DEVELOPMENT OF THE REAL-TIME DATA ACQUISITION SYSTEM FOR PHILIPS PATIENT MONITOR

By

FEI GUO

Submitted in Partial fulfillment of the requirements for the degree of

Master of Science

Thesis Advisor: Professor Kenneth A. Loparo, Ph.D.

Department of Electrical Engineering and Computer Science

CASE WESTERN RESERVE UNIVERSITY

August, 2014

CASE WESTERN RESERVE UNIVERSITY
SCHOOL OF GRADUATE STUDIES

We hereby approve the thesis/dissertation of

FEI GUO

Candidate for the Master of Science degree *.

Kenneth A. Loparo

(Chair of the committee)

Farhad Kaffashi

(Committee member)

Vira Chankong

(Committee member)

Date of Defense

07/16/2014

*We also certify that written approval has been obtained

for any proprietary material contained therein.

Contents

List of Figures

List of Tables

Acknowledgements

First, I would like to express the deepest appreciation to my academic advisor, Professor Kenneth A. Loparo for his immense knowledge, and patience. Every time I run into unexpected obstacle, Professor Loparo always points out the problems and offers suggestions and solutions, which inspire my research a lot. Without his guidance and help, this thesis would not have been possible.

I would also like to thank my committee members, Dr. Farhad Kaffashi, Professor Vira Chankong, for serving as my committee members. I also want to thank them for letting my defense be an enjoyable moment, and for their brilliant comments and suggestions.

I also want to thank Dr. Farhad Kaffashi and Dr. Hassan Mohammadi-Abdar who motivated me to look beyond the problems. Their expertise and work ethics teach me a lot.

Last but not the least, I would like to thank the Charles Stark Draper Laboratory for supporting my work through the University Research and Development grant.

Development of the real-time data acquisition system for Philips Patient Monitor

Abstract

by

FEI GUO

Effective acquisition and use of patient data in the Intensive Care Unit (ICU) is complex, involving the collection, interpretation and analysis of large volumes of high frequency physiological data provided by ICU monitors and devices. Data Acquisition Systems can play a critical role in clinical decisions support by providing real-time patient data at the bedside with the necessary visualization and analytic tools that transform the data to actionable information. In this thesis, we propose a Data Acquisition and Processing System for the IntelliVue Philips Patient Monitor via serial communication. The Data Acquisition System supports the collection of real-time physiological data, such as waveforms, numerics and alarm data, and supports fundamental interpretation and integrated analysis that could further assist medical personnel to estimate the relationships between clinical events and patient condition and to determine the appropriate interventions and assess their outcomes.

Keywords

Data Acquisition, Physiological data, ICU, Patient Monitor, Serial Communication

1 Introduction

1.1 Background

The intensive care unit (ICU), also termed critical care unit (CCU), intensive therapy unit or intensive treatment unit (ITU) is a particular department of a hospital or health care facility that provides intensive care medicine [1]. Intensive care units provide emergency treatment to patients with the most severe and life-threatening illnesses and injuries. It requires uninterrupted attentiveness and medical care from various specialists and medical equipment to sustain life and to ensure healthy bodily functions. They are staffed with highly trained doctors and critical care nurses who specialize in carrying out accurate and efficient emergency treatments for severely ill patients.

Patients may be transferred directly to an intensive care unit from an emergency department if required, or from a ward if they rapidly deteriorate, or immediately after surgery if the surgery is very invasive and the patient is at high risk of complications [2].

In 1950, anesthesiologist Peter Safar initiated the concept of "Advanced Support of Life", keeping patients sedated and ventilated in an intensive care environment [3]. Safar is considered to be the first practitioner of intensive care medicine as a specialist. In response to a polio epidemic (where many patients required constant ventilation and surveillance), Bjørn Aage Ibsen established the first intensive care unit in Copenhagen [4].

1.2 ICU present

More than five million Americans with life-threatening conditions are admitted to intensive care units each year. The intensive care unit is a complex, data-intense environment where physiologic signals are generated continuously or intermittently, using devices from a variety of different manufacturers to save patient’s lives [5].

Systemic parameters are monitored including electro-cardio signals, electroencephalogram, blood pressure, heart rate, respiratory rate, and pulse oximetry for days and nights. Critical care has been studies for almost forty years, and the number of monitors has grown exponentially and the development of monitoring technology has grown rapidly. However, the basic information technology architecture remains primordial and clinicians deal with the data essentially the same way [5]; the standard at most medical institutions and hospitals is still recording and logging data manually onto paper records.

Data overload can impede the clinicians' ability to identify important clinical data and may contribute to fatal medical error. There are staggering amounts of data, which are beyond the capability of any medical personnel to absorb, integrate and act upon reliably [6]. Understanding the information requirements of ICU providers will facilitate the development of information systems that prioritize the presentation of high-value data and reduce information overload [7]. Our objective is to develop an acquisition and analytic system to collect and extract physiological data based on the needs of ICU physicians, and integrate this data with the electronic medical record.

Monitors in the Intensive Care Unit display continuous recordings of multiple physiologic signals gathered from the patient [8]. Vital signs displayed simultaneously include cardiovascular data such as heart rate, arterial blood pressure, blood oxygen saturation, temperature, and respiratory data. However, the devices are not interoperable, and none of data is integrated or routinely archived. The inability to integrate and time-synchronize all physiologic data simultaneously into one dataset has been a major limiting factor in health care.

ICU monitoring systems have become increasingly complex, and the data rates necessary to support advanced analysis are the raw data cannot be utilized fully by medical personnel; since their main responsibility is patient care, observing/interpreting the data provided by ICU equipment is not possible [9]. Physiological parameters will usually be interpreted in the context of other physiological parameters together with events that have happened in the past [8]. Now clinicians are forced to do this in their heads: medical staff has to observe all the data generated from various devices and make clinical decisions intermittently.

ICU information systems have limited search capabilities ability because of the massive amounts and disparate data. It is difficult for data to be indexed, searched, and assembled to provide accurate information for medical personnel to take necessary action to treat patients because the original context of the data is lost [10].

While some monitors display raw trends even basic statistical analyses (mean, median, standard deviations) are elusive and nearly impossible to perform [11]. Although technologies in the modern age are sufficiently advanced to support more sophisticated analyses, such capability is unavailable in medical care systems at the bedside. ICU medical staff, therefore, could benefit greatly from a medical data collection and processing system to can assist them in the interpretation of massive and recondite data.

1.3 The ICU of the Future

We believe that the future of intensive care monitoring lies in the following aspects. First of all, the next generation of intelligent monitoring systems should have the functionality of integration and time-synchronization of multiple channels of physiological data continuously and simultaneously.

In addition, the system should be capable of processing physiological data in real-time, and using new dynamic analysis tools such as multivariate analysis and nonlinear time series analysis to facilitate rapid diagnoses and support clinical decision-making.

Last but not the least, the system should exhibit the clinical information in a user-friendly Graphical User Interface to facilitate clinical decision-making. The combination of all three elements, data integration, processing, and visualization, is far beyond the scope of what is commercially available today [5].

The development of the next generation of data acquisition systems requires a coordinated effort involving clinicians, engineers, and experts in informatics and complex biostatistics, and industry to truly move this field of “critical care bioinformatics” forward [6].

2 Protocol Concept

2.1 Definition of the Transport Protocol

The data acquisition system is designed to access the Philips IntelliVue Patient Monitor MP70 based on the programming guide provided by the Philips Healthcare Company.

The protocol is based on a Client/Server Model. The personal computer establishes a logical connection with the monitor. Communication occurs by sending and receiving two kinds of command messages; one is Association Control protocol, and the other is Data Export Protocol. Both the Association Control and Data Export functionalities in the IntelliVue monitor can be accessed via the MIB/RS232 interface.

For the MIB/RS232 interface, two transport protocols are supported: Fixed Baudrate Protocol at 19200 or 115200 baud and the protocol with Auto Speed Baudrate negotiated with the monitor. However, the Fixed Baudrate Protocol provides a transport protocol with minimal overhead and complexity, which is intended for Computer Clients which cannot use the Auto Speed Protocol. The protocol operates at a fixed baudrate and can be used with standard RS232 concentrators. It provides packet-oriented data exchange and checksum protection on top of the RS232 protocol. Therefore, the Fixed Baudrate Protocol a better choice for our application.

2.2 Protocol Model

The protocol is based on an object-oriented modelling concept. All information available through the Data Export Protocol is modelled as attribute values of information objects.

The following information object classes are supported by the IntelliVue monitor:

- **Medical Device System (MDS)**

The MDS object contains attributes representing dynamic state information and static device specific identification information.

- **Alert**

The Alert object contains attributes representing the current technical and patient alarms, as displayed on the IntelliVue monitor.

- **Numerics**

Numeric objects contain attributes representing the state and value of numerical measurements.

- **Waves**

Real-time wave objects contain attributes representing the state and value of wave data.

The object attributes can be accessed by a poll of the MDS object, which allows a query of the sets of attribute values from all objects of a specified class. The method can be called by sending a command message from a Computer Client to the IntelliVue monitor.

2.3 Protocol Dialog

The following diagram shows the protocol dialog between the Philips IntelliVue monitor and the Computer Client. The dialogs are request/response messaging pattern, that is: one party sends the request messages, and the receiving party returns with the response messages. To be more specifically, the Association Request Message sent from the Client will initialize a new communication, if the monitor accepts the association, it returns a MDS Create Event Report Message after a positive Association Result Message. After the association is established successfully, the dialog will enter into the Data Export section as shown. During these interactions, the client will send several Poll Data Request Messages to request for data, and the requested data will be contained in the Poll Data

Response Messages. The dialog can be ended after sending an Association Release Request Message and receiving an Association Release Result.

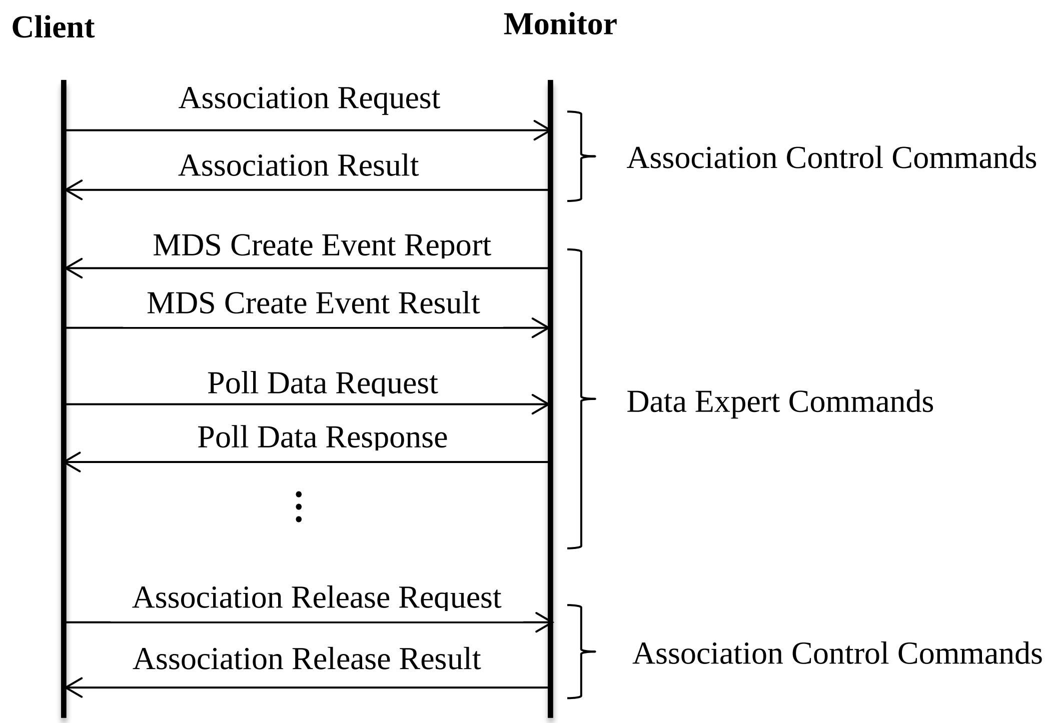

Figure 2-1 Protocol Dialog

Transport protocol-specific messages are not shown in the diagram. The Philips IntelliVue Patient Monitor processes global commands and sends response messages to the requests generated by the Computer Client.

2.4 Definition of the Association Control Protocol

2.4.1 Protocol Command Structure

The Protocol messages conform to the protocols defined by the ACSE Standard (ISO/IEC 8649 and ISO/IEC 8650), with some proprietary extensions, to establish the logical connection between the IntelliVue monitor and a Computer Client.

Protocol Command messages as defined in this section are the data structures that are transported within the transport layer messages. The following commands listed are available to establish a connection between a Computer Client and an IntelliVue monitor.

- **Association Request Message**

The Association Request message is sent from the Computer Client to the IntelliVue monitor when a new association needed to be established. The Association Request message contains information about the requested protocol and protocol options.

- **Association Response Message**

The Association Response message is sent by the IntelliVue monitor to confirm by the client if an Association Request message is received and interpreted successfully and the association is accepted.

- **Refuse Message**

If there is an error contained in Association Request message, or if the association cannot be accepted successfully, for instance, there is already another association in process, the IntelliVue monitor sends a Refuse message to reject the new association.

- **Release Request Message**

If the users intend to put the current association to an end, a Release Request message will be sent from the client to the monitor. The Release Request message does not contain variable data.

- **Release Response Message**

When the IntelliVue monitor receives a Release Request message, it sends a Release Response message as confirmation, and the client needs to parse and check the response messages to confirm the result from the monitor, whether the association release successfully or not. The Release Response message indicates that the association has been terminated.

- **Abort Message**

The Abort message terminates an association with no need for further confirmation because the Abort Message is generated automatically by the IntelliVue monitor for the sake of the system configurations. For example, the IntelliVue monitor sends an Abort message if an association is timed out.

2.4.2 Structure of Association Control Commands

All Association Control Commands share a common structure as shown in the Table 2-1.

Session Header
Session Data
Presentation Header
User Data
Presentation Trailer

Table 2-1 Protocol Commands for Association Control

The Session Headers can be used to identify the protocol commands. Each Session Header type maps to one protocol command. The Session Header occupies the first bytes of the message. The Computer Client can use the pre-defined building blocks for the Session Data, Presentation Header, and Presentation Trailer listed in the Appendix I and Appendix II to conveniently build valid messages. For some messages, the Session Data and the User Data block may be empty. Only the User Data block of the Association Request must be filled with Computer Client-specific data.

2.5 Definition of the Data Export Protocol

2.5.1 Definition Shared By Protocols

2.5.1.1 Byte Order

The protocol data structures use the Network Byte Order. This means that data are transmitted on the network with the most significant byte first. This may or may not match the order in which numbers are normally stored in memory for a particular processor. If the Computer Client is not using big-endian storage internally, for example, many common Personal Computer Platforms use little-endian storage, protocol data structures must be transformed before they are sent to an IntelliVue monitor or after they have been received from an IntelliVue monitor.

2.5.1.2 Byte Alignment

The Association Control protocols and Data Export protocols presume that there is no data alignment in data structure. However, most data types used have an even length for the reasons to increase the system’s performance. Many compilers have their own different default alignment settings. It is worth stressing that, the compiler in the application developed for this monitor, needs to use the right alignment when parsing and formatting protocol messages.

2.5.1.3 Bit Order

Bit order refers to the direction in which bits are represented in a byte of memory. The index for bits starts with zero for the most significant bit (MSB), the bit position having the greatest value, and ends with 15 for the least significant bit (LSB). Figure 2-2 shows the bits arranged into two bytes.

MSB														LSB	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Table 2-2 Bit Order of Two-Byte Data

2.5.1.4 Common Data Types

- **Basic Data Types**

The mapping of these types to data types used in a Computer Client application is machine specific and compiler dependent. The C language provides many basic types, and most of them are formed from one of the four basic arithmetic type specifiers (char, int, float and double), and optional specifiers (signed, unsigned, short, long). The C data types defined here make use of the following basic types:

u_8	unsigned 8 bit wide integer
u_16	unsigned 16 bit wide integer
u_32	unsigned 32 bit wide integer
i_8	signed 8 bit wide integer
i_16	signed 16 bit wide integer
i_32	signed 32 bit wide integer

Table 2-3 Basic Data Types of the C Language

- **Absolute Time**

The AbsoluteTime data type is used whenever data is time stamped and a temporal resolution of 1s is sufficient. The individual u_8 fields are encoded in BCD, they are not encoded as regular integer values. For example, the year 99 (decimal) is coded as 0x99. One detail need to be mentioned, an invalid time will indicated with “0xff” in all assigned byte fields.

The time resolution in the IntelliVue monitor with this format is 1 second. The sec-fractions element in the structure is not used.

```
typedef struct {
        u_8    century;
        u_8    year;
        u_8    month;
        u_8    day;
        u_8    hour;
        u_8    minute;
        u_8    second;
        u_8    sec-fractions;
} AbsoluteTime;
```

- **Relative Time**

The RelativeTime is like a high-resolution time label that maps a time relative to an event. It is used to place an event, like a particular event message, relative to each other with higher resolution. It is defined as follows:

```
typedef        u_32    RelativeTime;
```

The resolution of RelativeTime is 1/8ms, with a precision of 2 milliseconds. According to a known relation between AbsoluteTime and RelativeTime with a precision of about 1s, the Computer Client should have the capability to calculate the absolute time from RelativeTime.

- OID Type

The OIDType, which stands for Object Identifier Type, is a byte field that can identify all protocol elements.

typedef u_16 OIDType;

For different OIDType, for example, physiological identifiers, alert condition identifiers, units of measurement, etc, they have independent value ranges.

- **Private OID**

The PrivateOID type, which stands for Private Object Identifier, is a byte field that can identify of private or manufacturer specific elements.

typedef u_16 PrivateOID;

Values for the PrivateOIDs are listed whenever a PrivateOID is used.

- **TYPE**

Whenever the nomenclature value range of the OIDType is not clear from the context, the TYPE data type is used. Here, unlike the OIDType and PrivateOID, the nomenclature value range is explicitly identified, which means the code is only unique in a given partition. The data structure of the OIDType are defined in as shown.

```
typedef         u_16    NomPartition;
    typedef struct {
    NomPartition        partition;
    OIDType             code;
} TYPE;
```

- **Handle**

Object instances is defined in a 16 bit wide ID.

typedef u_16 Handle;

- **Global Handle**

The Protocol adopt by the monitor can support multiple measurement servers, where each measurement server assigns object handles uniquely and independently. To make sure the uniqueness of the handles within the context of a particular system, the Global Handle (GlbHandle) contains an additional identifier for the source system, such as, each measurement server will assign a distinct context id for identification. The context id is assigned dynamically and automatically when a measurement server is connected.

```
typedef              u_16    MdsContext;
typedef struct {
MdsContext    context_id;
Handle        handle;
} GlbHandle;
```

- **Managed Object Identifier**

The Managed Object Identifier is a byte field to play the role of identification of an object that contains an identifier for the object class, for instance, Alert object, Numerics object and Wave object, together with a Global Handle.

```
typedef struct {
        OIDType      m_obj_class;
        GlbHandle    m_obj_inst;
} ManagedObjectId;
```

- **Attribute Value Assertion**

Object attributes are presented in the way of data structures that contain an identifier, a length field and the actual value of the attribute.

The data structure of such an attribute names the Attribute Value Assertion, which is defined as follows:

```
typedef struct {
        OIDType   attribute_id;
        u_16      length;
        u_16      attribute_val;
} AVAType;
```

Specifically, the attribute_id identifies the type of the each different attribute; the length field contains the size of the attribute_val field in bytes; the attribute_val field is only a placeholder in this structure, which does not contain actual data. Hence, the parsing algorithm in the application must specify the attribute value to the data structure correctly.

- **Attribute List**

In a typical manner, object instances have multiple attributes that are assigned in a list with the following data type: byte fields of count, length and the real value.

```
typedef struct{
        u_16      count;
        u_16      length;
        AVAType   value[1];
} AttributeList;
```

Specifically, the count field contains the number of Attribute Value Assertion elements in the list and the length field contains the size of the list in bytes. The value field itself is only a placeholder data structure, which does not contain actual data, and the parsing algorithm in the client need to interpret the data structure correctly.

- **Variable Label**

The data structure is preceded by a length field, followed by the observed value of VariableLabel. The VariableLabel data type is encoded in 8-bit ASCII characters for the text, and the length is always even.

```
typedef struct {
        u_16    length;
        u_8     value[1];
} VariableLabel
```

- **TextId**

The TextId type is a 32-bit wide private ID.

```
typedef         u_32    TextId;
```

- **FLOAT Type**

For floating point numbers, a 32-bit wide format is used.

```
typedef         u_32    FLOATType;
```

FLOATType is interpreted as shown in Figure 2-4:

MSB			LSB
0 7	8 15	16 23	24 31

exponent 8 bits, signed   mantissa 24 bits, signed

Figure 2-2 Bit Order of FLOAT Type Data

The number of the FLOATType can be calculated by (mantissa)*(10exponent) with the exponent and mantissa in 2's complement.

2.5.2 Protocol Command Structure

Protocol Command messages are data structures transported within the transport layer message, such as UDP datagram, or IrDA and Fixed Baudrate Protocol messages, and have common structure for messages sent from the Computer Client to the IntelliVue monitor.

The Protocol Command messages represent the ISO/OSI layers 5 – 7 (session, presentation, and application layers). The message that transports a Protocol Command contains a checksum that should be validated by the Computer Client to detect corrupted messages.

The Protocol Command messages, that establish the logical association between the IntelliVue monitor and a Computer Client, conforms the Association Control Service Element (ACSE) Standard (ISO/IEC 8649 and ISO/IEC 8650). For Protocol Commands have message structure with the following basic format:

Session/Presentation Header
Remote Operation Header
Command Header
Command-Specific Parameter Data

Table 2-4 Basic Format of Message Structure

2.5.2.1 Session/Presentation Header

During the entire of the connection between the IntelliVue monitor and the Computer Client, the Session and Presentation Headers are set to fixed values in order to distinguish different association. Each protocol message starts with a common data structure constituting the session and presentation protocol. The first byte field of the session_id can be utilized by the Computer Client to identify between Data Export protocol commands and Association Control protocol commands.

```
typedef struct {
        u_16    session_id;         /* contains a fixed value*/
        u_16    p_context_id;       /* negotiated in association phase */
} SPpdu;
```

2.5.2.2 Remote Operation Header

The algorithm in the Computer Client uses the byte field of Remote Operation Header to distinguish between the different types of command messages, command response messages and error messages.

```
typedef struct {
        u_16        ro_type;        /* ID for operation */
        u_16        length;         /* bytes to follow */
} ROapdus;
```

ro_type	Description	Value
ROIV_APDU	invokes a remote operation	1
RORS_APDU	returns the result of a remote operation	2
ROER_APDU	returns an error for a remote operation	3
ROOLRS_APDU	returns parts of the result of a remote operation	5

Table 2-5 Types of Different Remote Operation

- **Remote Operation Invoke**

A Remote Operation Invoke message is defined as follows:

```
typedef struct {
        u_16          invoke_id;        /* identifies the transaction */
        CMDType       command_type;     /* identifies type of command */
        u_16          length;           /* no. of bytes in rest of message */
} ROIVapdu;
```

- **Remote Operation Result**

To respond the Operation Invoke message, a Remote Operation Result message can be used as a confirmation. The data structure of the Remote Operation Result message is defined in the data structure as follows:

```
typedef struct {
        u_16          invoke_id;        /* mirrored back from op. invoke */
        CMDType       command_type;     /* identifies type of command */
        u_16          length;           /* no of bytes in rest of message */
} RORSapdu;
```

- **Remote Operation Linked Result**

In some cases, if the Client requests for big size data, total data returned by the monitor may exceed the maximum message size defined by the monitor, hence the multiple Remote Operation Linked Result messages are used in these cases. That is to say, one big data may break down into several data packets according to size of the data. These data structures are defined as follows:

```
typedef struct {
```

```
        RorlsId      linked_id;       /* see below */
        u_16         invoke_id;       /* see below */
        CMDType      command_type;    /* identifies type of command */
        u_16         length;          /* no of bytes in rest of message */
} ROLRSapdu;
```

- **Remote Operation Error**

An error message will be returned from the monitor if an error is detected at the Remote Operation. Different errors can be identified by the invoke_id value.

```
typedef struct {
        u_16         invoke_id;
        u_16         error_value;
        u_16         length;
} ROERapdu;
```

2.5.2.3 Command Header

The Command Header contains the common part of the Command data structure identified in the Remote Operation Header. In each protocol message, a Command data structure is attached. The value of the CMDType field, which purpose is to identify the Remote Operation Invoke, Result and Linked Result, are shown in Table 2-6.

The IntelliVue monitor has the message mechanism of resending the message, if the result messages are not received successfully within 3 seconds. After resending the message three times, the association will be aborted by the IntelliVue monitor automatically.

CMDType	Description	Value
CMD_EVENT_REPORT	is used for an unsolicited event message	0
CMD_CONFIRMED_EVENT_REPORT	The confirmation to the event report	1
CMD_GET	Request attribute values of managed objects	3
CMD_SET	Set values of managed objects.	4
CMD_CONFIRMED_SET	Set attribute values of managed objects	5
CMD_CONFIRMED_ACTION	Invoke an activity on the receiver side	7

Table 2-6 CMD Types

2.5.2.4 Command Structure Summary

Table 2-7 shows a summary of command structure from which all the request and response messages are constructed.

SPpdu						
ROapdus						
ROIVapdu			RORSapdu/ROLRSapdu			ROERapdu
Event Report Argument	Action Argument	Get Argument	Event Report Result	Action Result	Get Result	Error Data
Event Data	Action Data	Set Argument	Event Result Data	Action Result Data	Set Result	

Table 2-7 Summary of Command Structure

2.5.3 Protocol Commands

The Protocol Commands are constructed based on the data types defined previously in section 2.5.2.4. A special marker is used to define how the command messages are constructed and generated because application must parse the individual elements of a command message separately.

2.5.3.1 MDS Create Event

The MDS Create Event is an important message that contains both the software and hardware configurations of the IntelliVue monitor. The Computer Clients needs to parse this message correctly to learn about the system configurations.

The MDS Create Event message has the structure as follows:

MDSCreateEventReport ::=

<SPpdu>

<ROapdus (ro_type := ROIV_APDU)>

<ROIVapdu (command_type := CMD_CONFIRMED_EVENT_REPORT)>

<EventReportArgument

(managed_object := {NOM_MOC_VMS_MDS, 0, 0},

event_type := NOM_NOTI_MDS_CREAT)>

<MDSCreateInfo>

The MDS Create Information uses the C type data structure definition as follows:

```
typedef struct {
ManagedObjectId     managed_object;
AttributeList       attribute_list;
} MdsCreateInfo;
```

2.5.3.2 MDS Create Event Result

The MDS Create Event Result is a confirmed operation corresponding to the MDS Create Event message. The Computer Client parses the MDS Create Event Result to confirm the configuration of the monitor.

The reply message has the structure as follows:

MDSCreateEventResult ::=

<SPpdu>

<ROapdus (ro_type := RORS_APDU)>

<RORSapdu

(invoke_id := mirrored from event report,

command_type := CMD_CONFIRMED_EVENT_REPORT)>

<EventReportResult

(managed_object := mirrored from event report,

event_type := NOM_NOTI_MDS_CREAT)

length := 0 >

2.5.3.3 Single Poll Data Request

Single Poll Data Request message can be sent by the Computer Client once the Association is established and accepts by the monitor. What's more, the MDS Create Event message and the MDS Create Event Result message sequence need to be completed. A method that returns IntelliVue monitor device data in a single response message is called by the message.

The message has the following structure:

MDSPollAction ::=

<SPpdu>

<ROapdus (ro_type := ROIV_APDU)>

<ROIVapdu (command_type := CMD_CONFIRMED_ACTION)>

<ActionArgument

(managed_object := {NOM_MOC_VMS_MDS, 0, 0},

action_type := NOM_ACT_POLL_MDIB_DATA)>

<PollMdibDataReq>

The number of incoming SINGLE POLL DATA REQUEST messages is limited by the IntelliVue monitor according to the maximum frequency and the size of each message. If the Computer Client sends messages with a frequency above the limit set in the configuration of the monitor, some of the messages will be discarded with no response is sent. The IntelliVue monitor will process a maximum of one POLL DATA REQUEST messages for each object type per second. An additional POLL DATA REQUEST for Numeric Observed Values is allowed.

2.5.3.4 Single Poll Data Result

Single Poll Data Result message is sent by the IntelliVue monitor in response to the Single Poll Data Request.

The message has the structure as follows:

MDSPollActionResult ::=

<SPpdu>

<ROapdus (ro_type := RORS_APDU)>

<RORSapdu (invoke_id := "mirrored from request message"

command_type := CMD_CONFIRMED_ACTION)>

<ActionResult

(managed_object := {NOM_MOC_VMS_MDS, 0, 0},

action_type := NOM_ACT_POLL_MDIB_DATA)>

<PollMdibDataReply>

2.5.3.5 Extended Poll Data Request

The Extended Poll Data Request allows the Computer Client to get access to the following types of data, such as numerics data, wave data, and alarm data. What’s more, Extended Poll Data Request also allows periodic Poll Replies with no need to send a Poll Request every time. The Computer Client enables periodic Poll Replies by setting timers for each type of data requested. The Extended Poll Data Request message allowed if the

Poll Profile Extensions optional package has been negotiated during the association phase, which refers to the Appendix II.

- **Accessing 12 second, 1 minute and 5 minute averaged Numeric data**

Within the Poll Profile Extensions optional package, the Computer Client and the IntelliVue monitor have negotiated which data source; either real-time data or averaged data is used to obtain the Numeric data. For the current version of system configuration, the IntelliVue monitor only allows the Computer Client to specify one data source for Numeric data.

The IntelliVue monitor shows a response or a reaction to the Extended Poll Data Request message with an Extended Poll Data Result message that contains the Numeric data from the source negotiated and specified in the Poll Profile Extensions optional package. The normal Poll Data Request message always returns data from real-time measurements.

- **Time Periodic Data Poll**

The Time Periodic Data Poll attribute is byte field reserved to afford the Computer Client to request periodic Poll Replies for a given time according to different type of data. If the Computer Client specifies the Time Periodic Data Poll attribute to the Extended Poll Data Request message, the IntelliVue monitor sends periodic Extended Poll Data Result messages automatically for the time specified according to different data type.

When the IntelliVue monitor receives an Extended Poll Data Request message, the first result message is sent immediately as a confirmation with the sequence number zero (see below), indicating to the Computer Client that its request was successful. The following messages are sent with the period specified in Table 2-8 below.

Data Source	Resulting Period
real-time waves	256ms
real-time measurements	1s
12 second averaged data	6s
1 minute averaged data	30s
5 minute averaged data	150s
alarm data	1s

Table 2-8 Time of Periodic Data Poll for All Type of Data

Before the time specified in the Time Periodic Data Poll attribute has expired, the Computer Client should send a new Extended Poll Request to keep the association alive; otherwise the monitor will abort the association because of the time out. As illustrated in Figure 2-3 below, taken the basic period of the reply 1 minute averaged data as an example, each new Extended Poll Request will be confirmed with an Extended Poll Result message immediately.

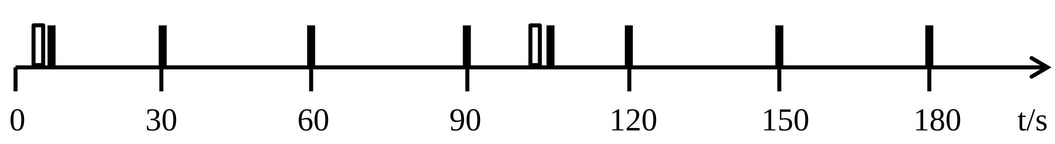

Figure 2-3 Basic Period of Replies

- **Limiting the Number of Objects in the Poll Result**

If the IntelliVue monitor is connected to a large number of measurement modules, a Poll Request for numerics will lead to a large amount of data being sent from the IntelliVue monitor, so the Computer Client must have the mechanism to limit the number of objects contained in a Poll Result Message. An internal priority table in the IntelliVue monitor determines which objects could be contained to the Poll Result. The priority table is constructed by the algorithm in the backend and if the system configuration changes, the system may need up to two minutes to respond and update the priority table. During this transition phase, the requested number of objects contained in the Poll Results sent by the monitor may contain less than regular number during the normal phase.

2.5.3.6 Extended Poll Data Result

When the IntelliVue monitor receives an Extended Poll Data Request message, it responds with a single or periodic Extended Poll Data Result messages. The parsing algorithm of the Computer Client will extract the data specified in the Extended Poll Data Request message and interpret the data according to the data structure of each kind of data.

2.5.3.7 Specifying Objects in the Poll Result

The Get and Set operations can be used to specify wave or numeric objects to be requested within the Poll Results. If no user defined priority list is detected, the monitor will return Extended Poll Result message according to the default priority list contained is the internal priority table and depends on the current system configuration. If there is a new user defined priority list, which replace the default priority list, more purposeful wave data or numerics data can be returned.

The Table 2-9 lists all the type of messages corresponding to the priority list and its functions.

Message Type	Description
Get Priority List Request	specifies multiple wave or numeric objects
Get Priority List Result	contains the requested attribute identifiers and values
Set Priority List Request	modifies a wave object priority list attribute
Set Priority List Result	returns the modified object priority

Table 2-9 Priority list Request and Response Message

3 Data Acquisition System

This chapter describes a real-time bedside acquisition system for physiological for intensive care units. The aim of this system is to collect and store real-time physiological data from patient monitors as a first step to providing complex data analytics that supports clinical decision-making.

3.1 Motivation

The ICU bedside decision support system requires the acquisition of high-resolution physiological data from the patient monitor that can then be analyzed and presented to clinical personnel for interpretation and decision-making. The interpretation of multi-modal physiological data is a nontrivial task that requires skilled and professionally trained clinical personnel to make reasonable diagnoses about the state and trajectory of a patient.

Previous studies have shown that physiological data contains important information, which not only indicates the current state of the patients, but is also important for assessing the interconnections between different organ systems and predicting the future state [12]. The dynamic changes in the physiological state of patients are more than isolated events, and are associated with organ system interconnections. Hence, the key to the design of the ICU bedside data acquisition system is to collect the physiological data, extract relevant clinical information from the multi-modal signals, quantify interactions between physiological subsystems, and identify important event information. We designed the data acquisition system with the primary goal of efficiently collecting high-resolution real-time physiological data to support future ICU bedside system development efforts.

3.2 Method

3.2.1 Overview

The real-time data acquisition system stated in the thesis is part of the Integrated Medical Environment being developed at University Hospitals Case Medical Center and Case Western Reserve University. The Integrated Medical Environment, as shown in Figure 3-1, is an open source architecture that provides the main frame for the ICU in the future, which could concisely be divided to three parts: (1) real-time data acquisition, integration and time-synchronization of physiological data; (2) a middleware informatics architecture that facilitates complex systems analysis methods and information extraction for hypothesis generation and testing; (3) a user-friendly graphics interface, which presents the processed information [6].

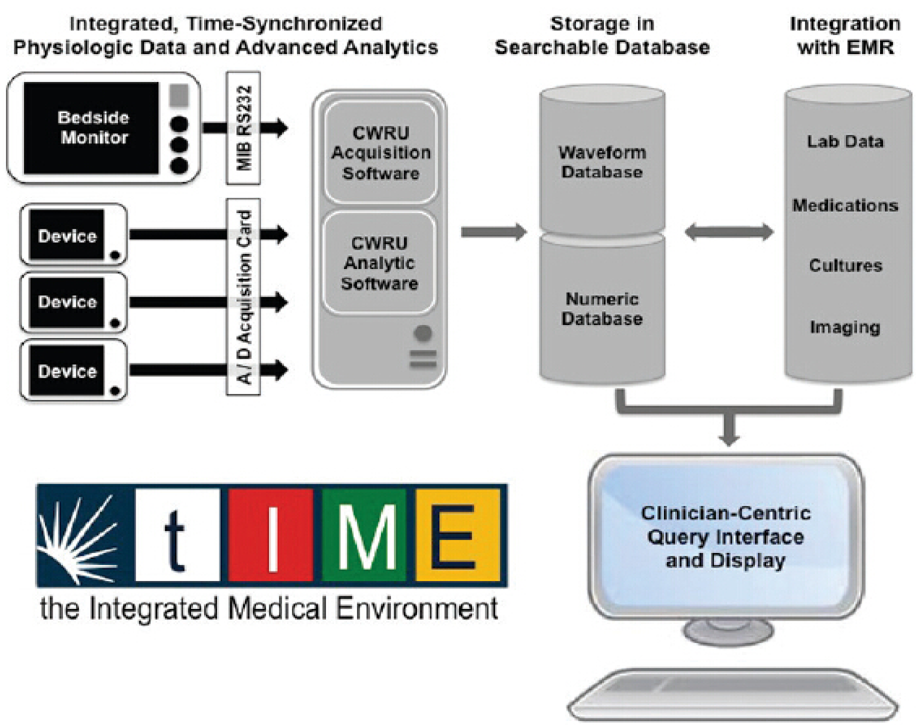

Figure 3-1 Schematic Illustration of the Integrated Medical Environment [6]

Real-time data acquisition at the bedside is the cornerstone of the Integrated Medical Environment. The prototype system developed in this work communicates with the MP70 Philips IntelliVue monitor in any intensive care unit to acquire physiological data via RS232 interface. In our implementation, the data acquisition software runs on a personal computer with a serial connection to the monitor, to manage the communications and acquisition and storage of all waveforms, numerics and alarm data from the monitor. The software includes three modules: the serial communication module, the data acquisition and processing module, and the data storage module.

3.2.2 System Components

3.2.2.1 Hardware

The hardware of the data acquisition system includes the Philips IntelliVue patient monitor, a computer with RS232 serial port and a RJ45 to RS232 cable. The IntelliVue monitor MIB/RS232 interface provides an eight-pin RJ-45 modular jack. The RS-232 connector of the MP70 Series monitor is shown in Figure 3-2 below.

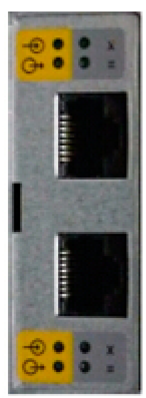

Figure 3-2 The RS-232 Connector of MP70 Series

- RJ45 to RS232 cable

The PC is connected to the MP70 monitor via RS232 using the cable configuration shown in Figure 3-3 below. The pins of the RJ45 connector are numbered from 1 for the highest pin to 8 for the lowest pin looking directly at the pins with the cable leaving the connector to the left.

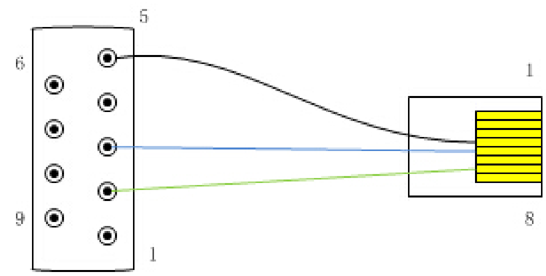

Figure 3-3 The Cable Configuration of RJ45 to RS232 Cable

The MP70 RS232 interface provides an RS232 port with the following pin assignment as shown in Table 3-1.

Computer Client	Pin and Signal Direction	IntelliVue monitor
GND	4<=>	GND
TxD	5 =>	RxD
RxD	7 <=	TxD

Table 3-1 Pin Assignment of RS232 Port

The TxD is the receive line and the RxD lines is the transmit line. The signals are referenced to the ground (GND).

3.2.2.2 Software

The function of the data acquisition module is to acquire real-time physiological data from the patient monitor using serial communications and save the extracted data into a hard disk, the data processing module enables the system to parse and interpret the data streaming from the monitor.

The design of software has been implemented using the MFC tools in Visual C++ 6.0 in the Windows XP operating system. The main programming processes are illustrated in Figure 3-4:

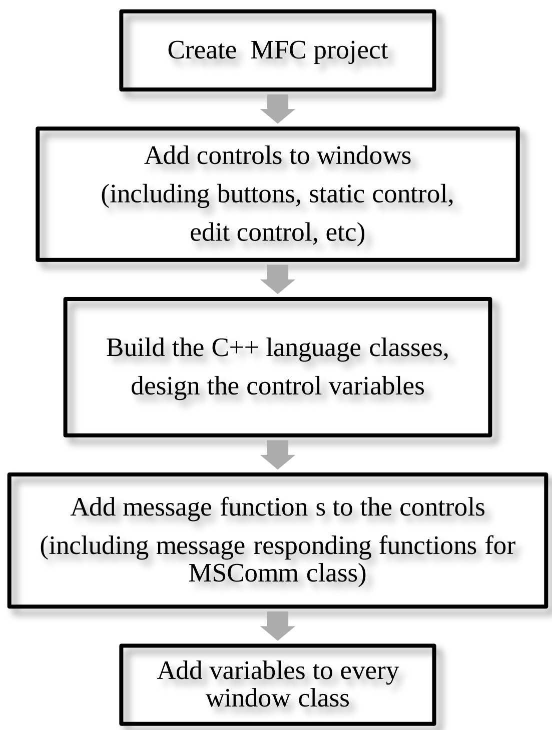

Figure 3-4 Programming Process of Data Acquisition Software

3.2.3 Serial Communication Module

The function of the serial communication module is to connect to the MIB/RS232 Interface using the Fixed Baudrate Protocol and to establish serial communication with the monitor. Most computers have serial communication ports COM1 to COM3 installed, and all available serial ports can be accessed using Visual C++ interface applications.

3.2.3.1 Introduction to ActiveX Controls

The ActiveX serial component is a software development kit that enables developers to establish a serial communication interface over various devices.

- **The Class Hierarchy of the ActiveX Component**

The ActiveX control functionality is encapsulated in a MFC class called COleControl. ActiveX control inherits all the functionality of the base class COleControl, and the class hierarchy of ActiveX controls is shown in Figure 3-5.

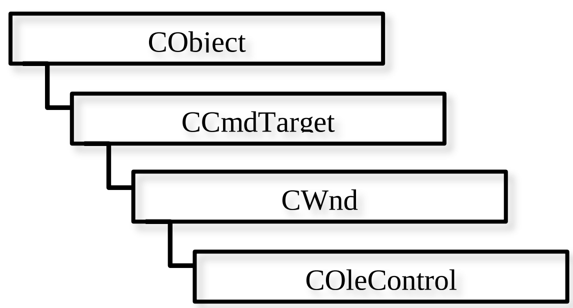

Figure 3-5 The Class Hierarchy for COleControl [13]

The ActiveX Controls have become the primary architecture for developing programmable applications, and the core is the Component Object Model (COM). ActiveX Control components can edit the COM interface by adding available Methods, Properties and Events, where the application accesses the ActiveX Controls via the interface provided by the ActiveX Controls.

Microsoft Communication Control (MSComm control) supports serial communications for the application by allowing the transmission and reception of data through a serial port. Each MSComm control can only correspond to one serial port. There are two

MSComm control methods for handling communication: the first one is event-driven communication, and the other is the polling method.

Event-driven communications is a very powerful method for handling serial port interactions. In many situations you want to be notified the moment an event takes place, such as when a character arrives or a change occurs in the Carrier Detect (CD) or Request To Send (RTS) lines. In such cases, the MSComm control OnComm event is used to trap and handle these communications events. The OnComm event also detects and handles communications errors.

3.2.3.2 Programming flowchart

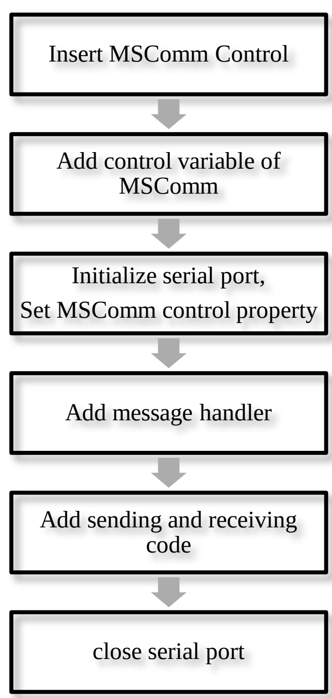

Figure 3-6 Programming Flowchart of Establishing Serial Communication

3.2.3.3 Properties of MSComm control

MSComm provides functionality for setting serial communication parameters such as the port number, Baud rate, parity, data bits, stop bits, handshaking and so on [14]. The method to establish serial communication is using parameter settings and related functions of the MSComm class in Visual C++, including CommPort, settings, PortOpen, Input, Output, OnComm, CommEvent, etc. The properties of MSComm control are listed in the Table 3-7.

Properties	Description
CommPort	Sets and returns the communications port number
Settings	Sets and returns the baud rate, parity, data bits, and stop bits as a string
PortOpen	Sets and returns the state of a communications port. Also opens and closes a port
Input	Returns and removes characters from the receive buffer
Output	Writes a string of characters to the transmit buffer

Figure 3-7 Properties of MSComm Control

3.2.4 Data acquisition and processing module

3.2.4.1 Main Interface

The main interface, shown in Figure 3-9, includes functionality for Serial Port Selection, Data Type Selection, Signals Selection, Start-Stop, and the Output Display Section.

- **Class Hierarchy of the CButton Control, CEdit and CCombo Box**

Button control is a small, rectangular child window that can be clicked on and off. Buttons can be used alone or in groups and can either be labeled or appear without text. A button typically changes appearance when the user clicks it.

Typical buttons include the check box, radio button, push button and combo box. A CButton object can become any of these, according to the button style specified at initialization by the Create member function. Figure 3-8 shows the class hierarchy of the CButton Control, CEdit control and CCombo Box control.

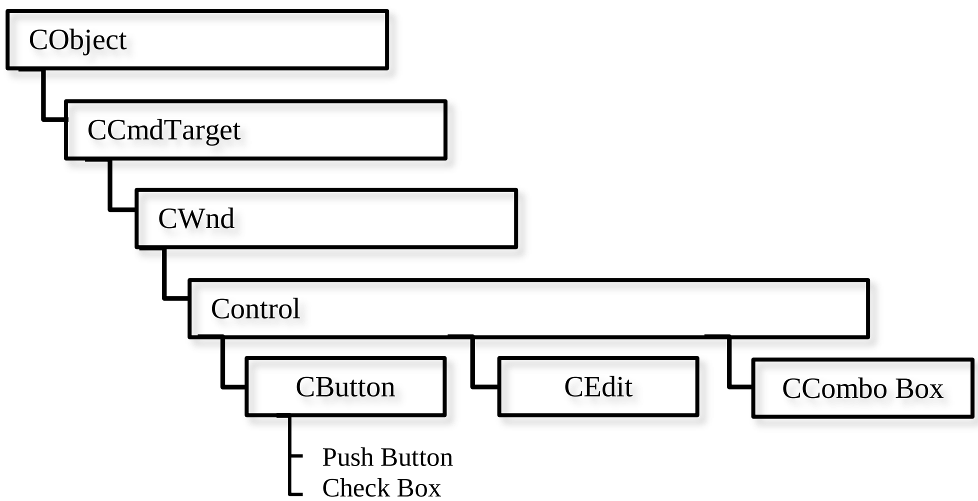

Figure 3-8 The Class Hierarchy for Cbutton, CEdit and CCombo Box [13]

- **Serial Port Selection**

Serial Port Selection is used to set initialization parameters for serial port communication. After the serial port been set, the handler of MSComm Control will transfer the parameters to messages to initialize the serial communication.

The Combo Box control "COM" is a drop-down list of all available serial Com ports from COM 1-COM 3. The Combo Box "Rate" is a drop-down list that includes the 2 available baudrate settings supported by the monitor, 19200 and 115200.

- **Data Type Selection**

The Data Type Selection is used to decide which kind of data will be polled in the Poll Data Request. The backend will be automatically enabled or disabled when the checkbox is checked or unchecked.

- **Signals Selection**

Wave data is chosen in Signals Selection when wave signals are to be acquired.

- **Start-Stop-Exit Button**

The Start button is used to initialize the application; the Stop button can pause the program and suspend the communication process, and pushing the Start button will turn the Stop button gray meaning that the Start button will disable the Stop button and vice versa. The Exit button is used to terminate serial communications and quit the application.

- **Output Display Section**

This display section is used to display all the commands, including the requests that are sent and the results that are received. It works like a printout log, to show whether the messages are sent correctly, or what kinds of messages are received.

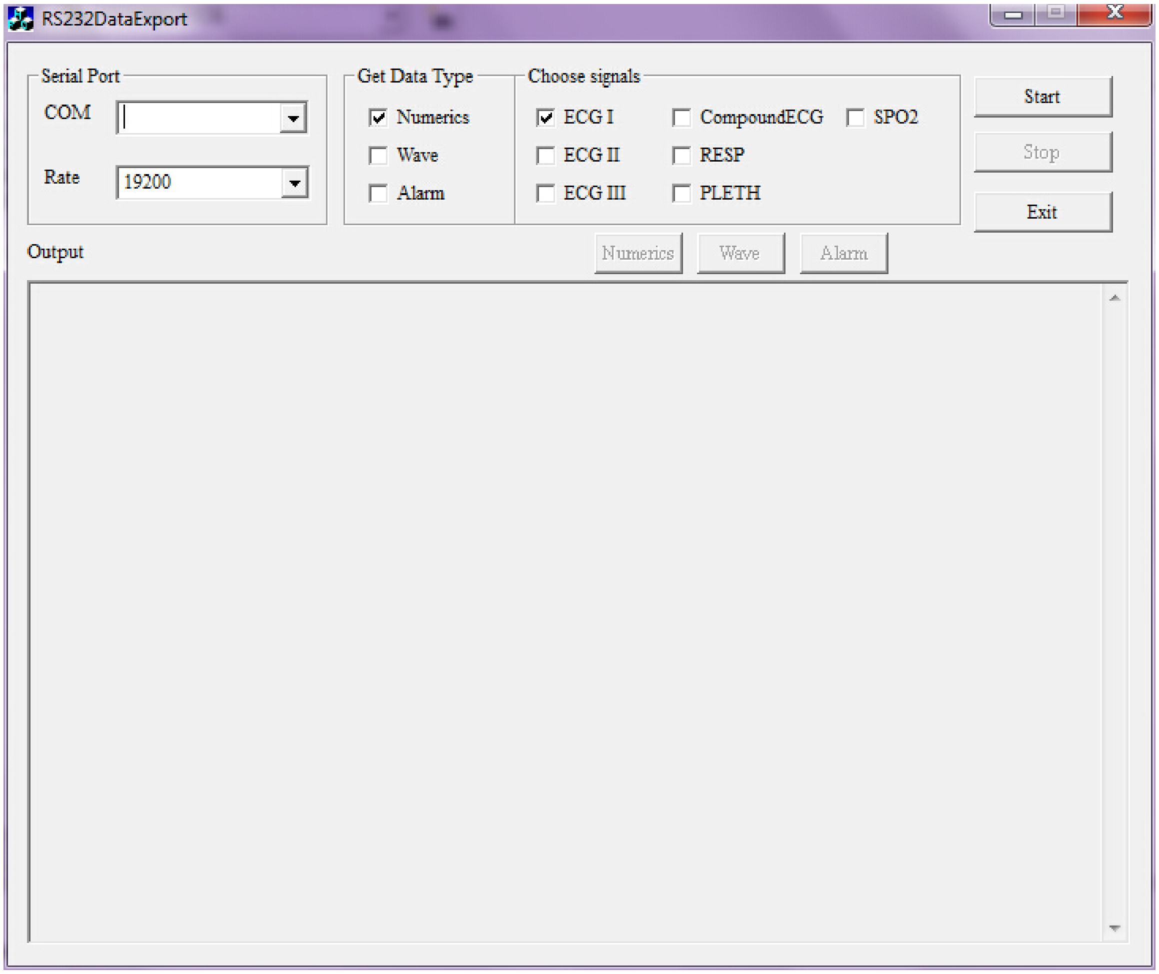

Figure 3-9 Main Interface

3.2.4.2 Single and Extended Polling

The Single Poll and Extended Poll Requests are used to access different types of data in the IntelliVue monitor. Those messages should be sent after the logical connection is established and the MDS Created Event/Reply message is finished.

If the application needs to access real-time numerics or wave data, it uses the Extended Poll Data Request command, which avoids sending poll requests with high frequency and thus reduces the communications that are required.

Considering the response time of the monitor, as listed in Table 3-2, the time span until all the requested data is collected depends on the internal update frequency of the data. That's to say, not all the data is available right after the association has been established.

Object Type	Max Time
Numerics(real-time)	< 2s
Numerics(12 second averaged)	< 18s
Numerics(1 minute averaged)	< 70s
Numerics(5 minutes averaged)	< 310s
Alarms	< 2s

Table 3-2 Response Time of the Monitor for Different Data Types

3.2.4.3 Parsing the Poll Result

A Poll Request message leads to an immediate response message from the monitor called a Poll Data Result. The Poll Data Result message is a transport message that includes a checksum that needs to be verified by the application. If the checksum is not correct, the message should be ignored.

3.2.4.4 Parsing Attributes

When parsing the attributes, the first thing to check is the length field. The length field denotes the length of the data appended to the message, excluding the size of the length field.

If the Computer Client fails to parse the message, a useful strategy is to compare the original raw data with the parsed data. If the Computer Client is using a different byte order and/or structure alignment these could result in inaccurate parsing of the attributes. Different alignment algorithms could result in the compiler inserting redundant bytes, which would cause the wrong interpretation of data structure.

3.2.4.5 Interpreting Numeric Data

The numeric data contains several important physiological parameters that need to be interpreted.

- **The Absolute Time Stamp**

The absolute time stamp defines a time tag for the current numeric value in the IntelliVue monitor. This attribute contains century, year, month and day as mentioned in the Common data type.

- **Numerics Observed Value**

The Numerics Observed Value attribute represents the measured value, along with state and identification data. Each observed value of numeric data has the same data structure.

The NuObsValue data type is defined as follows:

```
typedef struct {
        OIDType               physio_id;
        MeasurementState      state;
        OIDType               unit_code;
        FLOATType             value;
} NuObsValue;
```

The physio_id field identifies the physiological measurement. The unit_code field identifies the units of measure. The state field is a bit field structure defined by the monitor. The measurement is valid if the first octet of the state is all 0. Only if the state indicates a valid measurement, should the value field be interpreted. The value field is a floating point number.

- **Compound Numerics Observed Value**

The Compound Numerics Observed Value attribute represents multiple measured values contained in one Numeric object, along with state and identification data. The Compound Numerics Observed Value represents a blood pressure measurement by providing systolic, diastolic and mean values in a single Numeric object.

The NuObsValueCmp data type is defined as follows:

```
typedef struct {
        u_16 count;
        u_16 length;
        NuObsValue value[1];
} NuObsValueCmp;
```

3.2.4.6 Interpreting Alarm Data

The Alarm object represents the overall device alert condition and contains the global status and a list of active technical patient alerts. There are two groups of alarms available in the IntelliVue monitor; the active technical alarm (T-Alarm) and the active patient alarm (P-Alarm). Both the T-Alarm and P-Alarm share the same alarm data structure.

- **Device T-Alarm/ P-Alarm List**

The DevAlarmList is defined as follows:

```
typedef struct {
        u_16            count;
        u_16            length;
```

```
        DevAlarmEntry       value[1];
    } DevAlarmList;
```

The DevAlarmEntry is defined as follows:

```
    typedef struct {
        OIDType             al_source;
        OIDType             al_code;
        AlertType           al_type;
        AlertState          al_state;
        ManagedObjectId     object;
        PrivateOid          alert_info_id;
        u_16                length;
    } DevAlarmEntry;
```

3.2.4.7 Interpreting Wave Data

The data acquisition application can poll various combinations of wave signals, such as ECG lead I, II, III, Respiration and Pleth. The IntelliVue patient monitor supports the following wave types, which are defined by sample period, sample and array size, update period and bandwidth requirement, as shown in Table 3-3.

The application can poll up to three ECG waves simultaneously with sample rate at 500 samples per second for each signal by selecting the appropriate lead labels in the Wave Object Priority list. All the polled wave signals can be identified by their physiological identifier.

In addition, the application can poll a single compound ECG wave, which contains three channels with sample rate at 250 samples per second for each packet.

Besides the capability to poll up to three individual ECG waves with 500 samples/s each or the single Compound ECG wave, the application can select up to eight 125 samples/s or 62.5 samples/s waves simultaneously by selecting the appropriate lead labels in the Wave object priority list.

Wave Type	Sample Period	Sample Size	Array Size	Update Period	Bandwidth Requirement
500 samples/s (ECG)	2ms	16 bits	128 samples	256ms	1064 bytes/s
250 samples/s (Compound ECG)	4ms	16 bits	3*64 samples	256ms	1640 bytes/s
125 samples/s	8ms	16 bits	32 samples	256ms	296 bytes/s
62.5 samples/s	16ms	16 bits	16 samples	256ms	168 bytes/s

Table 3-3 Detail of Wave Data Supported by Monitor

3.2.5 Data Storage Module

3.2.5.1 Storage of Raw Data

After serial communication is established, the data acquisition system is initialized to receive data, although the monitor may be unable to return all the requests immediately due to the difference response times of the data. The system will automatically generate a txt file named “data.txt” to write the data in ASCII characters. After the data are written to a file, even during real-time data collection, the file can be accessed by other data analysis tools.

3.2.5.2 Storage of Parsed Data

The parsing algorithm instantly processes the raw data, the system reads the raw data from the buffer frame by frame and parses it immediately. The parsed data will be saved in the file “data_result.txt” in decimal integers.

4 Results and Discussion

4.1 Results

We have realized the data acquisition system that can collect real-time physiological data from the bedside monitor, parse the raw data and archive the extracted data for Intensive Care Unit applications. What's more, the archived data can support further related data analysis to support clinical decision-making at the bedside.

In order to evaluate the data acquisition system objectively, we ran the system on a DELL computer, Intel(R) Core(TM) 2 Quad CPU, 2 GB of RAM, 240 GB hard disk storage. It has the capability to collect real-time physiological data from the Philips IntelliVue MP 70 monitor via the serial communication during different durations of time from 1 hour to 12 hours.

The Philips IntelliVue Patient monitor with a touch screen displays all the waveforms and numeric data, as shown in the Figure 4-1. It supports both graphical and numerical data formats, juxtapose real-time measurements and trended data, and onscreen elements from waveforms to data labels can be organized as desired [15]. Specifically, in region I outlined by the red box, waveforms are displayed, 4 or 6 waveforms can be displayed simultaneously. The region II, outlined by the yellow box, numeric data is displayed. For instance, selection could be two ECG leads, Arterial Blood Pressure and Pleth; one ECG lead, Arterial Blood Pressure, Pleth and CO2, or three ECG leads, etc.

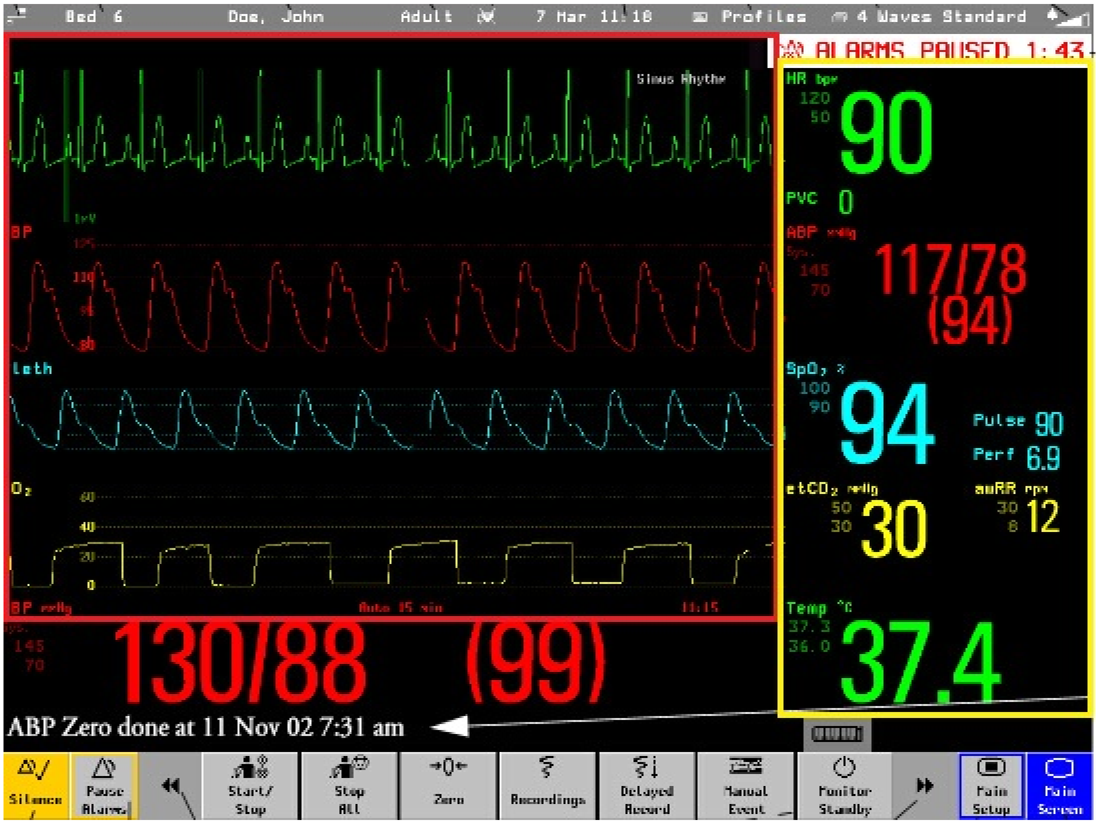

Figure 4-1 Display of the Screen of the Philips Bedside Monitor Showing Physiological Waves and Numerics

As a real-time data acquisition system, the performance was stable and was continuous until terminated by the user. The physiological data was archived in a file and ready for visual display by a custom developed MATLAB Graphical User Interface (GUI). The MATLAB GUI tool is similar to the patient monitor in displaying signals and numeric data. For ECG signal, the MATLAB GUI tool plots one waveform by extracting every 500 sample points. The signals are set to refresh every 10 seconds. Demonstration and display of 2 ECG channels is shown in Figure 4-2 as below.

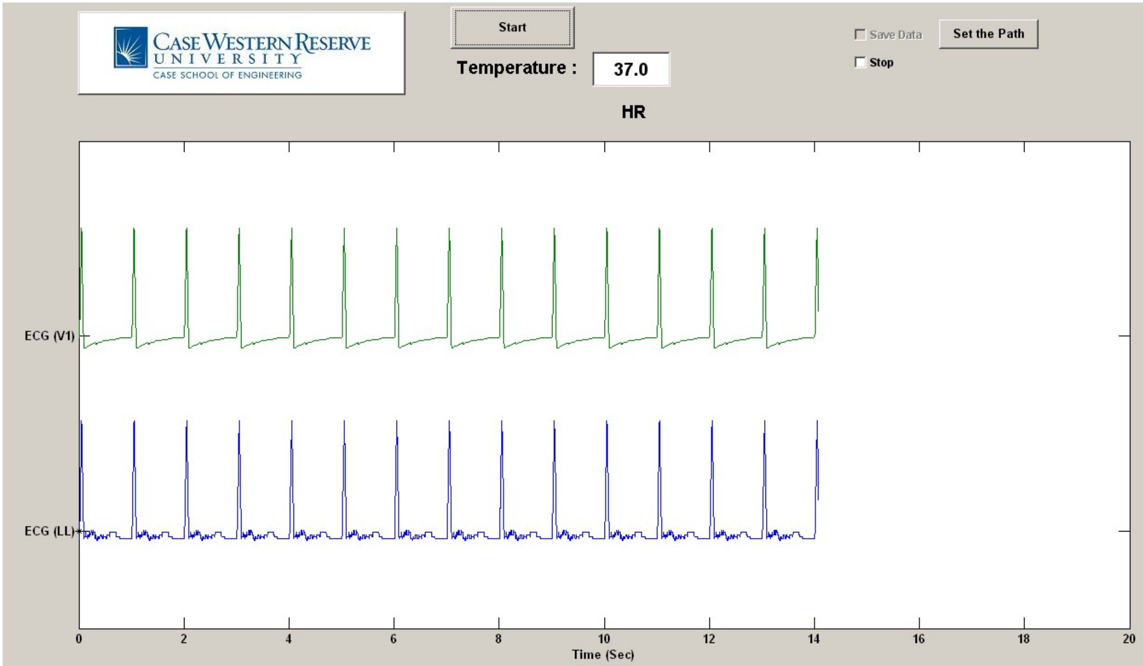

Figure 4-2 Demonstration of Reconstructed Two ECG Wave

4.1.1 Parsed Result of Numerics Data

We can extract information that indicates the physiological state of the patient from the numeric physiological data, e.g. heart rate, respiration rate, arterial blood pressure, arterial oxygen saturation, temperature and pulse rate from plethysmography, which are important vital signs that should be recorded in patient charts. What's more, the absolute time of monitor is contained in the numerics data.

- **Heart Rate**

A more precise method of determining pulse involves the use of an electrocardiograph, or ECG. Continuous electrocardiograph monitoring of the heart is routinely done in many clinical settings, especially in critical care medicine. The heart rate is measured from the ECG using the time between two consecutive R-waves to (R-R interval), as shown in Figure 4-3.

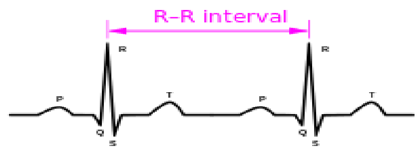

Figure 4-3 Definition of R-R Interval

- **Pulse Rate**

The pulse is the physical expansion of the artery. Its rate is usually measured either at the wrist or the ankle and is recorded as beats per minute. In some sense, pulse rate is supplementary to heart rate.

- **Respiration Rate**

The respiration rate measures the number of breathing cycles per minute, where each cycle includes one in inspiration and the subsequent exhalation. Normal range of resting respiratory rate is 12-15 breaths per minute for adults and 20-40 breaths per minute for children.

- **Arterial Blood Pressure**

Blood pressure is used to determine vascular resistance to blood flow, which is consists of systolic pressure and diastolic pressure. Systolic pressure measures the pressure exerted against the arterial walls when the heart is contracting. Diastolic pressure measures the pressure generated by the arterial walls when the heart is not contracting. The normal range for a resting, healthy adult human is 120 mm Hg systolic and 70 mm Hg diastolic, which can be written as 120/70 mm Hg.

Arterial blood pressure can be measured directly and correctly by inserting a cannula with calibrated transducer in the artery, which could convert pressure into an electrical signal. Non-invasive blood pressure can be measured indirectly by sphygmomanometer and the noninvasive measurements are simpler and quicker than invasive measurements, however, may yield somewhat lower accuracy and small systematic differences in numerical results.

- **Arterial oxygen saturation**

Oxygen saturation is the indicator of the percentage of hemoglobin binding sites in the bloodstream occupied by oxygen. At low partial pressures of oxygen, most hemoglobin is deoxygenated.

The pulse oximeter is a device that clips to the body, typically a finger but may be other areas, that uses the light absorption characteristics of saturated hemoglobin to estimate arterial blood oxygen saturation and pulse rate. The amount of light transmitted through the tissue is then converted to a digital value representing the percentage of hemoglobin saturated with oxygen.

Normal ranges of arterial oxygen saturation in healthy adults are from 97% to 99%. 95% oxygen saturation is clinically accepted in a patient with a normal hemoglobin level. Oxygen saturation values below 90% usually cause hypoxemia. Oxygen saturation levels of 65% and less will impair mental function. When the oxygen saturation decreases to 55% and less, temporary symptoms can occur, such as loss of awareness or consciousness [16]. However, oxygen saturation values are just one component of a complete evaluation of oxygenation and supplemental oxygen therapy. Data should be integrated into a complete assessment to determine the overall status of the patient.

- **Temperature**

Temperature is measured by a thermometer, which provides medical staff with information regarding the potential presence of infection and as well as metabolic response to exercise. The commonly accepted average body temperature taken internally is 37.0 °C (98.6 °F). In healthy adults, body temperature fluctuates about 0.5 °C (0.9 °F) throughout the day, while temperatures of different parts of the body vary about ± 0.4°C. The format of extracted numeric data from the Philips IntelliVue monitor operating in demo mode is listed in the Table 4-1.

Data type	Physio_id	Unit_code	Value
Heart rate	4182	beats per minute	60-100
Pulse rate	4822	beats per minute	60-100
Respiration rate	500A	respiration breathes per minute	12-15
Arterial blood pressure	4A04	mm mercury	120/80
Non-invasive blood pressure (NBP)	4A10	mm mercury	120/80
Arterial oxygen saturation	4BB8	percentage	95-100
Temperature	4B48	centigrade	36.5-37.2
Pulse from NBP	F0E5	beats per minute	60-100

Table 4-1 The Format of Numerics Data

4.1.2 Parsed Result of Alarm Data

The monitor alarm is important data which represent the overall device alert condition. It not only contains global alert status, but also includes a list of active technical and patient alerts.

The following alert information can be extracted from alarm data: alarm state, alarm type.

- **Alarm State**

The alarm state is used for monitoring the overall device alarm status and it will specify the state of each alarm. The alarm condition is inhibited, suspended or latched. The device condition is test mode, standby mode or demo mode.

- **Alarm type**

The alarm type shows the priority level of both technical and patient alarms, ranging from low to high.

For technical alarms, low priority alarms usually imply some minor system faults, such as the system being unable to process signals properly; medium priority alarms are generated during inoperable parameter measurement.

For patient alarms, low priority alarms are generated from abnormal changes of physical vital signs, such as temperature variation caused by fever; medium priority alarms show patient conditions due to violation of clinically defined limits, such as respiratory rate is lower than the minimum limit; high priority alarms are usually used to warn of a life threatening condition, for instance, cardiac arrest.

4.1.3 Parsed Result of Wave Data

Physiological data such as three individual ECG leads, one compound ECG lead, respiration wave, pleth wave, etc. are contained in the waveform data.

4.1.3.1 The Format of Extracted Individual ECG Wave

Each individual ECG wave data packet extracted might contain 1 to 3 ECG signals based on the user's selection. The Philips monitor supports Computer Client polling up to three individual ECG data with a sampling rate of 500 samples per second. The details of the extracted data are: 128 samples in each array, 16 bit sample size, for a total of 256 bytes for each ECG lead. For instance, if we choose three ECG leads: ECG I, ECG II, ECG III, the packet size would be tripled, which doesn't take into account the size of BOF, header, checksum, and EOF fields.

4.1.3.2 Format of Extracted Compound ECG Wave

Each single compound ECG wave contains three ECG leads. The details of extracted data are: 64 samples for each ECG lead, 16 bit sample size, for a total of 384 bytes for each compound ECG, which doesn't take into account the size of BOF, header, checksum, and EOF fields.

4.1.3.3 The Format of Extracted Respiration Wave

The size of the other wave data is much smaller than the ECG data. The system can poll respiratory wave with a sampling rate of 62.5 samples per second. The details of the extracted data are: 16 samples for each packet, 16 bit sample size, for a total of 32 bytes for each respiratory wave, which doesn't take into account the size of BOF, header, checksum, and EOF fields.

4.1.3.4 Format of Extracted Pleth Wave

The system can poll the pleth wave with a sampling rate of 62.5 samples per second. The details of the extracted data are: 16 samples for each packet, 16 bit sample size, for a total of 32 bytes for each pleth wave, which doesn't take into account the size of BOF, header, checksum, and EOF fields.

4.2 Discussion

The system developed in thesis for physiological data acquisition can collect numeric, alarm and wave signals simultaneously in real-time at the bedside. What's more, the system supports receiving and processing physiological data simultaneously, a functionality that previous studies do not have. The data storage method makes the file easy to be access and allows for both prospective and retrospective data analysis.

The major finding in this thesis is that the system improves the promptness and accuracy of data collection performance greatly. The previous version of our system was implemented by Matlab, which might contain 10 to 20 errors during one-minute sampling intervals due to communication issues, which is not an acceptable error rate. However, the final implementation has corrected these issues and clearly demonstrates that the acquisition system developed in this work has high accuracy and reliability.

The improved accuracy/reliability of the system compared to the Matlab implementation results from the fact that C/C++ generally outperforms Matlab in bit-accurate operations. While Matlab is a useful tool for establishing communication and signal processing, it can be quite slow when compared to C/C++ because of its matrix/block-based approach rather than sample-by-sample operations which are preferred in applications that require real-time communications. Matlab is also less efficient when dealing with several interactive feedback loops in the cope where C/C++ is perhaps the most ideal environment.

Additionally, the system is open source and is easy to modify to fit the different design specifications of both clinical and experimental studies. Compared to the system developed in the Linux operating system, the advantage of the Windows application is that it is more user-friendly. Software incompatibility for Linux applications and fewer options for interfaces that support visual display are additional drawbacks that make the Windows operating environment a reasonable choice for our application [17].

5 Conclusions and Future Works

In the previous chapters of this thesis, we have presented the design objectives, implementation processes, and performance of the data acquisition system. This chapter concludes the studies and findings, and discusses some suggestions for future research.

5.1 Conclusions

In the past 4 decades, considerable research has focused on the challenges in medical care in ICU, but this work is far from being complete. In this thesis, several problems related to the current status of data acquisition in the ICU have been addressed, which set the foundation for the current work.

Using details on the protocol that supports serial communications and the Philips Intellivue monitor, we developed a real-time acquisition system suitable for implementation in the ICU to acquire and display all waves, numerics and alarms from the monitor at the bedside, as well archive all data.

Through testing of the system, we observed that the system achieved the design objectives and had the capability to collect multiple channels of physiological data with supporting real-time display. The data acquisition system provided improved data acquisition and data processing capabilities and increased the data integrity in relationship to previous studies. High-level data integration will make enable improved ICU patient through the development of more intelligent and efficient analytics and visualization tools. The physiological data are time-stamped which allow synchronization with data from other ICU devices, supporting both prospective and retrospective data analysis, with the opportunity to significantly improve patient care.

Though there are numerous obstacles, such as insufficient computational power and incompatibility between different monitoring devices and data acquisition and analysis systems, that impede the implementation of the Integrated Medical Environment, we are progressing gradually realize our long term goal [6].

5.2 Future Works

The research presented in this thesis achieved all the basic functions of a data acquisition system for the ICU, but further development are required to meet the needs of critical care in treating patients in the ICU.

An earlier ICU revealed that with standalone pulse oximetry monitors up to 46.5 percent of low saturation alarms were neither observed nor responded to by any caregiver in large part due to constant false alarms associated with such devices [18]. Hence, the first area for future work would definitely be to integrate alarm events records with real-time recorded data from various devices. This requires integration and synchronization of multi-modal data from independent monitors and devises with the appropriate analytics that improves on the accuracy and reliability of identifying important critical events and patient state. Advanced signal processing techniques in combination with methods statistical decision-making and machine learning have the potential to greatly improve the accuracy of alarm data [19]. Once synchronized physiological signals are integrated with relevant alarm events, it is more likely to give clinicians early warnings for life-threatening situations.

As we continue to develop tIME with expanded functionality, the need to integrate more signals from different devices is apparent, including Infusion Pumps, Pulse Oximeters, Mixed Venous Oximeters, Ventilator, etc. [20]. Another information challenge is the integration of physiological data collected from a variety of monitors/devices with relevant clinical observations charted by clinicians within the ICU. The next generation of analysis tool will need to use aggregated data sets (physiological and observational) to assist clinical personnel in making personalized medical decisions.

Further, the development of interoperable ICU data acquisition and electronic medical record systems is essential to create interactive database management systems that span the data range from continuous physiological waveforms, to vitals, to the myriad of clinical patient data from laboratories, image centers, etc. The combination of the real-time bedside acquisition system with the EMR and other hospital information systems is the next major step for the Integrated Medical Environment.

APPENDIX I

Appendix I lists template of all the request and response messages. The messages are compose of the templates.

1. ASSOCIATION REQUEST

The following building blocks can be used to format an Association Request message:

AssocReqSessionHeader

0x0D <LI>

AssocReqSessionData

0x05 0x08 0x13 0x01 0x00 0x16 0x01 0x02

0x80 0x00 0x14 0x02 0x00 0x02

AssocReqPresentationHeader

0xC1 <LI> 0x31 0x80 0xA0 0x80 0x80 0x01

0x01 0x00 0x00 0xA2 0x80 0xA0 0x03 0x00

0x00 0x01 0xA4 0x80 0x30 0x80 0x02 0x01

0x01 0x06 0x04 0x52 0x01 0x00 0x01 0x30

0x80 0x06 0x02 0x51 0x01 0x00 0x00 0x00

0x00 0x30 0x80 0x02 0x01 0x02 0x06 0x0C

0x2A 0x86 0x48 0xCE 0x14 0x02 0x01 0x00

0x00 0x00 0x01 0x01 0x30 0x80 0x06 0x0C

0x2A 0x86 0x48 0xCE 0x14 0x02 0x01 0x00

0x00 0x00 0x02 0x01 0x00 0x00 0x00 0x00

0x00 0x00 0x61 0x80 0x30 0x80 0x02 0x01

0x01 0xA0 0x80 0x60 0x80 0xA1 0x80 0x06

0x0C 0x2A 0x86 0x48 0xCE 0x14 0x02 0x01

0x00 0x00 0x00 0x03 0x01 0x00 0x00 0xBE

0x80 0x28 0x80 0x06 0x0C 0x2A 0x86 0x48

0xCE 0x14 0x02 0x01 0x00 0x00 0x00 0x01

0x01 0x02 0x01 0x02 0x81

AssocReqUserData

Refer to Appendix II

AssocReqPresentationTrailer

0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

2. ASSOCIATION RESPONSE

The following building blocks can be used to format an Association Response message:

AssocRespSessionHeader

0x0E <LI>

AssocRespSessionData

0x05 0x08 0x13 0x01 0x00 0x16 0x01 0x02

0x80 0x00 0x14 0x02 0x00 0x02

AssocRespPresentationHeader

0xC1 <LI> 0x31 0x80 0xA0 0x80 0x80 0x01

0x01 0x00 0x00 0xA2 0x80 0xA0 0x03 0x00

0x00 0x01 0xA5 0x80 0x30 0x80 0x80 0x01

0x00 0x81 0x02 0x51 0x01 0x00 0x00 0x30

0x80 0x80 0x01 0x00 0x81 0x0C 0x2A 0x86

0x48 0xCE 0x14 0x02 0x01 0x00 0x00 0x00

0x02 0x01 0x00 0x00 0x00 0x00 0x61 0x80

0x30 0x80 0x02 0x01 0x01 0xA0 0x80 0x61

0x80 0xA1 0x80 0x06 0x0C 0x2A 0x86 0x48

0xCE 0x14 0x02 0x01 0x00 0x00 0x00 0x03

0x01 0x00 0x00 0xA2 0x03 0x02 0x01 0x00

0xA3 0x05 0xA1 0x03 0x02 0x01 0x00 0xBE

0x80 0x28 0x80 0x02 0x01 0x02 0x81

AssocRespUserData

Refer to Appendix II

AssocRespPresentationTrailer

0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

3. ASSOCIATION REFUSE

The following building blocks can be used to format a Refuse message:

RefuseSessionHeader

0x0C 0x03

RefuseSessionData

0x32 0x01 0x00

RefusePresentationHeader

This block is empty in the Refuse message.

RefuseUserData

This block is empty in the Refuse message.

RefusePresentationTrailer

This block is empty in the Refuse message.

4. RELEASE REQUEST

The following building blocks can be used to format a Release Request message:

ReleaseReqSessionHeader

0x09 0x18

ReleaseReqSessionData

This block is empty in the Release Request message.

ReleaseReqPresentationHeader

0xC1 0x16 0x61 0x80 0x30 0x80 0x02 0x01

0x01 0xA0 0x80 0x62 0x80 0x80 0x01 0x00

0x00 0x00 0x00 0x00

ReleaseReqUserData

This block is empty in the Release Request message.

ReleaseReqPresentationTrailer

0x00 0x00 0x00 0x00

5. RELEASE RESPONSE

The following building blocks can be used to format a Release Response message:

ReleaseRespSessionHeader

0x0A 0x18

ReleaseRespSessionData

This block is empty in the Release Response message.

ReleaseRespPresentationHeader

0xC1 0x16 0x61 0x80 0x30 0x80 0x02 0x01

0x01 0xA0 0x80 0x63 0x80 0x80 0x01 0x00

0x00 0x00 0x00 0x00

ReleaseRespUserData

This block is empty in the Release Response message.

ReleaseRespPresentationTrailer

0x00 0x00 0x00 0x00

6. ASSOCIATION ABORT

The following building blocks can be used to format a Association Abort message:

AbortSessionHeader

0x19 0x2E

AbortSessionData

0x11 0x01 0x03

AbortPresentationHeader

0xC1 0x29 0xA0 0x80 0xA0 0x80 0x30 0x80

0x02 0x01 0x01 0x06 0x02 0x51 0x01 0x00

0x00 0x00 0x00 0x61 0x80 0x30 0x80 0x02

0x01 0x01 0xA0 0x80 0x64 0x80 0x80 0x01

0x01 0x00 0x00 0x00 0x00 0x00 0x00

AbortUserData

This block is empty in the Abort message.

AbortPresentationTrailer

0x00 0x00 0x00 0x00

APPENDIX II

Appendix II lists the component of the user data of Association Request and Response message template.

User Data

The following section contains an example for the User Data which is contained in an Association Request message.

UserData

ASNLength length		: 72
		{0x48}

MDSEUserInfoStd

ProtocolVersion	protocol_version	: MDDL_VERSION1
NomenclatureVers.	nomenclature_version	: NOMEN_VERSION
FunctionalUnits	functional_units	: 0
SystemType	system_type	: SYST_CLIENT
StartupMode	startup_mode	: COLD_START

{0x80 0x00 0x00 0x00 0x40 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x80 0x00 0x00 0x00 0x20 0x00 0x00 0x00}

Option List

AttributeList	count	: 0
	length	: 0

{0x00 0x00 0x00 0x00}

Supported Profiles

AttributeList	count	: 1
	length	: 44

{0x00 0x01 0x00 0x2c}

AVAType

OIDType	attribute_id	: NOM_POLL_PROFILE_SUPPORT
u_16	length	: 40
		{0x00 0x01 0x00 0x28}

PollProfileSupport (attribute_val)

PollProfileRev.	poll_profile_revision	: POLL_PROFILE_REV_0
RelativeTime	min_poll_period	: 800000
u_32	max_mtu_rx	: 1000
u_32	max_mtu_tx	: 1000
u_32	max_bw_tx	: 0xffff 0xffff
PollProfileOpt.	options	: 0x6000 0x0000

{0x80 0x00 0x00 0x00 0x00 0x00 0x09 0xc4 0x00 0x00 0x09 0xc4 0x00 0x00 0x03 0xe8 0xff 0xff 0xff 0xff 0x60 0x00 0x00 0x00}

Optional Packages

AttributeList	count	: 1
	length	: 12

{0x00 0x01 0x00 0x0c}

AVAType

OIDType	attribute_id	: NOM_ATTR_POLL_PROFILE_EXT
u_16	length	: 8
		{0xf0 0x01 0x00 0x08}

PollProfileExt (attribute_val)

PollProfileExtOpt.	options	: POLL_EXT_PERIOD_NU_AVG_60SEC
AttributeList	count	: 0
	length	: 0

{0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00}

With this User Data, the length field of the Presentation Header must be set to 220 (0xDC) and the length field of the Session Header must be set to 236 (0xEC).

References

1. Intensive care unit - Wikipedia, the free encyclopedia. (n.d.). Retrieved from http://en.wikipedia.org/wiki/Intensive-care_unit

2. Smith, S. E. (2013-03-24). "What is an ICU". wiseGEEK. Bronwyn Harris, ed. Sparks, Nevada: Conjecture Corporation. Retrieved 2012-06-15.

3. What does Intensive-care unit mean? -definitions.(n.d.). Retrieved from http://www.definitions.net/definition/Intensive-care%20unit

4. Takrouri, M.S.M. (2004). "Intensive Care Unit". Internet Journal of Health (Sugar Land, Texas: Internet Scientific Publications) 3 (2). doi:10.5580/1c97. ISSN 1528-8315. OCLC 43535892. Retrieved 2007-08-25.

5. Research - Case Western Reserve University School of Medicine. (n.d.). Retrieved from http://casemed.case.edu/dept/neurology/Research.html

6. Michael A. D, and Kenneth A. L. The ICU of the Future: Translating Raw Data into Bedside Action. UH Neurological Institute Journal 2014, 6(2), 4-8.

7. Pickering B. W, Gajic O, Ahmed A, Herasevich V, and Keegan M. T. Data utilization for medical decision making at the time of patient admission to ICU. Critical Care Medicine 2013 Jun; 41(6):1502-10. doi: 10.1097/CCM.0b013e318287f0c0.

8. Apkar S, and Francis A. Towards an ICU Clinical Decision Support System using Data Wavelets. IJCA Special Issue on Systems and Data Processing (ICISD), pages 37-43, 2011.

9. Sukuvaara, T.I., Sydanmaa, M.E., Nieminen, H.O.,Heikela, A., Koski, E.M.J. (1993), Object-Oriented Implementation of an Architecture for Patient Monitoring, IEEE Engineering in Medicine and Biology, Volume 12,Issue 4, pages 69-81, December 1993.

10. Leonard Polizzotto. Integrated Medical Environment Decision Support - Institute of Medicine.(n.d.).Retrieved from www.iom.edu/~/media/Files/Activity%20Files/Quality/VSRT/IC%20Meeting%20Docs/DLC%2008-23-12/Len%20Polizzotto.pdf. August 23, 2012.

11. Hemphill J. C. and Michael D. G. Multimodal Monitoring and Neurocritical Care Bioinformatics: The Dawn of a New Age. Retrieved from http://patients.aan.com/news/?event=read&article_id=6972. December 23, 2008.

12. Brahm G, James M, Bruce A. M, Miles E, Susanna L, Zhiyoung S, Donald K, and Robert J. S. Physiologic data acquisition system and database for the study of disease dynamics in the intensive care unit. Critical Care Medicine. 03/2003; 31(2):433-41. DOI:10.1097/01.CCM.0000050285.93097.52.

13. Microsoft. MSDN Library for Visual Studio 2013. MFC Desktop Application's Hierarchy Chart. Retrieved from http://msdn.microsoft.com/en-us/library/ws8s10w4.aspx, 2014

14. Mahmoud M. Real-time Data Acquisition System for Monitoring Patients in Intensive Care Unit (ICU). Multisensor, Multisource Information Fusion: Architectures, Algorithms, and Applications 2003, Proceedings of SPIE Vol. 5099 (2003), 320-326.

15. Koninklijke Philips N.V.,(2014). IntelliVue MP60 and MP70 patient monitors Retrieved from http://www.healthcare.philips.com/main/products/patient_monitoring/products/intellivue_mp70_mp60/

16. Mike M. Oxymoron: Our Love-Hate Relationship with Oxygen. Albany Medical College, New York. 10/12/2010

17. Kropyvnytskyy I, Saunders F, Schierek P, et al. A computer system for continuous long-term recording, processing, and analysis of physiologic data of brain injured patients in ICU settings. BRAIN INJURY, July 2001; 15(7), 577-583.

18. R. M. Gardner and M. M. Shabot. Patient-Monitoring Systems. Medical Informatics Health Informatics 2001, 443-484.

19. Ying Z. (2003). Real-Time Analysis of Physiological Data and Development of Alarm Algorithms for Patient Monitoring in the Intensive Care Unit (Master Thesis). Massachusetts Institute of Technology, Cambridge, Massachusetts.

20. Reed M. G, William L. H, Thomas D. E, Thomas A. O, Hsueh-Fen W. Y. Real time data acquisition: recommendations for the Medical Information Bus (MIB). International Journal of Clinical Monitoring and Computing 8:251-258, 1992.