3-25-1996

Development of a security network (SECNET) based on integrated services digital network (ISDN)

Isidro Alvarez
Florida International University

FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

DEVELOPMENT OF A SECURITY NETWORK (SECNET) BASED ON INTEGRATED SERVICES DIGITAL NETWORK (ISDN).

A thesis submitted in partial satisfaction of the requirements for the degree of

MASTER OF SCIENCE

IN

COMPUTER ENGINEERING

by

Isidro Alvarez

1996

Thesis Committee Approval Sheet

To: Dean Gordon R. Hopkins
College of Engineering and Design

This thesis, written by ISIDRO ALVAREZ, and entitled, DEVELOPMENT OF A SECURITY NETWORK (SECNET) BASED ON INTEGRATED SERVICES DIGITAL NETWORK (ISDN), having been approved in respect to style and intellectual content, is referred to you for judgment.

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

Gustavo Roig

John C. Comfort /James R. Story

Subbarao V. Wunnava, Major Professor

Date of Defense: March 25, 1996

The thesis of Isidro Alvarez is approved.

Dean Gordon R. Hopkins
College of Engineering and Design

Dr. Richard L. Campbell
Dean of Graduate Studies

Florida International University, 1996

I dedicate this thesis to my to my mother Virginia Avalos Cabrera, to my father Isidro Alvarez Lopez, and to my sister Nieves A. Rodriguez. Without their support and love, I would not be what I am today.

ACKNOWLEDGMENTS

I would like to thank all my committee members: Dr. Subbarao V. Wunnava, Dr. Gustavo Roig, and Dr. John C. Comfort for their helpful comments, encouragement, and support. I also appreciate the support provided by Bellsouth, NORTEL, and Adtran Corporation.

I would like to thank especially Dr. Gustavo Roig because without his help I would not be doing the present presentation.

I would also like to thank the following persons that worked with me in the ISDN Lab at FIU: Mark W. Williams, Tom Gilbar, Pablo Perez, Carol Levay-Reyes, Hamid Ghassemi, Irma Becerra Fernandez, Margaret Dabdoub, Venkata K. Gandham, Miguel Rosario, Alvio Barrios, and Rick Cabrera for their support in one way or the other. I also wish to thank every one in the FIU Electrical & Computer Engineering Department for their support especially Dr. James R. Story, Mike Urucinitz, Pat Brammer, and Marbeth Cochran.

ABSTRACT OF THE THESIS

DEVELOPMENT OF A SECURITY NETWORK (SECNET) BASED ON INTEGRATED SERVICES DIGITAL NETWORK (ISDN)

by

Isidro Alvarez

Florida International University, 1996

Miami, Florida

Professor Subbarao V. Wunnava, Major Professor.

The progress in the computing and communication industries together with the fast evolution of the semiconductor industry has made possible advances in the communications field. These advances have been used by other related applications to improve the services that they bring about. On the other hand, business crimes have increased three digits orders of magnitude in one decade, making from 20% to 30% of small businesses fail. These conditions demand new solutions to make security systems more reliable and efficient.

The present work combines ISDN as a network with a security system to create a security network (SECNET). It will create intelligent and distributed security devices that communicate information from different places to a main security office by using the ISDN lines available at the premises. This work also introduces a new idea of individual equipment protection.

TABLE OF FIGURES

TABLE OF TABLES

Chapter I. Introduction.

During the 1980's, business crime and its cost has grown very fast. The private security industry experienced the greatest growth ever. It is expected that business crime will increase 75.4 % from 1990 to 2000. Also, other studies indicated that economic crime is the reason for 20 to 30 percent of small business failure [1].

Security systems commercially available nowadays are system based in the detection of an intruder into a room. This is accomplished by detecting an unauthorized opening of a door or window in a room under control. Also, there are more sophisticated systems that use wired or wireless communication path to check the identity of the person or persons accessing a room by checking certain form of identification. For example: magnetic badges can hold information about the person to whom was it assigned; also, smart cards can receive and transmit information about the holder. A different kind of system uses closed-circuit TV (CCTV), where security personnel check every possible entrance using TV cameras strategically distributed through the areas under surveillance. For private use, commercial system available use phone lines to communicate with an office that is in charge of notifying the police and the owner of the place of any intrusion.

Considering the above statistics and the basic principles of commercially available systems, the idea of creating a security system that operates almost independently from human interaction was developed. This security system uses computers, data communication, and telecommunications.

This system was conceptualized as a network of multiple computers interconnected using a communication medium. It has been christened: SECNET (SECurity NETwork). The communication network that has been selected is the Integrated Services Digital Network (ISDN). The selection was made because this is a new technology that is changing the way in which the telephone communications are made from the user loop (last link between the users to the Central Offices, CO). ISDN is a fast, very reliable digital communication link that allows transmission of voice, data, facsimile, and video at more than twice the speed of the fastest modems with error rates many times smaller. Also, ISDN is an emerging technology that is able to naturally incorporate new and future advances in telecommunications and it is replacing the old analog lines at increasing rate in the present years. At the time when the study was done, no security system was using ISDN as medium of communication, although there were some articles that started discussing about the possible benefits of ISDN for commercially security networks.

1.1 Objectives.

The objectives of the present work were then set. SECNET should be a system that would be largely compatible with other security systems. However, it should introduce new ideas to make the system unique with respect to the commercially available security systems. It also will make use of the ISDN lines to accomplish communication among the computers connected to the security network. A prototype has been built to demonstrate that SECNET can be done.

SECNET is the backbone for future additions of new features, and new technologies. These additions can be incorporated to the system by updating and upgrading the software without requiring total changes in the hardware, making the system more flexible.

1.2 Organization.

The present work consists of seven chapters. Each chapter explains different topics discussed in this thesis. Chapter 2 introduces what a security system is and the way in which SECNET is going to be implemented. Chapter 3 explains ISDN in detail. Chapter 4 illustrates the hardware used in SECNET. Chapter 5 details SECNET's system software. Chapter 6 illustrates the hardware and software integration in a system and future enhancements that can be incorporated into SECNET. Chapter seven shows the conclusion of this work.

Chapter II. Security Systems.

Economic or business crime continues to grow dramatically. As stated in Chapter I, business crime and its cost have experienced steady growth since the 1970's. Computer related crimes, stealing of valuable equipment, and other crimes are areas where the business crimes have been increased.

Studies in the 1970's by the United State Chamber of Commerce, Congress, and the American Management Association were in agreement that business crime costs $40 billion or more a year [1]. Following those studies, a projection of what business crime can cost up to year 2000 is shown in Figure 2.1 [1].

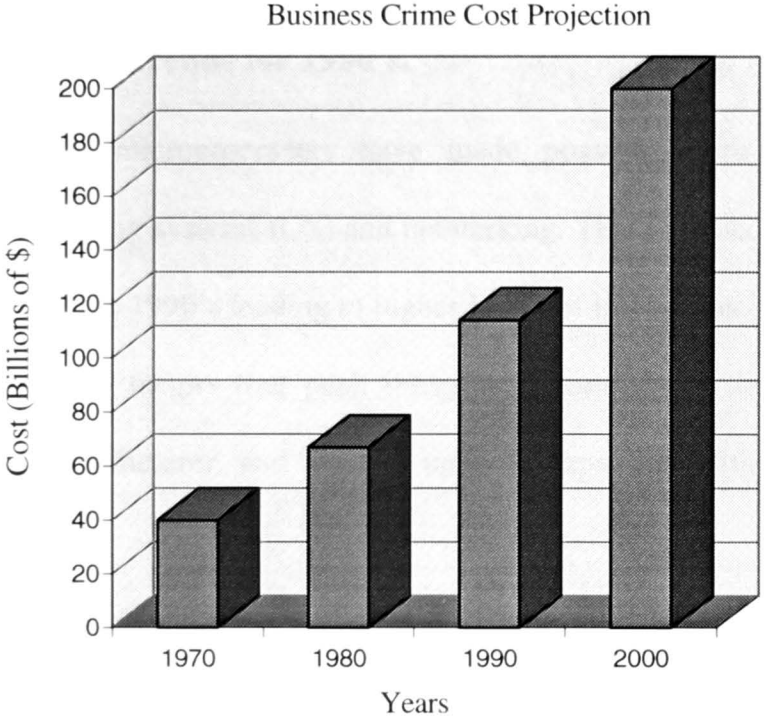


Figure 2.1 Business crime cost projection up to year 2000.

The impact of these costs reduces profits and increases the cost of insurance, security and internal control. The trend is to design proactive programs and controls to prevent losses [1].

There will be more automated systems doing complicated security tasks better than before. By decentralizing security, automating, and adding equipment where possible, companies can reduce the expenditure on security by upgrading their equipment and reducing security personnel [2].

Computers are used increasingly in security. They are used to help with operations, recordkeeping and decision making. Their use grew 9% between 1988 and 1989 alone [3].

2.1 Security System Trends for 1990's.

Advances in microprocessors have made possible more powerful personal computer (PC) operating systems (OS) and networking. These advances have set the pace for new products in the 1990's leading to higher levels of integration.

There are two factors that push integration: users want the best products, no matter who the manufacturer, and want to upgrade capability without discarding their original investment.

The best approach is the development of large distributed networks that reduce operating costs, then centralizing administration and security control for multiple sites, and downsize of integration to run on PC-sized systems. The trend on the 1990's will be

systems that integrate card access, alarm monitoring, CCTV control, biometrics, and administrative tasks [3].

Based on those predictions, SECNET should be a network system based on ISDN that allows administrative tasks and alarm monitoring. Also, it should be able to be upgrade without having to replace the hardware already in use.

2.2 SECNET General Concept.

SECNET is a network security system that connects a main security office (MSO) to a number of offices that have to be protected. The main link between the MSO and the offices is based on ISDN. ISDN lines installed in each of the offices and the MSO perform normal everyday communication. They can also be used to link together all the offices, including the MSO under security control. This is represented in Figure 2.2.1.

A number of offices under security control are connected to the MSO through available ISDN lines in each of the offices. Computers which can be doing different tasks are also running a software that allows them to interact with control units. The latter are in charge of supervising the equipment under security control. Based on this general idea, the SECNET concept was developed.

The MSO will collect all the information that each office will send to it. The number of offices that can be connected does not depend on of physical cabling limitations, nor is it required to have a certain number of dedicated lines to implement security function. They will use ISDN lines in circuit switched mode (CSM). In this mode, offices will use ISDN lines only on demand.

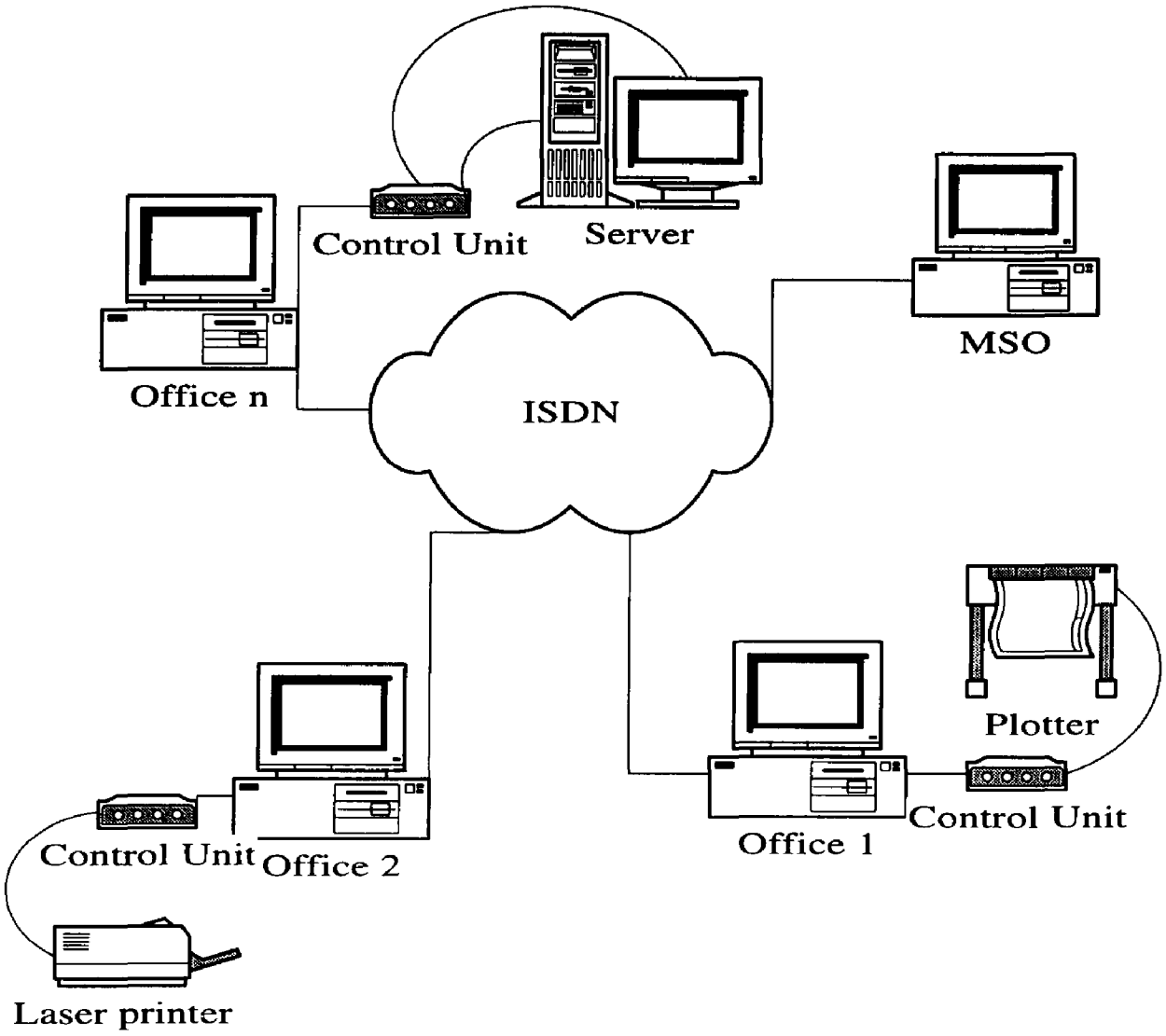


Figure 2.2.1 General view of SECNET.

In fact only if the control units detect an equipment failure. ISDN will be studied in much detail in the next chapter. However, it is important to point out that the advantages of ISDN over normal plain old telephone (POT) lines, ISDN makes connections in the order of milliseconds, it has more bandwidth and this bandwidth can be incremented on demand using statistical inverse multiplexing which is the junction of

the B channels together to form a channel with higher bandwidth. These characteristics make ISDN suitable to be used as a communication path in a security network. Also, a system based on ISDN is prepared to handle future upgrades such as sending still pictures full motion video, or audio.

SECNET, also, has to be designed to be as modular as possible to accommodate future upgrades without having to change drastically the system. To accomplish this, the hardware be used must be sufficiently flexible so that the upgrades can be made by changing only the software, which is of course much easier to upgrade than the hardware. The hardware should also be modular to facilitate incremental upgrades.

In addition to all of the above, a new direction in security systems will be taken: basically all the commercially available security systems control access points, identify the person who access the room by his identification card, and use cameras to monitor the rooms under protection. They can also control a number of equipment connected to a same point without any possible individual identification of them. SECNET will be designed using a different approach: it will be able to identify the location, and what kind of device is in that location. This approach makes SECNET a unique security network system. To the author's knowledge, there is no similar system on the market, today.

2.2 SECNET Design Considerations.

SECNET has been designed as a composite system consisting of a PC and a control unit. The former is in charge of administrative tasks such as maintaining the equipment's database, which will keep track of the problem that each of the equipment

has. It will establish communication to the MSO using ISDN lines and a Terminal Adapter (TA) connected to one of its serial ports. Also, it will communicate with the control unit to send and receive commands to make the system operational. It will perform the user/control unit interaction in a user friendly manner.

The control unit will be in charge of monitoring each unit of the equipment under control. The computer and the control unit interchange commands to add, enable, disable, and remove equipment. If something is wrong with one or more equipment the control unit will send an indication to the computer indicating the different equipment locations with problems.

These are the main specification for SECNET. In the following chapters its design is explained in detail.

Chapter III. Integrated Services Digital Network (ISDN).

3.1 Introduction

The public telephone and the telecommunication networks have gone through revolutionary changes in recent years. These changes have been driven by two fundamental factors: the broadening nature of user requirements and the technological changes that came about due to evolution in the microelectronics industry.

User requirements have increased from communications where only voice transmission was desired to include communication of text, graphics, facsimile, audio, and video. Commercial and residential users depend more and more on functions such as document transmission, data storage and retrieval, electronic mail, teleconferencing, and computer aided design. Along with these requirements, the use of all available communication options must be very flexible, simple and efficient, allowing higher levels of integration and standardization.

Advances in the microelectronics industry have made possible the digitization of the telecommunications. Moreover, contemporary high speed digital circuits make possible very high speed communications, very low noise in the transmission lines, less equipment required to perform the communication, and more efficient use of the bandwidth allocated for communications.

ISDN is a set of services provided by means of a limited set of standardized interfaces [4][5]. It supports services such as voice and data transmissions, using multipurpose user-to-network interfaces.

3.2 Evolution of ISDN.

A communication network has two main elements: transmission facilities and switching facilities. The former allows the transmission of signals between two locations. The latter links transmission facilities into a logical and well-structured network [5].

The telephone network was designed and developed based on analog devices. Analog transmission technology has been constantly expanded since 1900 to 1980. Bell Communication Laboratories (Bellcore) developed technologies like the analog carrier [5] to allow many different voice channels to be transmitted on the same pair of wires. This is accomplished by modulating each channel onto different frequencies. This technique is called Frequency Division Multiplexing (FDM). The simplest of these is a group of 12 channels, providing 48 kHz of bandwidth each. These systems made long-distance telephony affordable and popular [5]. However, the big drawback in this approach is that it tends to get noisier with increasing distance [5]. This is a problem very difficult to solve, because of the own nature of analog signal transmission.

The analog signal has to be refreshed after traveling a fixed distance. Every component involved in the communication link introduces noise into the connection, which also proportionally increases with the distance because more equipment is needed to form the link.

The last three decades have been very important in the evolution of telecommunications and public telephone networks. The development of digital signal processing and its introduction into the transmission technology took place in the beginning of the 1960's. The first digital transmission system was born: the T1 carrier.

This system originally was designed to carry voice by digitizing the voice in devices known as digital channel banks (D) by using Pulse Code Modulation (PCM) technique.

The T1 digital transmission system (it is also called Digital System 1, DS1) split the line into 24/8 bits channels sampling at a rate of 8000 times per second. The data bit rate for the T1 carrier is 1544 kbps (including bits for framing), and the signaling and control information is sent inband (data and control signal travel together). The development of T1 initiates the process of converting traditional analog transmission into digital transmission. The benefits of digital transmission are considerable: higher quality and reliability, lower power consumption, and lower cost. These benefits make the digital transmission system as the best transmission method in present days. The end of the 1970's probably marks the end of the analog transmission era [5].

Switching systems were also designed based on analog technology, using the inband signaling to carry control information to establish the calls. All telephone switches were analog up to the 1960's, when they started to be replaced by digital units. Voice was converted to digital streams for internal switching purposes. These systems are called Digital Private Exchanges (PBX). They were followed by Central Office (CO) switches; all were interconnected using T1 transmission systems. The reasons for the changes are that digital switching systems were cheaper to build, to install, and to maintain than analog ones. Even though both main systems were digital, all the signaling was done in analog fashion and inband.

ISDN is today's solution to a full and end-to-end digital network, from customer premises to the network system. It will help to achieve a maximum operational potential

for both transmission and switching digital systems. At the same time ISDN brings the potential and flexibility of the digital network to the user. ISDN integrates well circuit switching and packet switching. Its success is the integration of voice and data into a unique system. For many kinds of data transmission and voice applications, ISDN is the most appropriate and economical technology [5].

3.3 Integrated Service Digital Network (ISDN) Concept.

ISDN is defined by standards. These standards allow equipment portability among different fabricates and national Post, Telephone, and Telegraph (PTT) companies worldwide. From the very beginning, the International Telephone and Telegraph Consultative Committee (CCITT), more recently renamed as International Telecommunication Union (ITU), took the important role of defining a set of recommendations issued by different Study Groups (SG). These groups are specialized in certain topics like architecture (SG-XVIII), signaling protocol (SG-XI), and proposed new services (SG-I). Every four year, the ITU holds a plenary assembly where national delegations officially approve the work done by different study groups. The ground basis of the ITU is national standard committees. They allow many organizations, including manufacturers and users, to have indirect access to ITU, and at the same time a direct input to national standards [5].

ISDN basic structure is two channels with a data bit rate of 64 kbps (called Bearer or B channels) and a single channel with a data bit rate of 16 kbps, (called the D channel). They will be explained subsequently in this chapter.

ISDN integrates both circuit and packet switching. The most distinguishing characteristic of circuit switching is that provides a fixed amount of bandwidth (either analog or digital) for the full duration of the call. The main characteristics of data communications is traffic bursts, which makes Circuit Switched Data (CSD) an inefficient way to allocate bandwidth for this type of usage. On the other hand, Packet Switched Data (PSD) offers a better solution allowing costly high speed transmission facilities to be shared by multiple users using more bandwidth only when necessary.

Packet switching depends on the interaction of protocols. Some of the protocols are performed by the users' applications and are known as end-to-end protocols. Other protocols operate at between the user and the network as a whole, and require the coordinate actions of the network nodes at both ends of the connection. They are called edge-to-edge protocols. Still other protocols operate independently across individual links between the users and networks, they are known as hop-by-hop protocols [5]. This protocol architecture was developed using the Open System Interconnection (OSI) model [6].

The OSI model was develop to coordinate the standard development of system interconnection, while allowing existing standards to fit into the overall model. Figure 3.3.1 shows the OSI Reference Model and the name of each of its layers. These layers perform a subset of the functions required to communicate with other systems. Each layer provides services to the next higher layer; at the same time, it relies on the next lower layer to perform more elementary functions. Each layer should be independent of the other layers in the sense that changes implemented in one layer do not require that

changes be made to adjacent layers. This allows a certain degree of freedom among the different layers. This makes new models simple to operate, and at the same time, flexible enough to accommodate existing protocols into this layered system.

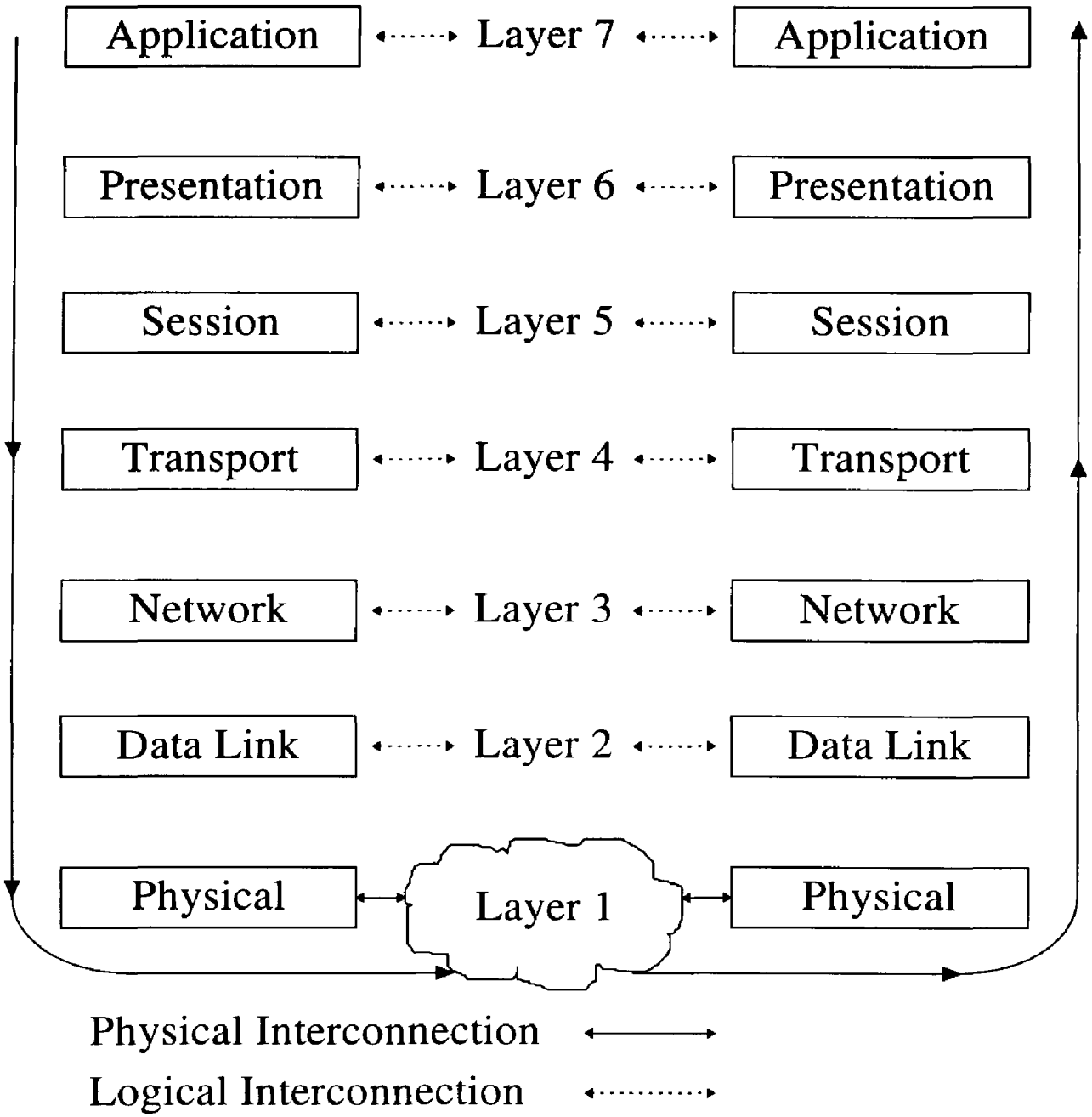


Figure 3.3.1 Open System Interconnection (OSI) model [6].

The layers as represented in Figure 3.3.1 are [7]:

- Layer 7: The Application Layer. Task to be performed (for example, file transfer, airline booking, messaging handling).
- Layer 6: The Presentation Layer. It establishes the common format used between terminals, using common rules to representing data.
- Layer 5: The Session Layer. This defines the way in which applications running at the two ends of the link intercommunicate, including initiation and termination of session and coordinating their activities during the session.
- Layer 4: The Transport Layer. It is the terminal-to-terminal layer. Data may be carried across the networks using different forms of Layer 1, 2 and 3 (i.e., via Local Area Network, LAN, and ISDN).
- Layer 3: The Network Layer. This ensures that messages are routed to the appropriate destinations, and provides a mechanism to ensure correct control and acknowledgment of messages.
- Layer 2: The Link Layer. It provides the discipline for assembling digits, error correction and detection by using frames. The format used in this layer is derived from a standard known as the High Level Data Link Control (HDLC).
- Layer 1: The Physical Layer: This defines the characteristics of the signal to be transferred over the physical media. It covers such things as pulse amplitude, line coding, transmission rates, connectors and anything else needed to transfer digits satisfactorily.

The lower three layers are called network service layers. These layers are defined in two planes: the control plane and the user plane. The latter uses service-specific protocols. The former is where ISDN signaling takes place [5]. As a network, ISDN is not concerned with layers four to seven. Layer 1 (the Physical Layer) is defined in recommendation I.430 and I.431. These recommendations specify the physical interface for both basic and primary accesses, and, since the B and D channels are multiplexed over the same interface, these standards apply to both types of channels. The protocol structure differs for the two channels in layers 2 and 3. Table 3.3.1 shows the protocol structure and its relationship with the OSI model. A new data link layer standard, Link Access Protocol D channel (LAPD) has been defined. This protocol is based on an HDLC modified protocol to meet the ISDN requirements.

<table>
<tr><td rowspan="8">Network

Data Link
Physical</td><td>Call</td><td>X.25</td><td>Further</td><td colspan="2"></td><td>X.25</td></tr>
<tr><td>Control</td><td>Packet</td><td>Study</td><td colspan="2"></td><td>Packet</td></tr>
<tr><td>I.451/Q.931</td><td>Level</td><td></td><td colspan="2"></td><td>Level</td></tr>
<tr><td colspan="3">LAPD (I.441/Q.921)</td><td colspan="2">Frame Relay</td><td>LAPB</td></tr>
<tr><td colspan="6">I.430 Basic Interface + I.431 Primary Interface</td></tr>
<tr><td>Signal</td><td>Packet</td><td>Telemetry</td><td>Circuit</td><td>Semiper_</td><td>Packet</td></tr>
<tr><td></td><td></td><td></td><td>Switched</td><td>manent</td><td>Switched</td></tr>
<tr><td colspan="3">D Channel</td><td colspan="3">B Channel</td></tr>
</table>

Table 3.3.1 ISDN Protocol Architecture (User-Network Interface Control Plane) [5][6].

Transmission on channel D is as LAPD frames exchanged between subscriber equipment and an ISDN switching element. These applications are supported (as shown in Table 3.3.1): control-signaling, packet switching, and telemetry [6]. Moreover, new standards have been defined to accommodate protocols used for Frame Relay (changing its name to LAPF).

For control signaling, a call control protocol has been defined: the I.451/Q.931. This protocol is used to establish, maintain, and terminate connections on the B channels. This is an advance with respect to normal telephone lines communications. The call control for the communication is not any longer inband; in ISDN is out of band. For packet switching services, the X.25 level 3 protocol is used to establish virtual circuits on the D channel to other users, and to exchange packetized data [6].

B channels can be used for circuit switching, semipermanent circuits and packet switching. In the case of circuit switching, a circuit is set up on demand on a B channel using the D channel to control the whole process. A semipermanent circuit is a B channel circuit that is set up by prior agreement between connected users in the network; it is the equivalent of dedicated lines in an analog environment. Both connections provide a transparent data path between end systems.

A packet-switched connection is set up on a B-channel between the user and a packet-switched node using also the call control protocol provided by the D channel. It uses X.25 protocol (layers 2 and 3) to establish the virtual circuit. Also as an alternative, ITU has recently defined frame relay services to be used on the B channels.

The usual diagram for describing these sets of protocols is shown in Figure 3.3.2. The two layer stacks used in the figure are the user plane, which match with the OSI model, and the control plane that is used for signaling. These two planes are the difference that differentiates the ISDN network from the general OSI model, although both planes follows the general objectives of the model. User information is carried through each layer, only changing its physical form (Layer 1) at intermediate points such

as NT1 or local exchanges. The signaling has only three defined layers 2 and 3, which are terminated in the exchange. They carry on the signaling to establish the communication, to make possible user-to-user applications.

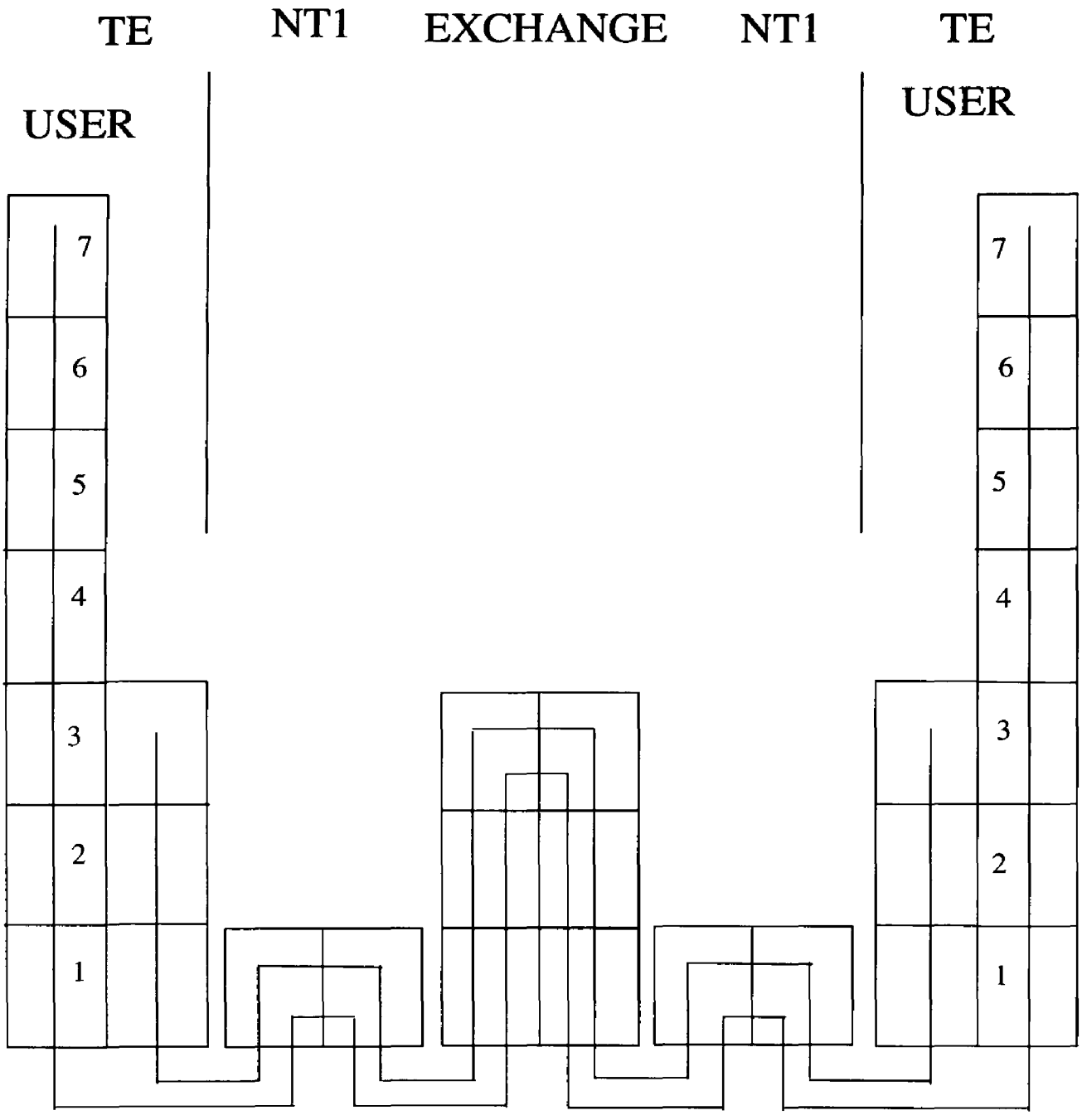


Figure 3.3.2 User and control planes used in ISDN [4][6].

3.4 ISDN Transmission Structure and Interface.

The selection of the rates at which the information is carried across the user-network interface is influenced by four important rate considerations [4]:

- Present and future end-user applications needs.
- Channel capacity and limitations imposed by the subscriber-access network and the digital transmission systems.
- Available transmission rates on the interchange network circuits.
- Cost and complexity of the interface.

The existing infrastructure of twisted pairs in the local-access plant is the most important communication access media that users have to access the network. Twisted pairs would be the main access media for ISDN for some time. Modern digital transmission technology allows twisted pairs to support data rates of up to several hundred of kbps, over distances of 10 miles, without repeating. Larger data rates usually require repeating every 1 to 2 miles. Broadband can only be achieved over higher bandwidth media such as fiber optic cables.

The main limitations on data rates in the interchange network are the result of a standard developed by the European Conference of Posts and Telecommunications Administrations (CEPT), ITU, and the American Telephone & Telegraph (AT&T) [4].

The basic rate is specified as multiples of 64 kbps; however, from this point up there are big differences. In North America, the existing multiplex structure is based on the primary multiplex format DS1 (T1). It combines 24 channels (at 64 kbps each) together with certain control information into a first-level multiplex arrangement

operating at transmission rates of 1.544 Mbps. On the other hand, CEPT standards are based on a multiplex arrangement of 32 channels (at 64 kbps each) for the first level transmission rate of 2.048 Mbps or multiplex format E1.

The channel structure is the total information carrying capacity of the interface. It is a constant value but depends on whether the interface supports basic, primary, or broadband access.

These channel structures are divided into one or more component channels, each with a specified data rate for independent portions of the total capacity. The entire channel is then transmitted synchronously over a single physical medium across the S, T or U references points. Figure 3.4.1 represents the ISDN user-to-network interface.

From figure 3.4.1, the reference points are defined as separate functional groups, and each of the boxes represents an arrangement of physical. Network Termination 1 (NT1) is the equipment that includes functions related to OSI layer 1. Network Termination 2 or NT2 is an intelligent device that includes OSI layers 2 and 3. It can perform switching and concentration functions (among them a digital PBX), a terminal controller or, Local Area Network (LAN) devices. The NT2/NT1 is equipment that can perform OSI functions for layer 1, 2 and 3. Terminal Equipment (TE1) refers to equipment that supports standard ISDN interface. The Terminal Adapter (TA) is a device that interfaces with non-ISDN equipment with the ISDN interface. Terminal Equipment 2 (TE2) are equipment that can not interface directly with the ISDN interface[5].

International standard ISDN is based upon 64 kbps circuit-switched channels that for a foreseeable future will be a standard for telephone purposes. Analogue signals are

band limited to 3.4 kHz and sampled at 8000 times per second. This sampling rate arises from the process related to sideband frequencies. These frequencies are integer multiples of 8 kHz. If the sampling rate is less than twice the bandwidth of the analogue signal, then the sideband will overlap (known as "aliasing") and subsequent decoding is no longer possible (Nyquist's Theorem). The amplitude samples are then measured and encoded into a binary number. The precision of the encoding process depends on the number of bits in that binary number. Empirically, it was found that to give an adequate speech quality 12 bits were needed in this encoding process. That is ± 2048 levels that could be identified by 2^{12} values or 12 bits. A non-linear compression function is introduced to translate the 12-bits code into an 8-bit code. Figure 3.4.2 shows diagrammatically how the uncompressed code could be mapped on to a compressed code for both polarities of signal.

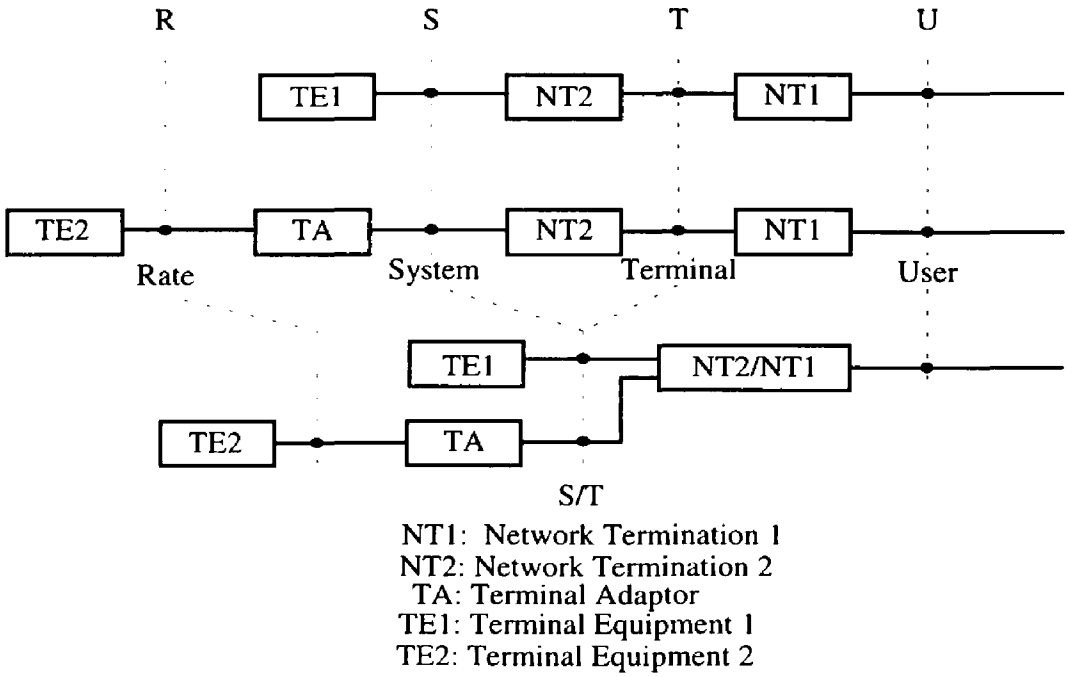


Figure 3.4.1 ISDN reference points and functional groups [6].

Problems arise for very low level signals, as the logarithm of very small numbers becomes negative. Somehow the compression law has to be forced to pass through the coordinates origin. There are two solutions for these problems. In North America, the two logarithmic curves were displaced towards the central vertical axis giving a transfer function of the form $y \propto (1+\mu \cdot x)$. This is known as "μ-Law." In Europe, a line was drawn tangentially to the two curves and hence, by symmetry, it passes through the origin. This curve is of the form $y \propto (1+A \cdot x)$ on the center part of the range, and $y \propto A \cdot x$ at the extremes of the range. This is known as "A-Law." These two compression laws is one of the main differences in the standard for ISDN.

By the above explained laws, Pulse Code Modulation (PCM) standards were chosen so that the transmission system could be embedded anywhere in an analog network. In ISDN, speech encoding and decoding are always performed at the user's TE, and the only level of variation is because of speaker differences, not network differences.

Component channels are integrated into a synchronous channel structure. This is accomplished using Synchronous Time Division Multiplexing (S-TDM). The channel structure consists of a periodic and continuous sequence of frames separated by framing channels and synchronously transmitted across the S, T and U interface points. A component channel is the result of permanent assignment of one or more specific, not necessary contiguous, time slots in every frame. The resulting capacity is then determined by: duration of the frame and time slot, number of data bits carried in a time slot, and the number of time slots [4].

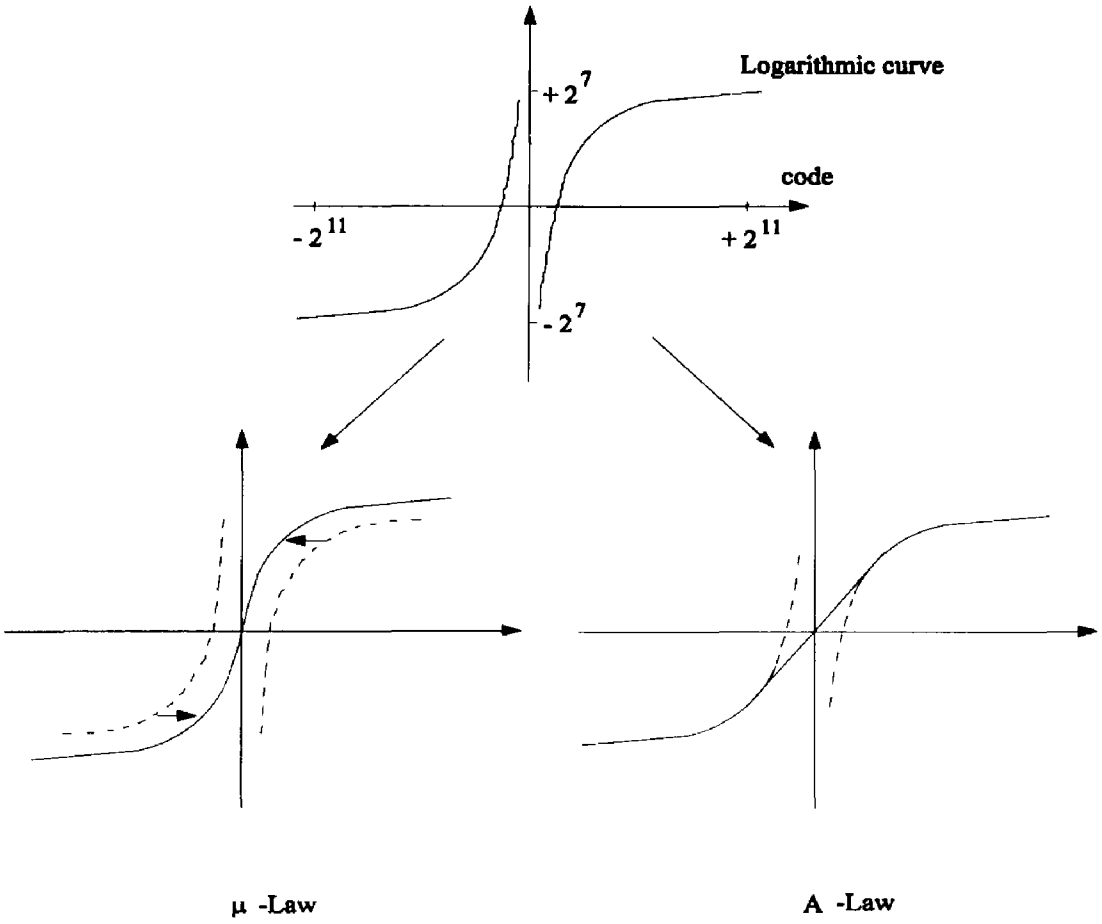


Figure 3.4.2. Logarithmic compression laws.

A typical frame has a duration of 125 μs divided into 32 times slots and carrying 256 bits of information. It yields a component channel capacity of m*64 kbps if m time slots are assigned to the channel. Typical values of m are 1, 6, 24 and 32. If m = 1, the channel is called a bearer channel, or B channel, that carries 64 kbps. It is also called DS0.

When m = 6, the data rate is 384 kbps (or six B channels). This kind of channel is used in specific applications, such as high resolution digital video and audio, for the distribution of television, teleconferencing, and surveillance, high speed file transfer, high resolution graphics and fast facsimile. This channel is also known as H0.

For m = 24, the channel is formed by 24 channels at 64 kbps plus 8 kbps of framing. That gives a 1.544 Mbps, which is called a DS1 or T1. If m = 32, the data rate is then 2.048 Mbps, called E1, which is the European standard. The latter two channels are considered Primary Rate Interface (PRI) in ISDN.

There is one more channel to mention: the D channel. The D channel can be found in the PRI frame where one of the 24 (DS1) or the 32 (E1) channels, is so designated. This channel is dedicated to control and signaling, and its data bit rate is 64 kbps. Also, it can be found in the Basic Rate Interface (BRI). It has a transfer rate of 16 kbps and is dedicated to control and signaling as well as packet-switched data transfers [4]. PRI and Broadband ISDN are beyond of the scope of this thesis.

The basic access channel structure is known as BRI. It consists of two 64 kbps B channels and one 16 kbps D channel. The aggregate data rate is 144 kbps. However, framing, synchronization, and other overhead brings the bit rate on the basic link to a total of 192 kbps. The B channels can be used either simultaneously in the same connection or independently on two connections. The former is achieved in two ways: Bonding or Point-to-Point Protocol (PPP). Bonding synchronizes both channels when a called is placed from the beginning of the call until the end. PPP uses the B channels on demand, and it is becoming the most used nowadays. Call control for both B channels is transmitted on the logically separate D channel. This creates an out-of-band signaling arrangement that allows the entire capacity of the B channels to be used for transmission of user information. Besides this, the D channel can be used to transmit low speed data

and control information between users. Figure 3.4.3 represents a BRI channel structure showing the 2B + D channels.

When out-of-band signaling is supported by separate and independent protocols, other protocols are implemented for connection control, network management, and user-to-user signaling. They may be defined with the evolution of the ISDN capabilities. They also can be designed to be highly flexible and sophisticated, to support and provide detailed call progress information, and other specialized connection-related services. In addition, they also support sophisticated network management capabilities during the life of the connection without interrupting the user transfer process.

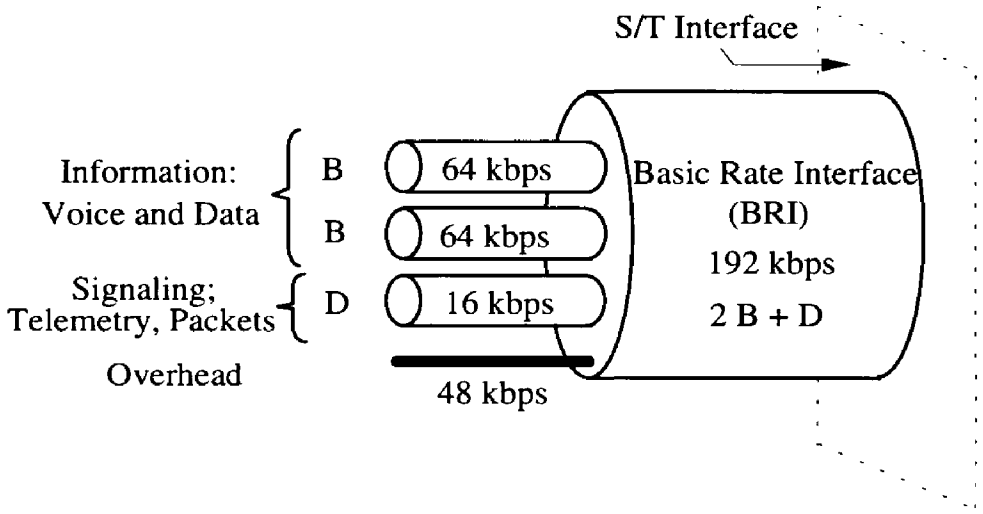


Figure 3.4.3 ISDN channel structure in the S and T interface points.

The BRI frame structure is represented in Figure 3.4.4. These are recurring frames of 48 bits with each frame being transmitted synchronously in 250 μs for an effective transmission rate of 4000 frames per second or 192 kbps. As previously stated, there is 144 kbps that corresponds to the two B channels and the D channel, the remaining 48 kbps are dedicated to layer 1 peer-to-peer protocol overhead and control information.

These frame structures are the same for point-to-point and for multipoint transmission. However they differ depending on the direction of transmission.

These 10 groups are individually DC balanced by appending to each group an L bit. Its logical value is chosen in such a way as to create a line signal with a zero average voltage over the balanced group. This is required by the fact that the transmit and receive circuits are transformer coupled to transmitters and receivers. F and F_A are used to provide frame synchronization, so that data contained in the frame may be properly demultiplexed at the receiver end. The activation and deactivation sequences from the network to the user are indicated by bit A.

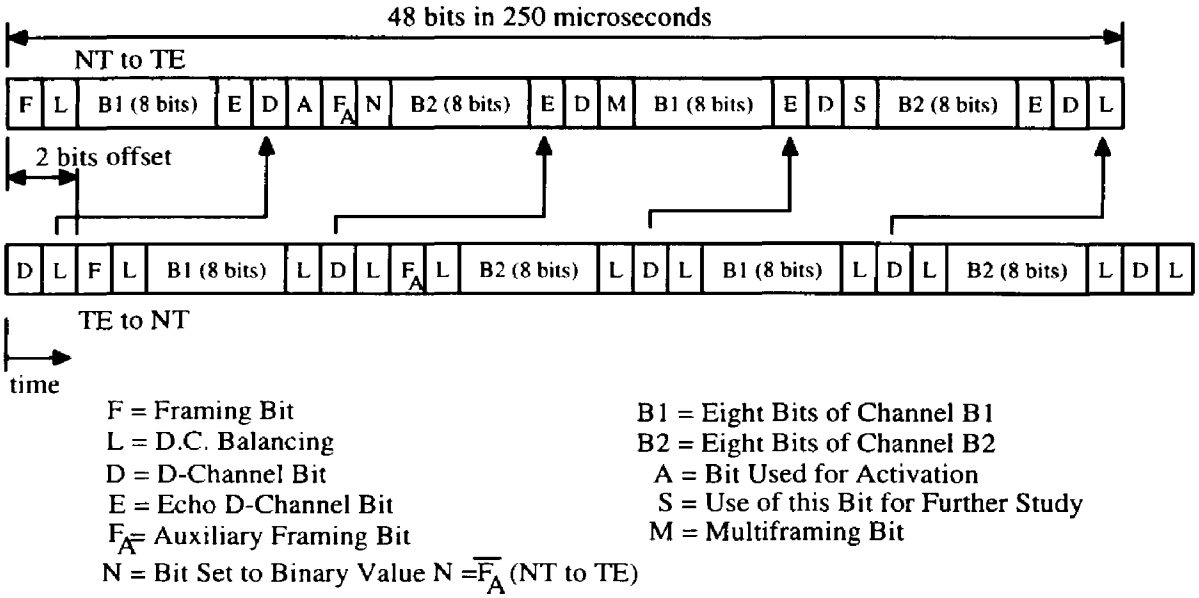


Figure 3.4.4 BRI frame structure at references points S and T [5][7].

Although the simplest way to transmit binary digits is to represent the "1" and "0" by two voltage levels, for linear transmission this process is rarely used for two main reasons:

- Maintenance Balance: By connecting a high-pass filter at the input and output of the generators, it allows power feeding along the same pair and separated from the signal. The high pass filter is a transformer, which also gives protection against power surges. If long strings of signals of the same polarity are transmitted then after passing through the high pass filter, they will be severely distorted. The simplest way of balancing the signal is transmit the logic 0 as 0V, and the logic one as alternately a positive and a negative signal as shown in Figure 3.4.5. This is known as bipolar pseudoternary or Alternate Mark Inversion (AMI). A modified way to send it is reversing the signal voltage levels by making the logical one as 0V and the logical zero as alternately positive and negative signals known as Modified AMI (MAMI). This technique is used mainly in the S/T reference points interface.

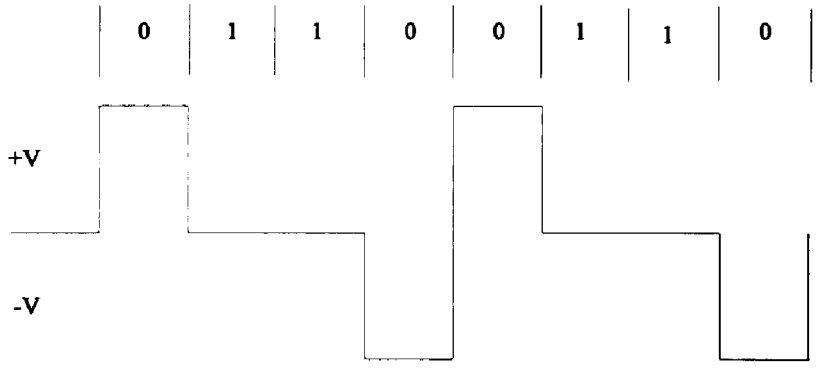


Figure 3.4.5. Modified AMI code for bit sequence 01100110.

Timing extraction: The regeneration process always involves retiming the signals and this is done by averaging the transitions over a considerable period, then using the extracted clock to retime subsequent pulses. AMI line code does not ensure this, in the case of a long string of zeros. In Europe, the modified code

is called High Density Bipolar 3 (HDB3). In North America, the modification of the AMI is called Bipolar with 8 Zeros Substitution (B8ZS). Both techniques are mainly used in PRI, although MAMI is also used in the S/T reference point for BRI.

The general frame structure for Layer 2 is shown in Figure 3.4.6. The frame has an integral number of octets, which are transmitted in an increasing order from top to bottom. Within each octet the bits are labeled from 1 to 8. Every frame contains at least

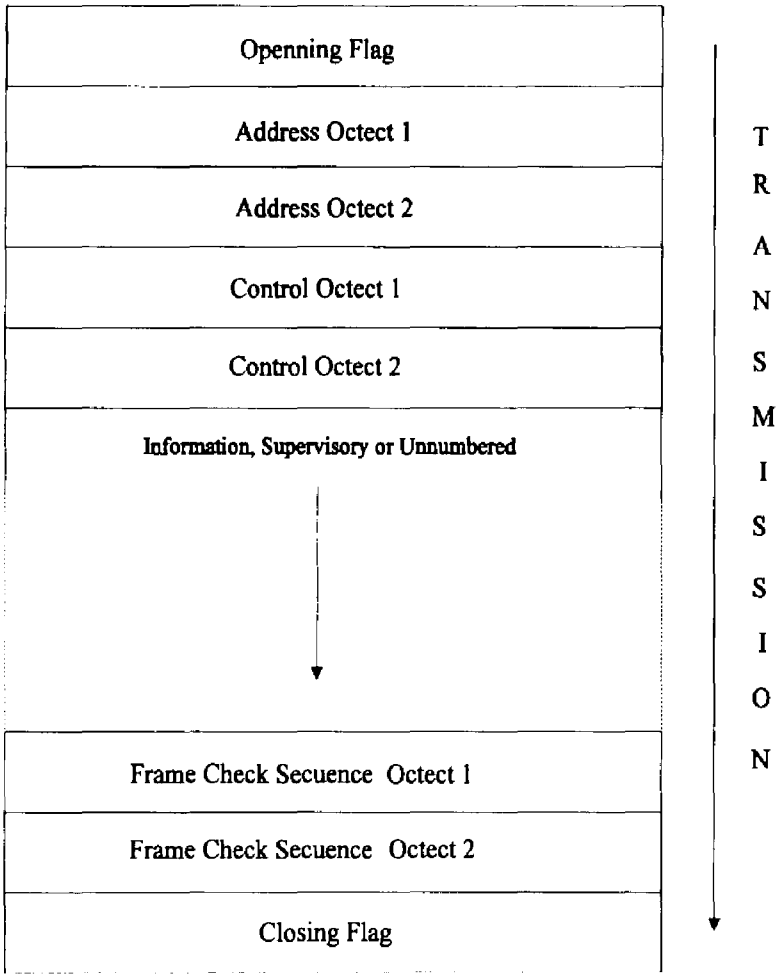
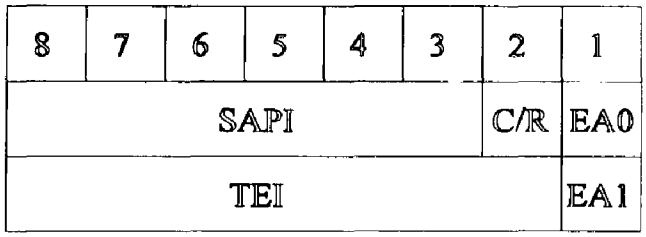


Figure 3.4.6 Layer 2 octets transmission.

five fields labeled: opening flag, address, control, frame check sequence (built in error correction and detection) and closing flag.

The first and last octets are unique markers to delimit the beginning and the end of the frame and they are symmetric bit patterns. They are the binary string 01111110 (Hexadecimal: 0x7F). The second and third octets are represented in Figure 3.4.7.



SAPI: Service Access Point Identifier
TEI : Terminal Equipment Identifier
C/R : Command/Response bit
EA0 : Extended Address 0
EA1 : Extended Address 1

Figure 3.4.7 Data link connection identifier octets [5].

They are the address fields that contain the Data Link Connection Identifier (DLCI). The DLCI is an identifier for a particular connection on which the frame is transmitted. The format of this field is shown in figure 3.4.7.

The second octet is the Service Access Point Identifier (SAPI) and the third octet contains the Terminal Equipment Identifier (TEI). The former refers to the peer data link layer identities that processes the data link layer frames. The latter is associated with the user's side of the network interface. It identifies a particular endpoint of connection within the TE for a point-to-point connection, or a group of endpoints within the same TE or different TE for a broadcast connection.

The U reference point is where the North American and European ISDN systems differ. In Europe, this point is defined to be as part of the PTT, while in North America this point is still part of the user interface. Its primary task for the BRI is the independent full duplex transmission across the U interface. User information, control information, and user-to-user signaling flow through this interface. All of this information is sent over the 2B+D channel structure. The data bit rate at this reference point is reduced to 160 kbps over a metallic twisted pair of wires. Digital transmission requires modulation of the information on to a suitable electrical signal. This is done to reduced crosstalk generated by the corresponding line signal and, also to narrow the power spectral density. ISDN uses 2 Binary 1 Quaternary (2B1Q) line code, which then reduces the data bit rate to 80 kbps. This code was initially proposed by the British Telecom Research Laboratories [8]. It associates pairs of binary digits with a single pulse chosen from four voltage levels. The pulses are known as quats, and they are symbolically represented by the alphabet: +3, +1, -1, -3. Figure 3.4.8. shows an example of this line coding for a sequence of bits.

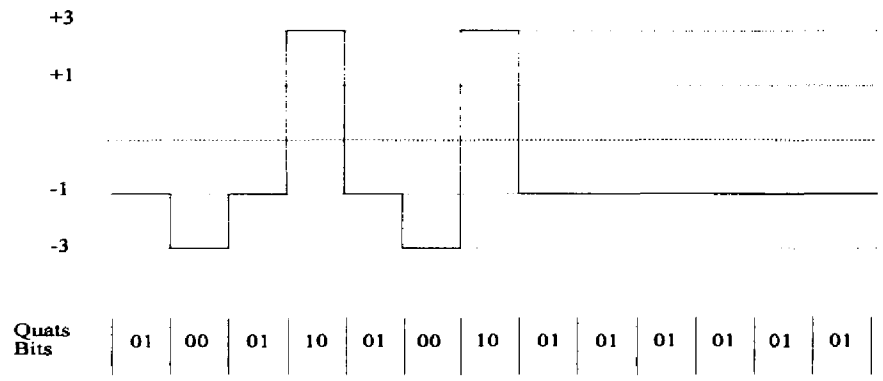


Figure 3.4.8 The 2B1Q code representation for letters: FIU.

Direct Current (DC) balance is achieved through scrambling of he raw bits. It requires the receivers to distinguish among four voltages. The line signaling rate of 160

kbps provides 16 kbps for timing and maintenance (2B + 1D = 144 kbps), in addition to multiplexing the 2B+D channel structure. Bell Communication Labs (Bellcore) defined this technique. It reflects the telephone company perspective. The Central Office (CO) could poll the NT1 for maintenance, put it into loopback mode, and perform testing functions. This standard is defined in the ANSI standard T1.601, and is used in the major deployments in North America. After applying the 2B1Q line coding format to the 160 kbps, it is reduced to 80 kbps.

Full duplex operation is required in digital loop transmission systems over a single twisted pair of wires. However, due to the complicated characteristic impedance of the cable, the transmitted signal is heavily distorted. The locally transmitted signal seriously interferes with the received signal. Methods are required to remove the local echo (and any other form impedance mismatch induced distortion from the received signal.

Three methods were developed. They are [7]:

- Frequency Division Duplex (FDD): The transmitter and the receiver operate at two different frequencies. It provides good tolerance to Near End CrossTalking (NECT); however, the tolerance to Far End CrossTalking (FECT), and noise in general is degraded because upper frequency channel suffers more attenuation.
- Time Division Duplex (TDD) or Time-Compression Multiplexing (TCM), also known as "burst-mode" or "ping-pong." Data are transmitted in one direction at the time, with transmission alternating between the two directions. It is very simple to realize; however, in order to accommodate full duplex transmission, the

bit rate during a burst is at least twice the required continuous rate; therefore, tolerance to crosstalk and noise is degraded. On shorter connections this mode offers a simple and effective means of duplex transmission.

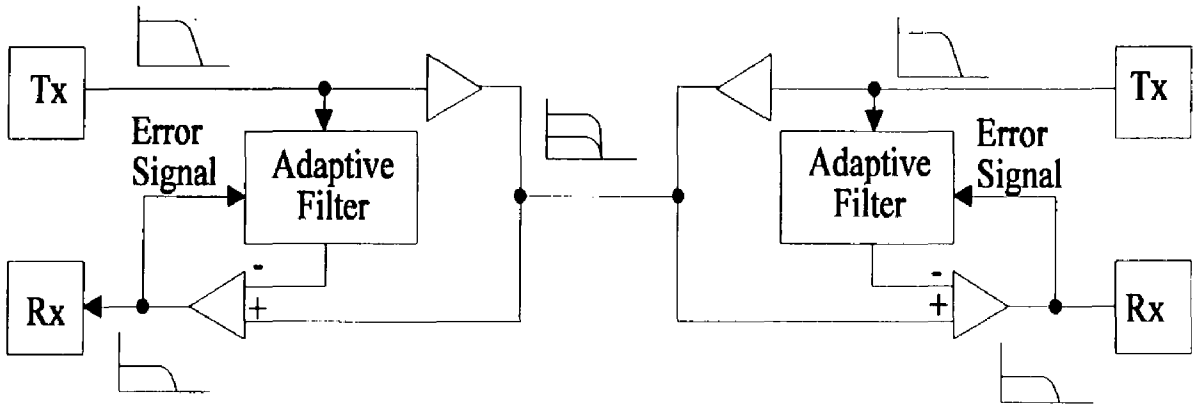


Figure 3.4.9. Echo cancellation [6].

- Echo Cancellation: This technique is known as adaptive hybrid technique and is represented in Figure 3.4.9. It removes the disadvantages of the above techniques. It involves adaptively forming a replica at the input of the receiver. The signals as shown in Figure 3.4.9. They can occupy the same frequency band and are continuous; therefore, the disadvantages of the previous techniques are avoided. It requires a more complex hardware implementation. However, modern Very Large Scale Integration (VLSI) technology enables this complexity to be realized at acceptable cost. At present, this technique offers the best solution t subscriber loops with longer reach (up to 18,000 feet). It is implemented in integrated circuits that perform Layer 1 functions, plus added digital signal processors which take care of the linear and non-linear echo cancellation.

3.5 ISDN Communications at FIU.

Basic rate access is intended to be the equivalent in ISDN terms, to simple telephone access, in analog terms. The connection of a multiplicity of analogue terminals (i.e., telephones, modems) is achieved by simply connecting them in parallel. The requirement for the ISDN interface is also to be able to simply connect terminals in parallel with the additional feature that terminals may be individually addressed by type or other identifier.

The two-wire transmission line from the local network is terminated by NT1. Although this NT1 is located on the customer's premises, it is on the network side of the ITU defined user-to-network interface and hence the responsibility of the network operator. The exception to this rule is in North America, where the user-to-network interface is the network side of the NT1. Figure 3.5.1 shows a representation of the network point-to-multipoint configuration.

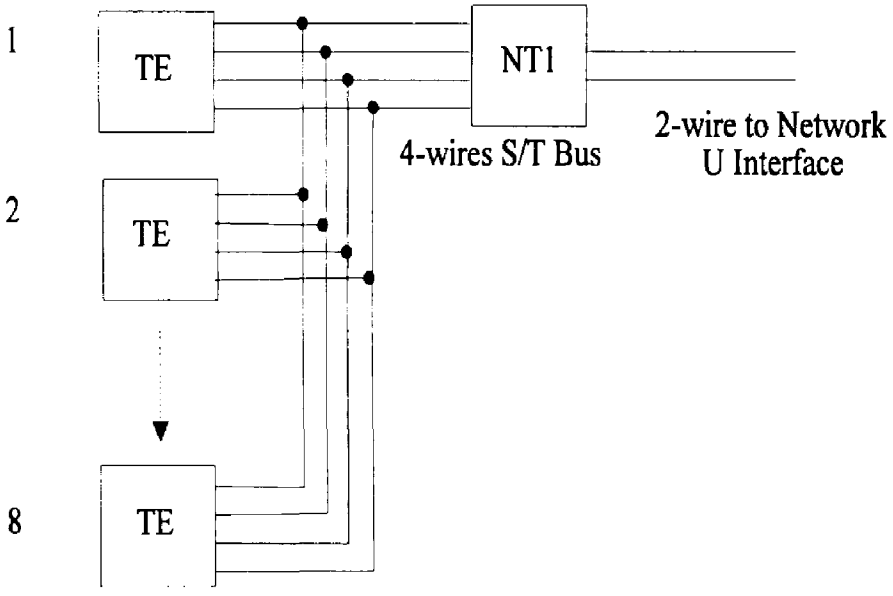


Figure 3.5.1 Point-to-multipoint configuration.

The NT1 converts from a two wire network to a four wire bus consisting in a transmit and receive pairs often known as S/T bus. This bus is designed to operate over a normal twisted pair internal cable as it may be provided for analog purposes. The only major difference is the type of connector used, which is an RJ-45 connector. This connector is an 8-pin connector defined in ISO specification 8877.

The mode of operations are:

- Point-to-Point Mode: Only one TE is connected at the end of up to a maximum of one km of cable. The actual limit is an attenuation of 96 dB at 96 kHz.
- Multipoint-to-Point: Up to eight terminals can be connected in parallel anywhere along the bus; however, the bus length is now limited to about 200 m by timing constraints. The terminal represents a high impedance (2500 Ω) on both the terminal and the NT1 allowing that the bus not be loaded, and the bus is terminated by 100 Ω resistors.

Over this bus passes the two transparent 64 kbps B channels and the 16 kbps D channel. However, all terminals have access to the D channel by the use of an access procedure. Each B channel is allocated to a particular terminal at the time of call set-up and is not capable of being shared among the terminals.

The power feed is provided across the user-network interface, to provide a basic telephone service, should there be a local mains failure. Figure 3.5.2. shows the physical arrangement of the power feed. The basic four-wire bus consists of two twisted pairs. A power source is available within the NT1 which can supply, via the centered tapped

transformer (known as phantom feed) a nominal voltage of 40 V, with up to 1 W of power.

The power under these nominal arrangements will probably be derived at the NT1 from a local main source. Under main failure condition, the power will be limited to 420 mW. To indicate this state to the terminals the polarity will be reversed. In this case, the power will be derived from the network. It is only intended to power a single digital telephone for emergency use. Two additional pairs are allocated for alternative power feeding arrangements, but it is not clear the extent to which these will be used.

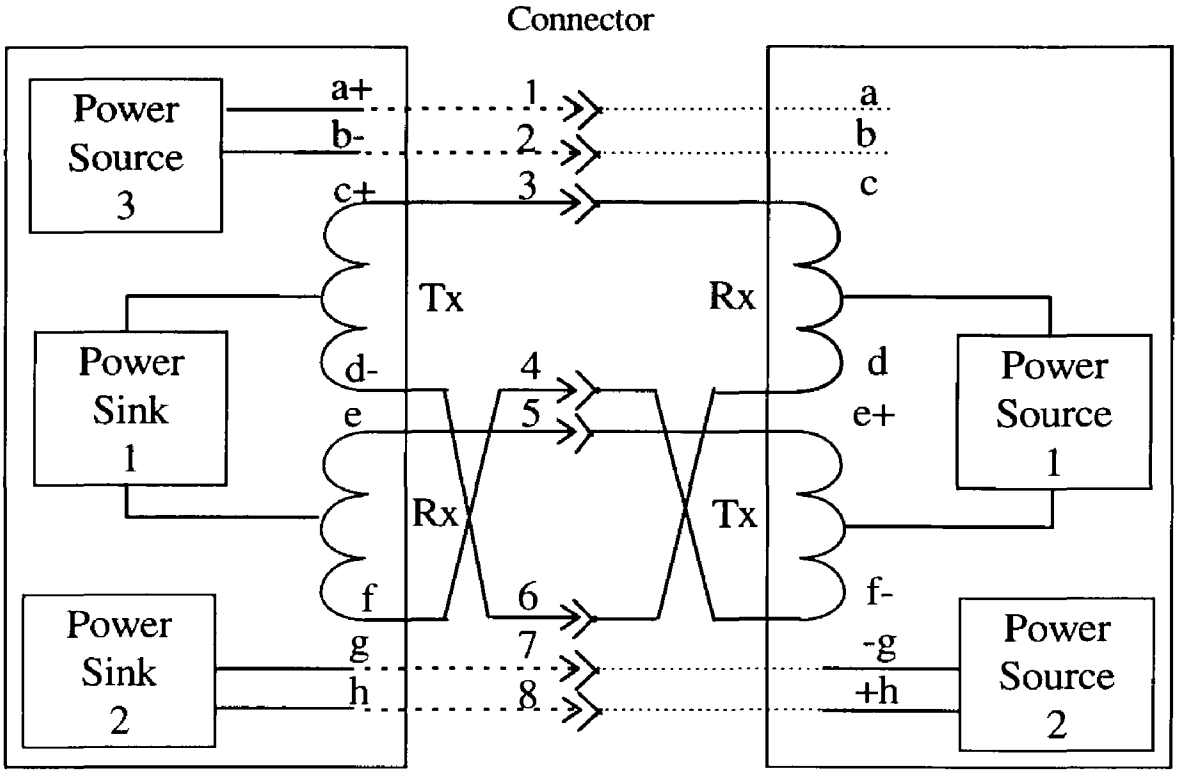


Figure 3.5.2. Physical Arrangement for power feeding for twisted pair [7].

The Electrical and Computer Engineering Department at Florida International University (FIU), together with Southern Bell and Northern Telecom, created a laboratory which has 5 ISDN lines to perform research and development of ISDN applications. The lines are configured as point-to-multipoint configuration. Figure 3.5.3. shows a configuration that is implemented today in the Lab. This configuration could be a typical configuration in either a commercial or a residential environment and it is the one that is going to be used for SECNET.

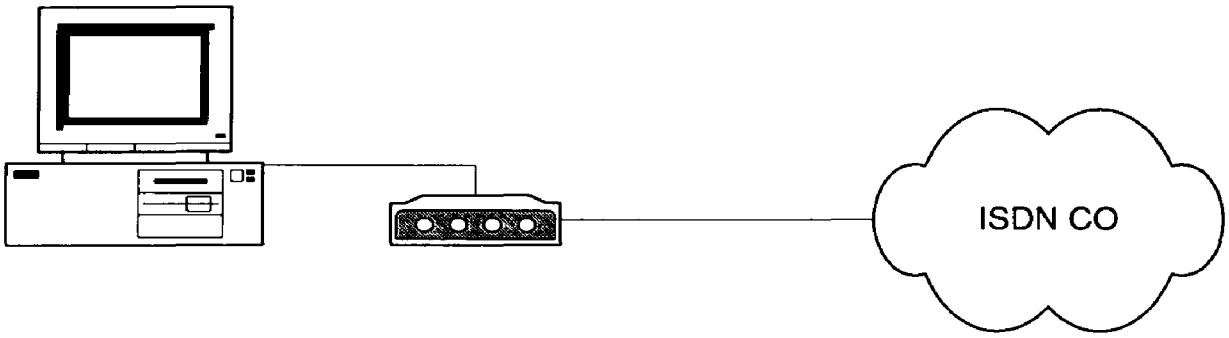


Figure 3.5.3. ISDN general line configuration at FIU.

The B channels have the Service Profile IDentifier (SPID) numbers. These numbers are assigned by the Telephone Company (TELCO), and the SPID is the one that identify the user's equipment during reset or power up. It is essentially a primary Directory Number (DN); however, it is also formed by the following sequence of numbers: [Prefix] Area Code - Phone Number [Suffix Numbers]. These numbers are explained in Table 3.5.1.

The computer is connected to the ADTRAN's Terminal Adapter through the serial port, and the latter is connected directly to the U interface.

TELCO	SPID Number				
				Suffix	
	Prefix	A. Code	Number	Proprietary	N-ISDN 1
AT&T Primary DN	01	XXX	XXXX	0	0001
Secondary DN	01	XXX	XXXX	1	0002;
Northern Telecom	----	XXX	XXXX	Last Digit	0000

Table 3.5.1 SPID definition for AT&T and Nortel.

This TA is capable of perform bonding to increase the transmission data bit rate up to 128 kbps. Using this configuration as a reference, each computer can simulate a different room or building. With this configuration, the possibilities of SECNET can be evaluated and tested.

Chapter IV. System Hardware.

4.1 Introduction

Computers manipulate information as commands and data. Both are represented as binary numbers. When information flows inside the computer all the bits go in different lines (depending on the data bus, 8, 16, 32, or more bits). The problem starts when it is necessary for the process to access external peripherals. The same information has to go through a physical medium, electrical wires to external peripherals and back to guarantee full duplex communication.

If the transmission of bytes is performed along eight wires, it is referred as parallel communication (e.g., printers use this mode). All eight bits are sent simultaneously, either from or to a peripheral device and a computer. Data transfer is accomplished at very high speeds. This method is a very efficient way to send information [8] to external peripherals. Its main disadvantage is that eight wires have to be connected between the computer and the peripherals, which is very impractical when the transmission process is between two devices separated for a long distance.

A second kind of transmission is known as serial communication. For this type of communication only two wires are needed to connect the computer and the peripheral. This arrangement allows to transmit one bit (out of eight) at a time. Although this communication method is neither fast nor efficient, it is much more cost effective for long distances than the parallel communication. Thus, it is the preferred way to communicate among computers.

To avoid overloading the computers' microprocessor with the serial communication process, some specialized integrated circuits were developed. Among them is, the National Semiconductor® INS8250, which is the standard universal asynchronous receiver-transmitter (UART) in modern microcomputers [8]. This Serial Communication Interface (SCI) was created to transmit information at data bit rate up to 19200 bits per second (bps). By the time computers grew more powerful and the necessity of transmitting megabytes of information, this UART was made, more sophisticated. Nowadays it can handle up to 115.2 kbps, and has internal buffers for transmission and reception that allow the communication process to be more effective.

The UART is in charge of all the serial communication processes using the computer's serial ports. However, to make different computer models compatible with each other when transmitting information through that port, an interface standard was developed. The Electronic Industries Association (EIA) was in charge of the development of this standard. Its title is "Interface Between Data Terminal Equipment (DTE) and Data Communication Equipment (DCE) Employing Serial Binary Data Interchange"; it is better known as Recommended Standard Number 232 revision C, or RS-232C [9]. The ITU also developed a similar standard and named V.24 and V.28 (electrical specifications)[9]. These standards define the mechanical, electrical and, and functional characteristics for the serial communication interface for the DTE and DCE equipment. It also includes a small subset of the functional characteristics for special equipment [9]. This allows users and communication equipment manufacturers to interact to make a more general interface.

SECNET system bases its operation on serial communication between their units using the RS-232C standard. Moreover, it also communicates with the Terminal Adapter (TA) device using the same standard. This communication link makes it possible for the computer accessing the ISDN lines to achieve connection to the main security office.

4.2 Serial Communication: The RS-232C Interface.

The RS-232C interface has been developed to define four main characteristics: mechanical, electrical, functional, and procedural. They specify the accurate nature of the interface between the Data Terminal Equipment (DTE, e.g., computers) and the Data Communication Equipment (DCE, e.g., modems) [10].

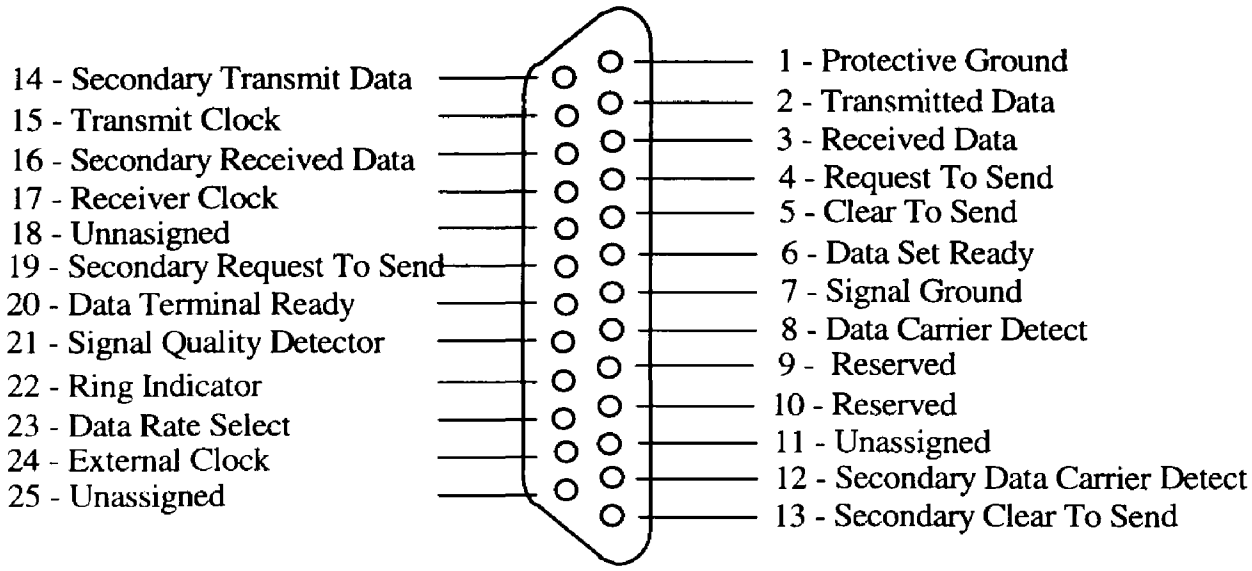


Figure 4.2.1 Typical DB-25 RS-232C connector [10].

This standard defines a 25-pin connector with a specific arrangement of leads, Figure 4.2.1 shows the physical pin connection for the (DB-25). This is the original RS-232 standard. For microcomputers, this interface is more commonly implemented using a subset of this standard in the IBM AT-class machines; it is a 9-pin connector that implements only

the most important functions of the interface. At first sight, this subset does not seem appropriate; however, from the programmer's point of view, it is good enough to carry out effective communications programming [8]. Table 4.2.1 represents the serial interface for an IBM AT microcomputer; it also shows the characteristics of the signal that each pin represents. Six signals are used to control the communication process between the DCE and the DTE. They are Data Carrier Detect (DCD), Data Terminal Ready (DTR), Data Set Ready (DSR), Request to Send (RTS), Clear To Send (CTS), and Ring Indicator (RI). Some of these signals are sent by the DTE and others by the DCE.

Pin	Symbol	Function Name	Characteristics
1	DCD	Data Carrier Detect	Data Link in progress *
2	RD	Receive Data	DCE sends data to DTE *
3	TD	Transmit Data	DTE sends data to DCE *
4	DTR	Data Terminal Ready	DTE ready to communicate with DCE *
5	--	Signal Common	
6	DSR	Data Set Ready	DCE ready to communicate with DTE *
7	RTS	Request To Send	DTE request to send to DCE *
8	CTS	Clear To Send	DCE ready to receive from DTE *
9	RI	Ring Indicator	DCE asserts RI when ring is detected *

* Asserted to logic state 1.

Table 4.2.1. The IBM-AT, DB-9 nine-pin, connector for the RS-232C Serial Interface [10].

The standard line parameters are the impedance, the maximum length of the cable to connect both DTE and DCE, the control procedure to send information, the bit transfer rate, and other attributes. It does not specify either the format or the content of the data being transferred [10]. Figure 4.2.2 shows the connection of the serial interface between the DTE and the DCE. The transfer of information among them is done by what is called "handshaking."

The DCE indicates to the DTE that certain operation has to be performed by asserting a logic state 1 in the appropriate line. The same operation occurs from the DTE to DCE. These signals will allow different equipment to determine when the other one is ready to receive or transmit charter through the transmitter and receiver lines, creating what is called hardware flow control..

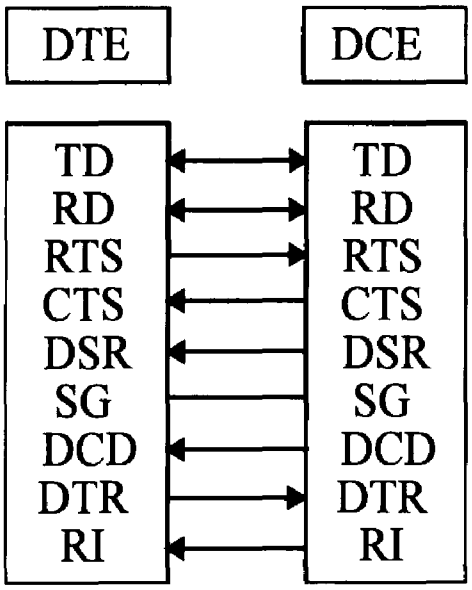


Figure 4.2.2 RS-232 C interface connection between DTE and DCE.

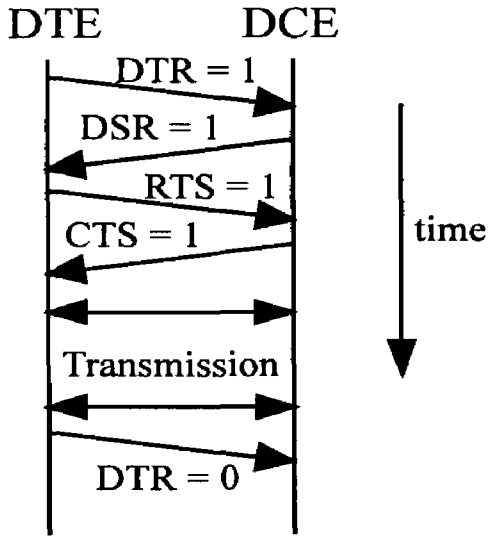


Figure 4.2.3 Handshaking process triggered by the DTE to start transmission.

Handshaking between the DTE and the DCE may have the following sequence, which is represented in Figure 4.2.3:

1. DTE asserts DTR to indicate that is active a ready to transmit.
2. DCE responds by asserting the DSR line, indicating that it also is ready for transmission.
3. DTE asserts the RTS line to indicate that is ready to start the transmission.
4. DCE answers by asserting the CTS line, indicating that it is ready to receive data.
5. Data transmission starts on the TD and RD lines.
6. DTE drops DTR to disconnect.

As it was previously stated, the format and content of the data being transmitted are not specified by the standard; however, both must be defined in serial communications. In a serial interface, data are transmitted one bit at a time. This transfer must have a way to be synchronized between the DTE and the DCE [8], and both devices have to agree upon them.

The first parameter to be defined is the bit transfer rate [11] expressed in bits per second (bps); the rest of the parameters are start bit, data bits, parity bits and stop bits. All parameters together specify the format of the data sent through the serial port [11], and the data bits themselves specify the data content. They together form the serial frame that the serial port sends for each character. The usage of each bit is explained below.

Start Bit: When a serial device is not transmitting, it asserts (sets the line to logic "1") the transmission line (TD). To start the transmission of a character, the transmitting serial device sends a start bit. This bit is a logical 0 and it last one fraction of the data

transfer rate time $= \left(\frac{1}{\text{data bit rate}} \right)$. The DCE is able to synchronize with the DTE by simply waiting for a change in the TD line from logical 1 to logical 0. Then, it halts for half a bit time, and then starts sampling the incoming data.

Data Bit: There are five to eight data bits. In the initial times of serial communications, all the data transfer was performed using the upper case part of the alphabet; also, cumulative timing error is smaller for lesser number of bits than for larger. However, since the computer equipment is byte-oriented, the natural way to handle data is in an eight bit fashion. The American National Standard Institute (ANSI) created the American Standard Code for Information Interchange or ASCII (ANSI Standard X.3.4-1977), intended to be primarily a "human readable code" [11], (Alphabet No. 5 of the ITU). ASCII is a seven-bit code, and the Extended ASCII is an eight bit code. The character code used in this thesis is the extended ASCII code with eight bits per character.

Parity: Parity is a rather ineffective and coarse form of error detection. There are five basic forms of parity: None, Even, Odd, Mark, and Space. A common setting is None (no-parity bit) when eight-bit of data are transmitted. Moreover, there are other better forms of error detection that improve the error detection and correction in serial communications.

Stop Bit: This bit signifies the end of the frame. It can be one, one and a half, and two bits. The transmitting device always sends a logical "1" for the stop bit. It also asserts the TD line.

All these bits together form the frame format for a data item that the serial port sends to the RS-232C interface. This frame format is shown in figure 4.2.4. This figure

represents the actual frame for number 6 (ASCII code 0x36 Hex). It has the start bit, eight data bits, and the stop bit, no parity bit.

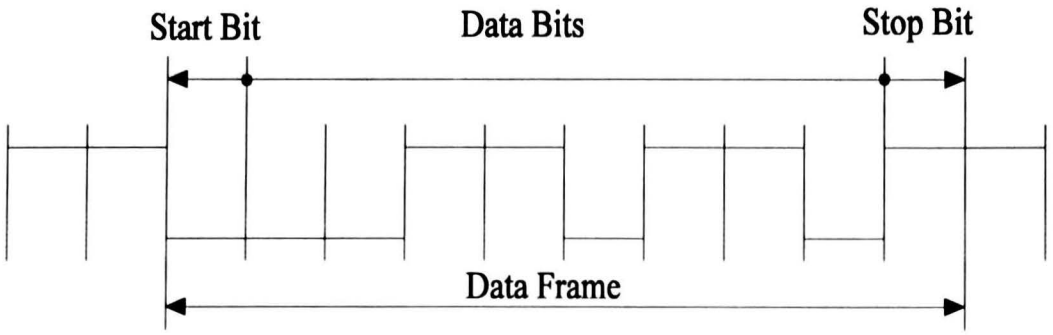


Figure 4.2.4 Frame format for ASCII code 0x36 Hex (number 6) for serial transmission.

Asynchronous transmission uses what is called character or word synchronization. A character is sent one bit at the time. This transmission uses a start and a stop bit, making it simple, cheap, and with modest time requirements. The receiver samples each bit at 50% of its time. If the receiver rate is in error by 5% (either slower or faster than the transmitter), the eighth bit of information will be displaced by a 45%; however, it is still correctly recovered. It has a 22% of overhead because the addition of the extra bits: one start, and 1 stop bits. Table 4.2.2 shows Circuit Switched Data (CSD), and Packet Switched Data (PSD) data bit rates.

	Data Transfer Rate	Data Bits	Parity	Stop Bits	Representation
CSD	38.4 kbps	8	None	1	38400 8 N 1
PSD	9.6 kbps	8	None	1	9600 8 N 1

Table 4.2.2. Data format to setup the computer's communication ports.

This format is selected for this work because it is simple, cheap, and has small overhead, as it has at least one fewer bit than the other formats (the parity bit). The parity bit, although important for error recovery, it is not necessary. In first place, other kinds of

error recovery methods are used; in second place, Integrated Services Digital Network (ISDN) has a built in error correction and detection.

All of these parameters defined in the RS-232C standard are considered the physical layer protocol, which is the first layer of the Open-System Interconnection (OSI) reference model. This model is a very important concept applied to data communication because it serves as a framework for the development of each communication protocol standard in the latest years [12]. All communication equipment manufacturers should build their equipment following the recommendations of the OSI, which will guarantee that the equipment will be able to communicate with equipment made by other manufacturers.

The serial communications may take place in four connections:

- Simplex: Communication in only one direction.
- Half-duplex: Communication is performed in both directions, one direction at a time.
- Full-duplex: Communication is performed in both directions, at the same time.
- Multiplex: This connection allows multiple serial communication channels to occur over the same serial communication line.

The serial communication parameters defined by the RS-232C standard are implemented in a piece of hardware so that the DTE can communicate with the DCE using the physical layer. This piece of hardware should free the microprocessor of all the burden that the serial communication protocol represents to it. This is done (as stated before) by the SCI's, especially the INS8250 family. It is the most commonly used communication integrated circuit in DTE equipment. In, accord with the OSI model (Figure 3.3.1), this

device operates in Layer 1 or the Physical Layer. The way that the upper layers communicate with this device is through software. The present thesis follows the OSI model by defining the low level functions implemented in the software part as the layer 1 of the cited model. These functions are in charge of the interaction between the software and the hardware. These software functions are explained in Chapter 4. They should give the information that is being received from the upper layers in the proper format to be used by the UART. In order to follow the specification for the serial transmission, it is necessary to know the functions of each of these registers.

4.3 The Universal Asynchronous Receiver Transmitter (UART).

All UART's have different registers to control all their operations. These registers can be accessed using input and output port operations. They can be read from and written to using the microprocessor in the computer.

These connectors are known as serial ports, and named COM1, COM2, COM3, and COM4. Each of them has an associated address that allows the microprocessor to communicate independently with each of them. Table 4.3.1 represents each port associated with each particular address in a PC/AT microcomputer.

Port	COM1	COM2	COM3	COM4
Base Address (Hex)	3F8	2F8	3E8	2E8

Table 4.3.1 Serial port base registers addresses.

The INS8250 has different registers that can be accessed using the input or output port instructions using the parallel port. Each of these registers executes its function or

functions depending upon their bit functions. They can be read only (status registers), write only (control registers), or read/write by the microprocessor.

Table 4.3.2 shows the names of the different registers, their function, and the bits involved on each function. Also, their memory locations are addressed with respect to the base address for COM1, the same scheme is used for COM2. This makes easier for the programmer to use a pointer and each register is addressed as an offset of that pointer.

DLAB Bit = 0										
Address	R/W	Name	7	6	5	4	3	2	1	0
0x3F8	r	RBR	d	d	d	d	d	d	d	d
	w	THR	d	d	d	d	d	d	d	d
0x3F9	r	IER	0	0	0	0	int.	error	tbe	rxrdy
	w	IER	0	0	0	0	int.	error	tbe	rxrdy
0x3FA	r	IIR	0	0	0	0	0	int. type		i.p.
0x3FB	r	LCR	dlab	break	parity settings			stop	data bits	
	w	LCR	dlab	break	parity settings			bits	data bits	
0x3FC	r	MCR	0	0	0	llbt	gpo2	gpo1	rts	dtr
	w	MCR	0	0	0	llbt	gpo2	gpo1	rts	dtr
0x3FD	r	LSR	0	txe	tbe	break	f.e.	p.e.	o.e.	rxrdy
0x3FE	r	MSR	dcd	ri	dsr	cts	Δdcd	Δri	Δdsr	Δcts

Table 4.3.2.a Registers for COM1 when the eighth bit of LCR register is 0, (DLAB = 0).

Each bit has associated with it a function, and its state indicates whether the function was performed or not. From the above table, these are their meanings: int. : interrupt; i.p.: interrupt pending; llbt: local loop-back test; f.e.: frame error; p.e.: parity error, o.e.: overrun error, tbe: transmitters buffer empty, and rxrdy: receiver ready.

DLAB Bit = 1										
Address	R/W	Name	7	6	5	4	3	2	1	0
0x3F8	R	DLL	d	d	d	d	d	d	d	d
	R	DHL	0	0	0	0	0	0	0	0
	W	DLL	d	d	d	d	d	d	d	d
	W	DHL	0	0	0	0	0	0	0	0

Table 4.3.2.b Registers for COM1 when the eighth bit of LCR register is 1, (DLAB = 1).

Receiver Buffer Ready (RBR) and Transmitter Holding Register (THR): The RBR and the THR have the same address, which is the base address of the serial ports. The RBR registers holds the data that came through the serial port. It can be accessed by reading from the base register. On the other hand, the THR holds the data to be sent through the serial port. Characters to be sent are to be written to this register. To access this registers, the Divisor Latch Access Bit (DLAB) bit 7 of the Line Control Register (LCR) must be set to 0.

Interrupt Enable Register (IER): This register enables the UART's interrupts. The address of this register is the base register address plus one) (0x3F9). Bit 2 is used to detect all error conditions that could happen when a character either is being received (break condition) or when the character has been received, parity, framing, or overrun. Each bit is explained in Table 4.3.3.

Bit	Symbol	Function
0	rxrdy	Receiver ready, set when a byte is in the rx buffer (RBR).
1	tbe	Transmitter buffer empty, set when tx is ready to send a byte.
2	error/break	Set, when the UART detects an error: parity, framing, overrun, etc.,
3	modem status	Set, when a change in one of the modem status inputs
4 - 7	always 0	Not used bits

Table 4.3.3 Control byte for the IER register.

Interrupt Identification Register: The IIR register indicates why an interrupt has

Bits				
2	1	0	Priority	Interrupt type
0	0	1	None	No interrupts pending
0	0	0	4	Modem status change
0	1	0	3	Transmit buffer empty (tbe)
1	0	0	2	Data received ready (rxrdy)
1	1	0	1	Error or break detected

Table 4.3.4 Interrupt cause and priority.

occurred. Bits 2 through 0 show what interrupt occurs and the order of priority that the interrupt has.

The two most significant bits of this registers indicate that the First-In-First-Out (FIFO) buffers are enabled.

Modem Status Registrar (MSR): The MSR provides the current state of the RS-232-C control lines. Each bit represents one of the lines and if set, they will generate an interrupt (if bit 3 of the IER is set). Table 4.3.5 shows the bit description for this register.

Bit	Name	Description
0	Delta CTS	if CTS changed state, set to 1
1	Delta DSR	if DSR changed state, set to 1
2	Delta RI	if RI changed state, set to 1
3	Delta DCD	if DCD changed state, set to 1
4	CTS	Current state of CTS
5	DSR	Current state of DSR
6	RI	Current state of RI
7	CD	Current state of CD

Table 4.3.5 Modem status register Bit Set.

Baud Rate LSB and MSB Divisor Latch Registers (DLL): This register can be accessed by setting DLAB bit. Register 0 becomes then the LSB DLL, and register 1 becomes the MSB DLL; the function of these two registers is to hold the data bit rate divisor for the UART.

Table 4.3.6 shows three of the most used data bit rates and the divisor number that this registers holds. To get the proper data bit rate, the main clock frequency given to the UART is divided by the constants, which are programmed to the device.

The value used to communicate through the serial port is going to be fixed to 38.4 kbps to achieve maximum data bit rate (dbr).

Data Rate (bps)	Latch Value			
	Decimal	Hex	MSB (Hex)	LSB (Hex)
9600	12	C	00	0C
19200	6	6	00	06
38400	3	3	00	03

Table 4.3.6 Divisor values for each data bit rates.

4.4 Serial Communication Ports for SECNET.

The communication ports are going to be set up at two data bit rates: 38.4 kbps for CSD using COM2, and 9.6 kbps for the SECNET control device attached to COM1. Also the frame used for the asynchronous communication is going to be 1 start bit, 8 data bits, no parity bit and 1 stop bit.

To use high data transfer rates, it is necessary that the microprocessor be aware of each data byte that is arriving to the serial port. By using interrupts, the processor can perform other tasks without losing the information being received through the serial port.

This is accomplished by enabling the microprocessor's interrupt handler and creating a function that services the interrupts. Table 4.4.1 shows the address for the UART registers in PC/AT or compatible computers. These definitions are necessary to allow the microprocessor and the UART to recognize each other and create the path to send and receive information from the higher levels of the communication process. Up to this point the system is able to interface with the Terminal Adapter (TA), either internally or externally. ISDN lines can now be accessed and then remote DTE's can interchange messages. Layer 1 in the OSI model in the DTE side is implemented. The next step is to create the physical interface, which is in charge of the security function for equipment and

facilities. The rest of this chapter is dedicated to this important part of the present project, and it has the introduction to a new technique that will improve the service that the security system performs to protect the facility where it is installed as well as the equipment in that facility. This new system is considered a new way to protect facilities and equipment.

	COM1		COM2	
Port base address	p_add	0x3F8	p_add2	0x2F8
Interrupt Service Routine	irq_add	0x0C	irq_add2	0x0B
Interrupt Request Line	irq_no	0x0EF	irq_no2	0x0EF
Interrupt Enable Register	IER	0x3F9	IER	0x3F9
Interrupt Status Register	ISR	0x3FA	ISR	0x3FA
Line Control Register	LCR	0x3FB	LCR	0x3FB
Modem Control Register	MCR	0x3FC	MCR	0x3FC
Line Status Register	LSR	0x3FD	LSR	0x3FD
Modem Status Register	MSR	0x3FE	MSR	0x3FE
LSB Divisor Latch	DLL	*	DLL	*
MSB Divisor Latch	DHL	0x00	DHL	0x00

* 0x03 for 38.4 kbps or 0x0C for 9.6 kbps

Table 4.4.1 Register denomination.

4.5 SECNET Equipment and Security Function.

The computer is the medium to communicate with the ISDN lines. It is the one that is connected to a TA (internal or external) through the serial port. However, the computer is used for other purposes also. In order to implement all the security functions, it is necessary to have equipment that performs operations to secure the room, the equipment in the room, and to detect intrusion. In the event of a detected threat, it communicates to a computer, which is connected to a Terminal Adapter (TA) to send specific information to the main security office through ISDN.

This equipment must have intelligence to perform the complex operation assigned to it. A microcontroller is an advanced single-chip Microprocessor Control Unit (MCU),

which has on-chip memory Random Access Memory (RAM), and Erasable Programmable Read Only Memory (EPROM). It also has different peripherals to implement a variety of functions or to allow the expansion of the MCU. These peripherals are: parallel ports, serial ports, timers, Analog-to-Digital (A/D) and Digital-to-Analog (D/A), and others. All of these peripherals make the single chip MCU a very powerful device that can be used to implement complex security applications at very low price.

The present work is based on the Motorola's MC68HC11 microcontroller unit. It is an advanced single-chip 8-bit MCU with on-chip memory and peripheral capabilities. The microcontroller clock runs 8 MHz with at a nominal bus speed of 2 MHz. The major peripheral functions are an eight-channel A/D converter with 8 bit of resolution; a 16-bit free running timer system with three input-capture lines; five output-compare lines, and a real-time interrupt function. An 8-bit pulse accumulator subsystem that allows to count external events or measure external periods of time; also, it has an asynchronous and a synchronous serial interface.

The instruction set consists of up to 110 base instructions; almost all the main instructions supports up to six addressing modes (increasing the number of instructions that the MCU can execute). It has two 8-bit accumulators (A and B) and they can be used as a single 16-bit accumulator by some instructions. This accumulator is called the D register.

The MCU itself can be expanded with additional ports that allow the connection of more peripherals and memory to the microcontroller in case of more powerful functions were required. At the same time it keeps part of the internal capabilities that it has.

The MC68HC11 has five ports, referred as: A, B, C, D, and E. Some of the pins in the ports have dual functions. Also, some ports themselves have more than one function to

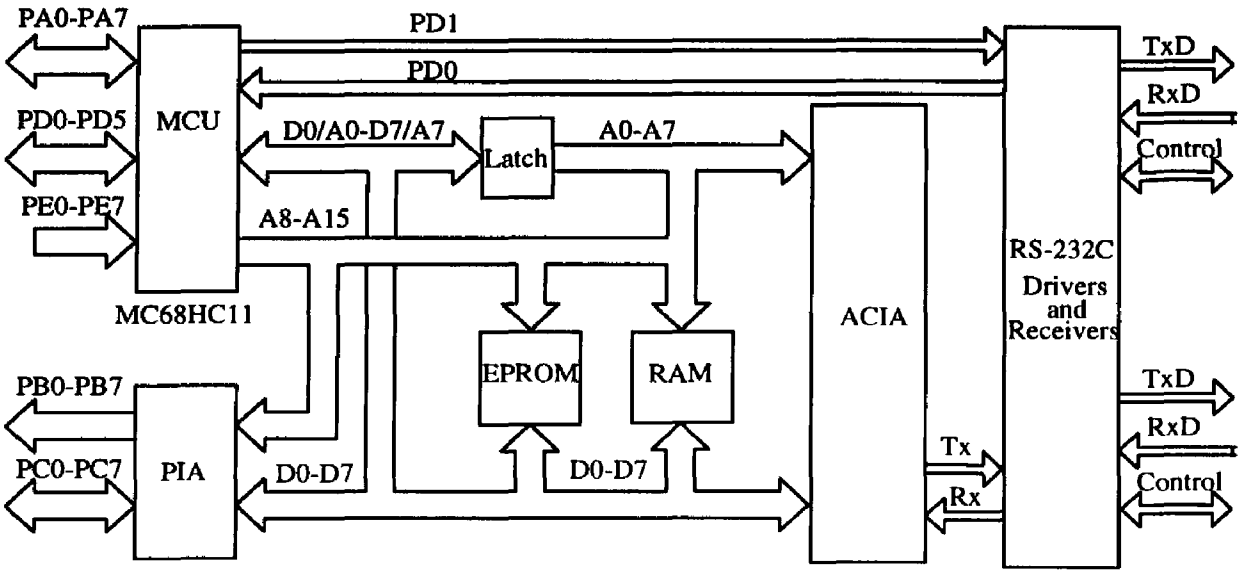


Figure 4.5.1 Block Diagram for an Expanded MCU as used in the EVB from Motorola, Inc. [13].

perform. Figure 4.5.1 shows the block diagram for the MC68HC11 EVB from Motorola Inc. [13]. Each bit of the different ports has an assigned function for ports A, D, and E. Table 4.5.1. shows the function of each port, and what each bit represents.

To use the MCU, it has to be expanded to use external memory because the internal memory the microcontroller has does not have enough space to handle the system code software. In this case, two ports are used to make the expansion. Those two ports are going to work as the external address and data bus (ports B and C). By doing the expansion, external memory and peripherals are going to be added to recover the functions of the lost ports. The external memory is going to be Random Access Memory (RAM) and Read Only

Mode (ROM). ROM memory holds the software code to make the system operational, and RAM memory holds the parameters that dynamically vary, when the program is executed.

		Bits								
Port	Function	7	6	5	4	3	2	1	0	Operation
A	General	I/O	O	O	O	O	I	I	I	Each Pin
	Timer	OC1	OC2	OC3	OC4	OC5				Each Pin
	PA	PAI					IC3	IC2	IC1	Each Pin
B	General	O	O	O	O	O	O	O	O	Full Port
	Address	A15	A14	A13	A12	A11	A10	A9	A8	Full Port
C	General	I/O	I/O	I/O	I/O	I/O	I/O	I/O	I/O	Full Port
	AD	AD7	AD6	AD5	AD4	AD3	AD2	AD1	AD0	Mux.
D	General	------	------	I/O	I/O	I/O	I/O	I/O	I/O	Each Pin
	SCI							TxD	RxD	Each Pin
	SPI			!SS	SCK	MOSI	MISO			Each Pin
E	General	I	I	I	I	I	I	I	I	Full Port
	A/D	I	I	I	I	I	I	I	I	Full Port

Table 4.5.1. Functional representation of ports in the MC68HC11.

From the table: I: Input; O: Output; PA: Pulse Accumulator; OC: Output Compare; IC: Input Compare; A: Address; AD: Address/Data; SCI: Serial Communication Interface; SPI: Serial Peripheral Interface; SCK: Serial Clock; !SS: Slave Select (active low); MOSI: Master Out/Slave In; MISO: Master In/Slave Out.

The final system is represented in Figure 4.5.1. The low address bus is latched to obtain 16-bit of addressing space (65 kbytes bus space), 8-bit data bus, an expansion Parallel Input Adapter port (PIA) and an Asynchronous Communication Interface Adapter (ACIA) port. These additions make the system more versatile and able to perform more effective communication tasks, because it has more serial ports to establish a link with other devices. Also, it allows the MCU to be connected to other similar units in a token ring

fashion, making the system more attractive for future addition of more equipment to be protected by the system.

This configuration allows the board to be connected to the computer using one of the serial ports. A second UART has to be added to use it to control each of the devices attached to the control unit.

The EVB connection to the PC is then accomplished through a serial port using RS-232C standard. The board does not have any hardware or software handshaking, therefore, a delay must be supplied to ensure the proper transmission/reception of bytes. The SCI allows full duplex communication. The parameters are data bit rate, data bits, number of stop characters, and parity. They can be programmed using the Serial Communication Control Register 1 (address: 102C Hex), which controls the number of transmitted and received data bits. Control Register 2 (address: 102D Hex), which enables and disables the interrupts; the Baud Rate Register (BAUD, address: 102B), which sets the prescaler to get a specific data bit rate.

The SPI is a synchronous interface that allows to several units to be connected in a token ring environment, using a master-slave configuration with other same units. This serial communication controller uses four basic signals MISO, MOSI, SCK and SS.

The last serial communication facility is the ACIA. This port is going to be used as a medium to communicate with the computer to create the link between the SECNET security control device and the ISDN line. This port has to match the configuration settings for COM1 that the computer has, that is 9600 bps N 8 1. Table 4.5.2. shows the ACIA registers and their functionality. The ACIA is a very simple communication port. It is very easy to

program and to use. Its only disadvantage is that the receiver and transmitter share the same register making the communication process looks like a half-duplex device. There are not two registers for the communication; this means that to send a character, the microprocessor has to be aware if a received character is in the buffer otherwise it will lose it. The way that this communication process is implemented is by making the reception of characters from the computer interrupt driven.

Address	Function	7	6	5	4	3	2	1	0
9800 read	Status Register	interrupt request	parity error	oe	frame error	CTS	CD	Tx Empty	Rx Ready
9800 write	Control Register	interrupt enable	00 Tx Int Dis 01 Tx Int En		Data Bits	Parity/Stop		Data Bit Rate divider preset	
9801	Tx / Rx	X	X	X	X	X	X	X	X

Table 4.5.2 ACIA register in the Motorola's EVB board.

4.6 Device Used to Secure the Equipment.

As stated in Chapter 1, the main objective of this thesis is to create a security system that be able to communicate through ISDN to a main security office, and to have a determined amount of equipment under very tight protection. The system hardware allows the system to communicate to a computer where the interface to the ISDN line is implemented.

To send and receive a message is one function that this piece of hardware has to be able to perform. This operation allows the system to send a piece of information to a specific protected device getting a proper answer back from that device. The MCU will be aware of each of the devices connected to the system keeping a close control of each device.

To allow the MCU to perform this tasks in a low cost and an efficient way different solutions where analyzed. Figure 4.6.1 shows the system implementation for the general idea. A device under protection (e.g., a printer) is connected to the SECNET device, which monitors whether the printer is connected to the protection system or has been disconnected.

The SECNET device will constantly send a signal to the device attached to the equipment under protection to make sure that it is in place and nothing has been altered.

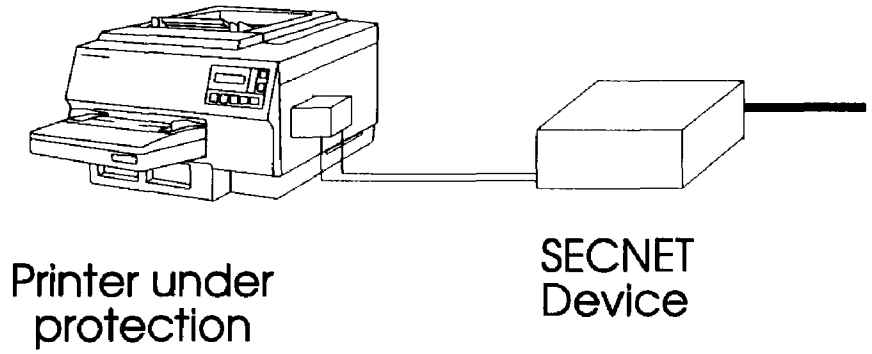


Figure 4.6.1 System implementation to protect a printer.

A signal will be sent every time that the equipment is queried to check if it has any problem. This is done by a serial port, which sends the signal to the device attached to the equipment under protection. The serial port should receive the answer back. In case that the signal is not received (or if it receives something different that the data sent), the circuit will alert the MCU. The MCU will send an alert signal to the computer indicating the equipment position.

The functional circuit is shown in figure 4.6.2. A serial port is in charge of sending a code signal to the device that is attached to the equipment under protection. The device should allow the serial port to receive this code sent for some data bit rates, and for other

should not allow to receive the code. This is passive device in order to simplify the design and reduce the cost.

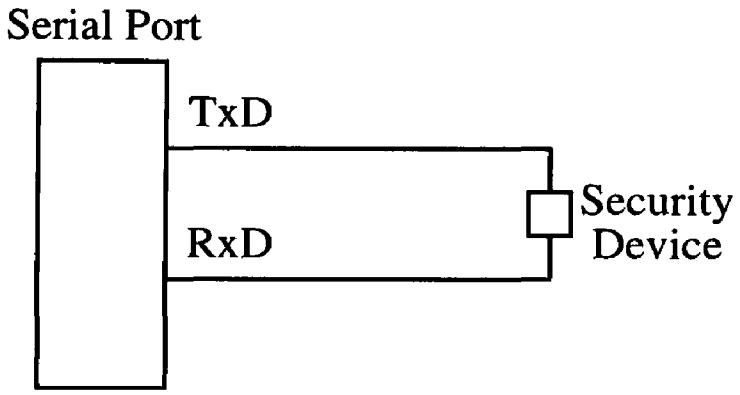


Figure 4.6.2 Functional Block for Equipment Protection.

The objective of allowing to pass some specified data bit rates and no others is to detect if the line could be either short-circuited or opened. For example, the serial port receives data bit rates of 0.6 kbps and less; on the other hand, it cannot receive data bit rates of 9.6 kbps or higher. One code is sent to the device under protection using the former rate, and a second code is sent at the latter data bit rate. The former is intended to be received and the latter is not. If someone short-circuits the lines, the receiver will get both codes, which means that the system is not performing under parameters and something happened to the equipment under protection. On the other hand, if the lines are opened, the receiver will receive neither the higher nor the lower data bite rates (bad operation). This also means that something could be happening to the equipment under protection (bad operation.)

For ease of discussion let us refer to a protected piece of equipment as a component. The above explanation is for the protection of only one component; however, the cost of this design does not justify a very powerful device to protect only one or a few component.

To make the system work protecting more than one equipment, it is necessary to create a method to send different codes to different components. It is not cost effective to have more than one serial port for each device because the system will become too expensive and will have serious space limitations.

Analyzing same kind of problems in real life, it was found that the same general idea is accomplished in a telephone network. One Central Office (CO) is in charge of supplying telephone communication to a very large number of users, by multiplexing several lines into a high speed single line. Figure 4.6.3 shows a functional diagram of this representation.

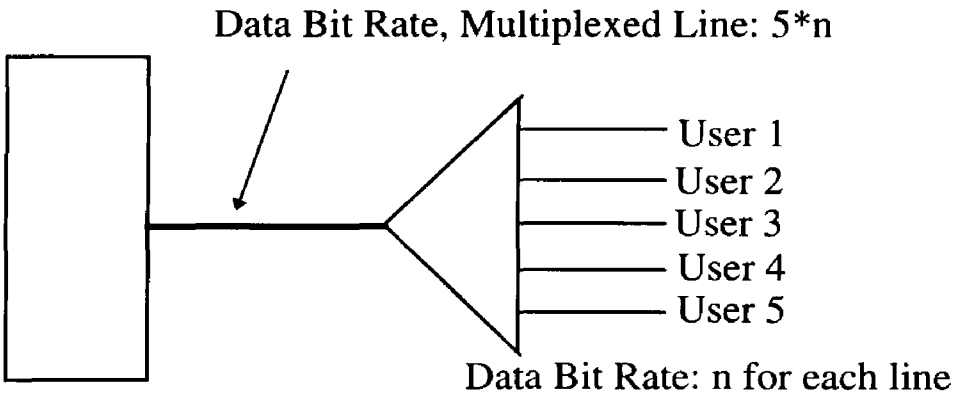


Figure 4.6.3 Functional representation of a Central Office (CO).

Considering the same principle, lets assume that the UART will be the CO by connecting a demultiplexer to the transmission pin. Multiple devices can be attached to only one transmission line. Each component will have a security device connected. The return line that should be connected to the receiver pin of the UART is then multiplexed to a line, which is connected to that pin. The main different between this system and the CO is that

the receiver in the UART does not have to be high speed. Only one component at a time will be polled.

A functional diagram for this system is shown in figure 4.6.4. Up to eight devices can be connected to the demultiplexer, and multiplexer. Both the multiplexer and demultiplexer can be controlled by the same three control lines, which simplifies the control circuit for this scheme. This scheme is used for both UART's used in the project. The operation of the system is as follows: a control signal selects the path through the multiplexer/demultiplexer to the Tx1 and Rx1 lines. Then the component's status is determined.

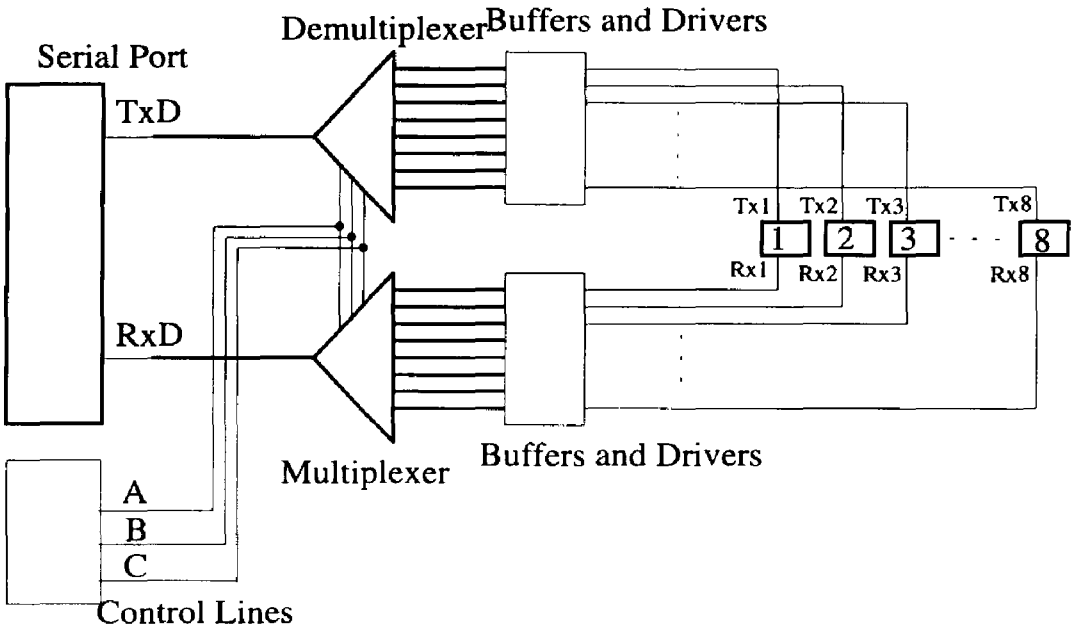


Figure 4.6.4. General diagram for the external circuitry used to implement the equipment protection.

The UART that was selected for this process is the MC68861 DUART. As its name indicates it has two identical serial ports built-in. UART1 is then programmed with a low

data bit rate and UART2 with at a high data bit rate. A code is sent to UART1. The code received is checked by the MCU. All the devices are checked and verified. The next step is to change the code and send it, using UART2, to each protected component to check all of them; in this case the code should not be received. The way the system operates is considered a unique aspect of the present work. Table 4.6.1. shows the control signal to manage the transmission and reception lines of the system.

From the table 4.6.1, letter C means a connection and letter Z means a high impedance state, which keeps the other devices disconnected when the selected one is receiving code from the MCU.

Transmission Lines								Control Lines			Reception Lines							
7	6	5	4	3	2	1	0	C	B	A	7	6	5	4	3	2	1	0
Z	Z	Z	Z	Z	Z	Z	C	0	0	0	Z	Z	Z	Z	Z	Z	Z	C
Z	Z	Z	Z	Z	Z	C	Z	0	0	1	Z	Z	Z	Z	Z	Z	C	Z
Z	Z	Z	Z	Z	C	Z	Z	0	1	0	Z	Z	Z	Z	Z	C	Z	Z
Z	Z	Z	Z	C	Z	Z	Z	0	1	1	Z	Z	Z	Z	C	Z	Z	Z
Z	Z	Z	C	Z	Z	Z	Z	1	0	0	Z	Z	Z	C	Z	Z	Z	Z
Z	Z	C	Z	Z	Z	Z	Z	1	0	1	Z	Z	C	Z	Z	Z	Z	Z
Z	C	Z	Z	Z	Z	Z	Z	1	1	0	Z	C	Z	Z	Z	Z	Z	Z
C	Z	Z	Z	Z	Z	Z	Z	1	1	1	C	Z	Z	Z	Z	Z	Z	Z

Table 4.6.1 Control signals and their relation with the transmission and reception lines.

To create the circuit that performs all previously explained operation a external circuitry was created in addition to the M68HC11EVB board. Figure 4.6.5 shows the block diagram for this circuitry.

To make the system more flexible and at the same time more cost effective, the design in figure 4.6.4 can be adapted to a system with a bus. A single integrated circuit with multiple parallel ports can be connected along with the single serial port making all the

outputs lines buffered. Moreover, multiplexer-demultiplexer input and output can also be buffered, and separated in cards that can be added on demand to the SECNET device. Users with less equipment to protect can buy the system with minimum requirements. For user who need more capabilities, they can be added by just plugging in cards to the SECNET device and then program the MCU to start checking the added equipment. This modular approach makes the system more cost effective and easy to upgrade.

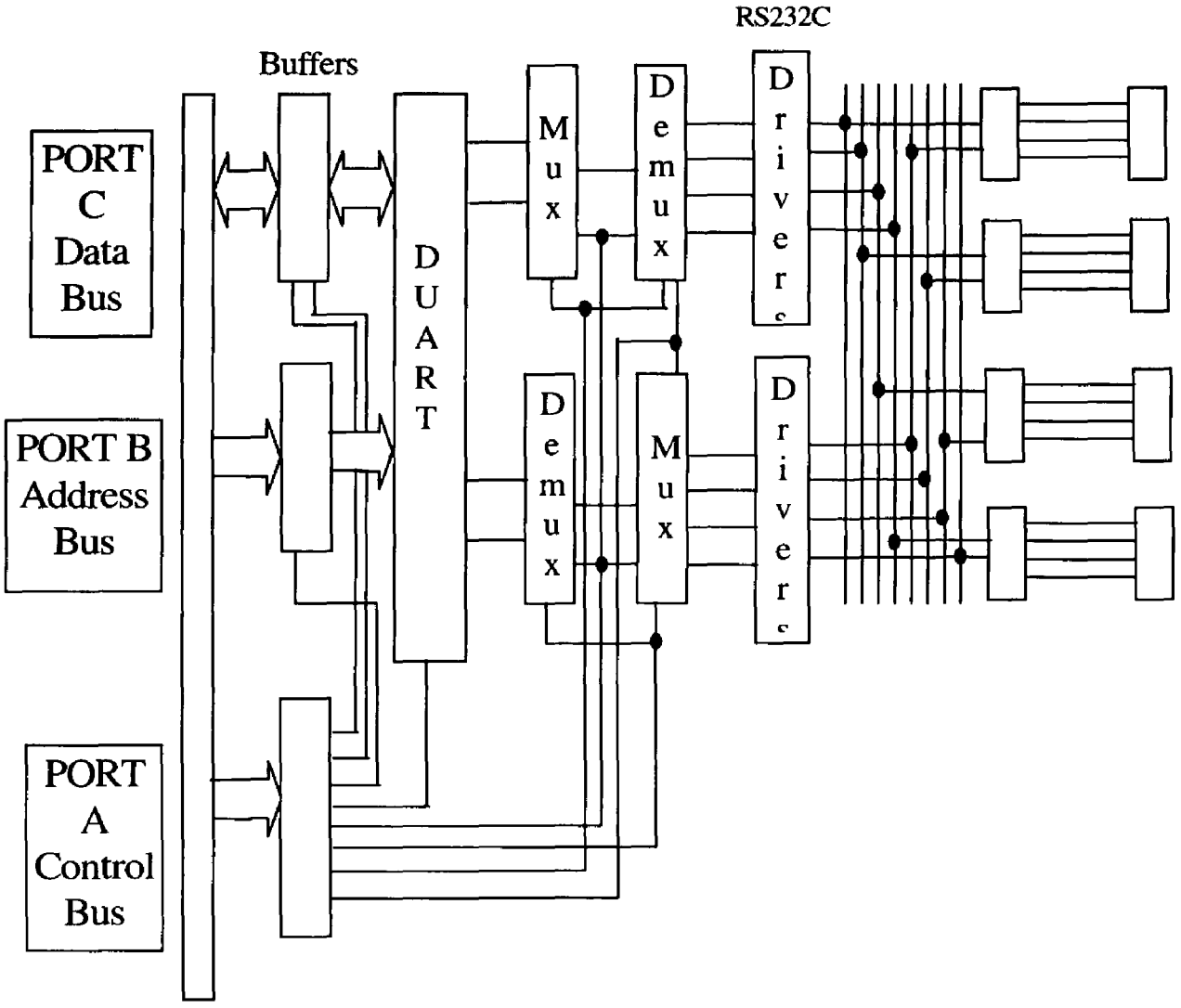


Figure 4.6.5. Modular representation of the external circuitry connected to the control unit.

Chapter V. System Software.

5.1. Introduction.

Software for SECNET is developed in two different parts: the one that operates in the computer and the one that operates in the security control device. The former is implemented using a high level language and the latter using assembly language. The present chapter explains the way in which both software parts are implemented and how the operate.

To implement the software, the OSI layered model explained in Chapter II was used as a guide for its design. The layered model offers a way to isolate functions that are related to the operations for which they are designed, and the results of those operations will be used be the adjacent layers. This makes the software easy to write and to debug. Also, it makes the software easy to upgrade by adding new functions that will improve the overall functionality of the complete system. Although, the OSI model can be taken exactly the way it is explained, an analogy is used to implement its functionality for the software.

The functions that have direct access to the hardware such as dirvers to read and write to the SCCs registers are considered as layer 1 functions. Functions that use the former but perform other operations to the data are considered functions in layer 2. The same definition process is used for the rest of the function. The result is a software highly modular and very easy to update and work.

The second methodology specially used for the software developed for the computer part is Object Oriented Design (OOD). The software is organized as a collection of discrete objects that are an abstract representation of some real world item or concept that incorporates both data structures and behavior [14]. This approach offers a good way to organize data and related actions [15].

The language used to implement the OOD is C++. It has become an increasingly popular programming language because of its data abstraction and object-oriented features. Its major addition is the introduction of classes that are the equivalent of objects. Classes allow the designer to define aggregated data types that include not only data members but also functions. They are implemented as structures. The way to access their members is either by value or by pointer. Classes inheritance extends data abstraction to object-oriented programming [16].

The software, then, is divided in software for the control unit written in assembly language and software for the computer written in C++ language. The rest of the chapter explains both parts of the software.

5.2. Control Unit Software.

The control unit has the function of interacting with the main computer receiving and sending commands and information. Also, it performs control operations over the devices that have been activated. A control operation is performed by sending a control byte at two different data bit rates (dbr). Then, detecting if the information in the unit

corresponds with the information that was sent. If something does not correspond a command is sent to the computer indicating the faulty component position or positions.

Functions	Layer	Comments
state1_func	6	state 1 operations
state2_fun	6	state 2 operations
state3_func	6	state 3 operations
state4_func	6	state 4 operations
state5_func	6	state 5 operations
state6_fun	6	state 6 operations
state7_func	6	state 7 operations
state8_func	6	state 8 operations
detect_dev_activ	5	detects if at least one device is active
chk_dev_bad	5	check if any device has problem
add_function	5	function to add a device to system
add_dev_to_q	4	add device to the right memory position
dis_function	5	function to disable a device
dis_dev	4	disable device in memory
en_function	5	function to enable a device
en_dev	4	enable device in memory
remov_function	5	function to remove a device
remov_dev_q	4	remove a device from memory
set_cntrl_table	3	set control table
chk_low_dbr	3	perform the control for low dbr operation
chk_hi_dbr	3	perform the control for high dbr operation
send_bad_dev	3	send which device is bad
outstring	2	output a string
enable_scc	2	enable the SCCs A and B
send_char_scca	2	send a character to SCC A
send_char_sccb	2	send a character to SCC B
init_scc_ports	2	initialize both SCC ports.
iniq	2	initialize the input buffer
init_ports	1	initialize the ports
acia_setup	1	setup the ACIA for communication
outacia	1	output function using the ACIA

Table 5.2.1 Software functions for the control unit.

The program is implemented in a structured manner. The use of independent

functions to implement the different tasks makes the software modular, easy to modify, debug, and update. This program contains three parts: initialization, input/output programming and operation.

The functions implemented in this part are described in Table 5.2.1. This table shows the name of the function, its layer, and its operation. The functions that are classified as layer 1 are the ones that perform operations directly on the Integrated Circuits (IC). Functions classified as layer 2 prepare the information received from layer 3 to pass it in the appropriate way to layer 1. The rest of the functions and layers operate analogously. Layer 7 is the application running on the MC68HC11 board, which is the main program running the initialization, the input/output programming, and the full operation.

5.3 Control Unit Software Operation.

The control unit, following the startup, performs an endless loop polling the components.

5.3.1 Initialization.

Initialization sets the variables to their starting values, and also sets the port parameters to desired mode of operation. It disables interrupts. Then, each of the port is setup to perform the operations it is assigned. Functions like init_port, aciasetup, iniq, init_scc_ports, enable_scc, and set_control_tbl are executed. After this, interrupts are enabled, and the microcontroller is ready to start the operation.

The ACIA is programmed to handle the communication with the computer. It is set to communicate at: 9600 bps, with no parity, eight data bits, and 1 stop bit (9600 N 8 1). Its receiver will request the microcontroller, indicating the it has a character that needs to be read. This method is used because characters arriving might be lost if the are not read fast enough. The transmission is performed by polling the transmitter control register to check if the transmitter is ready to send data.

The parallel ports A, B, and C are programmed in the following way: PORTC is assigned to have the same functionality of the Data Bus in the microcontroller. PORTB becomes the Address Bus, and PORTA becomes the Control Bus. After these ports are initialized, the I/O subsystem is set up. Port A controls the operation of the external circuitry. Port B addresses the DUART's internal registers, and port C is used to input and output the byte to be sent to each device attached to the MC68HC11 board.

Tables 5.3.1.1 shows how ports A and B are set up to perform their functions as control and address busses respectively.

Port	Function	7	6	5	4	3	2	1	0
A	Control	HDBR	LDBR	B	A	$\overline{CS}$	DIR	$\overline{EN}$	---
B	Address	----	----	----	R / $\overline{W}$	RB4	RB3	RB2	RB1

Table 5.3.1.1. Bit definition for Port A and Port B as Address and Control Busses respectively.

From Table 5.3.1.1, for Port A: HDBR and LDBR are High and Low Data Bit Rate respectively. They correspond with 19.2 kbps (bit 7), and 0.6 kbps (bit 6). A and B select the devices to be controlled 00 = device 1, 01 = device 2, 10 = device 3, and 11 = device 4. $\overline{CS}$ controls the DUART's control line. DIR controls the direction for the

buffer used as data bus (input and output). $\overline{EN}$ enables the same buffer. For Port B: RB1 through RB4 control the DUART's internal registers. $R/\overline{W}$ controls the read and write process for the DUART.

When all the ports are programmed to perform their functions, the external DUART can be programmed. The DUART has two serial ports. One port runs at 600 bps and the second at 19200 bps. Table 5.3.1.2 shows the way in which the two Serial Communication Controllers A and B (SCCA and SCCB respectively) are programmed.

SCC	DBR (bps)	Data Bits	Parity	Stop Bits
A	600	8	No	1
B	19200	8	No	1

Table 5.3.1.2. SCCA and SCCB programming mode.

The way the DUART's programming is done is by calling the function init_scc_ports (layer 2). This function sets the accumulator A (Acc. A) and accumulator B (Acc. B) of the MC68HC11 with the DUART's internal register address and the value to be programmed respectively. Then the function write_to_scc (layer 1) is called. This function lowers the DUART's Read / $\overline{\text{Write}}$ ($R/\overline{W}$) line, then writes the value stored in Acc. A into Port B (Address), and writes the content of Acc. B into Port C (Data). After this, Port C is programmed as an output by writing an 0xFF Hexadecimal (Hex) to register DDRC. Then, the DUART's Chip Select ($\overline{CS}$) is forced low by resetting bit 3 and sending that value to Port A (Control). This sequence guarantees the proper timing to write a byte to the DUART. $\overline{CS}$ is then disabled. The $R/\overline{W}$ line is set and Port C is programmed as an input to guarantee a high impedance state in Port C. This process is

done for each of the DUART's registers for SCCA and SCCB. After this process is done, the unit and the external circuitry are ready to start checking for the devices that are connected to the control unit to be controlled. This is done in the following step, which is the full operation process.

5.3.2 Full Operation.

A simple protocol was developed to handle the transfer of command between the computer and the microcontroller and vice versa. The protocol works in the following way: a small case 'c' opens the frame, then a second letter follows, and a third character follows. The second letter is the command to be executed by the microcontroller. Table 5.3.2.1 shows each character and its relation with the command to be executed by the control unit.

Each of these commands needs three bytes. For example to add a device to the control unit, the computer sends a command like: 'c' 'a' 1. This will be interpreted by the control unit as: a command is received; this command will add a device to the devices to be controlled in position 1. The control unit adds the device and sends to the computer an "all right" (OK) response or "bad" response. The way that the microcontroller sends the response back to the computer is: 'c' 'o' (for OK) or 'c' 'b' (for bad). These commands and responses will be issued in lower case letters. Capital letters or other type of letters that start a frame are echoed back from the control unit to the computer. This a way for the computer to detect if the control unit is still connected to the serial port.

The devices attached to the control unit are represented in a position in the control

Character	Function	Operation
c	-----	it indicates: command
a	add device	add a device to the control unit
d	disable device	disable the device (not controlled)
e	enable device	enable the device (controlled)
r	remove device	remove a device from the control unit

Table 5.3.2.1. Character-command correlation.

unit memory space. That position in memory holds a byte which bits have different meaning. Table 5.3.2.2 shows what each bit represents for each device in memory.

Bits	7	6	5	4	3	2	1	0
	DAD	DAC	HDBRP	LDBRP	----	----	----	----

Table 5.3.2.2. Bit representation for each device in memory.

From Table 5.3.2.2 each bit means:

- DAD: Device Added, it is set when a command to add a device is received by the control unit.
- DAC: Device Active, this bit is set when a command to enable a device is received by the control unit.
- HDBRP: High Data Bit Rate Problem. When a device that is active is detected with a problem using the High DBR check method, this bit is set, otherwise it remains 0.
- LDBRP: Low Data Bit Rate Problem. When a device that is active is detected with a problem using the Low DBR check method, this bit is set; otherwise, it remains 0.

When an "add device" function is issued the byte for the device that is added is cleared, and bit 7 is set. When the "enable device" is issued, bit 6 is set. The combination of these two bits set indicates that a component is present and active; the control unit must poll that component to detect any failure. On the other hand, if any of those bits is not set, the control unit will not poll that component.

The enable and disable commands were created to allow the movement of the devices from one position to another without having to deactivate the whole system.

After which command is in the buffer, program then goes to check for the devices. It checks the devices by sending a byte, using a the low dbr form, to each of the active devices in memory. If no device is present, it does not perform any further checking. If a device is active, then it sends a byte to that device by setting the proper control and address signals. Then, it waits until a byte is received. If the received byte does not correspond with the one sent or it does not receive any byte at all, bit 4 in the byte of memory reserved for that device is set. Otherwise it proceeds to check the next active device.

The program then follows the same procedure for the high dbr. When both checks are finished, then the status bytes for the devices are scanned. If any of bits 4 and 5 for any device is set, a command containing the device position is sent to the computer to alert it of a problem with that device.

This is the general way that the control unit operates. This operation does not depend of the computer. If the computer is disconnected after assigning the devices to each position, the unit is able to continuously operate protecting the devices in a stand

alone mode. Of course, another kind of operation should be developed to guarantee a stand alone control unit. However, this operation is out of the objective of the present work.

The second part of this system is the computer and the way it functions. This part is the one that makes the interaction between the user and the system more friendly. Also, it is the one that allows the control unit to complete its control operation by sending the information over the ISDN lines. The ISDN lines are connected to the computer through the Terminal Adapter (TA) attached to second serial port.

5.4 Computer Operation.

The software to run in the computer was designed to make the communication between the control unit and the ISDN line, and to serve as an interface between the user and the control unit.

It is structured in three main classes: DEVCOM, SERCOM, and MENU. The two first classes handle all the functions related with the initialization, control and utilization the serial ports in the computer; to receive and transmit information from, or to the TA and the control unit. In this case COM1 is used to communicate with the control unit and COM2 is used to communicate with the TA.

To accomplish an effective serial communication operation, it is necessary to have an interrupt driven operation for the receiving path; especially, when high dbr is needed for transmission of a high volume of information such as found in images, pictures or video. This is critical, because data can be lost if computer does not respond fast enough

to the reception. To accomplish this, the interrupt service routines for the serial ports are changed to functions that place in memory the received characters and update pointers. Then, the software can be able to check the buffers to detect any new characters. This allows having the two serial ports receive information simultaneously without losing any characters.

Functions	Layer	Comments
rx_char_com1	3	To handle all communication with the control unit
is_char_in1	2	To detect if a character is the receiver 1 buffer
purge_buffer1	2	To update pointers if no data is on the buffer
send_char1	1	To send a character using COM1
send_string1	1	To send a string of characters to COM1
transmitter_ready1	1	To detect if serial port is ready to send a character

Table 5.4.1 Functions defined in DEVCOM class.

DEVCOM is the class that define the functions to communicate with the control unit. Accordingly with the OSI model, the functions defined in this class are related with layers 1, 2, and 3 of the model.

Table 5.4.1 shows the functions and their peer layer. The function rx_char_com1 is in charge of managing all the communication process between the computer and the control device.

The function handles the commands sent by the control unit, and also send commands to the latter. The functions in layer 2 receive the information, process it, and call the functions in layer 1, which interact directly with the computer's hardware to send or receive the information from the control unit.

The second class SERCOM also defines functions in layers 1, 2, and 3 of the OSI model. This class is more complex than DEVCOM because the interaction with the TA is

more sophisticated than is that of the control unit. Moreover, the information to be sent to the TA needs to be correctly formatted. That is why this class has functions that can be placed at the same layer level as layers 4 and 5 of the OSI model. Table 5.4.2 shows the functions defined in this class and their peer layers

Functions	Layer	Comments
chat_mode	6	To handle the operations with COM2
tx_rx_mode	6	To handle the communications with the TA
rx_char_chat	5	To handle the characters from/to the console
rx_char_txrx	5	To handle the characters from/to the TA
select_mode	4	To select transmission or reception
csd_setup	3	To setup the TA for operation
get_time	3	To get the time from the system clock
room_number1	3	To get the room number to be sent
proc_char	3	To get character from buffer and process it
send_atcom	3	To send a string of AT commands to the TA
day_selector	3	To select the day from the system date
update_ptrs	2	To update the pointer if no characters in buffer
delay	2	To set an specific time delay using the system clock
send_string	2	To send a string of character using COM2
character_in	2	To check if a character is in the buffer
dial_up	2	To send the string of number to stablish the connection
header_block	2	To structure the information to be sent as a block
disconnect	2	To disconnect the ISDN line when the information is sent
open_port	1	To open the COM ports
setup_port1	1	To setup COM1
setup_port2	1	To setup COM2
transmitter_ready	1	To detect if the transmitter is ready for transmission
send_char	1	To send a character using COM2

Table 5.4.2 Functions defined in SERCOM class.

The functions chat_mode and tx_rx_mode are called by the application layer (7). These functions are in charge of the whole operation. chat_mode is designed to handle the

process when the computer is not communicating ISDN; it handles the interaction with the user through the console, and the communication with the control unit. tx_rx_mode handles the communication with the TA when the computer is transmitting information to another computer using ISDN.

chat_mode operates in a loop, which is broken if Carrier Detect (CD) bit in the status register of the COM2 port is set. This means that a connection has been stablished, and immediately, it switches to tx_rx_mode. Before any connection is established, it checks if any information has been received from a remote computer (connection is established by an incoming call). If no information has been received, it checks to see if any command has been received from the control unit. If none, it checks if any key has been pressed. If not, it checks CD. If no connection, then it restarts the cycle. This function has a built in counter that always is reset to zero when any character is received from the control unit. If no character has been received from the control unit after it reaches certain count, a flag is set and the function forces a connection to the main security office. This will inform that the control unit is not longer connected to the computer.

When a command is sent by the control unit, the third class MENU, which handles the operations in the computer memory is invoked. Also, it forces a call to the main security office. The information that it sends in this case is different from the previous one sent when the control unit and the computer lose contact with each other.

tx_rx_mode handles the communication process when a connection through the ISDN lines is active. It checks if any character is in the receiver buffer. If a character is

present, the function displays it in the console. If no character is present, then it checks if the connection was forced for the computer. If this is true, then it selects the proper operation (accordingly with the flag that was set). It also sends the information to the main security office, and automatically ends the connection and returns to the chat_mode function.

The third class MENU operates at higher levels than the previous two. This class is in charge of the processing of adding, enabling, disabling, and removing devices from the control unit. Moreover, if a command is received from the control unit, it indicates that a device has a problem.

This class is in charge of counting the number of problems for that device and preparing the information to be sent using the ISDN lines. The function members of this class can be placed in layers 6, 5, and 4. Table 5.4.3 shows the function members of this class and its relation with the OSI layer model. It works by placing a menu into the computer screen in which the functions to be executed are listed. When the user selects determined function, a process is performed to guarantee that the proper information is kept in memory. Then, the right command is issued to the control unit to be executed by this unit.

Function_menu is the function called when the user wants to send commands to the control unit. It is called from the chat_mode function previously explained. This function starts showing the main screen where the functions that can be performed by the control unit are displayed.

Each of the functions will start a process that allows the user to send a command function toward the control unit in a transparent mode (the users do not see this interaction.) These functions are the same functions explained in Table 5.3.2.1, (add, disable, enable, and remove). There also are two more functions: list and exit. For the former functions, the processes of adding, disabling, enabling, and removing are very similar to the ones executed in the control unit. Its only difference is that the computer after processing the information sends the command to the control unit. The function list gives general information to the user based on the type of device that is attached to each position, whether the device is active or not.

Functions	Layer	Comments
function_menu	6	To call the menu operations
show_device_menu	5	To show the device menu
show_device_enabled	5	To show devices that are enabled
show_device_disabled	5	To show devices that are disabled
show_screen	5	To show the first menu selection
add_function	5	To start the add device process
disable_function	5	To start the disable device process
enable_function	5	To start the enable device process
remove_function	5	To start the remove device process
list_function	5	To start the list devices process
exit_function	5	To exit the menu operations
check_keyboard	4	To get values input through the keyboard
set_position_flag	4	To set the flag indicating position used
set_device_active	4	To set the bit to indicate that device is active
set_device_inactive	4	To set the bit to indicate that device is inactive

Table 5.4.3. Functions defined in the MENU class.

These functions operate based on a structure that keeps the information for each of the devices attached to the control device. This structure has defined the following members: type, position, pos_used, enabled, problem, had_problem, numb_of_problems.

Type indicates what kind of device is attached to the control unit, for example: computer, monitor, printer, scanner, oscilloscope, etc. In this way each device is individually characterized in this security network. Position indicates the position where the device is going to be placed. Pos_used indicates if the position is already taken by other equipment. Problem is a flag that indicates if the control unit detected a problem with that device. Assuming that the problem was fixed, the unit still keeps a record of any previous problem with that device, and also the last member, numb_of_problems indicates how many problems that device had had before. In this way the computer keeps track of everything that is happening to the devices attached to the security network.

Chapter VI. System Integration and Future System Enhancements.

6.1 System Integration.

SECNET is based on a control unit and a computer operating together. The control monitors components connected to it. The computer runs the software that allows to send information over ISDN lines to a main security office.

Their operation is combined into one system. Its function is to secure the equipment in certain room and to communicate if something is not right with component under protection.

6.1.1 System Operation.

The control unit starts its operation when powered up. It starts by checking if any device is connected to the system, and if it is any active. Also, it starts sending information back to the computer about the status of the equipment connected to the unit if any. After the control unit starts, the software in the computer is started. When the software starts, it requests to the user that identify what operation it is going to be executing: a main security office or as part of the security of a room. This is necessary because the software operates in a different way if used in either one positions.

6.1.2. Main Security Office Operation.

The operation as a main security office is a basic one. It will check COM1 and COM2 for any incoming information; however, it will not have enable the detection of an active control unit for COM1. Also, it does not have to call a main security office because

it is working as one. Its function is to display any incoming information from the ISDN lines into the screen to alert the security personnel in the office of any problem in the rooms connected to the system.

6.1.3. Room Security Operation.

The software always starts asking if it will run as a main security office or not. By answering no, the steps that follow make it works as a room security system. Immediately, the user is asked for the room number and for the phone number of the main security office. These two steps guarantee that the correct communication is stablished and the right information is sent.

After this point, it starts checking COM1 and COM2. As previously stated, the control unit is already running and it is connected to COM1. The control unit is continually sending information to the computer. This information is the status of the devices connected to the system if any. The computer uses this information to reset a counter that is incremented if no communication is received from the control unit. If this counter is incremented up to certain value, this means that the control unit is no longer connected to the computer. Then, an alert signal is sent to the main security office indicating so. This signal is sent constantly until the problem is corrected.

In order to add equipment to be controlled by the system, the F1 key should be pressed. This brings up to the screen a menu showing all the functions that can be executed by the system. This menu is shown in Figure 6.1.3.1. It lists each function

associated with a number. By pressing the corresponding number the desired function can be executed.

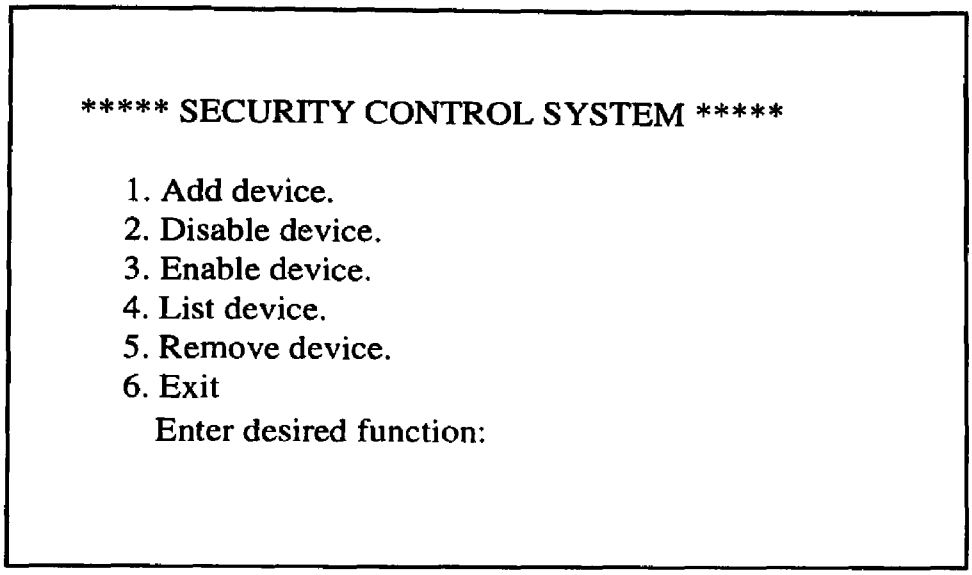


Figure 6.1.3.1. Main menu listing.

To add an equipment to the control unit, item one should be selected. This will start a process by bringing a second menu. This menu shows a list of equipment that can be added to the system. Again, each equipment is identified by a number. The user easily selects a determined equipment just by pressing the proper number. By selecting the component to be added, the equipment type is kept in memory and it is the one that will be sent if a fault is detected. Figure 6.1.3.2 shows the equipment listing for this screen.

After an equipment is selected, a third menu is shown. This menu shows the positions that have been taken, and it requests in what position the equipment being added is going to be placed. This screen shows the four positions that this system has available for the equipment. The screen is shown in Figure 6.1.3.3. It represents a hypothetical situation where two component have already been added to the system in

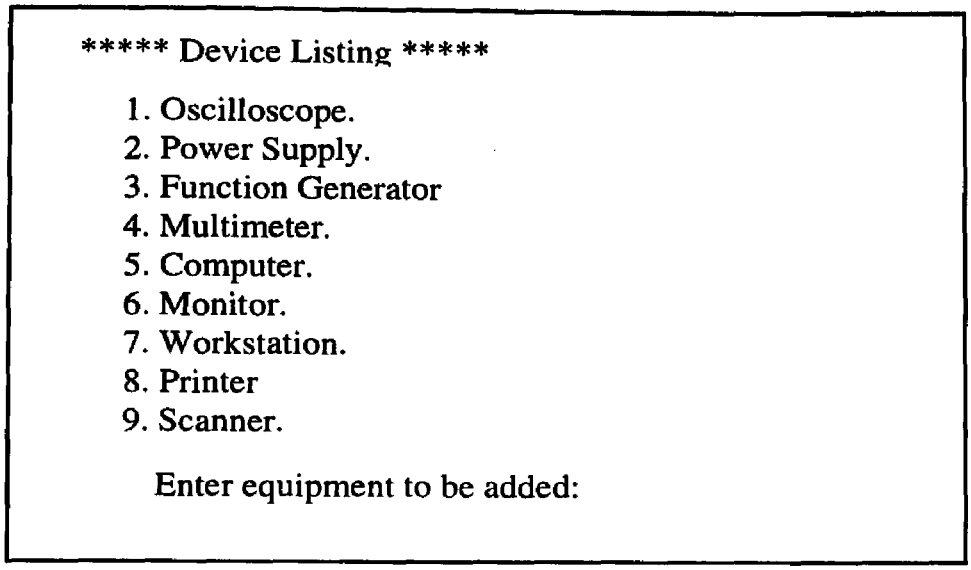


Figure 6.1.3.2. Component listing for the add component function.

positions 1 and 3, and position 2 and 4 are available.

After the position is entered the software sends a command to the control unit indicating that a component was added to position (as an example) two. The screen returns to the main menu represented in Figure 6.1.3.1 where it will wait for a new command or to exit to the main program.

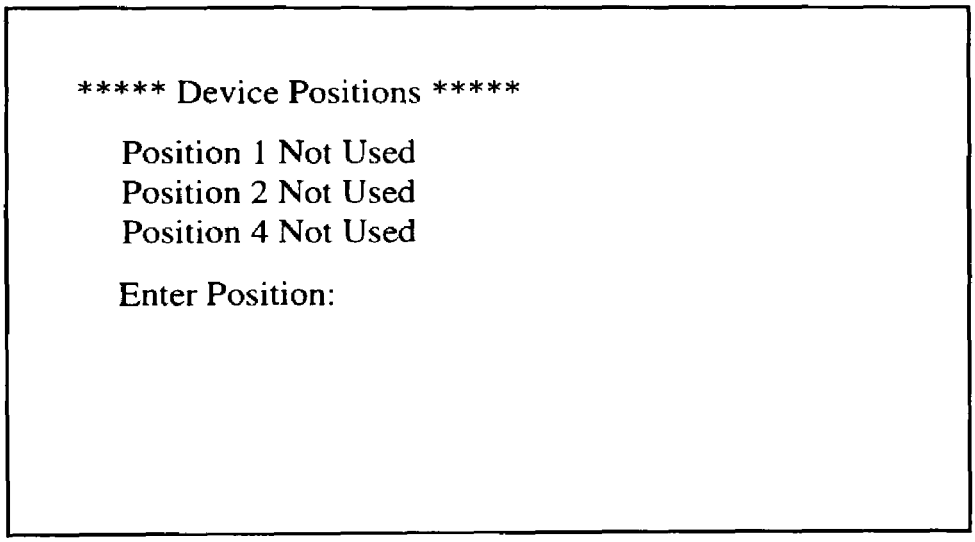


Figure 6.1.3.3. Screen showing the positions not taken.

To enable a device, the number that corresponds with enable device should be pressed.

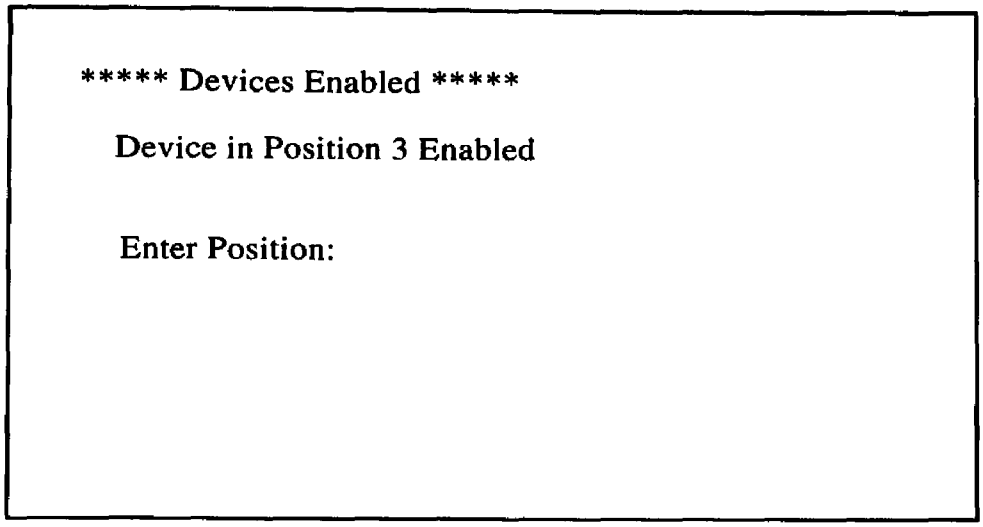


Figure 6.1.3.4. Screen showing the enabled positions.

A screen showing the positions that are added but that are still disable, is shown. This screen is shown in Figure 6.1.3.4. When the position to be enabled is selected, the enable command is sent by the computer to the control unit indicating to make the position selected active. This makes the control unit start checking the devices made active. This procedure is repeated for each device that is added and activated.

At this point the component that is going to be controlled must be connected to the control unit. Otherwise, the control unit will start sending commands to the computer indicating that something happened to that component.

6.1.3.1 Control Device.

The control device attached to the component under control has an internal connection for a jumper that disconnects the internal circuitry from the signal send by the

control unit. This condition will be detected by the control unit and immediately will send a signal to the computer indicating that the component in that position has failed the scan.

The control device is shown in figure 6.1.3.1.1. This figure shows a 3-D view of the control device and the jumper, to show how they are connected. The control device has the internal circuitry that allows signals no faster than 600 bps to pass through without any attenuation. On the other hand, signals faster than 9600 bps are attenuated. These characteristics match the specifications explained in Chapters 4 and 5 for the operation of the system. As it can be seen from Figure 6.1.3.1.1, the control device's body completely covers the jumper, avoiding the easy separation of the jumper from the control device's body.

The jumper block is attached to any part of the equipment to be protected. This part of the equipment selected should be one that is not easy to separate from the rest of the equipment, or that can be easily broken. One advantage of this method is that the component does not have to be opened or altered to attach the control device. Just by gluing the control device with a very powerful glue to the position will allow the system to operate.

If someone tries to separate the control device from the component, the full body is separated from the jumper. This opens the path for the signals and the low dbr control byte will not be received by the control unit. It will be the same if the line from the control unit to the control device is disconnected. On the other hand, if someone tries to tap into the line by short-circuiting the lines, (to externally emulate what the control

device is doing), a high dbr control byte will be received by the control unit yielding to a faulty operation for that component.

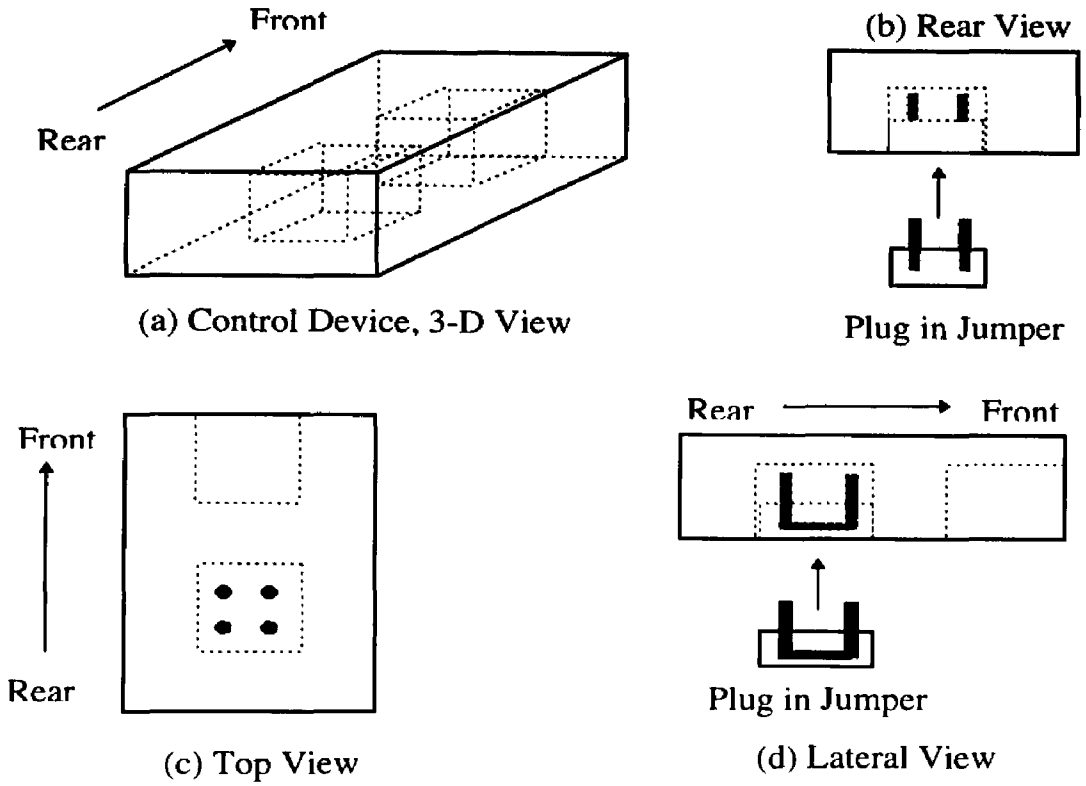


Figure 6.1.3.1.1. Control device positions: (a). 3-D view, (b). Rear view, (c). Top view, and (d). Lateral view.

The system is flexible enough to allow that the component be removed (e.g., to be repaired) without removing the control device or only leaving the jumper attached to the body of the component. This is done by the software by just disabling the position where the component is placed.

The last function is list component. This function will list the component attached to the control unit, their status, if they either have or had problems, and how many times they had problems.

6.2 System Signaling.

When the system is completely functional, it will be constantly checking if there is any information to be processed. If the control unit detects that a component has a problem, it will send a command to the computer with the position of the component with problem. The computer will receive this information, and immediately will send it to the main security office adding more information. The information added is the room number, number of times that problems have been detected in that room, date, time, type of component with the problem, and position in the system. This information will help the security personnel to exactly identify where the problem is located, when that problem happened, what kind of component they have to look for, and the exactly position inside the room where the component was placed. Figure 6.2.1 shows the screen at the main security office when a computer located in position 1, was disconnected from the control unit in room ECS258, at 11:15:02 AM Thursday, February 23, 1996.

A second condition is that the control unit is being disconnected from the computer. In this case, the computer will send information indicating the room number, the date and time, and a message indicating that the control unit has been disconnected from the computer. As previously stated, this information will also help the security personnel to know where the problem is located, when it happened at the cause of the problem. Figure 6.2.2 shows the computer's screen at the main security office when a control unit was disconnected from the computer.

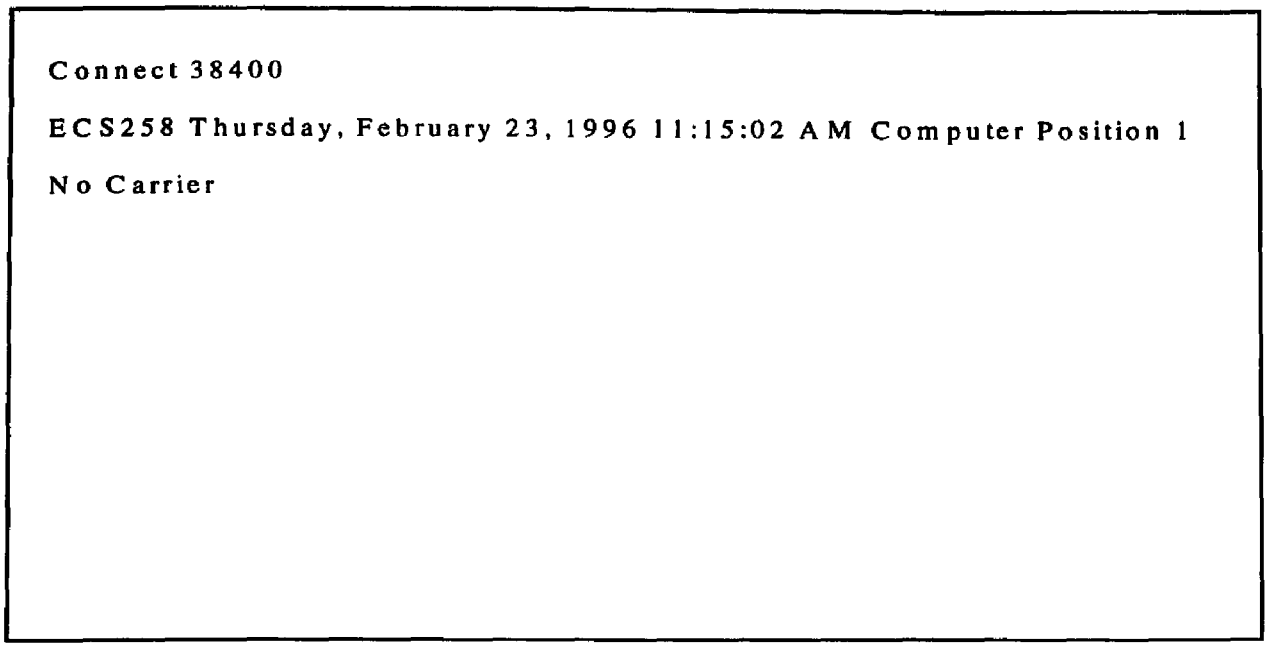


Figure 6.2.1 Screen at the Main Security Office (MSO) when an component was disconnected from the control unit.

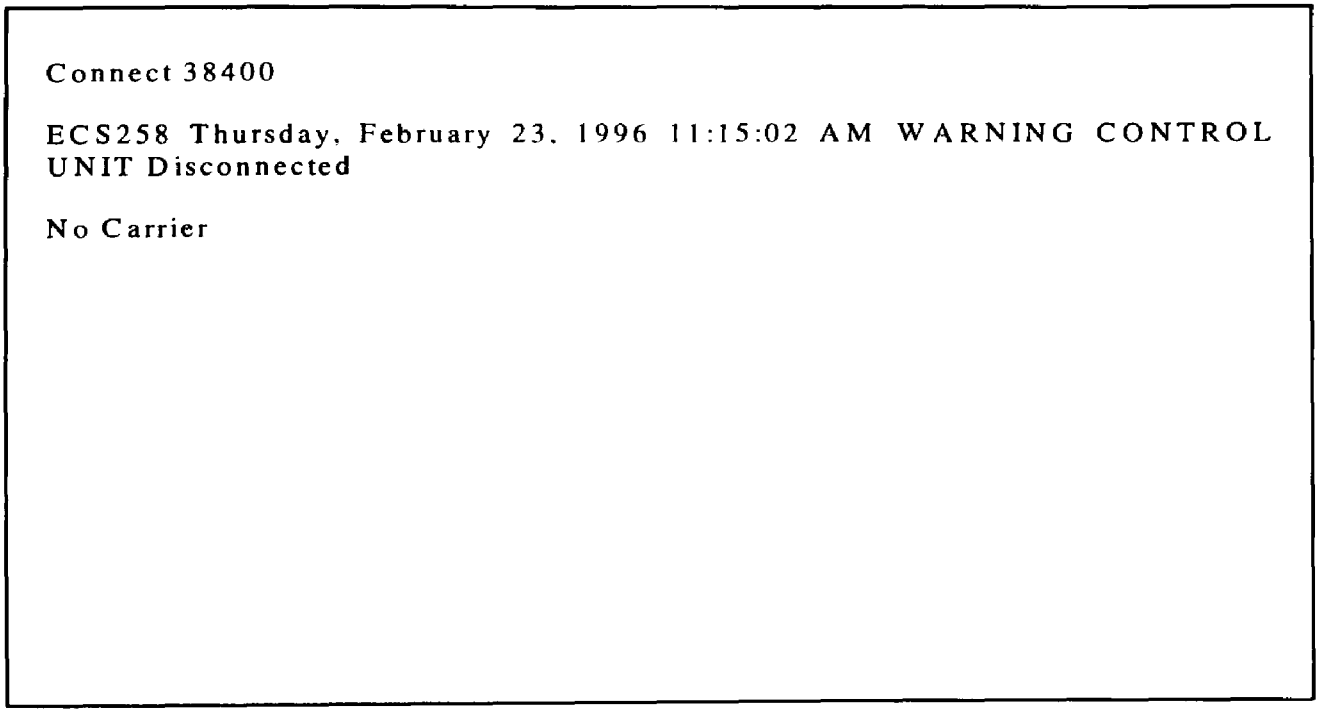


Figure 6.2.2. Screen at the Main Security Office (MSO) when a Control Unit was disconnected from the computer.

6.3 Future System Enhancements.

The SECNET system has great potential and wide scope for further development. There are new applications developed to utilize the full capacity of ISDN. These applications can be integrated to SECNET to obtain a more versatile system capable of offering a wide variety of uses in the security field.

6.3.1 Data Communication.

New techniques in the data communication area like data compression techniques, inverse multiplexing (bonding) and Point-to-Point Protocol (PPP) will increase the throughput of the data transmission over the ISDN lines. This allows that high resolution still images, full motion video and interactive video signal can be sent through ISDN lines. The latter two, bonding and PPP, allow the merging of channels in the ISDN lines. For example, for Basic Rate Interface (BRI), each B channel can go up to 64 kbps. By using those techniques, the speed of the communication line can be doubled to up to 128 kbps. Bonding was developed years ago. The way it operates is by synchronizing both B channels at the beginning of the connection, and keep it that way until the end of the connection. On the other hand, PPP is able to utilize the second channel statistically based on the demand for more bandwidth. The connection can be stablished with only one channel and if during the connection the second channel is needed, the PPP protocol will add that channel until it is no longer necessary. This will preserve a connection for other uses where it can be needed. Nowadays, PPP protocol is being selected as the default standard for obtain more speed from the communication lines.

PPP together with data compression, can increase the throughput of the data transmitted over the ISDN lines, to the point that high quality still picture and images can be sent in matter of seconds between two or more very remote places. Also, full motion images can be sent with reasonable quality using ISDN lines.

SECNET can incorporate these advantages and allow the security personnel in the main office to check by the usage of cameras each of the rooms under control. Also, by adding new devices to the system, it can check the person's or persons' identity in a matter of seconds allowing to exercise a more strict control over critical areas.

6.3.2 Networking.

The ability of interact in a complex system where different devices are connected and able to share information among them is what is called networking. Example of networking is the Local, Metropolitan and Wide Area Network. LAN, MAN and WAN respectively. Each of them has its own distinguishable characteristics. SECNET can take advantage of the three of them. However, because its characteristics of localize operation, the most useful will be LANs.

Ethernet addresses can be built into the control devices. The control unit will work as a server where a number of component under protection can be polled a higher speed to detect a failure on them. This will increase the number of component that a single control unit can control make the system more powerful, and at the same time, more economical. Figure 6.3.2.1 shows an implementation of SECNET as a LAN.

By allowing SECNET to be connected to a LAN, new possibilities for transmitting information, and control can be developed. For example, ISDN can be used as a back up link in case that the LAN is not operational. This will allow SECNET to be a fault tolerant system. If one of the communication links is broken the second can be used. This also brings the possibility of using Transmission Control Protocols / Internet Protocol (TCP/IP). This protocol is used to transfer data between two logically distinct user processes, such as file transfer or electronic mail servers. TCP/IP allows a reliable transmission of large files of data. These large files of data can be still pictures taken in a room under surveillance to be analyzed in the main security office [17].

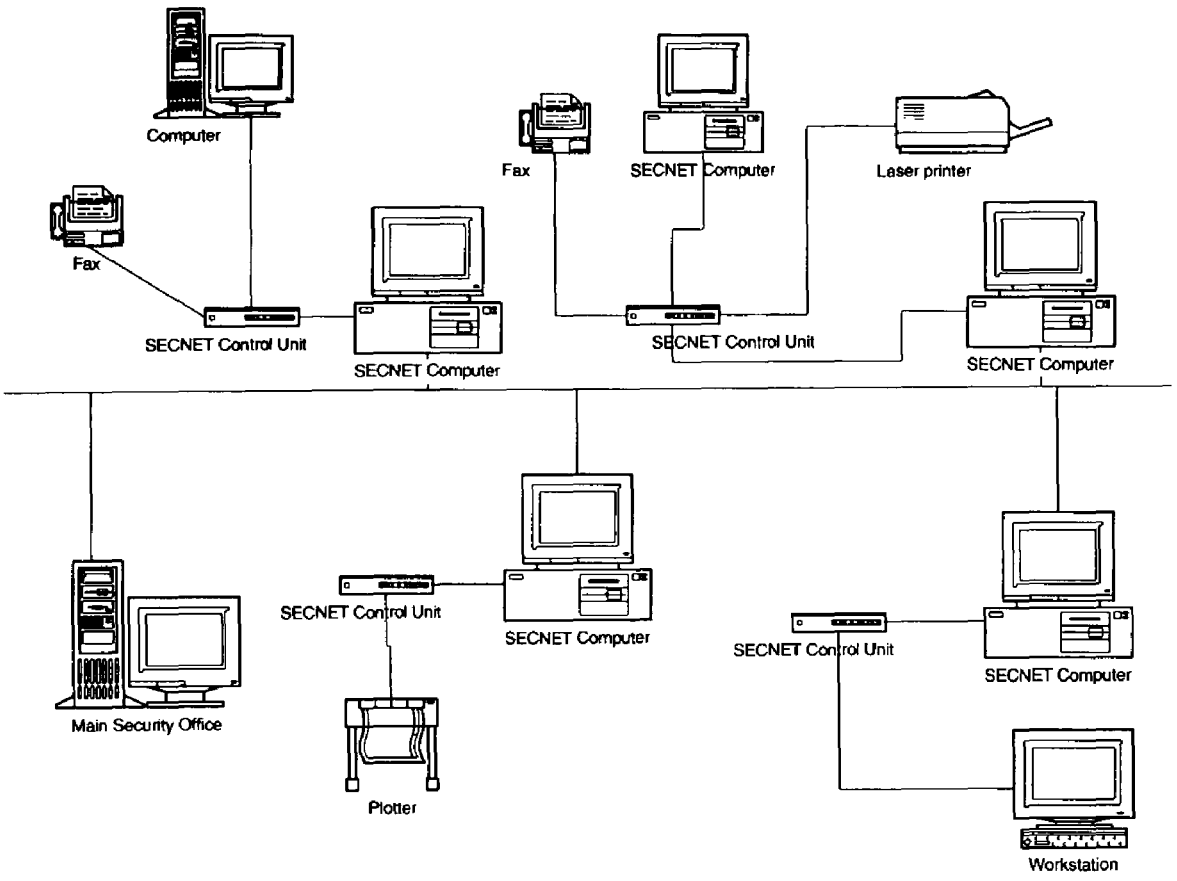


Figure 6.3.2.1. Local Area Network (LAN) implementation of SECNET.

6.3.3. Wireless Technology.

Wireless devices can be incorporated to SECNET in order to make the system even more reliable. SECNET can use the advantages of wireless technology as a way to backing up the main communication channels making the system less prone to fault ISDN lines or LAN connection.

By having wireless capabilities, SECNET will be able to send information to neighboring rooms attached to the system. Those rooms still have capacity of connecting to the main security office. Also, by using cellular technology, SECNET will be able to keep a secondary communication path with the main security office. Moreover, wireless LAN can be used to control different control units to the same computer making the system more efficient, flexible and economical.

6.3.4 Software.

Additional features can be added to the system software. These features can be added as a form of software upgrades. Among those features are the addition of database capabilities using encryption, the SQL programming language, database security, Zmodem File transmission protocol, etc.

Encryption is the process of transforming data into an intelligible sequence of binary information. This information is very difficult (if not impossible) to decrypt if the intruder does not have the key to transform those bits back to the original data. This is necessary because there is a security concern of using ISDN for data communication as well as any public network [18].

The SQL programming language is a very powerful tool in adding query related features to the system. Monitoring features as inventory schemes, room failure per day, month or year and many more can be implemented [18].

Database security will implement any access violation to SECNET; username and password to access the computers and the control units; logon and logoff records are some of the features that can be added to SECNET [18].

Zmodem file transmission capabilities that enable SECNET to send data in a continuous fashion, without waiting for acknowledgment of individual blocks. It also works in conjunction with software handshaking such as XON/XOFF [19]. This would allow SECNET to be more reliable when sending data because the data is being checked twice for transmission/reception errors.

These are the most important possible future enhancement for SECNET. These enhancements will make SECNET a very powerful security system that can be able to communicate with any other communication device because of its open system design philosophy.

Chapter VII. Conclusions.

SECNET was developed to create a new way to utilize new technologies in the security industry. These new technologies like Integrated Services Digital Networks (ISDN) makes this system faster and more reliable than one that uses normal telephone lines.

The task was divided into two parts. One port uses the Terminal Adapter (TA) to stablish a link with the main security office using ISDN. This is a computer, which has one of its serial port connected to the TA. The software can run under Windows™ as a DOS application, or it can run directly from DOS. This software is a specialized software communication package that is designed to continually check both serial communication ports in the computer. It is able to interact with the TA, and also with a control unit connected to the second serial port.

The second part is an intelligent control unit based on a microcontroller and a special external circuitry that allows the constant supervision of the component connected to it. A control device is attached to each component under control. If that control device is removed from the component, the control unit sends a command to the computer indicating the position where the problem was detected. The computer receives the information and sends it to the main security office via ISDN.

The objectives of the present work were accomplished. A security system, SECNET, was designed and a working prototype was created. This system has new characteristics that make SECNET unique. These characteristics are: individual

identification of component under surveillance, the utilization of control bytes that prevent the equipment to be either disconnected (opening the circuit), or make them appear to be connected (short-circuiting the control device). Also, the system can not be tapped to try to simulate the control process by transmitting the control bytes that are being sent by the control unit, using another device connected to the system to tap the communication with the equipment.

REFERENCES

[1] Zalud, Bill "What's Happening To Security?". Security, September 1990, pp 42-48.

[2] Russell, Rebecca D. "Security Costs and The.. Bottom Line". Security, June 1990, pp 42-48.

[3] Zalud, Bill "Charting Security's Course". Security, January 1990, pp 29-34.

[4] Helgert, Hermann J., Integrated Services Digital Networks: Architectures, Protocols, Standards, Addison-Wesley Publishing Company, Reading, Mass. 1991.

[5] Goldsten, Fred R., ISDN in Perspective, Addison-Wesley Publishing Company, Reading, Mass. 1992.

[6] Stalling, William, Data and Computer Communications, 4th Edition, Macmillan Publishing Company, New York, 1994.

[7] Stalling, William, ISDN and Broadband ISDN, 2nd Edition, Macmillan Publishing Company, New York. 1992.

[8] Goodwin, Mark, Serial Communications in C and C++, Management Information Source (MIS) Inc., New York, 1992.

[9] Friend, George E.; Fike, John L.; and others, Understanding Data Communications, Texas Instruments Information Publishing Center, Howard W. Sams & Company, Indianapolis, 1988.

[10] Monk, Timothy S., Windows Programmer's Guide to Serial Communications, Sam Publishing, Indiana, 1992.

[11] Campbell, Joe, C programmer's Guide to Serial Communications, Howard W. Sams & Company, Indiana, 1988.

[12] The Engineering Staff of TI Inc. Semiconductor Group, The TTL Data Book for Design Engineers, 2nd Edition, Texas Instrument Incorporated, 1976.

[13] Motorola Inc. M68HC11EVB Evaluation Board User's Manual, Motorola Inc., 1986.

[14] Rumbaugh, Janus and Others, Object-Oriented Modeling and Design, Prentice-Hall, Inc. Englewood Cliff, New Jersey 1991.

[15] Green, Curtis, "Hardware Modeling in C++". Embedded System Programming, Vol. 8 No. 10, October 95, pp 24-54.

[16] Dewhurst, Stephen, Programming in C++, Prentice-Hall, Inc. Englewwod Cliff, New Jersey 1989.

[17] Markley, Richard W., Data Communications and Interoperability, Prentice Hall, Englewood Cliffs, New Jersey 1990.

[18] William, Mark, Development of a Retail Network Using ISDN. Florida International University, Miami, Florida 1993.

[19] Nelson, Mark, Serial Communications: A C++ Developer's Guide, M&T Books, San Mateo, California, 1992.

APPENDIX A

```
* ************************************************************************************ *
* *                    SECurity NETwork (SECNET) Software HC11                       * *
* *                          Author: Isidro Alvarez, BSEE                            * *
* *                 Department of Electrical and Computer Engineering                * *
* *                          School of Engineering and Design                        * *
* *                          Florida International University                        * *
* *                                Miami, Florida, USA                               * *
* *                                    Spring 96                                     * *
* ************************************************************************************ *

* ************************************************************************************ *
* *                                  Definitions                                     * *
* ************************************************************************************ *

BASE_ADDR   equ     $1000
porta       equ     $00                     Port A address offset
portb       equ     $04                     Port B address offset
portc       equ     $03                     Port C address offset
pactl       equ     $26                     Port A control addr. offset
portcl      equ     $04                     Port C Latched Reg. addr. offset
ddrc        equ     $07                     Port C Data Dir. Control Reg. offset
oc1m        equ     $0D                     Output Comp. Mask Reg. offset addr.
tcnt        equ     $0E                     Timer Count. addr. reg. offset
toc1        equ     $16                     Timer Output Comp. addr. reg1 offset
toc2        equ     $18                     Timer Output Comp. addr. reg2 offset
tctl2       equ     $21                     Timer Control 2 addr. reg. offset
tmsk1       equ     $22                     Timer Mask 1 addr. reg. offset
tflg1       equ     $23                     Timer Flag 1 addr. reg. offset
tmsk2       equ     $24                     Timer Mask 2 addr. reg. offset
tflg2       equ     $25                     Timer Flag 2 addr. reg. offset
PORTA       equ     BASE_ADDR+porta         Port A address
PORTB       equ     BASE_ADDR+portb         Port B address
PORTC       equ     BASE_ADDR+portc         Port C address
PACTL       equ     BASE_ADDR+pactl         Port A Control address
PORTCL      equ     BASE_ADDR+portcl        Port C Latched Register address
DDRC        equ     BASE_ADDR+ddrc          Port C Data Dir. Control Register
OC1M        equ     BASE_ADDR+oc1m          Output Compare Mask Reg. addr.
TCNT        equ     BASE_ADDR+tcnt          Timer Countet address register
TOC1        equ     BASE_ADDR+toc1          Timer Output Compare addr. reg1
TOC2        equ     BASE_ADDR+toc2          Timer Output Compare addr. reg2
TCTL2       equ     BASE_ADDR+tctl2         Timer Control 2 address register
TMSK1       equ     BASE_ADDR+tmsk1         Timer Mask 1 address register
TFLG1       equ     BASE_ADDR+tflg1         Timer Flag 1 address register
TMSK2       equ     BASE_ADDR+tmsk2         Timer Mask 2 address register
TFLG2       equ     BASE_ADDR+tflg2         Timer Flag 2 address register
ACIA        equ     $9800                   ACIA control address register

* ************************************************************************************ *
* *                  Serial Communication Controller Degisters Definitions           * *
* ************************************************************************************ *

MR1A        equ     $00                     Channel A Mode Register
CSRA        equ     $01                     Clock Select Register Chanl. A
SRA         equ     $01                     Status Register A
CRA         equ     $02                     Command Register A
```

```
RxA     equ     $03             Receiver Register A
TxA     equ     $03             Transmit Register A
ACR     equ     $04             Auxiliary Control Register
IMR     equ     $05             Interrupt Mask Register
MR1B    equ     $08             Channel B Mode Register
CSRB    equ     $09             Clock Select Register Chanl. A
SRB     equ     $09             Status Register B
CRB     equ     $0A             Command Register B
RxB     equ     $0B             Receiver Register B
TxB     equ     $0B             Transmit Register B

* ************************************************************************************ *
* *                              Constant Definitions                                * *
* ************************************************************************************ *

CR              equ     $0D     Carriage Return definition
LF              equ     $0A     Line Feed definition
NUL             equ     $00     Null character definition
MS_600          equ     $84CD   msec. to get 1 char at 600bps
MS_19200        equ     $412    msec. to get 1 char at 19200 bps
SCC_EN          equ     $F7     SCC en. control bit (bit 3 =0)
SCC_DIS         equ     $08     SCC dis. control bit (bit 3 =1)
LOG_EN          equ     $7F     Logic en. control bit (bit 7 =0)
LOG_DIS         equ     $80     Logic dis. control bit (bit 7 =1)
CHNL_A_DIS      equ     $10     chnl A (600 bps) dis. (bit 4 =1)
CHNL_B_DIS      equ     $20     chnl B (19200 bps) dis. (bit 5 =1)
DBR_HI          equ     $40     DBR=19200 bps bit sel. (bit 6 =1)
SCC_WR          equ     $0F     SCC write enable (bit 4 =0)
SCC_RD          equ     $10     SCC read enable (bit 4 =1)
IN              equ     $00     Port C direction cntrl byte
OUT             equ     $FF     Port C direction cntrl byte
TRUE            equ     $01     Constant
FALSE           equ     $00     Constant

* ************************************************************************************ *
* *                           Interrupt Vector Definition                            * *
* ************************************************************************************ *

IRQIV           equ     $EE     Interrupt ReQuest Vector address
                org     IRQIV
                jmp     irqisr  int. request int. serv. routine

* ************************************************************************************ *
* *                                 Main Program                                     * *
* ************************************************************************************ *

*void main(void)
*{
                org     $C000   program starts
                sei             disable all interrupts
                lds     #STACK  load stack pointer
                jsr     clr_mem
                jsr     init_ports      initialize ports A and C
                jsr     aciasetup       ACIA setup
                jsr     iniq            Q initialization
                jsr     init_timer      initialize timer
                jsr     init_scc_ports  initialize the SCC ports
                jsr     crlf            start at column 0
                clra                    clear Acc A and
```

```
                clrb                    Acc B
                staa    TEMP            clear TEMP  memory
                staa    TEMP1           clear TEMP1 memory
                ldx     #MSG            load init message
                jsr     outstring       output message
                jsr     crlf            next line
                jsr     clear_q         clear Q
                clra                    clear Acc A and
                clrb                    Acc B
                staa    STATE           clear STATE selector
                jsr     enable_scc
                jsr     set_cntrl_tbl
                cli                     enable all interrupts
START_LOOP      ldaa    STATE           get STATE value
                cmpa    #$01            if STATE is 1
                beq     STATE1          then execute first state
                cmpa    #$02            if STATE is 2
                beq     STATE2          then execute second state
                cmpa    #$03            if STATE is 3
                beq     STATE3          then execute third state
                cmpa    #$04            if STATE is 4
                beq     STATE4          then execute fourth state
                cmpa    #$05            if STATE is 5
                beq     STATE5          then execute fifth state
                cmpa    #$06            if STATE is 6
                beq     STATE6          then execute sixth state
                cmpa    #$07            if STATE is 7
                beq     STATE7          then execute seventh state
                cmpa    #$08            if state is 8
                beq     STATE8          then execute eighth state
                clr     STATE           otherwise, clear STATE value
                inc     STATE           get 1 to start state machine
                bra     START_LOOP      restart the state machine
STATE1          jsr     state1_func     first state function call
                inc     STATE           prepare for next state
                bra     START_LOOP      return to the loop
STATE2          jsr     satate2_func    second state function call
                inc      STATE          prepare for next state
                bra     START_LOOP      return to the loop
STATE3          jsr     state3_func     third state function call
                inc     STATE           prepare for next state
                bra     START_LOOP      return to the loop
STATE4          jsr     state4_func     fourth state function call
                inc     STATE           prepare for next step
                bra     START_LOOP      return to the loop
STATE5          jsr     state5_func     fifth state function call
                inc     STATE           prepare for next state
                bra     START_LOOP      return to the loop
STATE6          jsr     state6_func     sixth state function call
                inc     STATE           prepare for next step
                bra     START_LOOP      return to the loop
STATE7          jsr     state7_func     seventh state function call
                inc     STATE           prepare for next state
                bra     START_LOOP      return to the loop
STATE8          jsr     state8_func     eighth state function call
                clr     STATE           clear state to start state
                inc     STATE           machine first state
                bra     START_LOOP      return to the loop
*)
```

```
*void aciasetup(void)
*{
aciasetup       equ     *
                ldx     #ACIA
                ldaa    #$03
                staa    0,X
                ldaa    #$96
                staa    0,X
                jsr     delay
                rts
*}

*void clr_mem(void)
*{
clr_mem         equ     *
                ldx     #CNTRL
                ldaa    #$00
do_again staa   0,x
                cpx     #DEV_LAST
                beq     end_clr_mem
                inx
                bra     do_again
end_clr_mem     rts
*}

*void set_cntrl_table(void)
*{
set_cntrl_tbl   equ     *
                ldx     #CNTRL_TABLE
                ldaa    #$4F
                staa    0,x
                ldaa    #$5F
                staa    1,x
                ldaa    #$6F
                staa    2,x
                ldaa    #$7F
                staa    3,x
                rts
*}

*void init_timer(void)
*{
init_timer      equ     *
                ldaa    #$00
                staa    TMSK2
                rts
*}

*void init_ports(void)
*{
init_ports      equ     *
                ldaa    #$80
                staa    PACTL
                ldaa    #IN
                staa    DDRC
                ldaa    #$F8
                staa    PORTA
                ldaa    #OUT
```

```
                staa     PORTB
                rts
*}

*void iniq(void)
*{
iniq            equ      *
                ldaa     #$00
                staa     HEAD
                staa     TAIL
                sta      COUNT
                ldx      #BUFF
LOOP1           staa     0,X
                inx      adresses
                cpx      #LAST
                bne      LOOP1
                rts
*}

*void dis_all_cntrl(void)
*{
dis_all_cntrl   equ      *
                pshx
                ldaa     #CHNL_A_DIS
                ora      #CHNL_B_DIS
                ora      #DBR_HI
                ora      #LOG_DIS
                ora      #SCC_DIS
                staa     CNTRL
                pulx
                rts
*}

*void init_scc_ports(void)
*{
init_scc_ports  equ      *
                jsr      dis_all_cntrl
                ldaa     #IMR
                ldab     #NUL
                jsr      write_to_scc
SCC_A           ldaa     #CRA
                ldab     #$10
                jsr      write_to_scc
                ldaa     #CRA
                ldab     #$22
                jsr      write_to_scc
                ldaa     #CRA
                ldab     #$38
                jsr      write_to_scc
                ldaa     #CRA
                ldab     #$50
                jsr      write_to_scc
                ldaa     #MR1A
                ldab     #$13
                jsr      write_to_scc
                ldaa     #MR1A
                ldab     #$07
                jsr      write_to_scc
                ldaa     #CRA
```

```
                ldab    #$10
                jsr     write_to_scc
                ldaa    #ACR
                ldab    #$80
                jsr     write_to_scc
                ldaa    #CSRA
                ldab    #$55
                jsr     write_to_scc
                ldaa    #CRA
                ldab    #$40
                jsr     write_to_scc
                ldaa    #CRA
                ldab    #$15
                jsr     write_to_scc
SCC_B           ldaa    #CRB
                ldab    #$10
                jsr     write_to_scc
                ldaa    #CRB
                ldab    #$22
                jsr     write_to_scc
                ldaa    #CRB
                ldab    #$38
                jsr     write_to_scc
                ldaa    #CRB
                ldab    #$50
                jsr     write_to_scc
                ldaa    #MR1B
                ldab    #$13
                jsr     write_to_scc
                ldaa    #MR1B
                ldab    #$07
                jsr     write_to_scc
                ldaa    #CRB
                ldab    #$10
                jsr     write_to_scc
                ldaa    #CSRB
                ldab    #$CC
                jsr     write_to_scc
                ldaa    #CRB
                ldab    #$40
                jsr     write_to_scc
                ldaa    #CRB
                ldab    #$15
                jsr     write_to_scc
                rts
*}

*void enable_scc(void)
*{
enable_scc      equ     *
                ldaa    #CRA
                ldab    #$05
                jsr     write_to_scc
                ldaa    #CRB
                jsr     write_to_scc
                rts
*}
```

```
*void write_to_scc((register -> A), (command -> B))
*{
write_to_scc    equ     *
                psha
                anda    #SCC_WR
                staa    PORTB
                stab    PORTC
                ldaa    #OUT
                staa    DDRC
                ldaa    CNTRL
                anda    #SCC_EN
                staa    PORTA
                ora     #SCC_DIS
                staa    PORTA
                pula
                ora     #SCC_RD
                staa    PORTB
                ldaa    #IN
                staa    DDRC
                rts
*}

*void read_from_scc(void)
*{
read_from_scc   equ     *
                clrb
                ora     #SCC_RD
                staa    PORTB
                ldaa    CNTRL
                anda    #SCC_EN
                staa    PORTA
                ldaa    #IN
                staa    DDRC
                ldab    PORTC
                ldaa    CNTRL
                ora     #SCC_DIS
                staa    PORTA
                ldaa    #IN
                staa    DDRC
                rts
*}

*void send_char_scca(void)
*{
send_char_scca  equ     *
                psha
                pshb
                ldaa    CNTRL
                anda    #SCC_EN
                staa    CNTRL
loop_char  ldaa #SRA
                jsr     read_from_scc
                andb    #$04
                cmpb    #$04
                bne     loop_char
                pulb
                pula
                jsr     write_to_scc
                rts
```

```
*}

*void send_char_sccb(void)
*{
send_char_sccb    equ     *
                  psha
                  pshb
                  ldaa    CNTRL
                  anda    #SCC_EN
                  staa    CNTRL
loop_chr ldaa     #SRB
                  jsr     read_from_scc
                  andb    #$04
                  cmpb    #$04
                  bne     loop_chr
                  pulb
                  pula
                  jsr     write_to_scc
                  rts
*}

* void state1_func(void)
*{
state1_func       equ     *
                  jsr     chk_command
                  jsr     delay1
                  jsr     delay1
                  rts
*}

*void state4_func(void)
*{
state4_func       equ     *
                  clr     DEV_COUNT
                  clr     DEV_PRBLM
                  jsr     delay
                  jsr     delay
                  rts
*}

*void state5_func(void)
*{
state5_func       equ     *
                  jsr     detect_dev_act
                  ldaa    DEV_ACTIVE
                  anda    #TRUE
                  cmpa    #FALSE
                  beq     no_dev_act
                  clr     DEV_ACTIVE
                  clr     PNTR
                  clr     PNTR1
chk_again_low     jsr     set_cntrl_byte
                  ldaa    DEV_ACTIVE
                  anda    #TRUE
                  cmpa    #TRUE
                  bne     cont_ste5
dev_active        jsr     chk_low_dbr
                  ldx     #DEV_BUFF
                  ldab    PNTR
```

```
                abx
                ldaa    DEV_PRBLM_LOW
                cmpa    #TRUE
                beq     dev_low_bad
dev_low_ok      ldaa    0,x
                ora     #$10
                staa    0,x
                jsr     dev_msg_ok
                bra     cont_ste5
dev_low_bad     ldaa    0,x
                anda    #$EF
                staa    0,x
                ldab    PNTR1
                ldx     #OUT_BUFF
                abx
                ldaa    PNTR
                staa    0,x
                inc     DEV_COUNT
                inc     PNTR1
                jsr     dev_msg_bad
cont_ste5       inc     PNTR
                ldaa    PNTR
                anda    #$0F
                cmpa    #$04
                bne     chk_again_low
                bra     end_ste5
no_dev_act      ldx     #MSGNODEV
                jsr     outstring
                jsr     crlf
end_ste5        rts
*}

*void state6_func(void)
*{
state6_func     equ     *
                jsr     detect_dev_act
                ldaa    DEV_ACTIVE
                anda    #TRUE
                cmpa    #FALSE
                beq     no_dev_act1
                clr     DEV_ACTIVE
                clr     PNTR
chk_again_high  jsr     set_cntrl_byte
                ldaa    DEV_ACTIVE
                anda    #TRUE
                cmpa    #TRUE
                bne     cont_ste6
dev_active1     jsr     chk_hi_dbr
                ldx     #DEV_BUFF
                ldab    PNTR
                abx
                ldaa    DEV_PRBLM_HI
                cmpa    #TRUE
                beq     dev_high_bad
dev_high_ok     ldaa    0,x
                ora     #$20
                staa    0,x
                jsr     dev_msg_ok
                bra     cont_ste6
```

```
dev_high_bad    ldaa    0,x
                anda    #$DF
                staa    0,x
                ldab    PNTR1
                ldx     #OUT_BUFF
                abx
                ldaa    PNTR
                staa    0,x
                inc     DEV_COUNT
                inc     PNTR1
                jsr     dev_msg_bad
cont_ste6       inc     PNTR
                ldaa    PNTR
                anda    #$0F
                cmpa    #$04
                bne     chk_again_high
                bra     end_ste6
no_dev_act1     ldx     #MSGNODEV
                jsr     outstring
                jsr     crlf
end_ste6        rts
*}

*void state7_func(void)
*{
state7_func     equ     *
                jsr     chk_dev_bad
                jsr     delay
                jsr     delay
                jsr     delay
                jsr     delay
                rts
*}

*void chk_low_dbr(void)
*{
chk_low_dbr     equ     *
                ldaa    #TxA
                ldab    #$55
                jsr     send_char_scca
                jsr     delay
                ldaa    #$02
                staa    TEMP1
read_again1     ldaa    #SRA
                jsr     read_from_scc
                andb    #$01
                cmpb    #$01
                beq     read_scc1
chk_temp1       dec     TEMP1
                ldaa    TEMP1
                anda    #$0F
                cmpa    #$00
                bne     read_again1
read_scc1       ldaa    #RxA
                jsr     read_from_scc
                cmpb    #$55
                beq     dev_ok
dev_prblm       ldaa    #TRUE
                staa    DEV_PRBLM_LOW
```

```
                staa    DEV_PRBLM
                bra     end_chk_low_dbr
dev_ok          ldaa    #FALSE
                staa    DEV_PRBLM_LOW
end_chk_low_dbr ldaa    #CNTRL
                ora     #LOG_DIS
                staa    PORTA
                rts
*}

*void chk_hi_dbr(void)
*{
chk_hi_dbr      equ     *
                ldaa    #TxB
                ldab    #$AA
                jsr
                jsr     delay2
loop_rx         ldaa    #SRB
                jsr     read_from_scc
                andb    #$03
                cmpb    #$01
                bne     rd_scc2_again
                bra     rd_scc2
rd_scc2_again   ldaa    #RxB
                jsr     read_from_scc
                bra     loop_rx
rd_scc2         ldaa    #RxB
                jsr     read_from_scc
                cmpb    #$AA
                bne     dev_okb
dev_prblmb      ldaa    #TRUE
                staa    DEV_PRBLM_HI
                staa    DEV_PRBLM
                bra     end_chk_hi_dbr
dev_okb         ldaa    #FALSE
                staa    DEV_PRBLM_HI
end_chk_hi_dbr  ldaa    #CNTRL
                ora     #LOG_DIS
                staa    PORTA
                rts
*}

*void set_cntrl_byte(void)
*{
set_cntrl_byte  equ     *
                ldab    PNTR
                ldx     #DEV_BUFF
                abx
                ldaa    0,x
                anda    #$C0
                cmpa    #$C0
                bne     dev_off
dev_on          ldy     #CNTRL_TABLE
                aby
                ldaa    STATE
                anda    #$05
                cmpa    #$05
                bne     high_dbr1
low_dbr1 ldaa   0,y
```

```
                anda    #$BF
                staa    CNTRL
                bra     enable_flag
high_dbr1       ldaa    0,y
                ora     #$40
                staa    CNTRL
enable_flag     ldaa    #TRUE
                staa    DEV_ACTIVE
                bra     end_cntrl
dev_off         ldaa    #FALSE
                staa    DEV_ACTIVE
end_cntrl       rts
*}

*void detect_dev_act(void)
*{
detect_dev_act  equ     *
                ldaa    #FALSE
                staa    PNTR
                staa    DEV_ACTIVE
chk_again_dev   ldab    PNTR
                ldx     #DEV_BUFF
                abx
                ldaa    0,x
                anda    #$C0
                cmpa    #$C0
                bne     dev_not_act
dev_is_act      ldaa    #TRUE
                staa    DEV_ACTIVE
                bra
dev_not_act inc PNTR
                ldaa    PNTR
                anda    #$0F
                cmpa    #$04
                bne
end_detect_dev  rts
*}

*void chk_dev_bad(void)
*{
chk_dev_bad     equ     *
                jsr     detect_dev_act
                ldaa    DEV_ACTIVE
                cmpa    #FALSE
                beq     no_dev_active
dev_act1  ldaa  DEV_PRBLM
                cmpa    #TRUE
                beq             dev_prblm1
                bra             no_dev_prblm1
dev_prblm1      jsr             send_bad_dev
                bra             dev_prblm2
no_dev_prblm1   ldx             #MSGDEVOK
                bra             send_info
no_dev_active   ldx             #MSGNODEV
                bra             send_info
dev_prblm2      ldx             #MSGDEVBAD
send_info       jsr             outstring
                jsr             crlf
                rts
```

```
*}

*void send_bad_dev(void)
*{
send_bad_dev    equ     *
                ldab    #FALSE
                stab    PNTR1
                ldaa    #'c'
                jsr     outacia
repeat          ldab    PNTR1
                ldx     #OUT_BUFF
                abx
                ldaa    0,x
                adda    #$30
                jsr     outacia
                inc     PNTR1
                ldab    PNTR1
                cmpb    DEV_COUNT
                blo     repeat
                ldaa    #'c'
                jsr     outacia
                jsr     crlf
                rts
*}

* void show_char(void)
*{
show_char       equ     *
LOOP_1          jsr     isempty
                beq     LOOP_2
                jsr     remove_q
                jsr     outacia
                bra     LOOP_1
LOOP_2          jsr     clear_q
                rts
*}

*void insert_q(void)
*{
insert_q        equ     *
                pshb
                pshx
                ldx     #BUFF
                ldab    TAIL
                abx
                staa    0,X
                inc     TAIL
                inc     COUNT
                pulx
                pulb
                rts
*}

* char remove_q(void)
*{
remove_q        equ     *
                pshb
                pshx
                ldx     #BUFF
```

```
                ldab    HEAD
                abx
                ldaa    0,X
                inc     HEAD
                dec     COUNT
                pulx
                pulb
                rts
*}

*void isempty(void)
*{
isempty         equ     *
                tst     COUNT
                rts
*}

* void clear_q(void)
*{
clear_q         equ     *
                clr     HEAD
                clr     TAIL
                clr     COUNT
                rts
*}

*void chk_command(void)
*{
chk_command     equ     *
                jsr     isempty
                bne     GET_CHAR
                jsr     clear_q
                jmp     END_CHK_CMD
GET_CHAR        jsr     remove_q
                cmpa    #'c'
                beq     CHK2_CMD
                jmp     RTN_CHR
CHK2_CMD        jsr     remove_q
                cmpa    #'a'
                beq     ADD_FUNC
                cmpa    #'c'
                beq     ENDCMD
                cmpa    #'d'
                beq     DIS_FUNC
                cmpa    #'e'
                beq     EN_FUNC
                cmpa    #'k'
                beq     CONT_FUNC
                cmpa    #'l'
                beq     LST_FUNC
                cmpa    #'r'
                beq     REMOV_FUNC
                cmpa    #'s'
                beq     STOP_FUNC
                cmpa    #NUL
                jsr     clear_q
                bra     END_CHK_CMD
RTN_CHR         jsr     return_char
                bra     END_CHK_CMD
```

```
ADD_FUNC        jsr     add_function
                bra     END_CHK_CMD
CONT_FUNC       jsr     cont_function
                bra     END_CHK_CMD
DIS_FUNC        jsr     dis_function
                bra     END_CHK_CMD
EN_FUNC         jsr     en_function
                bra     END_CHK_CMD
ENDCMD          jsr     endcmd_function
                bra     END_CHK_CMD
LST_FUNC        jsr     lst_function
                bra     END_CHK_CMD
REMOV_FUNC      jsr     remov_function
                bra     END_CHK_CMD
STOP_FUNC       jsr     stop_function
END_CHK_CMD     rts
*}

*void return_char(void)
*{
return_char     equ     *
                jsr     outacia
                jsr     crlf
                ldx     #MSGRTN
                jsr     outstring
                jsr     crlf
                jsr     delay
                rts
*}

*void add_function(void)
*{
add_function    equ     *
                ldx     #MSGADD
                jsr     outstring
                jsr     crlf
                jsr     remove_q
                jsr     add_dev_to_q
                jsr     chk_dev_added
                cmpa    #TRUE
                bne     dev_added_bad
dev_added_ok    ldx     #MSGDEVADDOK
                jsr     outstring
                jsr     send_msg
                jsr     crlf
                jsr     send_ok
                bra     end_add_dev
dev_added_bad   ldx     #MSGDEVADDBAD
                jsr     outstring
                jsr     send_msg
                jsr     crlf
                jsr     send_bad
end_add_dev     jsr     crlf
                rts
*}
```

```
*void add_dev_to_q(void)
*{
add_dev_to_q    equ     *
                psha
                pshb
                pshx
                ldx     #DEV_BUFF
                suba    #'0'
                staa    OFFSET
                tab
                abx
                ldaa    0,x
                anda    #$80
                cmpa    #$80
                beq     no_count
count           inc     DEV_COUNT
no_count        ldaa    #$80
                staa    0,x
                pulx
                pulb
                pula
                rts
*}

*void chk_dev_added(void)
*{
chk_dev_added   equ     *
                pshb
                pshx
                ldx     #DEV_BUFF
                ldab    OFFSET
                abx
                ldaa    0,x
                anda    #$80
                cmpa    #$80
                beq     deva_ok
                bra     deva_bad
deva_ok         ldaa    #TRUE
                bra     end_chk_dev
deva_bad        ldaa    #FALSE
end_chk_dev     pulx
                pulb
                rts
*}

*void cont_function(void)
*{
cont_function   equ     *
                ldx     #MSGCNT
                jsr     outstring
                jsr     crlf
                jsr     delay
                rts
*}
```

```
*void dis_function(void)
*{
dis_function    equ     *
                ldx     #MSGDIS
                jsr     outstring
                jsr     crlf
                jsr     remove_q
                jsr     dis_dev
                jsr     chk_dev_dis
                cmpa    #TRUE
                bne     dev_dis_bad
dev_dis_ok      ldx     #MSGDEVDISOK
                jsr     outstring
                jsr     send_msg
                jsr     crlf
                jsr     send_ok
                bra     end_dis_dev
dev_dis_bad     ldx     #MSGDEVDISBAD
                jsr     outstring
                jsr     send_msg
                jsr     crlf
                jsr     send_bad
end_dis_dev     jsr     crlf
                rts
*}

*void dis_dev(void)
*{
dis_dev         equ     *
                psha
                pshb
                pshx
                ldx     #DEV_BUFF
                suba    #'0'
                staa    OFFSET
                tab
                abx
                ldaa    0,x
                anda    #$BF
                staa    0,x
                pulb
                pula
                rts
*}

*void chk_dev_dis(void)
*{
chk_dev_dis     equ     *
                pshb
                pshx
                ldx     #DEV_BUFF
                ldab    OFFSET
                abx
                ldaa    0,x
                anda    #$40
                cmpa    #FALSE
                beq     dev_disok
                bra     dev_disbad
dev_disok       ldaa    #TRUE
```

```
                bra
dev_disbad      ldaa    #FALSE
end_chk_dev_dis pulx
                pulb
                rts
*}

*void en_function(void)
*{
en_function     equ     *
                ldx     #MSGEN
                jsr     outstring
                jsr     crlf
                jsr     remove_q
                jsr     en_dev
                jsr     chk_dev_en
                cmpa    #TRUE
                bne     dev_en_bad
dev_en_ok       ldx     #MSGDEVENOK
                jsr     outstring
                jsr     send_msg
                jsr     crlf
                jsr     send_ok
                bra     end_en_dev
dev_en_bad      ldx     #MSGDEVENBAD
                jsr     outstring
                jsr     send_msg
                jsr     crlf
                jsr     send_bad
end_en_dev      jsr     crlf
                rts
*}

*void en_dev(void)
*{
en_dev          equ     *
                psha
                pshb
                pshx
                ldx     #DEV_BUFF
                suba    #'0'
                staa    OFFSET
                tab
                abx
                ldaa    0,x
                ora     #$40
                staa    0,x
                pulx
                pulb
                pula
                rts
*}

*void chk_dev_en(void)
*{
chk_dev_en      equ     *
                pshb
                pshx
                ldx     #DEV_BUFF
```

```
                    ldab    OFFSET
                    abx
                    ldaa    0,x
                    anda    #$40
                    cmpa    #$40
                    beq     dev_enok
                    bra     dev_enbad
dev_enok            ldaa    #TRUE
                    bra     end_chk_dev_en
dev_enbad           ldaa    #FALSE
end_chk_dev_en      pulx
                    pulb
                    rts
*}

*void endcmd_function(void)
*{
endcmd_function     equ     *
                    ldx     #MSGEND
                    jsr     outstring
                    jsr     crlf
                    jsr     delay
                    rts
*}

*void lst_function(void)
*{
lst_function        equ     *
                    ldx     #MSGLST
                    jsr     outstring
                    jsr     crlf
                    jsr     delay
                    rts
*}


*void remov_function(void)
*{
remov_function      equ     *
                    psha
                    pshb
                    pshx
                    ldx     #MSGREMOV
                    jsr     outstring
                    jsr     crlf
                    jsr     remove_q
                    jsr     remov_dev_q
                    jsr     chk_dev_remvd
                    cmpa    #TRUE
                    bne     dev_remvd_bad
dev_remvd_ok        ldx     #MSGDEVREMOK
                    jsr     outstring
                    jsr     send_msg
                    jsr     crlf
                    jsr     send_ok
                    bra     end_rem_dev
dev_remvd_bad       ldx     #MSGDEVREMBAD
                    jsr     outstring
                    jsr     send_msg
```

```
                jsr     crlf
                jsr     send_bad
end_rem_dev     jsr     crlf
                pulx
                pulb
                pula
*}

*void remov_dev_q(void)
*{
remov_dev_q     equ     *
                psha
                pshx
                ldx     #DEV_BUFF
                suba    #'0'
                staa    OFFSET
                tab
                abx
                ldaa    #$00
                staa    0,x
                dec     DEV_COUNT
                pulx
                pulb
                pula
                rts
*}

* void chk_dev_remvd(void)
*{
chk_dev_remvd   equ     *
                pshb
                pshx
                ldx     #DEV_BUFF
                ldab    OFFSET
                abx
                ldaa    0,x
                anda    #$FF
                cmpa    #$00
                beq     devr_ok
                bra     devr_bad
devr_ok         ldaa    #TRUE
                bra     end_chkr_dev
devr_bad        ldaa    #FALSE
end_chkr_dev    pulx
                pulb
                rts
*}

*void stop_function(void)
*{
stop_function   equ     *
                ldx     #MSGSTP
                jsr     outstring
                jsr     crlf
                jsr     delay
                rts
*}
```

```
*void send_ok(void)
*{
send_ok         equ     *
                psha
                pshb
                pshx
                ldaa    #'c'
                jsr     outacia
                ldaa    #'o'
                jsr     outacia
                pulx
                pulb
                pula
                rts
*}

*void send_bad(void)
*{
send_bad        equ     *
                psha
                pshb
                pshx
                ldaa    #'c'
                jsr     outacia
                ldaa    #'b'
                jsr     outacia
                pulx
                pulb
                pula
                rts             return
*}

* void readacia(void)
*{
readacia        equ     *
                ldaa    ACIA
                bita    #$01
                beq     readacia
                ldaa    ACIA+1
                rts
*}

*void outacia(char accA)
*{
outacia         equ     *
                stab    TEMP3
                ldab    ACIA
                bitb    #$02
                beq     outacia
                anda    #$7F
                staa    ACIA+1
                cmpa    #CR
                bne     OUT_RTS
                ldaa    #LF
                bra     outacia
OUT_RTS         ldab    TEMP3
                rts
*}
```

```
*void outstring(void)
*{
outstring       equ     *
STRING          ldaa    0,X
                cmpa    #'$'
                beq     DONE
                jsr     outacia
                inx
                bra     STRING
DONE            rts
*}

*void send_msg(void)
*{
send_msg        equ     *
                psha
                pshb
                pshx
                ldx     #MSGDEV
                jsr     outstring
                ldaa    OFFSET
                adda    #$31
                jsr     outacia
                pulx
                pulb
                pula
                rts
*}

*void crlf(void)
*{
crlf            equ     *
                ldaa    #CR
                jsr     outacia
                rts
*}

*void dev_msg(void)
*{
dev_msg         equ     *
                pshx
                ldx     #MSGDEV
                jsr     outstring
                ldaa    PNTR
                adda    #$31
                jsr     outacia
                pulx
                rts
*}

*void dev_msg_ok(void)
*{
dev_msg_ok      equ     *
                pshx
                jsr     dev_msg
                ldx     #MSGOK2
                jsr     outstring
                jsr     crlf
```

```
                pulx
                rts
*}

*void dev_msg_bad(void)
*{
dev_msg_bad     equ     *
                pshx
                jsr     dev_msg
                ldx     #MSGBAD2
                jsr     outstring
                jsr     crlf
                pulx
                rts
*}

*void delay(void)
*{
delay           equ     *
                ldy     #$FFFF
DLY             nop
                nop
                dey
                bne     DLY
                rts
*}

*void delay1(void)
*{
delay1          equ     *
                ldy     #$000F
DLX             nop
                nop
                dex
                bne     DLX
                rts
*}

*void delay2(void)
*{
delay2          equ     *
                ldy     #$0008
DLX2            nop
                nop
                dex
                bne     DLX2
                rts
*}

*void irqisr(void)
*{
irqisr          equ     *
                psha
                pshb
                pshx
                ldab    ACIA
                bpl     IRQEND
                bitb    #$01
                beq     IRQISR1
```

```
                bne     inACIA
IRQISR1         ldab    ACIA
                bitb    #$02
                beq     IRQEND
                bne     outACIA
inACIA          jsr     readacia
                jsr     insert_q
                bra     IRQEND
outACIA         jsr     outacia
                bra     IRQEND
IRQEND          ldaa    #$96
                staa    ACIA
                pulx
                pulb
                pula
                rti
*)

MSG             equ     *
                FCC     'Starting... $'
MSGDEV          equ             *
                FCC     'DEVICE :  $'
MSGNODEV        equ     *
                FCC     'NO DEVICE ACTIVE $'
MSGRTN          equ     *
                FCC     'RTN CHAR $'
MSGADD          equ     *
                FCC     'ADD DEV $'
MSGDEVADDOK             equ             *
                FCC     'DEV ADD OK $'
MSGDEVADDBAD            equ             *
                FCC     'DEV ADD BAD $'
MSGCNT          equ     *
                FCC     'CONT $'
MSGDIS          equ     *
                FCC     'DISBL DEV $'
MSGDEVDISOK     equ     *
                FCC     'DEV DIS OK $'
MSGDEVDISBAD    equ     *
                FCC     'DEV DIS BAD $'
MSGEN           equ     *
                FCC     'ENBL DEV $'
MSGDEVENOK      equ             *
                FCC     'DEV EN OK $'
MSGDEVENBAD
                FCC     'DEV EN BAD $'
MSGEND          equ             *
                FCC     'END CMD $'
MSGLST          equ     *
                FCC     'LST DEV $'
MSGREMOV        equ     *
                FCC     'REMV DEV $'
MSGDEVREMOK     equ     *
                FCC     'RMVD DEV OK $'
MSGDEVREMBAD    equ     *
                FCC     'RMVD DEV BAD $'
MSGSTP          equ     *
                FCC     'STOP $'
MSGEND2         equ     *
```

```
            FCC     'END LOW DBR $'
MSGOK       equ     *
            FCC     'LOW DBR OK $'
MSGOK1      equ     *
            FCC     'HIGH DBR OK $'
MSGOK2      equ     *
            FCC     ' OK $'
MSGBAD      equ     *
            FCC     'LOW DBR BAD $'
MSGBAD1     equ     *
            FCC     'HIGH DBR BAD $'
MSGBAD2     equ     *
            FCC     ' BAD $'
MSGDEVOK    equ     *
            FCC     ' DEVICES OK $'
MSGDEVBAD   equ     *
            FCC     ' DEVICES BAD $'

CNTRL               RMB     1       control byte
OFFSET              RMB     1       device offset pointer
PNTR                RMB     1       device pointer
PNTR1               RMB     1       output offset pointer
DEV_ACTIVE          RMB     1       device active flag
DEV_PRBLM_LOW       RMB     1       device problem low flag
DEV_PRBLM_HI        RMB     1       device problem high flag
DEV_PRBLM           RMB     1       device problem flag
TEMP                RMB     1       temporary memory 0
TEMP1               RMB     1       temporary memory 1
TEMP2               RMB     1       temporary memory 2
TEMP3               RMB     1       temporary memory 3
TEMP4               RMB     1       temporary memory 4
STATE               RMB     1       state machine: state
HEAD                RMB     1         "   " Head
TAIL                RMB     1         "   " Tail
COUNT               RMB     1         "   " Counter
BUFF                RMB     64      addresses for Buffer
LAST                RMB     1       last address of Buffer
OUT_BUFF            RMB     16      output buffer
CNTRL_TABLE         RMB     16      control bytes table
DEV_HEAD            RMB     1       device buffer head
DEV_TAIL            RMB     1       device buffer tail
DEV_COUNT           RMB     1       device buffer counter
DEV_BUFF            RMB     64      device buffer
DEV_LAST            RMB     1       last device buffer position
                    RMB     256     addresses for Stack Point
STACK               RMB     1       initial add of Stack Point
```

APPENDIX B

```
/* ********************    MAIN FILE FOR COMMUNICATION PROGRAM    ****************** */
/* *                              Author: Isidro Alvarez, B.S.E.E.                    * */
/* *                  Master of Science in Computer Engineering Thesis Project        * */
/* *                              School of Engineering and Design                    * */
/* *                    Department of Electrical and Computer Engineering             * */
/* *                              Florida International University                    * */
/* *                                   Miami, Florida                                 * */
/* *                                    Spring 1996                                   * */
/* ********************************************************************************** */

#include  "c:\isidro\msvc\_ser41.h"
#include  "c:\isidro\msvc\sercl41.h"

void  room_numb(void);


void main(void)
{
        COMPORT port;
        char ch;

        system("cls");
        printf("\n  Is this the Main Security Office : ");
        while(!_kbhit())                                                        /* */
                ;                                                               /* */
        ch = _getche();                                                         /* */
        if ((ch == 'Y') || (ch == 'y'))
                sec_office = TRUE;
        else
        {
                sec_office = FALSE;
                room_numb();
        }

  system("cls");
        port.setup_ports();
        port.init_card();
        for(int i=0; i < 5; i++)
        {
                device[i].pos_used = FALSE;
                device[i].enabled  = FALSE;
                device[i].problem  = FALSE;
                device[i].had_problem = FALSE;
                device[i].numb_of_problems = 0;
        }
        device_problem_flag = FALSE;
        printf("Waiting for actions \n");
        while(TRUE)
        {
                if(!port.chat_mode())
                        break;
                port.tx_rx_mode();
        }

}
```

```
void room_numb(void)
{
        int flag = TRUE;
        char ch;

        system("cls");
        while(flag)
        {
                system("cls");
                printf("\n\n");
                printf("\n Enter room where the SECNET device is installed. \n");
                printf("\n Please follow the following format: ecs258 or ECS258 \n");
                printf("\n Enter room number :  ");
                scanf("%s", data_file_name);
                printf("\n You entered Room # = %s", data_file_name);
                printf("\n Is it correct (Y/N) : ");
                while(!_kbhit())                                        /* */
                        ;                                               /* */
                ch = _getche();                                         /* */
                if ((ch == 'Y') || (ch == 'y'))
                        flag = FALSE;
        }
        flag = TRUE;
        while(flag)
        {
                printf("\n\n");
                printf("\n Enter phone number of the Main Security Office. \n");
                printf("\n You can enter only the extension number... \n");
                printf("\n Enter phone number :  ");
                scanf("%s", phone_number);
                printf("\n You entered Phone # = %s", phone_number);
                printf("\n Is it correct (Y/N) : ");
                while(!_kbhit())                                        /* */
                        ;                                               /* */
                ch = _getche();                                         /* */
                if ((ch == 'Y') || (ch == 'y'))
                        flag = FALSE;
        }
        strcpy(dial, at_d);
        strcat(dial, phone_number);
        system("cls");
}
```

```
/* *********      CLASS DEFINITION HEADER FOR COMMUNICATION PROGRAM      ********** */
/* *                          Author: Isidro Alvarez, B.S.E.E.                      * */
/* *                Master of Science in Computer Engineering Thesis Project        * */
/* *                          School of Engineering and Design                      * */
/**                       Department of Electrical and Computer Engineering        * */
/* *                          Florida International University                     * */
/* *                                Miami, Florida                                  * */
/* *                                Spring 1996                                     * */
/* ************************************************************************************** */

#ifndef _SERCL41H_
#define _SERCL41H_

#include  "c:\isidro\msvc\_ser41.h"
#include  "c:\isidro\msvc\dv_com41.h"
#include  "c:\isidro\msvc\menu41.h"


        dev_MEMBERS *device2 = (dev_MEMBERS *) device;

void (interrupt _far *old_handler1)(void);
void (interrupt _far *old_handler2)(void);

void interrupt _far receive_com1()
{
    _disable();
    if(_inp(PORT_ADD1 | IIR) & RX_RDY);

    while(_inp(PORT_ADD1 | LSR) & DATA_RDY)
    {
          rx_buffer1[rx_buf_in1]=_inp(PORT_ADD1);
          rx_buf_in1++;
          if(rx_buf_in1 == BUF_MAX)
                rx_buf_in1 = 0;
    }
    _outp(ICR,EOI);
    _enable();
}

void interrupt _far receive_com2()
{
       _disable();
       if(_inp(PORT_ADD2 | IIR) & RX_RDY);

        while(_inp(PORT_ADD2 | LSR) & DATA_RDY)
        {
              rx_buffer2[rx_buf_in2]=_inp(PORT_ADD2);
              rx_buf_in2++;
              if(rx_buf_in2 == BUF_MAX)
                    rx_buf_in2 = 0;
        }
        _outp(ICR,EOI);
        _enable();
}

inline long int timing(void);
inline int time_out(int inc);
```

```
long int timing(void)
{
        time_t lt;
        lt = time(NULL);
        return(lt);
}

int time_out(int inc)
{
        long int newtime;


        newtime=timing();
        if ((newtime - start) < (inc + 1))
                return FALSE;
        else
                return TRUE;
}

class COMPORT
{

        friend class MENU;
        friend class PROTOCOL;

        /* ****************************  Private Function Declaration  ************************ */
        /* ********************************************************************************** */
        private:
                int  carrier(void);                     /* check if link is made          */
                unsigned char char_in(void);            /* get char in buf. & inc ptr     */
                unsigned char character(void);          /* get char in buf. ! inc ptr     */
                void close_port(void);                  /* close port to terminate        */
                void day_selector(int i);               /* day identificator              */
                void disconnect(void);                  /* end Established Connection     */
                void dial_up(void);                     /* automatic dial up              */
                void header_block(void);                /* header block constructor       */
                void init_port(int port);               /* port 1 initialization          */
                int  is_char_in(void);                  /* chk if a char is in buffer     */
                void month_selector(int i);             /* month Identificator            */
                void open_port(void);                   /* open ports to start            */
                void purge_buff(void);                  /* purge Input Buffer             */
                void setup_port1(void);                 /* parameters for port 1          */
                void setup_port2(void);                  /* parameters for port 2         */
                int  transmitter_ready(void);           /* check if Tx buffer is empty    */
                void update_ptrs(void);                 /* if End of Buff->Begining       */

        /* ****************************  Public Function Declaration  ************************ */
        /* ********************************************************************************** */

        public:

        /* ***************************** Class Constructor Function Definition ***************** */
        /* ********************************************************************************** */

                COMPORT(void){void open_port(void);};       /* constructor Declaration        */

        /* ***************************** Class Destructor Function Definition ****************** */
        /* ********************************************************************************** */
```

```
                public: ~COMPORT(void){void close_port(void);};     /* destructor Declaration      */

                int  chat_mode(void);                               /* chat. with remote terminal */
                void csd_setup(void);                               /* circuit Switched Data SetUp*/
                void delay(clock_t wait);                           /* function for delay         */
                void get_time(void);                                /* function to get date/time  */
                void init_card(void);                               /* hayes AT Commands CSD      */
                                                                    /*  PSD                       */
                void load_def(void);                                /* parameter Definition COM1  */
                                                                    /* COM2                       */
                void proc_char(unsigned char ch);                   /* display character typed    */
                void rx_char_chat(void);                            /* rx char. from rem. terminal*/
                void rx_char_rxtx(void);                            /* rx either Files or char.   */
                void setup_ports(void);                             /* function that setups ports */
                void select_mode(unsigned char ch);                 /* select remote term. operat.*/
                void selection(unsigned char ch);                   /* enter type of operation    */
                void send_char(unsigned char ch);                   /* function to Send a Charact.*/
                void send_atcom(char *ch);                          /* function to Send AT Comd   */
                void send_string(char *ch);                         /* send a string to port      */
                void send_numb(char *ch);                           /* send number string to port */
                void room_number(void);                             /* transmission file function */
                void tx_rx_mode(void);                              /* mode used to file Tx       */
                void room_number1(void);
                void header_block1(void);
};

int COMPORT::carrier(void)
{
        return _inp(PORT_ADD2 | MSR) & DCD ? TRUE : FALSE;
}

inline unsigned char COMPORT::char_in(void)
{
        return rx_buffer2[rx_buf_out2++];
}

inline unsigned char COMPORT::character(void)
{
        return rx_buffer2[rx_buf_out2];
}

int COMPORT::chat_mode(void)
{
  long int k;
        unsigned char c;

        MENU menu_list;
  DEVCOM device_com;

        hc_counter = 0;
        while(TRUE)
        {
                rx_char_chat();
                device_com.rx_char_com1();
                if(!sec_office)
                {
                        for(int i=0; i<3000; i++)
                                ;
```

```
                    hc_counter++;
                    if(hc_counter > 8000)
                    {
                            device_problem_flag = TRUE;
                            hc_flag = FALSE;
                    }
            }
            if (device_problem_flag)
            {
                    _putch(CR);
                    _putch(LF);
                    send_atcom(dial);
                    send_char(CR);
                    for (k=0; k<=100000; k++)
                    {
                            if(carrier())
                                    k = 200000;
                    }
                    rx_char_chat();
                    device_problem_flag = TRUE;
            }
            if(_kbhit())
            {
                    c = _getche();
                    if(c != ESC)
                    {
                            switch(c)
                            {
                                    case NULL:
                                    case Ext_Key:
                                            c=_getch();
                                            if((c == 'I')||(c == 'Q'))
                                                    dial_up();*/
                                            if(c == F1_Key)
                                                    menu_list.function_menu();
                                            break;
                                    default:
                                            proc_char(c);
                            }
                    }
                    else
                            break;
            }
            if(carrier())
            {
                    link = TRUE;
                    break;
            }
    }
    if(link == TRUE)
            return TRUE;
    else
            return FALSE;
}
```

```
void COMPORT::close_port(void)
{
        disconnect();
        _disable();
        p_add=PORT_ADD1
        _outp(p_add | LCR, port1_state);
        _outp(p_add | LCR, _inp(p_add | LCR | DLAB));
        _outp(p_add, port1_dll);
        _outp(p_add | 0x01, port1_dhl);
        _outp(p_add | LCR, _inp(p_add | LCR | ~DLAB));
        _outp(p_add  | IER, 0x00);
        _outp(IMR, _inp(IMR) | IRQ_ADD1);
        _outp(p_add  | MCR, _inp(p_add  | MCR) | 0x00);
        _dos_setvect(IRQ_ADD1, old_handler1);
        p_add=PORT_ADD2;
        _outp(p_add | LCR, port2_state);
        _outp(p_add | LCR, _inp(p_add | LCR | DLAB));
        _outp(p_add, port2_dll);
        _outp(p_add | 0x01, port2_dhl);
        _outp(p_add | LCR, _inp(p_add | LCR | ~DLAB));
        _outp(p_add  | IER, 0x00);
        _outp(IMR, _inp(IMR) | IRQ_ADD2);
        _outp(p_add  | MCR, _inp(p_add  | MCR) | 0x00);
        _dos_setvect(IRQ_ADD2, old_handler2);
        p_add=0;
        _enable();
        cout<<"Program Terminated..."<<endl;
        exit(ABORT);
}

void COMPORT::csd_setup(void)
{
        send_char(CR);
        _putch(SP);
        delay(DLY_TIME);
}

void COMPORT::day_selector(int i)
{
        switch(i)
        {
                case 0:
                        strcpy(date_wday,"Sun");
                        break;
                case 1:
                        strcpy(date_wday,"Mon");
                        break;
                case 2:
                        strcpy(date_wday,"Tue");
                        break;
                case 3:
                        strcpy(date_wday,"Wed");
                        break;
                case 4:
                        strcpy(date_wday,"Thu");
                        break;
                case 5:
                        strcpy(date_wday,"Fri");
```

```
                        break;
                case 6:
                        strcpy(date_wday,"Sat");
        }
}

void COMPORT::dial_up(void)
{
        for(int i=0; i<8; i++)
        {
                send_char(dial[i]);
                rx_char_chat();
                delay(DLY_TIME);
        }
        rx_char_chat();
        send_char(CR);
        _putch(LF);
        delay(DLY_TIME);
}

void COMPORT::delay(clock_t wait)
{
        clock_t goal;
        goal=wait + clock();
        while(goal > clock())
                ;
}

void COMPORT::disconnect(void)
{
        int i, j;
        link = FALSE;

        cout<<"\nDisconnecting Now\n";
        start=timing();
        _outp(PORT_ADD2 | MCR, GPO2);
        while (!carrier())
        {
                if(time_out(3))
                {
                        for(i=0;i<3;i++)
                        {
                                send_char(hang_up[i]);
                                _putch(hang_up[i]);
                                delay(DLY_TIME);
                        }
                        for(i=0;i<3;i++)
                        {
                                send_char(hang_up1[i]);
                                delay(DLY_TIME);
                        }
                        while(!time_out(5))
                                {       }
                        send_char(CR);
                        for(i=4;i<7;i++)
                        {
                                send_char(hang_up[i]);
                                printf("%c",hang_up[i]);
                                delay(DLY_TIME);
```

```
                        }
                        send_char(CR);
                        _putch(CR);
                        _putch(LF);
                        for(j=1; j<4; j++)
                        {
                                send_char(at_at[j]);
                                printf("%c",at_at[i]);
                                delay(DLY_TIME);
                        }
                        break;
                }
        }
}

void COMPORT::get_time(void)
{
        int date_y;
        char am_pm[]="AM";
        struct tm *newtime;
        time_t long_time;
        time(&long_time);
         newtime=localtime(&long_time);
        if(newtime->tm_hour > 12)
                strcpy( am_pm, "PM" );
        if( newtime->tm_hour > 12 )
                 newtime->tm_hour -= 12;
        if( newtime->tm_hour == 0 )
                newtime->tm_hour = 12;
        _itoa(newtime->tm_sec,time_s,10);
        _itoa(newtime->tm_min,time_m,10
        _itoa(newtime->tm_hour,time_h,10);
        _itoa(newtime->tm_mday,date_md,10);
        date_y = newtime->tm_year;
        if(date_y == 0)
                end_century = 1;
        if(end_century == 1)
        {
                date_y += 2000;
                _itoa(date_y, date_yr, 10);
        }
        else
        {
                date_y += 1900;
                _itoa(date_y, date_yr, 10);
        }
        strcpy(date_year,date_yr);
        strcpy(time_hour, time_h);
        strcpy(time_min, time_m);
        strcpy(time_sec, time_s);
        strcat(time_sec,'\0');
        strcpy(date_mday, date_md);
        strcat(date_mday,'\0');
        strcpy(pm_am, am_pm);
        day_selector(newtime->tm_wday);
        month_selector(newtime->tm_mon);
}
```

```
COMPORT::header_block(void)
{
  int i=0;

            send_string(comp_path);
            send_char(CR);
            _putch(CR);
            _putch(LF);
            send_string(date_wday);
            send_char(CR);
            _putch(SP);
            send_string(date_month);
            send_char(CR);
            _putch(SP);
            i=0;
            send_numb(date_mday);
            send_char(CR);
            _putch(SP);
            send_string(date_year);
            send_char(CR);
            _putch(SP);
            send_string(time_hour);
            send_char(CL);
            _putch(CL);
            send_string(time_min);
            send_char(CL);
            _putch(CL);
            send_numb(time_sec);
            send_char(SP);
            send_string(pm_am);
}

void COMPORT::header_block1(void)
{

   char Pos[]="Position ";
            char hc[]="WARNING SECNET Device disconnected ";
            char pos[5];
            int i=0;

            room_number1();
            get_time();

            delay(DLY_TIME1);
            delay(DLY_TIME1);
            delay(DLY_TIME1);
            send_string(comp_path);
            send_char(SP);
            _putch(SP);
            send_string(date_wday);
            send_char(SP);
            _putch(SP);
            send_string(date_month);
            send_char(SP);
            _putch(SP);
            send_char(SP);
            _putch(SP);
            i=0;
```

```
        send_numb(date_mday);
        send_char(SP);
        _putch(SP);
        send_string(date_year);
        send_char(SP);
        _putch(SP);
        send_string(time_hour);
        send_char(SP);
        _putch(SP);
        send_string(time_min);
        send_char(SP);
        _putch(SP);
        send_numb(time_sec);
        send_char(SP);
        _putch(SP);
        send_string(pm_am);
        send_char(SP);
        _putch(SP);

        if (!hc_flag)
        {
                send_char(SP);
                _putch(SP);
                send_string(hc);
                send_char(SP);
                _putch(SP);
                send_char(CR);
                _putch(CR);
                _putch(LF);
        }
        else
        {
                for (i = 1; i < 5; i++)
                {
                        if (device2[i].problem && device2[i].pos_used)
                        {
                                int j = 0;
                                while (device2[i].type[j] != NULL)
                                {
                                        send_char(device2[i].type[j]);
                                        _putch(device2[i].type[j]);
                                        j++;
                                }
                                send_char(SP);
                                _putch(SP);
                                send_string(Pos);
                                _itoa(i, pos, 10);
                                send_string(pos);
                                send_char(SP);
                                _putch(SP);
                                send_char(CR);
                        }
                }
        }
        send_char(NULL);
}
```

```
void COMPORT::init_card(void)
{
        printf("\n Circuit Switched Data (CSD) Configuration at 38.4 kbps \n\n");
        for(int i=0; i<100; i++)
                ;
        csd_setup();
}

void COMPORT::init_port(int port)
{
        p_add = port;
        _outp(p_add | LCR, _inp(p_add | LCR) | state);
        _outp(p_add | LCR, _inp(p_add | LCR) | DLAB);
        _outp(p_add, dll);
        _outp(p_add | 0x01, dhl);
        _outp(p_add | LCR, _inp(p_add | LCR) & 0x7f);
        _outp(p_add | MCR, _inp(p_add | MCR) | GPO2|DTRIRTS);
        _outp(p_add | IER, 0x01);
        _outp(IMR, _inp(IMR) & port_mask);
}

inline int COMPORT::is_char_in(void)
{
        return !(rx_buf_in2 == rx_buf_out2);
}

void COMPORT::month_selector(int i)
{
        switch(i)
        {
                case 0:
                        strcpy(date_month,"Jan");
                        break;
                case 1:
                        strcpy(date_month,"Feb");
                        break;
                case 2:
                        strcpy(date_month,"Mar");
                        break;
                case 3:
                        strcpy(date_month,"Apr");
                        break;
                case 4:
                        strcpy(date_month,"May");
                        break;
                case 5:
                        strcpy(date_month,"Jun");
                        break;
                case 6:
                        strcpy(date_month,"Jul");
                        break;
                case 7:
                        strcpy(date_month,"Aug");
                        break;
                case 8:
                        strcpy(date_month,"Sep");
                        break;
                case 9:
```

```
                    strcpy(date_month,"Oct");
                    break;
            case 10:
                    strcpy(date_month,"Nov");
                    break;
            case 11:
                    strcpy(date_month,"Dec");
        }
}

void COMPORT::open_port(void)
{
        _disable();
        p_add=PORT_ADD1;
        port1_state=_inp(p_add | LCR);
        _outp(p_add | LCR, _inp(p_add | LCR | DLAB));
        port1_dll=_inp(p_add);
        port1_dhl=_inp(p_add | 0x01);
        _outp(p_add | LCR, _inp(p_add | LCR | ~DLAB));
        p_add=PORT_ADD2;
        port2_state=_inp(p_add | LCR);
        _outp(p_add | LCR, _inp(p_add | LCR | DLAB));
        port2_dll=_inp(p_add);
        port2_dhl=_inp(p_add | 0x01);
        _outp(p_add | LCR, _inp(p_add | LCR | ~DLAB)); /
        p_add=0;
        _enable();
}

void COMPORT::proc_char(unsigned char ch)
{
        send_char(ch);
        if(ch == CR)
                _putch(LF);
}

void COMPORT::purge_buff(void)
{
        _disable();
        rx_buf_in2 = rx_buf_out2 = 0;
        _enable();
}

void COMPORT::rx_char_chat(void)
{
        while(is_char_in())
        {
                if(__isascii(rx_buffer2[rx_buf_out2]))
                {
                        _putch(rx_buffer2[rx_buf_out2++]);
                        update_ptrs();
                }
        }
}
```

```
void COMPORT::rx_char_rxtx(void)
{
        unsigned char c;

        while(is_char_in())
        {
                c=character();
                if(!tx_rx)
                {
                        if((c == CntrlR) || (c == CntrlT))
                        {
                                com_char1 = c;
                                cout<<c;
                                tx_rx = TRUE;
                        }
                }
                if(__isascii(c))
                {
                        _putch(c);
                        if(char_in() == CR)
                                _putch(LF);
                        update_ptrs();
                }
        }
}

void COMPORT::select_mode(unsigned char ch)
{
        switch(ch)
        {
                case CntrlT:
                        break;
                case CntrlR:
                        break;
        }
}

void COMPORT::selection(unsigned char ch)
{
        switch(ch)
        {
                case CntrlQ:
                        disconnect();
                        break;
                case CntrlR:
                        send_char(ch);
                        break;
                case CntrlT:
                        if (device_problem_flag == TRUE)
                        {
                                send_char(ch);
                                room_number1();
                                get_time();
                                header_block1();
                                for (int i=0; i < 5; i++)
                                {
                                        if(device[i].problem)
                                                device[i].problem = FALSE;
```

```
                                        }
                                        device_problem_flag = FALSE;
                                }
                                else
                                {
                                        send_char(ch);
                                        room_number();
                                        get_time();
                                        header_block();
                                }
                                break;
                        default:
                          proc_char(ch);

                }
}

void COMPORT::send_atcom(char *ch)
{
  int i=0;

                while(ch[i] != NULL)
                {
                        send_char(ch[i]);
                        _putch(ch[i]);
                        delay(DLY_TIME);
                        i++;
                }
}

void COMPORT::send_char(unsigned char ch)
{
                int status

                _outp(PORT_ADD2 | MCR, GPO2|DTR|RTS);
                while(!transmitter_ready())
                {
                        do
                                status=_inp(PORT_ADD2 | MSR);
                        while(status & (CTS | DSR) != 1);
                }
                _disable();
                _outp(PORT_ADD2, ch);
                _enable();
}

void COMPORT::send_string(char *ch)
{
                int j=0;

                while(ch[j] != NULL)
                {
                        send_char(ch[j]);
                        _putch(ch[j]);
                        j++;
                }
}
```

```
void COMPORT::send_numb(char *ch)
{
        for(int i=0; i<=2; i++)
        {
                send_char(ch[i]);
                _putch(ch[i]);
        }
}

void COMPORT::setup_ports(void)
{

        _disable();
        setup_port1();
        init_port(PORT_ADD1);
        old_handler1=_dos_getvect(int_irq);
        _dos_setvect(int_irq,receive_com1);
        setup_port2();
        init_port(PORT_ADD2);
        old_handler2=_dos_getvect(int_irq);
        _dos_setvect(int_irq,receive_com2);
        _enable();
}

void COMPORT::setup_port1(void)
{
        p_add=PORT_ADD1;
        port_mask=IRQ_NO1;
        int_irq=IRQ_ADD1;
        state=0x00;
        dll=0x0c;
        dhl=0x00;
}

void COMPORT::setup_port2(void)
{
        p_add=PORT_ADD2;
        port_mask=IRQ_NO2;
        int_irq=IRQ_ADD2;
        state=0x00;
        dll=0x03;
        dhl=0x00;
}

int COMPORT::transmitter_ready(void)
{
        return _inp(PORT_ADD2 | LSR) & TX_RDY ? TRUE : FALSE;
}

void COMPORT::tx_rx_mode(void)
{
        unsigned char c;
        int key_hit;
        int end_of_tx = 0;
        int counter = 0;

        tx_rx = FALSE;
```

```
            while(TRUE)
            {
              count=0;
                        rx_char_rxtx();
                        key_hit = FALSE;
                        if(device_problem_flag)
                        {
                                    c = CntrlT;
                                    selection(c);
                                    end_of_tx = 1;
                        }
                        else if(end_of_tx)
                        {
                                    c = CntrlQ;
                                    selection(c);
                                    end_of_tx = 0;
                        }
                        if(_kbhit())
                        {
                                    c = _getche();
                                    selection(c);
      }
                        else if(tx_rx)
                        {
                                    select_mode(com_char1);
                        }
                        tx_rx=FALSE;
                        if(!carrier())
                        {
                                    link = FALSE;
                                    break;
                        }
            }
}

void COMPORT::room_number(void)
{
            char num[4];
            char rnumb[10];
            char full_path[25];

            printf("\nEnter Room Number: ");
            scanf("%s", rnumb);
            strcpy(full_path,path);
            strcat(full_path,rnumb);
            file_counter++;
            _itoa(file_counter,num,10);
            strcat(full_path,num);
            strcat(full_path,ext);
            strcpy(comp_path,full_path);
}

void COMPORT::room_number1(void)
{

            char num[4];
            char full_path[25];
            char space[] = "  ";
```

```
            strcpy(full_path,data_file_name);
            strcat(full_path,space);
            file_counter++;
            _itoa(file_counter,num,10);
            strcat(full_path,num);
            strcpy(comp_path,full_path);
}

void COMPORT::update_ptrs(void)
{
            if (rx_buf_out2 == BUF_MAX)
                        rx_buf_out2 = 0;
}


#endif
```