

IRPA Grant Final Report

**DESIGN & DEVELOPMENT OF STREET LIGHT
MONITORING AND MANAGEMENT SYSTEM**

**MUSA BIN MOHD MOKJI
PROF IR DR SHEIKH HUSSAIN SHAIKH SALLEH
KAMARULAFIZAM BIN ISMAIL**

UNIVERSITI TEKNOLOGI MALAYSIA

TABLE OF CONTENT

CHAPTER	TITLE	PAGE
	ABSTRACT	ii
	TABLE OF CONTENT	iii
	LIST OF TABLES	vii
	LIST OF FIGURES	viii
	LIST OF APPENDICES	xi
CHAPTER I	INTRODUCTION	1
CHAPTER II	SYSTEM HARDWARE DESIGN	4
	2.1 Streetlight Monitoring Unit	4
	2.1.1 Introduction	4
	2.1.2 Feeder Interface Unit	8
	2.1.3 Feeder Controller Unit.	9
	2.1.4 Firmware Design	11
	2.1.4.1 Software Design of FIU	11
	2.1.4.2 Software Design of FCU	14
	2.1.4.3 Circuit Diagram of FIU and FCU	16
	2.2 Power Analysis Unit	24
	2.2.1 Introduction	24
	2.2.2 Analyzing measurement data	26
	2.2.3 Transients	27

CHAPTER	TITLE	PAGE
	2.2.4 Harmonics	27
	2.2.5 The Algorithm	28
CHAPTER III	SYSTEM SOFTWARE DESIGN	30
	3.1 Introduction	30
	3.2 Regional Control Centre-RCC	32
	3.2.1 Station Enrollment	34
	3.2.2 Setting	35
	3.2.3 System Log	42
	3.2.4 Internet/Networking	43
	3.2.5 GSM Control Panel	44
	3.3 National Control Centre-NCC	46
CHAPTER IV	COMMUNICATION INTERFACE	49
	4.1 GSM Network	49
	4.1.1 Why use SMS?	51
	4.2 PLCC	52
	4.2.1 Types of PLC technology	53
	4.2.1.1 Indoors/Short Range	53
	4.2.1.2 Outdoors/Long Haul	53
	4.2.1.3 Automotive	54
	4.2.2 Broadband over power lines	55
	4.2.3 IEEE	56
	4.3 TCPIP	57
	4.3.1 The Network Interface Layer	58
	4.3.1.1 PPP	59
	4.3.2 The Internet Layer	61

CHAPTER	TITLE	PAGE
	4.3.2.1 IP Addresses	64
	4.3.2.2 Conserving IP Addresses: CIDR, DHCP, NAT, and PAT	69
4.3.3	The Domain Name System	73
4.3.3.1	ARP and Address Resolution	75
4.3.3.2	IP Routing: OSPF, RIP, and BGP	76
4.3.3.3	IP version 6	78
4.3.4	The Transport Layer Protocols	79
4.3.4.1	Ports	79
4.3.4.2	TCP	81
4.3.4.3	UDP	83
4.3.4.4	ICMP	84
4.3.5	TCP Logical Connections and ICMP	86
4.3.6	The TCP/IP Application Layer	90
4.3.6.1	TCP and UDP Applications	90
4.4	Summary	93
CHAPTER V	FILED TEST	95
5.1	Introduction	95
5.2	Communication	96
5.3	GSM Modem	97
5.4	Client/Server Connectivity	98
5.5	Server System/NCC	100
CHAPTER VI	CONCLUSION	103
	REFERENCES	105

CHAPTER	TITLE	PAGE
	APPENDICES	106
	Appendix A: FCU Firmware Source Code	106
	Appendix B: FIU Firmware Source Code	114
	Appendix C: Client Software Source Code	128

ABSTRACT

The management of streetlights by the power utility company and local authorities are typically faced with the problem of high operational expenditure, low efficiency and increase customer complaint. They are also faced with increase customer complaint due to unattended faulty streets lights and frequent power outages. There is a significant pressure to reduce these operational expenses, improve efficiency and image. By operational efficiency we meant how faulty street lights are managed effectively through the use of a low cost automated system, thus improve efficiency and enhancing customer services. Operational cost reduction is achieved through accurately identification of faulty lights and timely action taken to rectify such fault. Currently these maintenance routines [i.e. random patrols around the street light zones] are conducted daily in parallel to records of faulty lights reported by customers. These incurred substantially high operational expenditure year to year. The objective of this project is to develop a low cost SLM system with features suffice enough for the utility companies to effectively manage and maintain street lights and also monitor power quality to ensure continuous and uninterrupted supply to customers both residential and industries. SLMS consist of interface modules (FC-Feeder Controller, FIU-Feeder Interface Units) installed at the substation or streetlights panel to collect the status of power over each feeder pillar. Information of faulty lights collected by the FIU (which will measure the current/power on the feeder and record any changes e.g. a drop in power indicating a faulty light in that feeder line) is passed back to FC using the PLC (Power line carrier) technique. The FC manages several feeder pillars and relay back the information received from FIU to a management system server located at the office (RCC-Regional Control Center & NCC-National Control Center) using "SMS" GSM network. RCC then sends an alert SMS to operational personnel to inform them of the faulty record. The management system keeps records for management reporting and analysis.

LIST OF APPENDICES

APPENDIX	TITLE	PAGE
A	FCU Firmware Source Code	106
B	FIU Firmware Source Code	114
C	Client Software Source Code	128

LIST OF FIGURES

FIGURE'S NO	TITLE	PAGE
2.1	Block Diagram of SLMS (<i>street lighting monitoring system</i>)	5
2.2	Drawing structures of SLMS (<i>street lighting monitoring system</i>)	6
2.3	Connection and location of FIU and FCU in SLMS (<i>Street Light Monitoring System</i>)	7
2.4	Feeder Interface Unit (FIU)	8
2.5	Block diagram of FIU (3 phase)	8
2.6	Inside view of FIU. FIU can add up to 4 cards (include main board) to monitor until 4 outputs feeder	9
2.7	Feeder Control Unit (FCU)	9
2.8	Block diagram of FCU	10
2.9	Inside view of FCU	11
2.10	Flow chart micro-controller program of FIU	12
2.11	Binary data format sending from FIU to FCU	14
2.12	Flow chart micro-controller program of FCU	15
2.13	Block diagram of FIU (3 phase)	16
2.14	50 Hz Amplifier circuit of FIU	16
2.15	RMS to DC converter circuit of FIU	17
2.16	DC adder circuit of FIU	17
2.17	Comparator circuit of FIU	18
2.18	Photo coupler circuit of FIU	18
2.19	Clock circuit of FIU	19
2.20	Micro-controller circuit of FIU	19
2.21	TTL to RS232 converter circuit of FIU	20

2.22	Total circuit for FIU main board	20
2.23	Simulated complete PCB of FIU	21
2.24	Schematic diagram of FCU	21
2.25	Photo coupler circuit of FCU	22
2.26	Micro-controller circuit of FCU	22
2.27	RS232-TTL converter circuit of FCU	23
2.28	Total circuit for FCU main board	23
2.29	Simulated complete PCB of FCU	24
2.30	This flow chart depicts an example of a data analysis system	26
2.31	Long-term monitoring of harmonic rms currents may be presented in a histogram	27
3.1	Authentication of RCC	33
3.2	Main user interface	34
3.3	New station enrollment	35
3.4	System Setting-General	36
3.5	System Setting-Communication	37
3.6	System setting- Security/Authentication	38
3.7	System Setting-Logging	39
3.8	System Setting-Fault Code	40
3.9	System setting-Database Management	41
3.10	System Setting-Sound Alert	42
3.11	System logging information	43
3.12	Internet/Networking control panel	44
3.13	GSM modem control panel	45
3.14	Login/ authentication dialog	46
3.15	Main Interface dialog	47
3.16	Internet/networking server control panel	48
4.1	GSM Communication	50
4.2	Abbreviated TCP/IP protocol stack	58
4.3	PPP frame format (using HDLC)	59
4.4	IP packet (datagram) header format	62

4.5	IP Address Format	64
4.6	Network Address Translation (NAT)	71
4.7	Port Address Translation (PAT)	72
4.8	TCP segment format	81
4.9	UDP datagram format	84
4.10	TCP logical connection phases	88
4.11	TCP/IP protocol suite architecture	93
5.1	Pilot plan implementation	95
5.2	FIU layout design	96
5.3	GSM modem control panel	98
5.4	Client configuration utility	100
5.5	Server socket starting interface	101

LIST OF TABLES

TABLE'S NO	TITLE	PAGE
3.1	Comparison of SQL Implementation	32
4.1	Subnet number of bits	65
4.2	IP address space	70
4.3	Well-known port number	80

CHAPTER 1

INTRODUCTION

Operation and maintenance of infrastructure and facilities of power distribution (at the feeder level) has always been a major concern of power utility companies and local authority (who manage the power distribution within their jurisdiction). Their main concern is usually the inefficient utilization of resource in order to give good customer service level. Managing the street lighting is an area of interest of TNB due to the vast distribution of lighting network implemented in cities and town. The size of the distribution has grown tremendously and as such there is a need to have a monitoring and management system to help reducing the operating cost while improving and optimize the utilization of workforce to handle maintenance activities.

This project will develop a mechanism to facilitate the monitoring, identifying/locating and rectifying the fault lights within the feeder network. A feeder control unit will measure the current/power on the feeder and record any changes e.g: drop in current indicating a faulty lamp in that particular feeder line.

Enhance feature of this product will include a smart module placed at individual lighting pole at the incoming feeder to collect data and distribute the status of these lights. Data collected at the feeder points will be remotely monitored by a central control system located in the central/regional offices.

In this period of research, we tend to prove on some concept and idea. This concept will bring together many parts from different technology and put them in a system to realize the idea.

At the end of this research, we manage to integrate this different kind of technology and make them talk to each other while passing the appropriate information. As the result, we have a system which monitor the status of the streetlight and power quality in our transmission and distribution system

Classification of power quality (PQ) related voltage and current waveform distribution is a key task power system monitoring. TNB engineers will use PQ to access the quality of power received by customers in an electrical power system. The study of power quality has been a major effect at any electrical utilities and industries.

Operation and maintenance of infrastructure and facilities of power distribution (at the feeder level) has always been a major concern of power utility companies and local authority (who manage the power distribution within their jurisdiction). Their main concern is usually the inefficient utilization of resource in order to give good customer service level. Managing the street lighting is an area of interest of TNB due to the vast distribution of lighting network implemented in cities and town. The size of the distribution has grown tremendously and as such there is a need to have a monitoring and management system to help reducing the operating cost while improving and optimize the utilization of workforce to handle maintenance activities.

This project will develop a mechanism to facilitate the monitoring, identifying/locating and rectifying the fault lights within the feeder network. A feeder control unit will measure the current/power on the feeder and record any changes e.g: drop in current indicating a faulty lamp in that particular feeder line.

Enhance feature of this product will include a smart module placed at individual lighting pole at the incoming feeder to collect data and distribute the status of these lights. Data collected at the feeder points will be remotely monitored by a central control system located in the central/regional offices.

The main feature of this system will prevent unauthorized access and usage of the streetlight monitoring and management system by both power utility company personnel and its approved sub-contractor. The proposed system can perform the following tasks:

- Monitor on/off of the streetlights
- Monitor status of feeder line
- Monitor abnormal lighting timing
- Data transfer based on compression technique to main station
- Analysis of feeder line using signal analysis technique for preventive maintenance

This project will consist of several units within the research environment, developing various component of the system. These units are.

- i. Controller access unit.
- ii. Back office and database unit.
- iii. DSP security interface
- iv. Communication interface unit
- v. Each unit is responsible to design, document and code their respective component.

CHAPTER 2

SYSTEM HARDWARE DESIGN

2.1 Streetlight Monitoring Unit

2.1.1 Introduction

There are various methods to monitor streetlights condition. The most common and easiest way is by visual checking during the streetlight when it is in operation. This procedure requires a number of resources including time, manpower and cost and no longer become an effective method of monitoring and maintenance task. The rapid development of technology has changed the style of monitoring streetlight from the conventional to the remote and distributed method. For instance, today we can monitor the streetlight from a centralized monitoring center by using a PC.

In Malaysia, Iconergy Sdn. Bhd with cooperation with University Technology Malaysia has developed a system which can manage and monitor the streetlight condition using a very advance method. The system is known as “*Power Quality Management and Street Light Monitoring System*” (PQSLM).

“*Power Quality Management and Street Light Monitoring System*” (PQSLM) is a system which monitor street light condition without having to go to the remote site. PQSLM uses 3 medium of data transfer from street light panel box to the central monitoring station. At the street light panel box, PQSLM will measure the current level

and process the data a dedicated microcontroller. The microcontroller will send data to the substation via power line communication protocol.

The above mentioned process is conducted by a hardware known as FIU (*feeder interface unit*). FIU is located inside the street lighting panel box. FIU consists of a several major part such as current sensor, microcontroller circuit and PLC module. The current sensor is a power clamp which also called CT coil (*current transformer coil*). This power clamp will transfer the actual current value to the small AC voltage with specific ratio depend on the type of power clamp. Then, the microcontroller will do some comparison with threshold voltage to determine the status of the streetlights. After the process, the microcontroller will also send the data to PLC module.

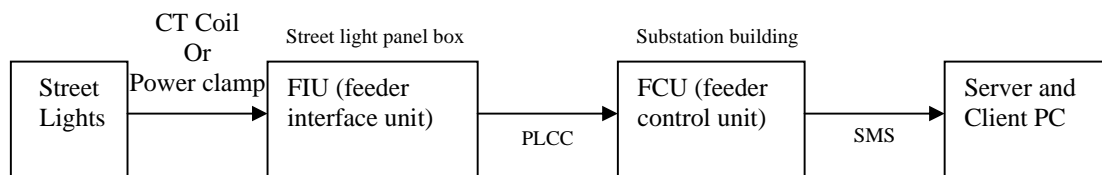


Figure 2.1: Block Diagram of SLMS (*street lighting monitoring system*).

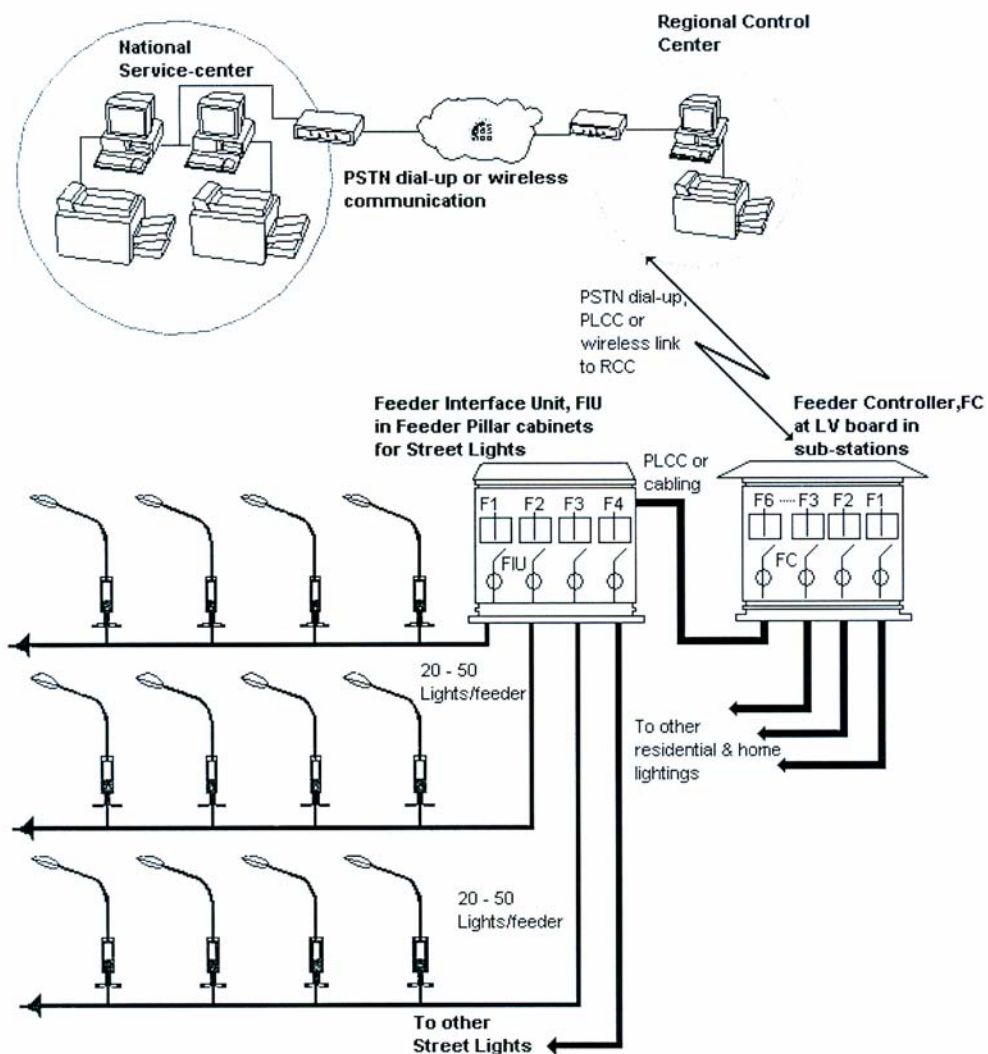


Figure 2.2: Drawing structures of SLMS (*street lighting monitoring system*).

According to the PLCC module specification, the data can be transferred in a distance of more than 500 meter.

FCU (*feeder controller unit*) is located inside the substation building. FCU received the data from FIU and then communicates with GSM modem to send SMS to the central station. FCU receives two dedicated input one from street light and another one from power quality analyzer unit. This power quality (PQ) analyzer unit utilizes a

high-speed digital signal processing (DSP) technology where the hardware is special design for complex and accurate mathematical calculation.

PQ is used for signal analysis to detect power the disturbance in 240Vac voltage signal. This FCU is also equipped with a backup power supply for continuous operation. The backup power supply is programmed to send a signal to FCU in case of power failure, and FCU will send SMS to the central station for immediately power shutdown information. For the security purpose, this system can easily detect the signal from the door sensors or motion sensors and then send the SMS to the central station. The FCU will continuously send the SMS if the sensors are still alerted. PQSLM is not just a monitoring system. It is also embedded with power disturbance and security solution.

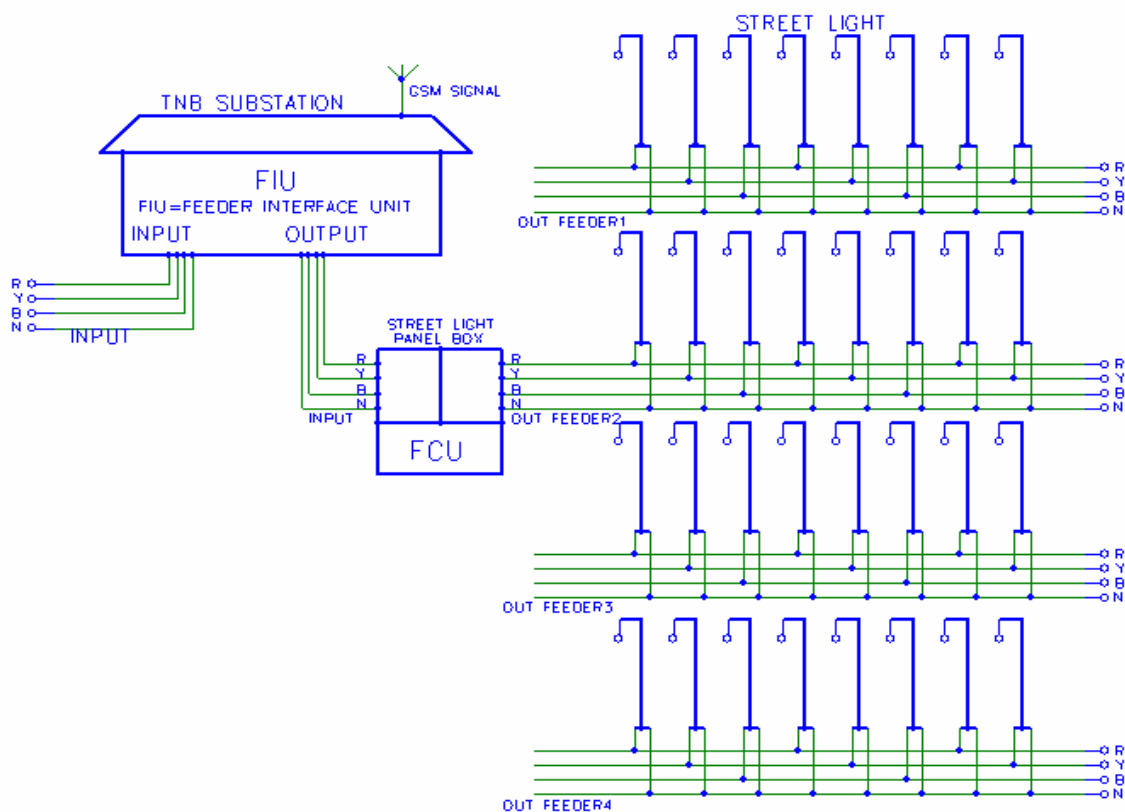


Figure 2.3: Connection and location of FIU and FCU in SLMS (*Street Light Monitoring System*).

Streetlight monitoring system usually implements one of these techniques which is either measuring the status of each and single streetlight or measuring the total current that flows to a certain feeder. The designed system measure the current level of a feeder which represents the status of streetlight for the whole feeder. This method can be explained by OHM law theory where the current (I) is equal to the voltage (V) divide by resistance (R). For instance, when we have 100 lights from one output feeder, each light consume 1 ampere current during normal operational time and by omitting the cable resistance, the total current will be 100-Ampere. So, if the current below the 100-A, we can detect a faulty of the light even if only one light goes OFF.

2.1.2 Feeder Interface Unit



Figure 2.4: Feeder Interface Unit (FIU).

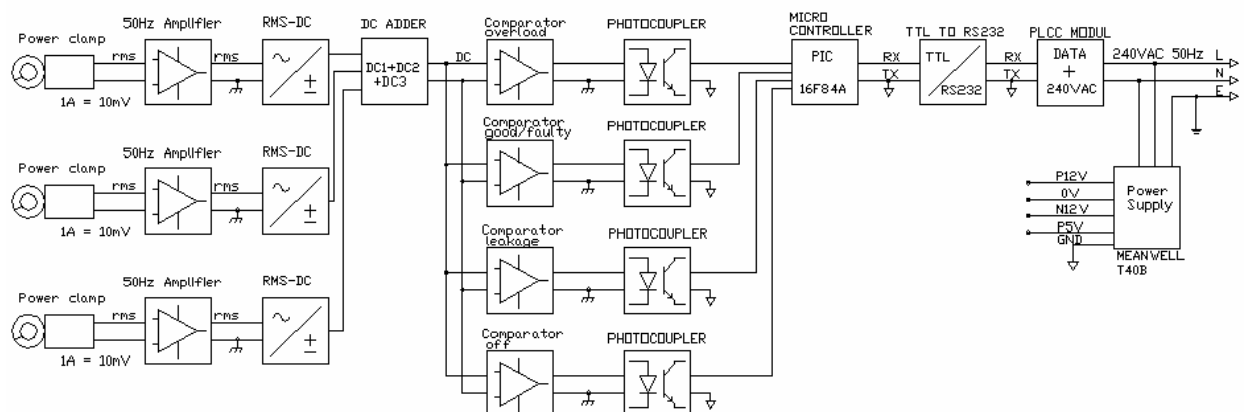


Figure 2.5: Block diagram of FIU (3 phase).

Figure 2.5 is a block diagram of an FIU. The FIU receives the input from a power clamp. Power clamp is used to measure the current that flow inside the supply cable for a streetlight feeder. The power clamp converts the current value from electromagnetic field to the small ac voltage (r.m.s) in according to the manufacturer defined ratio. FIU is using the power clamp with 1A/ 10mV rating.



Figure 2.6: Inside view of FIU. FIU can add up to 4 cards (include main board) to monitor until 4 outputs feeder.

2.1.3 Feeder Controller Unit.



Figure 2.7: Feeder Control Unit (FCU)

Basically, FCU are the main microcontroller, which control the data flow between FIU, FCU and central station. FIU incorporates a PLCC module as the data transceiver between FIU and FCU and GSM modem as the transceiver between FCU and central station. The FCU has one unique feature. It is pinging capability. It means that the user can ping the FCU to request the current status of the monitored substation. This is done by doing a missed call to the GSM modem terminal and the FCU will send the status of all connected substation.

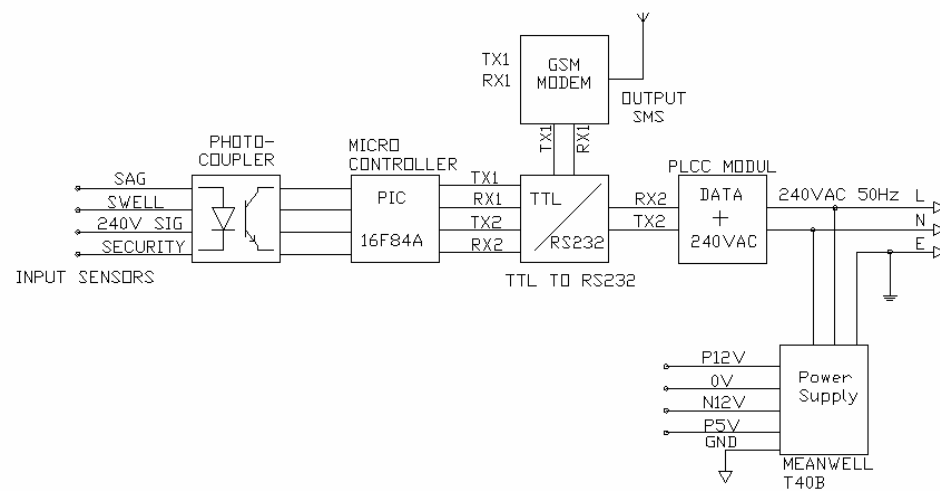


Figure 2.8: Block diagram of FCU.

The FCU is also designed to monitor input for power quality analyzer, security and 240Vac signal detector. FCU used switching power supply with 240Vac input and triple output +5, +12 and -12V DC. This power supply is connected from a UPS, so that the FCU can maintain the operation in case of power failure and send notification message to central station.



Figure 2.9: Inside view of FCU.

2.1.4 Firmware Design

2.1.4.1 Software Design of FIU

FIU uses a small microcontroller to control the whole circuit operation. The microcontroller is PIC16F84A from MICROCHIP. In this chapter will describe in details about the programming of the microcontroller by using assembly language.

We needs a hex file to be download to the flash ROM (*read only memory*) of microcontroller. After we write or burn the flash ROM, now it can works as programmed. Before we start making the assembly file, the first step is constructing the flow chart. Flow chart is important because it shows the structure of our program briefly. Then, from that we can easily to understand the flow of subroutine inside the programming.

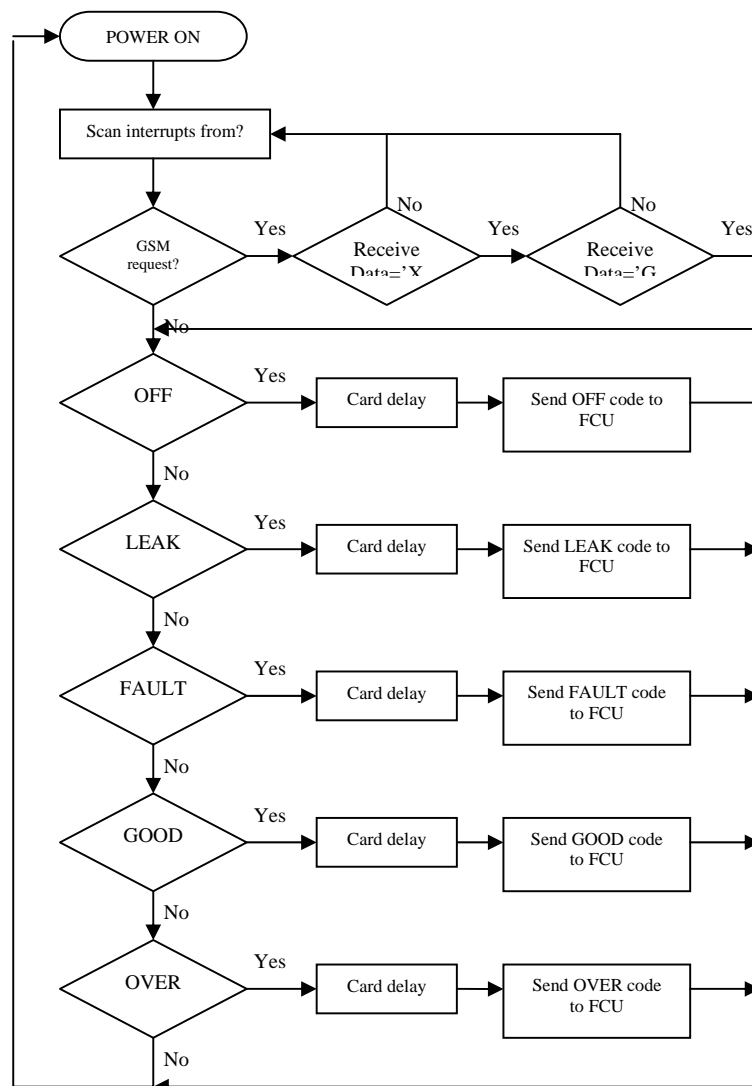


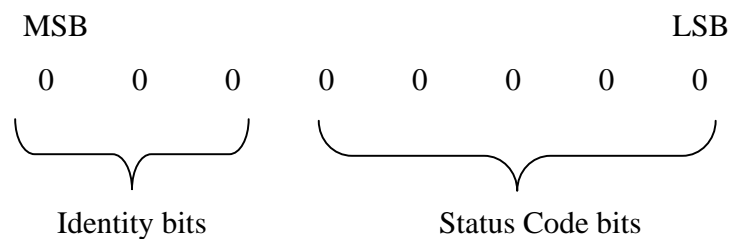
Figure 2.10: Flow chart microcontroller program of FIU.

On the initialization of the circuit, the FIU starts the scanning task especially on the interrupt signal from the all inputs. So far, it was designed have 5 inputs, which is coming from GSM MODEM, OFF, LEAK, FAULT, GOOD and OVER sensor. Data format from GSM MODEM is in serial data and another sensor is digital logic pulse.

The same process happen at the FIU when it receives any signal from the others sensors. FIU monitor the streetlight and directly send the code data depend on the streetlight status to FCU. This process continues until we switch OFF the power supply.

So it will be functioning continuously and have good capability to monitor in real time situation every day.

The figure below is a complete binary data format sending from FIU to FCU. This data call “*streetlight status data*” (SLSD). This type of data are 8 bit, none parity, stop bit 1 with speed 1200bps. The baud rate is 1200bps because PLCC module can send data maximum speed on that speed. Five bit data from the LSB is a *Status Code bits* data. Then, 3 bits data from the MSB is an *Identity bits*. Identity bits are used to separate the FIU data if more then one FIU was connected in the same phase in power line.



Below is the list down of data for one sample unit FIU. We can see the setting of Status Code and Identity bits. So, if we have additional FIU needs to put in the same phase power line, we just change the *Identity bits* to 010 or else than 001.

```

001 00000    ;main board off
001 00001    ;main board leak
001 00010    ;main board fault
001 00011    ;main board good
001 00100    ;main board over

001 00101    ;card1 off
001 00110    ;card1 leak
001 00111    ;card1 fault
001 01000    ;card1 good
001 01001    ;card1 over
  
```


001 01010	;card2 off
001 01011	;card2 leak
001 01100	;card2 fault
001 01101	;card2 good
001 01110	;card2 over
001 01111	;card3 off
001 10000	;card3 leak
001 10001	;card3 fault
001 10010	;card3 good
001 10011	;card3 over

Figure 2.11: Binary data format sending from FIU to FCU.

2.1.4.2 Software Design of FCU

FCU (*feeder controller unit*) also used same type of microcontroller as FIU (*feeder interface unit*). It also runs a similar firmware as which programmed in the flash ROM.

2.1.4.3 Circuit Diagram of FIU and FCU

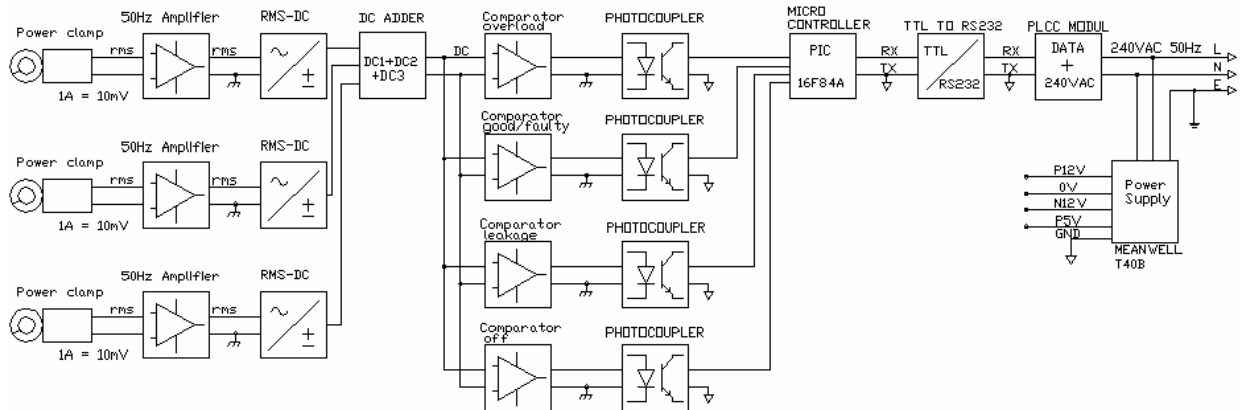


Figure 2.13: Block diagram of FIU (3 phase).

By refer to the block diagram above; it is easily to show the schematic diagram of FIU. In this chapter will explain the function of schematic diagram according to the block diagram. Subsequent figures show the circuit diagram of component for FIU and FCU.

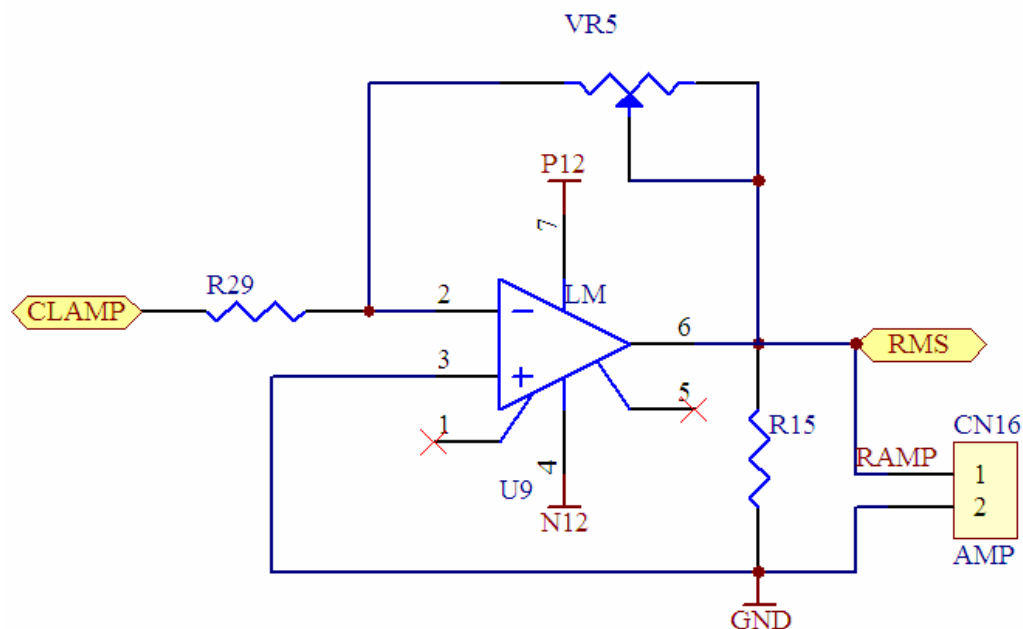


Figure 2.14: 50 Hz Amplifier circuit of FIU

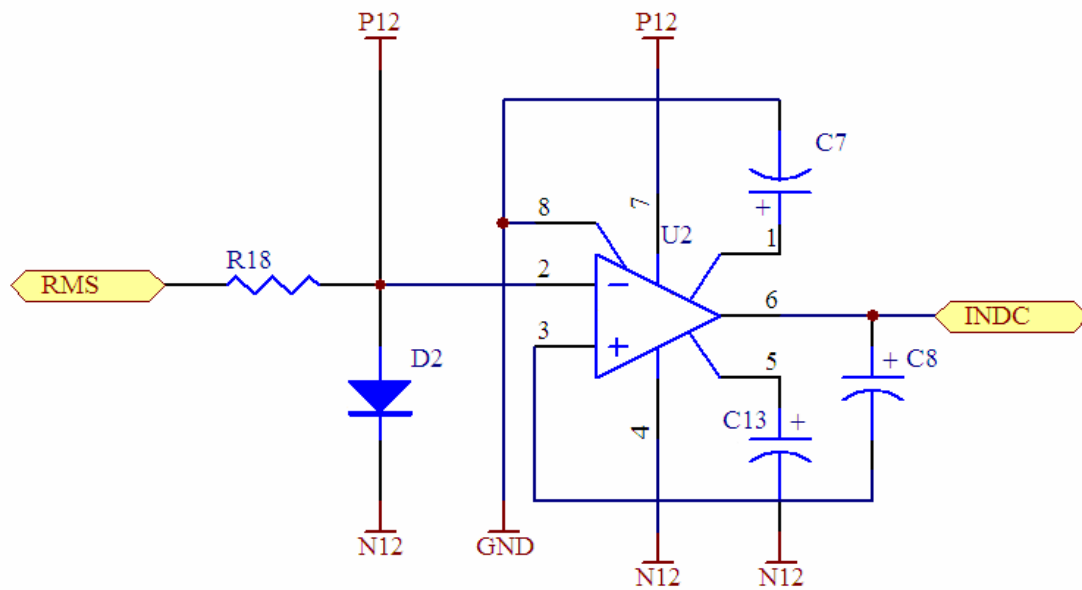


Figure 2.15: RMS to DC converter circuit of FIU

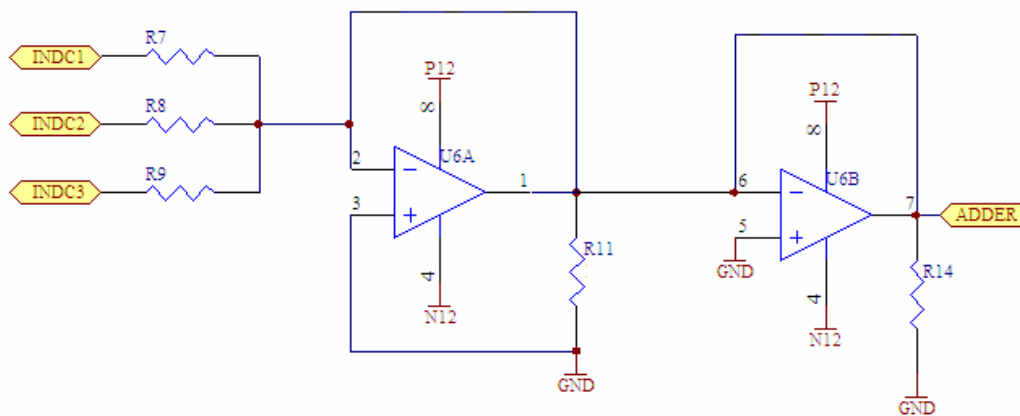


Figure 2.16: DC adder circuit of FIU

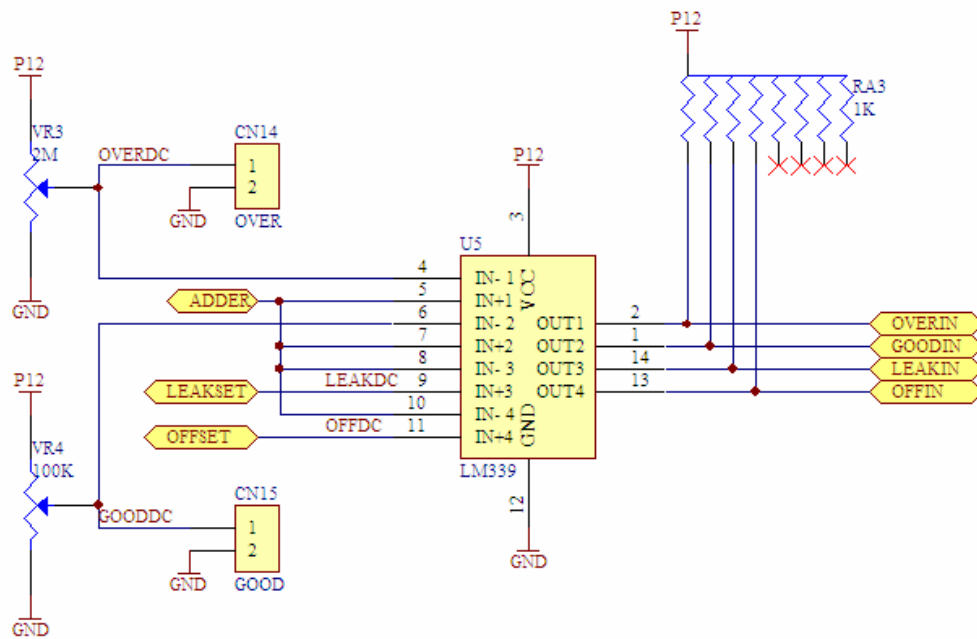


Figure 2.17: Comparator circuit of FIU

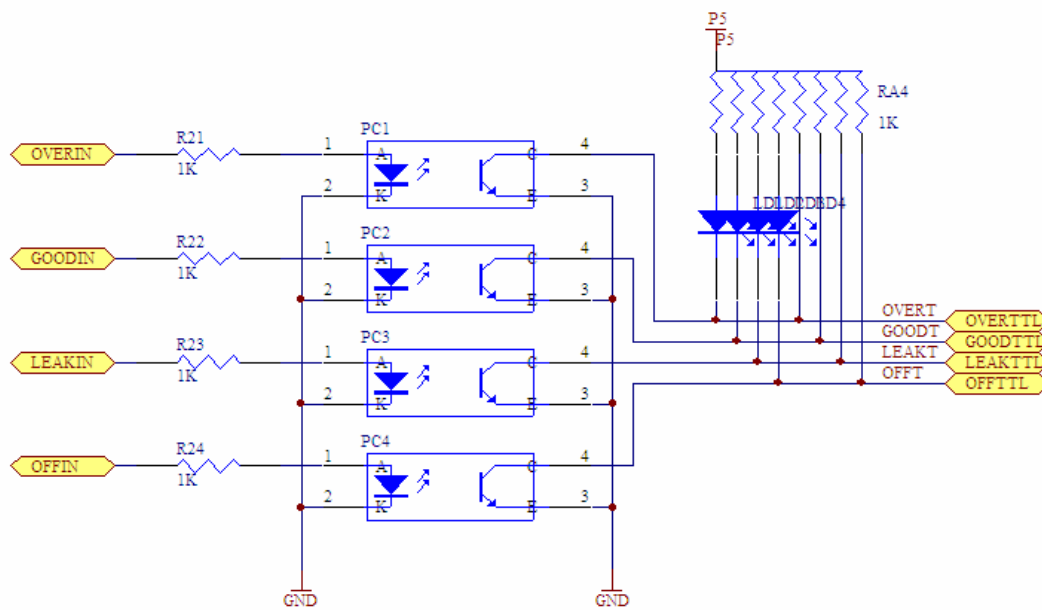


Figure 2.18: Photo coupler circuit of FIU

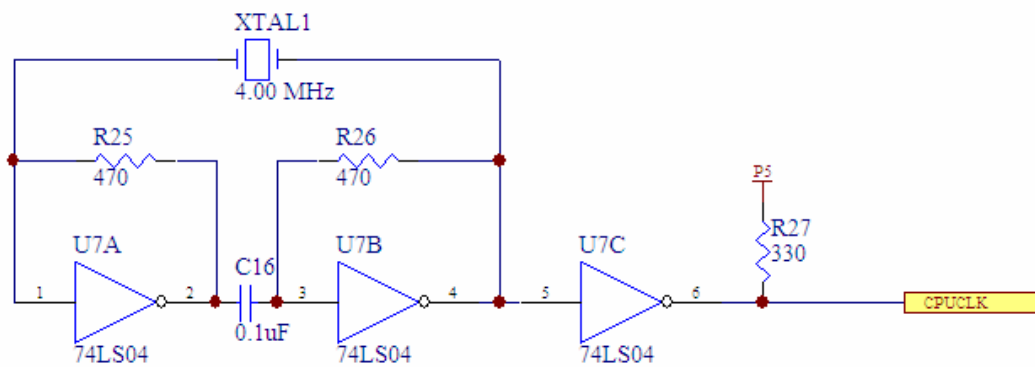


Figure 2.19: Clock circuit of FIU

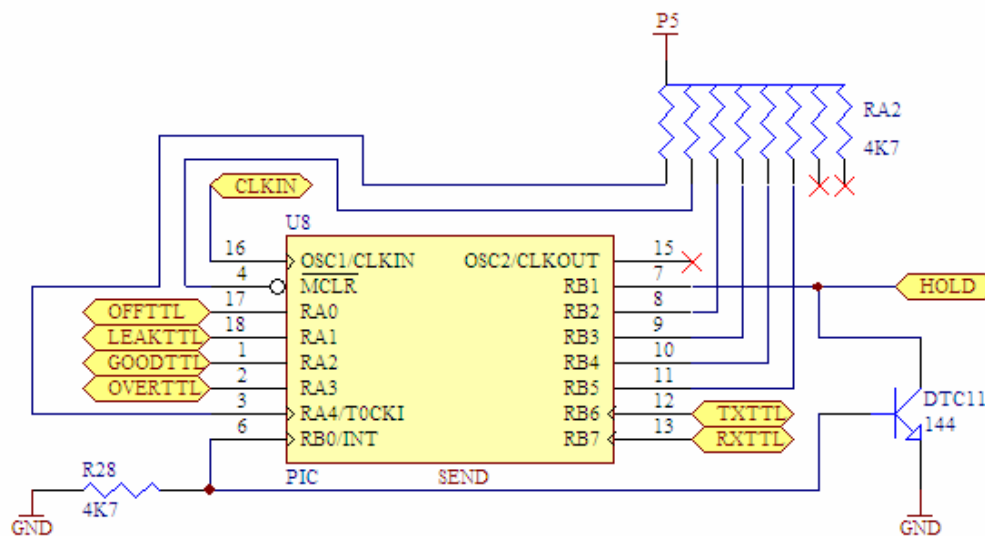


Figure 2.20: Microcontroller circuit of FIU

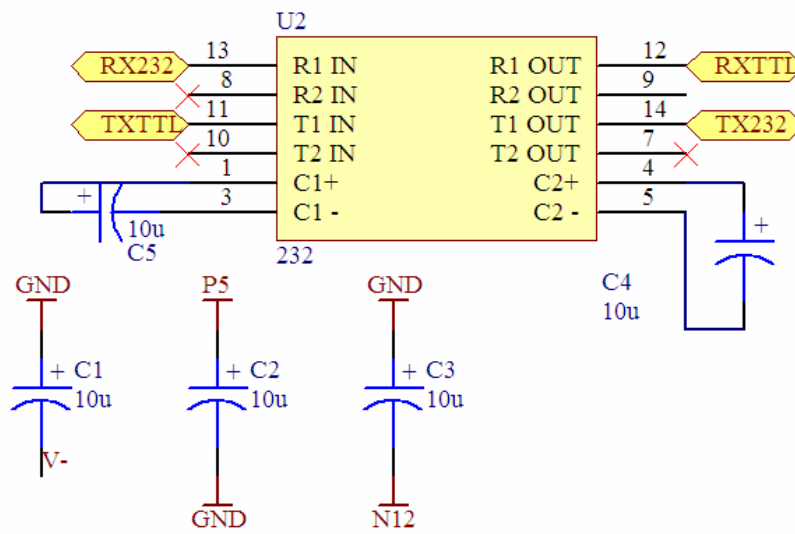


Figure 2.21: TTL to RS232 converter circuit of FIU

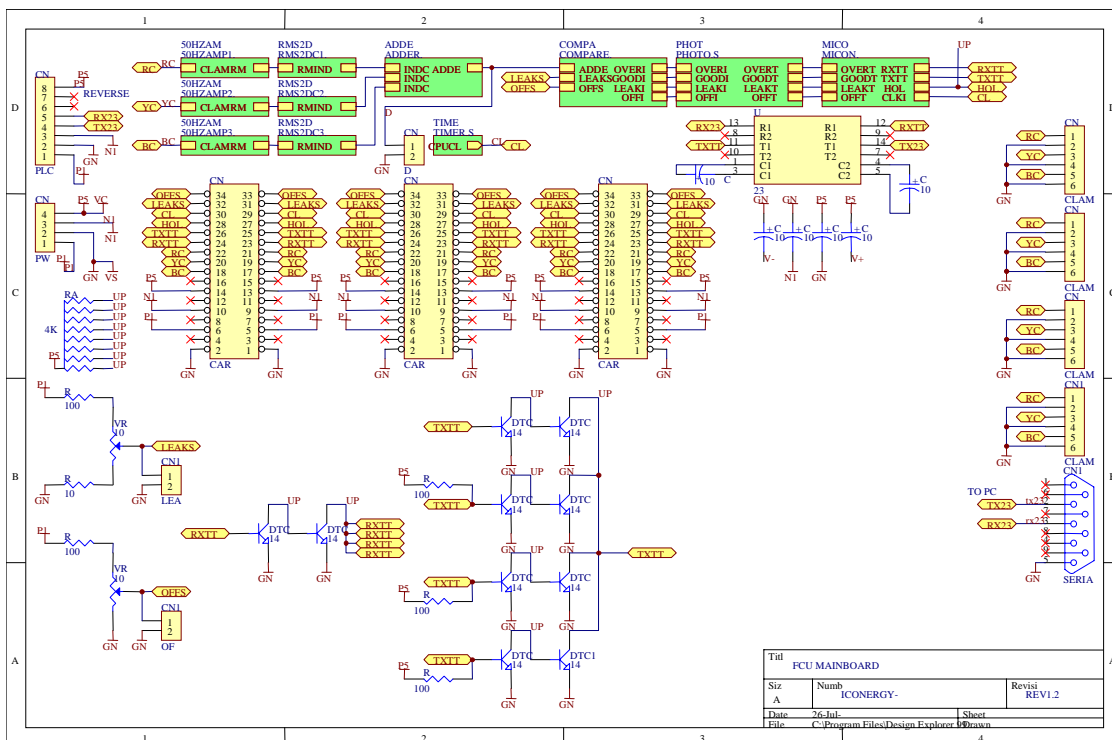


Figure 2.22: Total circuit for FIU main board

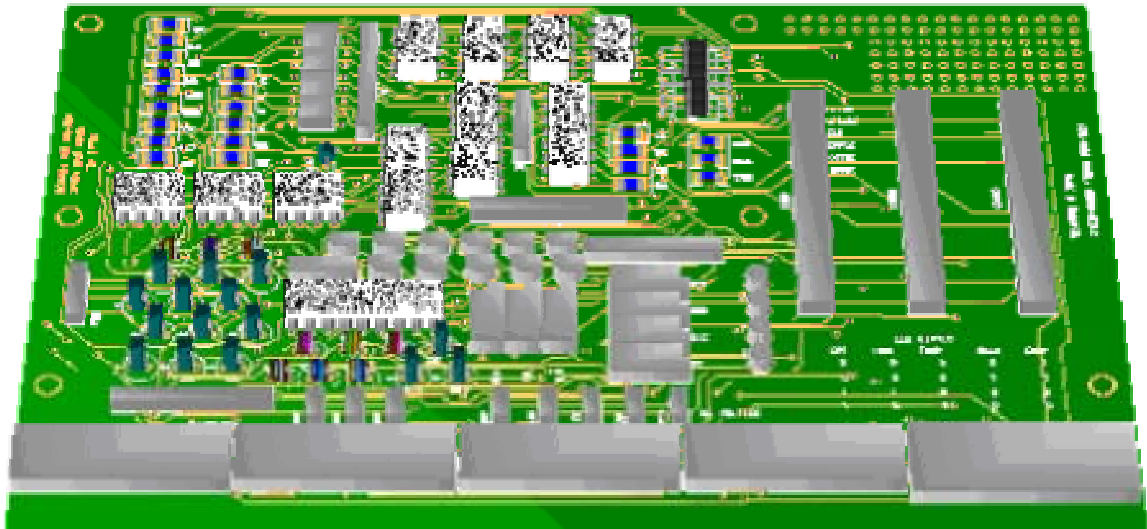


Figure 2.23: Simulated complete PCB of FIU

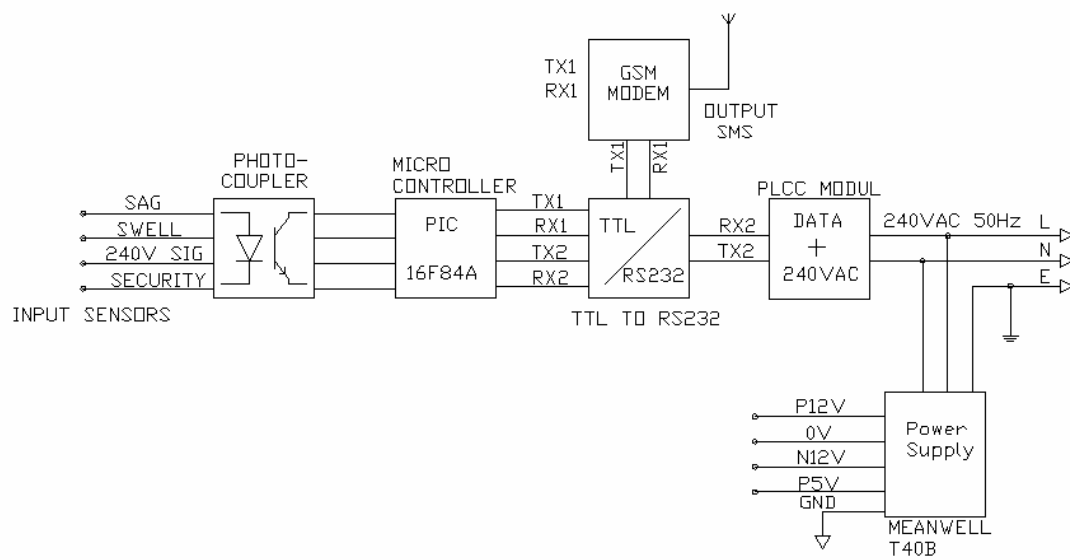


Figure 2.24: Schematic diagram of FCU

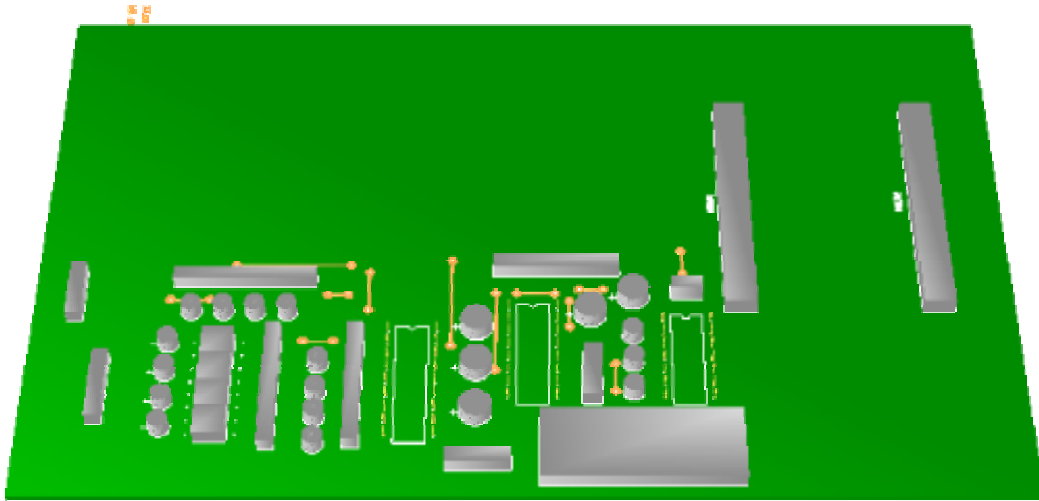


Figure 2.29: Simulated complete PCB of FCU

2.2 Power Analysis Unit.

2.2.1 Introduction.

Power quality, in recent years, has become an important issue and is receiving increasing attention by utility, facility, and consulting engineers. Present equipment setups and devices used in commercial and industrial facilities, such as digital computers, power electronic devices, and automated equipment, are sensitive to many types of power disturbances. Power disturbances arising within customer facilities have increased significantly due to the increasing use of energy efficient equipment such as switch-mode power supplies, inverters for variable speed drives, and more. The monitoring and data collection of power disturbances for power quality study therefore has to be conducted at the users' premises. To understand the power quality problem better requires a comprehensive monitoring and data capturing system that is used to characterize disturbances and power quality variations.

The threatened limitations of conventional electrical power sources have focused a great deal of attention on power, its application, monitoring and correction. Power economics now play a critical role in industry as never before. With the high cost of power generation, transmission, and distribution, it is of paramount concern to effectively monitor and control the use of energy. The electric utility's primary goal is to meet the power demand of its customers at all times and under all conditions. But as the electrical demand grows in size and complexity, modifications and additions to existing electric power networks have become increasingly expensive. The measuring and monitoring of electric power have become even more critical because of down time associated with equipment breakdown and material failures. For economic reasons, electric power is generated by utility companies at relatively high voltages (4160, 6900, 13,800 volts are typical). These high voltages are then reduced at the consumption site by step-down transformers to lower values which may be safely and more easily used in commercial, industrial and residential applications. Personnel and property safety are the most important factors in the operation of electrical system operation. Reliability is the first

consideration in providing safety. The reliability of any electrical system depends upon knowledge, preventive maintenance and subsequently the test equipment used to monitor that system.

Power quality has become a key concern for utility, facility, and consulting engineers because end-use equipment is now more sensitive to disturbances arising both from the utility power supply and within a customer's power distribution system. Also, this equipment is more interconnected in networks and industrial processes, meaning the effects of a problem with any one piece of equipment are much more severe.

This increased awareness of the importance of power quality has generated significant advances in monitoring equipment that can characterize disturbances and power quality variations. (“**Categories of Power Quality Variations**” below.) Today's monitoring tools can present information as individual events, like disturbance waveforms, trends, or statistical summaries. By comparing events with libraries of typical power quality variation characteristics and correlating them with system events, like capacitor switching, you can determine the causes of these variations.

In the same manner, you can correlate the measured data with verified effects on specific equipment to help characterize the sensitivity of that equipment to power quality variations. This will help you identify what equipment requires power conditioning as well as provide specifications for that protection

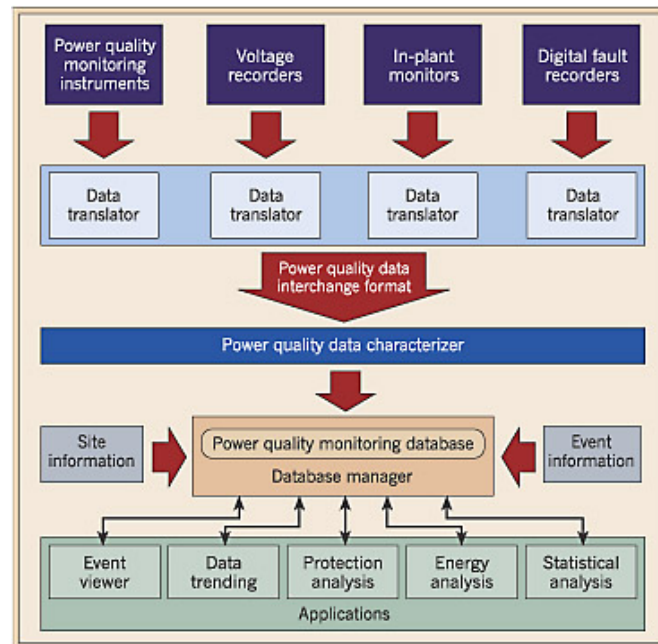


Figure 2.30: This flow chart depicts an example of a data analysis system.

2.2.2 Analyzing measurement data.

As we've seen, analyzing power quality measurements has become increasingly more sophisticated. It's not enough to simply look at rms quantities of voltage and current because some disturbances have durations in the millisecond range. In addition, today's end-use equipment is more sensitive to these very short disturbances. Finally, there is more equipment connected to our power systems that, in fact, cause disturbances or power quality problems.

Your data analysis system must be flexible enough to handle data from a variety of monitoring equipment as well as maintain a database you can use for many different applications (**Fig. 1** above).

Different types of power quality variations require different types of analysis to characterize system performance. And with a flexible system, you can customize these applications to individual user needs.

2.2.3 Transients.

These power quality variations are normally characterized by the actual waveform. However, you can develop summary descriptors for the following:

- Peak magnitude
- Primary frequency
- Time of occurrence
- Rate of rise

2.2.4 Harmonics.

Individual snapshots of voltage and current with the associated harmonic spectrums characterize these power quality variations. It's important to note that harmonic distortion levels are always changing, so these characteristics can't be represented with a single snapshot. As a result, you'll need to compile time trends and statistics. **Fig. 4** above shows an example time trend plot for one month, and **Fig. 5** below shows the statistics of harmonic current level for a much longer period of time.

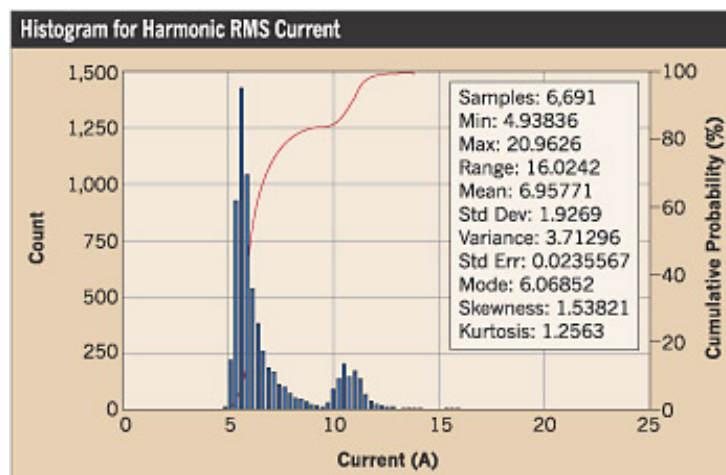


Figure 2.31: Long-term monitoring of harmonic rms currents may be presented in a histogram

A data analysis system for power quality measurements should be able to process data from a variety of instruments and support a range of applications for processing the generated data. With continuous power quality monitoring, it's very important to be able to summarize variations with time trends and statistics as well as characterize individual events.

2.2.5 The Algorithm

Consider the following power line carrier signal.

$$x(t) = A \sin(2 \pi ft). \quad (2.1)$$

To estimate its instantaneous energy and frequency, the equation (1) must be in complex or discrete-analytical form. So that the real and imaginary part can be separated and calculated independently.

$$z(n) = x(n) + jH[x(n)] \quad (2.2)$$

Where $H[]$ is the Hilbert transform of $x(n)$ which has approximately unity gain and introduce a $\pi/2$ phase shift with respect to the original signal. The Hilbert transform calculation method is

$$jX_i[n] = \frac{1}{N} \sum_{m=0}^{N-1} X_R[m] \otimes V_N[n-m] \quad (2.3)$$

$$V_N[n] = \begin{cases} -2j \cot(\pi n/N), & n=\text{odd.} \\ 0, & n=\text{even.} \end{cases} \quad (2.4)$$

Where $X_R[m]$ is the DFS[6] of $x(n)$ and the \otimes denote the circular convolution technique.

Another characteristic of analytical signal is the relationship between the signal amplitude and the instantaneous energy. The instantaneous energy is basically representing the temporal strength of a time varying signal.

$$E_z = z(n)z^*(n) = c(n)c^*(n). \quad (2.5)$$

The detection by Instantaneous Energy gives a unique feature that can represent every single disturbance associated with power quality disturbances such as voltage sag, transient voltage swell and waveform distortion.

CHAPTER III

SYSTEM SOFTWARE DESIGN

3.1 Introduction

This system comes together with a manager software. This software acts as an interface between the remote monitoring unit and a system operator who is responsible of handling the fault alert of a malfunctioned or damaged facility. This software consists of Regional Control Centre-RCC and National Control Centre-NCC. RCC is a software located at the local area covered by the substation that is being monitored. One RCC may consist of hundreds of substation depending on the geographical factor as well as the political establishments. For example, one RCC may control and monitor 500 substations located at a particular town or district. While another town or district will be covered by another RCC. While NCC is a software located at a central control station which serve as a server to all the RCC. RCC and NCC are connected through the internet.

At the NCC, one can observe the status and the condition of all the RCC as well as the status of each and every single substation. Here we deploy the concept of distributed and centralized system. The reason of having this kind of architecture is to help the local maintenance company to response to the fault alert as soon as possible once they received the alarm notification. But at the same time, the alert is also being sent to a centralized monitoring station. This method will minimize the task load of a particular manager software when the task loads are being shared among the other manager software which in this case the NCC and RCC. Using only one manager software is a way to reduce the cost but putting all the load to one software is not a wise decision. Just imagine what happen when all the substations are sending alert at the same time. This

situation will create a bottleneck where the software won't be able to serve every single alert properly due to the limitation on the bandwidth and processing speed. So, the proposed architecture will help to eliminate this problem where each RCC will act as a data aggregator and relayed the processed data to the NCC.

The software is designed using Microsoft Visual Studio C++ with the support of Microsoft Foundation Class library. This development environment is chosen due to its capability of handling various type of hardware and communication interface. In addition, the user interface is preferable compared to other development platform for instance visual basic, visual Interdev end etc.

The whole system uses MySQL database scheme to store information regarding to the substation profile, user account, log, fault, region information and others. It is chosen because MySQL is a free database system. Means that, no license required compared to other database scheme. MySQL is a lite version of Microsoft SQL Server database. So, it is simpler and faster. But the important issue such as security feature is improved.

SQL, or Structured Query Language is a specialize type of programming language developed to work with relational database such as MySQL, Oracle, Ms SQL Server, PostGre SQL, Informix and others.

SQL standard is defined by ANSI, American National Standard Institute in the ISO/IEC 9075:1992 document. Every relational database applies its own version of SQL standard; many enhance that standard. Standardizing the programming language allows the developer to address the database in much the same way from platform to platform

Table 3.1: Comparison of SQL Implementation


RDBMS	Advantages	Drawbacks
Oracle	Versatile, Stable, Secure	Potentially high total cost ownership (TCO)
Ms SQL Server	Stable and secure, Microsoft offer excellent support	Relatively high TCO,proprietary
Postgre SQL	Up-and-coming database with low TCO	Has yet to be widely implemented in large scale business use
Informix	Stable, good support available	Generally higher TCO
MySQL	Offer best case scenario database in many ways; low TCO, high stability, very high security and excellent support	Not all available version can offer the full range of MySQL capabilities

3.2 Regional Control Centre-RCC.

RCC holds the most important information as it is the closest interface to the remote monitoring unit namely FCU and FIU. It works in real-time environment and handles a lot of critical processes. The design has been associated with a lot of crashed-proof subroutine to prevent system crash upon system and technical failure. This is to ensure that the system is able to continue to perform the monitoring task in any condition without the need of user intervention. We tend to provide a fully automatic self manageable system which can assist the power utility company to maximize the productivity as well as providing the best service to the customer end.

As for security reason, RCC software comes with an authentication procedure to make sure only the authorized person is accessible to the data. During this procedure, the user is required to pass through two steps of authentication which is software and database authentication. Figure 3.1 shows the dialog that appears when the program is

executed. This software incorporates a multilevel security where different role of user may have different option of accessibility. For instance, an administrator may have greater access to the system than an operator. If the user enters the correct information, the system will proceed to the main interface; otherwise the system will be terminated



The screenshot shows a dialog box titled "Substation Monitor Client". It contains a license notice: "This software is licensed to Iconergy Sdn Bhd. The system is password protected. Enter your login ID and Password." Below this, there are two main sections for authentication. The first section has three fields: "Login Role" (a dropdown menu set to "Administrator"), "Username" (a text box containing "kamarulafizam"), and "Password" (a text box with four red dots). The second section, titled "Login to Database Server", has four fields: "HostName" (text box with "localhost"), "Port No." (text box with "3306"), "Username" (text box with "root"), and "Password" (text box with six red dots). At the bottom, there are "OK" and "Cancel" buttons.

Figure 3.1: Authentication of RCC

When the authentication procedure is successfully passed, then the main interface will appear. Here one can observe the menu on the left hand side and a list of registered substation on the right hand side. The menu consists of substation enrollment, setting, log, internet/networking, GSM modem control panel, and remote server desktop. When the software is executed, it will read the default setting from the computer registry and display the appropriate information according to the preloaded information. By using this way, user setting will remain every time a user executes the software.

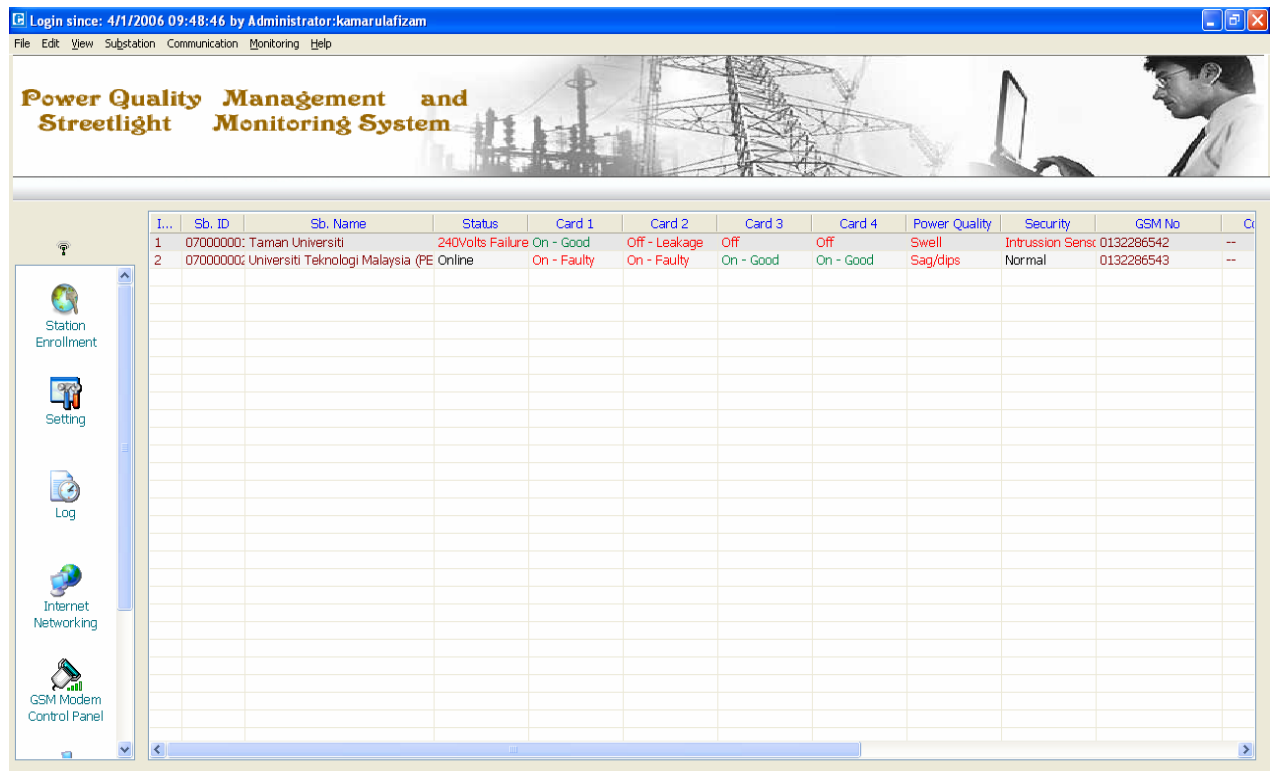


Figure 3.2: Main user interface.

3.2.1 Station Enrollment

Figure 3.3 shows the station enrollment dialog. Note that a station will have a unique identifier by a code number consist of area code and index of enrollment. There are also 4 field for feeder information which represented by card information. The user need to specify the detail information related to the location of the streetlight on a particular feeder. But if the card is not installed, or a particular feeder is not being monitored, the user is advised to leave the name field blank. Note that there are also 5 fields to specify the phone number of contact person. These fields are used to put down the contact of the persons who are responsible to attend to the alert on fault notification. One can enable or disable this contact person.

Substation Enrollment
Add new substation to be monitored

General Information

ID: 070000006 Serial Number:

Name: Prev. Substation:

Location: Next. Substation:

GSM ID: - PMU Source:

Alert to Mobile Phone via SMS
Note: Tick the checkbox to enable sending alert sms to the contact number

Contact I:

Contact II:

Contact III:

Contact IV:

Contact V:

Card Identification Name
Note: If the card is not installed. Leave the name field blank. Card 1 is built-in with the main board

Card 1:

Card 2:

Card 3:

Card 4:

Extra Note/Parameter:

OK Cancel

Figure 3.3: New station enrollment.

3.2.2 Setting

According to the standard interface building scheme, a software need to have a configuration panel to adjust and modify the way the program should run depending on the user. So in this software, the configuration panel has also being created. It consists of general setting, communication, security/authentication, logging, fault code, database management, sound and alert. Figure 3.4 to Figure 3.10 shows the related configuration panel.

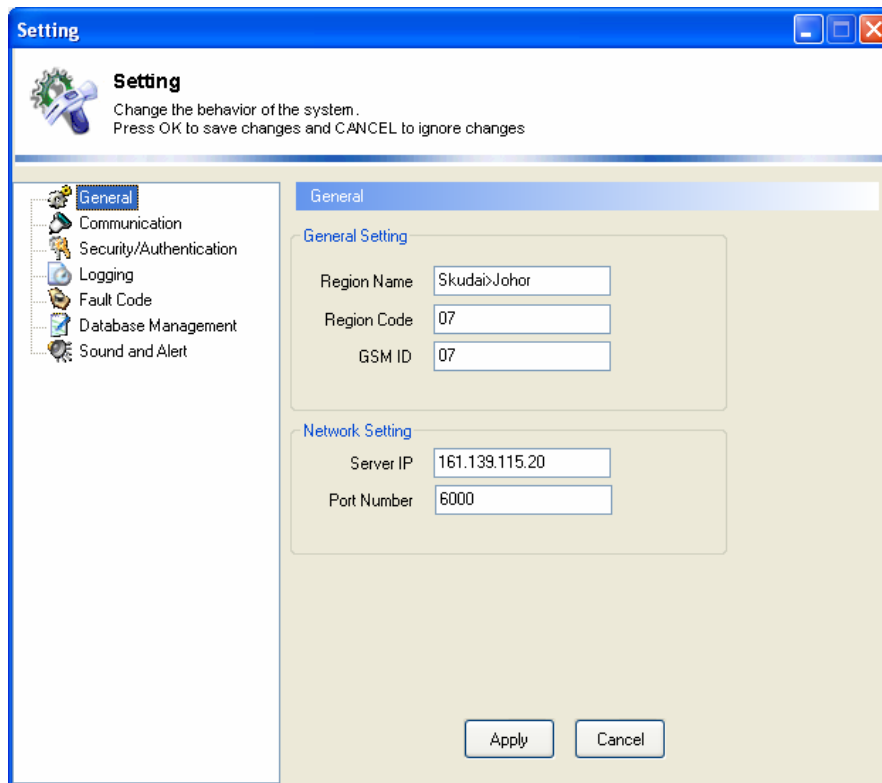


Figure 3.4: System Setting-General

Figure 3.4 shows a general setting of the system. As this software works on the regional basis, the region needs to be set prior to its operation. Here, the region name, region code and gsm id need to be specified. While for internet based communication parameter such as server ip address and communication channel or port need to be specified.

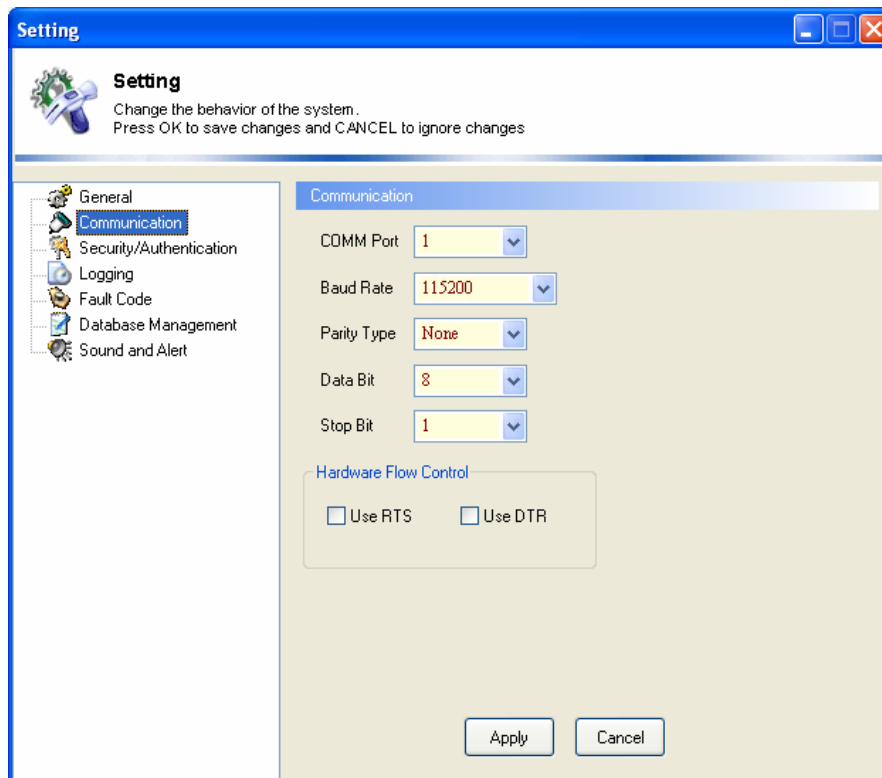


Figure 3.5: System Setting-Communication

This communication setting refers to the procedure to control and monitor the gsm modem. In this system, it uses the serial communication method where certain parameter needs to be specified before it is ready to exchange the data.

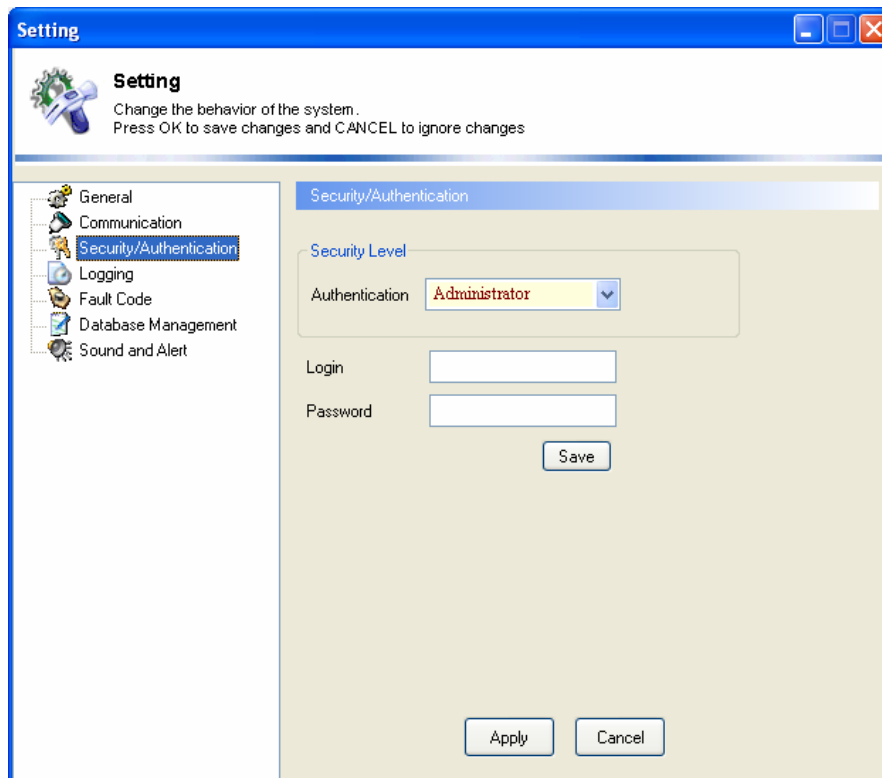


Figure 3.6: System setting- Security/Authentication

In software which implements security and authentication, the control panel to handle this kind of secret information is very important. User enrollment is necessary when more than one user is using the software and when certain privilege is assigned.

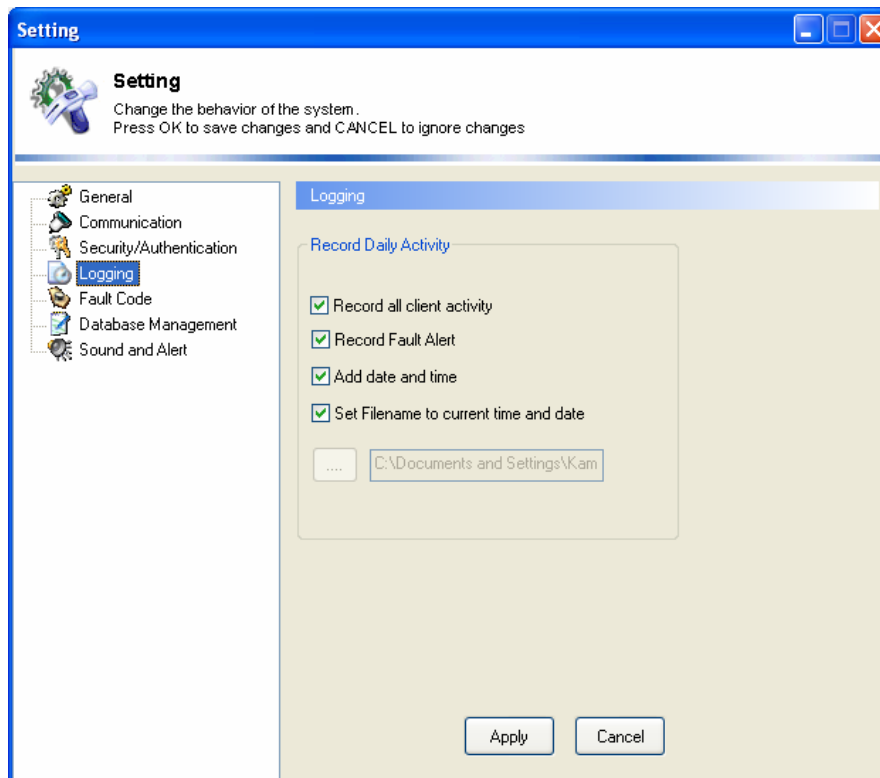


Figure 3.7: System Setting-Logging

The software is programmed to log all activities which involved the user intervention and self-operating routine and procedure. By this way, one can track the software activity as well a record to trace error. The log files can be set whether the to use a custom identifier or current date and time for the filename.

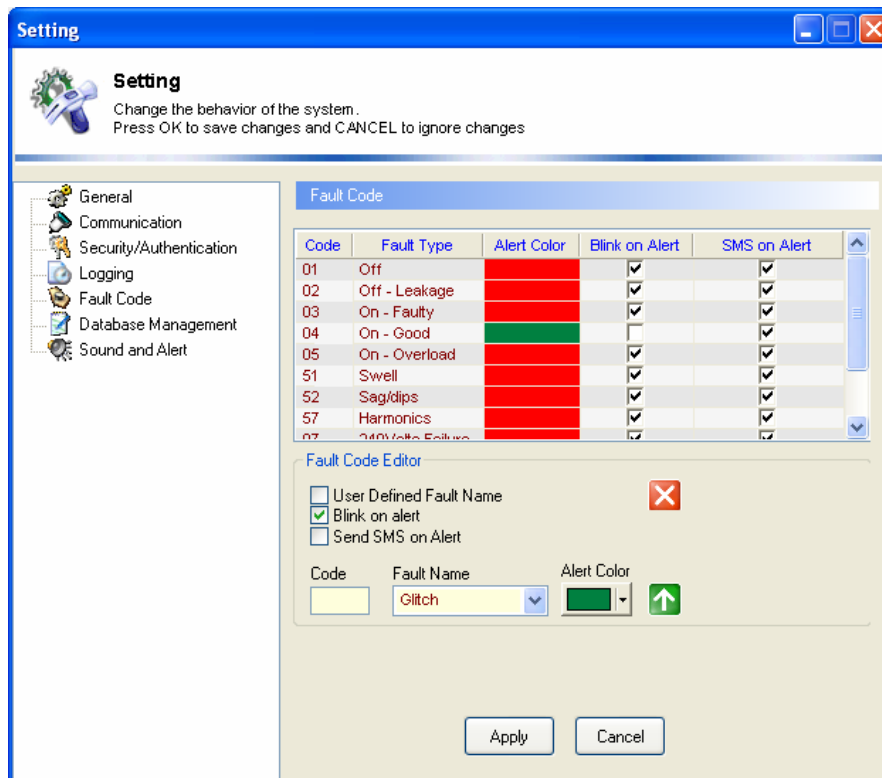


Figure 3.8: System Setting-Fault Code

This software is programmed to respond to the preset alert. The above figure shows a table of which the alert is registered as well as the system behavior once it received the alert. These include alert notification color, blinking alert and send sms to user alert. This setting can be change, add and delete and this framework has provide a flexible and a ready-to-expand system.

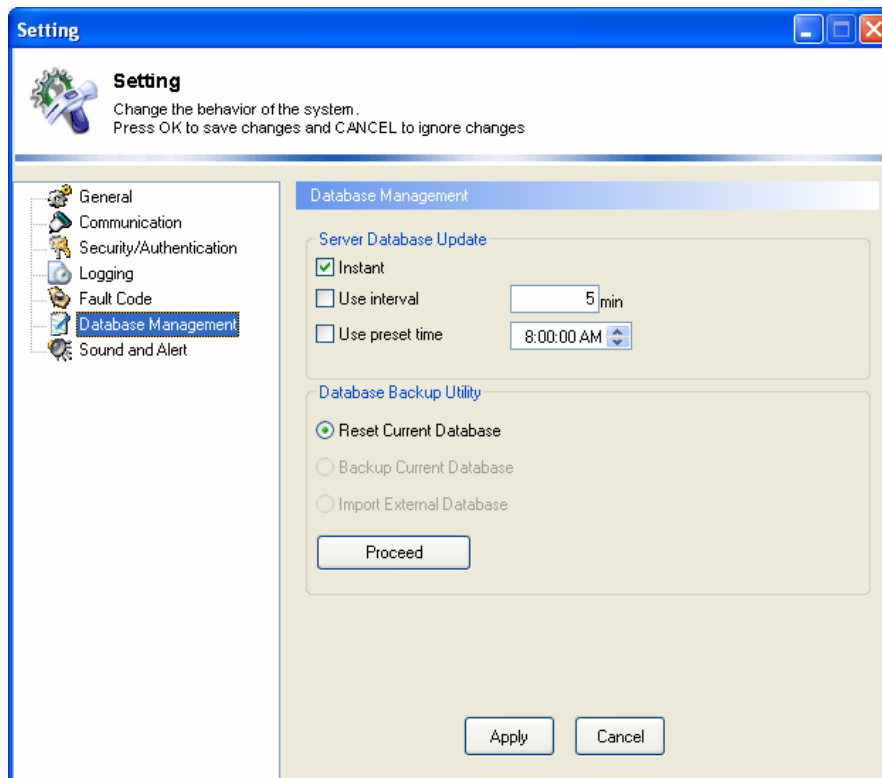


Figure 3.9: System setting-Database Management

As the monitoring system involved a lot of data, database management remain one of the important module. Here the database can be backup, import/export and deleted/refreshed. The system need to forward the alert to the NCC using the internet. But as the internet connection is not a guarantee, this system is ready to tolerate with connection failure. The data submission can be scheduled according to the preset hours, preset interval and instant submission.

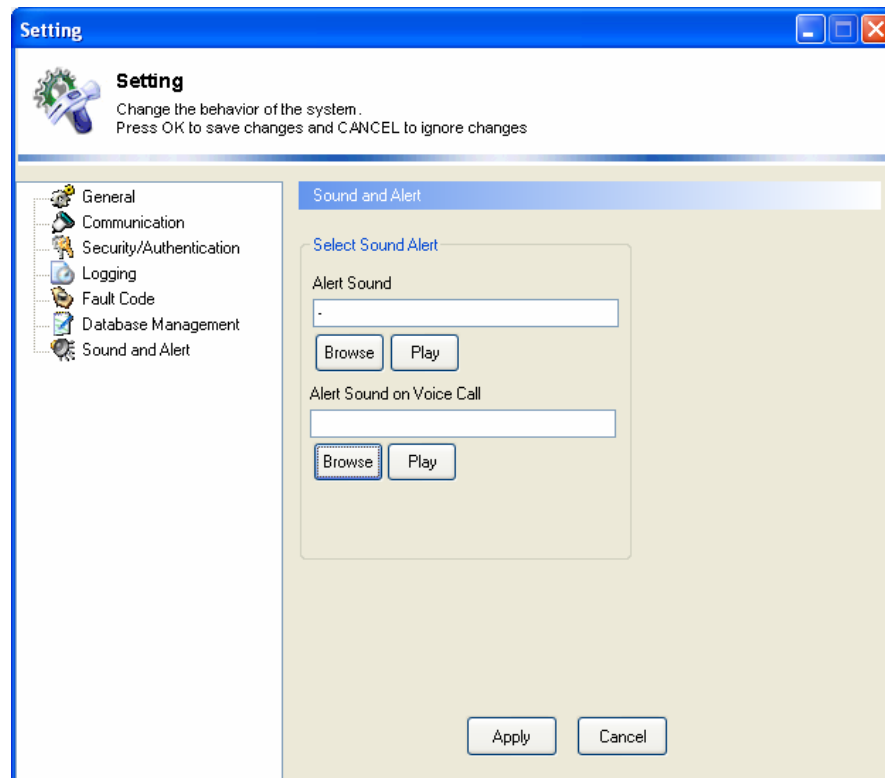


Figure 3.10: System Setting-Sound Alert

The most popular alert method is by using sound. In this system, one can preset the preferred sound scheme to indicate alert notification.

3.2.3 System Log

For further reference and analysis, the software incorporated a system log of the received alert. The log consists of complete information needed to identify a fault error including time, date, fault location, staff on duty and detail of the fault. This information can be viewed and sorted according to the date, source of fault and type of fault. Then one can transform the data into graphical representation for a better view and analysis. These include bar graph, pie chart and line graph.

Activity logging information

File View

Substation Logging Activity
 Substation message will be logged as the server receive the notification.
 The logging history can be review by the duration selected.
 Trend analysis is also provided. Explore the menu for the details

Date	Time	Station	Event	Detail	Staff on dut
Sat-01/10/2005	09:18:15 AM	Universiti Teknologi	On - Good	Card1	Administrator/ka
Sat-01/10/2005	09:20:28 AM	Universiti Teknologi	Sag/dips		Administrator/ka
Sat-01/10/2005	09:20:36 AM	Universiti Teknologi	Sag/dips		Administrator/ka
Sat-01/10/2005	09:25:49 AM	Taman Universiti	240Volts Rec		Administrator/ka
Sat-01/10/2005	09:51:40 AM	Taman Universiti	Off	Card2	Administrator/ka
Sat-01/10/2005	09:52:13 AM	Taman Universiti	Off	Card3	Administrator/ka
Sat-01/10/2005	09:52:48 AM	Taman Universiti	Off	Card4	Administrator/ka
Sat-01/10/2005	09:56:12 AM	Taman Universiti	Off	Card1	Administrator/ka
Sat-01/10/2005	09:59:10 AM	Universiti Teknologi	Off	Card1	Administrator/ka
Sat-01/10/2005	10:00:58 AM	Universiti Teknologi	Off	Card4	Administrator/ka
Sat-01/10/2005	10:03:55 AM	Universiti Teknologi	Off	Card1	Administrator/ka
Sat-01/10/2005	10:07:40 AM	Universiti Teknologi	Off	Card1	Administrator/ka
Sat-01/10/2005	11:32:02 PM	Universiti Teknologi	On - Good	Card1	Administrator/ka
Sat-01/10/2005	11:32:05 PM	Universiti Teknologi	On - Good	Card1	Administrator/ka
Sat-01/10/2005	11:32:10 PM	Universiti Teknologi	On - Good	Card1	Administrator/ka
Sat-01/10/2005	11:32:14 PM	Universiti Teknologi	On - Faulty	Card1	Administrator/ka
Sun-02/10/2005	08:57:35 AM	Taman Universiti	240Volts Rec		Administrator/ka
Sun-02/10/2005	08:59:59 AM	Taman Universiti	Off	Card1	Administrator/ka
Sun-02/10/2005	09:05:49 AM	Taman Universiti	Off	Card1	Administrator/ka

Clear All Items Load All Records Analysis

View Single Substation

View by Substation Name All Search

Select Duration to View

From 1/ 4/2006 To 1/ 4/2006 View by Time

Figure 3.11: System logging information

3.2.4 Internet/Networking.

As the system communicates with the server via the internet, RCC software has the capability to control and monitor the connection with the server. The internet/networking control panel has provided quite a powerful tool for this purpose.

Prior to establish connection, the user needs to specify the appropriate server ip and port number. This information is very important as the system will exchange a lot of data in the network system which might cover the whole world. The system has a multithreaded routine to listen the incoming connection and each connection will be assigned to a new process. By this way, each connection can be handled simultaneously utilizing optimized computer scheduling.

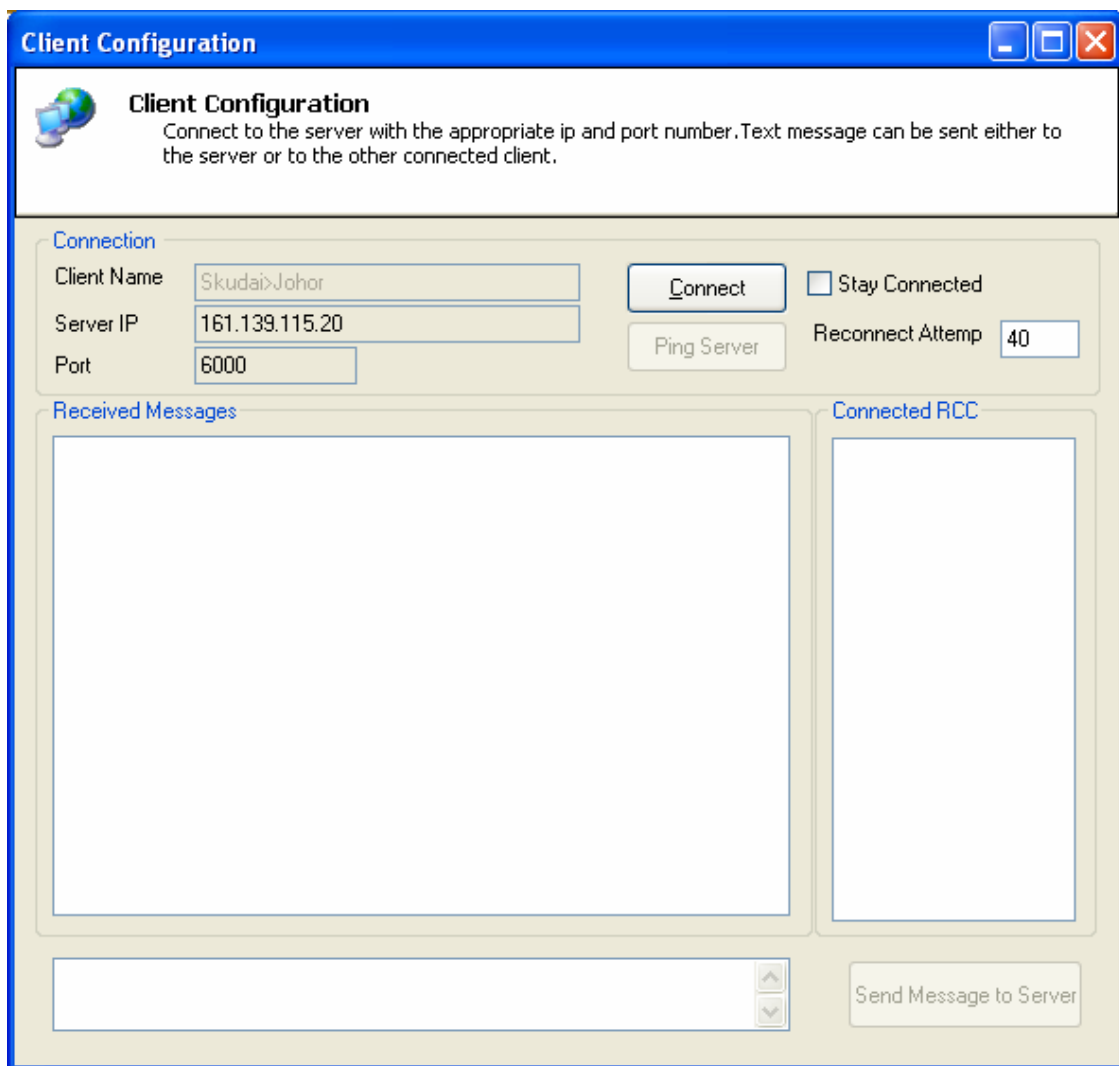


Figure 3.12: Internet/Networking control panel

3.2.5 GSM Control Panel.

The system monitor and control the gsm modem via the communication procedure of serial data transmission. The GSM modem is a very complex device where the information of the current gsm network setting is being transferred continuously. So the communication routine needs to accommodate these features by providing a robust communication interface to handle all sorts of messages and decode them into meaningful messages. The most important routine in this communication scheme is sms based messaging system. The system receives the fault notification via short message service. Thus, the sms identification and decoding scheme need to be very accurate to prevent loss of data of delayed of data.

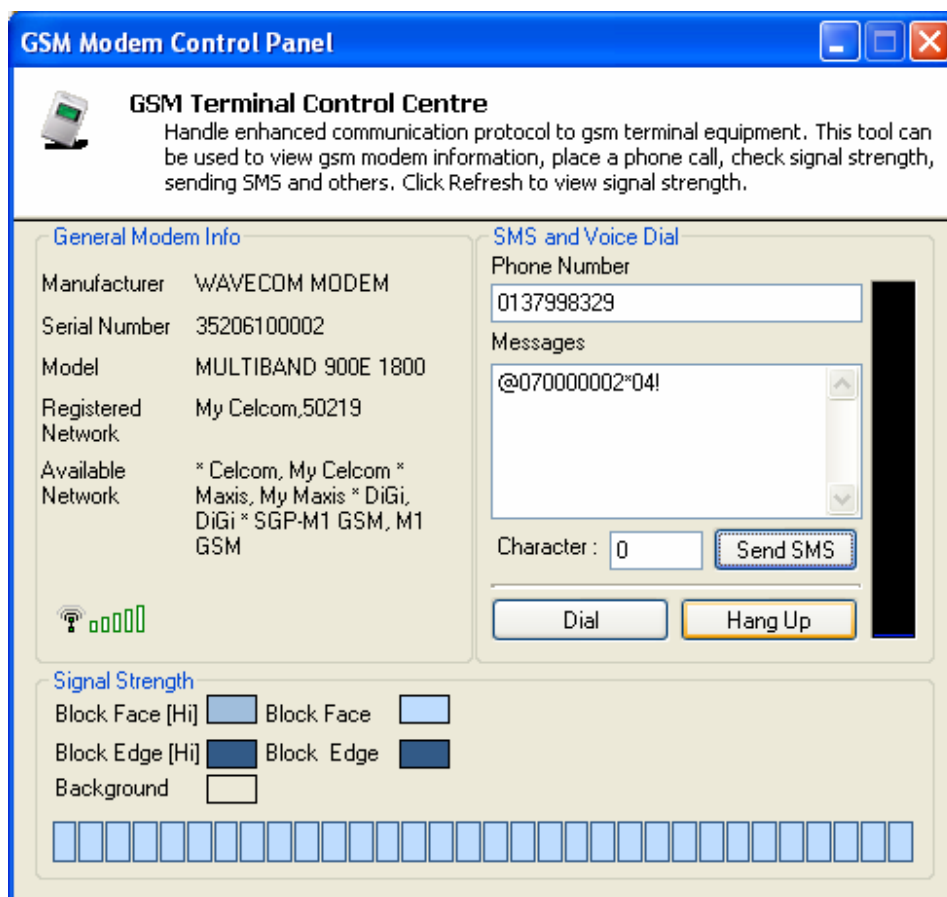


Figure 3.13: GSM modem control panel.

3.3 National Control Centre-NCC.

The NCC acts as centralized server where it monitors all the connected clients. It run in windows based computer with a reliable internet connection. As the RCC, NCC also deploys a multilevel access security where different role of login will have different option of privilege.

Figure 3.14: Login/ authentication dialog

When a user has successfully logged in, the main interface will appear. On the left hand side, one can observed a tree holding the information of regional control centre-RCC. And at the right hand side, one can observe a list containing the substation related to the RCC clicked on the tree. Note that, all the action being taken at the server level will be transmitted to the related client which hold the identical information and vice versa. One can add, delete and rename the regional information. The server will have the full control over the connected client including shutting down the program or the client computer. But this kind of features only available in the administrator mode.

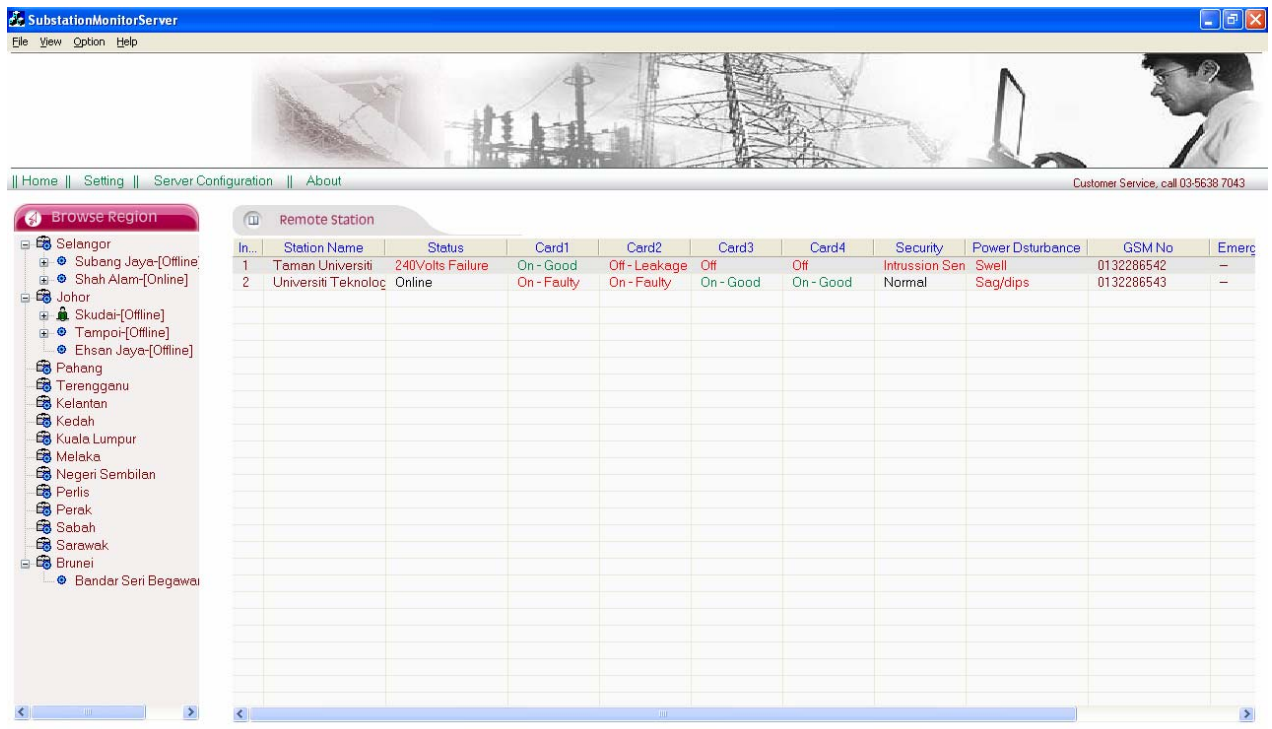


Figure 3.15: Main Interface dialog

Server holds all the client connectivity information. It runs an endless loop to listen to the client request or connection. The communication process exchanges the data in the secure manner where all the information is encrypted via a 128-bit blowfish algorithm. Furthermore, the communication packet is self defined where the data is exchanged with a stream of structured with holds the information agreed by both sender and recipient. The server can close connection to the client and might as well ping them to check the connection status.

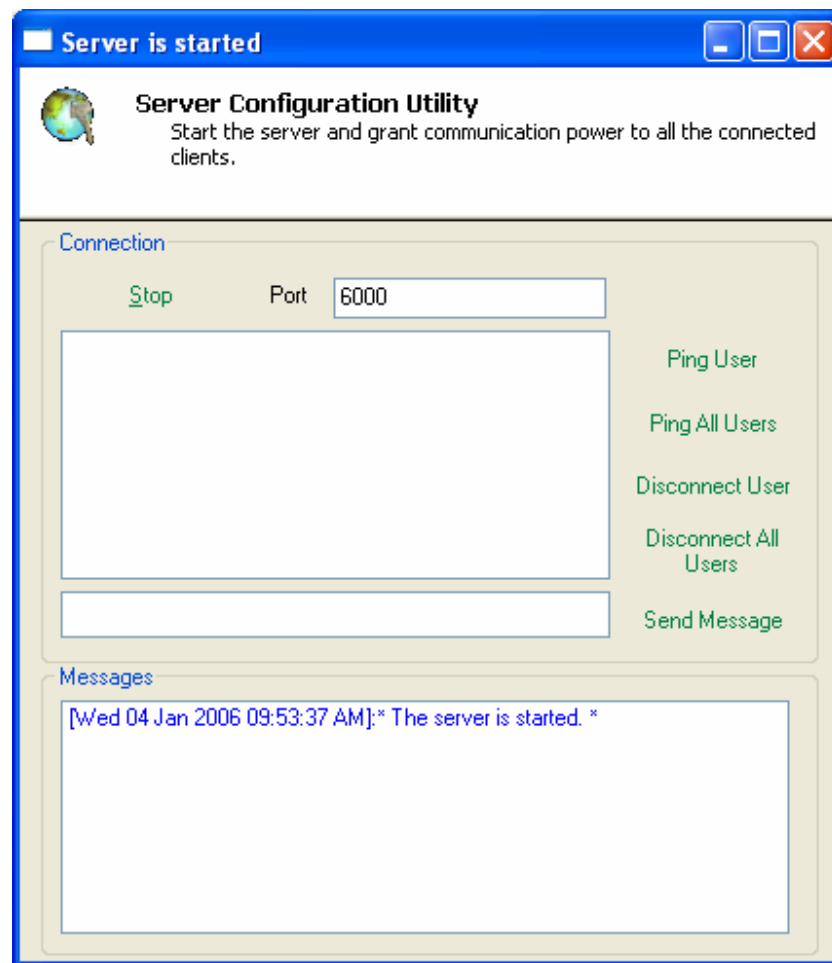


Figure 3.16: Internet/networking server control panel

CHAPTER IV

COMMUNICATION INTERFACE

4.1 GSM Network

The most important communication method used in our system is short message service using GSM networks. We use this facility to relay the fault notification message to a central regional station. This technology has been in the market since 20 years ago. A few years back, we have seen a lot of revolution occur in the communication area, especially in the mobile communication using GSM. Nowadays, GSM network is no longer carry short message and voice, but it also carry multimedia message and video streaming. Even though there many improvements occur in the GSM network, the messaging system still remains with the improved quality and reliability.

SMS stands for short message service. Simply put, it is a method of communication that sends text between cell phones, or from a PC or handheld to a cell phone. The "short" part refers to the maximum size of the text messages: 160 characters (letters, numbers or symbols in the Latin alphabet). For other alphabets, such as Chinese, the maximum SMS size is 70 characters.

Even if you are not talking on your cell phone, your phone is constantly sending and receiving information. It is talking to its cell phone tower over a pathway called a control channel. The reason for this chatter is so that the cell phone system knows which cell your phone is in, and so that your phone can change cells as you move around. Every so often, your phone and the tower will exchange a packet of data that lets both of them know that everything is OK.

Your phone also uses the control channel for call setup. When someone tries to call you, the tower sends your phone a message over the control channel that tells your phone to play its ring tone. The tower also gives your phone a pair of voice channel frequencies to use for the call.

The control channel also provides the pathway for SMS messages. When a friend sends you an SMS message, the message flows through the SMSC, then to the tower, and the tower sends the message to your phone as a little packet of data on the control channel. In the same way, when you send a message, your phone sends it to the tower on the control channel and it goes from the tower to the SMSC and from there to its destination.



Figure 4.1: GSM Communication

The actual data format for the message includes things like the length of the message, a time stamp, the destination phone number, the format, etc. For a complete byte-by-byte breakdown of the message format.

4.1.1 Why use SMS?

SMS has several advantages. It is more discreet than a phone conversation, making it the ideal form for communicating when you don't want to be overheard. It is often less time-consuming to send a text message than to make a phone call or send an e-mail. SMS doesn't require you to be at your computer like e-mail and instant messaging (IM) do -- although some phones are equipped for mobile e-mail and IM services.

SMS is a store-and-forward service, meaning that when you send a text message to a friend, the message does not go directly to your friend's cell phone. The advantage of this method is that your friend's cell phone doesn't have to be active or in range for you to send a message. The message is stored in the SMSC (for days if necessary) until your friend turns his cell phone on or moves into range, at which point the message is delivered. The message will remain stored on your friend's SIM card until he deletes it.

In addition to person-to-person messages, SMS can be used to send a message to a large number of people at a time, either from a list of contacts or to all the users within a particular area. This service is called **broadcasting** and is used by companies to contact groups of employees or by online services to distribute news and other information to subscribers.

In a 2004 University of Plymouth study on the psychology of SMS users, researchers found that mobile phone users were primarily either "texters" or "talkers". Compared to the talkers, the texters sent nearly double the number of SMS messages and made less than half as many voice calls per month. The texters preferred SMS to voice calls for its convenience as well as for the ability to review a message before sending it.

Companies have come up with many uses for the service beyond just your typical person-to-person message. Because SMS doesn't overload the network as much as phone calls, it is frequently used by TV shows to let viewers vote on a poll topic or for a contestant. As a promotional tool, wireless carriers put up giant screens at concerts and other large-scale events to display text messages from people in the audience.

You can use text messaging subscription services to get medication reminders sent to your phone, along with weather alerts, news headlines or even novels broken into 160-character "chapters." Internet search engines such as Yahoo! and Google have short messaging services that enable users to get information such as driving directions, movie show times or local business listings just by texting a query to the search engine's phone number. Social networking services such as Dodgeball use SMS to alert people who live in big cities when their friends or crushes are nearby. The possibilities for integrating SMS into your lifestyle seem endless.

4.2 PLCC

Another important block in our system is the power line communication carrier technology. Power line communications stands for the use of power supply grid for communication purpose. Power line network has very extensive infrastructure in nearly each building. Because of that fact the use of this network for transmission of data in addition to power supply has gained a lot of attention. Since power line was devised for transmission of power at 50-60 Hz and at most 400 Hz, the use this medium for data transmission, at high frequencies, presents some technically challenging problems. Besides large attenuation, power line is one of the most electrically contaminated environments, which makes communication extremely difficult. Further more the restrictions imposed on the use of various frequency bands in the power line spectrum limit the achievable data rates

Power line communication (PLC), also called Broadband over Power Lines (BPL) or Power Line Telecoms (PLT), is a wired technology that is able to use the current electricity networks for data and voice transmission. The carrier can communicate voice and data by superimposing an analog signal over the standard 50 or 60 Hz alternating current (AC). Traditionally electrical utilities used low-speed power-line carrier circuits for control of substations, voice communication, and protection of high-voltage transmission lines. One example of this technology is SCADA. More recently, high-

speed data transmission has been developed using the lower voltage transmission lines used for power distribution. A short-range form of power-line carrier is used for home automation and intercoms.

4.2.1 Types of PLC technology

4.2.1.1 Indoors/Short Range

Indoors, the PLC equipment can use the household electrical power wiring as a transmission medium. This is a technique used in home automation for remote control of lighting and appliances without installation of additional control wiring. The Home Plug system is an example of this technology. The X10 home automation system uses power line communication at the zero crossing voltage point in the AC wave.

Typically these devices operate by injecting a carrier wave of between 20 and 200 kHz into the household wiring at the transmitter. The carrier is modulated by digital signals. Each receiver in the system has an address and can be individually commanded by the signals transmitted over the household wiring and decoded at the receiver. These devices may either be plugged into regular power outlets or else permanently wired in place. Since the carrier signal may propagate to nearby homes (or apartments) on the same distribution system, these control schemes have a "house address" that designates the owner. There are also some very low-bit rate power line communication systems used for automatic meter reading.

4.2.1.2 Outdoors/Long Haul

Utility companies use special coupling capacitors to connect low-frequency radio transmitters to the power-frequency AC conductors. Frequencies used are in the range of 30 to 300 kHz, with transmitter power levels up to hundreds of watts. These signals may

be impressed on one conductor, on two conductors or on all three conductors of a high-voltage AC transmission line. Several different PLC channels may be coupled onto one HV line. Filtering devices are applied at substations to prevent the carrier frequency current from being bypassed through the station apparatus and to ensure that distant faults do not affect the isolated segments of the PLC system. These circuits are used for control of switchgear, and for protection of transmission lines. For example, a protection relay can use a PLC channel to trip a line if a fault is detected between its two terminals, but to leave the line in operation if the fault is elsewhere on the system.

While utility companies use microwave and now, increasingly, fiber optic cables for their primary system communication needs, the power-line carrier apparatus may still be useful as a backup channel or for very simple low-cost installations that do not warrant a fibre drop.

4.2.1.3 Automotive

Power-line technology enables in-vehicle network communication of Data, Voice, Music and Video signals by digital means over Direct Current (DC) battery power-line. Advanced digital communication techniques tailored to overcome hostile and noisy environment are implemented in a small size silicon device. One power-line can be used for multiple independent networks.

Prototypes are successfully operational in vehicles, using automotive compatible protocols such as CAN-bus, LIN sub-bus, and DC-bus.

Automotive applications include Mechatronics (e.g. Climate control, Door module, Immobilizer, Obstacle detector). Telematics and Multimedia. Benefits Saving the Cost and Weight of Ordinary Wiring, Flexible Modifications Simplicity of Installation.

4.2.2 Broadband over power lines

Broadband over Power Lines (BPL) aka Powerband is the use of PLC technology to provide broadband Internet access through ordinary power lines. A computer (or any other device) would need only to plug a BPL "modem" into any outlet in an equipped building to have high-speed Internet access.

BPL offers obvious benefits over regular cable or DSL connections: the extensive infrastructure already available would appear to allow more people in more locations to have access to the Internet. Also, such ubiquitous availability would make it much easier for other electronics, such as televisions or sound systems, to hook up. However, variations in the physical characteristics of the electricity network and the current lack of IEEE standards mean that provisioning of the service is far from being a standard, repeatable process and the amount of bandwidth a BPL system can provide compared to cable and wireless is in question.

High-speed data transmission or Broadband over Power Line uses the electric circuit between the electric substations and home networks. A standard used for this is ETSI PLT. PLC modems transmit in medium and high frequency (1.6 to 30 MHz electric carrier). The asymmetric speed in the modem is generally from 256 kbit/s to 2.7 Mbit/s. In the repeater situated in the meter room the speed is up to 45 Mbit/s and can be connected to 256 PLC modems. In the medium voltage stations, the speed from the head ends to the Internet is up to 135 Mbit/s. To connect to the Internet, utilities can use optical fiber backbone or wireless link.

Much higher speed transmissions using microwave frequencies transmitted via a newly discovered surface wave propagation mechanism have been demonstrated using only a single power line conductor. These systems have shown the potential for symmetric and full duplex communication in excess of 1 Gbit/s in each direction. Multiple WiFi channels as well as simultaneous analog television in the 2.4 and 5.3 GHz unlicensed bands have been demonstrated operating over medium voltage lines.

Differences in the electrical distribution systems in North America and Europe affect the implementation of BPL. In North America relatively few homes are connected to each distribution transformer, whereas European practice may have hundreds of homes connected to each substation. Since the BPL signals do not propagate through the distribution transformers, extra equipment is needed in the North American case.

4.2.3 IEEE

IEEE P1901 is a working group for delivering broadband over power lines. The aim is to define medium access control and physical layer specifications that can be all classes of BPL devices - from the long distance connections to those within the home. Other related IEEE groups are:

IEEE BPL — Standardization of Broadband Over Power Line Technologies

IEEE P1675 — Standard for Broadband over Power Line Hardware

IEEE P1775 — Another project approved by NesCom (IEEE Communications Society) on 12 May 2005 focuses on PLC equipment, electromagnetic compatibility requirements, and testing and measurement methods.

Power line communication technology is a new and fast growing technology due to its potential and huge demand. This is one of the most important future technologies with a lot of rooms of improvement. In our system, there is no specific requirement for this plcc. We just use a generic kind of plcc modem and that will make our system become vulnerable to technology changes.

4.3 TCPIP

As the world become smaller with the tremendous reach of mobile communication technology, the growth of the internet and networking technology has also shown a great potential to become a very important transport to carry information around the globe. Nowadays, almost every area on the earth has the internet facilities. The rapid development has made this facilities become very popular and the reliability and cheap service has made many people chose the internet as their main media of communication for business and others.

For the advantages, we chose the internet and networking facilities to be an important part of the system. The data can be sent and access from all over the world using the computers and network connectivity. This technology has shown a great demand and the development is very fast and has a very huge potential to be utilized in the future.

An increasing number of people are using the Internet and, many for the first time, are using the tools and utilities that at one time were only available on a limited number of computer systems (and only for really intense users!). One sign of this growth in use has been the significant number of TCP/IP and Internet books, articles, courses, and even TV shows that have become available in the last several years; there are so many such books that publishers are reluctant to authorize more because bookstores have reached their limit of shelf space! This memo provides a broad overview of the Internet and TCP/IP, with an emphasis on history, terms, and concepts. It is meant as a brief guide and starting point, referring to many other sources for more detailed information

TCP/IP is most commonly associated with the UNIX operating system. While developed separately, they have been historically tied, as mentioned above, since 4.2BSD Unix started bundling TCP/IP protocols with the operating system. Nevertheless, TCP/IP protocols are available for all widely-used operating systems today and native TCP/IP

support is provided in OS/2, OS/400, and Windows 9x/NT/2000, as well as most Unix variants.

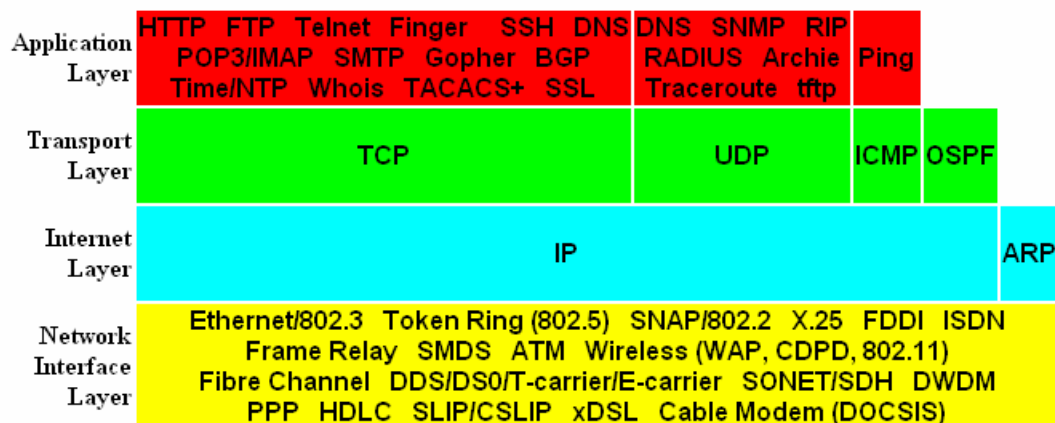


Figure 4.2: Abbreviated TCP/IP protocol stack

Figure 4.2 shows the TCP/IP protocol architecture; this diagram is by no means exhaustive, but shows the major protocol and application components common to most commercial TCP/IP software packages and their relationship.

4.3.1 The Network Interface Layer

The TCP/IP protocols have been designed to operate over nearly any underlying local or wide area network technology. Although certain accommodations may need to be made, IP messages can be transported over all of the technologies shown in the figure, as well as numerous others. It is beyond the scope of this paper to describe most of these underlying protocols and technologies.

Two of the underlying network interface protocols, however, are particularly relevant to TCP/IP. The Serial Line Internet Protocol (SLIP, RFC 1055) and Point-to-

Point Protocol (PPP, [RFC 1661](#)), respectively, may be used to provide data link layer protocol services where no other underlying data link protocol may be in use, such as in leased line or dial-up environments. Most commercial TCP/IP software packages for PC-class systems include these two protocols. With SLIP or PPP, a remote computer can attach directly to a host server and, therefore, connect to the Internet using IP rather than being limited to an asynchronous connection.

4.3.1.1 PPP

It is worth spending a little bit of time discussing PPP because of its importance in Internet access today. As its name implies, PPP was designed to be used over point-to-point links. In fact, it is the prevalent IP encapsulation scheme for dedicated Internet access as well as dial-up access. One of the significant strengths of PPP is its ability to negotiate a number of things upon initial connection, including passwords, IP addresses, compression schemes, and encryption schemes. In addition, PPP provides support for simultaneous multiple protocols over a single connection, an important consideration in those environments where dial-up users can employ either IP or another network Layer protocol. Finally, in environments such as ISDN, PPP supports inverse multiplexing and dynamic bandwidth allocation via the Multilink-PPP (ML-PPP) described in [RFCs 1990](#) and [2125](#).

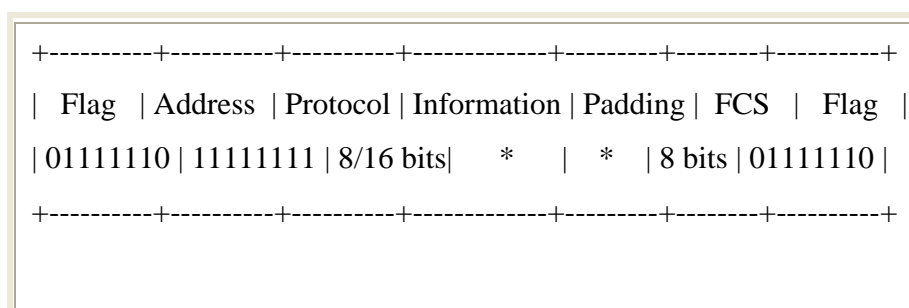


Figure 4.3: PPP frame format (using HDLC)

PPP generally uses an HDLC-like (bit-oriented protocol) frame format as shown in Figure 4.3, although RFC 1661 does not demand use of HDLC. HDLC defines the first and last two fields in the frame:

- *Flag*: The 8-bit pattern "01111110" used to delimit the beginning and end of the transmission.
- *Address*: For PPP, uses the 8-bit broadcast address, "11111111".
- *Frame Check Sequence (FCS)*: An 8-bit remainder from a cyclic redundancy check (CRC) calculation, used for bit error detection.

RFC 1661 actually describes the use of the three other fields in the frame:

- *Protocol*: An 8- or 16-bit value that indicates the type of datagram carried in this frame's Information field. This field can indicate use of a particular Network Layer protocol (such as IP, IPX, or DDP), a Network Control Protocol (NCP) in support of one of the Network Layer protocols, or a PPP Link-layer Control Protocol (LCP). The entire list of possible PPP values in this field can be found in the [IANA list of PPP protocols](#).
- *Information*: Contains the datagram for the protocol specified in the Protocol field. This field is zero or more octets in length, up to a (default) maximum of 1500 octets (although a different value can be negotiated).
- *Padding*: Optional padding to add length to the Information field. May be required in some implementations to ensure some minimum frame length and/or to ensure some alignment on computer word boundaries.

The operation of PPP is basically as follows:

1. After the link is physically established, each host sends LCP packets to configure and test the data link. It is here where the maximum frame length, authentication protocol (Password Authentication Protocol, PAP, or Challenge-Handshake Authentication Protocol, CHAP), link quality protocol, compression protocol, and other configuration parameters are negotiated. Authentication, *if* it used, will occur after the link has been established.

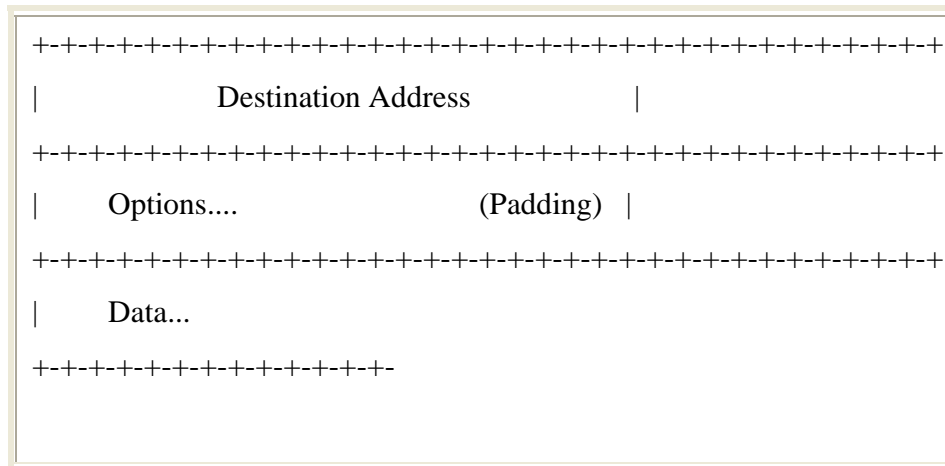


Figure 4.4: IP packet (datagram) header format.

The basic IP packet header format is shown in Figure 4.4. The format of the diagram is consistent with the RFC; bits are numbered from left-to-right, starting at 0. Each row represents a single *32-bit word*; note that an IP header will be at least 5 words (20 bytes) in length. The fields contained in the header, and their functions, are:

- *Version:* Specifies the IP version of the packet. The current version of IP is version 4, so this field will contain the binary value 0100. [NOTE: Actually, many IP version numbers have been assigned besides 4 and 6; see the [IANA's list of IP Version Numbers](#).]
- *Internet Header Length (IHL):* Indicates the length of the datagram header in 32 bit (4 octet) words. A minimum-length header is 20 octets, so this field always has a value of at least 5 (0101). Since the maximum value of this field is 15, the IP Header can be no longer than 60 octets.
- *Type of Service (TOS):* Allows an originating host to request different classes of service for packets it transmits. Although not generally supported today in IPv4, the TOS field can be set by the originating host in response to service requests across the Transport Layer/Internet Layer service interface, and can specify a service priority (0-7) or can request that the route be optimized for either cost, delay, throughput, or reliability.

- *Total Length*: Indicates the length (in bytes, or octets) of the entire packet, including both header and data. Given the size of this field, the maximum size of an IP packet is 64 KB, or 65,535 bytes. In practice, packet sizes are limited to the maximum transmission unit (MTU).
- *Identification*: Used when a packet is fragmented into smaller pieces while traversing the Internet, this identifier is assigned by the transmitting host so that different fragments arriving at the destination can be associated with each other for reassembly.
- *Flags*: Also used for fragmentation and reassembly. The first bit is called the More Fragments (MF) bit, and is used to indicate the last fragment of a packet so that the receiver knows that the packet can be reassembled. The second bit is the Don't Fragment (DF) bit, which suppresses fragmentation. The third bit is unused (and always set to 0).
- *Fragment Offset*: Indicates the position of this fragment in the original packet. In the first packet of a fragment stream, the offset will be 0; in subsequent fragments, this field will indicate the offset in increments of 8 bytes.
- *Time-to-Live (TTL)*: A value from 0 to 255, indicating the number of hops that this packet is allowed to take before discarded within the network. Every router that sees this packet will decrement the TTL value by one; if it gets to 0, the packet will be discarded.
- *Protocol*: Indicates the higher layer protocol contents of the data carried in the packet; options include ICMP (1), TCP (6), UDP (17), or OSPF (89). A complete list of IP protocol numbers can be found at the IANA's list of Protocol Numbers. An implementation-specific list of supported protocols can be found in the protocol file, generally found in the /etc (Linux/Unix), c:\windows (Windows 9x, ME), or c:\winnt\system32\drivers\etc (Windows NT, 2000) directory.
- *Header Checksum*: Carries information to ensure that the received IP header is error-free. Remember that IP provides an *unreliable* service and, therefore, this field only checks the header rather than the entire packet.
- *Source Address*: IP address of the host sending the packet.
- *Destination Address*: IP address of the host intended to receive the packet.

- *Options*: A set of options which may be applied to any given packet, such as sender-specified source routing or security indication. The option list may use up to 40 bytes (10 words), and will be padded to a word boundary; IP options are taken from the IANA's list of IP Option Numbers.

4.3.2.1 IP Addresses

IP addresses are 32 bits in length (Figure 4.5). They are typically written as a sequence of four numbers, representing the decimal value of each of the address bytes. Since the values are separated by periods, the notation is referred to as *dotted decimal*. A sample IP address is 208.162.106.17.

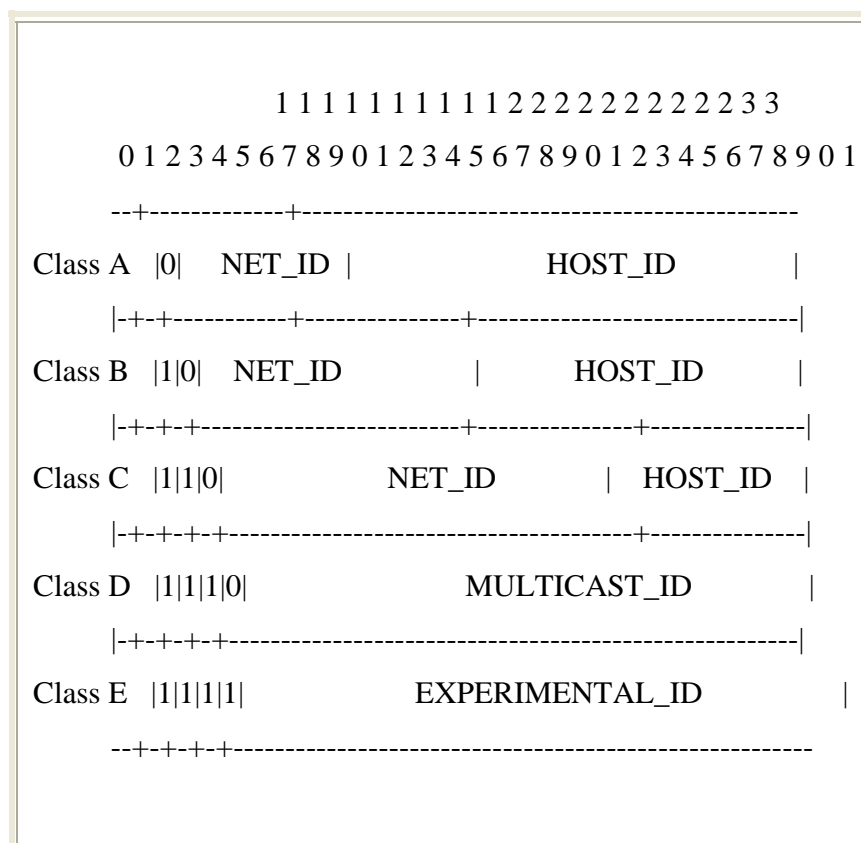


Figure 4.5: IP Address Format

IP addresses are hierarchical for routing purposes and are subdivided into two subfields. The Network Identifier (NET_ID) subfield identifies the TCP/IP sub network connected to the Internet. The NET_ID is used for high-level routing between networks, much the same way as the country code, city code, or area code is used in the telephone network. The Host Identifier (HOST_ID) subfield indicates the specific host within a sub network.

To accommodate different size networks, IP defines several *address classes*. Classes A, B, and C are used for host addressing and the only difference between the classes is the length of the NET_ID subfield:

- A Class A address has an 8-bit NET_ID and 24-bit HOST_ID. Class A addresses are intended for very large networks and can address up to 16,777,214 ($2^{24}-2$) hosts per network. The first bit of a Class A address is a 0 and the NETID occupies the first byte, so there are only 128 (2^7) possible Class A NETIDs. In fact, the first digit of a Class A address will be between 1 and 126, and only about 90 or so Class A addresses have been assigned.
- A Class B address has a 16-bit NET_ID and 16-bit HOST_ID. Class B addresses are intended for moderate sized networks and can address up to 65,534 ($2^{16}-2$) hosts per network. The first two bits of a Class B address are 10 so that the first digit of a Class B address will be a number between 128 and 191; there are 16,384 (2^{14}) possible Class B NETIDs. The Class B address space has long been threatened with being used up and it is has been very difficult to get a new Class B address for some time.
- A Class C address has a 24-bit NET_ID and 8-bit HOST_ID. These addresses are intended for small networks and can address only up to 254 (2^8-2) hosts per network. The first three bits of a Class C address are 110 so that the first digit of a Class C address will be a number between 192 and 223. There are 2,097,152 (2^{21}) possible Class C NETIDs and most addresses assigned to networks today are Class C (or sub-Class C!).

The remaining two address classes are used for special functions only and are not commonly assigned to individual hosts. Class D addresses may begin with a value between 224 and 239 (the first 4 bits are 1110), and are used for IP multicasting (i.e., sending a single datagram to multiple hosts); the IANA maintains a list of Internet Multicast Addresses. Class E addresses begin with a value between 240 and 255 (the first 4 bits are 1111), and are reserved for experimental use.

Several address values are reserved and/or have special meaning. A HOST_ID of 0 (as used above) is a dummy value reserved as a place holder when referring to an entire subnetwork; the address 208.162.106.0, then, refers to the Class C address with a NET_ID of 208.162.106. A HOST_ID of all ones (usually written "255" when referring to an all-ones byte, but also denoted as "-1") is a broadcast address and refers to all hosts on a network. A NET_ID value of 127 is used for loopback testing and the specific host address 127.0.0.1 refers to the *localhost*.

Several NET_IDs have been reserved in [RFC 1918](#) for private network addresses and packets will not be routed over the Internet to these networks. Reserved NET_IDs are the Class A address 10.0.0.0 (formerly assigned to ARPANET), the sixteen Class B addresses 172.16.0.0-172.31.0.0, and the 256 Class C addresses 192.168.0.0-192.168.255.0. An additional addressing tool is the *subnet mask*. Subnet masks are used to indicate the portion of the address that identifies the network (and/or subnetwork) for routing purposes. The subnet mask is written in dotted decimal and the number of 1s indicates the significant NET_ID bits. For "classful" IP addresses, the subnet mask and number of significant address bits for the NET_ID are:

Table 4.1: Subnet number of bits.

Class	Subnet Mask	Number of Bits
A	255.0.0.0	8
B	255.255.0.0	16
C	255.255.255.0	24

Depending upon the context and literature, subnet masks may be written in dotted decimal form or just as a number representing the number of significant address bits for the NET_ID. Thus, 208.162.106.17 255.255.255.0 and 208.162.106.17/24 both refer to a Class C NET_ID of 208.162.106. Some, in fact, might refer to this 24-bit NET_ID as a "slash-24."

Subnet masks can also be used to subdivide a large address space into subnetworks or to combine multiple small address spaces. In the former case, a network may subdivide their address space to define multiple logical networks by segmenting the HOST_ID subfield into a Subnetwork Identifier (SUBNET_ID) and (smaller) HOST_ID. For example, user assigned the Class B address space 172.16.0.0 could segment this into a 16-bit NET_ID, 4-bit SUBNET_ID, and 12-bit HOST_ID. In this case, the subnet mask for Internet routing purposes would be 255.255.0.0 (or "/16"), while the mask for routing to individual subnets within the larger Class B address space would be 255.255.240.0 (or "/20").

But how a subnet masks work? To determine the subnet portion of the address, we simply perform a bit-by-bit logical AND of the IP address and the mask. Consider the following example: suppose we have a host with the IP address 172.20.134.164 and a subnet mask 255.255.0.0. We write out the address and mask in decimal and binary as follows:

	172.020.134.164	10101100.00010100.10000110.10100100
AND	255.255.000.000	11111111.11111111.00000000.00000000
	-----	-----
	172.020.000.000	10101100.00010100.00000000.00000000

From this we can easily find the NET_ID 172.20.0.0 (and can also infer the HOST_ID 134.164).

As an aside, most ISPs use a /30 address for the WAN links between the network and the customer. The router on the customer's network will generally have two IP addresses; one on the LAN interface using an address from the customer's public IP

address space and one on the WAN interface leading back to the ISP. Since the ISP would like to be able to ping both sides of the router for testing and maintenance, having an IP address for each router port is a good idea.

By using a /30 address, a single Class C address can be broken up into 64 smaller addresses. Here's an example. Suppose an ISP assigns a particular customer the address 24.48.165.130 and a subnet mask 255.255.255.252. That would look like the following:

	024.048.165.130	00011000.00110000.10100101.10000010
AND	255.255.255.252	11111111.11111111.11111111.11111100
	-----	-----
	024.048.165.128	00011000.00110000.10100101.10000000

So we find the NET_ID to be 24.48.165.128. Since there's a 30-bit NET_ID, we are left with a 2-bit HOST_ID; thus, there are four possible host addresses in this subnet: 24.48.165.128 (00), .129 (01), .130 (10), and .131 (11). The .128 address isn't used because it is all-zeroes; .131 isn't used because it is all-ones. That leave .129 and .130, which is ok since we only have two ends on the WAN link! So, in this case, the customer's router might be assigned 24.48.165.130/30 and the ISP's end of the link might get 24.48.165.129/30. Use of this subnet mask is very common today (so common that there is a proposal to allow the definition of 2-address NET_IDs specifically for point-to-point WAN links).

- A last and final word about IP addresses is in order. Most Internet protocols specify that addresses be supplied in the form of a fully-qualified host name or an IP address in dotted decimal form. However, spammers and others have found a way to obfuscate IP addresses by supplying the IP address as a single large **decimal** number. Remember that IP addresses are 32-bit quantities. We write the address in dotted decimal for the convenience of humans; the computer still interprets dotted decimal as a 32-bit quantity. Therefore, writing the address as a single large decimal number will still allow the computer to see the address as a 32-bit number.

4.3.2.2 Conserving IP Addresses: CIDR, DHCP, NAT, and PAT

The use of class-based (or *classful*) addresses in IP is one of the reasons that IP address exhaustion has been a concern since the early 1990s. Consider an organization, for example, that needs 1000 IP addresses. A Class C address is obviously too small so a Class B address would get assigned. But a Class B address offers more than 64,000 addresses, so over 63,000 addresses are wasted in this assignment.

An alternative approach is to assign this organization a block of four Class C addresses, such as 192.168.128.0, 192.168.129.0, 192.168.130.0, and 192.168.131.0. By using a 22-bit subnet mask 255.255.252.0 (or "/22") for routing to this "block," the NET_ID assigned to this organization is 192.168.128.0.

This use of variable-size subnet masks is called *Classless Interdomain Routing (CIDR)*, described in RFCs [1518](#) and [1519](#). In the example here, routing information for what is essentially four Class C addresses can be specified in a single router table entry.

But this concept can be expanded even more. CIDR is an important contribution to the Internet because it has dramatically limited the size of the Internet backbone's routing tables. Today, IP addresses are not assigned strictly on a first-come, first-serve basis, but have been preallocated to various numbering authorities around the world. The numbering authorities in turn, assign blocks of addresses to major (or first-tier) ISPs; these address blocks are called *CIDR blocks*. An ISP's customer (which includes ISPs that are customers of a first-tier ISP) will be assigned an IP NET_ID that is part of the ISP's CIDR block. So, for example, let's say that *Gary Kessler ISP* has a CIDR block containing the 256 Class C addresses in the range 196.168.0.0-196.168.255.0. This range of addresses could be represented in a routing table with the single entry 196.168.0.0/16. Once a packet hits the Gary Kessler ISP, it will be routed to the correct end destination.

But don't stop now! By shrinking the size of the subnet mask so that a single NET_ID refers to multiple addresses (resulting in shrinking router tables), we could *extend* the size of the subnet mask to actually assign to an organization something smaller than a Class C address. As the Class C address space falls in danger of being exhausted,

users are under increasing pressure to accept assignment of these *sub-Class C addresses*. An organization with just a few servers, for example, might be assigned, say, 64 addresses rather than the full 256. The standard subnet mask for a Class C is 24 bits, yielding a 24-bit NET_ID and 8-bit HOST_ID. If we use a "/26" mask (255.255.255.192), we can assign the same "Class C" to four different users, each getting 1/4 of the address space (and a 6-bit HOST_ID). So, for example, the IP address space 208.162.106.0 might be assigned as follows:

Table 4.2: IP address space

NET_ID	HOST_ID range	Valid HOST_IDs
208.162.106.0	0-63	1-62
208.162.106.64	64-127	65-126
208.162.106.128	128-191	129-190
208.162.106.192	192-255	193-254

Note that in ordinary Class C usage, we would lose two addresses from the space — 0 and 255 — because addresses of all 0s and all 1s cannot be assigned as a HOST_ID. In the usage above, we would lose eight addresses from this space, because 0, 64, 128, and 192 have an all 0s HOST_ID and 63, 127, 191, and 255 have an all 1s HOST_ID. Each user, then, has 62 addresses that can be assigned to hosts.

The pressure on the Class C address space is continuing in intensity. Today, the pressure is not only to limit the number of addresses assigned, but organizations need to show *why* they need as many addresses as they want. Consider a company with 64 hosts and 3 servers. The ISP may request that that company only obtain 32 IP addresses. The rationale: the 3 servers need 3 addresses but the other hosts might be able to "share" the remaining pool of 27 addresses (recall that we lost HOST_ID addresses 0 and 31).

A pool of IP addresses can be shared by multiple hosts using a mechanism called Network Address Translation (NAT). NAT, described in [RFC 1631](#), is typically implemented in hosts, proxy servers, or routers. The scheme works because every host on the user's network can be assigned an IP address from the pool of RFC 1918 private addresses; since these addresses are never seen on the Internet, this is not a problem.

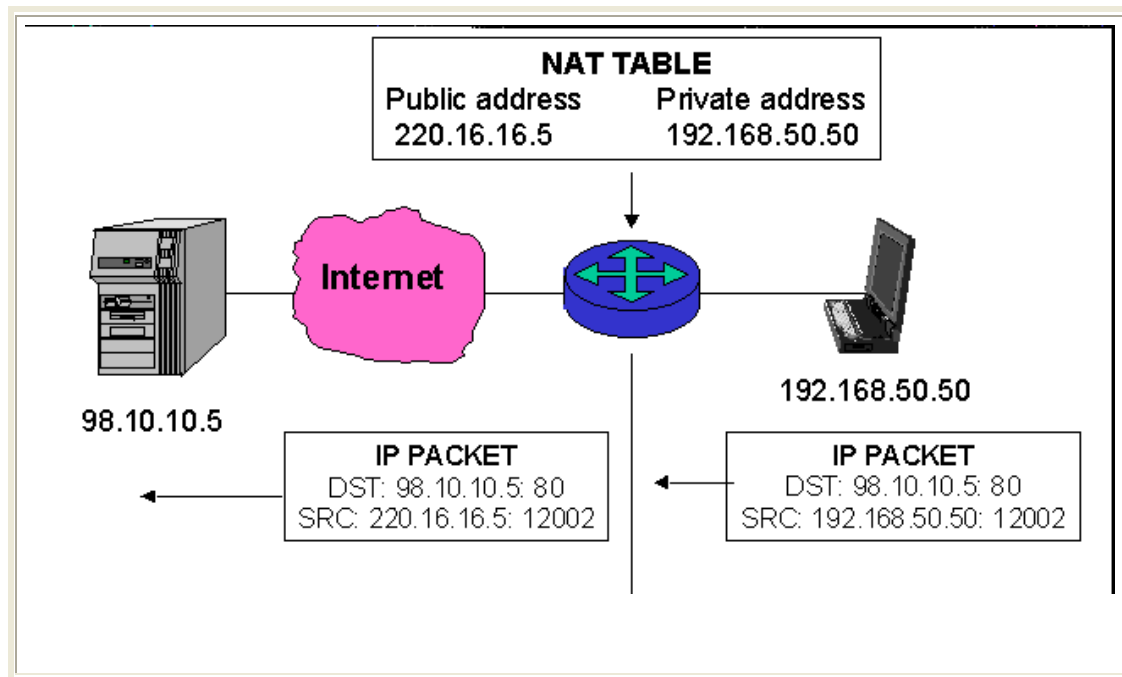


Figure 4.6: Network Address Translation (NAT)

Consider the scenario shown in Figure 4.6. When the user accesses a Web site on the Internet, the NAT server will translate the "private" IP address of the host (192.168.50.50) into a "public" IP address (220.16.16.5) from the pool of assigned addresses. NAT works because of the assumption that, in this example, no more than 27 of the 64 hosts will ever be accessing the Internet at a single time.

But suppose that assumption is wrong. Another enhancement, called Port Address Translation (PAT) or Network Address Port Translation (NAPT), allows multiple hosts to share a single IP address by using different "port numbers" (ports are described more in [Section 3.3](#)).

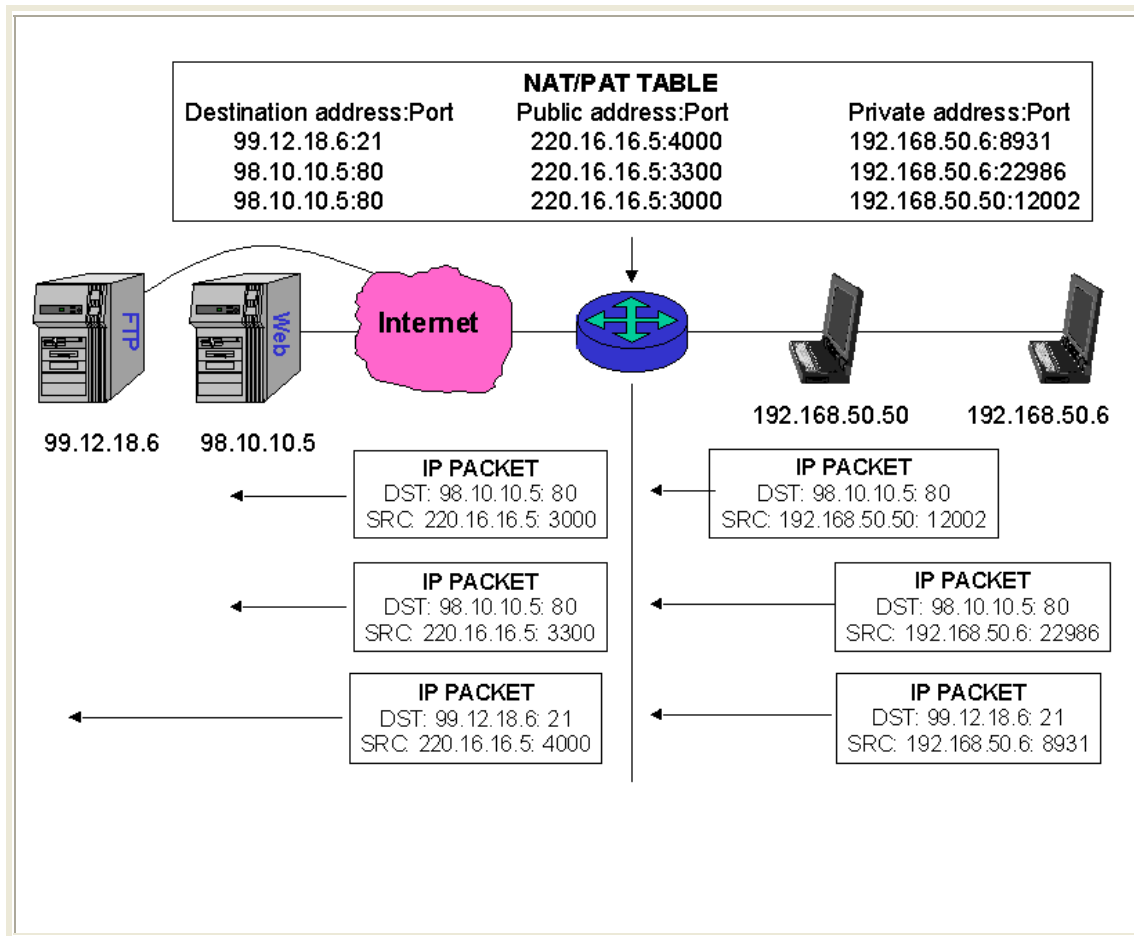


Figure 4.7: Port Address Translation (PAT)

Port numbers are used by higher layer protocols (e.g., TCP and UDP) to identify a higher layer application. A TCP connection, for example, is uniquely identified on the Internet by the four values (aka *4-tuple*) <source IP address, source port, destination IP address, destination port>. The server's port number is defined by the standards while client port numbers can be any number greater than 1023. The scenario in Figure 4.7 shows the following three connections:

- The client with the "private" IP address 192.168.50.50 (using port number 12002) connects to a Web server at address 98.10.10.5 (port 80).
- The client with the "private" IP address 192.168.50.6 (using port number 22986) connects to the same Web server at address 98.10.10.5 (port 80).

- The client with the "private" IP address 192.168.50.6 (using port number 8931) connects to an FTP server at address 99.12.18.6 (port 21).

PAT works in this scenario as follows. The router (running PAT software) can assign both local hosts with the same "public" IP address (220.16.16.5) and differentiate between the three packet flows by the source port.

A final note about NAT and PAT. Both of these solutions work and work fine, but they require that *every* packet be buffered, disassembled, provided with a new IP address, a new checksum calculated, and the packet reassembled. In addition, PAT requires that a new port number be placed in the higher layer protocol data unit and new checksum calculated at the protocol layer above IP, too. The point is that NAT, and particularly PAT, results in a tremendous performance hit.

One advantage of NAT is that it makes IP address renumbering a thing of the past. If a customer has an IP NET_ID assigned from its ISP's CIDR block and then they change ISPs, they will get a new NET_ID. With NAT, only the servers need to be renumbered.

Another way to deal with renumbering is to dynamically assign IP addresses to host systems using the Dynamic Host Configuration Protocol (DHCP). DHCP is also an excellent solution for those environments where users move around frequently; it prevents the user from having to reconfigure their system when they move from, say, the Los Angeles office network to the New York office.

4.3.3 The Domain Name System

While IP addresses are 32 bits in length, most users do not memorize the numeric addresses of the hosts to which they attach; instead, people are more comfortable with host names. Most IP hosts, then, have both a numeric IP address and a name. While this is convenient for people, however, the name must be translated back to a numeric address for routing purposes.

Earlier discussion in this paper described the domain naming structure of the Internet. In the early ARPANET, every host maintained a file called `hosts` that contained a list of all hosts, which included the IP address, host name, and alias(es). This was an adequate measure while the ARPANET was small and had a slow rate of growth, but was not a scalable solution as the network grew.

[NOTE: A `hosts` file is still found on UNIX systems although usually used to reconcile names of hosts on the local network to cut down on local DNS traffic; the file can usually be found in the `/etc` directory. On Microsoft Windows systems, the `HOSTS` file can typically be found in the `c:\windows` folder; in Windows NT and 2000, it can be found in `c:\winnt\system32\drivers\etc.`]

To handle the fast rate of new names on the network, the Domain Name System (DNS) was created. The DNS is a distributed database containing host name and IP address information for all domains on the Internet. There is a single *authoritative name server* for every domain that contains all DNS-related information about the domain; each domain also has at least one secondary name server that also contains a copy of this information. Thirteen *root servers* around the globe (most in the U.S., actually, with the remainder in Asia and Europe) maintain a list of all of these authoritative name servers.

When a host on the Internet needs to obtain a host's IP address based upon the host's name, a DNS request is made by the initial host to a local name server. The local name server may be able to respond to the request with information that is either configured or cached at the name server; if necessary information is not available, the local name server forwards the request to one of the root servers. The root server, then, will determine an appropriate name server for the target host and the DNS request will be forwarded to the domain's name server.

Name server data files contain the following types of records including:

- *A-record*: An address record maps a hostname to an IP address.
- *PTR-record*: A pointer record maps an IP address to a hostname.

- *NS-record*: A name server record lists the authoritative name server(s) for a given domain.
- *MX-record*: A mail exchange record lists the mail servers for a given domain. As an example, consider the author's e-mail address, *kumquat@sover.net*. Note that the "sover.net" portion of the address is a domain name, not a host name, and mail has to be sent to a specific host. The MX-records in the *sover.net* name database specifies the host *mail.sover.net* is the mail server for this domain.
- *CNAME-record*: Canonical name records provide a mechanism of assigning aliases to host names, so that a single host with a IP address can be known by multiple names.

4.3.3.1 ARP and Address Resolution

Early IP implementations ran on hosts commonly interconnected by Ethernet local area networks (LAN). Every transmission on the LAN contains the local network, or medium access control (MAC), address of the source and destination nodes. MAC addresses are 48-bits in length and are non-hierarchical, so routing cannot be performed using the MAC address. MAC addresses are never the same as IP addresses.

When a host needs to send a datagram to another host on the same network, the sending application must know both the IP and MAC addresses of the intended receiver; this is because the destination IP address is placed in the IP packet and the destination MAC address is placed in the LAN MAC protocol frame. (If the destination host is on another network, the sender will look instead for the MAC address of the default gateway, or router.)

Unfortunately, the sender's IP process may not know the MAC address of the intended receiver on the same network. The Address Resolution Protocol (ARP), described in RFC 826, provides a mechanism so that a host can learn a receiver's MAC address when knowing only the IP address. The process is actually relatively simple: the host sends an ARP Request packet in a frame containing the MAC broadcast address; the

ARP request advertises the destination IP address and asks for the associated MAC address. The station on the LAN that recognizes its own IP address will send an ARP Response with its own MAC address. As Figure 4.2 shows, ARP message are carried directly in the LAN frame and ARP is an independent protocol from IP. The IANA maintains a list of all ARP parameters.

Other address resolution procedures have also been defined, including:

- Reverse ARP (RARP), which allows a disk-less processor to determine its IP address based on knowing its own MAC address
- Inverse ARP (InARP), which provides a mapping between an IP address and a frame relay virtual circuit identifier
- ATMARP and ATMInARP provide a mapping between an IP address and ATM virtual path/channel identifiers.
- LAN Emulation ARP (LEARP), which maps a recipient's ATM address to its LAN Emulation (LE) address (which takes the form of an IEEE 802 MAC address).

[NOTE: IP hosts maintain a cache storing recent ARP information. The ARP cache can be viewed from a UNIX or DOS (in Windows 95/98/NT) command line using the `arp -a` command.]

4.3.3.2 IP Routing: OSPF, RIP, and BGP

As an OSI Network Layer protocol, IP has the responsibility to route packets. It performs this function by looking up a packet's destination IP NET_ID in a routing table and forwarding based on the information in the table. But it is *routing protocols*, and *not* IP, that populate the routing tables with routing information. There are three routing protocols commonly associated with IP and the Internet, namely, RIP, OSPF, and BGP.

OSPF and RIP are primarily used to provide routing within a particular domain, such as within a corporate network or within an ISP's network. Since the routing is *inside* of the domain, these protocols are generically referred to as *interior gateways protocols*.

The Routing Information Protocol version 2 (RIP-2), described in [RFC 2453](#), describes how routers will exchange routing table information using a distance-vector algorithm. With RIP, neighboring routers periodically exchange their entire routing tables. RIP uses hop count as the metric of a path's cost, and a path is limited to 16 hops. Unfortunately, RIP has become increasingly inefficient on the Internet as the network continues its fast rate of growth. Current routing protocols for many of today's LANs are based upon RIP, including those associated with NetWare, AppleTalk, VINES, and DECnet. The IANA maintains a list of RIP message types.

The Open Shortest Path First (OSPF) protocol is a link state routing algorithm that is more robust than RIP, converges faster, requires less network bandwidth, and is better able to scale to larger networks. With OSPF, a router broadcasts only changes in its links' status rather than entire routing tables. OSPF Version 2, described in RFC 1583, is rapidly replacing RIP in the Internet.

The Border Gateway Protocol version 4 (BGP-4) is an *exterior gateway protocol* because it is used to provide routing information between Internet routing domains. BGP is a distance vector protocol, like RIP, but unlike almost all other distance vector protocols, BGP tables store the actual route to the destination network. BGP-4 also supports policy-based routing, which allows a network's administrator to create routing policies based on political, security, legal, or economic issues rather than technical ones. BGP-4 also supports CIDR. BGP-4 is described in RFC 1771, while RFC 1268 describes use of BGP in the Internet. In addition, the IANA maintains a list of BGP parameters.

As an alternative to using a routing protocol, the routing table can be maintained using "static routing." One example of static routing is the configuration of a default gateway at a host system; if the host needs to send an IP packet off of the local LAN segment, it is just blindly forwarded to the default gateway (router). Edge router's, too,

commonly use static routing; the single router connecting a site to an ISP, for example, will usually just have a static routing table entry indicating that all traffic leaving the local LAN be forwarded to the ISP's access router. Since there's only a single path into the ISP, a routing protocol is hardly necessary.

All IP hosts and routers maintain a table that lists the most up-to-date routing information that that device knows. On a Windows system, you can examine the routing table by issuing a route print command; on UNIX systems, use netstat -r.

Figure 4.2 shows the protocol relationship of RIP, OSPF, and BGP to IP. A RIP message is carried in a UDP datagram which, in turn, is carried in an IP packet. An OSPF message, on the other hand, is carried directly in an IP datagram. BGP messages, in a total departure, are carried in TCP segments over IP.

4.3.3.3 IP version 6

The official version of IP that has been in use since the early 1980s is *version 4*. Due to the tremendous growth of the Internet and new emerging applications, it was recognized that a new version of IP was becoming necessary. In late 1995, IP version 6 (IPv6) was entered into the Internet Standards Track. The primary description of IPv6 is contained in [RFC 1883](#) and a number of related specifications, including ICMPv6.

IPv6 is designed as an evolution from IPv4, rather than a radical change. Primary areas of change relate to:

- Increasing the IP address size to 128 bits
- Better support for traffic types with different quality-of-service objectives
- Extensions to support authentication, data integrity, and data confidentiality
- The architecture and structure of IPv6 addresses is described in [RFC 2373](#). In July 1999, the IANA delegated the initial IPv6 address space to the worldwide

regional registries in order to begin immediate worldwide deployment of IPv6 addresses.

4.3.4 The Transport Layer Protocols

The TCP/IP protocol suite comprises two protocols that correspond roughly to the OSI Transport and Session Layers; these protocols are called the Transmission Control Protocol and the User Datagram Protocol (UDP). One can argue that it is a misnomer to refer to "TCP/IP applications," as most such applications actually run over TCP or UDP, as shown in Figure 4.2.

4.3.4.1 Ports

Higher-layer applications are referred to by a port identifier in TCP/UDP messages. The port identifier and IP address together form a *socket*, and the end-to-end communication between two hosts is uniquely identified on the Internet by the four-tuple (source port, source address, destination port, destination address).

Port numbers are specified by a 16-bit number. Port numbers in the range 0-1023 are called *Well Known Ports*. These port numbers are assigned to the server side of an application and, on most systems, can only be used by processes with a high level of privilege (such as root or administrator). Port numbers in the range 1024-49151 are called *Registered Ports*, and these are numbers that have been publicly defined as a convenience for the Internet community to avoid vendor conflicts. Server or client applications can use the port numbers in this range. The remaining port numbers, in the range 49152-65535, are called *Dynamic and/or Private Ports* and can be used freely by any client or server. Some well-known port numbers include:

Table 4.3: Well-known port number

Port #	Common Protocol	Service	Port #	Common Protocol	Service
7	TCP	echo	80	TCP	http
9	TCP	discard	110	TCP	pop3
13	TCP	daytime	111	TCP	sunrpc
19	TCP	chargen	119	TCP	nntp
20	TCP	ftp-control	123	UDP	ntp
21	TCP	ftp-data	137	UDP	netbios-ns
23	TCP	telnet	138	UDP	netbios-dgm
25	TCP	smtp	139	TCP	netbios-ssn
37	UDP	time	143	TCP	imap
43	TCP	whois	161	UDP	snmp
53	TCP/UDP	dns	162	UDP	snmp-trap
67	UDP	bootps	179	TCP	bgp
68	UDP	bootpc	443	TCP	https (http/ssl)
69	UDP	tftp	520	UDP	rip
70	TCP	gopher	1080	TCP	socks
79	TCP	finger	33434	UDP	traceroute

A complete list of port numbers that have been assigned can be found in the IANA's list of Port Numbers. An implementation-specific list of supported port numbers and services can be found in the services file, generally found in the /etc (Linux/Unix), c:\windows (Windows 9x, ME), or c:\winnt\system32\drivers\etc (Windows NT, 2000) directory.

4.3.4.2 TCP

TCP, described in [RFC 793](#), provides a virtual circuit (connection-oriented) communication service across the network. TCP includes rules for formatting messages, establishing and terminating virtual circuits, sequencing, flow control, and error correction. Most of the applications in the TCP/IP suite operate over the *reliable* transport service provided by TCP.

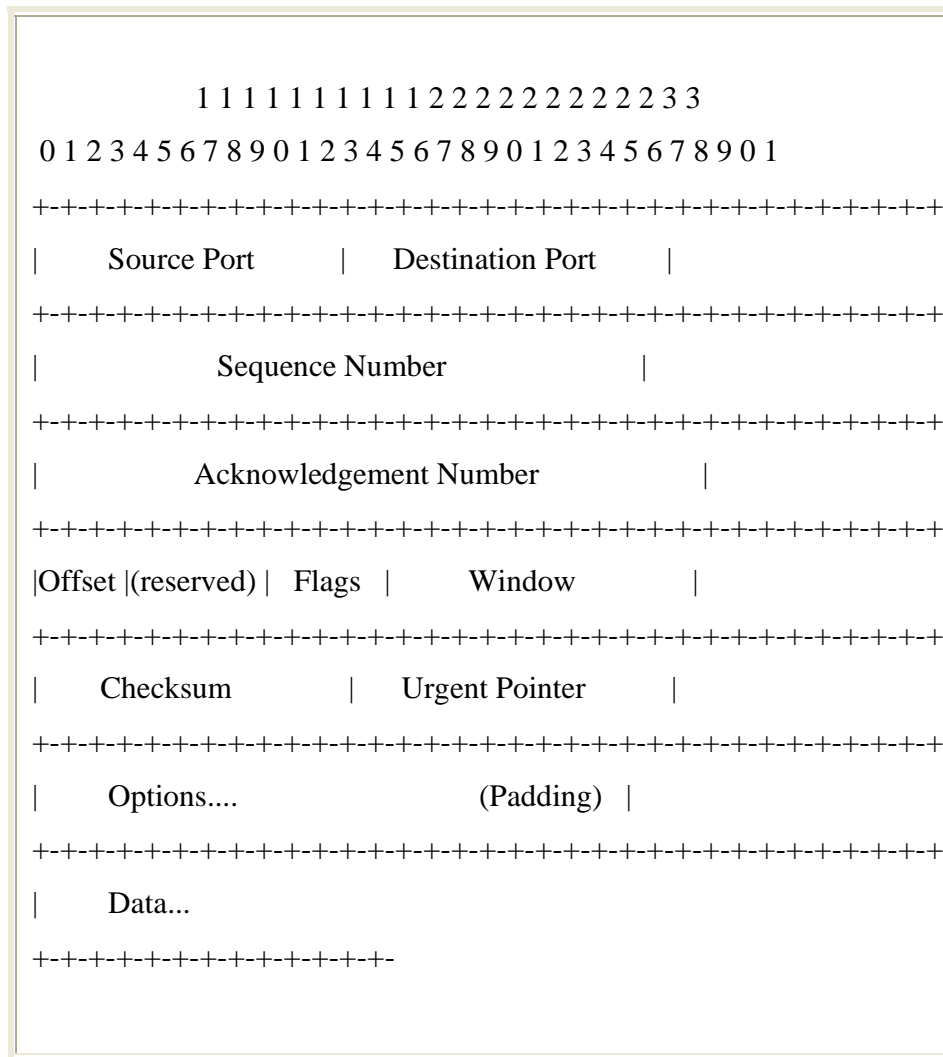


Figure 4.8: TCP segment format

The TCP data unit is called a *segment*; the name is due to the fact that TCP does not recognize messages, per se, but merely sends a block of bytes from the byte stream between sender and receiver. The fields of the segment (Figure 4.8) are:

- *Source Port and Destination Port*: Identify the source and destination ports to identify the end-to-end connection and higher-layer application.
- *Sequence Number*: Contains the sequence number of this segment's first data byte in the overall connection byte stream; since the sequence number refers to a byte count rather than a segment count, sequence numbers in contiguous TCP segments are not numbered sequentially.
- *Acknowledgment Number*: Used by the sender to acknowledge receipt of data; this field indicates the sequence number of the next byte expected from the receiver.
- *Data Offset*: Points to the first data byte in this segment; this field, then, indicates the segment header length.
- *Control Flags*: A set of flags that control certain aspects of the TCP virtual connection. The flags include:
 - *Urgent Pointer Field Significant (URG)*: When set, indicates that the current segment contains urgent (or high-priority) data and that the Urgent Pointer field value is valid.
 - *Acknowledgment Field Significant (ACK)*: When set, indicates that the value contained in the Acknowledgment Number field is valid. This bit is usually set, except during the first message during connection establishment.
 - *Push Function (PSH)*: Used when the transmitting application wants to force TCP to immediately transmit the data that is currently buffered without waiting for the buffer to fill; useful for transmitting small units of data.
 - *Reset Connection (RST)*: When set, immediately terminates the end-to-end TCP connection.
 - *Synchronize Sequence Numbers (SYN)*: Set in the initial segments used to establish a connection, indicating that the segments carry the initial sequence number.

- *Finish (FIN)*: Set to request normal termination of the TCP connection in the direction this segment is traveling; completely closing the connection requires one FIN segment in each direction.
- *Window*: Used for flow control, contains the value of the *receive window size* which is the number of transmitted bytes that the sender of this segment is willing to accept from the receiver.
- *Checksum*: Provides rudimentary bit error detection for the segment (including the header and data).
- *Urgent Pointer*: Urgent data is information that has been marked as high-priority by a higher layer application; this data, in turn, usually bypasses normal TCP buffering and is placed in a segment between the header and "normal" data. The Urgent Pointer, valid when the URG flag is set, indicates the position of the first octet of nonexpedited data in the segment.
- *Options*: Used at connection establishment to negotiate a variety of options; maximum segment size (MSS) is the most commonly used option and, if absent, defaults to an MSS of 536. Another option is Selective Acknowledgement (SACK), which allows out-of-sequence segments to be accepted by a receiver. The IANA maintains a list of all [TCP Option Numbers](#).

4.3.4.3 UDP

UDP, described in [RFC 768](#), provides an end-to-end datagram (connectionless) service. Some applications, such as those that involve a simple query and response, are better suited to the datagram service of UDP because there is no time lost to virtual circuit establishment and termination. UDP's primary function is to add a port number to the IP address to provide a socket for the application.

<pre> 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 3 3 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 </pre>
--

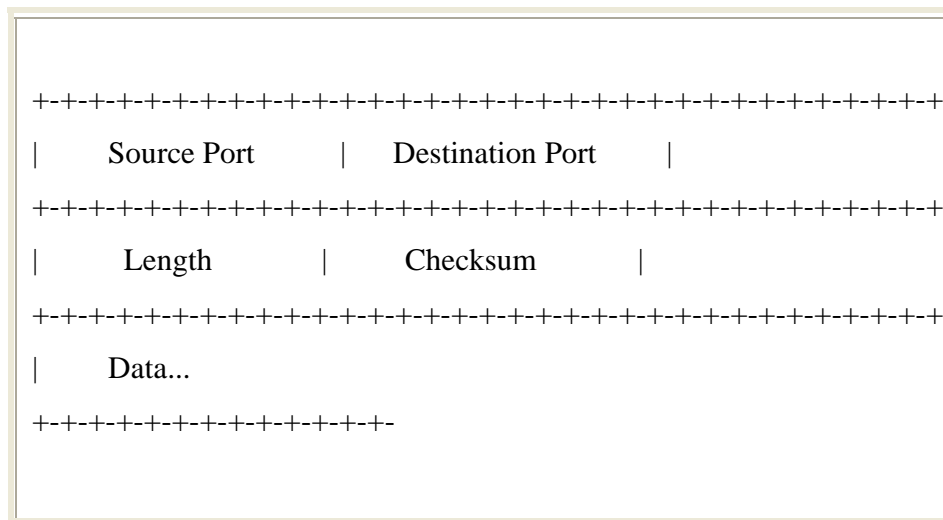


Figure 4.9: UDP datagram format

The fields of a UDP datagram (Figure 4.9) are:

- *Source Port*: Identifies the UDP port being used by the sender of the datagram; use of this field is optional in UDP and may be set to 0.
- *Destination Port*: Identifies the port used by the datagram receiver.
- *Length*: Indicates the total length of the UDP datagram.
- *Checksum*: Provides rudimentary bit error detection for the datagram (including the header and data).

4.3.4.4 ICMP

The Internet Control Message Protocol, described in [RFC 792](#), is an adjunct to IP that notifies the sender of IP datagrams about abnormal events. This collateral protocol is particularly important in the connectionless environment of IP. ICMP is not a classic host-to-host protocols like TCP or UDP, but is host-to-host in the sense that one device (e.g., a router or computer) is sending a message to another device (e.g., another router or computer).

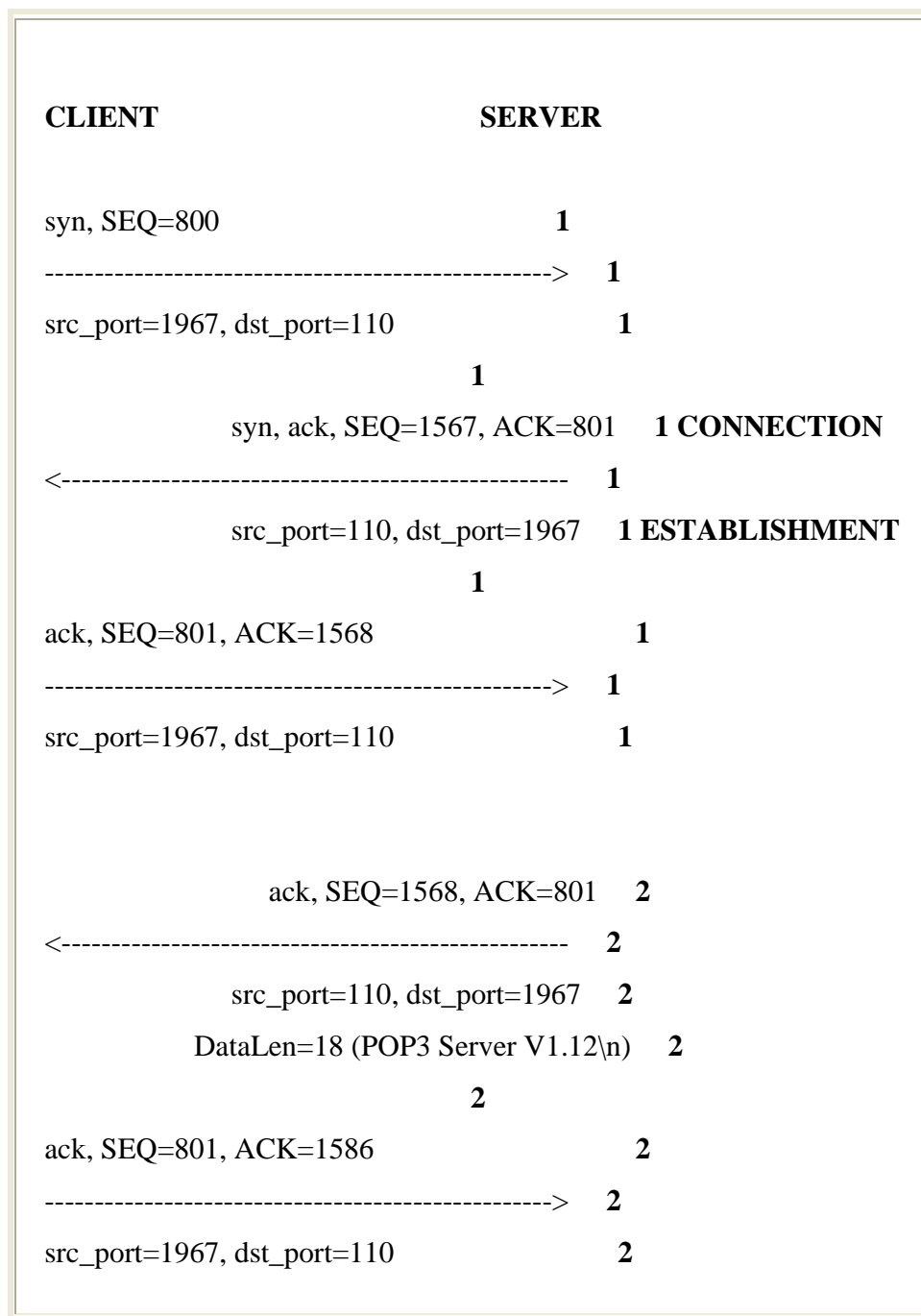
The commonly employed ICMP message types include:

- *Destination Unreachable*: Indicates that a packet cannot be delivered because the destination host cannot be reached. The reason for the non-delivery may be that the host or network is unreachable or unknown, the protocol or port is unknown or unusable, fragmentation is required but not allowed (DF-flag is set), or the network or host is unreachable for this type of service.
- *Echo and Echo Reply*: These two messages are used to check whether hosts are reachable on the network. One host sends an Echo message to the other, optionally containing some data, and the receiving host responds with an Echo Reply containing the same data. These messages are the basis for the Ping command.
- *Parameter Problem*: Indicates that a router or host encountered a problem with some aspect of the packet's Header.
- *Redirect*: Used by a host or router to let the sending host know that packets should be forwarded to another address. *For security reasons, Redirect messages should usually be blocked at the firewall.*
- *Source Quench*: Sent by a router to indicate that it is experiencing congestion (usually due to limited buffer space) and is discarding datagram's.
- *TTL Exceeded*: Indicates that a datagram has been discarded because the TTL field reached 0 or because the entire packet was not received before the fragmentation timer expired.
- *Timestamp and Timestamp Reply*: These messages are similar to the Echo messages, but place a timestamp (with millisecond granularity) in the message, yielding a measure of how long remote systems spend buffering and processing datagram's, and providing a mechanism so that hosts can synchronize their clocks.

ICMP messages are carried in IP packets. The IANA maintains a complete list of ICMP parameters.

4.3.5 TCP Logical Connections and ICMP

It is imperative to understand how a TCP connection is established to get a good feel for how TCP operates. TCP connections have three main parts: connection establishment, data exchange, and connection termination. The example below shows a POP3 server (listening on TCP port 110) being contacted by a client (using TCP port 1967).



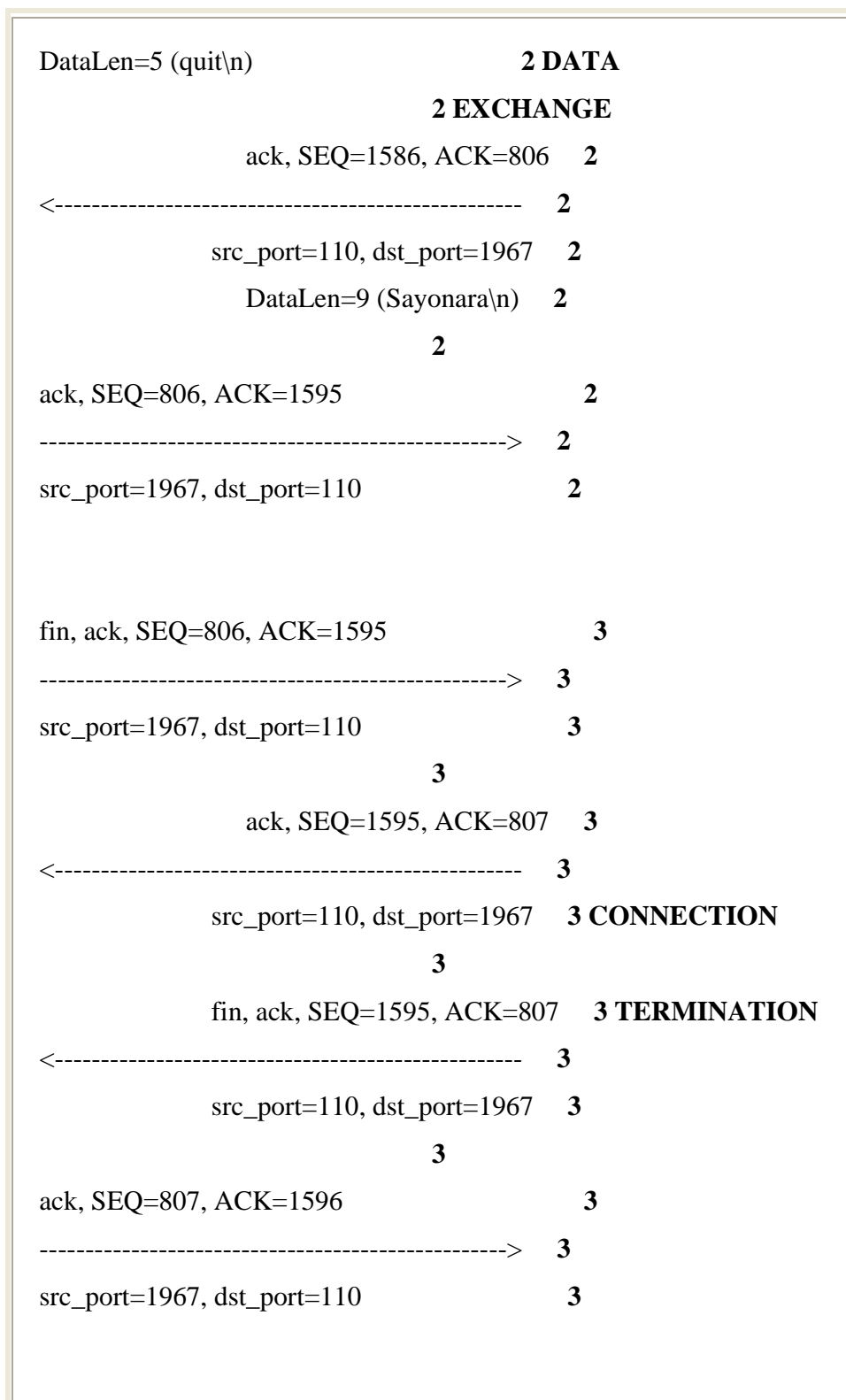


Figure 4.10: TCP logical connection phases

The *connection establishment* phase comprises a three-way handshake during which time the client and server exchange their initial sequence number (ISN) and acknowledge the other host's ISN. In this example, the client starts by sending the server a TCP segment with the syn-bit set and a Sequence Number of 800. The syn-bit tells the receiver (i.e., the server) that the sender (i.e., the client) is in "ISN initialization" mode and that the ISN hasn't yet been confirmed. The segment's Acknowledgement Number isn't shown because its value is, at this point, invalid.

The server responds with a segment with the syn- and ack-bits set, a Sequence Number of 1567, and an Acknowledgement Number of 801. The syn-bit and ISN of 1567 have the same meaning as above. The ack-bit indicates the value of the Acknowledgement Number field is valid and the ACK value of 801 is the way in which the server confirms the client's ISN.

The final part of the three-way handshake is when the client sends a segment with just the ack-bit set. Note that the Acknowledgement Number field (1568) is one greater than the server's ISN.

This three-way handshake is sometimes referred to as an exchange of "syn, syn/ack, and ack" segments. It is important for a number of reasons. For individuals looking at packet traces, recognition of the three-way handshake is how to find the start of a connection. For firewalls, proxy servers, intrusion detectors, and other systems, it provides a way of knowing the direction of a TCP connection setup since rules may differ for outbound and inbound connections.

The second part of the TCP connection is *data exchange*. The information here is more or less made up for example purposes only; it shows a POP server sending a banner message to the client system, the user sending the "quit" command, and the server signing off. (Note that the "\n" indicates an "end-of-line" indicator.) These segments show the changing of, and relationship between, the client's and server's sequence and acknowledgement numbers.

The final phase is *connection termination*. Although TCP connections are full-duplex (even if a given application does not allow two-way simultaneous communication), the TCP protocol views the logical connection as a pair of simplex links. Therefore, connection termination requires four segments or, more properly, two pair of segments. In this case, the client sends the server a segment with the fin- and ack-bits set; the server responds with a segment with just the ack-bit set and the Acknowledgment Number is incremented. The server then sends a fin/ack segment to the client.

The paragraphs above describe a normal scenario setting up a TCP connection between a client and server. Two UDP hosts communicate in a similar fashion; one host sends a UDP datagram to the other which is presumably listening on the port indicated in the datagram.

But what happens if a host isn't listening on a port to which a connection is attempted or the host doesn't actually exist? Here's what happens in these "abnormal" conditions:

- *Host not listening on TCP port:* If Host A attempts to contact Host B on a TCP port that Host B is not listening on, Host B responds with a TCP segment with the reset (RST) and acknowledge (ACK) flags set.
- *Host not listening on UDP port:* If Host A attempts to contact Host B on a UDP port that Host B is not listening on, Host B sends an ICMP *port unreachable* message to Host A.
- *Host does not exist:* If Host A attempts to contact Host B and Host B is not listening (e.g., Host B's IP address either doesn't exist or is unavailable), Host B's subnet's router will send an ICMP *host unreachable* message to Host A.

4.3.6 The TCP/IP Application Layer

The TCP/IP Application Layer protocols support the applications and utilities that are the Internet. This section will list a number of these applications and show a sample packet decodes of all protocol layers.

4.3.6.1 TCP and UDP Applications

Commonly used protocols (as shown in Figure 4.2) include:

- *Archie*: A utility that allows a user to search all registered anonymous FTP sites for files on a specified topic. Largely obsolete today, obviated by the World Wide Web.
- *BGP*: The Border Gateway Protocol version 4 (BGP-4) is a distance vector exterior gateway routing protocol, commonly used between two ISPs or between a customer site and ISP if there are multiple links.
- *DNS*: The Domain Name System (described in slightly more detail in Section 3.2.2 above) defines the structure of Internet names and their association with IP addresses, as well as the association of mail and name servers with domains.
- *Finger*: Used to determine the status of other hosts and/or users ([RFC 1288](#)).
- *FTP*: The File Transfer Protocol allows a user to transfer files between local and remote host computers ([RFC 959](#)).
- *Gopher*: A tool that allows users to search through data repositories using a menu-driven, hierarchical interface, with links to other sites. Largely obsolete today, obviated by the World Wide Web ([RFC 1436](#)).
- *HTTP*: The Hypertext Transfer Protocol is the basis for exchange of information over the World Wide Web (WWW). Various versions of HTTP are in use over the Internet, with HTTP version 1.0 ([RFC 1945](#)) being the most current. WWW pages are written in the Hypertext Markup Language (HTML), an ASCII-based, platform-independent formatting language ([RFC 1866](#)).

- *IMAP*: The Internet Mail Access Protocol defines an alternative to POP as the interface between a user's mail client software and an e-mail server, used to download mail from the server to the client and providing significant flexibility in mailbox management.
- *OSPF*: The Open Shortest Path First version 2 (OSPFv2) protocol is a link state routing protocol used within an organization's network. This is the preferred so-called *interior gateway protocol*.
- *Ping*: A utility that allows a user at one system to determine the status of other hosts and the latency in getting a message to that host. Uses ICMP Echo messages. For more information and insight, see [The Ping Page](#).
- *POP*: The Post Office Protocol defines a simple interface between a user's mail client software (e.g., Eudora, Outlook, or the e-mail capability of your browser) and an e-mail server, used to download mail from the server to the client and allows the user to manage their mailboxes. The current version is POP3 ([RFC 1460](#)).
- *RADIUS*: The Remote Authentication Dial-In User Service (RADIUS) is a remote-access protocol.
- *RIP*: The Routing Information Protocol (RIP) is a distance-vector routing protocol used within an organization's network.
- *SSH*: The Secure Shell is a protocol that allows remote logon to a host across the Internet, much like Telnet. Unlike Telnet, however, SSH encrypts passwords and data traffic.
- *SMTP*: The Simple Mail Transfer Protocol is the standard protocol for the exchange of electronic mail over the Internet ([RFC 821](#)). SMTP is used between e-mail servers on the Internet or to allow an e-mail client to send mail to a server. [RFC 822](#) specifically describes the mail message body format, and RFCs [1521](#) and [1522](#) describe MIME (Multipurpose Internet Mail Extensions). Reference books on electronic mail systems include *!%@:: Addressing and Networks* by D. Frey and R. Adams (O'Reilly & Associates, 1993) and *THE INTERNET MESSAGE: Closing the Book With Electronic Mail* by M. Rose (PTR Prentice Hall, 1993).

- *SNMP*: The Simple Network Management Protocol defines procedures and management information databases for managing TCP/IP-based network devices. SNMP ([RFC 1157](#)) is widely deployed in local and wide area networks. SNMP Version 2 (SNMPv2, [RFC 1441](#)) adds security mechanisms that are missing in SNMP, but is also very complex; widespread use of SNMPv2 has yet to be seen. Additional information on SNMP and TCP/IP-based network management can be found in *SNMP* by S. Feit (McGraw-Hill, 1994) and *THE SIMPLE BOOK: An Introduction to Internet Management, 2/e*, by M. Rose (PTR Prentice Hall, 1994).
- *SSL*: The Secure Sockets Layer (SSL), designed by Netscape, provides a mechanism for secure communications over the Internet, based on certificates and public key cryptography. The most commonly known SSL application is HTTP over SSL, commonly designated as *https*. The newest version of SSL is called Transport Layer Security (TLS) ([RFC 2246](#)). SSL is *not*, however, HTTP-specific; protocols such as IMAP4 (imaps), FTP (ftps), Telnet (telnets), and POP3 (pop3s) all have definitions for operation over SSL.
- *TACACS+*: The Terminal Access Controller Access Control System plus is a remote access protocol.
- *Telnet*: Short for *Telecommunication Network*, a virtual terminal protocol allowing a user logged on to one TCP/IP host to access other hosts on the network ([RFC 854](#)).
- *TFTP*: The Trivial File Transfer Protocol (TFTP) is used for some specialized simple file transfer applications.
- *Time/NTP*: Time and the Network Time Protocol (NTP) are used so that Internet hosts can synchronize their system time from well-known Internet time servers.
- *Traceroute*: A tool that displays the route taken by packets across the Internet between a local and remote host. The *traceroute* command is available on Linux/Unix systems; Windows systems starting with Windows 95 have a *tracert* command utility.
- *Whois/NICNAME*: Utilities that search databases for information about Internet domains and domain contact information ([RFC 3912](#)).

4.4 Summary

As this discussion has shown, *TCP/IP* is not merely a pair of communication protocols but is a suite of protocols, applications, and utilities. Increasingly, these protocols are referred to as the *Internet Protocol Suite*, but the older name will not disappear anytime soon.

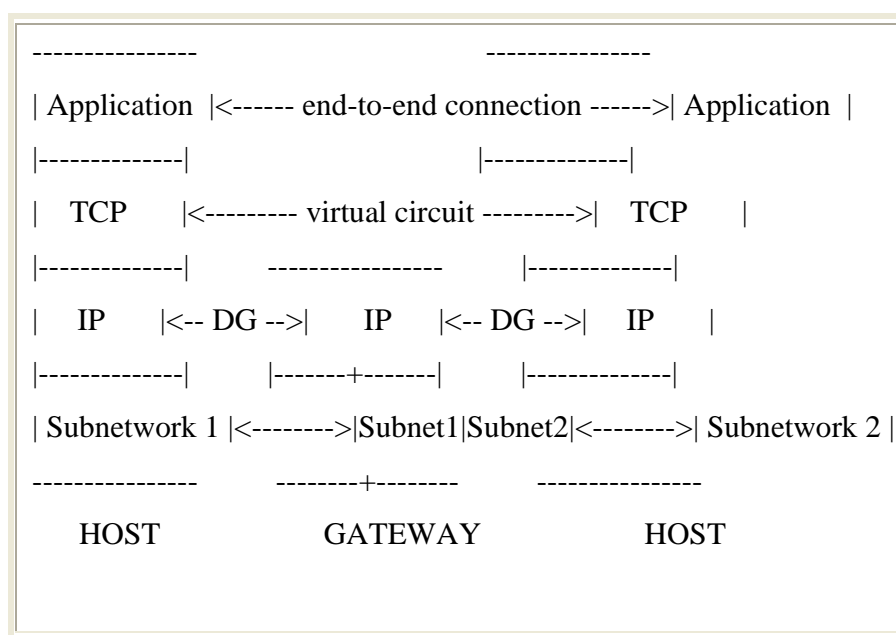


Figure 4.11: TCP/IP protocol suite architecture.

Figure 4.11 shows the relationship between the various protocol layers of TCP/IP. Applications and utilities reside in host, or end-communicating, systems. TCP provides a reliable, virtual circuit connection between the two hosts. (UDP, not shown, provides an end-to-end datagram connection at this layer.) IP provides a datagram (DG) transport service over any intervening subnetworks, including local and wide area networks. The underlying subnetwork may employ nearly any common local or wide area network technology. Note that the term *gateway* is used for the device interconnecting the two subnets, a device usually called a *router* in LAN environments or *intermediate system* in OSI environments.

CHAPTER V

FILED TEST

5.1 Introduction

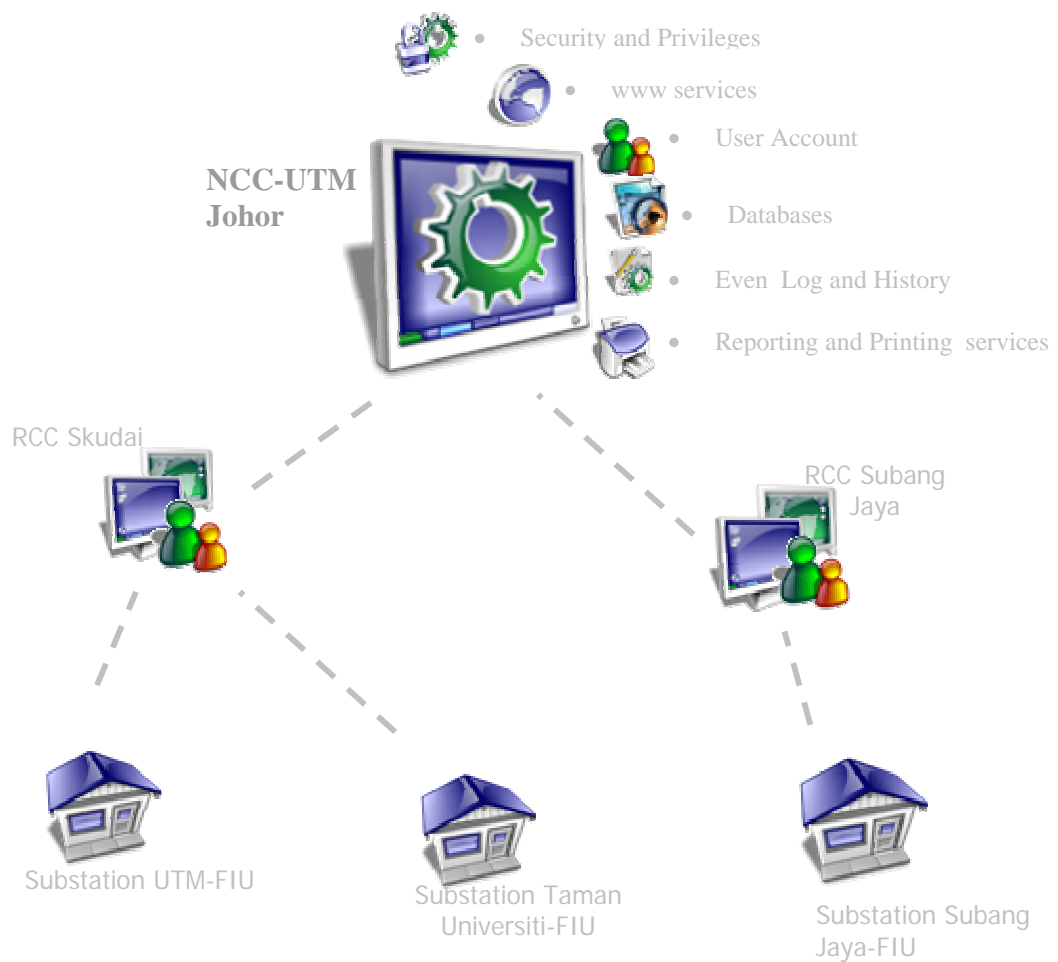


Figure 5.1: Pilot plan implementation

The above illustration describes the architecture of the pilot test plan. There are 3 substation will be monitored and each substation will monitor 4 feeder unit. Substation Taman Universiti and Substation UTM will be registered under RCC UTM while substation subang jaya will be monitored by RCC Subang Jaya. RCC Subang Jaya and RCC UTM will be connected to a centralized server system in UTM.

5.2 Communication

The communication between the substation (FIU) and RCC will be established using GSM modem utilizing Short Message Service (SMS) functionality. This mean, both FIU and RCC need to have the GSM modem and sms decoding function. FIU will monitor the feeder unit using Feeder Controller Unit (FCU).

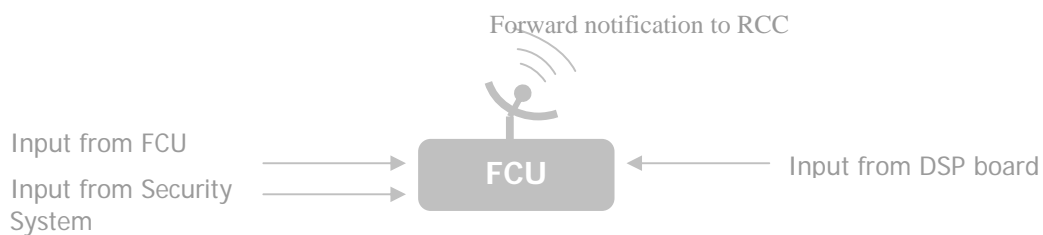


Figure 5.2: FIU layout design

FCU has the capability of monitoring multiple inputs at the same time and respond to RCC by sending the current situation at the substation. FCU will notify RCC the current status of the substation when it receives a status query call from the gsm modem. This feature is important when the RCC need to confirm the status which has been reported earlier.

RCC consist of software on windows platform PC running a monitoring application routine over the registered substation. It is designed with the latest interface building standard scheme according to Microsoft Developers Guidelines. It guarantees

the most beautiful user interface and improves handling capability. It is designed using Microsoft visual C++.NET environment utilizing various function and capability.

RCC handles communication between the substation and server. It receives notification from substation and does some necessary decoding and submits the notification to the server.

5.3 GSM Modem.

Interfacing of RCC and GSM modem is the most difficult part when designing RCC. The design of communication protocol need to consider a lot of circumstances as the gsm modem may be interfered with gsm networks message controlled by the service provide or operator. Furthermore, the RCC need to have 100% control of the GSM modem or the RCC system will crash if the gsm modem starts to behave strangely. All the activities occur in the GSM modem need to be monitored and controlled to prevent data loss and false alarm problem due to improper control of the GSM modem. RCC has the capability of the following details in handling GSM modem.

- Read, write, send, delete and decode sms in the sim card memory.
- Make and receive Voice calls.
- Display manufacturer and model information.
- Display available and registered networks.
- Display signal strength.
- Read sim card buffer/memory status.
- GSM modem connectivity status.

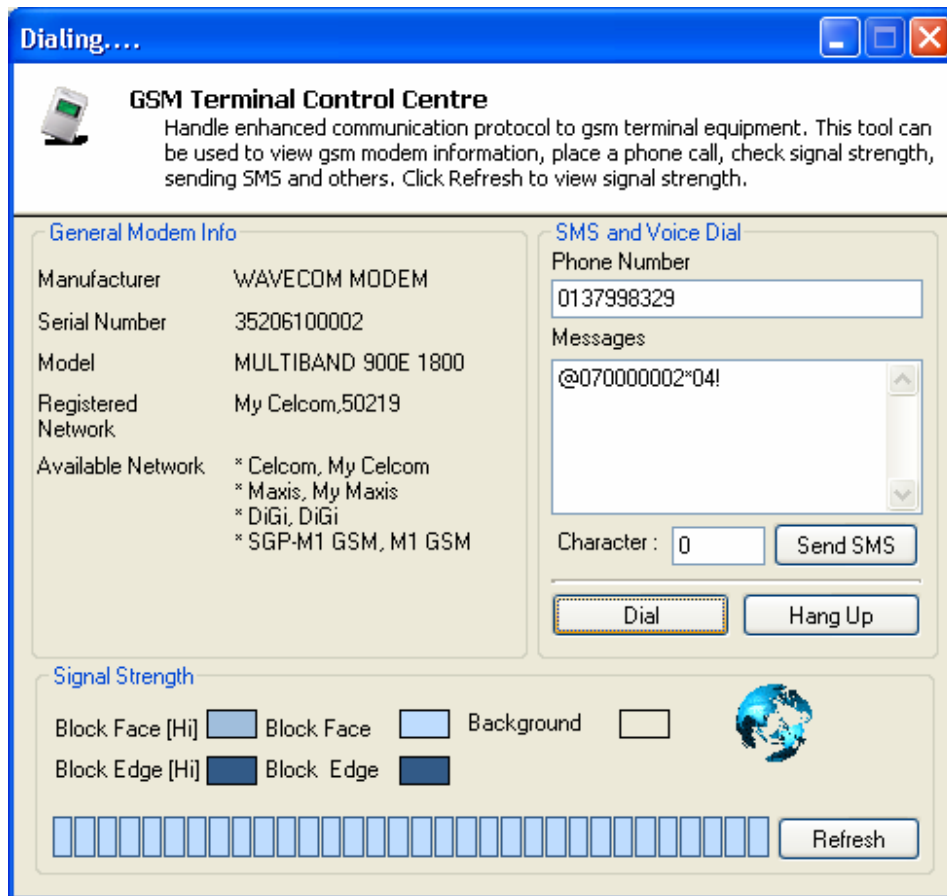


Figure 5.3: GSM modem control panel

5.4 Client/Server Connectivity

RCC send information to server using instant messaging system utilizing TCP/IP port called window socket. A socket is a data structure maintained by the system to handle network connections. Socket is created for moving a stream of binary data across machines with different platforms. Both client and server can run with any platforms, such as Window, UNIX, Macintosh, Linux, and others. This messaging system is very essential and popular in developing client and server system.

In order to establish connection, it requires a TCP/IP server and TCP/IP client. A TCP/IP server socket can be referred as to a socket that can accept many connections.

And a TCP/IP client socket is a socket that is connected to server socket. Once the server socket is turned on, the client can start to connect to the server by providing an appropriate server IP and port number. When communication established, data can be exchanged.

RCC reports the faults notification through TCP/IP socket communication. Once it gets connected to the server, it will send the list of currently connected user to the server. If any clients disconnected from the server, the server will inform everyone by sending updated information about the list of connected users. In this case, user refers to RCC.

RCC is designed to view only the information related to its own region. There is no possible access to the information in the other region. RCC only allowed accessing the registered substation in the region and conducting necessary operation such as delete, updating information and registering new substation. Once the RCC gets a fault notification, it will check whether the fault belongs to the registered substation or not. If yes, the notification will be processed and if not, the notification will be discarded and ignored. The fault notification can be sent to server instantly or by scheduled submission interval.

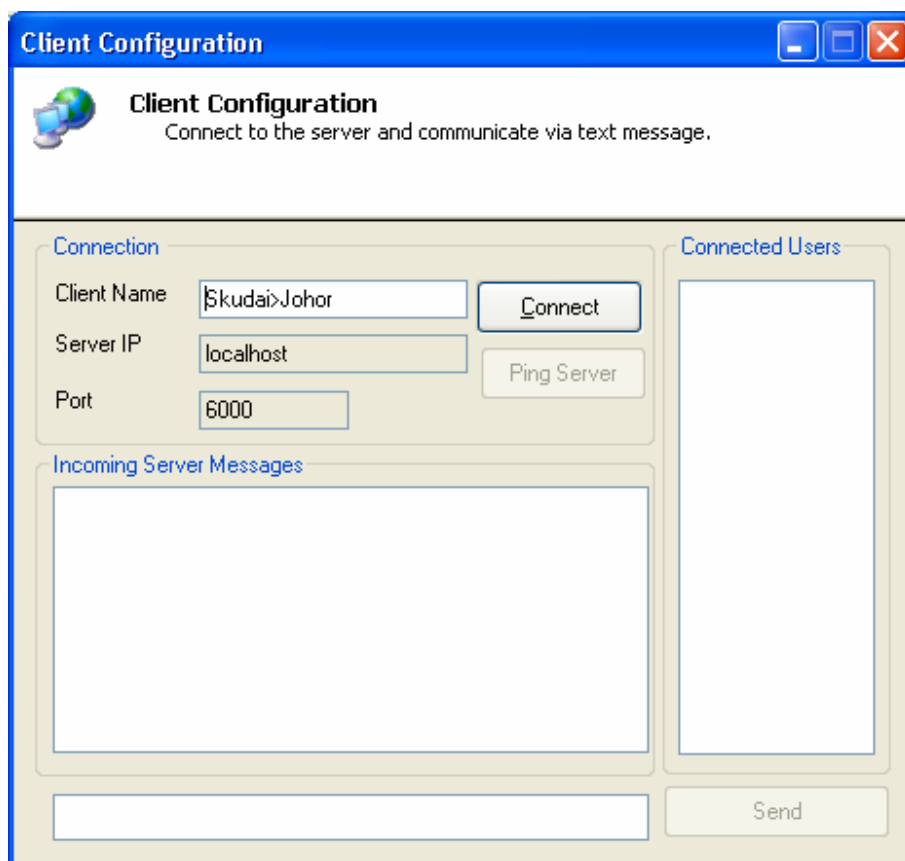


Figure 5.4: Client configuration utility

5.5 Server System/NCC.

NCC is a centre monitoring and control station. It can view all the connected RCC and perform operation such delete and enroll RCC. NCC can perform an intensive data analysis compared to RCC.

NCC starts by starting its TCP/IP socket server application and FTP server application. The TCP/IP server socket is used to exchange data between server and the connected clients while the ftp server is used as a file transfer utility.

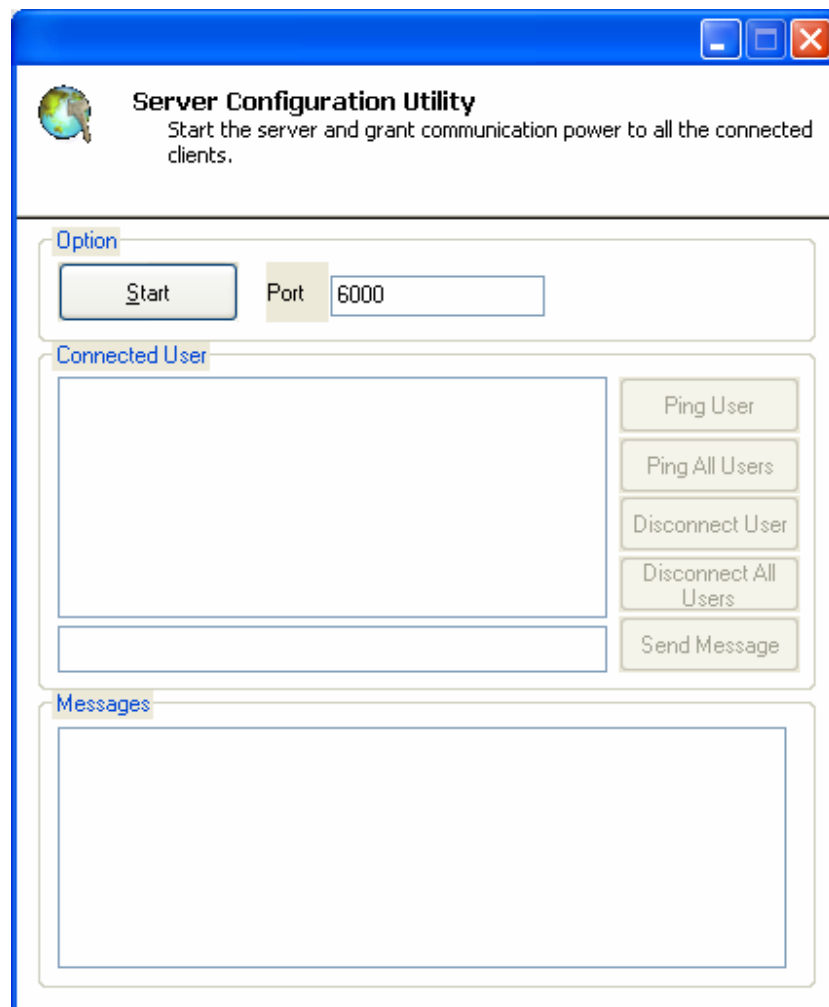


Figure 5.5: Server socket starting interface

A socket server needs to be able to listen on a specific port, accept connections and read and write data from the socket. A high performance and scalable socket server should use asynchronous socket IO and IO completion ports. Before we can start accepting connections we need to have a socket to listen on. Once the server started, it listens to the socket waiting for any connection by the client. Once a user request to connect, it will accept and store information regarding to the client. Then the server will return to its previous state which is listening to the socket.



Figure 5.6: Database and NCC software running on Dell Server

CHAPTER VI

CONCLUSION

The management of streetlights by the power utility company and local authorities are typically faced with the problem of high operational expenditure, low efficiency and increase customer complaint. They are also faced with increase customer complaint due to unattended faulty streets lights and frequent power outages. There is a significant pressure to reduce these operational expenses, improve efficiency and image. By operational efficiency we meant how faulty street lights are managed effectively through the use of a low cost automated system, thus improve efficiency and enhancing customer services.

Operational cost reduction is achieved through accurately identification of faulty lights and timely action taken to rectify such fault. Currently these maintenance routines [i.e. random patrols around the street light zones] are conducted daily in parallel to records of faulty lights reported by customers. These incurred substantially high operational expenditure year to year. The objective of this project is to develop a low cost SLM system with features suffice enough for the utility companies to effectively manage and maintain street lights and also monitor power quality to ensure continuous and uninterrupted supply to customers both residential and industries.

SLMS consist of interface modules (FC-Feeder Controller, FIU-Feeder Interface Units) installed at the substation or street lights panel to collect the status of power over each feeder pillar. Information of faulty lights collected by the FIU (which will measure the current/power on the feeder and record any changes e.g. a drop in power indicating a faulty light in that feeder line) is passed back to FC using the PLC (Power line carrier)

technique. The FC manages several feeder pillars and relay back the information received from FIU to a management system server located at the office (RCC-Regional Control Center & NCC-National Control Center) using “SMS” GSM network. RCC then sends an alert SMS to operational personnel to inform them of the faulty record. The management system keeps records for management reporting and analysis.

APPENDIX A

FCU FIRMWARE SOURCE CODE

```

;assembly language for FCU.
;fcu0 for motherboard micon AA BB CC DD EE 1 sec feedback
;terima x atau X baru send status
;hantar DUA aksara sahaja (sama gsm modem)final selepas delay pada
BEGIN
;serial17 trial for three phase (wiring on PCB must change!)
;hantar double data dan ada pull up pada a4(final)
;tambah delay selepas dapat x
include      16f84.h

.osc  hs
.wdt  off
.pwrt off
.protect off

btime equ    33          ; 9600bps @4.0MHz33
;btime      equ    83          ; 9600bps @10.0MHz

txd    equ    rb.6        ;ra.0 ok
rxd    equ    rb.7        ;ra.1
ra0    equ    ra.0
ra1    equ    ra.1
ra2    equ    ra.2
ra3    equ    ra.3
ra4    equ    ra.4
ra5    equ    ra.5

        org    0ch
ch      ds    1
rs      ds    1
rs1     ds    1
cn      ds    1
tm1     ds    1
tm2     ds    1
tm3     ds    1
tmoff   ds    1
tmleak  ds    1
tmfault ds    1
tmok    ds    1
tmover  ds    1
buffer1 ds    1
ch1     ds    1

        org    0
        goto  start
        org    4

```

```

start mov    !ra,#11111b
      mov    !rb,#10111111b

      clr    tm1
      clr    tm2
      clr    w
      clr    buffer1
;*****mula*****
mula  btfss  rxd
      goto  getgsm
      btfss  ra0
      goto  off
      btfss  ra1
      goto  leak
      btfss  ra3
      goto  over
      btfss  ra2
      goto  ok
      cje    ra,#11111b,fault
      goto  mula
;*****getgsm*****
getgsm call  terima
      mov    buffer1,w
      cje    buffer1,'#x',test ;receive data interupt X
      cje    buffer1,'#X',test ;receive data interupt x
      goto  mula
;*****test*****
test  call  wait5s          ;feedback 1s

      clr    tmoff
      clr    tmleak
      clr    tmfault
      clr    tmok
      clr    tmover
      cje    ra,#11100b,off
      cje    ra,#11101b,leak
      cje    ra,#11111b,fault
      cje    ra,#11011b,ok
      cje    ra,#10011b,over
      goto  test

;*****off*****
off   clr    tm1
      clr    tm2
      clr    tm3
off1  add    tm3,#1
off2  add    tm2,#1
off3  add    tm1,#1

      cje    ra,#11101b,leak
      cje    ra,#11111b,fault
      cje    ra,#11011b,ok
      cje    ra,#10011b,over

      cjne  tm1,#255,off3
      cjne  tm2,#255,off2
      cjne  tm3,#2,off1

```

```

off5  cje    tmoﬀ,#1,stop1
      mov    ch,#'A'
      call   trans

      call   wait1s
      call   wait1s
      call   wait1s
      call   wait1s

      mov    ch,#'A'
      call   trans
      mov    ch,#00001101b
      call   trans

      mov    tmoﬀ,#1
      clr    tmleak
      clr    tmfault
      clr    tmok
      clr    tmover

stop1  btfss  rxd
      goto   getgsm
      cje    ra,#11101b,leak
      cje    ra,#11111b,fault
      cje    ra,#11011b,ok
      cje    ra,#10011b,over
      goto   stop1
;*****leak*****
leak  clr    tm1
      clr    tm2
      clr    tm3
leak1 add    tm3,#1
leak2 add    tm2,#1
leak3 add    tm1,#1

      cje    ra,#11100b,off
      cje    ra,#11111b,fault
      cje    ra,#11011b,ok
      cje    ra,#10011b,over

      cjne   tm1,#255,leak3
      cjne   tm2,#255,leak2
      cjne   tm3,#2,leak1

leak5  cje    tmleak,#1,stop2
      mov    ch,#'B'
      call   trans

      call   wait1s
      call   wait1s
      call   wait1s
      call   wait1s

      mov    ch,#'B'
      call   trans
      mov    ch,#00001101b

```

```

        call    trans

        mov     tmleak,#1
        clr     tmoff
        clr     tmfault
        clr     tmok
        clr     tmover

stop2  btfss   rxd
        goto   getgsm
        cje    ra,#11100b,off
        cje    ra,#11111b,fault
        cje    ra,#11011b,ok
        cje    ra,#10011b,over
        goto   stop2
;*****fault*****
fault  clr     tm1
        clr     tm2
        clr     tm3
fault1  add     tm3,#1
fault2  add     tm2,#1
fault3  add     tm1,#1

        cje    ra,#11100b,off
        cje    ra,#11101b,leak
        cje    ra,#11011b,ok
        cje    ra,#10011b,over

        cjne   tm1,#255,fault3
        cjne   tm2,#255,fault2
        cjne   tm3,#2,fault1

        cje    tmfault,#1,stop3
fault5  mov     ch,#'C'
        call   trans

        call   wait1s
        call   wait1s
        call   wait1s
        call   wait1s

        mov    ch,#'C'
        call   trans
        mov    ch,#00001101b
        call   trans

        mov    tmfault,#1
        clr    tmoff
        clr    tmleak
        clr    tmok
        clr    tmover

stop3  btfss   rxd
        goto   getgsm
        cje    ra,#11100b,off
        cje    ra,#11101b,leak
        cje    ra,#11011b,ok

```

```

        cje    ra,#10011b,over
        goto  stop3

;*****ok*****
ok      clr    tm1
        clr    tm2
        clr    tm3
ok1     add    tm3,#1
ok2     add    tm2,#1
ok3     add    tm1,#1

        cje    ra,#11100b,off
        cje    ra,#11101b,leak
        cje    ra,#11111b,fault
        cje    ra,#10011b,over

        cjne   tm1,#255,ok3
        cjne   tm2,#255,ok2
        cjne   tm3,#2,ok1

ok5     cje    tmok,#1,stop4
        mov    ch,'#D'
        call   trans

        call   wait1s
        call   wait1s
        call   wait1s
        call   wait1s

        mov    ch,'#D'
        call   trans
        mov    ch,#00001101b
        call   trans

        mov    tmok,#1
        clr    tmoff
        clr    tmleak
        clr    tmfault
        clr    tmover

stop4   btfss  rxd
        goto  getgsm
        cje    ra,#11100b,off
        cje    ra,#11101b,leak
        cje    ra,#11111b,fault
        cje    ra,#10011b,over
        goto  stop4

;*****over*****
over    clr    tm1
        clr    tm2
        clr    tm3

over1   add    tm3,#1
over2   add    tm2,#1
over3   add    tm1,#1

```



```

    cje    ra,#11100b,off
    cje    ra,#11101b,leak
    cje    ra,#11111b,fault
    cje    ra,#11011b,ok

    cjne   tm1,#255,over3
    cjne   tm2,#255,over2
    cjne   tm3,#2,over1

over5    cje    tmover,#1,stop5
        mov    ch,'#E'
        call   trans

        call   wait1s
        call   wait1s
        call   wait1s
        call   wait1s

        mov    ch,'#E'
        call   trans
        mov    ch,#00001101b
        call   trans

        mov    tmover,#1
        clr    tmoff
        clr    tmleak
        clr    tmok
        clr    tmfault

stop5    btfss  rxd
        goto   getgsm
        cje    ra,#11100b,off
        cje    ra,#11101b,leak
        cje    ra,#11111b,fault
        cje    ra,#11011b,ok
        goto   stop5

;*****transmit*****
trans    bcf    txd            ;txd rb6
        mov    rs1,#20          ;300.55hz #20
        mov    rs,#255         ;segment rs, #255
trans10  djnz  rs,trans10      ;delay rasanya
ulang0   djnz  rs1,ulang0
        mov    cn,#8           ;segment cn, 8
        nop                    ;nop sekejap
trans0   rr     ch            ;rotate right segment ch
        nop                    ;nop sekejap
        movb   txd,c          ;
        mov    rs1,#20         ;#20
        mov    rs,#255        ;#255
trans11  djnz  rs,trans11
ulang1   djnz  rs1,ulang1
        djnz  cn,trans0
        nop
        nop
        nop
        nop

```

```

        nop
        bsf    txd
        mov    rs1,#20          ;#20
        mov    rs,#255         ;#255
trans12    djnz  rs,trans12
ulang2    djnz  rs1,ulang2
        ret
;*****
;*****receive*****
receive    btfsc rxd          ;test kalau rxd tak sama dengan bit mula
        goto  receive        ;if tak jumpa 1 ulang detect 1 "receive"
terima     mov    rs1,#10      ;300.55hz
        mov    rs,#200        ;segment rs, 9600Mhz
recv10     djnz  rs,recv10     ;delay rasanya, memang pun
recv20     djnz  rs1,recv20    ;if tolak rs dgn 1 = 0 goto recv10
        mov    cn,#8          ;segment cn = 8 bit
        nop
recv50     mov    rs1,#20
        mov    rs,#255
recv31     djnz  rs,recv31
recv13     djnz  rs1,recv13
        nop
        movb   c,rxid         ;bit c = bit rxd ; carry/borrow flag
        rr     ch1            ;rotate right sekali pd segment ch
        mov    w,ch1          ;ch1 dan w register only!
        djnz  cn,recv50      ;if tolak cn dgn 1 = 0 goto recv0
        ret
;*****
wa100m     clr    tm1          ;delay 100 milisecond
        clr    tm2
        clr    tm3
wait0      add    tm1,#1
wait1      add    tm2,#1
wait2      add    tm3,#1
        cjne   tm3,#239,wait2  ;239
        cjne   tm2,#28,wait1   ;28
        cjne   tm1,#1,wait0    ;1
        nop                                ;nop 1x
        ret

wa500m     clr    tm1          ;delay 100 milisecond
        clr    tm2
        clr    tm3
wait6      add    tm1,#1
wait7      add    tm2,#1
wait8      add    tm3,#1
        cjne   tm3,#255,wait8   ;239 tak betul lagi ni!
        cjne   tm2,#60,wait7   ;28
        cjne   tm1,#1,wait6    ;1
        nop                                ;nop 1x
        ret

wait1s     clr    tm1          ;delay 1 second
        clr    tm2
        clr    tm3
wait3      add    tm1,#1
wait4      add    tm2,#1

```

```
wait5 add    tm3,#1
      cjne   tm3,#105,wait5      ;105
      cjne   tm2,#24,wait4       ;24
      cjne   tm1,#2,wait3        ;2
      ret

wait5s    clr    tm1              ;delay 1 second
      clr    tm2
      clr    tm3
wait9 add   tm1,#1
wait10 add  tm2,#1
wait11 add  tm3,#1
      cjne   tm3,#255,wait11     ;105
      cjne   tm2,#100,wait10    ;24
      cjne   tm1,#8,wait9       ;2
      ret
```

APPENDIX B

FIU FIRMWARE SOURCE CODE

```

;assembly language for FCU.

        include      16f84.h

        .osc   hs
        .wdt   off
        .pwrt  off
        .protect off

btime equ    33                ; 9600bps @4.0MHz33

txd1 equ    rb.6                ;ra.0 ok
rxdl equ    rb.7                ;ra.1 ok
txd2 equ    rb.4                ;ra.0 ok
rxdl equ    rb.5                ;ra.1 ok
sag equ     rb.0
swell equ   rb.1
power equ   rb.2
motion equ  rb.3

        org    0ch
cha ds     1
chb ds     1
chl ds     1
ch2 ds     1
ch3 ds     1
cn ds     1
tm0 ds     1
tm1 ds     1
tm2 ds     1
tm3 ds     1
tmoff ds   1
tmleak ds  1
tmfault ds 1
tmok ds    1
tmover ds  1
buffer1 ds 1
buffer2 ds 1
buffer3 ds 1
buffer4 ds 1
count ds   1
count0 ds  1
count1 ds  1
count2 ds  1
count3 ds  1
count4 ds  1
lambat ds  1

        org    0
        goto   start

```

```

        org     4

start  mov     !ra,#0000b
       mov     !rb,#10101111b
;*****initial running
led*****
       mov     ra,#1110b    ;1 led padam
       call    wa500m
       mov     ra,#1101b
       call    wa500m
       mov     ra,#1011b
       call    wa500m
       mov     ra,#0111b
       call    wa500m
;*****initial send
data*****
       clr     lambat
       clr     count0
       clr     count1
       clr     count2
       clr     count3

delay0      add     count0,#1          ;delay untuk gsm initialize

delay1      add     count1,#1
delay2      add     count2,#1
           cjne    count2,#255,delay2      ;255
           cjne    count1,#255,delay1      ;255
           cjne    count0,#100,delay0      ;100

;*****set text mode pada
gsm*****
;text mov     ra,#1011b
;       mov     cha,#'A'          ;set text mode pada gsm modem
;       call    transa
;       mov     cha,#'T'
;       call    transa
;       mov     cha,#'+'
;       call    transa
;       mov     cha,#'C'
;       call    transa
;       mov     cha,#'M'
;       call    transa
;       mov     cha,#'G'
;       call    transa
;       mov     cha,#'F'
;       call    transa
;       mov     cha,#'='
;       call    transa
;       mov     cha,#'l'
;       call    transa          ;finish set text mode

;       mov     cha,#00001101b    ;send enter
;       call    transa
;       call    wait1s

```

```

;*****set
rts/dtr*****
rtsdtr    mov    cha,#'A'           ;set text mode pada gsm modem
          call   transa
          mov    cha,#'T'
          call   transa
          mov    cha,#'+'
          call   transa
          mov    cha,#'I'
          call   transa
          mov    cha,#'F'
          call   transa
          mov    cha,#'C'
          call   transa
          mov    cha,#'='
          call   transa
          mov    cha,#'0'
          call   transa           ;finish set rts/dtr

          mov    cha,#00001101b    ;send enter
          call   transa
          call   wait1s
          call   wait1s
          call   wait1s
          call   wait1s

;*****send start on
msg*****
poweron   mov    ch2,#'9'
          mov    ch3,#'9'
          goto   sms

;*****mula*****
*
mula      mov    ra,#1110b         ;1 led padam
          cje    rb,#11011111b,getfcu
          cje    rb,#01111111b,getgsm

          cje    rb,#11111110b,sag0    ;detect sag dsp
          cje    rb,#11111101b,swell0  ;detect swell dsp
          cje    rb,#11111011b,power0  ;detect power shutdown
          cje    rb,#11110111b,motion0 ;detect motion sensor

          cje    rb,#11011111b,getfcu
          cje    rb,#01111111b,getgsm
          goto   mula

getgsm    mov    ra,#1001b
          call   recva                ;check serial data dari gsm
          call   wal00m
          call   recval
          call   wait1s

          cje    cha,#00001101b,go1    ;on1
          goto   mula
go1       cje    ch1,#'G',miss        ;detect miss call +CRING
          goto   mula

```

```

miss clr    count0
miss0 clr   count1
      clr   count2
      clr   count3

      mov   ra,#1101b           ;1 led padam

      mov   chb,'#X'           ;send ! pada FCU utk interupt
      call  transb             ;ok
      mov   chb,'#I'           ;send ! pada FCU utk interupt
      call  transb             ;ok
      mov   chb,'#C'           ;send ! pada FCU utk interupt
      call  transb             ;ok
      mov   chb,'#O'           ;send ! pada FCU utk interupt
      call  transb             ;ok
      mov   chb,'#N'           ;send ! pada FCU utk interupt
      call  transb             ;ok
      mov   chb,'#E'           ;send ! pada FCU utk interupt
      call  transb             ;ok
      mov   chb,'#R'           ;send ! pada FCU utk interupt
      call  transb             ;ok
      mov   chb,'#G'           ;send ! pada FCU utk interupt
      call  transb             ;ok
      mov   chb,'#Y'           ;send ! pada FCU utk interupt
      call  transb             ;ok

scan0 add   count0,#1
scan1 add   count1,#1
scan2 add   count2,#1
scan3 add   count3,#1
      cje   rb,#11011111b,getfcu
      cjne  count3,#255,scan3    ;105
      cjne  count2,#255,scan2    ;24
      cjne  count1,#15,scan1     ;15
      cjb   count0,#6,miss0      ;if6 = 5 kali
      goto  mula

getfcu      clr    lambat
            mov   ra,#0111b
            call  recvb
dapat1     mov   buffer1,w
            cje   buffer1,#00100000b,on1 ;mainboard off
            cje   buffer1,#00100001b,on2 ;mainboard leak
            cje   buffer1,#00100010b,on3 ;mainboard fault
            cje   buffer1,#00100011b,on4 ;mainboard good
            cje   buffer1,#00100100b,on5 ;mainboard over

            cje   buffer1,#00100101b,on6 ;card1 off
            cje   buffer1,#00100110b,on7 ;card1 leak
            cje   buffer1,#00100111b,on8 ;card1 fault
            cje   buffer1,#00101000b,on9 ;card1 good
            cje   buffer1,#00101001b,on10 ;card1 over

```

```

cje  buffer1,#00101010b,on11 ;card2 off
cje  buffer1,#00101011b,on12 ;card2 leak
cje  buffer1,#00101100b,on13 ;card2 fault
cje  buffer1,#00101101b,on14 ;card2 good
cje  buffer1,#00101110b,on15 ;card2 over

cje  buffer1,#00101111b,on16 ;card3 off
cje  buffer1,#00110000b,on17 ;card3 leak
cje  buffer1,#00110001b,on18 ;card3 fault
cje  buffer1,#00110010b,on19 ;card3 good
cje  buffer1,#00110011b,on20 ;card3 over
goto  mula

;*****
on1  mov  ch2,#'1'
      mov  ch3,#'1'
      goto sms

on2  mov  ch2,#'1'
      mov  ch3,#'2'
      goto sms

on3  mov  ch2,#'1'
      mov  ch3,#'3'
      goto sms

on4  mov  ch2,#'1'
      mov  ch3,#'4'
      goto sms

on5  mov  ch2,#'1'
      mov  ch3,#'5'
      goto sms

;*****
on6  mov  ch2,#'2'
      mov  ch3,#'1'
      goto sms

on7  mov  ch2,#'2'
      mov  ch3,#'2'
      goto sms

on8  mov  ch2,#'2'
      mov  ch3,#'3'
      goto sms

on9  mov  ch2,#'2'
      mov  ch3,#'4'
      goto sms

on10 mov  ch2,#'2'
      mov  ch3,#'5'
      goto sms

;*****
on11 mov  ch2,#'3'

```



```

        mov    ch3,#'1'
        goto  sms

on12   mov    ch2,#'3'
        mov    ch3,#'2'
        goto  sms

on13   mov    ch2,#'3'
        mov    ch3,#'3'
        goto  sms

on14   mov    ch2,#'3'
        mov    ch3,#'4'
        goto  sms

on15   mov    ch2,#'3'
        mov    ch3,#'5'
        goto  sms

;*****
on16   mov    ch2,#'4'
        mov    ch3,#'1'
        goto  sms

on17   mov    ch2,#'4'
        mov    ch3,#'2'
        goto  sms

on18   mov    ch2,#'4'
        mov    ch3,#'3'
        goto  sms

on19   mov    ch2,#'4'
        mov    ch3,#'4'
        goto  sms

on20   mov    ch2,#'4'
        mov    ch3,#'5'
        goto  sms
;*****dsp analysis*****

sag0   mov    ch2,#'5'    ;dsp sag
        mov    ch3,#'1'
        goto  sms

swell0        mov    ch2,#'5'    ;dsp swell
        mov    ch3,#'2'
        goto  sms

power0       mov    ch2,#'9'    ;240v shutdown
        mov    ch3,#'7'
        mov    lambat,#'1'
        goto  sms

motion0      mov    ch2,#'9'    ;motion alert!
        mov    ch3,#'8'
        mov    lambat,#'2'

```

```

        goto sms

motion1  mov  ch2,#'9'    ;motion ok
        mov  ch3,#'6'
        clr  lambat
        goto sms
;*****

sms      mov  cha,#'A'    ;send sms
        call transa
        mov  cha,#'T'
        call transa
        mov  cha,#'+'
        call transa
        mov  cha,#'C'
        call transa
        mov  cha,#'M'
        call transa
        mov  cha,#'G'
        call transa
        mov  cha,#'S'
        call transa
        mov  cha,#'='    ;send sms
        call transa

nohp    mov  cha,#'"'    ;set phone no
        call transa
        mov  cha,#'0'    ;no kena tukar ler!
        call transa
        mov  cha,#'1'
        call transa
        mov  cha,#'3'
        call transa
        mov  cha,#'7'
        call transa
        mov  cha,#'9'
        call transa
        mov  cha,#'9'
        call transa
        mov  cha,#'8'
        call transa
        mov  cha,#'3'
        call transa
        mov  cha,#'2'
        call transa
        mov  cha,#'9'
        call transa
        mov  cha,#'"'    ;set phone no
        call transa

        mov  cha,#00001101b ;send enter
        call transa

        call wa500m

code    mov  cha,#'@'    ;region code
        call transa

```

```

mov    cha,#'0'
call  transa
mov    cha,#'7'
call  transa

mov    cha,#'0'    ;ID code
call  transa
mov    cha,#'0'
call  transa
mov    cha,#'0'
call  transa
mov    cha,#'0'
call  transa
mov    cha,#'0'
call  transa
mov    cha,#'0'
call  transa
mov    cha,#'0'
call  transa
mov    cha,#'1'    ;taman u @07 0000001*00!
call  transa

mov    cha,#'*'    ;FAULT code
call  transa

call  transa2      ;hantar FAULT code1 card
call  transa3      ;hantar FAULT code2

mov    cha,#'!'
call  transa
call  wait1s

mov    cha,#00011010b    ;CNTRL+Z 1A HEX
call  transa

mov    ra,#0000b    ;0 led on
call  wa500m
mov    ra,#1111b    ;1 led padam
call  wa500m

enter mov    cha,#00001101b    ;enter
call  transa

mov    cha,#'A'    ;delete all MESSAGE sms
call  transa
mov    cha,#'T'
call  transa
mov    cha,#'+'
call  transa
mov    cha,#'C'
call  transa
mov    cha,#'M'
call  transa
mov    cha,#'G'
call  transa
mov    cha,#'D'
call  transa

```

```

    mov    cha,#'='
    call  transa
    mov    cha,#'1'      ;delete all sms
    call  transa
    mov    cha,#','
    call  transa
    mov    cha,#'4'
    call  transa

    cje    lambat,#'1',lengaha      ;card2 off
    cje    lambat,#'2',lengahb      ;card2 off

dlay  clr    count0
      clr    count1
      clr    count2

dlay0 add    count0,#1
dlay1 add    count1,#1
dlay2 add    count2,#1
      cjne   count2,#255,dlay2      ;255
      cjne   count1,#255,dlay1      ;255
      cjne   count0,#20,dlay0       ;20
      goto   mula                   ;mula

lengaha  clr    count0                ;delay lambat sikit
          clr    count1                ;untuk motion dengan power down
          clr    count2                ;estimate
          clr    count3

lengah0  add    count0,#1
lengah1  add    count1,#1
lengah2  add    count2,#1
lengah3  add    count3,#1
          cjne   count3,#255,lengah3    ;255
          cjne   count2,#255,lengah2    ;255
          cjne   count1,#255,lengah1    ;255
          cjne   count0,#3,lengah0      ;3= 5-6 menit
          clr    lambat
          cje    rb,#11111011b,power0   ;detect power shutdown
          cje    rb,#11111111b,poweron  ;detect power on
          goto   mula                   ;mula

lengahb  clr    count0                ;delay lambat sikit
          clr    count1                ;untuk motion dengan power down
          clr    count2                ;estimate
          clr    count3

lengah4  add    count0,#1
lengah5  add    count1,#1
lengah6  add    count2,#1
lengah7  add    count3,#1
          cjne   count3,#255,lengah7    ;255
          cjne   count2,#255,lengah6    ;255
          cjne   count1,#255,lengah5    ;255
          cjne   count0,#3,lengah4      ;3= 5-6 menit

```

```

    clr    lambat
    cje    rb,#11110111b,motion0    ;detect motion sensor alert
    cje    rb,#11111111b,motion1    ;detect motion sensor ok
    goto   mula                      ;mula

;*****transmita*****
transa    bcf    txd1                ;txd rb6

        mov    tml,#20                ;300.55hz
        mov    tm0,#255                ;segment rs, 9600Mhz
ulanga    djnz  tm0,ulanga            ;delay rasanya
ulanga1   djnz  tml,ulanga1

        mov    cn,#8                  ;segment cn, 8
        nop

transal    rr     cha                  ;rotate right segment ch
        nop
        movb   txd1,c                  ;
        mov    tml,#20
        mov    tm0,#255
ulanga2    djnz  tm0,ulanga2
ulanga3    djnz  tml,ulanga3

        djnz   cn,transal

        nop
        nop
        nop
        nop
        nop
        nop
        bsf    txd1

        mov    tml,#20
        mov    tm0,#255
ulanga4    djnz  tm0,ulanga4
ulanga5    djnz  tml,ulanga5
        ret
;*****
;*****transa2*****
transa2    bcf    txd1                ;txd rb6

        mov    tml,#20                ;300.55hz
        mov    tm0,#255                ;segment rs, 9600Mhz
lagi1     djnz  tm0,lagi1            ;delay rasanya
lagi2     djnz  tml,lagi2

        mov    cn,#8                  ;segment cn, 8
        nop

lagi3     rr     ch2                  ;rotate right segment ch
        nop
        movb   txd1,c                  ;
        mov    tml,#20
        mov    tm0,#255

```

```

lagi4 djnz tm0,lagi4
lagi5 djnz tm1,lagi5

        djnz  cn,lagi3

        nop
        nop
        nop
        nop
        nop
        bsf   txd1

        mov   tm1,#20
        mov   tm0,#255
lagi6 djnz tm0,lagi6
lagi7 djnz tm1,lagi7
        ret
;*****
;*****transa2*****
transa3   bcf   txd1           ;txd rb6

        mov   tm1,#20           ;300.55hz
        mov   tm0,#255         ;segment rs, 9600Mhz
lagi8 djnz tm0,lagi8   ;delay rasanya
lagi9 djnz tm1,lagi9

        mov   cn,#8           ;segment cn, 8
        nop                   ;nop sekejap

lagi10    rr    ch3           ;rotate right segment ch
        nop                   ;nop sekejap
        movb  txd1,c          ;

        mov   tm1,#20
        mov   tm0,#255
lagi11    djnz  tm0,lagi11
lagi12    djnz  tm1,lagi12

        djnz  cn,lagi10

        nop
        nop
        nop
        nop
        nop
        bsf   txd1

        mov   tm1,#20
        mov   tm0,#255
lagi13    djnz  tm0,lagi13
lagi14    djnz  tm1,lagi14
        ret
;*****
;*****transmitb*****
transb    bcf   txd2           ;txd rb4

        mov   tm1,#20           ;300.55hz

```

```

        mov     tm0,#255           ;segment rs, 9600Mhz
ulangb  djnz   tm0,ulangb        ;delay rasanya
ulangb1 djnz   tml,ulangb1

        mov     cn,#8            ;segment cn, 8
        nop

transb1 rr     chb              ;rotate right segment ch
        nop                    ;nop sekejap
        movb   txd2,c           ;

        mov     tml,#20
        mov     tm0,#255
ulangb2 djnz   tm0,ulangb2
ulangb3 djnz   tml,ulangb3

        djnz   cn,transb1

        nop
        nop
        nop
        nop
        nop
        bsf    txd2

        mov     tml,#20
        mov     tm0,#255
ulangb4 djnz   tm0,ulangb4
ulangb5 djnz   tml,ulangb5
        ret
;*****
;*****receivel*****
recva   btfscl rxd1            ;rxd rb7 detect start bit 0 btfscl default
        goto   recva          ;if jumpa 1 go to terus

terusa  mov     tml,#10         ;300.55hz
        mov     tm0,#200       ;segment rs, 9600Mhz
recv1a  djnz   tm0,recv1a      ;delay rasanya, memang pun
recv2a  djnz   tml,recv2a      ;if tolak rs dgn 1 = 0 goto recv10
        mov     cn,#8          ;segment cn = 8 bit
        nop
recv3a  mov     tml,#20
        mov     tm0,#255
recv4a  djnz   tm0,recv4a
recv5a  djnz   tml,recv5a
        nop
        movb   c,rxd1          ;bit c = bit rxd ; carry/borrow flag

        rr     cha             ;rotate right sekali pd segment ch
;       mov     w,cha          ;cha dan w register only!
        djnz   cn,recv3a      ;if tolak cn dgn 1 = 0 goto recv0
        ret
;*****
;*****receivela*****
recval  btfscl rxd1            ;rxd rb7 detect start bit 0 btfscl default
        goto   recval          ;if jumpa 1 go to terus gsm modem

```

```

terusal    mov    tm1,#10                ;300.55hz
            mov    tm0,#200            ;segment rs, 9600Mhz
recv1a1    djnz   tm0,recv1a1 ;delay rasanya, memang pun
recv2a1    djnz   tm1,recv2a1 ;if tolak rs dgn 1 = 0 goto recv10
            mov    cn,#8                ;segment cn = 8 bit
            nop
recv3a1    mov    tm1,#20
            mov    tm0,#255
recv4a1    djnz   tm0,recv4a1
recv5a1    djnz   tm1,recv5a1
            nop
            movb  c,rxdl                ;bit c = bit rxd ; carry/borrow flag

            rr    ch1                    ;rotate right sekali pd segment ch
            djnz  cn,recv3a1 ;if tolak cn dgn 1 = 0 goto recv0
            ret
;*****

;*****receive2*****
recvb    btfsc  rxd2                ;rxd rb7 detect start bit 1
            goto  recvb                ;stabil la sikit.....:) 16 kali jer

terusb    mov    tm1,#10                ;300.55hz
            mov    tm0,#200            ;segment rs, 9600Mhz
recv1b    djnz   tm0,recv1b ;delay rasanya, memang pun
recv2b    djnz   tm1,recv2b ;if tolak rs dgn 1 = 0 goto recv10
            mov    cn,#8                ;segment cn = 8 bit
            nop
recv3b    mov    tm1,#20
            mov    tm0,#255
recv4b    djnz   tm0,recv4b
recv5b    djnz   tm1,recv5b
            nop
            movb  c,rxdd2                ;bit c = bit rxd ; carry/borrow flag

            rr    chb                    ;rotate right sekali pd segment ch
            mov    w,chb                ;cha dan w register only!
            djnz  cn,recv3b ;if tolak cn dgn 1 = 0 goto recv0
            ret
;*****
;*****

wa100m    clr    tm1                    ;delay 100 milisecond
            clr    tm2
            clr    tm3
wait0    add    tm1,#1
wait1    add    tm2,#1
wait2    add    tm3,#1
            cjne  tm3,#239,wait2        ;239
            cjne  tm2,#28,wait1        ;28
            cjne  tm1,#1,wait0         ;1
            nop                                ;nop 1x
            ret

wa500m    clr    tm1                    ;delay 100 milisecond
            clr    tm2

```



```
        clr    tm3
wait6   add    tm1,#1
wait7   add    tm2,#1
wait8   add    tm3,#1
        cjne  tm3,#255,wait8           ;239 tak betul lagi ni!
        cjne  tm2,#60,wait7           ;28
        cjne  tm1,#1,wait6            ;1
        nop
        ret                               ;nop 1x

wait1s  clr    tm1                       ;delay 1 second
        clr    tm2
        clr    tm3
wait3   add    tm1,#1
wait4   add    tm2,#1
wait5   add    tm3,#1
        cjne  tm3,#105,wait5          ;105
        cjne  tm2,#24,wait4           ;24
        cjne  tm1,#2,wait3            ;2
        ret
```

APPENDIX C

CLIENT SOFTWARE SOURCE CODE

Networking

```

void CSBClientConfigDlg::AddText(const CString& strText)
{
    LOG(CXListBox::Blue, CXListBox::White, 0, strText);
    SaveLog(strText);
}

void CSBClientConfigDlg::AddSystemText(const CString& strText)
{
    AddText(_T("* ") + strText + _T(" *"));
}

void CSBClientConfigDlg::OnMessage(CNDKMessage& message)
{
    CTime m_Time;
    m_Time = CTime::GetCurrentTime();
    CString strTime;
    strTime= m_Time.Format("[%a %d %b %Y %I:%M:%S %p]");

    switch (message.GetId())
    {
    case ChatUserJoin:
        {
            CString strNickname;

            message.GetAt(0, strNickname);

            CString strUserJoin;
            strUserJoin.Format(IDS_USER_JOIN, strNickname);

            AddSystemText(strUserJoin);
            LOGCON(CXListBox::Green, CXListBox::White,
0, strNickname);
            m_SvrCtrlPanel.AddNewNCCRCC(strNickname);
        }
        break;

    case SQLSyntax:
        {
        }
        break;

    case MobileRemoteCommand:
        {

```

```

        CString strNickname;
        CString strCommand;
        message.GetAt(0, strNickname);
        message.GetAt(1, strCommand);

        //calling parent to provide status
        if(strCommand=="#@&")
        {
            ::SendMessage(GetParent()->GetSafeHwnd(),
GSM_TERMINATE, GetDlgCtrlID(), 0);
            AddSystemText(strNickname+ " is sending command
to terminate the GSM Communication ");
        }
        if(strCommand=="*#%")
        {
            ::SendMessage(GetParent()->GetSafeHwnd(),
GSM_START, GetDlgCtrlID(), 0);
            AddSystemText(strNickname+ " is sending command
to initiate the GSM Communication ");
        }
    }
    break;

    case RequestStatusInitServer://receiving request frm server<<<---
-----
    {
        CString strNickname;
        CString strSBName;
        message.GetAt(0, strNickname);
        message.GetAt(1, strSBName);
        m_strPopSB = strSBName;
        AddSystemText(strNickname+ " is requesting status for
"+ strSBName);

        //calling parent to provide status
        ::SendMessage(GetParent()->GetSafeHwnd(),
REQUEST_NOW, GetDlgCtrlID(), 0);

    }
    break;

    case RequestStatusInitClient://Receiving status reply from server
<<<<-----
    {
        CString strNickname;
        CString strSBName;
        message.GetAt(0, strNickname);
        message.GetAt(1, strSBName);
        CString strRegion = m_SvrCtrlPanel.m_strParentSelect;
        CString strRcc = m_SvrCtrlPanel.m_strCurrentSelect;
        m_strPopSB = strRcc+">" + strRegion;
        AddSystemText(strSBName);

        m_SvrCtrlPanel.PushToTreeList(m_strPopSB, strSBName);
    }
}

```

```

        SaveLog(strSBName);
    }
    break;
case ChatText:
    {
        CString strNickname;
        CString strText;
        message.GetAt(0, strNickname);
        message.GetAt(1, strText);
        LOG(CXListBox::White, CXListBox::Navy, 0, strTime + " :
" + strNickname + _T(" : ") + strText);
        SaveLog(strNickname+" : "+strText);
    }
    break;

case ChatUserQuit:
    {
        CString strNickname;
        message.GetAt(0, strNickname);
        CString strUserQuit;
        strUserQuit.Format(IDS_USER_QUIT, strNickname);

        RemoveUser(strNickname);
        m_SvrCtrlPanel.RemoveNCCRCC(strNickname);
        LOG(CXListBox::Red, CXListBox::White, 0, strTime + " :
"+ strUserQuit);
        SaveLog(strUserQuit);
    }
    break;
}
}

```

// Called whenever an unexpected disconnection occurs. The only case when

// this method isn't call is when CloseConnection is used.

CloseConnection

// don't need to be called when when OnDisconnect is called. The derived

// class must override this method.

```
void CSBClientConfigDlg::OnDisconnect(NDKClientDisconnection
disconnectionType)
```

```
{
```

```
    //clear data..
```

```
    m_XListConnUser.ResetContent();
```

```
    m_SvrCtrlPanel.ResetListTree();
```

```
    UINT unResId = 0;
```

```
    switch (disconnectionType)
```

```
    {
```

```
    case NDKClient_NormalDisconnection:
```

```
        unResId = IDS_DISCONNECTED;
```

```
        break;
```

```
    case NDKClient_ServerCloseConnection:
```

```
        unResId = IDS_SERVER_CLOSE_CONNECTION;
```

```

        nReconnectAttemp=0;//reconnect flag counter
        SetTimer(501,60000,NULL);//negotiate connection
        break;

    case NDKClient_ServerStop:
        {
            unResId = IDS_SERVER_STOPPED;
            nReconnectAttemp=0;//reconnect flag counter
            SetTimer(501,60000,NULL);//negotiate connection
            break;
        }

    case NDKClient_ErrorSendingMessage:
        unResId = IDS_ERROR_SENDING_MESSAGE;
        nReconnectAttemp=0;//reconnect flag counter
        SetTimer(501,60000,NULL);//negotiate connection
        break;

    case NDKClient_ErrorReceivingMessage:
        unResId = IDS_ERROR_RECEIVING_MESSAGE;
        nReconnectAttemp=0;//reconnect flag counter
        SetTimer(501,60000,NULL);//negotiate connection
        break;

    default:
        break;
}

AddSystemText((LPCSTR)unResId);
bIsConnected = false;

UpdateUI();
}

// Called when the ping from the server is received. The number of
// milliseconds is returned since PingServer was called.
void CSBClientConfigDlg::OnPing(long lNbMilliseconds)
{
    CString strPing;
    strPing.Format(IDS_PING_RECEIVED, lNbMilliseconds);
    if(!m_bSessionKeepAlive)
    {
        AddSystemText(strPing);
        SaveLog(strPing);
    }
    m_bSessionKeepAlive = false;
}

void CSBClientConfigDlg::OnButtonConnect()
{/**
    CString strLocalIP;
    long lPort;
    SaveLog("Connecting to server.....");
    if (UpdateData(TRUE))
    {

```

```

if (IsConnected())
{
    SetWindowText("Disconnected..");
    SaveLog("Disconnected");
    CloseConnection();
    m_SvrCtrlPanel.ResetListTree();
    m_XListConnUser.ResetContent();
    KillTimer(502);
    m_bSessionKeepAlive=false;
    bIsConnected = false;
}
else
{
    if (OpenConnection(m_strIp, m_lPort))
    {
        GetIpAndPort(strLocalIP,lPort);
        CString strLocalInfo;
        strLocalInfo.Format("%s:%d",strLocalIP,lPort);
        CTime m_Time;
        m_Time = CTime::GetCurrentTime();
        CString strTime;
        strTime= m_Time.Format("[%a %d %b %Y %I:%M:%S
%p]");

        SetWindowText("Connected as " +m_strNickname +
["+strLocalInfo+ "] since "+ strTime);
        AddSystemText((LPCSTR)IDS_CONNECTED);
        SaveLog("Connected as "+m_strNickname +
["+strLocalInfo+" ] ");

        CNDKMessage message(ChatUserJoin);
        message.Add(m_strNickname);
        message.Add(strLocalInfo);

        SendMessageToServer(message);
        LOGCON(CXListBox::Blue, CXListBox::White,
0,"Server");

        SetTimer(502,600000,NULL);//ping every 10
mins....1000ms*60sec*10min
        KillTimer(501);//reconnect timer
        bIsConnected=true;
        PurgePendingMessage();

        //AfxMessageBox(strLocalIP);
    }
    else
    {
        CString str;
        str.Format(IDS_CANNOT_CONNECT);
        LOG(CXListBox::Red, CXListBox::Navy, 0,str);
        SaveLog(str);
        bIsConnected = false;
        if(m_bReconnect)

```

```

SetTimer(501,60000,NULL);//negotiate
connection
    }
    }
    UpdateUI();
}/**/

// AfxMessageBox(m_strPathName);
}

void CSBClientConfigDlg::OnButtonPingServer()
{
    PingServer();
    AddSystemText((LPCSTR)IDS_PING_SERVER);
    //+++++
    //simulate send request
    /*
    CNDKMessage message(RequestStatus);
    message.SetAt(0,"Bahaya");
    //getting target from UI
    message.SetAt(1,m_strNDKTarget);

    SendMessageToServer(message);
    /**/
}

void CSBClientConfigDlg::OnButtonSend()
{
    /**/

    if(!m_strChatInput.IsEmpty())
    {
        if (UpdateData(TRUE))
        {
            CTime m_Time;
            m_Time = CTime::GetCurrentTime();
            CString strTime;
            strTime= m_Time.Format("[%a %d %b %Y %I:%M:%S %p]");

            CNDKMessage message(ChatText);
            message.SetAt(0,m_strChatInput);
            //getting target from UI
            message.SetAt(1,m_strNDKTarget);

            SendMessageToServer(message);
            LOG(CXListBox::Red, CXListBox::White, 0,strTime + " :
"+ m_strChatInput);
            SaveLog(m_strChatInput);
            m_strChatInput.Empty();
            UpdateData(FALSE);
        }
    }
    else
        AfxMessageBox("Message Empty");
}

```

```

void CSBClientConfigDlg::SendNotification(CString strMessage,CString
strID)
{
    //    CStringArray    strArrayLog;
    if (UpdateData(TRUE))
    {
        CString strTime;
        CTime m_Time;
        CString strLog=strMessage;
        CString strServerMessage;
        m_Time = CTime::GetCurrentTime();
        strTime= m_Time.Format("[%a %d %b %Y %I:%M:%S %p] ");
        if(bInstantUpdate)//update instantly
        {
            strServerMessage = strMessage;
            if(bIsConnected)
            {
                CNDKMessage message(NotifyFault);
                message.Add(strServerMessage);
                SendMessageToServer(message);
                LOG(CXListBox::White, CXListBox::Red, 0,strTime
+" : " + strMessage);
                SaveLog(strMessage);
                UpdateData(FALSE);
                strArrayLog.RemoveAll();
            }
            if(!bIsConnected)
            {
                //CStringArray    strFaultPending;
                strFaultPending.Add(strServerMessage);
            }
        }
        else
        {
            strArrayLog.Add(strMessage);
        }

        //save to log//
        if(m_bRecord=="true")
        {
            FILE *fPtr;
            fPtr = fopen(m_strPathName,"a");
            if(fPtr==NULL)
            {
                m_strPathName="MyLog.txt";
                fPtr = fopen(m_strPathName,"a");
            }

            fprintf(fPtr,"%s\n",strTime+"\t"+ m_strNickname+"\t"
+strLog);

            fclose(fPtr);
        }
    }
}

```



```

}

void CSBClientConfigDlg::SendRequest(CString strFirst)
{
    //simulate replying status
    CNDKMessage message(RequestStatusInitServer);
    message.SetAt(0, strFirst);
    message.SetAt(1, "Server");

    //AddSystemText(" Sending status for "+ m_strPopSB);
    SendMessageToServer(message);
    SaveLog(strFirst);
}

void CSBClientConfigDlg::OnClntcfgQuerystat()
{
    // TODO: Add your command handler code here
    //AfxMessageBox("success");
}

void CSBClientConfigDlg::ShowServer()
{
    m_SvrCtrlPanel.ShowWindow(SW_SHOW);
}

LRESULT CSBClientConfigDlg::OnRequestUpdate(WPARAM wParam, LPARAM
lParam)
{
    //CSVRControlDlg m_SvrCtrlPanel;
    CString strRegion = m_SvrCtrlPanel.m_strParentSelect;
    CString strRcc = m_SvrCtrlPanel.m_strCurrentSelect;

    CNDKMessage message(RequestStatusInitClient);
    message.SetAt(0, strRcc+">"+strRegion);

    message.SetAt(1, "Server");

    SendMessageToServer(message);
    AddSystemText(" Requesting status of "+strRcc+" at "+ strRegion);
    SaveLog(" Requesting status of "+strRcc+" at "+ strRegion);

    return 0;
}

void CSBClientConfigDlg::SendOnlyToServer(CString strMessage)
{
    CTime m_Time;

```

```

        m_Time = CTime::GetCurrentTime();
        CString strTime;
        strTime= m_Time.Format("[%a %d %b %Y %I:%M:%S %p]");

        CNDKMessage message(ChatText);
        message.SetAt(0,strMessage);
        //getting target from UI
        message.SetAt(1,"Server");

        SendMessageToServer(message);
        LOG(CXListBox::Red, CXListBox::White, 0,strTime + " : "+
strMessage);
        SaveLog(strMessage);

    }

void CSBClientConfigDlg::SendSQLString(CString strSQL)
{
    CTime m_Time;
    m_Time = CTime::GetCurrentTime();
    CString strTime;
    strTime= m_Time.Format("[%a %d %b %Y %I:%M:%S %p]");

    CNDKMessage message(SQLSyntax);
    message.SetAt(0,strSQL);
    //getting target from UI
    message.SetAt(1,"Server");

    if(bIsConnected)
    {
        SendMessageToServer(message);
        LOG(CXListBox::Red, CXListBox::White, 0,strTime + " : "+
"SQL database string sent");
        SaveLog("SQL syntax has been sent to server");
    }
    // CStringArray strSQLPending;
    if(!bIsConnected)
        strSQLPending.Add(strSQL);
}

void CSBClientConfigDlg::PurgePendingMessage()
{
    //send the stored data
    CTime m_Time;
    m_Time = CTime::GetCurrentTime();
    CString strTime;
    strTime= m_Time.Format("[%a %d %b %Y %I:%M:%S %p]");

    //POSTING SQL SYNTAX-----
    CNDKMessage message(SQLSyntax);
    for(int i=0;i<strSQLPending.GetSize();i++)
    {

```

```

        if(!(strSQLPending.GetAt(i).IsEmpty()))
        {
            message.SetAt(0,strSQLPending.GetAt(i));
            message.SetAt(1,"Server");

            SendMessageToServer(message);
            LOG(CXListBox::Red, CXListBox::White, 0,strTime + " :
"+ "SQL database string sent");
            SaveLog("SQL syntax has been sent to server");
        }
    }
    strSQLPending.RemoveAll();

    //-----
    //POSTING PENDING FAULT MESSAGES...
    CNDKMessage FaultMessage(NotifyFault);
    for(int j=0;j<strFaultPending.GetSize();j++)
    {
        if(!(strFaultPending.GetAt(j).IsEmpty()))
        {
            FaultMessage.Add(strFaultPending.GetAt(j));
            SendMessageToServer(FaultMessage);
            LOG(CXListBox::White, CXListBox::Red, 0,strTime + " :
" + strFaultPending.GetAt(j));
            SaveLog(strFaultPending.GetAt(j));
        }
    }
    strFaultPending.RemoveAll();
}
}

```

Main Program

```

void CSubStationMonitorDlg::OnTimer(UINT nIDEvent)
{
    if( nIDEvent == 1)
    {
        nCounterDelay++;
        len = m_commctrl.ReadRs232Input(&ptr1, nComPorts-1);

        if(len > 0 )
        {
            m_BtnMonMode.SetIcon(IDI_GSMBUSY);
            str1.Empty();
            while(len--)
            {
                /*
                if((*ptr1 == 0x0D))
                {
                }
                */
            }
        }
    }
}

```

```

        else
        {
            str1 += *ptr1;
            m_strGSMRecieve = str1;
        }
        ptr1++;
    }

    OnGSMFilter(m_strGSMRecieve);
}
}
CDialog::OnTimer(nIDEvent);
}

LRESULT CSubStationMonitorDlg::OnSendSMS(WPARAM ch, LPARAM port)
{
    CString strNo;
    CString strText;

    //CGSM_SMS_Send          m_SendSMS;
    strNo = m_SendSMS.m_strDestNumber;
    strText = m_SendSMS.m_strSMSBody;
    bSendSMS = true;
    CString str1, str2;
    str1.Format("at+cmgs=%c%s%c", 34, strNo, 34);
    str1 += 0x0d;
    m_BtnMonMode.SetIcon(IDI_GSMBUSY);
    m_commctrl.SendCString(str1, nComPorts-1); // set port

    //CString str1, str2;
    str1.Format("%s%c", strText, 26); // 26 = CTRL_Z
    str1 += 0x0d;

    m_BtnMonMode.SetIcon(IDI_GSMBUSY);
    m_commctrl.SendCString(str1, nComPorts-1); // set port

    return 0;
}

LRESULT CSubStationMonitorDlg::OnHangUp(WPARAM ch, LPARAM port)
{
    CString strNo;
    //CGSM_SMS_Send          m_SendSMS;
    strNo = m_SendSMS.m_strDestNumber;
    CString str;
    str.Format("ATH%c", 0x0D);
    m_commctrl.SendCString(str, nComPorts-1);
    Sleep(100); // resend to confirm
    m_commctrl.SendCString(str, nComPorts-1); // set port
    return 0;
}

LRESULT CSubStationMonitorDlg::OnDial(WPARAM ch, LPARAM port)
{
    CString strNo;
    //CGSM_SMS_Send          m_SendSMS;
    strNo = m_SendSMS.m_strDestNumber;

```

```

    CString str;
    str.Format("ATD%s;%c",strNo,0x0D);
    m_commctrl.SendCString(str,nComPorts-1);
    return 0;
}

void CSubStationMonitorDlg::ProcessMessages(CString strMessage)
{
    //some init.
    CString      strSendSMSUser1;
    CString      strSendSMSUser2;
    CString      strSendSMSUser3;
    CString      strSendSMSUser4;
    CString      strSendSMSUser5;

    CString tmpMessage = strMessage;

    int SbIDLenght = 0;
    int SbCodeLength = 0;
    int nMessageLength = tmpMessage.GetLength();    //get the whole
message length
    int HatchPos = tmpMessage.Find("*",0);
    //-----
    //Message Segemntation
    //-----
    //Header identifier
    CString strHeader = tmpMessage.Left(1);
    //if(strHeader =="@")
    //{
    //substation ID and error code extraction
    CString SBID,SBFault,SBFault1,SBFault2;

    SbCodeLength = nMessageLength-HatchPos-1;

    SBID = tmpMessage.Left(HatchPos);
    SBFault1 = tmpMessage.Right(SbCodeLength);    //dummy code for
processing card
    SBFault2 = SBFault1.Left(2);
    int nFaultCodeMessage;
    nFaultCodeMessage = atoi(SBFault2);

    //get card number for substation fault
    CString      strActiveCard;
    CString strCardIndex="";
    CString strCardCode="";

    if(nFaultCodeMessage<50)
    {
    //-----retrieve card name*****
        //renew faultcode
        SBFault = SBFault2.Right(1); //take code for street light
problem...
        CString sbCode = SBID.Right(9);
        CString sbNami = GetSubstationName(sbCode);
        //get card id.
        strCardIndex =      SBFault2.Left(1);
        CString Data;

```

```

CString openformat;
//DWORD Index,iCode;
CString SQLString;

SQLString.Format("Select * FROM substationprofile WHERE
((UniqueID= '%s') AND "
"(Name = '%s'))",sbCode+"@"+m_strRegionCode,sbNami);

openformat.Format("%s",SQLString);
Connection con(use_exceptions);
try {
    ostringstream strbuf;

con.real_connect(m_strDatabaseName,m_strHostName,m_strClientUsern
ame,
m_strClientPassword,atoi(m_strPortNum),(int)0,60,NULL);
    Query query = con.query();
    query << openformat;
    Result res = query.store();
    Row row;
    Result::iterator i;

    for (i = res.begin(); i != res.end(); i++)
    {
        row = *i;
        CString strTmp;
        if(strCardIndex=="1")
        {
            strCardCode = "Card1";
            strTmp.Format("%s",row["Card1"]);
            strActiveCard = strTmp;
        }
        if(strCardIndex=="2")
        {
            strCardCode = "Card2" ;
            strTmp.Format("%s",row["Card2"]);
            strActiveCard = strTmp;
        }
        if(strCardIndex=="3")
        {
            strCardCode = "Card3" ;
            strTmp.Format("%s",row["Card3"]);
            strActiveCard = strTmp;
        }
        if(strCardIndex=="4")
        {
            strCardCode = "Card4" ;
            strTmp.Format("%s",row["Card3"]);
            strActiveCard = strTmp;
        }
        //reload sms tag..to send sms purpose

strSendSMSUser1.Format("%s",row["Contact1Enable"]);

```

```

strSendSMSUser2.Format("%s",row["Contact2Enable"]);
strSendSMSUser3.Format("%s",row["Contact3Enable"]);
strSendSMSUser4.Format("%s",row["Contact4Enable"]);
strSendSMSUser5.Format("%s",row["Contact5Enable"]);

        }

    } catch (BadQuery er)
    {
        cerr << "Error: " << er.error << " " << con.errnum()
<< endl;
        return ;
    }
}
else
{
    SBFault = SBFault2;
}
CString FaultType = m_SBSetting.GetFaultType(SBFault);

int ctlColor;
if(FaultType!="")
{
    CString sbNameBaru = SBID.Right(9);
    CString sbName = GetSubstationName(sbNameBaru);
    //look for the cell color
    COLORREF ColorLines;
    if(nSBHit+1%2==0)
        ColorLines = RGB(245,245,245);
    if(nSBHit+1%2==1)
        ColorLines = RGB(230,230,230);
    if(!sbName.IsEmpty())
    {
        CString fmt;
        if(strCardCode == "")
        {
            if((nFaultCodeMessage==99) ||
(nFaultCodeMessage==97) )
            {
                fmt.Format("UPDATE substationprofile SET
Status= '%s' WHERE ((UniqueID= '%s') AND "
"(Name =
'%s'))",FaultType,sbNameBaru+"@"+m_strRegionCode,sbName);
                ctlColor =
GetColorByFaultName(FaultType);

                m_ListView.SetItemText(nSBHit,3,FaultType,ctlColor,ColorLines);

                if(CheckBlinkFlag(FaultType))
                    SetCellBlink(FaultType, nSBHit, 3);
            }
        }
    }
}

```

```

        if((nFaultCodeMessage==98) ||
(nFaultCodeMessage==95) )
        {
            fmt.Format("UPDATE substationprofile SET
SecurityStat= '%s' WHERE ((UniqueID= '%s') AND "
            "(Name =
'%s'))",FaultType,sbNameBaru+"@"+m_strRegionCode,sbName);
            ctlColor =
GetColorByFaultName(FaultType);

            m_ListView.SetItemText(nSBHit,9,FaultType,ctlColor,ColorLines);

            if(CheckBlinkFlag(FaultType))
                SetCellBlink(FaultType, nSBHit, 9);
        }
        if((nFaultCodeMessage>=50) &&
(nFaultCodeMessage<=60) )
        {
            fmt.Format("UPDATE substationprofile SET
DSPStat= '%s' WHERE ((UniqueID= '%s') AND "
            "(Name =
'%s'))",FaultType,sbNameBaru+"@"+m_strRegionCode,sbName);
            ctlColor =
GetColorByFaultName(FaultType);

            m_ListView.SetItemText(nSBHit,8,FaultType,ctlColor,ColorLines);

            if(CheckBlinkFlag(FaultType))
                SetCellBlink(FaultType, nSBHit, 8);
        }
    }
    if(strCardCode =="Card1")
    {
        fmt.Format("UPDATE substationprofile SET
Card1Stat= '%s' WHERE ((UniqueID= '%s') AND "
            "(Name =
'%s'))",FaultType,sbNameBaru+"@"+m_strRegionCode,sbName);

        ctlColor = GetColorByFaultName(FaultType);

        m_ListView.SetItemText(nSBHit,4,FaultType,ctlColor,ColorLines);

        if(CheckBlinkFlag(FaultType))
            SetCellBlink(FaultType, nSBHit, 4);
    }
    if(strCardCode =="Card2")
    {
        //CString fmt;
        fmt.Format("UPDATE substationprofile SET
Card2Stat= '%s' WHERE ((UniqueID= '%s') AND "
            "(Name =
'%s'))",FaultType,sbNameBaru+"@"+m_strRegionCode,sbName);

        ctlColor = GetColorByFaultName(FaultType);
    }
}

```



```

m_ListView.SetItemText(nSBHit,5,FaultType,ctlColor,ColorLines);

        if(CheckBlinkFlag(FaultType))
            SetCellBlink(FaultType, nSBHit, 5);
    }

    if(strCardCode =="Card3")
    {
        //CString fmt;
        fmt.Format("UPDATE substationprofile SET
Card3Stat= '%s' WHERE ((UniqueID= '%s') AND "
        "(Name =
'%s'))",FaultType,sbNameBaru+"@"+m_strRegionCode,sbName);
        //save data to log

        ctlColor = GetColorByFaultName(FaultType);

m_ListView.SetItemText(nSBHit,6,FaultType,ctlColor,ColorLines);

        if(CheckBlinkFlag(FaultType))
            SetCellBlink(FaultType, nSBHit, 6);
    }

    if(strCardCode =="Card4")
    {
        //CString fmt;
        fmt.Format("UPDATE substationprofile SET
Card4Stat= '%s' WHERE ((UniqueID= '%s') AND "
        "(Name =
'%s'))",FaultType,sbNameBaru+"@"+m_strRegionCode,sbName);
        //save data to log
        ctlColor = GetColorByFaultName(FaultType);

m_ListView.SetItemText(nSBHit,7,FaultType,ctlColor,ColorLines);

        if(CheckBlinkFlag(FaultType))
            SetCellBlink(FaultType, nSBHit, 7);
    }

ViewDefaultDatabase(fmt);
//-----
//*****
CString faultMessage;
faultMessage.Format("%s\n*s*",sbName,FaultType);
//send message through TCP/IP network
CString      strTCPIPMessage;

    strTCPIPMessage.Format("%s@%s*s*s",sbNameBaru,sbName,FaultType,
strCardCode);

m_ClientConfig.SendNotification(strTCPIPMessage,sbNameBaru);
    //get phone number to alert..

    //check alert send SMS Flag
    if(CheckSendSMSFlag(FaultType) &&
strSendSMSUser1=="Yes")

```

```

        {
            m_strSMSAlertPhoneNo =
GetPhoneNumber( sbNameBaru,11);

            SendAlertViaSMS(m_strSMSAlertPhoneNo, strTCPIPMessage+ "-"
>"+strActiveCard);
                strSendSMSUser1 = "No";//reset value
        }

//=====
        m_AlertMsg.Alert( faultMessage, FaultType);
        m_AlertMsg.ShowWindow(SW_SHOW);
        //sending message to be save into database.....
        CString strTmpLog;

        strTmpLog.Format( "%s/%s", m_strOperatorRole, m_strCurrentOperator);

        m_SBLog.SetAlertLog( sbName, FaultType, strTmpLog, strCardCode, sbName
Baru+"@"+m_strRegionCode);
                //Sleep(500);
                //m_strSMSAlertPhoneNo =
GetPhoneNumber( sbNameBaru,12);

        //SendAlertViaSMS(m_strSMSAlertPhoneNo, strTCPIPMessage+ " at
"+strActiveCard);
                //-----
-
        }
    }

}

void CSubStationMonitorDlg::OnGSMNotify(CString strNotify)//first time
is alert..second query sms
{
//    m_wndTaskbarNotifier.Show(strNotify);

    nCounterDelay = 0;
    CString str;
    str = strNotify;
    CString strTmp, strTmp2="";
    CString      strMessageIndex;
    strMessageIndex = strNotify;
    //int nLength;

    //finding message header..get notification type
    int nstart = strNotify.Find('+',0);
    int nend = strNotify.Find(':');

    if(strNotify.GetLength()>4)
    {
        if((nstart>0 )&& (nstart<strNotify.GetLength()))
            for(int u=nstart; u<nend;u++)
                strTmp2+=strNotify.GetAt(u);
    }
}

```

```

    }
    //nGSMSMNotify++;
    //strDummyBuffer+=strNotify;
// strDummyBuffer+=strTmp2;
//TRACE("\n%s-->%d",strNotify,nGSMSMNotify);
//+++++
if(strTmp2=="+CMGL")//list message
{
    //strDummyBuffer+=strNotify;
    //*****RECORD TYPE*****
    /*
    CString strRecNum="";
    int nStart00 = strNotify.Find(':',0);
    int nEnd00 = strNotify.Find(',',nStart00);
    for(int u=nStart00+1; u<nEnd00;u++)
        strRecNum+=strNotify.GetAt(u);

    m_wndTaskbarNotifier.Show("Negotiating Transfer...\nPlease
Wait..");
    nSimcardIndex = atoi(strRecNum);
    if(nSimcardIndex==0)
        nSimcardIndex=1;
    CString str;
    str.Format("AT+CMGR=%d%c",nSimcardIndex,0x0D);
    m_BtnMonMode.SetIcon(IDI_GSMBUSY);
    m_commctrl.SendCString(str,nComPorts-1); //rileks dulu
    /*/
    //strDummyBuffer+=strNotify;
    return;
}

if(strTmp2=="+COPS")//network setting
{
    if(m_bAvailableNetwork)
    {
        m_wndTaskbarNotifier.Show(strNotify);

        m_strAvailableNetwork;
        m_bAvailableNetwork = false;
        KillTimer(9);
        KillTimer(8);
        KillTimer(7);
        m_wndTaskbarNotifier.Show("Information
Transferred!");
        //SetTimer(10,1000,NULL);
    }
    if(m_bRegisterNetwork)
    {
        m_wndTaskbarNotifier.Show(strNotify);
        m_strRegisterNetwork;
        m_bRegisterNetwork= false;
        KillTimer(8);
        SetTimer(9,10000,NULL);
    }
    return;
}

if(strTmp2=="+CPIN")//pin number

```

```

{
    CString strMode;
    int nStart = strNotify.Find(':',0);
    int nEnd = strNotify.GetLength();
    for(int u=nStart+2; u<nEnd;u++)
        strMode+=strNotify.GetAt(u);
    m_wndTaskbarNotifier.Show("Simcard Status\n"+strMode);
    strMode.TrimRight('\n');
    if(strMode=="READY")
    {
        //return;
    }
    if(strMode=="SIM PUK")
    {
        CMeterDecDlg dlg ;
        CString strPinNo="";
        dlg.m_Edit1.Format("");
        dlg.m_Title.Format("Enter SIM Card PUK Number.") ;

        if (dlg.DoModal() == IDOK)
        {
            strPinNo = dlg.m_Edit1;
            CString str;
            str.Format("AT+CPIN=%s%c",strPinNo,0x0D);
            m_commctrl.SendCString(str,nComPorts-1);
            return;
        }
        return;
    }
    if(strMode=="SIM PIN")
    {
        CMeterDecDlg dlg ;
        CString strPinNo="";
        dlg.m_Edit1.Format("");
        dlg.m_Title.Format("Enter SIM Card Pin Number.") ;

        if (dlg.DoModal() == IDOK)
        {
            strPinNo = dlg.m_Edit1;
            CString str;
            str.Format("AT+CPIN=%s%c",strPinNo,0x0D);
            m_commctrl.SendCString(str,nComPorts-1);
            return;
        }
    }

    Sleep(500);
    CString str;
    str.Format("AT+CREG?%c",0x0D);
    m_commctrl.SendCString(str,0);
    //DisplaySignalQualityMeter(nSignalQualityIndex);

    return;
}
if(strTmp2=="WIND")//sim card status
{
    CString strStatus="";

```

```

int nStart = strNotify.Find(':',0);
int nEnd = strNotify.GetLength();//Find(',',nStart);
for(int u=nStart+1; u<nEnd;u++)
    strStatus+=strNotify.GetAt(u);

int nstatus = atoi(strStatus);
m_BtnMonMode.SetIcon(IDI_GSM);
if(nstatus== 1)
{
    m_wndTaskbarNotifier.Show("Sim Card Detected");
    CString str;
    str.Format("AT%c",0x0D);
    m_commctrl.SendCString(str,nComPorts-1);

    Sleep(1000);
    str.Format("AT+CPIN?%c",0x0D);
    m_commctrl.SendCString(str,nComPorts-1);
    return;
}
if(nstatus== 0)
{
    m_wndTaskbarNotifier.Show("Sim Card Not Available");
    return;
}
}

if(strTmp2=="+CREG")//registration of simcard
{
    CString strStatus="";
    CString strReport="Unknown Network";
    int nStart = strNotify.Find(':',0);
    int nEnd = strNotify.GetLength();//Find(',',nStart);
    for(int u=nStart+1; u<nEnd;u++)
        strStatus+=strNotify.GetAt(u);

    int nstatus = atoi(strStatus);
    if((nstatus== 1)|| (nstatus==5))
    {
        strReport = "Logged On to Mobile Netwok";
        DisplaySignalQualityMeter(nSignalQualityIndex);
    }
    else
    {
        strReport = "Searching for Network..registering";
        m_BtnMonMode.SetIcon(IDI_GSM);
    }
    m_wndTaskbarNotifier.Show( strReport);
    return;
}
if(strTmp2=="+CME ERROR")//simcard error handling
{
    //AfxMessageBox("CME Error");
    CString strStatus="";
    int nStart = strNotify.Find(':',0);
    int nEnd = strNotify.GetLength();
    for(int u=nStart+1; u<nEnd;u++)
        strStatus+=strNotify.GetAt(u);
}

```

```

        int nstatus = atoi(strStatus);
        m_BtnMonMode.SetIcon(IDI_GSM);
        if(nstatus== 10)
        {
            m_wndTaskbarNotifier.Show("Sim Card not present..CME
ERROR 10");
        }
        if(nstatus== 13)
        {
            m_wndTaskbarNotifier.Show("Sim Card failure, insert
new one..CME ERROR 13");
        }
        if(nstatus== 14)
        {
            m_wndTaskbarNotifier.Show("Sim Card busy..CME ERROR
14");
        }
        if(nstatus== 15)
        {
            m_wndTaskbarNotifier.Show("Sim Card wrong..CME ERROR
15");
        }
        if(nstatus== 16)
        {
            m_wndTaskbarNotifier.Show("Incorrect Pin
Number..Please retry..");
            CString str;
            str.Format("AT%c",0x0D);
            m_commctrl.SendCString(str,nComPorts-1);

            Sleep(1000);
            str.Format("AT+CPIN?%c",0x0D);
            m_commctrl.SendCString(str,nComPorts-1);
            return;
        }
        if(nstatus== 20)
        {
            m_wndTaskbarNotifier.Show("Sim Card memory full..CME
ERROR 20");
        }
        if(nstatus== 20)
        {
            m_wndTaskbarNotifier.Show("No Network Service..CME
ERROR 30");
        }

        bSendSMS= false;
        return;
    }

    if(strTmp2=="+CMS ERROR")
    {
        m_wndTaskbarNotifier.Show("Negotiating Transfer....\nPlease
Wait..");
        bSendSMS= false;
        m_BtnMonMode.SetIcon(IDI_GSM0);
    }

```

```

        Sleep(1000);
        OnMonmode();
        return;
    }

    if(strTmp2=="+CMTI")
    {

        CString strIndex="0";
        CString ss;
        //get message index;
        int nIndex = strMessageIndex.ReverseFind(',');
        int nLength = strMessageIndex.GetLength();
        if((nIndex>0 ) && ( nIndex<50))
            for(int y=nIndex+1;y<nLength-1;y++)//find messae
index..not more than 10
                strIndex += strNotify.GetAt(y);
        nSimcardIndex = atoi(strIndex);
        CString str;
        str.Format("AT+CMGR=%d%c",nSimcardIndex,0x0D);
        m_BtnMonMode.SetIcon(IDI_GSMBUSY);
        m_commctrl.SendCString(str,nComPorts-1);
        return;
    }
    if(strTmp2=="+CRING")
    {
        //Start SMS query
        CString str,strNum;
        str = strNotify;
        int nStartS = strNotify.Find('"',0);
        int nFinishS = strNotify.ReverseFind('"');
        for(int u=nStartS+1; u<nFinishS;u++)
            strNum+=strNotify.GetAt(u);

        m_wndTaskbarNotifier.Show("Voice Call Received\n"+strNum);

        m_ClientConfig.SendOnlyToServer("Voice Call Received from -
-- "+strNum);
        strNum.Empty();
        m_BtnMonMode.SetIcon(IDI_GSMBUSY);
        return;
    }
    if(strTmp2=="+CSQ")//signal quality Check
    {
        CString str,strStrength;
        str = strNotify;

        int nStart = strNotify.Find(':',0);
        int nEnd = strNotify.Find(',',nStart);

        for(int t=nStart+1;t<nEnd;t++)
            strStrength +=strNotify.GetAt(t);
        if((strStrength.IsEmpty()) || (strStrength=="99"))
        {
            m_wndTaskbarNotifier.Show("Negotiating
Transfer....\nPlease Wait..");
            CString str;

```

```

        str.Format("AT+CSQ%c", 0x0D);
        m_BtnMonMode.SetIcon(IDI_GSMBUSY);
        m_commctrl.SendCString(str, nComPorts-1);
        //Sleep(100);
        return;
    }
    //m_BtnMonMode.SetIcon(IDI_GSM);
    int nQuality = atoi(strStrength);
    if((nQuality>0) && (nQuality<99))
    {
        nSignalQualityIndex=nQuality;
        m_SendSMS.SetSignalQualityScale(nQuality);
        DisplaySignalQualityMeter(nQuality);
    }

    CString sglQty;
    if(nQuality==0) sglQty="BAD";
    if((nQuality>0)&& (nQuality<=5)) sglQty="POOR";
    if((nQuality>5)&& (nQuality<=9)) sglQty="MEDIUM";
    if((nQuality>9)&& (nQuality<=14)) sglQty="GOOD";
    if((nQuality>14)&& (nQuality<=98)) sglQty="EXCELLENT";

    m_wndTaskbarNotifier.Show("Signal Quality\n" + sglQty);
    return;
}
if(strTmp2=="CMT")
{
    m_BtnMonMode.SetIcon(IDI_GSMBUSY);
    CString m_str, strTmp3;
    m_str = strNotify;
    int nLengthAll= strNotify.GetLength();

    int nStart= -1, nStart1=-1, nStart2=-1;
    int nStart3=-1, nStart4=-1, nStart5=-1, nStart6=-1, nStart7=-
1, nStart8=-1;

    //*****
    //*****Source Address*****

    CString SourceID;
    nStart = strNotify.Find('"', 0);
    nStart2 = strNotify.Find('"', nStart+1);
    for(int s=nStart+1; s<nStart2; s++)
        SourceID += strNotify.GetAt(s);

    //*****TIMESTAMP*****
    CString TimeStamp;
    nStart3 = strNotify.Find('"', nStart2+1);
    nStart4 = strNotify.Find('"', nStart3+1);

    for(int q=nStart3+1; q<nStart4; q++)
        TimeStamp += strNotify.GetAt(q);
    //*****MESSAGE BODY*****
    nStart7 = nStart4;
    nStart8 = nStart7;

    CString MessageBody;

```



```

nStart7 = strNotify.Find('@',nStart4);
nStart8 = strNotify.Find('!',nStart7+1);
CString strq;

//checking message body structure
if((nStart7>nStart4) && (nStart8>nStart7))//correct message
body found
{
    for(int w=nStart7;w<=nStart8;w++)//reading message
        MessageBody += strNotify.GetAt(w);
    int nSMSBodyLength = MessageBody.GetLength();
    if(nSMSBodyLength==14)
    {
        ProcessMessages(MessageBody);//main validation
selection
        nResendCntr = 0;//reset counter if message
valid
        //m_BtnMonMode.SetIcon(IDI_GSM0);
        DisplaySignalQualityMeter(nSignalQualityIndex);
        return;
    }
    else//request to read from simcard because data is
invalid
    {
        m_wndTaskbarNotifier.Show("Unknown
Message!!\n"+ MessageBody);
        //m_BtnMonMode.SetIcon(IDI_GSM0);
        DisplaySignalQualityMeter(nSignalQualityIndex);
        return;
    }
}
}
if(strTmp2=="+CMGR")//sms is read
{
    //m_wndTaskbarNotifier.Show("CMGR Detected!!");
    m_BtnMonMode.SetIcon(IDI_GSM0);
    CString m_str,strTmp3;
    m_str = strNotify;
    int nLengthAll= strNotify.GetLength();

    //*****RECORD TYPE*****
    CString RecType;
    int nStart= -1,nStart1=-1,nStart2=-1;
    int nStart3=-1,nStart4=-1,nStart5=-1,nStart6=-1,nStart7=-
1,nStart8=-1;

    nStart = strNotify.Find('"',0);
    nStart2 = strNotify.Find('"',nStart+1);

    for(int r=nStart+1;r<nStart2;r++)
        RecType+= strNotify.GetAt(r);
    //*****Source Address*****
    CString SourceID;
    nStart3 = strNotify.Find('"',nStart2+1);
    nStart4 = strNotify.Find('"',nStart3+1);

```

```

for(int s=nStart3+1;s<nStart4;s++)
    SourceID += strNotify.GetAt(s);
//*****TIMESTAMP*****
CString TimeStamp;
nStart5 = strNotify.Find('"',nStart4+1);
nStart6 = strNotify.Find('"',nStart5+1);

for(int q=nStart5+1;q<nStart6;q++)
    TimeStamp += strNotify.GetAt(q);

//*****MESSAGE BODY*****
CString MessageBody;
for(q=nStart6+2;q<nLengthAll;q++)
    MessageBody += strNotify.GetAt(q);

nStart7 = nStart6;
nStart8 = nStart7;
nStart7 = strNotify.Find('@',nStart6);
nStart8 = strNotify.Find('!',nStart7+1);
if((nStart7>nStart6) && (nStart8>nStart7))//correct message
body found
{
    MessageBody=" ";

    for(int w=nStart7;w<=nStart8;w++)//reading message
        MessageBody += strNotify.GetAt(w);
    int nSMSBodyLength = MessageBody.GetLength();

    m_SMSBuffer.PustSMSToBuffer(SourceID,TimeStamp,MessageBody);
    if(nSMSBodyLength==14)
    {
        m_BtnMonMode.SetIcon(IDI_GSMBUSY);
        CString str;

        str.Format("AT+CMGD=%d,3%c",nSimcardIndex,0x0D);
        m_commctrl.SendCString(str,nComPorts-1);
        Sleep(500);
        CString dat;
        dat.Format("SMS %d deleted",nSimcardIndex);
        m_wndTaskbarNotifier.Show(dat);
        //after delete then process
        ProcessMessages(MessageBody);//main validation
selection
        nResendCntr = 0;//reset counter if message
valid
        DisplaySignalQualityMeter(nSignalQualityIndex);
        return;
    }
    else//request to read from simcard because data is
invalid
    {
        m_wndTaskbarNotifier.Show("Unable to
decode...Unknown Message!!\n"+ MessageBody);
        DisplaySignalQualityMeter(nSignalQualityIndex);
        return;
    }
}

```

```

    }

    //*****
}
else//look like message incomplete..request for re-send
{
    nResendCntr++;//increase counter as resend is being
called
    if(nSimcardIndex==0)
        nSimcardIndex=1;//correct unexpected error of
accessing invalid sms memory
    CString str;
    str.Format("AT+CMGR=%d%c",nSimcardIndex,0x0D);
    if(nResendCntr<=3)
    {
        m_BtnMonMode.SetIcon(IDI_GSMBUSY);
        m_commctrl.SendCString(str,nComPorts-1);
        m_wndTaskbarNotifier.Show("Negotiating
Transfer....\nPlease Wait..");
        Sleep(100);
        return;
    }
    else
    {

        m_SMSBuffer.PustSMSToBuffer(SourceID,TimeStamp,MessageBody);
        m_wndTaskbarNotifier.Show("Request
timeout\nUnable to decode message");
        CString str;
        str.Format("AT%c",0x0D); //ping
        m_bPing= true;
        m_commctrl.SendCString(str,nComPorts-1);

        //DisplaySignalQualityMeter(nSignalQualityIndex);
        nResendCntr = 0;
        return;
    }
}
return;

}

if(strTmp2=="+CDSI")//Status delivery message
{
    m_wndTaskbarNotifier.Show("Status delivery report!!");
    return;
}

if(strTmp2=="+CPMS")//Simcard memory summary
{
    m_BtnMonMode.SetIcon(IDI_GSMBUSY);
    int Comma1,Comma2,Comma3;
    Comma1 = strNotify.Find(', ',0);
    Comma2 = strNotify.Find(', ',Comma1+1);
    Comma3 = strNotify.Find(', ',Comma2+1);

    CString    strCurrentBuffer;

```

```

CString      strTotalBuffer;
if((Comma1>=0)&&(Comma2>Comma1)&&(Comma3>Comma2))
{
    //get current buffer
    for(int z=Comma1+1;z<Comma2;z++)
        strCurrentBuffer+=strNotify.GetAt(z);

    //get Total buffer
    for(int d=Comma2+1;d<Comma3;d++)
        strTotalBuffer+=strNotify.GetAt(d);
    m_wndTaskbarNotifier.Show("Simcard Memory\nUsed
Buffer : "+strCurrentBuffer+
        "\nTotal Buffer: "+ strTotalBuffer);
    nSimCurrentMemory =atoi(strCurrentBuffer);
    nSimTotalMemory   = atoi(strTotalBuffer);
    //requesting unread sms
    Sleep(100);
    m_BtnMonMode.SetIcon(IDI_GSMBUSY);
    CString str;
    str.Format("AT+CMGL=%cREC
UNREAD%c%c",0x22,0x22,0x0D);
    m_commctrl.SendCString(str,nComPorts-1);
    return;
}
else
{
    m_BtnMonMode.SetIcon(IDI_GSMBUSY);
    m_wndTaskbarNotifier.Show("Fail to
deliver...\nRequesting!!");
    CString str;
    str.Format("AT+CPMS?%c",0x0D);
    m_commctrl.SendCString(str,nComPorts-1);
    Sleep(100);
    return;
}

}
strNotify="";
m_strGSMRecieve.Empty();
}

void CSubStationMonitorDlg::OnGsmPing()
{
    // TODO: Add your command handler code here
    CString str;
    m_bPing = true;

    str.Format("AT%c",0x0D);
    m_BtnMonMode.SetIcon(IDI_GSMBUSY);
    m_commctrl.SendCString(str,nComPorts-1);
}

void CSubStationMonitorDlg::OnGsmFlush()
{
    // TODO: Add your command handler code here

```

```

        if(AfxMessageBox("This will delete all old sms in the sim card.
Proceed?",MB_YESNO)==IDYES)
        {
            CString str;
            str.Format("AT+CMGD=1,3%c",0x0D);
            m_commctrl.SendCString(str,nComPorts-1);

            m_wndTaskbarNotifier.Show("SIM Card\nSMS Buffer Cleared");
        }
        else
            AfxMessageBox("Operation aborted");
    }

```

```

void CSubStationMonitorDlg::OnSimMem() //check sim card sms buffer
{
    // TODO: Add your command handler code here
    CString str;
    str.Format("AT+CPMS?%c",0x0D);
    m_commctrl.SendCString(str,nComPorts-1);
    //delete rec_read...prompt for read the rec_unread..ok
}

```

```

void CSubStationMonitorDlg::SendAlertViaSMS(CString strNo, CString
strMsg)
{
    bSendSMS = true;
    CString str1,str2;
    str1.Format("at+cmgs=%c%s%c",34,strNo,34);
    str1 += 0x0d;

    m_BtnMonMode.SetIcon(IDI_GSMBUSY);
    m_commctrl.SendCString(str1,nComPorts-1); // set port

    //CString str1,str2;
    str1.Format("%s%c",strMsg,26); // 26 = CTRL_Z
    str1 += 0x0d;
    m_BtnMonMode.SetIcon(IDI_GSMBUSY);
    m_commctrl.SendCString(str1,nComPorts-1); // set port
}

```

```

void CSubStationMonitorDlg::DisplaySignalQualityMeter(int nQual)
{
    int nScale = nQual;
    if(nScale==0)
    {
        m_BtnMonMode.SetIcon(IDI_GSM0);
    }
    if((nScale>0)&& (nScale<=5))
    {

```

```

        m_BtnMonMode.SetIcon(IDI_GSM1);
    }
    if((nScale>5)&& (nScale<=9))
    {
        m_BtnMonMode.SetIcon(IDI_GSM2);
    }
    if((nScale>9)&& (nScale<=14))
    {
        m_BtnMonMode.SetIcon(IDI_GSM3);
    }
    if((nScale>14)&& (nScale<=25))
    {
        m_BtnMonMode.SetIcon(IDI_GSM4);
    }
    if((nScale>25)&& (nScale<=98))
    {
        m_BtnMonMode.SetIcon(IDI_GSM5);
    }
}

void CSubStationMonitorDlg::SendToGSModem(CString strSms)
{
    int nLength = strSms.GetLength();
    char *m;
    m = new char[nLength];
    // = strTmp;
    for(int i=0;i<nLength;i++)
    {
        m[i] = strSms.GetAt(i);
    }
    m_SerialComm.WriteToPort(m);
    delete []m;
}

void CSubStationMonitorDlg::OnGSMFilter(CString strMessage)
{
    m_BtnMonMode.SetIcon(IDI_GSMBUSY);
    int nOkLength = strMessage.GetLength();

    CString strGoodMessage;
    strGoodMessage = strMessage;
    int nStartOK = strGoodMessage.Find('O',0);
    CString strOK;

    if(nStartOK>=0)//look for word OK....but check ok alone or ok
with message
    {
        strOK = strGoodMessage.GetAt(nStartOK);
        strOK += strGoodMessage.GetAt(nStartOK+1);
        if(strOK=="OK")
        {
            if(m_bPing)
            {
                m_bPing= false;
                DisplaySignalQualityMeter(nSignalQualityIndex);
                return;
            }
        }
    }
}

```

```

    }
    if(bSendSMS)
    {
        m_wndTaskbarNotifier.Show("Message Sent");
        bSendSMS= false;
        CString str;
        m_bPing = true;
        str.Format("AT%c",0x0D);
        DisplaySignalQualityMeter(nSignalQualityIndex);
        m_commctrl.SendCString(str,nComPorts-1);//ping
modem to clear connection

        return;
    }
    if(nOkLength<=4)
    {
        DisplaySignalQualityMeter(nSignalQualityIndex);
        CString str;
        m_bPing = true;
        str.Format("AT%c",0x0D);
        m_commctrl.SendCString(str,nComPorts-1);//ping
modem to clear connection
        return;
    }
}
}
if(bSMSList)
{
}

OnGSMNotify(strGoodMessage);
}

```