THE DEVELOPMENT AND IMPLEMENTATION OF A SINGLE-LINE INTELLIGENT DIGITAL TELEPHONE ANSWERING UNIT ON A PERSONAL COMPUTER

MARCHAND CHARL DU PLESSIS

Dissertation submitted in fulfilment of the requirements for the Degree

MAGISTER TECHNOLOGIAE:

ENGINEERING: ELECTRICAL

in the

Faculty of Engineering
Department of Electrical Engineering

at the

Technikon Free State

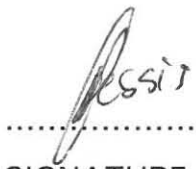
Supervisor: Prof. GJ Prinsloo, B.Eng; M.Eng; Ph.D.(Eng.)

BLOEMFONTEIN
April 1998

Central University of Technology, Free State

DECLARATION OF INDEPENDENT WORK

I, MARCHAND CHARL DU PLESSIS, do hereby declare that this research project submitted for the Degree MAGISTER TECHNOLOGIAE: ENGINEERING: ELECTRICAL, is my own independent work that has not been submitted before to any institution by me or anyone else as part of any qualification.



.....

SIGNATURE OF STUDENT

98/11/20

.....

DATE

ACKNOWLEDGEMENTS

First of all I would like to thank my Creator for the talents and strength He has given me in life.

There are several people who were major influences in my completion of this project. First and foremost was Professor GJ Prinsloo, my supervisor, who contributed countless hours reading and correcting my scribbles.

Thanks to my colleagues at Telkom SA for their support and ideas, especially Thys and Clive who made many constructive suggestions.

Thanks in particular goes to my parents, Ivor and Rosa, for all the support and opportunities they offered me.

Finally, I would like to express my deep and endearing appreciation to my wife, Sharon, for her encouragement and the time she sacrificed on my behalf throughout the project.

SUMMARY

Commercial telephone answering machines are limited to some extent by one or more of the following factors:

- limited facilities
- difficult to upgrade
- nonstandard telephone interfacing
- expensive
- lack of user-friendliness
- lack of dialogue and intelligence

The purpose of this study is to design an intelligent digital telephone system which will overcome as many of the above-mentioned problems as possible. The following features are proposed and will be discussed:

The use of a commonly available, but powerful, personal computer processor and memory instead of the elementary and rigid processor and magnetic tape storage units of the commercial telephone answering machine. This allows the quick storage and retrieval of digitized messages, each with its individual name, time and date stamp.

Using the personal computer's hardware and not duplicating the processor and memory units allows a more cost-effective system upgrade. Upgrades mainly consist of software changes and minor hardware changes. This means that an upgrade does not implicate a total hardware redesign.

Standards as prescribed by the local switching network standards and the Department of Post and Telecommunications, apply to this design and are applicable for licensing of the product.

It is evident that the cost of this project and design is kept minimal by not duplicating expensive components like the microprocessor and the memory units, although these are used in the design. In this respect upgrades are software orientated to further limit the costs.

The personal computer is equipped with a display which allows the user to make easy selections in order to execute the required instructions or to obtain information by using the help functions. This real-time help function eliminates the need for a user manual.

Dialogue between user and personal computer over the telephone network offers a simple method of delivering information without the need for any extra equipment such as modems, keyboards or display units.

The software used on the personal computer is designed in such a way that the system is intelligent and capable of decision making. Communication from the public telephone network is possible by using the telephone keypad and Dual Tone Multifrequency (DTMF) signalling.

OPSOMMING

Kommersieel beskikbare telefoonantwoordeenhede word deur een of meer van die volgende faktore beperk:

- fasiliteite
- moeilik opgradeerbaar
- onaanpasbaar by alle Suid-Afrikaanse skakelnetwerktoestande
- hoë koste by fasiliteituitbreiding en -opgradering
- beperkte gebruikersvriendelikheid
- beskik nie oor dialoogvoering of "intelligensie" nie

Die doel van die studie is om 'n intelligente digitale telefoonantwoordeenheid te ontwikkel wat die probleme en tekortkominge van bestaande telefoonantwoordmasjiene sal oplos. Die volgende doelwitte word daargestel en bespreek:

Die bestaande telefoonantwoordmasjien het 'n elementêre en rigiede prosesverwerking- en stooreenheid. Dit word vervang met dié van 'n persoonlike rekenaar wat kragtig, intelligent en veeldoelig is. Bykomende fasiliteite, soos die tyd en datum van elke boodskap wat gelaat is, word verskaf.

Deur gebruik te maak van die persoonlike rekenaar se hardeware, word duur verwerkings- en geheue-eenhede nie gedupliseer nie. Dit bied die daarstelling van 'n meer koste-effektiewe opgradeerbare eenheid. Opgraderings behels dan oorwegend hoofsaaklik sagtewareveranderinge en in 'n mindere mate hardewareveranderinge. 'n Totale hardewareherontwerp word op dié manier tydens opgraderings beperk.

Die ontwerp sal in oorleg met plaaslike skakelnetwerkstandaarde ontwikkel word. Dit sal verseker dat die eindproduk aan die Departement van Pos en Telekommunikasie se spesifikasies voldoen, soos van toepassing vir lisensiëring.

Kostes word laag gehou deur duur komponente, soos mikroprosesseerders en stoorkomponente, wat reeds bestaande komponente van die persoonlike rekenaar is, nie te dupliseer nie, maar wel in die ontwerp te gebruik. Omdat opgradering nou sagteware georiënteerd is, word kostes verder tot die minimum beperk.

Die rekenaar met 'n vertoonskerm maak dit egter moontlik om 'n gebruikersvriendelike intervlak daar te stel waar die gebruiker hoofsaaklik van keuseselektering en -uitvoering gebruik maak om die verlangde resultaat of toestand te aktiveer. Verder is dit moontlik om 'n intydse hulpfunksie daar te stel wat die gebuik van 'n handleiding sal uitskakel.

Interaktiewe spraak tussen rekenaar en mens is moontlik vanaf 'n telefoonnetwerk sonder die gebruik van addisionele toerusting soos modems, sleutelborde en vertooneenhede.

Die rekenaar maak dit moontlik om sagteware te ontwikkel en sodanig daarop aan te wend dat die stelsel oor besluitneming- en dialoogvoeringsfunksies beskik. Kommunikasie met die rekenaar kan deur middel van spraak of "Dual Tone Multifrequency"- (DTMF) seining geskied.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Central University of Technology, Free State

LIST OF ABREVIATIONS

A/D	Analog to Digital
AC	Alternating Current
ADM	Adaptive Delta Modulation
ADPCM	Adaptive Differential Pulse Code Modulation
APCM	Adaptive Pulse Code Modulation
CCITT	International Telegraph and Telephone Consultative Committee
CODEC	COder / DECoder
CPU	Central Processing Unit
D/A	Digital to Analog
DC	Direct Current
DM	Delta Modulation
DMA	Direct Memory Access
DOS	Device Operating System
DPCM	Differential Pulse Code Modulation
DRAM	Dynamic Random Access Memory
DSP	Digital Signal Processor
DTAS	Digital Telephone Answering System
DTMF	Dual Tone Mulifrequency
FIFO	First In First Out
FS	Frame Synchronization
IBM	International Business Machines
LDM	Linear Delta Modulation
LED	Light Emitting Diode
LPC	Linear Predictive Coder

LTS	Line Test System
MIDI	Musical Instrument Digital Interface
MPU	Microprocessor Unit
OMC	Operations and Maintenance Center
PAM	Pulse Amplitude Modulation
PC	Personal Computer
PCM	Pulse Code Modulation
PIC	Programmable Interrupt Controller
RAM	Random Access Memory
RTU	Remote Test Unit

CHAPTER 1

1. INTRODUCTION

"Speech is civilization itself. The word, even the most contradictious word, preserves contrast - it is silence which isolates" - Thomas Mann, The Magic Mountain, 1924.

Speech is the principal means of human communication and dominates the world's communications, of which the telephone system is the most extensive structure of all time. The familiarity with the telephone terminal is now converging with the growing tendency to store more and more information in computer databases. The ubiquity of the telephone system therefore presents an opportunity for direct access to the computer databases, provided that the problems of interactive speech systems can be overcome. The way the users adapt themselves to interfacing with the machine is an important aspect of a speech system. The key to success or failure can often be the user's motivation in using the technology.

"If speech is civilization, the future aim of designers must be to produce human computer interfaces that enable a more natural and useful dialogue in the form of a conversation" [46].

In this respect the following factors in commercial telephone answering systems were identified and need to be addressed to produce an acceptable man-machine communication system:

- limited facilities
- difficult to upgrade
- nonstandard telephone line interfacing
- expensive
- user-friendliness
- the lack of dialogue and intelligence

Figure 1.1 **Basic system setup**

1.1 PURPOSE OF THIS STUDY

The purpose of this study was to design an intelligent digital telephone answering system that will overcome as many of the above-mentioned problems as possible. The following features are proposed and will be discussed for the system as shown in figure 1.1:

- The use of the processor and memory of a commonly available, but powerful, personal computer instead of the elementary and rigid processor and magnetic tape storage units of the commercial telephone answering machine. This allows the quick storage and retrieval of digitized messages, each with its individual name, time and date stamp.

- Using the personal computer's hardware and by not duplicating the processor and memory units allows a more cost-effective system upgrade. Upgrades mainly consist of software changes and minor hardware changes. This means that an upgrade does not implicate a total hardware redesign.

- Standards as prescribed by the CCITT blue book and the Department of Post and Telecommunications apply to this design and are applicable for licensing of the product.

- It is evident that the cost of this project and design is kept minimal by not duplicating expensive components like the microprocessor and the memory units, although these are used in the design. In this respect upgrades are software orientated to further limit the costs.

- The personal computer is equipped with a display which allows the user to make easy selections in order to execute the required instructions or to obtain information by using the help functions. This real-time help function eliminates the need for a user manual.

- Dialogue between user and personal computer over the telephone network offers a simple method of delivering information without the need for any extra equipment such as modems, keyboards or display units [46].

- The software used on the personal computer is designed in such a way that the system is intelligent and capable of decision making. Communication from the public telephone network is possible by using the telephone keypad and Dual Tone Multifrequency signalling [27].

1.2 HYPOTHESIS

An intelligent digital telephone answering unit can be implemented on a personal computer by using an analog-to-digital hardware interface and designing modular software.

" 'n Intelligente digitale telefoon antwoordeenheid kan op 'n persoonlike rekenaar geïmplementeer word deur die ontwikkeling van modulêre sagteware en die gebruik van koppelvlak analoog na digitale omsetter hardeware."

1.3 SUMMARY

In this chapter we have described the purpose and the hypothesis of the study. In Chapter 2 we look at the various factors which influence the merging of speech and computer technology. In this chapter we also discuss the technology that is available to the designer and how it works. Chapter 3 describes the many variations and techniques for the digitization of speech signals. In chapter 4 we explain the development of the PC-based telephone answering system hardware and the factors which distinguish this design from commercial equipment. We then, in Chapter 5, explain the key procedures in the development of the software for the PC-based telephone answering system, and in Chapter 6 we look at the experimental methods and results for evaluating the system to see if it meets user requirements. Finally, in Chapter 7, we summarize the design of the PC-based telephone answering system.

CHAPTER 2

2. MERGING SPEECH AND COMPUTER TECHNOLOGY

2.1 A SHORT HISTORY - MODELLING THE VOCAL MECHANISM

Speech communication can be said to be the basic and most essential capability possessed by human beings. Without this important method and advanced form of communication the establishment of society as we know it would have been impossible. Although written language is effective for exchanging knowledge, the amount of information exchanged by speech is considerably larger and therefore clearly plays a very significant role in our lives [14].

Since man has always been fascinated by his ability to speak, as early as the eighteenth century attempts were made to model the human speech mechanism.

In 1780 Professor Christian Kratzentein designed a "Vox humana" capable of producing the five vowel sounds from a set of different and unusually shaped tubes.

In 1791 Wolfgang Ritter Von Kempelen constructed a talking machine which could produce about twenty different sounds! [46].

Without the aid of electricity these were physical analogs of the vocal apparatus - lungs, vocal cords, vocal tract, tongue, teeth and lips - but were accurate enough to yield sounds recognizable as human utterances. If such a model of speech production was possible using a purely physical analog, then clearly modelling with electrical impulses, and later with digital computers, was an easy extrapolation [6].

"Watson, I have another idea I haven't told you about that I think will surprise you. If I can get a mechanism which will make a current of electricity vary in its intensity as the air varies in density when a sound is passing through it, I can telegraph any sound, even the sound of speech." This became the central concept which came to fruition as the telephone in the following year. The invention of the telephone represents the first step in which speech began to be dealt with as an engineering target. Before the invention of pulse code modulation (PCM) in 1938, the speech wave had been dealt with by analog-processing techniques [14].

It was not until 1939, with a device called the "Voder" (Voice Operation Demonstrator), however, that a machine was able to produce connected speech. The Voder was produced at the Bell Telephone Labs in New Jersey, and could be operated by trained operators to emit speech sounds (see Figure 2.1). This electrical marvel used a set of 10 band-

pass filters, each controlled by one finger of the operator, to simulate the variable resonances of the vocal cavity.

After the Second World War, much more advanced speech synthesizers were produced using developments of the same technique. The development of digital circuits and electronic computers in the 1960's has made possible the digital processing of speech and has brought about remarkable progress in laboratories and universities around the world [6].

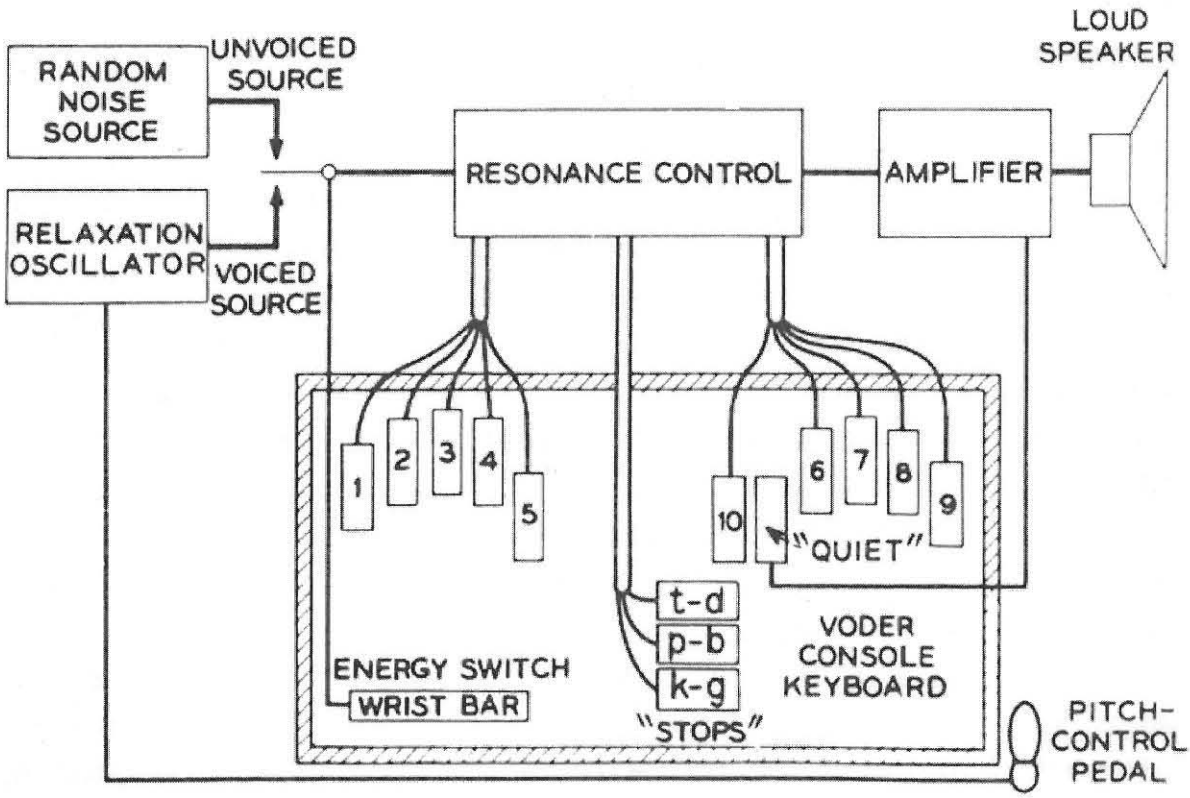


Figure 2.1 **Schematic diagram of the Voder**

2.2 VALUE OF SPEECH FOR MAN-MACHINE COMMUNICATION

Speech is by far the most widely used and natural means of communication between people and requires no special training. Although this is an effective way of transferring information, the problems regarding communication between human beings and information-processing machines become increasingly important. As information technology continues to make a greater impact on many aspects of our daily lives, there is a need for effective means of exchanging information between people and machines. Speech at first seems an obvious substitute, but this deceptively simple means of exchanging information is, in fact, extremely difficult. However, there is no practicable alternative for applications where the machine is only accessible from a standard telephone [17].

2.3 HUMAN-COMPUTER COMMUNICATION

2.3.1 The Human

The human, the user, is the one whom computer systems are designed to assist, and therefore the requirements of the user should be a priority in any interactive system design.

When a person interacts with a computer the user receives information that is conveyed by the computer, and responds by providing input to the computer. Information is received through the senses, particularly through sight, hearing and touch. It is stored, either temporarily in sensory or working memory, or permanently in long-term memory. This can then be used in reasoning and problem-solving. It is important to understand the capabilities and limitations of the human as information processor when designing interactive systems [8].

2.3.2 The Computer

The computer can be defined as the participant in the interaction that runs a program. What are we trying to achieve when we interact with computers? Relating to the reason for human-human interaction, the same reason applies to human-computer interaction. The same principles hold, whereby interaction is a process of information transfer from the user to the computer and from the computer to the user. The computer interactive input devices are split into two categories: those that allow text entry, and those that allow pointing, selection of particular items on the screen, and movement. The first category consists of devices such as keyboards and speech recognition systems, whilst the second category comprises devices such as the mouse, joysticks and touchscreens.

The one predominant output device is the computer screen. Other visual and auditory output devices that deserve mention range from modes of communication designed to supplement the screen, as well as those targeted as principal output devices. Visual outputs can take the form of analog representation of numerical values, such as dials, gauges or lights, to signify a certain system state. Gauges and dials are found in process control systems, whilst flashing light-emitting diodes (LED's) are used to signify the processor state in the backs of some computers.

The other mode of output, which is often designed to be used in conjunction with screen displays, is that of auditory signals. Sound offers an important level of feedback in interactive systems. It is evident that the sound occurring when a keyboard or telephone keypad is pressed signifies that the key has been successfully pressed, whilst the actual tone provides some information about the particular key that was pressed.

Like the human memory, computer memory can be grouped into short-term and long-term memory. Short-term memory is the most current active information which is held in random access memory (RAM). Most RAM is volatile, that is, its memory is lost when the power is turned off, but then again, it does have the faster access time.

2.4 USER-FRIENDLINESS AND USABILITY

2.4.1 User-friendliness

The so-called user-friendliness of equipment will, to a large extent, determine the satisfaction of the users and what their expectations are. It also appears that users who work several hours a day with a piece of unfriendly apparatus will have no difficulty with it. They learn to live with the unfriendly design and may even be proud of the fact that they are able to work with the equipment and others are not.

For users who make only sporadic use of that same equipment, or who often have to work with various types of equipment and many different facilities, the user-friendliness of such equipment soon becomes apparent. The success of equipment in this casual user market will ultimately depend on its user-friendliness. For casual users the ease of operation of the equipment is of great importance, and equipment for which the user manual has to be consulted for operation purposes will either not be used at all, or only a limited number of its facilities will be used.

Some common failings of human interface are:

- Reporting that an error has occurred without giving any indication of what remedial action is required or what has caused the error.
- No confirmation that an action has been carried out.
- Too little use of graphical information or colour.
- Lack of consistency in the use of abbreviations, names and symbols [10].

2.4.2 Usability

The primary objective of an interactive system is to allow the user to achieve particular goals in some application domain, that is, the interactive system must be usable. Therefore it is necessary to focus on the general principles which can be applied to the design of an interactive system, in order to promote its usability. The principles are divided into three main categories:

- Learnability - the ease with which new users can begin effective interaction and achieve maximal performance.
- Flexibility - the multiplicity of ways in which the user and system exchange information.
- Robustness - the level of support provided to the user in determining successful achievement and assessment of goals [17].

2.5 SPEECH SYNTHESIS

As discussed at the beginning of this chapter, several years ago the term "speech synthesis" was almost exclusively used for machine-generated speech sounds which modelled the human speaking system. These applications were mainly for research in speech production and perception.

Today, speech synthesis has come to mean provision of information in the form of speech from a machine, in which the messages are structured dynamically to suit the particular circumstances required. Some of the applications include simple information services, reading machines for the blind and communication aids for people with speech disorders. There is, however, also speech synthesis which can be an important part of complicated man-machine systems. In these systems various types of structured dialogue can be made using voice output, with either automatic speech recognition or by means of key pressing for the man-to-machine direction of communication [17].

2.5.1 Synthesis techniques

Speech synthesis techniques are divided into three main classes:

- Synthesis based on waveform coding, in which speech waves of a recorded human voice are stored after waveform coding or immediately after recording, and are used to produce desired messages.

- Synthesis based on the analysis-synthesis method, in which speech waves of recorded human voice are transformed into parameter sequences by the analysis-synthesis method and are stored with a speech synthesizer, driven by concatenated parameters to produce messages.

- Synthesis by rule, in which speech is produced based on phonetic and linguistic rules from letter sequences or sequences of prosodic features and phoneme symbols, which is the smallest unit in speech where substitution of one unit for another might make a distinction of meaning [35].

The synthesis system, based on the waveform coding method, is simple and provides high-quality speech, but the usage is limited to applications with few messages. This type of synthesis includes coding methods of differential pulse code modulation (DPCM), adaptive delta modulation (ADM), and adaptive differential PCM (ADPCM).

In synthesis based on the analysis-synthesis method, units are stored by source and spectral envelope parameters. Although this method is advantageous in that changing the speaking rate and smoothing the pitch and spectral change at connections can be performed by controlling the parameters, the naturalness of the synthesized speech is degraded. Speech synthesizers based on LPC analysis methods and channel vocoders are used for this purpose.

Synthesis by rule is a method by which future parameters for fundamental small units of speech such as syllables, phonemes or one-pitch-period speech, are stored and connected by rule. Although memory capacity can be greatly reduced when phonemes are taken as the fundamental units, the rules for connecting phonemes are so complicated that high-quality speech is hard to obtain. In this method, vocal tract analog, terminal analog, and LPC speech synthesizers are usually used for speech production [14].

In the next chapter the principles of some of these coding methods will be discussed in more detail.

2.6 SPEECH RECOGNITION

Speech recognition is the process of extracting and determining linguistic information from a speech signal by using computers or electronic circuits. As in the case of speech synthesis, the linguistic (phonetic) information is the most important information in the speech wave. Speech recognition also involves the extracting of individual information indicating who is speaking, which is called speaker recognition [14].

In the 1950's the earliest attempts at electronic speech recognition were made and mainly concentrated on the recognition of spoken digits. These earlier devices used decision logic to make the pattern match between the uttered word and the reference word [46].

2.6.1 Difficulties in speech recognition

It has been assumed that speech recognition could be easily achieved by machines, since it seems so natural and simple for humans. However, it has proved to be an extremely complex and difficult task to get machines to respond to even simple spoken commands.

Speech recognition is complicated by variations in pronunciation - a spoken word contains more acoustic information than the written equivalent and is rarely pronounced in the same way twice. Strong accents, a cold or emotion can also cause recognition problems and therefore the speaker's psychological and physiological state influences the speed and consistency of delivery. Speech recognition is hampered further by consonants adopting aspects of neighboring consonants and vowels in fluent speech [46].

2.6.2 Classification of speech recognition

Speech recognition can be classified, as shown in Table 2.1, into isolated word recognition and continuous speech recognition. In isolated word recognition, words uttered are recognized in isolation, whereas in continuous speech recognition continuously uttered sentences are recognized. Continuous speech recognition can be further classified into connected word recognition, which recognizes a relatively small vocabulary, and conversational speech recognition, which on the other hand recognizes a relatively large vocabulary. The latter, also called speech understanding, focuses on understanding the meaning of sentences rather than on recognizing each word and requires sophisticated linguistic knowledge [14].

	Speaker dependency	
Object	Speaker-dependent	Speaker-independent
1. Isolated word recognition	X	X
2. Continuous speech recognition		
2a. Connected word recognition	X	X
2b. Conversational speech recognition (Speech understanding)		

X = Commercialized

Table 2.1 **Classification of speech recognition methods**

From a different point of view, speech recognition can also be classified into speaker-dependent recognition and speaker-independent recognition. Speaker-independent systems can recognize speech uttered by any speaker, while speaker-dependent systems require the reference templates to be modified every time the speaker changes. Clearly, speaker-independent recognition is much more difficult than speaker-dependent recognition and therefore the former has been commercialized only for isolated word recognition [14].

2.7 SPEECH RECORDING

2.7.1 Why tapeless recording?

If one looks back over the past century it will be seen that tape as a medium has played a key role in improving audio quality and enabling possible operational techniques which otherwise would have been precluded. It must be said, none the less, that tape recording has its disadvantages. Not least among them is the time it takes to find a particular section of recorded material. Owing to the serial nature of the tape, it is often necessary to wind through large amounts of unwanted material before arriving at the desired section [36].

Magnetic tape recording has also been used in applications requiring speech announcements, for example in telephone announcement systems that informs you of the time. This device is costly because it requires a large number of tapes for different messages and will not let one create mixed messages for different situations. In contrast, this system could be replaced with a system with a few chips that will do the work of a large number of tapes. A voice stored in digital memories can create different messages by pulling up the different words required to create a specific message [22].

Furthermore, tape recorders remain electromechanical beasts which wear down, break down and are expensive to keep in good repair. In contrast, a fully electronic system composed of a few integrated circuits can usually be marketed in high volume and is inexpensive to maintain. Electronic speech will remain less expensive than tape [28].

2.7.2 Implications of digital recording

What is tapeless recording? It is obvious that this would appear to imply any recordings which are not made on tape. Recordings which make use of other media, such as solid-state memory and disk recordings, are tapeless recordings, the one principal common factor being the speed with which different parts of the recorded material may be accessed. In this context, tapeless recording relies heavily on computer technology and computer storage media. It is furthermore facilitated by two things: firstly, the fact that sound can be converted into numerical form (digital audio), and secondly, the fact that computer storage devices (mainly disk drives) have developed to such an extent that they are viable as stores for digital sound [37].

Digital storage of digitized voice is particularly appropriate for recorded announcements or for voice messaging. This is because the playback

quality does not deteriorate with time and individual announcements stored in memory or on disc can be randomly accessed [3].

The development of digital tapleless recording means that tape can no longer claim to have the highest sound quality. As a result of the merging of audio and computer technology, many other advantages will also be gained. Concepts such as the database may now be applied to sound, making it much easier to call up required material by telling the system what is required and leaving it to worry about where and when the material is actually stored [36].

The techniques used in digital recording are rather different from those in analog recording. One of the great merits of digital recording is that by expressing a signal in binary numerical form by sampling and quantizing, the quality becomes independent of the medium, whereas in analog recording the nature of the storage medium more directly affects the quality of the recorded signal [45].

With digital recording, noise or distortion introduced due to the recording or transmission process, may well not affect the sound quality of replay. Once the sound is represented by a string of numbers, the sound may be changed by changing the numbers electronically. A further point to consider is that computers are used to handling numbers for a variety of

non-audio purposes and are mainly concerned with moving numbers from place to place in their own time. The "real-time" entity of audio requires that it should be played back continuously and should not contain pauses. This is in contrast with the "nature" of a computer which may take a few seconds to display a text file stored on a disk, with short waiting periods while more data are collected from disk. Therefore the designer of a digital recording system will need to rethink the usage of computer technology to ensure a constant output of sound when required [36].

2.7.3 Digital recording systems

2.7.3.1 Historical precedent

It is interesting to see from which roots the modern digital recording system has evolved. Commercial tapeless systems have been conceived from the start as professional audio recording and editing systems. However, there are a small number which stem from the root of music sampling and synthesis. This can be seen in the MIDI-controlled music sampler, which is really a basic tapeless recorder. MIDI stands for Musical Instrument Digital Interface and is widely used in audio systems as a means of remote control for musical instruments and other devices which are related to music systems. It is important to understand the difference between a sampler which uses RAM to store sound, which

may be triggered under MIDI control, and a tapeless recorder which uses a disk drive to store sound files. The sampler may also employ a disk drive to store sounds in the long term, but they must be loaded to RAM before they can be accessed fast enough [36].

2.7.3.2 Technical concepts - Digital system overview

It should not be forgotten that the aim of tapeless digital recording is to arrive at a flexible sound recording and retrieval system which operates seamlessly as far as the user is concerned. Therefore various technical concepts are important to the understanding of the way in which this is achieved. Although in general every commercial system available differs in the implementation of the under-mentioned principles, it is possible to describe a basic block diagram of a generalized tapeless digital recording system.

The system, as shown in Fig 2.2, consists firstly of a user interface which displays information and handles commands, communicating with the central processor (CPU). The CPU handles the overall control of operations of the system. The digital signal processor (DSP) handles the real-time processing of audio data. A block of solid state RAM (random access memory) acts as a temporary store for audio data. Furthermore the system consists of a DMA (direct memory access) controller and a

storage device which is likely to be a disk drive with its associated controller. There are also analog-to-digital and digital-to-analog interfaces, connected to the system via further memory to act as a buffer. First-in-first-out (FIFO) buffering is employed to smooth inputs and outputs. A timing interface controls the synchronization in the system.

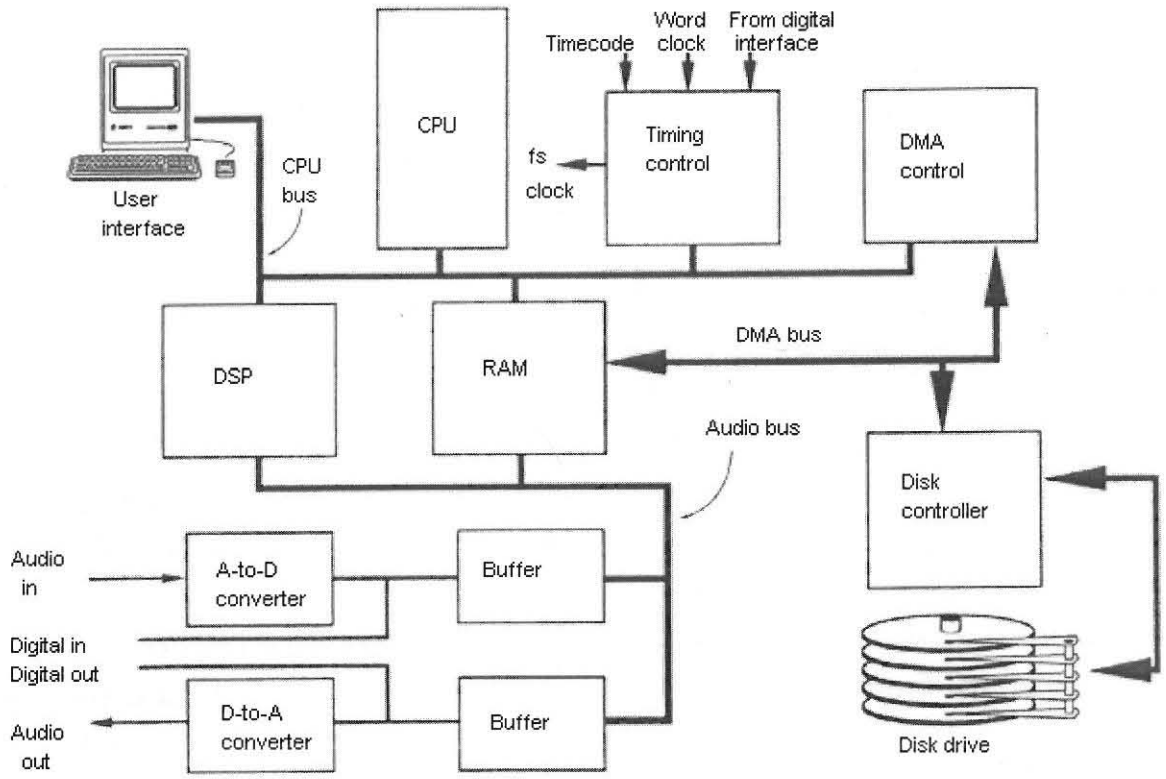


Figure 2.2 **Conceptual block diagram of a tapeless recording system**

In the operation of the system, audio data are transferred to and from the store to the RAM. Usually a technique known as direct memory access is used, whereby data may be transferred directly from the store to the RAM or visa versa without having to pass via the central processor. This assists in speeding up the transfer of data. Data are transferred between RAM and the audio interface via the buffers, under the control of the central processor which takes commands from the user interface [36].

2.8 MODERN RECORDING AND ANSWERING EQUIPMENT

Oki semiconductor produces different types of solid state recorders in the 62XX series. The MSM6258 contains an adaptive differential PCM (ADPCM) speech processor implemented in CMOS technology, as illustrated in Figure 2.3.

The MSM6258 is a stand-alone version with a voice detector circuit and a 8-bit MPU interface. Analog-to-digital and digital-to-analog converters are contained in both chips and can be looped to connect external devices. They also feature a static RAM interface accepting a maximum of 128 Kbytes, and a static RAM interface for a maximum of two megabytes. Furthermore the recorder contains recording and playback outputs, and three sampling frequencies are possible : 4.0, 5.3, and 8.0 kHz at 4.096 MHz [22].

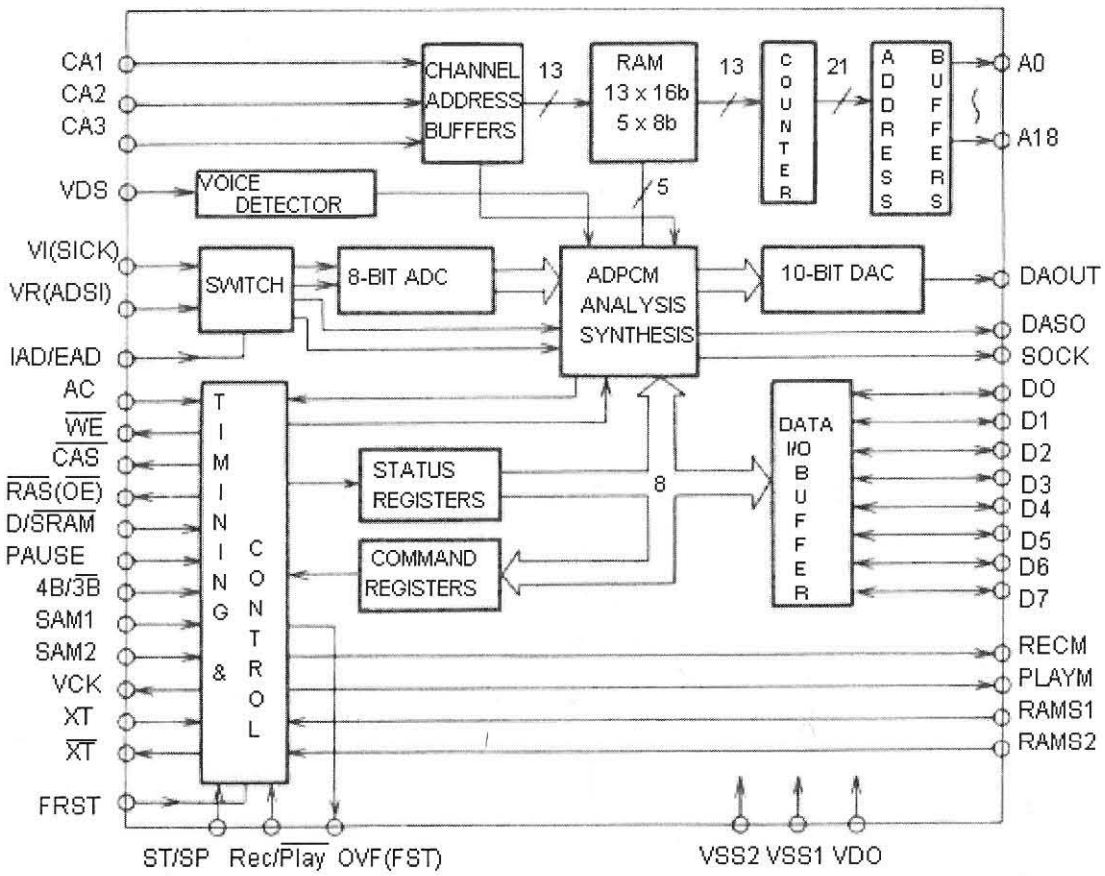


Figure 2.3 **The MSM 6258 Oki Semiconductor solid state recorder**

Dallas semiconductor also produces a digital answering machine processor - the DS2132A. This is a Digital Signal Processor (DSP) optimized for the compression/expansion of PCM coded voice to/from an extremely low bit rate. The standard Record/Playback algorithm compresses speech to 9.8Kbps or 4.9 Kbps. The DS2132A detects and generates all 16 DTMF tones and can also generate a wide variety of call progress tones. The system, as shown in figure 2.4, consists of a

standard telephone CODEC (COder-DECoder) device, the DS2132A, a microcontroller, and a bank of DRAM. Although a wide variety of CODEC's and microcontrollers can be used with the DS2132A, the implementation shown is with a Hitachi CODEC and a 8051-type microcontroller.

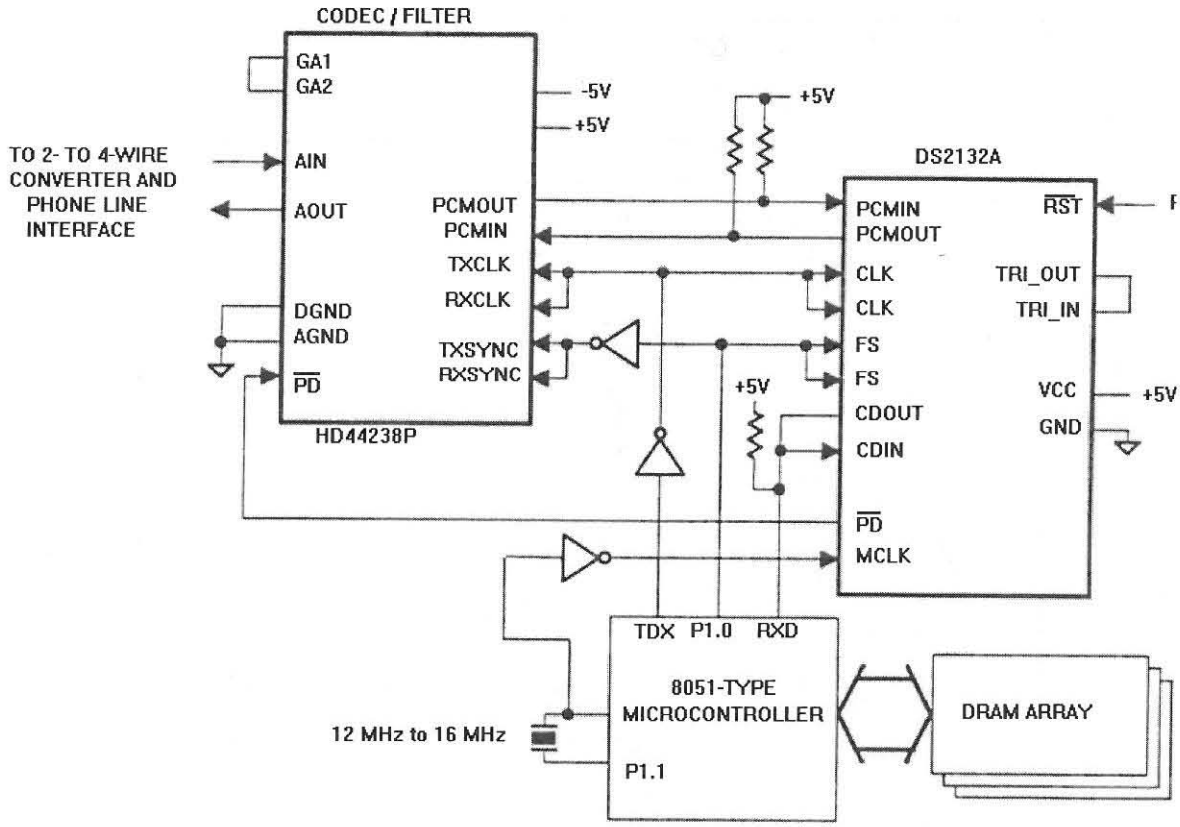


Figure 2.4 **A typical commercial digital answering machine**

In the operation of the system the microcontroller creates the clock (CLK) and frame synchronization (FS) that is applied to both the CODEC and the DS2132A. In this manner the DS2132A shares the signals necessary

to drive the CODEC. The following occurs in the process of recording an audio signal. The analog signal applied at the AIN pin of the CODEC is converted to eight-bit values and output at PCMOUT every 125μs. The DS2132A then takes these eight-bit samples in at the PCMIN pin and effectively compresses them to either 9.8 Kbps or 4.9 Kbps. The compressed data is then passed to the microcontroller via the CDOUT pin. The microcontroller then stores the compressed speech in the DRAM array. The inverse of this process is required to "play back" the message at the AOUT pin of the CODEC [19].

2.9 CONCLUSION

This chapter has covered some of the technical and human-factor tradeoffs involved in merging speech and computer technology. Since the invention of the telephone and the Voder, man-machine communication systems have witnessed remarkable progress. From the basic principles of various related speech technologies, as discussed in this chapter, it is evident that continuing improvement in effectiveness and naturalness of recording devices, such as digital telephone answering systems, will depend on creative synthesis of concepts and results from many fields. Given the exceptional flexibility of tapeless digital recording, it may be that both economic and creative advantages exit in its adoption.

However, when scrutinizing modern recording and answering equipment used in association with personal computers, it is clear that the general trend is to duplicate the processing and memory units on these systems. Thus, the feasibility of not duplicating expensive processor and memory modules in the system design, when implemented on a personal computer, needs to be investigated. Such a design will be proposed in chapter 4.

CHAPTER 3

3. DIGITIZATION OF SPEECH

3.1 INTRODUCTION

Chapter 2 looked at the various factors which influence the merging of speech and computer technology. Chapter 2 also discussed the technology that is available to the designer and how it works. This chapter describes the many variations and techniques for the digitization of speech signals.

The rapid development of computers and the growth of digital communications networks have encouraged the application of digital processing techniques to speech processing. It is possible to convert the speech signal into a electrical signal by using a microphone. The electrical signal is then transformed from an analog into a digital signal, because digital techniques facilitate highly sophisticated signal processing and are far more reliable than analog techniques. Analog-to-digital conversion, commonly referred to as digitization, consists of sampling, quantizing and coding processes [14].

Due to its usefulness in a variety of applications and its interesting nature, digitization of speech continues to be an area of intense research and has produced many types of speech digitization algorithms. The implementation cost and the performance requirements implied by the application will determine the type of digitization chosen. Digitization techniques can be broadly categorized into two classes. The first category is the encoding of the waveform as faithfully as possible and this is representative of the general problem of analog-to-digital and digital-to-analog conversions. Pulse code modulation (PCM), differential PCM (DPCM) and delta modulation (DM) are the most common techniques used to encode a voice waveform.

The second category of voice digitization is used by digital storage devices with limited capacity, and is concerned primarily with producing very low data rate speech encoders and decoders, commonly referred to as a "vocoder" (voice coder). Vocoder techniques generally produce unnatural or synthetic sounding speech and do not provide adequate quality, although such techniques are capable of producing intelligible speech.

Considering these digitization categories, the following discussions will emphasize the basic principals and applications of PCM waveform coding [3].

3.2 PULSE CODE MODULATION

3.2.1 Sampling

Sampling is the process for depicting a continuously varying signal as a periodic sequence of values. Waveform coders attempt to copy the actual shape of the waveform. The Nyquist Sampling theorem shows that it is theoretically possible to reconstruct the original waveform exactly from a specification in terms of the amplitudes of regularly spaced ordinate samples taken at a frequency of at least twice the bandwidth, and the waveform can be completely recovered if the samples occur often enough. In its conceptually simplest form a waveform coder consists of a low-pass filter, a sampler and a device for coding the samples. This technique, represented in Figure 3.1, involves the amplitude modulation of a chain of pulses by the analog audio signal in a process known as pulse amplitude modulation (PAM) [3].

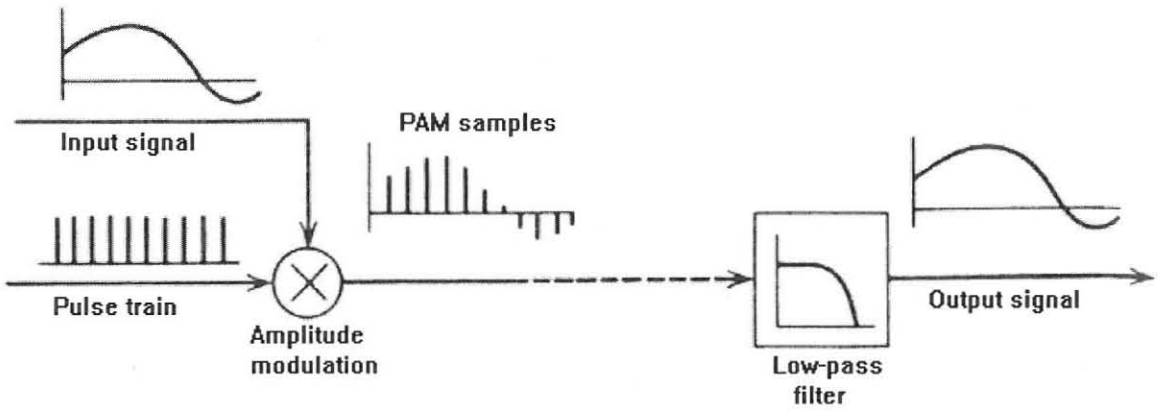


Figure 3.1 **Pulse amplitude modulation**

3.2.2 Foldover Distortion (Aliasing)

Foldover distortion is caused when the waveform of a PAM signal is sampled contrary to the sampling theorem. The input waveform cannot be recovered without distortion. Frequencies higher than the Nyquist frequency (half the sampling frequency) will "alias"; in other words, they will be reflected back into the original spectrum, and thus another term for this is aliasing distortion. For those signals with an unknown frequency bandwidth, it is vital that a low pass filter is employed before conversion to ensure that frequencies which would otherwise alias, are rejected.

Fig 3.2(a) shows an example of a 4kHz signal, with bandwidth W which is restricted under 4kHz and sampled at 10 kHz. FIG 3.2(b), indicates an aliasing process occurring in speech if the same signal is sampled at a 6kHz rate. S = 1/T [Hz], where S is called the sampling frequency and T is termed the sampling period. Notice how aliasing distorts the high-frequency components of the signal which is discontinuous in the time domain but still continuous in the amplitude domain [14].

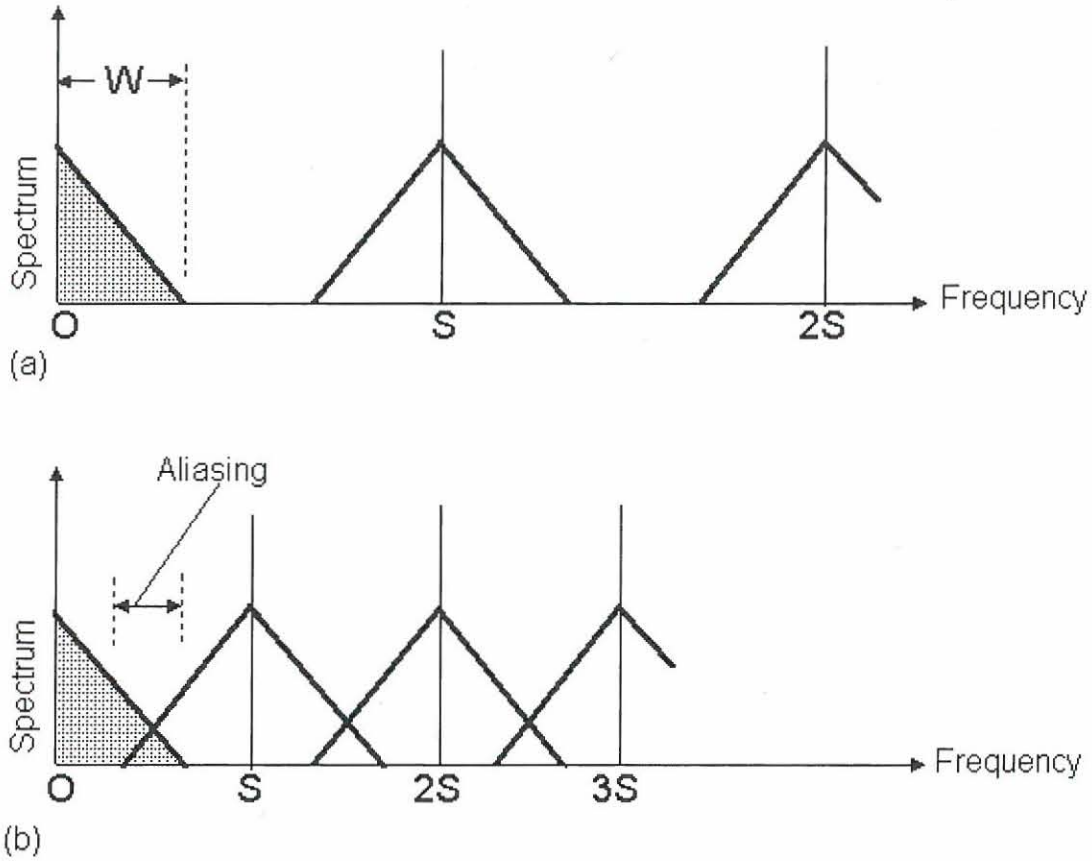


Figure 3.2 **Sampling in the frequency domain:**

(a) correct sampling (S ≥ 2 W)

(b) incorrect sampling (S < 2W)

3.2.3 Quantization

The amplitude samples produced by pulse amplitude modulation (PAM) are still analog waveforms of the voice signal. The amplitude samples can be processed much easier if they are available in digital form. Quantization is the process of assigning discrete binary values to sampled PAM pulses, resulting in PCM (pulse code modulation). This principle of coding was suggested by Reeves (1938), and is now widely used for

feeding analog signals into computers or other digital equipment for subsequent processing [17].

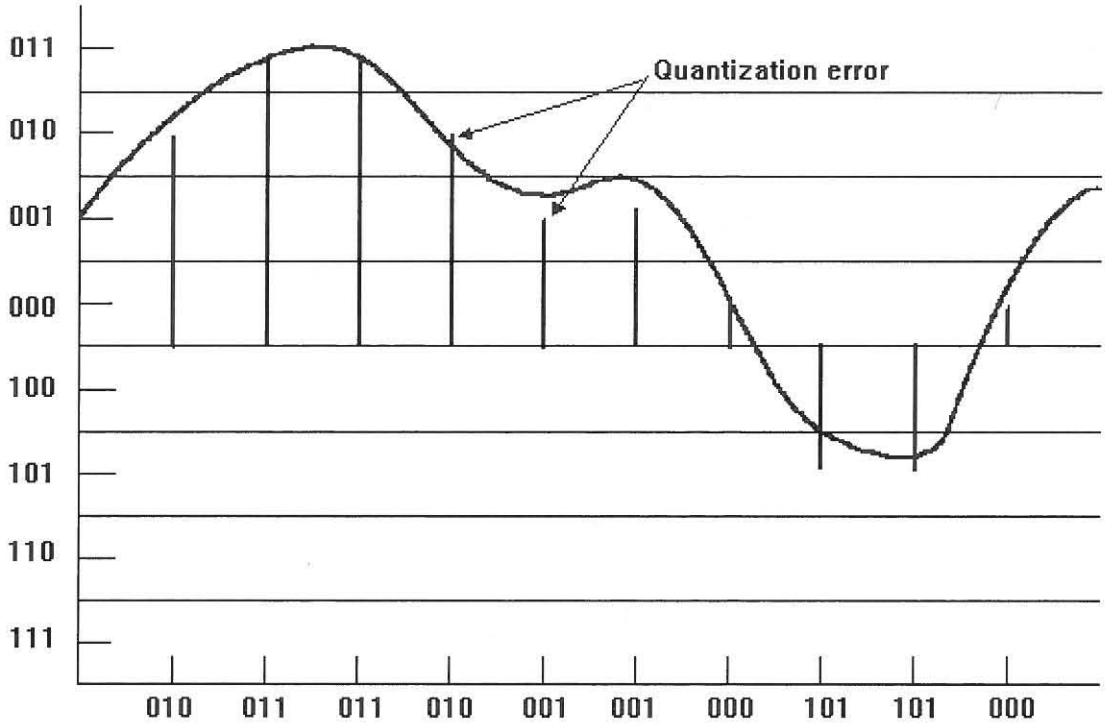


Figure 3.3 **Quantization of analog samples**

A typical quantization process, as shown in figure 3.3, depicts the manner in which a set of quantization intervals is associated in a one-to-one fashion with a binary code word. Decision values are decisive for quantizing as they form the limits between any two quantizing intervals. Amplitudes exceeding the decision value are assigned the quantizing value above the limit, while small amplitudes are assigned the quantizing value below the limit [3].

3.2.4 Quantizing distortion

As shown in figure 3.4, an error not greater than one-half of the smallest quantizing interval may result when the sample magnitude lies somewhere in between one quantizing level and another, which will be quantized to the nearest level.

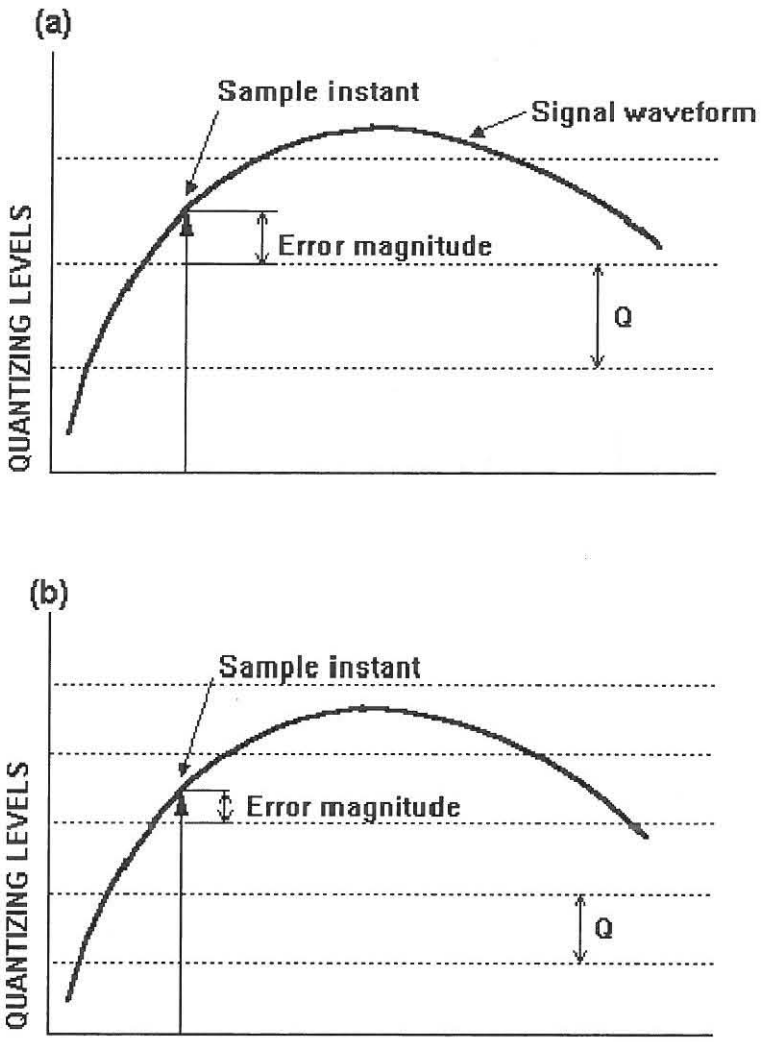


Figure 3.4 **Quantizing intervals**

At (a) a large quantizing error results from a small number of quantizing intervals, whereas at (b) a smaller error occurs from a larger number of quantizing levels.

This quantization error is called quantization distortion or quantization noise, because it results in the output of the system being slightly different from that which was presented to the input [36].

The greater the number of bits per sample, the more quantizing intervals and the smaller the chance for a potential error. Quantizing noise has properties not obviously related to the structure of the speech, and its effect is then subjectively equivalent to adding a small amount of flat spectrum random noise to the signal. If the number of digits in the binary code is small or if the input signal level exceeds the permitted coder range, the quantizing noise will have different properties and will be highly correlated with the speech signal. In this case the fidelity of the reproduction of the speech waveform will obviously be much worse, but the degradation will no longer sound like the addition of random noise. It will be more similar subjectively to the result of non-linear distortion of the analog signal. Such distortion produces many intermodulation products from the main spectral components of the speech signal, but even when extremely distorted the signal usually contains sufficient spectral features of the original signal for much of the intelligibility to be retained. Even though the information content of a speech signal is almost entirely carried by the spectrum above 300Hz, the sound pressure

waveform of the signal has a substantial proportion of its total power in the frequency range below 300 Hz [17].

3.2.5 Linear quantization

If the quantizing intervals are equally spaced over the entire amplitude range, the resulting approximation is usually referred to as linear quantization. It is called linear because the numerical equivalent of each code is proportional to the quantized sample value it represents. The characteristics clearly show that the quantizing intervals are of equal size over the entire input signal range. (Figure 3.5) On the output side, there can be an error of up to half a quantizing interval, which means that low-level inputs produce relatively large quantizing errors which may be in the same order of magnitude as the speech signal itself. This results in an undesirably small signal-to-noise ratio where the errors for small signals will be audible.

For peak-level input signal amplitudes, on the other hand, one half of a quantizing interval is only 1/256 of the amplitude and will give inaudible errors. (An 8-bit binary code permits an arrangement of $2^8 = 256$ quantized values). Linear quantization would only be optimum if a given amount of error was of equal importance at all the signal levels [28].

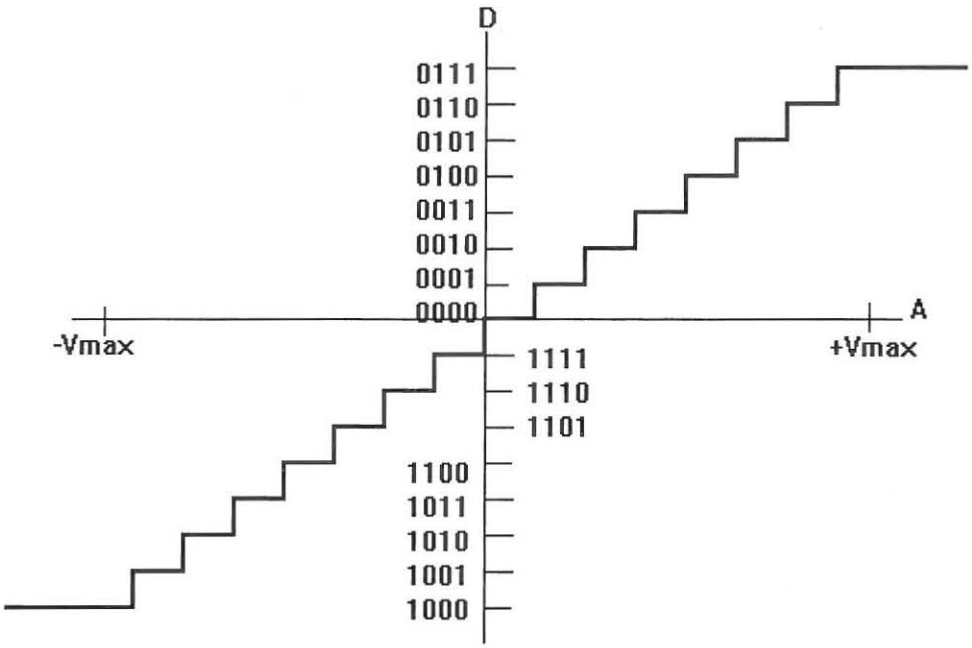


Figure 3.5 **Linear PCM Quantization**

3.2.6 Companding

A uniform PCM system provides unneeded quality for large signals which means that the system is inefficiently utilized. As has already been mentioned, more efficient coding procedure is achieved if the quantization intervals are not uniform but are allowed to increase with the sample value. When the quantization intervals are directly proportional to the sample value, signal-to-noise value is constant for all the signal levels. With this technique an adequate dynamic range for large signals and fewer bits per sample for a specified signal-to-noise

ratio are provided. A non-linear relationship exists between the code word and the samples they represent when the quantization intervals are not uniform.

Non-linear quantizing of speech signals is achieved by means of companding. As shown in figure 3.6, companding is the process of first compressing and then expanding a signal. The analog input sample is first compressed and then expanded and quantized with uniform quantization intervals. A non-uniform PCM decoder is then used to expand the compressed value, using an inverse compression characteristic to recover the original sample value.

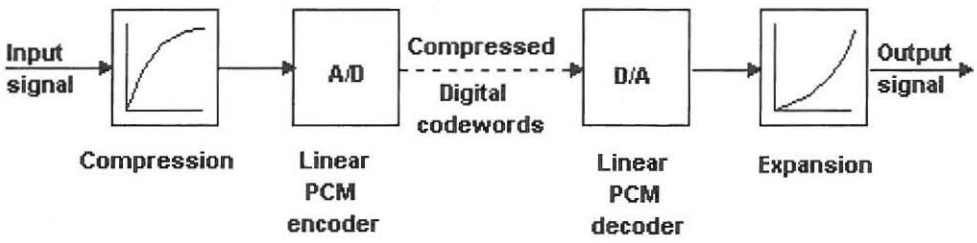


Figure 3.6 **Companded PCM with analog compression and expansion**

In figure 3.7 the effect of the compression operation is shown. Notice that quantizing intervals are distributed in such a way that closely spaced levels are used for low-input signals, and widely spaced levels for larger input signals. The larger the sample value, the more it is compressed into constant-length quantization intervals [3].

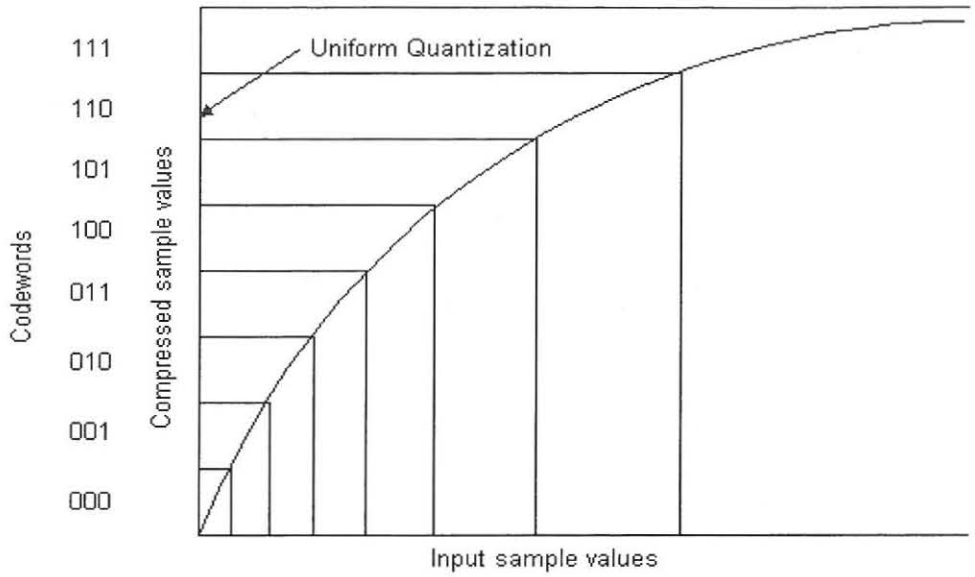


Figure 3.7 **Typical compression characteristic**

3.2.7 Pre-emphasis and De-emphasis

As quantizing noise has a flat spectrum its effect on the signal-to-noise ratio is much more serious for the weaker components, but more important for the higher-frequency components. In speech signal processing a considerable performance improvement for PCM can be obtained by taking into account this property of speech production, and by applying pre-emphasis to the speech signal with a simple linear filter to make the average spectrum more nearly flat. This is done by emphasizing the higher-frequency components roughly 6 dB/oct prior to low-pass filtering for A/D conversion. Pre-emphasis can also be accomplished after A/D conversion through differential calculation or

through application of first-order digital filtering. To maximize the signal-to-noise ratio as much as possible it is necessary that pre-emphasis be applied prior to A/D conversion.

After PCM decoding the received signal can be restored to its original spectral shape by de-emphasis. This is done by adding a tilt of 6dB/oct to reproduce the original spectral tilt, so reducing the higher-frequency components of the quantizing noise. For normal communication purposes it is not necessary that the de-emphasis should match the pre-emphasis, since speech intelligibility is actually improved by attenuating the low-frequency components, as it reduces the upward spread of auditory masking [14].

3.3 DIFFERENTIAL PULSE CODE MODULATION

A conventional PCM system encodes each sample of the input waveform independently from all other samples. Thus a PCM system is inherently capable of encoding an arbitrary waveform whose maximum frequency component does not exceed one-half the sampling rate. Analysis of a typical speech waveform, however, indicates that there is considerable redundancy from one sample to another. Differential pulse code modulation (DPCM) is designed specifically to take advantage of such sample-to-sample redundancies. Since the range of sample differences is less than the range of individual samples, the bits needed to encode

difference samples are fewer. Often the same sampling rate and band-limiting filters are used as for conventional PCM systems. The method of generating the difference samples for a DPCM coder is to store the previous input sample directly in a sample-and-hold circuit and to use an analog subtractor to measure the change. This change in the signal is then quantized and encoded.

Figure 3.8 shows a more complicated DPCM structure, because the previous input value is reconstructed by a feedback loop that integrates the encoded sample differences. The feedback signal is an estimate of the input signal as obtained by integrating the encoded sample differences, which means that the feedback signal is obtained in the same manner used to reconstruct the waveform in the decoder. By implementation of the feedback, the quantization errors are prevented from accumulating indefinitely. If there is an accumulation of quantization errors the feedback will drift from the input signal, but the next encoding of the difference signal will automatically compensates for the drift. This is a great advantage because quantization errors might accumulate without bound in a system without feedback. The analog-to-digital conversion process can be uniform or companded like in PCM systems. DPCM systems with first-order prediction typically provides a full 1-bit reduction in the code word size with respect to standard PCM encoding [3].

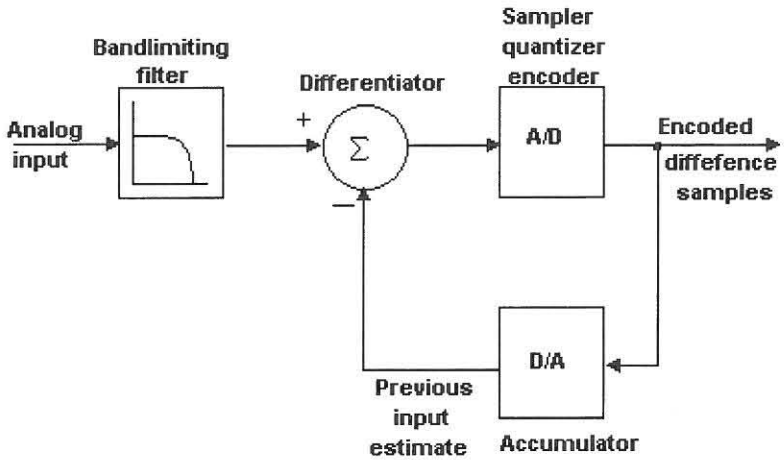
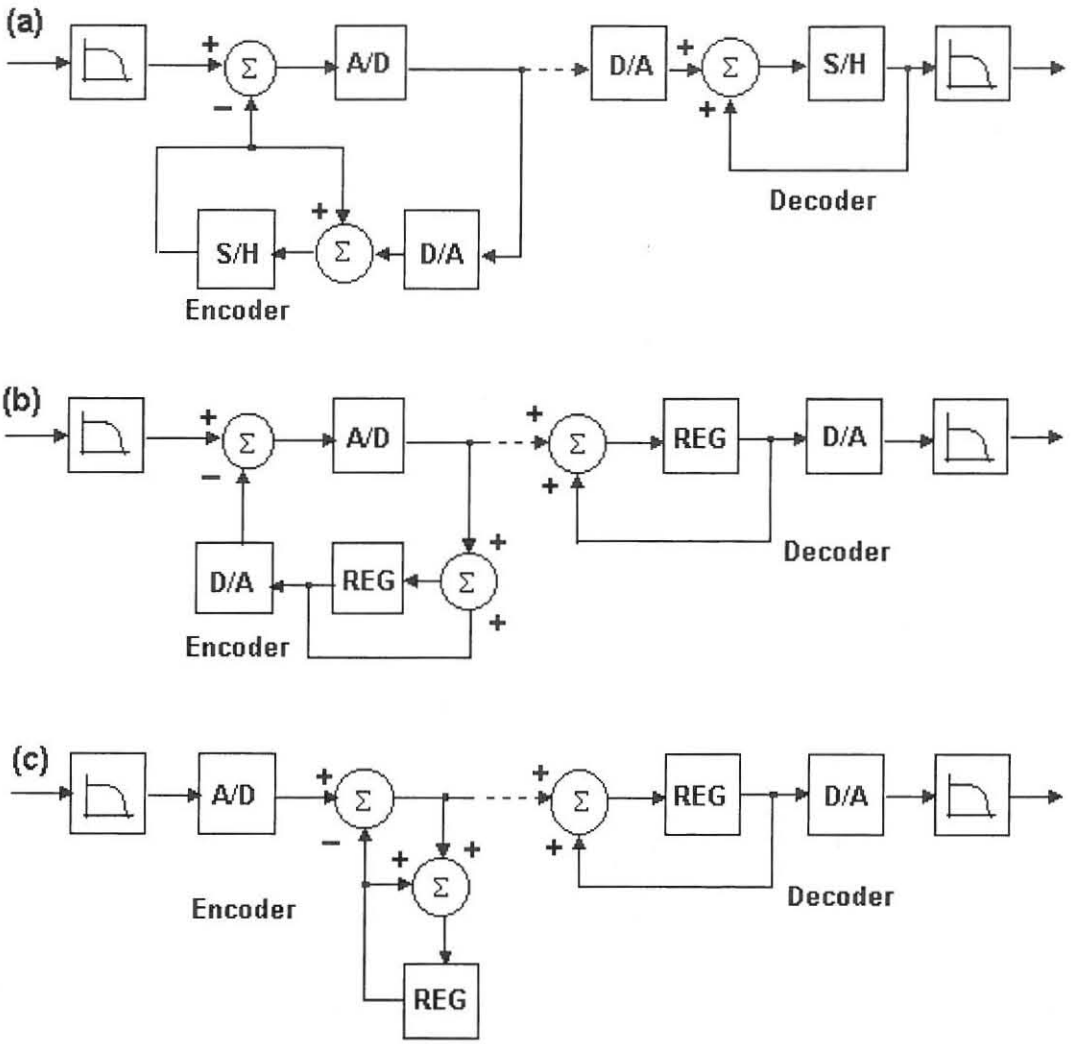


Figure 3.8 **Functional block diagram of differential PCM**

3.3.1 DPCM Implementations

Differential PCM encoders can be implemented at the one extreme with analog circuitry by using the differencing and integration functions, while at the other extreme all signal processing can be implemented digitally using conventional PCM samples as input. Block diagrams of three different implementations with differing amounts of digital signal processing are shown in figure 3.9.



- **S/H** **Sample and Hold**
- **REG** **Register**

Figure 3.9 **DPCM implementations.**
(a) Analog integration. (b) Digital integration. (c) Digital differencing.

Figure 3.9a shows a system using differencing and integration. The analog-to-digital conversion is performed on the difference signal, and conversion from digital to analog for the feedback loop is immediately performed on the limited-range difference code. Storage in a sample-and-

hold (S/H) circuit and analog summation is used to provide integration. In figure 3.9b the system performs the integration function digitally. The difference code is summed and stored in a data register to generate a digital representation of the previous input sample, whereas in figure 3.9a the difference code is immediately converted back to analog for feedback. The analog feedback for differencing in figure 3.9b is generated by full-scale D/A converters, which must provide full amplitude range conversion, whereas the D/A converters in figure 3.9a convert the more limited difference signals.

Figure 3.9c shows a system where digital logic circuits are used to perform signal processing. Digitally generated approximations of the previous amplitude code are compared to the full amplitude range sample codes produced by the A/D converter. In this case the A/D converter must encode the entire dynamic range of the input, whereas only the difference signals are used for the operation of the A/D converters in figures 3.9a and 3.9b. Digital processing, as in figure 3.9c, is generally the most effective means of implementing a DPCM algorithm. This is due to the availability of digital signal processing (DSP) components, some of which contain internal A/D converters [3].

3.3.2 Higher order prediction

In the simplest case of a linear predictor the feedback signal of a DPCM system represents first-order prediction of the next sample. The

difference between predicted and actual values is a prediction error. If the DPCM concept can be extended to incorporate more than one past sample value into the prediction circuitry, the additional redundancy available from all previous samples can be weighted and summed to produce a better estimate of the next input sample. With a better estimate, the range of the prediction error decreases to allow encoding with a compressed bit rate. Results have shown that for systems with constant predictor coefficients, the most realizable improvement occurs when only the last three sample values are used for prediction.

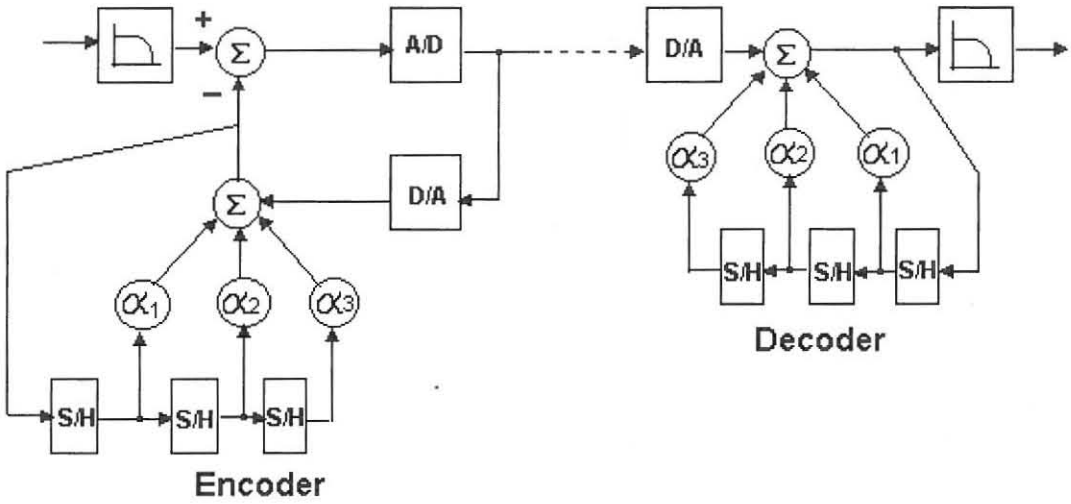


Figure 3.10 **Extension of DPCM to third-order prediction**

For conceptual purposes Figure 3.10 shows the implementation of an analog differencing and integration third-order linear predictor. In practice the use of digital memory, multiplication and addition in a DSP component could be more effective, as many applications involve

already digitized (PCM) signals. DPCM systems utilizing third-order prediction can provide reductions of up to 2 bits per sample [14].

3.4 ADAPTIVE DIFFERENTIAL PCM

By adding adaptation logic to the basic DPCM implementation the quantization step for the first-order difference is controlled to adapt to the input speech. This method is referred to as Adaptive Differential PCM or ADPCM. In spite of its simple structure ADPCM produces high quality by using backward adaptive quantization and fixed prediction methods. This method produces a Signal-to-Noise ratio of roughly 22dB at a sampling rate of 8 kHz and 4-bit quantization (32kbps), which is roughly 8dB higher than that of an ordinary telephone system. Subjective evaluation by the preference method indicates that the quality of 4-bit ADPCM is between that of 6-bit and 7-bit standard PCM. This implies savings of roughly 2.5 bits per sample. One reason for this improvement is that ADPCM can cover a wider amplitude range than standard PCM with the same bit rate. The other reason is that the power spectral distribution for the quantization noise is not homogeneous but is concentrated in the lower-frequency range. This means that the speech spectrum can easily mask the quantization noise. Figure 3.11 shows the block diagram of an advanced ADPCM system incorporating both adaptive quantization and adaptive prediction capabilities [14].

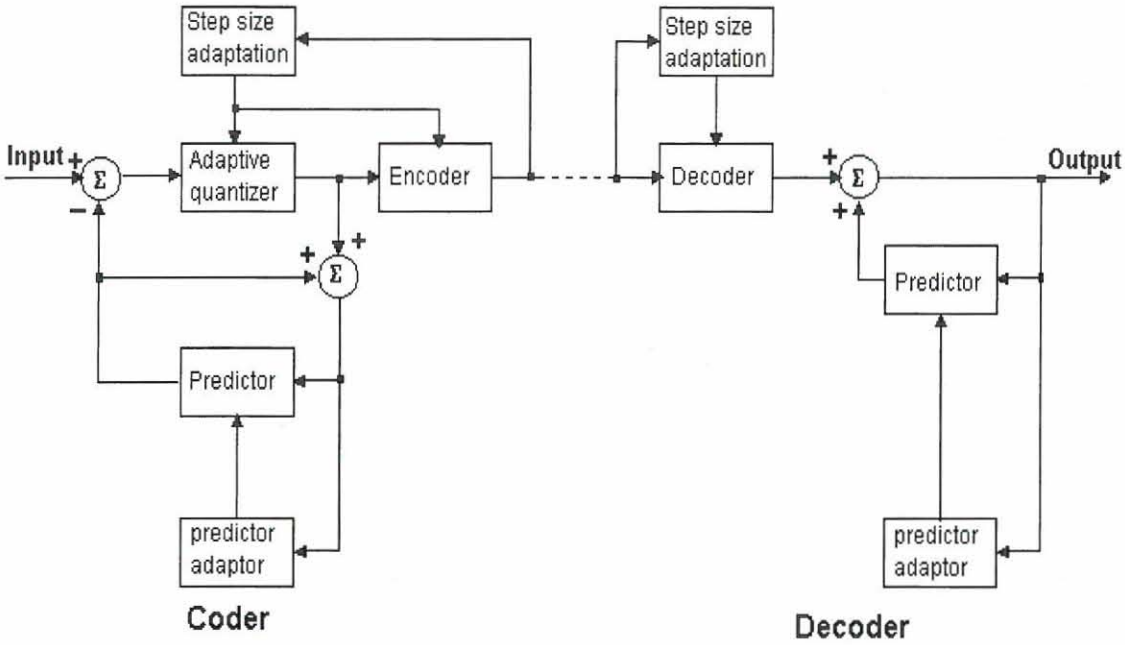


Figure 3.11 **ADPCM, including adaptive quantization and adaptive prediction.**

3.5 DELTA MODULATION

Delta modulation (DM) is another digitizing technique using an extreme method of differential quantization, in which the sampling frequency is raised so high that the difference between adjacent samples can be approximated by a 1-bit representation. A basic implementation of a DM encoder and decoder is shown in figure 3.12. DM can be considered as a special case of DPCM where a single bit specifies merely the polarity of the difference sample and thereby indicates whether the signal has increased or decreased since the last sample. An approximation of the

input signal is constructed in the feedback path by stepping up one quantization level when the difference is positive and stepping down when the difference is negative.

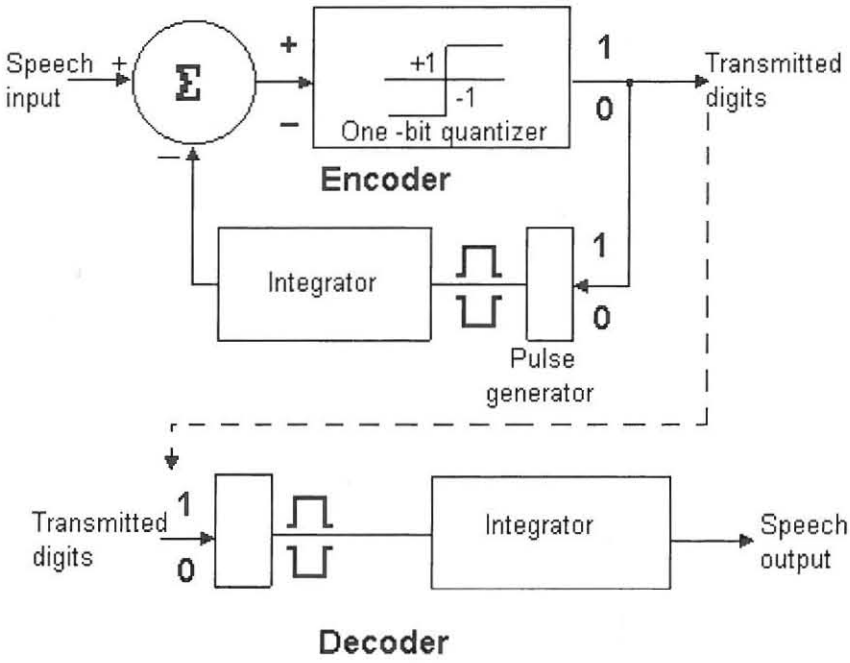


Figure 3.12 **Block diagram of a delta modulator**

A DM approximation of a typical waveform is shown in figure 3.13 where the input is encoded as a sequence of "ups" and "downs" in a manner resembling a staircase. The feedback signal continues to step in one direction until the input is crossed again. This means that when tracking the input signal, the DM output "bounces" back and forth across the input waveform, allowing the input to be accurately reconstructed by a smoothing filter. Different from normal PCM or

multibit DPCM systems, the DM system requires a sampling rate higher than the minimum Nyquist sampling rate of twice the bandwidth. Since each encoded sample contains a relatively small amount of information (1 bit), "oversampling" is needed to achieve better prediction from one sample to the next.

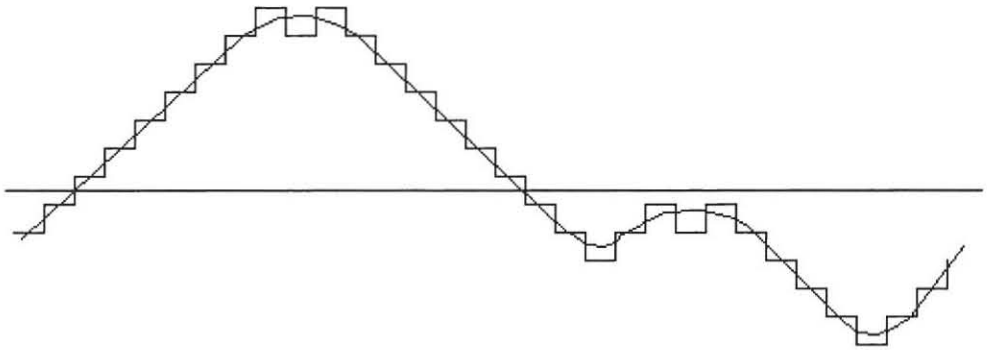


Figure 3.13 **Waveform encoding by delta modulation**

The main attraction of DM is its simplicity. Notice that the A/D conversion function is provided by a simple comparator which produces a "0" from a negative and "1" from a positive difference voltage. A two-polarity pulse generator provides the D/A function in the decoder and in the feedback path of the encoder. In the simplest form the integrator can consist of nothing more than a capacitor to accumulate the change from the pulse generator. Furthermore the delta modulator also allows the use of simple filters for bandlimiting the input and smoothing the output [17].

3.5.1 Slope overload and granular noise

Although the conceptual operation shown in figure 3.13 indicates that the encoded waveform is never much more than a step size away from the input signal, it sometimes happens that the DM may not be able to keep up with rapid changes in the input signal and thus falls more than a step size behind. When this happens the delta modulator is said to be experiencing "slope overload" as shown in figure 3.14. This condition occurs when the rate of change of the input exceeds the maximum rate of change that can be generated by the feedback loop.

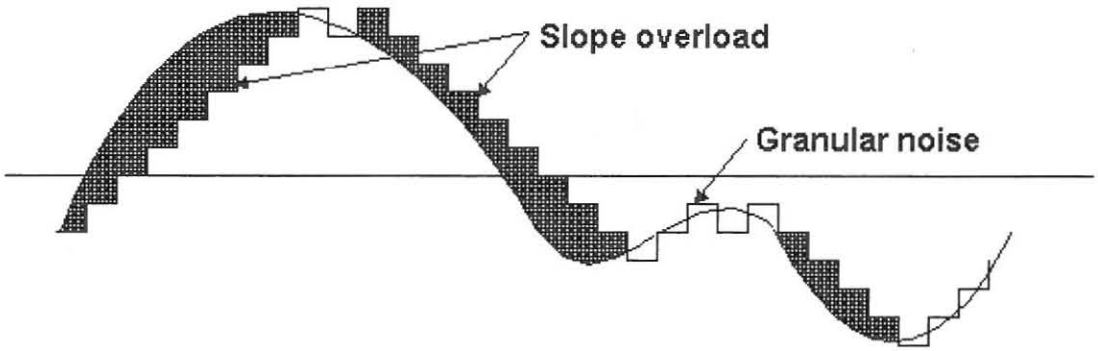


Figure 3.14 **Slope overload and granular noise of a delta modulation system**

On the other hand, when there is no input signal or when the input wave changes only slightly and very slowly, the quantization outputs alters between 0 and 1. This causes a type of distortion, which is referred to as granular noise. Slope overload distortion can only be reduced by

increasing the step size. When the step size is too large, however, the granular noise increases. Therefore the step size is usually set at a compromise value which minimizes the mean square quantization error [3].

3.5.2 Linear and adaptive delta modulation

Uniform or linear delta modulation (LDM) uses a fixed step size and is therefore the simplest delta modulator. The fundamental concerns when designing a linear delta modulator is selecting a fixed step size and sampling rate to satisfy the criteria as mentioned in paragraph 3.5.1. The signal-to-slope overload distortion ratio must be some minimum value for the highest level signal to be encoded. Secondly, the signal-to-granular noise ratio must be some minimum value for the lowest level signal to be encoded. In order to satisfy the slope overload criteria it is necessary to design the system so that slope overload is just on the verge of occurring at the highest level of input. This means that the step size for the highest level signal is not optimum in a perceptual sense or even in terms of minimizing the sum of granular noise and overload distortion [3].

A coding method, called adaptive delta modulation (ADM), is more effective in handling the above-mentioned problems. ADM uses an algorithm in which the step size is changed adaptively with respect to the input speech waveform. Basically, all the various ADM methods are

based on the backward adaptation (feedback adaptation) technique, in which the minimum step size is adjusted according to the output code sequence. Since the step size is varied exponentially in ADM, the slope overload distortion and the granular noise in ADM are smaller than those in LDM. Experimental evaluation in this regard has indicated that an ADM system with a sampling frequency of 56 kHz has almost the same quality as a 7-bit normal PCM application with 88kHz sampling [14].

3.6 SUBBAND CODING (SBC)

Subband coding makes use of the method in which the coder first divides the input spectrum into separate bands using bandpass filters. As shown in figure 3.15, the signal passing through each of the subbands is individually encoded by an adaptive coding method such as APCM at the Nyquist rate. The individual bit streams are then multiplexed for transmission to the decoder where the inverse procedures reproduce the original signal.

Separately encoding each subband is advantageous for two reasons. Firstly by making the bands as narrow as the ear's critical bands, the quantizing noise in each band can be largely masked by the speech signal in the same band.

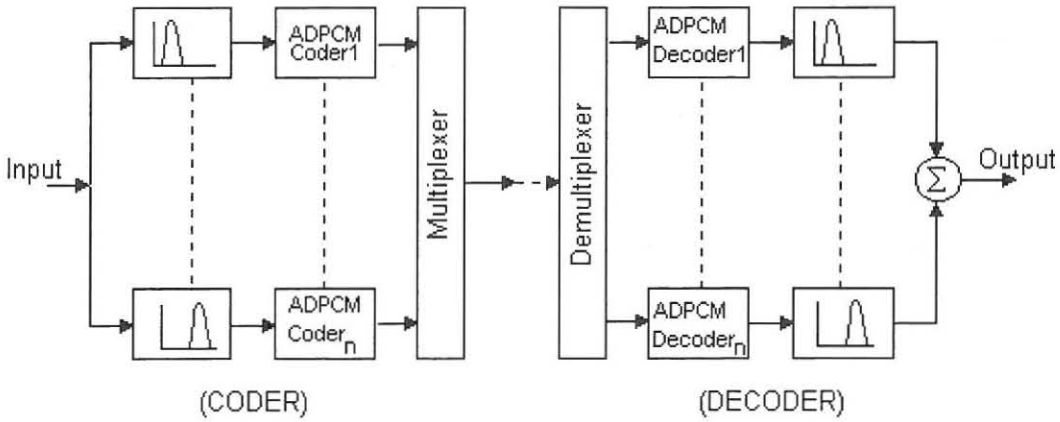


Figure 3.15 **Block diagram of SBC**

A second advantage is the fact that a higher bit rate can be allocated to those bands which are subjectively more important. In this way this method can produce less-perceptible quantizing noise at the same or a lower bit rate. In addition, this method is also beneficial in that the low-level speech input will not be corrupted by quantization noise produced in another band [14].

3.7 VOCODERS

Most of the previously described encoding and decoding algorithms have been concerned primarily with reproducing the input waveform as accurately as possible, and they have assumed little or no knowledge of

the nature of the signal they process. The basic goal of a vocoder (a contraction of voice coder) is to encode only the perceptually important aspects of speech with fewer bits than the more general waveform encoders. For this reason they are not applicable to portions of the pubic telephone network in which other signals, such as modem signals, must be accommodated. Although vocoders furthermore produce unnatural or synthetic sounding speech, they can be used in limited bandwidth applications like message recording, encrypted voice transmission over narrowband HF radio, digital cellular radio, analog telephone circuits, computer output and games. Many forms and variations of vocoders exist, but the three most basic vocoding techniques are:

- the channel vocoder;
- the formant vocoder; and
- the linear predictive coder [3].

3.7.1 The channel vocoder

The first vocoder was invented in 1939 by H. Dudley of Bell Telephone Laboratories. This vocoder uses a bandpass filter bank for spectral analysis and is now referred to as the channel vocoder [14].

The bandpass filters are used to separate the speech energy into subbands that are full-wave rectified and filtered to determine relative power levels. Each power level is encoded and transmitted to the destination. Recent advances in digital technology have introduced the use of digital signal processing to determine the input spectrum. Modern channel vocoders can also determine the nature of speech excitation (voiced or unvoiced) and the pitch frequency of the voiced sounds. Determining the pitch of voiced sounds is the most difficult aspect of most vocoder realizations. Since certain sounds are not clearly classifiable as purely voiced or purely unvoiced, accurate excitation information is furthermore a desirable extension of a basic vocoder [3].

3.7.2 Formant vocoder

The spectral envelope of a speech sound shows a small number of well-defined peaks called formants. A formant vocoder determines the location and amplitude of these spectral peaks and transmits this information instead of the entire spectrum envelope. Analysis of the spectral envelope is the main difficulty with formant vocoders. For this reason, no formant vocoders have yet been used operationally. Although computationally expensive, some of these vocoders have been successfully demonstrated in the research environment [17].

3.7.3 LPC vocoders

The linear predictive coder (LPC) is a popular vocoder which analyzes a speech waveform to produce a time-varying model of the vocal tract excitation and transfer function. The LPC extracts perceptually significant features of speech directly from a time waveform rather than from the frequency spectra, as do channel and formant vocoders. At the receiving end, a synthesizer recreates the speech by passing the specified excitation through a mathematical model of the vocal tract. The synthesizer adapts to changes by periodically updating the parameters of the model and the specification of the excitation. The vocal tract is assumed to represent a linear time-invariant process during any one specification interval. Most of the predictive coders mentioned previously base their prediction on past measurements (backward estimation). In contrast the LPC uses prediction parameters based on the actual input segments to which the parameters are applied (forward estimation). Thus, the LPC does not measure and encode difference waveforms or error signals, but the error signals are minimized in a mean-squared sense when the predictor coefficients are determined. As with other vocoders, the nature of excitation is established by determining whether strong periodic components exist in the waveform. Pitch is determined by measuring periodicity when it exists. In addition to measuring pitch with techniques similar to those used by other vocoders, an LPC encoder/analyzer has particular properties that aid in pitch determination.

The overall result is that LPC's provide more natural sounding speech than the purely frequency-domain-based vocoders [3].

3.8 CONCLUSION

This chapter has described many variations and techniques for the digitization of speech signals. When selecting an encoder / decoder for some application, it is necessary to compare the various algorithms in terms of cost, complexity and voice quality.

Although delta modulation is by far the simplest algorithm, which therefore can be implemented at the lowest cost, this advantage is insignificant, due to the high volume and competitive production of standard PCM codecs. Furthermore, considering voice quality, PCM and DPCM applications would have been the only acceptable algorithms if it were not for the fact that some applications can tolerate lower levels of quality.

The next chapter we explains the development of the PC-based telephone answering system hardware and the factors which distinguish this design from commercial equipment.

CHAPTER 4

4. THE DEVELOPMENT OF THE PC-BASED TELEPHONE ANSWERING SYSTEM HARDWARE

4.1 INTRODUCTION

Chapter 3 looked at the many variations and techniques for the digitization of speech signals. Chapter 4 explains the development and the operation of the PC-based telephone answering system hardware interface modules.

The historical development of recording systems, as described in chapter 2, showed the advantages of using modern computer storage devices instead of the conventional magnetic tape. Advances in digital audio and in computer industry have led to the development of this recording system which offers a flexibility in operation that cannot be matched with tape.

In order to distinguish this design from commercially available telephone answering and recording equipment it is necessary to produce a tapeless, cost-effective computer-based telephone answering system by not duplicating the expensive processing and memory units on the computer

interface card, and by utilizing the computer's processor and memory for system control, buffering and information storage.

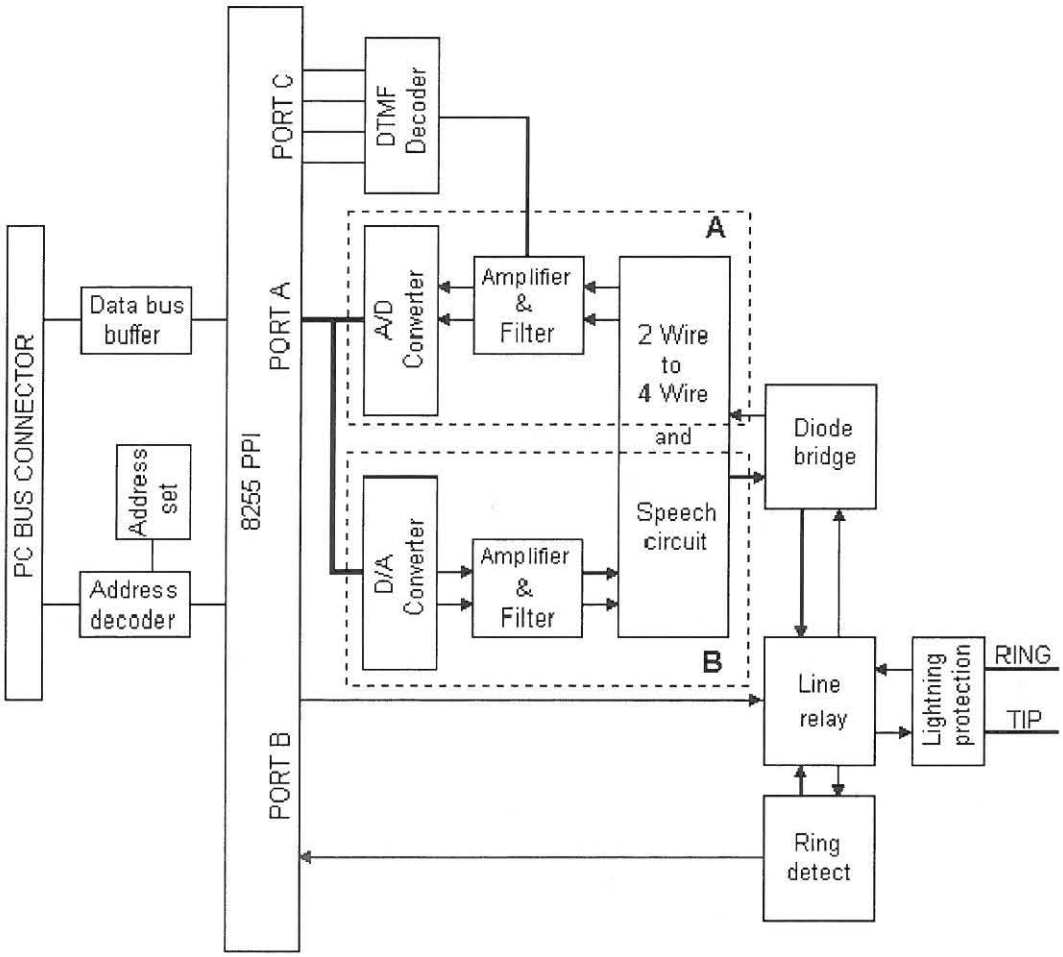
In pursuit of a cost-effective telephone answering system based on the above-mentioned system characteristics, the system adapted and evolved into a design which can still accommodate as many facilities as possible with the minimum hardware requirements.

4.2 GENERAL DESCRIPTION

The digital telephone answering system (DTAS) hardware consists mainly of analog and digital devices fitted to a printed circuit board plugged into the PC host's IBM Bus expansion slot. This DTAS interface, under control of software on the PC, makes the Man- (Analog) Machine (Digital) communication possible.

The block diagram, as shown in figure 4.1, represents the different units and associated equipment on the card. These units can be grouped together to form the following modules of the system:

- Telephone line interface (Appendix A, sheets 3&4)
- Analog-to-digital converter (Appendix A, sheet 2)
- Filters (Appendix A, sheet 2)
- Digital-to-analog converter (Appendix A, sheet 2)
- PC Bus Interface (Appendix A, sheet1)



- **A Recording Path**
- **B Play Back Path**

Figure 4.1 **Block diagram of the Digital Telephone Answering System**

It can be seen from the block diagram that the interface card consists of two main paths, namely a recording path and a playback path. The one part of the circuit carries the analog information, which is converted to digital format, acceptable for the PC. In the other part, information is carried from the PC, is converted from digital to analog format and is then applied to the telephone line.

The system operation, recording and playback modes will be further discussed in chapter 5, under System Software.

4.3 SYSTEM DESCRIPTION

4.3.1 Telephone line interface (Appendix A, sheets 3 & 4)

The line interface is a standard telephone speech circuit that complies fully with Telkom SA telephone specifications. The lightning protection consists of a primary and a secondary protection. A 350V gas arrestor (GS1), two 10 ohm series resistors (R38,R39) and a 290V dual symmetrical transient voltage suppressor (XB1) form the primary protection. The earth path for the lightning protection is connected via the metal card bracket to the chassis of the computer. An 18V transient voltage suppressor (U12) forms the secondary protection. A diode bridge

(D1-D4) protects the circuit against line polarity reversals. The line is looped or opened with relay contacts (RL1).

The line interface circuit is built around speech amplifier U11. U11 is powered from the exchange line voltage. U11 is also DC-isolated from the rest of the circuitry and the PC. U11 automatically loops the line with 20mA to 85mA, depending on the line length.

The speech amplifier performs the two wires (line) to four wires (input signal, output signal) conversion by means of a Wheatstone bridge.

The AC outgoing signal (applied at pins 1 and 16) is sent to one diagonal of the bridge (pins 6 and 9). A small percentage of the signal is lost on R5, R6 an C3 (being much bigger than the line impedance). The main part of the signal is sent to the line via R10. In receiving mode, the AC signal from the line is sensed across the second diagonal of the bridge (pins 11 and 10). After amplification it is applied to pins 12 and 13.

The speech amplifier automatically adjusts the gain of the sending and receiving amplifiers to compensate for line attenuation. The line voltage is sensed from resistive dividers R35 and R36.

R32 is a bias resistor that assures the minimum operating current for U11. R37 fixes the DC characteristics. C24 is a matching capacitor for a capacitive line.

4.3.1.1 Interface with the speech amplifier

The signal to the speech amplifier U11 is coupled with isolation transformer T1. T1 provides isolation between U11 and the rest of the circuit. It also ensures that the conversation loss is minimal. Conversion loss is caused by difference in impedance between tip and ground and between ring and ground.

The signal from the speech amplifier U11 is coupled with an isolation transformer T2. T2 provides isolation between U11 and the rest of the circuit. It also ensures that the conversion loss is minimal. The signal is further amplified by the differential amplifier U10/A.

4.3.1.2 DTMF receiver

U13 is a complete integrated DTMF receiver. The input signal is applied to pins 1 and 2, which are the inputs of an operational amplifier. R39 and R40 determine the gain of this amplifier. The oscillator of the receiver is driven by a 3,57954mhz crystal X1. Pin 15 (STD) goes high if a valid tone pair is received. There is a short delay after presenting the

signal to the input before STD goes high. This delay is needed for the validation period and the time for the output latch to settle. R43 and C29 determine the delay time that is in the order of 20ms.

DTMF dialing specifications stipulate that the digit and the interdigit pause must be longer than 65ms.

The detected tone is presented in binary code form at the data lines Q1-Q4. The function truth table is identical to the one for a DTMF generator (see table 4.1). TOE (pin 10) is permanently pulled high, which means that the data on Q1-Q4 is ready and valid when STD is high.

NUMBER	D3	D2	D1	D0	TOE	F-LOW	F-HIGH
ANY	Z	Z	Z	Z	L	-	-
1	0	0	0	1	L to H	967	1209
2	0	0	1	0	L to H	967	1336
3	0	0	1	1	L to H	967	1477
4	0	1	0	0	L to H	770	1209
5	0	1	0	1	L to H	770	1336
6	0	1	1	0	L to H	770	1477
7	0	1	1	1	L to H	852	1209
8	1	0	0	0	L to H	852	1336
9	1	0	0	1	L to H	852	1447
0	1	0	1	0	L to H	941	1209
*	1	0	1	1	L to H	941	1336
#	1	1	0	0	L to H	941	1477

Table 4.1 **Input / Output relation of U13**

4.3.1.3 Ring current detector

In "on hook" condition, the ring current detector (U14) is connected to the line via the relay contacts of RL1. U14 is an opto coupler that isolates the ring voltage from the rest of the circuit. The output (pin5) goes low for the duration of a ring burst. This output is applied to the inverter (U7B). The output of the inverter represents the occurrence of ring current which is sensed and counted by the system software.

4.3.2 Analog to Digital converter (Appendix A, sheet 2)

The ADC0804 (U6) is a low-cost encoding device. This CMOS 8-bit successive approximation A/D converter accepts an analog signal (Vi) and an analog reference (Vref) as inputs and generates a digital output (Do) which is acceptable to personal computer.

The reference voltage of the converter is set with potentiometer R16. This voltage is set to obtain a full 8-bit resolution. The device is operated in the free-running mode by connecting INTR to WR. This means that no external signals are needed to initialize each conversion cycle. When an analog-to-digital conversion is complete, output INTR will go low and is then applied to the hardware interrupt - IRQ5 on the PC Bus. The interrupt is sensed and controlled by the system software to obtain system synchronization during recording and playback periods.

The system clock for the A/D converter is obtained from the RC logic oscillator (R10, C6) and needs 72 clock pulses (64 + 8) to complete one conversion.

In order to determine the clock frequency of the RC oscillator the Nyquist criterion formula would apply, and is defined by the relation:

$$Fs > 2BW$$

where Fs = sampling frequency

BW = bandwidth of the input signal [3]

This means:

Sampling Frequency > 2 x Bandwidth

> 2 x 3400 Hz

> 6800 Hz

Then the A/D needs to be clocked at more than 6800 x 72 =

489 600 Hz

The RC oscillator is tuned by the variable resistor R10 to obtain an oscillating frequency of 500kHz, which proves to give satisfactory sampling results.

4.3.3 PC bus interface (Appendix A, sheet 1)

U1 is an 8-bit identity comparator that is used as an address bus decoder. It compares the status of the address bus with a preset address on the card. When they are equal, pin 19 (P=Q) goes low which enables data communication with the card. A switch (SW1) allows setting of the physical base I/O address of the 8255 anywhere within the range $000 hex to $3FC hex.

The output from either U1 or the NAND gate (U2A) will go low when the predefined address is selected, but it cannot be directly used to generate the chip select input for the 8255. In order to generate this signal it needs to be combined with three other signals derived from the PC bus. These are the Input/Output Read, or IOR line, the IOW line and the address enable, or AEN line. First the AEN line and the output from the address decoder are fed into a NOR gate, and the IOR and IWR lines into a NAND gate (U3B), then the output from these two gates is fed into another NAND gate (U3A) to finally produce the CS or chip select input for the 8255.

U4 is an octal bus transceiver with 3-state outputs. DIR (pin 1) determines the transmit/receive mode of U4 . A high on DIR enables the data flow from the PC bus to U5 (8255). A low on DIR enables the data

flow from U5 to the PC bus. G (pin 19) enables U4. A low on G enables data flow and a high isolates U4 from the data bus of the PC.

U5 is a mapped programmable peripheral interface. This port expander can be programmed to multiplex the data bus to 3 separate ports that can be set up as read or write latches. These ports are eight-bit I/O registers and the fourth is a control register. A control word must be programmed to set up the ports before they can be addressed.

System in recording mode:

Port	Status	Address
A	Read	Base address + 0H
B	Read	Base address + 1H
C	Read	Base address + 2H
Control word	099H	Base address + 3H

System in playback mode:

Port	Status	Address
A	Write	Base address + 0H
B	Read	Base address + 1H
C	Read	Base address + 2H
Control word	089H	Base address + 3H

The control register determines how the 8255 functions. It determines the mode in which the chip will operate, which in turn determines whether a particular I/O line functions is an input or an output line.

The 8255 is directly connected to the IOR, IWR and Reset line inputs from the PC bus, as well as address lines A0 and A1 and the eight data lines D0 through D7.

4.3.4 Digital-to-Analog converter (Appendix A, sheet 2)

U7 is an 8-bit resolution (255-step), high conversion speed (1μs) digital-to-analog device used to convert the digital signal coming from the PC back into an analog format. The digital input is latched, so that updating from the PC bus can be handled. The reference voltage (Vref, pin 5) is set with potentiometer R13. The analog output is passed through a low-pass filter (U9C). This ensures that the sample frequency and its harmonics are suppressed during the reproduction of the analog signals.

4.4 SUMMARY

The purpose of the hardware design was to produce cost-effective computer-based hardware interface. The design proved that it is possible to build such a software controlled interface by not duplicating expensive processing and memory units. In this chapter we also discussed the

hardware associated with the different modules, and the operation of the various circuits was described in conjunction with the circuit diagrams included in appendix A. In the next chapter the development of the software for the PC-based telephone answering system will be addressed.

CHAPTER 5

5. DEVELOPMENT OF THE SOFTWARE FOR THE PC-BASED TELEPHONE ANSWERING SYSTEM

5.1 INTRODUCTION

In the pevious chapter we discussed the hardware design and the operation of the various circuits of the system. Chapter 5 discusses the development of the system software and gives an explanation of the key procedures in the system control.

Humans are able to think about more than one thing at a time, and in accomplishing some piece of work they frequently interrupt their current train of thought to pursue some other related piece of work. A personal computer system which forces the user to progress in a certain order through all of the tasks needed to achieve some objective, from beginning to end without any diversions, does not correspond to that standard working pattern. If the personal computer is to be an effective dialogue partner, it must be as flexible in its ability to "change the topic" as the human is.

Computers for the most part react to stimuli provided by the user, so they are quite amenable to a wandering dialogue which is initiated by the

user. It is therefore necessary for the computer dialogue partner to present the context of each thread of dialogue so that the user can distinguish between them.

With these objectives in mind, the main aims of the system software could, therefore, be listed as follows:

- To separate physically the presentation of the different logical threads of user-computer conversation on the display device.

- To ensure natural and effective interaction between the caller and the system.

- To utilize the intelligence of the personal computer as far as possible in the software.

.

5.2 SOFTWARE CONTROL FUNCTIONS

All the system software as described in this chapter is written by using Turbo Pascal version 6.0.

The system software comprises various modules which are responsible for control functions. The following are brief descriptions of the main software control functions:

5.2.1 Display interface control

When the system is operational and in standby mode, a title bar is displayed at the top of the screen, identifying the answering system's name and version for the user. A "command" bar at the bottom of the screen allows the user to either enter the system for administration purposes or to exit the system. Between these two bars lies an "activity" field, indicating the successful initialization and loading of data on the system. This area also serves as an information field, updated with the name, date and time, relevant to incoming calls answered by the system.

When the system is entered, an access check is performed whereby the user is prompted for his/her user name and password. If successful the user progresses to the main menu which is presented within a window. To achieve dialogue partitioning on the display device, the software adopts "windowing" as a mechanism to direct the user when switching from one option to another. The rest of the main menu, and other menus accessible from this window, will be discussed in the pages to follow.

5.2.2 Telephone line interface control

When the system is in standby mode the main program continuously monitors the telephone line for an incoming call. If an incoming call is present the ring detect bit on port C of the PPI indicates the presence of ring current. The system then counts the ring bursts until these correlate with the amount entered in the system data and operates the line relay on the line interface which extends the telephone line to the speech amplifier connected to the analog-to-digital and digital-to-analog circuitry. The system then plays the "Greeting.msg" message to the caller. At the same time the system also checks for a "#" entry, which would indicate a remote message inquiry, otherwise it initiates the line recording of the message.

Interaction between the caller and the system is performed under software control, which also translates the dual-tone multifrequency (DTMF) signalling tones generated from the caller's telephone keypad [46]. The software then uses this information for decision-making when caller messaging is in progress and when remote telephone access to messages is required. At the end of message recording and after remote message enquiry the telephone line interface control procedures restore the line relay and repair the system for the next incoming call. The relevant telephone line control procedures will be explained in the pages to follow.

5.2.3 Message recording and playback control

As has already been mentioned, incoming messages are registered on the "activity" screen, but are not accessible from that screen. Recorded messages are accessible from the "Mail Menu", with the "Play Mail Messages" option.

Since speech is "continuous" by nature, it does not lend itself to being played in segmented form, as the hearing process is highly sensitive and cannot be fooled. This is in contrast with a computer system, which transfers information to and from a disk in bursts. Therefore, to ensure continuous recording and playback control, the sampled and retrieved information is stored in a RAM buffer on the host computer. Under software control the process of reading out from the buffer begins before the file has been read completely into the buffer, otherwise (a) there will be unacceptable delay between asking for the file and getting it, and (b) the size of the buffer would have to be great enough to hold the largest message file entirely. The factor which controls reading to and writing from the buffer is the state of emptiness of the buffer, and it is this that the software control monitors to determine whether any action needs to be taken to prevent the buffer from either overflowing or emptying. The RAM buffer area (32k-byte) is reserved (set aside) at the initialization of the system.

5.2.4 File saving and retrieval control

Each time a message is in the process of being recorded the software creates a unique filename for the file, which is then stored on disk. When a system message needs to be recorded, this is done by entering the "Mail Menu", where the user obtains access to the "Record System Messages" option. Messages on disk are stored and labeled in such a way that the control will only recall and list the messages relevant to the user name and password. System messages like "greeting.msg" can only be recorded and are only accessible by the administrator of the system. After the system recording has been made the administrator is prompted to rename the file in order to make it recognizable as a system file, e.g.: "Greeting.msg".

If the user needs to listen to a message, the message has to be retrieved from disk. This can be done locally or remotely by telephone. The mail data, containing the user name and filename, are kept separate in a mail record on disk. Local retrieval is done by entering the "Play Mail Messages" option, and the user obtains a list of relevant messages available on disk. The user selects the required message by moving the cursor down on the list and entering the required message. The file will then be partially transferred to the buffer, as mentioned in paragraph 5.2.3, from where it will be converted from digital to analog format to make it audible. Remote retrieval of messages is done under DTMF tone control from the telephone keypad.

5.2.5 System setup control

Although the software controls various background setup functions, like the configuration of the interface ports, there are a few parameters that the user, or rather the system administrator, may change. From the "User Menu", the user data can be changed. The administrator is allowed to add users by entering their "UserNames", "UserID" and "Password". Users can also be deleted from the system in the same manner.

From the "Setup Menu" the administrator is allowed to change the "Admin_Password", "Ring_Count" which indicates the amount of ring bursts allowed before answering, and "Recording_Time", indicating to the software the maximum recording time allowed for a message.

5.2.6 User support control

The user support software, better known as the help system, is designed integrally with the rest of the system. The help function forms part of each menu and may be accessed from any window in the system. This makes the use of the help function much more acceptable than a manual. The help data is stored in a single text file on disk, from where it is displayed when required.

5.3 SYSTEM SOFTWARE EXPLANATION

The digital telephone answering machine system software has been written in Borland Turbo Pascal version 6.

The key software procedures is found in Appendix B1 and B2, which includes the source code for the system, consisting of the following:

B1 - MMI_MAIN.PAS : Man-Machine Interface Main Program

B2 - MMI_CORE.PAS : Man-Machine Interface Core Module

Appendix B3 to B7 forms the support modules of the system. The source code of the these modules are provided on a diskette at the back of the paper and consists of the following:

B3 - MMI_MENU.PAS : Man-Machine Interface Menu Module

B4 - MMI_LBOX.PAS : Man-Machine Interface ListBox Module

B5 - MMI ERROR.PAS : Man-Machine Interface Error Module

B6 - MMITOOLS.PAS : Man-Machine Interface Tools Module

B7 - HELPMENU : Text for Help Module

The following is a description of the procedures and functions from the main program and core module. By describing the main program and core module, the relevant contents of the remaining modules will also be addressed simultaneously.

5.3.1 Program MMI_MAIN (Appendix B1)

MMI_MAIN is the main program which sets up the "activity" screen, and controls the menus and windows, as well as the access to the system. It calls for the initialization and closure of the mail, user and system data. Furthermore, when the system is in standby mode, MMI_MAIN continuously monitors the telephone line for an incoming call.

The main module consists of the following procedures:

MyExitProc: Should an error occur in the system, this procedure will get the error description from the MMI ERROR module and display it. This procedure is also responsible for the "cleaning up" of the user, mail and system data when the system is in exit.

UserData: This procedure contain the menu options for the construction of the user data window. A "popup" window is constructed for this purpose, which is called from the MMI_MENU module.

MailData: This procedure contains the dimensioning and menu options for the construction of the mail data window. A "popup" window is constructed for this purpose, which is called from the MMI_MENU module.

SysSetup: This procedure contains the dimensioning and menu options for the construction of the mail data window. A "popup" window is constructed for this purpose, which is called from the MMI_MENU module.

MainMenu: This procedure contains the dimensioning and menu options for the construction of the main menu window. A "popup" window is constructed for this purpose, which is called from the MMI_MENU module.

PasswordCheck: This procedure checks the validity of the user name and password by comparing them to the data in the user record. A "popup" window is constructed for this purpose, which is called from the MMI_MENU module.

5.3.2 Module MMI_CORE (Appendix B2)

The MMI_CORE module is probably the most important unit, containing the software liable for speech recording and playback. The software in this module also controls the telephone line interface, PC host interface, conversion of data, storage and retrieval of data, and software timing of the system.

The core module consists of the following procedures and functions:

ClearBuffer: This procedure sets the buffer pointer to "0" and fills the complete buffer with zeros to prepare a clean RAM-buffer, which acts as a temporary data store

IntSample: This procedure is an interrupt service routine, called continuously at the sampling rate, to provide a constant control time base in the system. Depending on the system status, whether "record" or "play", this procedure allows for a sample to be sent from D/A converter to buffer or to be sent from buffer to A/D converter respectively. In order to put this procedure into perspective, an explanation of this subroutine will be given later in this chapter.

Enable_IRQx: This procedure gets the applicable interrupt mask register from the 8259A programmable interrupt controller (PIC) and clears the mask bit of the interrupt service routine.

Disable_IRQx: This procedure gets the applicable interrupt mask register from the 8259A programmable interrupt controller (PIC) and sets the mask bit of the interrupt service routine.

IntDisable: This procedure disables the new interrupt service routine and sets the system interrupt vector back to point to its original handler.

IntEnable: This procedure overwrites the system vector for the interrupt in question to point to a new service routine.

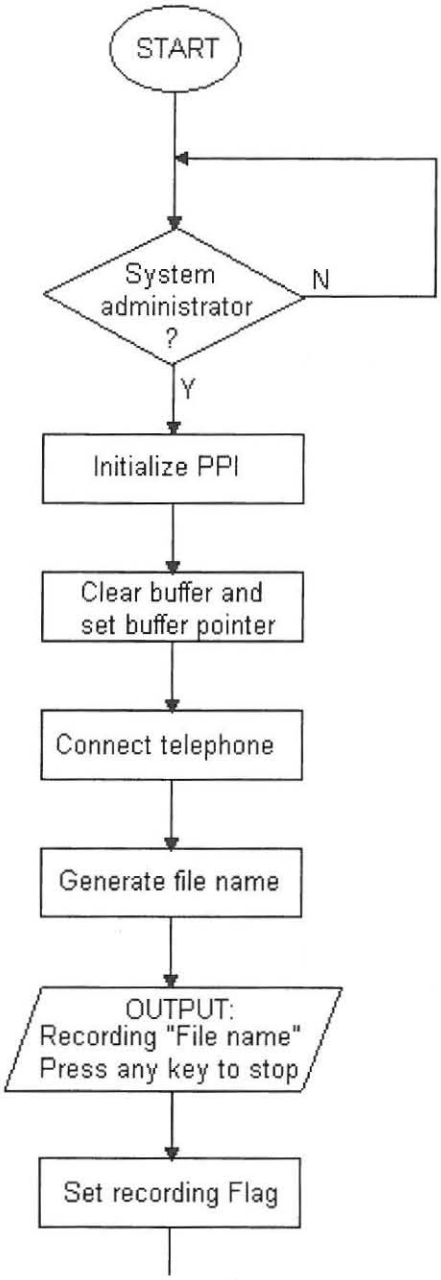


Figure 5.1 **"MakeSystRec" Procedure** (Continued over page)

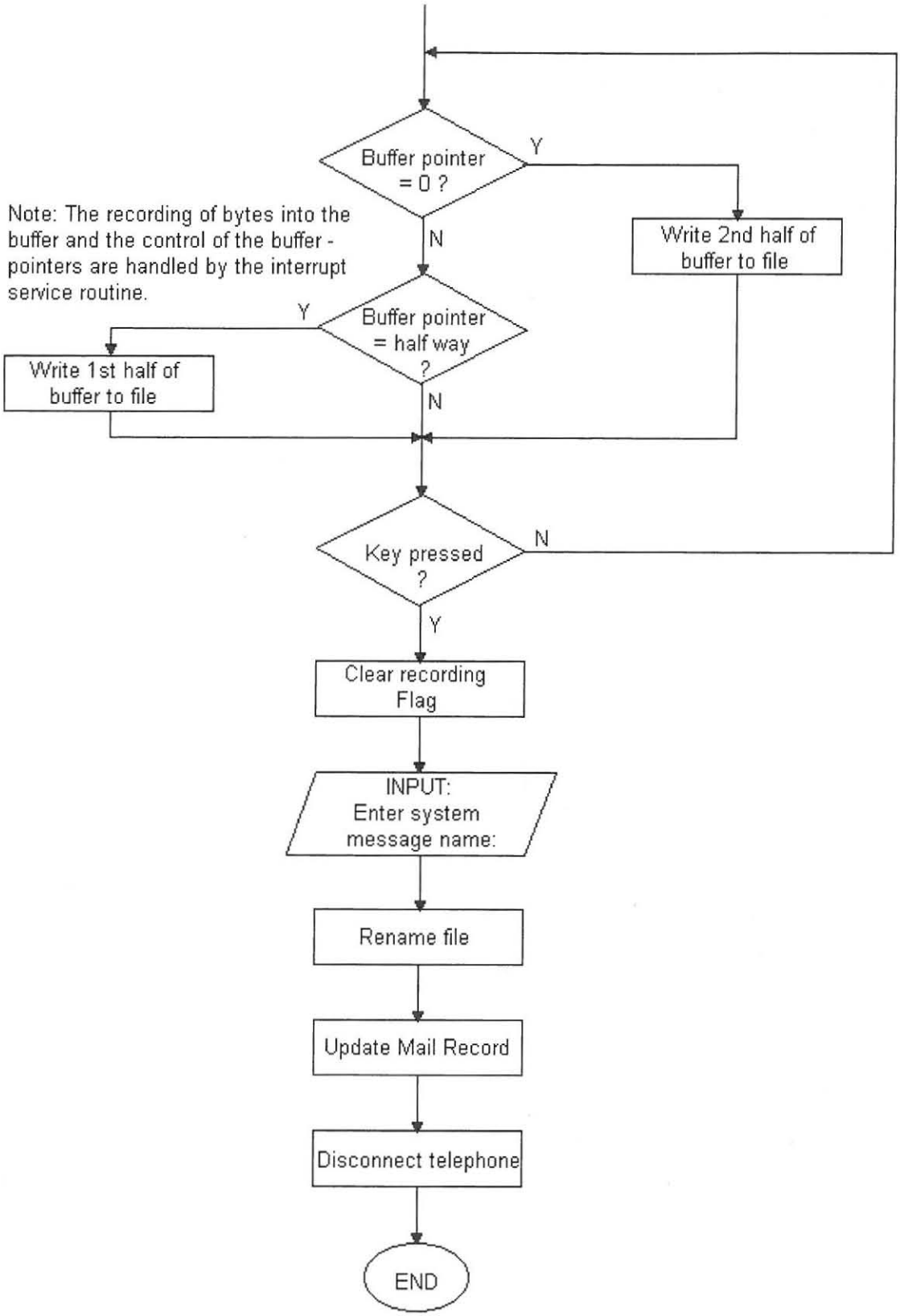


Figure 5.1 **"MakeSystRec" Procedure**

MakeSystRec: This procedure verifies the access to the system, initializes the 8255 programmable peripheral interface (PPI) and sets port-A on the PPI to input. It calls for the generation of a random file name, to which the recorded data is to be saved. Furthermore, it controls the buffer pointers and the transfer of the buffer information to disk. Since this procedure handles the system recordings, it also controls the display output, which prompts the user to rename the file. A "popup" window is constructed for this purpose, which is called from the MMI_MENU module. This procedure also provides error handling, should an error occur during recording. Figure 5.1 represents a flowchart of the "MakeSystRec" procedure.

MakeLineRec: The functionality of this procedure is very similar to "MakeSystRec". It also initializes the PPI, sets port-A to input, generates a random file name and controls the buffers. The difference is, since this is a telephone line recording, it registers who the message is for and also registers the date and time of the message. The user name and all the relevant information is then listed on the activity screen for the user's attention. Figure 5.2 represents a flowchart of the "MakeLineRec" procedure.

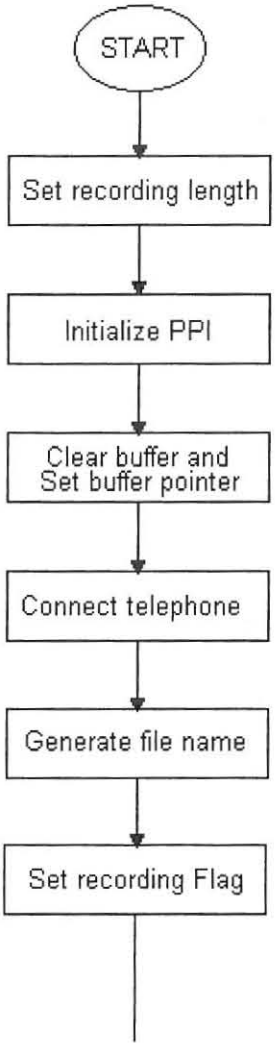


Figure 5.2 **"MakeLineRec" Procedure** (Continued over page)

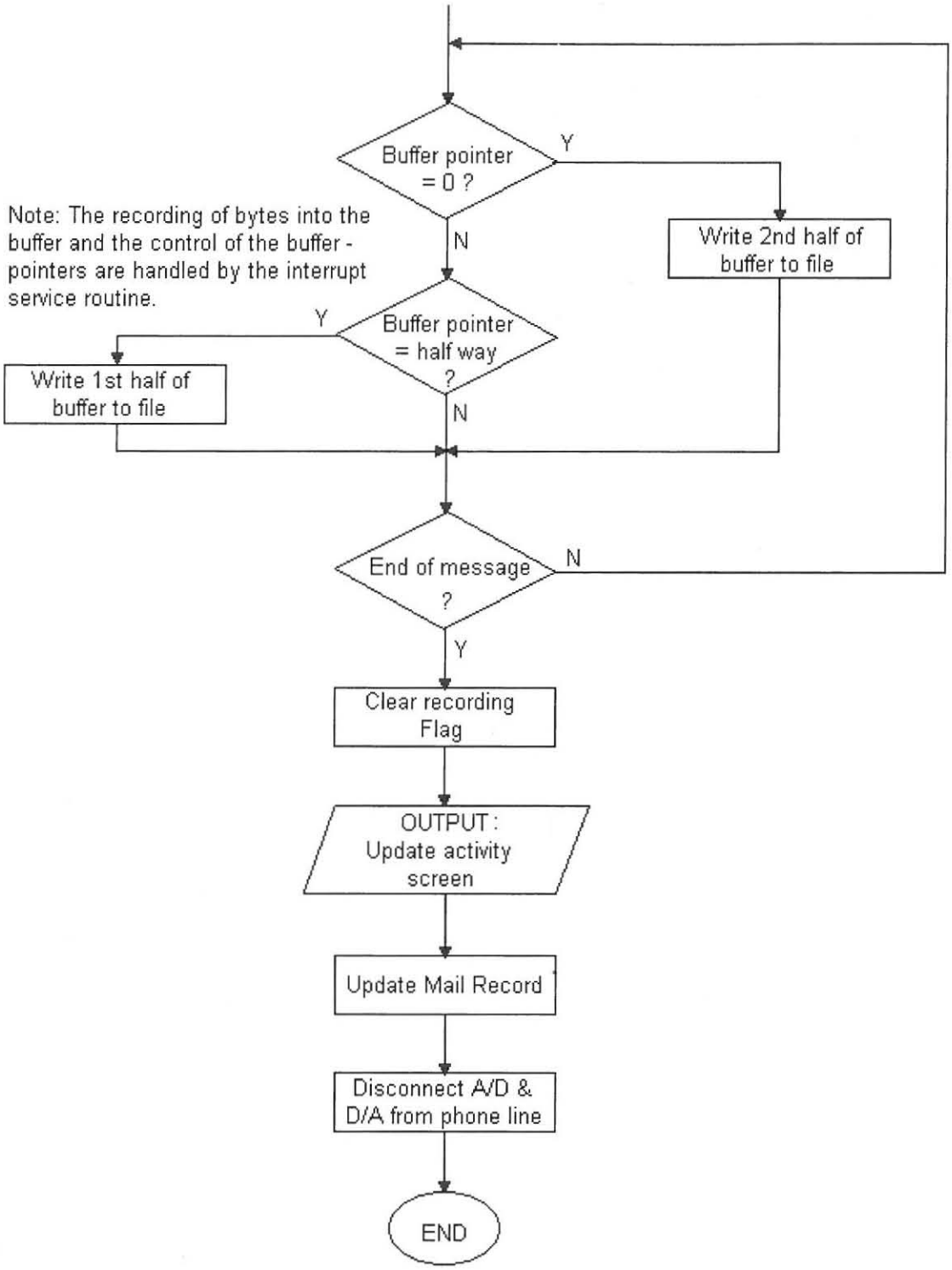


Figure 5.2 **"MakeLineRec" Procedure**

PlaySystRec: This procedure sets port-A of the PPI for output to the D/A converter. It assigns the "PlayFile" and transfers it in blocks of data to the buffer from where it is applied to the D/A converter. It displays the file name while being played. A "popup" window is constructed for this purpose, which is called from the MMI_MENU module. This procedure also provides error handling, should an error occur during playback. Figure 5.3 represents a flowchart of the "PlaySystRec" procedure.

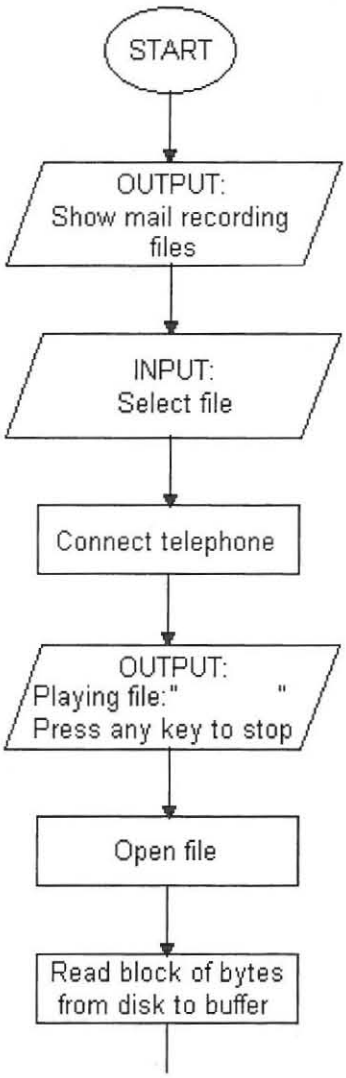


Figure 5.3 **"PlaySysRec" Procedure** (Continued over page)

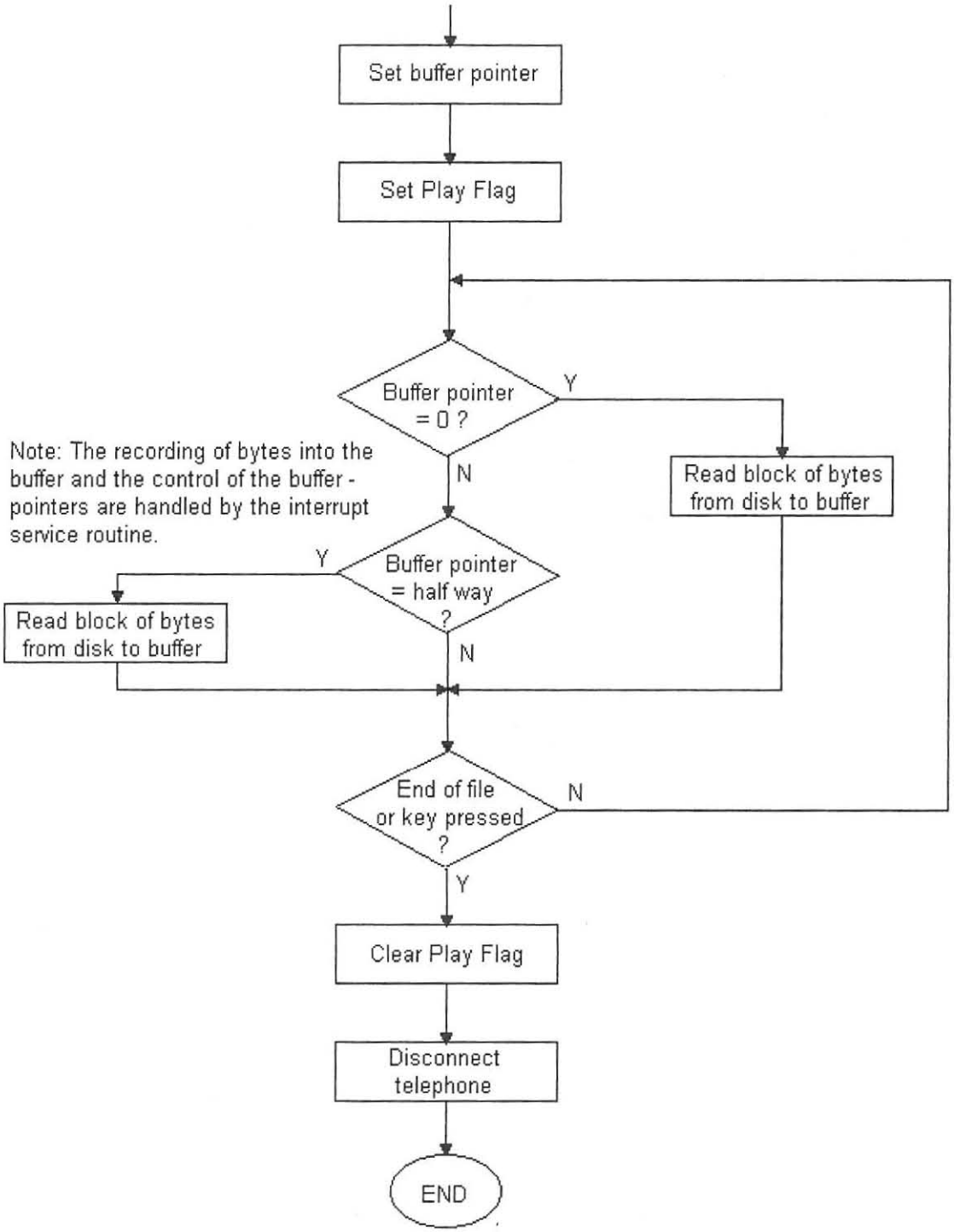


Figure 5.3 **"PlaySysRec" Procedure**

PlayLineRec: This is a function with an operation very similar to the "PlaySystRec" procedure. It also sets port-A of the PPI for output to the D/A converter, assigns the "PlayFile", and transfers the data in blocks from disk to the buffer. Although this function does not need to update to the "activity screen", it does monitor the DTMF decoder. If a "#" enters the decoder this function returns the character which stopped the play state. This is required for remote retrieval of messages. Figure 5.4 represents a flowchart of the "PlayLineRec" procedure.

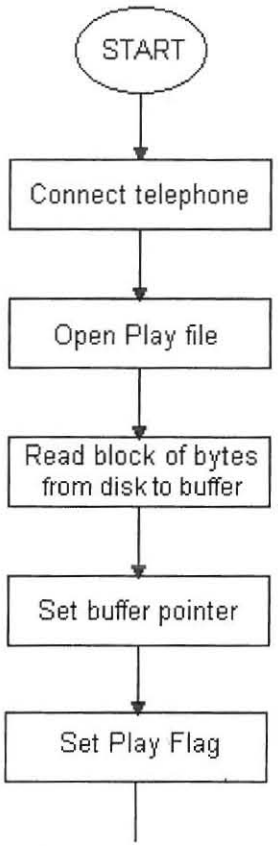


Figure 5.4 **"PlayLineRec" Procedure** (Continued over page)

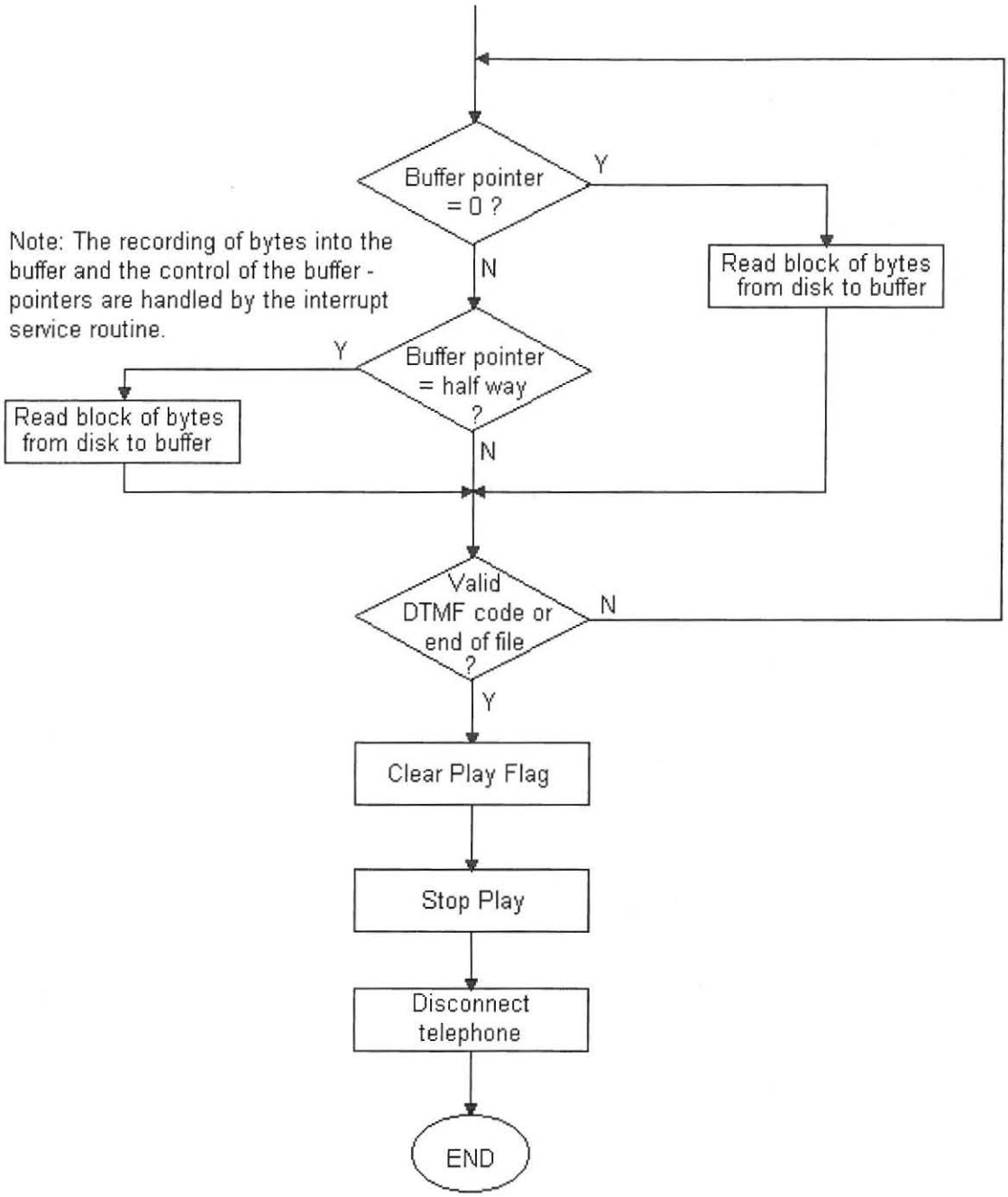


Figure 5.4 **"PlayLineRec" Procedure**

SetBit: This procedure sets the required bit of port-B on the PPI.

TestB: This function determines the status of port-B on the PPI and returns the Boolean value.

TestC: This function determines the status of port-C on the PPI and returns the Boolean value.

Timer: This procedure, together with the timer counters in the interrupt procedure, controls a 100-millisecond timer. If a parameter of, say, 10 is passed to this procedure it will calculate to: 100ms x 10 = 1000ms = 1second.

DTMF: This is a function responsible for sensing a bit on port-C for an indication of a DTMF signal. If a DTMF signal is present the code is converted to the hexadecimal format of its ASCII value and is stored in an array.

LoadMailDatabases: This procedure opens the "MailDataFile"and reads the data into an array. It also does file handling, should an error occur.

LoadUserDatabases: This procedure opens the "UserDataFile"and reads the data into an array. It also does file handling, should an error occur.

LoadSystDatabases: This procedure opens the "SystDataFile"and reads the data into an array. It also does file handling, should an error occur.

SaveMailRecord: This procedure saves the mail information by adding it to the "MailDataFile"on disk.

SaveUserRecord: This procedure saves the user information by adding it to the "UserDataFile"on disk.

SaveSystRecord: This procedure saves the system information by overwriting the "SysFile"on disk.

ShowMailRecords: This procedure is used by the "MailData" procedure in the main program. The "ShowMailRecords" procedure also uses a "PopUp" window, and by calling the "Init_lBox" procedure from the MMI_LBOX module, a list box is constructed in which the relevant mail records or files are displayed. By using the "Do_lBox" function from the same procedure, a file can be selected. This procedure then uses the "PlaySystRec" procedure from the core module, as described earlier, to play the message.

ShowUserRecords: This procedure is used by the "UserData" procedure in the main program. The "ShowUserRecords" procedure also uses a "PopUp" window, but no list box is required, since the user information is obtained from the "UserData" array.

ShowSystRecords: This procedure is used by the "SystData" procedure in the main program. The "ShowSystRecords" procedure also uses a "PopUp" window, in which the system information is displayed.

DelMailRecord: This procedure is used by the "DelMail" procedure which will be described later. The DelMailRecord procedure takes the file name passed to it and removes it from the disk. It also updates the mail data file and array with the latest information.

DelUserRecord: This procedure is used by the "DelUser" procedure which will be described later. The DelUserRecord procedure takes the user name passed to it and removes the relevant user information. It also updates the user data file and array with the latest information.

InitialiseMail: This function is used by the main program. It reserves RAM for the mail data and opens a "MailData.dat" file if it does not already exist. It also loads the "Maildata" file information into RAM and returns a Boolean value.

InitialiseUser: This function is used by the main program. It reserves RAM for the user data and opens a "UserData.dat" file if it does not already exist. It also loads the "Userdata" file information into RAM and returns a Boolean value.

InitialiseSyst: This function is used by the main program. It reserves RAM for the system data and opens a "SystData.dat" file if it does not already exist. It also loads the "SystData" file information into RAM and returns a Boolean value.

CleanupUser: This procedure gives the reserved user RAM back to DOS.

CleanupMail: This procedure gives the reserved mail RAM back to DOS.

CleanupSyst: This procedure gives the reserved user system RAM back to DOS.

AddUser: This procedure handles the "popup" window and the display dialogue when entering user information. It also uses the "SaveUserRecord" procedure, as earlier described, to store the new information.

EntrSetup: This procedure handles the "popup" window and the display dialogue when entering user information. It also uses the "SaveSystRecord", as earlier described, to store the new information.

DelMail: As mentioned earlier, this procedure uses the "DelMailRecord" procedure. The "DelMail" procedure handles the popup window, the display dialogue and the selection of the file to be deleted.

DelUser: As mentioned earlier, this procedure uses the "DelUserRecord" procedure. The "DelUser" procedure handles the popup window, the display dialogue and the entry of information of the user to be deleted.

RandomFileName: This is a function responsible for the generation of a random file name, used in the "MakeLineRec" and "MakeSystRec" procedures.

ShutDown: This procedure is used by the main program to give the memory, allocated to the mail and user data, back to DOS.

PlayMessages: This procedure registers the characters used for remote access to the messages on the system, checks validity of the password entered, extracts the relevant messages linked to the user name and starts playing them. Furthermore, the procedure checks for the appropriate entry of character, to "skip" some of the messages. A "beep" separates the messages from each other, and after the last message this procedure calls for the play of the "NoMore.msg" file as a

no-more- messages indication. Figure 5.5 represents a flowchart of the "PlayMessages" procedure.

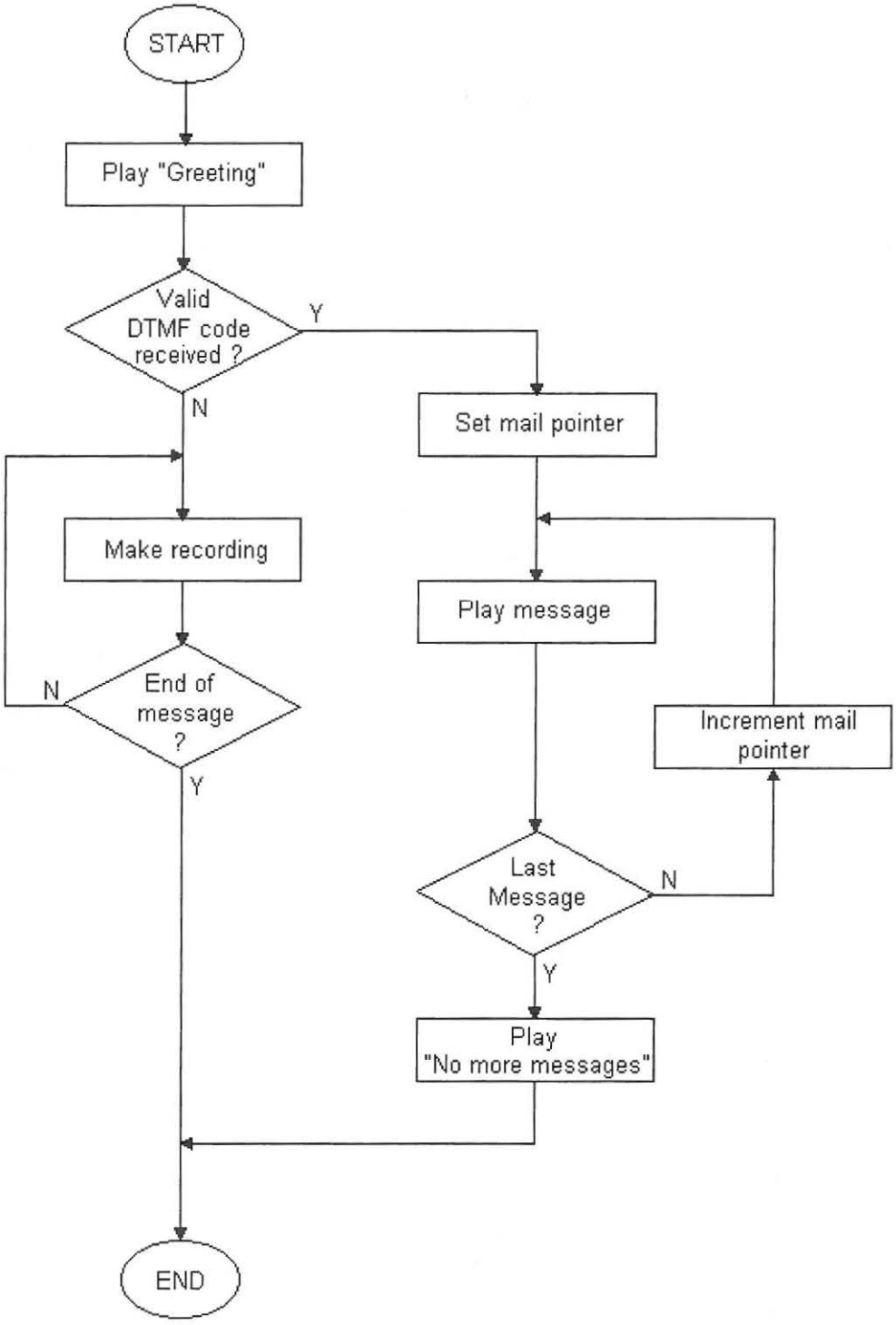


Figure 5.5 **"PlayMessages" Procedure**

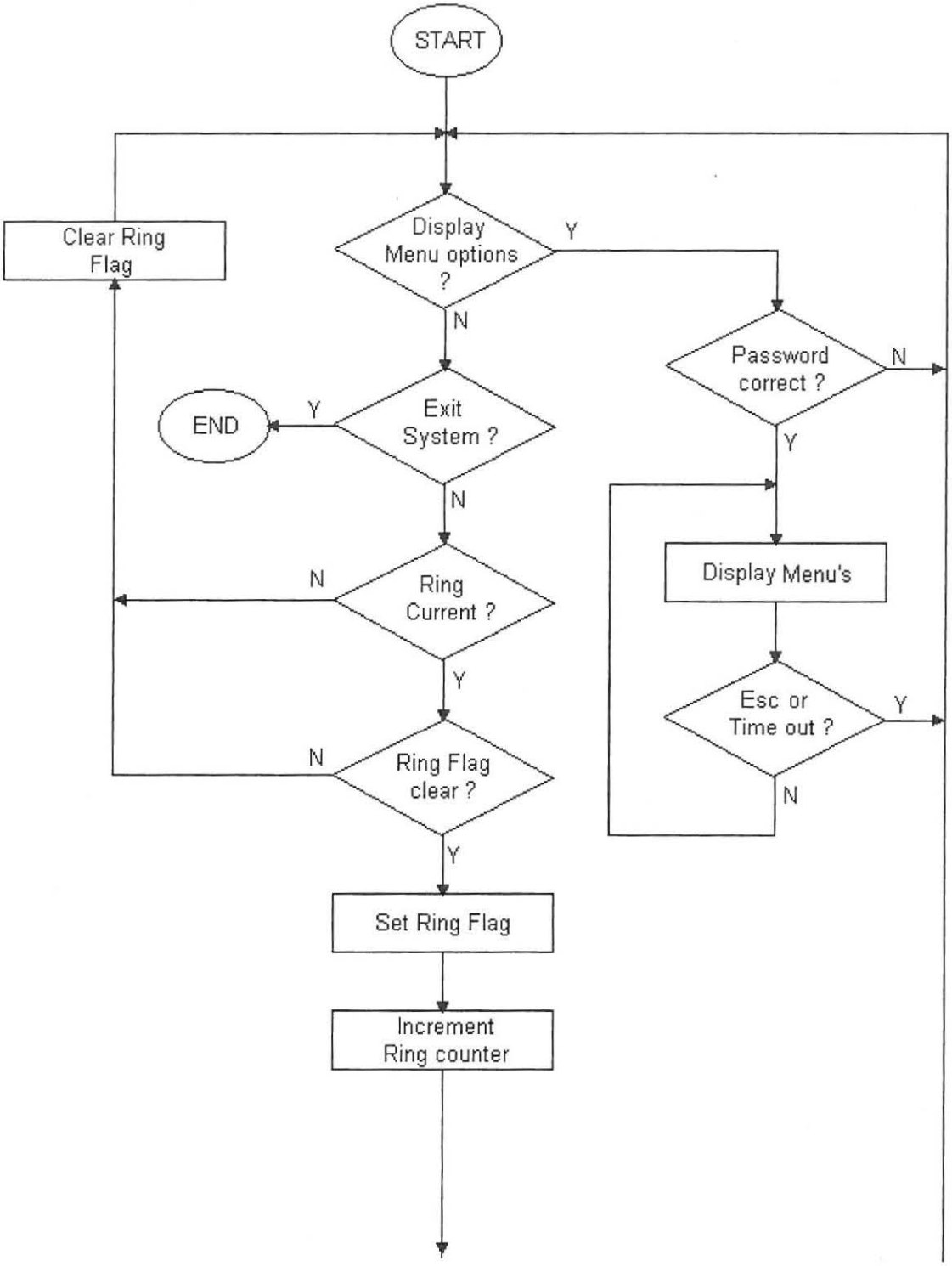


Figure 5.6 **"TestLine" Procedure** (Continued over page)

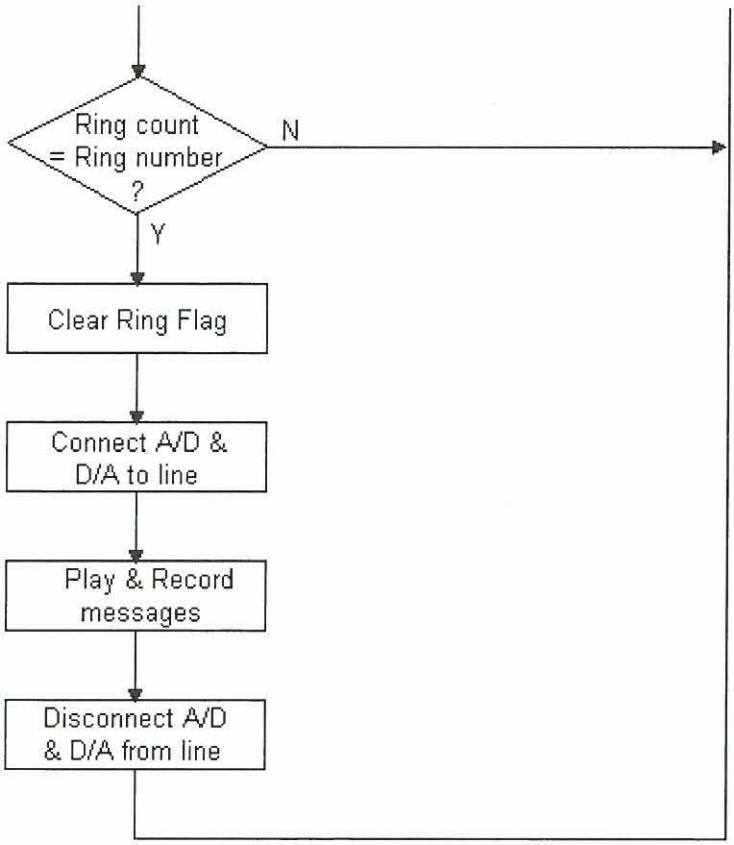


Figure 5.6 **"TestLine" Procedure**

TestLine: This procedure is used by the main program to continuously monitor the telephone line for an incoming call, when the system is in standby mode. It checks the ring detect bit on port-C of the PPI, which will indicate the presence of ring current. If ringing is present this procedure counts the ring bursts until the number correlates with the amount entered in the system data. It operates the line relay on the line interface and plays the "Greeting.msg" message to the caller. It also checks for a "#" entry with the use of the "GetDtmfChar" function, which would indicate a remote message enquiry; it otherwise initiates the line recording of the message. Figure 5.6 represents a flowchart of the "TestLine" procedure.

GetUserName: This is a function which receives the user's password and looks up the user name from the user data array. This information is then passed to the "MakeLineRec" procedure in order to display the message information (user name, date and time) on the "activity" screen.

ShowHelp: This procedure uses a text file, which contains the relevant user support information. It also uses a popup window in which the help information is displayed. Although only one help file is used, only the relevant information to the menu from where the help is required, will be displayed.

TogglePortBits: This procedure sets and resets the relevant bit on port-B, according to the parameter passed to it.

GetDtmfChar: This is a function used by the "TestLine" procedure. As has already been mentioned, this procedure checks for the entry of a "#" from the telephone keypad. It is also responsible for the conversion of any other data presented by the DTMF decoder.

5.3.3 Interrupt service routine

The interrupt service routine is introduced to the system to secure a constant time base between the add-on hardware interface and the PC host. This principle proves to make the digital telephone answering machine transportable, between PC's with different processor speeds, without any software timing problems. The A/D conversion hardware continuously generates interrupt signals at the sampling rate. These are interrupt requests (IRQ's) and can be temporarily turned off or masked by a software command. The "enable_IRQx" procedure gets the applicable interrupt mask register from the 8259A-PIC and clears the mask bit of the interrupt service routine. The "IntEnable" procedure then overwrites the system vector for the interrupt in question to point to a new service routine. The hardware signals that it needs attention on the that Interrupt line, while the CPU finishes its current instruction and then signals that it is ready by sending an "interrupt acknowledge" signal back to the hardware interface. On each interrupt request the status of the recording and play flags are checked and processing takes place as follows:

REC Flag is set - Sample is read from A/D to buffer

Play Flag is set - Byte is written to D/A from buffer

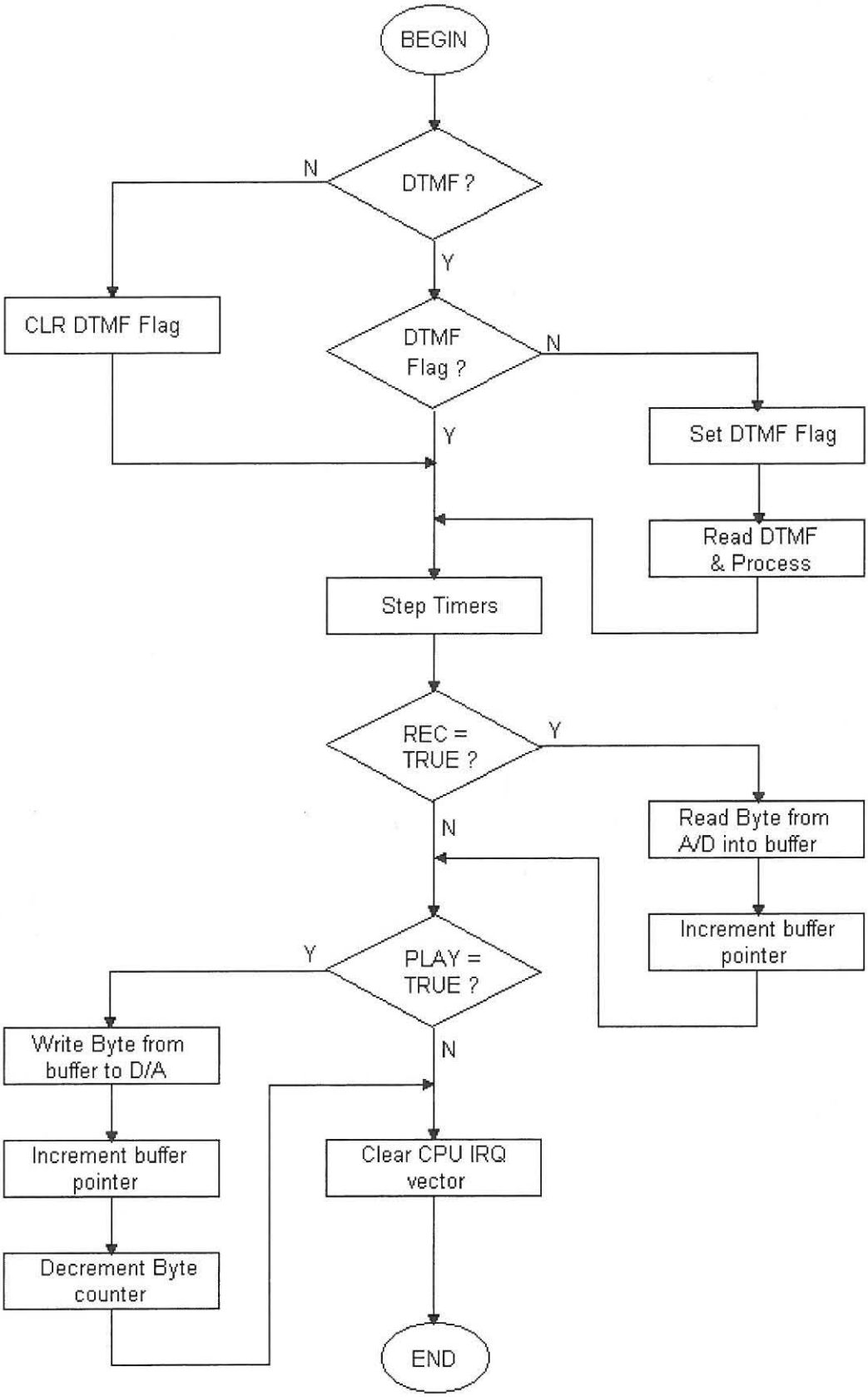


Figure 5.7 **"IntSample" Procedure**

If the system is shut down the IRQ's are restored. The "Disable_IRQx" procedure gets the applicable interrupt mask register from the PIC and sets the mask bit of the interrupt service routine. The "IntDisable" procedure then disables the new interrupt service routine and sets the system interrupt vector back to point to its original handler. A flowchart of the Interrupt Service Routine is shown in Figure 5.7.

5.4 APPEARANCE OF THE DISPLAY INTERFACE SOFTWARE

During the design the display interface system software was assessed against the following criteria, as proposed by Nielsen and Molich's [8] heuristic evaluation model:

- Is the system behaviour predictable?
- Is the system behaviour consistent?
- Is feedback provided?
- Is the user's memory not perhaps overloaded?
- Is the dialogue task orientated?

5.4.1 Predictability

Figure 5.8 represents the main menu of the telephone answering system. The user has sufficient information to predict the result of his future interaction with the interactive system, for example that by pressing F1 or by entering on the highlighted area, the mail menu will be displayed.

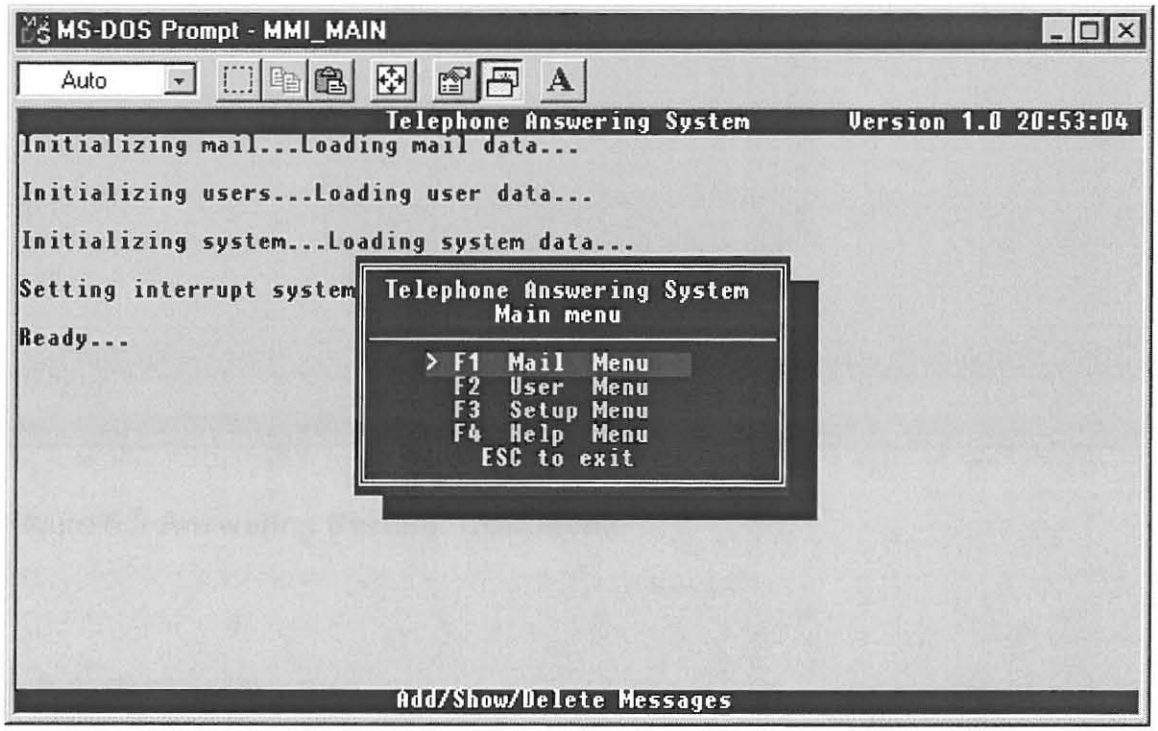


Figure 5.8 **Telephone Answering System: Main Menu**

Operation visibility equips the user with the ability to know which operations can be performed and which operations he might like to invoke, but which cannot be performed. For example, as shown in figure 5.9, the user is not allowed to proceed to the next level because, as

depicted at the bottom of the screen, only the system administrator is allowed to enter the next level.

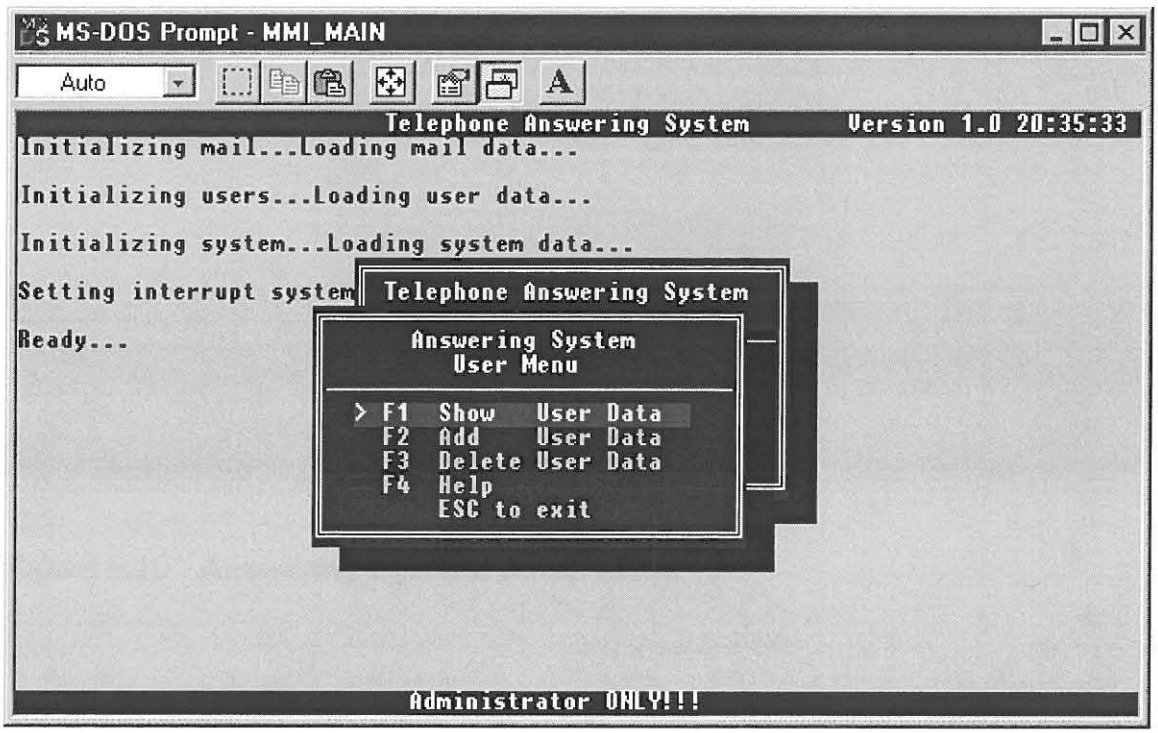


Figure 5.9 **Answering System: User Menu**

5.4.2 Consistency

By comparing figures 5.10 and 5.9 in the previous paragraph, it can be seen that the consistency relates to the likeness in system behaviour arising from similar situations or similar task objectives.

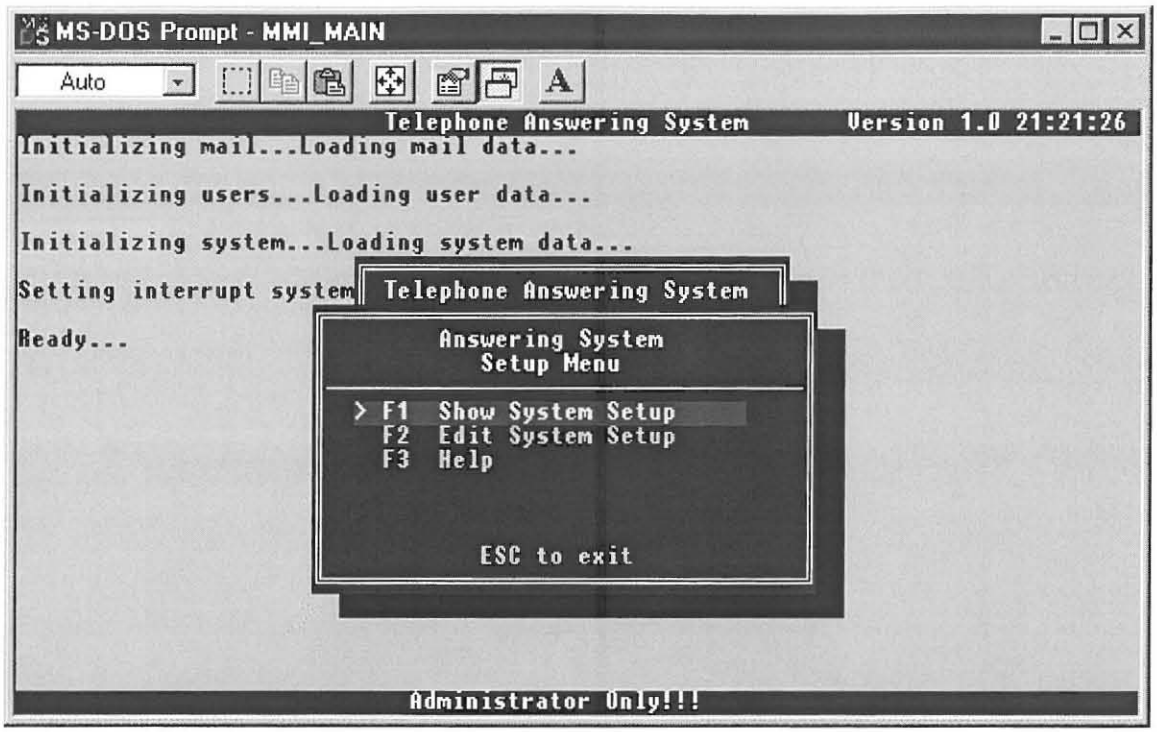


Figure 5.10 **Answering System: Setup Menu**

The system has consistency in command or option naming and also in command / argument invocation. Input expressions and output responses are consistent with respect to the meaning of the action to be taken.

5.4.3 Feedback

Figure 5.11 shows an example of feedback or form of response of the system.

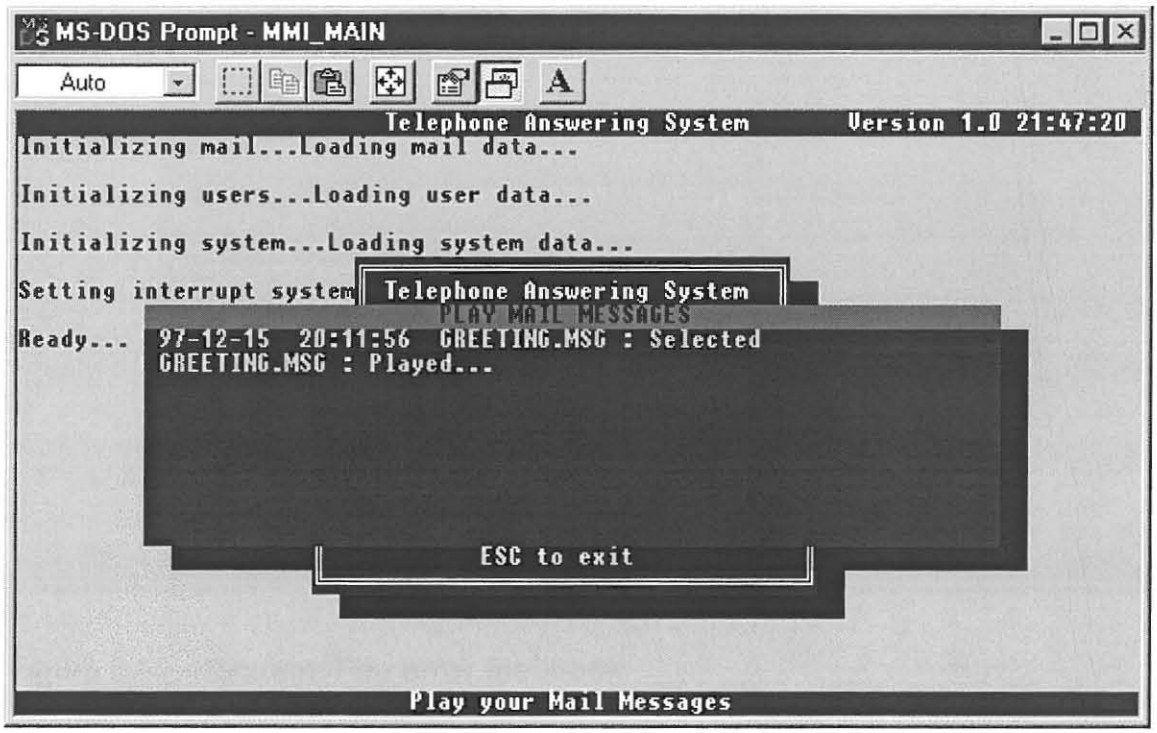


Figure 5.11 **System Feedback**

Other examples of feedback are found in error messages given by the system, for instance when a file has been erased from outside the system, the file name will still exist in the system's database file, and when the system is requested to retrieve the file, it will prompt an error as shown in figure 5.12.

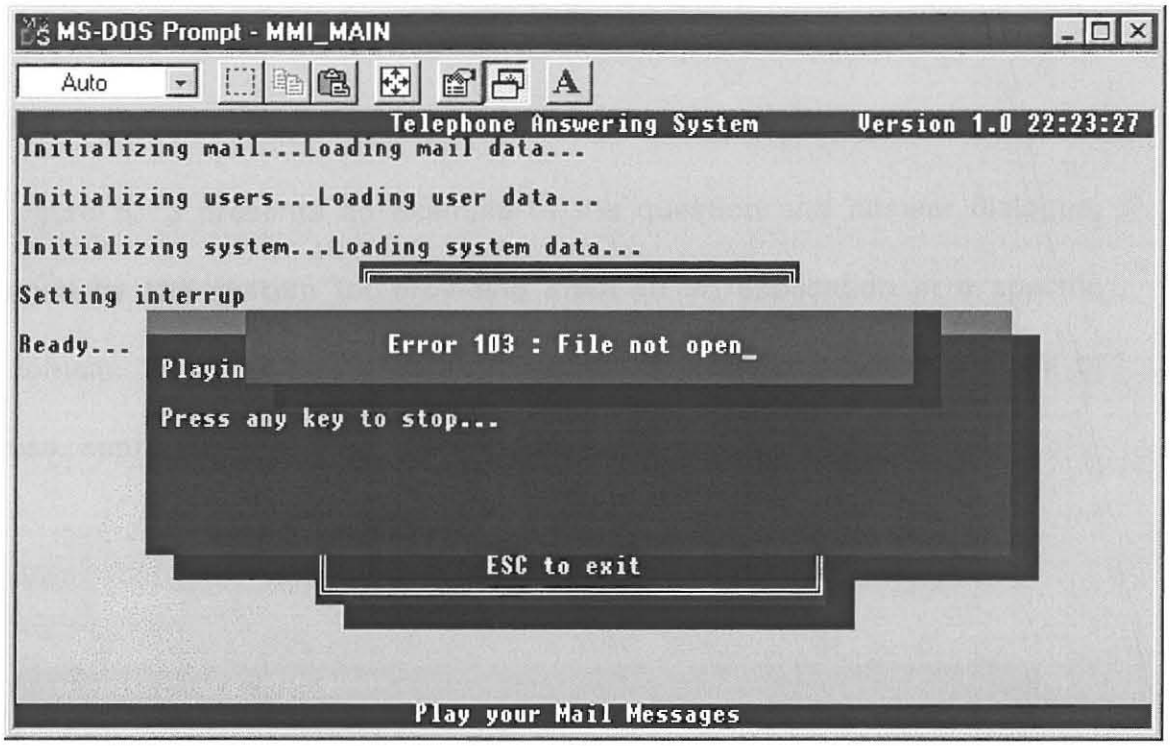


Figure 5.12 **System: File error feedback**

5.4.4 User memory overload

From the figures shown in the previous paragraphs it is evident that the system uses a menu-driven interface, with the set of options available to the user, displayed on the screen. Since the options are visible they are less demanding on the user, relying on recognition rather than recall. The windows have a graphical component in which the menu appears within a rectangular box. Furthermore, to ensure that the user's memory is not overloaded, the menus, windows and boxes are logically grouped and hierarchically ordered.

5.4.5 Task-orientated dialogue

Figure 5.13 presents an example of the question and answer dialogue, used by the system for providing input to an application in a specific domain. This is a simple task-orientated mechanism, which is easy to use, appropriate for the novice and casual user.

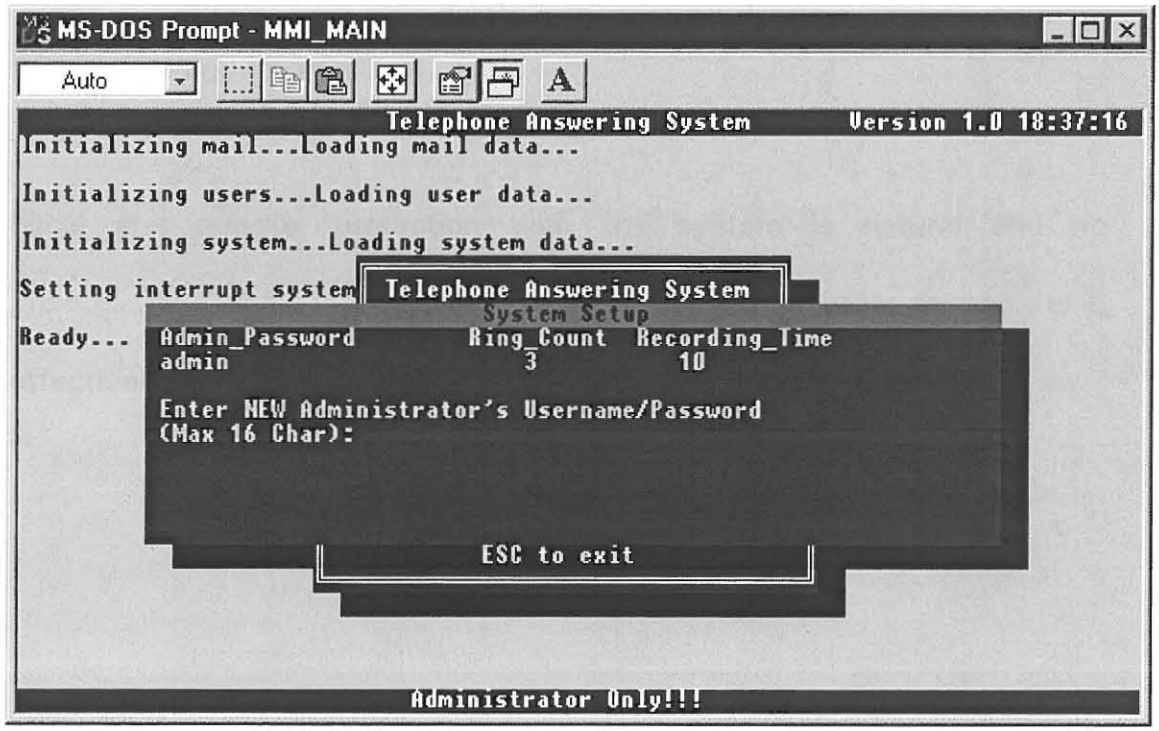


Figure 5.13 **Task-orientated dialogue**

5.5 CONCLUSION

The system software provides the user with menu options inside windows which prove to be logical and simple to use.

The display interface software is predictable and consistent in behaviour. Usable feedback is provided by the system, and the user's memory is not overloaded. Furthermore the display dialogue is task orientated, meaningful and informative.

Local and remote interaction with the system is natural and no prerequisite knowledge of the system is required in order to operate it effectively.

CHAPTER 6

6. EXPERIMENTAL EVALUATION

6.1 INTRODUCTION

The previous chapters discussed methodologies and models which supported the system design. However, although these techniques were employed, the role of the evaluation is to assess the design to ensure that it actually behaves as we expect and meets the requirements of the user.

Analytical and informal experimental testing was continuously performed throughout the design and assessed in identifying specific problems with the system, which resulted in modifications to the design.

This chapter will discuss the aims, methods and results of the following tests performed on the system:

- Evaluation of the telephone line interface
- Evaluation of the speech paths
- Evaluation of the ring detector control

6.1.1 Evaluation of the telephone line interface

Tests were performed on the telephone line interface to ensure that the interface complies with TELKOM SA specifications. Line measurements were made from the TELKOM SA Operation and Maintenance Centre (OMC) by using the Line Test System (LTS) Remote Test Unit (RTU). The method and results of this experiment will be discussed in paragraph 6.2.1.

6.1.2 Evaluation of the speech paths

This experiment was performed to determine the sound quality of the speech circuits in the answering system. An Auto-Tims analyser was connected to the system which performed various audio tests on the system. The method and results of this experiment will be discussed In paragraph 6.2.2.

6.1.3 Evaluation of the ring detector control

This experiment was performed to evaluate the functionality of the ring current detector under software control. The ring detector response was measured to determine the total delay in operation of the line relay circuit. The measured value was then compared to the preset software control

value. The method and results of this experiment will be discussed in paragraph 6.2.3.

6.2 EXPERIMENTS

6.2.1 Experiment 1: Evaluation of the telephone line interface

6.2.1.1 Objective

Verification test to determine whether the answering system's telephone line interface unit conforms with TELKOM SA specifications.

6.2.1.2 Method

Line measurements were made from the TELKOM SA Operation and Maintenance Centre (OMC) by using the Line Test System (LTS) Remote Test Unit (RTU). The RTU can test all subscriber lines originating from over 40 types of analog and digital switching machines. The RTU's basic testing features are listed below:

- DC leakage and voltage
- AC voltage
- AC induced current
- AC impedance
- Capacitance
- C-Message noise measurements

The RTU is installed in the electronic exchange in which the telephone line is to be tested. The RTU uses programmable accessing information to match testing needs. Tests are initiated by typing text commands on a terminal in the Operation and Maintenance Centre.

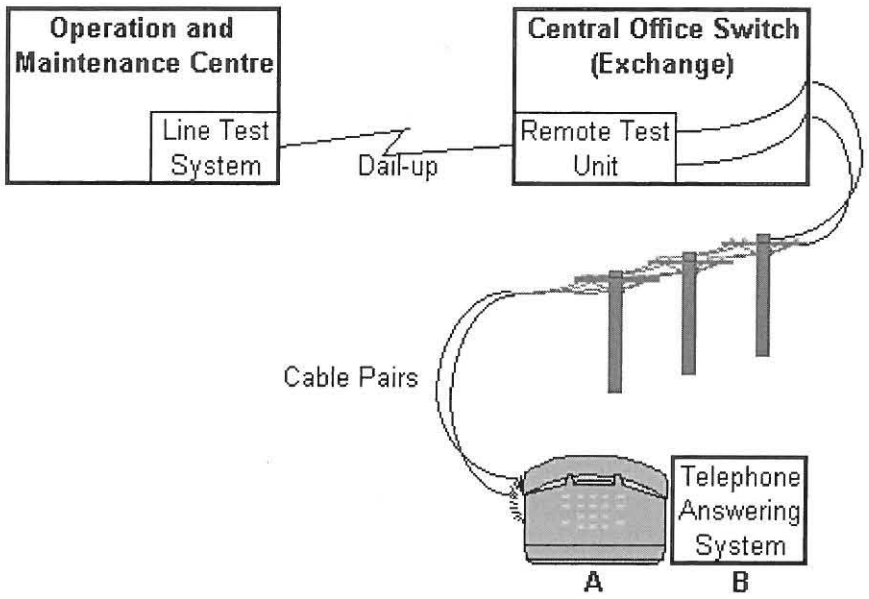


Figure 6.1 **LTS RTU Test setup**

As shown in Figure 6.1, the basic test was performed from the Operation and Maintenance Centre by dialing up the Remote Test Unit in the relevant exchange. The Remote Test Unit then performed the tests, as described in the following paragraph, on the subscriber line. The first test was on a standard telephone line with only a DTFM telephone (A) connected to the line. Secondly the same test was performed on the same telephone line with the DTMF telephone and answering system's

telephone line interface (A&B) connected to the telephone line. Both tests produced test results which will be discussed in the paragraphs to follow.

6.2.1.3 Test Results

A screen print of the test results is shown in figures 6.2 and 6.3. The results presented in figure 6.2 are measurements made on a standard DTMF telephone connected to the exchange line, while figure 6.3 represents measurements made with the telephone and answering system connected to the line.

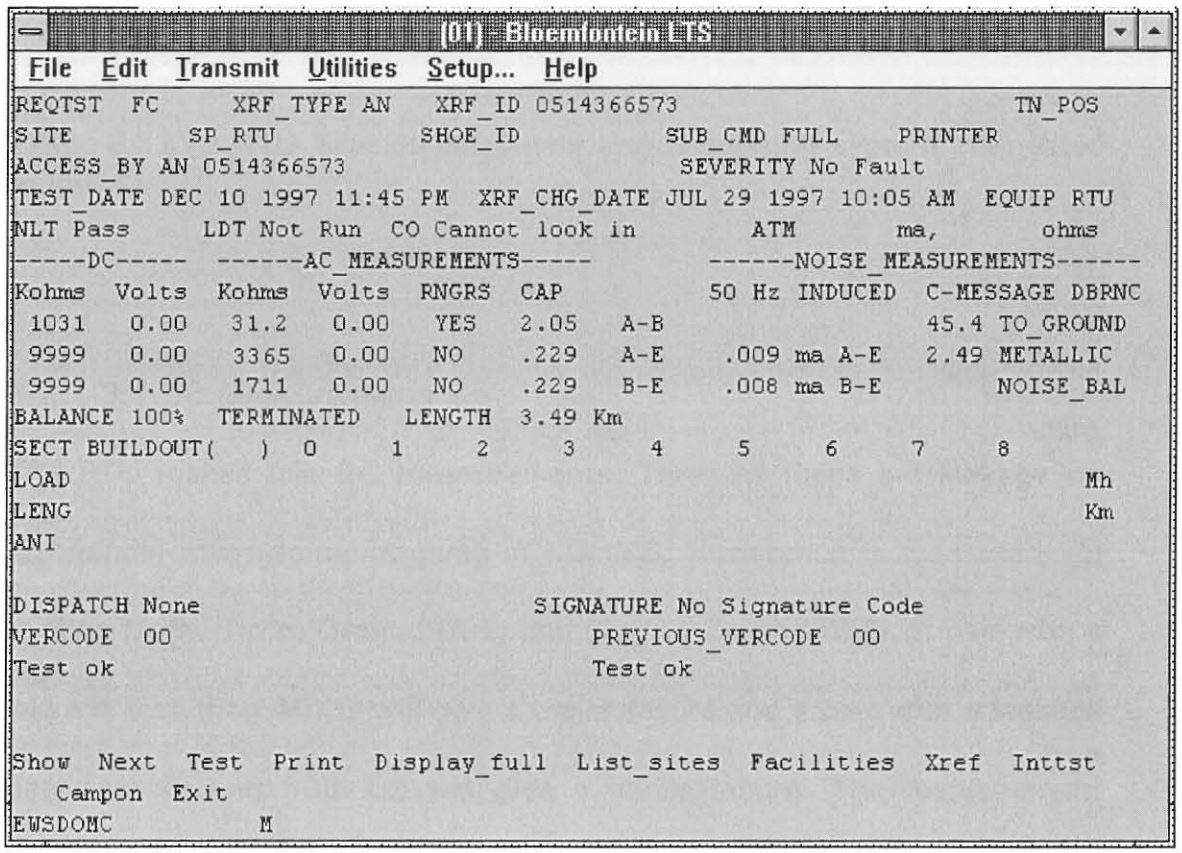


Figure 6.2 **LTS RTU Test on telephone connected to line**

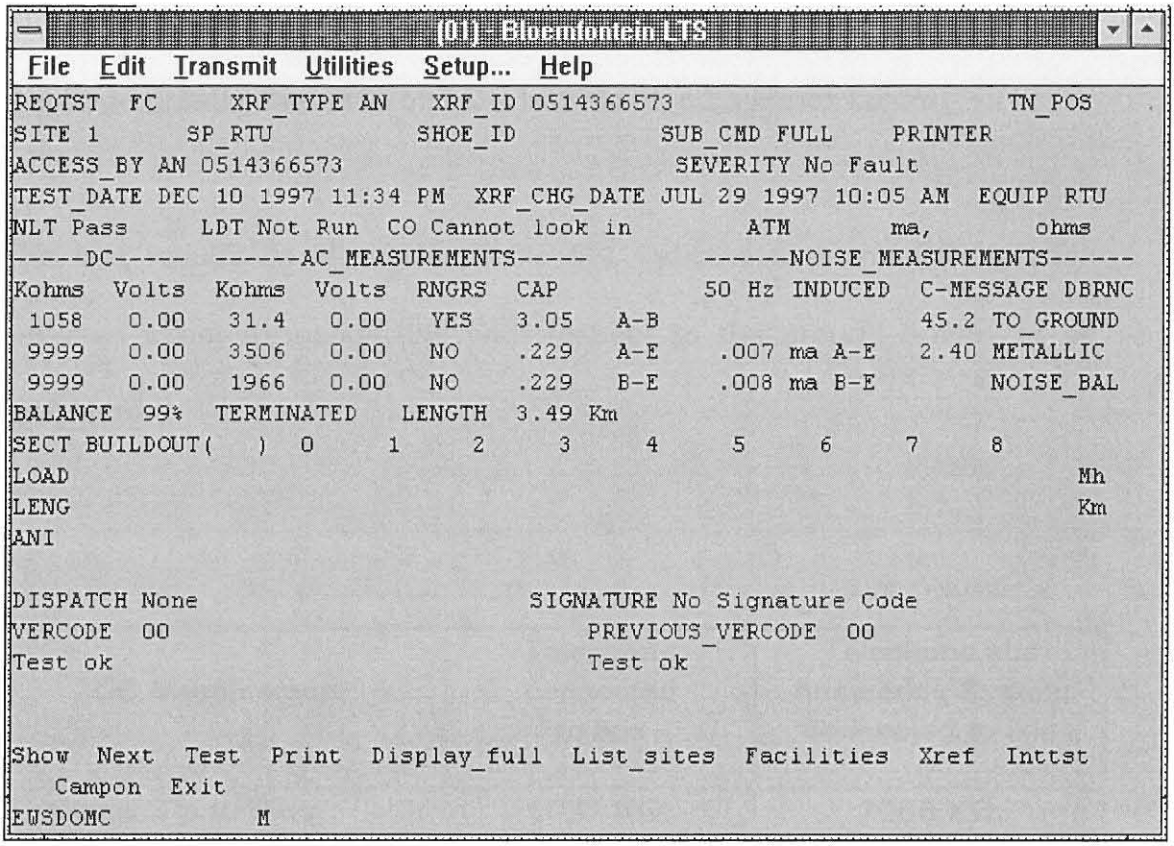


Figure 6.3 **LTS RTU Test on telephone and answering system connected to line**

6.2.1.3.1 DC Measurements

The RTU makes five DC measurements. Three of these are leakage (or resistance) measurements given in KOHMS. Resistance is measured Tip to Ring (A-B), Tip to Ground (A-E) and Ring to Ground (B-E). A pair with a leakage less than 40KΩ will give a major failure and a pair with a leakage between 40 and 500 KΩ will give a minor failure. The measurement range of these values is 0.00 to 9999KΩ.

The other two DC measurements are the voltage components of the pair given in VOLTS DC. Voltage is measured Tip to Ground and Ring to

Ground and from that the voltage Tip to Ring is calculated. Any pair with a voltage greater than 10Volts DC is considered a minor failure.

The test results as shown in Table 6.1 indicate that the DC isolation between either exchange line with respect to the ground proves to be satisfactory.

DC Measurement	Telephone connected to line	Telephone and Answering System connected to line
Leakage: Tip to Ring	1031 KΩ.	1058 KΩ.
Leakage: Tip to Ground	9999 KΩ.	9999 KΩ.
Leakage: Ring to Ground	9999 KΩ.	9999 KΩ.
Voltage: Tip to Ring	0.00 V	0.00 V
Voltage: Tip to Ground	0.00 V	0.00 V
Voltage: Ring to Ground	0.00 V	0.00 V

Table 6.1 **DC Measurements**

6.2.1.3.2 AC Measurements

The AC components of the line are measured at 30 Hz. An AC voltage above 15 VAC will give a minor failure. This test also determines the capacitance and AC impedance of the line. The AC test results are shown in Table 6.2 .

AC Measurement	Telephone connected to line	Telephone and Answering System connected to line
Voltage: Tip to Ring	0.00 VAC	0.00 VAC
Voltage: Tip to Ground	0.00 VAC	0.00 VAC
Voltage: Ring to Ground	0.00 VAC	0.00 VAC
Capacitance: Tip to Ring	2.05μF	3.05μF
Capacitance: Tip to Ground	0.229μF	0.229μF
Capacitance: Ring to Ground	0.229μF	0.229μF
Impedance: Tip to Ring	31.2KΩ	31.4KΩ
Impedance: Tip to Ground	3365KΩ	3506KΩ
Impedance: Ring to Ground	1711KΩ	1966KΩ

Table 6.2 **AC Measurements**

6.2.1.3.3 Noise Measurements

The RTU makes a series of noise measurements on the line. These include a measurement of 50 HZ induced current on the Tip and Ring to ground and C-Message noise measurements, both metallic and longitudinal. 50 Hz induced currents below 0.2 mA are ignored. C-Message noise on the line is measured in dBRNC (decibels referenced to noise, C-Message weighted) and subjected to the standard C-Message filtering (300 to 3000Hz). Longitudinal noise values measured to ground greater than 80 dBRNC and metallic noise across Tip and Ring above 20 dBRNC will give a minor failure during a C-Message noise test. The noise test results are shown in Table 6.3.

Noise Measurement	Telephone connected to line	Telephone and Answering System connected to line
50 Hz: Tip to Ground	0.009 ma	0.007 ma
50 Hz: Ring to Ground	0.008ma	0.008 ma
C-Message: Longitudinal noise	45.4 dBRNC	45.2 dBRNC
C-Message: Metallic noise	2.49 dBRNC	2.40 dBRNC

Table 6.3 **Noise measurements**

6.2.1.4 Conclusion

The DC measurements made show that the resistance between Tip and Ring, measured with the line interface connected to the exchange line, is 1058 KΩ. This is a satisfactory resistance with a value far less than the 40 KΩ failure limit. The Tip-to-earth and Ring-to-earth reading is 9999 KΩ, which means that earth isolated from the exchange pairs.

The AC and DC isolation between either exchange line and with respect to ground is 0.00V. This indicates high isolation at the line interface. AC capacitance and impedance readings are close to that of a standard telephone, which means that the line interface does not influence these components of the line.

The longitudinal noise value is 45.2 dBRNC for the telephone and line interface connected to the exchange line. This value is far from the 80 dBRNC limit and is more or less the same as the value obtained for only a standard telephone connected to the exchange line.

The test results obtained prove that the telephone line interface fully complies with Telkom SA specifications and can be connected to a subscriber line without exceeding the DC, AC and noise test tolerances as prescribed.

6.2.2 Experiment 2: Evaluation of the speech paths

6.2.2.1 Objective

Verification test to determine the answering system's speech circuit, amplifiers and filters, analog-to-digital and digital-to-analog converter circuit functionality and sound quality.

6.2.2.2 Method

Measurements were made on the telephone answering system by using the Auto-Tims analyser. As shown in Figure 6.4 the analyser was connected to the Tip and Ring of the Telephone Answering System. To perform the tests in the following paragraphs, the Auto-Tims transmitted (TX) the required signals for each test. The Telephone Answering System then recorded the signals. This operation prepared the signals to be played back by the system. The Auto-Tims then analysed the received (RX) signals. A printer which was connected to the analyser printed all the test results.

The following tests were performed with the Auto-Tims and will be discussed in the following paragraphs:

- Voice frequency sweep
- Frequency shift
- Frequency offset
- Phase jitter
- Noise / noise with tone
- Non-linear distortion

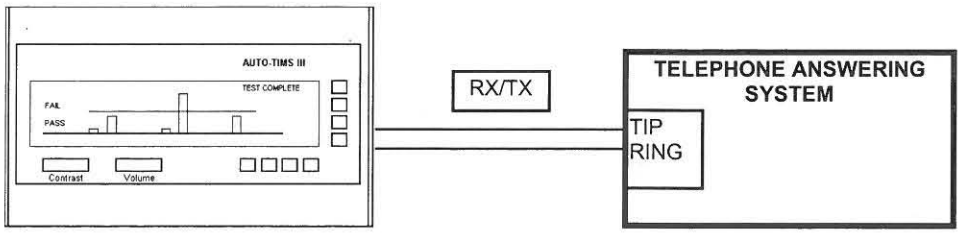


Figure 6.4 **Auto-Tims Test Setup**

6.2.2.3 Test results

6.2.2.3.1 VF sweep

The Voice Frequency Level Sweep test allowed testing of the Telephone Answering System in the frequency range 200 to 4000 Hz. The test was set up as explained in paragraph 6.2.2.2 and Figure 6.4. Table 6.4 shows the test parameters used for the experiment and Figure 6.5 shows the graphic test results obtained from the test.

VF SWEEP	
Lower TX Freq.	200 Hz
Upper TX Freq.	4000 Hz
Reference Freq.	800 Hz
Sweep Method	Auto
Sweep Speed	70 Millisec/ Step
Freq. Step	100 Hz
Transmit Level	-6 dBm
Vertical Scale	10/-40 dB

Table 6.4 **VF Sweep Parameters**

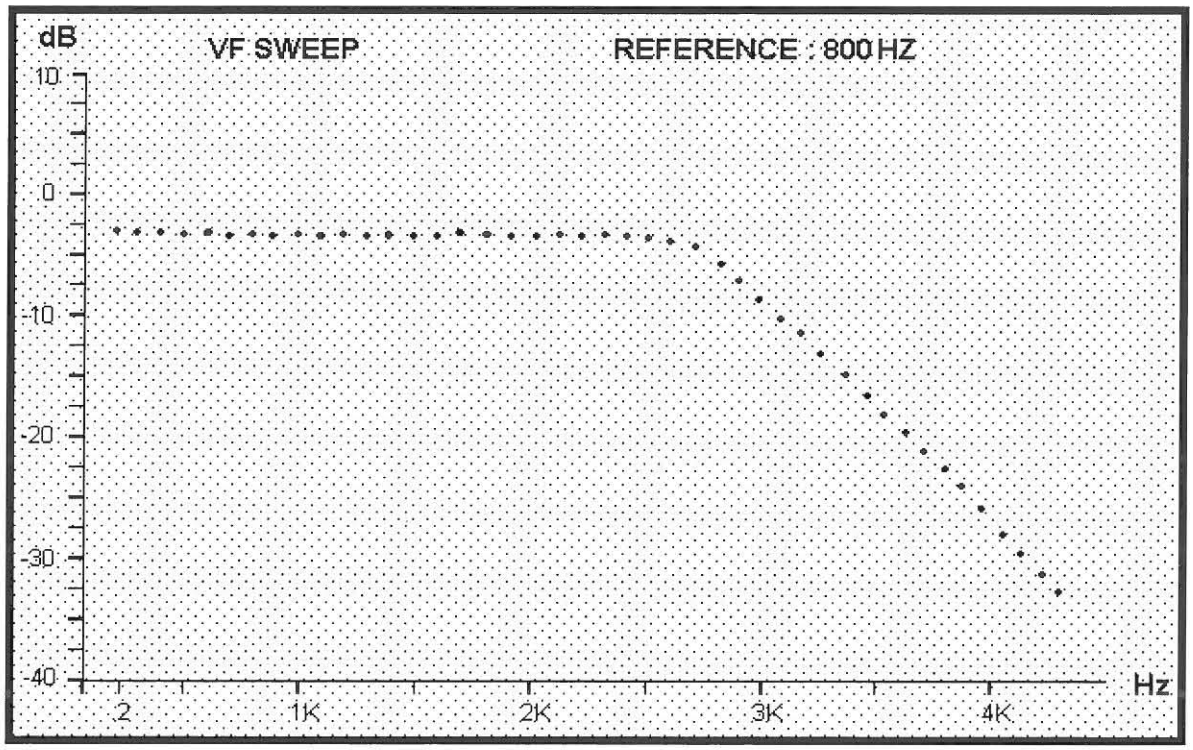


Figure 6.5 **VF Sweep Frequency Response**

6.2.2.3.2 Frequency Shift

The Frequency Shift measurement, as shown in Table 6.5, is calculated from a 1020 Hz and another tone with a 2:1 harmonic relationship, simultaneously transmitted.

FREQUENCY SHIFT			
TRANSMIT		RECEIVE	
Level	: -6.0 dBm	Offset	: -0.01 Hz
Impedance	: 600 Ohm		
TX/RX Term.	:Term/Term	Level	: -8.5 dBm
TX/RX	:Normal		
Reference	: 0.0 dBr		

Table 6.5 **Frequency Shift**

6.2.2.3.3 Frequency Offset

The Frequency Offset measurement, as shown in Table 6.6, is calculated from a 1020 Hz tone. This offset could occur if the sample and playback clock frequency were offset.

FREQUENCY OFFSET			
TRANSMIT		RECEIVE	
Level	: -6.0 dBm	Offset	: -0.00 Hz
Freq.	: 1020 Hz		
Impedance	: 600 Ohm	Tone Level	: -8.5 dBm
TX/RX Term.	:Term/Term		
TX/RX	:Normal		
Reference	: 0.0 dBr		

Table 6.6 **Frequency Offset**

6.2.2.3.4 Phase Jitter

The Phase Jitter measurement, as shown in Table 6.7, detects phase jitter in the 4 to 300 Hz range, on the 1020 Hz test tone. Common sources of phase jitter are power supply ripple and its harmonics.

PHASE JITTER			
TRANSMIT		RECEIVE	
Level	: -6.0 dBm	Offset	: -0.01 Hz
Freq.	: 1020 Hz		
Impedance	: 600 Ohm	Tone Level	: -8.5 dBm
TX/RX Term.	:Term/Term		
TX/RX	:Normal		
Reference	: 0.0 dBr		

Table 6.7 **Phase Jitter**

6.2.2.3.5 Noise / Noise with tone

The circuit noise with a tone may be different from the pure noise on the circuit. This is due to companding and quantizing, that are active only when a signal is present. This test also provides a signal-to-noise ratio measured at 1020 Hz. This test is performed by removing the received 1020Hz holding tone with a notch filter and measuring the noise level through the selected noise filter. The test results are shown in Table 6.8.

NOISE / NOISE WITH TONE	
TRANSMIT	RECEIVE
Notch Filter : Auto	Noise : -42.3 dBm
Noise Filter : Psopho	
Impedance : 600 Ohm	Tone Freq : 1020 Hz
TX/RX Term. : Term/Term	Tone Level : -8.5 dBm
TX/RX : Normal	
Reference : 0.0 dBr	S/N Ratio : 34 dB

Table 6.8 **Noise / Noise with tone**

6.2.2.3.6 Non-linear Distortion

Non-linearity in a circuit can cause distortion of a signal. The transmitted signal consists of four tones - two tones centered at 860 Hz, 6 Hz apart, and two tones centered at 1380 Hz, 16 Hz apart. The receiver measures the 3rd-order products in a narrow band centered at about 1.9 kHz, and the 2nd-order products in the narrow bands centered at about 520Hz and 2240 Hz. The results are shown in Table 6.9.

NON - LINEAR DISTORTION	
TRANSMIT	RECEIVE
Level : -6.0 dBm	2nd Order : -39.1 dB
4 Tones	3rd Order : -17.8 dB
Impedance : 600 Ohm	
TX/RX Term. : Term/Term	Signal Level : -8.5 dBm
TX/RX : Normal	

Table 6.9 **Non-linear Distortion**

6.2.2.4 Conclusion

The Voice Frequency Sweep test showed that the cutoff frequency of the system is evident above the 3kHz range. The Frequency Shift and Frequency Offset tests indicated that there is no deviation in the sampling and playback clock frequencies of the system. The Phase Jitter test also proved to be satisfactory. Finally, the results of the Noise and Distortion tests showed that the audio signal of the system is at an acceptable standard.

Audio tests performed by the Auto-Tims analyser proved that the system's speech path is in acceptable hardware functionality under software control.

6.2.3 Experiment 3: Evaluation of the ring detector control

6.2.3.1 Objective

Verification test to evaluate the functionality and response of the ring current detector under software control.

6.2.3.2 Method

The ring detector response was measured to determine the total delay in operation of the line relay circuitry. The software parameter value of Ring_Count was set to the value of 2. Test calls were made to the

system and the time was measured from the first ring burst until the line was looped by the RL1 relay. The measured values were then plotted on a graph from which the average delay was calculated. The time value of the various durations and average value ares measured in seconds. This experiment was carried out several times to produce enough values to determine the average delay in the detection and relay operation circuitry under software control.

6.2.3.3 Test results

The results of the ring detector control experiment are shown in Figure 6.6. The maximum delay measured was 7 seconds and the minimum delay was 3 seconds. These values give an average total delay of 5 seconds in circuit operation.

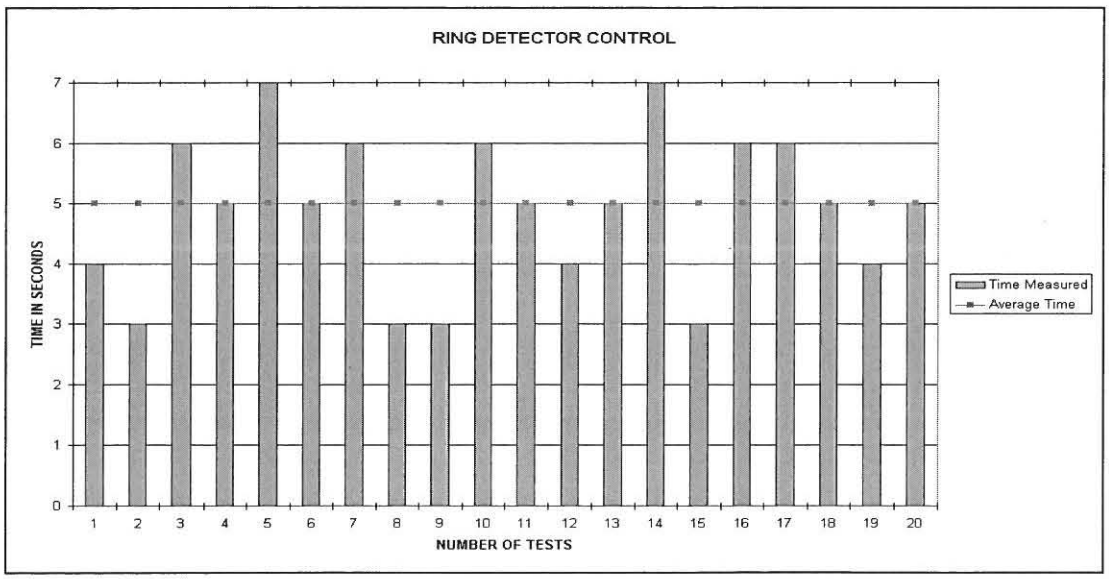


Figure 6.6 **Response of the Ring Detector Control**

6.2.3.4 Conclusion

The ring detector response experiment showed that the ring detector and associated resistor/capacitor (RC) circuit causes an average delay of 5 seconds in relay operation when the Ring_Count value is set to 2. It is known that the ring current cycle generated by the telephone network is 3 seconds long. This cycle can be broken up into the following segments:

1 Ring cycle = (0,4on + 0,2off + 0,4on + 2off) Seconds = 3 Seconds.

Two ring cycles would then be 6 seconds long. If this value is compared to the average delay of 5 seconds, as calculated from the experiment, it is evident that the values are very close and although fluctuations do occur, these will not be more than the duration of 1 ring cycle.

6.3 CONCLUSION

The test results from the first experiment, which was performed on the telephone line interface proved that the interface complies with TELKOM SA specifications and that the system can be connected to a telephone network access line without exceeding the DC, AC and noise test tolerances as prescribed.

The second experiment was performed to determine the sound or audio quality of the speech circuits in the answering system.

The results from this test indicated that the amplifiers and filters, analog-to-digital and digital-to-analog converter produce an audio signal with acceptable sound quality under software control.

From the third experiment we saw that the ring detector response and associated RC circuits cause an average delay which is very similar to the value which can be determined from the programmed value. We also discovered that the variation in delay will not be longer than the duration of 1 ring cycle.

All the experimental results obtained prove that the digital telephone answering system complies with the necessary standards and can be used on a subscriber line without exceeding the prescribed tolerances. Recorded and recovered message quality is satisfactory and proves acceptable hardware functionality under software control.

Results from messages obtained during the test period showed that more than 80% of the users make use of the menu options offered by the system and leave messages successfully.

Adequate system security is provided to prevent unauthorized access to messages and variable parameters. The system operation is user-friendly and easy access to the user's own messages is allowed.

All the system software as described in chapter 5 is written by using Turbo Pascal, which provides an integrated debugger which was continuously used to prove the correctness of the code in the software modules and the correct functionality of the executable program.

CHAPTER 7

7. SUMMARY

The aim of this study was to design an intelligent low-cost digital telephone answering system on a personal computer by developing modular software and an analog-to-digital hardware interface.

The "non audio nature" of a computer is mainly concerned with moving numbers from one place to another in its own time. This is in contrast with the "real-time" entity of audio which requires sound to be played back continuously and without any breaks. Therefore the designers of PC-based commercial digital recording systems overcome this problem by duplicating the processor and memory units on the system to handle the real-time processing of audio data.

The design as described in chapters four and five broke away from the commercial concepts of message recording and adopted a design which provides optimal PC-host hardware utilization and the efficient use of the system resources without the duplication of expensive hardware.

The system is designed on a modular basis. Each module generates specific functions and contributes towards the overall system function.

The interfacing amongst software modules, and between the software and hardware modules, acts as checking medium for the correctness of data and task transfers. Each module accesses only those data and parameters it needs to know about, and has as little common data as possible. The program does status checks on data access in conjunction with hardware status registers and program loop detection methods.

In the event of software or hardware failures, an error message is displayed to assist in fault detection in abnormal conditions.

The system allows day-to-day variable parameters and data to be modified according to demand. Passwords are available on two levels, that is, administrator and normal user level, to prevent unauthorized access to the variable parameters, but to allow access on a selective basis to the user's own messages.

This study proved that an intelligent digital telephone answering system implemented on a personal computer, without the use of digital speech processor technologies, has an economic and creative advantage over existing answering systems.

From the principles of various related technologies, as discussed in this paper, the researcher gained knowledge with regard to the following:

- Speech recording methods
- Telephone answering equipment
- Digitization principles
- Computer hardware
- Computer interfacing
- Telephone line interfacing
- A/D and D/A conversion implementation
- DTMF signalling
- Modular software design
- Functionality and advantages of using interrupts
- Hardware design
- Evaluation and testing

Future research which may emerge from this study could be the implementation of an artificial intelligent voice-operated telephone answering system on a personal computer without the incorporation of expensive speech synthesis and recognition IC's.

This means that we will have to prove that we do not hear with our ears exclusively, but also with the "hardware" and "software" of the brain.....

8. REFERENCES

[1] Ambardar, A. **Analog and Digital Signal Processing.** Boston: PWS Publishing Company, 1995. 478-484 pp.

[2] Analog Devices, **Analog-Digital conversion handbook.** London: Prentice Hall, 1986. 316-347 pp.

[3] Bellamy, J. **Digital Telephony.** USA: John Wiley & Sons, 1991. 93-158 pp.

[4] Bradley, A.C. **Peripherals for Computer Systems.** London: Macmillan Education Ltd, 1991. 96 pp.

[5] Brey, **Microprocessor/Hardware Interfacing and Applications.** Ohio: Bell & Howell Company, 1984. 157-173 pp.

[6] Bristow, G. **Electronic Speech Recognition.** London: William Collins Sons & Co. Ltd, 1986. 7 pp.

[7] CCITT, **Man-Machine Language (MML).** Geneva: IX Plenary Assembly, 1988.

[8] Dix, A. **Human-Computer Interaction.** Hertfordshire: Prentice Hall International Ltd, 1993. 9-416 pp.

[9] Dixey, G. **Microprocessor Interfacing.** Cheltenham: Stanley Thornes (Publishers) Ltd, 1991. 50-51 pp.

[10] Duuren, J. **Telecommunication Networks and Services.** Cornwall: Addison-Wesley Publishing Company Inc, 1992. 188-189 pp.

[11] Edis, E.A. **Newnes Telecommunications Pocket Book.** Oxford: Butterworth-Heinemann Ltd, 1992. 114 pp.

[12] Freeman, R.L. **Reference Manual for Telecommunications Engineering.** USA: John Wiley & Sons, 1985.

[13] Freeman, R.L. **Telecommunications System Engineering.** USA: John Wiley & Sons, 1989.

[14] Furui, S. **Digital Speech Processing, Synthesis and Recognition.** New York: Marcel Dekker Inc, 1989. 139-204 pp.

[15] Hall, D.V. **Microprocessors and Interfacing.** Singapore: McGraw-Hill Book Co, 1986. 232-280 pp.

[16] Haykin, S.S. **An Introduction to Analog and Digital Communications.** Singapore: John Wiley & Sons Inc, 1989. 177-187 pp.

[17] Homes, J.N. **Speech Synthesis and Recognition.** Berkshire: Van Nostrand Reinhold (UK) Co. Ltd, 1988. 54-72 pp.

[18] Hutchings, H. Interfacing with C, **Electronics World + Wireless World**, April 1990. pp.280-288.

[19] Internet: Dallas Semiconductor Corp: Data Sheets - PDF Format, http://cosmo.tky.hut.fi/~kosonen/prod/dallas/dallas.html.

[20] James. M. **LSI Interfacing.** Oxford: Heinemann Professional Publishing Ltd, 1998. 136-140 pp.

[21] Jayant, N.S. **Digital Coding of Waveforms.** U.S.A: Bell Telephone Laboratories Inc, 1984. 221-225 pp.

[22] Jiménez, R. **Designing with Speech Processing Chips.** California: Academic Press Inc, 1991. 1-28 pp.

[23] Lathi, B.P. **Modern Digital and Analog Communication Systems.** Florida: The Dryden Press, 1989.

[24] Laurel, B. **The Art of Human-Computer Interface Design.** USA: Apple Computer Inc, 1990.

[25] Lawton, L.S. **Integrated Digital Networks.** Wilmslow: Sigma Press, 76-82 pp.

[26] Mano, M.M. **Computer System Architecture.** New Jersey: Prentice-Hall Inc, 1976.

[27] Minoli, D. **Telecommunication Technology Handbook.** USA: Artech House, 1991.

[28] Morgan, N. **Talking Chips.** U.S.A: McGraw-Hill Inc, 1984.

[29] Morris, J.C. **Analog Electronics.** London: Thomson Litho Ltd, 1991. 67-82 pp.

[30] Niewiadomski, S. **Filter Handbook.** Oxford: Heinemann Professional Publishing Ltd, 1989.

[31] Noll, A.M. **Introduction to Telecommunication Electronics.** Norwood: Artech House Inc, 1988. 254-255 pp.

[32] Northrop, R.B. **Analog Electronic Circuits.** USA: Addison-Wesley, 1990. 476-477 pp.

[33] Rafiquzzaman, M. **Microprocessor and Microcomputer-based System Design.** Florida: CRC Press, 1990. 25 pp.

[34] Roden, M. S. **Digital Communication Systems Design.** UK: Prentice-Hall International Editions, 1988.

[35] Roe, D.B. **Voice Communication Between Humans and Machines.** Washington: National Academy Press, 1994. 6-8 pp.

[36] Rumsey, F. **Tapeless Sound Recording.** Oxford: Butterworth & Co. Publishers Ltd, 1990. 1-57 pp.

[37] Rumsey, F. **The Digital Interface Handbook.** Oxford: Butterworth-Heinemann Ltd, 1993. 24-37 pp.

[38] Rumsey, F. **Sound and Recording: An Introduction.** Oxford: Butterworth-Heinemann Ltd, 1992. 176-190 pp.

[39] Seals, R.C. **Microprocessor-based Systems.** Leckhampton: Stanley Thorns Publishers Ltd, 1992. 191-192 pp.

[40] Sokolowski, S. **The Talking Telephone.** USA: McGraw-Hill, 1990. 65 pp.

[41] Tooley M.H. **Electronic Circuits Handbook: Design, Testing and Construction.** Oxford: Butterworth-Heinemann Ltd, 1995. 190-195 pp.

[42] Uffenbeck, J. **The 8086/8088 Family - Design, Programming and Interfacing.** New Jersey: Prentice-Hall, 1987.

[43] Vassos, B.H. **Analog and Computer Electronics for Scientists.** New York: John Wiley & Sons Inc, 1993. 255-291 pp.

[44] Verburg, W.P. **Principles of Digital Systems and 8-bit Microprocessors.** Pretoria: Dirk, 1989. 230-242 pp.

[45] Watkinson, J. **Coding for Digital Recording.** London: Focal Press, 1990. 1-8 pp.

[46] Wheddon, C. & Linggard, R. **Speech Synthesis - Application of Computer Systems.** Suffolk: Chapman and Hall, 1990. 1-26 pp.

References consulted in developing the software:

[47] Crawford, J.H. **Programming the 80386.** Alameda: Sybex Inc, 1987.

[48] Hergert, D. **Turbo Pascal Programming for the PC.** Carmel: Sams, 1991.

[49] Mojena, R. **Turbo Pascal.** Boston: PWS Publishing Company, 1993.

[50] O' Brien, S.K. **Turbo Pascal 5.5: The Complete Reference.** Berkley: Osborne McGraw-Hill Inc, 1989.

[51] O' Brien, S.K. **Turbo Pascal 6: The Complete Reference.** Berkley: Osborne McGraw-Hill Inc, 1991.

[52] Ohlsen, C. & Stoker, G. **Turbo Pascal Advanced Techniques.** Carmel: Que Corporation, 1989.

[53] Smith, J. **Getting the Most from Turbo Pascal.** New York: Micro Text Productions, 1987.

[54] Swan, T. **Mastering Turbo Pascal Files.** USA: Howard W. Sams & Co, 1987.

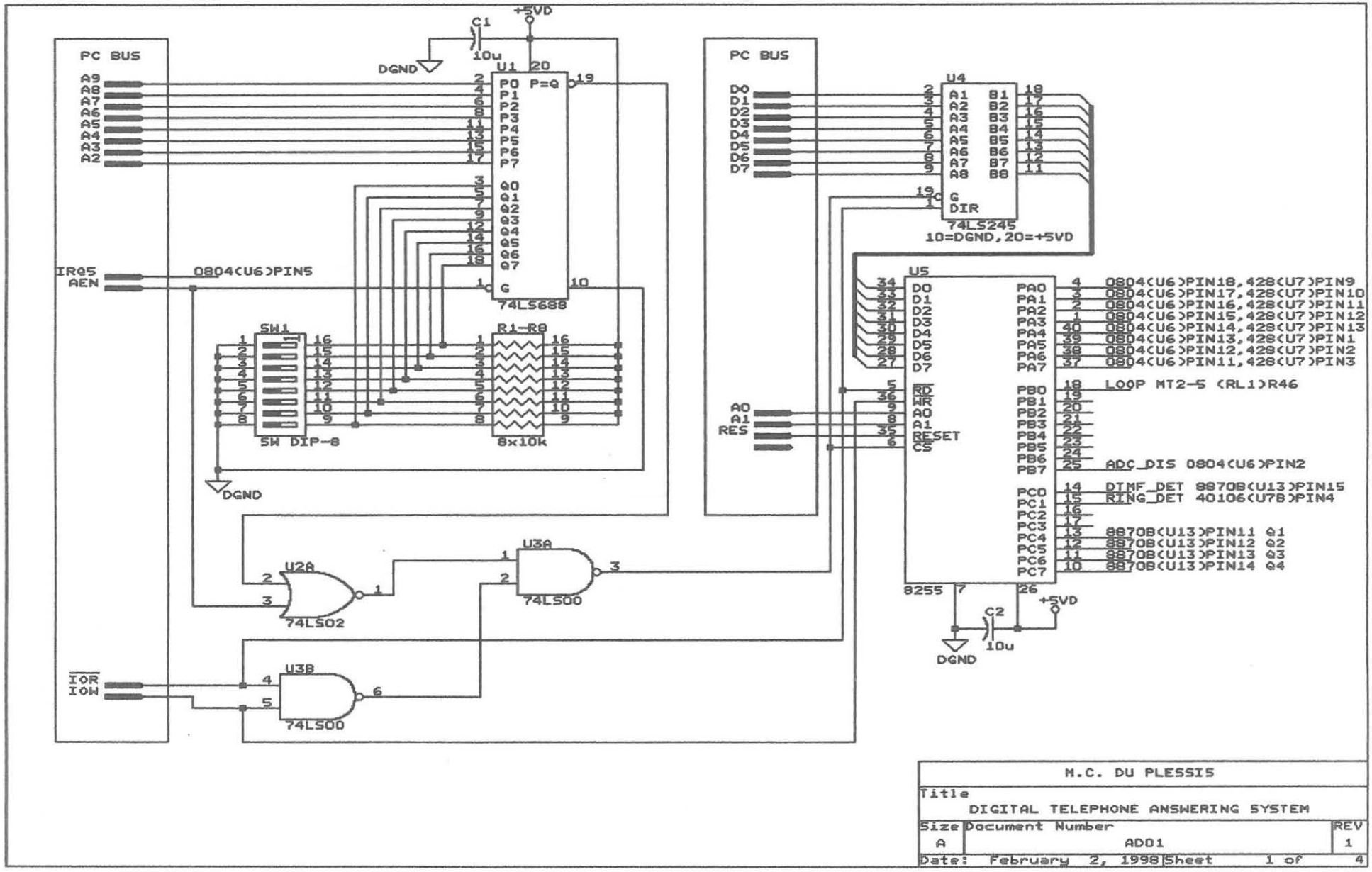
[55] Swan, T. **Mastering Turbo Pascal 6.** Carmel: Hyden Books, 1993.

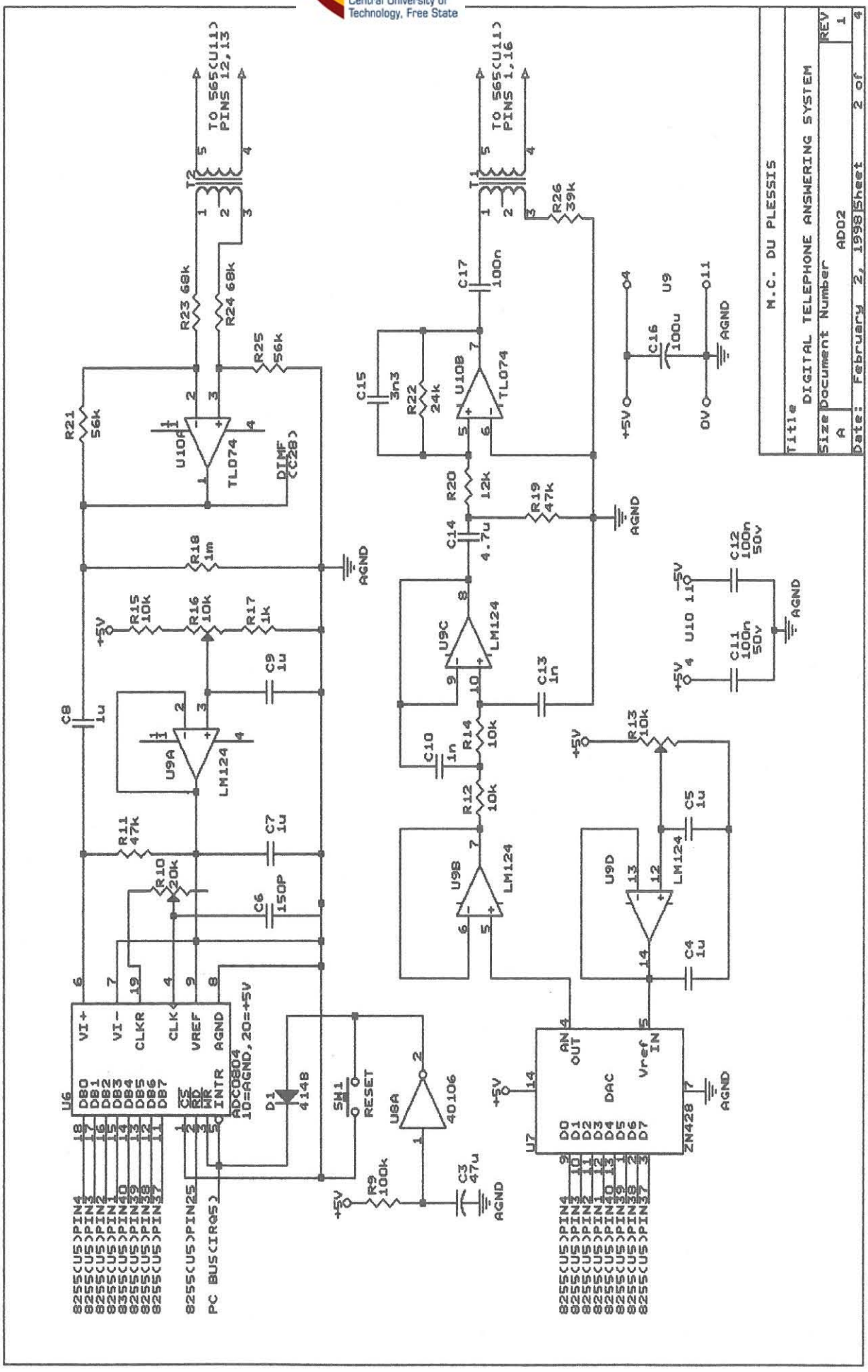
[56] Tanik, M.M. **Library of Turbo Pascal Programs.** Plano: Wordware Publishing Inc, 1987.

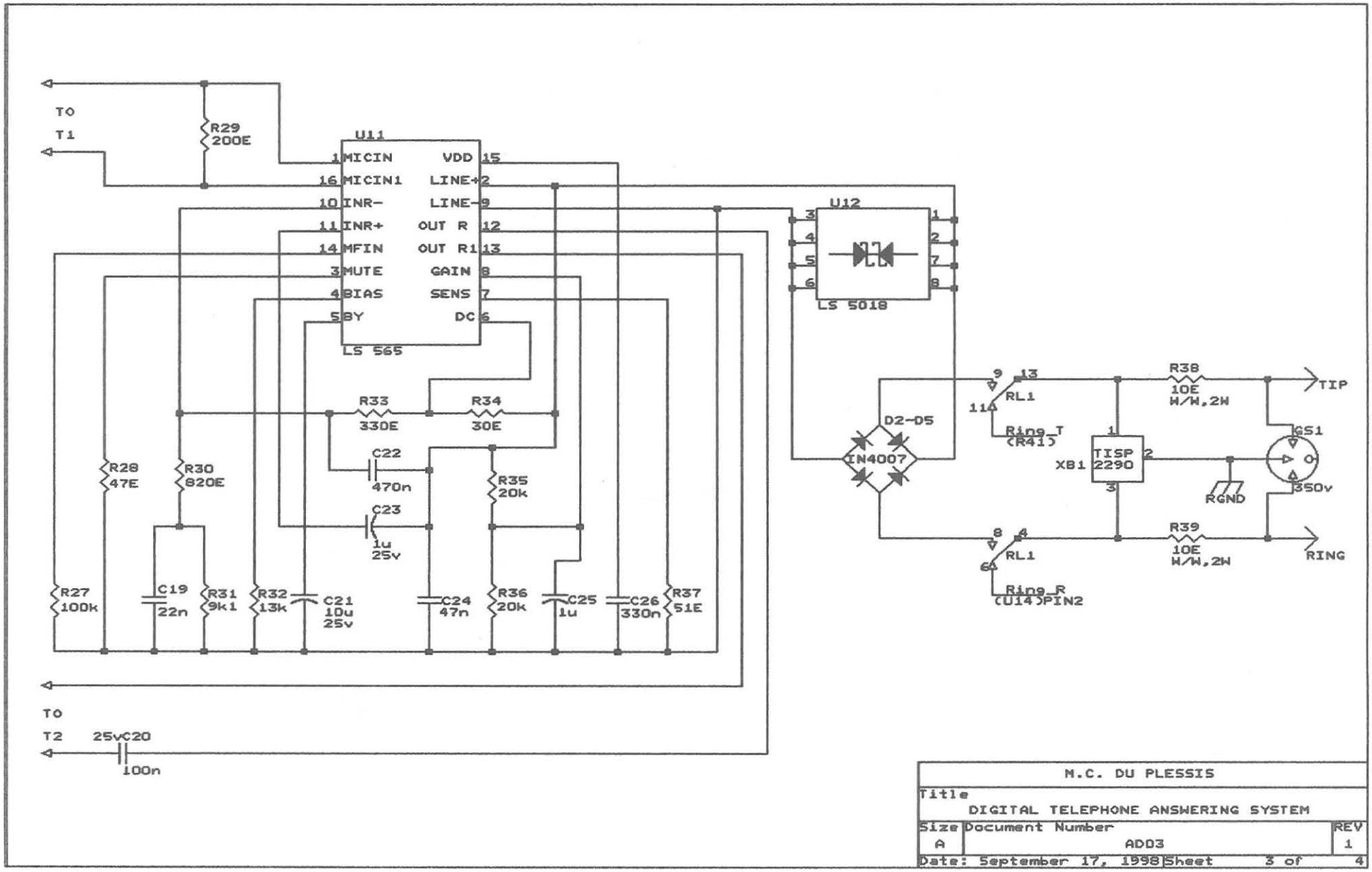
[57] Yester, M. **Using Turbo Pascal.** Carmel: Que Corporation, 1991.

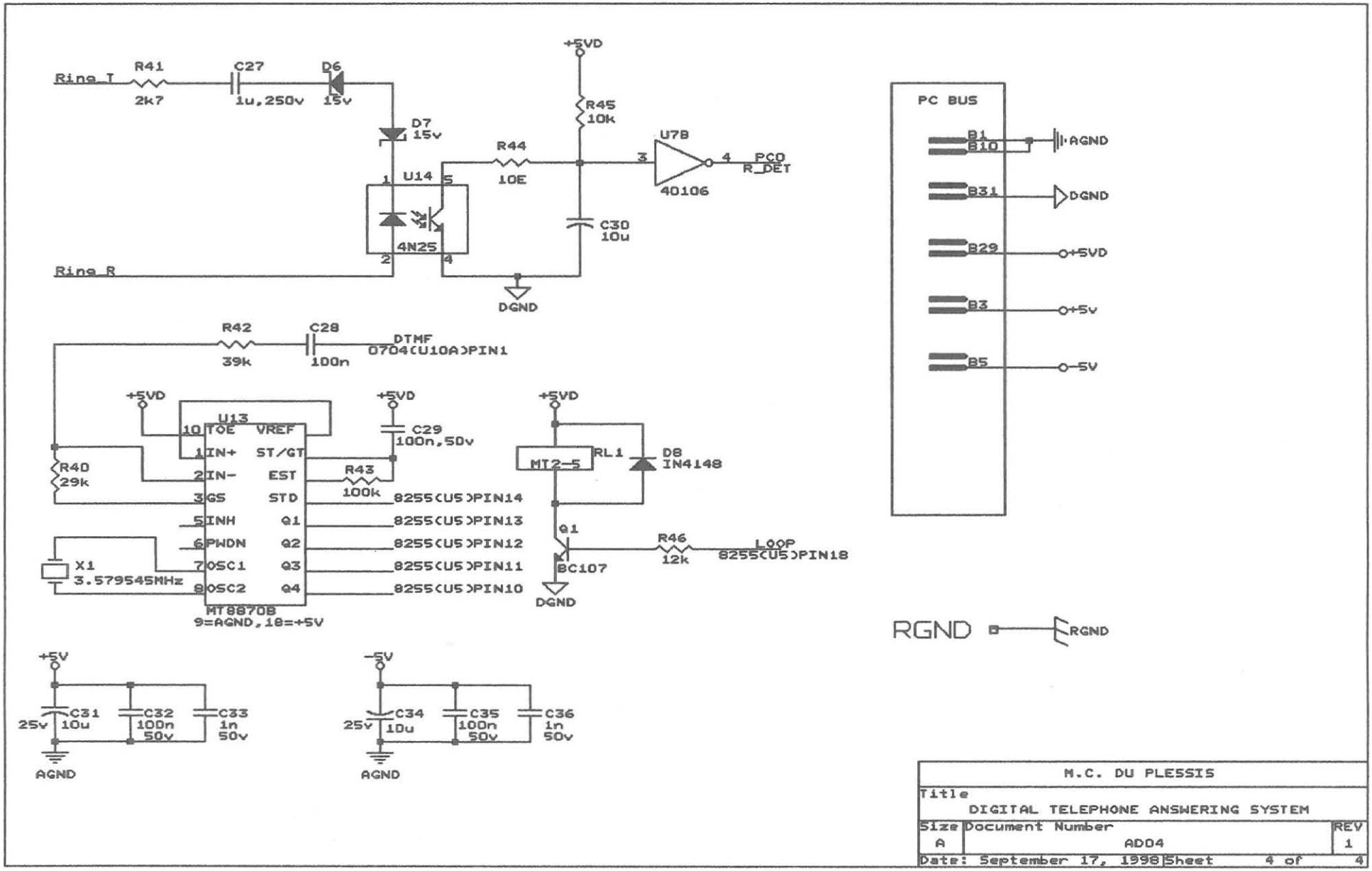
Central University of Technology, Free State

APPENDIX A: SYSTEM CIRCUIT DIAGRAMS









Central University of Technology, Free State

APPENDIX B1 - B2 : SYSTEM SOURCE CODE

APPENDIX B1 : MMI_MAIN PROGRAM

```
{ ************************************************************}
{*******SOFTWARE WRITTEN IN TURBO PASCAL FOR A DIGITAL*******}
{***************TELEPHONE ANSWERING SYSTEM****************}
{************************************************************}

{$A+,B-,D+,E+,F-,G+,I+,L+,N-,O-,R-,S+,V+,X+}
{$M 48000,0,655360}
{$Define  PRODUCTION}

PROGRAM MMI_MAIN;

Uses Crt, Dos, MMI_Core, MMI_Menu, MMIError, MMI_LBox, MMITools;

Var  OExitProc : Pointer;

Procedure MyExitProc;
Begin
    ShowError(ExitCode,1,'System shutdown : Runtime Error ','Error
    ' + IntStr(ExitCode) + ' : ' + ErrorList(ExitCode));
    While menun>0 do Restore;
    CleanupUser;
    Cleanupmail;
    CleanupSyst;
    ExitProc:=OExitProc;
    Halt(ExitCode);
end;


Procedure UserData;
var choice : byte;
begin
  BlueMenu;
    mil.head1:='Answering System';
    mil.head2:='User Menu';
    mil.itm[0]:='F1  Show   User Data  ';
    mil.hlp[0]:='Administrator ONLY!!! ';
    mil.itm[1]:='F2  Add    User Data  ';
    mil.hlp[1]:='Administrator ONLY!!! ';
    mil.itm[2]:='F3  Delete User Data  ';
    mil.hlp[2]:='Administrator ONLY!!! ';
    mil.itm[3]:='F4  Help              ';
    mil.hlp[3]:='Help on User Data options ';
    mil.foot:='ESC to exit';
    mil.n:=3;
    popup(22,9,52,18,White,Blue,White,Blue,wbox+wdouble+wshadow+wlight+
    whoriz,'',0,0,):
    choice:=0;
    repeat
```

```
        choice:=menu(choice,true,true);
        case choice of
          0 : ShowUserRecords(true);
          1 : AddUser;
          2 : DelUser;
          3 : ShowHelp('{UserHelp}');
        end;
    until choice=27;
    restore;
end;


Procedure MailData;
var choice : byte;
begin
    BlueMenu;
    mil.head1:='Answering System';
    mil.head2:='Mail Menu';
    mil.itm[0]:='F1  Play Mail Messages    ';
    mil.hlp[0]:='Play your Mail Messages';
    mil.itm[1]:='F2  Delete Mail Messages  ';
    mil.hlp[1]:='Delete Your Messages  ';
    mil.itm[2]:='F3  Record System Messages';
    mil.hlp[2]:='Administrator ONLY!!!  ';
    mil.itm[3]:='F4  Help                  ';
    mil.hlp[3]:='Help on mail options   ';
    mil.foot:='ESC to exit';
    mil.n:=3;
    popup(22,9,57,20,White,Blue,White,Blue,wbox+wdouble+wshadow+wlight+
          whoriz,'',0,0,1);
    choice:=0;
    repeat
       choice:=menu(choice,true,true);
       case choice of
         0 : ShowMailRecords;
         1 : DelMail;
         2 : MakeSystRec(IntStr(0));
         3 : ShowHelp('{MailHelp}');
       end;
    until choice=27;
    restore;
end;


Procedure SysSetup;
var choice : byte;
begin
    BlueMenu;
    mil.head1:='Answering System';
    mil.head2:='Setup Menu';
    mil.itm[0]:='F1  Show System Setup     ';
    mil.hlp[0]:='Administrator Only!!!  ';
    mil.itm[1]:='F2  Edit System Setup     ';
    mil.hlp[1]:='Administrator Only!!!  ';
    mil.itm[2]:='F3  Help                  ';
    mil.hlp[2]:='Help on System Setup options';
```

```
    mil.foot:='ESC to exit';
    mil.n:=2;
    popup(22,9,57,20,White,Blue,White,Blue,wbox+wdouble+wshadow+wlight+
          whoriz,'',0,0,1);
    choice:=0;
    repeat
       choice:=menu(choice,true,true);
       case choice of
         0 : ShowSystRecords(True);
         1 : EntrSetup;
         2:  ShowHelp('{SetupHelp}');
       end;
       Nosound;
    until choice=27;
    restore;
end;


Procedure MainMenu;
var choice : byte;
begin
  BlueMenu;
    mil.head1:='Telephone Answering System';
    mil.head2:='Main menu';
    mil.itm[0]:='F1  Mail  Menu  ';
    mil.hlp[0]:='Add/Show/Delete Messages';
    mil.itm[1]:='F2  User  Menu  ';
    mil.hlp[1]:='Add/Show/Delete Users  ';
    mil.itm[2]:='F3  Setup Menu  ';
    mil.hlp[2]:='System Setup Options  ';
    mil.itm[3]:='F4  Help  Menu  ';
    mil.hlp[3]:='Help on Menu Options  ';
    mil.foot:='ESC to exit';
    mil.n:=3;
    popup(25,7,55,16,White,Blue,White,Blue,wbox+wdouble+wshadow+wlight+
    whoriz,'',0,0,1);
    choice:=0;
    repeat
       choice:=menu(choice,true,true);
       case choice of
         0 : MailData;
         1 : UserData;
         2 : SysSetup;
         3 : ShowHelp('{MenuHelp}');
       end;
    until choice=27;
    restore;
end;


Procedure PasswordCheck;
Var i : byte;
begin
  popup(10,9,70,18,White,Blue,White,Blue,wshadow+wlight,' ACCESS CHECK
```

```
  ',Black,Green,0);
  Clrscr;
  write('Username : ');
  Readln(UserRecord.UserName);
  write('Password : ');
  Readln(UserRecord.Password);
  restore;
   if UserRecord.Username = SystData.SystAdmin then
   begin
      UserRecord.UserName:='Administrator';
      UserRecord.UserId:='0';
      UserRecord.Password:='';
       mainmenu;
   end
   else
   begin
    i:=0;
    while (UserPointer^[i].UserName<>UserRecord.UserName) and (i<15) do inc(i);
    if (i<15) and (UserPointer^[i].UserName<>'') then
        if UserPointer^[i].Password=UserRecord.Password then
      begin
        UserRecord:=UserPointer^[i];
        MainMenu;
      end;
   end;
end;


{************************************************}
{**************  Main Program  ******************}
{************************************************}


Begin
   PortBmem:=0;
   Port[PortB]:=PortBmem;
   DtmfHp:=0;
   DtmfTp:=0;
   DtmfFlag:=False;
   OExitProc:=ExitProc;
   ExitProc:=@MyExitProc;
   SwapVectors;
   Colour(White,Blue,0);
   ClrScr;
   Centre(1,'~T~elephone ~A~nswering ~S~ystem',Yellow,Blue);
   Writeat(60,1,'Version 1.0',0,0);
   Writeat(20,25,'~F10~ - Menu',Yellow,Blue);
   Writeat(45,25,'~Ctrl-F10~ - Exit',Yellow,Blue);
   Window(1,2,80,24);
   Colour(Black,LightGray,0);
   ClrScr;
   InitMenu;
   BlueMenu;
   SysError:=0;
   UserPointer:=nil;
   MailPointer:=nil;
```

```
    write('Initializing mail...');
    InitialiseMail;  {Do this first}
    if SysError<>0 then Shutdown;
    write('Initializing users...');
    InitialiseUser;  {Do this first}
    if SysError<>0 then Shutdown;
    write('Initializing system...');
    InitialiseSyst;  {Do this first}
    if SysError<>0 then Shutdown;
    ClearBuffer;
    HalfWay:=MaxBufLen div 2;
    Port[Control]:=$99;
    tbase:=80;
    t10ms:=0;
    t100ms:=0;
    Rec := False;
    Play := False;
    RingFlag := False;
    RingCnt := 0;

    {$IfDef PRODUCTION}
       writeln('Setting interrupt system...');
       IntEnable;  {(@IntSample);}
    {$EndIF}
     writeln;
     writeln('Ready...');
     Writeln;
     Repeat
       Repeat
         DtmfTp:=DtmfHp;
         TestLine;
       Until Keypressed;
       CursorX:=Wherex;
       CursorY:=Wherey;
       Ch:=GetKey;
       if Ch=_F10 then PasswordCheck;
       Gotoxy(CursorX,CursorY);
     Until Ch=Ctrl_F10;
     Window(1,1,80,25);
      Colour(lightgray,Black,0);
     Clrscr;
     Port[$2b1] := $00;
     {$IfDef PRODUCTION}
        IntDisable;
     {$EndIf}
     CleanupUser;
     Cleanupmail;
     CleanupSyst;
     Clrscr;
     ExitProc:=OExitProc;
     SwapVectors;
End.
```

APPENDIX B2 : MMI_CORE UNIT

```
Unit MMI_CORE;
{$X+}
Interface
Uses Dos, Crt, Printer, MMIError, MMITools, MMI_Menu, MMI_LBox;

  Const
    MaxBufLen = $8000; (* Same as 32768*)
    Control       = $2B3;
    PortA         = $2B0;
    PortB         = $2B1;
    PortC         = $2B2;
    RelayOpBit  = $01;
    RingAckBit  = $02;
    RingDetBit  = $02;
    ADC_RD_Bit = $80;
    DTMFDetBit    = $01;
    Stop          = #27;
    MailMax       = 512;
    MailMax1    = 513;

Type
 Arry = ARRAY[0..MaxBufLen-1] OF BYTE;
    Str  = String[20];
    String_type = string[80];

   MailRec = Record
      UserID      : String[1];
      FileName: String[12];
   end;

   UserRec = Record
      UserName   : String[16];
      UserID      : String[1];
      PassWord   : String[3];
   end;

   SystRec = Record
      SystAdmin  : String[16];
      RingCount  : String[1];
      RecTime    : String[3];
   end;

   CodeStr = String[3];

   MailData = Array[0..MailMax] of MailRec;
   MailDataPointer = ^MailData;

   UserData = Array[0..15] of UserRec;
   UserDataPointer = ^UserData;

Var
```

```
        DtmfBuff : Array[0..15] of byte;
        DtmfHp, DtmfTp : Byte;
        DtmfFlag : Boolean;
        PortBmem    : Byte;
        Sample      : Byte;
        HalfWay     : Word;
        OldVec      : Pointer;
        Buffer      : Arry;
        BufPtr        : Word;
        ByteCnt     : longint;
        RingCnt     : Word;
        OutFile     : File;       { Of Arry;}
        InFile      : File;
        OutName     : Str;
        InName      : Str;
        Code        : CodeStr;
        Ch          : Char;
        Rec,Play    : Boolean;
        RingFlag    : Boolean;
        PortBstatus : Byte;
        PortCstatus : Byte;
        CursorX     : Byte;
        CursorY     : Byte;
        SysError    : Byte;
        Tbase,t10ms,t100ms : byte;
        My10msDc, My10msUc, My100mst, My1sect : Integer;

        SystData    : SystRec;
        SysFile     : File of SystRec;

        MailPointer  : MailDataPointer;
        MailDataFile: File;

        UserPointer : UserDataPointer;
        UserDataFile: File;

        UserRecord : UserRec;

Function DTMF(Nchar:byte; Var TCode:CodeStr):byte;
Procedure PUTS (x,y:INTEGER; str:string_type);
Procedure ClearBuffer;
Procedure Enable_IRQx(IRQ : BYTE);
Procedure Disable_IRQx(IRQ : BYTE);
Procedure IntDisable;
Procedure IntEnable; {(Routine:Pointer);}
Procedure MakeSystRec(Uid:String);
Procedure MakeLineRec(Uid:String;RecLeng:Integer);
Procedure PlaySystRec(PlayFile:String);
Function PlayLineRec(PlayFile:String):char;
Procedure Setbit(Bit:byte);
Procedure Clear(Bit:byte);
Function TestB(Bit:Byte):Boolean;
Function TestC(Bit:Byte):Boolean;
Procedure timer(Time:Integer);
Procedure LoadMailDatabases;  {Only called when starting program}
Procedure LoadUserDatabases;  {Only called when starting program}
```

```
Procedure LoadSystDatabases;  {Only called when starting program}
Procedure SaveMailRecord(ID,Fname:String);
Procedure SaveUserRecord(ID,PassWord,UserName:String);
Procedure SaveSystRecord;
Procedure ShowMailRecords; {All record with ID}
Procedure ShowUserRecords(wait:boolean); {All record with ID}
Procedure ShowSystRecords(wait:boolean); {All record with ID}
Procedure DelMailRecord(FileName:String);
Procedure DelUserRecord(UserName:String);
Procedure ShutDown;
Procedure TestLine;

Function  InitialiseMail:Boolean;  {Do this first}
Function  InitialiseUser:Boolean;  {Do this first}
Function  InitialiseSyst:Boolean;  {Do this first}
Function  RandomFileName:String;
Function  GetUserName(UserId:String):String;
Function  GetDtmfChar:Char;


Procedure CleanUpMail;     {Do this last }
Procedure CleanupUser;     {Do this last }
Procedure EntrSetup;
Procedure AddUser;
Procedure DelUser;
Procedure DelMail;
Procedure ShowHelp(Context:string);
Procedure TogglePortBbits(Mask:byte);


IMPLEMENTATION

Procedure PUTS (x,y:INTEGER; str:string_type);
Begin
   Gotoxy(x,y);
   Write(str);
End;


Procedure ClearBuffer;
Begin
   RingCnt := 0;
   BufPtr:=0;
   FillChar(Buffer,SizeOf(Buffer),0);
End;


Procedure IntSample; Interrupt;
Begin
   Inline($FB);     (*STI*)
   if TestC(DtmfDetBit) then
   begin
      if not DtmfFlag then
      begin
         DtmfFlag:=True;
         DtmfBuff[DtmfHp]:=Port[PortC] shr 4;
```

```
            DtmfHp:=(DtmfHp+1) and $0F;
        end;
    end
    else
    begin
        DtmfFlag:=False;
    end;
    memw[$b000:0000]:=Bufptr;
    begin
        Dec(tbase);
        if tbase=0 then
        begin
            tBase:=80;
            dec(My10msDc);
            inc(My10msUc);
            inc(t10ms);
            if t10ms=10 then
            begin
                t10ms:=0;
                dec(my100mst);
                inc(t100ms);
                if t100ms=10 then
                begin
                    t100ms:=0;
                    dec(my1sect);
                end;
            end;
        end;
    end;
    if Rec then
    begin
        Buffer[BufPtr] := Port[PortA];          {Read data from port A }
        Inc(BufPtr);                            {Increment buffer pointer}
        BufPtr:=BufPtr and (MaxBufLen-1);       {Mask off MSb's if $0400 then $0000}
    end;
    if Play then
    begin
        Port[$2b0] := Buffer[BufPtr];           {Move data to port}
        Inc(BufPtr);                            {Increment buffer pointer}
        Dec(ByteCnt);                           {Count off bytes remaining}
        BufPtr:=BufPtr and (MaxBufLen-1);       {Mask off MSb's if $8000 then $0000}
    end;
    Inline($FA);    (*CLI*)                     {Enable interrupts      }
    Port[$20] := $20;                           {Signal end of interrupt  }
End;


Procedure Enable_IRQx(IRQ : BYTE);
Var
 imr, mask : Integer;

Begin
  mask   := not (1 shl IRQ);
  imr    := Port [$21];                         {get Interrupt mask register from 8259}
  imr    := imr and mask;                       {clear mask bit of IRQ  }
```

```
  Port[$21]:= imr;                          {and return to controler}
End;



Procedure Disable_IRQx(IRQ : BYTE);
Var
  imr,mask  : Integer;

Begin
    mask    := (1 shl IRQ);
    imr     := Port[$21];                   {get Interrupt mask register from 8259}
    imr     := imr or mask;                 {set mask bit of IRQ  }
    Port[$21]:= imr;
End;



Procedure IntDisable;
Begin
   disable_IRQx(5);
   SetIntVec($0d,OldVec);
End;



Procedure IntEnable; {(Routine:Pointer);}
Begin
  GetIntVec($0d,OldVec);
  SetIntVec($0d,@IntSample);
  Enable_IRQx(5);
End;



Procedure MakeSystRec(Uid:String);
Var SysMes:string[64];
   fPath : DirStr;
   fName : NameStr;
   fExt  : ExtStr;
   Begin
      if UserRecord.UserId<>'0' then exit;
      popup(10,9,70,18,White,Blue,White,Blue,
      wshadow+wlight,' RECORDING ',Black,Green,1);
      clrscr;
      PORT [Control] := $99;  {initialize 8255}
      TogglePortBbits($01);     {Operate line relay}
      clearBuffer;
      BufPtr:=1;
      OutName:=RandomFileName;
      Assign(OutFile,OutName);
      writeln;
      Writeln('Recording file : ',OutName);
      Writeln;
      Write('Press any key to stop...');
      {$i-} Rewrite(OutFile,1); {$i+}
```

```
        SysError:=IoResult;
        if SysError=0 then
        begin
           Rec := True;
           ClrKeyBuffer;
           Repeat
              if BufPtr = 0 then BlockWrite(OutFile,Buffer[Halfway],HalfWay);
              if BufPtr = HalfWay then BlockWrite(OutFile,Buffer[0],HalfWay);
           Until keypressed;
              if BufPtr<HalfWay then
                  BlockWrite(OutFile,Buffer[0],BufPtr)
              else
                  BlockWrite(OutFile,Buffer[HalfWay],BufPtr-HalfWay);
                 ClrKeyBuffer;
                 Rec := False;
                 Close(OutFile);
                 clrscr;
                 popup(10,9,70,18,White,Blue,White,Blue,
                 wshadow+wlight,' Save as ',Black,Green,0);
                 repeat
                    Writeln('Enter System Message Name... ');
                    Readln(SysMes);
                    Fsplit(SysMes,fPath,fName,fExt);
                    SysMes:=Strup(fName+'.MSG');
                    {$i-}
                    Rename(OutFile,SysMes);
                    {$i+}
                 until IoResult=0;
                 SaveMailRecord(Uid,SysMes);
                 Restore;
                 { IntDisable;}
        end
        else
        begin
           ShowError(SysError,1,'Gen Filename : Error ',ErrorList(SysError));
           {$i-}
           Close(OutFile);
           Erase(OutFile);
           {$i+}
        end;
        TogglePortBbits($01);
        restore;
        ClrKeyBuffer;
     End;



Procedure MakeLineRec(Uid:String;RecLeng:Integer);
Var MSGname:string;
Begin
   My100mst := RecLeng;
   {PORT [Control] := $99;  initialize 8255}
   {Port[PortB] := $01;     Operate line relay, See:Linetest Procedure}
   clearBuffer;
   BufPtr:=1;
```

```
    OutName:=RandomFileName;
    Assign(OutFile,OutName);
    {$i-} Rewrite(OutFile,1); {$i+}
    SysError:=IoResult;
    if SysError=0 then
       begin
          Rec := True;
          Repeat
             if BufPtr = 0 then BlockWrite(OutFile,Buffer[Halfway],HalfWay);
             if BufPtr = HalfWay then BlockWrite(OutFile,Buffer[0],HalfWay);
          Until (my100mst<=0);
           If BufPtr<HalfWay then
             BlockWrite(OutFile,Buffer[0],BufPtr)
           else
             BlockWrite(OutFile,Buffer[HalfWay],BufPtr-HalfWay);
             Rec := False;
             Writeln('MESSAGE FOR : ',
             PadRight(GetUserName(Uid),' ',16),'  ',SysDate,'  ',SysTime);
             Close(OutFile);
             SaveMailRecord(Uid,OutName);
       end
       else
       begin
            {$i-}
            Close(OutFile);
            Erase(OutFile);
            {$i+}
       end;
       {Port[Portb]:=00;}
       ClrKeyBuffer;
End;


Procedure PlaySystRec(PlayFile:String);
Var  BytesRead : word;
Begin
   popup(10,9,70,18,White,Blue,White,Blue,
   wshadow+wlight,' PLAYING ',Black,Green,1);
   Clrscr;
   Port[Control] := $89;   {Initialize 8255}
   TogglePortBbits($80);
   TogglePortBbits($01);
   Assign(InFile,PlayFile);
   InName := PlayFile;
   Writeln;
   Writeln('Playing file : ',PlayFile);
   Writeln;
   Write('Press any key to stop...');
   {$i-} Reset(InFile,1); {$i+}
   SysError:=IoResult;
   if SysError=0 then
      BlockRead(InFile,Buffer,MaxBufLen,BytesRead)
   else
      begin
         ShowError(SysError,1,'Open Play File : Error ',ErrorList(SysError));
```

```
          Restore;
        end;
    ByteCnt:=BytesRead;
    BufPtr:=1;
    Play := True;
    REPEAT
      if BufPtr = 0 then
      begin
        BlockRead(InFile,Buffer[HalfWay],HalfWay,BytesRead);
        inc(ByteCnt,BytesRead);
      end;
      if BufPtr = HalfWay then
      begin
        BlockRead(InFile,Buffer[0],HalfWay,BytesRead);
        inc(ByteCnt,BytesRead);
      end;
    UNTIL (ByteCnt<=0) or KeyPressed;
    Play := False;
    close(InFile);
    TogglePortBbits($80);
    TogglePortBbits($01);
    Restore;
    ClrKeyBuffer;
End;


Function PlayLineRec(PlayFile:String):char;
Var  BytesRead : word;
     plr : char;
Begin
    {Port[Control] := $89;   Initialize 8255}
    {Port[PortB] := $81;}
    Assign(InFile,PlayFile);
    InName := PlayFile;
    {Writeln('Playing file : ',PlayFile);}
    {$i-} Reset(InFile,1); {$i+}
    SysError:=IoResult;
    if SysError=0 then
       BlockRead(InFile,Buffer,MaxBufLen,BytesRead)
    else
    begin
       ShowError(SysError,1,'Open Play File : Error ',ErrorList(SysError));
       Restore;
       exit;
    end;
    ByteCnt:=BytesRead;
    BufPtr:=1;
    Play := True;
    REPEAT
       if BufPtr = 0 then
          begin
             BlockRead(InFile,Buffer[HalfWay],HalfWay,BytesRead);
             inc(ByteCnt,BytesRead);
          end;
       if BufPtr = HalfWay then
```

```
            begin
                BlockRead(InFile,Buffer[0],HalfWay,BytesRead);
                inc(ByteCnt,BytesRead);
            end;
        UNTIL (ByteCnt<=0) or (DtmfHp<>DtmfTp);
        PlayLineRec:=PLR;
        Play := False;
        close(InFile);
        ClrKeyBuffer;
End;


Procedure Setbit(Bit:byte);
Begin
  PortBstatus:=(PortBstatus or Bit);
  Port[PortB]:=PortBstatus;
End;


Procedure Clear(Bit:byte);
Begin
  PortBstatus:=PortBstatus and (not Bit);
  Port[PortB]:=PortBstatus;
End;


Function TestB(Bit:Byte):Boolean;
Begin
  TestB:=(PortBstatus and Bit) = Bit;
End;


Function TestC(Bit:Byte):Boolean;
Begin
  PortCstatus:= Port[PortC];
  TestC:=(PortCstatus and Bit) = Bit;
End;


Procedure timer(Time:Integer);
Begin
    Clrscr;
    My100mst:=(Time);
    repeat
    until (My100mst<=0) or KeyPressed;
End;


Function DTMF(Nchar:byte; Var TCode:CodeStr):byte;
Var Tone     : Byte;
    ToneFlag : Boolean;
    Timeout  : Boolean;
    TonePtr  : Word;
Begin
  ClrKeyBuffer;
  TCode:='';
```

```
  Timeout:=false;
  ToneFlag:=false;
  TonePtr :=0;
  my10msdc:=500;
  Repeat
    IF ToneFlag and not TestC(DTMFDetBit) then ToneFlag:=false;
    IF  TestC(DTMFDetBit)  and not ToneFlag THEN
    begin
       My10msDc:=500;
       {WriteLn('Waiting for DTMF signal....');}
       Tone := Port[PortC] shr 4;
       ToneFlag:=true;
       Case Tone of
          $1..$9   : Tone := Tone + $30; {1-9 = ASCII 49-57}
          $A    : Tone := $30;       {0 = ASCII 48}
          $B       : Tone := $2A;       {* = ASCII 42}
          $C       : Tone := $23;       {# = ASCII 35}
    end;
    Inc(TonePtr);
    Tcode[0] := chr(TonePtr);
    Tcode[TonePtr] := chr(tone);
    Timeout:=(My10msDc<=0);
  Until  Timeout or ((TonePtr=Nchar) and not ToneFlag);
  DTMF:=0;
  If timeout then DTMF:=1;
End;



Procedure LoadMailDatabases;  {Only called when starting program}
Begin
    writeln('Loading mail data...');
    Assign(MailDataFile,'MailData.DAT');
    {$i-} Reset(MailDataFile,1); {$i+}
    SysError:=IoResult;
    if SysError=0 then
       BlockRead(MailDataFile,MailPointer^[0],MailMax1*Sizeof(MailRec))
    else
    begin
       ShowError(SysError,1,'Loading Mail : Error ',ErrorList(SysError));
       Restore;
    end;
    Close(MailDataFile);
End;



Procedure LoadUserDatabases;  {Only called when starting program}
Begin
    writeln('Loading user data...');
    Assign(UserDataFile,'UserData.DAT');
    {$i-} Reset(UserDataFile,1); {$i+}
    SysError:=IoResult;
    if SysError=0 then
       BlockRead(UserDataFile,UserPointer^[0],16*Sizeof(UserRec))
```

```
    else
    begin
       ShowError(SysError,1,'Loading users : Error ',ErrorList(SysError));
       Restore;
    end;
    Close(UserDataFile);
End;


Procedure LoadSystDatabases;  {Only called when starting program}
Begin
    writeln('Loading system data...');
    Assign(SysFile,'SystData.DAT');
    {$i-} Reset(SysFile); {$i+}
    SysError:=IoResult;
    if SysError=0 then
       Read(SysFile,SystData)
    else
    begin
       ShowError(SysError,1,'Loading Mail : Error ',ErrorList(SysError));
       Restore;

    end;
    Close(SysFile);
End;



Procedure SaveMailRecord(ID,Fname:String);
Var i : word;
Begin
  i:=0;
    While (MailPointer^[i].UserID<>'') and (i<MailMax) do inc(i); {Scan for blank
record}
    if i>=MailMax then
       Writeln('Mail File Is Full')
    else
    Begin
       MailPointer^[i].UserID:=ID;
       MailPointer^[i].FileName:=Fname;
       Reset(MailDataFile,1);
       BlockWrite(MailDataFile,MailPointer^[0],MailMax1*Sizeof(MailRec));
       Close(MailDataFile);
    end;
End;



Procedure SaveUserRecord(ID,PassWord,UserName:String);
```

```
Var i : word;
Begin
  i:=0;
   While (UserPointer^[i].UserID<>'') and (i<15) do inc(i); {Scan for blank record}
   if i>=16 then
      Writeln('User File Is Full')
   else
   begin
      UserPointer^[i].UserID:=ID;
      UserPointer^[i].UserName:=UserName;
      UserPointer^[i].PassWord:=PassWord;
           Reset(UserDataFile,1);
      BlockWrite(UserDataFile,UserPointer^[0],16*Sizeof(UserRec));
           Close(UserDataFile);
   end;
End;



Procedure SaveSystRecord;
Var i : word;
begin
  Rewrite(SysFile);
  Write(SysFile,SystData);
  Close(SysFile);
end;



Procedure ShowMailRecords; {All records with ID}
Var i : word;
   ch : Char;
   Selfile,fname,uname : string;
   s : SearchRec;
   id : string[3];
Begin
   popup(10,9,70,18,White,Blue,White,Blue,Wshadow+wlight,
        ' PLAY MAIL MESSAGES ',Black,Green,1);
   ClrScr;

   id:=UserRecord.UserId;
   Init_LBox(black,lightgray,Black,lightgray,White,Blue);
   i:=0;
   While (i<MailMax) and (MailPointer^[i].UserId<>'') do
   begin
      if (MailPointer^[i].UserId=id) or (UserRecord.UserId='0') then
      begin
         FindFirst(MailPointer^[i].Filename,AnyFile,s);
         If Doserror=0 then
            Add_lbox(Fdate(s.time)+'  '+Ftime(s.time)+'  '+s.Name)
         else
            Add_LBox(MailPointer^[i].Filename+'  Not found!!!');
      end;
      inc(i);
   end;
   Selfile:=do_lbox(15,5,55,20,'Filenames');
```

```
    if SelFile<>'' then
    begin
        Writeln(Selfile,' : Selected');
        Writeln(Copy(SelFile,21,12),' : Played...');
        PlaySystRec(Copy(SelFile,21,12));
        ch:=GetKey;
    end;
    Restore;
    PortBmem:=0;
    Port[PortB]:=PortBmem;          {Release line Relay}
End;



Procedure ShowUserRecords(wait:boolean); {All records with ID's}
Var i : word;
   ch : Char;
Begin
  if UserRecord.UserId<>'0' then exit;
  if wait then popup(10,9,70,18,White,Blue,White,Blue,Wshadow+wlight,
                ' DISPLAY USER DATA ',Black,Green,1);
  ClrScr;
   i:=0;
      Writeln(PadRight('Usernames',' ',20),' ', 'UserID':3,'     ','Password':4);
      While UserPointer^[i].UserID<>'' do
      begin
         Writeln(PadRight(UserPointer^[i].UserName ,' ',20),'
',UserPointer^[i].UserID:3,'          ',
         UserPointer^[i].Password:4);
         inc(i);
      end;
      ClrKeyBuffer;
      if wait then
      begin
         Write('Press any key to continue...');
         ch:=GetKey;
         Restore;
      end;
End;



Procedure ShowSystRecords(wait:boolean); {All record with ID}
Var i : word;
   ch : Char;
Begin
   if UserRecord.UserId<>'0' then exit;
   if wait then popup(10,9,70,18,White,Blue,White,Blue,Wshadow+wlight,
                      ' DISPLAY SYSTEM DATA ',Black,Green,1);
   ClrScr;
   i:=0;
   Writeln(PadRight('Admin_Password',' ',20),'  ',
                          'Ring_Count':3,'  ',
                          'Recording_Time');
```

```
    Writeln(PadRight(SystData.SystAdmin ,' ',20),'    ',
       SystData.RingCount:3,'          ',
       SystData.RecTime);
    ClrKeyBuffer;
    if wait then
    begin
       Write('Press any key to continue...');
       ch:=GetKey;
       Restore;
    end;
End;



Procedure DelMailRecord(FileName:String);
Var i : Word;
   TfileName : String;
Begin
  TFileName:=Copy(Filename,1,8);
  i:=0;
   while (Pos(TFileName,MailPointer^[i].FileName)=0) and
(MailPointer^[i].FileName<>'') and
          (i<MailMax) do inc(i);
   if Pos(TfileName,MailPointer^[i].FileName)=1 then
      Move(MailPointer^[i+1],MailPointer^[i],(MailMax1-i)*Sizeof(MailRec));
          { Source          , Dest          , Number of Bytes}
      Reset(MailDataFile,1);
   BlockWrite(MailDataFile,MailPointer^[0],(MailMax1)*Sizeof(MailRec));
      Close(MailDataFile);
End;



Procedure DelUserRecord(UserName:String);
Var i : Word;
Begin
  i:=0;
   while (UserPointer^[i].UserName<>UserName) and (i<15) do inc(i);
   if i<MailMax then
      Move(UserPointer^[i+1],UserPointer^[i],(15-i)*Sizeof(UserRec));
          { Source          , Dest          , Number of Bytes}
      Reset(UserDataFile,1);
   BlockWrite(UserDataFile,UserPointer^[0],16*Sizeof(UserRec));
      Close(UserDataFile);
End;



Function InitialiseMail:boolean;  {Do this first}
Var i : Word;
Begin
   InitialiseMail:=False;
   Getmem(MailPointer,(MailMax1)*Sizeof(MailRec)); {Get memory for data}
   if MailPointer<>nil then
```

```
    begin
      if not Exist('MailData.DAT') then    {File has not been created yet}
      begin
        for i:=0 to MailMax do             {clear all records}
        begin
          MailPointer^[i].UserID:='';
          MailPointer^[i].FileName:='';
        end;
        Assign(MailDataFile,'MailData.DAT');   {Write data to file}
        Rewrite(MailDataFile,1);
        Blockwrite(MailDataFile,MailPointer^[0],(MailMax1)*Sizeof(MailRec));
        Close(MailDataFile);
      end;
      LoadmailDatabases;    {Normal load of databases}
      writeln;
      if SysError=0 then
        InitialiseMail:=true;
    end
    else
      writeln('Mail initialisation failed...');
End;



Function InitialiseUser:Boolean;  {Do this first}
Var i : Word;
Begin
    InitialiseUSer:=False;
    Getmem(UserPointer,16*Sizeof(UserRec)); {Get memory for data}
    if UserPointer<>nil then
    begin
      if not Exist('UserData.DAT') then    {File has not been created yet}
      begin
        for i:=0 to 15 do             {clear all records}
        begin
          UserPointer^[i].UserName:='';
          UserPointer^[i].UserID   :='';
          UserPointer^[i].Password:='';
        end;
        Assign(UserDataFile,'UserData.DAT');   {Write data to file}
        Rewrite(UserDataFile,1);
        Blockwrite(UserDataFile,UserPointer^[0],16*Sizeof(UserRec));
        Close(UserDataFile);
      end;
      LoadUserDatabases;    {Normal load of databases}
      writeln;
      if SysError=0 then
        InitialiseUser:=true;
    end
    else
      writeln('User initialisation failed...');
End;


Function InitialiseSyst:Boolean;  {Do this first}
Var i : Word;
```

```
Begin
   InitialiseSyst:=False;
   if not Exist('SystData.DAT') then    {File has not been created yet}
   begin
         SystData.SystAdmin:='admin';
         SystData.RingCount:='3';
         SystData.RecTime  :='10';
         Assign(SysFile,'SystData.DAT');   {Write data to file}
         Rewrite(SysFile);
         write(SysFile,SystData);
         Close(SysFile);
   end;
   LoadSystDatabases;    {Normal load of databases}
   writeln;
   if SysError=0 then
   InitialiseSyst:=true;
End;



Procedure CleanupUser; {Do this last}
Begin
   Freemem(UserPointer,16*Sizeof(UserRec));   {Give memory back to DOS}
End;



Procedure CleanUpMail; {Do this last}
Begin
   Freemem(MailPointer,(MailMax1)*Sizeof(MailRec));   {Give memory back to DOS}
End;



Procedure AddUser;
Var User: UserRec;
Begin
 if UserRecord.UserId<>'0' then exit;
 popup(10,9,70,18,White,Blue,White,Blue,wshadow+wlight,' Add User Data
',Black,Green,1);
 ShowUserRecords(false);
 Writeln('');
 With User do
 begin
   Repeat
       Write('Enter User Name (Max 20 Char):');
       Readln(UserName);
       Write('Enter User ID (1-9):');
       Readln(UserID);
       Write('Enter Password (Max 3 Digits):');
       Readln(Password);
       Writeln;
    until (UserName<>'') and (UserID<>'') and (Password<>'');
    SaveUserRecord(UserID,Password,UserName);
end;
Restore;
```

```
End;


Procedure EntrSetup;
var OldSys : String;
Begin
  if UserRecord.UserId<>'0' then exit;
  popup(10,9,70,18,White,Blue,White,Blue,wshadow+wlight,' System Setup
',Black,Green,1);
  ShowSystRecords(false);
  Writeln('');
  With SystData do
  begin
    Repeat
       Writeln('Enter NEW Administrator''s Username/Password');
       Write('(Max 16 Char):');
       OldSys:=SystAdmin;
       Readln(SystAdmin);
          if SystAdmin='' then SystAdmin:=Oldsys;
       Write('Ring Count (3). [1-9]:');
       Readln(RingCount);
          if RingCount='' then RingCount:='3';
       Write('Recording time (10sec) [10-30]:');
       Readln(RecTime);
       if RecTime='' then RecTime:='10';
       Writeln;
    Until (RingCount<>'') and (RecTime<>'') and (SystAdmin<>'');
    SaveSystRecord;
 end;
 Restore;
End;



Procedure DelMail;
Var i : word;
    Tfile :File;
    ch : Char;
    Selfile,fname,uname : string;
    s : SearchRec;
    id : string[3];
Begin
    popup(10,9,70,18,White,Blue,White,Blue,Wshadow+wlight,
         ' DELETE MAIL MESSAGES ',Black,Green,1);
    ClrScr;
    id:=UserRecord.UserId;
    Init_LBox(black,lightgray,Black,lightgray,White,Blue);
    i:=0;
    While (i<MailMax) and (MailPointer^[i].UserId<>'') do
    begin
       if (MailPointer^[i].UserId=id) or (UserRecord.UserId='0') then
       begin
          FindFirst(MailPointer^[i].Filename,AnyFile,s);
          if Doserror=0 then
             Add_lbox(Fdate(s.time)+'  '+Ftime(s.time)+'  '+s.Name)
          else
```

```
                Add_LBox(MailPointer^[i].Filename+'  Not found!!!');
        end;
        inc(i);
    end;
    Selfile:=do_lbox(15,5,55,20,'Select Filename to DELETE');
    if SelFile<>'' then
    begin
        Writeln(Selfile,' : Selected');
        DelMailRecord(Copy(SelFile,21,12));
        assign(Tfile,copy(SelFile,21,12));
        erase(TFile);
        Writeln(Copy(SelFile,21,12),' : Deleted...');
        ch:=GetKey;
    end;
    Restore;
End;



Procedure DelUser;
Var UserName: String;
Begin
  if UserRecord.UserId<>'0' then exit;
  popup(10,9,70,18,White,Blue,White,Blue,wshadow+wlight,' Delete USER DATA
',Black,Green,1);
  ShowUserRecords(false);
  begin
    Writeln('Enter User Name to Delete:');
    Readln(UserName);
    DelUserRecord(UserName);
  end;
  Restore;
End;



Function RandomFileName:String;
Var RFN : String;
Begin
   Repeat
    Rfn:=PadLeft(IntStr(Random(65535))+'.msg','0',12);
    until not exist(rfn);
    RandomFileName:=Rfn;
End;



Procedure ShutDown;
Begin
    if MailPointer<>nil then Freemem(MailPointer,MailMax1*Sizeof(MailRec));   {Give
    memory back to DOS}
    if UserPointer<>nil then Freemem(UserPointer,16*Sizeof(UserRec));   {Give
    memory back to DOS}
    Writeln('System shutdown caused by : ',Errorlist(SysError));
    halt(0);
    Restore;
```

```
End;



Procedure PlayMessages;
Var  i : Byte;
     j,k : Byte;
     id: string[3];
     Password:string;
     DtmfDetect : Boolean;

Begin
    Code:='';
    for i:=1 to 3 do Code:=Code+GetDtmfChar;
    Password:= code;
    {Writeln(code);   eg.#111}
     i:=0;
     while (code<>UserPointer^[i].Password) and (i<15) do inc(i);
     if UserPointer^[i].Password=Code then
    begin
        id:=UserPointer^[i].userid;
        j:=0;
        K:=0;
        while (MailPointer^[j].UserId<>'') and (DtmfHp=DtmfTp) do
        begin
            If mailpointer^[j].userid  =  id then
            begin
                inc(k);
                PlayLineRec(MailPointer^[j].FileName);
                if DtmfTp<>DtmfHp then
                if DtmfBuff[DtmfTp]<>$0C then DtmfTp:=DtmfHp;
                PlayLineRec('Beep.Msg');
            end;
            inc(j);
        end;
        While DtmfHp<>DtmfTp do GetDtmfchar;
        if K=0 then
            PlayLineRec('NoMsg.msg')
        else
            PlayLineRec('NoMore.msg');
    end;
    PortBmem:=0;
    Port[PortB]:=PortBmem;     {Release Line Relay}
End;



Procedure TestLine;
Var RingCount:Integer;
     Timeout  : Boolean;
     i          : integer;
     ch       : Char;

Begin
    If NOT TestC(RingDetBit) Then Exit;
    RingCount := 0;
```

```
    Repeat
       Inc(RingCount);
       My100mst := 45;  {4.5 sec ring cycle timeout }
       My10msDc := 150;  {1.5 sec ring timeout }
    {$IfDef PRODUCTION}
       Repeat Until (Not TestC(RingDetBit)) or (my10msDc=0);
    {$Else}
       Repeat Until (Not TestC(RingDetBit)) or Keypressed;
       If keypressed then exit;
    {$EndIf}
       TimeOut:=(My100mst<=0);
       Repeat Until TestC(RingDetBit) or TimeOut;
    Until (RingCount=StrInt(SystData.RingCount)) or TimeOut;
    If TimeOut Then EXIT;
    PORT [Control] := $89;  {initialize 8255 for Play}
    TogglePortBbits($80);
    TogglePortBbits($01);    {8=D-A on for play and 1=Opp line Relay}
    PlayLineRec('Greeting.msg');
    if DtmfHp=DtmfTp then PlayLineRec('Menu.msg');
    If DtmfHp<>DtmfTp Then   {Valid code received?, code represents ID}
     begin
       ch:=GetDtmfChar;
       if ch='#' then
          PlayMessages
       else
       begin
          Code:='';
          Code:=Code+ch;
          PlayLineRec('Beep.msg');
          PORT [Control] := $99;  {initialize 8255 for Recording}
          TogglePortBbits($80);
          MakeLineRec(Code,((StrInt(SystData.RecTime))*10));
       end
     end
    else
    begin
       PlayLineRec('Beep.msg');
       PORT [Control] := $99;  {initialize 8255 for Recording}
       TogglePortBbits($80);
       MakeLineRec('0',((StrInt(SystData.RecTime))*10));
    end;
    PortBmem:=0;
    Port[PortB]:=PortBmem;    {Reset port memory}
End;



Function  GetUserName(UserId:String):String;
Var i : Byte;
Begin
  i:=0;
   While (i<16) and (UserPointer^[i].UserId<>UserId) do inc(i);
  GetUserName:=UserPointer^[i].UserName;
End;
```

```
Procedure ShowHelp(Context:string);
var Hfile : Text;
    Line : String;
     Key : char;
Begin
    Assign(Hfile,'MMI_HELP.txt');
    Assign(HFile,'MMI_HELP.txt');
    {$i-} Reset(HFile); {$i+}
    SysError:=IoResult;
    if SysError=0 then
      begin
        reset(Hfile);
        popup(5,4,75,21,White,Blue,White,Blue,wshadow+wlight,' HELP
',Black,Green,1);
            clrscr;
        repeat
            Readln(hfile,line);
         until Pos(Context,line)>0;
         Readln(hfile,line);
         repeat
            writeln(line);
            readln(hfile,line);
         until pos('{END}',line)>0;
        write('Press any key to return...');
      end
    else
    begin
        ShowError(SysError,1,'Loading Mail : Error ',ErrorList(SysError));
    end;
    close(hfile);
    Key := Getkey;
    ClrKeyBuffer;
    restore;
End;


Procedure TogglePortBbits(Mask:byte);
Begin
    PortBmem:=PortBmem Xor Mask;
    Port[PortB]:=PortBmem;
End;


Function GetDtmfChar:Char;
Var Tone : Byte;
Begin
    My10MsDc:=500;
    While (DtmfHp=DtmfTP) and (My10MsDc>0) do;
    if My10MsDc=0 then
    begin
        GetDtmfChar:=#0;
        Exit;
```

```
    end;
    if DtmfHp = DtmfTp then
        GetDtmfChar: = #0
    else
    begin
        Tone: = DtmfBuff[DtmfTp];
        Case Tone of
            $1..$9 : Tone : = Tone + $30; {1-9 = ASCII 49-57}
            $A     : Tone : = $30;        {0 = ASCII 48}
            $B     : Tone : = $2A;        {* = ASCII 42}
            $C     : Tone : = $23;        {# = ASCII 35}
        end;
        GetDtmfChar: = chr(Tone);
        {Write(Chr(tone));}
        inc(DtmfTp);
        DtmfTp: = DtmfTp and $0F;
    end
End;

End.
```