task is at the heart of sampled-data control of mechanical devices. The models of the device to be controlled are calculated on the assumption that the sample time is known and fixed. The control algorithm is, in turn, calculated from the device model, reinforcing the dependence on a known and stable sample time. Any jitter in the sample time leads to imprecision in the control-system performance.

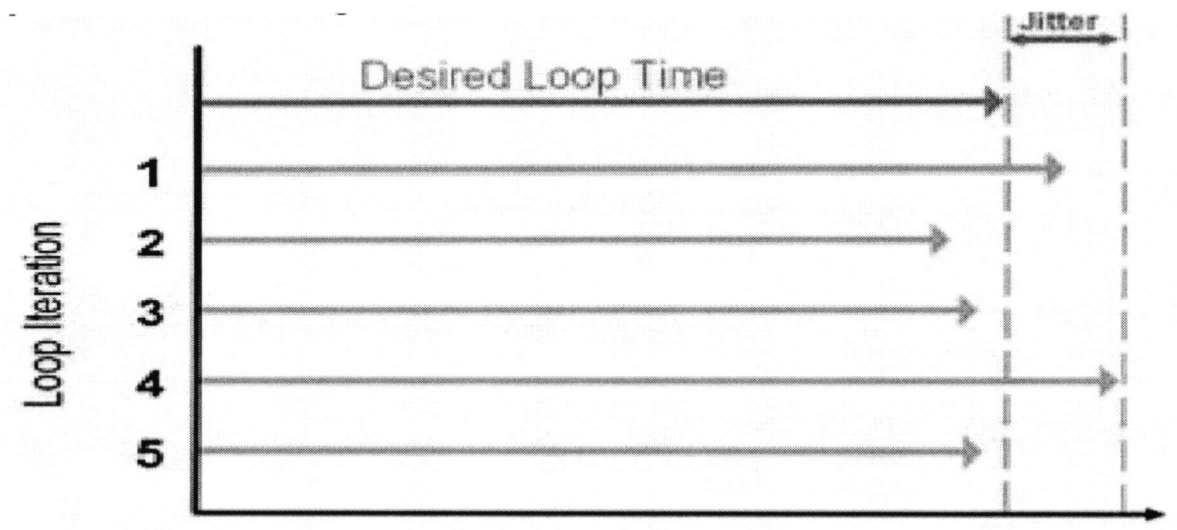


Figure 2.3: Periodic Jitter

Based on the discussion above it can be concluded that the faster response time, reduced jitter and multi tasking facility make the RT-Linux operating systems best suitable for time critical application such as for biomedical experiments. As my thesis work is concerned, the proposed system is the development of bio signal analyzer in hard real-time environment. In the following section different types of bio signals and their corresponding sensors required for the experiments will be discussed.

2.7 Biosignals

Biosignal is a summarizing term for all kinds of signals that can be (continually) measured and monitored from biological beings. Bio signals are used primarily for extracting information on a biological system under investigation. The process of extracting information could be as simple as feeling the pulse of a person on the wrist or as complex as analyzing the structure of internal soft tissues by an ultrasound scanner. In the following section different types of biosignal and their corresponding sensors are discussed.

2.8 Source and Characteristics of Different Biosignals

Biomedical signals [18,19] originate from a variety of sources such as: bioelectric signal, biomechanical signal, biochemical, biomagnetic, biooptical signal etc. Here brief ideas about these signals are given below.

Bioelectric Signals: These are unique to the biomedical system. They are generated by nerve cells and muscle cells. Their basic source is the cell membrane potential under which certain conditions may be excited to generate an action potential. The electric field is generated by the action of many cells constitutes the bio-electric signal. The most common examples of bioelectric signals are the ECG (electrocardiographic) and EEG (electroencephalographic) signals.

Biomechanical Signals: These signals originate from some mechanical function of the biological system. They include all types of motion and displacement signals, pressure and flow signals etc. The movement of the chest wall in accordance with the respiratory activity is an example of this type of signal.

Biochemical Signals: The signals which are obtained as a result of chemical measurement from the living tissue or from samples analyzed in the laboratory. The examples are measurement of partial pressure of carbon-dioxide (pco2), Partial pressure of oxygen (po2) and concentration of various irons in blood.

Biomagnetic Signals: Extremely weak magnetic fields are produced by various organs such as the brain, heart, and lungs. The measurement of these signals provides information which is not available in other types of bio-signals such as bio-electric signals. A typical example is that of magneto-encephalograph (MEG) signal from the brain.

2.9 Signals and Sensors

A biosensor [20, 21] consists of two elements: bio-element and sensor-element. Bioreceptor is the bio-element and transducers are: the sensor-element. The bioreceptor is a bimolecular object that recognizes the target analyte. It can be enzyme, antibody, tissue, etc. and the transducer should be capable of converting the bio recognition event into a measurable signal.

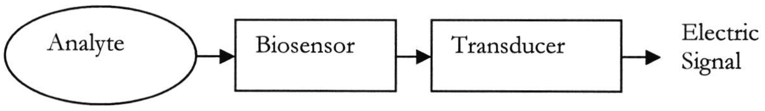


Figure. 2.4: Conversion of bio signal to electric signal

TABLE 2.1 DIFFERENT TYPES SIGNALS AND SENSORS

Signals	**Sensors**
Electrocardiogram	ECG Electrodes
Oxygen Saturation	Pulse Oximeter
Body Temperature	Thermal Probe
Aortic Pressure	Strain Gauge Sensor
Respiratory Rate	Impedence Pneumography Electrode

The output of different sensors for different biosignals is shown in Figure 2.5.

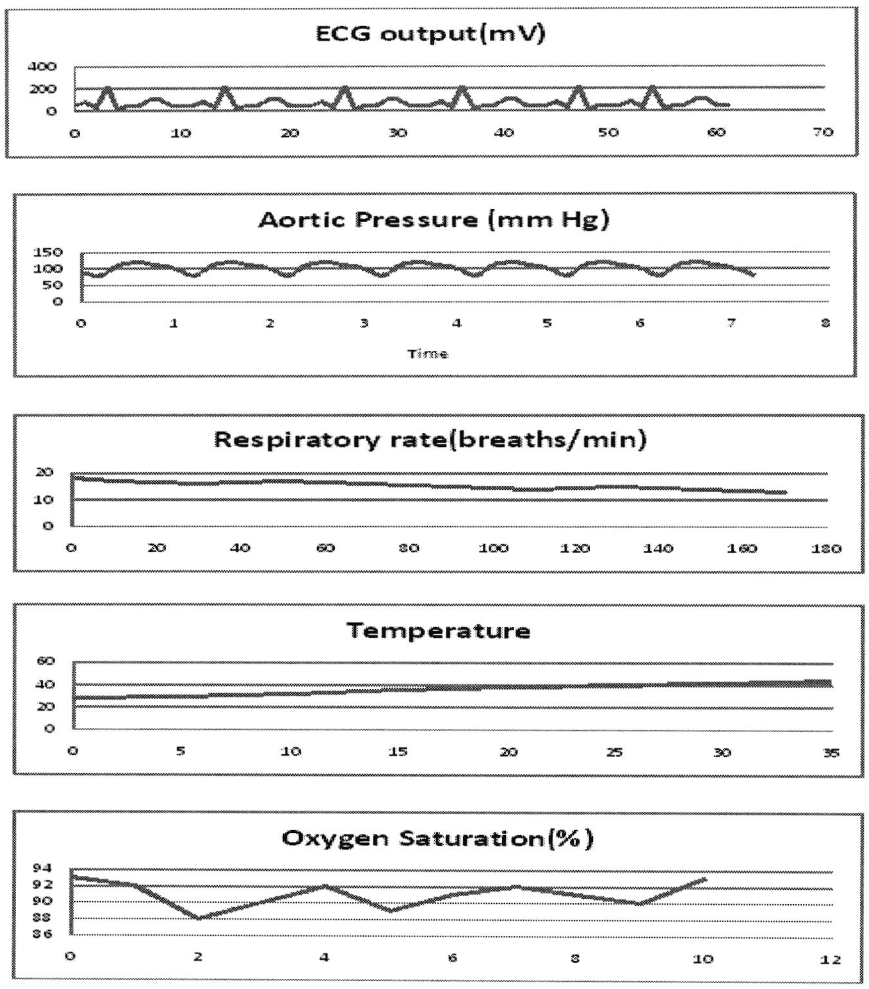


Figure 2.5: Simulated Output of Biosignal

For the time being five sensors are considered: ECG sensor to observe heart condition, pulse oximeter sensor to measure the oxygen saturation, temperature sensor to measure the body temperature and blood pressure monitoring sensor and respiratory rate to measure the breathing rate per minute. Other channels will be kept reserved for future implementation of more signals. Following section of the thesis give details about these five signals and their corresponding sensor and next section discuss the sensors and their corresponding signal conditioning circuit.

2.10 Electrocardiogram (ECG) Signal Monitoring System

Electrocardiogram (ECG) measures the electrical activity of the heart. Then heart is a muscular organ that beats in rhythm to pump the blood through the body. The heart muscles create electrical waves when they pump. These waves pass through the body and can be measured at electrodes attached to the skin. ECG signal bandwidth is .05Hz to 150 Hz and amplitude range is up to 10mV [22].

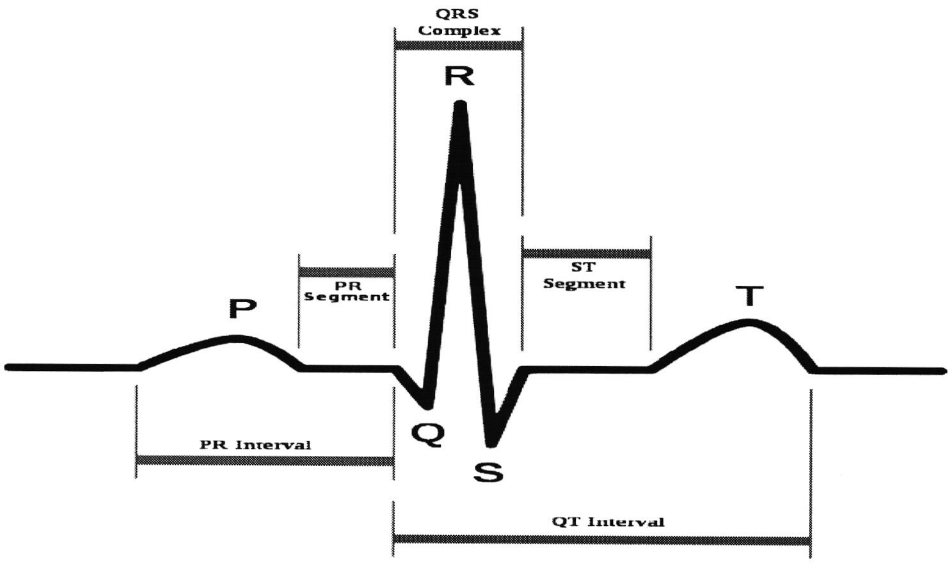


Figure 2.6: ECG Signal

2.10.1 ECG Leads

ECG signal is measured from electrodes applied to the surface of the body. The waveform of this signal is varied dependent on the placement of the electrodes. The term lead is used to indicate a particular group of electrodes. Electrodes are placed on each arm and leg, and six electrodes are placed at defined locations on the chest. These electrode leads are placed at defined location on the chest. These are two basic types of ECG leads: bipolar and unipolar. Bipolar leads utilize a single positive and a single negative electrode between which electrical potentials are measured. Unipolar leads (augment leads and chest leads) have single positive recording electrodes and utilize a combination of the other electrodes to serve as a composite negative electrode. Depending on the placing of electrodes there are three types of electrodes-Limb leads, Augmented leads, Chest leads.

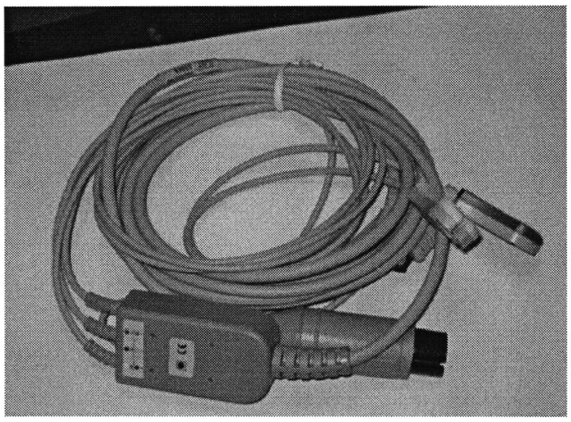


Figure 2.7: Three leads ECG

2.11 Pulse Oximetry Signal Monitoring System

The measurement of the oxygen in the blood is very important to analyze a patient medical condition. Pulse oximetry is used for this purpose. It is a non-invasive system to monitor the percentage of hemoglobin which is saturated with oxygen. It consists of a probe attached to the patient's finger or ear lobe. Acceptable normal ranges are from 95 to 100 percent although values down to 90% are common. In case of critical patient there may be risk of respiratory failure, so to know how well the arterial blood is oxygenated is very important for the patients. Pulse oximetry is also being used in the monitoring of pulmonary disease in adults and in the investigation of sleep disorders.

2.11.1 Principles of Pulse Oximetry

The principle of the pulse oximetry [23] is based on two physical principles. One is the presence of a pulsatile signal generated by arterial blood, which is relatively independent of non pulsatile arterial blood, venous and capillary blood and other tissues and the another one is the fact that oxy hemoglobin and reduced hemoglobin have different spectra. A Pulse Oximeter is shown in Figure 2.8.

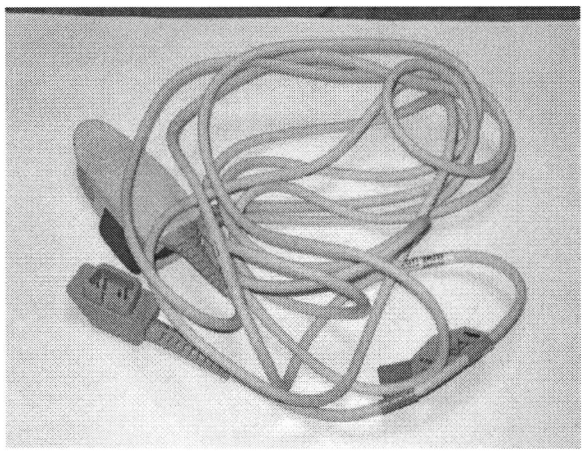


Figure 2.8: Spo_2 sensor

Pulse Oximeter is based on the red and infrared light absorption characteristics of oxygenated and deoxygenated hemoglobin. The light is partly absorbed by hemoglobin by amounts which differ depending on whether it is saturated or desaturated with oxygen. By calculating the absorption at the two wavelengths the processor can compute the proportion of hemoglobin which is oxygenated. Oxygenated hemoglobin absorbs more infrared light and allows more red lights to pass through. Deoxygenated (or reduced) hemoglobin absorbs more red light and allows more infrared light to pass through. Red light is in the 600-750 nm wavelength light band. Infrared light is in the 850-1000 nm wavelength light band.

2.12 Blood Pressure Measurement Technique

Blood pressure is the most often measured and most intensively studied parameter in medical and physiological practice [24]. All blood pressure measurements are made with reference to the atmospheric pressure. The frequency range for aortic pressure measurement signal is 0-60Hz. There are two basic methods for measuring blood pressure- direct and indirect.

The indirect method consists of simple equipment and cause very little discomfort to the subject but they are intermittent and less informative. They are based on the adjustment of a known external pressure equal to the vascular pressure so that the vessel just collapses. On the other hand, the direct method provides continuous and much more reliable information about the absolute vascular pressure from probes or transducers inserted directly into the blood stream. But the additional information is obtained at the cost of increased disturbance to the patient and complexity of the equipment.

The direct method of blood pressure measurement is considered for this system. For direct measurement a catheter tip probe can be used in which a sensor is mounted on the tip of the probe and the pressure exerted on it is converted into electrical signals.

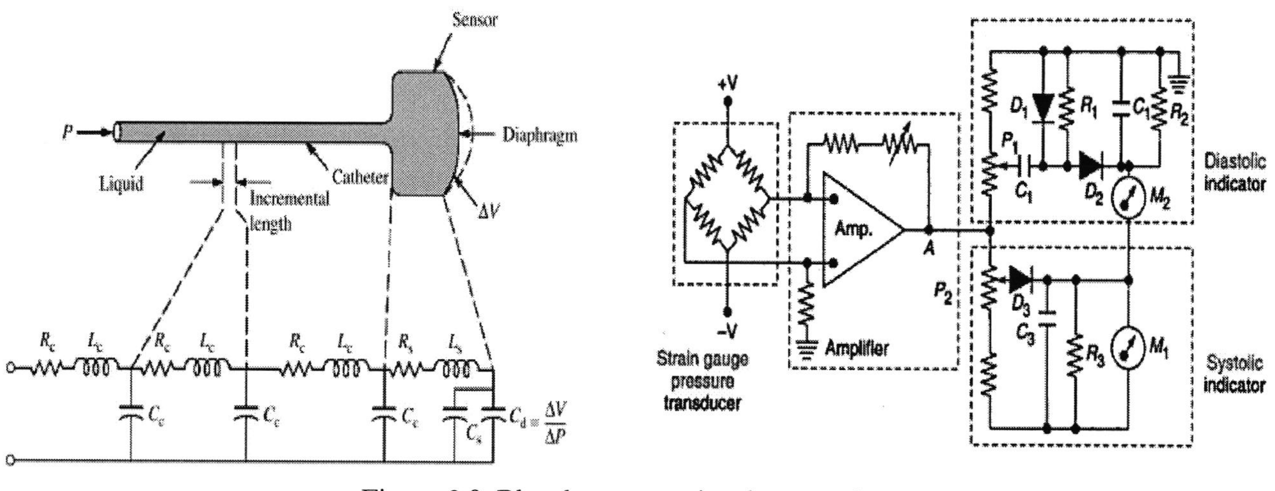


Figure 2.9: Blood pressure signal processing

Strain gauge transducer is used for processing the arterial pressure. The electrical signals corresponding to arterial pressure are amplified using an operational or carrier amplifier.

2.13 Temperature Measurement

Body temperature indicates the measure of the body ability to generate and get rid of heat. It is an indication of many types of illness. It can provide useful information about the severity of the illness [25, 26,27]. The truest, accessible, core temperature is measurement of the pulmonary artery temperature with a thermistor. Thermistor is a type of resistor with resistance varying according to its temperature. A thermistor temperature sensor is showed in Figure 2.10.

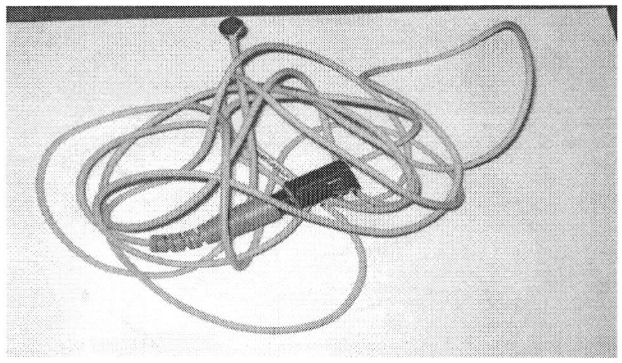


Figure 2.10: Temperature sensor

In this thesis above thermistor probe has been used to collect the data for temperature. It offers high precision thermistor elements, customizable probes and assemblies to provide precise and reliable temperature measurement in the most demanding applications. There are also some other type of temperature sensors such as catheter type thermistor, glass mercury thermometer.

2.14 Respiratory Rate Measurement

One of the options to measure respiratory rate is impedance pneumography. The objective of this technique is to measure the change of electrical impedance of the person's thorax caused by respiration. In this case oscillator output is applied to the two/four outer electrodes. The electrode used for this case is same as ECG electrode. The measuring range of the amplifier is .1 to 3 ohm with a frequency response of .2 to 3.0 Hz corresponding to respiratory rate of 12 to 180 per minute [28,29].

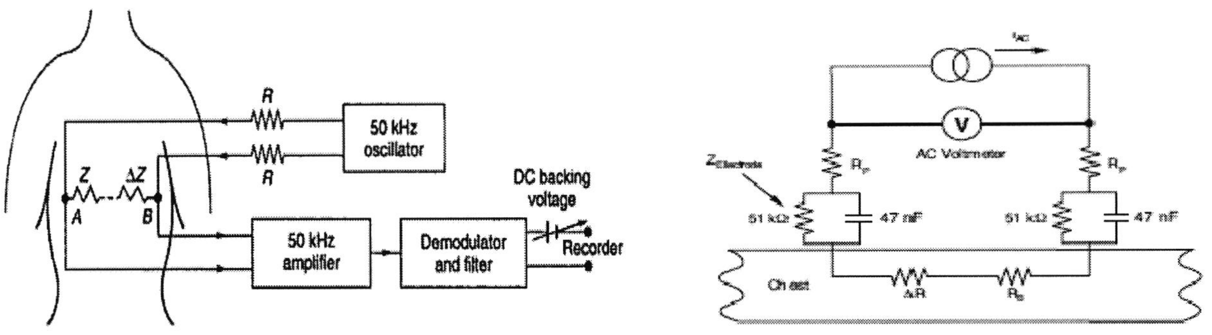


Figure 2.11: Respiratory Rate Signal Conditioner

In both case (either two/four electrodes) a high frequency ac current is applied into the tissue through the electrodes. The ac current produce a potential difference across the two points of the drive electrodes. The potential difference is related to the resistivity of the tissue between the voltage sensing electrodes.

In this section different bio signals and their corresponding sensors with their signal conditioning circuit are presented. The circuitry for temperature and ECG measurement will be discussed in the experimental result section. How these bio sensors are communicating with the outer world will be presented in next section.

2.15 Biomedical Signal Acquisition System

In order to use these biomedical signals in their various applications, a system must be available that can acquire, analyze, display these various signals. Although there are many design options available for creating such a workstation, two of the most common approaches are to either develop embedded or PC-based system. Here PC based biomedical system is considered where a data acquisition board USB-4716 is used for interfacing biosignals with computer.

2.16 Data Acquisition Board Fundamentals

A DAQ board is a basic A/D converter that is coupled with an interface that allows a PC to control the action of the A/D and capture the digital output information from the converter. A DAQ board is designed to plug directly into a personal computer's bus, with all the power required for the A/D converter and associated interface components being directly obtained from the bus. Moreover, it should be noted that a DAQ board is more than a simple A/D function on a board. A data acquisition may include discrete bi-directional I/O lines, counter timers, and D/A converters for outputting analog signals for control applications.

The purpose of a DAQ board is to convert analog data to digital data that a computer is able to manipulate. There are three primary methods available to transfer digitized data between the DAQ board and computer memory. These three methods for data transfer are direct memory access (DMA), interrupt, and programmed I/O transfers. For programmed I/O transfers, data are transferred between the CPU and the PC whenever the CPU receives a software code to either acquire or generate a single data point. Interrupt data transfer occurs when the DAQ board sends an interrupt to the CPU. This interrupt causes the CPU to either read acquired data from the DAQ board or write data to the DAQ board. DMA transfers use a DMA controller instead of the CPU to move acquires data between the board and computer memory. Even though high-speed data transfer can occur with interrupt and programmed I/O transfers, they require the use of the CPU to transfer data, Consequently, DMA transfer are able to acquire data at high speeds and keep the CPU free for performing other task at the same time [30].

2.16.1 Hardware Overview

The data acquisition board is been used for signal generation and communicating with computer is a DAQ from Advantech Company model number USB-4716. The USB-4716 is a true Plug & Play portable data acquisition device. USB-4716 has 16 single-ended/8 differential inputs with 16-bit resolution and a 200 kHz maximum sampling rate. Each individual input channel is software selected. However while there are multiple analog input channels the USB-4716 only contains one A/D converter. Consequently, when more than one channel is being utilized to acquire data, the sampling rate for each channel is defined as: (Sampling Frequency)/ (Number of Channel used). Hence, when all 16 differential analog input channels are used, the maximum sampling rate for each channel is 12.5 kHz.

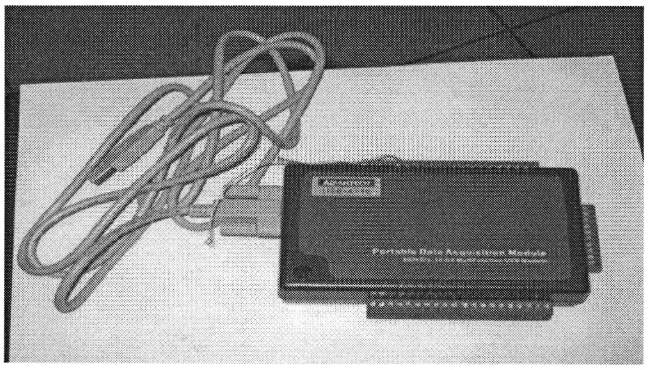


Figure 2.12: USB-4716 Data Acquisition Card

Acquired data are stored in a buffer located on the DAQ board. Data are then transferred, using the USB bus of the computer, via the direct memory access (DMA). Data are transferred from the FIFO to the computer once the amount of samples stored in the buffer reaches a specific value, which is known as count value. The USB-4716 also has additional functionality besides acquiring analog signals. The DAQ board contains two 16 bits analog output channels. Analog output is also stored in a buffer. Although digital input and output signals are not used during the course of the thesis project, it should be noted that the board contain 8 digital input/output channels.

2.16.2 USB-4716 Device Driver

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 for using the DAQ system and representing the data. The functions primarily classified as two categories which are ActiveDAQ Pro device control and ActiveDAQ Pro GUI control. The device control functions are used to manipulate the data coming through the DAQ card. The device control functions were integrated to Graphical User Interface (GUI) to control the data coming from the DAQ card. The device control functions are ADvAI.dll (analog input control), ADvAO.dll (Analog output control), ADvDIO (Digital input output control).

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

2.17 Interfacing Technique for Windows

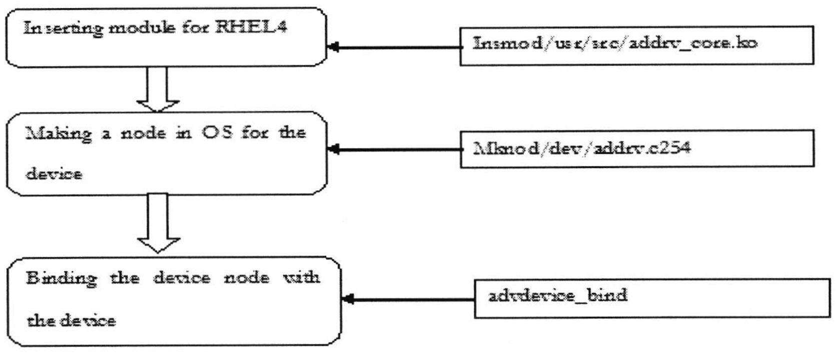
Interfacing between the DAQ card and the software is most important part in the project. The card receives the data from the sensors and sends the data to the computer. The computer gets a digital data and software takes the responsibility for further processing of the data and shows it in a specific manner. So at first the communication between the software and the card is necessary [31].

Different properties and function of ADvAI.dll is used for analog signal processing in the software. First the device was selected by calling SelectDevice function. It makes sure that the correct version of the product which is USB-4716 is in use. After that device name and device number was selected by using DeviceName and DeviceNumber properties. Then by using DataAnalog properties to control analog input data coming from the sensors. After getting the analog input data software processes it as needed and then displays it to graphical user interface. C# language was used for the graphical user interface and specified functions in C# development environment wad added as reference.

2.18 Interfacing Technique for Linux

To interface the Linux with the hardware the driver named advdaq-1.09.0001-el4.i386 in Red Hat Linux4 (RHEL4) operating system needed to compile. By compiling the driver file two modules insmod/usr/src/adddrv_core.ko and insmod/usb4716.ko were inserted in a module for RHEL4. After this the process becomes a part of the Linux OS. 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 DAQ system and Linux OS and step2 makes the hardware-software contract with the DAQ system with the Linux software [32].

In step 1, after inserting the two modules a file is created automatically which is a addrev file: /proc/device/addrev. Then the major type of the file was found. Using the major type of the file a node was made of the USB hardware with the OS by using mknod/dev/addrv.c254. After making the node the node of the OS to the hardware was bound by using binding command advdevice_bind. After binding the device with the OS hardware-hardware contact part is done. Now the Linux OS is ready for getting analog data through the channels of the DAQ card. The binding process is shown in the flow chart below.



Inserting module for RHEL4
Insmod/usr/src/addrv_core.ko
Making a node in OS for the device
Mknod/dev/addrv.c254
Binding the device node with the device
advdevice_bind

Figure 2.13: Procedures of Linux-DAQ communication

After making the communication between Linux and DAQ it needs to make communication with the process of Linux to obtain analog data through the hardware channels. For that it is necessary to communicate with the device node that was made within the Linux OS.

For obtaining analog data as input several functions were used given by the DAQ card manufacturer. First the device is opened 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. If it opens successfully, it will return output or otherwise will generate an error.

After opening up the device, it is configured for obtaining the analog data correctly. For obtaining the analog data specific channel and a gain code need to set up. 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 DRV_AIvoltageIn functions which have two parameters where both are pointer have been used. One is retrieved from DRV_DeviceOpen and another is from PT_AIConfig. List of functions used for this project are DRV_DeviceOpen, DRV_GetErrorMessage, DRV_DeviceClose, DRV_AIConfig, DRV_AIVoltageIn. The software-hardware communication process flow charts are given below.

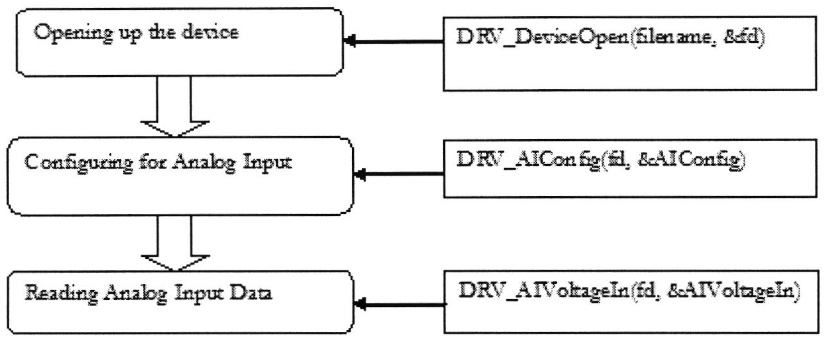

Opening up the device
DRV_DeviceOpen(filename, &fd)
Configuring for Analog Input
DRV_AIConfig(fd, &AIConfig)
Reading Analog Input Data
DRV_AIVoltageIn(fd, &AIVoltageIn)

Figure 2.14: Procedures of Software-DAQ Communication

2.19 Interfacing Technique for Real-Time Linux

Results were obtained for Windows and Linux but the real-time kernel for Linux was not available. Therefore the interfacing could not be done between the RT-Linux and DAQ. In case of RT-Linux on the availability of real-time kernel the inter process communication is needed between the real-time kernel and non real-time GUI done. Real-time thread and Linux program need to access the same piece of shared memory so have to connect up to the same 'tag' by named mbuff.h header file. The mbuff_alloc and mbuff_free functions calls are used as Linux system call function thus need to be called inside the init_module and cleanup_module functions respectively, and not called within the thread code itself. To implement shared memory in RT-Linux, the following code additions are made:

- volatile int *variable; //global declaration of shared variable in both the real time module and Linux program

- variable=(volatile int*) mbuff_alloc("tag name", size); //assign space to the pointer variable in both the real-time module and Linux program

- mbuff_free ("tag name", (void*)variable); //deallocate process from shared memory space in both the real-time module and Linux program

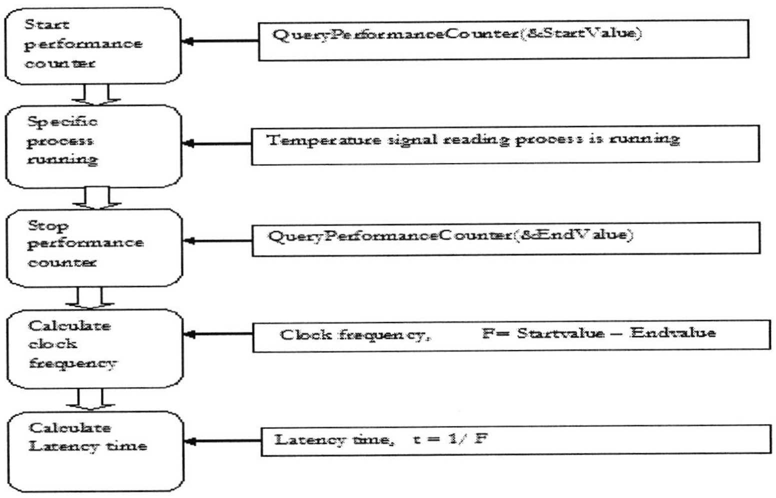
In this section the brief overview of the data acquisition system under different operating systems used for the experiments is presented. The interfacing technique between the DAQ card and operating system is shown. Next section is followed by the conducted experiments and the test results.

2.20 Experimental Results

The experiments done in this project, techniques used for the experiments and the result or significance of the experiments are explained in the following section. For the experiments, two operating systems Windows XP and Red Hat Linux4 are used where one is a general purpose operating system and another is soft real time operating system. The latency for signal processing was found out in both environments.

2.21 Time Measurement Technique for Windows

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. By subtracting start time clock frequency from end time clock frequency the number of frequency needed for the process is obtained. 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 the function QueryPerformanceCounter (&value) was used. Windows latency measurement technique flow chart for temperature signal is given below.



Start performance counter
QueryPerformanceCounter(&StartValue)
Specific process running
Temperature signal reading process is running
Stop performance counter
QueryPerformanceCounter(&EndValue)
Calculate clock frequency
Clock frequency, F= Startvalue – Endvalue
Calculate Latency time
Latency time, t = 1/ F

Figure 2.15. Time latency measurement steps in windows

2.22 Time Measurement Technique for Linux

To measure time latency in Linux environment a function given by the Linux developer is used. 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 it does not need to calculate time from frequency rather the time is obtained directly from the function. To use the function gettimeofday we need to include a header file include <sys/time.h>.

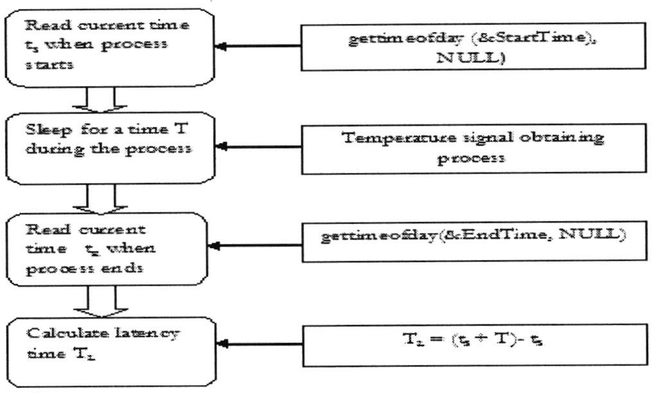

Read current time t_s when process starts
Sleep for a time T during the process
Read current time t_e when process ends
Calculate latency time T_L
gettimeofday (&StartTime), NULL)
Temperature signal obtaining process
gettimeofday(&EndTime, NULL)
$T_L = (t_s + T) - t_e$

Figure2.16. Time latency measurement procedures in Linux

2.23 Comparison of Latency in Two OS

The latency time both for Windows and Linux was obtained and compared. Biomedical temperature sensor for the signal processing purpose is used and 100 samples were taken for each unit of average data for both in windows and Linux. Process was run for 5 times that means 500 data were taken in windows and Linux. The average time latency for both the systems is given below.

TABLE 2.2 DEPENDENCY ON RUNNING PROCESSES

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

From the Table 2.2 a decision about the time latency in Windows and Linux OS can be made. It can be seen 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 Windows which is definitely a system drawback. Other than that it was observed some time Windows 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.

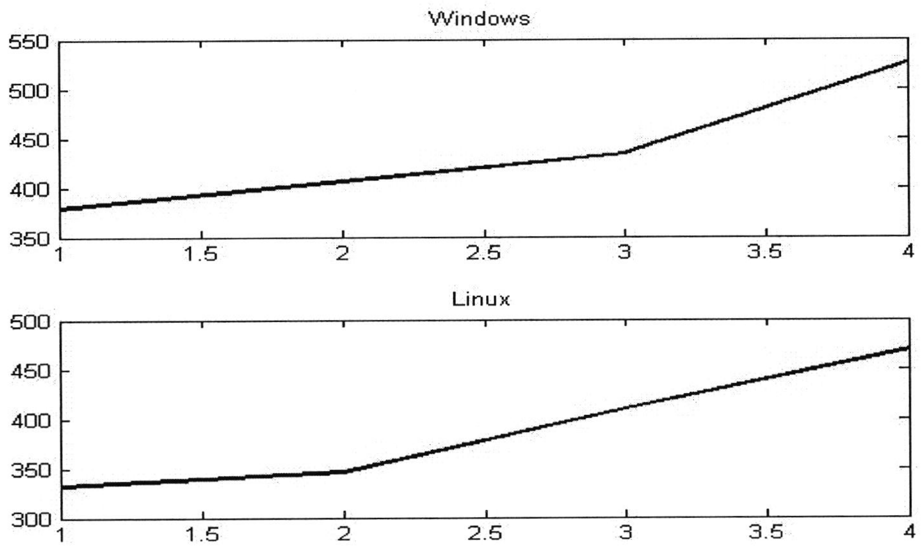
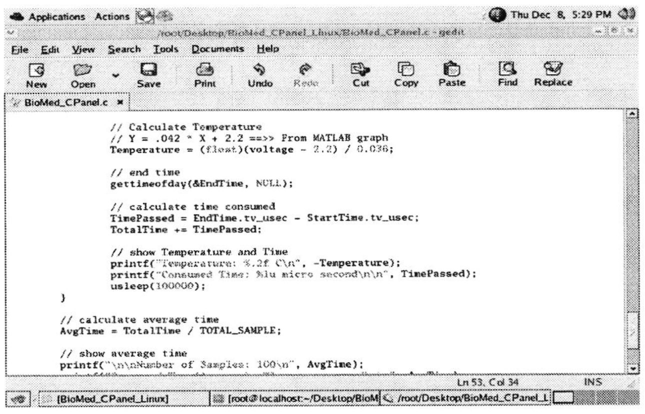


Figure 2.17: Windows and Linux time latency variation depending on running processes


Applications Actions
Thu Dec 8, 5:29 PM
/root/Desktop/BioMed_CPanel_Linux/BioMed_CPanel.c - gedit
File Edit View Search Tools Documents Help
New Open Save Print Undo Redo Cut Copy Paste Find Replace
BioMed_CPanel.c
Ln 53, Col 34
INS
[BioMed_CPanel_Linux]
[root@localhost:~/Desktop/BioM
/root/Desktop/BioMed_CPanel_L

```
            // Calculate Temperature
            // Y = .042 * X + 2.2 ==>> From MATLAB graph
            Temperature = (float)(voltage - 2.2) / 0.036;

            // 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("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: 100\n", AvgTime);
```

Figure 2.18: Linux code for time latency measurement

2.24 Comparison of Time Delay in Windows, Linux and RT-Linux

To show the Comparison of task completion time in Windows, Linux and RT-Linux under multitasking and without multitasking environment, the results were taken from the robotic arm manipulator experiment done in our lab. In that experiment the total task completion time is the time taken by the robotic arm to pick an object and pass it to the conveyer belt and collect that object by another robotic arm. It can be seen (Table 2.3) [33] in case of Windows task completion time under multitasking environment is 7.52s and without multitasking 6.24s. For Linux task completion time with multitasking is 6.34s and without multitasking it is 6.0s. But in case of RT-Linux task completion time is 4s. These results give the verification that RT-Linux is better than Linux and much better than Windows.

TABLE 2.3 TOTAL TIME DELAY IN WINDOWS, LINUX AND RT-LINUX

Operating System	Task Completion Time (T1) (with multi tasking)	Task Completion Time(T2) (without multitasking)
Windows (1)	**7.52s**	**6.24s**
Linux (2)	**6.34s**	**6.0s**
RT-Linux (3)	**4.0s**	**4.0s**

2.25 Temperature Measurement Technique

Temperature measurement can be done in some specific ways. Here a noninvasive biomedical temperature sensor is used which will give some potential difference (p.d) according to the body temperature. After the digitization of the taken p.d the temperature of the human body is shown in a workstation which is in Windows and Linux environment.

The measurement of temperature will be accommodated with other measuring parameters of animal body through the graphical user interface.

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

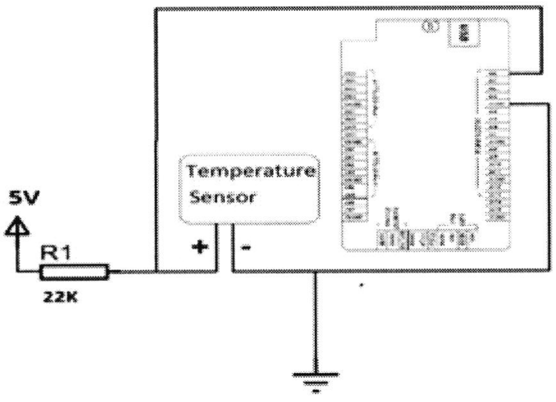


Figure 2.19: The circuit of temperature sensor

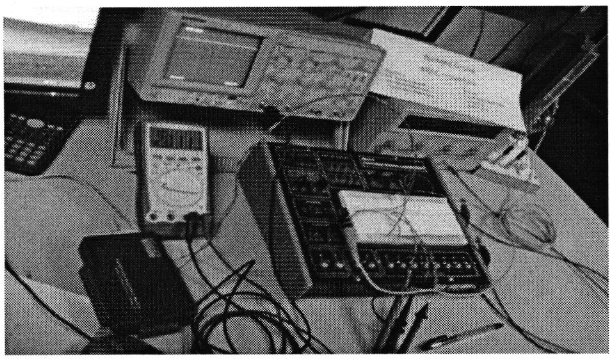


Figure 2.20: Lab Setup

2.25.1 Test Results

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

TABLE 2.4 TEMPERATURE AND CORRESPONDING VOLTAGES

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 data were plotted in MATLAB and using curve fitting method equation was obtained. The equation is needed to use in the developed software and the equation is, y=0.042x+2.2 where y is voltage and x is temperature.

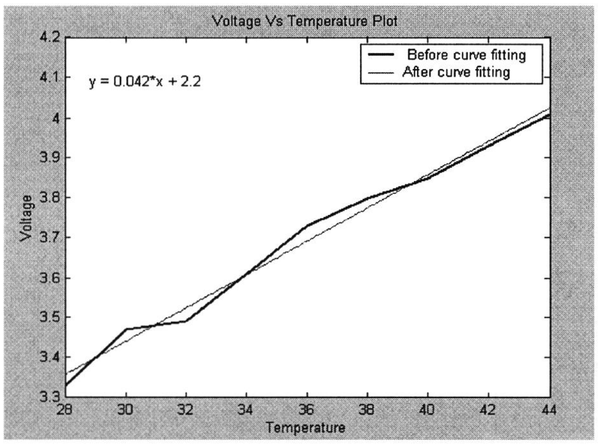
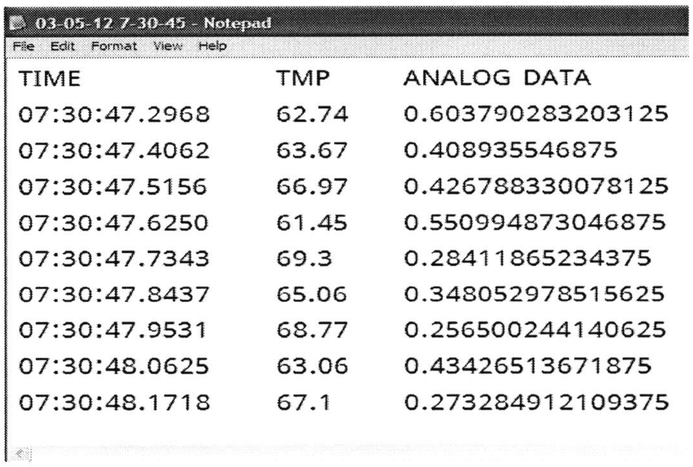


Figure 2.21: Curve fitting for temperature equation

It was obtained that the temperature x= y-3.16/0.042. This equation was used in the software to show temperature from the sensor. These data are also saved in a data logger so that these information can be used for research purpose if offline mode.



```
TIME              TMP      ANALOG DATA
07:30:47.2968     62.74    0.603790283203125
07:30:47.4062     63.67    0.408935546875
07:30:47.5156     66.97    0.426788330078125
07:30:47.6250     61.45    0.550994873046875
07:30:47.7343     69.3     0.28411865234375
07:30:47.8437     65.06    0.348052978515625
07:30:47.9531     68.77    0.256500244140625
07:30:48.0625     63.06    0.43426513671875
07:30:48.1718     67.1     0.273284912109375
```

Figure 2.22: Output of Data Logger

2.26 Data Deployment on developed Software

After getting the equation x= y-3.16/0.042 where x= temperature and y=voltage, the equation was used in our developed software to get voltage and temperature shown in Graphical User Interface (GUI). Voltage and temperature is also shown in a graph. The temperature graph shown in the GUI has been given below.

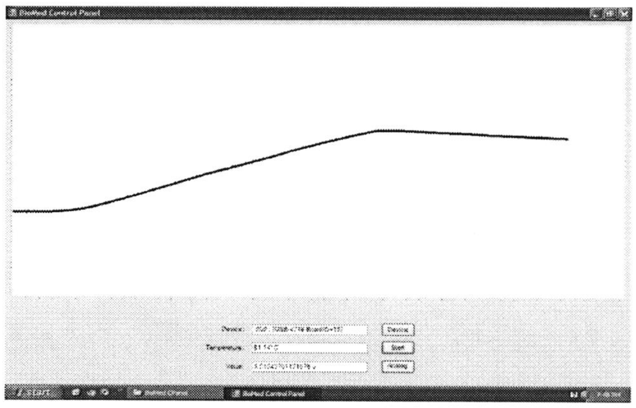


Figure 2.23: Temperature shown in GUI

2.27 Experiments with ECG Sensor

Electrocardiogram records the electrical activity of the heart. Signal is received from human/animal object through ECG electrodes. Three leads ECG sensor is used. The received signal is passed to the signal processing circuit to make necessary amplification and noise removal. The ECG signals were generated from ECG simulator and the output is displayed in digital oscilloscope (Figure 2.24).

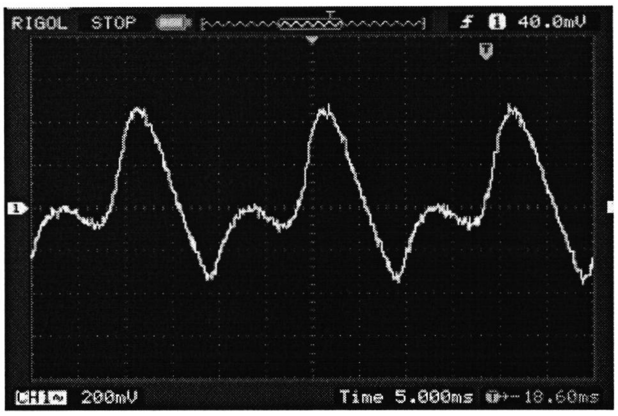


Figure 2.24 The Circuit of ECG Sensor and ECG output

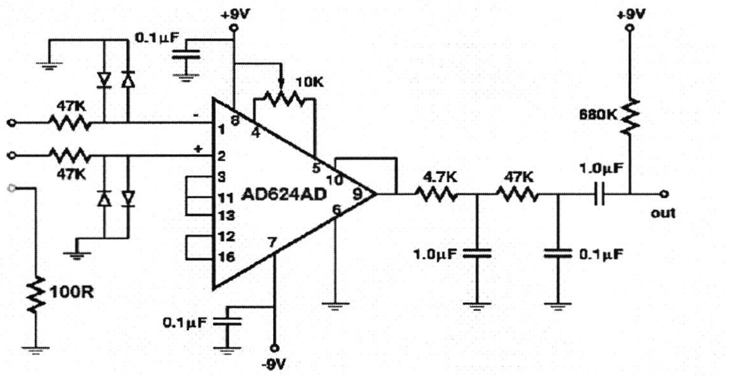


Figure 2.25 ECG Signal Processing Circuit

The signal processing circuit consists of AD624 amplifier. Diode protections were added to the input of the amplifier (Figure 2.25). Finally for filtering high frequency signal, a low pass filter is used.

In this chapter a details about biosignal processing system along with different experimental results are presented. Time measurement technique along with the results is shown in Windows and Linux environment. Latency time is calculated and compared. Experiments on ECG are done and the result is displayed on digital oscilloscope. Next chapter will discuss about the theoretical analysis of the time delay of sampled data acquisition system.

Chapter 3

TIME DELAY ANALYSIS OF SAMPLED DATA SYSTEM

3.1 Introduction

The increasing use of PC hardware is one of the most important developments in high-end embedded system. In the real-time servo Biomedical System loop the controlled variable is sampled (A/D) at a suitable constant based sampling rate and control algorithm uses this information to compute a new control action which is reconstructed (D/A) at a faster rate (integer multiple of based sampling rate) and applied to the continuous servo plant (Figure 3.1). Complexity arises when time delays are introduced over a communication control network, where the signal is sampled and then sent to the controller node. For an example when the communication is considered over a field bus where the signal sampling rate is higher than the sampling rate of the controller and the signal is sent to the controller node. Due to the sampling and synchronization of the signals between the I/O device (ADC and DAC) and the controller, a delay variation may occur, which affects the performance of the system.

3.2 Time Delay of Sampled Data System

A sampled-data system is a control system in which continuous-time is controlled with a digital device. Under periodic sampling, the sampled-data system is time-varying but also periodic. Time delays are introduced over a communication control network, where the signal is sampled and then sent to the controller node. This system is called sampled data system. The system block diagram with different delays at different layer is given below. This set up will give the system with varying time delays from sensors to the computer, i.e., the variation of control delay which is the time from when the measurement signal is sampled till it reaches the digital signal processor (DSP). These delay sources are combined together and gives the total time delay for the system. It will be shown in the next topics how this delay at each layer contributes to the system.

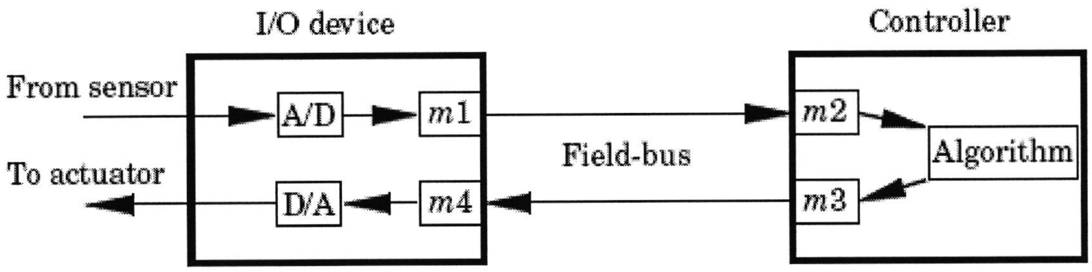


Figure 3.1 System Block Diagram with Different Source of Delays

In the data acquisition module (Figure3.1), after sensor receiving the signal, the signal is transferred to the ADC. One delay is coming when the signal is sampled in the ADC and the sampled output is copied into memory m1. Here the delay is τ_{SC}. After that the data is transferred from m1 to the memory of the computer m2. During this transfer the field bus delay is τ_{bus}. Then τ_{CC} gives the computational delay. For the close loop case, the signal will go to the actuator causing delay τ_{ca} at DAC. Thus all source of delay can be accumulated as,

$$\tau_{total} = \tau_{SC} + \tau_{CC} + \tau_{bus} + \tau_{ca}$$

This sampled data systems are divided into different levels. The top level is considered in the data acquisition part (ADC/DAC). Memory device and field buses are considered as higher level and the controller is considered as lower level '0'.

3.2.1 Timing Diagram of the System

Timing diagram of typical cycle in a partial synchronous loop is shown in the Figure below.

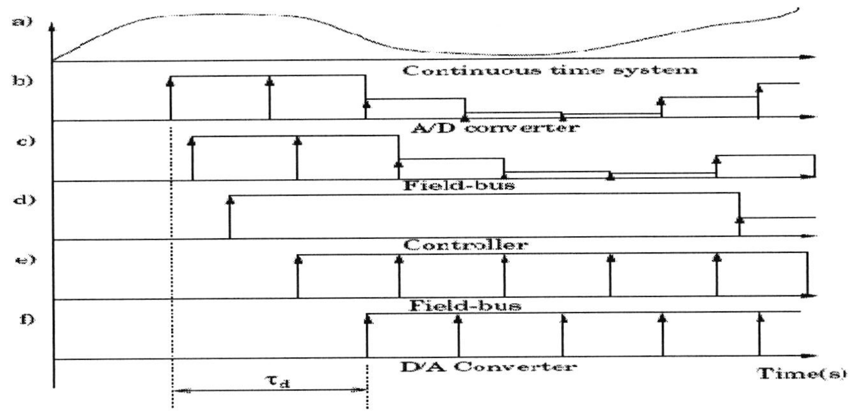


Figure3.2 Timing Diagram of the System

In the diagram 1st the continuous time signal that is to be processed is shown. Then the signal is A/D converted which is the value corresponding the value of memory m1 (Figure 3.1). Then the signal is sent to the DSP using field bus. One delay is there. After that the signal will go to the DAC through the field bus. The total delay thus is τ_d.

3.2.2 Time Delay in Definite Synchronous System

For the simplicity two layers are considered and time delays are observed for three different cases [34, 35, 36]. Here the 1st layer (in this case layer '0') is doing the necessary computation to produce output and another layer (layer '1') is taking the sample from the experimented object. In the first case the sampling time $h_0=h_1$, in the second case $h_0 \geq h_1$ where the layer 0 sampling time is greater than layer 1 which cause reject sampling and finally

$h_0 \leq h_1$, the sampling time in layer 0 is less than layer 1, which cause vacant sampling in layer 0.

Delays in a two layer system are presented below in the diagram.

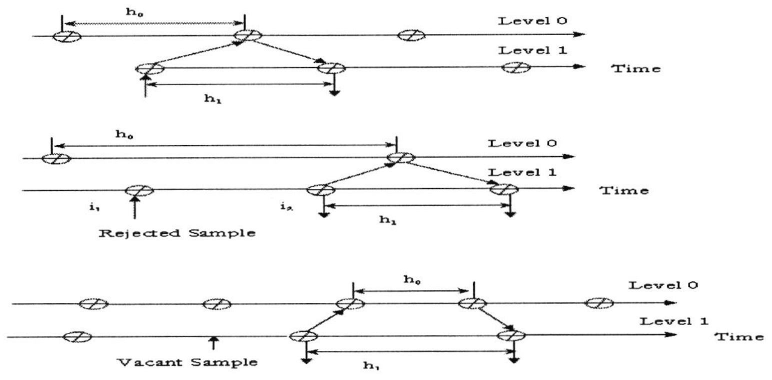


Figure 3.3 Delays in a Definite Synchronous System with Two Layers

3.2.2.1 Time Delay on Layer i

Following the example above it can be said that if any signal enters at any level at time $n_i h_i$ and leaves that level at time $n_i h_i + k_i h_i$, total time delay for sampling is $k_i h_i$ where k_i is a positive integer. Therefore on layer L (bottom layer) the delay is $\tau_L = k_L h_L$..(i)

Now two cases can be considered. First, the delay in layer i is less than the sampling interval of layer i+1, $\tau_L \leq h_{i+1}$. Now τ_i is drifting in relation to h_{i-1} due to the different sampling rates. The possibilities that τ_i falls within a sampling period of layer i+1 is, $(h_{i+1} - \tau_{i)} / h_{i+1}$ [37, 38]. Therefore the delay at layer i+1 is $(h_{i+1}-\tau_i)/h_{i+1.}$ This gives

$$P(k_{i+1} = 1) = \frac{h_{i+1} - \tau_i}{h_{i+1}} \quad \text{...(ii)}$$

In that case the delay at layer i+1 would be $\tau_i = h_{i+1}$ .... (i')

Now let's consider the second case where $\tau_i = 2h_{i+1}$. .... (i'')

This occurs if and only if a sampling time in layer i+1 falls in the interval τ_i. Thus probability of delay for this case is:

$$P(k_{i+1}=2)=\frac{\tau_i}{h_{i+1}} \quad \text{(iii)}$$

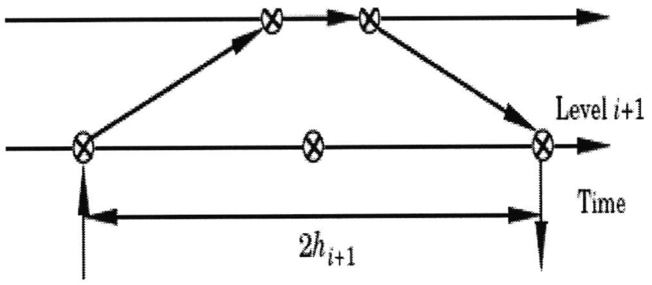


Figure 3.4 Time Delay on layer i+1 when $\tau_i \leq h_{i+1}$.

Now a longer time delay where $\tau_i > h_{i+1}$, then the delay is written as $\tau_i = nh_{i+1} + \overline{\tau_i}$

and the delay at layer i+1 is $\tau_{i+1} = nh_{i+1} + h_{i+1}$ .... (iv)

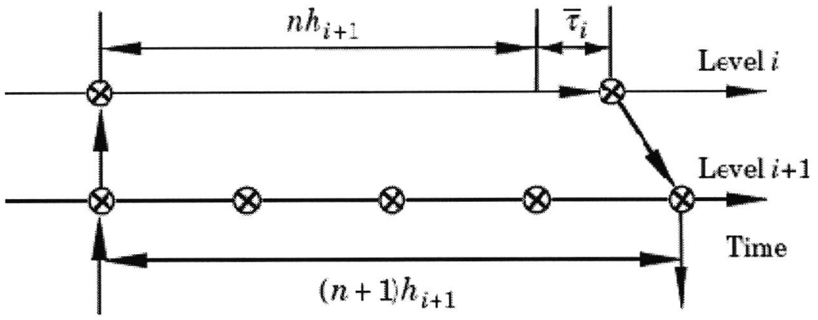


Figure 3.5 Time Delay on layer i+1 when $\tau_i \geq h_{i+1}$

Analogously to (ii) and (iii) and replacing τ_i by $\overline{\tau_i}$ gives,

$$P(k_{i+1}=n+1)=\frac{h_{i+1}-\overline{\tau_i}}{h_{i+1}} \quad \text{(v)}$$

$$P(k_{i+1}=n+2)=\frac{\overline{\tau_i}}{h_{i+1}}=\frac{\tau_i - nh_{i+1}}{h_{i+1}} \quad \text{(vi)}$$

3.2.2.2 Example to Illustrate Definite Synchronous Model

Let's consider the sampling time $\tau_i = h_1 = 264 > h_2 = 260$ ms. From equation (iv) the delay is found 520 or 780 ms. The probabilities for the two delays now can be calculated from (v), and (vi).

$$P(\tau_a = 520) = \frac{256}{260} = .9846$$

$$P(\tau_a = 784) = \frac{4}{260} = .015$$

Now if we consider $\tau_i = h_i = 260 < h_2 = 264$ms, in that case the delay will be 264 ms or 528 ms. The probabilities for the two delays can now be calculated from equation (ii) and (iii).

$$P(\tau_a = 264) = \frac{4}{264} = .015$$

$$P(\tau_a = 528) = \frac{260}{264} = .9848$$

3.2.3 Possible Number of Delay in a Partial Synchronous System

The system where both input and output have the same sampling rate but do not occur at the same time is defined as partial synchronous system. The system considered in my work is a partial synchronous system. There is constant time difference in the partial synchronous system which is called as phase difference. If the constant phase difference between the inputs and outputs θ_H is introduced for the higher level, delays of the partially synchronous systems are shown in Figure 3.6 below:

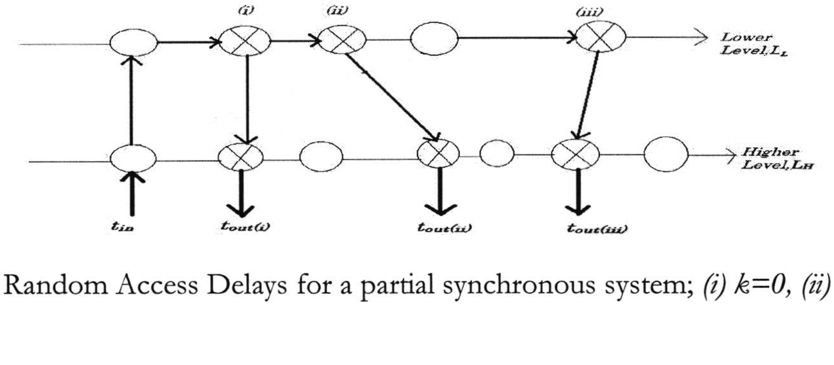


Figure 3.6 Random Access Delays for a partial synchronous system; *(i) k=0, (ii) k=, 1 (iii) k=2*

If a signal enters the layer i+1 at time t_{in} and the corresponding signals exist the layer t_{out} then the delay in layer i+1 is, $\tau_{i+1}=k_{i+1}h_{i+1}+\theta_{i+1}$................................(i''')

Let's consider the case where $\tau_i<h_{i+1}$. Based on the idea of definite synchronous model the probabilities of delay was found that,

$$P(k_{i+1}=0)=\frac{\theta_{i+1}-\tau_i}{h_{i+1}}$$

$$P(k_{i+1}=1)=\frac{h_{i+1}-|\theta_{i+1}-\overline{\tau_i}|}{h_{i+1}}$$

$$P(k_{i+1}=2)=\frac{\overline{\tau_i}-\theta_{i+1}}{h_{i+1}}$$

For the case $\tau_i>h_{i+1}$ using the same idea in definite synchronous system the probability of delay can be written as:

$$P(k_{i+1}=n)=\frac{\theta_{i+1}-\tau_i}{h_{i+1}} \quad \text{..........................(vii)}$$

$$P(k_{i+1}=n+1)=\frac{h_{i+1}-|\theta_{i+1}-\overline{\tau_i}|}{h_{i+1}} \quad \text{................(viii)}$$

$$P(k_{i+1}=n+2)=\frac{\overline{\tau_i}-\theta_{i+1}}{h_{i+1}} \quad \text{..........................(ix)}$$

3.2.3.1 Joining Levels

The possible delays on layer L, the bottom layer are given by $k_Lh_L+\theta_L$. The probability for any one k is the additive conditional probabilities for all time delay in the layer above. Thus the total probability is written as:

$$\begin{aligned}P(k_{i+1}=n) &= P(k_{i+1}=n|\tau_i=\theta_{i+1})P(\tau_i=\theta_{i+1})\\ &+ P(k_{i+1}=n|\tau_i=h_{i+1}+\theta_{i+1})P(\tau_i=h_{i+1}+\theta_{i+1})\\ &+ P(k_{i+1}=n|\tau_i=2h_{i+1}+\theta_{i+1})P(\tau_i=2h_{i+1}+\theta_{i+1})\\ &+\dots\end{aligned}$$

3.2.3.2 Example to Illustrate Partial Synchronous Model

Let's consider the sampling time h_1 =264>h_2 =260 ms, θ_1.=160 ms and θ_2 =180 ms. First layer 1 is considered. There are two possible time delays in layer 1, θ_1 and θ_1+h_1. From equation (i''') the delay is found 160 or 424 ms. The probability from this delay can be calculated from equation vii and viii.

$$P(\tau_1 = \theta_1) = P(\tau_1 = 160) = \frac{160}{264} = .606$$

$$P(\tau_1 = \theta_1 + h_1) = P(\tau_1 = 424) = \frac{104}{264} = .3939$$

Now the possible time delay in layer 2 correspondent to 180 ms and 440 ms due to the first value of τ_i and 440ms and 700ms due to second value of τ_1 .Thus from (vii),(viii)and (ix) we get,

$$P(\tau_a = 180) = P(k_{i+1} = 0 \mid \tau_i = 160) P(160) = \frac{180-160}{260} = .0769 \times .606 = .0466$$

$$P(\tau_a = 440) = P(k_{i+1} = 1 \mid \tau_i = 160) P(\tau_i = 160) + P(k_{i+1} = 1 \mid \tau_i = 422) P(\tau_i = 424)$$

$$= \left[\frac{260 - |180-160|}{260} \times .606\right] + \left[\frac{260 - |180-424|}{260} \times .3939\right] = .559 + .02424 = .5832$$

$$P(\tau_a = 700) = P(k_{i+1} = 2 \mid \tau_i = 424) P(\tau_i = 424) = \frac{424-180}{260} \times .3939 = .3696$$

3.2.3.3 Maximum Time Delays for Partial Synchronous System

There is no time delay on layer 0, on layer 1 there are two possible time delays and on layer 2 there are four possible time delay. Thus for each delay on a particular time delay there are two possible time delays on the layer below. Thus for layer L there are 2^Lpossible time delay.

3.2.3.4 Example to Illustrate Time delay for the proposed System

- **<u>Probable Time Delay for ECG Signal</u>**

Let's consider ECG signal. As we know the bandwidth for ECG signal is .05 Hz-150 Hz, the sampling frequency must be greater than twice the bandwidth according to Nyquist criterion. Now if the sampling time taken by DAQ for the ECG signal (level 2) is h_2= 3 ms (Assuming sampling frequency 300 Hz) and the sampling time of computer is h_1= 2 ms then the delay (equation i' and i") will be 3ms or 6ms. The probabilities of this delay can be found from equation ii and equation iii as,

$$P\ (\tau_d{=}3\text{ ms}) = \frac{(3-2)}{3} = \frac{1}{3} = .33$$

$$P\ (\tau_d{=}6\text{ ms}) = \frac{2}{3} = .66$$

Now if the sampling time of the computer (level 1) is 5 ms then the total delay can be found from equation (iv) is 6 ms or 9 ms. The probabilities of this delay can be found from v and vi as,

$$P\ (\tau_d{=}6\text{ ms}) = \frac{(3-5+3)}{3} = \frac{1}{2} = .33$$

$$P\ (\tau_d{=}9\text{ ms}) = \frac{5-3}{3} = .66$$

Now let's consider additional delays are inserted into the system which can be due to noise. Let's take θ_1=1 ms and θ_2=4 ms. Then the two possible time delays are 1 ms and 6 ms (equation i''') and the probability of this delay is (vii and viii)

$$P\ (\tau_d{=}1\text{ ms}) = \frac{1}{5} = .2$$

$$P\ (\tau_d{=}6\text{ ms}) = \frac{5-1}{5} = .8$$

Possible time delays in layer 2 corresponding to 2 ms and 7ms due to first value of τ_d and 12 ms due to second value of τ_d. From (vii,viii and ix) we get,

$$P\ (\tau_d{=}6\text{ ms}) = \frac{4-1}{3}\times.2 = 1\times.2 = .2$$

$$P\ (\tau_d{=}7\text{ ms}) = \left[\frac{(3-|4-1|}{3}\times.2 + \frac{3-|4-6|}{3}\times.8\right] = 0 + .26 = .26$$

$$P\ (\tau_d{=}12\text{ ms}) = \frac{6-4}{3}\times.8 = .53$$

- **Practical Time Delay Calculation**

Practical Data was found by sending data to the computer via data acquisition card and receiving the same data via DAC and then the total time delay was calculated. Practically the delay was found as: 18.6-17.6+4.12 =3.12 ms.

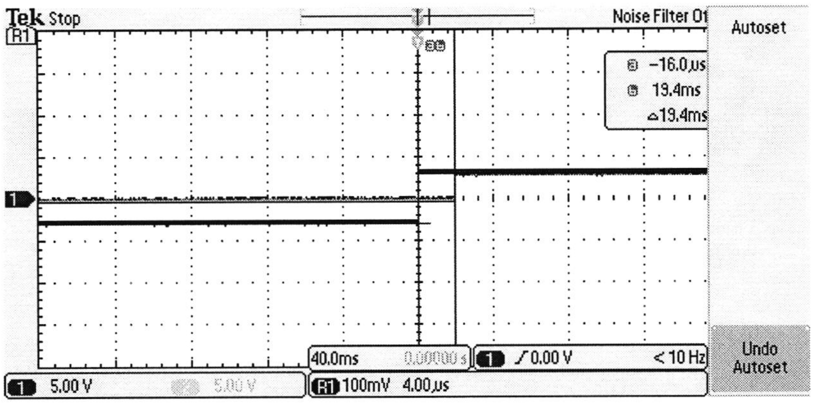


Figure 3.7 Digital Oscilloscope Output of Total Time Delay

- **Probable Time Delay for Oxygen Saturation Measurement Signal**

Let's consider oxygen saturation measurement using pulse oximeter. If the sampling rate of ADC is h_2=20 ms (sampling frequency 50 Hz) and the sampling time of computer is h_1=15 ms then the delay from equation i' and i" is either 15 ms or 30ms [24]. Then the probabilities of delay can be found from equation ii and iii as,

$$P\ (\tau_d = 15\ \text{ms}) = \frac{(20-15)}{20} = \frac{5}{20} = .25$$

$$P\ (\tau_d = 30\ \text{ms}) = \frac{15}{20} = .75$$

Now if the sampling time of computer is 25 ms then the delay will be 40 ms or 60 ms and the probabilities for the delay can be found from v and vi as,

$$P\ (\tau_d = 40\ \text{ms}) = \frac{20-25+20}{20} = \frac{15}{20} = .75$$

$$P\ (\tau_d=60\text{ ms}) = \frac{25-20}{20} = \frac{5}{20}. = 25$$

Now let's consider additional delays are inserted into the system which can be due to noise. Let's take θ_1=10ms and θ_2=15ms. Then the two possible time delays are 10 ms and 35ms (equation i''') and the probability of this delay is (vii and viii)

$$P\ (\tau_d=10\text{ ms}) = \frac{10}{25} = .4$$

$$P\ (\tau_d=35\text{ ms}) = \frac{25-10}{25} = .6$$

Possible time delays in layer 2 corresponding to 15 ms and 35ms due to first value of τ_d and 55 ms due to second value of τ_d. From (vii,viii and ix) we get,

$$P\ (\tau_d=15\text{ ms}) = \frac{12-10}{20}\times .4 = .1\times .4 = .04$$

$$P\ (\tau_d=35\text{ ms}) = \left[\frac{(20-|12-10|}{20}\times .4 + \frac{20-|15-35|}{20}\times .6\right] = +.36+0 = .36$$

$$P\ (\tau_d=55\text{ ms}) = \frac{35-15}{20}\times .6 = .6$$

Similarly for all channels this delay can be calculated separately. For respiratory rate and aortic pressure measurement the sampling frequency is 50 Hz and 60 Hz respectively. This delay will be added with the computation delay where RT-Linux solution is proposed to guarantee the hard real-time requirement.

Another thing need to be examined is the aliasing effect. Aliasing effect will occur if the sampling time is less than the ADC conversion time. The ADC conversion time is (1/clock frequency). For the USB-4716 DAQ card the clock frequency is 10 MHz. The ADC conversion time is $= \frac{1}{1\times 10^6} = .1\,\mu s$ which is much less than the sampling time taken by ECG (3 ms) and oxygen saturation (20 ms). So aliasing effect will not occur.

In the discussion above, time delays for the sampled data system is analyzed. It was seen that the theoretical maximum occurs when the phase of the input and the output layers all are synchronized. It can be seen that the maximum time delay is doubled the added value of all layers other than the layer 0. A possible delay in layer 0 is then added which is the computational delay. It was found that there is two possible time delay for the delay in layer above. Example is shown for the time delay ECG and oxygen saturation signal and aliasing effect is analyzed. . Now the question arises how the delay is affecting the system and how to solve the problem?

3.3 Effect of Time Delay on Sampled Data System

System performance can be adversely affected by the presence of delays [34,35] in the loop. In fact small or large delays and their associated phase lag provide unwanted oscillation in the system response, which may even lead to instability. The reason for such delays can be due to sensor characteristics, insufficient processing speed, or communication lags etc. Other delays that can also be inherent in the process are neglected. The varying time delay in computing and transmitting the control output and its negative effects on real-time control systems are classified into delay and loss problems. The non zero time varying delay shorter than the sampling interval yields the delay problem and the delay greater than the sampling interval cause loss problem. Due to the time delay some errors occurs in the system such as Jitter, transient errors which is discussed below.

3.3.1 Jitter

Jitter can be defined as time variations in actual start times of actions as opposed to stipulated start times. Jitter depends on clock accuracy, scheduling algorithms and computer hardware architecture. One related issue connected to scheduling is intentional changes in sampling period. Typically the jitter change at each sampling interval. For a particular system load, if the system is reasonably predictable, it should be possible to determine the values that the sampling interval actually takes. If the jitter is known beforehand, its effect can be analyzed and compensated.

3.3.2 Transient Errors

Transient errors occur due to loss or corruption of signals during communication. It has increased data delay therefore time variations are introduced in the system. To recover from such errors, one way is to detect loss of measurement (vacant sampling) and predict the output of the process. Another serious error, a temporary blackout refers to transient fault which cause the system to behave in an unpredicted way (for example, no action or erroneous action) for some period of time. In safety issues it is important to consider these problems and appropriate measures need to be taken.

Many research works have been done on the effect of time delay on the performance of sampled data system. In paper [39] the stability and worst case performance of network embedded system is analyzed for sampled data control system. Analysis is done considering both input and output jitter and for pure output jitter conservativeness of a previously stability theorem for pure output jitter was reduced and stability criterion is developed based on small gain theorem.

In other researches paper [40, 41] it is shown the effect of communication delay can strongly affect the stability and performance and the network experience higher level of data drop out and corruption due to noise, interference etc. and the relationship between performance degradation and feedback delay is shown.

In the paper [42] effect of computational delay on the performance of hybrid ACC is shown. The computational delay effect the hybrid control system in terms of peak error, RMSE and control energy and a computational delay compensator was applied with the controller.

This paper [43] proposes a scheme for real time feedback controller taking the effect of deadline missing of the controller due to delay and uncertainity of the plant due to time delay cause system failure.

Keeping this constraints in mind a controller need to be designed to control the stability of the system and. In next chapter mathematical modeling of the controller will be shown with necessary example.

Chapter 4

MATHMATICAL MODEL OF SAMPLED DATA CONTROL SYSTEM

4.1 Introduction

Sampled data systems are hybrid systems, involving both continuous time and discrete time signals. Such a system operates in continuous time but some continuous time signals are sampled at certain time instants (usually periodically), yielding discrete time signals. Before designing any sophisticated sampled data system (such as automatic heart pump for animal/human) it is important to design the controller which will take necessary action to minimize the error of the system and maintain the stability of the system caused by time delay. To design the controller the mathematical model need to be analyzed. As discussed in previous chapters that the system proposed is having multiple input and multiple output yielding MIMO system. In this chapter first single input system with two rates is considered and then using lifting technique two rates is converted into single rate and finally mutilate input output system is shown.

4.2 Automatic Heart Pump Sampled Data Control System

An example is given below to show the biomedical automatic heart pump control system. Sensor receives the information about patient heart beat and sends this information to controller. Based on the control, decision this signal is sent to the motor which then rotates according to the set speed by the controller. The controller has another job of maintaining the stability of the system by calculating the time delay.

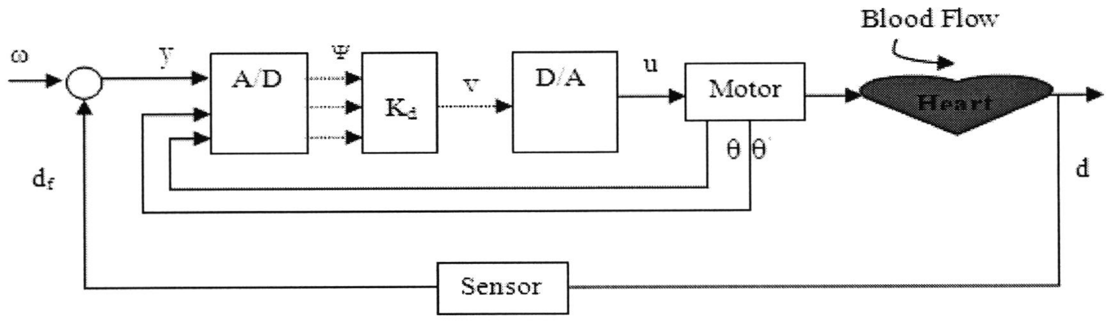


Figure 4.1 Automatic Heart Pump Control Systems

The mathematical representation of above biosignal processing system is below:

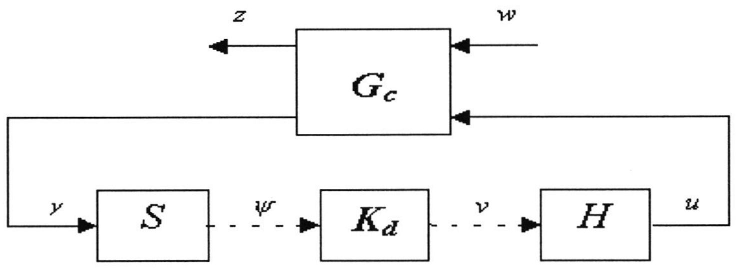


Figure 4.2 Mathematical Representation of Automatic Heart Pump Control System

Here, z = signal to be controlled = w-d, in this case, w=r

y= measured signal input to the digital controller = $\begin{bmatrix} r-d_f \\ \theta \\ \theta' \end{bmatrix}$

w= exogenous input consisting of reference commands such as from sensors, noise

u= control input= output of the digital controller

K_d= controller

Ψ and v are input and output of the controller.

The continuous time plant Gc is defined as follows.

$$G_c(s) = \begin{bmatrix} A & B_w & B_{u_c} \\ C_z & D_{zw} & D_{zu_c} \\ C_{y_c} & D_{y_c w} & D_{y_c u_c} \end{bmatrix}$$

$$H : l(Z) \Rightarrow L(R), u = Hv , u(kh + t) = v(k), 0 < t \le h$$

$$S : L(R) \Rightarrow l(Z), \Psi = Sv , \Psi(k) = y(kh)$$

$$\begin{aligned} \dot{x}(t) &= Ax(t) + B_w w(t) + B_{u_c} u_c(t) \\ y_c(t) &= C_{y_c} x(t) + D_{y_c w} w(t) \\ z(t) &= C_z x(t) + D_{zu_c} u_c(t) \end{aligned}$$

For the sampled data controller problem the equivalent discrete time representation of the plant dynamics and average measurement are as follows:

$$\begin{aligned} x^d(k+1) &= A^d x^d(k) + w^d(k) + B^d \hat{u}(k) \\ \hat{y}_c(k) &= C^d x^d(k) \end{aligned}$$

A sampled-data system can be decomposed into four components. These components are shown in Figure 4.3 (a), are: the plant, which is to be controlled, the analogue-to-digital (A/D) converter, the digital device that implements the controller, and the digital-to-analogue (D/A) converter.

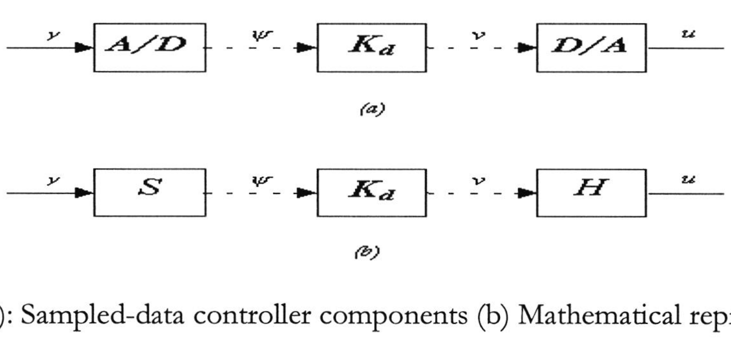


Figure 4.3 (a): Sampled-data controller components (b) Mathematical representation

The plant in sampled-data control systems is the continuous-time device to be controlled. The output of the plant, which it is to be controlled, is called the controlled variable. A regulator is one type of sampled-data control system and its purpose is to maintain the controlled variable at a preset value (animal heart rate, cardiac flow rate etc) or the process at a constant value. This input is known as the reference or set point. The second type of sampled-data system is a servomechanism whose purpose is to make the controlled variable follow an input variable.

The analogue-to-digital (A/D) converter changes the sampled signal into a binary number so that it can be used in calculations by the digital compensator. The word length (number of bits of resolution) of the A/D converter limits the fundamental precision of the control system as well as determines its maximum speed of operation. Typical word lengths are 10-12 bits but greater precisions are available for some applications. For example, a 10-bit A/D converter quantizes the analogue signal into 2^{10}, or 1024, discrete levels, which approximately 0.1% resolution. The conversion of a continuously valued signal into one of 2^n allowed values creates the equivalent of an additive noise called quantization noise. The word length of the A/D and subsequent computations is selected to keep this noise to an acceptable level. The A/D converter also sets the maximum speed of operation of sampled-data control system since it takes some time, usually microseconds, to effect the conversion. The Nyquist criterion requires that a system be sampled at greater than twice its maximum frequency component to in order to properly represent a signal. Since the A/D converter is used to convert plant feedback signals, it is the dynamics of the plant, which determine the minimum sampling rate and this is reflected in this section of the A/D converter conversion speed. Typical practical sampling rates are 8 to 10 times the maximum plant frequency. The mathematical representation of the A/D converter is the ideal sampler, *S* (Figure 4.4). It periodically samples $y(t)$ to yield the discrete-time signal $\psi(k)$. Let h denote the sampling period. Thus

$\psi(k) := y(kh)$. In general, $y(t)$ and $\psi(k)$ are both vectors, of the same dimension.

The digital controller shown in the typical sampled-data control system of Figure 7.1 takes the digitized value of the analogue feedback signals and combines them with the set point or desired trajectory signals to compute a digital control signal to actuate the plant through the D/A converter. A controller is used to modify the feedback signals in such a way that the dynamic performance of the plant is improved relative to some performance index. In Figure 4.2, K_d is a finite-dimensional (FD), linear time-invariant (LTI), causal, discrete-time system. Its input and output at time k are $\psi(k)$ and $v(k)$.

Since a digital controller computes the control signal used to drive the plant, a digital-to-analogue (D/A) converter must be used to change this binary number to an analogue voltage. The mathematical representation of the D/A converter is the hold operator, H (Figure 4.3). It converts the discrete-time signal v into the continuous-time signal $u(t)$ simply by holding it constant over the sampling intervals. Thus, $u(t) = v(k)$ for $kh \leq t < (k+1)h$. S and H are synchronized, physically by a clock. They are ideal system elements: S instantaneously samples its input; the output of H instantaneously jumps at the sampling instants.

Sampled-data control systems are designed by first developing a mathematical model for the plant or process to control. From this model, inherent capabilities can be computed and performance deficiencies identified. A sampling rate can be selected and a controller designed through well-established procedures in order to meet the desired performance measure.

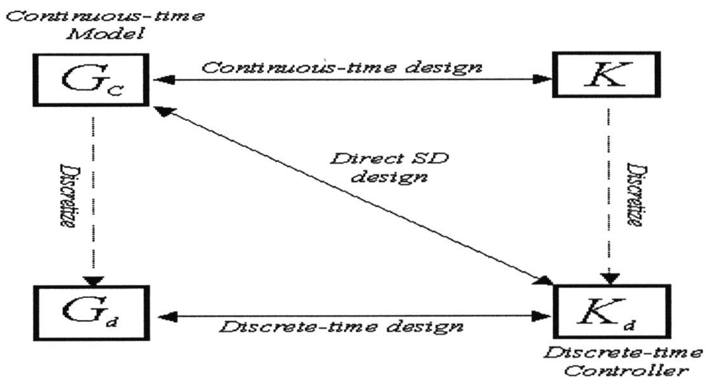


Figure 4.4: Approaches for the design of a discrete-time controller for a continuous-time process

Controllers can be designed (Figure 4.4) either through a direct digital approach operating on sampled-data signals, or they can be designed as continuous systems and then converted to sampled-data systems with some trial and error required in both cases to meet the desired performance goals.

4.3 Inter sample Behavior of Sampled Data System

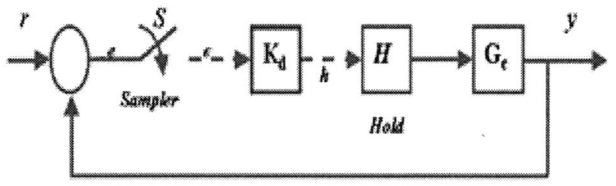


Figure 4.5: A Sampled Data Tracking System

Let's consider the plant, $G_c(s) = \frac{1}{(10s+1)(25s+1)}$

and the reference input r is the unit step. The goal is that the plant output 'y' should track input 'r' optimally.

To design the controller the plant need to be discretized first. Let's take the sampling period h=1s which is much smaller than the time constants of the plant (10 and 25s). The discretized plant has the transfer function:

$$\hat{G}_d(\lambda) = \frac{2.0960\times10^{-3}\lambda(\lambda+1.0478)}{(\lambda-1.0408)(\lambda-1.1052)}$$

$$= \left[\begin{array}{cc|c} 0.8675 & -0.0037 & 0.9325 \\ 0.9325 & 0.9981 & 0.4773 \\ \hline 0 & 0.0040 & 0 \end{array}\right]$$

And the discretized system is shown in figure .Here $\rho = Sr, \varepsilon = Se$ and $\psi = Sy$. Since r is the continuous time unit step, ρ is the discrete time unit step, 1_d. The optimal tracking problem is to design and LTI K_d to achieve internal stability and minimize $\|\varepsilon\|_2$. This performance criterion ignores intersample behavior: $\|\varepsilon\|_2$ could be small and yet $\|e\|_2$ could be large. This is an important point and it will be analyzed later. The formula for K_d is,

$$\hat{k}_d = \frac{\hat{q}}{1-\hat{G}_d\hat{q}}$$

The transfer function from ρ to ε is $1-G_d\hat{q}$

Thus $\hat{\varepsilon}(\lambda) = [1-G_d(\lambda)\hat{q}(\lambda)]\dfrac{1}{1-\lambda}$

So $\hat{q}(\lambda) = 1+(1-\lambda)\hat{q}_1(\lambda), \hat{q}_1(\lambda)$

And then $\hat{\varepsilon}(\lambda) = t_1 - \hat{t}_2\hat{q}_1$

where $\hat{t}_1(\lambda) = \dfrac{1-\hat{G}_d(\lambda)}{1-\lambda}$, $\hat{t}_2(\lambda) = \hat{G}_d(\lambda)$

The time domain equation is $\varepsilon = (T_1 - T_2Q_1)\omega$

where ω is the unit impulse. The problem is in the standard form namely,

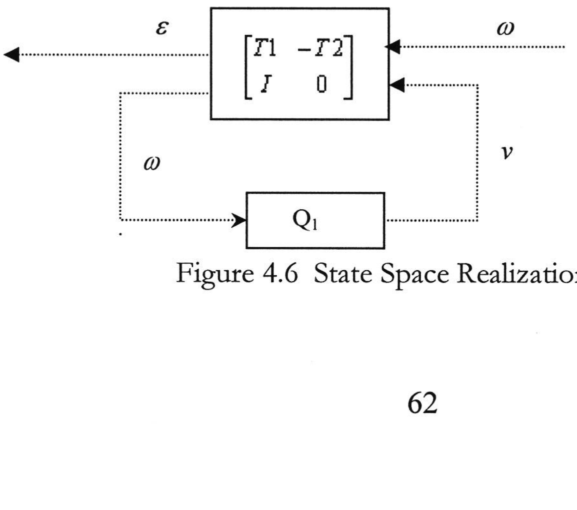


Figure 4.6 State Space Realization of the System

Now bring in realization of T_1 and $T_{2,}$

$$\hat{t}_1(\lambda) = \left[\begin{array}{c|c} A_{t1} & B_{t1} \\ \hline C_{t1} & D_{t1} \end{array}\right]$$

$$\hat{t}_2(\lambda) = \left[\begin{array}{c|c} A_{t2} & B_{t2} \\ \hline C_{t2} & D_{t2} \end{array}\right]$$

The induced realization for

$$G_{tmp} = \begin{bmatrix} T_1 & -T_2 \\ 0 & 0 \end{bmatrix}$$

is

$$\left[\begin{array}{c|cc} A & B_1 & B_2 \\ \hline C_1 & D_{11} & D_{12} \\ 0 & I & 0 \end{array}\right] = \left[\begin{array}{cc|cc} A_{t1} & 0 & B_{t1} & 0 \\ 0 & A_{t2} & 0 & B_{t2} \\ \hline C_{t1} & -C_{t2} & D_{t1} & -D_{t2} \\ 0 & 0 & I & 0 \end{array}\right]$$

For the data at hand, the numbers are

$$A = \begin{bmatrix} 0.8675 & -0.0037 & 0 & 0 \\ 0.9325 & 0.9981 & 0 & 0 \\ 0 & 0 & 0.8675 & -0.0037 \\ 0 & 0 & 0.9325 & 0.9981 \end{bmatrix}$$

$$B1 = \begin{bmatrix} 0.9325 \\ -249.5227 \\ 0 \\ 0 \end{bmatrix} \qquad B2 = \begin{bmatrix} 0 \\ 0 \\ 0.9325 \\ 0.4773 \end{bmatrix}$$

$C_1 = [0 \quad -0.0040 \quad 0 \quad -0.0040\ ]$, $D_{11}=1$, $D_{12}=0$

Since, $D_{12}=0$ But,

$$\|\varepsilon\|_2^2 = \|\varepsilon(0)\|^2 + \|\varepsilon\|_2^2$$

If ε denotes the state of $G_{tmp,}$ then $\varepsilon = C_1\xi + D_{11}\omega$

So $\varepsilon(0) = D_{11}\omega(0)$

Hence $\|\varepsilon\|_2^2 = \|D_{11}\omega(0)\|^2 + \|\varepsilon\|_2^2$

And an equivalent optimization problem is to minimize $\|\dot{\varepsilon}\|_2$.

For $K \geq 0$ we have

$$\begin{aligned}\varepsilon &= C_1\xi \\ &= C_1 A\xi + C_1 B_1 w + C_1 B_2 v\end{aligned}$$

Thus the equivalent problem pertains to the generalized plant

$$\left[\begin{array}{c|cc} A & B_1 & B_2 \\ \hline C_1 A & C_1 B_1 & C_1 B_2 \\ 0 & I & 0 \end{array}\right]$$

In this case, the values are

$$\begin{aligned}C_1 A &= [-0.0037 \quad -0.0040 \quad -0.0037 \quad -0.0040] \\ C_1 B_1 &= 0.9981 \\ C_1 B_2 &= -0.0019 \\ M &= (C_1 B_2)'(C_1 B_2) \\ &= 3.6451\times 10^{-6}\end{aligned}$$ and

$$A - B_2 M^{-1} B_2' C_1' C_1 A = \begin{bmatrix} 0.8675 & -0.0037 & 0 & 0 \\ 0.9325 & 0.9981 & 0 & 0 \\ -1.8219 & -1.9500 & -0.9544 & -1.9538 \\ -0.9325 & -0.9981 & 0 & 0 \end{bmatrix}$$

The latter matrix is singular; the generalized eigen problem for the sympletic pair is,

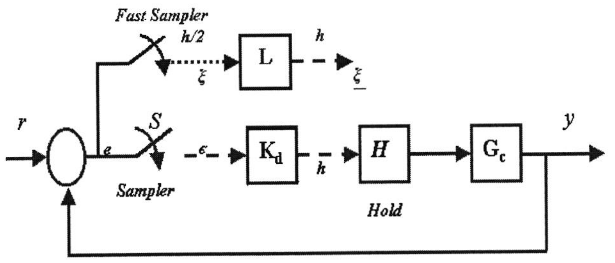
$$\begin{aligned}X &= 0_{4\times 4} \\ F &= -(M + B_2' X B_2)^{-1}(B_2' X A + B_2' C_1' C_1 A) \\ &= [-1.9538 \quad -2.0911 \quad -1.9538 \quad -2.0911] \\ F_0 &= -(M + B_2' X B_2)^{-1}(B_2' X B_1 + B_2' C_1' C_1 B) \\ &= 522.7759\end{aligned}$$

From theorem, $\hat{q}_1(\lambda) = \left[\begin{array}{c|c} A + B_2 F & B_1 + B_2 F_0 \\ \hline F & F_0 \end{array}\right]$

By back substitution the value of K$_d$ the optimal controller is,

$$\hat{k}_d(\lambda) = \frac{-477.1019(\lambda - 1.1052)(\lambda - 1.0408)}{(\lambda + 1.0478)(\lambda - 1)}$$

The plant K_d contain the pole at λ=1 required for step tracking. In addition it cancels all the stable poles and zeros of $\hat{G}_d$. For this controller, the sampled error ε is the impulse $\varepsilon = \delta_d$. The discretized system has a deadbeat response, the plant output ψ requiring only one discrete time step (I s in real time) to reach its final value. The simulated output of the given system is given in Figure 4.7.



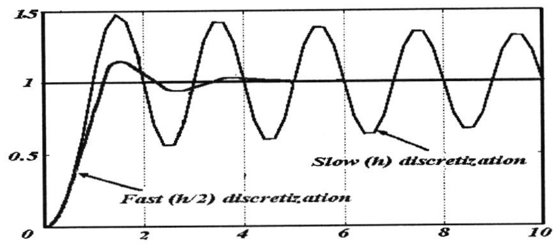


Figure 4.7 Effect of Slow and Fast Sampling

As it can be seen from the graph from if the signal is sampled at the sampling frequency of the sampler it shows ripple in the system. To solve this ripple problem signal is sampled with a frequency faster than the frequency of the controller. And the new controller can solve the ripple problem (as shown in graph). And the fast discretized controller is,

$$\hat{k}_d(\lambda) = \frac{-488.85(\lambda - 1.1052)(\lambda - 1.0408)}{(\lambda + 1.3955)(\lambda - 1)}$$

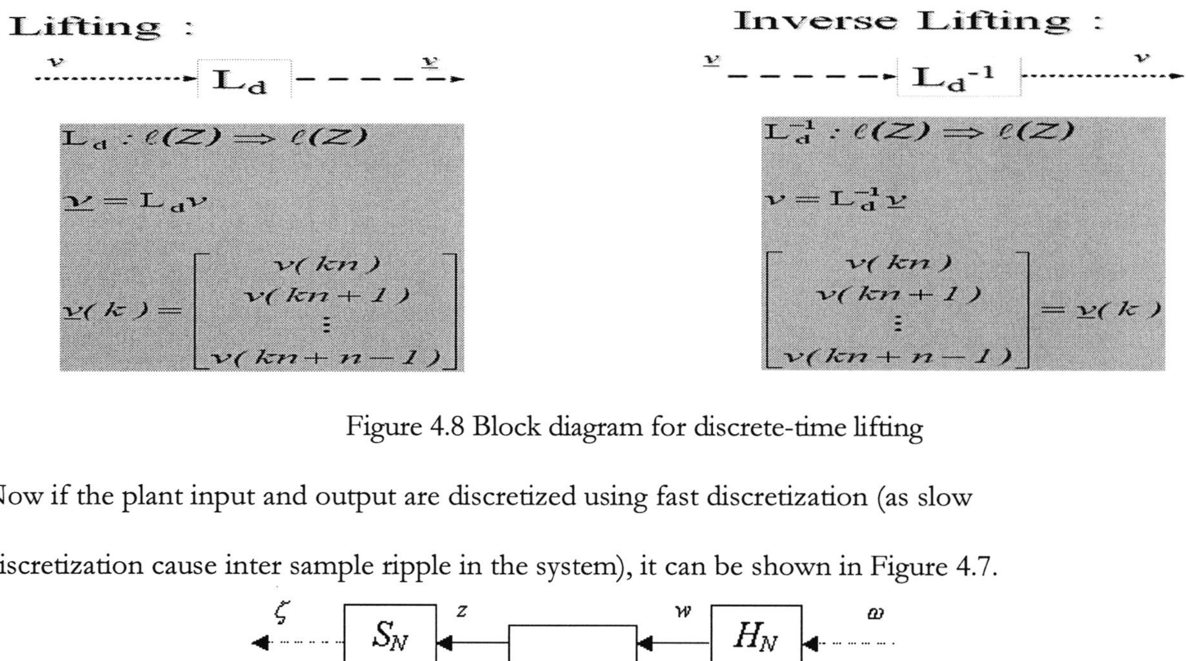
But it causes the multi rate in the system and the presence of multi rate in the system which makes the system time variant. Moreover multi rate in the system cause synchronization problem among input, output and controller and thus affect the system performance. To solve this problem lifting technique is used which converts the multi rate system into single rate and make the system time variant.

4.4 Lifting Techniques for Periodic Systems

The notion of lifting consists of the transformation of a periodic system to an equivalent discrete-time shift invariant one. That in turn enables the use of the tools of linear time-invariant systems theory for the analysis of periodic systems. The lifting technique is a very important developmental tool. There are two types of lifting, discrete-time and continuous-time. Here the system analysis is done in discrete time. Therefore discrete time lifting technique is applicable [44].

4.4.1 Discrete-time lifting

Discrete-time lifting is commonly used in multi-rate signal processing . By using lifting one can convert a multi-rate periodic system to a single-rate system. Lifting can be done in two ways: either slow rate sampling can be transferred into fast rate or fast rate sampling can be transferred into slow rate. Here, the signal v is considered with its fast sampling period h/N, and so the lifted associate $\underline{v}$ can be referred to period h. Thus the dimension of the lifted signal $\underline{v}$ is N times that of v. There exists the inverse of lifting which is causal but time-varying and is defined below.



$$\mathrm{L_d} : \ell(Z) \Rightarrow \ell(Z)$$
$$\underline{v} = \mathrm{L_d} v$$
$$\underline{v}(k) = \begin{bmatrix} v(kn) \\ v(kn+1) \\ \vdots \\ v(kn+n-1) \end{bmatrix}$$

$$\mathrm{L_d^{-1}} : \ell(Z) \Rightarrow \ell(Z)$$
$$v = \mathrm{L_d^{-1}} \underline{v}$$
$$\begin{bmatrix} v(kn) \\ v(kn+1) \\ \vdots \\ v(kn+n-1) \end{bmatrix} = \underline{v}(k)$$

Figure 4.8 Block diagram for discrete-time lifting

Now if the plant input and output are discretized using fast discretization (as slow discretization cause inter sample ripple in the system), it can be shown in Figure 4.7.

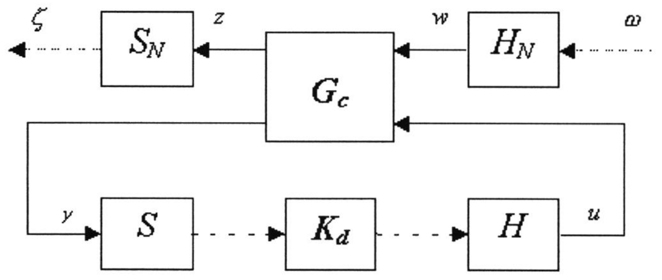


Figure 4.9 Fast discretization of SD system

Figure 4.9 displays a multi-rate SD system and the generalized plant $\boldsymbol{G_c}$ is continuous-time fast discretized linear time invariant (FDLTI) system with

$$G_c(s) = \begin{bmatrix} A & B_w & B_{u_c} \\ C_z & D_{zw} & D_{zu_c} \\ C_{y_c} & D_{y_c w} & D_{y_c u_c} \end{bmatrix}$$

and the controller $\boldsymbol{K_d}$ is discrete-time FDLTI. Samplers S and S_N are periodic of Periods h and h/N, respectively, and synchronized with them correspondingly are hold device H and H_N. This is an example of N-periodic systems for which the output shifts by N samples if the input does. Similar to the pure sampled-data case, discrete lifting can be used to associate an LTI system to this periodic system. First absorb the samplers and holds into the plant G_c and then introduce the discrete-time lifting operator and its inverse in this setup to get the setup in Figure 4.10. The system from $\underline{\omega}$ to $\underline{\zeta}$ is a single rate system.

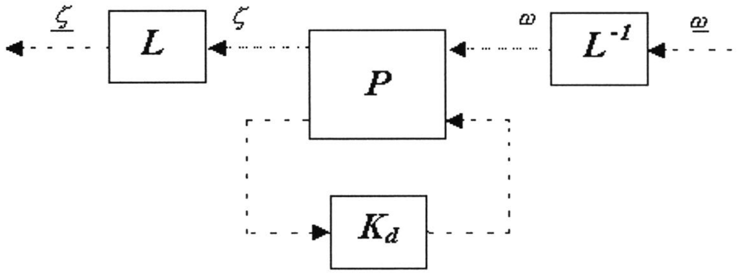


Figure 4.10 Two rate discrete system with lifting

Absorbing the lifting and its inverse into P as in Figure 4.11 where

$$\underline{P} := \begin{bmatrix} L & \\ & I \end{bmatrix} P \begin{bmatrix} L^{-1} & \\ & I \end{bmatrix}.$$

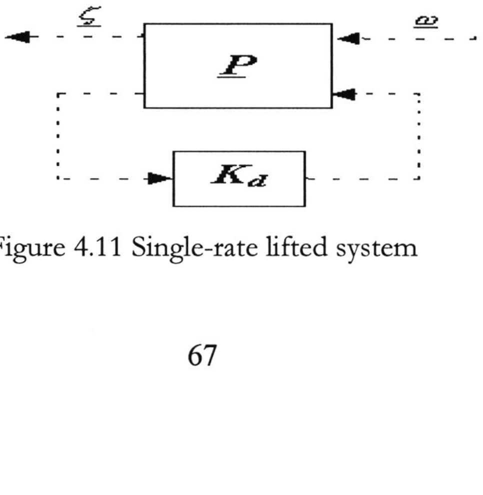


Figure 4.11 Single-rate lifted system

4.5 The Structure of Multi-Rate System

A general MRSD feedback system, $\Sigma[\boldsymbol{G_c}, H\boldsymbol{K_d}S\ ]$ to be considered in this section is shown in Figure 4.11. The plant $\boldsymbol{G_c}$ is FDLTI continuous-time system and the multi-rate sampled-data controller $H\boldsymbol{K_d}S$ is linear and causal, where $\boldsymbol{K_d}$ is the discrete-time controller. It is synchronized with S and H such that it inputs a value from the i-th channel at times lL_ih and outputs a value to the j-th channel at K_jh. A real number h is a basic sampling interval, l is the least common multiple and L_i, K_j are positive integers with S and H the multi-rate sampler and hold (zero-order) respectively.

$S\ := \text{diag}\ [\ S_{p_1h},\ S_{p_2h} \ldots\ S_{p_{i'}h}]$ $\qquad y = S\,y_c : y_i(l) = y_{ci}(lL_ih), \quad i = 1,2 \ldots i'$ and

$H\ := \text{diag}\ [\ H_{m_1h},\ H_{m_2h} \ldots\ H_{m_{j'}h}].\ u_c = H\,u : u_{cj}(k(K_jh)+t) = u_j(k), \quad 0<t\leq K_jh, \quad j= 1,2\ldots j'$

Here, output vector, $y_c = [y_{c1}, \ldots y_{cp}]^T$ and input vector, $u_c = [u_{c1}, \ldots, u_{cm}]^T$ are continuous-time signals and output vector, $y = [y_1, \ldots, y_p]^T$ and input vector $u = [u_1, \ldots, u_m]^T$ are discrete-time signals.

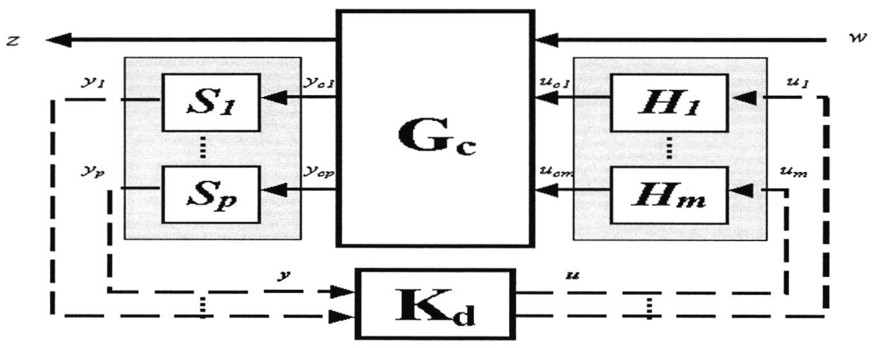


Figure 4.12: Multi-rate sampled-data system (periodic)

The continuous-time plant $\mathbf{G_c}$ of Figure 4.12 is supposed to have the following partition,

$$\begin{bmatrix} z \\ y_c \end{bmatrix} = \begin{bmatrix} G_{zw} & G_{zu_c} \\ G_{y_cw} & G_{y_cu_c} \end{bmatrix} \begin{bmatrix} w \\ u_c \end{bmatrix}$$

Assume that the transfer matrix from u_c to y_c is strictly proper which guarantees the existence of closed-loop transfer matrix. Then the state-space description of $\mathbf{G_c}$ with $D_{y_c u_c} = 0$ is,

$$\begin{bmatrix} \dot{x}(t) \\ z(t) \\ y_c(t) \end{bmatrix} = \begin{bmatrix} A & B_w & B_{u_c} \\ C_z & D_{zw} & D_{zu_c} \\ C_{y_c} & D_{y_c w} & 0 \end{bmatrix} \begin{bmatrix} x(t) \\ w(t) \\ u_c(t) \end{bmatrix}$$

where, $x(t) \in \Re^n$, $w(t) \in \Re^{m'}$, $u(t) \in \Re^m$ and $z(t) \in \Re^{p'}$ are the state vector, disturbance input, control signal input and controlled output respectively. In general, the state space realization of $\boldsymbol{G_c}$ is,

$$G_c(s) = \begin{bmatrix} A & B_w & B_{u_c} \\ C_z & D_{zw} & D_{zu_c} \\ C_{y_c} & D_{y_c w} & D_{y_c u_c} \end{bmatrix}$$

Let L_c be the continuous-time lifting operator (where $c=lh$) mapping a continuous signal to a discrete sequence taking values in $l_2\,[0, c)$. If L_p the p fold and L_m the m fold discrete-time lifting operator.

Multirate Sampler:

$$S := diag\,[S_{p_1h}, S_{p_2h} \ldots S_{p_ih}]$$
$$y := Sy_c : y_i(l) = y_{ci}(lL_ih),\ i = 1,2 \ldots i'$$

Multirate Hold:

$$H := diag\,[H_{m_1h}, H_{m_2h} \ldots H_{m_jh}]$$
$$u_c = H_u : u_q(k(k_jh) + t) = u_j(k), 0 < t < k_jh,\ j = 1,2 \ldots j'$$

Lifting Operator:

$$L_p := diag\,\{L_{\bar{p}_1} \ldots L_{\bar{p}_{i'}}\}$$
$$L_M := diag\,\{L_{\bar{m}_1}, \ldots L_{\bar{m}_{j'}}\}$$

Lifted Plant:

$$G = \begin{bmatrix} L_c & \\ & L_p S \end{bmatrix} G_c \begin{bmatrix} L_c^{-1} & \\ & \mathcal{H} L_M^{-1} \end{bmatrix}$$

Thus, the multi-rate system of Figure 4.12 is equivalent to single-rate system in Figure 4.13.

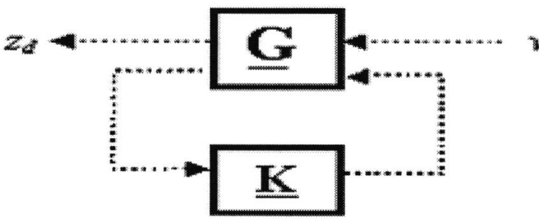


Figure 4.12 Single-rate LTI discrete system

In this chapter the mathematical model of the sampled data system is presented and the mathematical equation for plant was found. Based on the equation the controller for the system can be designed.

Chapter5

Conclusion and Future Work

In this thesis time delay analysis of biomedical data acquisition sampled data system is presented. Experiment on sensor is done and after proper interfacing with the USB-4716 data acquisition card the signal was shown in the developed GUI. Time delay of an open loop system is observed under multi tasking environment. It has been observed that the time delay in hard real-time Linux reduces compared to windows and soft real-time Linux. As in case of open loop case stability is not a major problem. So for open loop case time delay can be minimized to some extent by using hard real-time operating system. Although outputs from two channels are shown, for future work, more channels will be added.

The effect of time delay was analyzed for closed loop biomedical system. The total delay of the sampled data system was found both theoretically and practically for ECG and temperature measurement and it was found that delay in both cases were quite alike. It was found that in case of closed loop system time delay cause the stability problem and performance degradation of the system. To minimize the effect of time delay the controller was needed. A mathematical modeling of the controller was developed. Discrete time analysis of the plant is done and Lifting technique was used to convert the multi rate signal into single rate.

Possible future works include designing the controller for the system. As the example given in the system such as automatic heart pump, this sort of highly sophisticated biomedical system's controller designing is a very challenging task. The controller will not only control the intervened signal sent to human body but also take care of the stability of the system.

References

[1] Cardenas-Flores F, Benitez-Perez H, Garcia Nocetti F, "Study of Concurrent Queued System For Soft Real-Time Purpose: Bioengineering Case Study", Procedings of 4th IEEE Congress of Electronics, Robotics and Automotive Mechanics, 25th -28th Sep, 2007, pp. 68-73.

[2] E, Tavares, P. Maciel, B. Silva, "Modelling and Scheduling Hard Real-Time Biomedical System with Time and Energy Constraints", IEEE Electronic Letter, 2007, vol. 43, no 19. pp.1015-1017.

[3] Shahriar Iravanian and David J. Christini "Optical Mapping System with Real-Time Control Capability", American Journal of Physiology- Heart and Circulatory Physiology, 2007, pp. H2605-H2611

[4] John-Olof Nilsson, Issac Skog and Peter Handel, " Joint State and Measurement Time Delay Estimation of Non Linear State Space System", Procedings of 10th International Conference on Information AScience, Signal Processing and their Application (ISSPA 2010), pp. 324-328.

[5] Hwang-Cheng Chow, Wan-Tin Lin, "High Resolution Sucessive Approximation ADC for Low Power Biomedical Applications", Procedngs of International Conference on Advances Science and Contemporary Engineering (ICASCE 2012), 2012, vol. 50, pp. 275-283.

[6] K. Vinecore, et al , "Design and Implementation of a Portable Physiologic Data Acquisition System," Pediatric Critical Care Medicine, vol. 8, 2007, pp. 563-569.

[7] Hisaya Fujioka, "A Discrete-Time Approach to Stability Analysis of Systems with Aperiodic Sample and Hold Devices, "Procedings of IEEE Transaction on Automatic Control, October 2009, vol. 54, pp 2440-2445.

[8] Alexandre Seuret, "Stability Analysis for Sampled Data System with a time varying period", Procedings of 48th IEEE Conference on Decision and Control, Shanghai, China, December 16-18, 2009, pp. 8130-8135.

[9] A.M. Azad, "Multi Rate Sampled Data System with Decentralized Control Structure", Procedings of The IET China Ireland International Conference on Information and Communication Technologies, CIICT 2007", pp. 129-136.

[10] Mehnaz Akhter Khan, " Development of a Low Cost Microcontroller and PC Based Patient Monitoring System for Intensive Care Unit of Hospitals", a thesis submitted to the Department of Electrical and Electronic Engineering of BUET in partial fulfilment of the Requirements for the degree of M. Sc, 2008.

[11] Data Display, Acquisition and Feedback System for Biomedical Experiments", A Major Qualifying Project Report Submitted to the Faculty of the Worcester Polytechnique Institute, in partial fulfillment of the requirement for the Degree of Bachelor of Science by Patrick J. Bonneau, 2006.

[12] A. D. Dorval, D. J. Christini and J. A. White, "Real-Time linux dynamic clamp: A fast and flexible way to construct virtual ion channel in living cells," Annals of Biomedical Engineering, 2001,vol. 29, pp. 897-907.

[13] W. C. Kao, W. H. Chen, C. K. Yu, C. M. Hong, and S. Y. Lin, " A real time system for portable homecare applications," in Proc. 9th Int. Symp.Consum. Electron. (ISCE 2005), vol. 14, pp. 369-374.

[14] Xiang Feng, "Towards Real-Time Enabled Microsoft Windows" Procedings of the 5th ACM International Conference on Embedded Software, 2005, pp. 142-146.

[15] http://support.microsoft.com

[16] "Introduction to Linux for Real-Time Control", Introductrory Guidelines and References for Control Engineers and Managers by National Institute of Standards and Technology.

[17] J. W. S. Liu, Real-Time Systems. Englewood Cliffs, NJ: Prentice-Hall, 2000.

[18] L. Cromwell, Weibel, and Pfeiffer, Biomedical Instrumentation and Measurements. Pearson Education, 1980.

[19] G. L. Cote, M. Lec, M. V. Pishko, " Emerging Biomedical Sensing Technologies and Their Applications", procedings of IEEE Sensors Journal, 2003, vol. 3, pp. 251-266.

[20] The Biomedical Engineering Handbook, edited by J. D. Bronzino, Boc Raton, Fl: CRC, 1995.

[21] http://www.adinstruments.com

[22] Kao and T.L.J. Hwang, eds., "Computer Analysys of the Electrocardiograms from ECG Paper Recordings", Procedings of IEEE Conference on Engineering in Medicine and Biology Society, 2001, vol.4.

[23] Documentation on "Precision Signal Conditioning in Portable Pulse Oximeter Application", by National Semiconductor.

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

[25] K. Cronin, "Temperature taking in the ICU: which route is best?" Australian Critical Care, 2000, vol. 13, pp. 59-64.

[26] http:/www.temperatures.com/sensors.html.

[27] Joseph J. Carr , John M Brown, "Introduction to Biomedical Equipment Technology", 4th Edition, Pearson Education.

[28] Leslie Cromwell, Fred J. Weibell, Erich A. Pfeiffer, "Biomedical Instrumentation and Measurements", 2nd Edition.

[29] Amit K Gupta, Application Report on "Respiration Rate Measurement Using Impedance Pneumography", February, 2011

[30] User Manual of USB-4716 (Enhanced Multi-Lab Card) by Advantech (Automation with PCs).

[31] J. Hossain, K. Razin, M. Hasan, " Data Processing Through Biosensors and Development of Simulation Software in Windown and RT-Linux", A thesis submitted to Electrical and Electronic Engineering Department, BRAC University.

[32] M. B. Yehuda, "Introduction to Linux Device Drivers". IBM Hafia Research Labs and Haifux, January, 2005.

[33] A. A. Moshi, S. S. Cynthia, E. Islam, R. Rahman and A. M. Azad, "Performance Analysis of Robotic Arm Manipulators Control System Under Multitasking Environment". Procedings of IEEE 18th International Conference on Industrial Engineering and Engineering Management (IE&EM), 3rd -5th September, 2011, pp. 613-617.

[34] Karl J. Astrom, Bjorn Wittenmark, "Computer Control Systems: Theory and Design", 3rd Edition, Prentice Hall.

[35] Nilsson, J.(1998): Real-Time Control Systems with Delays. PhD thesis ISRN LUTFD2/TFRT—1049—SE, Department of Automatic Control, Lund Institute of Tchnology, Lund Sweden.

[36] Björn Wittenmark , Ben Bastian , Johan Nilsson, "Analysis of Time Delays in Synchronous and Asynchronous Loops", Procedings of 37th IEEE Conference on Decision and Control.

[37] Sheldon M. Ross, "Introduction to Probablity Models" 10th Edition, Elsevier

[38] Sheldon M. Ross, " Probablity and Statistics for Engineers and Scientists", 4th Edition, Elsevier

[39] Anton Cervin, "Stability and Worst Case Performance Analysis of Sampled Data Control System with Input and Output Jitter", Procedings of American Control Conference (ACC), 27th -29th June, 2012, pp. 3760 – 3765.

[40] Li, Pengfei ,Wang, Yannian , Zhou, Wei, "Performance Analysis of Hybrid Selection and Closed-loop Transmit Diversity Systems in the Presence of Feedback Delay", Procedings of IET Conference on Wireless, Mobile and Sensor Networks, 12th-14th December, 2012, pp.1051-1054.

[41] Seuret, A, Simon, D. " Robust Control Under Weakened Real-Time Contraints", Procedings of 50th International Conference on Decision and Control, 12th -15th December, 2011, pp. 2016-2021.

[42] Junmin Wang, Raul G. Longria "Effect of Computational Delay on the performance of a Hybrid Adaptive Cruise Control System", SAE World Congress 3rd-6th April, 2006.

[43] Kawka, P.A. " Stability and Performance of Packet Based Feedback Control Over a Markov Channel", Procedings of American Control Conference, 2006, 14th -16th June, 2006.

[44] T. Chen and B. Francis, "Optimal Sampled-Data Control Systems, Springer", 1995.

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;
using System.IO;


namespace Biomed_Control_Panel_v2
{

  public partial class MainForm : Form
  {
    int x = 0, y = 0;
    private bool isStop = false;
    Bitmap bmp = new Bitmap(1024, 768);
    private FileStream fs;
    private StreamWriter file;
    GraphForm gf;

    public MainForm()
    {
      InitializeComponent();

    }

    private void cmdSelectDevice_Click(object sender, EventArgs e)
    {
      // selecting between the different model of usb daq devic (imp)
      axAdvAI1.SelectDevice();
      axAdvAI1.SetValueRange(0, -10, +10);
      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 - 3.16) / 0.042, 2);
            //double Temperature = axAdvAI1.DataAnalog;
            // 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";
            this.file.WriteLine(DateTime.Now.ToString("hh:mm:ss.ffff") + "\t" +
Temperature.ToString() + "\t" + axAdvAI1.DataAnalog.ToString());
            gf.setData(Temperature);
        }

        private void btnStart_Click(object sender, EventArgs e)
        {
            if (!this.isStop)
            {

                this.isStop = true;
                this.btnStart.Text = "Stop";


                this.file = new StreamWriter("logs\\"+DateTime.Now.ToString("dd-MM-yy H-
mm-ss") + ".txt", true);
               this.file.WriteLine("TIME\t\tTMP\tANALOG DATA");


               gf = new GraphForm();
               gf.Show();

                timer1.Enabled = true;
                timer1.Start();
            }
            else
            {
                this.isStop = false;
                this.btnStart.Text = "Start";
                timer1.Stop();
                this.file.Flush();
                this.file.Close();
            }
        }

        private void pictureBox1_Click(object sender, EventArgs e)
        {

        }
    }
}
```

Graph form.cs

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


namespace Biomed_Control_Panel_v2
{
    public partial class GraphForm : Form
    {
        private Timer tm;
        private PointPairList list;
        private int totalInGraph = 36;
        private GraphPane myPane1;
        private GraphPane myPane2;

        public GraphForm()
        {
            InitializeComponent();
        }

        private void GraphForm_Load(object sender, EventArgs e)
        {
            // Get a reference to the GraphPane instance in the ZedGraphControl
            myPane1 = zg1.GraphPane;
            myPane2 = zg2.GraphPane;
        // Set the titles and axis labels
        myPane1.Title.Text = "Temperature";
        myPane1.XAxis.Title.Text = "Time, Second";
        myPane1.YAxis.Title.Text = "Temperature";
        ///***myPane.Y2Axis.Title.Text = "Parameter B";

        // Set the titles and axis labels2
        myPane2.Title.Text = "Demonstration of Dual Y Graph 2";
        myPane2.XAxis.Title.Text = "Time, Second";
        myPane2.YAxis.Title.Text = "ECG";

        // Make up some data points based on the Sine function
        this.list = new PointPairList();
        ///***PointPairList list2 = new PointPairList();
    /*      for (int i = 0; i < this.totalInGraph; i++)
        {
            double x = (double)i * 5.0;
            double y = Math.Sin((double)i * Math.PI / 15.0) * 16.0;
            ///***double y2 = y * 13.5;
            list.Add(x, y);
            ///***list2.Add( x, y2 );
        }
     *
     * */

        // Generate a red curve with diamond symbols, and "Alpha" in the legend
        LineItem myCurve = myPane1.AddCurve("Alpha",
            list, Color.Red, SymbolType.Diamond);
        // Fill the symbols with white
        myCurve.Symbol.Fill = new Fill(Color.White);

        LineItem myCurve2 = myPane2.AddCurve("Gamma", list, Color.Blue,
SymbolType.Square);
        myCurve2.Symbol.Fill = new Fill(Color.White);


        ///***// Generate a blue curve with circle symbols, and "Beta" in the legend
        ///***myCurve = myPane.AddCurve( "Beta",
        ///***list2, Color.Blue, SymbolType.Circle );
        // Fill the symbols with white
        ///***myCurve.Symbol.Fill = new Fill( Color.White );
        // Associate this curve with the Y2 axis
        ///***myCurve.IsY2Axis = true;

        // Show the x axis grid
        myPane1.XAxis.MajorGrid.IsVisible = true;
        myPane2.XAxis.MajorGrid.IsVisible = true;
        // Make the Y axis scale red
        myPane1.YAxis.Scale.FontSpec.FontColor = Color.Red;
        myPane1.YAxis.Title.FontSpec.FontColor = Color.Red;
        myPane2.YAxis.Scale.FontSpec.FontColor = Color.Red;
        myPane2.YAxis.Title.FontSpec.FontColor = Color.Red;

        ///***// turn off the opposite tics so the Y tics don't show up on the Y2 axis
        ///***myPane.YAxis.MajorTic.IsOpposite = false;
        ///***myPane.YAxis.MinorTic.IsOpposite = false;
        // Don't display the Y zero line
        myPane1.YAxis.MajorGrid.IsZeroLine = false;
        myPane2.YAxis.MajorGrid.IsZeroLine = false;
        // Align the Y axis labels so they are flush to the axis
        myPane1.YAxis.Scale.Align = AlignP.Inside;
        myPane2.YAxis.Scale.Align = AlignP.Inside;
        // Manually set the axis range
        myPane1.YAxis.Scale.Min = 20;// -10;// -2;// 0;
        myPane1.YAxis.Scale.Max = 45;// 10;// 2;// 120;
        myPane2.YAxis.Scale.Min = -.02;//-2;// 0;
        myPane2.YAxis.Scale.Max = .02;//2;// 120;
        myPane1.XAxis.Scale.Min = 0;
        myPane1.XAxis.Scale.Max = 20;// 60;
        myPane2.XAxis.Scale.Min = 0;
        myPane2.XAxis.Scale.Max = 20;// 60;

        ///***// Enable the Y2 axis display
        ///***myPane.Y2Axis.IsVisible = true;
        ///***// Make the Y2 axis scale blue
        ///***myPane.Y2Axis.Scale.FontSpec.FontColor = Color.Blue;
        ///***myPane.Y2Axis.Title.FontSpec.FontColor = Color.Blue;
        // turn off the opposite tics so the Y2 tics don't show up on the Y axis
        ///***myPane.Y2Axis.MajorTic.IsOpposite = false;
        ///***myPane.Y2Axis.MinorTic.IsOpposite = false;
        ///***// Display the Y2 axis grid lines
        ///***myPane.Y2Axis.MajorGrid.IsVisible = true;
        ///***// Align the Y2 axis labels so they are flush to the axis
        ///***myPane.Y2Axis.Scale.Align = AlignP.Inside;

        // Fill the axis background with a gradient
        myPane1.Chart.Fill = new Fill(Color.White, Color.LightGray, 45.0f);
        myPane2.Chart.Fill = new Fill(Color.White, Color.LightGray, 45.0f);

        // Add a text box with instructions
        TextObj text = new TextObj(
           "Zoom: left mouse & drag\nPan: middle mouse & drag\nContext Menu: right
mouse",
           0.05f, 0.95f, CoordType.ChartFraction, AlignH.Left, AlignV.Bottom);
        text.FontSpec.StringAlignment = StringAlignment.Near;
        myPane1.GraphObjList.Add(text);
        myPane2.GraphObjList.Add(text);

        // Enable scrollbars if needed
        zg1.IsShowHScrollBar = true;
        zg1.IsShowVScrollBar = true;
        zg1.IsAutoScrollRange = true;
        zg2.IsShowHScrollBar = true;
        zg2.IsShowVScrollBar = true;
        zg2.IsAutoScrollRange = true;
        ///***zg1.IsScrollY2 = true;

        // OPTIONAL: Show tooltips when the mouse hovers over a point
        zg1.IsShowPointValues = true;
        zg1.PointValueEvent += new
ZedGraphControl.PointValueHandler(MyPointValueHandler);
        zg2.IsShowPointValues = true;
        zg2.PointValueEvent += new
ZedGraphControl.PointValueHandler(MyPointValueHandler);

        // OPTIONAL: Add a custom context menu item
        zg1.ContextMenuBuilder += new
ZedGraphControl.ContextMenuBuilderEventHandler(
                MyContextMenuBuilder);
        zg2.ContextMenuBuilder += new
ZedGraphControl.ContextMenuBuilderEventHandler(
                MyContextMenuBuilder);

        // OPTIONAL: Handle the Zoom Event
        zg1.ZoomEvent += new ZedGraphControl.ZoomEventHandler(MyZoomEvent);
        zg2.ZoomEvent += new ZedGraphControl.ZoomEventHandler(MyZoomEvent);

        // Size the control to fit the window
        SetSize();

        // Tell ZedGraph to calculate the axis ranges
        // Note that you MUST call this after enabling IsAutoScrollRange, since
AxisChange() sets
        // up the proper scrolling parameters
        zg1.AxisChange();
        zg2.AxisChange();
        // Make sure the Graph gets redrawn
        zg1.Invalidate();
        zg2.Invalidate();

        ///////////////////
        tm = new Timer();
        tm.Interval = 1000;
        tm.Tick += new EventHandler(Timer_Tick);
        //   tm.Start();
          ///////////////////
        }
        private void Timer_Tick(object sender, EventArgs eArgs)
        {
          for (int i = 1; i < list.Count; i++)
            list[i - 1] = new PointPair(list[i - 1].X, list[i].Y);
          zg1.Invalidate();
          zg2.Invalidate();
        }

        /// <summary>
        /// On resize action, resize the ZedGraphControl to fill most of the Form, with a
small
        /// margin around the outside
        /// </summary>
        private void Form1_Resize(object sender, EventArgs e)
        {
          SetSize();
        }

        private void SetSize()
        {
          zg1.Location = new Point(10, 10);
          // Leave a small margin around the outside of the control
          zg1.Size = new Size(this.ClientRectangle.Width - 20,
               (this.ClientRectangle.Height - 30) / 2);

          zg2.Location = new Point(10, (this.ClientRectangle.Height - 30) / 2 + 20);
          // Leave a small margin around the outside of the control
          zg2.Size = new Size(this.ClientRectangle.Width - 20,
               (this.ClientRectangle.Height - 30) / 2);
        }

        public void setData(double y)
        {
          if (list.Count < this.totalInGraph)
          {
            list.Add((double)list.Count, y);
          }
          else
          {
            for (int i = 1; i < list.Count; i++)
              list[i - 1] = new PointPair(list[i].X, list[i].Y);
            list[list.Count-1]=new PointPair((double)(list[list.Count-2].X)+1.0, y);

            myPane1.XAxis.Scale.Min++;
            myPane1.XAxis.Scale.Max++;
            myPane2.XAxis.Scale.Min++;
            myPane2.XAxis.Scale.Max++;

         }
         zg1.Invalidate();
         zg2.Invalidate();
      }


      /// <summary>
      /// Display customized tooltips when the mouse hovers over a point
      /// </summary>
      private string MyPointValueHandler(ZedGraphControl control, GraphPane pane,
                  CurveItem curve, int iPt)
      {
         // Get the PointPair that is under the mouse
         PointPair pt = curve[iPt];

         return curve.Label.Text + " is " + pt.Y.ToString("f2") + " units at " +
pt.X.ToString("f1") + " days";
      }

      /// <summary>
      /// Customize the context menu by adding a new item to the end of the menu
      /// </summary>
      private void MyContextMenuBuilder(ZedGraphControl control, ContextMenuStrip
menuStrip,
                  Point mousePt, ZedGraphControl.ContextMenuObjectState objState)
      {
         ToolStripMenuItem item = new ToolStripMenuItem();
         item.Name = "add-beta";
         item.Tag = "add-beta";
         item.Text = "Add a new Beta Point";
         item.Click += new System.EventHandler(AddBetaPoint);

         menuStrip.Items.Add(item);
      }

      /// <summary>
      /// Handle the "Add New Beta Point" context menu item.  This finds the curve with
      /// the CurveItem.Label = "Beta", and adds a new point to it.
      /// </summary>
      private void AddBetaPoint(object sender, EventArgs args)
      {
         // Get a reference to the "Beta" curve IPointListEdit
         IPointListEdit ip = zg1.GraphPane.CurveList["Beta"].Points as IPointListEdit;
         if (ip != null)
         {
            double x = ip.Count * 5.0;
            double y = Math.Sin(ip.Count * Math.PI / 15.0) * 16.0 * 13.5;
            ip.Add(x, y);
            zg1.AxisChange();
            zg1.Refresh();
          }
        }

        // Respond to a Zoom Event
        private void MyZoomEvent(ZedGraphControl control, ZoomState oldState,
                ZoomState newState)
        {
          // Here we get notification everytime the user zooms
        }

    }
}
```

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.6 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
```

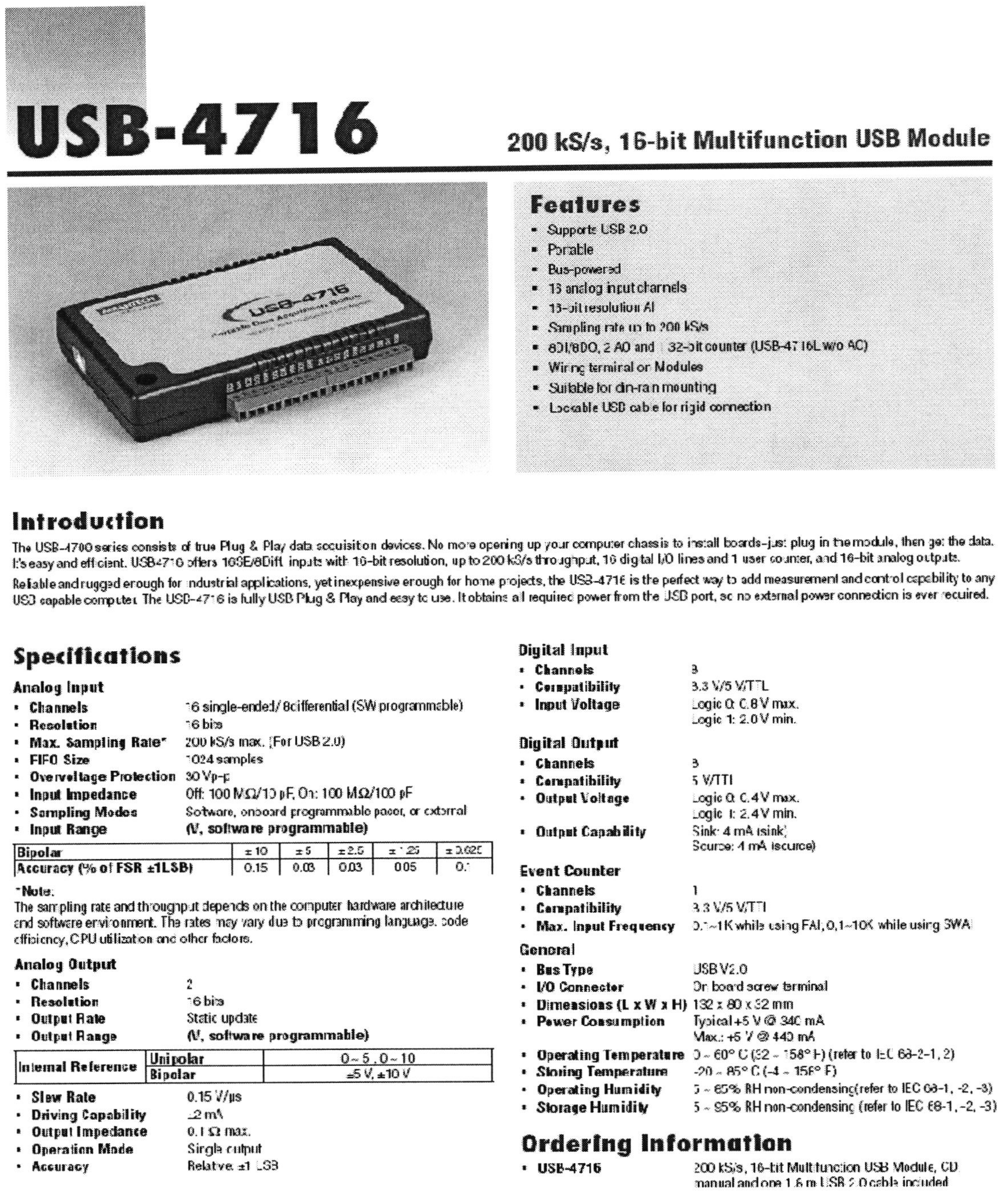
```
Voltage: 1.420593 V
Consumed Time: 424 micro second

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

B.2 Sample Data Set for Linux Time Measurement

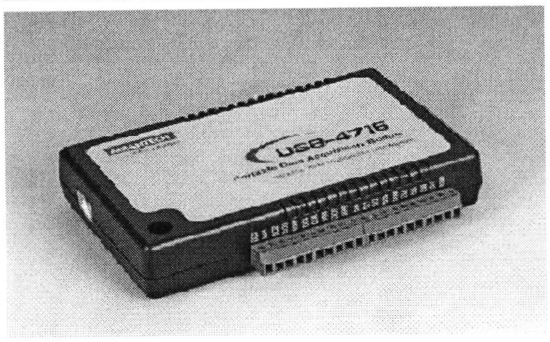
```
Start: BioMed Control Panel
File: /dev/advdaq0
Channel: 0
Range: -+10 V

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

Temperature: 26.39 C
Voltage: 1.249962 V
Consumed Time: 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
```

Appendix C

C.1 Main Features of USB-4716 Data Acquisition Card



USB-4716 **200 kS/s, 16-bit Multifunction USB Module**



Features
- Supports USB 2.0
- Portable
- Bus-powered
- 16 analog input channels
- 16-bit resolution AI
- Sampling rate up to 200 kS/s
- 8DI/8DO, 2 AO and 1 32-bit counter (USB-4716L w/o AO)
- Wiring terminal on Modules
- Suitable for din-rail mounting
- Lockable USB cable for rigid connection

Introduction
The USB-4700 series consists of true Plug & Play data acquisition devices. No more opening up your computer chassis to install boards-just plug in the module, then get the data. It's easy and efficient. USB-4716 offers 16SE/8Diff inputs with 16-bit resolution, up to 200 kS/s throughput, 16 digital I/O lines and 1 user counter, and 16-bit analog outputs.

Reliable and rugged enough for industrial applications, yet inexpensive enough for home projects, the USB-4716 is the perfect way to add measurement and control capability to any USB capable computer. The USB-4716 is fully USB Plug & Play and easy to use. It obtains all required power from the USB port, so no external power connection is ever required.

Specifications

Analog Input

Channels	16 single-ended/ 8 differential (SW programmable)
Resolution	16 bits
Max. Sampling Rate*	200 kS/s max. (For USB 2.0)
FIFO Size	1024 samples
Overvoltage Protection	30 Vp-p
Input Impedance	Off: 100 MΩ/10 pF, On: 100 MΩ/100 pF
Sampling Modes	Software, onboard programmable pacer, or external
Input Range	**(V, software programmable)**

Bipolar	±10	±5	±2.5	±1.25	±0.625
Accuracy (% of FSR ±1LSB)	0.15	0.03	0.03	0.05	0.1

***Note:**
The sampling rate and throughput depends on the computer hardware architecture and software environment. The rates may vary due to programming language, code efficiency, CPU utilization and other factors.

Analog Output

Channels	2
Resolution	16 bits
Output Rate	Static update
Output Range	**(V, software programmable)**

Internal Reference	**Unipolar**	0 ~ 5 , 0 ~ 10
	Bipolar	±5 V, ±10 V

Slew Rate	0.15 V/µs
Driving Capability	±2 mA
Output Impedance	0.1 Ω max.
Operation Mode	Single output
Accuracy	Relative ±1 LSB

Digital Input

Channels	8
Compatibility	3.3 V/5 V/TTL
Input Voltage	Logic 0: 0.8 V max. Logic 1: 2.0 V min.

Digital Output

Channels	8
Compatibility	5 V/TTL
Output Voltage	Logic 0: 0.4 V max. Logic 1: 2.4 V min.
Output Capability	Sink: 4 mA (sink) Source: 4 mA (source)

Event Counter

Channels	1
Compatibility	3.3 V/5 V/TTL
Max. Input Frequency	0.1~1K while using FAI, 0.1~10K while using SWAI

General

Bus Type	USB V2.0
I/O Connector	On board screw terminal
Dimensions (L x W x H)	132 x 80 x 32 mm
Power Consumption	Typical +5 V @ 340 mA Max.: +5 V @ 440 mA
Operating Temperature	0 ~ 60° C (32 ~ 158° F) (refer to IEC 68-2-1, 2)
Storing Temperature	-20 ~ 85° C (-4 ~ 185° F)
Operating Humidity	5 ~ 85% RH non-condensing (refer to IEC 68-1, -2, -3)
Storage Humidity	5 ~ 95% RH non-condensing (refer to IEC 68-1, -2, -3)

Ordering Information

USB-4716	200 kS/s, 16-bit Multifunction USB Module, CD manual and one 1.8 m USB 2.0 cable included