

DEVELOPMENT OF MULTITHREADED REAL TIME DATA ACQUISITION  
SOLUTIONS

by

Gajendra Singh

A thesis submitted to the faculty of  
The University of North Carolina at Charlotte  
in partial fulfillment of the requirements  
for the degree of Master of Science in the  
Department of Electrical and Computer Engineering

Charlotte

2006

Approved by:

---

Dr. James M. Conrad

---

Dr. Ivan L. Howitt

---

Dr. Linda J. Xie



## ABSTRACT

GAJENDRA SINGH. Development of Multithreaded Real Time Data Acquisition Solutions. (Under the direction of DR. JAMES M. CONRAD).

The purpose of a data acquisition system is to provide reliable and timely information. The received information can be then analyzed in real-time or remotely to estimate the state of the measured environment. This thesis and work are an effort toward the development of communication Application Programming Interfaces (APIs). The developed APIs can be used to provide reliable communication between a host PC and an embedded target.

This work is justified by the increasing need of host side software, which can be used to communicate with an embedded target using a variety of communication interfaces. A multithreaded application has been developed to address the implementation of APIs. This application monitors and logs the sensors' data on the host PC. A way to monitor and display several sensors' data received on a single communication line is proposed in the application. While developing the APIs, an effort was made towards data abstraction. An object oriented approach has been utilized for all the communication interfaces. The protocols tested for communication are RS-232, USB and Ethernet. The interface implementation is used to build communication classes for the classified protocols. The classes constructed for interfacing can be easily integrated with any application software. Using existing work, a developer will have to do a thorough investigation of all protocols to achieve reliable data acquisition. This work provides a simplified API solution for three major communication protocols in an easy-to-use toolkit which can be used by developers with reasonable software design skills.

## ACKNOWLEDGEMENTS

I would like to thank my graduate adviser Dr. James M. Conrad for his extraordinary support and understanding in guiding me through this thesis successfully. I would also like to thank Dr. Ivan Howitt and Dr. Linda Xie for providing valuable support.

I would like to express my special thanks to Sandeep Sirpatil, Gurudatt Mysore and Sonia Thakur for providing me valuable help to accomplish my thesis implementation. I also want to express my appreciation to developers who post valuable information and their experience on open source website [www.codeproject.com](http://www.codeproject.com).

## TABLE OF CONTENTS

LIST OF FIGURES .....	x
LIST OF TABLES .....	xii
CHAPTER 1: INTRODUCTION .....	1
1.1 Motivation.....	2
1.2 Previous Work .....	3
1.3 Description of the proposed thesis work.....	5
1.4 Organization of Thesis.....	5
CHAPTER 2: COMMUNICATION INTERFACES .....	7
2.1 Overview of RS-232 .....	7
2.1.1 D9-Connector Pins .....	9
2.1.2 Medium Access.....	10
2.1.3 RS-232 Limitations .....	11
2.1.4 Programming a Serial Connection .....	11
2.1.4.1 Opening a Port .....	12
2.1.4.2 Configuring a Serial Port .....	12
2.1.4.3 Configuring Time-outs.....	12
2.1.4.4 Writing to a Serial Port .....	12
2.1.4.5 Reading from a Serial Port.....	13
2.1.4.6 Closing a Serial Port .....	13
2.2 Overview of USB.....	13
2.2.1 USB Signals and Packets .....	15
2.2.2 USB Transfers.....	17

2.2.3 Enumeration .....	18
2.2.4 Writing PC Software .....	20
2.3 Overview of Ethernet .....	22
2.3.1 Ethernet Frame .....	23
2.3.2 Medium Access Control .....	25
2.3.3 Programming for Network Communications .....	25
2.3.3.1 Sockets, Ports and Addresses .....	26
2.3.3.2 Creating a Socket .....	26
2.3.3.3 Making a Connection .....	26
2.3.3.4 Sending and Receiving Messages .....	27
2.3.3.5 Closing the Connection .....	27
2.3.3.6 Detecting Errors .....	27
2.4 Comparison of Interfaces .....	27
CHAPTER 3: DEVELOPMENT AND TESTING .....	29
3.1 Development Phase .....	29
3.1.1 RS-232 Interface Specifications .....	28
3.1.2 RS-232 Interface Development .....	29
3.1.2.1 Opening Serial Port .....	30
3.1.2.2 Configuring Serial Port .....	30
3.1.2.3 Setting Timeouts .....	31
3.1.2.4 Reading and Writing to the Serial Port .....	32
3.1.2.5 Closing Serial Port .....	32
3.1.3 USB Interface Specifications .....	32

3.1.4 USB Interface Development .....	32
3.1.5 Ethernet Interface Specifications .....	34
3.1.6 Ethernet Interface Development .....	34
3.1.6.1 Accept Function .....	35
3.1.6.2 Connect Function .....	35
3.1.6.3 Send and Receive Function.....	35
3.1.6.4 Close Function .....	36
3.2 Testing Phase .....	36
3.2.1 RS-232 Interface Testing .....	36
3.2.1.1 System Description .....	37
3.2.1.2 Clam Sensor Bay.....	38
3.2.1.3 YSI Sensor Bay .....	38
3.2.1.4 Motor Controller .....	39
3.2.1.5 Single Board Computer.....	40
3.2.1.6 Complete Hardware .....	41
3.2.1.7 Graphical User Interface .....	42
3.2.1.8 Window Design of GUI.....	42
3.2.1.9 System Requirements.....	44
3.2.1.9.1 Clam Sensor Board Requirements .....	44
3.2.1.9.2 YSI Sensor Board Requirements .....	46
3.2.1.9.3 Motor Controller Board Requirements .....	47
3.2.1.9.4 Persistor Requirements .....	48
3.2.1.9.5 Graphical User Interface Requirements.....	48

3.2.1.10 Software Design of GUI .....	49
3.2.3 USB Interface Testing.....	50
3.2.3 Ethernet Interface Testing.....	52
3.3 Multitasking .....	52
3.3.1 Idle Process Thread.....	53
3.3.2 Independent Threads .....	53
3.3.3 Inter-tasks Communication .....	53
3.3.4 Building a Multitasking Application .....	54
CHAPTER 4: DEVELOPMENT TOOLS.....	55
4.1 Introduction to Windows Programming .....	55
4.2 Integrated Development Environment.....	56
4.2.1 The Editor .....	56
4.2.2 The Compiler .....	56
4.2.3 The Linker.....	57
4.2.4 The Libraries .....	57
4.3 Using Visual Studio IDE .....	58
4.3.1 Creating Project Workspace.....	59
4.3.2 Building Project .....	63
4.3.3 Debugging.....	65
CHAPTER 5: FUTURE DEVELOPMENT .....	67
5.1 Conclusion .....	67
5.2 Future Work.....	68
5.2.1 API and SDK supported by WinCE.....	68



5.2.2 Why WinCE? .....	69
REFERENCES .....	70
APPENDIX .....	73

## LIST OF FIGURES

FIGURE 2.1 RS-232 voltage levels.....	8
FIGURE 2.2 D9 pinouts [26].....	10
FIGURE 2.3 Serial logic waveform [26].....	10
FIGURE 2.4 Comparison of USB, RS-232 serial & parallel connectors [16].....	15
FIGURE 2.5 Enumeration process [9].....	19
FIGURE 2.6 Re-enumeration [9].....	19
FIGURE 2.7 Layers of PC-host software [16].....	21
FIGURE 2.8 Network protocol stack [3].....	23
FIGURE 2.9 Basic socket connection [6].....	26
FIGURE 3.1 Data logging system .....	37
FIGURE 3.2 Data acquisition system.....	37
FIGURE 3.3 Clam sensor board .....	38
FIGURE 3.4 YSI sensor board .....	39
FIGURE 3.5 Motor controller board .....	40
FIGURE 3.6 Single board computer.....	41
FIGURE 3.7 Complete hardware system.....	41
FIGURE 3.8.a Tab one displaying biological data .....	43
FIGURE 3.8.b Tab two displaying environmental data .....	44
FIGURE 3.9 Software design flowchart.....	50
FIGURE 3.10 USB data acquisition hardware setup.....	51
FIGURE 3.11 USB data acquisition on host-PC .....	52
FIGURE 4.1 Windows program structure [14].....	56

FIGURE 4.2 VC++ inactive IDE.....	58
FIGURE 4.3 Opening a new project using MFC AppWizard.....	59
FIGURE 4.3.a Step-1 of creating a project framework .....	60
FIGURE 4.3.b Step-2 of creating a project framework .....	60
FIGURE 4.3.c Step-3 of creating a project framework .....	61
FIGURE 4.3.d Step-4 of creating a project framework .....	61
FIGURE 4.3.e Project information window .....	62
FIGURE 4.4 Set active configurations .....	62
FIGURE 4.5 Active project workspace .....	63
FIGURE 4.6 Files created after building the project .....	64
FIGURE 4.7 Starting debugger in VC++ .....	66
FIGURE 4.8 Using debugger in VC++ .....	66

## LIST OF TABLES

TABLE 2.1 D9 pins description [4].....	9
TABLE 2.2 RS-232 cable length [26] .....	11
TABLE 2.3 Relative performances of USB versions [16] .....	14
TABLE 2.4 USB packet types [16] .....	16
TABLE 2.5 USB transfers [16] .....	18
TABLE 2.6 Descriptors [9] .....	20
TABLE 2.7 Ethernet frame [3] .....	24
TABLE 2.8 Comparison Chart [3] .....	28
TABLE 3.1 Clam sensor data format .....	45
TABLE 3.2 YSI data analysis.....	47
TABLE 4.1 Description of files created after building the project [14].....	64

## LIST OF ABBREVIATIONS

ADC	Analog to Digital Converter
API	Application Program Interface
ATL	Active Template Libraries
CR2032	3V Lithium Coin Cell
CSMA-CA	Carrier Sense Multiple Access with Collision Avoidance
DCB	Data Control Block
DTE	Data Terminal Equipment
GUI	Graphical User Interface
IDE	Integrated Development Environment
IFG	Inter Frame Gap
ISDN	Integrated Services Digital Network
MAC	Medium Access Control
MFC	Microsoft Foundation Classes
MSDN	Microsoft Developer Network
PID	Packet Identifier
SDK	Software Development Kit
TCP/IP	Transmission Control Protocol /Internet protocol
USB	Universal Serial Bus
WinCE	Windows Compact Edition

## CHAPTER 1: INTRODUCTION

Most of the proposed research projects analyze data to answer an anticipated research problem. It is very important to collect correct data in order to successfully address the objective of the research. Recent advent of single board computers has redefined the data acquisition methodologies. Almost every single board computer supports more than one communication interface. This gives flexibility to choose a specific communication interface suitable for data acquisition systems. The large amount of data collected by a single board computer can not be analyzed on a real-time basis because of the memory constraints and the absence of a user interface. When data is sent to a host PC or a handheld device, then it can be analyzed in real-time or be stored on hard disk for remote analysis. The proposed work is an effort to develop a communication interface between an embedded target controllers and a host PC. This thesis work not only explores the conventional serial data acquisition protocol based on RS-232, but also gives a simplified solution to advanced data communication interfaces such as USB and Ethernet.

The main reasons why USB and Ethernet were chosen along with traditional RS-232 protocol are:

- They are versatile. An Ethernet has ability to transfer short messages to huge files by use of the higher level TCP/IP protocol. USB has the ability to serve many devices on a single line of interface.
- They have operating system support.

- They provide a much higher data rate in comparison to RS-232.
- There is an increasing support of USB and Ethernet on embedded targets.

### **1.1 Motivation**

This work initially started with development of a RS-232 based data acquisition interface for the Durham-based Nekton Research Inc. The task of developing a RS-232 based data acquisition interface was challenging without the support of a well defined host side communication interface. As a result, the development a well defined communication toolkit was identified and efforts were made to produce and demonstrate the usefulness of the interface for application software. The success of a research objective depends on the analysis of uncorrupted data. Therefore, data acquisition becomes the utmost important issue of a research project. This work provides an easy-to-use communication interface toolkit for developers to carry out effective data acquisition. The aim of the work is to provide a reliable communication between the host PC and the embedded target to ensure a successful logging of the data at host side. In order to provide abstraction, an object oriented approach has been chosen for the implementation of the communication interfaces.

A substantial amount of resources were identified to work in the direction of developing communication interfaces. RS-232 is a well supported interface and the Microsoft Developer Network (MSDN) provides the necessary information to access RS-232 on the PC host side. The work by John Hyde [16] provides an adequate foundation to understand USB interface. Microsoft Developer Network offers support for programming Ethernet interface and served as a useful resource. Three different descend classes in C++ are developed from Microsoft Foundation Classes (MFC) and member functions to

access USB, Ethernet and RS-232 ports are added to them.. These classes can be integrated with any user interface application and can serve as an added capability to an application software used to access embedded targets. With the use of provided interface a developer obtains an easy-to-use communication interface solution.

## **1.2 Previous Work**

The proposed work is influenced by the continuous enduring work on improving data acquisition methodologies. The proposed work ensures to provide a toolkit that can be a valuable addition to the never ending research on improvisation of data acquisition.

Work by S. Martin proposes an alternative to single user; single tasking, simple, low cost IBM 'PC/XT/AT' family of computers. These PC/XT/AT computers were designed with office automation in mind to excel in word processing and spreadsheet applications. They were not up to the mark for real time requirements of high performance data acquisition. He has proposed to put a high performance processor right on the data acquisition board [17]. Interface developed for the anticipated work, can be an excellent addition to high performance processor in order to achieve reliable, and fast data acquisition.

Work by Payne and Menz have listed all the important goals of a guaranteed high speed data acquisition and illustrated a case study consistent to their work. Their work does not consider communication interfaces into account [21]. A lack of sophisticated communication interface can be a major limiting factor in achieving real time requirements. The anticipated work for thesis can add to the case study discussed in the work done by Payne and Menz and can deal with communication issues.



Postolache, Pereira and Girao have proposed a sensing unit as a part of PC based water quality monitoring system [22]. It is almost similar to projected work in this thesis. The main difference between two works is that the latter one provides a flexible set of communication interfaces which can be utilized to deal with soft as well as hard real time requirements. The former work just discusses the RS-232 based system and does not consider higher data rate acquisition interfaces.

Nigus and Dyer have proposed the design of a system which is host independent [20]. Their work can be utilized by any computer, provided it has a RS-232 port. The thesis work adds to the flexibility of their design by introducing two more interfaces to system: USB and Ethernet.

The article by Bruce discusses USB protocol for accessing data from embedded targets and served as an essential resource in developing a USB based data acquisition system [23].

The article by B. Zdanivsky discusses to create an Internet browser based acquisition system. In addition to generating an Internet feed, the system collects heart rate, blood pressure, and temperature data [30]. The proposed system three interfaces: RS-232, Ethernet, and USB to communicate between different modules of the system. A communication toolkit, similar to one developed for projected thesis work can save a lot of time in developing such system.

A careful study of all the previous work done, gives adequate motivation to work on managing a real time data acquisition in a better way.

### **1.3 Description of Proposed Thesis Work**

The main goal of the proposed thesis work is to provide an easy-to use toolkit. This toolkit can be used to provide an efficient communication interface. An effort to improve data acquisition has been made by providing a reliable communication interface. An Application Programming Interface has been developed and tested for three major communication interfaces: RS-232, USB, and Ethernet.

In order to address the objective of the proposed thesis work, a RS-232 communication interface class was written. An application software was developed to test the RS-232 interface class. A hardware system was configured to collect data from 23 different sensors. The data collected was sent to PC-host on a single RS-232 line. A user interface application running on PC-host monitored the data received on the serial port. The data received is not only displayed in real-time but also logged on a file for remote analysis. The successful data acquisition by the application tested the efficiency of developed RS-232 interface and further motivated to extend the work. In order to complete the development of toolkit, classes for USB and Ethernet interface was implemented and tested.

### **1.4 Organization of Thesis**

This thesis is divided into five major chapters. Chapter Two of the thesis report discusses communication interfaces developed and tested for the toolkit. It also explains the Medium Access Control mechanism of RS-232, USB and Ethernet protocols and also illustrates a comparison between them. Chapter Three describes the development and testing of Interfaces. It also discusses the demonstrated hardware used for interfacing sensors and sending data to host machine via different interfaces.

Chapter Four describes the development tools used to create, compile and debug interface and application software. Chapter Five summarizes the results and suggests future enhancements.

## CHAPTER 2: COMMUNICATION INTERFACES

As a part of data acquisition, there are many choices of interfaces available. Deciding on an interface suitable for an application depends on a lot of factors. Choosing a particular host-target interface would depend on:

- Support of interfaces on both PC-host and target, so that writing low level drivers can be avoided.
- Fast, so that the interface can deal real-time data acquisition.
- Reliable, so that validity of data can be provided.
- Inexpensive solution

Some times more than one interface is essential for an application to meet the requirements. Designing a GUI supporting a specific communication interface calls for a lot of learning of that particular interface. To use a particular communication interface efficiently, it has to be configured properly. In this section of proposed work, three communication interfaces are selected and medium access control mechanism of these interfaces is explained.

### 2.1 Overview of RS-232

RS-232 is very well known single-ended serial communication protocol. It is intended to support efficient data communication at low baud rates (<20kbps). The RS-232 signals are represented by voltage levels with respect to common ground. Signal definitions in RS232 standard are:

- Ground
- Primary communication channel for data exchange and flow control.
- Secondary communication channel for controlling remote modem and for handshaking.

A signal between -3v to -25v signifies logic '1' while a signal between +3v to +25v signifies logic '0'. Thus a serial communication in RS232 has a 50v voltage swing. The region between -3v to +3v is not defined for RS232 standard.

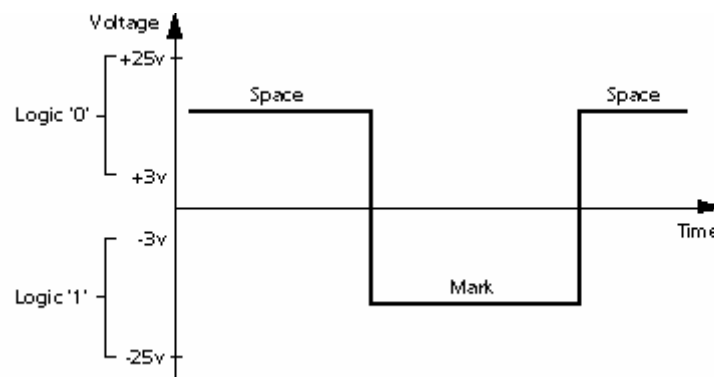


FIGURE 2.1 RS-232 voltage levels

Ports used for RS232 interface are of two types. There are D-type 25-pin connectors and D-type 9-pin connectors. D-type 25 pin connectors are obsolete but d9 are commonly used. Connecting just primary channel of these connectors make them a Null Modem. A Null Modem is used to connect two DTE's (Data Terminal Equipment) together. This is commonly used method in communicating an embedded device with a PC. This involves just a three wire (Transmit, Receive, and Ground) connection. The grounds of both terminals are supposed to be common. Programming a serial port is fairly easy. Exchanging data by way of RS-232 port is similar to reading from or writing to a file

[19]. Windows support RS-232 serial communications drivers. The only task which a developer has to do is to develop an interface to access the port to exchange data.

### 2.1.1 D9-Connector Pins

Table 2.1 and Figure 2.1 shows pin connections for a 9 Pin D-type connector [4].

TABLE 2.1 D9 pin description [4]

D9 Connector	Abbreviation	Full name
Pin 3	TD	Transmit Data
Pin 2	RD	Receive Data
Pin 7	RTS	Request To Send
Pin 8	CTS	Clear To Send
Pin 6	DSR	Data Set Ready
Pin 5	SG	Signal Ground
Pin 1	CD	Carrier Detect
Pin 4	DTR	Data Terminal Ready
Pin 9	RI	Ring Indicator

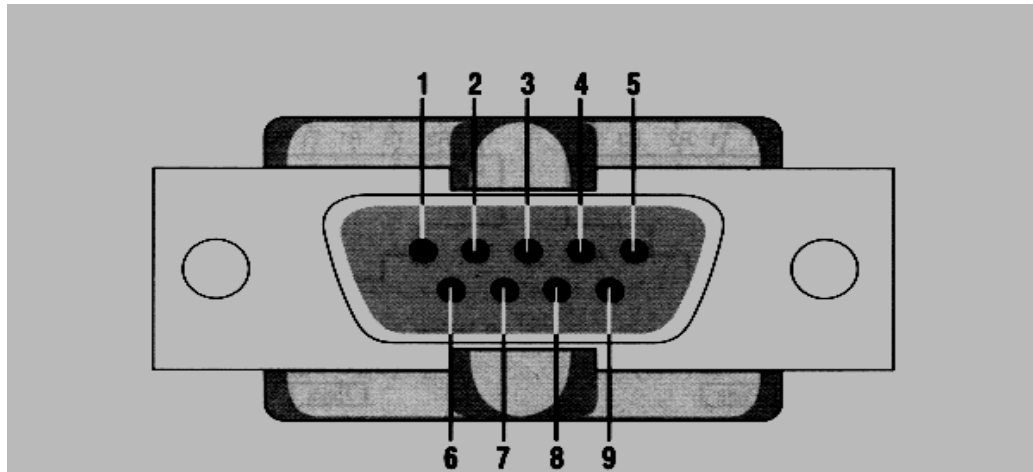


FIGURE 2.2 D9 pinouts [26]

### 2.1.2 Medium Access

RS-232 communication is asynchronous which means no clock signal is sent with data. Data is synchronized by use of a transmitted start bit and a receiver side clock which keeps tab on timing [4]. A transition from high (logic '1') to low (logic '0') signifies a start bit. After the start bit the data bits are sent one at a time until a stop bit is received. A transition from high to low signifies a stop bit. For validating data, parity is often used. A common waveform from the port is shown in Figure 2.3.

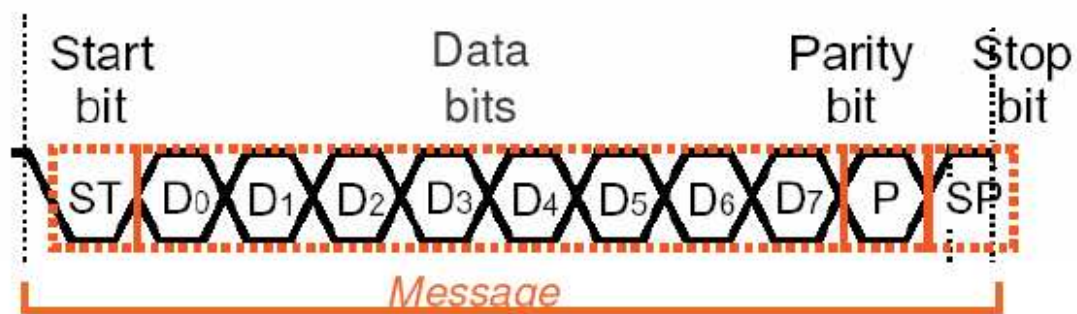


FIGURE 2.3 Serial logic waveform [26]

### 2.1.3 RS-232 Limitations

The cable length is the major issue regarding RS-232 communications. The maximum length which can be used for efficient data communications depends on some factors. Using a cable with low capacitance allows you to span longer distances without going beyond the limitations of the standard. The Baud rate required also plays a role in determining cable length. If the transmission speed is reduced by a factor two or four, the maximum length increases dramatically as shown in Table 2.2 [26]. Other minor difficulties with RS-232 interface can be:

- Misconnection of handshaking signals, resulting in buffer overflow.
- Incorrect pin configuration preventing connectors to mate properly.

TABLE 2.2 RS-232 cable lengths [26]

Baud Rate	Cable Length in feet
19200	50
9600	500
4800	1000
2400	3000

### 2.1.4 Programming a Serial Connection

Programming a serial connection involve following functions:

- Opening a port
- Configuring a Serial port
- Configuring time-outs



- Writing to a serial port
- Reading from a serial port
- Closing serial port

#### **2.1.4.1 Opening a port**

The CreateFile function is called to open a serial port. CreateFile opens the COM port and returns a handle that can be used to access an object. Port name is given while using CreateFile function. If port does not exist then CreateFile returns 'ERROR\_FILE\_NOT\_FOUND' to notify user [19].

#### **2.1.4.2 Configuring a Serial Port**

This is the most critical phase and it involves configuring the port setting with the DCB structure. It is very common to erroneously initialize the DCB structure. A call to CreateFile opens a port with default settings. To configure port according to application requirement, SetCommState function is called [19].

#### **2.1.4.3 Configuring Time-Outs**

A timeout function is very important, in case there is no activity on a port. The COMMTIMEOUTS structure can be used to configure timeouts. If this structure is not configured, the port uses default time-outs supplied by the driver, or time-outs from a previous communication application [19].

#### **2.1.4.4 Writing to a Serial Port**

Once a serial port is opened and configured, the WriteFile Function is called, which helps transfer data through serial the connection to another device. WriteFile basically writes data to a file at a position indicated by the file pointer. Once data is been written to the file, the file pointer value is adjusted to point a new location on file [19].

#### **2.1.4.5 Reading from a Serial Port**

The Readfile function is used to receive data from a device at other end of the transmission. It is similar to WriteFile as it takes the same parameter and it reads data from a file from a location indicated by a file pointer. Once reading is been done, the pointer is adjusted by number of bytes read [19].

#### **2.1.4.6 Closing a Serial Port**

The CloseHandle function is used to close a serial port. This function basically closes an open object handle. There is a two seconds delay after CloseHandle is called before the port is closed and the resource is freed. This delay helps in completing pending operations [19].

### **2.2 Overview of USB**

The major limitation with the legacy PC environment is a limited number of peripheral devices that can be attached to standard connectors. The serial connector described in the last section also supports just a single device. The speed of the interface is another issue taken into consideration with the legacy PC environment. The emergence of USB created a method of attaching and accessing a number of devices at a reduced overall cost. It also simplifies attachment and configuration from the end user perspective. The design goals of USB include [16]:

- A single connector type to connect any PC peripheral.
- Ability to attach more than one peripheral to same connector.
- A method of easing the system resource conflicts.
- Hot plug support
- Automatic detection of devices.

- Low cost solution.
- Enhanced performance capability.
- Support for legacy hardware and software.
- Low power implementation.

USB fits very well in category of an interface that enhances performance capability.

USB is available in three different versions and it supports three transmission rates as shown in Table 2.3 [16].

TABLE 2.3 Relative performances of USB versions [16]

Performance	Applications	Attributes
USB1.0:Low Speed 1.5Mbps	Keyboard, Mouse Stylus, Game Peripherals	Low cost Easy to use Multiple devices
USB1.1:Full Speed 12Mbps	ISDN, PBX, POTS, Scanner	Low cost Easy to use Multiple devices
USB2.0:High Speed 480Mbps	Video Conferencing, Imaging, Mass storage	Low cost Easy to use Multiple devices High Bandwidth Guaranteed Latency

USB uniquely provides benefits to not only the developer, but also to users using a USB interface. USB's defined cable standards and automatic error checking eases a lot of work a developer would have to do. USB has the flexibility of accessing medium for communication. It supports four different kinds of data transfers. There are transfers suited for exchanging large and small blocks of data, with and without time constraints [16]. USB is very useful in the case of meeting real-time requirements.

USB is not a simple protocol. In order to program a USB peripheral, one needs a fair knowledge of rules to exchange data on the serial line.

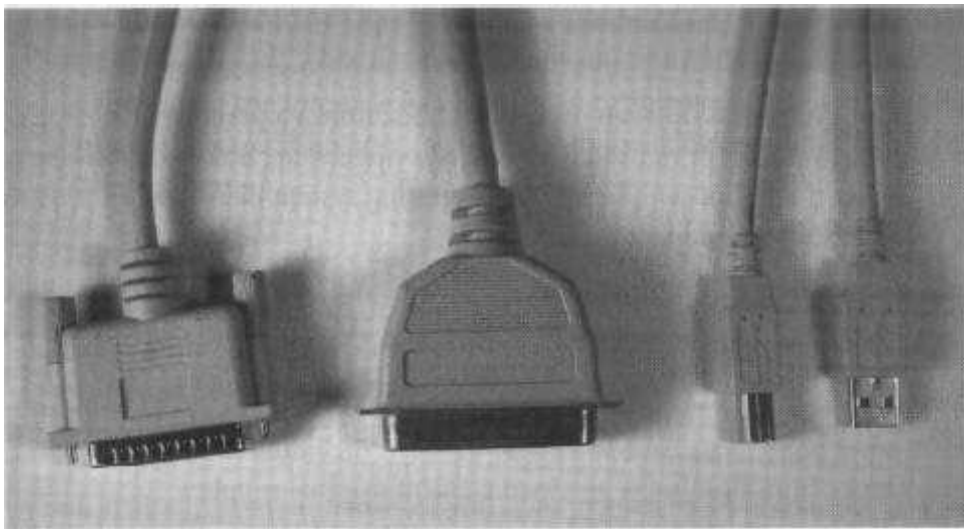


FIGURE 2.4 Comparison of USB, RS232 serial and parallel connectors [3].

### 2.2.1 USB Signals and Packets

USB uses differential signaling. There are 4 wires: two data wires D+ and D- , power and ground. The D+ and D- wires are driven at the same time and typically in anti-phase. The USB data wires do not include a clock signal, thus it is asynchronous. The base speed of the data wires are negotiated during enumeration and a SYNC signal is sent, so

that receiver can tune its bus clock to exact transitions. The fundamental element of communication on USB bus is a packet. A packet has a start, some information and an end. All packets fall in one of four categories: token packets, data packets, handshake packets, and special packets. Packet information can be as small as 1 Byte and as large as 3074 Bytes. Table 2.4 gives a brief description of packets used in USB:

TABLE 2.4 USB packet types[16]

<b>PID Value</b>	<b>Packet Type</b>	<b>Packet Category</b>
0101	SOF	Token
1101	SETUP	Token
1001	IN	Token
0001	OUT	Token
0011	DATA0	Data
1011	DATA1	Data
0111	DATA2	Data
1111	MDATA	Data
0010	ACK	Handshake
1010	NAK	Handshake
1110	STALL	Handshake
0110	NYET	Handshake
1100	PRE	Special
1100	ERR	Special
1000	SPLIT	Special
0100	PING	Special
0000	(Reserved)	(Reserved)

The first byte of every packet is a Packet Identifier (PID) that defines how other information bytes should be interpreted. A PID is formed with 4 bits and the complement of the 4 bits to provide an error check.

### **2.2.2 USB Transfers**

USB device deals with mainly two kind of communication. The first one is to detect and configure a device and the second one is to carry out actual data transfers. The first one occurs when a host enumerates the device on power up or attachment.

While configuring, a device's firmware responds to a number of requests made by the host. Host's operating system is responsible for this enumeration process. Therefore, for enumeration, a user does not have to write any code for host side; enumeration is done at kernel level.

All communication falls in one of the four transfers supported by USB. Control transfer is needed to configure a device on attachment or power up. The PC host software has two parameters to deal with- delivery time accuracy and delivery quality accuracy. The time and quality attributes have different importance for different data types. The host guarantees quality accuracy by using handshake mechanism. In general [16]:

- If data is received correctly, then an ACK handshake is generated.
- If there is a problem with the data transfer, a NAK handshake is generated.
- If the data receiver is confused, a STALL handshake is generated.

Table 2.5 gives a brief outline of transfers involved in USB interface:

TABLE 2.5 USB transfers [16].

<u>Maximum size of Frame</u>					
Type	Important Attributes	Low speed	Full Speed	High Speed	Examples
Interrupt	Quality + Time	8	64	3072	Mouse, keyboard
Bulk	Quality	-	64	512	Printer, Scanner
Isochronous	Time	-	1023	3072	Audio, Video
Control	Quality + Time	8	64	64	System Control

### 2.2.3 Enumeration

It is important to know, how a USB device describes itself to host-PC. The first step is to detect the device. This is done by detecting a rise of D+ or D- above ground. Once a device is detected, a series of steps takes place which actually enumerates the device. The PC host sends requests to the device. These transfers are called control transfers. Control transfers consist of two stages, SETUP and STATUS. The device decodes the request from the host. The firmware provides all the necessary information needed by the host to load the proper driver for the device. This identification information consists of Device Descriptors, Configuration Descriptors and Interface Descriptors. Figure 2.5 shows the enumeration process.

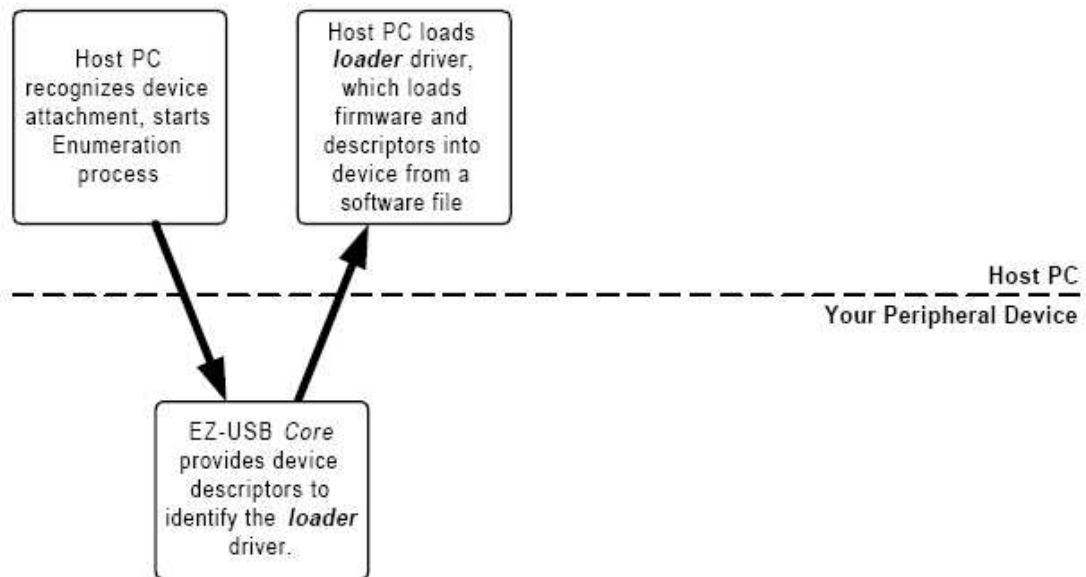


FIGURE 2.5 Enumeration process[9].

Once RAM is loaded with descriptors and code that define final device, the device needs to enumerate once again. This process is shown in Figure 2.6:

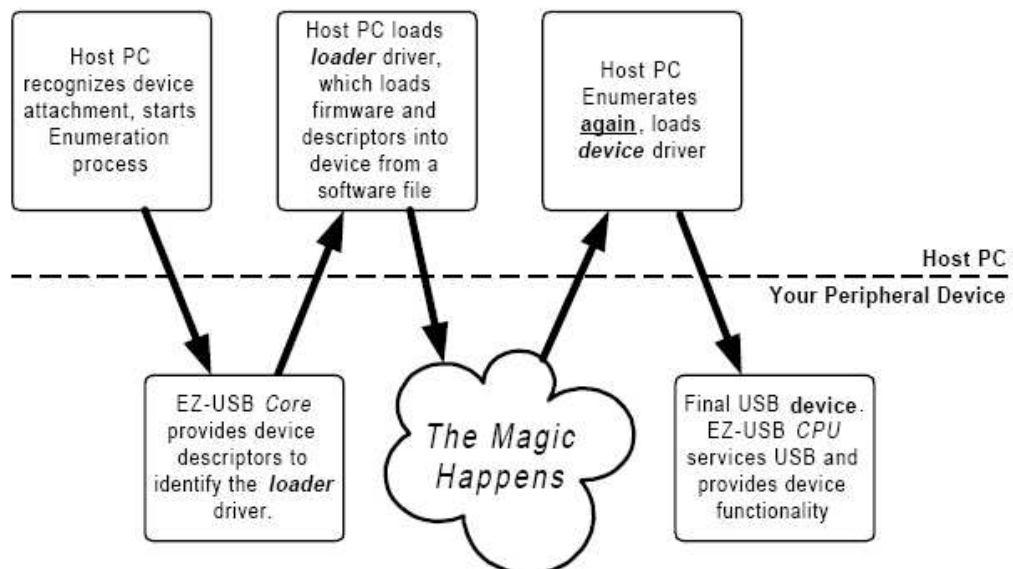


FIGURE 2.6 Re-enumeration [9]



Table 2.6 gives a brief description of the various descriptors a PC needs to identify a device:

TABLE 2.6 Descriptors [9]

<b>Descriptor Type</b>	<b>Description</b>
Device	Describes an entire device.
Configuration	Describes one of the configurations of a device.
Interface	Describes one of the interfaces that is part of configuration.
Endpoint	Describes one of the endpoints belonging to an interface.
String	Contains a human readable Unicode string describing the device, a configuration, an interface, or an endpoint.

#### 2.2.4 Writing PC Software

A PC host running USB-aware operating system software supports two distinct USB functions- initialization and runtime. The USB initialization software is active at all times and can add devices at anytime. Once a device is enumerated, it is given an identifier by the PC host. This identifier is used during run-time. Figure 2.7 shows the structure of software written for full functioning of a USB device.

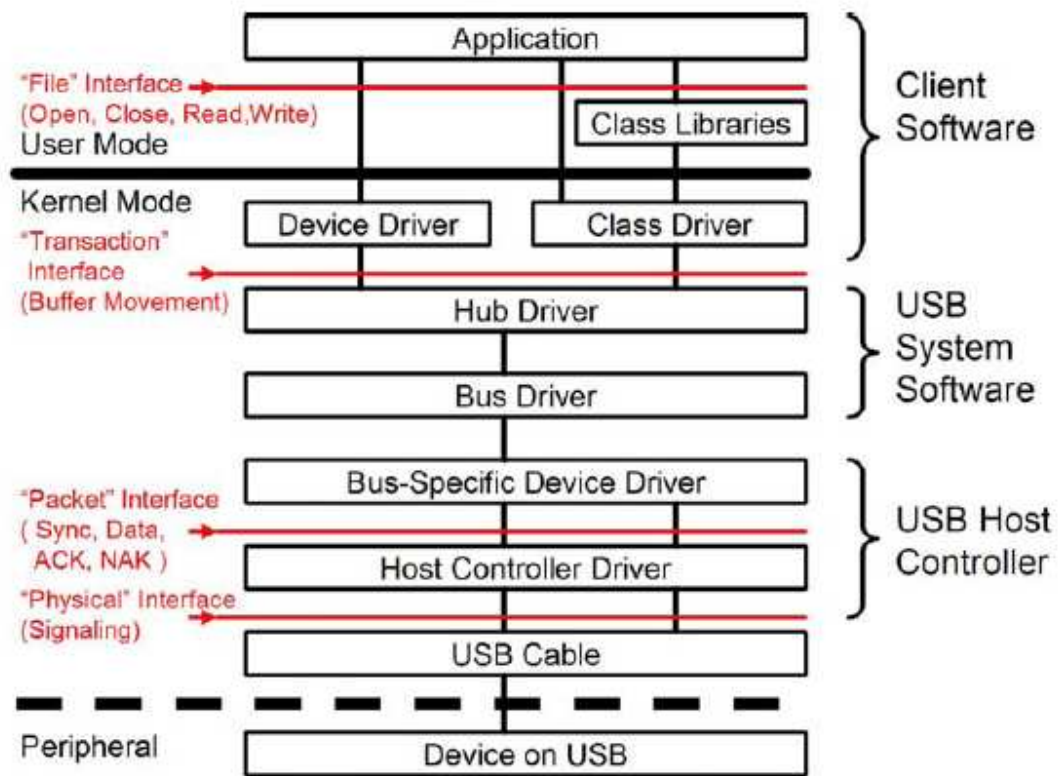


FIGURE 2.7 Layers of PC-host software [16].

Similar to serial connection programming, USB programming also involves a number of steps. Generally the kind of transfer involved in the application also distinguishes the design of the host software. A basic design template will have following parts:

- Initializing program: initializing a handle to the host controller.
- Enumerating the host controller, identifying the host controller.
- Identifying the root hub node by using DeviceIoControl system call.
- Probing root hub connections for collecting node connection data structures.
- Collecting Descriptor data.
- Interpreting the Configuration Descriptor.

### 2.3 Overview of Ethernet

Host machines can easily communicate with near by devices using interfaces like USB and RS-232, but with Ethernet a computer can communicate over greater distances. Ethernet networks are capable and flexible. A computer on a network has to agree on the following aspects of sharing the network [3]:

- Standard rules, which specify when a computer may transmit.
- Identifying a transmission's intended destination.
- Format of information send over the network.

Every computer has networking support that has a layered model, where each layer manages a portion of the job. Today growth of embedded devices supporting Ethernet interface has greatly increased [3]. They are also supposed to support a protocol stack for supporting networking. A stripped off version of TCP/IP stack is written in C and can easily be ported to resource limited embedded devices. But since not all the features can be supported by a software based TCP/IP stack, nowadays a hardware TCP/IP stack is implemented on embedded devices. Figure 2.8 shows a computer's network protocol stack:

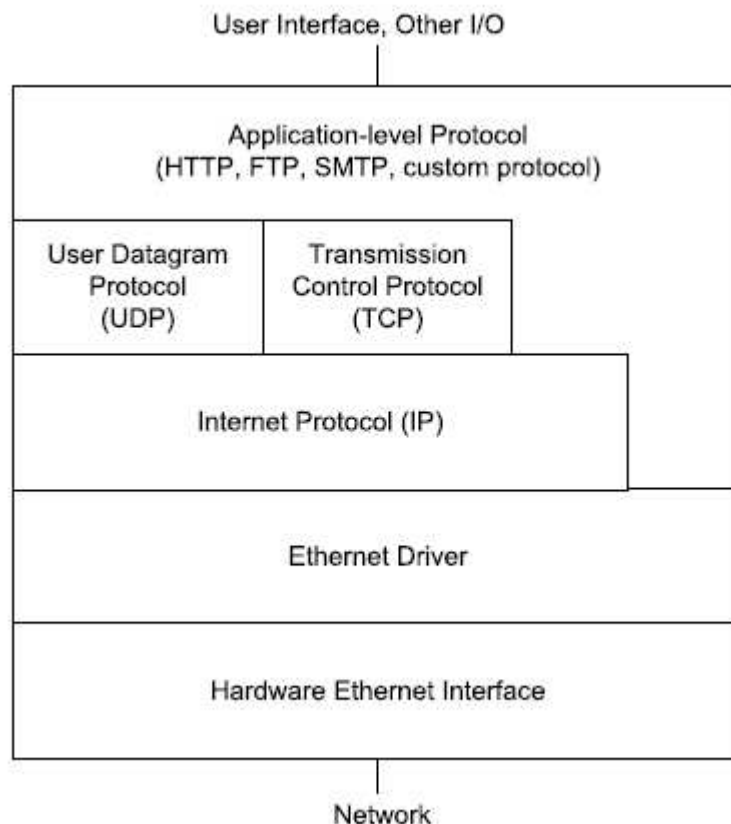


FIGURE 2.8 Network protocol stack [3].

### 2.3.1 Ethernet Frame

The Ethernet standard has been published as IEEE 802.3. All data in an Ethernet travels in structures called frames. Each frame has defined fields for data and other information to help the efficient transmission of data to its destination. A single frame has seven fields. Ethernet communication is asynchronous, which means that interfaces do not share a clock. For asynchronous RS-232 communication, a start bit and stop bit is enough to distinguish between two frames. But an Ethernet frame consists of 1000 bits. Detecting a single voltage change at the beginning of the frame isn't enough to enable the interface reliably predicting when to read all the bits that follow. So the solution to this is to start each frame with a known bit pattern that contains many transitions. This pattern is

provided by Preamble and Start of Frame Delimiter. Table 2.7 shows all seven fields of an Ethernet frame:

TABLE 2.7 Ethernet frame [3]

<b>Field</b>	<b>Length in bytes</b>	<b>Purpose</b>
Preamble	7	Synchronization pattern.
Start Frame Delimiter	1	End of synchronization pattern.
Destination Address	6	Ethernet hardware address the frame is directed to.
Source Address	6	Ethernet hardware address of the sender.
Length or Type	2	If 1500 (05DCh) or less, the length of the data field in bytes. If 1536 (0600h) or greater, the protocol used by the contents of the data field.
Data	46 to 1500	The information the source wants to send to the destination.
Frame Check Sequence	4	Error- checking value.

### **2.3.2 Medium Access Control**

A standard which allows devices on a LAN to share their interconnecting media is called medium access control. One of the ways of achieving medium access control is using a master slave concept. USB interface use master slave concept for medium access control. One another way is token passing in which, computers in the network take turns. A token can be setting a sequence of bits that indicates the possession. However Ethernet uses carrier sense multiple access with collision detection or CSMA/CD. The node, which wants to transmit must monitor the network and should transmit only if the carrier is absent. Multiple access means that no single interface controls the network traffic. Any interface that has been idle for at least the amount of time as interframe gap (IFG) can attempt to transmit. It's a job of Ethernet controller to handle sending and receiving of frames, including detecting collisions and deciding when to try again after a collision [3]. To send an Ethernet frame on network, a computer places its physical addressing the source address field and places destination's physical address in the destination address field. A physical address has two parts, a 24 bit Organizationally Unique Identifier (OUI) that identifies the interface's manufacturer and an additional 24 bits that are unique to the piece of hardware. This address is also called MAC address. It is often expressed as a series of six hexadecimal bytes.

### **2.3.3 Programming for Network Communications**

Applications requiring network capabilities are built around Winsock interface. A descended class can be created from available MFC Winsock classes. Application communicating over a network waits for another application to open a communication connection. They 'listen' for this connection request. The application, which tries to

connect to another one over the network, looks for its address and once communication is made between them, messages can be passed back and forth between them. Application use TCP/IP network protocol for communication.

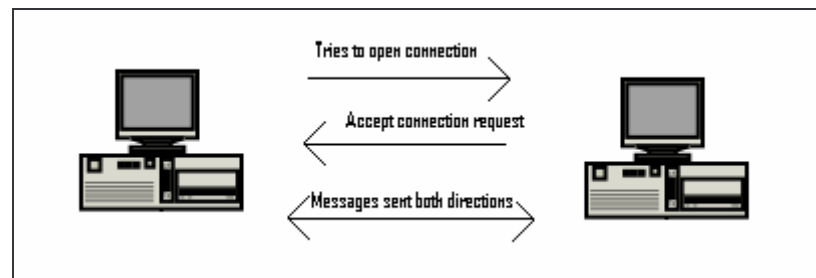


FIGURE 2.9 Basic socket connections [6]

### 2.3.3.1 Sockets, Ports and Addresses

Sockets were first developed on UNIX at UC Berkley. In order to read or write a file, a file object must be used to point to file. A socket is also an object used to read and write messages that travel between applications. To open a socket, a computer name is required on which other application is running and also the port to which that application is listening. Computer name is nothing but the address of the computer.

### 2.3.3.2 Creating a Socket

A base class `CAsyncSocket` is used to obtain a complete event driven socket communication. An object of this class can be used to call create function, which is called without any parameters, so that it can connect to another application (client) [19].

### 2.3.3.3 Making a Connection

Once a socket is been created, object of the class `CAsyncSocket` can again be called to open the connection. There are three steps associated with opening a connection. Two of them take place on the server, and third one on the client side. Once a connection is

made, an event is triggered to let application know that it is been connected or there were problems while making a connection. The server is made to listen for incoming connection. Once connection is made, server accepts the connection [19].

#### **2.3.3.4 Sending and Receiving Messages**

A pointer to a generic buffer is passed to send and receive functions. While sending, buffer should contain the data and while receiving the data incoming data is copied in the buffer. Sending data is simpler than receiving. While receiving, as soon as incoming data comes, an event is triggered [19].

#### **2.3.3.5 Closing the Connection**

Once the complete switching of data is done between a client and a server, the socket is supposed to be closed. This can be done by simply calling a close function of class CAsyncSocket [19].

#### **2.3.3.6 Detecting Errors**

Another important member function of class CAsyncSocket can be used to get the error code. This function is GetLastError ( ) and it only returns error codes [19].

### **2.4 Comparison of Interfaces**

After having a brief discussion of three serial interfaces, it is important to write down their comparative capabilities. Even though making a data acquisition system may require using all available interfaces, a comparison always helps in taking a wise decision of using an interface in specific application. The embedded targets manufactured today have multiple interfaces and making a decision to use an interface for a specific part of application can be difficult without a comparison. Table 2.8 covers important parameters considered while choosing an interface.



TABLE 2.8 Comparison chart [3]

<b>Interface</b>	<b>Format</b>	<b>Number of Devices (maximum)</b>	<b>Length (maximum feet)</b>	<b>Speed (Maximum bps)</b>
RS-232	Async	1	50-100	20k
USB	Async	127	16	1.5M,12M,480M
Ethernet	Async	1024	1600	10M,100M,1G

## CHAPTER 3: DEVELOPMENT AND TESTING

The objective of the thesis was to develop a host-PC communication toolkit. The communication interfaces considered for the proposed work were RS-232, USB1.1, and Ethernet. A considerable amount of time was spent on gathering requirements for RS-232, USB 1.1, and Ethernet interfaces. An object oriented interfaces were finally written in C++ to complete development phase of RS-232, USB1.1 and Ethernet based toolkit.

### **3.1 Development Phase**

Descendent classes were developed for RS-232, USB, and Ethernet interface during development phase using MFC classes. These classes can serve as extremely useful tool for developing user friendly software applications.

#### **3.1.1 RS-232 Interface Specifications**

- Using standard Readfile/Writefile functions to receive and transmit data.
- No handshaking is used in the interface.
- Communication is event driven.

#### **3.1.2 RS-232 Interface development**

Developing a serial class is divided into main six sections: opening serial port, configuring serial port, configuring timeouts, reading from port, writing to port, and closing the port.

### 3.1.2.1 Opening Serial Port

It is simple to open a serial port using windows file I/O. One important step is to include windows.h header file in the code developed. For opening a serial port, a handle to the serial port is created:

```
hComm = CreateFile (portname,
                    GENERIC_READ | GENERIC_WRITE,
                    0,
                    0,
                    OPEN_EXISTING,
                    0,
                    0);
```

The variable hComm calls CreateFile. The first argument is the name of the file to be opened. In this case, it is the name of the port say 'COM1', or 'COM2'. The next argument informs windows whether the port is going to be read or written. Argument 5 is open an existing file. If existing file is not present, error message is displayed. This handle can be utilized to get hold of the port.

### 3.1.2.2 Configuring Serial Port

Once we obtain the handle to the port, a Data Control Block (DCB) structure is used to configure the port for a certain set of parameters (baud rate, data bits, stop bit, parity).

```
DCB m_dcb;
if (!GetCommState(hComm, &m_dcb)) {
//error getting state
}

m_dcb.BaudRate = CBR_19200;
m_dcb.ByteSize = 8;
m_dcb.StopBits = ONESTOPBIT;
m_dcb.Parity = NOPARITY;
```

In this function first of all a variable of type DCB is created. Then GetCommState function is called. This function takes handle to the port and DCB structure variable as argument to fill in the parameters currently in use by serial port.

### 3.1.2.3 Setting Timeouts

If an application is continuously polling serial port, and in case there is not any data coming into serial port, then the application hangs while waiting for data to show up. So timeouts can be configured to fix this problem.

```
COMMTIMEOUTS m_CommTimeouts;

m_CommTimeouts.ReadIntervalTimeout = 50;
m_CommTimeouts.ReadTotalTimeoutConstant = 50;
m_CommTimeouts.ReadTotalTimeoutMultiplier = 10;
m_CommTimeouts.WriteTotalTimeoutConstant = 50;
m_CommTimeouts.WriteTotalTimeoutMultiplier = 10;

if (!SetCommTimeouts(hComm, &m_CommTimeouts)){
    //set error message
}
```

In this function we declare a variable of type COMMTIMEOUTS structure provided by windows. In this function some terms are used which are explained as:

- ReadIntervalTimeout specifies how long to wait between characters before timing out.
- ReadTotalTimeoutConstant specifies how long to wait before returning.
- ReadTotalTimeoutMultiplier specifies how much additional time to wait before returning for each byte that was requested in read operation.
- WriteTotalTimeoutConstant and WriteTotalTimeoutMultiplier does same thing, just for write instead of read.

#### 3.1.2.4 Reading and Writing to the Serial Port

For reading and writing, ReadFile/ WriteFile function are used. WriteFile takes handle to the port, byte to be written and NULL.

```
if (WriteFile (hComm, &bybyte , 1, &iBytesWritten, NULL)==0)
    return false;
```

ReadFile takes handle to the port, a buffer to store data, number of bytes to read, a pointer to an integer that will be set to the number of bytes actually read and NULL.

```
if (!ReadFile (hComm, &rxbuff , 1, &dwBytesTransferred, 0)){
    //error
}
```

#### 3.1.2.5 Closing Serial Port

Once serial communication is complete, it is extremely important to close the handle, which is pretty simple.

```
CloseHandle(hComm);
return;
```

### 3.1.3 USB Interface Specifications

- A choice of transfer type is made.
- A series of request to the target is made during enumeration process.
- A device address is assigned to the device.
- A specific ENDPOINT is chosen for the transfer of information.
- Writing lower level drivers are avoided.

#### 3.1.4 USB Interface Development

USB is much complex interface to implement. Initially, when a device is connected to PC, it is enumerated using control transfers. A firmware code has to be written on

hardware in order to answer the request made by host-PC during enumeration. Once enumeration is over, a host side interface can be used to address the hardware. A multithreaded application based class was written to fulfill the requirements of the USB Interface on host-PC. Isochronous transfer type was chosen for the interface. The USB interface code mainly depends on a procedure used to call driver to access the device. The procedure opens the host controller device driver using the symbolic name. In this function, a pointer to device driver handle where the file handle is placed is given as shown:

```
*hostcontrollerHandle=CreateFile(
    completecontrollername,
    GENERIC_WRITE,
    FILE_SHARE_WRITE,
    NULL,
    OPEN_EXISTING,
    0,
    NULL);
```

Using standard system call, the name of the node at the next level can be identified [16]. A handle to device can be obtained by making CreateFile system call. Using this handle, device can be probed for gathering information about the device. A DeviceIoControl system call is used for this. These system calls are a part of enumeration [16].

```
hDeviceHandle = CreateFile (
    completeDeviceName,
    GENERIC_WRITE,
    FILE_SHARE_WRITE,
    NULL,
    OPEN_EXISTING,
    0,
    NULL);
```

```

Flag = DeviceIoControl(
    hDeviceHandle,
    IOCTL_USB_GET_DESCRIPTOR_FROM_NODE_CONNECTION,
    &packet,
    Sizeof(packet),
    NULL,
    0,
    &BytesReturned ,
    NULL);

```

A further DeviceIoControl system call is made to start the transfer between the node and host [9]. In the proposed work isochronous USB transfer was used for data acquisition. This is implemented in code as follows:

```

ISO_TRANSFER_CONTROL IsoControl;

IsoControl.PacketCount = GetDlgItemInt(IDC_PacketCount,&Success,FALSE);
IsoControl.PipeNum = GetDlgItemInt(IDC_PipeNum ,&Success,FALSE);
IsoControl.BufferCount = GetDlgItemInt(IDC_BufferCount, &Success,FALSE);
IsoControl.FramesPerBuffer =
GetDlgItemInt(IDC_FramesPerBuffer,&Success,FALSE);

bResult = DeviceIoControl (hDevice,
    IOCTL_EZUSB_START_ISO_STREAM,
    &IsoControl,
    sizeof(ISO_TRANSFER_CONTROL),
    NULL,
    0,
    (LPDWORD)(&nBytes),
    NULL);

if (bResult != TRUE)
{
    hOutputBox->SendMessage (LB_ADDSTRING, 0, (LPARAM)"ISO
Transfer Failed");
    CloseHandle (hDevice);
    return;
}

```

A false value of flag ‘bResult’ in the code shown above can stop the transfer.

### 3.1.5 Ethernet Interface Specification

- The interface can be utilized to access devices supporting embedded Ethernet protocol.
- The interface can serve a client-server model based application.
- The Interface is a descendent of the CAsyncSocket Class.

### 3.1.6 Ethernet Interface Development

For Ethernet, a dialog based application is created. A descendent class was inherited from CAsyncSocket class. Development of Ethernet interface class is divided as: making a connection, sending and receiving messages and closing the connection. A client – server is model is developed for the interface. The descendent class MySocket.cpp is created. The primary reason for creating a descendent class is to capture the events when messages are received, or connections are completed. The CAsyncSocket class has a series of functions for each of the events. A brief description of all the event functions written for the interface is in following section.

#### 3.1.6.1 Accept Function

This function is called on a listening socket to signal that a connection request from another application is waiting to be accepted [6].

```
class CMySocket : public CAsyncSocket
CMySocket m_sListenSocket;
CMySocket m_sConnectSocket;

// Accept the connection request
m_sListenSocket.Accept(m_sConnectSocket);
```



### 3.1.6.2 Connect Function

This function is called on a socket to signal that the connection with another application has been completed and that the application can now send and receive messages through socket [6].

```
// create a default socket
m_sConnectSocket.create ( );
m_sConnectSocket.Connect (Server Name, Port number)
```

### 3.1.6.3 Send and Receive Function

Send function is called to signal that socket is ready to send data. This function is called right after connection is completed. Receive on the other hand signal that data has been received through socket and is ready to be retrieved in receive buffer [6].

```
m_sConnectSocket.Send (LPCTSTR("string"), length of message);
irec = m_sConnectSocket.Receive (Buf, BufSize);
```

‘irec’ flag is used to validate proper reception.

### 3.1.6.4 Close Function

This function signals that application on other end of the connection has closed its socket. This should be followed by closing the socket that received this notification.

```
m_sConnectSocket.Close();
```

## 3.2 Testing Phase

For partial fulfillment of the work, a dialog based application was written for testing RS-232 interface. This application was written for Nekton Research Inc. a Durham based company. This application was a part of project Biobay, which was a system to efficiently perform water quality measurement with extensive data collection and logging. In order to test USB 1.1, a USB I2C/IO development board from Devasys was used. An application hardware interface was set, in which ADC0848 was interfaced with

Devasys board. The data collected from ADC was sent to host-PC via Devasys board and USB interface class was used to collect data at host-PC end. For testing Ethernet interface, a dialog based application was developed that can function as either client or server in a Winsock connection. A server application was developed that can listen and accept connections from other network application.

### 3.2.1 RS-232 Interface Testing

A complete multithreaded real time data acquisition system is developed. The data coming from the sensors is monitored and data coming from them is displayed on individual screen. At the same time the data is logged on computer disk for analyzing data later. The complete interface exchanges data using RS-232 communication interface. The choice of RS-232 is based on the requirements of the complete system, which is efficiently fulfilled by RS-232 interface.

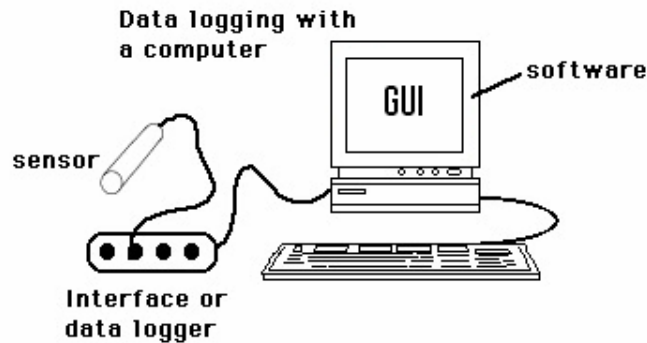


FIGURE 3.1 Data logging system.

### 3.2.1.1 System Description

A design of the system is described here. This system has four modules constituting the hardware to collect sensor data and a graphical user interface running on PC to finally display and log data. Figure 3.2 shows a block diagram of the data acquisition system.

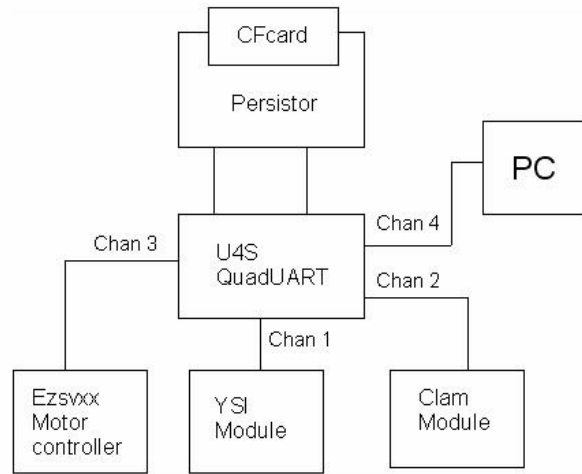


FIGURE 3.2 Data acquisition system.

### 3.2.1.2 Clam Sensor Bay

This board consists of 16 clam sensors. The magnitude of contraction or expansion of the clam is measured by the Hall Effect sensors, which is connected to the clam via a plunger. The analog data from the Hall Effect sensor is then send to MX1270 (12 bit, 8 channels ADC). The PIC16F876A microcontroller then transmits the ADC data to SBC via RS-232 interface. A snapshot of the clam sensor board is taken shown in Figure 3.3.

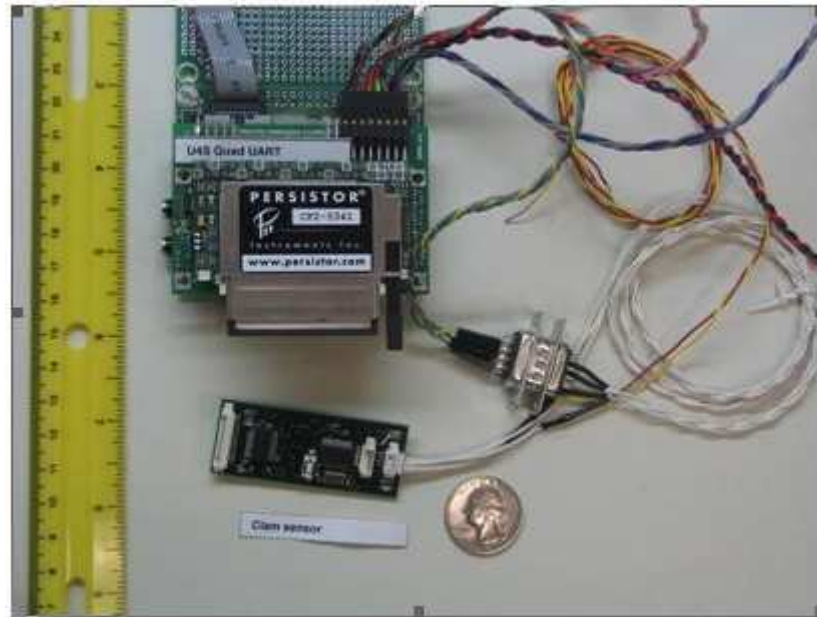


FIGURE 3.3 Clam sensor board.

### 3.2.1.3 YSI Sensor Bay

This board is a 6-Series Environmental monitoring system. It is used in our project to interface the following sensors.

- Temperature
- Conductivity
- Dissolved oxygen
- Depth
- pH
- Turbidity
- Chlorophyll

This board is configured to collect data and send it to the single board computer (persistor board). A snapshot of the YSI sensor board is taken shown in Figure 3.4.

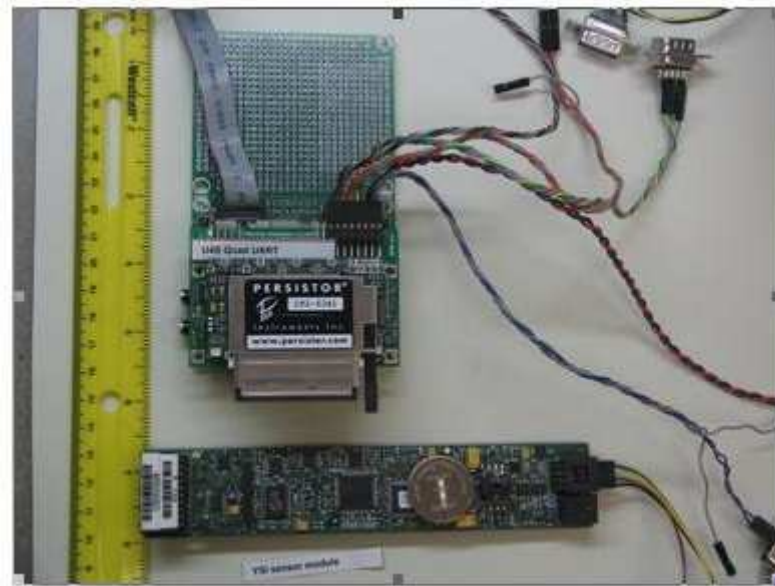


FIGURE 3.4 YSI sensor board

#### 3.2.1.4 Motor controller

A motor controller board is used to control the motor that drives the water sampler. A snapshot of the YSI sensor board is taken and shown in Figure 3.5.

#### 3.2.1.5 Single Board Computer

This module consists of Persistor, which is the main module and is responsible for communication between all different modules. It is responsible for sending all data collected from other hardware modules to GUI running on PC for displaying and logging the data. The Sensor fusion algorithm runs on this board, which decide whether a water sample has to be taken or not. A snapshot of the single board computer is taken and shown in Figure 3.6.



FIGURE 3.5 Motor controller board



FIGURE 3.6 Single board computer

### 3.2.1.6 Complete Hardware

A snapshot of complete hardware module is taken and shown in Figure 3.7. The interface used to communicate between single computer board and other sub modules is RS-232.

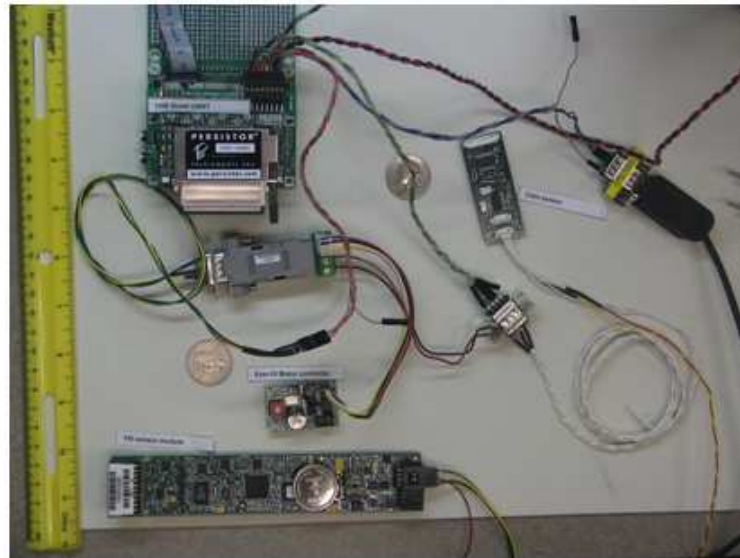


FIGURE 3.7 Complete hardware system.

### 3.2.1.7 Graphical User Interface

A simple, well organized graphical user interface was designed for windows environment. Clarity and consistency were the basis of the design. The design had four phases: Analysis, Design, Construction and Testing.

- **Phase 1: Analysis:** In this phase, GUI requirements written by the sponsor was discussed; collected and documented. High level user activities were identified. All possible design constraints were also taken into account. A user case scenario was created which consisted of all the tasks, describing how a user would use GUI.

- **Phase 2: Design:** All the information collected from the analysis phase was utilized to create a high level construction. Major modules of the GUI were discussed and designed. The modular design of the GUI gave flexibility to the programming of the integrated software.
- **Phase 3: Construction:** A detailed and realistic prototype of the GUI was created.
- **Phase 4: Testing:** In this phase tests were performed on the designed GUI to check if the final version was able to communicate efficiently with the selected communication interface. Fulfillment of all the requirements was also determined. All the shortcomings of the designed window software were eliminated.

#### **3.2.1.8 Window Design of GUI**

A snapshot of the GUI displaying the collected data is shown here in Figure 3.8. The user interface is supposed to monitor the data from 16 different biological sensors and 7 different environmental sensors. Displaying all the incoming data on a single screen had a trade off with efficient display screen for one single sensor, therefore a tabbed frame view is been utilized. Tab-1 was programmed to display data coming from biological sensors, while Tab-2 was programmed to display data coming from environmental sensors.



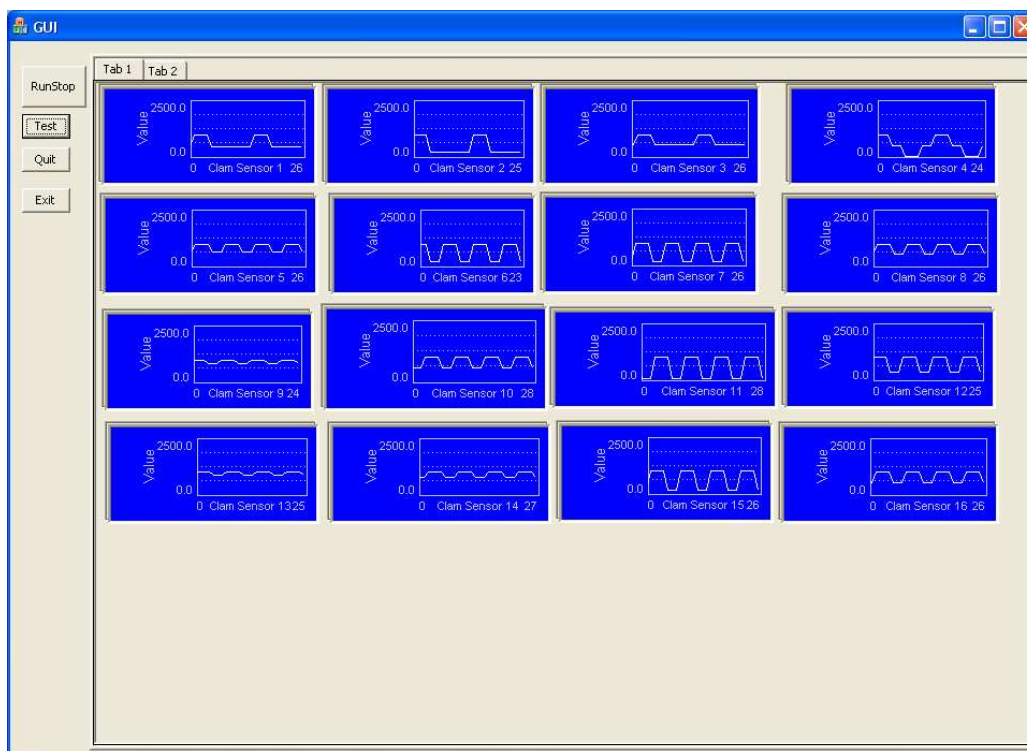


FIGURE 3.8.a Tab-1 displaying biological data.

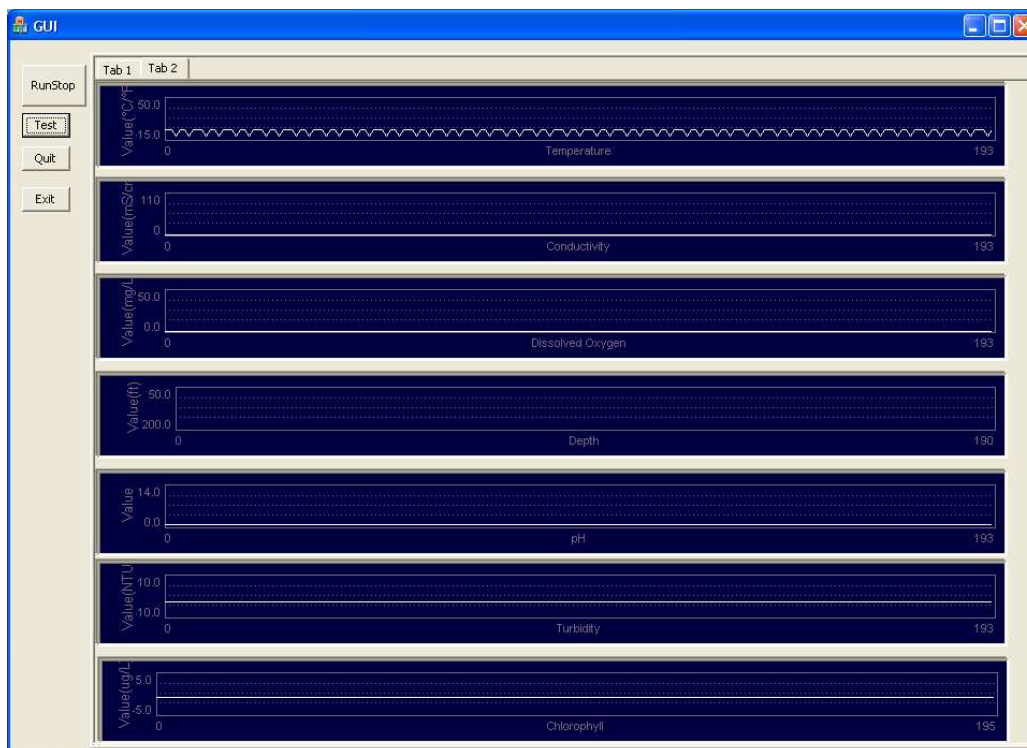


FIGURE 3.8.b Tab-2 displaying environmental data.

### 3.2.1.9 System Requirements

Before developing the system, a set of requirements were set and then the whole system was designed by keeping those requirements in mind. A list of the system requirements is presented here.

#### 3.2.1.9.1 Clam Sensor Board Requirements

- Power supply range is 7.5-12V, current 1A.
- One RS232 interface runs at 9600 Baud rate to send and receive data to the Persistor board. The maximum Baud rate can be 460 kbps. The Baud rate is currently set to 57600.
- The frame format is one start bit 8 data bits and 1 stop bit (8N1). The data is in 8-bit ASCII format.
- The cable connecting the Persistor and Clam Sensor bay should be shielded to prevent interference.

Table 3.1 shows a list of sensors used with YSI board and an analysis of the number of bytes sent from the sensors to YSI board.

TABLE 3.1 Clam sensor data format

Sensors	Data	No of Bytes	Data Format
	BBDS	4	
	Comma	1	
1	Sensor Data	4	xxxx
	Comma	1	
2	Sensor Data	4	xxxx
	Comma	1	
3	Sensor Data	4	xxxx
	Comma	1	
4	Sensor Data	4	xxxx
	Comma	1	
5	Sensor Data	4	xxxx

	Comma	1	
6	Sensor Data	4	xxxx
	Comma	1	
7	Sensor Data	4	xxxx
	Comma	1	
8	Sensor Data	4	xxxx
	Comma	1	
9	Sensor Data	4	xxxx
	Comma	1	
10	Sensor Data	4	xxxx
	Comma	1	
11	Sensor Data	4	xxxx
	Comma	1	
12	Sensor Data	4	xxxx
	Comma	1	
13	Sensor Data	4	xxxx
	Comma	1	
14	Sensor Data	4	xxxx
	Comma	1	
15	Sensor Data	4	xxxx
	Comma	1	
16	Sensor Data	4	xxxx
	Comma	1	
	Check sum	3	xxx
	Carriage return	1	
	Line Feed	1	
	Total	90	

### 3.2.1.9.2 YSI Sensor Board Requirements

- The YSI board uses the RS-232 standard for communication, with following specifications.
  - Baud rate 9600
  - Frame type 8N1
  - Power supply 12 V

- It uses command set comprising of ASCII characters, for configuration and calibration.
- The YSI board is connected to a PC through the RS-232 serial port during the initial configuration and calibration.
  - Hyper terminal is used for this set up.
  - The board should be connected to a PC for all recalibrations.
  - The final calibration is done onsite.
- The YSI board is configured to ‘power up to run’ mode with discrete sampling.  
In this mode, the YSI board starts to run as configured on power up.
- The YSI board is configured to sample data at the system sample rate.

Table 3.2 shows analysis of the number of bytes sent from the YSI board.

TABLE 3.2 YSI data analysis.

SENSORS	Data	no of bytes	Units	Range	Digits/Resolution
1	Temp	5	°C or °F	-5° - +45° C	0.01°C
	space	1			
2	Conductivity	5	mS/cm (milliSiemens/cm)	0-100 mS/cm	0.001-0.1 mS/cm
	space	1			
3	DO	6	mg/L (milligrams/L)	0 – 50 mg/L	0.2 mg/L
	space	1			
4	Depth	7	ft or m	0 – 30 ft (0 - 9 m)	0.001ft or 0.0003m
	space	1			
5	pH	5	0 – 14	0-14	0.2
	space	1			
6	Turbidity	5	NTU (nephelometric turbidity units)	0 – 1000 NTU	0.1 NTU
	space	1			
7	Chlorophyll	6	µg/L (micrograms/L)	0 – 400 µg/L	0.1µ/L
	CR	1			
	LF	1			
TOTAL		47			

### **3.2.1.9.3 Motor Controller Requirements**

- The motor is controlled by the persistor using RS-232 protocol.
- The fixture is manually aligned to an initial zero position before deploying the system.
- To collect the first water sample the motor is turned by 45°, by issuing the command P (move motor relative in positive direction).
- For subsequent samples the motor is moved by 90° using the same command.

### **3.2.1.9.4 Persistor Requirements**

- A supply of 4-9V DC is required to power the board.
- The persistor's time source is used as the time reference for the entire system.
- The persistor needs a CR2032 Lithium Cell for the Real Time clock.
- The persistor needs 4 RS-232 ports to communicate with other boards. The extra ports are provided by the add-on card U4S.
- It communicates with YSI (U4S-port 1) and clam sensor board (U4S-port 2) and motor control board (U4S-port 3) over 3 RS-232 ports at 9600 Baud and 8N1 format.
- It communicates with the PC (U4S-port 4) over RS-232 at 19200 Baud and 8N1 format.
- It gets data from the YSI and the clam sensor board in ASCII format. The data is saved onto the compact flash card and is also sent over to the PC with time stamp.
- The sensor fusion algorithm is executed on this board. Based on the result the persistor will command the motor controller to index and take a water sample.
- It will implement and execute the various commands from the PC.
  - Start sampling.

- Stop sampling.
- Set algorithm parameters.

#### **3.2.1.9.5 Graphical User Interface Requirements**

- The GUI runs on a Pentium-class PC with a minimum of 256MB of RAM, 1GHz processors speed or above, and 50MB free hard drive space.
- The GUI runs on a PC with a screen of minimum 800 by 600 pixels.
- One RS-232 interface port running 19200 baud is used to communicate with the hardware collecting data.
- The GUI will set up the Persistor board for the starting and stopping of data collection.
- The GUI displays the sensor's data on their respective display window. A log file of the data collected is also stored for further analysis of data.

#### **3.2.1.10 Software Design for GUI**

For testing the developed communication interface an efficient multithreaded GUI was designed. The interface is integrated with other modules and a fully functional dialog based user application is tested for fulfilling set requirements. A flowchart (Figure 3.9) defines the tasks running for GUI.

The application does multiple tasks by sharing processor time. The application not only monitors RS-232 port, it also displays collected data and writes data to a file on the hard disk. This capability of the application is because of its multithreaded nature.

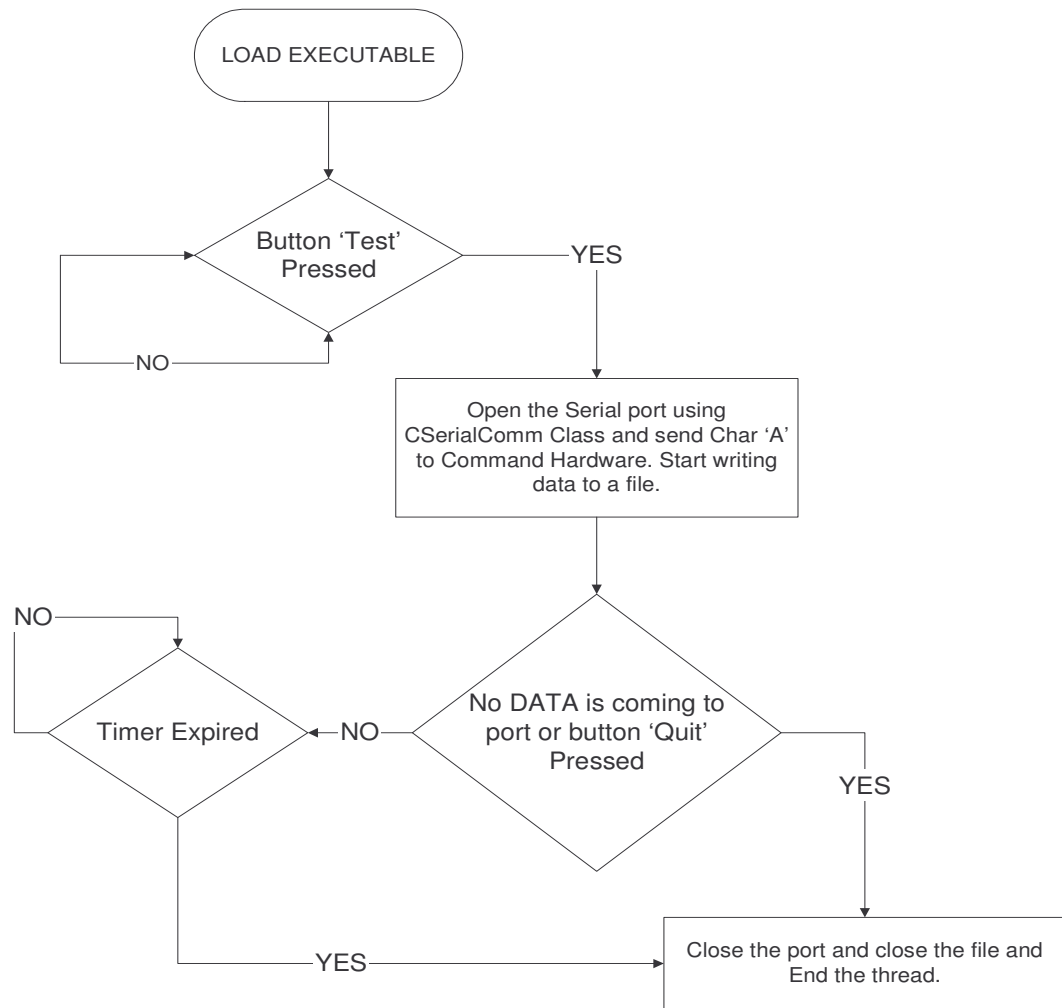


FIGURE 3.9 Software design flowchart

### 3.2.2 USB Interface Testing

For testing USB PC-host side interface, another data acquisition system was set. The hardware setup for this data acquisition system is shown in the Figure 3.10.

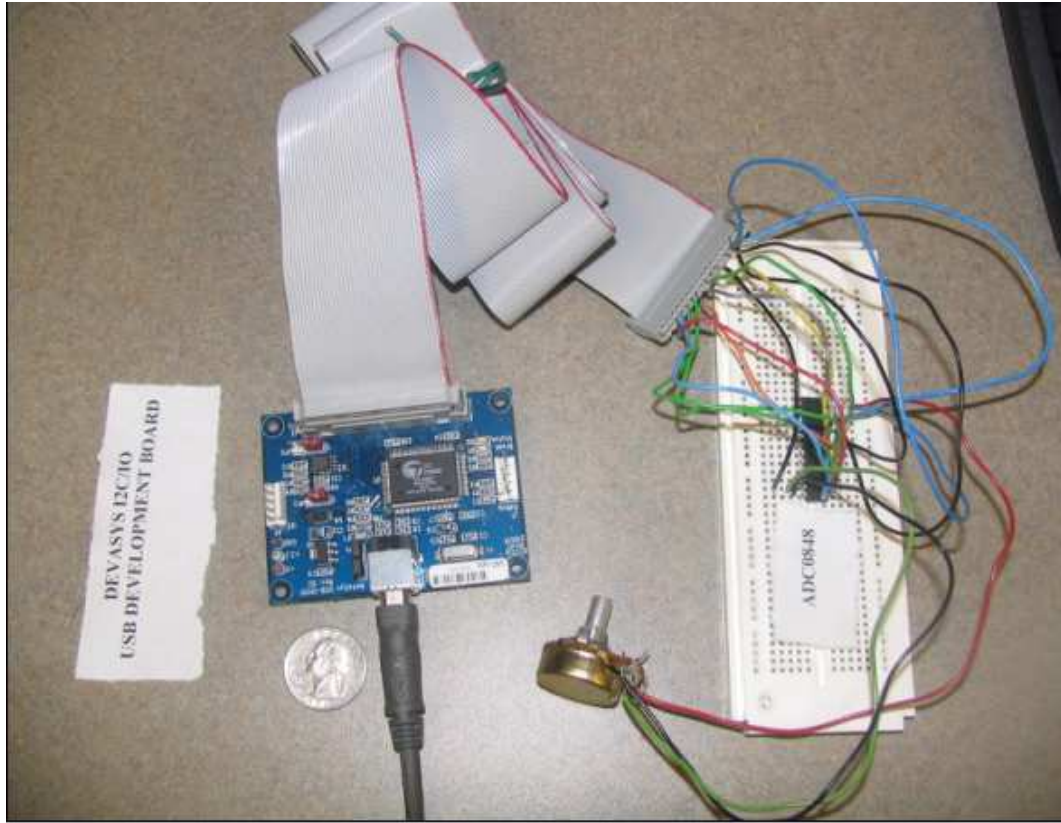


FIGURE 3.10 USB data acquisition hardware setup

A USB development board from Devasys is used for the USB data acquisition system. This board is interfaced with the ADC0848 and data collected from ADC chip is sent to host-PC via Devasys board. An application integrated with the USB interface class is used to monitor the incoming data. The data retrieved is displayed in a dialog based application as shown in Figure 3.11.



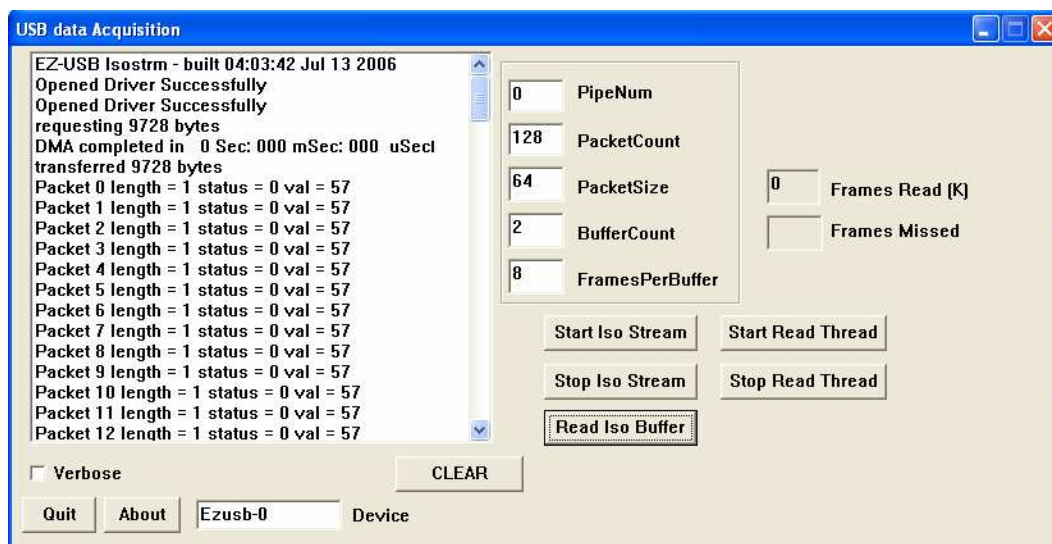


FIGURE 3.11 USB data acquisition on host-PC

### 3.2.3 Ethernet Interface Testing

For the proposed work, no embedded target has been tested for testing the Ethernet interface. A desktop based client-server model however is created to test the working of event functions developed for Ethernet interface.

### 3.3 Multitasking

Applications used during time of Windows 3.x were single threaded, with only one path of execution at any point in time. Then cooperative multitasking was offered, in which each individual application decides about when to give up the processor for another application to perform any waiting processing. But in worst case of such a concept, an application would be held in waiting state, if another application got stuck in some never ending loop. Then preemptive multitasking was introduced, in which a higher priority waiting task pre-empts current task. Applications running on a PC are divided into processes and threads. Processes are various applications running in the kernel mode of the PC sharing CPU time and every process has the capability to execute multiple

threads at anytime. Threads basically run in the user mode to avoid overhead of operating system interference with their execution.

### **3.3.1 Idle Process Thread**

Adding Idle threads are called when there are no messages in application message queue. While an application is idle, it can perform work such as cleaning memory or writing to a print spool. The function used in developing window application is OnIdle() and it's a holdover from the Windows 3.x days. For the proposed work, no task is defined for this function.

### **3.3.2 Independent Threads**

In order to do a long background task without interfering with other running tasks, an independent thread has to be created. To create and start an independent thread there are many methods. One of them is calling AfxBeginThread function from windows API. A function to call can be passed to this function for performing thread's task. A pointer is returned to a CWinThread class object, which actually runs as an independent thread. The threads are even prioritized. This priority of the threads control how much of CPU time thread gets in comparison to other threads. Every thread has its own stack and always has some default value, which makes it optional.

### **3.3.3 Inter-tasks Communication**

Sometimes it becomes necessary for one task to communicate with other tasks. This is one of the most complex issues because, while communicating, one task should not get into other's way when engaging in critical activities. The issue is mainly shared data corruption. Managing access to shared resources is most challenging task, while developing multithreaded application. Sharing does not work too well in a multithreaded

application. There are ways to limit access to a common resource to only one thread at a time. Some of them are:

- Defining critical sections
- Using Mutexes
- Use of Semaphores

### **3.3.4 Building a Multitasking Application**

There are many ways to add multithreading capability to an application. Microsoft provides an efficient API to add multithreading capability to an application. With MFC a descend class can be developed. A simpler way is to give a standard call to the independent function, which will be doing its task in the background without interfering with other ongoing tasks of application.

## CHAPTER 4: DEVELOPMENT TOOLS

The communication interfaces for the proposed work are developed in Microsoft Visual C++ 6.0 Integrated Development Environment. Windows programming approach has been used in order to make it easy to write user interface application on top of communication interface classes. This chapter is a rapid tour of working in this IDE. The best approach to getting familiar with it is to work through creating, compiling and executing simple program.

### **4.1 Introduction to Windows Programming**

A windows program has a different structure to that of typical DOS program, and it's rather more complicated. In a DOS program, keyboard can give input and can be written to the display directly, whereas a windows program can only access the input and output facilities of the computer by way of windows functions; no direct access to these hardware resources is permitted. Since several programs can be active at one time under windows, windows has to determine which application a given input is destined for and signal the program concerned accordingly. Windows has primary control of all the communication with the user. The user actions are all regarded by windows as events, and result in a particular piece of program code being executed [14]. A windows program is basically written to customize windows to provide a particular set of capabilities. Even an elementary windows program involves quite a few lines of code. MS VC++ AppWizard makes things easy and provides a readymade framework to begin coding. A structure of windows program is shown in Figure 4.1.

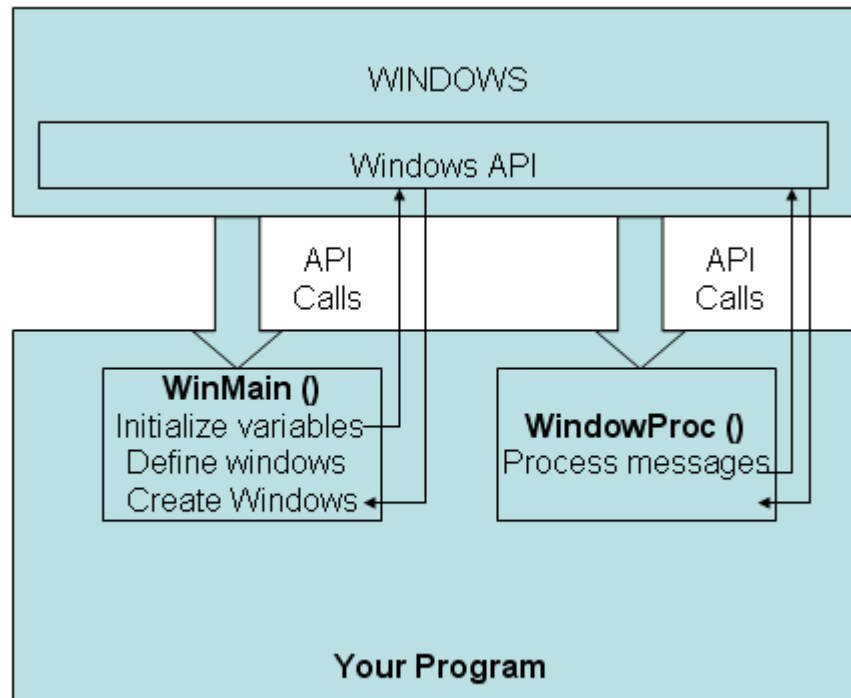


FIGURE 4.1 Windows program structure[30]

## 4.2 Integrated Development Environment

The IDE of Microsoft VC++ 6.0 is a self contained environment for creating, compiling, linking and testing windows programs. The fundamental components of the IDE are the editor, the compiler, the linker and the libraries.

### 4.2.1 The Editor

The editor provides an interactive environment for creating and editing the source code. The editor automatically recognizes fundamental words in C++ language and assigns different color to them based on what category they fit in.

### 4.2.2 The Compiler

The compiler converts source code into machine language, and detects and reports errors in the compilation process. The compiler has the ability to detect a wide range of

errors occurring due to unrecognized code. The output from a compiler is called object code. This object code is stored in files with extension .obj.

### **4.2.3 The Linker**

The linker is used to combine various modules generated by the compiler from source code files, adds required code modules from program libraries supplied as a part of C++, and welds everything to generate an executable. The linker can also report errors. If a part of the program is missing, a non-existent library component is referenced.

### **4.2.4 The Libraries**

A library support extends C++ language capability to include ready made routine sets to the application development. There is a basic set of routines common to all C++ compilers which make up Standard Template Library (STL). The window provides an application programming interface (API). It consists of a large set of functions that provide all lower level interfaces. The problem with windows API is, it has thousands of functions and it was not written keeping Visual C++ in mind. API has to be usable in programs written in a variety of languages, most of which are not object oriented. So MFC was developed which is nothing but a set of classes upon which windows programming with visual C++ is built. These classes represent an object oriented approach to windows programming that encapsulates the windows API. There is another library provided by Visual C++ which is Active template library (ATL). ATL provides structure for writing specialized windows programs.

### 4.3 Using Visual Studio IDE

The most important tools VC++ 6.0 which work in an integrated way to help write windows programs are AppWizard and ClassWizard. The AppWizard generates a basic framework for writing windows programs. The class provides an easy way to extend the classes provided by AppWizard. Most of the program development and execution is performed within IDE. When VC++ is started first time with no project active, a window shown in Figure 4.2 pops up. The workspace window helps to navigate through all the program files of the project. The editor window is the place where the source files can be written and modified, and the output window displays messages that result from compiling and linking the program.

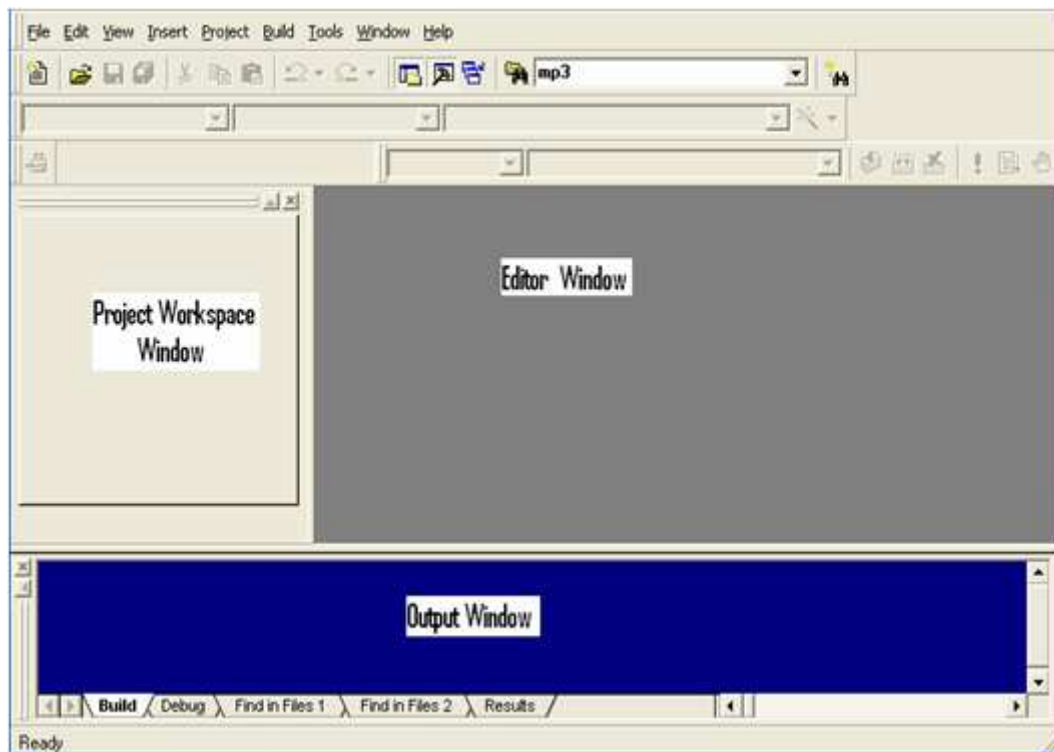


FIGURE 4.2 VC++ inactive IDE

### 4.3.1 Creating Project Workspace

After opening VC++, one is all set to start making a workspace for their project. A project workspace is a folder in which all the information relating to a project is stored. Once a project is created, a project workspace is created automatically, and VC++ maintains all the source code and other files in the workspace folder. The proposed work consists of the code developed by using MFC application wizard. The code written for proposed work is a win32 application, using MFC for support. The workspace folder holds the project definition files. The project definition includes a project name, a list of source files, the options set for editor, compiler, linker and other components of VC++. The basic definition of a project is actually stored in a file with the extension .dsp. A walkthrough of making a project is shown in following Figures:

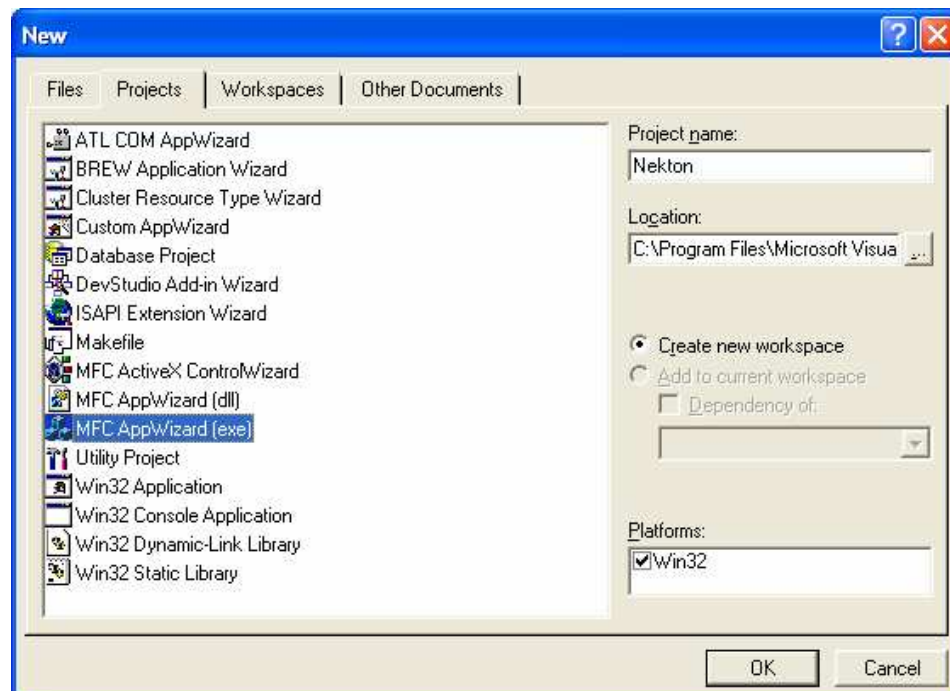


FIGURE 4.3 Opening a new project using MFC AppWizard



The code written for the proposed work is a dialog based application so 'Dialog based' is selected in MFC AppWizard step1. The process of creating a framework is continued there after.

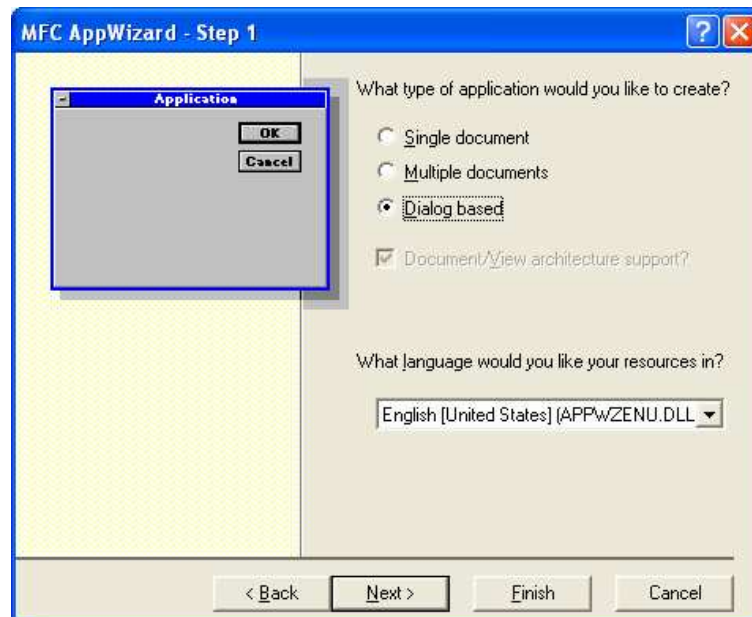


FIGURE 4.3.a Step-1 of creating a project framework

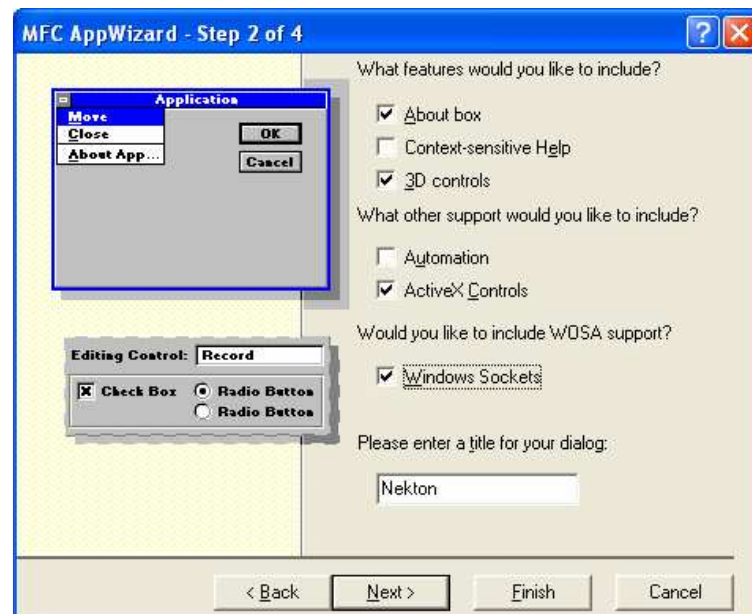


FIGURE 4.3.b Step-2 of creating a project framework

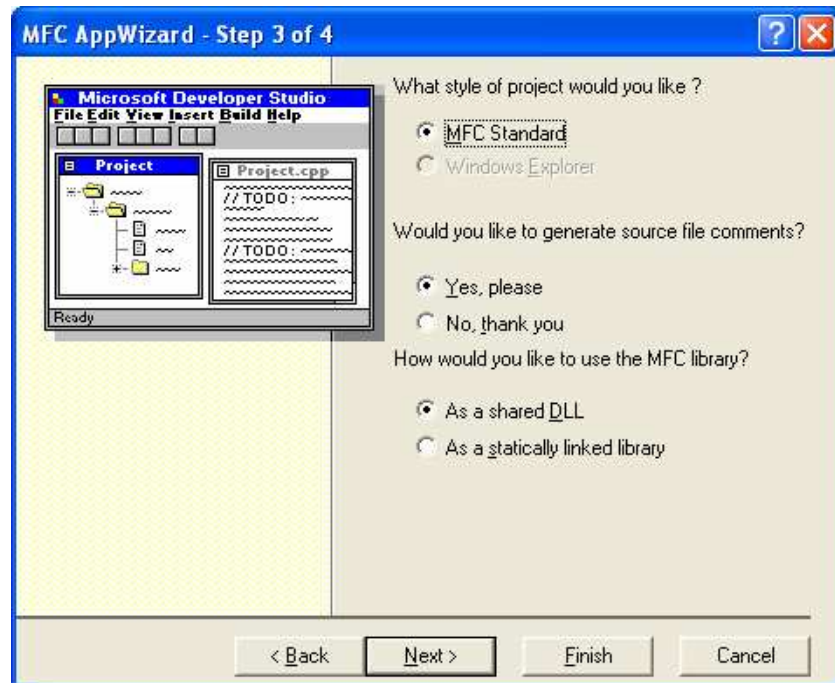


FIGURE 4.3.c Step-3 of creating a project framework

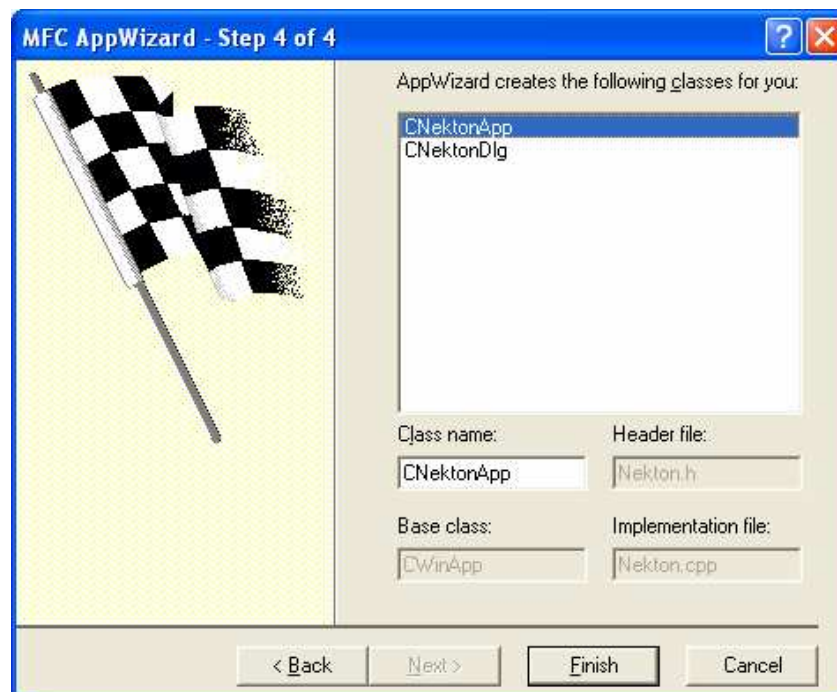


FIGURE 4.3.d Step-4 of creating a project framework

Once the AppWizard completes all steps to create a framework, all the information about the project is displayed. It is always important to make sure that all the features of application, which were supposed to be set, are displayed accordingly. An example is shown in Figure 4.3.e.

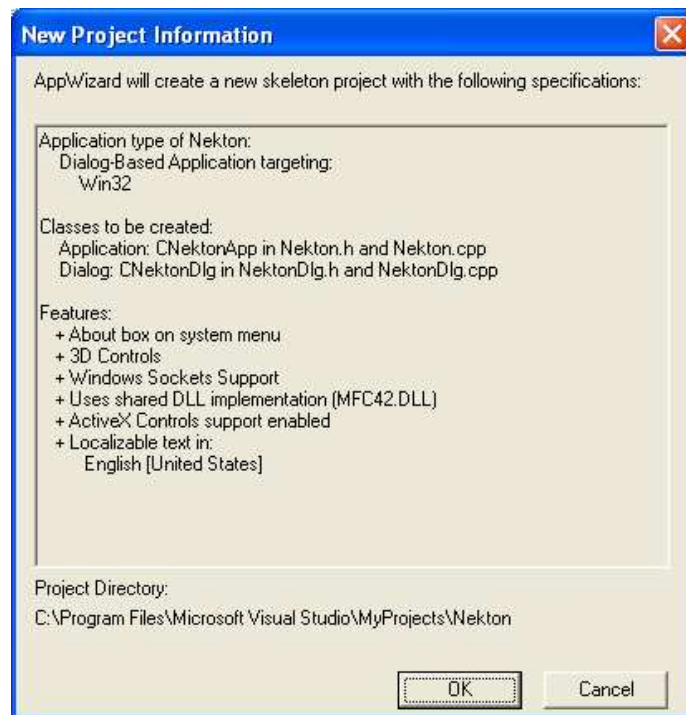


FIGURE 4.3.e Project information window

VC++ also provides option through project | settings menu to determine how a source code is to be processed during compile and link stages. The set of options that produces a particular executable version of program is called configuration as shown in Figure 4.4:

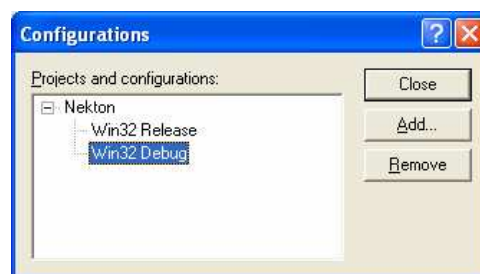


FIGURE 4.4. Set active configurations

Once framework is set by AppWizard, the project workspace with desired features is created and it looks as shown in Figure 4.5.

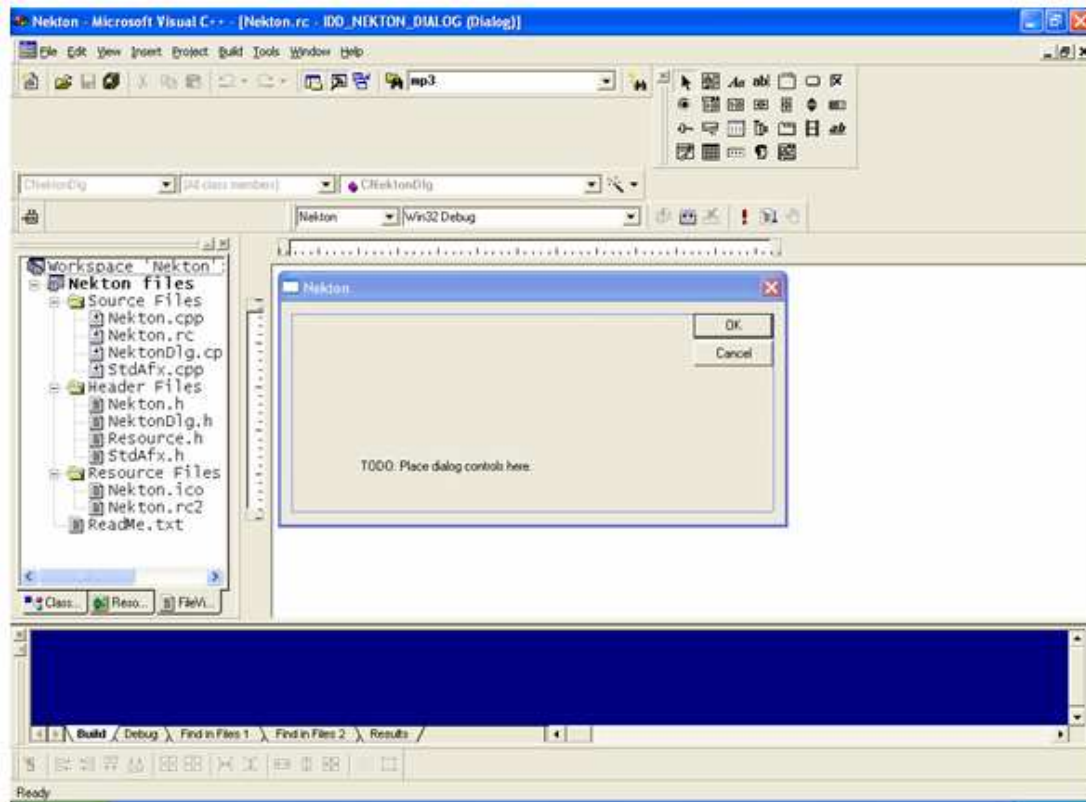



FIGURE 4.5.Active project workspace

### 4.3.2 Building Project

After creating the active project, it is build using a build icon  on the tool bar. Once the example is built without any error, a new sub- older either release or debug depending on the configuration of the project is created in the project folder. This folder contains output of build that is performed on the project. This folder contains seven new files. The Figure 4.6 shows the debug folder created after building the active project.

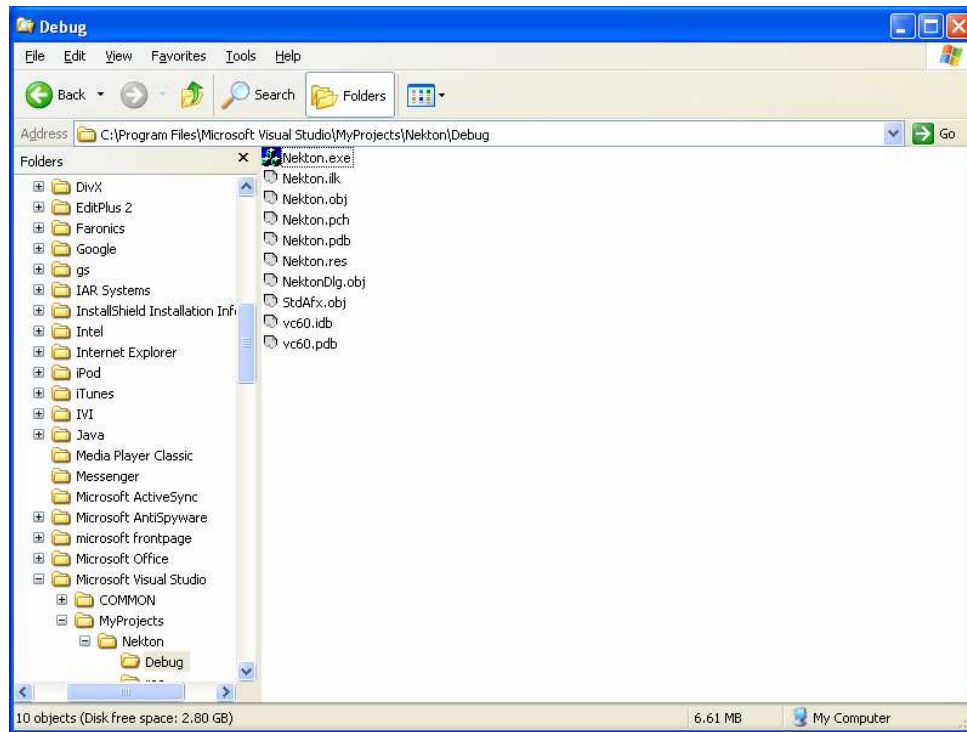



FIGURE 4.6 Files created after building the project

Table 4.1 gives a quick run through of the use of kind of files created after building the project.

TABLE 4.1 Description of files created after building the project [14].

File Extension	Description
*.exe	This is executable file which is obtained by successful compiling and linkage.
*.obj	This is object file which is obtained after compiling and is used to create executable file by linker.
*.ilk	This is used by the linker to link libraries with the modified code object files.
*.pch	This is a pre compiled header file. Because of this file, lot of time is saved to rebuild the whole program.
*.pdb	This file contains debugging information that is used when program is executed in debugging mode.
*.idb	This file contains additional debug information.

Once a project is build without any errors, application can be executed by clicking the execution icon .

#### 4.3.3 Debugging

Bugs are the errors in the program. Debugging errors are an integral part of programming. Debugging is the major activity performed during the testing phase of the application development. There are following major strategies followed to make debugging as painless as possible:

- Using available library facilities as much as possible. This helps in using as much pre-tested code as possible.
- Developing and testing code incrementally. This helps in reducing development process.
- Adding debugging code.
- Using debugger programs available in IDE.

Debugger programs were available long before IDEs, and they can still be used from the command line. However, using a built in debugger of an IDE is pretty simple. A debugger execute program incrementally rather than all at once. After each increment of the program executes, the debugger pauses, and contents of variables can be viewed. Once the variables are examined carefully, debugger can be directed to execute next statement. A small module of the program can also be debugged by setting breakpoints between the start and end of that module of the code. Figure 4.7 and Figure 4.8 show how to set a current project in debug mode.



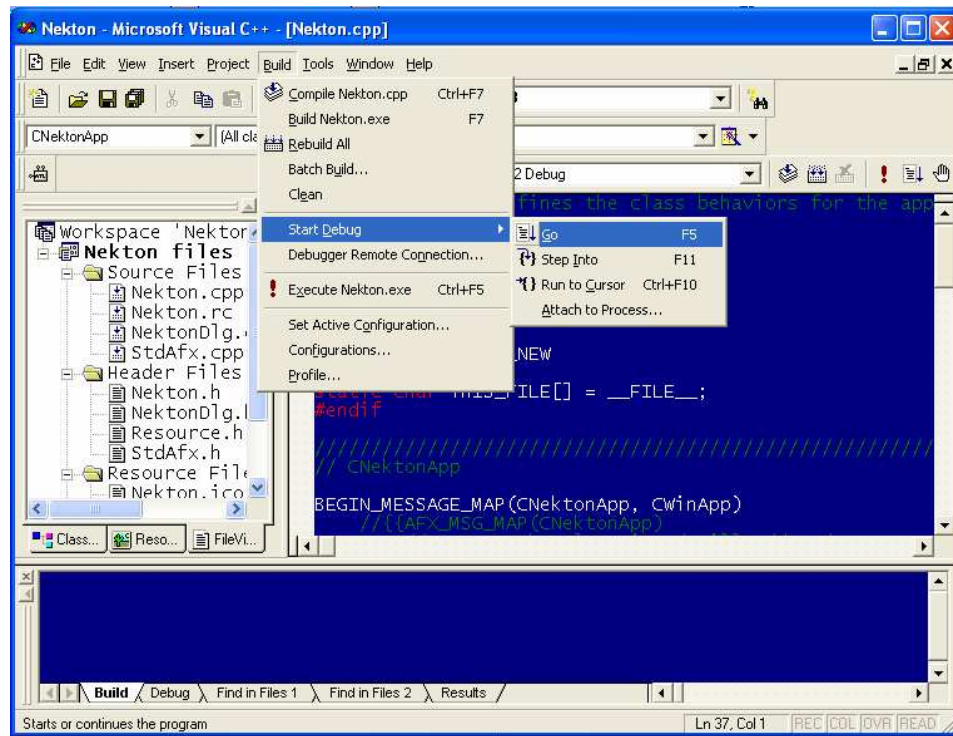


FIGURE 4.7. Starting debugger in VC++

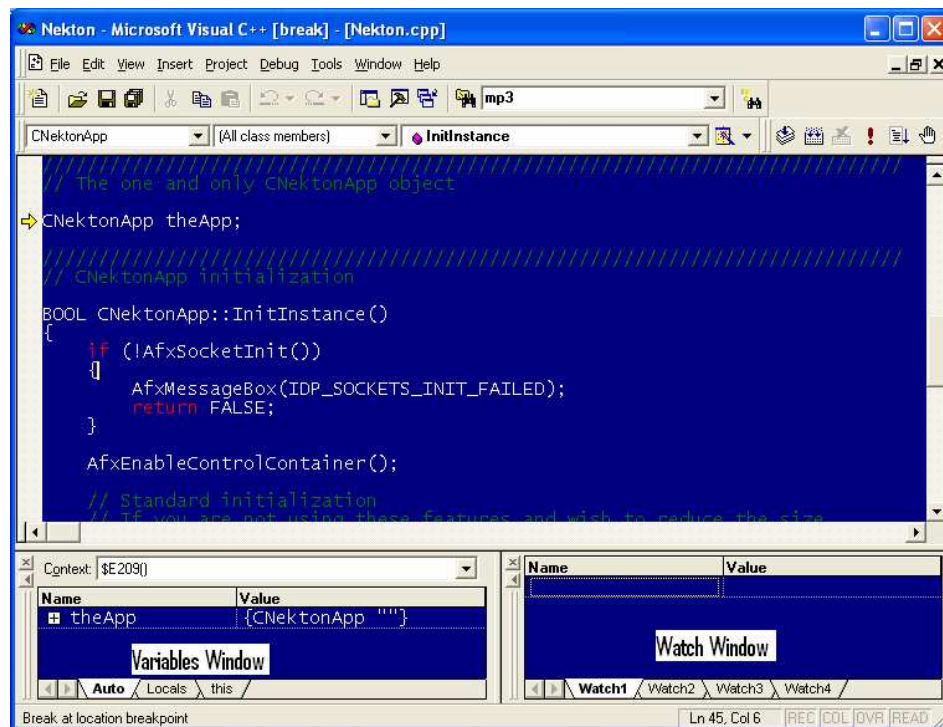


FIGURE 4.8. Using debugger in VC++

## CHAPTER 5: FUTURE DEVELOPMENT

### 5.1 Conclusion

The proposed work is a step to provide an easy-to-use interface solution for real-time data acquisition. This work has the potential use in providing flexibility and in decreasing development time significantly for any application built on top of the interface. This work can be used in the following areas:

- Communicating embedded targets supporting RS-232, USB or Ethernet interface.
- Developing other user friendly applications without much effort using the presented interfaces.
- Logging large amount of data on hard disk for remote analysis.
- Displaying data received from a large number of sensors on a single serial line on their respective display windows.
- Porting the Win32 based code written for RS-232, USB, and Ethernet interface in WinCE environment for their use in handheld devices.
- Providing a better way for Nekton Research Inc., a Durham based company to monitor data received from their environmental and biological sensors.
- Adding a valuable resource to the ongoing research at UNC Charlotte's Department of Electrical and Computer Engineering.



## 5.2 Future Work

The presented work was done for Win32 environments. The application developed can easily be made to run on desktop computers. But there will be times when tasks have to finish by specific times. In those times, the need of real-time environment will be seen. There are many real time operating systems available and they can be configured for handheld devices. One of such real-time operating systems is WinCE and it can be designed from the ground up to be as small as possible. Although both desktop operating system and Windows CE are feature driven, WinCE takes measures to ensure that features can be selectively included or excluded, depending on the specific needs of the hardware on which it is ported.

### 5.2.1 API and SDK supported by WinCE

Over the years, Win32 has accumulated a huge number of peripheral SDKs and API toolkits for development. WinCE does not support all of them but Microsoft Inc. chooses those that are crucially important, they also incrementally add new ones too. Some of the most used APIs supported by WinCE are:

- Multimedia
- WinSock
- Remote Access Services
- Windows Networking

Apart from the listed APIs, one most important API supported by WinCE is Setup. Installing programs on a WinCE device is different than on the desktop because it must be done remotely over some communication medium. For this reason, WinCE does not implement desktop setup APIs, but provides new functionality of its own [19].

### 5.2.2 Why WinCE?

Choosing specifically WinCE for future development is based on the comparison of the Real Time OS done by Dr. Timmerman and Dr. Perneel [28]. Their work is limited to three major real time operating systems used in the industry:

- VxWorks RTOS from Wind River Systems
- WinCE 5.0 from Microsoft
- Montavista Linux Professional 2.1 from Montavista

All the important issues in choosing a real time operating system are considered in their work. A careful examination of the work done by Dr. Timmerman and Dr. Perneel gives more credits to WinCE 5.0. Above all, the code written for presented work is in the Win32 environment and it makes more sense to use a compact version of Windows to add real time capability to the applications.

## REFERENCES

- [1] A. Abdusalam, A.R. Bin Ramli, N.K. Noordin, Md.L Ali, "Real Time Data Acquisition and Remote Controlling Using World Wide Web", *Student Conference on Research and Development*, July 2002, pp. 456-459.
- [2] P.C Abegglen, W.R. Faris, W.J. Hankley, "Design of a Real-Time Central Data Acquisition and Analysis System", *Proceedings of IEEE Conference*, Vol. 58, Jan. 1970, pp. 38-48.
- [3] J. Axelson, "Embedded Ethernet and Internet complete", *Designing and Programming Devices for Networking, 2003*, First Edition, Penram International Publishing (INDIA) Pvt. Ltd, Mumbai.
- [4] Beyond Logic Tutorials on RS-232, USB protocols.  
Website: <http://www.beyondlogic.org/serial/serial.htm#1>
- [5] T.J. Cacicchi, "Experimentation and Analysis: SigLab/MATLAB Data Acquisition Experiments for Signal and Systems", *IEEE Transactions on Education*, Vol. 48, Aug. 2005, pp. 540-550.
- [6] D. Chapman, J. Bates, "Teach yourself VC++ 6.0", *Sams Techmedia*, INDIA.
- [7] J.S. Chen, C.J. Wang, S.J. Chen, G.J. Jan, "A Graphical User-interface Control System at SRRC", *Proceedings of 1993 Particle Accelerator Conference*, Vol. 3, May 1993, pp. 1878-1880.
- [8] D.B. Crosetto, "Real-time System Design Environment for Multi-channel High-Speed Data Acquisition System and Pattern Recognition", *11th IEEE NPSS Real Time Conference*, June. 1999, pp. 329-336.
- [9] Cypress USB training material  
Website: <http://www.cypress.com/portal/server.pt>
- [10] B.S. Drakulic, S.J. Berry, M.N. Gold, Z. Konstantinovic, "A Real Time Data Acquisition and Signal Processing Unit for Biomedical Applications", *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, Vol. 3, Nov. 1988, pp. 1260-1261.
- [11] D. Engberg, T. Glanzman, "A Small Unix Based Data Acquisition System", *IEEE Transactions on Nuclear Science*, Vol. 41, Issue. 1, Feb. 1994, pp. 77-79.

- [12] A. Gani, A.J.E. Salami, "A Lab VIEW Based Data Acquisition System for Vibration Monitoring and Analysis", *Student Conference on Research and Development*, July 2002, pp. 62-65.
- [13] Z. Guzik, S. Borsuk, K. Traczyk, M. Plominski, "Enhanced 8K Pulse Height Analyzer and Multi-Channel Scaler (TUKAN) with PCI or USB Interfaces", *IEEE Nuclear Science Symposium Conference Record*, Vol. 3, Oct. 2004, pp.1444-1447.
- [14] I. Horton, "Beginning Visual C++ 6", 1998, Sixth Edition, Wrox Press, Inc, USA.
- [15] B. Hubbs, "A Survey of Highly Integrated Ethernet DataComm Devices", *IEEE Aerospace Conference*, Vol. 4, Mar. 1998, pp. 489-498.
- [16] J. Hyde, "USB design by example", *A Practical Guide to Building I/O Devices*, 2001, Second Edition, Intel Press, USA.
- [17] S. Martin, "PC-based Data Acquisition in an Industrial Environment", *IEE Colloquium on PC-Based Instrumentation*, 1990, pp. 2/1-2/3.
- [18] R. Moody, P.Turner, "Clam Gape Sensing Equipment for Water Monitoring", *Sea-Technology Magazine*, March 2006, pp. 28-32
- [19] MSDN online  
Website: <http://msdn2.microsoft.com/en-us/default.aspx>.
- [20] D.D. Nigus, S.A. Dyer, "An Easy-to-use, Host-independent Data Acquisition System", *6<sup>th</sup> IEEE Instrumentation and Measurement Technology Conference*, April. 1989, pp. 86-91.
- [21] J.R. Payne, B.A. Menz, "High Speed PC-Based Data Acquisition Systems", *IEEE Industry Applications Conference*, Vol. 3, Oct. 1995, pp. 2140-2145.
- [22] O. Postolache, J.M.D Pereira, P.S. Girao, "An Intelligent Turbidity and Temperature Sensing Unit for Water Quality Assessment", *Canadian Conference on Electrical and Computer Engineering*, Vol. 1, May 2002, pp. 494-499.
- [23] B. M. Pride, "Simple USB Data Acquisition", *Circuit Cellar*, April 2005, pp. 20-26.
- [24] V. De Rossi, P. Batsomboon, S. Tosunoglu, D.W. Repperger, "Interactive Modular Graphical User Interface Development for Telesensation Systems", *IEEE International Conference on Systems, Man and Cybernetics*, Vol. 2, 1997, pp. 1604-1608 .

- [26] RS-232 Specification and Standard.  
Website: [http://www.lammertbies.nl/comm/info/RS-232\\_specs.html](http://www.lammertbies.nl/comm/info/RS-232_specs.html)
- [27] D.J. Sides, "A Dynamically Adaptable Real Time Data Acquisition and Display System", *Proceedings of Real-Time Technology and Applications Symposium*, May. 1995, pp. 50-51.
- [28] M. Timmerman, L. Perneel "Understanding RTOS Technology and markets"  
Website: <http://www.download.microsoft.com>
- [29] A. Yiming, T. Eisaka, "An Ethernet Protocol for Real-Time Communications", *SICE Annual Conference*, Vol. 2, Aug. 2004, pp. 1905-1908.
- [30] B. Zdanivsky, "Browser-Based Telemetry System", *Circuit Cellar*, December 2005, pp. 12-18.
- [31] S. Zimmermann, V.H. Areti, G.W. Foster, U. Joshi, K. Treptow, "FASTBUS Readout Controller Card for High Speed Data Acquisition", *Conference Record of the 1991 IEEE Nuclear Science Symposium and Medical Imaging Conference*, Vol. 2, 1991, pp. 794-798.
- [32] A.C. Zoric, S.S. Ilic, "PC-Based System for Electrocardiography and Data Acquisition", *7th International Conference on Telecommunications in Modern Satellite, Cable and Broadcasting Services*, Vol. 2, Sep. 2005, pp. 619-622.

## APPENDIX

Due to the extensive size of the project, program code is not attached. If project code is needed, please contact the following individuals:

Dr. James Conrad

Email: [jmconrad@uncc.edu](mailto:jmconrad@uncc.edu)

Gajendra Singh

Email: [to.gsingh@gmail.com](mailto:to.gsingh@gmail.com)