12-2-2002

Remote experimental station for engineering education

Muralidhar Doddapuneni
Florida International University

DOI: 10.25148/etd.FI15101207

FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

REMOTE EXPERIMENTAL STATION FOR ENGINEERING EDUCATION

A thesis submitted in partial fulfillment of the

requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

by

Muralidhar Doddapuneni

2002

To: Dean Vish Prasad
College of Engineering

This thesis, written by Muralidhar Doddapuneni, and entitled Remote Experimental Station for Engineering Education, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this thesis and recommend that it be approved.

Malcolm Heimer

Tadeusz M. Babij

Subbarao V. Wunnava, Major Professor

Date of defense: December 2, 2002

The thesis of Muralidhar Doddapuneni is approved.

Dean Vish Prasad
College of Engineering

Dean Douglas Wartzok
University Graduate School

Florida International University, 2002

DEDICATION

I dedicate this thesis to my parents Vijaya Kumari and Krishnaiah, my sister Neelima and my friend Vidya. Without their patience, understanding, support, and most of all love and caring, this work would not have been possible.

ACKNOWLEDGMENTS

I wish to express my sincere gratitude to my major professor, Dr. Subbarao Wunnava, for his continued support and for providing me with remarkable opportunities throughout my masters. I would like to thank my committee members Dr. Tadeusz Babij and Dr Malcolm Heimer for contributing their valuable time and experience.

I extend my gratitude to HSTN lab members for their kindness, friendship and support. Finally I would like to thank Pat Brammer in Electrical and Computer engineering for all her help through out my masters.

ABSTRACT OF THE THESIS

REMOTE EXPERIMENTAL STATION FOR ENGINEERING EDUCATION

by

Muralidhar Doddapuneni

Florida International University, 2002

Miami, Florida

Professor Subbarao V. Wunnava, Major Professor

This thesis provides a distance-learning laboratory for students of electrical and computer engineering department where the instructor can conduct experiments on a computer and send the results to the students at remote computers. The output of the experiment conducted by the instructor is sampled using a successive approximation Analog to Digital (A/D) converter. A microcontroller collects samples using high speed queued serial peripheral interface clock and transmits data to IBM-compatible personal computer over a serial port interface, where the samples are processed using Fast Fourier Transforms and graphed. The client/server application developed transfers the acquired samples over Transmission Control Protocol/Internet Protocol (TCP/IP) network with operational Graphical User Interface (GUI) to the remote computers where the samples are processed and presented to students.

The application was tested on all Windows platforms and various Internet speeds (56k modem, Digital Subscriber Line (DSL), Local Area Network (LAN)). The results were analyzed and appropriate methodology of Remote Experimental Station was formulated.

TABLE OF CONTENTS

CHAPTER PAGE

LIST OF FIGURES

FIGURE PAGE

1.0 Introduction

Our industries face stiff competition in the current progress towards globalization of the economy, as manufacturers search for new ways to increase production at a lower cost with higher quality. To address these challenges, industries need highly skilled personnel tuned to modern technology. However, rising costs, reduced budgets, difficulties in retaining high quality students and lack of technical and laboratory resources are some of the challenges that beset engineering education. Further exigencies include timetable clashes and time constraints within a flexible semesterised system offering multiple options. Therefore, innovative teaching methods are necessary to circumvent some of these problems. An essential factor in student learning is motivation. Studies [1] note the limitations of traditional classroom teaching in today's changing environment.

Presenting the course material in an attractive manner to encourage their participation. For example: combining text, sound, graphics, interactivity and motion to animate technical concepts helps students understand concepts and enables them to do far more than through listening alone. Thus, a well-implemented hypermedia-enhanced curriculum could be an effective tool in technical education. The task must be perceived by students to be relevant to their needs as future professionals, and must include their direct involvement with problem solving and exploring possible options in plant conceptualization, design and operation. To overcome these difficulties, we have to develop a computer based instruction system with real-world case studies.

1.1 Web-Based Learning

A major shortcoming with conventional engineering education is the exigency of providing equipment and laboratory tools caused by rising costs and infrastructure requirements. Besides the difficulties in managing and maintaining units, additional problems arise in upgrading these facilities to continue providing leading edge education. We contend that it is now partly possible to facilitate higher level learning in practice-oriented courses with the availability of affordable computers and supporting software. Our early experience shows that in a highly dynamic and technologically fast-moving environment, web-based laboratory work may be a suitable option for sustainably providing high-level skills to engineering students.

From its inception, the web was recognized as a low cost, flexible, and platform-independent means for information exchange. In its infancy, web-based education relied on the distribution of static pages. In this delivery mode, the only advantages the web offered over its Internet predecessors, such as the News Groups and Gopher servers, were ease of use and the ability to embed graphical content. However, the current interactivity available with a web-based system provides significant advantages over these early versions.

1.1.1 Hypermedia

As the World Wide Web (WWW) evolves into an important instructional platform, educational hypermedia is gaining increasing attention. Hypermedia is made up of nodes that can contain text, graphics, audio, video, even entire programs, and is an open system that allows users to read from, append or write materials to shared structures. Consequently, educational hypermedia provides flexible means of accessing

instructional information and supports various learning styles. From an instructional perspective, a critical feature of hypermedia is that it provides a non-sequential information presentation that differs markedly from the text-based material used in conventional instructional systems. Sound structures and learning guidance, however, are as important as is navigational freedom. A hypermedia learning system must be able to assist students in determining their learning performance levels. A hierarchical learning environment may help students not only during non-sequential information searches and browsing, but also during knowledge construction and integration [1]. Thus, hypermedia courseware has the potential to offer distributed, interactive, student-centered learning. Students will have greater flexibility and control to study when and where they desire. The courseware will be interactive, adaptive and responsive to the pedagogical needs of the students. Classroom instruction can be transformed from primarily lecturer-based to predominantly student-centered. It is a well-established fact that students tend to retain more by interactive doing and reading. Coupled with hypermedia, the methodology of student learning is set to undergo fundamental transformations.

1.2 Classification of Computer-Based Education Systems

Computer-based educational systems are classified, depending on the main research focus, as follows [2]:

Computer-aided instruction system: Most educational systems using computers belong to this general category. It developed with the advances of computer and computing technologies and is being used in a broad term. They are more interested in and satisfied with transferring educational contents efficiently even if the implemented system and methodology are quite different from the real counterpart.

Multimedia/virtual laboratory: Using computers, advanced graphics, and multimedia technology, e.g. head-mounted displays, 3D sound, and artificial sensory devices, they would like to make a mockup of the real part as closely as possible.

Distance-learning system on the Web: On the Internet or intranet, the education is performed with less direct interference between the instructor and the trainee, usually by downloading and installing the necessary computer programs and/or connecting to the education server. The instructor becomes free from directly managing the education session compared with general computer-aided instruction systems.

Intelligent tutoring system: Intangible education method itself, especially the role of human instructor who is adaptively controlling the information transfer rate, is the objective to try to replace with the computer. How to react and proact to the trainee's changing knowledge level and how to interactively give only the necessary feedback are main focus of the research and development in this area.

1.2.1 Virtual Laboratory

A Virtual Laboratory is a heterogeneous, distributed problem solving environment that enables a group of researchers located around the world to work together on a common set of projects. As with any other laboratory, the tools and techniques are specific to the domain of the research, but the basic infrastructure requirements are shared across disciplines. Although related to some of the applications of tele-immersion, the virtual laboratory does not assume a priori the need for a shared immersive environment.

1.2.2 Components of Virtual Laboratory

- The components of a virtual laboratory include [4]:
- Computer servers capable of handling very large-scale simulations and data reductions.
- Data bases that contain application specific information such as simulation initial and boundary condition, experimental observations, customer requirements, manufacturing constraints, as well as distributed, application specific resources such as the human genome repositories.
- Scientific instruments that are connected to the network.
- Collaboration tools, sometimes including tele-immersion
- Software assets. (Each virtual laboratory is based around specialized software for simulation, data analysis, discovery and reduction, and visualization. Most of this software was originally designed for "stand-alone" use on a single machine.
- Tightly coupled, multi-disciplinary computations place great stress on network bandwidth. Low latency is critical and computer system resource scheduling must be coupled with bandwidth reservation services. Multicast protocols and technology are critical to the collaborative nature of an experiment in a virtual laboratory where people, resources, and computations are widely distributed. Information streams in these experiments might combine voice, video, real-time data streams from instruments, and large bursts of data from simulations and visualization sources. Fig 1.1 below shows you the basic system architecture of a virtual lab.

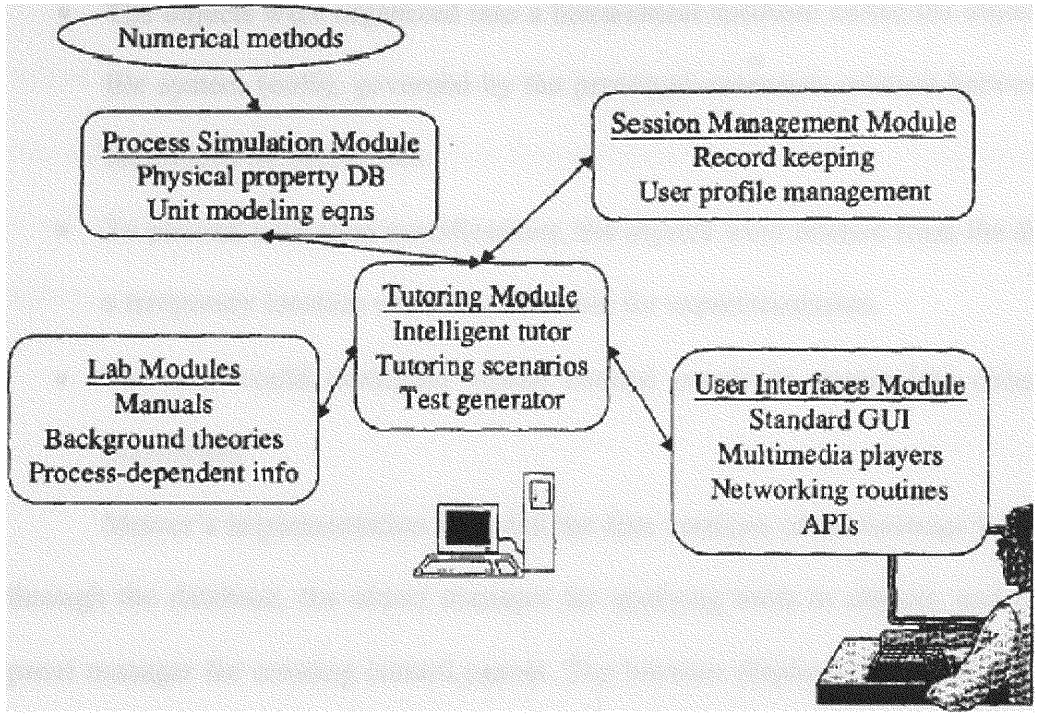

Figure 1.1. System Architecture [2]

1.3 Previous Work

Work on the Virtual Laboratory began in 1989, motivated by the need to organize a fast-growing database of experiments related to plant modeling and fractal generation Using L-systems. The original concept and design were introduced and related to previous research results by Mercer, Prusinkiewicz and Hanan, and detailed in Mercer's M.Sc. thesis. This work introduced several key elements of vlab design that remained essential to the subsequent implementations and extensions:

- Related data files were grouped into objects, complete with a specification file detailing which tools (programs with options and argument files) apply to this object.

- The objects were organized into a hierarchical database called the object-oriented file system (oofs), governed by the prototype-extension relation between objects introduced by Lieberman.
- To prevent unwanted modifications, the objects were fetched from the database to a temporary location called the lab table for experimentation.
- The user could configure virtual control panels to manipulate chosen model parameters.

Mercer's implementation included the first versions of the browser for navigating through the database, the object manager for applying tools to objects, and the control panel manager for creating control panels. The browser displayed only a limited view of the database (current object and its immediate extensions) and did not provide adequate support for moving objects within the hierarchy [3].

1.4 Design Objective of The Thesis

With the rapid changes in technology and varying marketing conditions, the education system is challenged with providing increased educational opportunities without increase in budgets. Many educational institutions are answering this challenge by developing distance education programs.

The distance education takes place when a physical distance separates teacher and student, and technology (i.e. voice, video and data) often in concert with face-to-face communication, is used to bridge the instructional gap. These types of programs can provide adults with a second chance at a college education, reach those disadvantaged by limited time, distance or physical disability, and update the knowledge base of workers at their places of employment

On the other side, it is often difficult for students, especially for students who follow at-distance flavors of engineering schools or masters, to have access to a good laboratory over full time of the course. Remote experimental station would allow students to practice experiments in undergrad curriculum of Electrical and computer engineering major.

2.0 Design Considerations for Remote Experimental Station

The main purpose of this thesis activity is to provide a good remote laboratory for undergraduate students of Electrical and Computer Engineering department. In this chapter various design issues to develop Remote Experimental Station are discussed

2.1 Analog To Digital Converters

As the output from student experiment board in labs like (Circuits lab, Electronics I lab, Electronics II lab, Integrated Circuits lab) will be analog we will be working mostly with analog signals, we have to convert the acquired signal to digital samples for computer to understand. So we have to choose a better a/d converter from widely available varieties by considering factors like cost, complexity, efficiency and reliability

2.1.1 Successive Approximation Converter

A successive approximation converter provides a fast conversion of a momentary value of the input signal. It works by first comparing the input with a voltage which is half the input range. If the input is over this level it compares it with three-quarters of the range, and so on. Twelve such steps gives 12-bit resolution. While these comparisons are taking place the signal is frozen in a sample and hold circuit. After A-D conversion the resulting bytes are placed into either a pipeline or buffer store. A pipeline store enables the A-D converter to do another conversion while the previous data is transferred to the computer. Buffered A-D converters place the data into a queue held in buffer memory [7]. The computer can read the converted value immediately, or can allow values to accumulate in the buffer and read them when it is convenient. This frees the computer from having to deal with the samples in real time, allowing them to be processed in convenient batches without losing any data.

2.1.2 Dual Slope Integrating Converter

This converter reduces noise but is slower than the successive approximation type. It lets the input signal charge a capacitor for a fixed period and then measures the time for the capacitor to fully discharge at a fixed rate. This time is a measure of the integrated input voltage, which reduces the effects of noise.

2.1.3 Charge Balancing Converter

The input signal again charges a capacitor for a fixed time, but in this converter the capacitor is simultaneously discharged in units of charge packets: if the capacitor is charged to more than the packet size it will release a packet, if not a packet cannot be released. This creates a pulse train. The input voltage is determined by counting the pulses coming out of the capacitor. Noise is reduced by integrating the input signal over the capacitor charging time.

2.1.4 Flash Converter

A flash converter is the fastest type of converter we use. Like the successive approximation converter it works by comparing the input signal to a reference voltage, but a flash converter has as many comparators as there are steps in the comparison. An 8-bit converter, therefore, has 2 to the power 8, or 256, comparators [7].

2.1.5 Sigma-Delta Converter

This converter digitizes the signal with very low resolution (1-bit) and a very high sampling rate (MHz). By over sampling, and using digital filters, the resolution can be increased to as many as 20 or more bits. Sigma-delta converters are especially useful for high-resolution conversion of low-frequency signals as well as low-distortion conversion of signals containing audio frequencies. They have good linearity and high accuracy.

2.2 Specifications of Analog-Digital Converters

Many types of specifications for A-D converters are quoted by hardware manufacturers. Here are some of the specifications, which are in practice, namely resolution, linearity, offset errors, sample and hold acquisition time, throughput, integration time and re-calibration.

2.2.1 Resolution

The resolution of the A-D converter is the number of steps the input range is divided into. The resolution is usually expressed as bits (n) and the number of steps is 2 to the power n. A converter with 12-bit resolution, for instance, divides the range into 2^{12}, or 4096, steps. In this case a 0-10 V range will be resolved to 0.25 mV, and a 0-100 mV range will be resolved to 0.0025 mV. Although the resolution will be increased when the input range is narrowed, there is no point in trying to resolve signals below the noise level of the system.

2.2.2 Linearity

Ideally an A-D converter with n-bit resolution will convert the input range into (2 to the power n)-1 equal steps (4095 steps in the case of a 12-bit converter). In practice the steps are not exactly equal, which leads to non-linearity in a plot of A-D output against input voltage.

2.2.3 Sample and Hold Acquisition Time

A sample and hold circuit freezes the analogue input voltage at the moment the sample is required. This voltage is held constant while the A-D converter digitizes it. The acquisition time is the time between releasing the hold state and the output of the sample

circuit settling to the new input voltage value. Sample and hold circuits are not used with integrating converters.

2.2.4 Throughput

The throughput is the maximum rate at which the A-D converter can output data values. In general it will be the inverse of the (conversion time + the acquisition time) of the A-D converter. Thus a converter that takes 10 microseconds to acquire and convert will be able to generate about 100 000 samples per second. Throughput can be increased by using a pipelined A-D converter, so a second conversion can start while the first is still in progress. Throughput may be slowed down, however, by other factors, which prevent data transfer at the full rate.

2.2.5 Integration Time

An integrating A-D converter measures the input voltage by allowing it to charge a capacitor for a defined period. The integration averages the input signal over the integration time, which if chosen appropriately will average over a complete mains cycle thereby helping to reduce mains frequency interference. The throughput of an integrating converter is not the inverse of the integration time, as throughput also depends on the maximum discharge time.

2.2.6 Re-Calibration

Some A-D converters are able to re-calibrate themselves periodically by measuring a reference voltage, and compensating for offset and gain drifts. This is useful for long term monitoring since drifts do not accumulate. If the re-calibrations are set too far apart there may appear to be small discontinuities in the recorded data as the re-calibrations occur. (If you have a reading other than zero for a zero condition, then you

have an offset error: every reading will be inaccurate by this amount. When the A-D converter is preceded by signal conditioning circuits offset errors need not normally be considered. Drift occurs because components in the amplifier change over time and with temperature. Drift is usually only significant for people trying to measure low-level signals a few millivolts over long periods of time or in difficult environmental conditions).

2.3 Interfacing to Computer

The samples acquired from a/d converter have to be transferred to the computer, for this we need a microprocessor which can store the digital samples in a buffer and send it to the computer as the speed of acquiring samples from a/d converter is different from the speed of transferring the samples to computer. The interfacing techniques that are widely used are serial port interface (RS-232 interface), parallel interface

2.3.1 Serial Port Interface

The serial communications are used for transferring data over long distances, because parallel communications requires too many wires. Serial data received from a modem or other devices are converted to parallel so that it can be transferred to the PC bus. The serial communications equipment can be divided into simplex, half-duplex and full duplex. A simplex serial communication sends information only in one direction (i.e. a commercial radio station). Half-duplex means that data can be send in either direction between two systems, but only in one direction at a time. In a full-duplex transmission each system can send and receive data at the same time. There are two ways to transmit serial data: synchronously or asynchronously. In a synchronous transmission data is sent

in blocks, the transmitter and the receiver are synchronized by one or more special characters called sync characters [10].

The serial port of the PC is an asynchronous device. For asynchronous transmission, a bit identifies its start and 1 or 2 bits identify its end, don't need any synchronization. The data bits are sent to the receiver after the start bit. The least significant bit is transmitted first. A data character usually consists of 7 or 8 bits. Depending on the configuration of the transmission a parity bit is send after each data bit. It is used to check errors in the data characters. Finally 1 or 2 stop bits are send.

Description of the port:

The serial port of the PC is compatible with the RS-232C standard. This standard was designed in the 1960s to communicate a data terminal equipment or DTE (the PC in this case) and a data communication equipment or DCE (usually a modem).

The standard specifies 25 signal pins, and that the DTE connector should be a male and the DCE connector should be a female. The most used connectors are the DB-25 male, but many of the 25 pins are not needed. For that reason in many modern PCs a DB-9 male connector is used. So you will find one or more of these connectors in the rear panel of the PC. The voltage levels are between -3V and -15V for logic high. A logic low is a voltage between +3V and +15V. The commonly used voltages are +12V and -12V.

The most commonly used signals are listed below:

DTR (Data-Terminal-Ready): The PC tells the modem that is powered up and ready to send data.

DSR (Data-Set-Ready): The modem tells the PC it is powered up and ready to transmit or receive data.

RTS (Request-To-Send): The PC sets this signal when has a character ready to be sent.

CD (Carrier-Detect): The modem sets this signal when has detected the computer.

CTS (Clear-To-Send): The modem is ready to transmit data. The computer will start sending data to the modem.

TxD: The modem receives data from de PC.

RxD: The modem transmits data to the PC.

The integrated circuits that convert the serial data lines to parallel and vice versa are called UART (Universal Asynchronous Receiver-Transmitter). The typical PC UART is the Intel 8251A, this IC can be programmed like a synchronous or an asynchronous device.

Eight data bits (D0-D7) connect the 8251A to the data bus of the PC. The chip select (/CS) input enables the IC when is asserted by de control bus of the PC system. This IC has two internal addresses, a control address and a data address. The control address is selected when the C-/D input is high. The data address is selected when the C/D input is low. The RESET signal resets the IC. When the /RD is low the computer reads a control or a data byte. The /WR enables the PC to write a byte. Both signals are connected to the system signals with the same names.

The UART includes four internal registers:

THR: Temporary output register.

TSR: Output register.

RDR: Input register.

RSR: Temporary input register.

Every character to be transmitted is stored in the THR register. The UART adds the start and stop bits. Then copies all bits (data, start and stop bits) to the TSR. To finish the process the bits are sent to the line by the TD signal.

Every character received from the line RD is stored in the RSR register. The start and stop bits are eliminated and the UART writes this character to the RDR. To finish the process the character is read for the PC.

Addressing the port.

There are two ways to address the serial port, by the 14H BIOS interrupt and by the 21H DOS interrupt.

The 14H BIOS interrupt uses four functions to program the serial port. Each function is selected assigning a value to the AH register of the microprocessor. We list the four functions below:

- Function 00H: Initializes the serial port and sets the speed, data and stop bits and the parity parameters.
- Function 01H: Sends a character to the specified serial port.
- Function 02H: Reads a character from the specified serial port.
- Function 03H: Returns the state of the specified serial port.

There are three functions in the 21H DOS interrupt related to the operation of the serial port:

- Function 03H: Reads a character from the COM1 serial port.
- Function 04H: Writes a character to the COM1 serial port.

- Function 40H: It is a common out function for all files and devices that use a handle access. This function sends a number of bytes from a buffer to the specified device.

2.3.2 Parallel Port Interface

A parallel interface is used for connecting an external device such as a printer. Most personal computers have both a parallel port and at least one serial port. On PCs, the parallel port uses a 25-pin connector (type DB-25) and is used to connect printers, computers and other devices that need relatively high bandwidth. It is often called a Centronics interface after the company that designed the original standard for parallel communication between a computer and printer [10].

The Parallel port is a standard designed to connect a printer to a computer. It is used for the CPU to send data to a printer. This interface drives some input and output signals. The purpose of these signals is to let the computer know the state of the printer and control it. Eight data bits carry all the information sent with each clock pulse. The hardware of this port consists of 8 output data bits, 5 input control bits and 5 output control bits. The control signals are listed below:

Outputs:

- STROBE: Tells the printer when the eight data bits are ready to be read. Turns to a low logic level when the data are ready.
- INIT: Reset the printer.
- SLCT IN: Selects the printer when it turns to a low logic level.
- AUTO FD: Tells the printer to print an empty line followed by a carriage return.
- D0-D7: Data bits.

Inputs:

- ACK: Tells the CPU that the data has been correctly received.
- BUSY: The printer sets this line when its buffer is full. The computer will stop sending more data.
- SLCT: Tells the computer that a printer is present.
- ERROR: An error has occurred. The CPU stop sending more data.
- PE: The printer is out of paper.

All these signals are connected to a 25 PIN connector. All the bits have TTL logic levels.

Addressing the port.

In the MS-DOS operative system three parallel ports, called LPT1, LPT2 and LPT3, are supported. So we can find three addresses dedicated to these ports in the memory map of the PC. Let's study the addresses dedicated to LPT1 first. Each parallel port uses three addresses of the I/O map. For LPT1 these addresses are 378H, 379H and 37AH.

- 378H PORT: In this address the CPU writes the data to be sent to the printer. It is an output port. The eight data bits (D0-D7) are latched to appear in the output connector. In the table 2.1 we can see which pins of the connector are used.
- 379H PORT: This is an input port. These signals are used by the CPU to know the state of the printer.
- 37AH PORT: In this port the computer writes the signals that control the printer.

3.0 Client/Server Model Using Winsock

3.1 Introduction

The Windows Sockets (also known as WinSock) specification defines a network-programming interface for Microsoft Windows, which is based on the "socket" paradigm popularized in the Berkeley Software Distribution (BSD) from the University of California at Berkeley. It encompasses both familiar Berkeley socket style routines and a set of Windows-specific extensions designed to allow the programmer to take advantage of the message-driven nature of Windows.

The Windows Sockets Specification is intended to provide a single API to which application developers can program and multiple networks software vendors can conform. Furthermore, in the context of a particular version of Microsoft Windows, it defines a binary interface (ABI) such that an application written to the Windows Sockets API can work with a conformant protocol implementation from any network software vendor. This specification thus defines the library calls and associated semantics to which an application developer can program and which a network software vendor can implement.

Network software, which conforms to this Windows Sockets specification, will be considered "Windows Sockets Compliant". Suppliers of interfaces which are "Windows Sockets Compliant" shall be referred to as "Windows Sockets Suppliers". To be Windows Sockets Compliant, a vendor must implement 100% of this Windows Sockets specification. Applications which are capable of operating with any "Windows Sockets Compliant" protocol implementation will be considered as having a "Windows Sockets Interface" and will be referred to as "Windows Sockets Applications".

This version of the Windows Sockets specification defines and documents the use of the API in conjunction with the Internet Protocol Suite (IPS, generally referred to as TCP/IP). Specifically, all Windows Sockets implementations support both stream (TCP) and datagram (UDP) sockets [16].

3.2 Client Server Model

Every network specification has a communication endpoint. There are two types of endpoints: clients and servers. By definition, a client sends the first packet, and a server receives it Fig 3.1. This is not the extension of their functionality, but is characterized by the role it plays during this initial communication.

It is possible for two network applications to begin simultaneously, but it is impractical to require it. It is very difficult to start two applications (programs) at the same instant, and the nature of a network-with varying traffic loads and the like-make the arrival of packet at each end unpredictable. For these reasons, we design each pair of network applications to perform complementary network operations in sequence, rather than simultaneously.

The server application executes first and waits to receive; the client executes second and sends the first network packet.

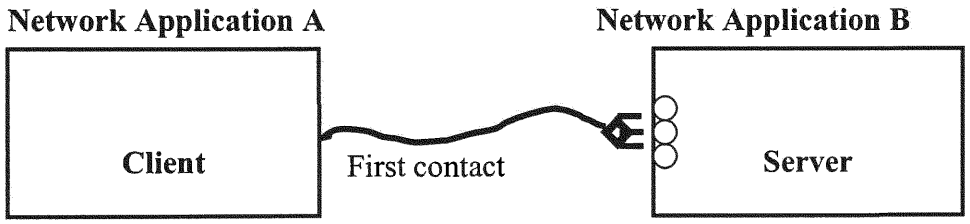

Figure 3.1. Client Server Application Model [10]

After initial contact, either the client or the server is capable of sending and receiving data. The initial contact is used to characterize their relationship only for the purpose of definition. The services these applications provide can be reverse this relationship any time after their first communication between each other.

3.3 Client And Server Model Association

Socket types are associated with the protocols they support. For two sockets to communicate as a client and server, they must have same socket types. Either both sockets must be stream (TCP) or both must be datagram (UDP), Client applications must be able to locate and identify a server's socket. A server application names its sockets to establish its identity, so clients can reference to it. A socket name (for TCP) consists of the IP address and port number, as well as the protocol. A client can use the Window Sockets service name functions to find out the standard service's port number, and a server's IP address is easy to find with Windows Sockets host name resolution functions if server's host name is known.

When the client socket successfully contacts a server socket, their two names combine to form an association. When one socket bonds with another to form an association, the association establishes the identity of both sockets. At that point, each socket is uniquely identified by the combination of its own name and that of its peer. This association has five elements [16]:

- Protocol (same for both client and server sockets)
- Client IP address
- Client port number

- Server IP address
- Server port number

For stream (TCP) a connection-oriented protocol, the life of an association corresponds directly to the creation and destruction of the TCP virtual circuit between a client and server. Since the datagram (UDP) socket is connectionless, the protocol does not clearly define the life of a UDP association. In theory, each datagram packet transmitted creates and destroys an association. But most UDP applications use the same association for the life of the socket.

3.4 Network Programming Sketch

There are five basic steps to network programming [10]:

- Open a socket.
- Name the socket.
- Associate with another socket.
- Close the socket.

The steps mentioned above are similar in operation to file I/O operations.

Table 3.1. Socket I/O Operation [10]

File I/O	Network I/O
Open a file	Open a socket
	Name the Socket
	Associate with another socket
Read and write	Send and receive between sockets
Close the file	Close the socket

In the table above, the first operation of opening file corresponds to three operations on a Socket (open, name and associate). A file already has a identity when it is opened, and system device is already associated with it. But the socket when opened has a incomplete identity. To complete this identity, a network application must assign it a name and associate it with another socket. One significant similarity between opening a socket and a file is that both operations return a handle, an arbitrary descriptor with which to access the opened resource. The point to note is that handles can make the source codes non- portable because they are non-equivalent on all operating systems[10].

3.5 Client Server Sketches

It is not possible to make a single network client and server pair of network programs that can be used for either datagram (UDP) or stream sockets(TCP), without some modification. This is not surprising, given the differences between the connection-oriented and connectionless paradigms. Below is a "bare-bone" superstructure for all Window Sockets Network applications, so it is worthwhile to present them [10].

Table 3.2. Connection- Oriented (TCP) Network Application [10]

Client	Server
Socket( )	Socket( )
Initialize sockaddr_in structure with server (remote) socket name	Initialize sockaddr_in structure with server (local) socket name
	bind( )
	listen( )
connect( )---------->	
<association created, either side can send or receive>	
send( ) ---------->	recv( )
recv( ) <----------	send( )
closesocket( )	closesocket( ) (connected socket)
	closesocket( ) (listening socket)

Table 3.3. Set the Remote Socket Name Once [10]

Client	Server
socket( )	socket( )
Initialize sockaddr_in structure With server (remote) socket name	Initialize sockaddr_in structure With server (local) socket name
	bind( )
connect( )	
send( ) ---------->	recv( )
<association created either side can send or receive>	
recv( ) <----------	send( )

Table 3.4. Set the Remote Socket Name for Each Datagram [10]

Client	Server
socket( )	socket( )
Initialize sockaddr_in structure With server (remote) socket name	Initialize sockaddr_in structure With server (local) socket name
	bind( )
Sendto( ) ---------->	recvfrom( )
recvfrom( ) <----------	sendto( )
closesocket( )	closesocket( )

3.6 The Operation Modes

The connection established via Window Sockets has three distinct operation modes. They are as follows:

- Blocking Mode
- Non-Blocking Mode.
- Asynchronous Mode.

A blocking mode if fails will block all attempts to run any other WinSock function call within a task (or thread). Blocking operation if pending can be problematic if they usually initiate network operations. Though simpler to design such an operation gives rise to the problem of reentry (into the program). Clearly such a mode does not suffice our design needs if the application is to run on a Window platform, where calls to other window messages need to be handled simultaneously.

The problem with non-blocking mode is that is that an application needs to retry function over and over-polling to complete an operation, or detect its completion. This incurs significant system overhead if polled to often, or adversely affect application performance such as data throughput-if not polled often enough. Besides that it complicates the code.

Then how do solve the problem? The answer to this is quite simple, suppose the responsibility to callback is left for the remote application i.e. after initiating a network call the application goes about doing its chores and WinSock DLL sends the application a message when the event for which a callback was requested occurs. Fortunately Window Sockets provides us with such an operational mode, which is called as Asynchronous mode of operation. It is non-blocking since it returns from a call before the operation is

completed and relies on the "call you back" concept to complete its pending operation. The asynchronous mode of operation is non compatible with UNIX based Berkeley Sockets, since it was designed to use the inheritant message generating/operating property of Windows platform.

3.7 Byte Ordering

When transferring numeric data over a network it is important to consider the byte order used for representing numbers. PCs store the least significant byte at the lower memory address (little-endian byte order). However the big-endian byte order is the standard network byte order defined for the Internet. A number of functions defined in the socket interface require numeric parameters specified in network byte order. The following functions perform byte order conversions.

- ntohlO (network to host long)

  Converts a 32-bit value from network to host byte order.

- ntohsO (network to host short)

  Converts a 16-bit value from network to host byte order.

- htonlO (host to network long)

  Converts a 32-bit value from host to network byte order.

- htonsO (host to network short)

  Converts a 16-bit value from host to network byte order.

4.0 Audio and Video on the Internet

4.1 Introduction

Until recently, video on the web has been a down load-and-play technology. It worked as follows. the user clicked on a URL corresponding to the video file; the standard HTTP protocol was used to fetch that file to the local disk, based on the extension of the file (.avi or mov), Microsoft's Media player or Apple's QuickTime player was invoked to display the video. This basic solution, while having the advantage of implicitly, had a fundamental flaw. The user had to wait for the entire video file to download before he or she could see any of its content. Because video files are usually very large, the only video found on the web was short 30-second clips. Even these short clips were avoided by most users connected via 28.8 Kbps modems to the Internet

4.2 Streaming Audio and Video -Applications

Enabling compelling video on the web required the innovation of streaming video. With streaming video technology, the video is displayed to the end-user as it streams over the network in real-time. Other than a few seconds latency at the beginning, there is no wait associated with watching a video clip and it doesn't matter whether the clip is 30 seconds or I hour long. The user, in general, has the capability of skipping around within the video, just like one can within CD for music.

As real-time audio-video streaming over the Internet has become prominent, with products from Macromedia, Microsoft Net Show, Netscape, Progressive Networks, V DO, Vivo, VXtreme, Xing, and others, there are two major approaches emerging for streaming multimedia content. The first is the "server-Iess" approach, where a standard web server is used to supply data to the client. The second is the server-based approach,

where a specialized streaming server is used to deliver video to the client. In this paper we compare the advantages and disadvantages of the two approaches. Before we discuss relative merits, let's take a look at the way each approach works.

Applications:

- Streaming Video Creates A Media Rich Environment
- Migrating From A Text Only Published Site To A Compelling Media Rich Environment
- Increasing sales with better presentation
- Bringing training materials directly to the desktop
- Dramatically reducing the cost of attending meetings and seminars by bringing the meeting directly to the desktop
- Seamless integration with comp. any newsletters, announcements and HR materials

4.3 Video Compression for the Internet

One of the biggest challenges in enabling video over the Internet is developing video compression technology that can cope with the skinny pipes and heterogeneity of the Internet. The importance of compression is easy to see if we look at the storage and bandwidth requirements of uncompressed digital video.

4.3.1 Frame Quality

There are several dimensions to frame quality, including frame resolution, color depth, and distortion.

4.3.2 Frame

The frame resolution is measured as the width and height of the frame in pixels on the screen. For instance, for NTSC video:

- 640x480 video corresponds to full-screen resolution.
- 320x240 represents quarter-screen resolution.
- 160x120 represents quarter-quarter-screen resolution

The higher the resolution, the sharper the image appears to the eye. The lower the resolution, the fuzzier the image appears to the eye. By lowering the resolution of the frames, significantly higher-compression ratios can be achieved. For example, a 160x120 frame contains only 1/1.6th the pixels of a 640x480 frame. For this reason, most vendors use a resolution of 160x120 when delivering video over 28.8 Kbps channels.

4.3.3 Color Depth

The second dimension of frame quality, the color depth, refers to the number of bits used to represent each pixel in the image. 24-bits per pixel offers the highest quality (16 million colors) 16-bits per pixel (more common, difference in quality to 24-bit is small) 8-bits per pixel (more common substantial difference in quality to 16-bit)
From a compression perspective, reducing the number of bits used per pixel in the original image can give higher compression ratios. Unfortunately, quality that is lost in the original image due to such a conversion can never be recovered at the decoder.

4.3.4 Distortion

The third dimension of frame quality is the amount of distortion or error between the original frame and the compressed-then-decompressed frame. There are several established quantitative measures for distortion, including absolute error per pixel (the

average error between each pixel of the original frame and decoded frame), mean-squared-error (MSE) per pixel, or in signal processing theory terms "signal-to-noise ratio" (SNR).

Unfortunately, all of the above metrics offer good insight only within narrow regions of the evaluation space, that is, when the artifacts caused by the two compression algorithms being compared are of similar nature (e.g., blockiness observed in block-based algorithms, or DCT artifacts observed in the MPEG-family of algorithms) and of similar magnitude. The reason for the limited usefulness of distortion measures is that ultimately a human is judging the quality, and the human visual system is highly non-linear. For example, the human visual system may mask a particular type of error in one region of the image, and amplify the same error in another region of the image based on the surrounding context. For example, blockiness artifacts stand out much more in smooth regions of the image, like in a sky region, than they do in complex portions of the image, like in a flowerbed region.

The eventual task of evaluating video quality is further complicated by the fact that the temporal dimension (frame rate) and the spatial dimension (frame quality) interact with each other in complex ways. For example, the perceived resolution of the image and our sensitivity to errors is reduced when there is high motion in the scene. Given a bandwidth limit, most compression algorithms have the capability of trading off between offering higher frame rate or higher individual frame quality.

Some codec implementations fix the frame rate, and simply vary the frame quality to meet the bandwidth constraints. Others maintain a certain frame quality, but drop frames to meet the bandwidth constraints. Still others vary both parameters and attempt to

optimize visual quality for the end user. In summary to measure video quality we need to consider both the frame rate and frame quality at the same time. We emphasize that video quality is a subjective experience, quantitative measures offer only a rough guide to human perception.

4.4 Audio and Video Compression Algorithms

4.4.1 Bandwidth Scalability

This is the codec's ability to deliver a compressed video over a wide range of bandwidths. (From 20kbps to several Mbps)

4.4.2 Resolution, Frame-Rate, Frame Quality Scalability

When providing bandwidth scalability, the codec should allow the application to freely trade-off among video resolution, frame-rate, and individual frame quality. discussed in the previous section, for action-oriented videos, high frame-rate may be more important than frame resolution. Conversely, for educational videos, frame resolution may be more important than frame rate.

4.4.3 Fast Compression/Decompression

Most existing codecs do not allow for real-time compression in software requiring expensive compression hardware. In order for Internet video to become pervasive, the codec should be able to compress frames in real-time in software without taking up too much of the CPU time. More importantly, it should be able to decode rapidly the compressed stream as users receive It over the network. In fact, to allow multi-way video conferencing, the codec needs to be cheap enough computationally to allow multiple simultaneous decodes and one encode.

4.4.4 Ability to Cope With Network Losses

The Internet is a lossy transport medium and data packet can be lost in the delivery process. The compressed video stream must be structured so that individual packet losses do not severely degrade the quality of the video. For example, significant dependencies between encoded frames, as found in P (predicted) and B (bi-directionally-predicted) frames in MPEG, can cause a substantial reduction in visual quality in the event of packet loss. Video compression traditionally involves stripping out redundancy, whereas delivery over lossy networks often involves introducing redundancy. By optimizing these conflicting processes jointly rather than separately, high quality video can be delivered at lower bandwidths. An important implication is that live/on-demand video servers that provide codec-agnostic network delivery do not offer the best possible quality to end-users.

4.4.5 Encoding and Decoding Latency

In interactive applications such as video conferencing, it is important that the video and audio latency (i.e., the time between when the speaker says something and when the person on the other end observes it) be very small. Delays should not be introduced for the sake of high compression efficiency. For example, the B frames (bi-directional frames) in MPEG are encoded as a difference from both earlier and later frames. At 10 fps, waiting for the next few frames to occur can introduce several hundred milliseconds of latency into the video stream, which is a noticeable lag.

The codec should allow this type of latency to be disabled. To summarize, codecs for use on the Internet have different and more stringent requirements than those designed for playback from a CD-ROM or for TV broadcast. Codecs designed for the Internet require

greater scalability, lower computational complexity, and greater resiliency to network losses, and lower encode/decode latency for video conferencing.

4.5 Windows Media Encoder

Microsoft Windows Media Encoder is an easy-to-use, powerful production tool for converting both live and prerecorded audio and video to Windows Media Format. With Windows Media Encoder, you can deliver live content in real time to client computers or to a file for later use. Real-time sources of audio and video content include anything that you can plug into your audio or video card, including a CD player, microphone, VCR, video camera, or video player. Stored sources are audio or video files. Users can view encoded Windows Media-based content with Microsoft Windows Media Player or with any application built using the Windows Media Format.

4.5.1 Special Considerations for High Bandwidth

To take advantage of the high-bandwidth features and codecs in Windows Media Audio and Video 8, you will need to be able to capture audio and video at its highest quality. A computer that can capture a small image size at 15 frames per second may not be able to handle video with four times the image size and twice the frame rate.

When video is converted from analog to digital, each frame is broken up into hundreds of pixels. Each pixel is represented by one or more bytes that represent the color of that small area of the image. Each video conversion requires a certain amount of computer memory and computation time. The higher the quality of video, the greater the number of frames per second; and the larger the image size, the greater the number of pixels that must be converted in a given period of time. For example, when capturing video at 30 frames per second, the computer must not only be capable of handling many

pixel conversions, it must also perform many conversions very quickly in order to keep up with the continuous stream of video.

The computer must also be able to handle the audio conversion simultaneously. The smallest unit of audio is a called a sample. High-quality audio requires more samples per second, plus a computer that has the speed and memory to process the continuous stream. The optimal system described previously takes advantage of the new codecs and features in Windows Media Audio and Video 8.

4.5.2 Considering Compression

The digital audio and video streams of samples and pixels are measured as the bit rate of the content, such as 700 kilobits per second (Kbps). High-quality professional video has a bit rate that far exceeds the capacity of most computers and networks: 270 megabits per second (Mbps). However, there is a way to maintain high quality while lowering the bit rate and file size so that the content can be delivered over a network more easily. This method is called digital compression.

There are many digital compression techniques, or algorithms, which can be made available to programs for capturing, encoding, and playing back digital media. A compression algorithm that is created for this type of general use is called a codec. The Microsoft Windows Media Audio codec and the Microsoft Windows Media Video codec compress very high bit rate raw audio and video to a rate that can be streamed over a network or saved to a file and downloaded. The codecs offer a great deal of user flexibility because they are highly scalable. Scalability enables you to choose the amount of compression and bit rate for a given situation. For example, you can sacrifice some quality and compress to very low bit rates for streaming media at telephone modem

speeds, or maintain high quality and high bit rates to stream or download over a high-bandwidth network or to save to a CD.

Windows Media Audio and Video codecs enable you to put your audio and video on the Internet or an internal intranet and quickly reach many end users. When designing your production workstation, you must consider the codec and bit rates you will use. The analog-to-digital conversion requires a certain amount of speed and memory, and compression requires additional processing power from your computer. However, if you follow the recommendations for the optimal system, your workstation will be able to handle the additional workload

4.5.3 High-Bandwidth Features of Windows Media Technologies

Several features introduced in Windows Media Technologies 7 are aimed at improving the quality of high-bandwidth content. With the introduction of Windows Media Audio and Video, the quality of the codecs has been greatly improved. The Windows Media Video codec offers near-VHS quality at 250 kilobits per second (Kbps) and near-DVD quality at 500 Kbps when creating Windows Media files for download.
As you can see, the need is greater than ever to upgrade the quality and speed of a production system to take advantage of all that Windows Media Audio and Video 8 has to offer. The following list briefly describes the most important high-bandwidth features:

Deinterlacing: When encoding a video file captured at the full frame size of 640 x 480 pixels, the two interlaced fields contained in a single frame of NTSC video must be converted to one complete frame that can be displayed on a computer monitor. Computer monitors use a different method of displaying video called progressive scanning, which does not use interlaced fields. The deinterlacing feature converts interlaced video frames

into progressively scanned frames, creating a cleaner, sharper image with fewer motion artifacts at both a full frame size and at 320 x 240 pixels.

Inverse telecine: To make the conversion from film, which plays back at 24 frames per second, and video, which plays at 29.97 frames per second, a telecine, such as a film scanner, adds redundant fields to the video. When encoding captured video of a film, you can use the inverse telecine filter to remove redundant fields that were added by the telecine. The final encoded video appears more like the original film; plus, with fewer frames, the file size and bit rate decrease.

60 frames per second: high-quality video can be created that has very smooth and crisp motion by using the deinterlacing filter to encode from 640 x 480 pixels to 320 x 240 pixels, and then converting the 60 fields per second into 60 frames per second.

Variable bit rate: Any on-screen movement results in an increase in the bit rate of a video, because new pixels must be generated from frame to frame. One of the benefits of using Windows Media Technologies for streaming over a network is that it maintains a constant bit rate (CBR) regardless of how much change occurs in the video. To maintain a constant bit rate, however, Windows Media Encoding Utility must compromise playback quality by dropping frames or dynamically reducing the number of new pixels. Playback is not always as smooth as one would like, but the end user experiences as smooth a presentation as possible for a given bandwidth. With the introduction of Windows Media Audio and Video, you have the option of encoding CBR video for streaming over a network or variable-bit-rate (VBR) video for situations where bit rate is not an issue. VBR video cannot be streamed. You should use VBR video when video is to be downloaded and played back locally or over a fast network. With VBR, the

integrity of high motion or rapid changes in the video is maintained by simply allowing the bit rate to vary as needed. You set the desired quality level, and the bit rate changes to maintain that quality.

Two-pass encoding: When using two-pass encoding, the Windows Media Encoding Utility reads through the file first to analyze the complexity of the content. It then goes through the content a second time and encodes a file based on that analysis. Two-pass encoding can be used with either the CBR or VBR methods and produces a much cleaner and smoother video than the one-pass CBR method used in previous versions of Windows Media Encoder. Windows Media Encoder currently supports only one-pass CBR, making it suitable for encoding broadcast or on-demand streams.

By using these features with the improved Windows Media Audio and Video codecs, you can create high-quality pictures and sound at a fraction of the file size and bit rate of conventional digital media. The typical computer monitor has a far higher resolution and faster frame rate than an NTSC television. This enables you to produce video on a computer monitor that is higher quality than standard television—and with Windows Media Audio and Video codecs, you can produce video and audio that can be downloaded or streamed. However, to do all this you need a system that can capture and maintain high quality.

4.5.4 Intelligent Streaming and Multiple-bit-rate Video

Intelligent streaming is a set of features in Microsoft Windows Media Technologies that automatically detects network conditions and adjusts the properties of a video stream to maximize quality. Because Windows Media Technologies is a client/server system, the server and the client communicate with each other to establish

actual network throughput and automatically make a series of adjustments to maximize the quality of the stream. With intelligent streaming, users receive a continuous flow of content tailored to their connection speeds.

To take full advantage of intelligent streaming, you must encode your content using multiple bit rates. To encode content at multiple bit rates, you create a single Windows Media Format stream or file containing multiple streams (audio, video, and script) that are encoded at different bit rates. When a multiple-bit-rate Windows Media file or live stream is received by a player, only one of the video streams is played: the one that is the most appropriate for current bandwidth conditions. The process of selecting the appropriate stream is handled by the Windows Media server and Windows Media Player and is completely invisible to the user.

Windows Media Encoder provides predefined multiple-bit-rate profiles, making encoding content for multiple bit rates an easy process. You can also create your own multiple-bit-rate profiles using Profile Manager.

4.5.5 Looping

Looping is a feature of Windows Media Encoder where file-based content is played repeatedly until encoding is complete and ensures that you have no interruption in the stream as a result of reaching the end of a file before completing a broadcast or capture session.

Looping is available for broadcast and capture scenarios. In a broadcast scenario, and is applied automatically when one of your source groups is a file. In a capturing scenario, looping is applied automatically when you have a configuration that includes multiple source groups. In both cases, looping is applied to only the active source group.

Looping is not available for file-conversion scenarios or when you are sourcing content from an .mpg file and is also not invoked when you use the time compression feature to accelerate or decelerate the playback of your content.

4.5.6 Codecs

Audio and video content can consume a lot of bandwidth on a network when it is streaming. By compressing the content, it can be broadcast over common Internet bandwidths. You can compress content by applying compression algorithms to the data, taking into account the desired output quality and available bandwidth. Before the stream is played on a player, it is decompressed using decompression algorithms. These compression and decompression algorithms are called codecs.

Codecs are designed to compress a stream to a certain bit rate. The target bit rate determines the amount of compression applied. Codecs that do not compress source content as much produce content that usually sounds and looks richer and more dynamic, but requires more bandwidth to stream.

Windows Media Audio and Windows Media Video are software codecs used to decrease the bit rate of digital media files so they can be delivered efficiently over a network. Windows Media Encoder uses these codecs to compress the data for streaming, while Microsoft Windows Media Player decompresses the data for playback.

The Windows Media Audio and Windows Media Video codecs offer excellent compression quality and efficiency. The Windows Media Audio codec delivers a .wma file of the same quality as an .mp3 file, but at nearly one-third the size. The codec is fully compatible with Windows Media Player and all portable devices supporting Windows Media Audio. While the quality of encoded video depends on the content being encoded,

Windows Media Video can deliver near-VHS-quality at bit rates ranging from 250 kilobits per second (Kbps) to 450 Kbps, and near-DVD-quality at 500 Kbps to several megabits per second (Mbps). Windows Media Video 8 codec is appropriate for both streaming and downloading digital media files. It is also compatible with Windows Media Player.

5.0 Implementation

5.1 Introduction

In this section we will discuss the architecture of the Remote Experimental Station and various tools and methods used to implement system. The theoretical perspective of the thesis study is to provide a distance-learning laboratory for the students in undergrad curriculum of Electrical and computer engineering major where a student can learn experiments from a remote location under the supervision of the instructor. Interactive Remote Experimental station provides a network-based interactive environment between students and instructor, which is used to bridge instructional gap

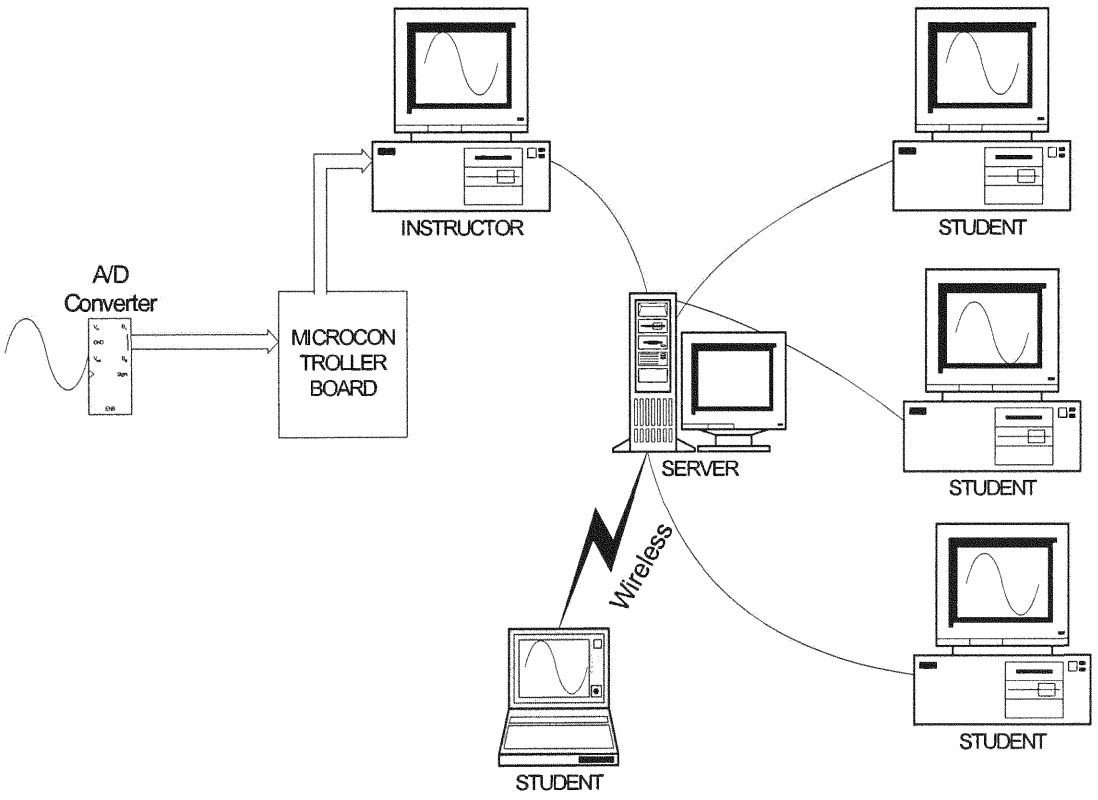

Figure 5.1. Remote Experimental Station Architecture

5.2 Operation of A/D Converter

The MAX1284 is a12-bit analog-to-digital converter (ADC), which combines a high-bandwidth track/hold (T/H), a serial interface with high conversion speed, an internal +2.5V reference, and low power consumption. The MAX1284 operates from a single +4.5V to +5.5V supply. The MAX1284 is a high-speed, 12-bit data-acquisition system. Resistor R1 (1k.) and capacitor C1 (0.01μF) form a single-pole, low-pass anti-aliasing filter with a nominal 10μs time constant and a corner frequency of approximately 16kHz. C3 and C4 bypass the analog-to-digital converter's (ADC's) voltage reference. When plugged into the micro controller (68HC16) module, the VDD circuit is powered by +5V. Fig shows the various connections and pin out, the specifications and pin outs are referred in appendix.

The MAX1284 use an input T/H and successive-approximation register (SAR) circuitry to convert an analog input signal to a digital 12-bit output. Figure 5.3 shows the MAX1284 in its simplest configuration. The internal reference is trimmed to +2.5V. The serial interface requires only three digital lines (SCLK, CS, and DOUT) and provides an easy interface to microprocessors (μPs).

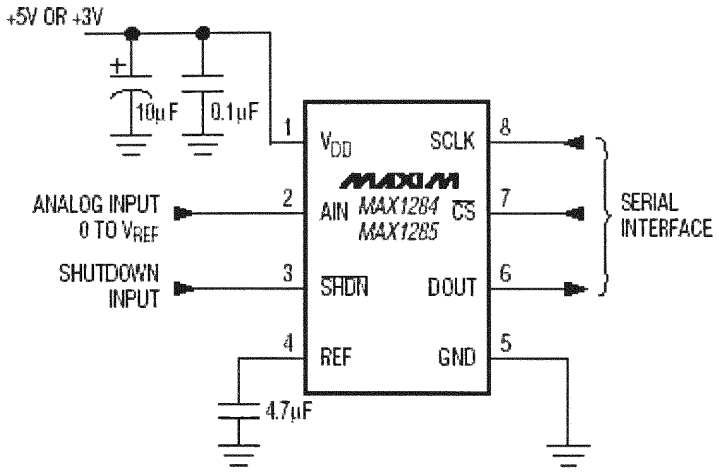

Figure 5.2. Typical Operating Circuit [15]

The MAX1284/MAX1285 have two modes: normal and shutdown. Pulling SHDN low shuts the device down and reduces supply current to below 2µA (typical), while pulling SHDN high puts the device into operational mode. Pulling CS low initiates a conversion that is driven by SCLK. The conversion result is available at DOUT in unipolar serial format. The serial data stream consists of three zeros, followed by the data bits (MSB first). All transitions on DOUT occur 20ns after the rising edge of SCLK. Fig 5.3 illustrates the sampling architecture of ADC's comparator. The full-scale input voltage is set by the internal reference (V_{REF} = +2.5V)

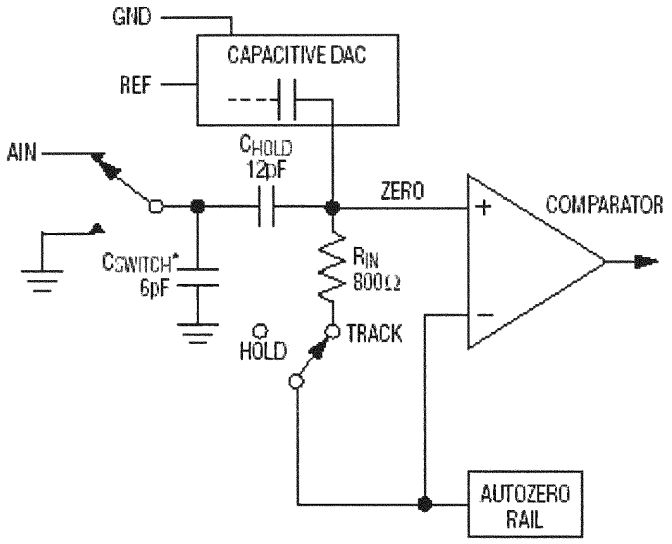

Figure 5.3. Equivalent Input Circuit [7]

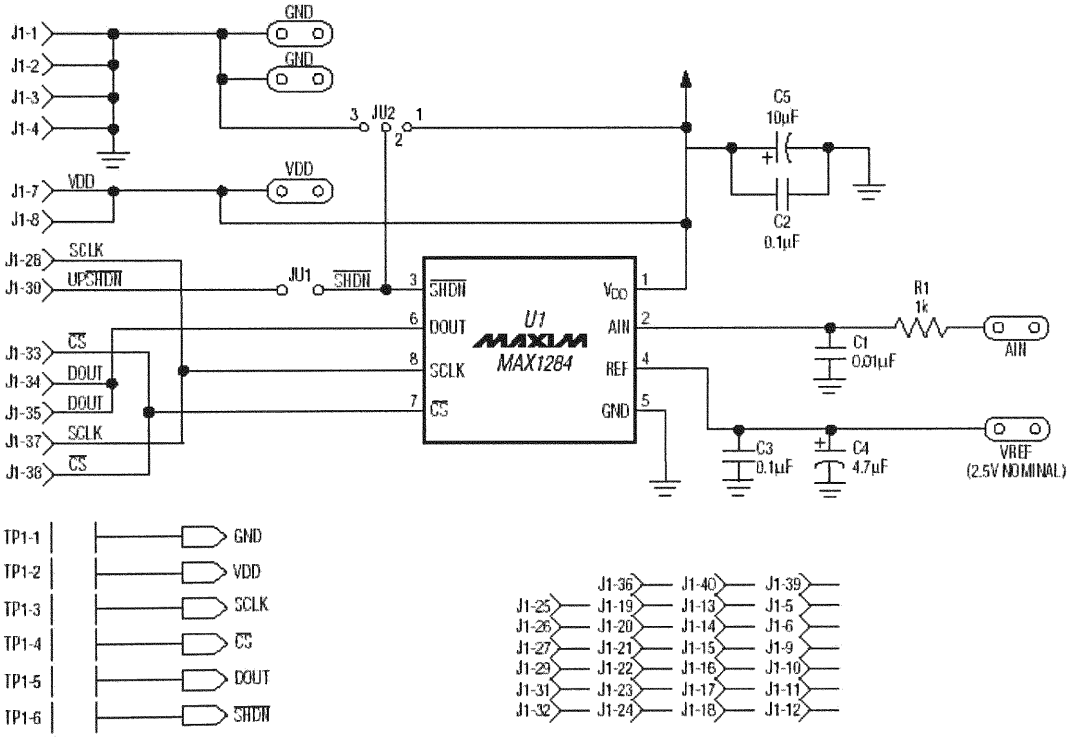

Figure 5.4. A/D Converter Schematic Diagram [17]

5.2.1 Track/Hold

In track mode, the analog signal is acquired and stored in the internal hold capacitor. In hold mode, the T/H switch opens and maintains a constant input to the ADC's SAR section. During acquisition, the analog input (AIN) charges capacitor CHOLD. Bringing CS low, ends the acquisition interval. At this instant, the T/H switches the input side of CHOLD to GND. The retained charge on CHOLD represents a sample of the input, unbalancing node ZERO at the comparator's input. In hold mode, the capacitive digital-to-analog converter (DAC) adjusts during the remainder of the conversion cycle to restore node ZERO to 0 within the limits of 12-bit resolution. This action is equivalent to transferring a charge from CHOLD to the binary-weighted capacitive DAC, which in turn forms a digital representation of the analog input signal. At the conversion's end, the input side of CHOLD switches back to AIN, and CHOLD charges to the input signal again. The time required for the T/H to acquire an input signal is a function of how quickly its input capacitance is charged. If the input signal's source impedance is high, the acquisition time lengthens and more time must be allowed between conversions. The acquisition time (t_{ACQ}) is the maximum time the device takes to acquire the signal, and is also the minimum time needed for the signal to be acquired. Acquisition time is calculated by:

$t_{ACQ} = 9(RS + RIN) \times 12pF$

Where RIN = 800Ω ,Fig 4.4 RS = the input signal's source impedance, and t_{ACQ} is never less than 625ns. Source impedances below 2kΩ do not significantly affect the ADCs AC performance. Higher source impedances can be used if a 0.01μF capacitor is connected to

the analog input. Note that the input capacitor forms an RC filter with the input source impedance, limiting the ADCs input signal bandwidth.

5.2.2 Input Bandwidth

The ADCs' input tracking circuitry has a 6MHz small-signal bandwidth, so it is possible to digitize high-speed transient events and measure periodic signals with bandwidths exceeding the ADC's sampling rate, by using under-sampling techniques. To avoid aliasing of unwanted high-frequency signals into the frequency band of interest, anti-alias filtering is used.

5.2.3 Internal Reference

The MAX1284 have an on-chip voltage reference trimmed to 2.5V. The internal reference output is connected to REF and also drives the internal capacitive DAC. The output can be used as a reference voltage source for other components and can source up to 800μA. Bypass REF with a 4.7μF capacitor. Larger capacitors increase wake-up time when exiting shutdown. The internal reference is disabled in shutdown

5.2.4 Starting a Conversion

When power is first applied, and if SHDN is not pulled low, it takes the fully discharged 4.7μF reference bypass capacitor up to 2ms to provide adequate charge for specified accuracy. No conversions should be performed during this time. To start a conversion, pull CS low. At CS's falling edge, the T/H enters its hold mode and a conversion is initiated. Data can then be shifted out serially with the external clock which in this case is provided by the micro-controller.

5.2.5 Timing and Control

Conversion-start and data-read operations are controlled by the CS and SCLK digital inputs. The timing diagrams of Figures 8 and 9 outline serial-interface operation. A CS falling edge initiates a conversion sequence: the T/H stage holds the input voltage, the ADC begins to convert, and DOUT changes from high impedance to logic low. SCLK is used to drive the conversion process, and it shifts data out, as each bit of conversion is determined. SCLK begins shifting out the data after the rising edge of the third SCLK pulse. DOUT transitions 20ns after each SCLK rising edge. The third rising clock edge produces the MSB of the conversion at DOUT, followed by the remaining bits. Since there are twelve data bits and three leading zeros, at least fifteen rising clock edges are needed to shift out these bits. Extra clock pulses occurring after the conversion result has been clocked out, and prior to a rising edge of CS, produce trailing zeros at DOUT and have no effect on converter operation. Pull CS high after reading the conversion's LSB. For maximum throughout, CS can be pulled low again to initiate the next conversion immediately after the specified minimum time (t_{CS})

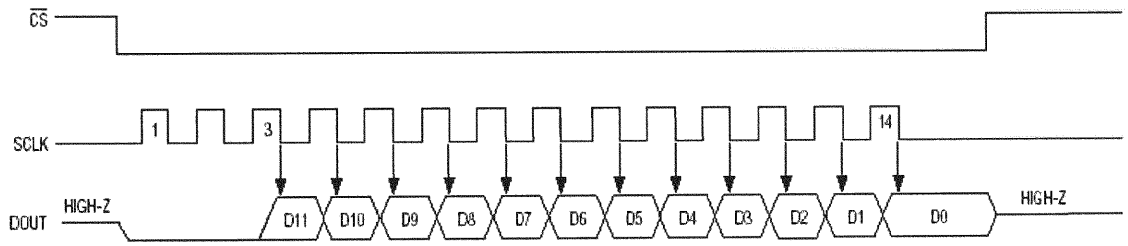

Figure 5.5. QSPI Interface Timing Sequence [17]

5.2.6 QSPI Interface

QSPI allows the minimum number of clock cycles necessary to clock in the data. The MAX1284 require 15 clock cycles from the µP to clock out the 12 bits of data. Figure 4.5shows a transfer using CPOL = 0 and CPHA = 1. The conversion result contains two zeros followed by the 12 bits of data in MSB-first formatted.

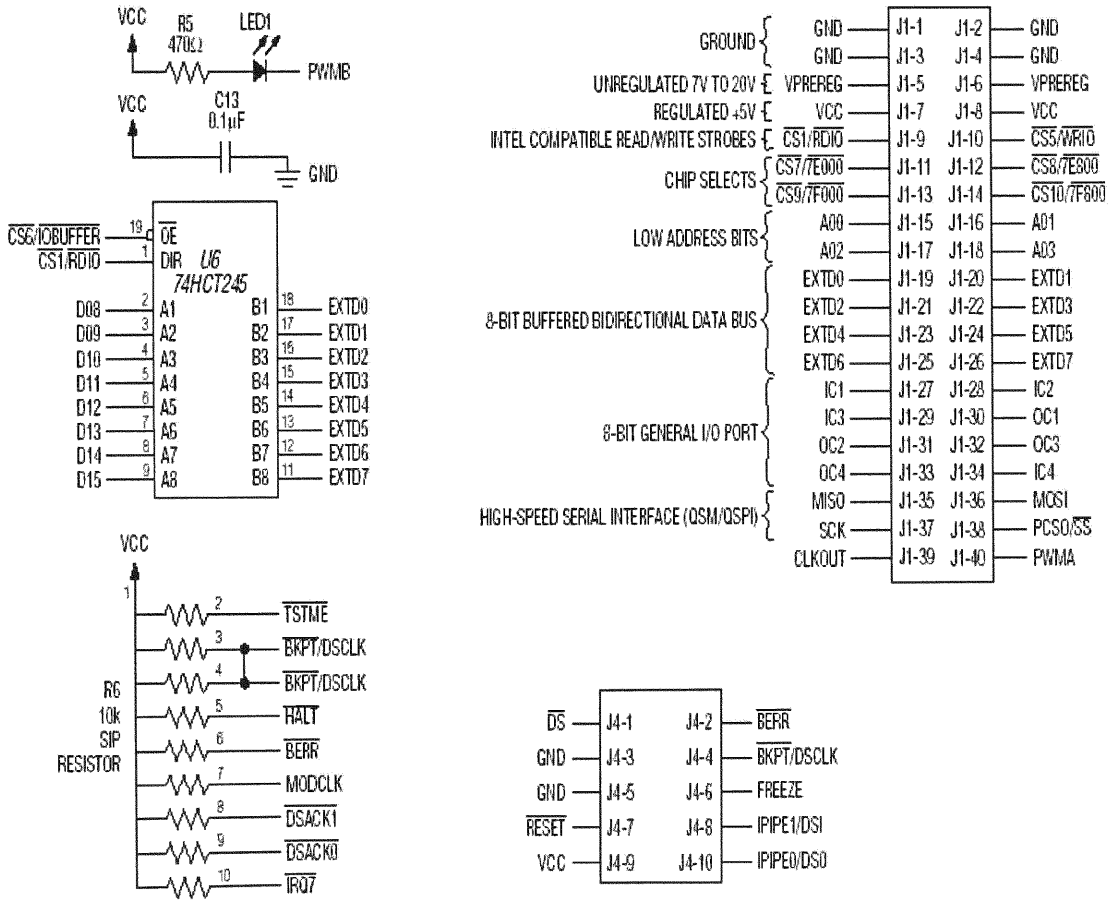

Figure 5.6. Schematic Diagram of Microcontroller (68HC16) Module (1) [15]

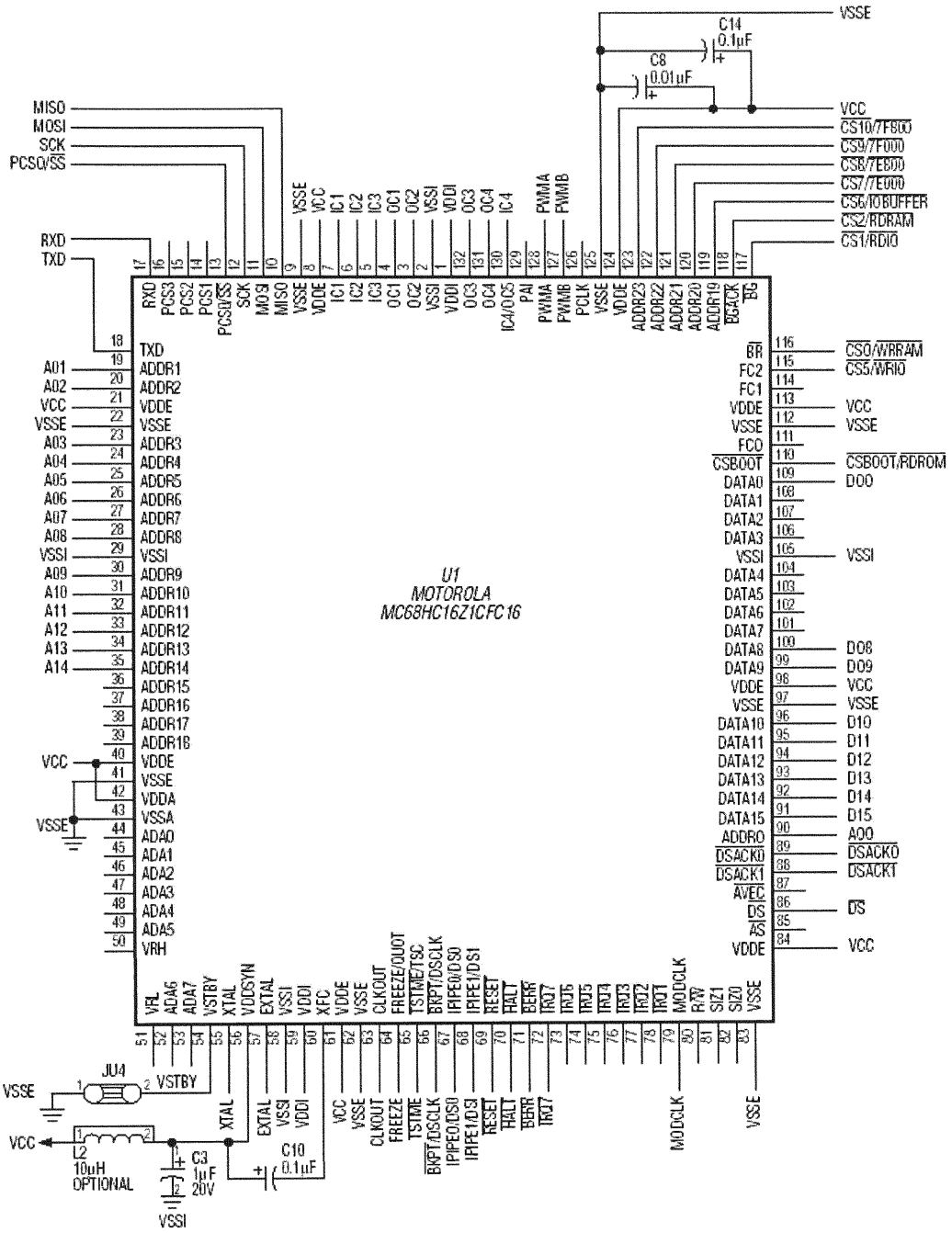

Figure 5.7. Schematic Diagram of Microcontroller (68HC16) Module (2) [15]

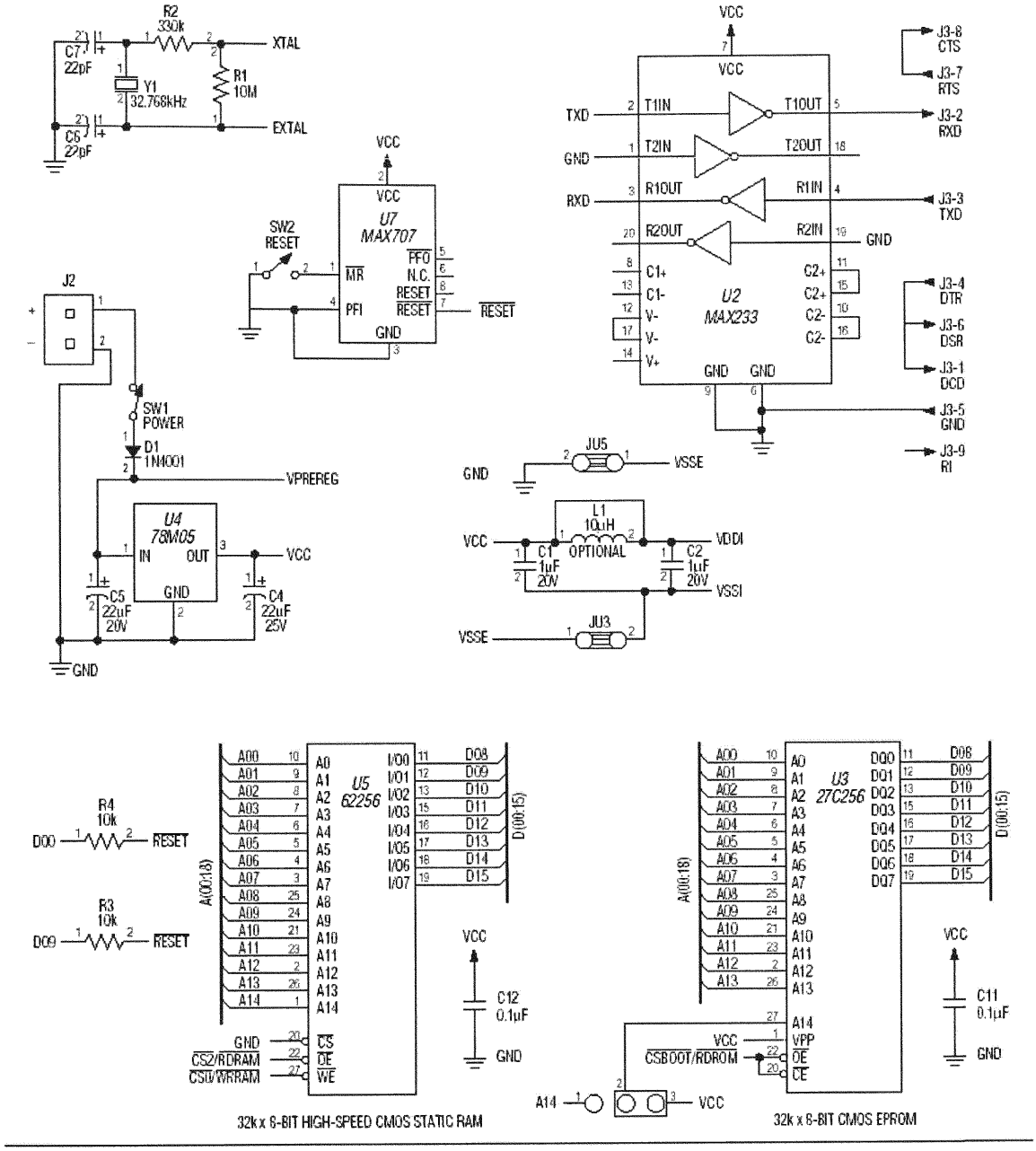

Figure 5.8. Schematic Diagram of Microcontroller (68HC16) Module (3) [15]

5.3 Microcontroller Module (68hc16)

Fig 5.6 to 5.8 shows the schematics of the microcontroller module. The 68HC16 module is an assembled printed circuit board from Motorola. The module uses an inexpensive 8-bit implementation of Motorola's MC68HC16Z1 microcontroller (μC) to collect data samples at high speed using the Queued Serial Peripheral Interface (QSPI). The module draws its power from user supplied power source connected to terminal block J2.A three terminal 5V regulator allows input voltages between 8V and absolute maximum of 20V. The module requires 200mA of input current.

The MC68HC16Z1 (Fig 5.7) is a member of Motorola's modular microcontroller family, a series of 16-bit and 32-bit devices constructed from standard on-chip peripheral modules that communicate by means of a standard inter module bus. The MC68HC16Z1 is a sophisticated single chip control system that incorporates a 16-bit CPU module, a system integration module (SIM), a general-purpose timer (GPT), a queued serial module (QSM), an 8-channel analog to digital converter (ADC), and 1-Kbyte stand by RAM. The MCU thus provides a designer with many options, ranging from reset configuration to interrupt generation, that must be considered during the design phase [15].

MAX 707 on the module monitors the 5v logic supply, generates the power on reset , and produces whenever reset button is pressed. The module uses a phase-locked loop (PLL) to set its bus speed. Crystal Y1 is a 32.768kHz frequency reference. The internal oscillator runs 256 times faster than the external crystal. When the 68HC16 is reset, it waits for the PLL to lock before it executes any software. After the PLL locks onto the reference frequency, the software doubles the clock speed by writing to the clock synthesizer control register, selecting a bus speed of 16.78MHz.

The 74HCT245 octal buffer lets the 68HC16 module access an 8-bit port on the 40-pin interface connector. This memory-mapped port consists of separate read and write strobes, four chip selects, four address LSBs, and eight data bits.

5.3.1 Serial Communications

J3 is an RS-232 serial port, designed to be compatible with the IBM PC 9-pin serial port. Use a straight through DB9 male-to-female cable to connect J3 to this port. If the only available serial port has a 25-pin connector, you may use a standard 25-pin to 9-pin adapter. The MAX233 is an RS-232 interface voltage level shifter with two transmitters and two receivers. It includes a built-in charge pump with internal capacitors that generates the output voltages necessary to drive RS-232 lines.

The connections and their descriptions are referred in Appendix B.

5.4 Software

The software used to develop application is C++ and the compiler is Visual C++ compiler. Pre defined Microsoft Foundation Class (MFC) were also used to reduce the time for development.

5.4.1 Data Display/Processing Application

The data from the microcontroller module is acquired using RS-232 cable. The software main window controls the serial clock speed and sample rate. It displays the voltage and output code as well as some statistics of the input signal. A separate graph shows the data changing in the real time. The update rate is limited to about 10 samples per second due to com port bandwidth limitations.

5.4.2 Client/Server Application

This application was developed using C++ and windows sockets Application Programming Interface (API). Various classes were written in C++, which implements the data transfer and connectivity between clients. The server application consists of list and information of the clients that are connected and plays the role of transferring the data from one client to other client. The instructor and student uses the client part and the server will be running on a different machine, this type of architecture was implemented keeping in mind that if instructor application is implemented in the server than every client connected to the server has to reconnect again if there is a connection problem in the instructor machine. The client part also has the functionality of displaying the digital code acquired from the microcontroller module in form of a graph so the data samples can be transferred to the other clients and displayed.

5.4.3 Audio/Video Streaming Application

Microsoft Windows Media Encoder is a audio /video streaming application which uses the latest audio and video compression technologies, the Microsoft Windows Media Audio codec and Microsoft Windows Media Video codec for real-time capture and streaming applications. These codecs, or compressor/decompressors, deliver incomparable audio and video quality at lower bit rates.

Windows Media Encoder includes multiple input sources, on-the-fly source switching for live or on-demand events, and creating and modifying encoding profiles. In addition, the encoder includes four new profiles that were designed specifically for streaming to take advantage of the quality and efficiency of the Windows Media Audio 8

and Windows Media Video 8 codecs. The audio/video data streamed by the windows media encoder can be viewed in windows media player.

6.0 Results, Conclusion and Future work

6.1 Results

The following figures 6.1 to 6.3 show the waveforms of actual signal, which is regenerated from samples acquired using A/D converter and microcontoller module. The fig shows the graph part where the analog signal is displayed and to the right we have the volts/div and time/div which means that each division is to be multiplied by the number shown. Voltage/amplitude is represented on y-axis and time is represented on x-axis. The "UpDate" button will draw the graph with the updated samples.

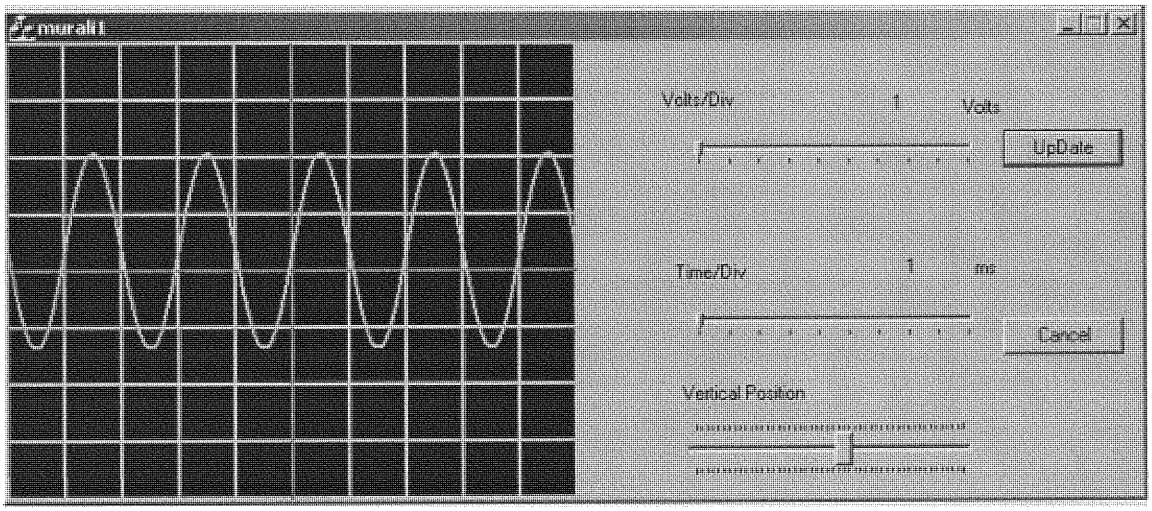

Figure 6.1. Sine Waveform

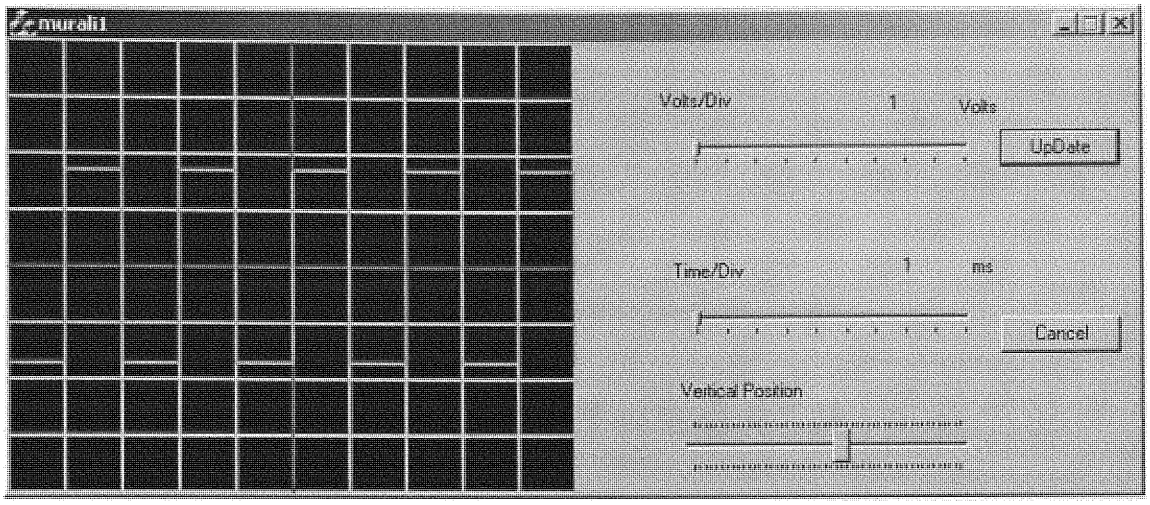

Figure 6.2. Square Waveform

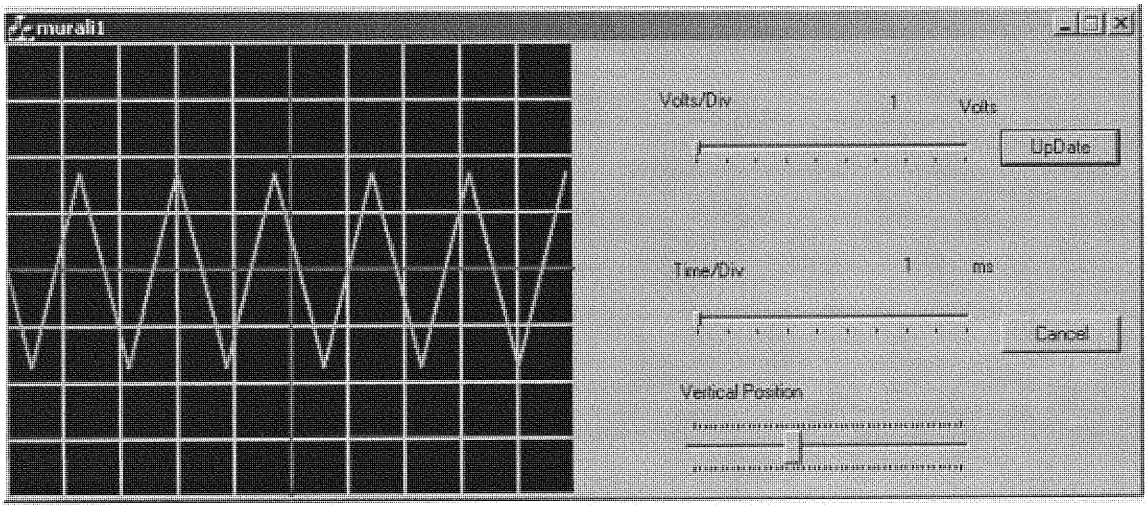

Figure 6.3. Triangular Waveform

The following figure (Fig 6.4) is screen shot of windows media encoder which encodes and broadcasts the audio and video captured by the web camera. Traversing through "session" item in the menu gets you the new session wizard where you can specify the streaming speed, title and other data

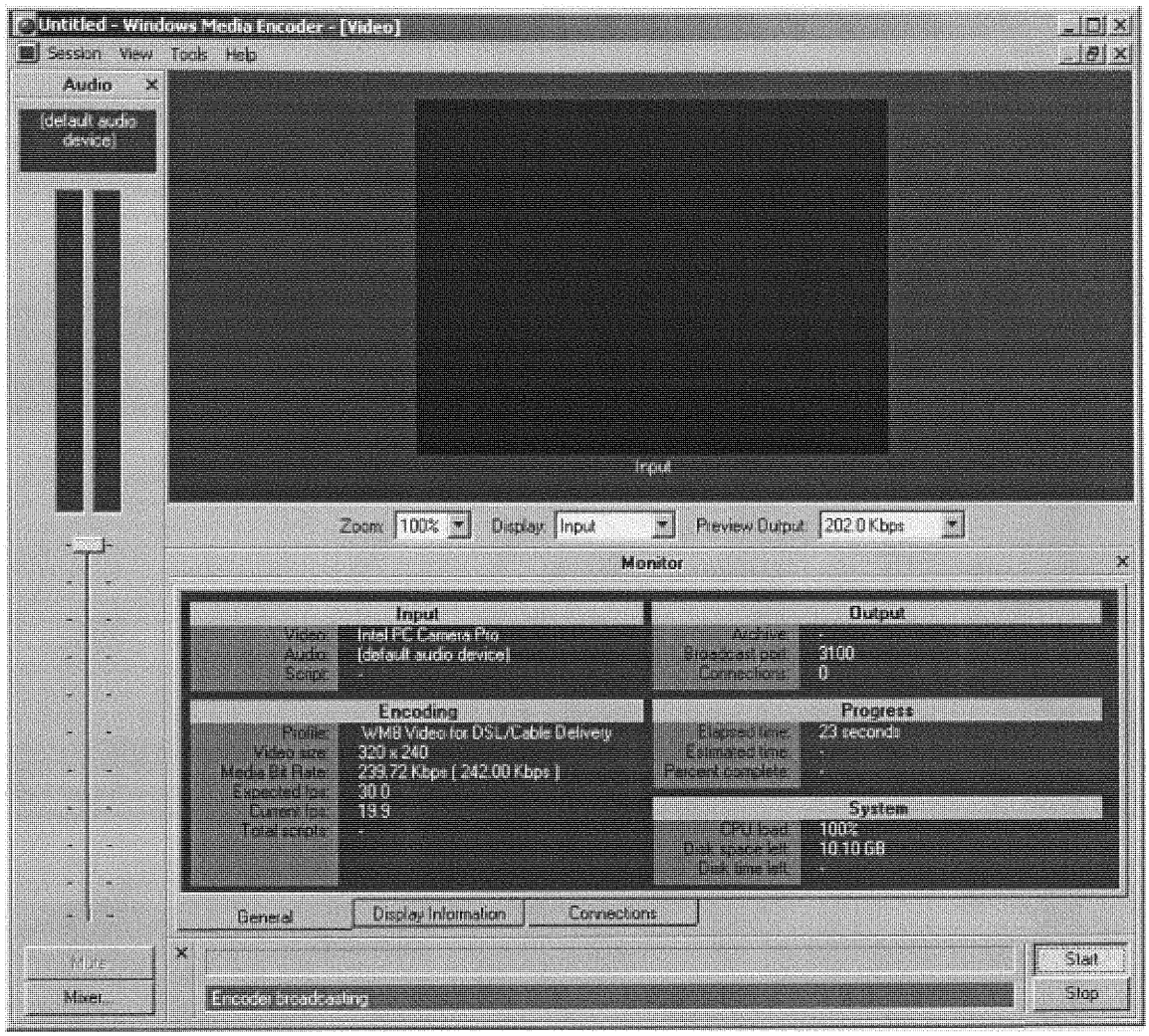

Figure 6.4. Windows Media Encoder

Fig 6.5 is screen shot of the client, which establishes the connection between instructor and student. Where the right side list box shows the list of the users connected, typing a text in the edit box and left clicking the mouse button on "send" will pass the message to all users. Other functionalities like different fonts and font colors are implemented. The "Graph" button on clicking with mouse left button displays the graph shown above in Fig 6.1. The 'C' menu item is to connect to the server and one should be aware of the IP address of the server to connect.

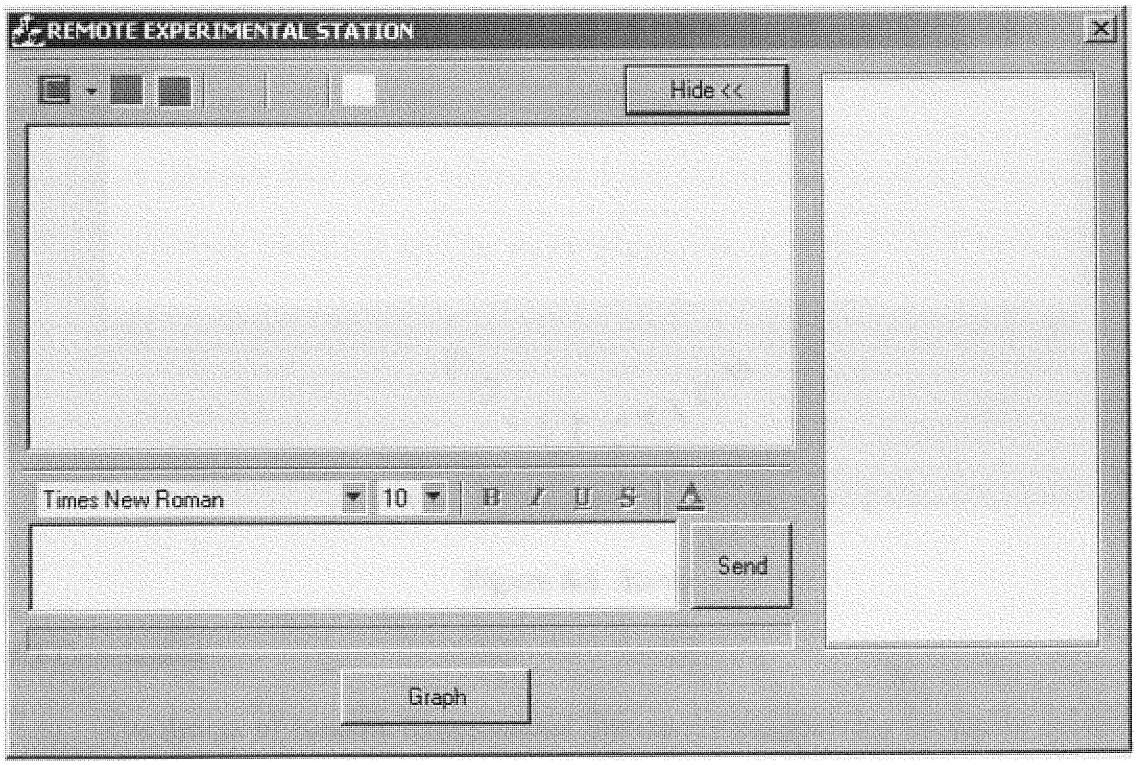

Figure 6.5. Client

Fig 6.6 is the screenshot of a server, which connects all the clients and enables the data transfer between them. The list box shows the users connected to server. The

"start/stop" button toggles between the on and off stages of the listening socket (i.e. server will not be able to accept connection if it is stopped).

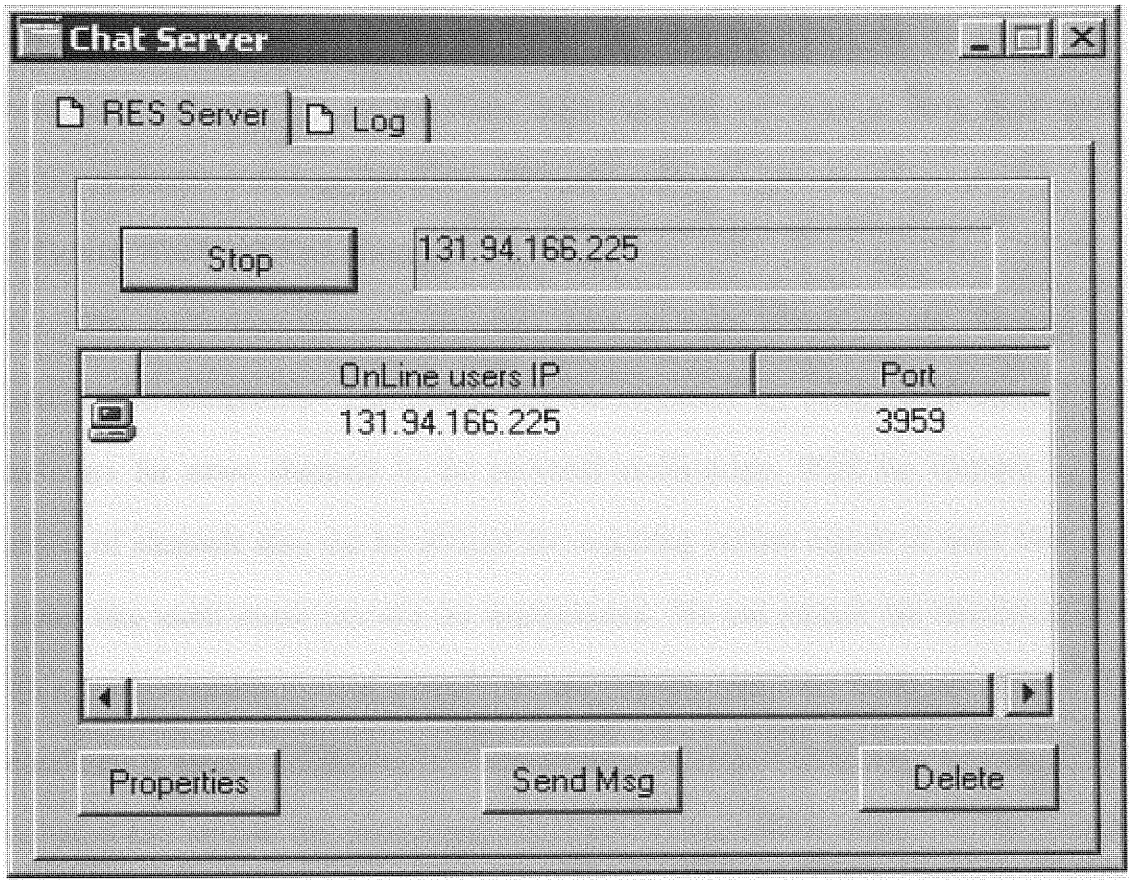

Figure 6.6. Server

6.2 Conclusion

The design of Remote Experimental Station For Engineering Education achieved its goal using the analog to digital converter for sampling analog signal and a microcontroller module to collect the samples and transmitting them to computer over serial cable (RS 232) where the samples were analyzed and analog signal is regenerated and displayed in a pc based oscilloscope. Client/Server application plays role of

transmitting the samples to remote computers and presenting the output signal to the students. Windows Media Encoder incorporated audio/video conference for the system. The system designed is a cost effective solution as a/d converters and the microcontroller units can be purchased for as low as $10.00 per each unit. To view the audio/video streaming broadcasted by the windows media encoder we need to have high-speed Internet connections such as DSL/Cable for home users.

6.3 Future Work and Enhancements

The application design is presently done on a single channel a/d converter for avoiding the complexity. The system can be enhanced by using multi channel a/d converters for more channels in the pc-based oscilloscope. With the increase in the number of channels they can be utilized for acquiring various signals at same time and utilities like multi meter can also be implemented. We can replace the microcontroller with re-programmable devices like CPLDs and FPGAs, which are faster and reliable. As technology is rapidly increasing a/d converters with greater sampling rate and greater resolution can replace the present a/d converters. Enhancement of the present Internet protocols for smooth audio/video streaming over the networks without loss of data.

REFERENCES

[1] Vincent G. Gomes, Bruce Choy, Geoff W. Barton, Jose A. Romagnoli," Web-Based Courseware in Teaching Laboratory-Based Courses", Department of Chemical Engineering, The University of Sydney, Sydney, NSW 2006, Australia © 2000 UICEE Global J. of Engg. Educ., Vol.4, No.1.

[2] Dongil Shin a,,, En Sup Yoon a, Sang Jin Park b, Euy Soo Lee b," Web-based interactive virtual laboratory system for unit operations and process systems engineering education" Computers & Chemical Engineering.Volume 24, Issues 2-7, 15 July 2000, Pages 1381-1385.

[3] L, Mercer, P, Prusinkiewicz, J, Hanan, "The concept and design of a Virtual Laboratory". In Graphics Interface '90 Conference proceedings, pages 149-155. Canadian Information Processing Society, 1990.

[4] The Virtual Laboratory, "An Application Environment for Computational Science and Engineering", Http://www.internet2.edu/html/virtual_laboratory.html.

[5] Hesselink, L, Rizal, D, Bjornson, E, " CyberLab, A New Paradigm in Distance Learning", Stanford University, 2000.

[6] Hoon, P, S, "Conducting Experiments Over the Internet using Lab VIEW and Component Works", Department of Electrical Engineering, Singapore Polytechnic, 2000.

[7] Analog-Digital Conversion Handbook: by The Engineering Staff of Analog Devices, Prentice hall, 1986.

[8] Richard, M, Jones, "Introduction to MFC Programming with Visual C++" Prentice hall, 2000.

[9] Sanjit, K, Mitra, "Digital Signal Processing: A Computer based approach " - 2nd edition, McGraw-Hill, 2001.

[10] Bob, Quinn, Dave, Shute, "Windows Socket Network Programming ", Adison-Wesley, 1997.

[11] David, J, Kruglinsky, George, Shepherd, Scot, Wingo, "Programming Microsoft Visual C++", Microsoft Press, 1998.

[12] Sharon, Darley, Charles, Melear, "An Introduction to the MC68HC16Z1 ", Motorola Semiconductor Device Tutorial, Motorola INC, 1996.

[13] Lang, George, Fox, " Quantization noise in A/D converters ", S V SOUND VIB. Vol. 34, no. 11, pp. 5-6. Nov 2000.

[14] Chiorboli, Giovanni; Fontanili, Massimo, "Cross-correlation noise measurements in A/D converters", IEEE TRANS. INSTRUM. MEAS. Vol. 48, no. 6, pp. 1282-1286. Dec 1999.

[15] Maxim Integrated Products, "Understanding SAR ADCs", Mar 2001.

[16] William, Stalling, "Data And Computer Communications", Prentice Hall, 1996.

[17] Maxim Integrated Products, "MAX1284 Evaluation System", June 2000.

APPENDIX A

ADC and DAC Glossary

Terms

acquisition time (for track/hold): The amount of time it takes for the analog-to-digital converter (ADC) input to "acquire" the input signal. The RC time constant formed by the ADC's "track/hold" circuitry and the accuracy to which the ADC must digitize the input signal determine the length of acquisition.

aliasing: In sampling theory, when an input signal frequency component exceeds the Nyquist limit, the signal is "aliased" or "folded back" or replicated at other frequencies in the frequency spectrum above and below Nyquist. Normally, aliasing is due to unwanted signals beyond the Nyquist limit. To prevent aliasing, all undesired signals must be filtered adequately so that they are not digitized by the analog-to-digital converter (ADC). Aliasing can be used advantageously when undersampling.

aperture delay: In an analog-to-digital converter (ADC), the delay in time from when the user requests the analog input to be sampled (that is, when the user puts the ADC's track/hold in "hold" mode) and the actual time when this occurs.

aperture jitter: The amount of variance in the aperture delay. This value is typically much smaller than that of the aperture delay.

binary coding (unipolar): A digital coding scheme in which zero is represented by all zeros (00000000) and full scale by all ones (11111111).

bipolar: A signal that swings both above and below analog ground, thus having positive and negative values.

common-mode rejection (CMR): The ability of a device to reject a signal that is "common" to or applied to both input terminals. The common-mode signal can be either an AC or a DC signal or a combination of the two. CMR is often expressed in decibels.

common-mode rejection ratio(CMRR): It is the ratio of the differential signal gain to the common-mode signal gain.

crosstalk: For an analog-to-digital converter (ADC) with more than one input, crosstalk is the amount of signal from one analog input that appears on the measured analog input. This value is typically specified in decibels. For a digital-to-analog converter (DAC),

crosstalk is the amount of noise that appears on the DAC output(s) when another DAC is being updated.

decibel (dB): A unit of relative amplitude defined on a logarithmic scale. For voltage values, dB is given by 20 log Va/Vb. For power, it is 10 log Pa/Pb. DBc is dB referenced to a carrier signal, and dBm is dB referenced to 1mW. For dBm, the load resistance must be known for the specification to determine the voltage or current equivalence (that is, 1mW into 50 ohms).

differential nonlinearity (DNL) error: Assuming an ideal analog-to-digital converter (ADC) with finite digital codes exactly 1LSB apart (DNL error = 0) or an ideal digital-to-analog converter (DAC) with analog output values exactly one code apart (DNL error = 0), the DNL error is defined as the difference between the ideal and the measured code transitions for successive codes for an ADC or the difference between the ideal and the measured output value between successive DAC codes. As an example, when using an ADC, as the analog input voltage increases, if the next code transition occurs at a voltage 1.5LSB away from the previous transition, the DNL error is 0.5LSB; if it occurs at 0.5LSB away, the DNL error is -0.5LSB.

dynamic range: The range of signal amplitudes (or signal strengths) a converter can resolve, typically expressed in decibels. A converter with a dynamic range of 60dB means that it can resolve signals in the range in amplitude from x to 1000x. Dynamic range is important in communication applications where signal strengths vary dramatically. For an analog-to-digital converter (ADC), if the signal is too large, it will over-range the ADC input. If it is too small, the signal will get lost in the quantization noise of the converter.

effective number of bits (ENOB): The measured performance (in bits) of an analog-to-digital converter (ADC) with respect to input frequency f_{IN}. As f_{IN} increases, overall noise (particularly the distortion components) also increases, thereby reducing the ENOB and SINAD(signal-to-noise and distortion ratio). ENOB is related to SINAD by the following equation:

$$ENOB = \frac{SINAD - 1.76}{6.02}$$

full-scale (FS) error: For the ideal transfer curve, the code edge transition that causes all ones in the digital code to occur 1.5LSB below full scale. The full-scale error is the difference between this code transition of the ideal transfer function and the actual measured value at this code transition. Unlike gain error, offset error is included in the FS error measurement.

full-power bandwidth (FPBW) : A large-signal (i.e. at or near full scale) analog input is applied to the analog-to-digital converter (ADC), and the input frequency is swept up

to the point where the amplitude of the digitized conversion result has decreased by -3dB. This point is defined as the "full-power bandwidth frequency."

gain error: With a full-scale analog input voltage applied to the analog-to-digital converter (ADC) (resulting in all ones in the digital code) or a digital code of all ones applied to the digital-to-analog converter (DAC), gain error is defined as the amount of deviation between the ideal transfer function and the measured transfer function (with the offset error removed). Gain error is usually expressed in LSB or a percent of full-scale range (%FSR). It can be calibrated out with hardware or in software.

glitch impulse energy: The amount of energy that appears at the digital-to-analog converter (DAC) output when a major carry transition occurs. It is measured typically in nV*s and given by the area under the curve on a voltage-versus-time graph.

integral nonlinearity (INL) error: The amount of deviation of the measured transfer function of an analog-to-digital converter (ADC) or a digital-to-analog converter (DAC) from the ideal transfer function (defined as a straight line drawn from zero to full scale). (Sometimes a "best-fit" straight line is used where the ideal transfer function is represented by a straight line drawn between the end points of the actual transfer function. This method is referred to as "end-point linearity.") The INL error is also defined as the sum of the DNL errors starting from code 000 to the code where the INL measurement is desired.

intermodulation distortion (IMD): An analog-to-digital converter (ADC) test where two signals (f1 and f2) of equal amplitude that are also very close in frequency are applied to the ADC input. A frequency spectrum plot of the data reveals the amount of distortion due to the ADC's digitizing of the two signals. IMD is typically specified in decibels.

$$\text{IMD} = 20\text{Log}_{(10)} \frac{\text{RMS sum of distortion terms}}{\text{Input (Volts, RMS)}}$$

where the distortion terms are given by

2nd-order terms: - f1 + f2, f1 - f2

3rd-order terms: - 2f1 + f2, 2f1 - f2, f1 + 2f2, f1 - 2f2

least significant bit (LSB): In the binary number scheme, the bit position in a group of bits given the smallest weighted value. The LSB is typically the furthest-right bit in a grouping of data bits. The LSB weight is given by the full-scale range of the converter divided by 2^N, where N is the converter's resolution.

major carry transition: The midscale point where the MSB changes from low to high and all other bits change from high to low, or where the MSB changes from high to low

and all other bits change from low to high. This point is often where the worst switching noise occurs. *See also* most significant bit (MSB).

monotonic (referring to the DAC): A digital-to-analog converter (DAC) is said to be monotonic if, as the DAC code is increased, the analog output never decreases in value. A +/-1LSB DNL guarantees that a DAC is monotonic.

most significant bit (MSB): In the binary number scheme, the bit position in a group of bits given the largest weighted value. The MSB is typically the furthest-left bit in a grouping of data bits.

multiplying DAC: A digital-to-analog converter (DAC) that allows an AC signal to be applied to the reference input pin. This allows the DAC to be used as a digital attenuator by feeding the signal of interest into the reference pin and using the DAC codes to scale it.

no missing codes (referring to the ADC): An analog-to-digital converter (ADC) has no missing codes if, as a ramp signal is applied to the ADC input, all digital codes appear in the resulting conversion data.

Nyquist frequency: In sampling theory, the Nyquist frequency is the maximum frequency that can be applied to the analog-to-digital converter (ADC) input with no aliasing effects. The Nyquist frequency is defined as the "sampling frequency/2."

offset binary coding: For bipolar signals, offset binary is a digital coding scheme in which the most negative value is represented by all zeros (00000000) and the most positive value is represented by all ones (11111111).

offset error (bipolar): When using a bipolar converter, the ideal midscale transition occurs at AGND -0.5LSB. Bipolar offset error is the measured deviation from this ideal value.

offset error (unipolar): For an ideal converter, the first transition occurs at 1/2LSB, above zero. Offset error is the amount of deviation between the measured first transition point and the ideal point. For an analog-to-digital converter (ADC), 0V is applied to the ADC input and then the analog input voltage is increased until the first transition occurs. This voltage can be converted to LSBs by dividing it by the LSB step size, which is $VREF/2^N$. For a digital-to-analog converter (DAC), offset error is determined by loading a code of all zeros into the DAC and measuring the analog output voltage. Offset error is easily calibrated out in software or with hardware.

oversampling: Sampling the analog-to-digital converter (ADC) input at a much higher frequency than that of Nyquist. This technique provides a processing gain by effectively reducing the noise floor of the converter. A 2X increase in oversampling rate theoretically improves signal-to-noise by 3dB.

power-supply rejection (PSR): The amount of change in the converter's value (often measured at full scale), as the power-supply voltage changes from its nominal value. PSR assumes that the converter's linearity is unaffected by changes in the power-supply voltage. It is often measured in LSB/V. Power-supply rejection ratio (PSRR) is the ratio of the input signal change (in volts) to the change in the converter output (in volts). It is measured typically in decibels.

quantization error: When a continuous time signal is digitized, because there isn't an infinite number of discrete digital levels, the difference between the actual analog value and the digital representation of that value is defined as the quantization error.

ratiometric measurement (referring to an ADC): Rather than using a voltage reference with an absolute value, a ratio of the signal applied to the transducer (that is, the load cell or bridge) is also applied to the voltage reference input of the analog-to-digital converter (ADC), thereby eliminating any errors introduced by a changing reference.

resolution: When an analog signal is digitized, it is represented by a finite number of discrete voltage levels. The resolution is the number of discrete levels that are used to represent the signal. To more accurately replicate the analog signal, the resolution must be increased. Resolution is usually defined in bits. Using converters with higher resolutions will reduce the quantization error.

root mean square (RMS): The effective value or effective DC value that an AC signal represents. For a sine wave, the RMS value is 0.707 times the peak value, or 0.354 times the peak-to-peak value.

sampling rate/sampling frequency: The rate at which a converter acquires the input signal, digitizes it, and outputs data to a μC/DSP. It is specified in samples per second or Hertz (Hz) and is also referred to as the "throughput rate."

settling time (referring to the DAC): The actual amount of time it takes a digital-to-analog converter (DAC) output to reach its final value (within a certain percentage) once the DAC has accepted a command to change its output value. This time can be affected by the slew rate of an output amplifier and by the amount of amplifier ringing and signal overshoot.

signal-to-noise and distortion ratio (SINAD): The RMS value of the sine wave f(IN) (input sine wave for an ADC, reconstructed output sine wave for a DAC) to the RMS value of the noise of the converter from DC to the Nyquist frequency, including harmonic content. It is typically expressed in decibels.

$$\text{SINAD} = 20\text{Log}_{(10)} \frac{\text{Input (Volts, RMS)}}{\text{Noise + Harmonics}}$$

signal-to-noise ratio (SNR): The RMS value of the sine wave $f_{(IN)}$ (input sine wave for an ADC, reconstructed output sine wave for a DAC) to the RMS value of the noise of the converter from DC to Nyquist frequency, excluding noise at DC and harmonic distortion content. It is typically expressed in decibels.

$$SNR = 20Log_{(10)} \frac{\text{Input (Volts, RMS)}}{\text{Noise}}$$

signed binary coding: A coding scheme in which the MSB represents the sign (positive or negative) of a binary number. In this scheme, "-2" is represented by 10000010 and "2" is represented by 00000010.

slew rate: The maximum rate at which the digital-to-analog converter (DAC) output can change, or the maximum rate at which an input signal can change without resulting in an error in the digitized representation of the input signal.

small-signal bandwidth: A small-amplitude signal is applied to the analog-to-digital converter (ADC) in such a way that the signal's slew rate will not limit the ADC's performance, and the input frequency is swept up to the point where the amplitude of the digitized conversion result has decreased by -3dB. The small-signal bandwidth is often limited by the track/hold amplifier performance.

spurious-free dynamic range (SFDR): The ratio of the RMS value of the sine wave $f_{(IN)}$ (input sine wave for an ADC, reconstructed output sine wave for a DAC) to the RMS value of the peak spur observed in the frequency domain. It is typically expressed in decibels. SFDR is important in certain communication applications that require maximizing the dynamic range of the converter.

total harmonic distortion (THD): The RMS value of the distortion appearing at multiples (harmonics) of the input (or output for a DAC) frequency to the RMS value of the input (or output) sine wave. Only harmonics within the Nyquist limit are included in the measurement. It is typically expressed in decibels.

$$THD = 20Log_{(10)} \frac{\text{Noise}}{\text{Input (Volts, RMS)}}$$

transition noise (referring to the ADC): The range of input voltages that cause the analog-to-digital converter (ADC) result to toggle between codes. As the input voltage is increased, the voltage that defines where a code transition occurs (code edge) has an associated amount of noise due to this transition. This specification is often given as an RMS value rather than peak-to-peak.

two's complement coding: A binary digital coding scheme for negative numbers that simplifies addition and subtraction computations. In this scheme, the number "-2" is represented by 11111110 and "2" is represented by 00000010.

undersampling: A technique whereby the analog-to-digital converter (ADC) sampling rate is lower than the input frequency (which normally results in a loss of signal information), which causes aliasing. With proper filtering of the input signal and proper frequency selection (including f_{IN} and f_{SAMPLE}), the aliased components that contain the signal information can be shifted from a higher frequency to a lower frequency and converted. This method effectively uses the ADC as a downconverter by shifting higher bandwidth signals into the ADC's desired band of interest. The bandwidth of the ADC's track/hold must be capable of handling the highest frequency signals for this technique to work.

unipolar: A signal that swings from zero to the positive full-scale range, thus having only a positive polarity.

APPENDIX B

Maxim Max 1284 Evaluation System

The MAX1284 evaluation system (EV system) is a complete data-acquisition system consisting of a MAX1284 evaluation kit (EV kit) and a Maxim 68HC16MODULEDIP microcontroller (μC) module. The MAX1284 is a high-speed, 12-bit data acquisition system. Windows 95/98 software provides a handy user interface to exercise the MAX1284's features.

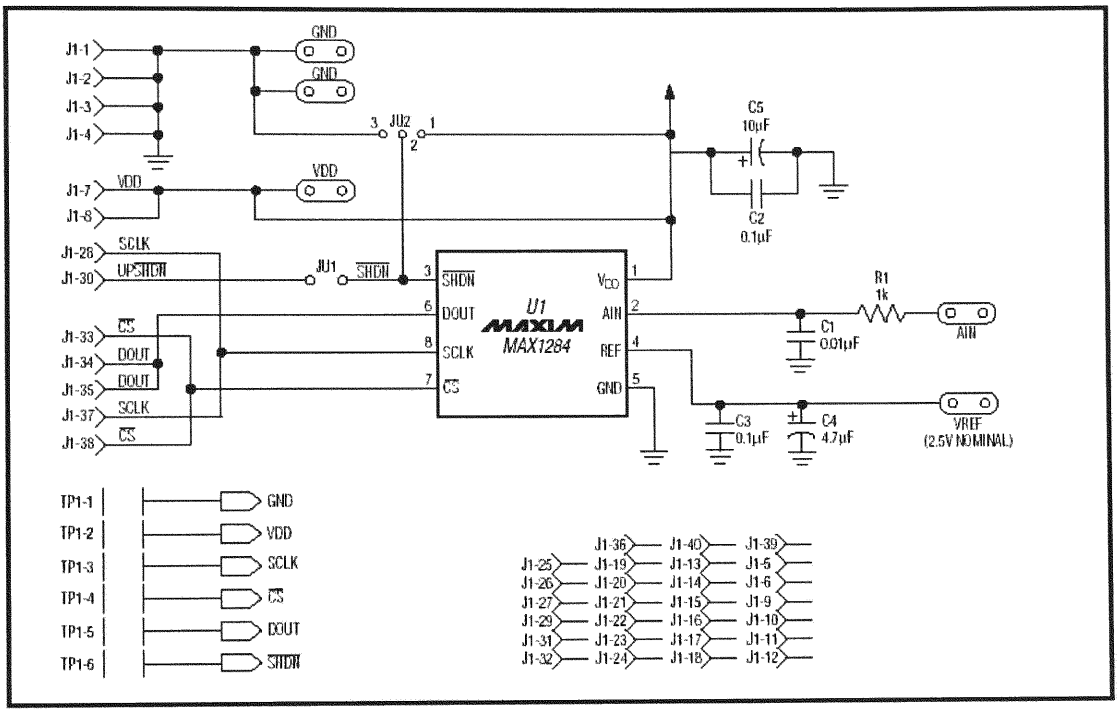

Figure 1. MAX1284 EV Kit Schematic Diagram

Detailed Hardware Description

The MAX1284, U1, is a high-speed, 12-bit data-acquisition system. Resistor R1 (1k.) and capacitor C1 (0.01μF) form a single-pole, low-pass anti-aliasing filter with a nominal 10μs time constant and a corner frequency of approximately 16kHz. C3 and C4 bypass the analog-to-digital converter's (ADC's) voltage reference. When plugged into the 68HC16MODULE, the VDD circuit is powered by +5V.

Problem: No output measurement. System seems to report zero voltage or fails to make a measurement.

Check the VDD supply voltage. Check the reference voltage using a DVM. Use an oscilloscope to verify that the conversion-start signal is being strobed. Verify that SHDN is being driven high.

Problem: Measurements are erratic, unstable; poor accuracy.

Check the reference voltage using a DVM. Use an oscilloscope to check for noise. When probing for noise, keep the oscilloscope ground return lead as short as possible, preferably less than 1/2in (10mm).

JUMPER	POSITION	FUNCTION
JU1	Closed	$\overline{SHDN}$ driven by µC; JU2 must be open
	Open*	$\overline{SHDN}$ set by JU2
JU2	1-2*	Operate
	2-3	Shutdown

*** Default Configuration**

Table 1. Jumper JU1 Functions

DESIGNATION	QTY	DESCRIPTION
C1, C2, C3	3	1µF ceramic capacitors
C4, C5	2	22µF, 25V radial-lead electrolytic capacitors
C6, C7	2	22pF capacitors
C8	1	0.01µF capacitor
C9	0	Reference designator, not used
C10–C14	5	0.1µF capacitors
D1	1	1N4001 diode
J1	1	40-pin right-angle male connector
J2	1	2-circuit terminal block
J3	1	Right-angle printed circuit board mount, DB9 female socket
J4	0	Open
JU1	0	Open
JU2	0	Reference designator, not used
JU3	0	Open
JU4	0	Open
JU5	0	Open
L1	0	Open
L2	0	Open
LED1	1	Light-emitting diode
R1	1	10MΩ, 5% resistor

Table 2. 68HC16 Module Component List

DESIGNATION	QTY	DESCRIPTION
R2	1	330kΩ, 5% resistor
R3, R4	2	10kΩ, 5% resistors
R5	1	470Ω, 5% resistor
R6	1	10kΩ SIP resistor
SW1	1	Slide switch
SW2	1	Momentary pushbutton switch
U1	1	68HC16 µC MC68HC16Z1CFC16 (132-pin plastic quad flat pack)
U2	1	Maxim MAX233CPP
U3	1	27C256 EPROM containing monitor program
U4	1	7805 regulator, TO-220 size
U5	1	62256 (32K x 8) static RAM
U6	1	74HCT245 bidirectional buffer
U7	1	Maxim MAX707CPA
Y1	1	32.768kHz watch crystal
None	4	Rubber feet
None	1	28-pin socket for U3
None	1	20-pin socket for U6
None	1	3" x 5" printed circuit board
None	1	Heatsink for U4, thermalloy # 6078

Table 3. 68HC16 Module Component List

68HC16 Module

General Description

The 68HC16 module is an assembled and tested printed-circuit board intended for use with Maxim's high speed serial-interface evaluation kits (EV kits). The module uses an inexpensive 8-bit implementation of Motorola's MC68HC16Z1 microcontroller (µC) to collect data samples at high speed using the QSPI™ interface. It requires an IBM-compatible personal computer and an external DC power supply, typically 12V DC or as specified in EV kit manual. Maxim's 68HC16 module is provided to allow customers to evaluate selected Maxim products. It is not intended to be used as a microprocessor development platform, and such use is not supported by Maxim.

Detailed Description

Power Input Connector J2

The 68HC16 module draws its power from a user-supplied power source connected to terminal block J2. Be sure to note the positive and negative markings on the board. A three-terminal 5V regulator allows input voltages between 8V and an absolute maximum of 20V. The 68HC16 module typically requires 200mA of input current.

68HC16 Microcontroller

U1 is Motorola's 68HC16Z1 µC. Contact Motorola for µC information, development, and support. Maxim EV kits use the high-speed queued serial peripheral interface (QSPI) and the internal chip-select generation. A MAX707 on the module monitors the 5V logic supply, generates the power-on reset, and produces a reset pulse whenever the reset button is pressed.

Serial Communications

J3 is an RS-232 serial port, designed to be compatible with the IBM PC 9-pin serial port. Use a straightthrough DB9 male-to-female cable to connect J3 to this port. If the only available serial port has a 25-pin connector, you may use a standard 25-pin to 9-pin adapter. Table 1 shows the pinout of J3.
The MAX233 is an RS-232 interface voltage level shifter with two transmitters and two receivers. It includes a built-in charge pump with internal capacitors that generates the output voltages necessary to drive RS-232 lines.

40-Pin Data Connector J1

The 20 x 2 pin header connects the 68HC16 module to a Maxim EV kit. Table 2 lists the function of each pin. Note that 68HC16 object code is not compatible with 68HC11 object code. Use the 68HC16 module only with those modules that are designed to support it, and only download code that is targeted for the 68HC16 module. Downloading incorrect object code into the 68HC16 module will have unpredictable results.

Address Ranges

The 68HC16 µC generates various enable signals for different address ranges. The ROM and RAM enable signals are fed directly to the respective chips. Several additional signals (J1.11–J1.14) are available on the data connector to be used by Maxim EV kits. Table 6 outlines the address ranges for each of the elements found on the 68HC16 module, and Table 7 is a truth table that describes the logic for each of the 68HC16's chip-select outputs. Because the addresses are not completely decoded, the boot ROM and user RAM have shadows.

PIN	NAME	FUNCTION
1	DCD	Handshake; hard-wired to DTR and DSR
2	RXD	RS-232-compatible data output from 68HC16 module
3	TXD	RS-232-compatible data input to 68HC16 module
4	DTR	Handshake; hard-wired to DCD and DSR
5	GND	Signal ground connection
6	DSR	Handshake; hard-wired to DCD and DTR
7	RTS	Handshake; hard-wired to CTS
8	CTS	Handshake; hard-wired to RTS
9	None	Unused

Table 4. Serial Communications Port J3

PIN	NAME	FUNCTION
1–4	GND	Ground
5, 6	VPREREG	Unregulated input voltage
7, 8	VCC	+5V from on-board regulator
9	RD	Read strobe
10	WR	Write strobe
11	7E000	Chip select for 7E000–7E7FF
12	7E800	Chip select for 7E800–7EFFF
13	7F000	Chip select for 7F000–7F7FF
14	7F800	Chip select for 7F800–7FFFF
15	A00	Address bit 0 (LSB)
16	A01	Address bit 1
17	A02	Address bit 2
18	A03	Address bit 3
19	EXTD0	Buffered data bus 0 (LSB)
20–26	EXTD1–7	Buffered data bus bits 1–7
27	IC1	General I/O port bit 0 (LSB)
28	IC2	General I/O port bit 1
29	IC3	General I/O port bit 2
30	OC1	General I/O port bit 3
31	OC2	General I/O port bit 4
32	OC3	General I/O port bit 5
33	OC4	General I/O port bit 6
34	IC4	General I/O port bit 7
35	MISO	QSPI master-in, slave-out
36	MOSI	QSPI master-out, slave-in
37	SCK	QSPI serial clock
38	PCS0/SS	QSPI chip-select output
39	CLKOUT	System clock output
40	PWMA	Pulse-width-modulator output

Table 5. 40 Pin Data-Connector Signals

PIN	FUNCTION
00000–07FFF	Boot ROM (U3, strobed by CSBOOT)
08000–0FFFF	Shadow of boot ROM
10000–17FFF	User RAM (U5, strobed by CS0 and CS2)
18000–1FFFF	Shadow of user RAM
20000–203FF	Internal standby RAM; 1kbyte
20400–7DFFF	Unused
7E000–7E7FF	External chip select (J1 pin 11) (CS7)
7E800–7EFFF	External chip select (J1 pin 12) (CS8)
7F000–7F7FF	External chip select (J1 pin 13) (CS9)
7F800–7FFFF	External chip select (J1 pin 14) (CS10)
80000–F7FFF	Not accessed by the 68HC16
F8000–FF6FF	Unused
FF700–FF73F	68HC16's built-in ADC (not used)
FF740–FF8FF	Unused
FF900–FF93F	General-purpose timer module (GPT)
FF940–FF9FF	Unused
FFA00–FFA7F	System integration module (SIM)
FFA80–FFAFF	Unused
FFB00–FFB07	Internal standby RAM (SRAM) control registers
FFB08–FFBFF	Unused
FFC00–FFDFF	Queued serial module (QSM)
FFE00–FFFFF	Unused

Table 6. 68HC16 Module Memory Map
(all address values are in 20-bit hex)

Boot ROM

The boot ROM, U3, is configured as an 8-bit memory device. Resistor R4 pulls data bit 0 low during system reset, forcing the µC to fetch instructions using only the upper eight data bits. The boot ROM checks the system and waits for commands from the host.

68HC16 Module Self Check

To test the 68HC16 module's integrity, connect the power supply to the power terminals (J2). Do not connect anything to J1 or J3. Slide the power switch SW1 to the "ON" position. The LED will light up, and will flash within 5 seconds. If the LED flashes with a 50%-on/50%-off duty cycle, then it passed its self-check. Note that this test does not exercise the RS-232 port or the EV kit 40-pin interface, but it does confirm that the power supply, microprocessor, ROM, and RAM passed the self-test. If the LED flashes with a 10%-on/90%-off duty cycle, then it failed its self-check. Most likely, the RAM chip (U5) is bad. If the LED remains on and does not flash, then the problem is U3 (the

EPROM), U1 (the microprocessor), U4 (the regulator), the MAX707 reset generator, or the power supply. Use a voltmeter to verify that the power supplies are good. Check the power-supply input and the +5V output from the regulator. Use an oscilloscope to see if the 32.768kHz reference oscillator is running.

ADDRESS RANGE	CSBOOT	CS0	CS1	CS2	CS5	CS6	CS7	CS8	CS9	CS10
0xxxx read	L	H	H	H	H	H	H	H	H	H
1xxxx read	H	H	H	L	H	H	H	H	H	H
1xxxx write	H	L	H	H	H	H	H	H	H	H
7E0xx read	H	H	L	H	H	L	L	H	H	H
7E0xx write	H	H	H	H	L	L	L	H	H	H
7E8xx read	H	H	L	H	H	L	H	L	H	H
7E8xx write	H	H	H	H	L	L	H	L	H	H
7F0xx read	H	H	L	H	H	L	H	H	L	H
7F0xx write	H	H	H	H	L	L	H	H	L	H
7F8xx read	H	H	L	H	H	L	H	H	H	L
7F8xx write	H	H	H	H	L	L	H	H	H	L

Table 7. 68HC16 Chip-Select Outputs Truth Table

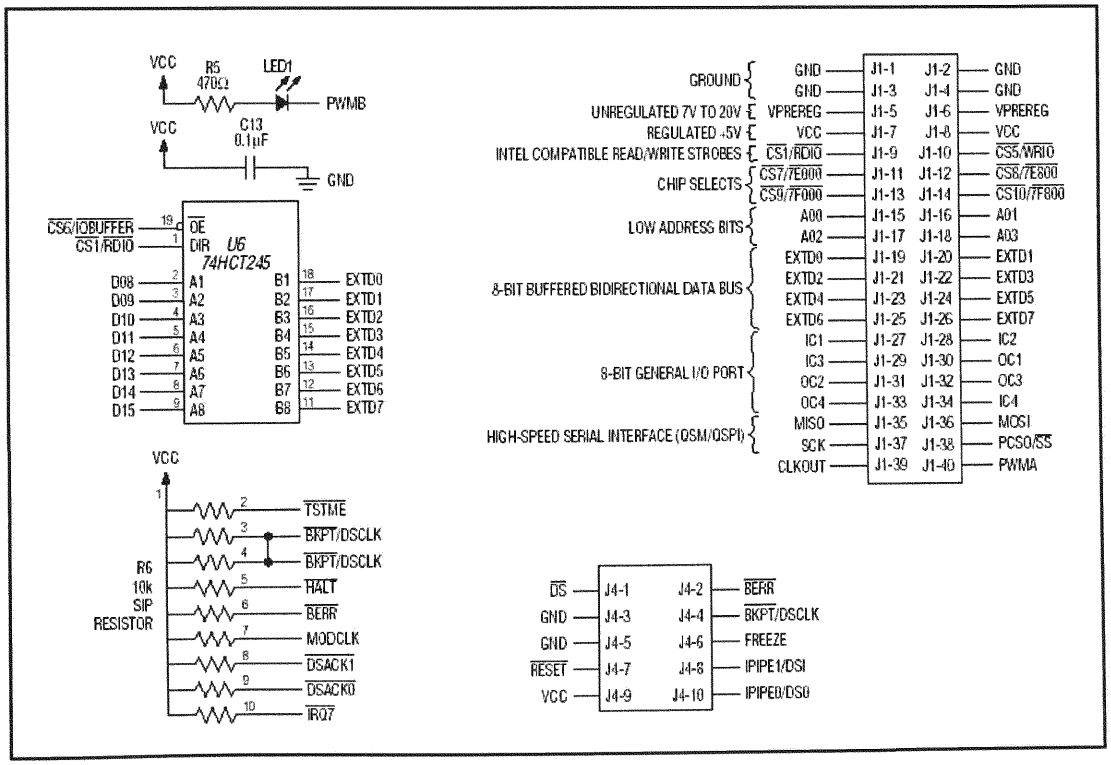

Figure 5. 68HC16 Module Schematic

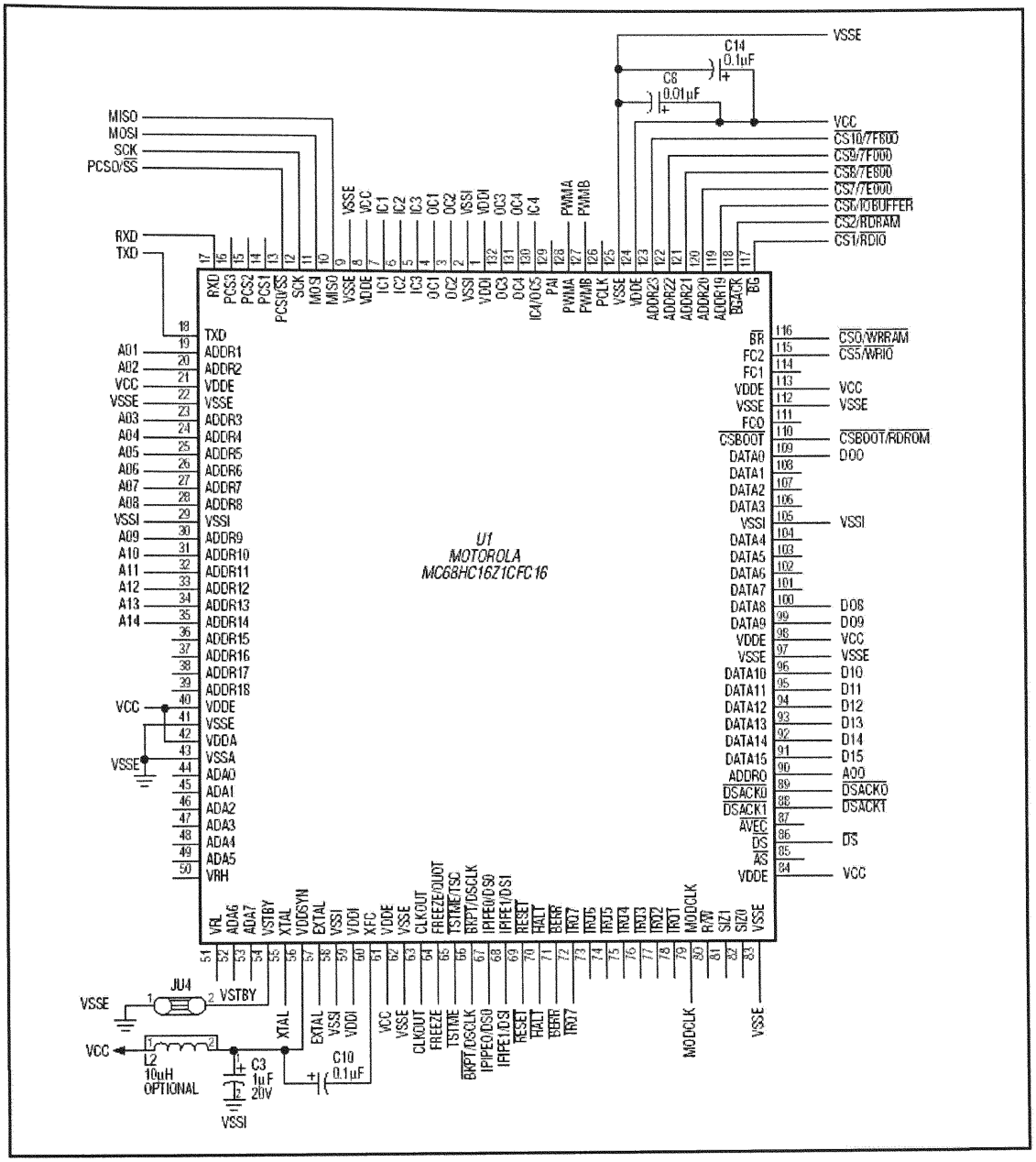

Figure 6. 68HC16 Module Schematic (continued)

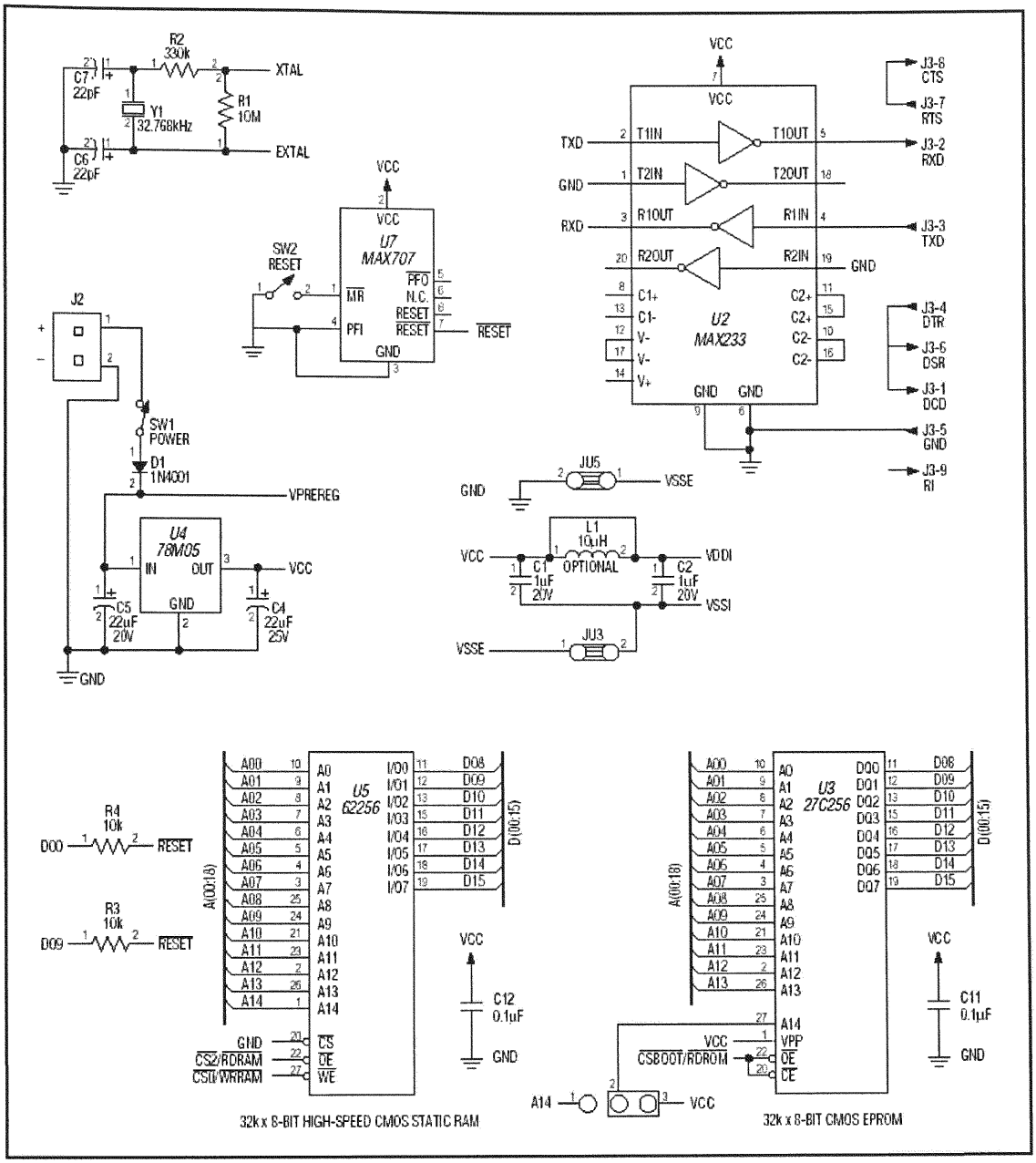

Figure 7. 68HC16 Module Schematic (continued)

APPENDIX C

Client/Server Implementation Code

```
//client source code
//These are some of the main function implemented

void CChatRoomDlg::OnBtnsend()
{
        // TODO: Add your control notification handler code here
        SetDefaultCharFormat4Input();
        SendPkg();
}
void CChatRoomDlg::OnClose()
{
        // TODO: Add your message handler code here and/or call default
        if(bConnected || bSignIn){
                MessageBox("Pls log off before you
leave.",NULL,MB_OK|MB_ICONWARNING);
                return;
        }
        Finalize();
        SaveIniFile();
        CExpandingDialog::OnClose();
}
void CChatRoomDlg::Finalize()
{
        if(m_pArchiveIn != NULL){
                delete m_pArchiveIn;
                m_pArchiveIn = NULL;
        }
        if(m_pArchiveOut != NULL){
                delete m_pArchiveOut;
                m_pArchiveOut = NULL;
        }
        if(m_pSocketFile != NULL){
                delete m_pSocketFile;
                m_pSocketFile = NULL;
        }

        if(m_pClientSocket != NULL && bConnected){
                bConnected = FALSE;
                bSignIn = FALSE;
                m_pClientSocket->Close();
                delete m_pClientSocket;
        }
```

```
}

void CChatRoomDlg::AssemblePkg(CPkg& pkg)
{
        pkg.Init();
        //the next 5 are for client side

        m_wndInput.GetWindowText(pkg.strText ); //1
        pkg.strName = m_strScreenName;                  //2
        pkg.bAway = bAway;                                      //3
        pkg.iMyIcon = m_iMyIcon;                                        //4
        pkg.bBold = bCharBold;                                  //5
        pkg.bItalic  = bCharItalic;                     //6
        pkg.bUnderLine  = bCharUnderline;       //7
        pkg.bStrikeOut = bCharStrikeOut;        //8
        pkg.clrText  = clrChar;                                 //9
        pkg.fontName  = m_fontName;                             //10
        pkg.fontSize  = m_fontSize;                     //11

//      pkg.ipAddress                                           //12    reserved for
Server
//      pkg.request = UNDEFINED;                        //13    reserved
//      pkg.pSocket                                                     //14
        reserved for Server
//      pkg.port                                                        //15
        reserved for Server


}

void CChatRoomDlg::SendPkg()
{

        if(!bConnected || !bSignIn){
                m_wndInput.SetWindowText("");
                return;
        }

        CPkg pkg;

        AssemblePkg(pkg);

        pkg.request = MESG;

        try{
```

```
                pkg.Serialize(*m_pArchiveOut);
                m_pArchiveOut->Flush();
                m_wndInput.SetWindowText("");
        }
        catch(CFileException* e){
                TCHAR lpszError[255];
                e->GetErrorMessage(lpszError,255);
                AfxMessageBox(lpszError);
                exit(0);
        }
}

void CChatRoomDlg::ConnectServer(CString serverIp,UINT port)
{
        //Create the socket, as well as the initialization of the socketfile and archive
        m_pClientSocket = new CChatSocket(this);
        m_pClientSocket->Create();

        m_pSocketFile = new CSocketFile(m_pClientSocket);
        m_pArchiveIn = new CArchive(m_pSocketFile,CArchive::load);
        m_pArchiveOut = new CArchive(m_pSocketFile,CArchive::store);

        while(!m_pClientSocket->Connect(serverIp,port)){
                if(MessageBox("Not Connected. Do Yo want to try again?","Connecting
Server...",MB_ICONQUESTION|MB_YESNO)==IDNO){
                        Finalize();
                        return;
                }
        }
        bConnected = TRUE;
}

void CChatRoomDlg::ParsePkg()
{
        CPkg pkg;

        try{
                do{
                        pkg.Serialize(*m_pArchiveIn);
                }while(!m_pArchiveIn->IsBufferEmpty());
        }catch(CFileException* e){
                m_pArchiveIn->Abort();
        }

//      if(pkg.request == SIGNIN){
```

```
//              bSignIn = TRUE;
//              m_hIcon = (HICON)m_imgContactList.ExtractIcon(m_iMyIcon);
                //only now, all the authentication done, we can really begin talking...
                //set the default char format for Input window - yh
//              SetDefaultCharFormat4Input();

//      }

        if(pkg.request == SIGNOUT){
                bSignIn = FALSE;
                m_wndShow.ShowMessage(pkg);
//              OnSignOut();
        }

        if(pkg.request == SVRSTOP){
                m_wndShow.ShowMessage("Server",pkg.strText);
        }

        if(pkg.request == SVRSTART){
                m_wndShow.ShowMessage("Server",pkg.strText);
        }

        if(pkg.request == NEW){
                //update the contact list - yh
                CString strTmp = pkg.strName;
                m_nameString.RemoveAll();
                m_wndShow.ShowMessage("Server",pkg.strText);

                int index;
                while((index=strTmp.Find(";"))!=-1){
                        CString name = strTmp.Left(index);
                        m_nameString.Add(name);
                        strTmp = strTmp.Right(strTmp.GetLength() -(index+1));
                }

                UpdateListBox();
        }
        if(pkg.request == OFF){

                m_wndShow.ShowMessage("Server",pkg.strText);
                CString strTmp = pkg.strName;
                m_nameString.RemoveAll();

                int index;
                while((index=strTmp.Find(";"))!=-1){
```

```
                    CString name = strTmp.Left(index);
                    m_nameString.Add(name);
                    strTmp = strTmp.Right(strTmp.GetLength() -(index+1));
                }

                UpdateListBox();
        }
//      if(pkg.request == SERVER){
//              m_wndShow.ShowMessage(pkg);
//      }

        if(pkg.request == CONN){
                m_wndShow.ShowMessage(pkg);
        }

        if(pkg.request == MESG){
                if(!pkg.strText.IsEmpty())
                        m_wndShow.ShowMessage(pkg);
        }

}

void CChatRoomDlg::OnConnect()
{
        // TODO: Add your command handler code here
        CConnectDlg dlg;
        if(bConnected){
                bConnected = FALSE;
//              if(bSignIn){
                        bSignIn = FALSE;
//                      CString tmp;
//                      tmp.Format("%s just logged off.",m_strScreenName);
//                      m_wndShow.ShowMessage("Server",tmp);
//              }

                SendPkg(OFF);
                return;
        }
        if(dlg.DoModal() == IDOK){
                CString strTmp = dlg.GetIpAddress();
                m_strServerIP = strTmp;
                ConnectServer(strTmp,1500);
                return;
        }
}
```

```
void CChatRoomDlg::OnSignIn()
{
	// TODO: Add your command handler code here
	CSignInDlg dlg;
	if(!bConnected) return;
	if(bSignIn) return;
	if(dlg.DoModal() == IDOK){
		m_strScreenName = dlg.GetName();
		if(m_strScreenName.IsEmpty()) return;
		SendPkg(NEW);	//sign in, then send the NEW request to the server -
yh
		bSignIn = TRUE;
		SetDefaultCharFormat4Input();

	}
}


CString CChatRoomDlg::GetServerIP()
{
	return m_strServerIP;
}

CString CChatRoomDlg::GetScreenName()
{
	return m_strScreenName;
}

void CChatRoomDlg::OnMylogo()
{
	// TODO: Add your command handler code here
	CIconDlg dlg;
	if(dlg.DoModal() == IDOK){
		m_iMyIcon = dlg.GetMyIcon();
		m_hIcon = (HICON)m_imgContactList.ExtractIcon(m_iMyIcon);
		SetIcon(m_hIcon,TRUE);
		SetIcon(m_hIcon,FALSE);
	}
}
void CChatRoomDlg::OnAway()
{
	// TODO: Add your command handler code here
	bAway = !bAway;
}
void CChatRoomDlg::OnDetails()
```

```
{
	// TODO: Add your control notification handler code here
}

// update the list box on the right regarding the current name list -yh
void CChatRoomDlg::UpdateListBox()
{
	m_wndContactList.ResetContent();
	for(int i=0;i<m_nameString.GetSize();i++){
		CString name = m_nameString.GetAt(i);
		m_wndContactList.InsertString(i,name);
	}
}

// send the pkg stream to the server with indicating request - yh
void CChatRoomDlg::SendPkg(int request)
{
	CPkg pkg;
	AssemblePkg(pkg);
	// send a quest to the server
	pkg.request = request;
	try{
		pkg.Serialize(*m_pArchiveOut);
		m_pArchiveOut->Flush();
		m_wndInput.SetWindowText("");
	}
	catch(CFileException* e){
		TCHAR lpszError[255];
		e->GetErrorMessage(lpszError,255);
		AfxMessageBox(lpszError);
		exit(0);
	}
}

void CChatRoomDlg::OnUpdateConnect(CCmdUI* pCmdUI)
{
	// TODO: Add your command update UI handler code here
	pCmdUI->SetCheck(bConnected);
}
void CChatRoomDlg::OnUpdateAway(CCmdUI* pCmdUI)
{
	// TODO: Add your command update UI handler code here
	pCmdUI->Enable(bConnected && bSignIn);
	pCmdUI->SetCheck(bAway);
}
```

```
void CChatRoomDlg::OnUpdateSignIn(CCmdUI* pCmdUI)
{
	// TODO: Add your command update UI handler code here
	pCmdUI->Enable(bConnected);
	pCmdUI->SetCheck(bSignIn);
}
void CChatRoomDlg::OnAppAbout()
{
	// TODO: Add your command handler code here
		CAboutDlg dlgAbout;
		dlgAbout.DoModal();

}
void CChatRoomDlg::OnSignOut()
{
	// TODO: Add your command handler code here
	if(bConnected && bSignIn){
		bConnected = FALSE;
		bSignIn = FALSE;
		CString tmp;
		tmp.Format("%s just logged off.",m_strScreenName);
		m_wndShow.ShowMessage("Server",tmp);
		SendPkg(OFF);
		m_nameString.RemoveAll();
		m_wndContactList.ResetContent();
		return;
	}
}

void CChatRoomDlg::LoadIniFile()
{
	CFile iniFile;

	if(iniFile.Open("Config.dat",CFile::modeNoTruncate|CFile::modeCreate|CFile::m
odeRead)){

		if(iniFile.GetLength() == 0) return;

		CArchive* ar = new CArchive(&iniFile,CArchive::load);

		*ar >> m_strServerIP;
		*ar >> m_strScreenName;

		*ar >> bCharBold ;
		*ar >> bCharItalic ;
```

```
                *ar >> bCharUnderline ;
                *ar >> bCharStrikeOut ;
                *ar >> clrChar ;
                *ar >> m_fontName ;
                *ar >> m_fontSize ;

                //for combo
                int nIndex;
                *ar >> nIndex;
                m_wndRichTextBar.m_cmbFontSize.SetCurSel(nIndex);

                *ar >> nIndex;
                m_wndRichTextBar.m_cmbFontName.SetCurSel(nIndex);

                delete ar;

                iniFile.Close();
        }


}
void CChatRoomDlg::SaveIniFile()
{
        CFile iniFile;
        if(iniFile.Open("Config.dat",CFile::modeNoTruncate|CFile::modeCreate|CFile::m
odeReadWrite)){

//              if(iniFile.GetLength() == 0) return;

                CArchive* ar = new CArchive(&iniFile,CArchive::store);

                *ar << m_strServerIP;
                *ar << m_strScreenName;

                *ar << bCharBold ;
                *ar << bCharItalic ;
                *ar << bCharUnderline ;
                *ar << bCharStrikeOut ;
                *ar << clrChar ;
                *ar << m_fontName ;
                *ar << m_fontSize ;

                //for combo
                int nIndex = m_wndRichTextBar.m_cmbFontSize.GetCurSel();
                *ar << nIndex;
```

```
                nIndex = m_wndRichTextBar.m_cmbFontName.GetCurSel();
                *ar << nIndex;

                ar->Flush();

                delete ar;
                iniFile.Close();
        }
}

//SERVER IMPLEMENTATION SOURCE CODE


BOOL CChatServer::OnInitDialog()
{
        CPropertyPage::OnInitDialog();

        // Create the ToolTip control.
        m_tooltip.Create (this);
        m_tooltip.Activate (TRUE);

        // TODO: Use one of the following forms to add controls:
        // m_tooltip.AddTool (GetDlgItem (IDC_<name>), <string-table-id>);
        // m_tooltip.AddTool (GetDlgItem (IDC_<name>), _T ("<text>"));

        //set imagelist for the listctrl
        m_imgComputer.Create(16,16,TRUE,1,1);
        m_imgComputer.Add(AfxGetApp()->LoadIcon(IDI_ICON1));

        //start listening
        BeginListening(1500);


        return TRUE;
}

void CChatServer::OnStart()
{
        // TODO: Add your control notification handler code here
        bListening = !bListening;

        CString strTitle;
        CButton* pButton = (CButton*)GetDlgItem(IDC_START);
        pButton->GetWindowText(strTitle);
```

```
        if(strTitle == "Start"){
                if(!m_ContactList.IsEmpty()){
                        UpdateInfo2All("Server is now starting listening
again.",SVRSTART);
                }
                strTitle = "Stop";
                CStatic* pStatic = (CStatic*)GetDlgItem(IDC_IPADDRESS);
                pStatic->SetWindowText(GetServerIP());
                pStatic->EnableWindow(TRUE);
        }
        else{
                UpdateInfo2All(
                        " The Server is now stopping listening.But your Link still exists.
Pls try again later. Sorry for any inconvinience.",
                        SVRSTOP);
                strTitle = "Start";
                CStatic* pStatic = (CStatic*)GetDlgItem(IDC_IPADDRESS);
                pStatic->SetWindowText(GetServerIP());
                pStatic->EnableWindow(FALSE);
        }


        pButton->SetWindowText(strTitle);

}

void CChatServer::SetOwner(CMainSheet *pMainSheet)
{
        m_pMainSheet = pMainSheet;
}

BOOL CChatServer::BeginListening(UINT port)
{
        ASSERT(m_pListenSocket == NULL);

        if(!m_ContactList.IsEmpty())
                CloseAllClients();

        m_pListenSocket = new CListeningSocket(this);
        m_pListenSocket->Create(port);
//      bListening = TRUE;
        return m_pListenSocket->Listen();
//              ((CEditView*)m_viewList.GetHead())->SetWindowText(_T("Begin
listening ..."));
}
```

```
void CChatServer::ProcessClientPkg(CClientSocket *pSocket)
{
	CPkg pkg;
	ASSERT(pSocket);

	try{

		// the next line of code was first not included in the bListening == FALSE
block
		// and caused some error: for that part the Archive buffer was just full.
		// Now I moved it outside of the if-else block so that it applys both parts.
		pSocket->ReceivePkg(&pkg);
		if(bListening){
			if(pkg.request == NEW){
				if(IsThisNameBeingUsed(pkg,pSocket)){
					Server2Indivisual(pSocket,"Sorry. The screen name
is being used. Try using another one.Thank you.",SIGNOUT);
					return;
				}

				UpdateClientInfo(&pkg,pSocket);
				CString strReminder;
				strReminder.Format("%s  just signed in.", pkg.strName);

				UpdateListCtrl();
				UpdateInfo2All(strReminder,NEW);

				// log message sent to the log page
				CTime time= CTime::GetCurrentTime();;
				CString strTmp = time.Format(" Time is:  %H:%M:%S.
Date is: %A, %B %d, %Y");
				strReminder += strTmp;
				m_pMainSheet->SetLogMessage(strReminder);

			}
			if(pkg.request == OFF){
				CString strReminder;
				strReminder.Format("%s  just logged off.", pkg.strName);

				DeleteIndivisual(pkg.strName);
				UpdateListCtrl();
				UpdateInfo2All(strReminder,OFF);

				// log message sent to the log page
				CTime time= CTime::GetCurrentTime();;
```

```
                    CString strTmp = time.Format(" Time is:  %H:%M:%S.
Date is: %A, %B %d, %Y");
                    strReminder += strTmp;
                    m_pMainSheet->SetLogMessage(strReminder);
                }
                if(pkg.request == MESG ){
                    UpdateClientInfo(&pkg,pSocket);
                    Send2AllClients(&pkg);
                }
            }
        }
        catch(CFileException* e){
            TCHAR szCause[255];
            e->GetErrorMessage(szCause,255);
            AfxMessageBox(szCause);
        }
}

void CChatServer::Send2AllClients(CPkg *pkg)
{
        for(POSITION pos = m_ContactList.GetHeadPosition(); pos != NULL;){
            CPkg* pClient = (CPkg*)m_ContactList.GetNext(pos);
            ASSERT(pClient);
            CClientSocket* pSocket = (CClientSocket*)pClient->pSocket;
            ASSERT(pSocket);
            try{
                pSocket->SendPkg(pkg);
            }
            catch(CFileException* e){
                TCHAR szCause[255];
                e->GetErrorMessage(szCause,255);
                AfxMessageBox(szCause);
            }
        }
}

void CChatServer::CloseAllClients()
{
        for(POSITION pos = m_ContactList.GetHeadPosition();pos != NULL;){
            CPkg* pClient = (CPkg*)m_ContactList.GetNext(pos);
            ASSERT(pClient);
            CClientSocket* pSocket = (CClientSocket*)pClient->pSocket;
            pSocket->Close();
            if(pSocket != NULL) {delete pSocket;pSocket = NULL;}
            //              pSocket->Finalize();
```

```
            delete pClient;
            pClient = NULL;
        }
        m_ContactList.RemoveAll();
}

void CChatServer::CloseListening()
{
        if(m_pListenSocket == NULL) return ;

        //close the linked client and clear up the contact list
        CloseAllClients();
        m_pListenSocket->Close();
        delete m_pListenSocket;
        m_pListenSocket = NULL;
        bListening = FALSE;
}


//when got a connection message from a client
void CChatServer::ProcessAccept()
{
        CClientSocket* pSocket = new CClientSocket(this);
        m_pListenSocket->Accept(*pSocket);
        pSocket->Init();

        CString ipAddress;
        UINT port;
        pSocket->GetPeerName(ipAddress,port);



        //send the connection confirmation to the client,add to the list - yh
        CPkg pkg;
        pkg.Init();
        pkg.strName = _T("Server");
        pkg.strText = "Connected. Please Sign in now ...";
        pkg.clrText = RGB(0,0,0);
        pkg.bBold = 1;//bold
        pkg.fontName = "system";
        pkg.fontSize = 12;
        pkg.clrText = RGB(0,0,255);//blue
        pkg.request = CONN;

        pSocket->SendPkg(&pkg);
```

```
        CPkg* pTmp;
        pTmp = new CPkg;
        ASSERT(pTmp);

        pTmp->Init();
        pTmp->ipAddress = ipAddress;
        pTmp->port = port;
        pTmp->pSocket = (DWORD)pSocket;

        if(m_ContactList.IsEmpty()){
                m_ContactList.AddHead(pTmp);
        }
        else{
                m_ContactList.AddTail(pTmp);
        }
}
void CChatServer::Server2Individual(CString name, CString strMessage,int request)
{
        for(POSITION pos = m_ContactList.GetHeadPosition(); pos != NULL;){
                CPkg* pClient = (CPkg*)m_ContactList.GetNext(pos);
                ASSERT(pClient);
                if(pClient->strName != name) continue;
                try{
                        CClientSocket* pSocket = (CClientSocket*)pClient->pSocket;
                        ASSERT(pSocket);

                        CPkg pkg;
                        pkg.Init();
                        pkg.strName = _T("Server");
                        pkg.strText = strMessage;
                        pkg.clrText = RGB(0,0,0);
                        pkg.bBold = 1;//bold
                        pkg.clrText = RGB(0,0,255);//blue
                        pkg.fontName = "system";
                        pkg.fontSize = 12;
                        pkg.request = request;
                        pSocket->SendPkg(&pkg);

                }
                catch(CFileException* e){
                        TCHAR szCause[255];
                        e->GetErrorMessage(szCause,255);
                        AfxMessageBox(szCause);
                }
        }
```

```
}

void CChatServer::Server2Individual(CClientSocket* pSocket, CString strMessage,int
request)
{
	for(POSITION pos = m_ContactList.GetHeadPosition(); pos != NULL;){
		CPkg* pClient = (CPkg*)m_ContactList.GetNext(pos);
		ASSERT(pClient);
		if((CClientSocket*)pClient->pSocket != pSocket) continue;
		try{
			CClientSocket* pSocket = (CClientSocket*)pClient->pSocket;
			ASSERT(pSocket);

			CPkg pkg;
			pkg.Init();
			pkg.strName = _T("Server");
			pkg.strText = strMessage;
			pkg.clrText = RGB(0,0,0);
			pkg.bBold = 1;//bold
			pkg.fontSize = 12;
			pkg.clrText = RGB(0,0,255);//blue
			pkg.fontName = "system";
			pkg.request = request;
			pSocket->SendPkg(&pkg);

		}
		catch(CFileException* e){
			TCHAR szCause[255];
			e->GetErrorMessage(szCause,255);
			AfxMessageBox(szCause);
		}
	}
}

void CChatServer::UpdateInfo2All(CString msg,int request)
{
	CPkg pkg;
	CString nameString = "";
	for(POSITION pos = m_ContactList.GetHeadPosition(); pos != NULL;){

		CPkg* pClient = (CPkg*)m_ContactList.GetNext(pos);
		nameString += pClient->strName;
		nameString += ";";
	}
	for(pos = m_ContactList.GetHeadPosition(); pos != NULL;){
```

```
                CPkg* pClient = (CPkg*)m_ContactList.GetNext(pos);
                pkg.bBold = 1;//bold
                pkg.clrText = RGB(0,0,255);//blue
                pkg.strName = nameString;
                pkg.strText = msg;
                pkg.request = request;

                try{
                        CClientSocket* pSocket = (CClientSocket*)pClient->pSocket;
                        ASSERT(pSocket);
                        pSocket->SendPkg(&pkg);

                }
                catch(CFileException* e){
                        TCHAR szCause[255];
                        e->GetErrorMessage(szCause,255);
                        AfxMessageBox(szCause);
                }
        }
        nameString = "";
}

void CChatServer::UpdateClientInfo(CPkg* pPkg,CClientSocket* pSocket)
{
        ASSERT(pSocket);
        ASSERT(pPkg);
//      CString name = pPkg->strName;

        for(POSITION pos = m_ContactList.GetHeadPosition();pos != NULL;){
                CPkg* pTmp = (CPkg*)m_ContactList.GetNext(pos);
                if(pSocket == (CClientSocket*)pTmp->pSocket){

                        pTmp -> strText         = pPkg->strText;        //update the
pkg info from client
                        pTmp -> strName         = pPkg->strName;
                        pTmp -> bAway           = pPkg->bAway;
                        pTmp -> iMyIcon         = pPkg->iMyIcon;
                        pTmp -> bBold           = pPkg->bBold ;
                        pTmp -> bItalic         = pPkg-> bItalic;
                        pTmp -> bUnderLine  = pPkg->bUnderLine;
                        pTmp -> bStrikeOut  = pPkg->bStrikeOut;
                        pTmp -> clrText         = pPkg->clrText;
                        pTmp -> fontName    = pPkg->fontName;
                        pTmp -> fontSize    = pPkg->fontSize;
```

```
//                      pTmp -> ipAddress    = pPkg->ipAddress;
//                      pTmp -> port         = pPkg->port;
//                      pTmp -> request              = pPkg->request;
//                      pTmp -> pSocket              = pPkg->pSocket;
                        pPkg->ipAddress = pTmp->ipAddress;
                        pPkg->port = pTmp->port;
                        pPkg->pSocket = pTmp->pSocket;

                        return ; //successful in update
                }
        }

}

void CChatServer::DeleteIndivisual(CString name)
{
        for(POSITION pos = m_ContactList.GetHeadPosition();pos != NULL;){
                CPkg* pClient = (CPkg*)m_ContactList.GetAt(pos);
                ASSERT(pClient);
                if(pClient->strName == name){
                        m_ContactList.RemoveAt(pos);
                        CClientSocket* pSocket = (CClientSocket*)pClient->pSocket;
                        pSocket->Close();
                        delete pSocket;
                        pSocket = NULL;
                        delete pClient;
                        pClient = NULL;
                        break;
                }
                else{
                        m_ContactList.GetNext(pos);
                }
        }
}

void CChatServer::UpdateListCtrl()
{
        m_ClientListCtrl.DeleteAllItems();
        m_ClientListCtrl.SetImageList(&m_imgComputer,LVSIL_SMALL);
        LV_ITEM lvItem;

        int iActualItem,iItem,iSub;

        int length = m_ContactList.GetCount();
        for(iItem=0;iItem<length;iItem++){
```

```
                CPkg* pTmp =
(CPkg*)m_ContactList.GetAt(m_ContactList.FindIndex(iItem));
                LPTSTR ipAddress = (pTmp->ipAddress).GetBuffer(pTmp-
>ipAddress.GetLength());
                CString tmp;
                tmp.Format("%d",pTmp->port);
                LPTSTR port = tmp.GetBuffer(tmp.GetLength());

                for(iSub = 0;iSub<3;iSub++){
                        lvItem.mask = LVIF_TEXT|(iSub == 0? LVIF_IMAGE:0);
                        lvItem.iItem = (iSub == 0)? iItem : iActualItem;
                        lvItem.iSubItem  = iSub;
                        lvItem.iImage = 0;
                        lvItem.pszText = "";   //even you don't need this member,still
declare it!
                        if(iSub == 0)
                         iActualItem = m_ClientListCtrl.InsertItem(&lvItem);
                        else{
                                if(iSub == 1)
                                        lvItem.pszText  = ipAddress;
                                else if(iSub == 2)
                                        lvItem.pszText = port;
                                m_ClientListCtrl.SetItem(&lvItem);
                        }
                }
        }
}

CString CChatServer::GetServerIP()
{
        char szHostName[200];
        gethostname(szHostName,strlen(szHostName));
        LPHOSTENT pHost;
        pHost = gethostbyname(szHostName);
        struct in_addr* ptr = (struct in_addr*)pHost->h_addr_list[0];
        int a = ptr->S_un.S_un_b.s_b1;
        int b = ptr->S_un.S_un_b.s_b2;
        int c = ptr->S_un.S_un_b.s_b3;
        int d = ptr->S_un.S_un_b.s_b4;

        CString strTmp;
        strTmp.Format("%d.%d.%d.%d",a,b,c,d);
        return strTmp;

}
```