1-31-1991

Integrated digital speech system with PC

Zhen Zhu
New Jersey Institute of Technology

Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

Abstract

Title of Thesis :
Integrated Digital Speech System with PC

Name :
Zhen Zhu
Master of Science in Electrical Engineering
Department of Electrical and Computer Engineering

Thesis directed by :
Dr. Chung H. Lu
Associate Professor
Department of Electrical and Computer Engineering

A PC-based real-time digital speech processing system is designed. Speech synthesis, analysis, storage and retrieval are implemented in the system with intelligent telephone functions. Speech data are stored on floppy disk and easy to retrieve. Though floppy disk is slow in speed, real-time processing is achieved with low bit rate coding. The computer gets speech data from a digital speech processing unit or send data to it through a synchronous serial communication port. The analog input of digital speech processing unit can be selected from the telephone line or the microphone, and output of the digital speech processing unit can be sent to the earphone or telephone line. The software resides in the PC RAM. It is activated by hot-key combination or the ring detect signal. The possibility of speedy playback without change of pitch and intonation is also discussed in this thesis.

Integrated Digital Speech System With PC

BY
Zhen Zhu

Thesis submitted to the faculty of the graduate school of
the New Jersey Institute of Technology
in partial fulfillment of the requirements for the degree of
Master of Science in Electrical Engineering

1991

APPROVAL SHEET

Title of Thesis: Integrated Digital Speech System
with PC

Candidate: Zhen Zhu
Master of Science in Electrical Engineering, 1991

Thesis and Abstract Approved by the Examining Committee:

Dr. Chung H. Lu, Advisor / Date
Associate Professor
Department of Electrical and Computer Engineerig

Dr. Anthony Robbi / Date
Associate Professor
Department of Electrical and Computer Engineerig

Dr. Sol Rosenstark / Date
Professor
Department of Electrical and Computer Engineerig

New Jersey Institute of Technology, Newark, New Jersey.

VITA

Name : Zhen Zhu

Present address :

Date of Birth :

Place of Birth :

Education :

1. New Jersey Institute of Technology
   Electrical Engineering Department, M.S. E.E.
   January, 1989 - January, 1991

2. Luwan College of Technology
   Electrical Engineering Department, B.S. E.E.
   September, 1978 - June, 1982

ACKNOWLEDGEMENTS

The author wishes to express his sincere gratitude to Dr. Chung H. Lu, his thesis advisor, for his valuable guidance, support, inspiration and encouragement during the entire course of this thesis. Appreciation is also extended to other committee members, Dr. Anthony Dr. Robbi and Dr. Rosenstark, for their support and serving on the examination committee.

Contents

1 Introduction 1

I. Introduction 1

II. General Description of the System 3

2 Principal Digital Speech Technology 6

I. Introduction to LPC 6

II. Realization of Linear Predictive Coding in System 12

3 PC XT Interfacing Circuitry 22

I. Introduction 22

II. DTMF Transceiver 24

III. Telephone Line Interface 26

IV. 2-4 Converter 28

4 Software for IBM PC XT 31

I. Introduction to Terminate-and-Stay-Resident Program 31

II. System Functions 33

5 Conclusion and Suggested Future Work 42

A Circuit Diagrams 44

B Software for IBM PC XT 46

List of Figures

1.1 System Overview 4

2.1 The Source-Filter Model 7
2.2 Curve Fitting 10
2.3 Linear Prediction used in the System 12
2.4 Block Diagram of DSPU 13
2.5 Companding Curve of the μ-law Compander 14
2.6 Block Diagram of μPD7720 16

3.1 Block Diagram of PC Interface 23
3.2 Analog Audio Signal Flowchart 24
3.3 Schematic of Telephone Line Interface 26
3.4 Schematic of 2-4 Converter 28

4.1 Flowchart of the Program for PC 34
4.2 Menu Structure 35
4.3 Telephone Call Progress 36

List of Tables

2.1 Bit requirements for each parameter in LPC-10 11

3.1 DTMF Coding and 75T2090 Digit In/Out 25

Chapter 1

Introduction

I. Introduction

Speech is an everyday, essential communication medium. It also has advantages over other means of communication. You can listen while shaving, doing the housework, or driving the car. Speech leaves hands and eyes free for other tasks. The most important convenience feature of speech communication is the telephone, and this gives people the advantage of being able to communicate over long distances. You can go into a phone booth anywhere in the world, carrying no special equipment, and transmit your message to a machine or a person by either voice or keystrokes. People are always trying to improve this communication medium for more convenience and efficiency.

With fast development of computer technology, machines including the computer itself are more intelligent and they can do more and more tasks for people in their daily lives. More and more work is done by a machine rather than a person, therefore irrespective of whether they like it or not, people have to 'talk' with a machine more than they did before. As a result, practical applications of speech signal processing based on speech signal analysis, synthesis, recognition, transmission and recording

have attracted much more attention than before.

Meanwhile, the Personal Computer (PC), since its introduction in the late 1970s, has become an integral part of modern daily life. With a steady reduction in price, the PC has made substantial inroads into business offices, classrooms and private residences. Researchers say some 28 million US workers use personal computers in their offices, and that some 8.2 million more PCs are used in schools. Along with this development, the home computer market has grown to 3.7 million units annually. Computer literacy is an order of magnitude higher than it was five years ago. All these developments are accomplished with a succession of quantum jumps in computing power. Today's PC which can be the size of a briefcase, in comparison with the room size mainframe in its infancy, contains greater computing power, larger memory space and faster speed, and most importantly, it is easier to use, yet does not cost more. Needless to say, speech processing with a PC will let more people get benefits and conveniences from this advanced technology.

The purpose of this project is to take advantage of digital speech processing technology, telephony technology, and the phenomenal growth of personal computers. A system that uses a PC was built. This system compresses live speech signals to 2400 bits per second. The speech data can be stored and retrieved in real time on a floppy disk. A 360k bytes floppy disk can store up to 20 minutes speech data. A 1.2M bytes floppy disk can store more than one hour of speech data.

Fully integrated, the system functions as a powerful, intelligent, digital speech system. All features of telephone functions (including telephone usage accounting), messaging systems, transcription systems, and various other automation functions

can be implemented.

II. General Description of the System

The system hardware contains a digital speech processing unit (DSPU), interfacing circuitry, a regular household telephone set and a personal computer. The initial phase of the project uses an IBM PC/XT. In the future, with minor modifications, the whole system can be ported to any other PCs, such as IBM PC/AT, IBM PS/1, and MacIntosh. Figure 1.1 shows the system overview.

The system software mainly consists of two parts: one in the PC, and the other in the DSPU. The program for PC terminates and stays resident after it is executed on the command line and can be invoked by a hot key stroke, timer or ring detector. After it has been loaded, the user can still communicate with it by executing the same program to suspend, restart, or unload itself. This program does tasks such as reading and writing a diskfile, editing the user's phone directory, dialing, checking ID, automatically making a call or answering a call. The function of the DSPU is to convert and compress the speech signal to a low bit rate (2400 bps) digital data. In 1976, the US Government defined a standard coding scheme called LPC-10 for a 10-pole prediction with a data rate of 2400 bit/s. Today, many systems can achieve even lower bit rates, and are simpler to operate and are more flexible. In this project, we chose the VDT2400 developed by UNIDATA, with necessary modifications. It consists of a speech analyzer, a synthesizer, and a pitch extractor.

The function of the interface is twofold. First, it gets the digital speech data from the DSPU and then writes them on a floppy disk. Second, it feeds the DSPU with dig-

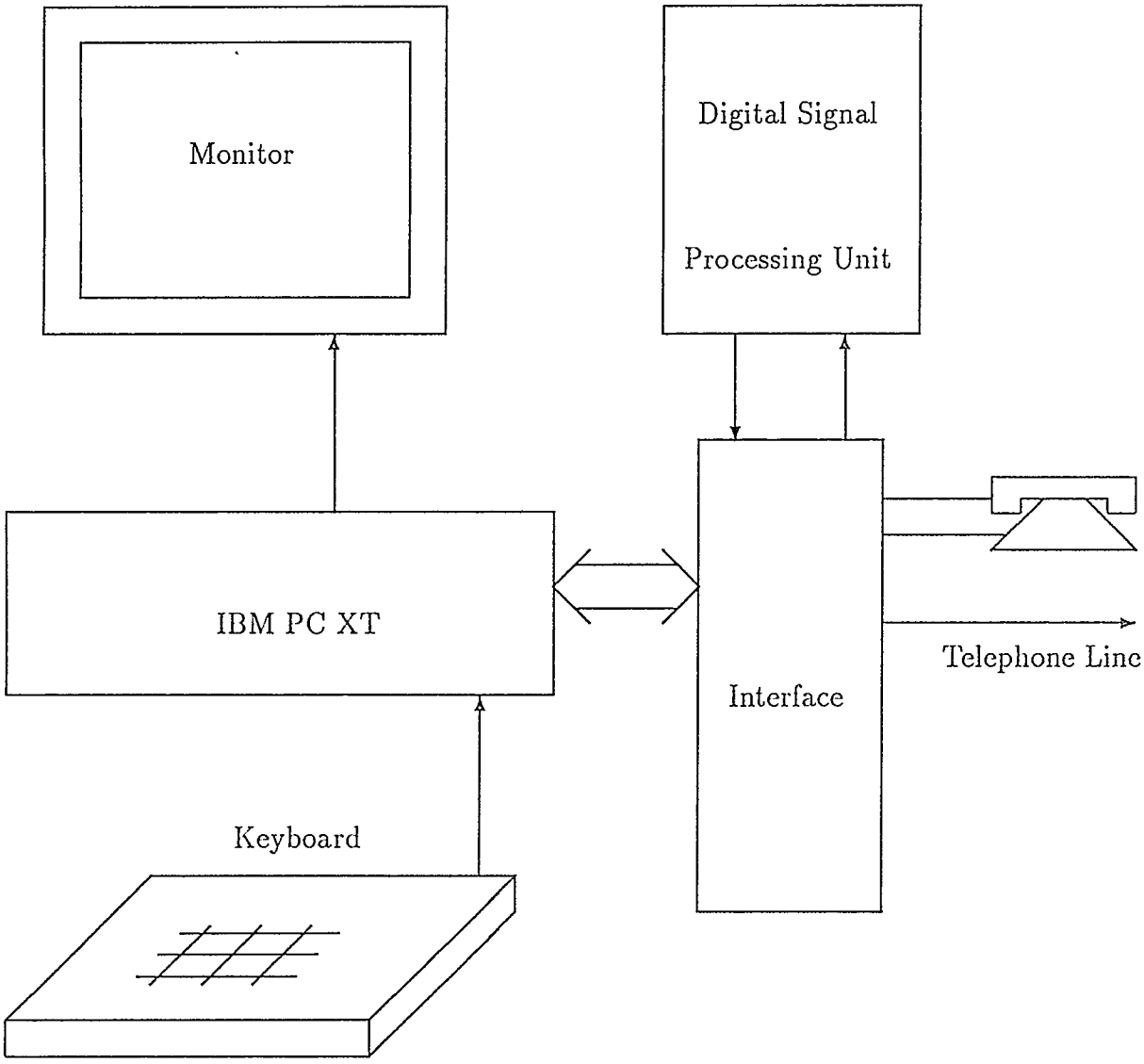


Figure 1.1: System Overview

ital speech data from the floppy disk and then outputs an analog speech signal. Both functions are performed in real time. The hardware also contains a 2-4 converter, a DTMF (dual tone multiple frequency) transmitter and receiver, interrupt logic, one input port and one output port. The basic functions of the system are to pick up a message recorded previously, to answer calls and to record incoming messages. It can also interactively prompt users for options.

Chapter 2

Principal Digital Speech Technology

I. Introduction to LPC

Human speech is produced by the combined action of the vocal cords and the vocal tract, which consists of the throat, mouth, and nose resonance cavities. The three different sound types can be classified as:

- voiced
- unvoiced (fricative)
- unvoiced (aspirated)

Vocal cord vibration rate (which determines the pitch) varies from 60-70 Hz for the lowest male voice to 1200-1300 Hz for the upper limit of a soprano. Typically, a female voice approximates 400 Hz for a voiced sound, a male voice hovers between 80 and 200 Hz. Predominant frequencies in unvoiced sounds are above 2000 Hz.

The speech can be produced by a source-filter model which indicates the features of the human voice and simulates human vocal tract effects with filters, as shown in

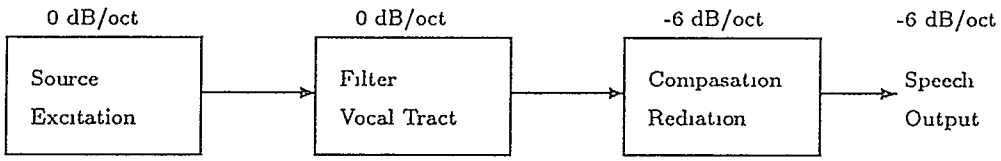


Figure 2.1: The Source-Filter Model

Figure 2.1. The excitation source with a flat spectrum and the radiation compensation with 6 dB/octave fall-off are often preferred to give the required trend in the output spectrum. This idea is useful for Speech Linear Predictive Coding.

Digital speech processing can be made much more intelligent than analog processing. For example, it is difficult to provide rapid access to messages stored in analog form. But if a directly digitized audio waveform is used for transmission and storage, the data rate is very high. Suitable coding can reduce the data-rate of speech to as little as one hundredth of that needed by direct digitization. For many coding method, speech parameters like intonation and amplitude are separated out from the articulation of the speech and stored, or transmitted, instead of simply the digitized waveform.

Linear Predictive Coding is a relatively new method of speech analysis-synthesis. It is primarily a time-domain coding method but can be adapted for frequency-domain parameters like formant frequency, bandwidth and amplitude. We briefly introduce it below.

A speech sample $x(n)$ can be predicted quite closely by the previous samples improved by multiplying the previous samples by a set of numbers, say $a_1, \ldots, a_k$,

which are adapted on a syllabic time-scale. The prediction error could be written as

$$\begin{aligned} e(n) &= x(n) - a_1 x(n-1) - a_2 x(n-2) - \cdots - a_p x(n-p) \\ &= x(n) - \sum_{k=1}^{p} a_k x(n-k) \end{aligned}$$

The multipliers a_k should be adapted to minimize the error signal. It turns out that they must be re-calculated and transmitted on a time-scale that is rather faster than syllabic but much slower than the basic sampling rate: intervals of 10-25 msec are usually used (compare this with the 125 μsec sampling rate for telephone-quality speech). Turning the above relationship into z-transforms gives

$$E(z) = X(z) - \sum_{k=1}^{p} a_k z^{-k} x(z) = (1 - \sum_{k=1}^{p} a_k z^{-k}) X(z)$$

Rewriting the speech signal in terms of the error,

$$X(z) = \frac{1}{1 - \sum_{k=1}^{p} a_k z^{-k}} E(z)$$

Speech can be viewed as an excitation source passing through a vocal tract filter, followed by another filter to model the effect of radiation from the lips as mentioned before. The overall spectral levels can be reassigned as shown in Figure 2.1, so that the excitation source has a 0dB/octave spectral profile, and hence is essentially impulsive. Considering the vocal tract filter as a serial connection of digital formant filters, its transfer function is the product of terms like

$$\frac{1}{1 - b_1 z^{-1} + b_2 z^{-2}}$$

where b_1 and b_2 control the position and bandwidth of the formant resonances. The -6 dB/octave spectral compensation can be modeled by the first-order digital filter

$$\frac{1}{1 - bz^{-1}}$$

The product of all these terms, when multiplied out, will have the form

$$\frac{1}{1 - c_1 z^{-1} - c_2 z^{-2} - \cdots - c_q z^{-q}}$$

where q is twice the number of formants plus one, and the c's are calculated from the positions and bandwidths of the formant resonances and the spectral compensation parameter. Hence the z-transform of the speech is

$$X(z) = \frac{1}{1 - \sum_{k=1}^{p} c_k z^{-k}} I(z)$$

where $I(z)$ is the transform of the impulsive excitation.

This is similar to the linear prediction relation. If p and q are the same, then the linear predictive coefficients a_k form a p'th order polynomial which is the same as that obtained by multiplying together the second-order polynomial representing the individual formants (together with the first-order one for spectral compensation). Furthermore, the predictive error $E(z)$ can be identified with the impulsive excitation $I(z)$. It is possible to parametrize the error signal by its frequency and amplitude (two relatively slowly-varying quantities) instead of transmitting it sample by sample (at an 8 kHz rate). Compared to the speech sampling rate, the source parameters vary relatively slowly and this leads to an extremely low data rate.

Linear prediction can also be considered as a kind of curve-fitting technique. Figure 2.2 illustrates how four samples of speech signal can predict the next one. If the order of linear prediction is high enough (at least 10) with the correct coefficients, the prediction will closely model the resonances of the vocal tract. The prediction error will be very small until the next pitch period begins.

Practical linear predictive coding schemes operate with a value of p between 10 and 15. The a_k's are recalculated every pitch period from 10 to 25 msec. The pitch

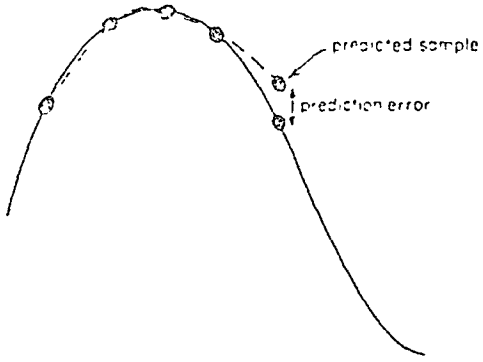


Figure 2.2: Curve Fitting

and amplitude of the speech are estimated and transmitted at the same rate. If the speech is unvoiced, there is no pitch value: an "unvoiced flag" is transmitted instead.

At the receiver, the excitation waveform is reconstructed. For voiced speech, it is impulsive at the specified frequency and with the specified amplitude, while for unvoiced speech it is random, with the specified amplitude. This signal $e(n_0)$, together with the transmitted parameters $a_1, \ldots, a_p$, is used to regenerate the speech waveform by

$$x(n) = e(n) + \sum_{k=1}^{p} a_k x(n-k)$$

The key problem in linear predictive coding is to determine the values of the coefficients $a_1 \ldots$ to minimize

$$\sum_n e(n)^2$$

There are several methods to solve it. We can find their advantages and disadvantages and even programs in many books. We will not discuss them.

The linear predictive parameters that need to be stored or transmitted are: 1. pitch, 2. voiced-unvoiced flag, 3. overall amplitude level, and 4. filter coefficients.

Table 2.1: Bit requirements for each parameter in LPC-10

	Voiced Sounds	Unvoiced Sounds	Comment
Pitch/Voicing	7	7	6 bits pitch, 1 voicing 60 values, semilog
Energy	5	5	32 values, semilog
k_1	5	5	Coded by table lookup
k_2	5	5	Coded by table lookup
k_3	5	5	Linear
k_4	5	5	Linear
k_5	4	-	Linear
k_6	4	-	Linear
k_7	4	-	Linear
k_8	4	-	Linear
k_9	3	-	Linear
k_{10}	2	-	Linear
Synchronization	1	1	Alternating 1s and 0s
Error Detection/ Correction	-	21	
TOTAL	54	54	

Frame Rate 44.4 Hz (22.5 msec)

The DVT2400 is improved LPC which still has some similarity to the US Government standard coding scheme LPC-10. The bit assignment for the parameters in LPC-10 is shown in Table 2.1.

In the system, all these parameters must be transmitted from DSPU to the PC through a synchronous communication port and stored on the floppy disk when recording or vice versa when playback, as shown in Figure 2.3.

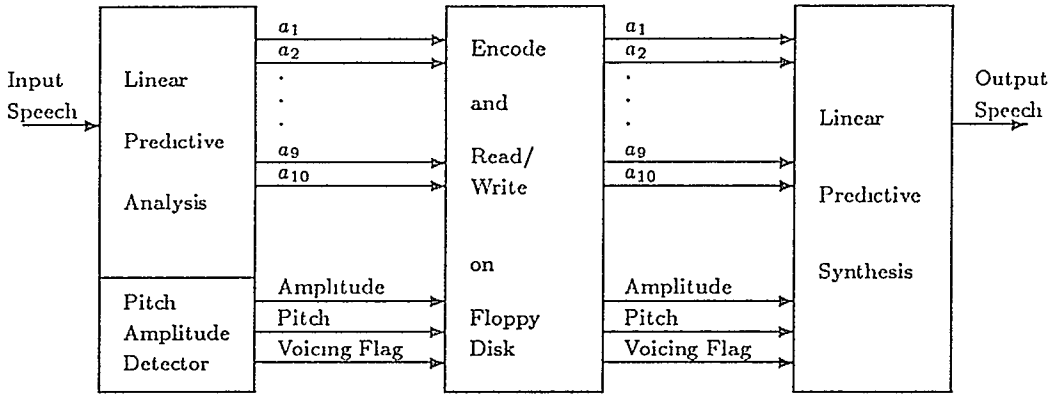


Figure 2.3: Linear Prediction used in the System

II. Realization of Linear Predictive Coding in System

The DSPU may be divided into four functional sections, as shown in Figure 2.4. they are: A digital speech processing section, a host processor section (including control, memory, etc.), a digital I/O port, and an analog I/O port.

The first step of speech processing is coding. In the coding procedure, an analog speech signal from the handset microphone is supplied to an industry standard μ-law coding and decoding (CODEC) chip. This signal is then passed through an anti-aliasing filter and sampled at a 6.4KHz rate using pulse code modulation (PCM). It is then converted to an 8-bit serial data stream. In the decoding procedure or the receive path (synthesis), the output from the digital processing section is converted back to an analog speech signals by the CODEC, then fed to a low pass filter and the handset earpiece.

A μ-law CODEC chip S3507 is used in the system. It contains independent circuitry for processing transmitting and receiving signals. Switched capacitor filters provide the necessary bandwidth limiting of voice signals in both directions. Circuitry

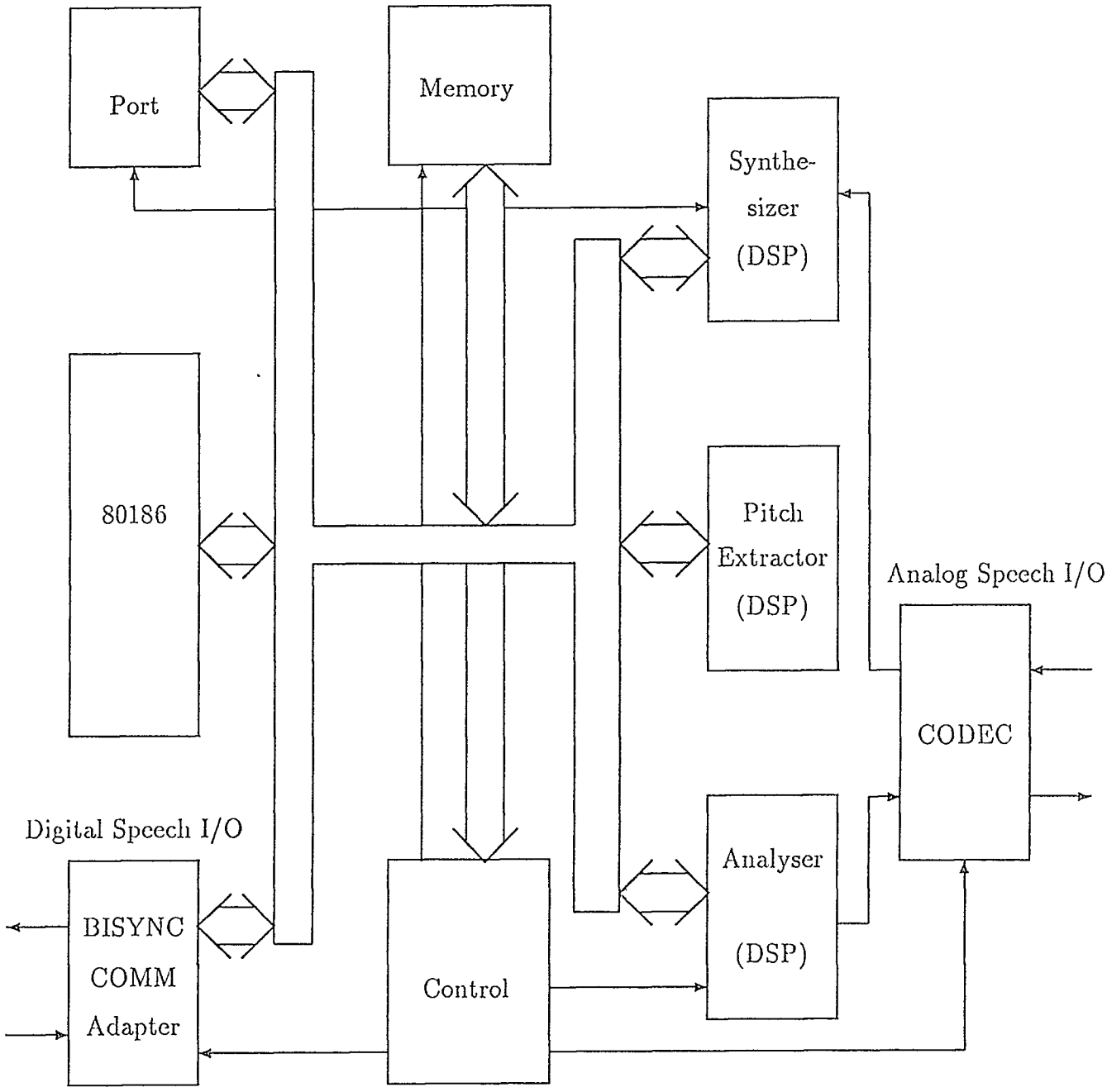


Figure 2.4: Block Diagram of DSPU

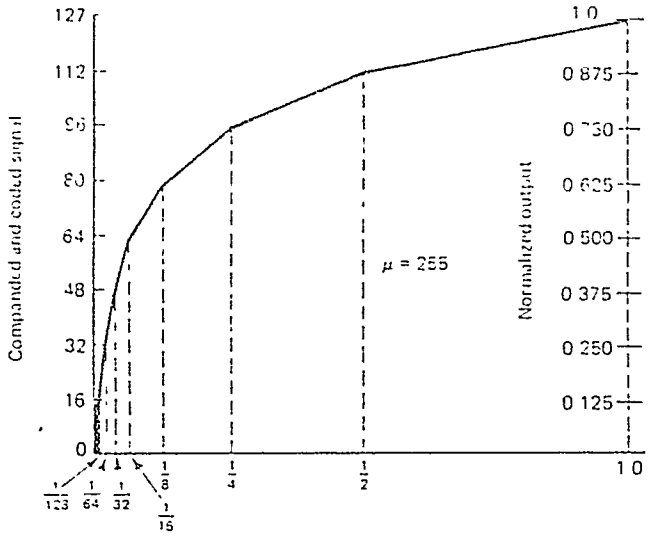


Figure 2.5: Companding Curve of the μ-law Compander

for coding and decoding operates by the principle of successive approximation, using charge redistribution in a binary weighted capacitor array to define segments and a resistor chain to define steps. A bandgap voltage generator supplies the reference level for the conversion process. The PCM data word is formatted according to the μ-law companding curve with the sign bit and the ones complement of the 7 magnitude bits. The μ-law quantization is given by:

$$y(n) = s_{max} \frac{\log[1 + \mu \frac{|s(n)|}{s_{max}}]}{1 + \log(1 + \mu)} sgn[s(n)]$$

where

- s_{max} is the maximum absolute value of the signal;
- $sgn[s(n)] = \pm 1$ when $s(n)$ is positive or negative,
- $|\, s(n)\,|$ is the absolute value of $s(n)$;
- μ is a parameter that determines the level of compression.

Figure 2.5 shows companding curve of μ-law compander.

This process of compression-expansion avoids enlarging the quantization stepsize for weaker signal and then impacts the speech quality when the quantizer is designed to accommodate strong signals. The 8-bit PCM data is clocked out by transmission shift clock which can vary from 64KHz to 2.048MHz. The chip also has noise suppression when the channel is idle.

From the CODEC, the serial data stream goes to a digital speech processing section which consists of three custom 16 bit digital signal processor chips. All of them are μPD 7720 which is introduced here.

Fabricated in high-speed NMOS by NEC, the μPD7720 signal processing interface (SPI) is a complete 16-bit microcomputer on a single chip, as shown in Figure 2.6. ROM space is provided for program and data/coefficient storage, while the on-chip RAM may be used for temporary data, coefficients and results. Computational power is provided by a 16-bit Arithmetic/Logic unit(ALU) and a separate 16 $\times$ 16-bit fully parallel multiplier. This combination allows the implementation of a "sum of products" operation in a single 250 ns instruction cycle. In addition, each arithmetic instruction provides for a number of data movement operations to further increase throughput. Two serial I/O ports are provided for interfacing to CODECs and other serially oriented devices while a parallel port provides both data and status information to conventional μP for more sophisticated applications.

Features of the μPD7720 chip include:

1 Fast instruction execution: 250 ns/8 MHz clock;

2 16-bit data word;

3 Multioperation instructions for optimizing program execution;

4 Large memory capacity;

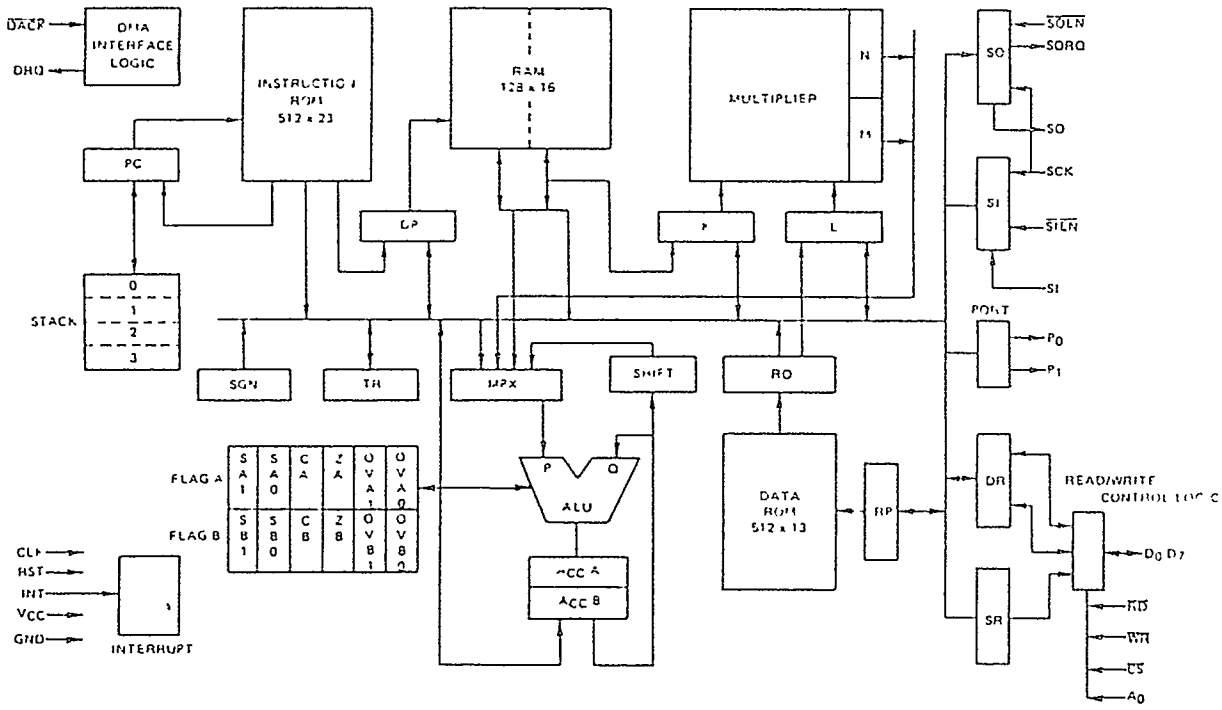


Figure 2.6: Block Diagram of μPD7720

5 Program ROM, 512 × 23 bits;

6 Data/coefficient ROM, 510 × 13 bits;

7 Data RAM, 128 × 16 bits;

8 Fast (250ns/ 8MHz) 16 × 16-bit parallel multiplier with 31-bit result;

9 Four-level subroutine stack for program efficiency;

10 Multiple I/O capabilities.

Each instruction is 23-bit long which is divided into 10 fields related to different tasks, such as Jump, ALU, addressing etc. Each instruction can be expressed as a set of six instructions of assembly language. Program for a DSP chip is straightforward to save time. The PASCAL and C program for calculating LPC parameters can be ported to μPD7720 assembly.

All three NEC 7720, programmed differently, provide speech analysis, pitch extraction and speech synthesis. The speech data stream from the CODEC is fed to the signal processors for analysis and pitch extraction. The analysis models the data, simultaneously processing voiced and unvoiced sounds while compressing and removing information unnecessary to reproduce speech faithfully and intelligibly. This method allows for the transmission of sounds that traditional low bit rate voice digitizers cannot classify, resulting in improved voice quality and speaker recognition, even in noisy environments.

As a host processor, the 80186 integrates a chip-select logic unit, two independent high-speed DMA channels, three programmable timers, a programmable interrupt controller and a clock generator. All the 80186 integrated peripherals are controlled by 16-bit registers contained in a 256-byte control block, which may be mapped into

either the memory or I/O space. The chip-select logic unit, timer, interrupt controller and clock generator are used in the system. Six memory chip-select outputs are provided for 3 address areas: upper memory, lower memory, and midrange memory. The range of each chip-select is user programmable. The 80186 can also generate chip-selects for up to seven peripheral devices. The 16-bit programmable timers are used for real-time coding and time delays. The interrupt controller is used for three DSP chips and 8274 communication chip.

The data from the digital signal processors are fed to the host microprocessor in the form of pitch frequency information and linear predictive coding parameters. The host microprocessor then performs a quantization operation on the data, formats it to 54 bit serial-synchronous data frames, each containing 53 data bits and a synchronization bit and is outputed in RS-232C standard.

Data received from the disk is fed to the host microprocessor where it is processed and fed to the synthesizing signal processor. This chip synthesizes the digital signals into a serial data stream which is then sent to the CODEC to be converted back analog voice signals. Therefore, the system must have a capability of getting data from DSPU through the synchronous communication adapter and storing them or vice versa in real-time processing. Because the speed to access a floppy disk is much slower than à harddisk, during real-time operation, it may be too late to start disk operation when a buffer is already full or empty. In the design two buffers with length of 512 bytes each are used. When one buffer is busy with transmission or reception, the other is used to read from or write on to a floppy disk.

Two interrupts which connect to IRQ_3 and IRQ_4 are used for receiving and transmitting respectively. An error flag is used to tell that the disk file cannot be read or written for some reasons as well as due to recording timeout. When it is set,

the recording or playback operation is stopped. The program also checks if any key is entered to stop recording or playback. The programs below show the interrupt service routine and recording precedure.

```
void interrupt ssi() {
      disable();
      *(pointer++) = inportb(DATA_PORT);
      counter++;
      outportb(PORT_8259_20, 0x20);
      enable();
}


record() {
      int i, numwrite;
      unsigned data;
      char key;
      if( (fp = fopen(filename,"wb")) != NULL ) {
            disable();
            old_vect = getvect(0xb);
            setvect(0xb,ssi);
            enable();
      }
      else return;
      pointer = buff0;
      outportb(MODE_INITIAL, 0x98);
      outportb(INTERNAL_CONTROL, 0x02);
```

```
outportb(EXT_MODEM_INTERFACE, 0x10);
outportb(EXT_MODEM_INTERFACE, 0x00);
outportb(USART_STATUS, 0x0C);
outportb(USART_STATUS, 0x32);
outportb(USART_STATUS, 0x32);
outportb(USART_STATUS, 0x04);
inportb(DATA_PORT);
data = inport(PORT_8259_20);
outport(PORT_8259_20, data & 0xf7ff);
outportb(PORT_8259_20, 0x20);
while ( !keyhit() ) {
    if ( counter >= BUFFLENGTH ) {
        counter = 0;
        if (bufflag) {
            pointer = buff0;
            numwrite = fwrite(buff1,1,BUFFLENGTH,fp);
        }
        else {
            pointer = buff1;
            numwrite = fwrite(buff0,1,BUFFLENGTH,fp);
        }
        bufflag = 1;
        if (numwrite != BUFFLENGTH)
            error_flag = 1;
    }
```

```
        if(error_flag) break;
    }
    outportb(EXT_MODEM_INTERFACE, 0x10);
    outportb(EXT_MODEM_INTERFACE, 0x00);
    outport(PORT_8259_20, data );
    outportb(PORT_8259_20, 0x20);
    fclose(fp);
    error_flag = 0;
    disable();
    setvect(0xb,old_vect);
    enable();
}
```

Chapter 3

PC XT Interfacing Circuitry

I. Introduction

Interfacing circuitry deals with the connection between IBM PC XT, DSPU, and a telephone line, as shown in Figure 3.1.

When the computer is off, the system can work as an ordinary telephone. So the external set (the base of ordinary telephone set) has ringer, dialer, and 2-4 convertor. The handset which includes mouth and ear pieces switches to either internal or external circuits. Three signals: ring detect, DTMF detect, remote off-hook detect connect to the IBM PC XT through an AND gate. Each flag for DTMF data received or off-hook detected is latched in a D flip-flop for CPU to check and they can be enabled or disabled to generate an interrupt. With an 8-bit read/write port, the CPU is able to check call progress and control a telephone set.

The compressed speech data are transmitted between DSPU and the IBM PC XT through synchronous serial port. Each frame has fifty-four bits including one synchronizing bit alternated with 1 and 0 pattern.

The main function of the telephone line interfacing is to feed the telephone line with analog speech signal of one side and get one of the other side from the telephone line. The speech signal to feed the telephone line can be either from the mouth piece

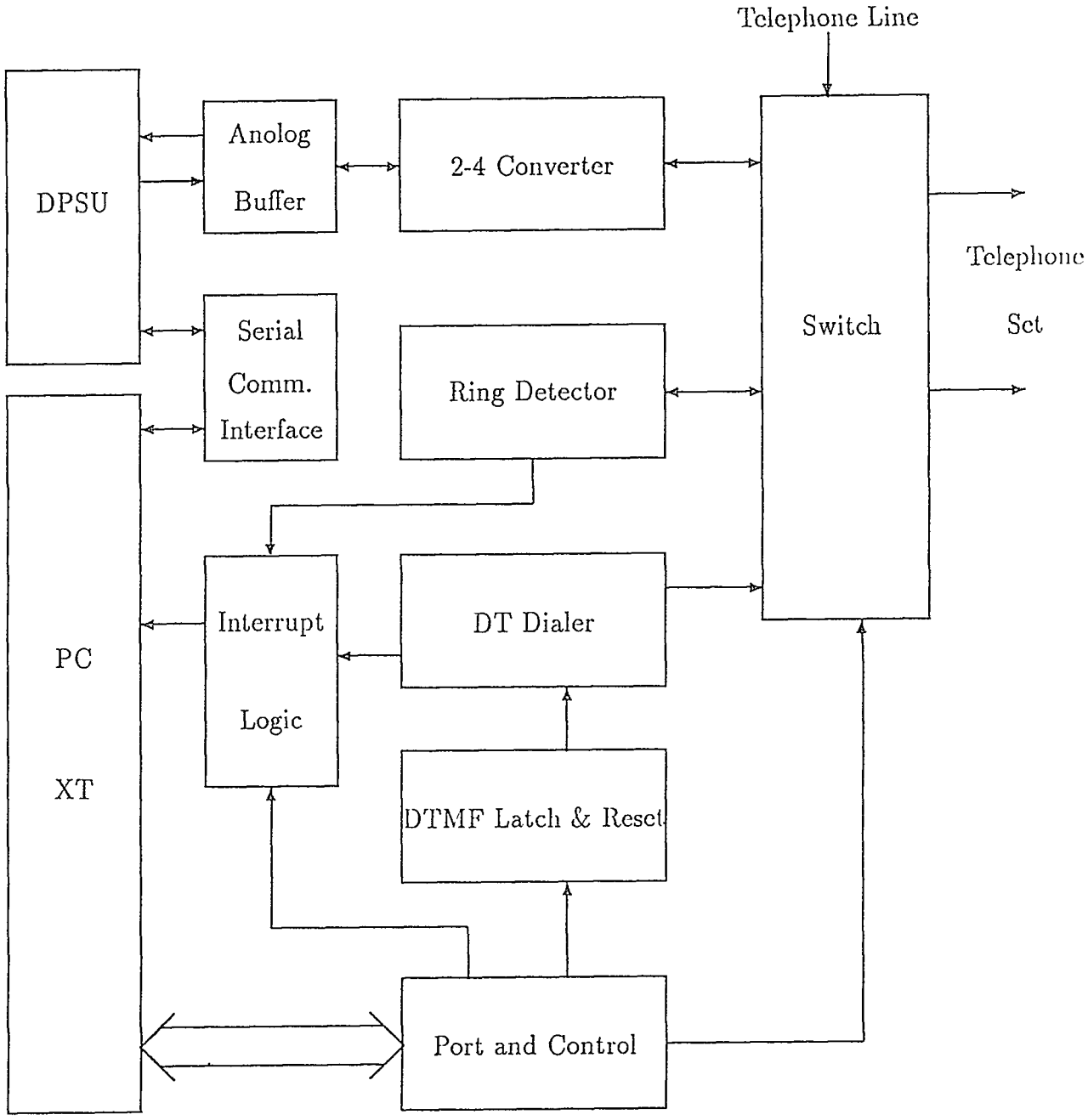


Figure 3.1: Block Diagram of PC Interface

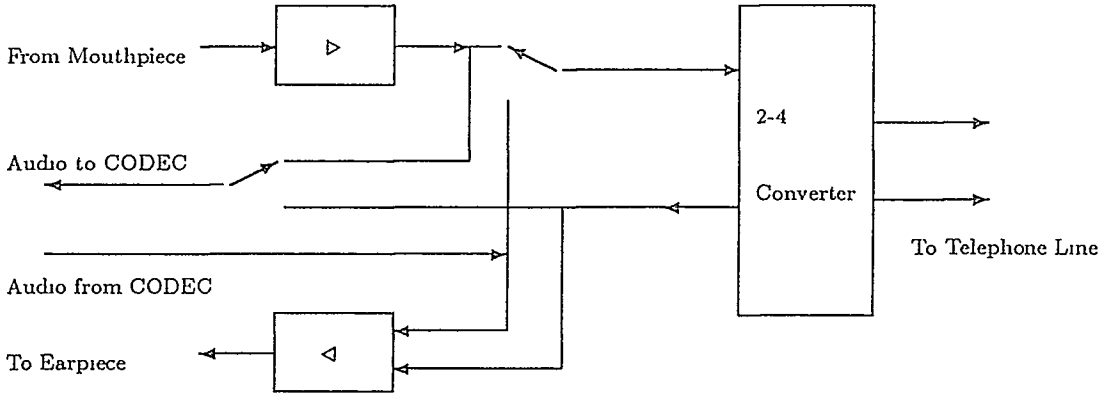


Figure 3.2: Analog Audio Signal Flowchart

or from CODEC and the speech signal either from the telephone line or from the mouth piece can connect to CODEC as shown in Figure 3.2. The speech signal is switched by an analog bilateral switch HC4066 which connects to +5V and −5V power lines and an MC1488 is used to shift the logic level to control the switches.

II. DTMF Transceiver

When you make a phone call, traditionally, pulses generated by shorting and opening the pair of telephone line, tells the connection device the number of the destination. Now, DTMF (dual tone multi-frequency) is often used. It can still be sent out after the connection established.

Silicon Systems' SSI 75T2090 is used to transmit and receive DTMF signals as well as to detect call progress signals in the system. It is a Dual-Tone Multi-Frequency (DTMF) Transceiver that can both generate and detect all 16 standard Touch-Tone.

The DTMF Receiver in the SSI 75T2090 detects the presence of a valid tone pair on a telephone line or other transmission medium. The analog input is pre-processed by 60Hz reject and band-splitting filters, then is fed into an Automatic Gain Controller. Eight bandpass filters detect the individual tones. The chip provides the

Table 3.1: DTMF Coding and 75T2090 Digit In/Out

	Hexadecimal Code				Frequency	
Digit in /out	D_7 D_3	D_6 D_2	D_5 D_1	D_4 D_0	Low Group f_0 Hz	High Group f_0 Hz
1	0	0	0	1	697	1209
2	0	0	1	0	697	1336
3	0	0	1	1	697	1447
4	0	1	0	0	770	1209
5	0	1	0	1	770	1336
6	0	1	1	0	770	1447
7	0	1	1	1	852	1209
8	1	0	0	0	852	1336
9	1	0	0	1	852	1447
0	1	0	1	0	941	1336
.	1	0	1	1	941	1209
#	1	1	0	0	941	1447
A	1	1	0	1	697	1633
B	1	1	1	0	770	1633
C	1	1	1	1	852	1633
D	1	0	0	0	941	1633

coded digital outputs which can drive standard CMOS circuitry, and are three-state enabled to facilitate bus-oriented architectures.

The DTMF generator on the SSI 75T2090 responds to a hexadecimal code input with a valid tone pair. Pins D_4-D_7 are the data inputs for the generator. A high to low transition on LATCH causes the hexadecimal code to be latched internally and the appropriate DTMF tone pair to begin. The DTMF output is disabled by a high on RESET and will not resume until new data are latched in.

The call progress detector consists of a bandpass filter and an energy detector for turning the on/off cadences into a microprocessor compatible signal.

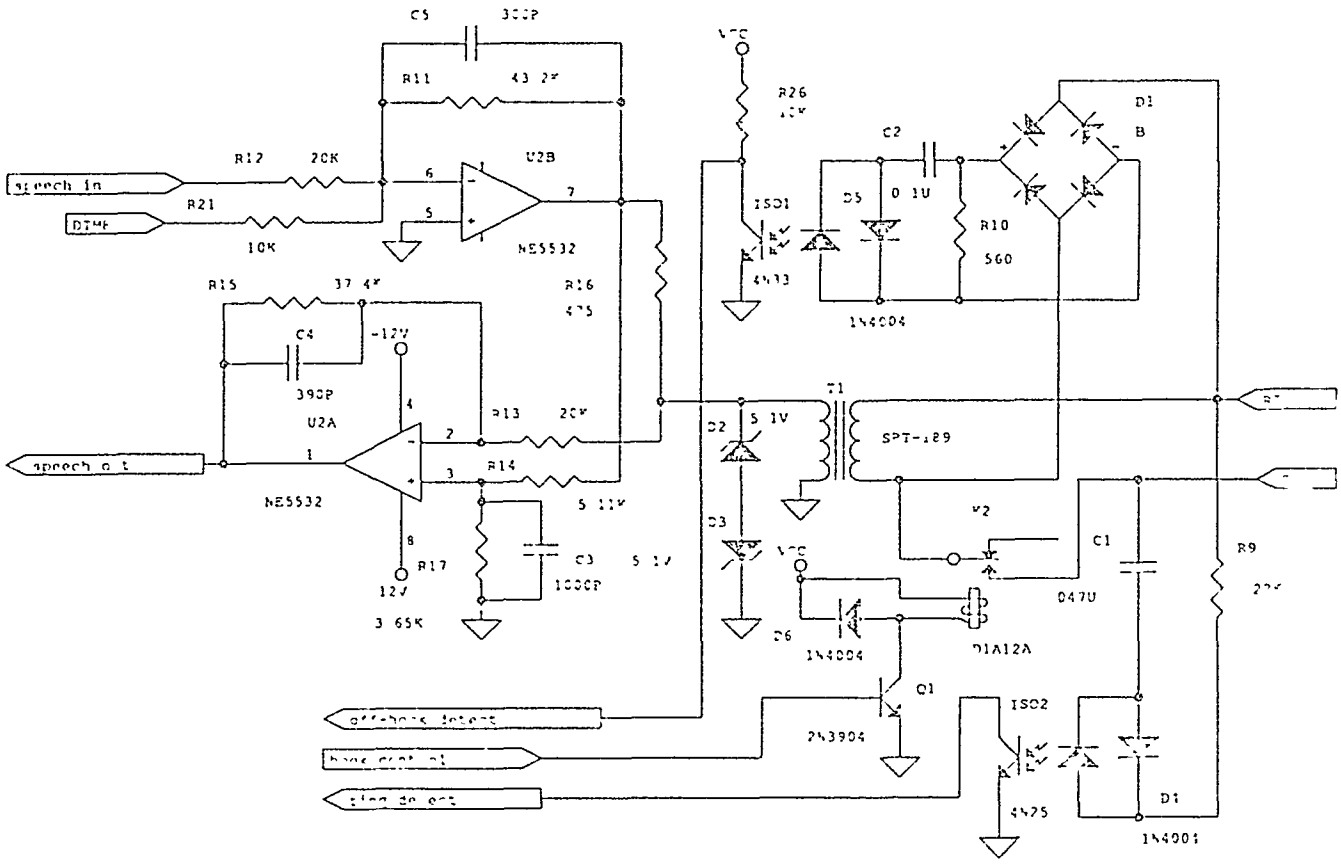


Figure 3 3: Schematic of Telephone Line Interface

III. Telephone Line Interface

The telephone line interface comprises of a ring detector, a remote off-hook detector, aon/off hook controller, a 2-4 convertor and a DTMF tranceiver. In this section, we will only briefly discuss the first three. Two photo couplers and a transformer are used to isolate the telephone line from the internal circuit. The schematic is shown in Figure 3.3.

In "on-hook" condition, no DC loop exists through the telephone set. Only ring signals come in to signal a incoming call. When you pick up the telephone set, the DC loop is closed, the resulting DC current tells the central office that one wants to make

a call and it provides a dial tone if the line is not busy. If the call is completed after dialing, the central office will send a ring voltage to the destination and a ringback to the caller. When the destination picks up the telephone-set to signal an "off-hook" condition, the central office will remove the ring voltage and connect the two parties. In the system, the computer has to 'pick up' hand-set, check call progress, and dial.

During the receiving of a call procedure, ISO_2 conducts because AC ring voltage comes in and it will generate an interrupt. The ring detect interrupt service routine will check ring voltage again and make Q_1 on. When Relay K_2 contacts close, they provide a DC path across the telephone line through B and R_{10}. The resulting DC current through R_{10} signals an "off-hook" condition to the central office, which then removes the ring voltage and connects the calling party to the line. Bridge rectifier BR_1 ensures that, regardless of the phone-line polarity, the end of R_{10} that's connected to C_2 is always positive with respect to the other end. When the other party hangs up the telephone hand-set, a pulse of high voltage will be generated and it instantly lessens the DC voltage appearing between tip and ring. That voltage change is coupled through C_2 as a gated pulse that appears on IC8 pin3, momentarily illuminating the internal LED. The ISO_1 phototransistors then conduct, placing a brief low in pins 3 of U9A. That will be latched into a D flip-flop for the CPU to check and generate an interrupt when it is enabled.

When the computer makes a call, it closes K_2 and then checks the dialing tone from the output of U2A. After making sure of the dialing tone on the line, it sends a U2B DTMF signal to dial and check which signal appears on the line. If it is a ringback, the computer will continue to check and send the message to the input of U2B after the ringback disappears and before a timeout.

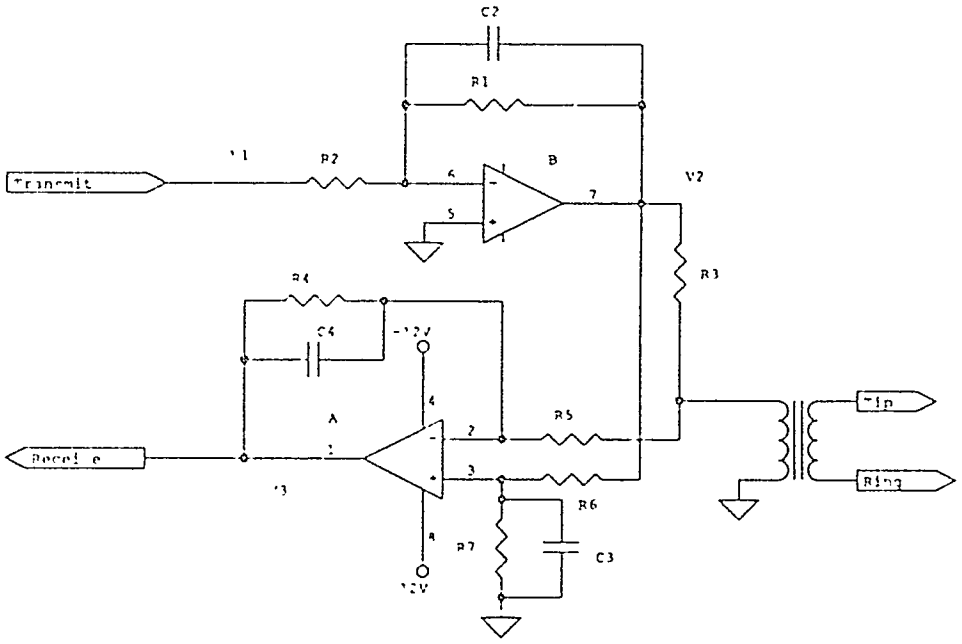


Figure 3.4: Schematic of 2-4 Converter

IV. 2-4 Converter

The signal on the phone line is the sum of the transmit and receive signals. The 2-4 converter subtracts the transmitted signal from the signal on the line to form the received signal. It is important to match the hybrid impedance as closely as possible to the telephone line to produce only the received signal. Figure 3.4 shows the schematic of the 2-4 converter.

It uses two operational amplifiers, one in the transmit path and the other in the receive path. The input from the CODEC or mouth piece amplifier provides a gain of 6 dB over the transmit signal level desired at the line. Under ideal conditions, with no loss in the transformer and perfect line matching, the signal level at the line will then be the desired -9dBm. In practice however there is impedance mismatch and loss in the coupling transformer. Therefore it may be desired to provide a gain in the

transmit and receive paths to overcome the loss. The receive gain G_R and transmit gain G_T are set by the ratios of resistors R_4, R_5 and R_1, R_2 respectively. The circuit can be analyzed as follows:

$$V_R = -\frac{R_4}{R_5}(V_{TR} + V_{RX}) + \left(1 + \frac{R_4}{R_1}\right)\left(\frac{R_7}{R_6 + R_7}\right) V_2$$

$$V_R = -\frac{R_4}{R_5} V_1$$

If R_1/R_2 is chosen to equal the loss in the transformer, it can be assumed the V_2 is twice as high as V_{TX} (transmit portion of the total line signal). Since $V_{TR} = V_{TX} + V_{RX}$ and $V_2 = 2\ V_{TX}$

$$\begin{aligned} V_R &= -\frac{R_4}{R_5}(V_{TR} + V_{RX}) + \left(1 + \frac{R_4}{R_1}\right)\left(\frac{R_7}{R_6 + R_7}\right) 2V_{TX} \\ &= -\frac{R_4}{R_5} V_{RX} + \left[(V_{TR} + V_{RX}) + \left(1 + \frac{R_4}{R_1}\right)\left(\frac{2R_7}{R_6 + R_7}\right) - \frac{R_4}{R_5}\right] V_{TX} \end{aligned}$$

To eliminate any transmit signal from appearing at the received signal input, the second term in the above equation must be set to zero, giving

$$\frac{R_4}{R_5} = \left(1 + \frac{R_4}{R_1}\right)\left(\frac{R_7}{R_6 + R_7}\right)$$

Solving for R_6/R_7, we get

$$\frac{R_6}{R_7} = 1 + \frac{2R_5}{R_4}$$

Additionally, we have

$$G_R = \frac{R_4}{R_5}$$

and

$$G_T = \frac{R_1}{R_2}$$

These equations can be solved to select component values that meet the desired requirements.

Chapter 4

Software for IBM PC XT

I. Introduction to Terminate-and-Stay-Resident Program

A Terminate-and-Stay-Resident (TSR) program is employed in this design. A general idea about TSR is given here.

TSR programs are loaded from the command line, and reside in memory above MS-DOS and beneath transient programs. After the TSR is made resident, DOS takes over, and the user can execute other programs. In the system, TSR interrupt service routine attaches the keyboard interrupt, timer, and IRQ_2, so that the TSR can pop up as the result of one of the events, They can be a hot key, the timer when it reaches a previous set time, or interrupt source IRQ_2. These events can occur while the system is executing another TSR, running a transient task, or waiting for a command. When the TSR is finished, the interrupted task resumes.

As a single-task operation system, MS-DOS controls various resources to support a single task. The TSR program, when it interrupts another program and pops up, needs the services of MS-DOS. So the conditions available for popping up and context switching must be handled with care to prevent the system from crashing.

When a TSR interrupts a task, the context items that have to be switched are:

- Stack
- Program segment prefix (PSP)
- Control-Break setting
- Critical error interrupt handler
- Disk transfer address (DTA)
- Video memory
- Screen cursor

Switching the stack can guarantee enough stack for the TSR. The TSR must switch the DOS PSP pointer to its own PSP which contains fields that control how the program operates. The TSR program should not be terminated when the user presses Control-Break or Control-C, usually to terminate the current program. The DTA has to also be saved because DOS reads and writes records from files opened with the FCB functions and searches directory in this area. And the video mode, video memory contents, cursor position, and cursor configuration must be saved and restored since they all are used by TSR.

Most MS-DOS functions are not reentrant. The typical TSR cannot operate without DOS functions. So the following flags are established to tell TSR if it can pop up or not.

- Own running flag (avoid reentrant execution of itself)
- Disk operations flag
- ROM-BIOS video functions flag

- DOS busy flag

In the system, the basic condition for TSR to pop up is set by three interrupt service routines. They are key ISR, timer ISR and an ISR that attaches to IRQ_2.

Another feature of the TSR in the system is that it is able to communicate with the TSR after it is resident. It can be suspended, unloaded and can not be loaded for a second time if it is already memory resident.

II. System Functions

In this section, the software related to the system functions is explained. The main structure and flowchart of the software are shown in Figure 4.1 and menu structure in Figure 4.2.

During telephone line connection, call progress signals the status of the telephone line. Call progress is measured by the presence of one or more supervisory signals on the phone line. There is the dial tone that tells you that the line is ready for use, the busy signal that tells you a call can not be completed, the ringback that simulates the ringing sound of the phone at the other end, and finally, an obscure reorder signal that tells you that the call only went halfway through and has to be repeated. Figure 4.3 shows all the standards for those new or precise call-progress signals.

Two methods of detecting a call progress signal are by the frequency and their cadence. For the most reliable detection, both of those should be used together.

Unfortunately, through the output of SSI 75T2090, the computer can only know if a signal in the frequency range from 350Hz to 620Hz is present or not. If such a signal is present, the computer must identify the call progress by its cadence.

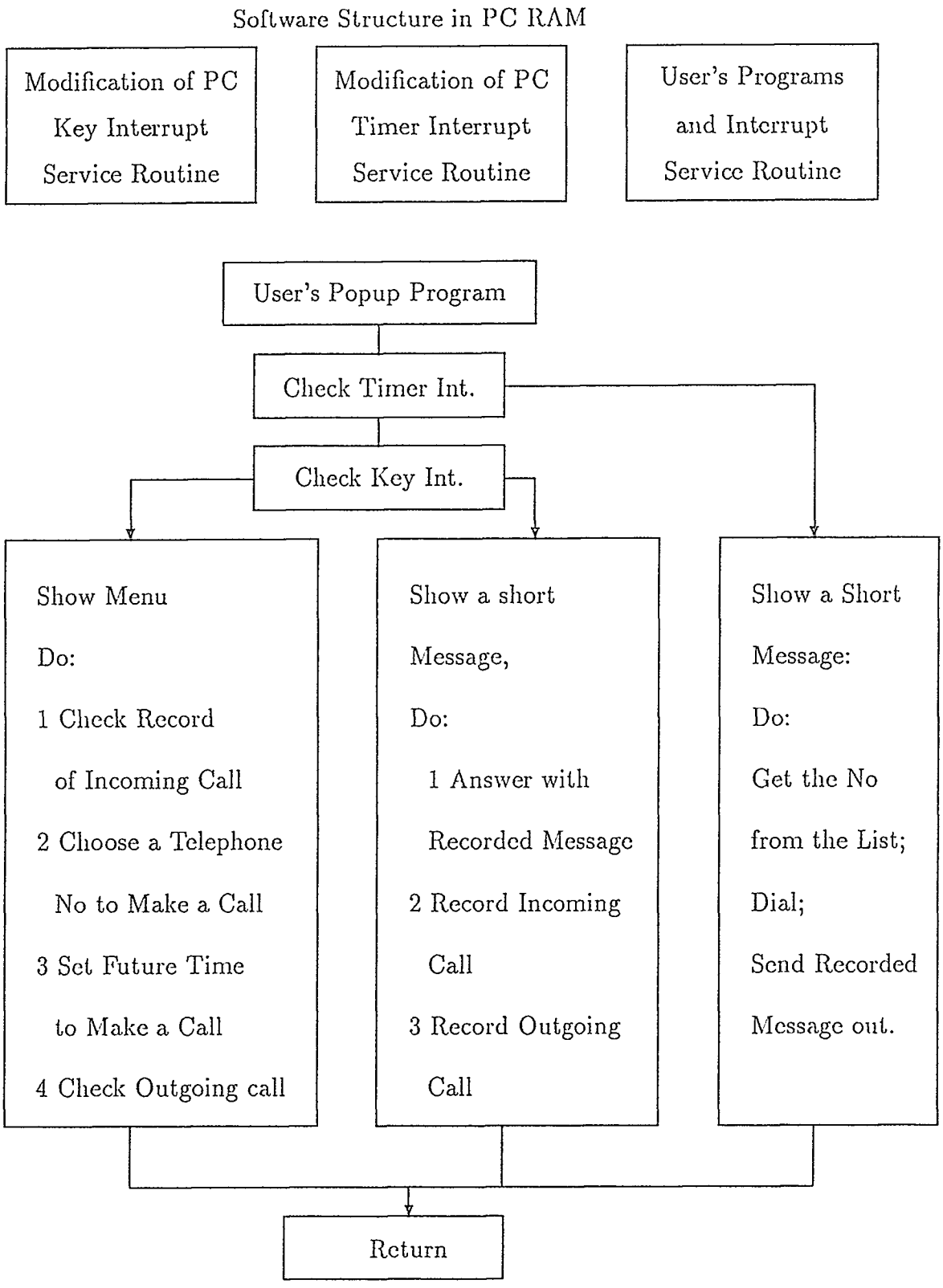


Figure 4.1: Flowchart of the Program for PC

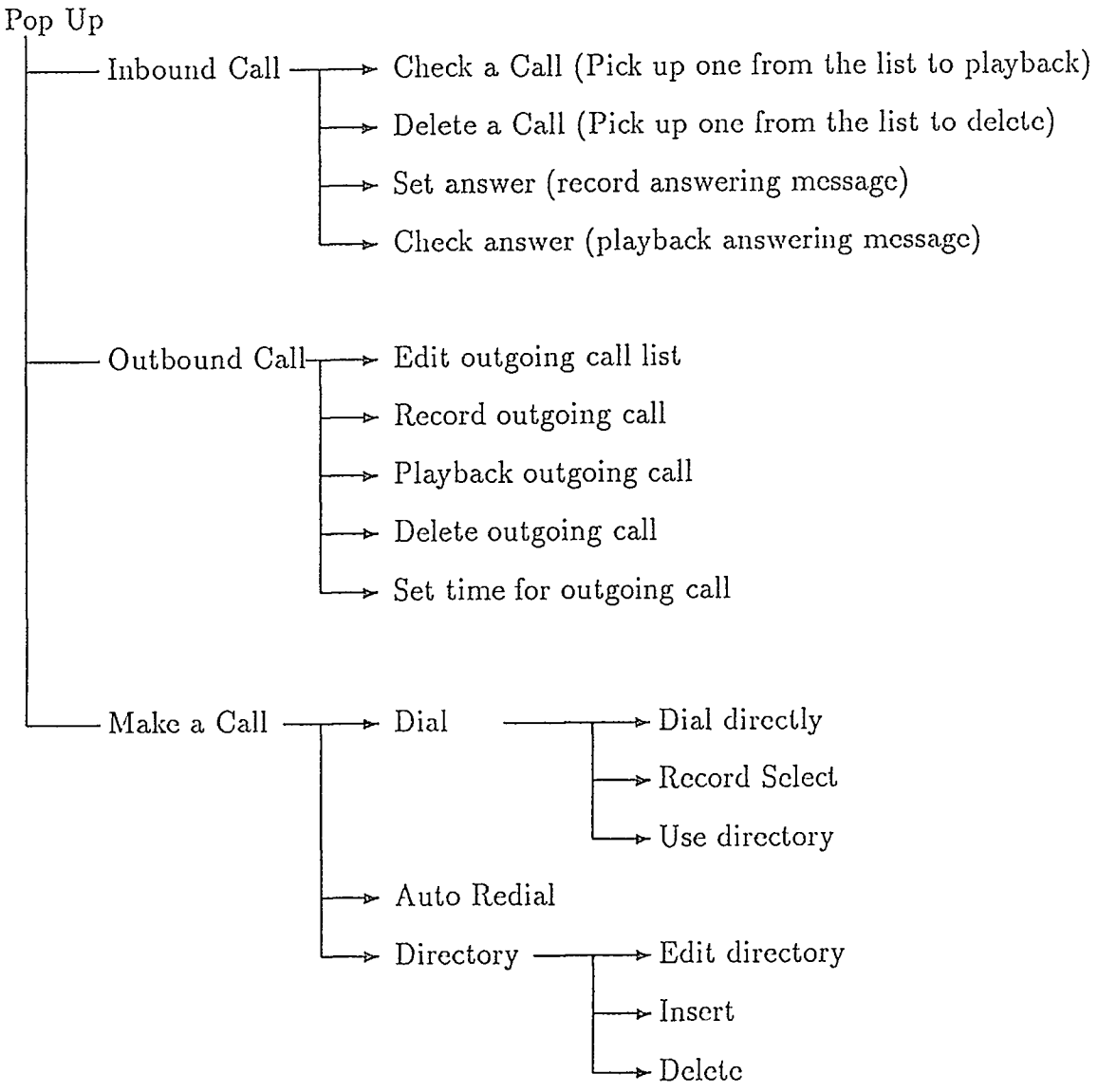


Figure 4.2: Menu Structure

The DIAL TONE frequencies are a 350 Hz sinewave and a 440 Hz sinewave that are continuously present until dialling is begun.

The RINGBACK simulates the phone being rung at other end. this is a 440 Hz sinewave and a 480 Hz sinewave with a cadence of 2 seconds on and 4 seconds off.

The BUSY signal announces the call cannot be completed. This is 480 Hz sinewave and a 620 Hz sinewave with a cadence of 0.5 seconds on and 0.5 seconds off.

The REORDER signal announces the call went only part way through the phone system. Like the busy signal, this is also a 480 Hz sinewave and a 620 Hz sinewave The cadence is 0 25 seconds on and 0 25 seconds off.

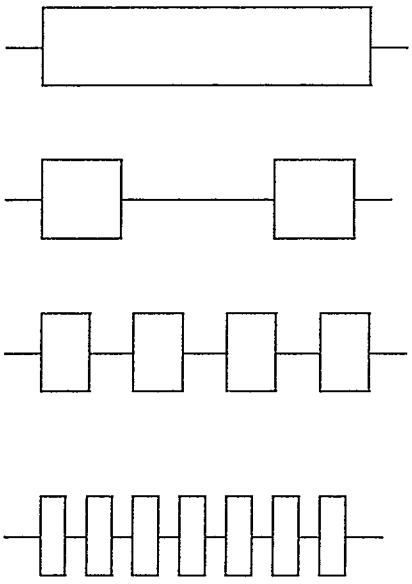


Figure 4.3: Telephone Call Progress

The computer samples and stores call progress detect from the output of SSI 75T2090 at every fixed time period for thirty times. The computer also stores the ideal patterns for each call progress signal. For example, the pattern of reorder signal can be '11110000111100001111000011110000' and '11111111000000001111111100000000' for a busy signal. After all samples have been finished, the computer compares input sample pattern to each ideal pattern. The program has a threshold and shift function in case that the phases are different and noisy. When two patterns compare, they shift relatively to get the minimum number. The program checks if it is less than the threshold. If so, the number of that ideal pattern is returned. The programs are shown below.

```
#define WAIT_TIME 5
enum react { NOTHING,DIALTONE,RINGBACK,BUSY,REORDER };
#define detect_tone() (((inportb(FONEPORT)&0x80) == 0)?0:1)
unsigned one_num( long in ) {
```

```
    int i,cnt;
    for(i=0,cnt=0; i<32; i++){
        cnt += in & 1;
        in >>= 1;
    }
    return( cnt );
}
#define shift(x) ((x<<1)—( ((x&0x80000000)==0)?0:1 ))
unsigned related(long l1,long l2){
    unsigned min=32,tmp,i;
    for(i=0; i<31; i++){
        if( ( tmp = one_num(l1Î2) ) < min) min = tmp;
        l2 = shift(l2);
    }
    return min;
}
enum react test_tone(){
    long buff;
    unsigned i,cnt;
    for( i=0,cnt=0; i< WAIT_TIME * 32; i++){
        delay(63);
        if( detect_tone() == 0 ) cnt = 0;
        else{
            if( cnt == 0 ) cnt++;
            else break;
```

```
        } /* else */
    }/* for */
    if( i >= WAIT_TIME * 32 ) return( NOTHING );
    buff = 0x3;
    for( i=2; i<32; i++){
        delay(63);
        buff <<= 1;
        buff += detect_tone();
    }/* for i */
    if( related( buff,0xf0f0f0f0 ) < 8 ) return( REORDER );
    if( related( buff,0xff00ff00 ) < 8 ) return( BUSY );
    buff = 0;
    for( i=0; i<32; i++){
        delay(188);
        buff <<= 1;
        buff += detect_tone();
    }/* for i */
    if( related( buff,0x000007ff ) < 10 ) return( RINGBACK );
    if( related( buff,0xffffffff ) < 8 ) return(DIALTONE);
    return(NOTHING);
}/* test_tone */
```

When the computer records every speech data file, it must get a file name for that speech data in order not to be mixed up. The file name for each incoming call is defined as the specific time when a call comes in. The first letter of a file name with

A to L indicates January to December, the next two digits are day of month, the next two digits are hours and the next two are minutes, and the last digit indicates seconds. So the file name for incoming speech data gets a format of YDDHHIMMS.in and the program for the record file name is shown below.

```
getrecordname() {
    char head[12] = "ABCDEFGHIJKL";
    struct tm *curtime;
    time_t bintime;

    time(&bintime);
    curtime = localtime(&bintime);
    sprintf(filename,"%c%02d%02d%02d%d.in",head[curtime->tm_mon],
        curtime->tm_mday,curtime->tm_hour,curtime->tm_min,
        curtime->tm_sec/6);
}
```

When a user wants to make future calls, he just selects 'outbound call' from the main menu and tells the computer the time, number and message for each call. The computer can do it one by one without attention. When the user sets a future call, an execution-time-pointer will point the nearest time he sets. The timer interrupt service routine will check the time each minute. The program for calling pops up if it reaches that time. The timer interrupt service routine also checks the hot key-combination for the program popping up. The system has a function that it will stop after recording a certain period of time when it processes real-time speech data. A timer-counter is set in the main program and is deceased in the timer ISR. If the

timer-counter is zero, nothing happens. If it is nonzero, it will be decreased by one, and as soon as it is decreased to zero, the error-flag is set and real-time recording or playback ends. The following shows the timer interrupt service routine.

```
void interrupt newtimer()
{
    (*oldtimer)();
    if (!scancode){
        kbval = peekb(0,0x417);
        if (!resoff && ((kbval & keymask) << keymask) == 0)
            if(!running){
                popflg = 1;
                popkind = 'k';
            }
    }
    if( (*pt.PP)%1092 == 0){
        if(timer_count != 0 && running == 1)
            if(--timer_count == 0) error_flag = 1;
        if(exe_t_pt != -1 && running == 0){
            popflg=1;
            popkind = 't';
        }
    }
    if (popflg && peekb(dosseg, dosbusy) ==0)
        if (diskflag == 0 && videoflag == 0){
            outportb(0x20,0x20);
```

```
            popflg = 0;
            dores();
        }
}
```

The ring detector and data valid flag of DTMF receiver connect to IRQ_5. Both of them can be read, reset and disabled. The program is activated when a call comes in. The previously recorded message will be sent out and the incoming message will be recorded on the disk. The information about all the incoming calls will be listed on the screen for the user to check or delete.

A directory of up to one hundred phone numbers is built up for the user's convenience. The user can use this directory or just dial to make a phone call. He also can record the speech of either partner. When the line is busy, the user can set to automatic redial mode and the computer will dial again and again for him.

Chapter 5

Conclusion and Suggested Future Work

Digital speech signal processing has lots of advantages over analog speech signal processing. The integrated PC based Digital Speech System can perform detailed call accounting, message indexing for easy retrieval, and timed distribution of messages to pre-selected telephone numbers. The potential applications are many. In the future, with added features of speech recognition, speaker recognition, and text to speech conversion, the system can become more versatile and perform more useful automation functions.

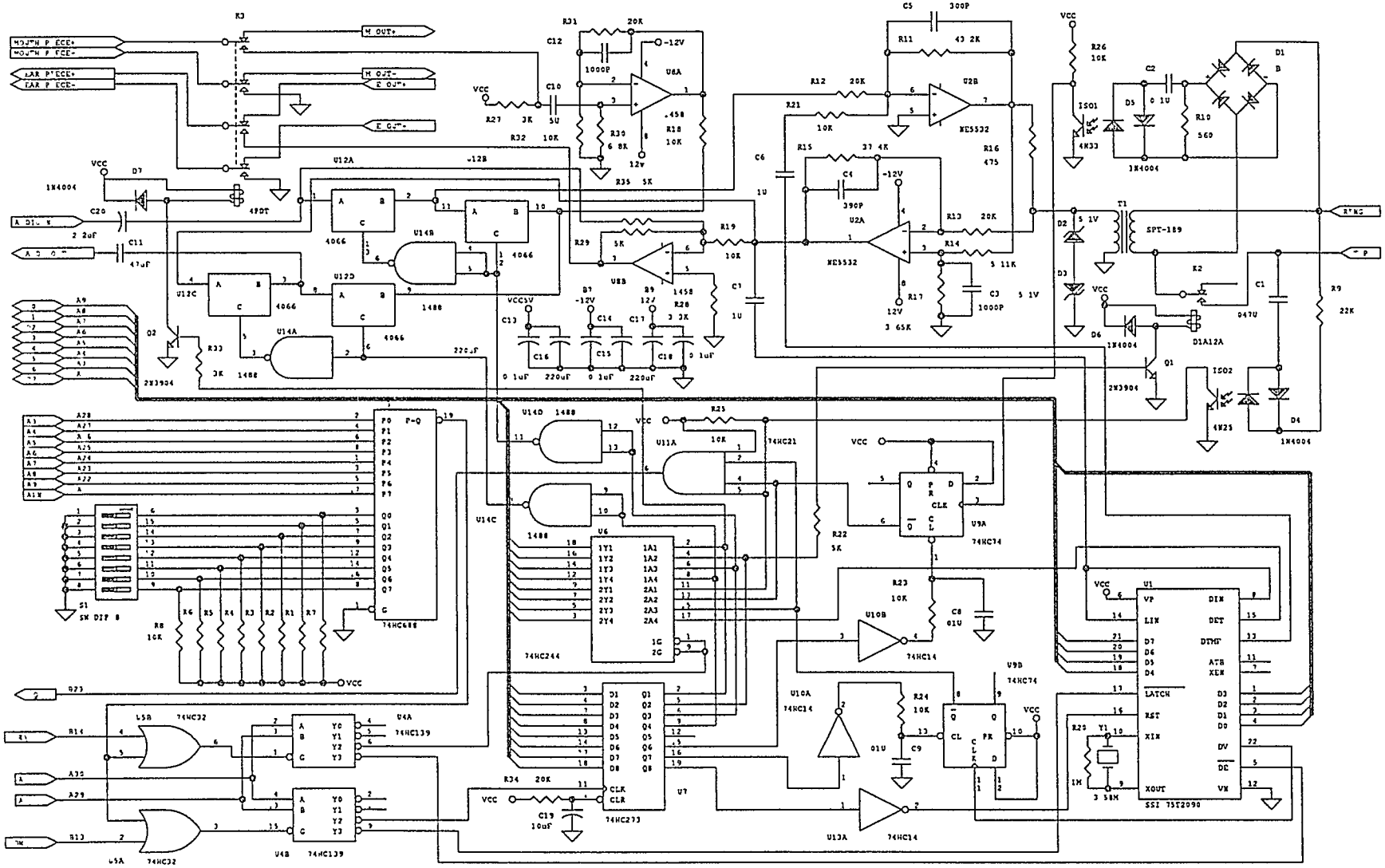
The speed of speech communications mostly depends on the speaking not listening. Sometimes, it is desirable to listen to previously recorded thirty minutes of speech within twenty minutes or even less with the same quality, intonation. There is no problem recognizing a speaker who speeds up his speech. You also can adjust the time of speech to fit it into a certain interval. This, as a suggested future work, can be achieved in the system. In this design, all compressed speech data are stored on the disk in a format of 54-bit frame and can be processed further. Normally, from forty to fifty percent of speech data frames are silent. Some of the silent frames can

be removed and the speech is still clear. Some unsilenced frames can be removed from a steady group of speech frames. It is possible that up to fifty percent of time can be saved. This processing can be realized by checking each frame, removing some of silent and redundant ones to decrease time duration of a steady segment, and then joining the remaining frames together.

The system has been implemented with speech synthesis and analysis. If speech segments are stored instead of messages, the computer can create a sentence by itself. With implementation of speech recognition, it is possible that a PC based digital speech system with more intelligent functions can be developed.

Appendix A

Circuit Diagrams



Title	INTEGRATED DIGITAL SPEECH SYSTEM
Size	Document Number
A	Designer Zhen Shu

Appendix B

Software for IBM PC XT

```
#include <dos.h>
#include <stdio.h>
#include <string.h>
#include <dir.h>
#include "tsr.h"

#include <conio.h>
#include <ctype.h>

#define table_len 100
#define file_len  10
#define ON  1
#define OFF 0

#define MENU_ROW 5
#define MENU_COL 60
#define MENU_LEN 18
#define MENU_HIG 12

#define LIST_IN_ROW 8
#define LIST_IN_COL 12
#define LIST_IN_LEN 37
#define LIST_IN_HIG 12

#define DIR_ROW 3
#define DIR_COL 5
#define DIR_LEN 35
#define DIR_HIG 12

#define LIST_OUT_ROW 5
#define LIST_OUT_COL 9
#define LIST_OUT_LEN 50
#define LIST_OUT_HIG 12

/* key code */
#define BACK  8
#define TAB   9
#define CR    0xd
#define ESC   0x1b
#define SPACE 0x20

#define FUN   0
#define PgUp  0x49
#define PgDn  0x51
#define UP    0x48
#define DOWN  0x50
#define LEFT  0x4b
#define RIGHT 0x4d
#define DEL   0x53
#define HOME  0x47
#define END   0x4f
#define F3    0x3d
```

```
#define F4    0x3e
#define F5    0x3f

#define PORT_8259_20 0x20
#define PORT_8259_21 0x21

#define INT_LEVEL    0xb

struct sheet{
       char NAME[15];  /* user name */
       char NUM[15];   /* user's teltphone number */
}table[table_len];

#include <time.h>
#define List_len 10
union TIME_UNION{
         char c[4];
         long l;
};
struct List_struct{
       char NAME[15];
       char NUM[15];
       union TIME_UNION TIME;
       char STATE;
} List[List_len];
int exe_t_pt=-1;

static union{
       long far *PP;
       long II;

} pt;

static int menu;
static int error_flag = 0;     /* stop record or playback when ON */
static int timer_count = 0;    /* use for duration of recording */
FILE *fp;
char filename[13];
static char sv1[2400], sv2[700], sv3[300];

static union REGS rg;

extern unsigned _heaplen = 128;
extern unsigned _stklen = 4096;
unsigned scancode = 52;
unsigned keymask = 12;
char signature[] = "PHONE";     /* TSR SIGNATURE */

static void interrupt ifunc();

main(argc, argv)
char *argv[];
```

```
{
  int ivec;

  if((ivec = resident(signature, ifunc)) != 0){
     if(argc > 1){
       rg.x.ax = 0;
       if (strcmp(argv[1], "quit") == 0)
          rg.x.ax = 1;
       else if (strcmp(argv[1], "restart") == 0)
          rg.x.ax = 2;
       else if (strcmp(argv[1], "wait") == 0)
          rg.x.ax = 3;
       if(rg.x.ax){
            /* call the communications interrupt */
          int86(ivec, &rg, &rg);
          return;
       }
     }
  printf("\nDigiPhone is already resident");
  }
  else {
       /* ------ initial load of TSR program ------ */
       printf("\nResident DigiPhone is loaded");
       openfiles();
       resinit();
  }
}

void interrupt ifunc(bp,di,si,ds,es,dx,cx,bx,ax)
{
  if (ax == 1)          /* "quit " */
       terminate();
  else if (ax == 2)     /* "restart" */
       restart();
  else if (ax == 3)     /* "wait" */
       suspend();
}

#define DTMFPORT 0x22e
#define FONEPORT 0x22c

enum funct   { PICK_UP,
               ON_HOOK,
               DETECT_TONE,
               CHECK_RING,
               CHECK_REMOTE_HOOK,
               CHECK_DTMF_RX,
               LINE2CODEC,
               MOUTH2CODEC,
               MOUTH2LINE,
               CODEC2LINE,
               RES_OFFHOOK_INT,
```

```
            DIS_OFFHOOK_INT,
            RES_DTMF_RX_INT,
            DIS_DTMF_RX_INT,
            RES_DTMF_TX,
            DIS_DTMF_TX,
            HANDSET_INTL,
            HANDSET_EXTL,
            RESET_ALL
}

do_function(function_num)
enum funct function_num;
{
  static char fone_status;

  switch (function_num) {
    case PICK_UP:
        fone_status |= 2;
        outportb(FONEPORT, fone_status);
        break;
    case ON_HOOK:
        fone_status &= 0xd;
        outportb(FONEPORT, fone_status),
        break;
    case DETECT_TONE:
        return (((inportb(FONEPORT)&0x80) == 0)?0:1);
    case CHECK_RING:
        return (((inportb(FONEPORT)&0x10) == 0)?0:1);
    case CHECK_REMOTE_HOOK
        return (((inportb(FONEPORT)&0x20) == 0)?0:1);
    case CHECK_DTMF_RX:
        return (((inportb(FONEPORT)&0x40) == 0)?0.1);
    case LINE2CODEC:
        fone_status |= 8;
        outportb(FONEPORT, fone_status),
        break;
    case MOUTH2CODEC:
        fone_status &= 7;
        outportb(FONEPORT, fone_status);
        break;
    case MOUTH2LINE:
        fone_status &= 0xb;
        outportb(FONEPORT, fone_status);
        break;
    case CODEC2LINE:
        fone_status |= 4;
        outportb(FONEPORT, fone_status);
        break;
    case RES_OFFHOOK_INT:
        fone_status |= 0x20,
        outportb(FONEPORT, fone_status),
        fone_status &= 0xdf;
```

```
            outportb(FONEPORT, fone_status);
            break;
        case DIS_OFFHOOK_INT:
            fone_status |= 0x20;
            outportb(FONEPORT, fone_status);
            break;
        case RES_DTMF_RX_INT:
            fone_status |= 0x40;
            outportb(FONEPORT, fone_status);
            fone_status &= 0xbf;
            outportb(FONEPORT, fone_status);
            break;
        case DIS_DTMF_RX_INT.
            fone_status |= 0x40;
            outportb(FONEPORT, fone_status);
            break;
        case RES_DTMF_TX:
            fone_status |= 0x80;
            outportb(FONEPORT, fone_status);
            fone_status &= 0x7f;
            outportb(FONEPORT, fone_status);
            break;
        case DIS_DTMF_TX:
            fone_status |= 0x80;
            outportb(FONEPORT, fone_status),
            break;
        case HANDSET_INTL:
            fone_status |= 1;
            outportb(FONEPORT, fone_status);
            break;
        case HANDSET_EXTL:
            fone_status &= 0xfe;
            outportb(FONEPORT, fone_status),
            break;
        case RESET_ALL:
            outportb(FONEPORT, 0xff);
            fone_status = 0;
            outportb(FONEPORT, fone_status);
    }
}

#define WAIT_TIME 5
enum react { NOTHING,DIALTONE,RINGBACK,BUSY,REORDER };

#define detect_tone() (((inportb(FONEPORT)&0x80) == 0)?0:1)

unsigned one_num( long in ){
        int i,cnt;
        for(i=0,cnt=0; i<32; i++){
                cnt += in & 1;
                in >>= 1;
        }
```

```
        return( cnt );
}

#define shift(x) ((x<<1)|( ((x&0x80000000)==0)?0:1 ))
unsigned related(long l1,long l2){
        unsigned min=32,tmp,i;
        for(i=0; i<31; i++){
                if( ( tmp = one_num(l1^l2) ) < min) min = tmp;
                l2 = shift(l2);
        }
        return min;
}

enum react test_tone(){
        long buff;
        unsigned i,cnt;

        for( i=0,cnt=0; i< WAIT_TIME * 32; i++){
                delay(63);
                if( detect_tone() == 0 ) cnt = 0,
                else{
                        if( cnt == 0 ) cnt++;
                        else break;
                } /* else */
        }/* for */
        if( i >= WAIT_TIME * 32 ) return( NOTHING );

        buff = 0x3;
        for( i=2; i<32; i++){
                delay(63);
                buff <<= 1;
                buff += detect_tone();
        }/* for i */

        if( related( buff,0xf0f0f0f0 ) < 8 ) return( REORDER );
        if( related( buff,0xff00ff00 ) < 8 ) return( BUSY );
        buff = 0;
        for( i=0; i<32; i++){
                delay(188);
                buff <<= 1;
                buff += detect_tone();
        }/* for i */
        if( related( buff,0x000007ff ) < 10 ) return( RINGBACK );
        if( related( buff,0xffffffff ) < 8 )  return(DIALTONE);
        return(NOTHING);
}/* test_tone */

dial_one( char c){
        outportb(DTMFPORT,c),
        delay(150);
        do_function(RES_DTMF_TX);
        delay(100);
```

```
}

dial( char buff[] ){
        int i=0;
        while( buff[i] != 0 ){
                if( buff[i] >= '1' && buff[i] <= '9' )
                        dial_one( buff[i]<<4 );
                else if(buff[i] == '0')
                        dial_one(0xa0);
                i++;
        }
}

#include <stdlib.h>

#define KEYBD   9
#define TIMER   0x1C
#define DISK    0x13
#define VIDEO   0x10
#define ZERODIV 0
#define INT28   0x28
#define CRIT    0x24
#define CTRLC   0x23
#define CTRLBRK 0x1B

static void interrupt (*oldbreak)();
static void interrupt (*oldctrlc)();
static void interrupt (*oldtimer)(),
static void interrupt (*old28)();
static void interrupt (*oldkb)();
static void interrupt (*olddisk)();
static void interrupt (*oldvideo)();
static void interrupt (*oldcrit)(),
static void interrupt (*ZeroDivVector)(),

static void interrupt newtimer();
static void interrupt new28();
static void interrupt newkb();
static void interrupt newdisk(),
static void interrupt newvideo();
static void interrupt newcrit();
static void interrupt newbreak(),
static void interrupt ring();
static union REGS rg;
static struct SREGS seg;

static unsigned sizeprogram;
static unsigned dosseg;
static unsigned dosbusy;
static char far *mydta;
static unsigned myss;
static unsigned stack;
```

```
static unsigned intpsp;
static unsigned psps[2];
static int pspctr;
static int resoff;
static int running;
static int popflg;
static int diskflag;
static int videoflag;
static int cflag;
static char popkind;

static void resterm(void);
static void pspaddr(void);
static void dores(void);

void resinit()
{
  segread(&seg);
  myss = seg.ss;
  stack = _SP;
  rg.h.ah = 0x34;
  intdos(&rg, &rg);
  dosseg = _ES;
  dosbusy = rg x.bx;
  mydta = getdta();
  pspaddr();
  oldtimer = getvect(TIMER);
  old28 = getvect(INT28);
  oldkb = getvect(KEYBD);
  olddisk = getvect(DISK);
  oldvideo = getvect(VIDEO);
  setvect(TIMER, newtimer);
  setvect(KEYBD, newkb);
  setvect(INT28, new28);
  setvect(DISK, newdisk);
  setvect(VIDEO, newvideo);
  setvect(INT_LEVEL,ring);

  do_function(RESET_ALL);
  do_function(DIS_OFFHOOK_INT);
  do_function(DIS_DTMF_RX_INT),
  do_function(HANDSET_INTL);
  outport(PORT_8259_20, inport(PORT_8259_20) & 0xf7ff);
  outportb(PORT_8259_20, 0x20);
  pt.II = 0x0040006cL;
  sizeprogram = myss + (( stack+50 ) / 16 ) - _psp;
  setvect(ZERODIV, ZeroDivVector),
  keep(0, sizeprogram);
}

void interrupt newvideo(bp,di,si,ds,es,dx,cx,bx,ax)
{
```

```
  static int hbx;
  static unsigned vidbp[5];
  static unsigned vidoutbp;

  vidbp[videoflag++] = _BP;
  _BX = bx;
  _BP = bp;
  (*oldvideo)();
  vidoutbp = _BP;
  hbx = _BX;
  _BP = vidbp[--videoflag];
  ax = _AX;
  bx = hbx;
  cx = _CX;
  bp = vidoutbp;
  dx = _DX;
  es = _ES;
  di = _DI;
}

static void interrupt newbreak()
{
  return;
}

void interrupt newcrit(bp,di,si,ds,es,dx,cx,bx,ax,ip,cs,flgs)
{
  ax = 0;
  cflag = flgs;
}


void interrupt newdisk(bp,di,si,ds,es,dx,cx,bx,ax,ip,cs,flgs)
{
  diskflag++;
  (*olddisk)();
  ax = _AX;
  newcrit();
  flgs = cflag;
  --diskflag;
}

static int kbval;

void interrupt newkb()
{
  extern unsigned scancode;
  extern unsigned keymask;

  if (scancode && inportb(0x60) == scancode){
     kbval = peekb(0, 0x417);
     if (!resoff && (kbval & keymask) == keymask){
```

```
        kbval = inportb(0x61);
        outportb(0x61, kbval | 0x80);
        outportb(0x61,kbval);
        disable();
        outport(0x20,0x20);     /* send eoi to 8259 */
        enable();
        if (!running){
          popflg = 1;
          popkind = 'k';
        }
        return;
      }
    }
    (*oldkb)();
}

void interrupt newtimer()
{
  extern unsigned scancode;
  extern unsigned keymask;

  (*oldtimer)();
  if (!scancode){
     kbval = peekb(0,0x417);
     if (!resoff && ((kbval & keymask) << keymask) == 0)
        if(!running){
          popflg = 1;
          popkind = 'k';
        }
  }

  if( (*pt.PP)%1092 == 0){
    if(timer_count != 0 && running == 1)
       if(--timer_count == 0) error_flag = 1;
    if(exe_t_pt != -1 && running == 0){
        popflg=1;
        popkind = 't';
    }
  }

  if (popflg && peekb(dosseg, dosbusy) ==0)
    if (diskflag == 0 && videoflag == 0){
        outportb(0x20,0x20);    /* send soi to 8259 */
        popflg = 0;
        dores();
    }
}

void interrupt new28()
{
  (*old28)();
  if (popflg&& peekb(dosseg, dosbusy) != 0){
```

```
    popflg = 0;
    dores();
  }
}

void resident_psp()
{
  int pp;

  intpsp = peek(dosseg, *psps);
  for(pp = 0; pp < pspctr; pp++)
    poke(dosseg, psps[pp], _psp);
}

void interrupted_psp()
{
  int pp;

  for (pp =0; pp < pspctr, pp++)
  poke(dosseg, psps[pp], intpsp);
}

static void dores()
{
  static char far *intdta;
  static unsigned intsp;
  static unsigned intss;
  static unsigned ctrl_break;
  time_t tnow;
  struct tm *tmnow;
  union TIME_UNION tunow;

  running = 1;
  disable();
  intsp = _SP;
  intss = _SS;
  _SP = stack;
  _SS = myss;
  enable();
  oldcrit = getvect(CRIT);
  oldbreak = getvect(CTRLBRK);
  oldctrlc = getvect(CTRLC);
  setvect(CRIT, newcrit);
  setvect(CTRLBRK, newbreak);
  setvect(CTRLC, newbreak),
  ctrl_break = getcbrk();
  setcbrk(0);
  intdta = getdta();
  setdta(mydta);
  resident_psp();
  switch( popkind ){
    case 't':
```

```
        time(&tnow);
        tmnow = localtime(&tnow);
        tunow.c[0] = tmnow->tm_min;
        tunow.c[1] = tmnow->tm_hour;
        tunow.c[2] = tmnow->tm_mday;
        tunow.c[3] = tmnow->tm_mon;
        if(tunow.l == List[exe_t_pt].TIME.l)
           popup();
        break;
      case 'k':
        popup();
        break;
      case 'r':
        popup();
        break;
    }

    interrupted_psp();
    setdta(intdta);
    setvect(CRIT, oldcrit);
    setvect(CTRLBRK, oldbreak);
    setvect(CTRLC, oldctrlc);
    setcbrk(ctrl_break);
    disable();
    _SP = intsp;
    _SS = intss;
    enable();
    running = 0;
}

static int avec = 0;

unsigned resident(signature, ifunc)
char *signature;
void interrupt (*ifunc)();
{
char *sg;
unsigned df;
int vec;

segread(&seg);
df = seg ds - seg cs,
for (vec = 0x60; vec < 0x68; vec++){
  if (getvect(vec) == NULL){
     if ( !avec)
        avec = vec;
     continue;
  }
  for (sg = signature; *sg; sg++)
    if (*sg != peekb(peek(0,2+vec*4)+df, (unsigned)sg))
       break;
    if (!*sg)
```

```
        return vec;
}
if (avec)
   setvect(avec, ifunc);
return 0;
}

static void pspaddr()
{
  unsigned adr = 0;
  unsigned enddos;

  rg.h.ah = 0x52;
  intdos(&rg, &rg);
  enddos = _ES;
  enddos = peek(enddos, rg.x.bx-2);
  while (pspctr <2 &&
          (unsigned)((dosseg << 4) + adr) < (enddos << 4)){
    if (peek(dosseg, adr) == _psp){
       rg.h.ah = 0x50;
       rg.x.bx = _psp +1;
       intdos(&rg, &rg);
       if (peek(dosseg, adr) == _psp + 1)
          psps[pspctr++] = adr;
       rg.h ah = 0x50;
       rg.x.bx = _psp;
       intdos(&rg, &rg);
    }
    adr++;
  }
}

static void resterm()
{
  closefiles();
  setvect(TIMER, oldtimer);
  setvect(KEYBD, oldkb);
  setvect(INT28, old28);
  setvect(DISK, olddisk);
  setvect(VIDEO, oldvideo);
  setvect(avec, (void interrupt (*)()) 0);
  freemem(peek(_psp, 0x2c));
  freemem(_psp);
}

void terminate()
{
  if (getvect(VIDEO) == (void interrupt (*)()) newvideo)
    if (getvect(DISK) == (void interrupt (*)()) newdisk)
       if (getvect(KEYBD) == newkb)
          if (getvect(INT28) == new28)
             if (getvect(TIMER) == newtimer){
```

```
                resterm();
                return;
              }
  resoff = 1;
}

void restart()
{
   resoff = 0;
}

void suspend()
{
   resoff = 1;
}

int get_char()
{
  static int c,flag=0;
  if(flag == 0){
    while (1) {
      rg.h.ah = 1;
      int86(0x16, &rg, &rg);
      if(rg.x.flags & 0x40) {
        int86(0x28, &rg, &rg),
        continue;
      }
      rg.h.ah = 0;
      int86(0x16, &rg, &rg);
      if (rg.h.al == 0)
        c = rg.h.ah | 128;
      else
        c = rg.h al;
      break;
    }/*while*/
    if(c<128)
      return(c);
    else{
      flag = 1;
      return(0);
    }
 }else{
      flag = 0;
      return(c&0x7f);
 }
}

set(x,y,on,num)
int x,y,on,num;
{
   union REGS regs;
   regs.h.ah = 2;/* set cursor position */
```

```
    regs.h.dh = x;
    regs.h.dl = y;
    regs.h.bh = 0;
    int86(0x10,&regs,&regs);
    regs.h.ah = 1;/* turn on/off cursor */
    regs.h.ch = (on==1)?num·0x20;
    regs.h.cl = 11;
    int86(0x10,&regs,&regs);
}

window_frame(y,x,l,h,title,cursor_on)
int x,y,l,h,cursor_on;
char title[];
{
        int i,j,len;
        char buff[80];
        textattr(YELLOW + (BLUE<<4));

        len = strlen(title),
        set(x,y,OFF,13);
        putch('╒');
        for(i=1; i<(l-len)/2 ;i++) putch('═'),
        cprintf("%s",title);
        i += len;
        for( ; i<l-1; i++) putch('═');
        putch('╕');
        for(i=1; i<h-1; i++){
                set(x+i,y,OFF,13);  putch('│');
                set(x+i,y+l-1,OFF,13),putch('│');
        }
        set(x+h-1,y,OFF,13);
        putch('╘');
        for(i=2; i<l ;i++) putch('═');
        putch('╛');

        for(i=0; i<80; i++)
                buff[i] = ' ';
        buff[l-2] = '\0';
        for(i=h-2;i>0;i--){
                set(x+i,y+1,OFF,13);
                cprintf("%s",buff);
        }
        set(x+1,y+1,cursor_on,10),
        textattr(LIGHTGRAY + (BLUE<<4));
}

show_menu(select_group, row, col, hight_choice)
int select_group, row, col, hight_choice;
{
 int i;
 static char *fun_name[6][4]
                = {
```

```
                    {"Exit          ",
                     "Inbound call ",
                     "Outbound call",
                     "Make a call  "
                    },
                    {
                     "Check a call ",
                     "Delete a call",
                     "Set Answer   ",
                     "Check Answer ",
                    },
                    {
                     "Exit         ",
                     "Dial         ",
                     "Auto Redial  ",
                     "Directory    ",
                    },
                    {
                     "Exit         ",
                     "Edit         ",
                     "Insert       ",
                     "Delete       ",
                    },
                    {
                     "Exit         ",
                     "Dial Directly",
                     "Use Directory",
                     "Record Select",
                    },
                    {
                     "Exit         ",
                     "Make a Call  ",
                     "Check a Call ",
                     "Delete a Call ",
                    }
                   };

 for (i=0, i<4; i++) {
    set(row+(2*i), col, OFF, 13);
    if (i == hight_choice)
        textattr(BLACK + (LIGHTGRAY<<4));
    else
        textattr(LIGHTGRAY + (BLUE<<4));
    cprintf("%-14s",fun_name[select_group][i]);
 }
}

message(num, row, col)
char num;                       /* num = 0  clear message window */
int row, col;
{
    char saveline[80];
```

```
    static char *msg[10] = {"Playing back, Press any key to quit",
                            "Recording... Press any key to quit ",
                            "^Y^X: Move;  Ent: select, ESC. quit  ",
                            "        Do not interrupt!           ",
                            "      Waiting for dial tone         ",
                            "    Waiting for a while, please     ",
                            "   Not complete, dial again? Y/N    ",
                            "    Press any key, when finish.     ",
                            "            Pick up? Y/N            ",
                            "            Quit ?   Y/N            "};

    if(num != 0) {
       gettext(col,row+1,col+36,row+1,sv3),
       set(row,col,OFF,13);
       textattr(LIGHTGRAY + (BLUE<<4));
       cprintf("%-36s",msg[num - 1]);
    }
    else
       puttext(col,row+1,col+36,row+1,sv3),
}

edit(high_row, row, col, begin)
int high_row, row, col, begin;
{
    char buff[20], c;
    int i = 0;
    char key1;

    set(row+high_row, col+3, ON, 10);
    strcpy(buff,table[begin+high_row] NAME),

    while (1) {
       c = get_char();
       switch(c) {
         case FUN:
            key1 = get_char();
            switch(key1){
            case RIGHT:
                if(i<13 & buff[i] != '\0') {
                   i++;
                   putchar(buff[i-1]);
                }
                if(i>15 & i<29 & buff[i-16] != '\0') {
                   i++;
                   putchar(buff[i-17]);
                }
                break;
            case LEFT:
                if((i>1 & i<14) | (i>16)) {
                  i--;
                  putchar(BACK);
                }
```

```
                break;
            }
            break;
        case BACK:
            if(i>0) {
                i--;
                putchar(BACK);
                putchar(' ');
                putchar(BACK);
            }
            break;
        case CR:
            if (i<15) {
                set(row+high_row, col+18, ON, 10);
                /* store new name */
                buff[i] = '\0';
                strcpy(table[begin+high_row].NAME,buff);
                i = 16;
                strcpy(buff,table[begin+high_row].NUM);
            }
            else {
                /* store new number */
                buff[i-16] = '\0';
                strcpy(table[begin+high_row].NUM,buff);
                return;
            }
            break;

        default:
            if(i<14 & isprint(c) != 0) {
                putchar(c);
                buff[i++] = c;
            }
            if (i>15 & i<30 & isdigit(c) != 0) {
                putchar(c);
                buff[i-16] = c;
                i++;
            }
        }
    }
}


show_dial(high_row, row, col, begin, menu)
int high_row, row, col, begin, menu;
{
    int i;
    for (i = 0; i < 10; i++) {
        set(row+i, col, OFF, 13);
        if (!menu & i == high_row)
            textattr(BLACK + (LIGHTGRAY<<4));
        else
```

```
            textattr(LIGHTGRAY + (BLUE<<4));
          cprintf("%2d %-14s %-14s",begin+i,table[begin+i].NAME,
              table[begin+i].NUM);

      }
}

getrecordname() {
  char head[12] = "ABCDEFGHIJKL";
  struct tm *curtime;
  time_t bintime;

  time(&bintime);
  curtime = localtime(&bintime);

  /* filename is up to the time when a call is just coming, first letter
    A to L indicate Jan to Dec, next two are day of month, so are hours,
    mins, and the last letter indicates digit of seconds. YDDHHMMS.in */

  sprintf(filename,"%c%02d%02d%02d%d.in",head[curtime->tm_mon],
        curtime->tm_mday,curtime->tm_hour,curtime->tm_min,
        curtime->tm_sec/6);
}

#define EXT_MODEM_INTERFACE 0x3A1
#define INTERNAL_CONTROL    0x3A2
#define MODE_INITIAL        0x3A3
#define DATA_PORT           0x3A8
#define USART_STATUS        0x3A9


#define BUFFLENGTH 1024

int bufflag = 0;
char *pointer;
int  counter = 0;
void interrupt  (*old_vect)();
char buff0[BUFFLENGTH],buff1[BUFFLENGTH];
struct file_{
      char name[13];
      long filesize;
} file[file_len];
char month[12][4] = {"Jan", "Feb", "Mar", "Apr", "May",
                "Jun", "Jul", "Aug", "Sep", "Oct", "Nov","Dec"};

void interrupt sso(){
  disable();
  outportb(DATA_PORT,*(pointer++) );
  counter++;
  outportb(PORT_8259_20, 0x20);     /* write eoi to 8259 */
  enable();
}
```

```
playback()                      /* playback data file = filename */
{
  int i, numread;
  unsigned data;

  if( (fp = fopen(filename,"rb")) != NULL ){
    disable();
    old_vect = getvect(0xc);
    setvect(0xc,sso);
    enable();
  }
  else return;

  fread(buff0,1,BUFFLENGTH*2,fp),
  pointer = buff0;

  outportb(MODE_INITIAL, 0x98);
  outportb(INTERNAL_CONTROL, 0x02);
  outportb(EXT_MODEM_INTERFACE, 0x10);
  outportb(EXT_MODEM_INTERFACE, 0x00);
  outportb(USART_STATUS, 0x0C);
  outportb(USART_STATUS, 0x32);
  outportb(USART_STATUS, 0x32);
  outportb(USART_STATUS, 0x30);

  data = inport(PORT_8259_20);
  outport(PORT_8259_20, data & 0xefff);  /* Enable 8259 Int Level 4 */

  inportb(DATA_PORT);
  outportb(PORT_8259_20, 0x20);         /* write eoi to 8259 */
  outportb(USART_STATUS, 0x21);
  outportb(DATA_PORT, 0x05);

  while ( !keyhit() ) {
    if ( counter >= BUFFLENGTH ) {
      counter = 0;
      if (bufflag) {
        pointer = buff0;
        numread = fread(buff1,1,BUFFLENGTH,fp);
      }
      else {
        pointer = buff1;
        numread = fread(buff0,1,BUFFLENGTH,fp);
      }
      bufflag ^= 1;
      if (numread != BUFFLENGTH)
        error_flag = 1;          /* end of file */
    }

    if( error_flag > 0) break;

  }
```

```
    outportb(EXT_MODEM_INTERFACE, 0x10);
    outportb(EXT_MODEM_INTERFACE, 0x00);
    outportb(USART_STATUS, 0x0C);
    outportb(USART_STATUS, 0x32);
    outportb(USART_STATUS, 0x32);
    outportb(USART_STATUS, 0x30);
    outport(PORT_8259_20, data );   /* Restore 8259 Int Mask */
    inportb(DATA_PORT);
    outportb(PORT_8259_20, 0x20);   /* write eoi to 8259 */

    fclose(fp);
    error_flag = 0;

    disable();
    setvect(0xc,old_vect);
    enable();
}

void interrupt ssi(){
    disable();
    *(pointer++) = inportb(DATA_PORT);
    counter++;
    outportb(PORT_8259_20, 0x20);    /* write eoi to 8259 */
    enable();
}

record() {
    int i, numwrite;
    unsigned data;
    char key;

    if( (fp = fopen(filename,"wb")) != NULL ){
      disable();
      old_vect = getvect(0xb);
      setvect(0xb,ssi);
      enable();
    }
    else return;

    pointer = buff0;

    outportb(MODE_INITIAL, 0x98);
    outportb(INTERNAL_CONTROL, 0x02);
    outportb(EXT_MODEM_INTERFACE, 0x10);
    outportb(EXT_MODEM_INTERFACE, 0x00);
    outportb(USART_STATUS, 0x0C);
    outportb(USART_STATUS, 0x32);
    outportb(USART_STATUS, 0x32);
    outportb(USART_STATUS, 0x04);
    inportb(DATA_PORT);

    data = inport(PORT_8259_20),
```

```
    outport(PORT_8259_20, data & 0xf7ff); /* Enable 8259 Int Level 3 */
    outportb(PORT_8259_20, 0x20);         /* write eoi to 8259 */

    while ( !keyhit() ) {
       if ( counter >= BUFFLENGTH ) {
          counter = 0;
          if (bufflag) {
             pointer = buff0;
             numwrite = fwrite(buff1,1,BUFFLENGTH,fp);
          }
          else {
             pointer = buff1;
             numwrite = fwrite(buff0,1,BUFFLENGTH,fp);
          }
          bufflag ^= 1;
          if (numwrite != BUFFLENGTH)
             error_flag = 1;          /* disk full */
       }

       if( error_flag > 0) break;

    }

    outportb(EXT_MODEM_INTERFACE, 0x10);
    outportb(EXT_MODEM_INTERFACE, 0x00);
    outport(PORT_8259_20, data );        /* Restore 8259 Int Mask */
    outportb(PORT_8259_20, 0x20);        /* write eoi to 8259 */

    fclose(fp);
    error_flag = 0;

    disable();
    setvect(0xb,old_vect);
    enable();
}


int keyhit() {
    static union REGS rg;

    rg.h.ah = 1;
    int86(0x16, &rg, &rg);
    if (rg.x.flags & 0x40)
        return 0,
    rg.h.ah = 0;
    int86(0x16, &rg, &rg),      /* eat the keystroke */
    return 1;
}


show_list_in(high_row, row, col, menu, num)  /* menu and file */
int high_row, row, col, menu, num;
```

```
{
        int i;
        for (i = 0; i < 10; i++) {
            set(row+i, col, OFF, 13);

            if (!menu & i == high_row)
              textattr(BLACK + (LIGHTGRAY<<4));
            else
              textattr(LIGHTGRAY + (BLUE<<4));
            if (i < num)
            cprintf(" %2d  %-3s  %c%c,  %c%c:%c%c    %5d Sec.",i,
                 month[(int) file[i].name[0] - 0x41],
                 file[i].name[1],file[i].name[2],
                 file[i].name[3],file[i] name[4],
                 file[i].name[5],file[i].name[6],
                 file[i].filesize/300);
            else
            cprintf("                                ");
        }
}

auto_redial()
{
    int i;
    char dialnum[13];

    readkey(dialnum,11,20,6,"DIAL NUMBER");
    for (i=0; i<10; i++)  {
        do_function(PICK_UP),
        message(5,24,30);
/*      if(test_tone() == DIALTONE){*/
                dial(dialnum);
/*      }*/
        message(0,24,30);
        if ( test_tone() != RINGBACK) {
             do_function(ON_HOOK);
             delay(50000);
             continue;
        }
        else  break;
    }
    message(8,24,30);
    while ( !keyhit() ) ;
    message(0,24,30);
    do_function(ON_HOOK);

}

void inbound_call()
{
   int i,count=0,file_row=0;
   long totalsize;
```

```
  char key,key1;

  int menu = ON, menu_choice = 0;

  struct ffblk fileinfo;

  for(i = 0; i < file_len; i++) {
    strcpy(file[i].name, NULL);
    file[i].filesize = 0;
  }

  if (findfirst("*.in", &fileinfo, FA_SYSTEM) ==0)  {
       strcpy( file[0].name, fileinfo.ff_name);
       file[0].filesize = fileinfo.ff_fsize;

       i = 1;
       count = 1;
       while (findnext(&fileinfo) == 0 && i < 10) {
         count++;
         file[i].filesize = fileinfo.ff_fsize;
         strcpy(file[i].name, fileinfo.ff_name);
         i++;
       }
  }

   gettext(LIST_IN_COL+1,  LIST_IN_ROW,  LIST_IN_COL+1+LIST_IN_LEN,
LIST_IN_ROW+LIST_IN_HIG, sv1);

  window_frame(LIST_IN_COL, LIST_IN_ROW, LIST_IN_LEN, LIST_IN_HIG," Call List ",OFF),

  window_frame(MENU_COL, MENU_ROW, MENU_LEN, MENU_HIG, " MENU ",OFF);

  do {

    show_list_in(file_row, LIST_IN_ROW+1, LIST_IN_COL+1, menu, count);

    show_menu(1, MENU_ROW+2, MENU_COL+2, menu_choice),

    key = get_char();
    if (menu == OFF) {
     switch ( key ) {
       case FUN:
         key1 = get_char();
         switch( key1 ) {
           case UP:
             if(file_row > 0 ) file_row--;
             break;
           case DOWN:
             if(file_row < 9 ) file_row++;
         }
         break,
```

```
    case CR:
        switch (menu_choice) {
          case 0:              /* check */
             message(1,22,30);
             strcpy(filename, file[file_row].name);
             playback();
             message(0,22,30);
             break;
          case 1:              /* delete */
             unlink(file[file_row].name);
             for (i=file_row;i<count-file_row;i++) {
                strcpy (file[i].name, file[i+1].name),
                file[i].filesize = file[i+1].filesize;
             }
             count--;
             break;

        }
        menu = ON,
        break;
    case ESC:
        menu = ON;
        key = HOME;       /* keep prog on inbound sub menu */
        break;
  }
}
else {                    /* menu is ON */
  switch ( key ) {
    case FUN:
      key1 = get_char();
      switch( key1 ) {
        case UP·
          if(menu_choice > 0 ) menu_choice--;
          break;
        case DOWN·
          if(menu_choice < 3 ) menu_choice++;
          break;
      }
      break;

    case CR:
        switch (menu_choice) {
          case 2.              /* set answer */
                message(2,24,30);
                do_function(MOUTH2CODEC);
                strcpy(filename,  "answerin msg"),
                record();
                message(0,24,30);
                break;
          case 3:              /* check answer */
                if (findfirst("answerin.msg", &fileinfo, FA_SYSTEM) ==0) {
                    message(1,24,30);
```

```
                        strcpy(filename, fileinfo.ff_name);
                        playback();
                        message(0,24,30);
                    }
                    else {              /* print error message */
                        set(MENU_ROW+10, MENU_COL+1, OFF, 13);
                        textattr(LIGHTGRAY + (BLUE<<4));
                        cprintf("%-14s"," No Answer MSG");
                    }
                    break;
                case 0:
                case 1: menu = OFF;
                    break;
            }
        }
    }
  } while(key != ESC | menu == OFF);
  puttext(LIST_IN_COL+1, LIST_IN_ROW, LIST_IN_COL+1+LIST_IN_LEN,
        LIST_IN_ROW+LIST_IN_HIG, sv1);

}

get_near_t(){
  int i;
  time_t tnow;
  struct tm *tmnow;
  union TIME_UNION tunow;

  time(&tnow);
  tmnow = localtime(&tnow);
  tunow.c[0] = tmnow->tm_min;
  tunow.c[1] = tmnow->tm_hour;
  tunow.c[2] = tmnow->tm_mday;
  tunow.c[3] = tmnow->tm_mon;

  for(i=0; i<List_len; i++)
    if( List[i].TIME.l < tunow.l) List[i].STATE = OFF;

  for(i=0; i<List_len; i++){
    if(List[i].STATE == ON) break;
  }

  if(i<10){
    exe_t_pt = i;
    for(i++; i<List_len; i++)
      if( List[i].STATE==ON && List[i].TIME.l<List[exe_t_pt].TIME.l)
        exe_t_pt = i;
  }
  else
    exe_t_pt = -1;
}
```

```
int pos[6]={4,20,35,39,42,45};
void show_list_out(high_row, high_col, row, col, menu )
int high_row, high_col, row, col, menu;
{
  int i,j;
  for(i=0; i<List_len; i++){
     set(col+i,row,OFF,0);
     textattr(LIGHTGRAY + (BLUE<<4));
     cprintf("%2d:",i);

     if (!menu & i == high_row & high_col == 0)
       textattr(BLACK + (LIGHTGRAY<<4));
     else
       textattr(LIGHTGRAY + (BLUE<<4));
     set(col+i,row+pos[0],OFF,0);
     cprintf("%s",List[i].NAME);

     if (!menu & i == high_row & high_col == 1)
       textattr(BLACK + (LIGHTGRAY<<4));
     else
       textattr(LIGHTGRAY + (BLUE<<4));
     set(col+i,row+pos[1],OFF,0);
     cprintf("%s",List[i].NUM);

     if (!menu & i == high_row & high_col == 2)
       textattr(BLACK + (LIGHTGRAY<<4));
     else
       textattr(LIGHTGRAY + (BLUE<<4)),
     set(col+i,row+pos[2],OFF,0);
     cprintf("%s",month[List[i] TIME.c[3]]);

     if (!menu & i == high_row & high_col == 3)
       textattr(BLACK + (LIGHTGRAY<<4)),
     else
       textattr(LIGHTGRAY + (BLUE<<4));
     set(col+i,row+pos[3],OFF,0);
     cprintf("%-2d",List[i].TIME.c[2]);

     if (!menu & i == high_row & high_col == 4)
       textattr(BLACK + (LIGHTGRAY<<4));
     else
       textattr(LIGHTGRAY + (BLUE<<4));
     set(col+i,row+pos[4],OFF,0);
     cprintf("%02d",List[i].TIME.c[1]);
     textattr(LIGHTGRAY + (BLUE<<4));
     cprintf(":");

     if (!menu & i == high_row & high_col == 5)
       textattr(BLACK + (LIGHTGRAY<<4)),
     else
       textattr(LIGHTGRAY + (BLUE<<4));
     set(col+i,row+pos[5],OFF,0);
```

```
        cprintf("%02d",List[i].TIME.c[0]);
    }/* for i */
}

void outbound_call(){
    char key,key1;
    int  row=0,col=0;
    int pt,i;
    static char *help_name[7]
                    = {
                        "Move:   Arrow ",
                        "Select: Enter ",
                        "Set T.: Enter ",
                        "Delete: F3    ",
                        "Record: F4    ",
                        "PlayBk: F5    ",
                        "Quit:   Esc.  "
                       };

    gettext(10, 5, 10+50, 5+12, sv1);
    window_frame(LIST_OUT_COL,LIST_OUT_ROW,LIST_OUT_LEN,LIST_OUT_HIG,"Directory
",OFF);
    window_frame(MENU_COL, MENU_ROW, MENU_LEN, MENU_HIG, " HELP ",OFF);

    for (i=0; i<7; i++) {
      set(MENU_ROW+1+i, MENU_COL+1, OFF, 13);
      cprintf(" %-14s",help_name[i]);
    }

    do{
        show_list_out(row,col,10,6,OFF);
        key = get_char();
        switch(key){
        case FUN:
            key1 = get_char();
            switch(key1){
            case DOWN:
                if(++row>9) row = 0;
                break;
            case UP:
                if(--row<0) row = 9;
                break;
            case RIGHT:
                if(++col>5) col = 0;
                break;
            case LEFT:
                if(--col<0) col = 5;
                break;
            case F3:
                strcpy( List[row] NAME, "..........." );
                strcpy( List[row].NUM,". ... . .-. ..");
                List[row].TIME.l = 0x00010000;
```

```
            List[row].STATE = OFF;
            sprintf(filename,"%d.out",row);
            unlink(filename);
            break;
        case F4:
            sprintf(filename,"%d.out",row);
            message(2,24,40);
            record();
            message(0,24,40);
            break;
        case F5:
            sprintf(filename,"%d.out",row);
            message(1,24,40);
            playback();
            message(0,24,40);
            break;

        }
        break;
    case CR:
        textattr(BLACK + (LIGHTGRAY<<4));
        List[row].STATE = ON;
        switch(col){
        case 0:
           for(pt=0,key1=0; key1 != CR; ){
              set(LIST_OUT_ROW+row+1,LIST_OUT_COL+pos[col]+pt+1,ON,10);
              key1 = get_char();
              if( isalnum(key1) || key1==' ' || key1=='-' ){
                 for(i=13; i>pt; i--)
                    List[row].NAME[i] = List[row].NAME[i-1];
                 List[row].NAME[pt] = key1;
                 set(LIST_OUT_ROW+row+1,LIST_OUT_COL+pos[0]+1,ON,10);
                 cprintf("%s",List[row].NAME),
                 if(++pt==14) pt --;
              }
              else if( key1 == BACK ) pt--,
              else if( key1 == FUN ){
                 key1 = get_char();
                 switch(key1){
                 case LEFT:
                    if(--pt<0) pt++;break,
                 case RIGHT:
                    if(++pt>13) pt--,break;
                 case DEL:
                    for(i=pt; i<13; i++)
                       List[row].NAME[i] = List[row].NAME[i+1];
                    List[row].NAME[13] = ' ';
                    set(LIST_OUT_ROW+row+1,LIST_OUT_COL+pos[col]+1,ON,10);
                    cprintf("%s",List[row].NAME);
                    break;
              . case HOME:
                    pt=0; break;
```

```
            case END:
                pt=13; break;
            }
        }
    }
    break;
case 1:
    for(pt=0,key1=0; key1 != CR; ){
        set(LIST_OUT_ROW+row+1,LIST_OUT_COL+pos[col]+pt+1,ON,10);
        key1 = get_char();
        if( isdigit(key1) || key1==' ' || key1=='-' ){
            List[row].NUM[pt] = key1;
            set(LIST_OUT_ROW+row+1,LIST_OUT_COL+pos[col]+1,ON,10);
            cprintf("%s",List[row] NUM);
            if(++pt==14) pt --;
        }
        else if( key1 == BACK ) pt--;
        else if( key1 == FUN ){
            key1 = get_char();
            switch(key1){
            case LEFT:
                if(--pt<0) pt++;break;
            case RIGHT:
                if(++pt>13) pt--;break;
            case DEL:
                for(i=pt; i<13, i++)
                    List[row] NUM[i] = List[row].NUM[i+1];
                List[row].NAME[13] = ' ';
                set(LIST_OUT_ROW+row+1,LIST_OUT_COL+pos[col]+1,ON,10);
                cprintf("%s",List[row] NUM);
                break;
            case HOME:
                pt=0, break;
            case END:
                pt=13, break,
            }
        }
    }
    break;
case 2:
    if(++List[row].TIME.c[3]==12) List[row].TIME.c[3] = 0;
    break;
case 3:
    if(++List[row].TIME.c[2]==32) List[row].TIME.c[2] = 1;
    break;
case 4:
    if(++List[row].TIME.c[1]==24) List[row].TIME.c[1] = 0;
    break;
case 5:
    if(++List[row].TIME c[0]==60) List[row].TIME.c[0] = 0;
    break;
}
```

```
            break;
        }
    }while(key != ESC );

    get_near_t();

    puttext(10, 5, 10+50, 5+12, sv1);

}

void dialmenu()
{
    int i;
    char key,key1;
    char dialnum[13];

    int dial_row = 0, page_begin = 0, menu_choice = 0,
    int record_flag = OFF;      /* Off: record the other party */
    int menu = ON;              /* switch between menu and directory */

    gettext(DIR_COL+1, DIR_ROW, DIR_COL+1+DIR_LEN, DIR_ROW+DIR_HIG, sv1);
    window_frame(DIR_COL,DIR_ROW,DIR_LEN,DIR_HIG," Dial Directory ",OFF);

    window_frame(MENU_COL, MENU_ROW, MENU_LEN, MENU_HIG, " DIAL MENU ",OFF),

    while (1) {

      show_dial(dial_row, DIR_ROW+1, DIR_COL+1, page_begin, menu);
      show_menu(4, MENU_ROW+2, MENU_COL+2, menu_choice);

      set(MENU_ROW+10, MENU_COL+1, OFF, 13);
      textattr(LIGHTGRAY + (BLUE<<4));

      if (record_flag)
         if (record_flag == 1)
            cprintf("%-14s","* Record other "),
         else
            cprintf("%-14s","* Record self  ");
      else
         cprintf("%-14s","* No Recording ");

      key = get_char();
      if (menu == OFF) {
        switch ( key ) {
          case FUN:
            key1 = get_char();
            switch( key1 ) {
              case UP:
                if(dial_row > 0 ) dial_row--;
                else if (page_begin > 0) page_begin--;
                break;
              case DOWN:
```

```
                if(dial_row < 9 ) dial_row++;
                else if (page_begin < 90) page_begin++;
                break;
              case PgUp:
                if ((page_begin -= 10) < 0)
                    page_begin = 0;
                break;
              case PgDn:
                if ((page_begin += 10) > 90)
                    page_begin = 90,
                break;
              }
            break;

          case CR:          /* choice is made, do whatever */
                /* switch menu_choice */
              if (menu_choice == 2) { /* use directory */
                    do_function(PICK_UP);
                    message(5,24,30);
/*                  if(test_tone() == DIALTONE){*/
                    delay(500);       /* use this instead */
                          dial(table[dial_row+page_begin].NUM);
/*                  }*/
                    message(0,24,30),
                    message(8,24,30);
                    if (record_flag) {
                       getrecordname();
                       message(0,24,30);
                       message(2,24,30);
                       record();
                       message(0,24,30);
                       message(8,24,30);
                    }
                    while ( !keyhit() ),
                    message(0,24,30);
                    do_function(ON_HOOK);
                    do_function(MOUTH2CODEC);
              }
              menu = OFF;
              break;

          case ESC: menu = ON;

       }
     }
     else {
       switch ( key ) {
         case FUN:
           key1 = get_char();
           switch( key1 ) {
             case UP:
               if(menu_choice > 0 ) menu_choice--;
```

```
                break;
              case DOWN:
                if(menu_choice < 3 ) menu_choice++;
                break;
              }
            break;

          case CR:
             switch (menu_choice)  {
                case 0:  puttext(DIR_COL+1, DIR_ROW, DIR_COL+1+DIR_LEN,
                           DIR_ROW+DIR_HIG, sv1);
                         return;
                case 1: /* dial directly */
                         do_function(PICK_UP);
                         readkey(dialnum,11,20,6,"DIAL NUMBER");
                         message(5,24,30);
/*                       if(test_tone() == DIALTONE){*/
                                dial(dialnum);
/*                       }*/
                         message(0,24,30);
                         message(8,24,30);
                         if (record_flag) {
                            getrecordname();
                            message(0,24,30);
                            message(2,24,30),
                            record();
                            message(0,24,30);
                            message(8,24,30),
                         }
                         while ( !keyhit() );
                         message(0,24,30);
                         do_function(ON_HOOK);
                         do_function(MOUTH2CODEC);
                         break;
                case 2: /* use directory */
                         menu = OFF;
                         break;
                case 3: /* record select */
                         if (++record_flag > 2) record_flag = 0;
                         if (record_flag)
                            if (record_flag == 1)
                               do_function(LINE2CODEC);
                            else
                               do_function(MOUTH2CODEC);
                         else
                            do_function(MOUTH2CODEC);

             }

       }
    }
  }
```

```
}

int readkey(buff,len,x,y,title)
char buff[];
unsigned len;
int x,y;
char title[];
{
        char b[30],c;
        int i=0;

        gettext(y+1, x, y+len+1+strlen(title)+7, x+3, sv3);

        window_frame(y,x, len + strlen(title) + 7,3,"",ON);
        cprintf(" %s: ",title);
        do{
                c = get_char();
                switch(c){
                case ESC:
                        return(-1),
                case BACK.
                        if(i>0){
                                i--;
                                putchar(BACK),
                                putchar(' ');
                                putchar(BACK),
                        }
                        break;
                case FUN:
                        get_char();
                        break;
                case CR:
                        break,
                default:
                        if(i<len){
                                putchar(c);
                                b[i++] = c;
                        }
                }/* switch c */
        }while( c != CR );
        b[i] = '\0';
        b[len] = '\0';
        strcpy(buff,b);
        puttext(y+1, x, y+len+1+strlen(title)+7, x+3, sv3);
        return(strlen(buff));
}

void directory()
{
  int i;
  char key,key1;
```

```
int dial_row = 0, page_begin = 0, menu_choice = 0;

int menu = ON;              /* switch between menu and directory */

gettext(DIR_COL+1, DIR_ROW, DIR_COL+1+DIR_LEN, DIR_ROW+DIR_HIG, sv1);
window_frame(DIR_COL,DIR_ROW,DIR_LEN,DIR_HIG," Dial Directory ",OFF);

window_frame(MENU_COL, MENU_ROW, MENU_LEN, MENU_HIG, " MENU ",OFF);

while (1) {

  show_dial(dial_row, DIR_ROW+1, DIR_COL+1, page_begin, menu);
  show_menu(3, MENU_ROW+2, MENU_COL+2, menu_choice);

  key = get_char();
  if (menu == OFF) {
    switch ( key ) {
      case FUN.
        key1 = get_char();
        switch( key1 ) {
          case UP:
            if(dial_row > 0 ) dial_row--;
            else if (page_begin > 0) page_begin--,
            break;
          case DOWN:
            if(dial_row < 9 ) dial_row++;
            else if (page_begin < 90) page_begin++;
            break,
          case PgUp:
            if ((page_begin -= 10) < 0)
                page_begin = 0,
            break;
          case PgDn·
            if ((page_begin += 10) > 90)
                page_begin = 90;
            break;
          }
        break;

      case CR:          /* choice is made, do whatever */
            /* switch menu_choice */
          switch (menu_choice) {
            case 0:               /* case = 0, no directory */
               menu = ON;
               break;
            case 1:
               edit(dial_row, DIR_ROW+1, DIR_COL+1, page_begin);
               break;
            case 2·               /* insert */
               for(i = table_len-1,i > dial_row + page_begin;i--) {
                  strcpy(table[i] NAME, table[i-1] NAME);
                  strcpy(table[i] NUM, table[i-1] NUM),
```

```
                }
                strcpy( table[dial_row+page_begin].NAME, "....." ),
                strcpy( table[dial_row+page_begin].NUM, ". ... ...-....");
                break;
            case 3:             /* delete */
                for (i=dial_row + page_begin;i<table_len-1;i++) {
                    strcpy (table[i] NAME, table[i+1].NAME);
                    strcpy (table[i].NUM, table[i+1].NUM);
                }
                strcpy( table[table_len-1].NAME, "....." );
                strcpy( table[table_len-1].NUM, ". ... ...-....");
          }
          break;

        case ESC: menu = ON;

      }
    }
    else {
      switch ( key ) {
        case FUN:
          key1 = get_char();
          switch( key1 ) {
            case UP:
              if(menu_choice > 0 ) menu_choice--;
              break;
            case DOWN:
              if(menu_choice < 3 ) menu_choice++;
              break;
            }
          break;

        case CR:
            if (menu_choice == 0)  {
                puttext(DIR_COL+1, DIR_ROW, DIR_COL+1+DIR_LEN,
                      DIR_ROW+DIR_HIG, sv1);
                return;
            }
            menu = OFF;

      }
    }
  }
}

void interrupt ring(){
  disable();
  do_function(RES_DTMF_RX_INT);
  popkind = 'r';
  popflg = 1;
  outportb(PORT_8259_20, 0x20);     /* write eoi to 8259 */
  enable();
```

```
}

void popup()
{
  int i,j, xpos, ypos;
  char key,key1;

  int menu_choice = 0;

  menu = ON;

  xpos = wherex(),
  ypos = wherey();

  if (popflg == 'r') {
     message(9,3,32),
     message(0,3,32);
     set(0, 0, ON, 10);
     gotoxy(xpos, ypos);
     return;
  }

  gettext(MENU_COL+1, MENU_ROW, MENU_COL+1+MENU_LEN, MENU_ROW+MENU_HIG,
sv2);

  do{
    window_frame(MENU_COL, MENU_ROW, MENU_LEN, MENU_HIG, " MENU ",OFF);
    show_menu(0, MENU_ROW+2, MENU_COL+2, menu_choice),
    key = get_char(),
    switch (key) {
      case FUN:
        key1 = get_char();
        switch( key1 ) {
          case UP:
            if(menu_choice > 0 ) menu_choice--;
            break,
          case DOWN:
            if(menu_choice < 3 ) menu_choice++;
            break;
        }
        break;

      case CR:        /* choice is made, do whatever */
            /* switch menu_choice */
          switch (menu_choice) {
            case 0:
                  key = ESC;
                  break;
            case 1:  inbound_call(),
                  break;
            case 2:  outbound_call(),
                  break;
```

```
            case 3: window_frame(MENU_COL, MENU_ROW, MENU_LEN, MENU_HIG, " Make
Call ",OFF);

                    do {
                      show_menu(2, MENU_ROW+2, MENU_COL+2, menu_choice);
                      key = get_char();
                      switch (key) {
                        case FUN:
                          key1 = get_char();
                          switch( key1 ) {
                            case UP.
                              if(menu_choice > 0 ) menu_choice--;
                              break;
                            case DOWN:
                              if(menu_choice < 3 ) menu_choice++;
                              break;
                          }
                          break,

                        case CR:        /* choice is made, do whatever */
                              /* switch menu_choice */
                            switch (menu_choice) {
                              case 0: break;
                              case 1: dialmenu(); /* dial */
                                      break;
                              case 2: auto_redial();
                                      break,
                              case 3: directory(),
                            }
                      }
                    } while (menu_choice!=0 | key != CR);

          }
       }/* switch key */
  }while(key != ESC);
  puttext(MENU_COL+1, MENU_ROW, MENU_COL+1+MENU_LEN, MENU_ROW+MENU_HIG,
sv2);
  set(0, 0, ON, 10);
  gotoxy(xpos, ypos);

  if( (fp = fopen("table.txt","wb")) == NULL ) {
    printf("can not write table.txt!"),
    return;
  }
  else {
    fwrite(table,1,table_len*sizeof(struct sheet),fp);
    fclose(fp);
  }

  if( (fp = fopen("list.txt","wb")) == NULL ) {
    return;
  }
```

```
  else {
    fwrite(List,1,List_len*sizeof(struct List_struct),fp);
    fclose(fp),
  }
}

/* -- start-up function, to be used in TSR application -- */
void openfiles()
{
      int i;
      FILE *fp;

      /* open table.txt */
  if( (fp = fopen("table.txt","rb")) == NULL ){
            if( (fp = fopen("table.txt","wb")) == NULL ){
                  printf("can not write table.txt'\n");
                  return;
            }
            else{
                  for(i=0; i<table_len; i++){
                        strcpy( table[i].NAME, "  . .. ....." );
                        strcpy( table[i].NUM," .   ..-....");
                  }/* for i */
                  fwrite(table,1,table_len*sizeof(struct sheet),fp);                  fclose(fp);
                  fclose(fp);
            }/* if fp w */
  }
  else{
            fread(table,1,table_len*sizeof(struct sheet),fp);
            fclose(fp);
  }

  if( (fp = fopen("list.txt","rb")) == NULL ){
            if( (fp = fopen("list txt","wb")) == NULL ){
                  printf("can not write list.txt!\n");
                  return;
            }
            else{
                  for(i=0, i<List_len; i++){
                        strcpy( List[i] NAME, "... ....." );
                        strcpy( List[i].NUM,". ... . .- ...");
                        List[i].TIME.l = 0x00010000;
                        List[i] STATE = OFF;
                  }/* for i */
                  fwrite(List,1,List_len*sizeof(struct List_struct),fp);
                  fclose(fp);
            }/* if fp w */
  }
  else{
            fread(List,1,List_len*sizeof(struct List_struct),fp);
            fclose(fp),
  }
```

```
    get_near_t(),

}

void closefiles()
{
}
```

Bibliography

[1] Motorola *Data Book.*, 1984.

[2] Silicon Systems *Data Book.*, 1990.

[3] NEC Electronics Ins. *Microcomputer Products.*, 1984.

[4] Markel, J.D.and Gray, A.H. *"Linears prediction of speech", Springer Verlag, Berlin.* 1976.

[5] Witten, I.H.*Principles of Computer Speech.*,1982.

[6] Viswanathan,R.,and J. Makhoul. *"Quantization Properties of Transmission Parameters in Linear Predictive Systems." IEEE Trans. Acous., Speech and Signal Proc., Vol. ASSP-23:309-321* June 1975.

[7] *Documentation of the Government Standard LPC-10 Algorithm.*

[8] Singhal,S. and B.S.Atal. *Improving Performance of Multi-Pulse LPC Coders at Low Bit Rates." Proc. IEEE Int. Conf.Acous., Speech, and Signal Proc.*(march 1984):1.3 1-1.

[9] Sugamura, N., and F Itakura. *'Speech Data Compression by LPC Analysis-Synthesis Technique." Trans. IECE '81/8, Vol. J64-A, No.8.599-606.*