A PROJECT REPORT ON

REMOTE MONITORING OVER INTERNET

SUBMITTED TO THE UNIVERSITY OF PUNE, PUNE
IN THE PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE AWARD OF THE DEGREE

OF

BACHELOR OF ENGINEERING (Information Technology)

BY

Arjun Kumar Ganjoo	**Exam No: B3058507**
Priyanka R. Menon	**Exam No: B3058561**
Prateek Bhatnagar	**Exam No: B3058594**
Sarfaraz A. Sayyed	**Exam No: B3058609**

DEPARTMENT OF INFORMATION TECHNOLOGY

PUNE INSTITUTE OF COMPUTER TECHNOLOGY

Survey number 27, Pune Satara Road, Dhanakawadi

PUNE 411043

APRIL 2007

CERTIFICATE

This is to certify that the project report entitled

"REMOTE MONITORING OVER INTERNET"

Submitted by

Arjun Kumar Ganjoo	Exam No: B3058507
Priyanka R. Menon	Exam No: B3058561
Prateek Bhatnagar	Exam No: B3058594
Sarfaraz A. Sayyed	Exam No: B3058609

is a bonafide work carried out by them under the supervision of Prof. Tushar Rane and it is approved for the partial fulfillment of the requirement of University of Pune, Pune for the award of the degree of Bachelor of Engineering (Information Technology).

This project work has not been earlier submitted to any other Institute or University for the award of any degree or diploma.

(Prof. Tushar Rane)	(Prof. G.P Potdar)
Guide & Project Coordinator	Head,
Department of Information Technology	Department of Information Technology

(Prof. Arun Gaikwad)
Principal,
PICT, Pune

Place: Pune
Date:

ACKNOWLEDGEMENT

Executing a systems project involves quite a lot of effort. We take this opportunity to thank all those people who have been instrumental, directly or indirectly, in making our project successful.

First of all we would like to thank our Head of the Information Technology Department, Prof.G.P.Potdar for providing us with necessary computing resources and facilities. We also extend our sincere thanks to Mr. Ashok Patil, our external project guide at Honeywell Automation India Ltd. (HAIL) for his perseverance, valuable suggestions and eagerness to help us in all possible ways. Furthermore we are appreciative of Ms. Meghna, Mr. Pandurang Khutal, Mr. Sahil and others from the HAIL fraternity for their valuable and timely inputs.

We are grateful to Prof. Tushar A. Rane, our guide at Pune Institute of Computer Technology, for his undivided attention and support without whom this project was not possible to complete.

Ms. Sanika Bapat deserves a special mention here for providing us with valuable feedback and suggestions during the progress of RMOI. We are indebted to her for her valuable time she spared to communicate with us over the issues that arose during design and implementation of RMOI.

We would like to express our heartfelt gratitude toward all our colleagues and friends for all the support and encouragement that they have extended to us. All of this would have never been possible without the tremendous support that our parents gave us during the project. Their cheerful encouraging words and the faith they showed in us helped us a lot during our testing times.

Arjun Kumar Ganjoo
Priyanka R. Menon
Prateek Bhatnagar
Sarfaraz A. Sayyed

Abstract

*"Remote Monitoring" refers to accessing and monitoring a device from a distant location. At times it is not possible and feasible to monitor a device by being physically present along with it. This leads to problems such as improper maintenance as the breakdown takes a long time to get reported and fixed. This delay renders the unattended device unusable for days. Such problems can be solved by constantly monitoring the unattended devices and the environmental conditions around them from a remote location (through internet). "**RMOI**" system makes use of a Serial to Ethernet development kit (i.e. **RCM4000)** that provides the physical interface between serial device and Internet. RCM4000 can be programmed by using **Dynamic C** programming environment. This system allows user to access or modify various features of the Serial Device from internet. User can view details through the web pages that are served by the web server (embedded) running on the RCM4000 kit which is connected to the Serial Device. Additional features such as **E-diagnostics** & **E-mail alarms**, **Authenticating Users** are also supported.*

KEYWORDS:-

RCM 4000, RMOI, Dynamic C, E-Diagnostic's-mail Alarms, Authenticating Users.

Table of Contents

LIST OF DIAGRAMS

LIST OF TABLES

Chapter One

Overview

1. OVERVIEW

1.1 INTRODUCTION

Several asynchronous serial interfaces, including RS-232, RS-422, RS-485, LonWorks, CANbus, Profibus, and Interbus are currently in use to connect various types of devices, such as sensors, card readers, meters, analyzers, converters and PLCs. Devices that convert data between the serial and Ethernet interfaces allow engineers to take advantage that's better of two worlds. The "serial world" is the world of sensors, actuators, modems, and basic RS-485 networks that transmit data between devices and computers. The "Ethernet world" is the world of NICs (Network Interface Cards), the Internet, and open protocols used to whisk information from host to host.

Serial-to-Ethernet refers to any product or process used in the marriage of the serial and Ethernet interfaces. In general, this is an important field for both business and industry, since millions of legacy serial devices, most without built-in Ethernet ports, are still in common use today. An *Ethernet communications card allows* companies to connect legacy serial devices to an Ethernet LAN/WAN, providing many more options for data acquisition, device management, and industrial control than would otherwise be available.

1.2 PROJECT IDEA

A serial device has been placed at a remote location and we want to monitor the serial device from different parts of the globe, so to do this we need the help of internet but in spite of it we don't have the right software that would make the interaction between the person who's suppose to monitor the serial device and the serial device itself simple enough, thus the idea is to develop a software interface that would understand the serial device as well as the medium being used to monitor it.

Thus to enable this we used the help of a development kit called "Rabbit Core Module Development Kit". We used the RCM 4000 series of the kit to enable us to provide a platform for serial to Ethernet conversion vice-versa. It has a predefined TCP/IP stack which we have exploited to design a software code of serial to Ethernet conversion and vice-versa. This kit will accept the data sent by the user and convert it to serial data and forward it to the serial device. Serial device will read the data sent and perform the desired operation and then send the response to the kit, which will then frame it into the TCP/IP packets and send it over the internet to the user. The implementation details have been discussed later.

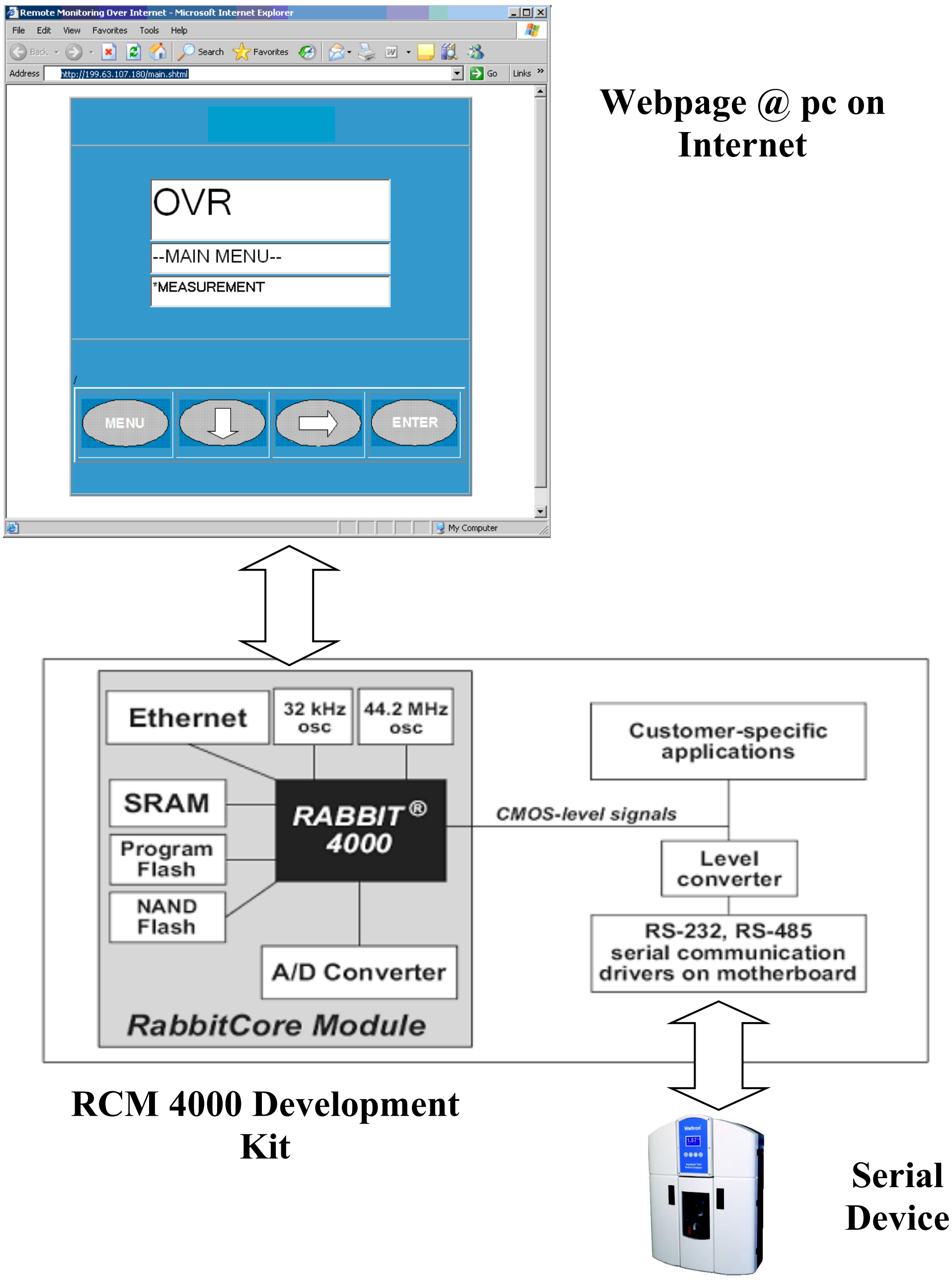

Fig 1.1 **<u>Project Idea</u>**

1.3 NEED OF THE PROJECT

"*Remote Monitoring*" refers to accessing and monitoring a serial device from a distant location. At times it is not possible and feasible to monitor a device by being present physically. This leads to problems such as improper maintenance as the breakdown takes a long time to get reported and fixed. This delay renders the unattended device unusable for days. Such problems can be solved by constantly monitoring the unattended devices and the environmental condition around them from a remote location (internet).Fig 1.1 shows the device to be monitored.

Fig 1.2Serial Device

1.4 LITERATURE SURVEY

The transformation between the serial and Ethernet interfaces takes place at the *electronic signal* and *network protocol levels*, such as in the transformation of data from the RS-232 format into a format suitable for a TCP/IP network. *Ethernet communication card)*, is a smart, standalone device with a tiny embedded operating system and CPU that is, nevertheless, large enough to contain its own operating system, as well as the necessary software protocols, such as the TCP/IP stack. A Serial Device Server also comes equipped with the required hardware interfaces, such as RS-232, RS-422, and RS-485 ports. The device server can transfer, and even process data between the serial and Ethernet interfaces to carry out pre-defined tasks

- Dynamic ‘C’ Guide book.
- Rabbit Development Guide for understanding protocol stack and development kit for TCP/IP-SERIAL communication.
- Study of present code to understand its functionality.

1.4.1 DYNAMIC ‘C’

Dynamic C is an integrated development system for writing embedded software. It is designed for use with Z-World controllers and other controllers based on the Rabbit microprocessor.

<u>The Nature of Dynamic C</u>

Dynamic C integrates the following development functions:

- Editing.
- Compiling.
- Linking.
- Loading.
- Debugging.

In fact, compiling, linking and loading are one function.

1. Dynamic C has an Easy-to-use, built-in, full-featured, text editor.
2. Dynamic C programs can be executed and debugged interactively at the source-code or machine-code level.
3. Pull-down menus and keyboard shortcuts for most commands make Dynamic C easy to use. Dynamic C also supports assembly language programming. It is not necessary to leave C or the development system to write assembly language code. C and assembly language may be mixed together.
4. Debugging under Dynamic C includes the ability to use “printf” commands, watch expressions and breakpoints. Watch expressions can be used to compute C expressions involving the target’s program variables or functions. Watch expressions can be evaluated while stopped at a breakpoint or while the target is running its program

5. Dynamic C 10.09 introduces advanced debugging features such as execution and stack tracing. Execution tracing can be used to follow the execution of debuggable statements, including such information as function/file name, source code line and column numbers, action performed, time stamp of action performed and register contents. Stack tracing shows function call sequences and parameter values.
6. Dynamic C provides extensions to the C language (such as *shared* and *protected* variables, co statements and co functions) that support real-world embedded system development. Dynamic C Supports cooperative and preemptive multitasking.
7. Dynamic C comes with many function libraries, all in source code. These libraries support real-time programming, machine level I/O, and provide standard string and math functions.

Dynamic C Enhancements:

Dynamic C differs from a traditional C programming system running on a PC or under UNIX. The reason? To be better help customers write the most reliable embedded control software possible.

It is not possible to use standard C in an embedded environment without making adaptations. Standard C makes many assumptions that do not apply to embedded systems. For example, standard C implicitly assumes that an operating system is present and that a program starts with a clean slate, whereas embedded systems may have battery-backed memory and may retain data through power cycles. Rabbit Semiconductor has extended the C language in a number of areas.

Many enhancements have been added to Dynamic C. Some of these are listed below:

1. *Function chaining*, a concept unique to Dynamic C, allows special segments of code to be embedded within one or more functions. When a named function chain executes, all the segments belonging to that chain execute. Function chains allow software to perform initialization, data recovery, or other kinds of tasks on request.
2. *Co-statements* allow concurrent parallel processes to be simulated in a single program.
3. *Co-functions* allow cooperative processes to be simulated in a single program.
4. *Slice statements* allow preemptive processes in a single program.
5. Dynamic C supports embedded *assembly code* and stand-alone assembly code.

6. Dynamic C has *shared* and *protected* keywords that help protect data shared between different contexts or stored in battery-backed memory.

Dynamic C has a set of features that allow the programmer to make fullest use of extended memory.

1. Dynamic C supports the 1 MB address space of the microprocessor. The address space is segmented by a memory management unit (MMU). Normally, Dynamic C takes care of memory management, but there are instances where the programmer will want to take control of it.
2. Dynamic C has keywords and directives to help put code and data in the proper place. The keyword root selects root memory (addresses within the 64 KB physical address space). The keyword xmem selects extended memory, which means anywhere in the 1024 KB or 1 MB code space. root and xmem are semantically meaningful in function prototypes and more efficient code is generated when they are used. Their use must match between the prototype and the function definition. The directive #memmap allows further control.

Dynamic Differences

The main differences in Dynamic C are summarized below:

1. If a variable is explicitly initialized in a declaration (e.g., int x = 0;), it is stored in flash memory (EEPROM) and cannot be changed by an assignment statement. Such a declaration will generate a warning that may be suppressed using the const keyword: const int x = 0 .To initialize static variables in Static RAM (SRAM) use #GLOBAL_INIT sections. Note that other C compilers will automatically initialize all static variables to zero that are not explicitly initialized before entering the main function. Dynamic C programs do not do this because in an embedded system you may wish to preserve the data in battery-backed RAM on reset.

2. The numerous include files found in typical C programs are not used because Dynamic C has a library system that automatically provides function prototypes and similar header information to the compiler before the user's program is compiled. This is done via the #use directive. This is an important topic for users who are

writing their own libraries. It is important to note that the #use directive is a replacement for the #include directive, and the #include directive is not supported.

3. When declaring pointers to functions, arguments should not be used in the declaration. Arguments may be used when calling functions indirectly via pointer, but the compiler will not check the argument list in the call for correctness.
4. Bit fields are not supported.
5. Separate compilation of different parts of the program is not supported or needed.

<u>TCP/IP Libraries</u>

Dynamic C includes extensive TCP/IP libraries that serve as application templates for fast program development.

1. **HTTP** — Hypertext Transfer Protocol. Protocol for web browsers and servers to transfer files, such as text and graphics. Contains facilities for Server Side Includes (SSI) and CGI routines.
2. **POP3** — Post Office Protocol. Standard protocol to retrieve e-mail.
3. **TFTP** — Trivial File Transfer Protocol. Simplified version of FTP that allows files to be transferred from one computer to another over a network. Client and server available.
4. **FTP** — File Transfer Protocol. Application protocol in TCP/IP stacks for transferring files between network nodes. Server with password support for file transfers between network nodes.
5. **SMTP** — Simple Mail Transfer Protocol. Internet protocol providing e-mail services.
6. **DHCP** — Dynamic Host Configuration Protocol. A method for a device to assign its network configuration information from a central server.
7. **Socket-Level UDP** — User Datagram Protocol. Protocol exchanging datagram's without acknowledgements or guaranteed delivery.
8. **Socket-Level TCP** — Transmission Control Protocol. Reliable full-duplex data transmission.
9. **ICMP** — Internet Control Message Protocol. Network protocol to verify connecting to another host. (PING)

1.4.2 HTTP

An HTTP (Hypertext Transfer Protocol) server makes HTML (Hypertext Markup Language) pages and other resources available to clients (that is, web browsers). HTTP is implemented by HTTP.LIB, thus you need to write #use "http.lib" near the top of your program. HTTP depends on the Dynamic C networking suite, which is included in your program by writing #use "dcrtcp.lib".

Setting up the network subsystem is a necessary pre-requisite for use of HTTP. In the file tcp_config.lib are predefined configurations that may be accessed by#define of the macro TCPCONFIG.

<u>HTTP Server Data Structures:</u>

<u>HttpState</u>

Use of the HttpState structure is necessary for CGI functions.

HTTP File Upload

The HTTP library provided with Dynamic C allows the association of C functions with web page URLs. When the user, via their web browser, retrieves a specified resource, the C function may be called from the HTTP server. Such a function is called a Common Gateway Interface (CGI) function, and it is responsible for generating a response to the user's request.

The advantage of using a CGI is that it can generate web page content on-the-fly, and cause the browser to display or do anything that it is capable of. In addition, the CGI is able to read data that was sent by the browser.

Steps to use CGI :

1. #use "dcrtcp.lib", and specify network configuration options.
2. #use <filesystem(s) of choice>, and specify the file system configuration.
3. #define USE_HTTP_UPLOAD
4. #use "http.lib"

5. Create an initial web page with a form asking for the file(s) to be uploaded. The main requirement is that you specify `enctype="multipart/form-data"` inside the <FORM> tag(s).
6. Write a CGI function (if not using the default one provided).
7. Create an initial resource table containing at least an entry for each of the above two resources (the web page and the CGI).
8. Create a list of content type mappings, i.e., the MIME table.
9. Create rules which limit the upload facility to select user groups.
10. Create a set of user IDs.
11. In the main program, call `http_handler ()` in a loop.

1.4.3 SMTP

SMTP (Simple Mail Transfer Protocol) is one of the most common ways of sending e-mail. SMTP is a simple text conversation across a TCP/IP connection. The SMTP server usually resides on TCP port 25 waiting for clients to connect.
SMTP has following components that were useful in designing our program:-

1. **Envelope**: it usually contains the senders address.
2. **Message**: it contains **Headers & Footers**

   Headers: they define the sender the receiver and the subject of the message along with other information.

3. **Address**: to deliver mail a mail handling system must use an addressing system with unique addresses. It has two parts

   - **Local part:** it defines the name of a special file called user mailbox where al the mails for the user are stored and retrieved by the user.

   - **Domain name:** it signifies the region of the network from which mail was sent or received.

4. **User Agent (UA):** it composes reads, replies and forwards messages.

To send mails using SMTP we need *client SMTP and server SMTP*

1.4.4 RABBIT CORE MODULE (RCM4000):

Table: 1.1 Rabbit Core Module (RCM4000) Specification

Rabbit Core RCM4000 Specifications		
Features	RCM4000	RCM4010
Microprocessor	Rabbit 4000 @ 58.98 MHz	
EMI Reduction	Spectrum spreader for reduced EMI (radiated emissions)	
Ethernet Port	10Base-T, RJ-45, 2 LEDs	
Flash	512K (16-bit)	
SRAM	512K (16-bit)	
NAND Flash	32MB	—
General-Purpose I/O	19 digital I/O lines, configurable with up to four layers of alternate functions	25 digital I/O lines, configurable with up to four layers of alternate functions
Analog Inputs	8 channels single-ended (11-bit resolution) or 4 channels differential (12-bit resolution)	—
Additional Inputs	2 Startup Mode, Reset In, CONVERT	2 Startup Mode, Reset In
Additional Outputs	Status, Reset Out, Analog	Status, Reset Out.
Auxiliary I/O Bus	8 data and up to 6 address (shared with I/O), plus I/O read/write	
Pulse-Width Modulators	-------------------------------------	Two channels synchronized PWM with 10-bit counter Two channels variable-phase or synchronized PWM with 16-bit counter

Features	RCM4000	RCM4010
Serial Ports	Five shared high-speed, CMOS-compatible ports • All 5 are configurable as asynchronous (with IrDA) • 4 configurable as clocked serial (SPI) • 1 configurable as SDLC/HDLC • 1 clocked serial port dedicated for A/D converter (RCM4000) • 1 asynchronous serial port dedicated for programming	
Serial Rate	Max. asynchronous baud rate = CLK/8	
Backup-Battery	Connection for user-supplied battery (to support RTC and data SRAM)	
Slave Interface	Slave port permits use as master or intelligent peripheral with master controller.	
Real-Time Clock	Yes	
Timers	Ten 8-bit timers (6 cascadable from the first), one 10-bit timer with 2 match registers, and one 16-bit timer with 4 outputs and 8 set/reset registers	
Watchdog/Supervisor	Yes	
Input Capture	-------------------------------------	2-channel input capture can be used to time input signals from various ports.
Quadrature Decoder	-----------------------------------	2-channel Quadrature decoder accepts inputs from external incremental encoder modules.
Power	3.0 – 3.6 V DC, 90 mA @ 3.3 V (preliminary, pins unloaded)	
Operating Temp.	0°C to +70°C	
Humidity	5–95%, non-condensing	
Connectors - Headers	One 2 x 25, 1.27 mm pitch IDC signal header. One 2 x 5, 1.27 mm IDC programming header	

Features	RCM4000	RCM4010
Board Size	1.84” x 2.42” x 0.77” (47 mm × 61 mm × 20 mm)	
Part Number	20-101-1094	20-101-1112
Development Kit	U.S. 101-1114, Int'l 101-1115	

Key Features

- Rabbit® 4000 microprocessor with Integrated Ethernet
- Clock speed at 58.98 MHz with 16-bit memory
- 32 MB NAND flash for data
- 512K Flash / 512K SRAM
- 8 channel 12-bit A/D converter
- Up to 5 CMOS-compatible serial ports
- Optimized for use with Dynamic C® 10

Design Advantages

- Designed for embedded networking with intelligence and I/O control
- Serves web pages, controls remote devices, links equipment to the Internet
- Security-key feature with “tamper detect” and encryption capabilities adds peace of mind for OEMs and systems integrators
- Complete microprocessor, on-board memory, development software with royalty-free TCP/IP stack, and hundreds of sample programs reduce time-to-market .

Applications

- Serial-to-Ethernet applications
- Remote monitoring and communications
- Web-enabling devices
- Device/data management and control

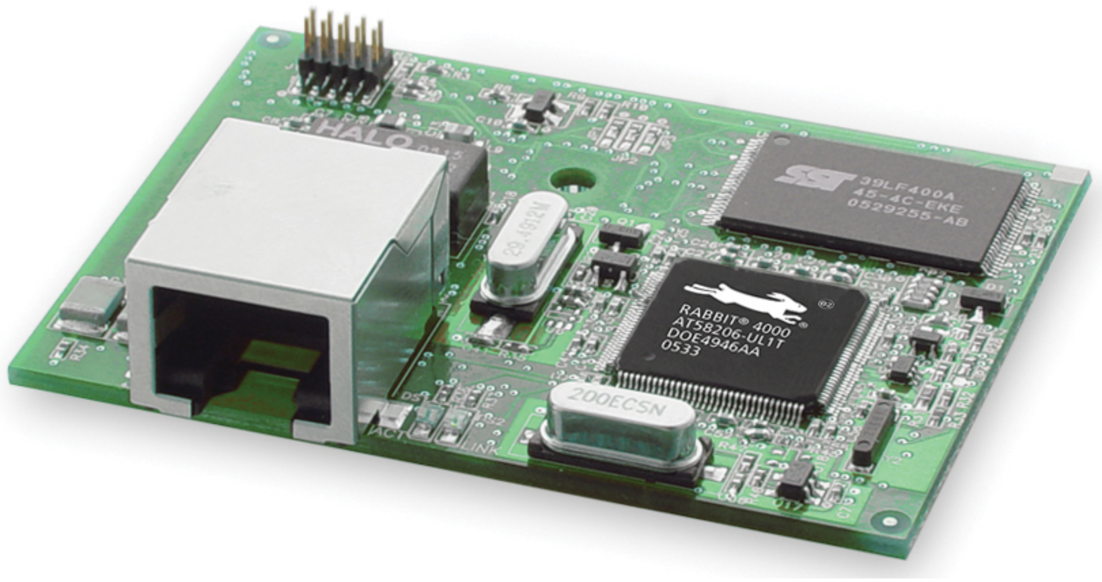

Fig: 1.3 <u>RCM 4000 Module</u>

Chapter two

Problem Definition and Scope

2. PROBLEM DEFINITION AND SCOPE

2.1 PROBLEM DEFINITION

To develop an Ethernet Communications Interface. Ethernet Communications Interface will provide communications for a Serial device. The Interface essentially should –

- Allow remote monitoring of processes from a serial device through Ethernet Communication Interface.
- Use Industry standard Ethernet TCP/IP to access information.

2.2 SCOPE

To enable the Serial device to interact with the Rabbit Core Module 4000. The scope is divided into three phases:

- The first consists of wired communication between Serial device & RCM4000 Kit using RS 232 cable.
- The second consists of connecting the middleware (RCM4000) to the internet using Ethernet interface.
- The third consists of Displaying GUI of Serial device, Triggering Alarms and E-mail Alerts to Authorized users between PC on internet and Serial device using RCM4000.Authenticating users will also be supported.

Chapter Three

Software Requirements and Specifications

3. SOFTWARE REQUIREMENTS SPECIFICATION

3.1 INTRODUCTION

The system will provide Serial to Ethernet communication between PC at the user-end (connected to internet) and a Serial device at the application end via Rabbit Core Development Module.

3.1.1 PURPOSE

The purpose of this project is to enable authorized users to monitor the remotely placed Serial device via internet.

3.1.2 SCOPE

This project has a wide scope in industrial, government and military use. The software should be able to handle multiple users. Once a request has been initiated and authenticated the user can toggle the menu contents and change the settings according to his needs. In case of emergency the Serial device self triggers alert emails to the authenticated users. The email would contain the details regarding alerts and its cause, so that users can take appropriate action.

The use of this software will be to provide a user access to any information embedded device, which could be some control information used by the device as a parameter for the function that the device is supposed to perform. However this software would not only make the above process user friendly but make this information accessible over the internet thus enabling the user to be located anywhere as long as he has access to the network on which the device is present.

3.1.3 Overall description

The system will consist of following modules:-

- Serial to Ethernet Conversion and vice versa.
- Authenticating Users.
- Email Triggering.

.

3.2 INFORMATION FLOW DESCRIPTION

The Information Flow representation diagram graphically represents the Interplay between PC and Serial device.

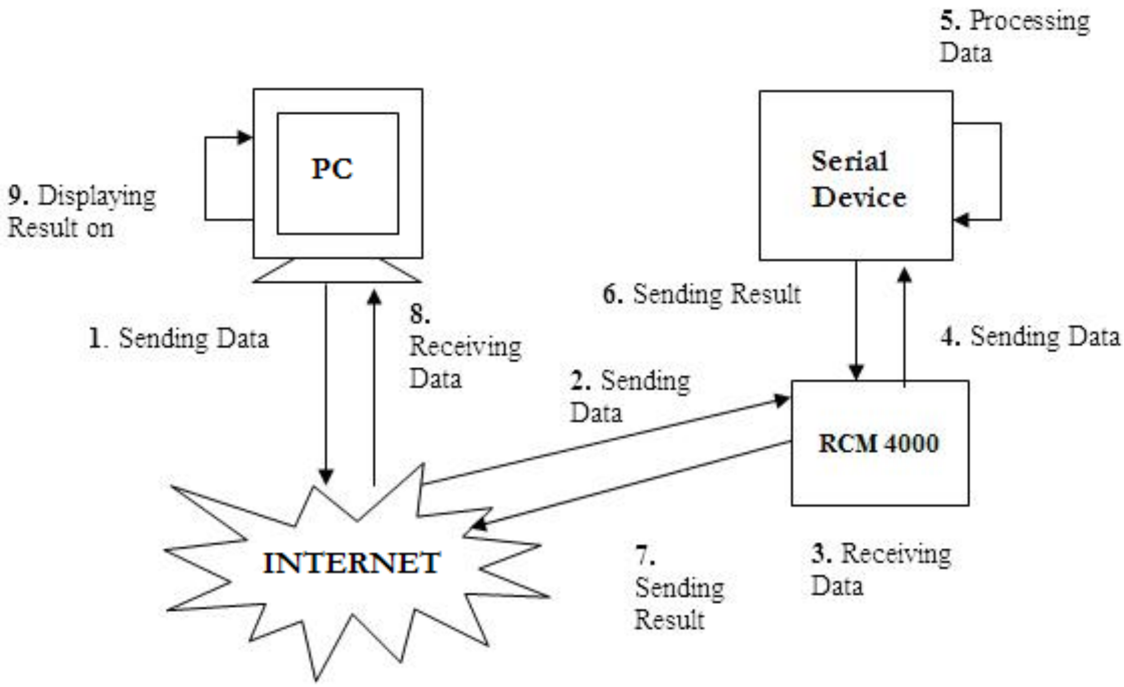

Fig: 3.1 <u>Information flow</u>

According to the pressed button on the GUI at the user end the corresponding key code will be sent to the RCM 4000 from PC connected via internet. The RCM 4000(Serial-Ethernet Module) will form the Serial Data and send it to the Serial device end. The analyzer will use the Serial Data sent by kit and perform the respective operation as per the data (Key Code) sent to it via RCM 4000.

3.3 FUNCTIONAL DESCRIPTION

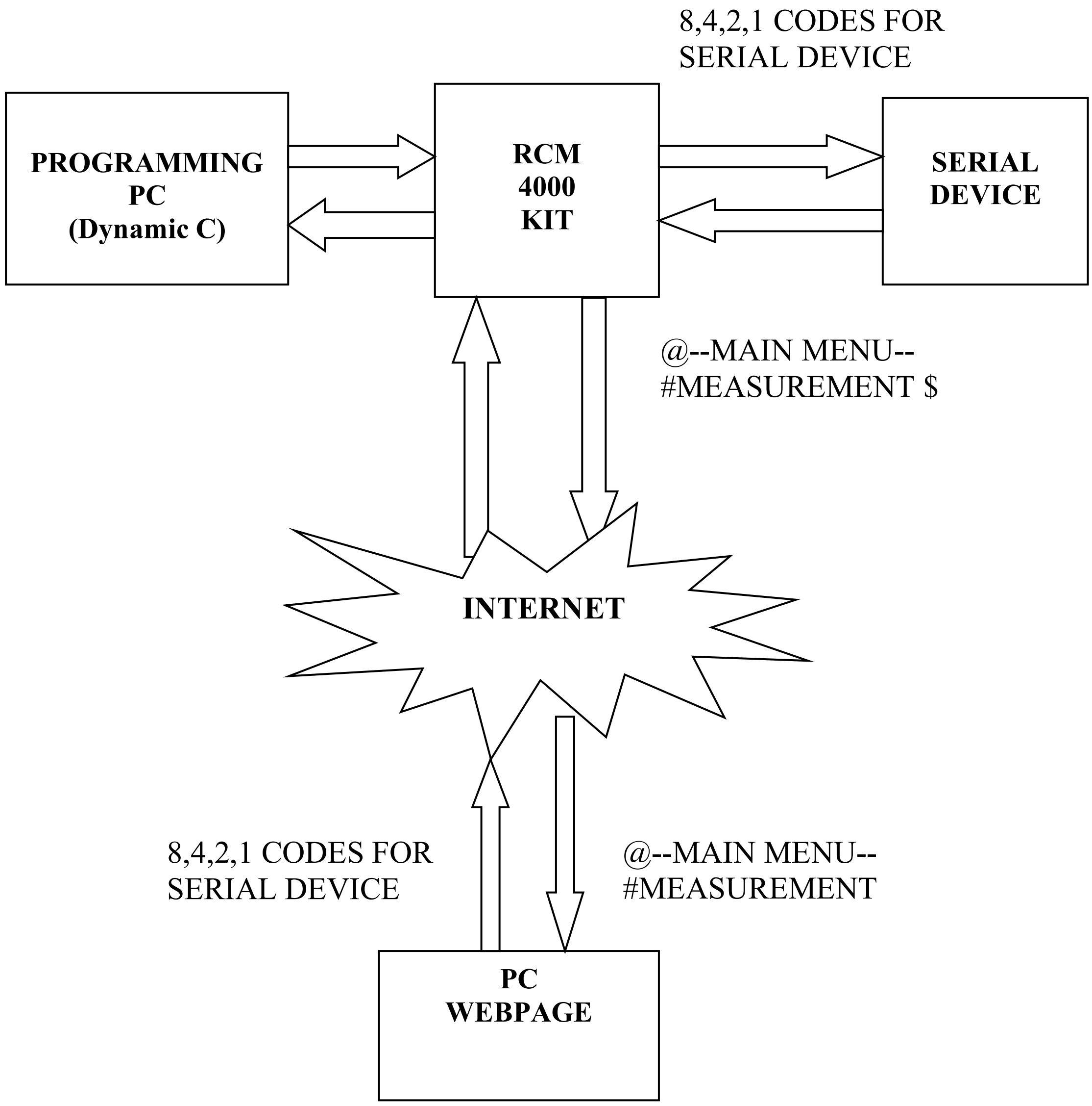

Fig. 3.2 **<u>Functional Working</u>**

3.4 BEHAVIOURAL DESCRIPTION

3.4.1 Coding:

1. Writing Functions in Dynamic ‘c’, using the libraries defined in them at the RCM 4000 Development kit end.
2. Writing and Modifying Functions in ICC AVR Studio, using the libraries defined in them at the serial device end.
3. Writing HTML pages in Interdev.

3.4.2 Documents:

All the following documents are prepared as per the Software engineering Concepts and will be provided with the system.

1. Abstract.
2. Software Requirement Specification.
3. Project Plan.
4. System Design.
5. Test Plan

System States:

1. Receive HTTP request.
2. Parse request.
3. Perform Ethernet to serial conversion.
4. Communicate with serial device.
5. Perform serial to Ethernet conversion.
6. Send HTTP response.
7. Trigger email in case of emergency.

3.4.4 Events and Actions

1. The user can view the status of the Serial device system by accessing the web page via the internet. The GUI of the web page is designed as per the interface of the Serial device. The buttons provided on the GUI will help user navigate through the different options of the Serial device displayed on screen at user end.

2. RCM 4000 application is developed for Serial device to interact with PC connected to internet. It will process the key code sent by the PC. RCM will then communicate through RS 232 with application on Serial device and send the corresponding result to PC.

Chapter Four

Project Plan

4. PROJECT PLAN

4.1 PROJECT ESTIMATES

4.1.1 Hardware:

1. Serial Device.
2. RS 232, RJ 45 Cables.
3. Rabbit Core Module 4000 Developer kit.

4.1.2 Software:

1. User End – Interdev (HTML).
2. Serial Device End – ICC AVR, AVR Studio 4.
3. RCM 4000 Module – Dynamic ‘ C ’ (v10.09)

4.2 PROJECT SCHEDULE

Table 4.1: Monthly Schedule

SNO	MODULE	START DATE	END DATE	NO OF DAYS	DONE BY
1	Comparative study of kits / kit finalization	12/26/06	12/29/06	4 days	Project Team
2	Kit Purchasing	1/2/07	1/23/07	3 weeks	HAIL
3	Getting familiar with the analyzer, Working on existing analyzer communication - PC to analyzer serial communication, understanding existing analyzer firmware	1/2/07	1/15/07	2 weeks	Project Team

4	Ground work on finalizing specifications, Defining scope (preparing Final specification document) - e.g. What all is configurable as a part of this project, DHCP support, email support, static IP support, alarm recipients configuration, password access protection etc, what happens if two users access the analyzer from two different PCs on a network?	1/2/07	1/23/07	3 weeks	Project Team + HAIL
5	Getting functional with the kit	1/22/07	2/5/07	2 weeks	Project Team
6	Implementation - Browser related	2/5/07	3/19/07	6 weeks	Project Team
7	Implementation - Analyzer firmware	2/5/07	3/19/07	6 weeks	Project Team
8	Implementation - Kit firmware	2/5/07	3/19/07	6 weeks	Project Team
9	Testing	3/19/07	4/2/07	2 weeks	Project Team
10	Report Generation and Finalization	4/2/07	4/16/07	2 weeks	Project Team

Chapter Five

Design and Implementation

5. <u>DESIGN AND IMPLEMENTATION</u>

5.1 DESIGN

5.1.1 USECASE DIAGRAMS

Use Case Diagram 1

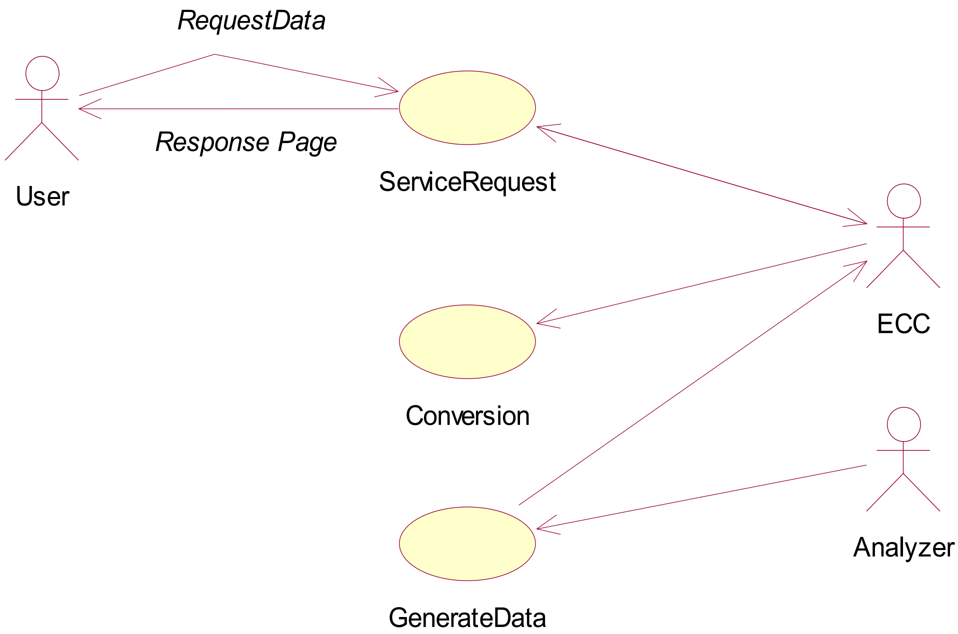

Fig. 5.1 <u>Use Case Level 0</u>

Use Case Diagram 2

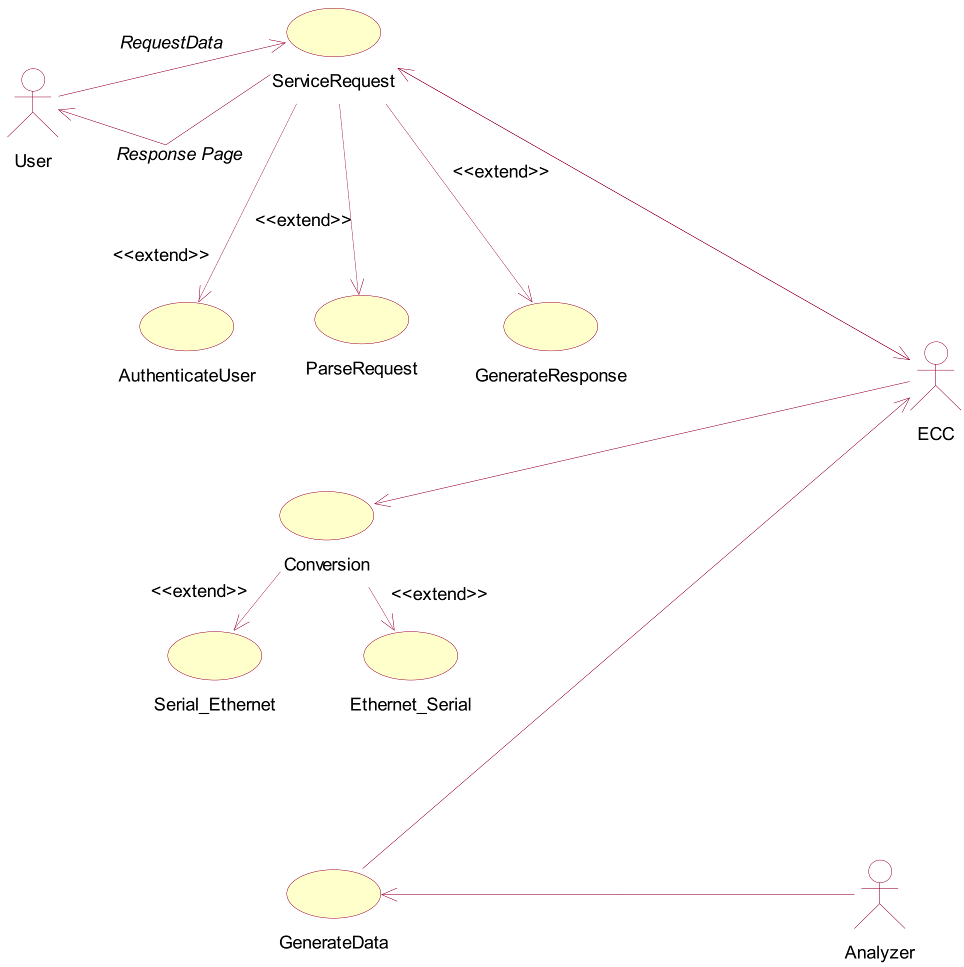

Fig. 5.2 <u>Use Case Level 1</u>

5.1.2 SEQUENCE DIAGRAM

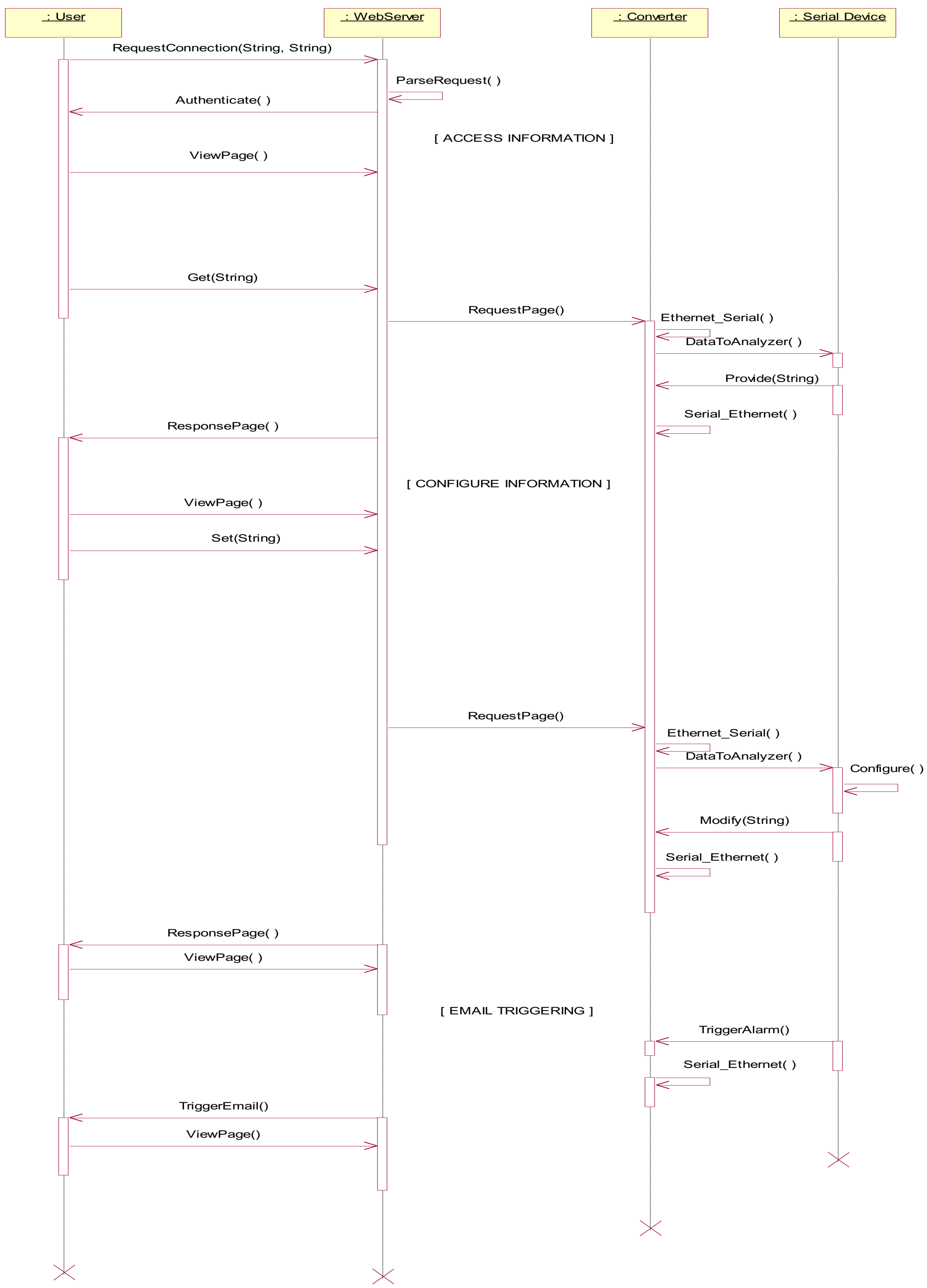

Fig. 5.3 <u>Sequence Diagram</u>

5.1.3 CLASS DIAGRAM

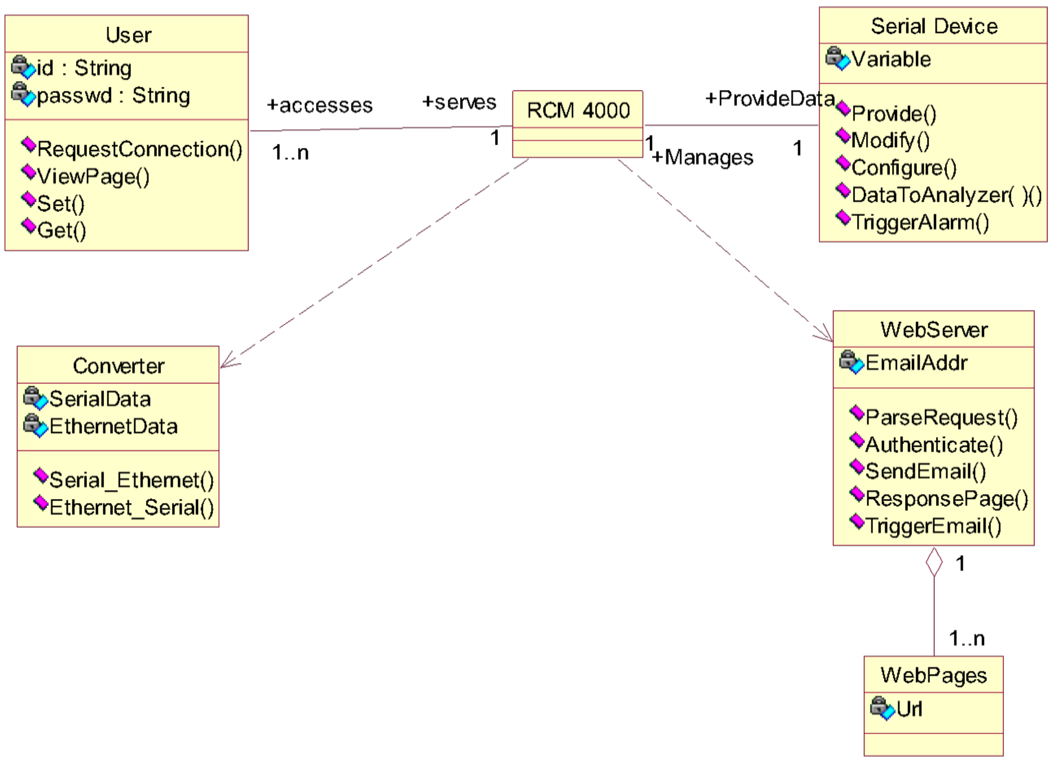

Fig. 5.4 Class Diagram

5.1.4 COLLABRATION DIAGRAM

Reading Parameters

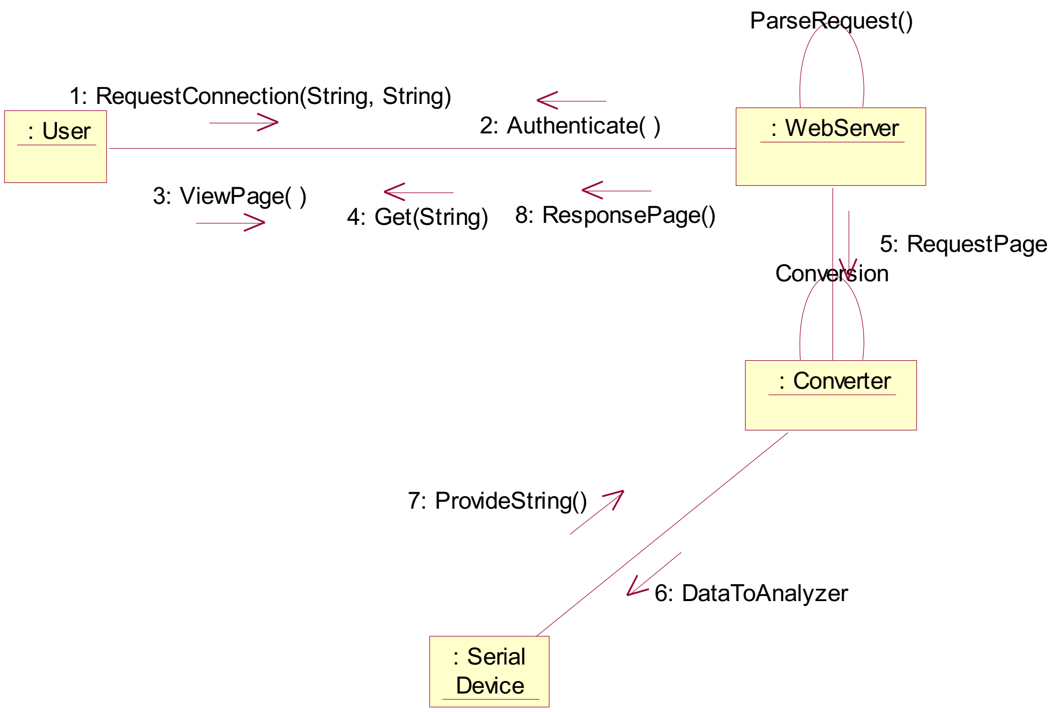

Fig. 5.5 <u>Collaboration Diagram (Reading Parameters)</u>

Configuring Parameters

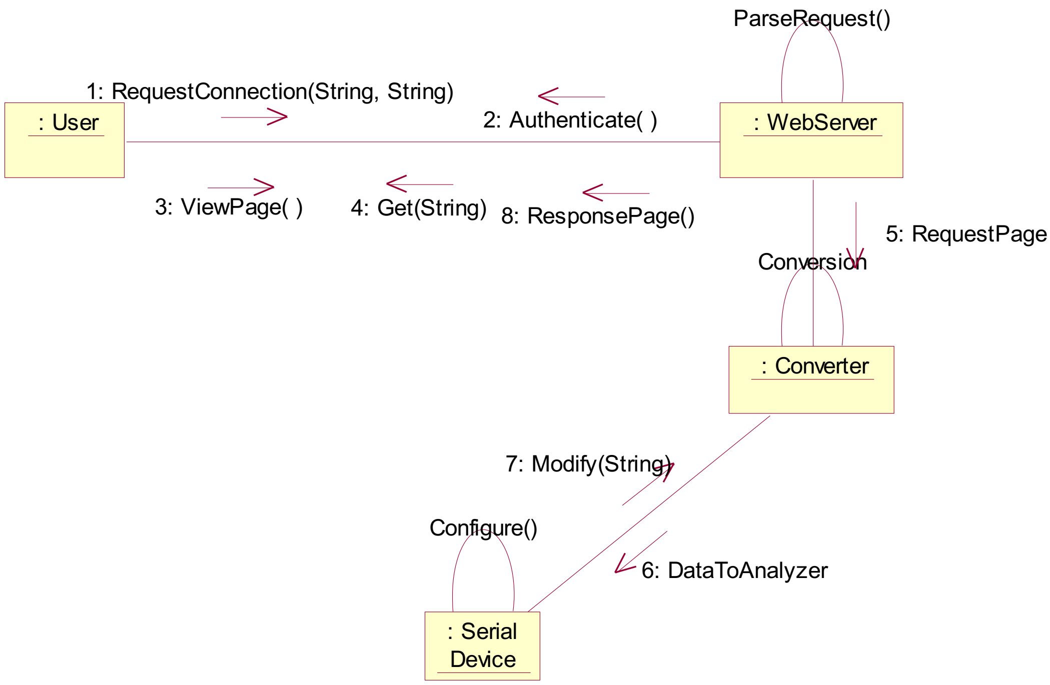

Fig. 5.6 <u>Collaboration Diagram (Configuring Parameters)</u>

Email Triggering

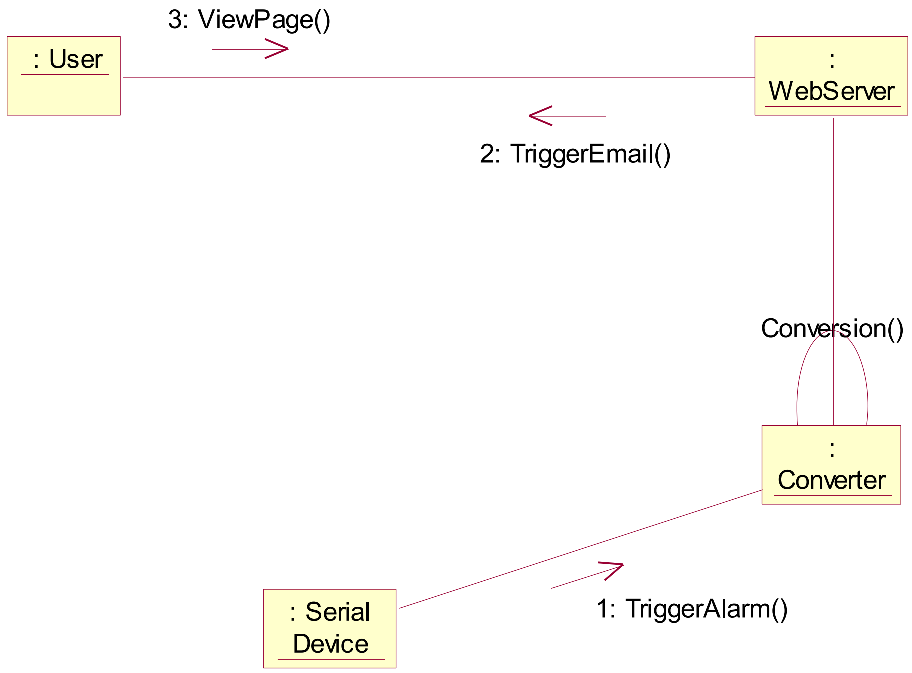

Fig. 5.7 **Collaboration Diagram (Email Triggering)**

5.1.5 DEPLOYMENT DIAGRAM

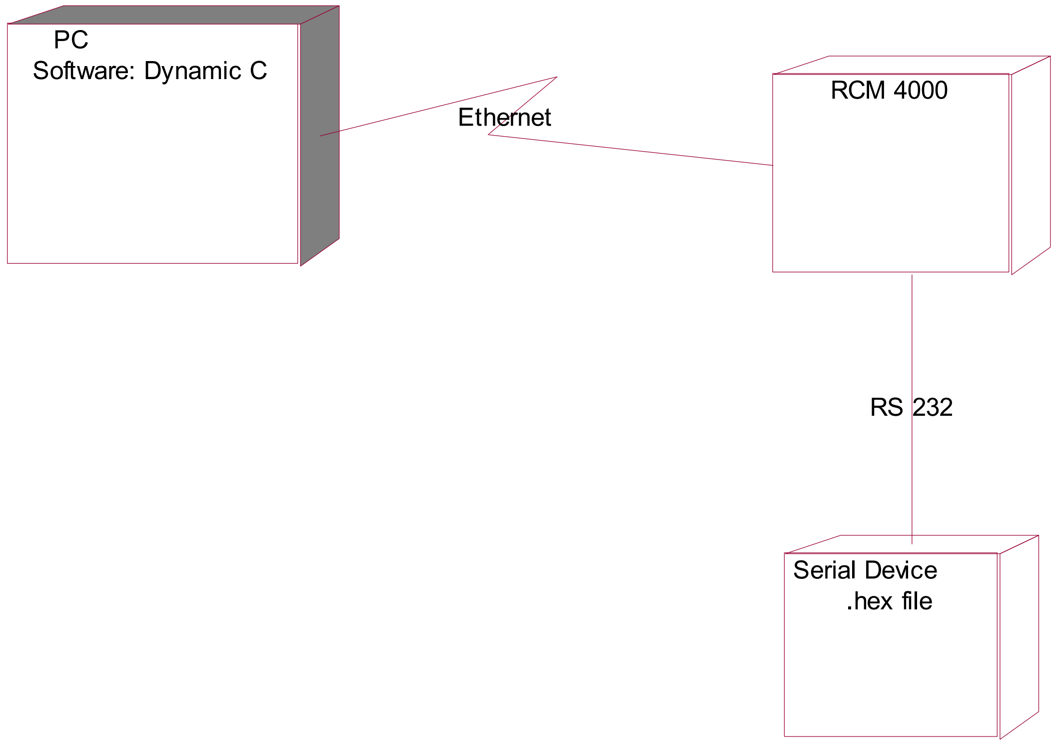

Fig 5.8 Deployment Diagram

5.2 IMPLEMENTATION

5.2.1 CODE EXAMPLES

5.2.1.1 SERIAL DEVICE CODE

```
/*-------------------------------------------------------------------------------------------

        Functional Description:
        Sending data displayed on LCD's MAIN_DISP to RCM 4000 Kit

-------------------------------------------------------------------------------------------*/
        for(i=0;i<5;i++)
                chSend_buffer1[i]='\0';

        chSend_buffer1[0]='~';          // Prefixing code for Main Display
        i_cnt = 1;
        while(i_cnt<4)// Collecting Displayed data into buffer
        {
                chSend_buffer1[i_cnt]=*chStore++;
                        i_cnt++;
         }

        i_cnt=0;
        while(i_cnt<4)
        {
                if(buff1[i_cnt]!=chSend_buffer1[i_cnt])
                {
                        compare_flag1=1;
                }
                i_cnt++;
        }
        if(compare_flag1==1)
        {
```

```
                i_cnt=0;
                while(i_cnt<4)
                {
                        buff1[i_cnt]=chSend_buffer1[i_cnt];
                        i_cnt++;
                }
                USART_Transmit_Buffer(chSend_buffer1,4);
        }

        for(i=0;i<4000;i++)
  {
  WDR();
  }
/*--------------------------------------------------*/
                break;

  case MSG_1:

          cstrcpy(buffer,"        ");
          Disp_TxtMsg_L(buffer, BLANK_PAGE, SET_YADD, 8, 0, 0);
          Disp_TxtMsg_R(buffer, BLANK_PAGE, SET_YADD, 8, 0, 0);

          if(chCurrStart <= 8)
          {
                Disp_TxtMsg_L(pchDispData, PG_ADD_MSG1, SET_YADD,
8, chCurrStart, chCurrEnd);
           Disp_TxtMsg_R(pchDispData + 8, PG_ADD_MSG1, SET_YADD,
8, 0, 0);
          }
          if(chCurrStart > 8)
          {
                Disp_TxtMsg_L(pchDispData, PG_ADD_MSG1, SET_YADD,
8, 0, 0);
```

```
            Disp_TxtMsg_R(pchDispData + 8, PG_ADD_MSG1, SET_YADD,
8, chCurrStart - 8, chCurrEnd - 8);
            }

/*------------------------------------------------------------------------------------------


            Functional Description:
            Sending data displayed on LCD's MSG_1 to RCM 4000


-------------------------------------------------------------------------------------------*/
                for (i=0;i<18;i++)
                        chSend_buffer2 [i] ='\0';

                chSend_buffer2 [0] = '@';
             i_cnt=1;
             While (i_cnt<=17)
             {
                  chSend_buffer2 [i_cnt] =*chMsg1++;
                   I_cnt++;
             }
                i_cnt = 0;
                while (i_cnt<=17)
                {
                  if (buff2[i_cnt]!=chSend_buffer2[i_cnt])
                        {
                compare_flag2=1;
                         }
                        i_cnt++;
                }
               If (compare_flag2==1)
               {
                         i_cnt=0;
                         While (i_cnt<=17)
                         {
```

```
                    buff2 [i_cnt]=chSend_buffer2[i_cnt];
                        I_cnt++;
                    }
                    USART_Transmit_Buffer (chSend_buffer2, 18);
            }

        for (i=0;i<4000;i++)
        {
        WDR ();
        }
        /*---------------------------------------------------*/
        break;

    case MSG_2:

        cstrcpy (buffer,"        ");
        Disp_TxtMsg_L(buffer, BLANK_PAGE, SET_YADD, 8, 0, 0);
        Disp_TxtMsg_R(buffer, BLANK_PAGE, SET_YADD, 8, 0, 0);

        if (chCurrStart <= 8)
        {
            Disp_TxtMsg_L (pchDispData, PG_ADD_MSG2, SET_YADD,
8,                                                        chCurrStart,
chCurrEnd);
         Disp_TxtMsg_R (pchDispData + 8, PG_ADD_MSG2, SET_YADD,
8, 0, 0);
        }
        If (chCurrStart > 8)
        {
            Disp_TxtMsg_L(pchDispData,  PG_ADD_MSG2,  SET_YADD,
8, 0, 0);
         Disp_TxtMsg_R (pchDispData + 8, PG_ADD_MSG2, SET_YADD,
8, chCurrStart - 8, chCurrEnd - 8);
        }
```

```
/*-----------------------------------------------------------------------------------------

        Functional Description:
        Sending data displayed on LCD's MSG_2 to RCM 4000 Kit
-----------------------------------------------------------------------------------------*/
//          icount=2;

      for(i=0;i<19;i++)
                        chSend_buffer3[i]='\0';



           //chSend_buffer3[0] ='#';
                  //chSend_buffer3[0] = '#';


     chSend_buffer3[1] ='#';
                  chSend_buffer3[1] = '#';


               i_cnt= 2;
           While (i_cnt<=17)
      {
        chSend_buffer3 [i_cnt] =*chMsg2++;
                        i_cnt++;
           }


      i_cnt=0;
               while(i_cnt<=17)
               {
                 if (buff3[i_cnt]!=chSend_buffer3[i_cnt])
                     {
               compare_flag3=1;
                     }
                     i_cnt++;
               }
```

```
                if (compare_flag3==1)
                {
                        i_cnt=0;
                        while(i_cnt<=17)
                        {
                          buff3[i_cnt]=chSend_buffer3[i_cnt];
                                i_cnt++;
                        }
                        chSend_buffer3[18]='$';
          USART_Transmit_Buffer(chSend_buffer3,19);

          //chSend_buffer3[0] = '$';
          //USART_Transmit_Buffer(chSend_buffer3,1);
                }

                for(i=0;i<4000;i++)
           {
          WDR();

           }
          /*-----------------------------------------------------*/

    break;
    default:
      break; }
    }
```

5.2.2 SOURCE CODE (Dynamic C)

Function Description

1. **sock_init();**

   DESCRIPTION: This function initializes the packet driver and DCRTCP using the compiler defaults for configuration. This function should be called before using other DCRTCP functions.

2. **http_init();**

DESCRIPTION: Initialize the http daemon. This must be called after sock_init (), and before calling http_handler () in a loop. This sets the root directory to "/" and sets the default file name to "index.html".You can change these Defaults by calling http_set_path () after this function.

3. **cgi_redirectto();**

SYNTAX: int cgi_redirectto (HttpState* state, char* url);

DESCRIPTION: Utility function that may be called in a CGI Function, to redirect the user to another page.

4. **serDputc();**

SYNTAX: int serDputc (char c);

DESCRIPTION: Write a character to serial port D write buffer.

5. **serDopen();**

SYNTAX: int serDopen (long baud);

DESCRIPTION: Opens the D serial port.

6. **http_handler();**

SYNTAX: int http_handler (void);

DESCRIPTION: Tick function to run the http daemon. Must be Called periodically for the daemon to work.

7. **serDgetc();**

SYNTAX: int serDgetc();

DESCRIPTION: Get next available character from serial port D read buffer

8. **sspec_adduser();**

SYNTAX: int sspec_adduser (int sspec, int userid);

DESCRIPTION: Add to the read permission mask for the given resource. The group(s) which 'userid' is a member of are ORed in to the existing permission mask for the resource. The write permissions are not modified.

9. smtp_sendmail();

SYNTAX: void smtp_sendmail (char* to, char* from, char* subject, char*message)

DESCRIPTION: Start an e-mail being sent. This function is intended to be used for short messages which are entirely constructed prior to being sent.

```
/*
	FUNCTION DESCRIPTION:
		1) Function to send an email.
		2) Makes use of SMTP.lib library.
		3) Sends an E-Mail in case of emergency.
		   Eg: when the alarm levels for serial device are not within the
		   specified range.
*/

void sendmail(char two[30])
{

		smtp_sendmail(two, FROM, SUBJECT, BODY);

		while(smtp_mailtick()==SMTP_PENDING)
		continue;

		if(smtp_status()==SMTP_SUCCESS)
		{
		printf("Message sent\n");
		}
		else
		{
		//printf("Error sending message\n");
		}
```

```
}


/*
    FUNCTION DESCRIPTION:
        1) Sends 8 to the serial device firmware
        2) Redirects to the main web page (main.shtml) when “MENU” is pressed on
           the GUI.
*/
int menutoggle(HttpState* state)
{

  serDputc('8');
  flag=1;

  // Return an HTTP response that advises the client to request an updated Web page.
  cgi_redirectto(state,REDIRECTTO);
  return 0;
}

/*
    FUNCTION DESCRIPTION:
        1) Sends 2 to the serial device firmware
        2) Redirects to the main web page (main.shtml) when “SIDE ARROW” is
           pressed on the GUI.
*/
int sdtoggle(HttpState* state)
{

  serDputc('2');

  nIn1=serDread(&prev,100,600);
  printf("\n%s",prev);
  nIn1=serDread(&prev,100,600);
```

```
  flag=0;

  // Return an HTTP response that advises the client to request an updated Web page.
  cgi_redirectto(state,REDIRECTTO);
  return 0;
}

/*
    FUNCTION DESCRIPTION:
        1) Sends 4 to the serial device firmware
        2) Redirects to the main web page (main.shtml) when "DOWN ARROW" is
           pressed on the GUI.
*/
int dntoggle(HttpState* state)
{

  serDputc('4');
  flag=1;

  // Return an HTTP response that advises the client to request an updated Web page.
  cgi_redirectto(state,REDIRECTTO);
  return 0;}
/*
    FUNCTION DESCRIPTION:
        1) Sends 1 to the serial device firmware
        2) Redirects to the main web page (main.shtml) when "ENTER" is pressed on
           the GUI.
        3) Triggers E-MAIL when unexpected condition occurs.
*/
int entoggle(HttpState* state)
{

  serDputc('1');
  flag=1;
```

```
if((strstr(data3,"HIGH")==NULL) && (strstr(data3,"LOW")==NULL))
{
 if((strstr(data3,"LO")!=NULL) && (strstr(data3,"HI")==NULL))
 {
     k=0;
     while(1)
     {
         if(isdigit(data3[k]))
         {
           dg1[0]=data3[k];
           k++;
           dg1[1]=data3[k];
           k++;
           dg1[2]=data3[k];
           k++;
           dg1[3]='\0';
           k++;

           dg2[0]=data3[k];
           k++;
           dg2[1]=data3[k];
           k++;
           dg2[2]='\0';
           break;
         }
         k++;
     }

     d1=atoi(&dg1);
           d2=atoi(&dg2);

     if(d1>100)
     {
```

```
            memset(BODY,'\0',strlen(BODY));
            strcpy(BODY,"LOW alarm value is...");
            strcat(BODY,dg1);
            strcat(BODY,".");
            strcat(BODY,dg2);
            sendmail(TO1);
            sendmail(TO2);
            sendmail(TO3);
            sendmail(TO4);
        }
}

if((strstr(data3,"HI")!=NULL) && (strstr(data3,"LO")==NULL))
{

    k=0;
    while(1)
    {
        if(isdigit(data3[k]))
        {
            dg1[0]=data3[k];
            k++;
            dg1[1]=data3[k];
            k++;
            dg1[2]=data3[k];
            k++;
            dg1[3]='\0';
            k++;//for .

            dg2[0]=data3[k];
            k++;
            dg2[1]=data3[k];
            k++;
            dg2[2]='\0';
```

```
                break;

            }
            k++;
        }

        d1=atoi(&dg1);
        d2=atoi(&dg2);

        if(d1>100)
        {
            memset(BODY,'\0',strlen(BODY));
            strcpy(BODY,"HIGH alarm value is...");
            strcat(BODY,dg1);
            strcat(BODY,".");
            strcat(BODY,dg2);

            sendmail(TO1);
            sendmail(TO2);
            sendmail(TO3);
            sendmail(TO4);
        }
    }

 }
 // Return an HTTP response that advises the client to request an updated Web page.
 cgi_redirectto(state,REDIRECTTO);
 return 0;
}
```

```
/*
        PROGRAM CODE:
                Parsing of data coming from Serial Device and putting it into variables
                that will be reflected on the Web Page.
*/

while (1)
  {

        if(flag==1)     //parsing of the data starts here
        {
          while ((nIn1=serDgetc()) == -1);    //Gets data character by character
                                                          from Serial Port
          ch=(char)nIn1;

          if((ch>='A' && ch<='Z') || (ch=='*')|| (ch=='-') ||(ch=='')||(ch=='0')||(ch=='1')||
            (ch=='2')||(ch=='3')||(ch=='4')||(ch=='5')|| (ch=='6')||(ch=='7')||(ch=='8')||
            (ch=='9')||(ch==':')||(ch == '/')||(ch == '.')|| (ch >= 'a' && ch <= 'z')||(ch ==
'&'))

         dt[i++]=ch;

          /*if(ch=='~')    //for ovr etc
          {
              aflag=1;
          }
          if(aflag==1)
          {
            if(ch=='@')
            {
                dt[i]='\0';
                i=0;
                k=0;
                while(dt[i]!='\0')
```

```
        data1[k++]=dt[i++];

        data1[k]='\0';
        i=0;
        flag=0;
        aflag=0;
    }
  } */

 if(ch=='œ')
 { //if 1
      dt[i]='\0';
      i=0;
      k=0;

      while(dt[i]!='\0')
      data3[k++]=dt[i++];

      data3[k]='\0';
      i=0;
      flag=0;
 } //if 1

 else if(ch=='#')
 {        //if 2
      dt[i]='\0';
                  i=0;
                  k=0;
      while(dt[i]!='\0')
      data2[k++]=dt[i++];
                  data2[k]='\0';
                  i=0;
 } //if 2
```

```
else if(ch=='$')
{      //if 3
      dt[i]='\0';
      i=0;
      k=0;

      while(dt[i]!='\0')
      data3[k++]=dt[i++];

      data3[k]='\0';
      i=0;
      flag=0;
      if((strstr(data3,"HIGH")==NULL) && (strstr(data3,"LOW")==NULL))
      {            //if high
        if((strstr(data3,"HI")!=NULL) || (strstr(data3,"LO")!=NULL))
        {           //if hi
          memset(data2,'\0',strlen(data2));

          if((strstr(data3,"HI")!=NULL) && (strstr(data3,"LO")==NULL))
          {
                i=0;
                while(data3[i]!=':')
                {
                  i++;
                }
                i++;

                if(data3[i]=='*')
                flag=0;

                else
                flag=1;
          }
          else
```

```
        {
            i=0;
            while(1)
            {
                while ((nIn1=serDgetc()) == -1);
                ch=(char)nIn1;

      if((ch>='A' && ch<='Z') || (ch=='*')|| (ch=='-')
||(ch=='')||(ch=='0')||(ch=='1')||
        (ch=='2')||(ch=='3')||(ch=='4')||(ch=='5')||(ch=='6')||(ch=='7')||(ch=='8')||
        (ch=='9')||(ch==':')||(ch == '/')||(ch == '.')|| (ch >= 'a' && ch <= 'z')||(ch == '&'))

                dt[i++]=ch;

                if(ch=='œ')
                {
                  dt[i]='\0';
                  i=0;
                  k=0;
                  strcat(data3,dt);
                  break;
                }
            }
            flag=1;
        }
      }//if hi

      else
      {
              flag=0;
      }

    } //if high
```

```
            } //if 3

         }//if  flag      //parsing of the data ends here

   }//while

/*
         PROGRAM CODE:
                  Define user's name and password and associates them with the web page
                  for authentication purpose.
*/

  http_setauthentication (HTTP_BASIC_AUTH);

  user1_enabled = 1;
  user1 = sauth_adduser("rmoi", "rmoi", SERVER_HTTP);

  page1 = sspec_addxmemfile("/", index_html, SERVER_HTTP);
  sspec_adduser(page1, user1);
  sspec_setrealm(page1, "Admin");

  page2 = sspec_addxmemfile("index.html", index_html, SERVER_HTTP);
  sspec_adduser(page2, user1);
  sspec_setrealm(page2, "Admin");
```

PROGRAM CODE: HTML Page : “main.shtml”

```
<HTML>
    <HEAD>
         <TITLE>Remote Monitoring Over Internet</TITLE>
    </HEAD>

<BODY>

<center>
```

```
<TABLE style="WIDTH: 523px; HEIGHT: 400px" cellSpacing=1 cellPadding=2
width=523 bgColor=steelblue border=2>

 <TR>
        <TD><FONT color=white size=10>

                <CENTER>SERIAL DEVICE</CENTER></FONT>

        </TD>
 </TR>



 <TR>
        <TD>

        <P><center> </center>
        <CENTER> </CENTER>
        <CENTER>
        <INPUT style="FONT-SIZE: xx-large; WIDTH: 312px; HEIGHT: 81px"
        size=39 name=text1 border=0 value="OVR" readonly></CENTER>
        <CENTER>
        <INPUT id=text1 style="FONT-SIZE: large; WIDTH: 312px; HEIGHT: 41px"
        size="19" name=text2 border=0 value="<!--#echo var="data2"-->"
        readonly></CENTER>
        <CENTER>

        <INPUT id=text2 style="FONT-SIZE: larger; WIDTH: 312px; HEIGHT: 41px"
        size=39 name=text3 border=0 value="<!--#echo var="data3"-->"
        readonly></CENTER>



        <CENTER> </CENTER>
        <CENTER> </CENTER>
```

```
        </TD>
 </TR>

 <TR>
   <TD>

     <P style="FONT-SIZE: larger"> </P>
     <P>
     <TABLE style="WIDTH: 506px; HEIGHT: 98px" cellSpacing=4 cellPadding=1
     width=506 border=2>
       <TR>
         <TD><A href="/menutoggle.cgi"><IMG style="BORDER-LEFT-COLOR:
         steelblue; BORDER-BOTTOM-COLOR: steelblue; BORDER-TOP-COLOR:
         steelblue; BORDER-RIGHT-COLOR: steelblue" src="mn.gif" >



         </A>
         </TD>
        <TD><A href="/dntoggle.cgi"><IMG style="BORDER-LEFT-COLOR:
        steelblue; BORDER-BOTTOM-COLOR: steelblue; BORDER-TOP-COLOR:
        steelblue; BORDER-RIGHT-COLOR: steelblue" src="dn.gif" >
        </A>
        </TD>

        <TD><A href="/sdtoggle.cgi"><IMG style="BORDER-LEFT-COLOR:
        steelblue; BORDER-BOTTOM-COLOR: steelblue; BORDER-TOP-COLOR:
        steelblue; BORDER-RIGHT-COLOR: steelblue" src="sd.gif" >
        </A>
        </TD>

        <TD><A href="/entoggle.cgi"><IMG style="BORDER-LEFT-COLOR:
        steelblue; BORDER-BOTTOM-COLOR: steelblue; BORDER-TOP-COLOR:
        steelblue; BORDER-RIGHT-COLOR: steelblue" src="en.gif" >
```

```
          </A>
          </TD>
        </TR>
      </TABLE></P>
    <P> </P>
     </TD>
  </TR>
</TABLE>
<P> </P></center>
<FORM action="reg.shtml" method=post>
           <INPUT type=submit value=REGISTER>
</FORM>
</BODY>
</HTML>
```

5.3 GUI SNAPSHOTS

1. User gets the following screen, when the system gets initialized after authentication.

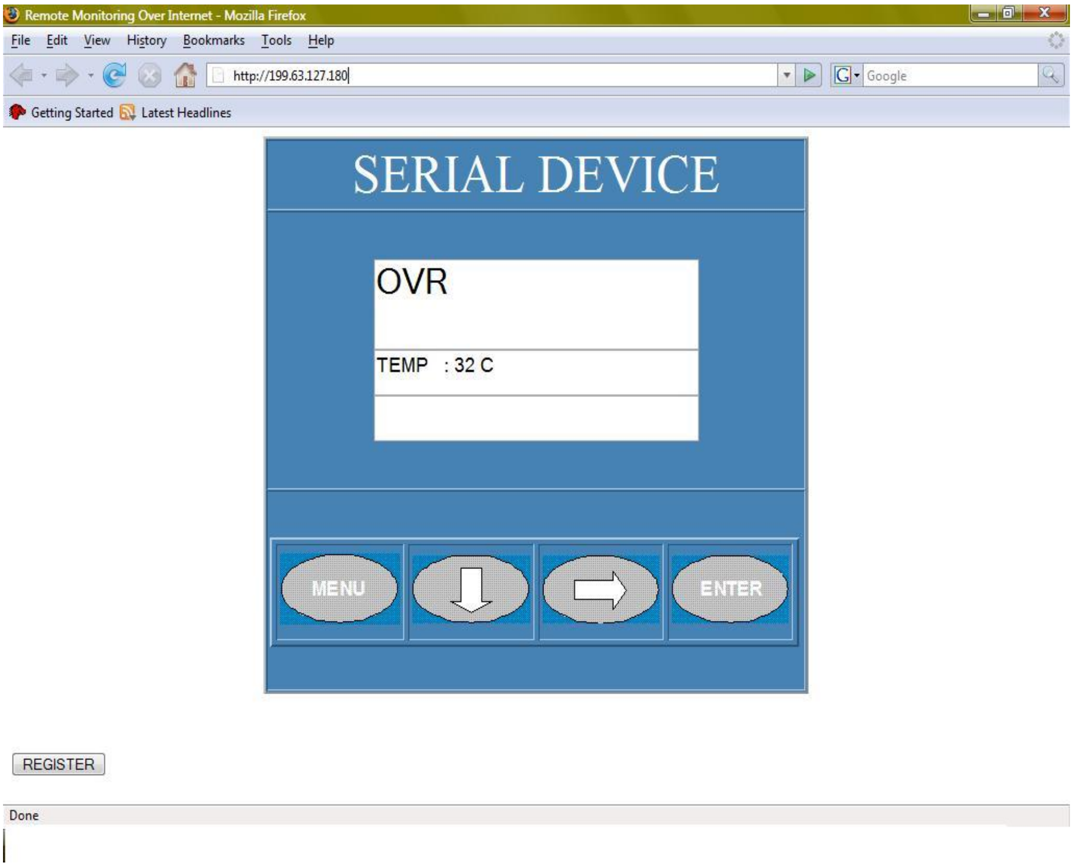

Fig 5.9 <u>Serial device Initialization Display Screen</u>

2. When User Presses the Register Button Following Screen Gets Displayed And user Enter the Email addresses of the concern person.

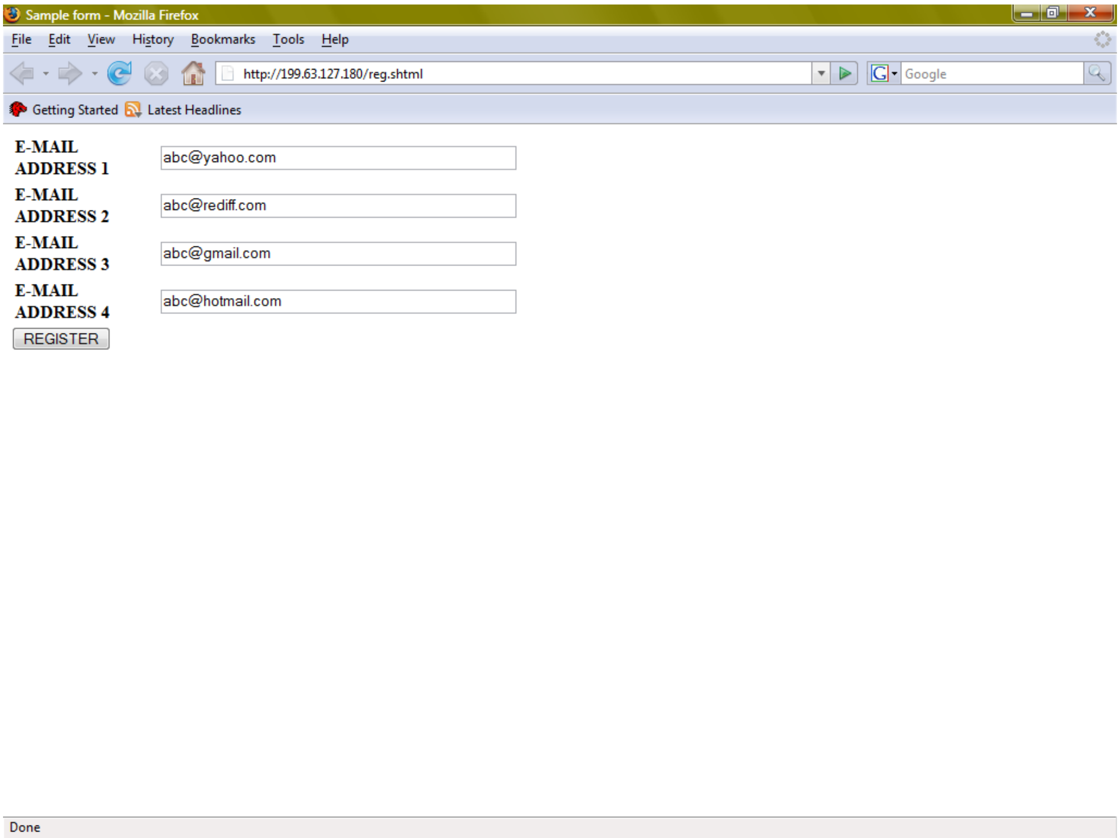

Fig 5.10 <u>E-mail Registration Form</u>

3. After the registration is over and on click of the register button this page is displayed (Home Page).

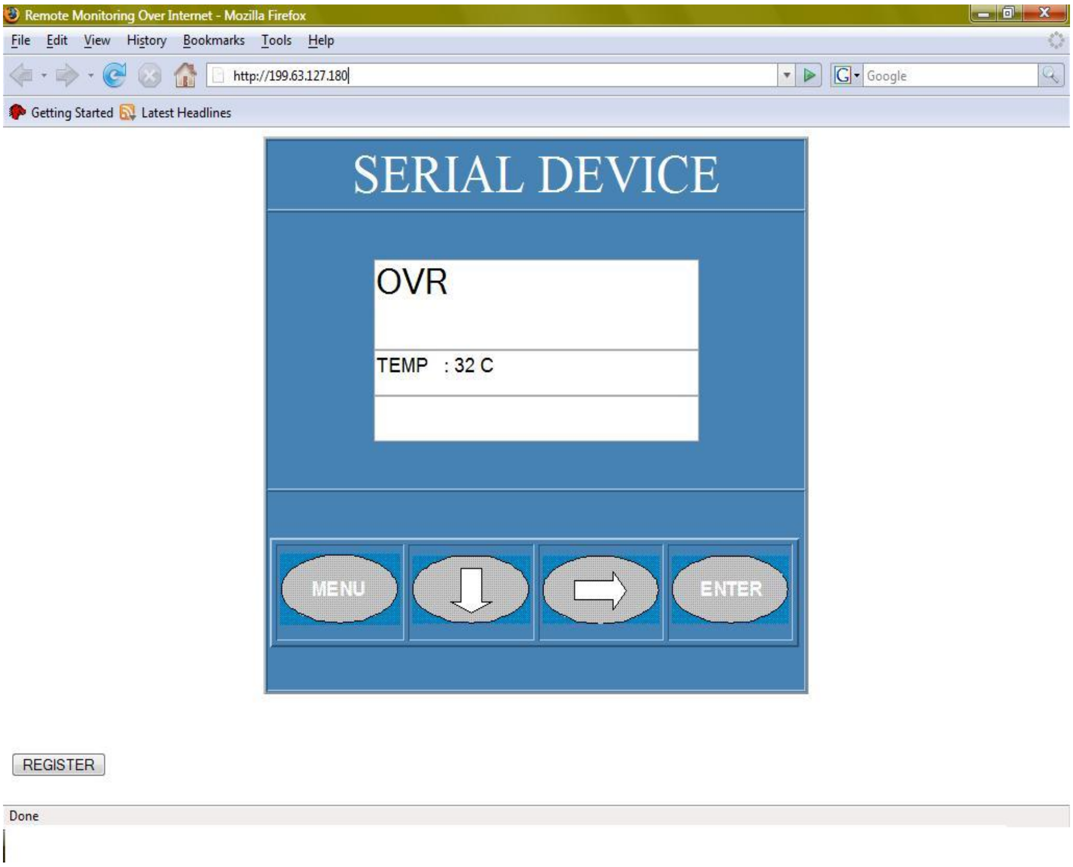

Fig 5.11 <u>Initialized screen Form</u>

4. When the User Presses Menu Button the Following Screen is displayed.

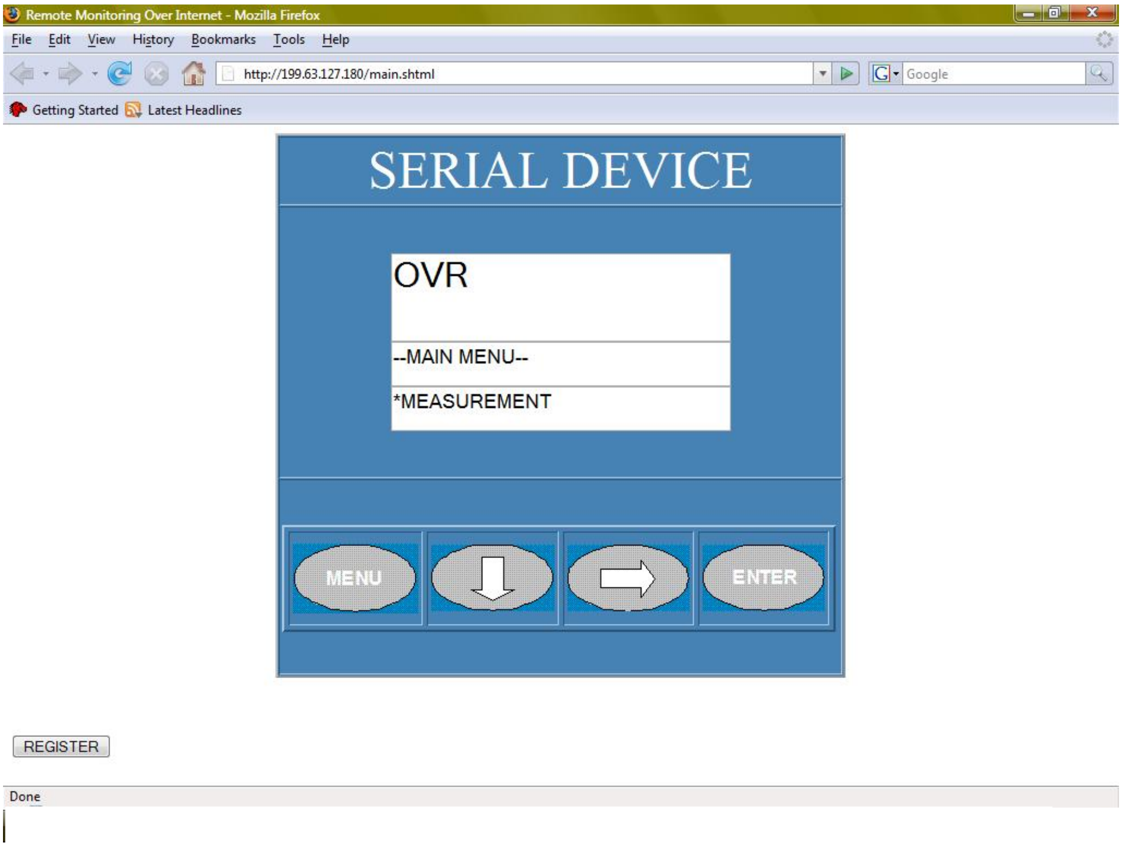

Fig 5.12 MAIN MENU form

Chapter Six

Test Plan

And

Reports

6. TEST PLAN

6.1 OBJECTIVE

The software test plan is designed to prescribe the scope, approach, resources and schedules of all testing activities. To check the system across it's functionality.

Objective

The main purpose of the system is to monitor the Serial Data remotely. So the system will be tested for functionality like data transfer from PC to RCM4000 and RCM4000 to Serial Data and vice versa.

Testing Strategies

Testing is the process of analyzing software items to detect the differences between existing and required conditions and to evaluate features of the software Items.

Purpose of the test:

To check the system for proper functionality.

Items to be tested

- Interfaces
- Code

Features to be tested

Consistency

Scope

In the existing environment, all operations of Serial Data are done by directly interacting with it. Here the user needs to press the buttons on the analyzer to carry out required functions. This system will replace the direct interactions by computerized interface. The analyzer's interface will be simulated on the computer screen. From this screen the user will interact with the actual analyzer.

In this system user will click the MENU button provided on the GUI and will select required option. This action will transfer the corresponding key code to the analyzer wirelessly. The analyzer will get this key code on its serial port. It will process the key code and will display corresponding output. This output will send back to computer interface. The user sitting at that place will take next appropriate actions.

Reference Material

- Abstract
- Software requirement specification

6.2 TEST ITEMS

6.2.1 Program Modules

Code

1. Functions for sending data on a serial port (for both PC and SERIALDEVICE).
2. Functions for receiving data from serial port (for both PC and SERIALDEVICE).
3. Functions for Parsing.
4. Functions for authenticating Users.

Operator Procedures

For testing, the system will be first installed to check whether it works properly in the given environment conditions. Those conditions could be like when we install system where it would be used and also on different versions of windows.

6.3 FEATURES TO BE TESTED

1) To check whether data received is consistent.
2) To check interfacing between PC and RCM 4000 Kit
3) To check interfacing between Serial Device and RCM 4000 kit

6.4 FEATURES NOT TO BE TESTED

Documentation

6.5 APPROACH

6.5.1 Module Testing

In this testing, each module is checked after adding code needed to modify that Module. Testing is done such as every statement of each program is executed at least once. For do-while, for, if statements are tested for every loop.

6.5.2 Integration Testing

This test is mainly conducted to uncover errors associated with interfacing each module with another, though individual module work well. All modules are checked together according to program specifications. Bottom-Up integration testing is used.

6.5.3 Regression Testing

Test report which is generated during the testing process will be used by the developing team to correct faults occurred in the system. So the changes will be introduced in the system. To check whether the system works fine or not after changes introduced, regression testing will be done.

6.5.4 System Testing

In the system testing the entire system is checked whether it is giving desired Output.

6.6 PASS OR FAIL CRITERIA

The data transfer between PC and Analyzer should be as per specified requirement.

6.7 TESTING PROCESS

6.7.1 Resources

For testing purpose different resources like hardware and software resources as Specified in software requirement specification will be given along with that user manual in the form of word document will be provided to the testing group.

6.8 ENVIRONMENTAL REQUIREMENTS

6.8.1 Software Requirements

- InterDev – for developing application at user end
- ICC AVR – for writing the Analyzer code
- Dynamic ‘C’ – for programming RCM 4000 Kit.

6.9 TEST CASES

6.9.1 UNIT TESTING AT USER END

Table 6.1: Unit Testing

No.	Conditions to be tested	Expected Results	Actual Results
1	MENU button pressed	The data on serial device screen should be displayed on GUI	The data on serial device screen seen on GUI
2	DOWN button pressed	The data on serial device screen should be displayed on GUI	The data on serial device screen seen on GUI
3	SIDE button pressed	The data on serial device screen should be displayed on GUI	The data on serial device screen seen on GUI
4	ENTER button pressed	The data on serial device screen should be displayed on GUI	The data on serial device screen seen on GUI

6.9.2 FUNCTIONAL TESTING AT USER END

Table 6.2: Functional Testing at User End

No.	Conditions to be tested	Expected Results	Actual Results
1	Click on MENU	The key code ‘8’ should get transmitted.	The key code ‘8’ getting transmitted.
No.	Conditions to be tested	Expected Results	Actual Results
2	Click on DOWN	The key code ‘4’ should get transmitted.	The key code ‘4’ getting transmitted.

3	Click on SIDE	The key code '2' should get transmitted.	The key code '2' getting transmitted.
4	Click on ENTER	The key code '1' should get transmitted.	The key code '1' getting transmitted.
5	Received data	The Received data prefixed by '~' should be displayed in first text box, The Received data suffixed by '#' and 'œ' (specially for alarm values) should be displayed in second text box '$' should be displayed in third text box	The received data is getting displayed at the specified positions.

6.9.3 FUNCTIONAL TESTING OF CODE WRITTEN FOR RCM4000

Table 6.3: Functional Testing of code written for RCM4000

No.	Conditions to be tested	Expected Results	Actual Results
1	Transmitting data to serial device	It should collect Ethernet data sent by PC.It should convert Ethernet data to serial data. It should transfer this data to the serial device.	It collects Ethernet data sent by PC.It converts Ethernet data to serial data. It transfers this data to the serial device.
2	Transmitting data to PC	It should collect serial data sent by serial device from serial port of device. It should convert serial data to Ethernet format. It should transfer this data to the PC.	It collects serial data sent by serial device from the serial port. It converts serial data to Ethernet format. It transfers this data to the PC.
3	Triggering Email	When kit receives alarm	It sends email to four

		values from serial device, it should send email to four fixed email addresses.	fixed email addresses.

6.9.4 FUNCTIONAL TESTING OF CODE WRITTEN FOR SERIAL DEVICE

Table 6.4: Functional Testing of code written for serial device

No.	Conditions to be tested	Expected Results	Actual Results
1	Receive routine is receiving data	It should collect data from USART.If data is 8/4/2/1 then set flag to recognize key is received remotely.	It is collecting data from USART. It is setting flag to recognize key is received remotely if data is 8/4/2/1.
2	Key process is checking received data is key.	It should set key recognition flag true and key press event as per key received	It is setting key recognition flag true and key press event as per key received
3	Print_LCD is getting correct data to be displayed on PC and calling function to transmit it on serial port.	It should collect data to be displayed on PC, prefix it with predefined character '~' for first line or '@' for second line or '#' for third line. It should call function to transmit this data on serial port.	It is collecting data to be displayed on PC, prefixing it with predefined character. It is calling function to transmit this data on serial port.
4	Transmit routine is receiving data	It should transmit data on USART.	It is transmitting data on USART.

Chapter Seven

Cost Estimation

7. COST ESTIMATION

7.1 INTRODUCTION

Software cost estimation is the process of predicting the amount of effort required to build a software system. Cost estimates are needed throughout the software lifecycle. Preliminary estimates are required to determine the feasibility of a project. Detailed estimates are needed to assist with project planning. The actual effort for individual tasks is compared with estimated and planned values, enabling project managers to reallocate resources when necessary.

7.2 SIZE

Size is a primary cost factor in most models for cost estimation. There are two common ways to measure software size: ***lines of code*** and ***function points***.

7.2.1 Function Points

The function point measure is used for calculating cost estimation for this project.

- **Function points (FP)** measure size in terms of the amount of functionality in a system. Function points are computed by first calculating an unadjusted function point count (UFC). Counts are made for the following categories
- **Number of user inputs** – those items provided by the user that describe distinct application-oriented data (such as file names and menu selections)
- **Number of user outputs** – those items provided to the user that generate distinct application-oriented data (such as reports and messages, rather than the individual components of these)
- **Number of user inquiries** – an inquiry is defined as an online input that results in generation of some immediate software response in the form of online output. Each distinct enquiry is counted.
- **Number of files** – Each logical master file (i.e. a logical grouping of that may be one part of a large database or a separate file), is counted
- **Numbers of External Interfaces** – All machine-readable interfaces that are used to transmit information to another system are counted.
- Once this data has been collected, a complexity rating is associated with each count according to Table 7.1.

Table 7.1 Function point complexity weights.

	Weighting Factor (WF)		
Item	**Simple**	**Average**	**Complex**
Number of user inputs	3	4	6
Number of user outputs	4	5	7
Number of user inquiries	3	4	6
Number of files	7	10	15
Number of External Interfaces	5	7	10

Each count is multiplied by its corresponding complexity weight and the results are summed to provide the UFC.

To compute function points (FP), the following relationship is used: $FP = UFC * [0.65 + 0.01 * \sum F_i]$

The F_i (1 to 14) are "complexity adjustment values" called Technical Complexity Factor (TCF), based on responses to questions. Following table 7.2 shows the these factors

Table 7.2 Components of the technical complexity factor

F1	Reliable back-up and recovery	F8	Online update
F2	Data communications	F9	Complex interface
F3	Distributed functions	F10	Complex processing
F4	Performance	F11	Reusability
F5	Heavily used configuration	F12	Installation ease
F6	Online data entry	F13	Multiple sites
F7	Operational ease	F14	Facilitate change

Each component is rated from 0 to 5, where

0- No influence

1- Incidental

2- Moderate

3- Average

4-Siginificant

5-Essential

The TCF can then be calculated as:

$$TCF = [0.65 + 0.01 * \sum F_i]$$

Finally Function point can be calculated as

$$FP=UFC * TCF$$

Once function points have been calculated, they are used in a manner analogous to LOC to normalize measures of software productivity, quality and other attribute: $ per FP, FP per person-month, and errors per FP etc.

7.3 COST ESTIMATION FOR APPLICATION

Table 7.3 Computing Function Point Metrics

Item	Count		**Average WF**		Total
Number of user inputs	6	*	4	=	024
Number of user outputs	3	*	5	=	015
Number of user inquiries	40	*	4	=	160
Number of files	1	*	10	=	010
Number of External Interfaces	3	*	7	=	021
Unadjusted Function Point Count (UFC)					230

Table 7.4 Computing Technical Complexity Factor

Factor	Description	Rated value
F1	Reliable back-up and recovery	0
F2	Data communications	5
F3	Distributed functions	0
F4	Performance	4
F5	Heavily used configuration	2

F6	Online data entry	4
<u>Factor</u>	<u>Description</u>	<u>Rated value</u>
F7	Operational ease	4
F8	Online update	4
F9	Complex interface	1
F10	Complex processing	2
F11	Reusability	4
F12	Installation ease	3
F13	Multiple sites	0
F14	Facilitate change	3
$\sum F_i$		36

TCF = [0.65 + 0.01 * $\sum F_i$]

TCF = [0.65 + 0.01 * 36]

TCF = 1.01

Therefore,

FP = UFC * TCF

FP =230*1.01

FP = 232.3

So,

Function point Calculated is 232.31

Chapter Eight

Future Work

8. FUTURE WORK

1. Providing facility for *m* users to monitor *n* analyzers.
2. Providing to maintain logs of system for users.
3. Upgrading to Rs-485 Modbus Interface.
4. Registering the application in internet domain such that it can be accessed through the public IP.
5. Providing DHCP support for the application.
6. Adding more enhancements to user interface.
7. We can develop the generalized system which will work for all types of serial devices as shown in the figure below.

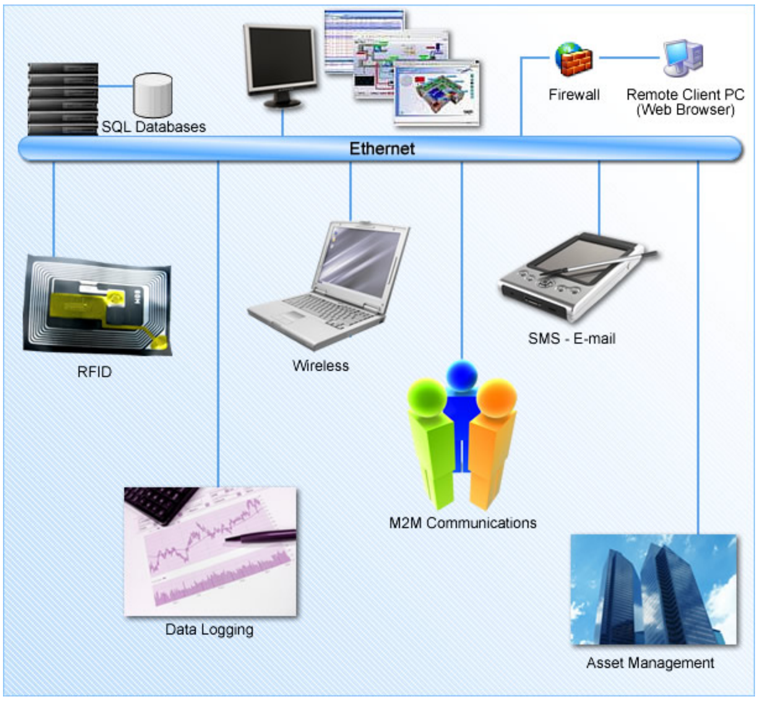

FIG 8.1 GENERALIZED SYSTEM

Chapter Nine

Summary

9. SUMMARY

The user tries to access the remotely placed serial device via internet. He views the GUI and accordingly clicks the buttons on the interface to manipulate the serial device. The buttons correspond to certain numbers (data) that are converted to serial data by RCM 4000 Module. This module then further converts the data i.e. serial data; to Ethernet data when serial device transmits any data towards the user.

Incase of any alarm the serial device sends the alarm alert to RCM 4000 module which in turn triggers 5 static email ids. The User Interface is available at a static IP address and requires the users to authenticate through before accessing the GUI.

Chapter Ten

Conclusion

10. CONCLUSION

Using the knowledge and Code as described in previous chapters and the successful working of the following modules:-

1. TCP/IP to Serial Conversion.
2. Serial Conversion to TCP/IP.
3. Email Triggering
4. User Authentication

Along with a profound satisfaction of our sponsor "*HONEYWELL AUTOMATION INDIA LIMITED*", we conclude that the project "*Remote Monitoring over Internet* " has been implemented and completed successfully and a platform for future enhancements has been laid herein.

Chapter Eleven

References

11. REFERENCES

WEBSITES

[1.] (DCPUM_Dynamic C.pdf) [www.zworld.com]

[2.] Rabbit Kit Details [www.zworld\RCM4000\rcm4000.htm]

BOOKS AND AUTHORS

[1.] [**Pressman Roger S.]** Software Engineering, a Practitioners Approach, Sixth Edition, 2005.

[2.] **[Hans-Erik Eriksson, Magnus Penker]** UML 2.0 Toolkit Reprint 2006.

[3.] **[Andrew S. Tanenbaum]**Computer Networks, Third Edition

[4.] **[Dynamic C User Manual]** Functional Guide and Specification booklet.

[5.] **[Rabbit 4000]** User Guide and Functional Reference.