DEVELOPMENT OF A PROGRAMMABLE TIME-DOMAIN SPEECH-SYNTHESIS SYSTEM.

Karel Nicolaas van der Walt

Dissertation submitted in compliance with the requirements for the Master's Degree in Engineering in the Department of Electrical Engineering at the Technikon OFS.

July 1995

Supervisor: Mr. G. D. Jordaan

Acknowledgments

I would like to thank the following people who helped make this project a reality:

First off all, a big thank you to my wife, Magda, for all her help and support.

Thank you to all my colleagues off department Bio engineering for their support and ideas, especially Cassie, who was always willing to supply me with source material for my sound sampling experiments. Thank you also to Kobus who is always willing to lend a helping hand with the mechanical aspects of my projects.

Thank you to department Bio physics for allowing me to make use of their equipment. Thank you Thys for scanning the photos in chapter 6.

Most of all I would like to thank my project leader, Mr. GD Jordaan, without whose tireless help I would never have been able to complete these pages.

Extract

Problem

Many existing speech-synthesis systems are limited by one or more of the following disadvantages:

- expensive
- inflexible
- difficult to program
- often limited to an Anglo-Saxon language
- resultant speech is often unnatural and "mechanical" in nature
- some speech synthesizers are difficult to control and require a microprocessor
- a thorough phonetic knowledge is sometimes a prerequisite for programming of systems

Hypothesis

The purpose of this study was to design a speech-synthesis system which overcome as much of the above mentioned problems as possible. The following features were considered for such a system:

- The use of electronic recordings of real speech in Erasable Programmable Read Only Memory, instead of a conventional speech-synthesizer. This allows natural speech in any language without any prior phonetic knowledge.
- An unlimited vocabulary is provided as the user is able to make recordings himself.
- User-friendly software greatly simplifies programming of the system.
- Design of the system is such that it run independent of a computer once programmed.

Method of research

By means of experimental research a hardware and software prototype was developed to demonstrate and evaluate the working principles of such a system.

Results and conclusion

A very flexible system was developed which is easy to program and use. Clearly understandable speech may be reproduced in any language without specialized technical or phonetic knowledge.

Uittreksel

Probleemstelling

Verskeie bestaande spraaksintese sisteme word beperk deur een of meer van die volgende nadele:

- duur
- onbuigbaar in gebruik
- moeilik om te programmeer
- dikwels beperk tot 'n Anglo-Saksiese taal
- resulterende spraak is dikwels onnatuurlik en masjienagtig
- spraaksintetiseerders is dikwels moeilik om te beheer en vereis die gebruik van 'n mikroverwerker
- 'n deeglike kennis van fonetika mag 'n voorvereiste wees vir die effektiewe programmering van sommige sisteme

Hipotese

Die doel van die studie was die ontwerp van `n spraaksintese stelsel wat soveel moontlik van genoemde probleme oorkom.

Die volgende is as eienskappe van so `n sisteem oorweeg:

- Die gebruik van elektroniese opnames van werklike spraak in Uitwisbare Programmeerbare Lees Alleen Geheue, in plek van `n konvensionele spraaksintetiseerder. Dit verseker spraak in enige taal sonder dat enige fonetiese kennis nodig is.
- 'n Onbeperkte woordeskat word verseker deurdat die gebruikers self sy eie opnames kan maak.
- Gebruikersvriendelike sagteware vereenvoudig grootliks die programmering van die sisteem.
- Ontwerp van die sisteem is sodanig dat dit onafhanklik van `n rekenaar gebruik kan word, sodra dit geprogrammeer is.

Metode van ondersoek

Deur middel van eksperimentele navorsing is `n harde- en sagteware prototipe ontwikkel om die werksbeginsel van so `n stelsel te demonstreer en te evalueer.

Resultate en gevolgtrekking

`n Baie buigbare sisteem is ontwikkel wat maklik is om te gebruik en maklik is om te programmeer. Duidelik verstaanbare spraak kan in enige taal weergegee word sonder enige gespesialiseerde tegniese of fonetiese kennis.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION **1**

1.1 PURPOSE OF THIS STUDY 2

1.2 METHOD OF RESEARCH 2

1.3 PROBLEMS ENCOUNTERED. 2

1.4 CONCLUSION 3

CHAPTER 2

IN PURSUIT OF ARTIFICIAL SPEECH **4**

2.1 HUMAN SPEECH PRODUCTION 4

2.2. EARLY SPEECH SYNTHESIS 5

2.2.1 Once upon a time... 5

2.2.2 Early electronic speech. 6

2.3 MODERN RESULTS 7

2.3.1 Direct waveform-digitization 8

2.3.2 Phoneme synthesis 10

2.3.3 Linear-predictive coding 13

2.4 CONCLUSION 15

CHAPTER 3

PULSE CODE MODULATION **16**

3.1 INTRODUCTION 16

3.2 PULSE CODE MODULATION AS RECORDING MEDIUM. 16

3.2.1 Alias distortion 16

3.2.2 Quantization error 19

3.2.3 Solutions to quantization noise 21

3.2.3.1 Companders 21

3.2.3.2 Companding ADC/DAC 22

3.2.3.3 Pre-emphasis 25

3.3 CONCLUSION 25

CHAPTER 4

SYSTEM HARDWARE 26

4.1 INTRODUCTION 26

4.2 SYSTEM DESCRIPTION 26

4.2.1 Choice of sampling frequency 28

4.2.2 Choice of word length 29

4.2.3 Reduction of quantization noise 29

4.2.4 Distinguishing between different messages 30

4.3 RECORD/PLAYBACK/PROGRAMMER MODULE 31

4.3.1 Input path 31

4.3.1.1 Input amplifiers 31

4.3.1.2 Amplitude control 32

4.3.1.3 Low-pass filter 32

4.3.1.4 Compressor 34

4.3.1.5 Analogue-to-Digital Converter 34

4.3.2 Output path 35

4.3.2.1 Digital-to-Analogue Converter 35

4.3.2.2 Low-pass fi....36

4.3.2.3 Expander....36

4.3.2.4 Amplitude control....36

4.3.2.5 Output amplifiers....37

4.3.3 EPROM programmer and PIA....37

4.3.4 Voltage regulators....38

4.4 COMPUTER INTERFACE CARD....38

4.5 SPEECH CIRCUIT....39

4.5.1 Operation of the Speech Circuit....39

4.6 CONCLUSION....45

CHAPTER 5

SYSTEM SOFTWARE....47

5.1 INTRODUCTION....47

5.2 SOFTWARE DESCRIPTION....48

5.2.1 Variables and Functions....49

5.2.1.1 Record Buffer....49

5.2.1.2 Record Buffer Pointer....49

5.2.1.3 Record Monitor....49

5.2.1.4 Programme Status....49

5.2.1.5 Reset EPROM address....50

5.2.1.6 Clock EPROM address....50

5.2.1.7 Write EPROM / Read EPROM....50

5.2.1.8 EPROM Size....50

5.2.1.9 LIMIT1 and LIMIT2....50

5.2.2 Subroutines....51

5.2.2.1 Interrupt Service Routine....51

5.2.2.2 RECORD 52

5.2.2.3 PLAY 54

5.2.2.4 MUTE/LISTEN 55

5.2.2.5 COPY 55

5.2.2.6 MOVE 55

5.2.2.7 DELETE 56

5.2.2.8 SAVE 57

5.2.2.9 LOAD 58

5.2.2.10 PATH 58

5.2.2.11 WriteE 59

5.2.2.12 ReadE 62

5.2.2.13 ListE 62

5.2.2.14 HELP 62

5.2.2.15 128K/256K 62

5.2.2.16 QUIT 62

5.3 CONCLUSION 64

CHAPTER 6

EVALUATION 65

6.1 INTRODUCTION 65

6.1.1 Computer interface 65

6.1.2 Record/Playback/Programmer module 65

6.1.3 Speech circuit 67

6.1.4 Software 67

6.2 USING THE SYSTEM 67

6.2.1 Recording, playback and programming. 67

6.2.2 Speech circuit in use 69

6.3 EVALUATING THE SYSTEM.....69

6.3.1 Measurements.....69

6.3.2 Aural tests.....75

6.4 CONCLUSION.....76

CHAPTER 7

SUMMARY.....78

APPENDIXES

APPENDIX A.....79

APPENDIX B.....97

APPENDIX C.....104

REFERENCES.....182

LIST OF DIAGRAMS

Chapter 2

Figure 2.1 Simplified electronic representation of the human speech production system.....5

Figure 2.2 The first electronic device to produce individual speech sounds.....6

Figure 2.3 Block diagram of the 'VODER' built by H Dudley in 1939.....7

Figure 2.4 National Semiconductor Digitalker.....9

Figure 2.5 Phoneme-based SPO256 speech synthesizer.....12

Figure 2.6 Votrax SC-01 phoneme speech synthesizer.....13

Figure 2.7 Ten-stage lattice filter as used in LPC synthesizers.....14

Chapter 3

Figure 3.1 Showing the sampling process in (a) time domain and (b) frequency domain.....17

Figure 3.2 If sampling frequency is too low, aliasing occurs.....18

Figure 3.3 Alias distortion.....18

Figure 3.4 Quantization stage characteristic.....19

Figure 3.5 Probability density function of quantization error.....20

Figure 3.6 (a) Input to compressor, (b) Output from compressor.....21

Figure 3.7 Compressor time constants.....23

Chapter 4

Figure 4.1 Speech-circuit pulse diagram.....44

Chapter 5

Figure 5.1 Appearance of the system software.....48

Figure 5.2 Interrupt service routine flowchart.....52

Figure 5.3 RECORD procedure flowchart.....53

Figure 5.4 PLAY procedure flowchart.....54

Figure 5.5 COPY procedure flowchart....56

Figure 5.6 SAVE procedure flowchart....57

Figure 5.7 LOAD procedure flowchart....58

Figure 5.8 WriteE procedure flowchart....60

Figure 5.9 ReadE procedure flowchart....63

Chapter 6

Figure 6.1 Computer Interface Card....66

Figure 6.2 Record/Playback/Programmer module....66

Figure 6.3 Speech circuit together with memory expansion unit....68

Figure 6.4 Software as seen on a computer monitor....68

Figure 6.5 Setup of complete system....70

Figure 6.6 Speech circuit under aural evaluation....70

Figure 6.7 1kHz reference input signal....72

Figure 6.8 1kHz signal recovered from the DAC....73

Figure 6.9 1kHz signal at output of system....74

Figure 6.10 Frequency response of input/output low pass filter....75

CHAPTER 1

INTRODUCTION

SPEECH-SYNTHESIS — the artificial generation of that series of sounds known as 'speech' — has advanced beyond the novelty stage to become a real alternative to the simple audible and visible indicators and displays common to so much of modern society. There is very little that can be done with a light or buzzer that can't be done better with a 'spoken' word.

'Toys' such as Texas Instruments Speak and Spell have been recognized as effective learning tools. Through the electronic mouths of those machines, children are exposed to new words in an exciting and interactive way. Talking calculators (Williams, 1994: 67) and timepieces have expanded the horizons of the blind. Pilots and drivers are relieved of the need to watch their meters and gauges continuously as alarms can be given with instructions as to what actions should be taken. Speech-synthesis devices are finding a myriad of uses in communications, appliances, automotive applications, clocks, instrumentation, language translators etc. (Savon, 1982: 62).

Despite the obvious advantages speech-synthesis provide, many systems are limited in their application possibilities due to one or more of the following disadvantages:

- expensive
- inflexible
- difficult to program
- limited to an Anglo-Saxon language
- resultant speech is unnatural and 'mechanical' in nature
- speech synthesizer is difficult to control and require a microprocessor

- a thorough phoneti… …quisite for programming the system

1.1 Purpose of this study

The purpose of this study was to design a speech-synthesis system which overcome as much of the above mentioned problems as possible. The following features were proposed for such a system:

- The use of electronic recordings of real speech, instead of a speech-synthesizer. This allows natural speech in any language without any prior phonetic knowledge.
- An unlimited vocabulary is provided as recordings are made by the user himself.
- User-friendly software greatly simplifies programming of the system.
- Design of the system is such that the final speech-circuit, once programmed, runs independent of a computer.

1.2 Method of research

Studying the available literature provided a background for initial ideas. Experimental research turned the initial ideas into a prototype. Software was generated through study and experimental research to test the validity of the design. Eventually hardware and software was integrated into a final working prototype.

1.3 Problems encountered.

- During the initial stages of the project, the high capacity EPROM's (Erasable Programmable Read Only Memory) used in the circuit, was relatively expensive and not freely available.

1.4 Conclusion

Existing speech-synthesis systems are limited by a number of disadvantages as discussed.

Through this study an attempt was made to provide a relatively cheap speech-synthesis system which provide clear, natural speech in any language. The resulting system is easy to program by the user, as no prior technical or phonetic knowledge or experience is needed.

CHAPTER 2

IN PURSUIT OF ARTIFICIAL SPEECH

2.1 Human speech production

To produce speech sounds, the air from the lungs is forced through the vocal cords, or folds, which are located in the larynx. As the air flow builds up, complex pressure differences are produced by the nature of the glottis, causing the vocal folds to vibrate in a manner similar to the reed in a wind instrument. This process is known as phonation and sounds produced in this way are termed `voiced` sounds. The vowel sound's frequency is determined by the tension in the vocal folds. The range of pitch (vibration) for adults is typically two octaves with the range for females being about an octave higher than for males. Another source of sound generation is breath noise. Sounds produced in this manner are termed `un-voiced` sounds such as `s` and `ch` (Owens, 1993: 3).

As these sounds produced by the vibration of the vocal cords or breath noise pass along the vocal tract and out through the mouth, the characteristics of speech are impressed upon them by two types of modulation processes which produce forms of resonance known as formants. One source of resonance can be described as a shaping of the energy-frequency distribution and is achieved by the passage of the sound through the multiple resonance chamber formed by the mouth and tongue. Further shaping may be affected by the lips and teeth. The other modulation process is achieved by closing off the vocal tract by the tongue, lips or teeth and then releasing the sound energy. The sounds thus produced are termed plosives or stops, of which `p` and `t` are examples (Owens, 1993: 3).

The human vocal tract is capable of making an infinite number of distinct sounds. At the linguistic level the basic unit of speech is known as the phoneme. The phoneme is considered as a working definition of the perceptual unit of language and the manifestation of each phoneme depends on the word being spoken and the position of the phoneme within the word. The vocal tract may be

reproduced electronically by emulating the various functions of the human voice production process as illustrated in Figure 2.1 (Owens, 1993: 4).

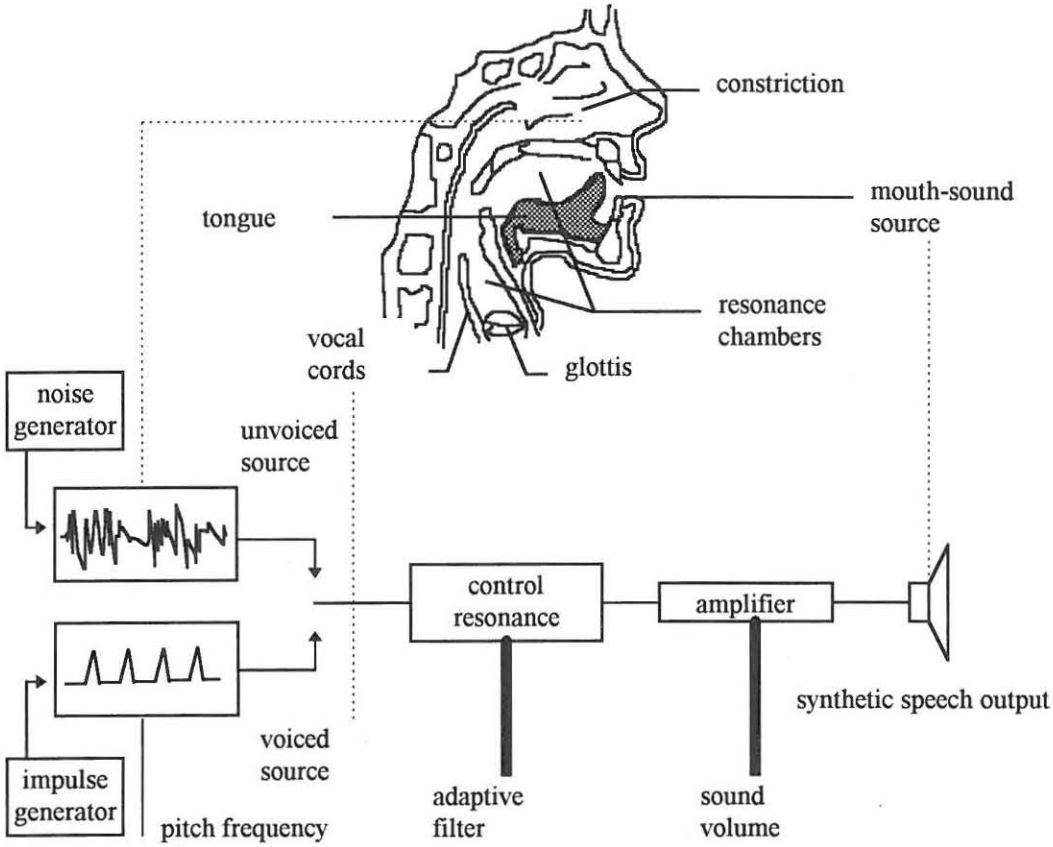


Figure 2.1 Simplified electronic representation of the human speech production system.

2.2. *Early speech synthesis*

2.2.1 Once upon a time...

In 1779 a prize was offered by the Imperial Academy of St. Petersburg for a scientific explanation of the physiological differences between the five vowel sounds and demonstrating an apparatus for producing the sounds. By 1780 Professor Christian Kratzentein had designed a `Vox humana` capable of producing the vowel sounds from a set of different shaped tubes. Some unusual shapes were created in an attempt to emulate the human vocal tract (Wheddon and Linggard, 1990: 7).

In 1791 Wolfgang Ritter Von K... talking machine which he began designing in 1769; it consisted of a bellows, a mouth shape, nostrils and whistles. The machine included a compressible leather tube and an air chamber equipped with a reed leading to a soft leather resonator which could be manually shaped for the formation of the vowel sounds. Consonants were created by holes which the `player` closed by movement of the fingers. The Von Kempelen machine could produce about twenty different sounds! (Furui, 1989: 206 and Morgan, 1984: 2).

2.2.2 Early electronic speech

In 1936 K W Wagner constructed an electric speech synthesizer consisting of a pulse generator, connected to a series of formant filters. Figure 2.2 shows the synthesizer in the form of a block diagram (Baumann, 1981:5.18).

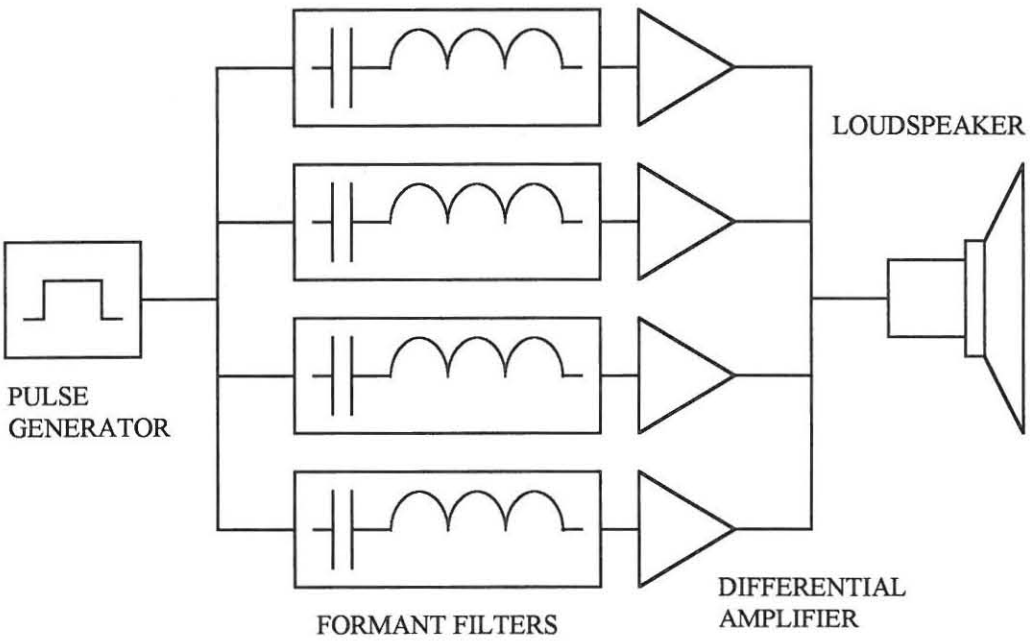


Figure 2.2 The first electronic device to produce individual speech sounds.

This simple device was able to imitate all the vowels and several voiced consonants (lll, mmm, nnn) surprisingly well (Baumann, 1981: 5.17).

A few years later Homer Dudley, [illegible]e Laboratories, managed to go one step further. Based on the Wagner device, he built the `VODER` (Voice Operation DEmonstratoR) in 1939. This speech synthesizer could successfully imitate all the vocal sounds, provided it was skillfully operated. The `VODER` was not a standalone talking machine, it was played like an instrument by a trained operator. The resulting acoustic output, however, bore a striking resemblance to the human voice (Morgan, 1984: 2). As Figure 2.3 shows, the `VODER` is such a complicated device, all things considered, that the operator has to be a real artist to be able to control it skillfully (Bristow, 1986: 7).

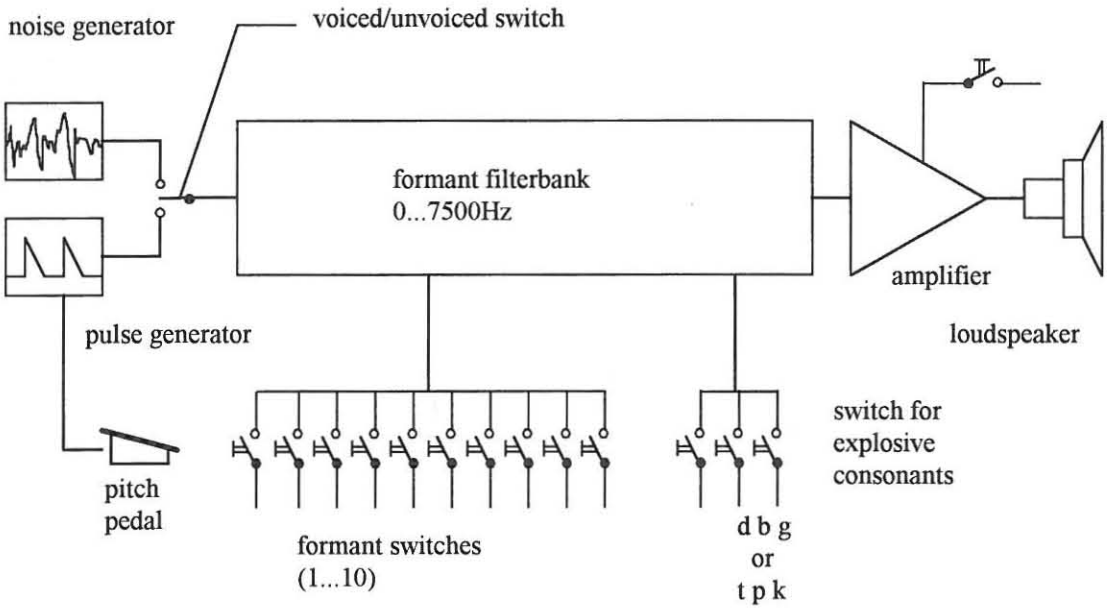


Figure 2.3 Block diagram of the 'VODER' built by H Dudley in 1939.

2.3 Modern results

Surprisingly, the fundamental theories involved in speech synthesis have not changed greatly from those used at the time of the `VODER`.

While there are more similarities than differences among the approaches to speech synthesis, they can be categorized into two main groups:

- Time-domain synthesis ... subdivided into:
  - Direct waveform-digitization which is a high bit-rate technique
  - Phoneme synthesis which is a low bit-rate technique
- Frequency-domain synthesis
  - The main frequency-domain synthesis technique is linear-predictive coding which is an electronic model of the vocal tract (Savon, 1982: 65 and Morgan, 1984: 17).

2.3.1 Direct waveform-digitization

Direct waveform-digitization is the process of taking a live or recorded speech-waveform and passing it through an analogue-to-digital converter. The final quality of the reconstructed speech (obtained by passing the digitized data through a digital-to-analogue converter) depends on several factors, including the rate at which the speech waveform is sampled and the number of amplitude levels into which it is segmented. As the number of levels increases, more information must be processed and stored in memory; as the number of levels decreases the memory and processing requirements decrease — at the expense of intelligibility (Savon, 1982: 63);

It is necessary to sample the waveform at a frequency that is at least twice as high as the highest frequency in the original waveform[1] . If the sampling frequency is too low, aliasing, which causes the higher frequencies to appear falsely as lower frequencies, takes place (Watkinson, 1991, 14).

Playback of the digitized signal is (in theory) a simple matter of recalling the sequences of data stored in memory and processing them with a digital-to-analogue converter (Kaufman and Seidman, 1988: 24).

[1] see paragraph 3.2.1

Direct waveform-digitization of human speech is highly redundant, wasting a lot of memory space storing drawn-out sounds like `oooooh` and 'aaaaaah'. Methods used to reduce the memory requirements and rate of processing of redundant information may actually cause this synthesis technique to be classified in one of the other simulation groups (Savon, 1982, 63).

National Semiconductor uses a direct-digitization method that reconstructs the speech using pulse-code modulation[2]. It calls its system the *Digitalker*. Compressed speech data is stored along with frequency and amplitude information in a read-only-memory (ROM). Figure 2.4 shows a block diagram of the speech processor chip, referred to as the SPC (Sokolowski, 1990: 201).

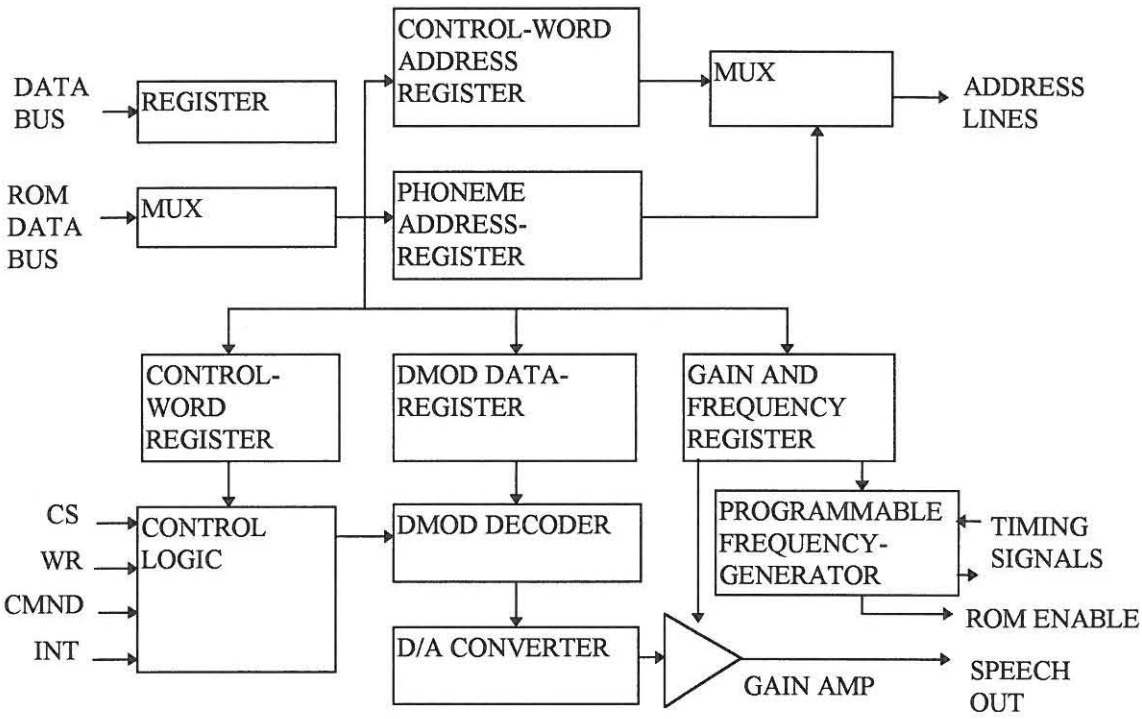


Figure 2.4 National Semiconductor Digitalker.

The key to the practicality of this, as well as many other systems, is speech-compression — coding of the signals that minimizes the redundant information. National points out that its system produces speech quality far better than the crude sound associated with early demonstrations of speech digitization. In terms of memory space, the result is that male voices require about 100 bits per word

[2] see chapter 3

and female voices somewhat more. The system is more like a digital recorder — that digitizes actual voices, stores, and then plays back — than the other methods which model the vocal tract (Sokolowski, 1990: 198).

The compression method combines three techniques. First, redundant pitch periods are removed. Then, adaptive delta-modulation, which uses the difference between two successive sampling points, rather than the value at those points, is used to conserve memory space by storing only the difference information. Phase adjustments and half-period zeroing, remove the `direction` component of the waveform; that information is not needed for intelligibility, so can be safely done away with. Computer processing is used to accomplish the compression. National claims that its compression scheme, combined with waveform digitization, is very competitive with other systems, including linear-predictive coding (Jim②nez, 1991: 17).

The *Digitalker* is programmed with control information that instructs it how many times to repeat a specific waveform. A programmable frequency generator is used to add inflection. The system is easy to use because it requires only a start pulse and an 8-bit address to trigger any message. Simple switches can be used if a microprocessor is not justified. One *Digitalker* can produce up to 256 different messages. (Savon, 1982: 63).

2.3.2 Phoneme synthesis

Phoneme synthesis works by combining basic sound elements (phonemes) into complete words and sentences. In theory any spoken word can be synthesized by stringing phonemes together. The quality of the resulting speech depends on the extent of the phoneme library. To cover a wide range of inflection requires a correspondingly large number of phonemes. Compromise is necessary to put together a practical system. Redundancy-elimination coding techniques are used (Walker, 1984: 24).

The phoneme method is particu... ...e extent or type of vocabulary required is not fixed: When there is no definite vocabulary list, speech cannot be constructed from stored words and phrases but must be generated from the more elemental phoneme sounds (Walker, 1984: 24).

General Instruments (GI) has taken the approach just described, combining phoneme synthesis with a digital filter. The design is versatile enough to operate in linear-predictive filter[3] mode. GI has released a product specification that describes the SPO256 speech-processor IC as an LSI n-channel metal-gate device that can synthesize up to 256 sound sequences (Jim②nez, 1991: 5).

Figure 2.5 shows a block diagram of the processor (Savon, 1982: 64). The speech process is started by addressing the ROM location that contains the phoneme desired. Up to 256 phonemes can be stored in the 16K bits of the on-board ROM, but that can be extended to as many as 3825 phonemes (or, more usually, complete words or phrases) through the addition of up to 491K bits of ROM.

The device includes a controller and a vocal-tract model (VTM). The VTM is a digital filter similar to that used in linear-predictive coding. The system generates complex sound-sequences under the control of 15 slowly varying parameters including: repeat count, pitch period, source amplitude, and 13 digital filter-coefficients. The controller is a sequential processor that gets its instructions and data from the ROM and alters the contents of the 12 parameter-registers controlling the VTM (Jim②nez, 1991: 6).

One-byte inputs specify 256 entry points. The entry points in the internal ROM are spaced by 8-byte increments, meaning that each phoneme can be defined by 8 bytes (64 bits). Expanding the input format to 2 bytes increases the number of entry points. The controller has 16 executable instructions and supports one level of subroutine nesting. JMP (JuMP) and JSR (Jump to Sub Routine) instructions are included to allow chaining of program segments and the reuse of code sequences (Savon, 1982: 64).

---

[3] see paragraph 2.3.3

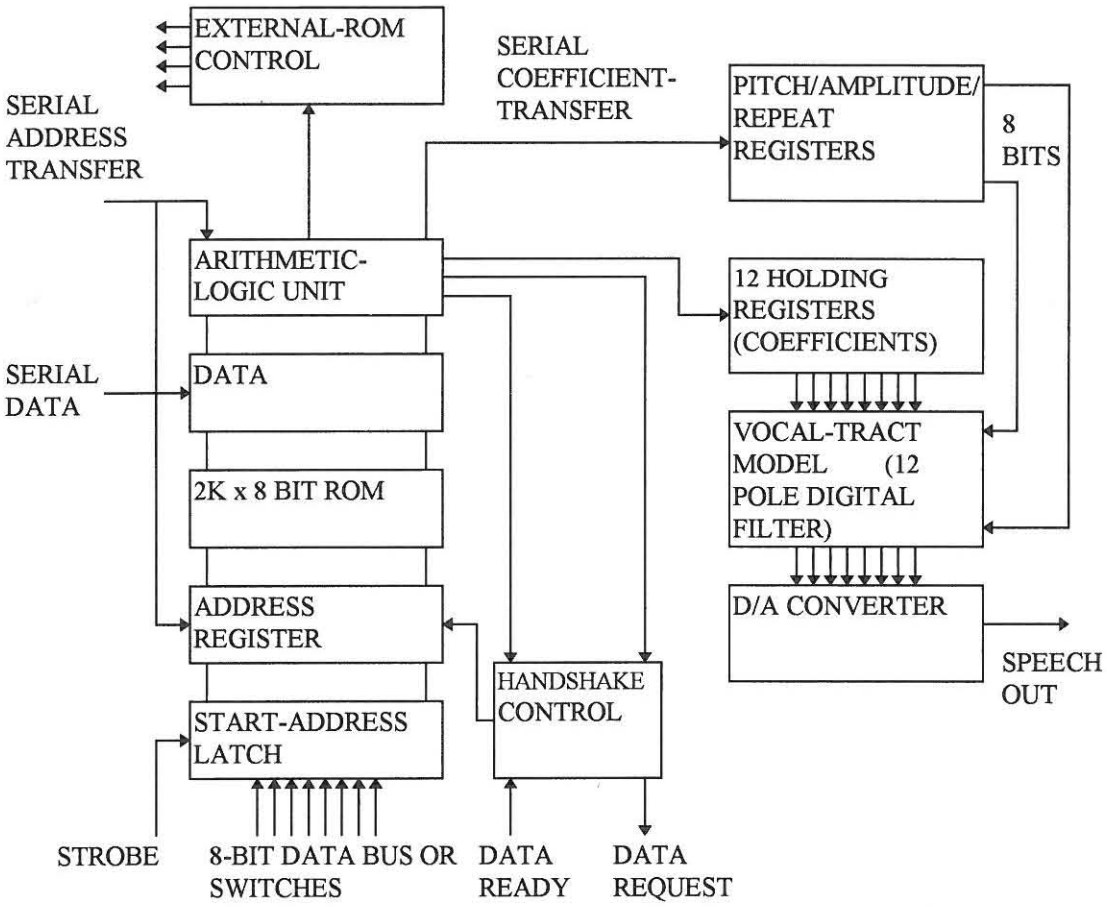


Figure 2.5 Phoneme-based SPO256 speech synthesizer.

External ROM interfaces to the speech-processor device through serial input and output lines. Sixteen-bit serial addresses are used for addressing. Serial-to-parallel conversion is handled internally by the SPO256. The analogue output is developed by a 7-bit pulse width modulation, digital-to-analogue converter. The output is passed through a 5-kHz-cutoff low-pass filter and is then amplified externally (Jim②nez, 1991: 7).

Votrax offers another phoneme-based system consisting of a switched capacitor filter in a CMOS-technology LSI integrated circuit. Phonemes require an average of 6 bits and each phoneme is 40 to 200 milliseconds long. The device has a 64-phoneme library accessed by a 6-bit code.

Typical speech-data uses 70 bits per second. Votrax plans to provide a text-to-phoneme translator algorithm that will let the user do his own programming. Figure 2.6 shows a block diagram of the system (Reese and Keller, 1982: 57).

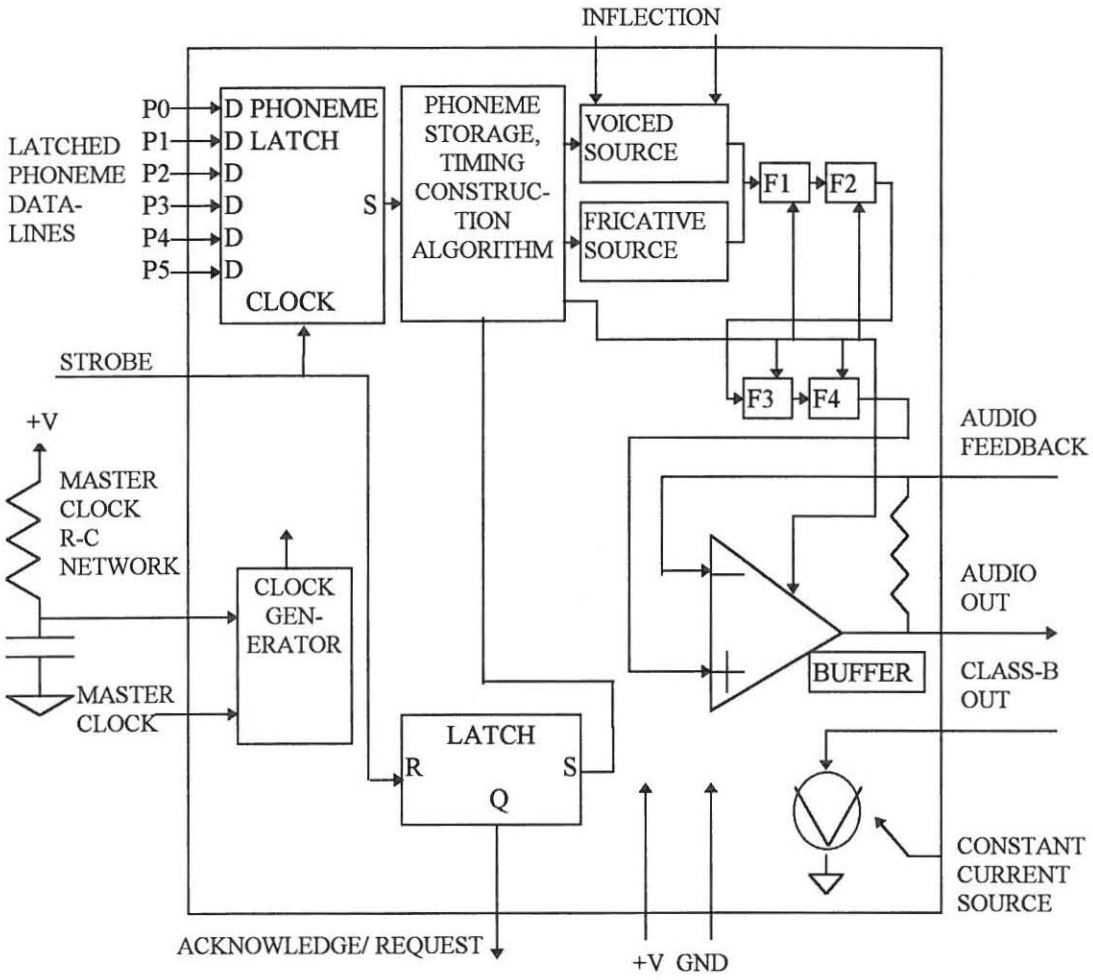


Figure 2.6 Votrax SC-01 phoneme speech synthesizer.

2.3.3 Linear-predictive coding

Linear-predictive coding (LPC)—used extensively by Texas Instruments—is a combination of techniques that model the vocal tract electronically. Noise and tonal sources generate signals that are processed by the LPC filter. The method derives its name from the fact that it predicts the parameters

of the next speech sample using a linear combination of the preceding speech samples. That results in a major reduction in the amount of memory required for the storage of speech data (Furui and Sondhi, 1992: 209).

Speech-synthesis techniques using this method require as few as three integrated circuits: a digital lattice filter, a ROM, and a controller. Present systems are based on the 10-stage filter shown in Figure 2.7: its output is set by pitch, amplitude and filter coefficients (Savon, 1982: 65).

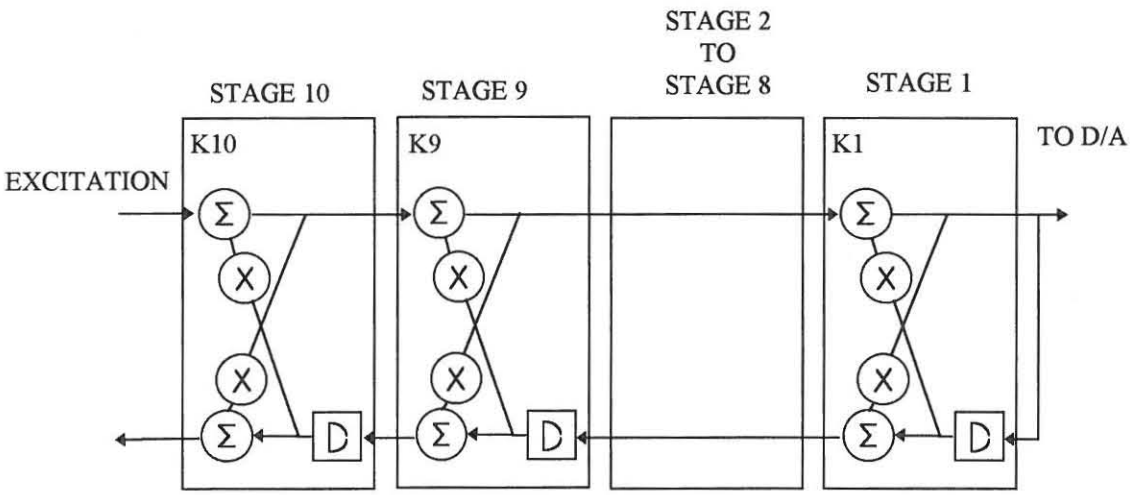


Figure 2.7 TEN-STAGE LATTICE FILTER as used in LPC synthesizers.

The lattice-filter structure includes multiplication, summation, and delay blocks. Digital filters are constructed of memory-register delay components and of summers and multipliers connected in either feedback or non-recursive, direct path configurations. Multiplication takes longer than addition, so pipeline techniques are used to synchronize the filter's operation to the sampling rate of the system (Savon, 1982: 65).

The voice-synthesis processor performs about 400 000 multiplication and additions each second. A frame (complete set of data) is supplied to it about every 25 milliseconds. Data from adjacent frames is interpolated about every 3 milliseconds to generate a smooth output. A tradeoff is made among

flexibility, fixed or variable vocabulary, speech quality, and cost-per-second of speech output (Savon, 1982: 65).

A computer program is used to produce an optimal set of coefficients for the vocal tract filter, the 8-bit energy and pitch information, and to generate the one-bit repeat codes. A technician corrects any audible deficiencies (Savon, 1982: 65).

2.4 Conclusion

Mechanical speech has a long and fascinating history. This century saw the introduction of the first connected speech synthesizer, the `VODER`. Recent advances in IC technology have permitted complicated systems similar to the `VODER` to be placed on a single piece of silicon (Morgan, 1984: 6).

Speech synthesis systems can be categorized into two main groups:

- Time-domain synthesis
    - Direct waveform-digitization which is a high bit-rate technique
    - Phoneme synthesis which is a low bit rate technique
- Frequency-domain synthesis
    - The main frequency-domain synthesis technique is linear-predictive coding which is an electronic model of the vocal tract.

CHAPTER 3

PULSE CODE MODULATION

3.1 Introduction

The principle of pulse code modulation (PCM) was first suggested by Reeves in 1938, and is now widely used for feeding analogue signals into computers and other digital equipment for subsequent processing. PCM, as waveform coder, attempt to copy the actual shape of the waveform produced by, for example, a microphone and its associated analogue circuits (Holmes, 1988: 56). This is done by sampling the input signal at regular time intervals. The analogue input signal can assume an infinite number of different levels between the two limit values which define the range of the signal. As a correspondingly infinite number of digital codes would be impractical, PCM uses a fixed range of levels to which sampled values are rounded or quantized (Roddy and Coolen, 1984: 608). The number of levels depends on the accuracy required in a specific application. The samples need to be taken frequently enough to capture the quickest variation in the input signal.

3.2 Pulse code modulation as recording medium.

The waveform recovered from a digital sampling process is never quite the same as the waveform that went in. Sampling by its very nature, breaks continuous waveforms into discontinuous elements. Without taking special precautions, the result will be recognizable, but will be noisy and distorted (Chappell, 1987: 11). This waveform degradation is mainly caused by alias distortion and quantization noise.

3.2.1 Alias distortion

Although the sampling operation may seem to introduce a rather drastic modification of the input

signal (as it ignores all the signal changes that occur between the sampling times), it can be shown that the sampling process in principle removes no information whatsoever, as long as the sampling frequency is at least twice that of the highest frequency component present in the input signal. This is the well known Nyquist theorem on sampling. The Nyquist theorem can be verified if we consider the frequency spectra of the input and output signals in Figure 3.1 (Baert, Theunissen and Vergult, 1992: 29).

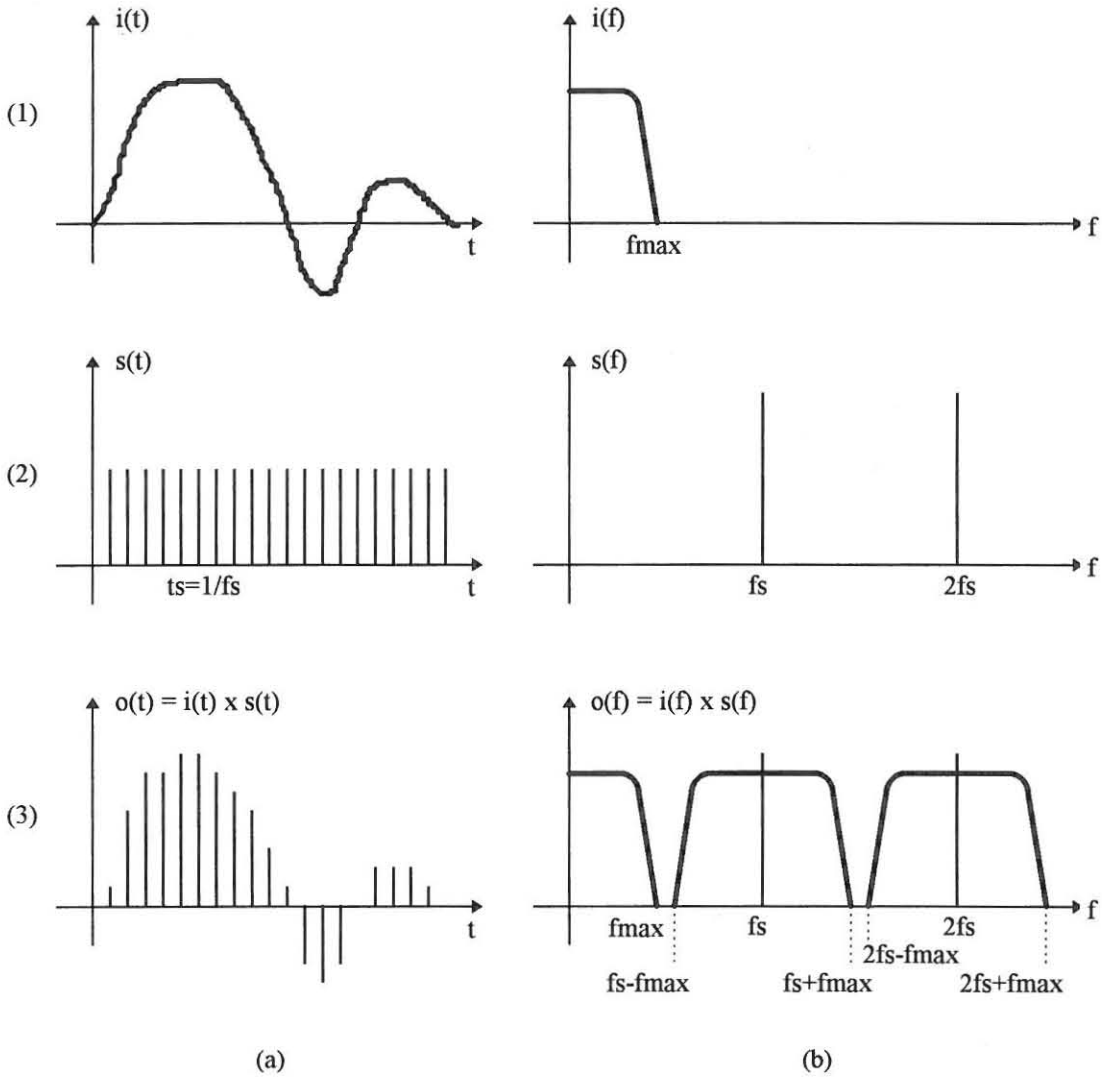


Figure 3.1 Showing the sampling process in (a) time domain and (b) frequency domain.

An analogue signal i(t) (Figure 3.1.1 a) which has a maximum frequency fmax, will have a spectrum having any form between 0Hz and fmax (Figure 3.1.1b). The sampling signal s(t) (Figure 3.1.3 a), having a fixed frequency fs, can be represented by one spectral line at fs (Figure 3.1.1b). The

sampling process is equivalent to a multiplication of i(t) and s(t) and the spectrum of the resultant signal (Figure 3.1.3a) can be seen to contain the same spectrum as the analogue signal, together with repetitions of the spectrum modulated around multiples of the sampling frequency (Figure 3.1.3 b). As a consequence, low-pass filtering can completely isolate and thus completely recover the signal (Ramsey and Watkinson, 1993: 32).

If allowed to remain, the transposed replicas of the signal spectrum will sound like harmonic distortion. In practice, reconstruction is imperfect. Imperfect filters are used, which permit small amounts of unwanted high frequencies to remain (Morgan 1984: 18). Figure 3.1.3 also shows that fs must be greater than 2fmax otherwise the original spectrum would overlap with the modulated part of the spectrum, and consequently be inseparable from it (Figure 3.2) (Petersen, 1992: 152).

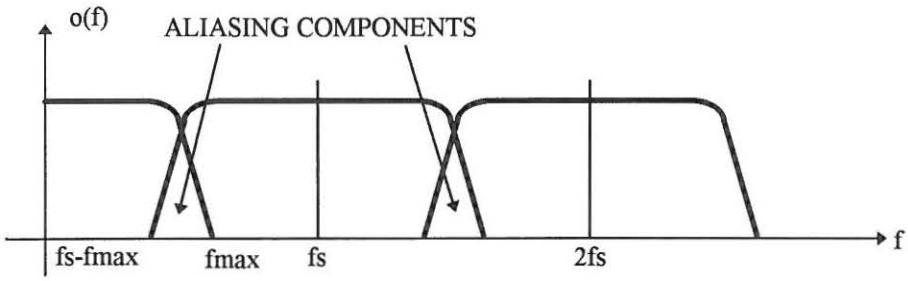


Figure 3.2 If sampling frequency is too low, aliasing occurs.

This phenomenon is known as aliasing. Figure 3.3 shows how this occur, where the dotted line signal is produced instead of the actual signal, shown as a solid line. (Baert, Theunissen and Vergult, 1992: 33).

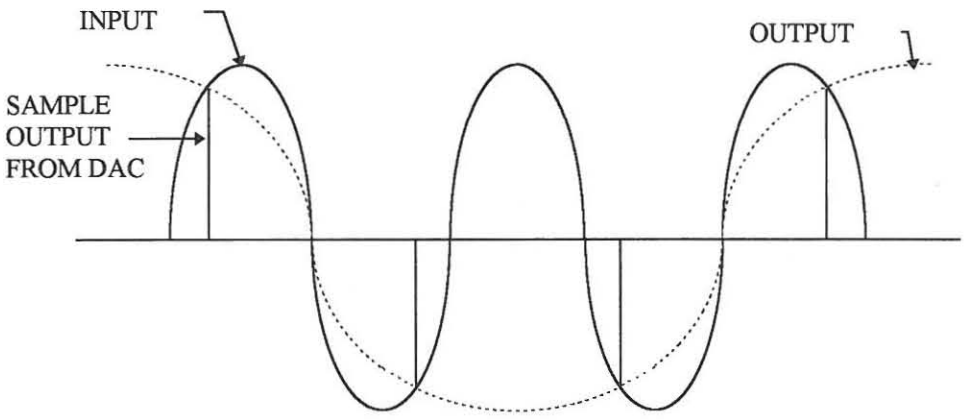


Figure 3.3 Alias distortion.

3.2.2 Quantization error

By definition, because all voltages in a certain quantization interval are represented by the voltage at the center of this interval, the process of quantization is a non-linear process and creates an error, called quantization error. The maximum quantization error is obviously equal to half the quantization interval Q, exccpt in a case where the input voltage widely exceeds the maximum quantization levels (+ or - Vmax), when the signal will be rounded to these values (Figure 3.4) (Furui, 1989: 102 and Haykin, 1988: 190).

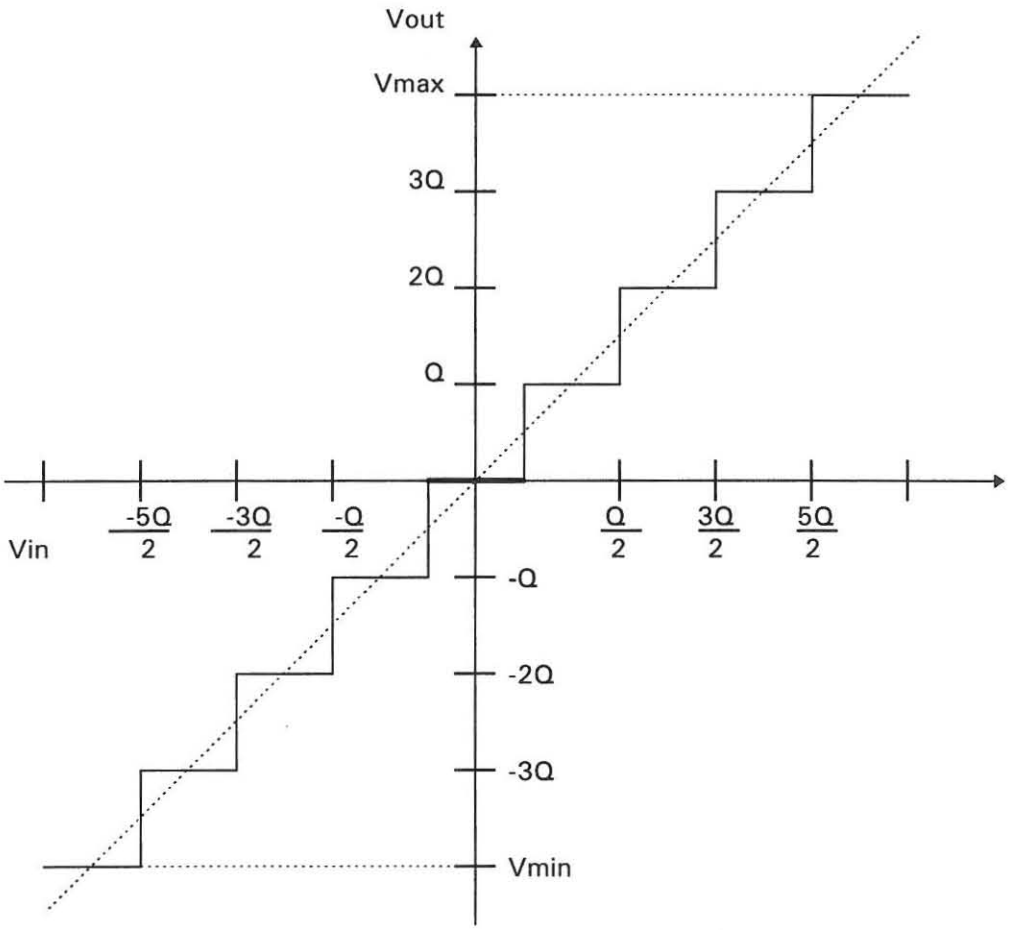


Figure 3.4 Quantization stage characteristic.

Generally, however, such overflows or underflows are avoided by careful scaling of the input signal. So, in the general case, it can be said that

$$-Q/2 < e_{(n)} < Q/2 \qquad (3.1)$$

where e(n) is the quantization error for a given sample n. It can be shown that, with most types of input signals, the quantization errors for several samples will be randomly distributed between these two limits, or in other words, its probability density function is flat (Figure 3.5) (Ramsey and Watkinson, 1993: 46).

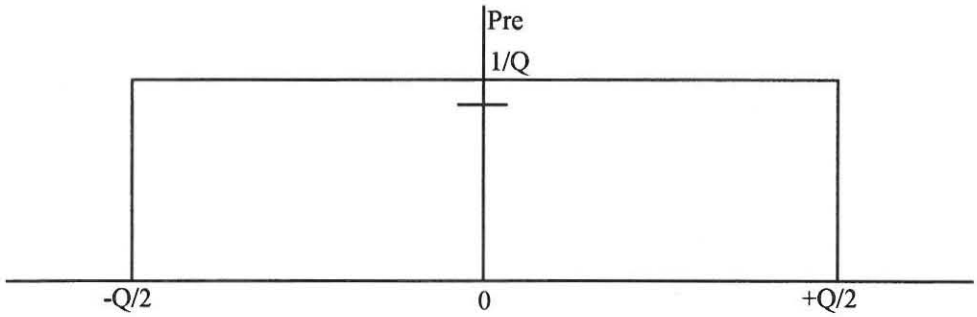


Figure 3.5 Probability density function of quantization error.

There is a good analogy between quantization error in digital systems and noise in analogue systems: one can indeed consider the quantized signal as a perfect signal plus quantization error (just like an analogue signal can be considered to be the sum of the signal without noise plus a noise signal). (Rumsey, 1990: 22).

If the number of bits per sample is fairly large the quantization noise has properties not obviously related to the structure of the signal, and its effect is then subjectively equivalent to adding a small amount of flat-spectrum random noise to the signal. If the number of digits in the binary code is small or if the input signal exceeds the permitted coder range, the quantization noise will have different properties, and will be highly correlated with the signal. In this case the fidelity of reproduction of the signal waveform will obviously be much worse, but the degradation will no longer sound like the addition of random noise. It will be more similar subjectively to the result of non-linear distortion of the analogue signal. Such distortion produces many intermodulation products from the main spectral components of the signal, but even when extremely distorted the signal usually contains sufficient of the spectral features of the original signal for much of the intelligibility to be retained (Holmes, 1988: 56).

A theoretical maximum of 48 dE [illegible] for an 8-bit system is only possible if the signal is large enough to make use of all 256 available quantization levels. Unfortunately, the signal-to-noise ratio degrades very quickly as the amplitude of the input signal drops. For an input 20dB below the overload point (the point at which the A-D converter runs out of codes) the S/N ratio is a mere 28dB. The reason for this degradation is fairly obvious; in the extreme case the input may be so small that it only alters the least significant bit in the binary code. The resulting variation between two possible output levels would not track all the nuances of the input with any degree of precision (Unknown author 1, 1987: 21).

3.2.3 Solutions to quantization noise

3.2.3.1 Companders

The simplest solution to the quantization noise problem is to adjust the input so that the ADC is operated as close as possible to its overload level without actually exceeding it. This can be done electronically. Devices which do this are called companders (compressor/expanders). In essence, the compressor section will reduce the amplitude of signals that are above a certain pre-defined level and increase the amplitude of small signals, so that the range of amplitudes is close to a constant level as depicted in Figure 3.6 (Chappell, 1987: 12).

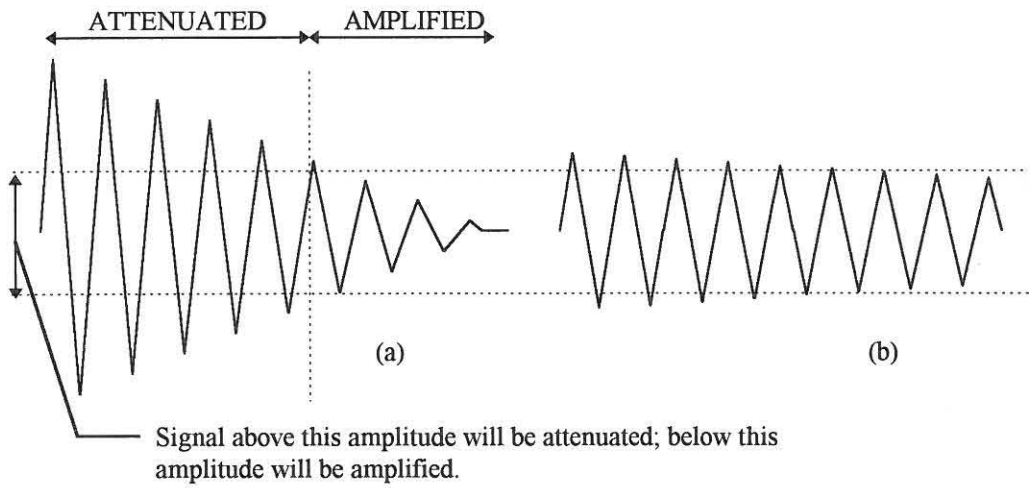


Figure 3.6 (a) Input to compressor, (b) Output from compressor.

Thus, the output will have a sub... compressed, dynamic range and the signal can be kept close to the overload point of the ADC at all times. The expander section returns the dynamic range to normal after the signal has been translated back into the analogue form. Companders work by taking an average of the signal level and using this average to control a voltage-controlled-amplifier (VCA) which modifies the output level (Chappell, 1987: 11).

The averaging process involves rectifying the input and integrating the result as shown in figure 3.7 (Chappell, 1987: 11). A short time constant on the integrator will give an accurate average but will result in a good deal of ripple on the VCA control signal as in (b), while a long time constant gets rid of the ripple but will make the circuit slow to respond to amplitude changes as in (c) and (d). (Chappell, 1987: 12).

As rapid changes of gain in the VCA cause considerable distortion, compander circuits tend to 'err' on the side of a slightly longer time constant. The resulting amplitude changes cause an effect often described as 'breathing' (Chappell, 1987: 12).

3.2.3.2 Companding ADC/DAC

The obvious way to increase the dynamic range of the sampling process is to use a high resolution ADC. A 12-bit ADC will give 4096 quantization levels and a S/N of 72dB at best; a 16-bit ADC have approximately 65 000 levels and a maximum S/N ratio of 96dB. However, these improvements come at high cost regarding expensive converters (A/D and D/A) and huge amounts of memory needed to store samples. An alternative would be non-linear ADC and DACs.

For example: Around the 0V level the first eight quantization levels are spaced 0.025% of full scale. The next eight are spaced 0.05% of full scale, the eight after that 0.1% and the eight after that 0.2% of full scale. The resolution around 0V is now excellent (compared to a linear 8-bit system). The penalty is that in the final section, which takes care of the wave peaks, the resolution is only equivalent to a 5-bit ADC.

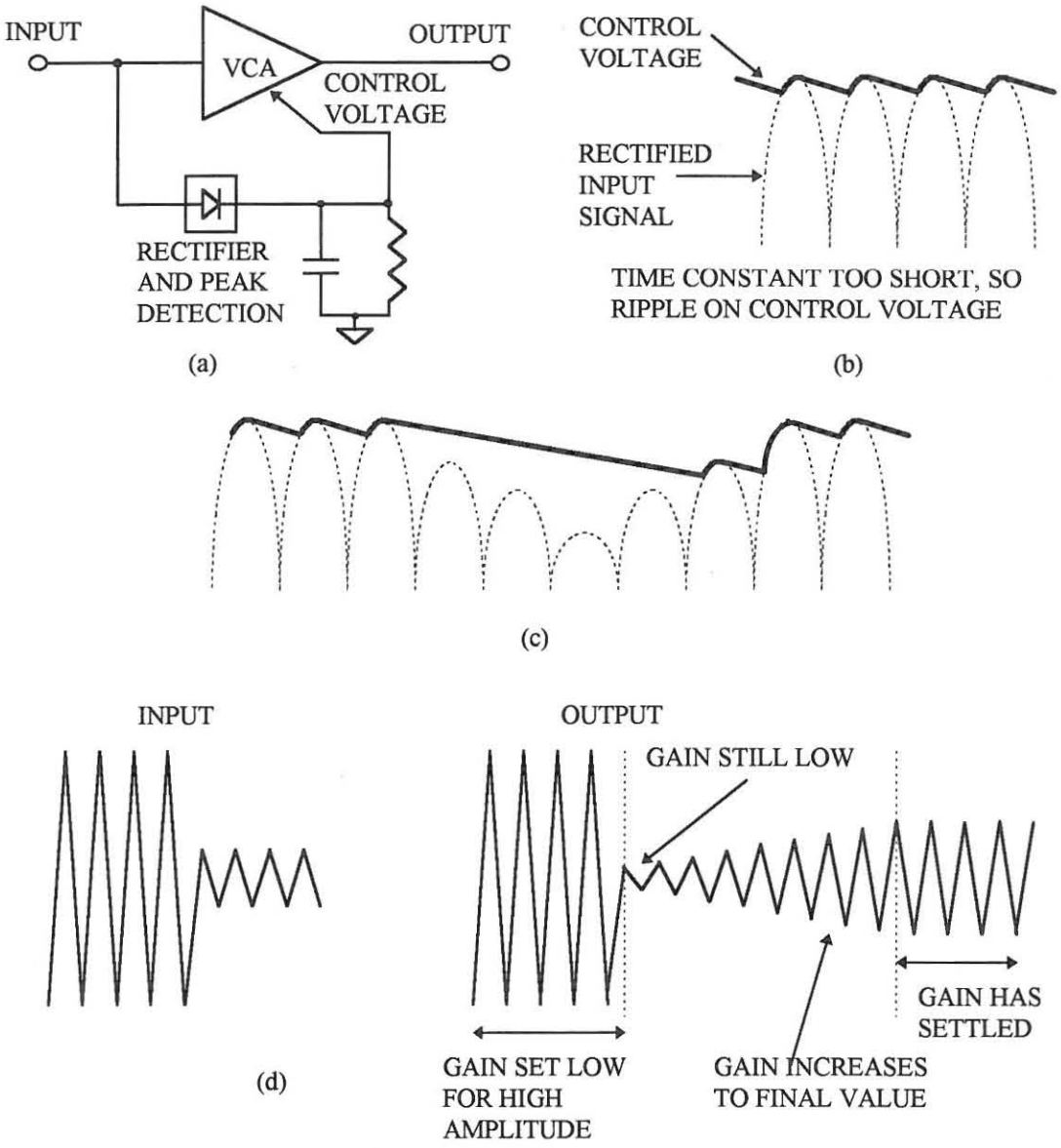


Fig 3.7 Compressor time constants.

There are two strong points in favor to this as opposed to linear conversion. First of all, audio signals generally have an amplitude spectrum concentrated in small signals rather than large ones and will often have crest factors (the ratio of peak to RMS amplitude) of five or more. Secondly, the human ear is much more sensitive to any inaccuracies of the wave around the zero-crossing point than it is to slight variations in the peaks (Noll, 1988: 250).

In IC form, there are two versio... DAC. The first follows a law known as μ255, developed by Bell Laboratories.

The idealized law is:

$$Y = 0.18\ \ln(\ 1 + \mu X\ ) \tag{3.2}$$

where :

n : number of bits in converter

μ : 255 for an 8-bit converter ($\mu = 2^{n}-1$)

X : analogue signal level as a fraction of full scale

This law is approximated by seven 'chords' which double in step size as described above. The rival system, the A-law, approximates the function:

$$Y = 0.18(\ 1 + \ln AX\ ) \text{ for } 1/A \leq X \leq 1 \tag{3.3}$$

$$Y = 0.18AX \text{ for } 0 \leq X \leq 1/A \tag{3.4}$$

where:

n : number of bits in converter

A : 87.6 for an 8-bit converter

X : analogue signal level as a fraction of full scale

The expression gives the magnitude of the signal levels. A sign bit in the DAC is used to say whether they are above or below zero (Chappell, 1987: 13).

3.2.3.3 Pre-emphasis

Yet another way to improve sound quality is to apply pre-emphasis to the input signal. The theory is much the same as that employed in Dolby systems and the like. The higher frequencies of the passband are selectively amplified before A-D conversion, then attenuated after D-A conversion, decreasing a portion of the noise-spectrum with the high frequencies (Chappell, 1987: 14).

3.3 Conclusion

Pulse Code Modulation is a widely used technique for coding analogue signals into a digital format for subsequent processing.

The main causes of PCM waveform degradation is alias distortion and quantization noise.

The input signal needs to be sampled at a frequency at least twice that of the highest frequency component present in the input signal to avoid alias distortion. This is known as the Nyquist theorem on sampling. Quantization noise can be reduced by one of the following techniques:

- companders
- companding ADC and DAC
- pre-emphasis

CHAPTER 4

SYSTEM HARDWARE

4.1 Introduction

Considering the speech systems described in chapter 2, it should be clear that manufacturers did their utmost to produce high quality speech systems with the minimum memory requirements. Although they largely succeed in their quest, this unfortunately complicates the task of the user who has to put up with synthesizers that are often difficult to program and control.

The simplest approach to a talking machine would probably be to record the message on a tape-recorder and to replay it at will. While this would certainly work fine in some applications (telephone answering machines for example) it would be totally unsuitable in others. Reasons for this may be:

- Inability to easily access recorded messages in an arbitrary order
- Mechanical wear on the mechanism cause breakdowns
- Tape-recorders are not as robust as electronic circuits
- Tape based speech is more expensive than electronic speech

4.2 System description

The purpose of this study is to produce a system capable of bridging the gap between the ease of use of a tape recorder and some of the benefits of using a speech synthesizer like the Digitalker. The system is intended to be used in any application that requires an easily programmable speech system capable

of clear natural sounding speech ... any language. The simple controls of the speech circuit allows it to be used in applications which does not justify the use of a microprocessor. Possible uses may include:

- Helping the visually handicapped; for example a lift that announces the floors at which it stops.
- Warning indicators that alert an operator without having to be continuously visually monitored.

The intended applications of the project calls for a circuit in which the messages once recorded, would seldom need change. For this reason, a record facility on each speech circuit would be an unnecessary expenditure adding to circuit complexity. A better option would be one common record facility. All messages are recorded on this facility and are then transferred to individual speech circuits. Following this line of thought, the system may be subdivided into two main parts:

- **Message record:** This part of the project consists of a Record/Playback/Programmer module, a personal computer, an interface card to connect the Record/Playback/Programmer module to the computer and software to control the system. The user records messages by means of a microphone and the Record/Playback/Programmer module onto the computer. Playback is possible for monitoring purposes. The software allows a limited amount of editing to the recorded message; represented as a waveform on the computer screen. When satisfied, the user may transfer the message to an EPROM by means of the EPROM-programmer, integral to the Record/Playback/Programmer module. The programmed EPROM need simply be plugged into a speech circuit to be ready for use.
- **Message replay:** The speech circuit is the equivalent of a mini tape recorder capable of replaying a total of 30 seconds of recorded speech (expandable up to 90 seconds by means of a memory expansion card). This may consist of up to 15 different messages.

The system is easy ... requires only a start pulse and a 4-bit address to trigger any specific message. (Similar to the Digitalker but on a smaller scale.)

Recording is accomplished by means of 8-bit pulse code modulation without any data compression. Despite the fact that PCM makes the system very memory-hungry, it might still be considered worthwhile because of the following:

- It is very easy to program
- Resultant speech is clear and natural sounding
- Speech in any language is possible
- Memory prices have dropped considerably over recent years

4.2.1 Choice of sampling frequency

The internationally accepted bandwidth for speech is 300-3400Hz (Kennedy, 1984: 504). Taking the Nyquist criteria stating that the sampling frequency should be at least twice that of the highest frequency component present in the signal being sampled (Takasaki, 1991: 9), into consideration, this would infer a minimum sampling frequency of:

3400Hz * 2 = 6800Hz

The sampling frequency settled for in this system is 6944Hz[4]

[4] see paragraph 4.3.1.5

4.2.2 Choice of word length

The choice of word length in analogue-to-digital and digital-to-analogue converters is mainly limited to 8, 12 and 16 bit types. Although the compact disk quality of 16-bit may be tempting, these converters are expensive and would have twice the memory requirements of a similar 8-bit system.

An 8-bit system would have a signal-to-noise ratio of 48dB for a signal which is large enough to make use of all 256 available quantization levels. Similar results may be expected of a cheap domestic tape recorder (Chappell, 1987: 11), making it perfectly suitable for the intended application[5].

4.2.3 Reduction of quantization noise

To combat the signal-to-noise ratio degradation resulting from a drop in input signal amplitude, some form of noise reduction may be considered a necessity. After careful consideration, compansion was chosen as a means of noise reduction. Reasons for this are:

- effective: noise reduction may be as much as 40dB (King, 1977: 17)
- complete compander systems like the NE571 is generally available
- simple construction: only a few external components need be added to a NE571 integrated circuit to construct a complete compressor and expander
- low cost

Although the use of companding ADC and DAC's would avoid the breathing effects and other problems usually associated with noise reduction systems, such converters might have proven difficult to obtain.

[5] see paragraph 4.2

4.2.4 Distinguishing between different messages

The following system has been adopted to distinguish between messages of different length while still retaining optimal memory use and short access times.

- To distinguish different messages from each other, an unique 'header' character is used, which does not occur inside any message. The character singled out for this purpose is '0'. This imply that all zero's that occur inside messages must be replaced by '1'. Although this has a slight clipping effect on the signal, it is not noticeable. '0' is chosen as header as it lies on one signal extreme (the other being 255) and is thus least likely to cause distortion if removed.

- The header is followed by an identification byte. Each of the 15 possible messages receive an unique address, defined by the user, by which it can be called up. The ID byte may take any value between 1 and 15. Message ID no's need not be allocated in any specific sequence.

- The ID byte is followed by the actual message. This consists of linear PCM with the only provision being that no '0's occur.

- To speed up the search for individual messages, messages are only allowed to start on memory addresses which are multiples of 128. This way, memory searches may be conducted in steps of 128 instead of checking every byte for a start of message header ('0'). The disadvantage of this method is that up to 127 bytes may be wasted with every message to fill the gap from the end of a message to the next multiple of 128 address. This is acceptable considering that in a worst case, when 128 bytes need to be added to each of the 15 possible messages, it will still amount to less than 0.75% wastage of a 256K Byte EPROM's capacity. The gap left between messages is filled with 127, which represents silence (halfway mark between signal extremes 1 and 255).

4.3 Record/Playback/Programmer Module [6]

As can be seen from the block diagram of the Record/Playback/Programmer Module in appendix A, this circuit can be sub-divided into four main blocks.

- Input path
- Output path
- PIA and EPROM programmer
- Voltage regulators

4.3.1 Input path[7]

The input path consists of:

- Input amplifiers
- Amplitude control
- Low-pass filter
- Compressor
- A to D converter

4.3.1.1 Input amplifiers[8]

By means of SW201 it is possible to select between a microphone input, amplified by U201 and associated components, or a direct line input, buffered by U202 and associated components.

---

[6] see circuit diagram 3, appendix A

[7] see circuit diagrams 4 - 8, appendix A

[8] see circuit diagram 4, appendix A

4.3.1.2 Amplitude control[9]

The purpose of the amplitude control circuit is to allow the user to control the amplitude of the input signal with software. The input signal is buffered by U204C and fed to U203 consisting of three CMOS bilateral analogue switches. These switches switch different combinations of R207 to R211 into the circuit to alter the amplitude of the input signal. U203 is controlled by three control lines, allowing 8 different amplitude levels.

4.3.1.3 Low-pass filter[10]

The low pass filter is an anti-aliasing filter used to remove all frequency components above half the sampling frequency from the input signal. Without the anti-aliasing filter, unrecoverable noise and distortion would be added to the signal during the sampling process[11] .

A sixth order filter with an attenuation of 36dB per octave is used. The filter has a Q factor of one, implying a gain of about 1dB at the cutoff frequency. The upper cutoff frequency is chosen around 10% below the required '-3dB point'. This cancels out the gain (Schaerer, 1982: 3.45). The following formulas apply to the filter:

$$\text{cutoff frequency} \quad fp = \frac{1}{2 \cdot \pi \cdot R \cdot C} \tag{4.1}$$

-3dB threshold frequency $fG = 1{,}1 \cdot fp$ (R222 = 6k8)

cutoff frequency gain $\approx$ 1dB (R222 = 6k8)

---

[9] see circuit diagram 5, appendix A

[10] see circuit diagram 6, appendix A

[11] see paragraph 3.2.1

slope in hold range 36dB/ octave
120dB/decade

gain in the forward bias range $A = \left(\frac{R218}{R219} + 1 \right) \bullet \left(\frac{R222}{R223} + 1 \right)$ (4.2)

$A = 3.85\ (11.6dB)$

For an intended bandwidth of 300 - 3400 Hz this would mean:

fp = 10% lower than 3400Hz

3400*90/100 = 3060Hz

If C is taken as 10n

THEN $R = 1 / (fp\ 2\pi\ C)$ (4.3)

$= 1 / (3060 * 6{,}28 * 10E\text{-}9)$

$= 5199{,}05$ OHM

The closest easily obtainable value to 5199 ohm is 5,6 kilo-ohm.

Calculating fp from this value:

$fp = 1/(2\pi.R.C)$

$= 1/(6{,}28.5600.10E\text{-}9)$

$= 1/0{,}0003516$

$= 2844.14$ Hz

Practically testing the circuit with these values yielded quite satisfactory results[12], effectively suppressing alias distortion.

[12] see figure 6.10

4.3.1.4 Compressor[13]

The compressor compresses the dynamic range of the input signal so that the signal may be kept close to the overload point of the DAC at all times. The compressor accomplishes this by reducing the amplitude of signals that are above a certain predefined level and increasing the amplitude of small signals, so that the range of amplitudes is close to a constant level[14]. The purpose of this is to cover as much of the ADC conversion range as possible to minimize quantization noise. The whole compander is based on a ready-made IC and is used as prescribed by the manufacturer. Noise reduction with this sort of system can be as much as 40dB, and because it works over the entire audio frequency range, will provide 'wide-band' noise reduction (King, 1977: 16).

4.3.1.5 Analogue-to-Digital Converter[15]

The ADC convert the analogue signal into a digital format acceptable to the computer. U503D, U207B, X201 and associated components form a clock oscillator for the ADC (U206).

As the ADC needs 72 clock pulses to complete one conversion, and a minimum sampling frequency of at least 5688Hz (2844 x 2) is required[16], the ADC needs to be clocked at:

5688 * 72 = 409 536 Hz.

An easily achievable oscillator frequency is 500kHz giving a sampling frequency of

500 000 / 72 = 6944.4 Hz.

---

[13] see circuit diagram 7, appendix A

[14] see paragraph 3.2.3.1

[15] see circuit diagram 8, appendix A

[16] see paragraph 4.2.1 and 4.3.1.3

The ADC is used in free-running mode, which means that no external signals are needed to initialize each conversion cycle. The end of conversion (EOC) pulse is used to give an interrupt to the host computer, while at the same time is used to start a new conversion cycle. The host must read the current converted value before the end of the next conversion.

U503F and associated components are used to 'kickstart' the first conversion process into operation after powering up. After the first conversion the process is self sustaining. The purpose of RV202 and RV203 is to adjust the amplitude and offset of the input signal to be compatible with the ADC conversion range.

4.3.2 Output path[17]

The output path consists of:

- Digital-to-analogue converter
- Low-pass filter
- Expander
- Amplitude control
- Output amplifiers

4.3.2.1 Digital-to-Analogue Converter[18]

The purpose of the DAC is to convert the digital signal from the computer back into an analogue format. The DAC consists of U301 which is an 8-bit latched input monolithic linear D-to-A converter.

---

[17] see circuit diagrams 9 - 13, appendix A

[18] see circuit diagram 9, appendix A

4.3.2.2 Low-pass filter[19]

This filter removes by-products of the sampling process from the DAC output signal and is similar to the input filter[20].

4.3.2.3 Expander[21]

The expander returns the signal amplitude, previously compressed by the compressor, to its normal dynamic range. The whole compander is based on a ready made IC and is used as prescribed by the manufacturer.

4.3.2.4 Amplitude control[22]

The purpose of the amplitude control circuit is to allow the user to control the amplitude of the output signal by software. The output signal is buffered by U303C and fed to U304 which consists of three CMOS bilateral analogue switches. These switches switch different combinations of R327 to R331 into the circuit to alter the amplitude of the output signal. U304 is controlled by three control lines; allowing 8 different amplitude levels.

---

[19] see circuit diagram 10, appendix A

[20] see paragraph 4.3.1.3

[21] see circuit diagram 11, appendix A

[22] see circuit diagram 12, appendix A

4.3.2.5 Output amplifiers[23]

U306 is a small audio amplifier capable of driving headphones or a speaker. U305 is a unity-gain buffer amplifier to make the output signal available to the user.

4.3.3 EPROM programmer and PIA[24]

The purpose of the EPROM programmer is to program the final message into an EPROM. The programmer is controlled by U401, a 8255 peripheral interface adapter, which is connected to the computer data bus[25].

U501 and U502[26], are two binary counters. By clocking the counters under computer control, consecutive memory locations in the EPROM are addressed. A program pulse to the program pin of the EPROM, through inverter U503B, program data applied to the data lines of the EPROM into the EPROM memory. Every programmed byte gets verified by enabling the EPROM output through inverter U503A and reading the data-lines of the EPROM. To program the EPROM a 12.5V potential is needed on the Vpp pin of the EPROM. Relay K501 replace the normal 5V with 12.5V during programming. The relay is switched on under computer control through the 8255 peripheral interface. To verify a byte written to the EPROM, one of the 8255 ports need to be changed from an output to an input port. This is done by reconfiguring the 8255's ports. However, the effect of this is that the output of all 8255 ports, including the program-voltage signal, default to zero. The same is true when the port is reversed from an input to an output port again. The program-voltage signal thus need to be switched on each time the 8255 is reconfigured; that is twice for every write verify cycle. To overcome the problem of the program-voltage signal continuously switching between '1' and '0', the signal is

---

[23] see circuit diagram 13, appendix A

[24] see circuit diagrams 14 & 15, appendix A

[25] see circuit diagram 14, appendix A

[26] see circuit diagram 15, appendix A

applied to a mono-flop constructed around U503E, U503C and associated components. This ensure that relay K501 is not continuously switched on and off between data-program and data-verify cycles.

4.3.4 Voltage regulators[27]

The voltage applied to the Record/Playback/Programmer module is regulated by 4 voltage regulators. They are:

- U601 : 12.5V for EPROM programming
- U602 : 12V for audio output amplifier and compander
- U603 : +5V positive half of the +/- split rail for op-amps as well as the 5V necessary for the logic circuits.
- U607 : -5V negative half of split-rail for op-amp's

C609 to C616 are de-coupling capacitors scattered throughout the circuit.

4.4 Computer interface card[28]

The purpose of the computer-interface card is to link the Record/Playback/Programmer module to the computer. U103, U104 and U105 form the address decoder. U102 buffers the computer control lines and U101 is a bi-directional data buffer.

---

[27] see circuit diagram 16, appendix A

[28] see circuit diagram 17, appendix A

4.5 Speech circuit

The speech circuit relies on a 256K byte EPROM, to provide a total of 30 seconds of recording time consisting of up to 15 different messages. Memory expansion is possible through the inclusion of a memory expansion module; boosting recording time to 90 seconds.

4.5.1 Operation of the Speech Circuit[29]

The Speech circuit is capable of operating in three different modes. They are:

- ***Standby mode***: Much of the circuit is disabled to lower power consumption.
- ***Search mode***: The circuit is searching for the requested message.
- ***Speech mode***: The circuit is uttering the requested message.

For the sake of this explanation it is assumed that the Speech circuit is in ***Standby mode***, implicating the following

- The *Mode flip flop* (U1B) is reset, which means:
  - The Q output is low:
    - The switch formed by transistor Q1 is on
  - The NOT Q output is high:
    - The NOT enable pin of the DAC (U10) is high (through D6), disabling the DAC
    - The reset pin of *Binary counter 1* (U7) is high, disabling the counter.

[29] see circuit diagrams 1 & 2, appendix A

- The reset pin of *Binary counter 2* (U8) is high, (because of R1 and because the *Search flip flop* (U5B) is reset) disabling the counter.
- The NOT CE and NOT OE pins of the EPROM (U9) is high, disabling the EPROM and its output lines respectively.

- The *Search flip flop* (U5B) is reset, meaning:
  - The Q output is low:
    - The A=B input of the *Magnitude comparator* (U2) is low, making it impossible to match the data bus with the 4-bit word on its input.
  - The NOT Q output is high:
    - The reset pin of *Binary counter 2* is high, (because of R1 and because the *Mode flip flop* (U1B) is reset) disabling the counter.

To initialize any specific message in the Speech circuit, the user need to specify the ID number of that message. This is done by placing a four bit code on the four address lines connected to the *Magnitude comparator* (U2). To start the message-search the Enable line connected to the set pin of the *Search flip flop* (U5B) is taken high. This set the *Search flip flop*, switching the circuit into ***Search mode***, with the following effect:

- The *Search flip flop* (U5B) is set, implicating:
  - The Q output goes high:
    - A=B input of the *Magnitude comparator* goes high, allowing it to start comparing the data on the data bus with the 4-bit address placed on its input lines.
  - The NOT Q output goes low:

- The reset line of *Binary counter 2* (U8) goes low (through D8), allowing it to start counting.
- The NOT CE and NOT OE lines of the EPROM (U9) goes low, selecting the EPROM and enabling the output (data lines) respectively.

Because transistor Q1 is switched on, (by the low Q output of the *Mode flip flop* (U1B)) clock pulses from the oscillator circuitry (U6A, U6B, U6C, U5A) is able to clock binary counter (U8) directly. *Binary counter 1* (U7) is still disabled because its reset pin is held high by the *Mode flip flop* (U1B). This has the effect that the addresses generated by the two binary counters jump a step of 128 followed by a step of 1 (the clock signal tied to the input of *Binary counter 1* (U7) is also tied to address line A0 of the EPROM (U9)) followed by a step of 128, followed by a step of 1 and so on. The reason for the steps of 128 is the system adopted for optimizing memory usage and minimizing access time (see paragraph 4.2.4). The reason for the step of 1 is to compare the ID byte of messages with the requested message address. A less complicated circuit is possible if the ID byte is always checked than it would be if only checked when the preceding byte is a message header.

The requested message is found when a message header is followed by an address byte coinciding with the address placed on the input lines of the *Magnitude comparator* (U2). The *Magnitude comparator* (U2) indicates a match by its A=B output going high. Message Headers (indicated by a '0') are detected by the *Zero detector* (U4). Detection is indicated by the *Zero detector's* (U4) output going high.

Note:

When the address placed on the address lines of the EPROM (U9) changes, there is a corresponding change of data on the data lines. There is, however, a brief period during this changeover of data when the data on the data lines is not valid. Consequently the output of the *Magnitude comparator* (U2) and the *Zero detector* (U4) is not valid for this period either. To bypass this problem, the output of these two circuits are only sampled when the data lines have

settled and are valid again. [illegible] for the *Magnitude comparator* (U2) and U3B for the *Zero detector* (U4), which sample the output of the circuits half a clock pulse after change of data.

To match the output of the *Zero detector* (U4) in time with that of the *Magnitude comparator* (U2) (remember that the ID byte follows the Message header and is thus one clock pulse later in time), the output of the *Zero detector* (U4) is delayed one clock pulse by U3A. If both the output of the delayed *Zero detector* (U4) and the *Magnitude comparator* (U2) goes high simultaneously (as detected by the AND gate formed by D1, D4 and R3), it indicates that the desired message has been found and that the circuit may proceed to ***Speech mode***.

Speech mode is entered when the AND gate formed by D1, D4 and R3 set the *Mode flip flop*. This has the following effect:

- *Mode flip flop* (U1B) is set implicating:
  - The Q output goes high:
    - Transistor Q1 is switched off
    - The *Search flip flop* (U5B) is reset
  - The NOT Q output goes low:
    - The reset pin of *Binary counter 1* (U7) is no longer held high and it can start counting
    - The NOT enable pin of the DAC (U10) is taken low by action of D6 and R4, enabling the DAC (if both the signals tied to D5 and D7 are low).
    - The reset pin of *Binary counter 2* (U8) and the NOT CE and NOT OE inputs of the EPROM (U9) is no longer held low by the *Search flip flop* (U5B) as it has been reset. This task is taken over by the *Mode flip flop* via D2.

- The *Search flip flop* (U5B, is reset implicating:
  - The Q output goes low:
    - The A=B input of the *Magnitude comparator* (U2) goes low, effectively disabling it
  - The NOT Q output goes high:
    - The reset pin of *Binary counter 2* (U8) and the NOT CE and NOT OE inputs of the EPROM (U9) can no longer be held low by the *Search flip flop* (U5B) through D3. This task is taken over by the *Mode flip flop* through D2.

The two binary counters are now configured in such a way that it generates consecutive addresses to the EPROM that increment in steps of 1. The effect of this is that the message is read out, appearing as an analogue signal at the output of the DAC.

This process continues until the *Zero detector* (U4) detects another message header ('0') on the data bus, indicating the start of the next message. This resets the *Mode flip flop* (U1B), switching the circuit back into ***Standby mode***.

Note:

- The purpose of the OR gate formed by D5, D6, D7 and R4 is to:
  - disable the DAC (via D5) when a message header ('0') is present on the data bus as these would cause a loud 'pop' sound
  - disable the DAC during the brief invalid data periods as these cause a background noise on the DAC (via D7)
- U3B is set by the RC network formed by C3 and R11 during power up to ensure that the circuit is in ***Standby mode*** when first switched on to avoid spontaneous message utterances.

Figure 4.1 shows all the relevant pulse trains during the different phases of the circuit.

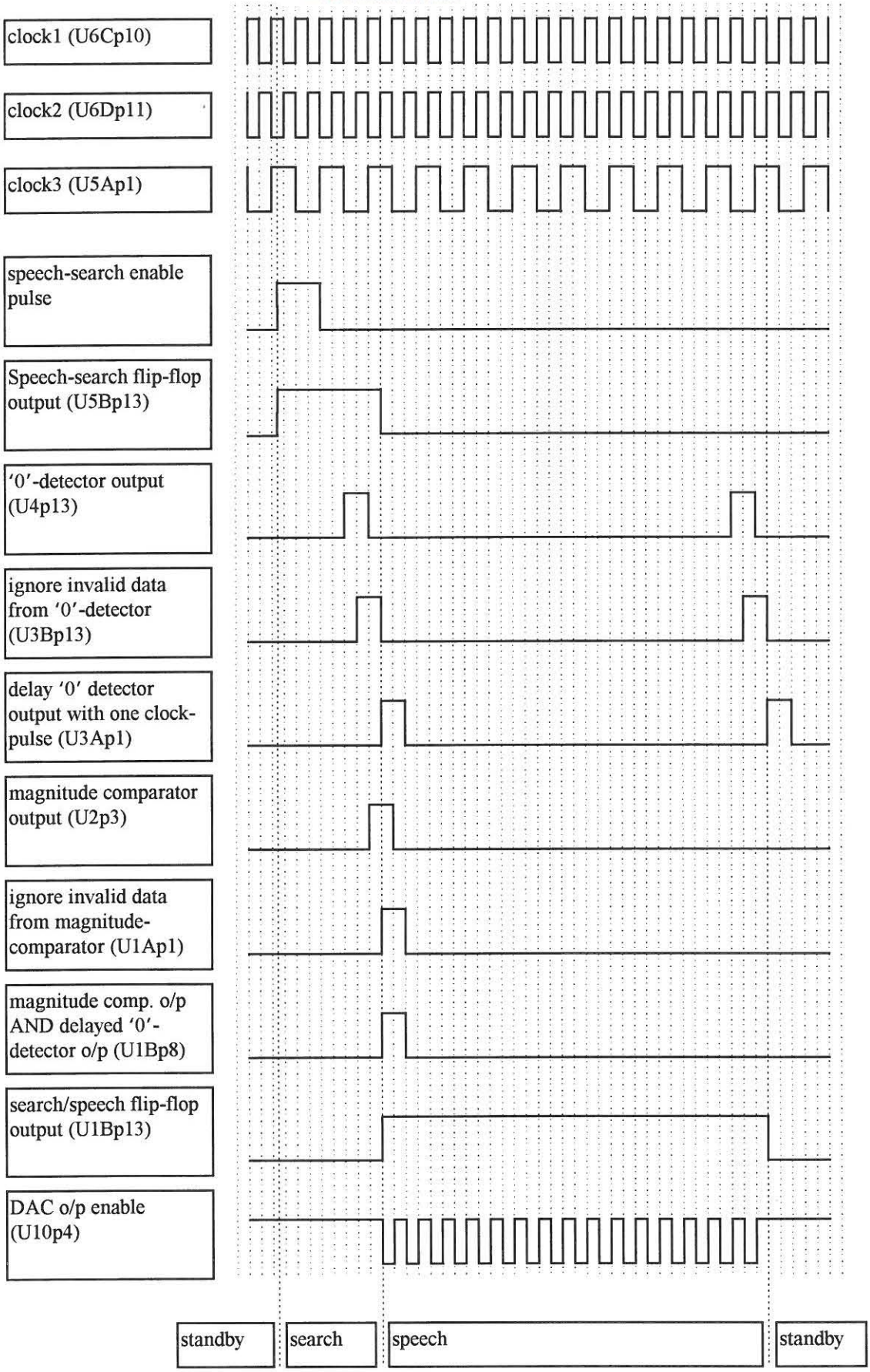


Figure 4.1 Speech-circuit pulse diagram.

4.6 Conclusion

The purpose of this study was to produce a system capable of bridging the gap between the ease of use of a tape recorder and the benefits of using a system like the Digitalker. Eight-bit Pulse Code Modulation without any form of data compression and a sampling rate of 6944Hz[30] is used as recording system. Quantization noise is limited by a compander.

The project comprises playback modules (speech circuits) and a system to program the playback modules.

The following has been devised to distinguish between messages of different length while still retaining optimal memory use and access time.

- Each message starts with a header (indicated by a zero ('0')) to distinguish messages from each other
- The byte following the message header, is an identification byte specific to that message. It may take on any value between 1 and 15.
- The message content follows the message ID. This consists of linear PCM with the only provision being that no zeroes occur.

The programming system may be subdivided into:

- Record/Playback/Programmer module
- Interface card

[30] see paragraph 4.3.1.5

- Software

The Record/Playback/Programmer module may be further subdivided into:

- Input path
- Output path
- 8255 PIA and EPROM programmer
- Voltage regulators

The playback modules consist of a small circuit board which is the equivalent of a mini tape recorder capable of replaying a total of 30 seconds consisting of up to 15 different messages.

CHAPTER 5

SYSTEM SOFTWARE

5.1 Introduction

The ease of use of the system as a whole is largely determined by the level of software user-friendliness. It was thus important to ensure an intuitive user interface that is both efficient and easy to use.

The software serves two main purposes:

- ***Hardware control:*** Control over the Record/Playback/Programmer module including:
  - recording and playback of samples
  - control of record and playback levels
  - writing samples to and reading samples from EPROM

- ***Software control:***
  - Subsequent processing of recorded messages, including: message editing (copy, move, delete)
  - writing messages to and reading messages from disk.

5.2 Software description

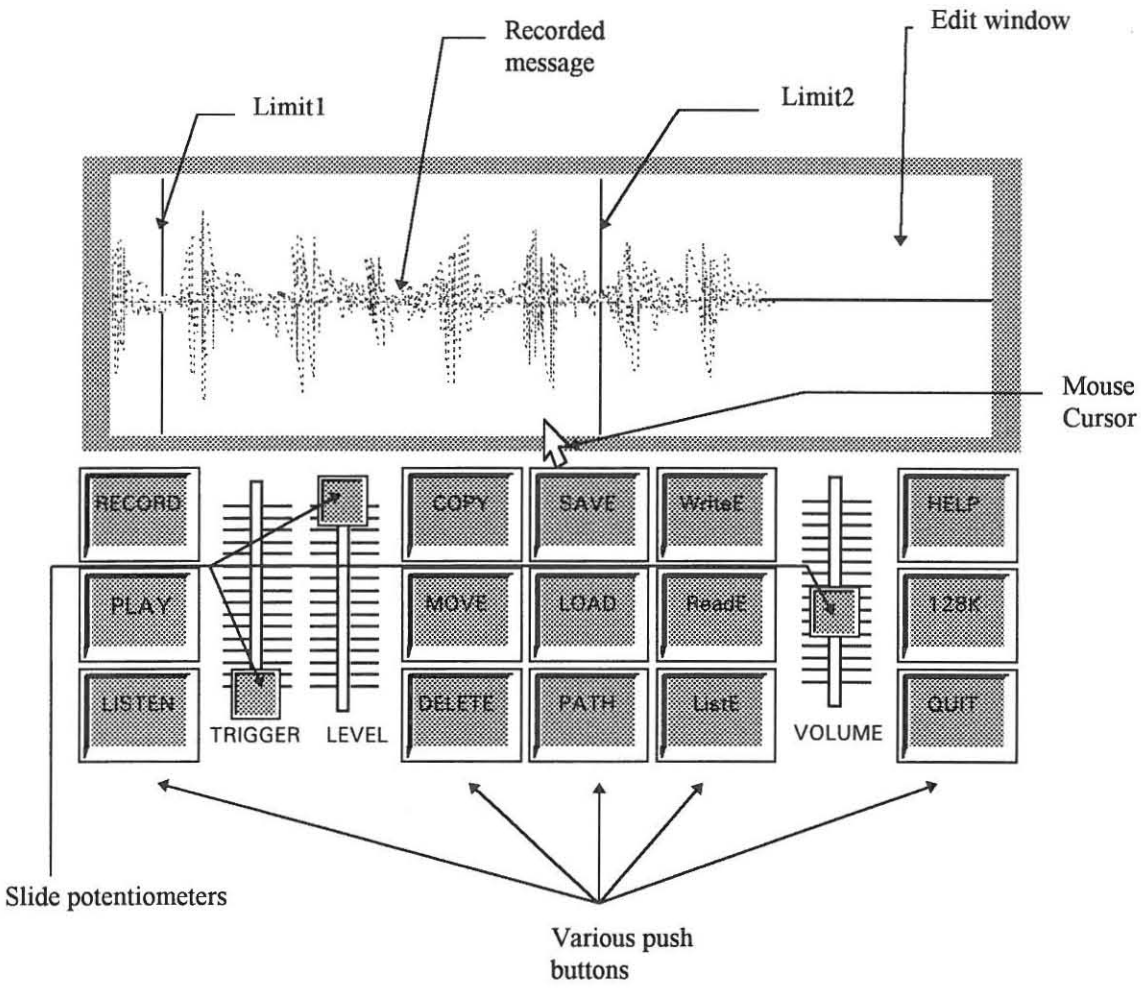


Figure. 5.1 Appearance of the system software.

The programme is designed in such a way that the screen represents a machine with push buttons, slider potentiometers and a display screen. To push a button, the cursor is moved over the desired button and the left mouse key clicked. A recorded message is represented as a waveform on the display screen (Edit window).

In the pages to follow, the various subroutines used in accomplishing the software functions will be discussed, starting with brief explanations of some of the variables used in the subroutines.

5.2.1 Variables and Functions

5.2.1.1 Record Buffer

All samples, whether recorded, loaded from disk or loaded from EPROM, are stored in the Record Buffer. This is a part of the host computer's RAM and consists of about 60k bytes, allowing for approximately 8.6 seconds of sound at a sampling frequency of 6944Hz[31] .

5.2.1.2 Record Buffer Pointer

The Record Buffer Pointer keeps track of where the next byte is to be written into, or read from the Record Buffer.

5.2.1.3 Record Monitor

This defines the status of the monitor function which allows the user to monitor an input signal that has passed through the whole process of analogue-to-digital and digital-to-analogue conversion in real time. Two states are possible: LISTEN and MUTE, which is on and off respectively.

5.2.1.4 Programme Status

The Programme Status variable is the program's way of keeping track of the mode of operation it is in. This would be either Record, Play or Idle.

---

[31] see paragraph 4.3.1.5

5.2.1.5 Reset EPROM address

The address lines of the EPROM in the EPROM programmer socket is tied to the output lines of a binary counter. The Reset EPROM function resets this counter, consequently addressing memory location '0' of the EPROM. (See circuit diagram 15 in Appendix A)

5.2.1.6 Clock EPROM address

The Clock EPROM function increments the binary counter addressing the EPROM, with a subsequent increment in the addressed memory location of the EPROM. (See circuit diagram 15 in Appendix A)

5.2.1.7 Write EPROM / Read EPROM

This function Write and Read bytes to and from an EPROM respectively.

5.2.1.8 EPROM Size

Variable that defines the inserted EPROM's size. It must be set by the user. Provision has been made for two possible sizes: 128k and 256k bytes.

5.2.1.9 LIMIT1 and LIMIT2

Limit1 and Limit2 are pointers used in the EDIT window to mark specific parts of the recorded signal for the purpose of playback, copy, move, delete and saving to EPROM or disk. Limit1 is moved by positioning the mouse cursor over the desired part of the waveform in the EDIT window and clicking the left mouse button. Limit2 is moved in a similar way by clicking the right mouse button. Limit1 indicates the start of the marked signal block and Limit2 the end. Limit1 can never move to the right

of Limit2 as this would indicate a signal block chaining before it has started, or a reversal of the recorded signal.

5.2.2 Subroutines

5.2.2.1 Interrupt Service Routine

The purpose of the Interrupt Service Routine is to establish communication between the Record/Playback/Programmer module and the software. The Record/Playback/Programmer module continuously generates interrupt signals at a rate of 6944 per second, which is the sampling frequency[32]. On receipt of an interrupt, programme execution is temporarily halted and control transferred to the Interrupt Service Routine. The Interrupt Service Routine would react in one of three ways, depending on the status of the Programme Status variable. The three possible states are:

- RECORD : Record a byte into the Record Buffer
- PLAY : Replay a byte from the Record Buffer
- IDLE : Do nothing

Figure 5.2 represents a flowchart of the Interrupt Service Routine.

The main reason for using an interrupt handler routine instead of polling, is that it allows RECORD and PLAY functions to be performed in the background while a routine may handle other tasks like updating the display, without the risk of missing samples. It also has the effect of greatly simplifying the RECORD and PLAY procedures.

[32] see paragraph 4.3.1.5

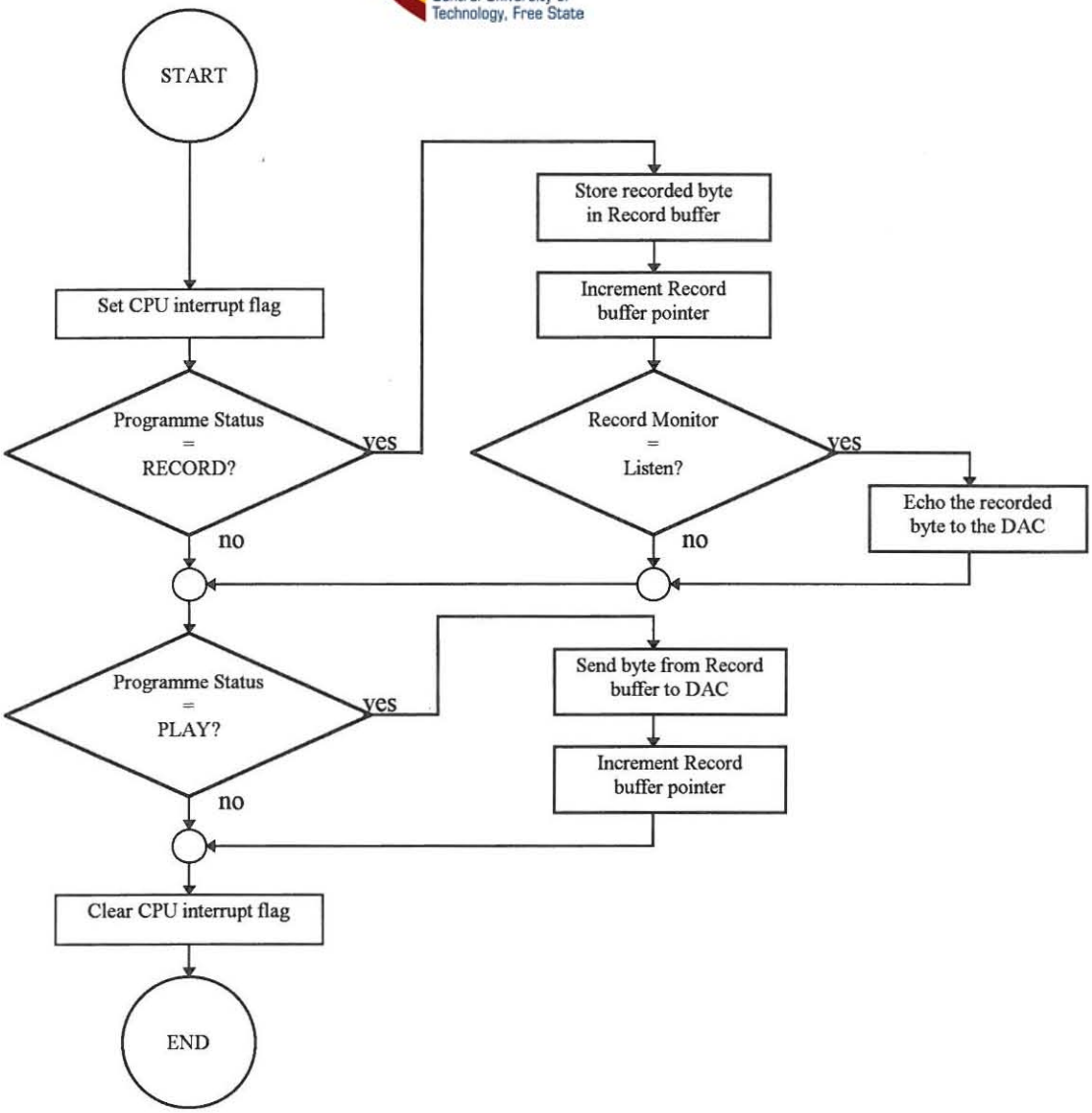


Figure 5.2 Interrupt Service Routine flowchart.

5.2.2.2 RECORD

Sounds may be recorded into the system by either using a microphone or directly by means of the auxiliary input. The recording process is initiated by 'pressing' the RECORD button. The record level can be controlled by dragging the LEVEL control potentiometer up or down with the mouse. The recording process is delayed until the input signal exceeds a value defined by the TRIGGER potentiometer. This function may be overridden by setting the TRIGGER potentiometer to its minimum level. The recorded signal is displayed as a graphical waveform in the Edit window.

Note: It is not possible to alter t... ...neters while the system is recording or playing back a signal.

Figure 5.3 is a simplified flowchart of the RECORD process.

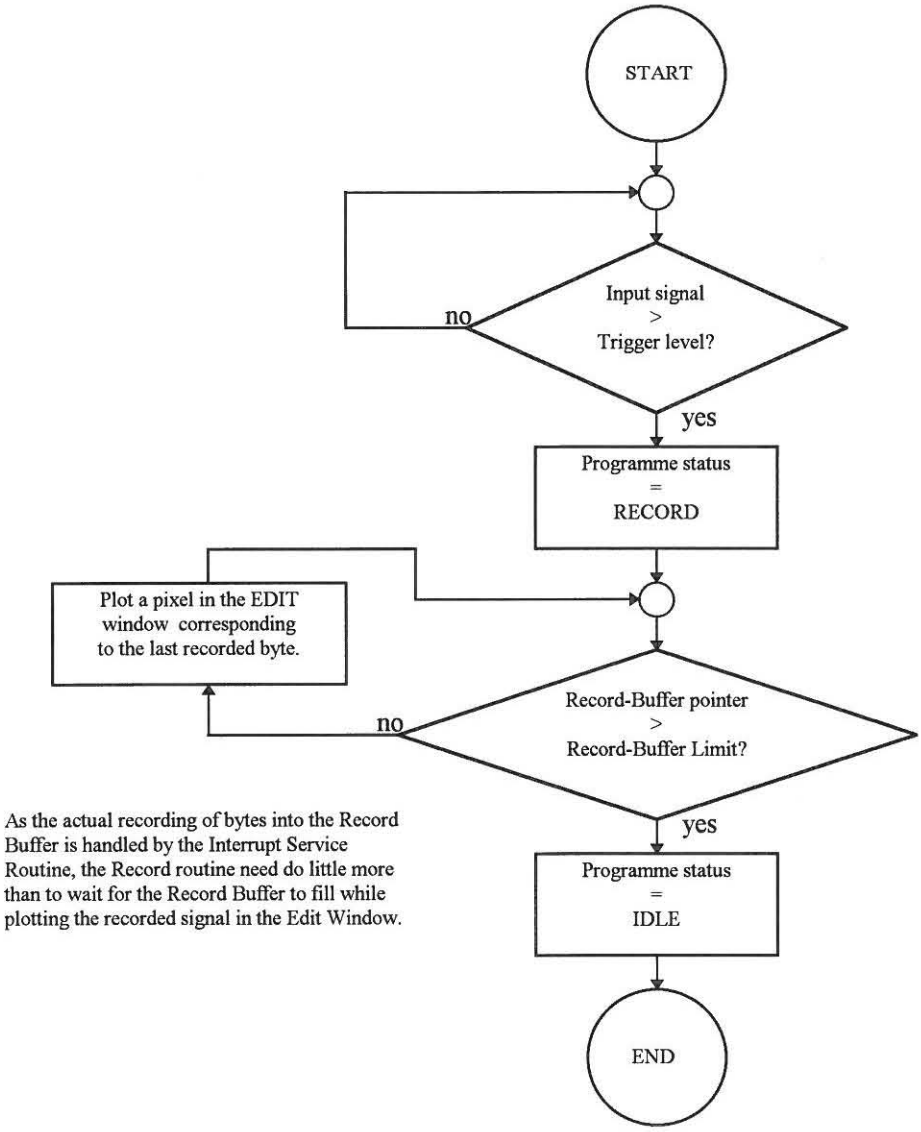


Figure 5.3 RECORD procedure flowchart

The RECORD procedure switch the Programme Status variable from IDLE to RECORD. Each time an interrupt request is received from the Record/Playback/Programmer module, programme flow is redirected to the Interrupt Service Routine, where another byte will be recorded into the Record Buffer. Once the Record Buffer is full, the Programme Status variable is switched back to IDLE, completing the RECORD routine.

5.2.2.3 PLAY

Recorded signals may be played back by 'pressing' the PLAY button. Specific parts of the recorded signal may be played back by marking it with Limit1 and Limit2[33]. It is possible to control the playback volume by altering the setting of the VOLUME potentiometer by means of the mouse.

Figure 5.4 represents the PLAY procedure flowchart.

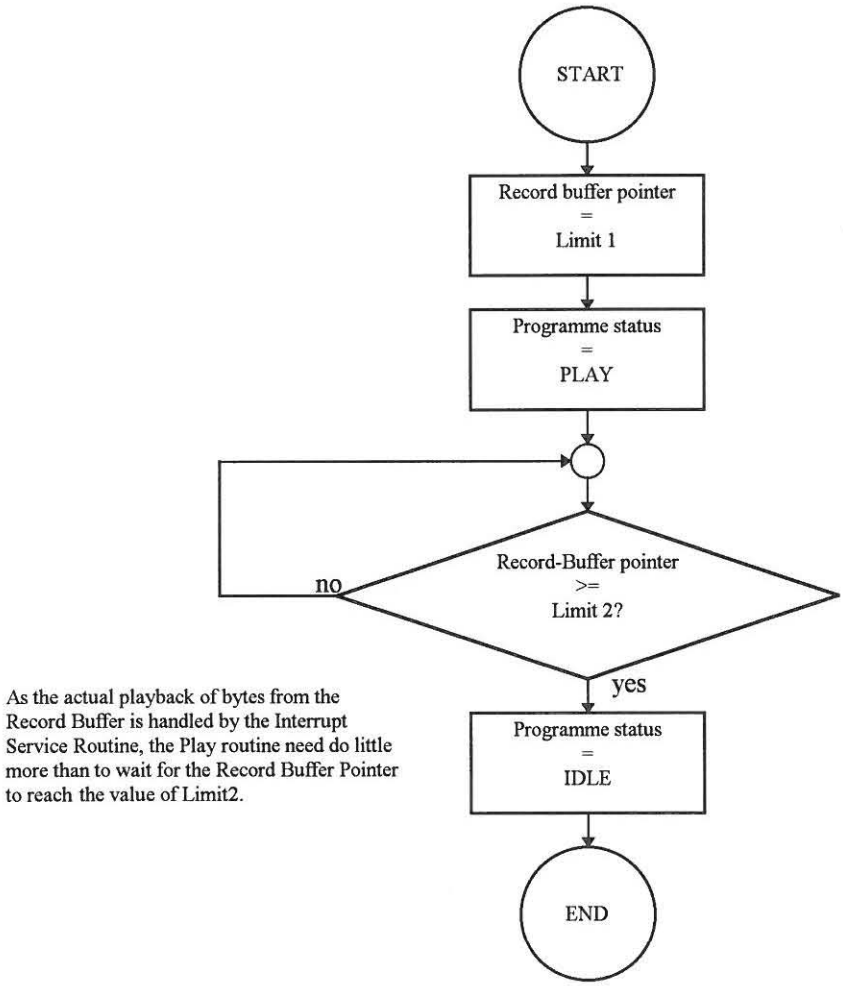


Figure 5.4 PLAY procedure flowchart

[33] see paragraph 5.2.1.9

The PLAY procedure is very similar to the RECORD procedure, except that the Programme Status variable is set to PLAY instead of RECORD. During Playback the Interrupt Service Routine handles all the data manipulation to accomplish the replay process. PLAY is terminated as soon as the value of the Record Buffer Pointer is equal to that of Limit2, at which point the Programme Status variable is switched back to IDLE, completing the play routine.

5.2.2.4 MUTE/LISTEN

When in use, the system continuously convert the input signal from analogue-to-digital and from digital-to-analogue in real-time, allowing the user to monitor the input signal, thus giving an indication of what a recorded signal would sound like. This feature may be disabled by 'pressing' the LISTEN button, turning it into MUTE.

5.2.2.5 COPY

Any part of the recorded signal may be copied from one part of the EDIT window to another. This is accomplished by marking the desired part of the signal in the EDIT window by means of Limit1 and Limit2[34]. 'Pressing' the COPY button highlights the marked part of the EDIT window. It is now possible to drag a copy of the marked signal to any part of the EDIT window by means of the mouse cursor. Figure 5.5 is a flowchart of this process.

5.2.2.6 MOVE

MOVE is similar to COPY, except that in this case it is not a copy of the marked signal that is moved, but the signal part itself. The space originally occupied by the signal part is filled with silence, represented by '127'.

---

[34] see paragraph 5.2.1.9

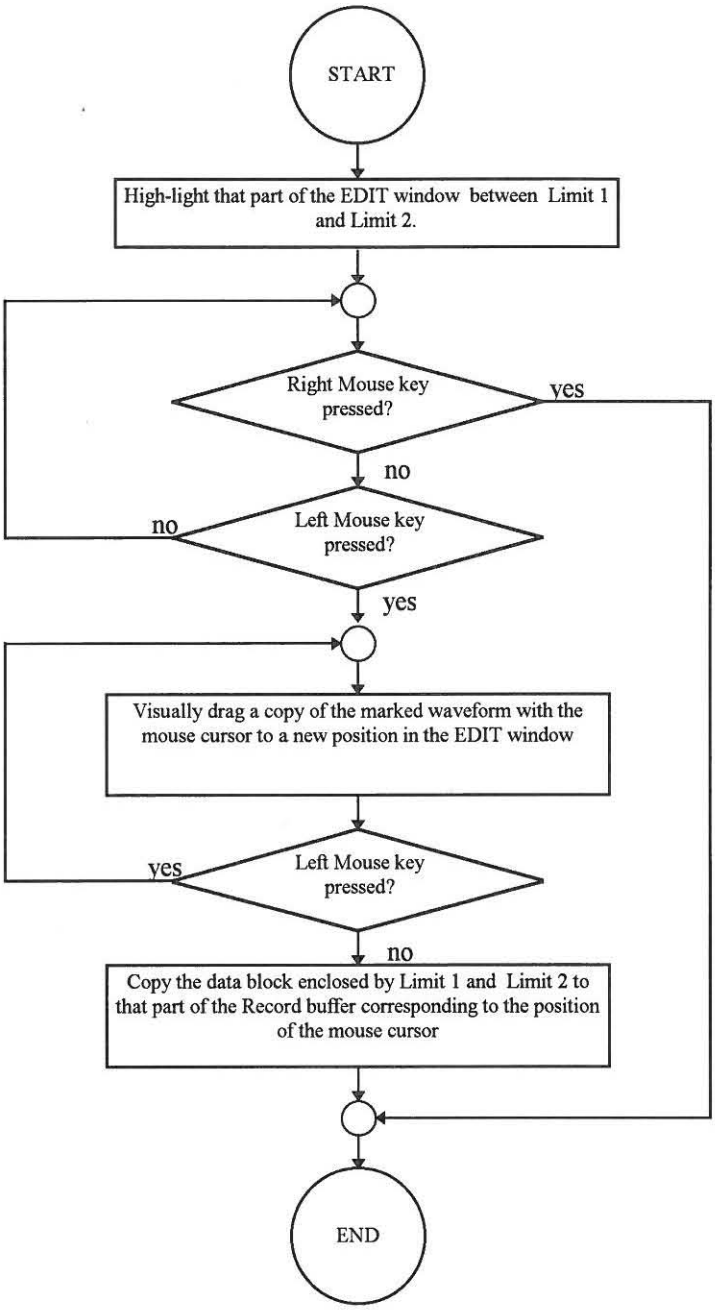


Figure 5.5 COPY procedure flowchart

5.2.2.7 DELETE

Similar to COPY and MOVE, a part of the signal is marked by means of Limit1 and Limit2 in the EDIT window. The marked part is deleted by pressing the DELETE button and filling the corresponding Record Buffer positions with silence, represented by '127'.

5.2.2.8 SAVE

'Pressing' the SAVE button save that part of the recorded signal between Limit1 and Limit2, in the EDIT window, to disk. The user is prompted for a file name. The flowchart of this procedure is shown in Figure 5.6

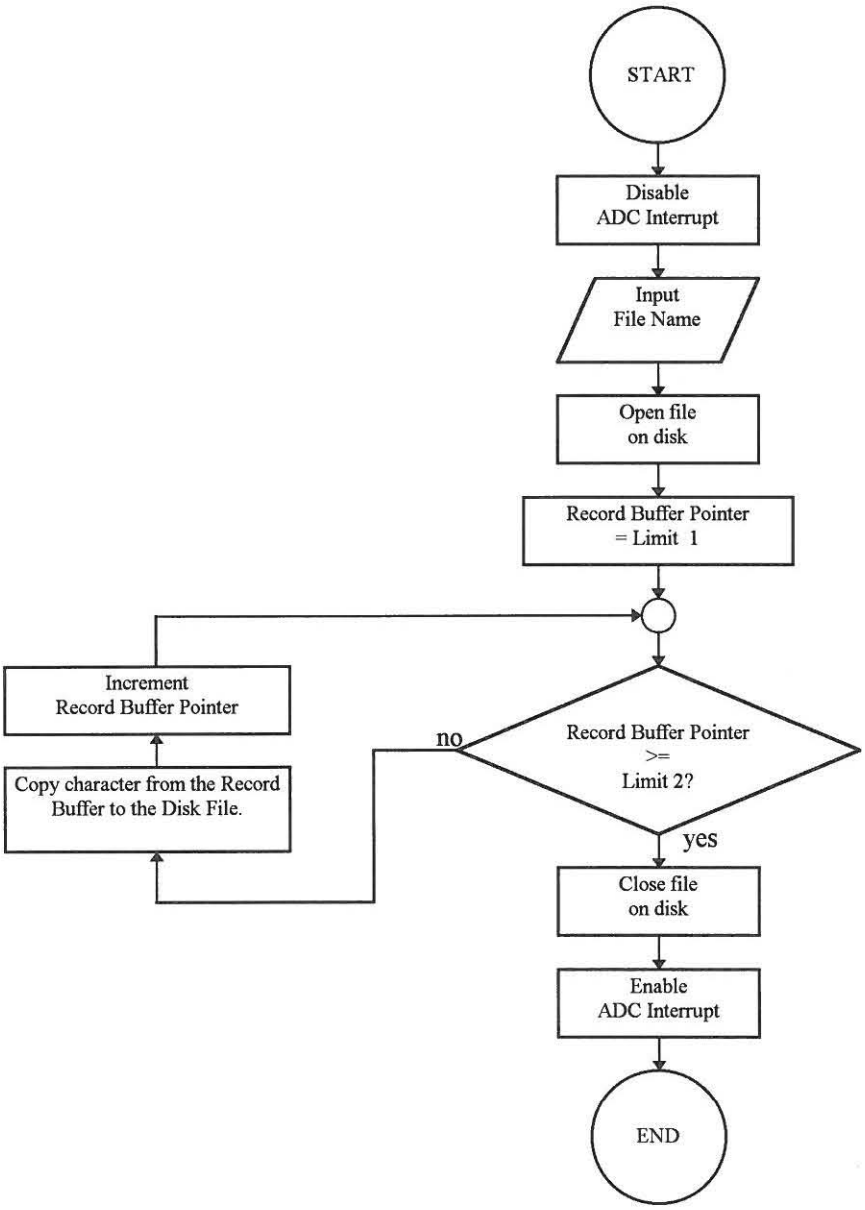


Figure 5.6 SAVE procedure flowchart.

5.2.2.9 LOAD

A file, previously saved to disk, is retrieved by 'pressing' the LOAD button. A list of all the files in the current directory with a .TLK extension is displayed. A file is selected by clicking its name with the mouse cursor. See Figure 5.7 for a flowchart of the LOAD procedure.

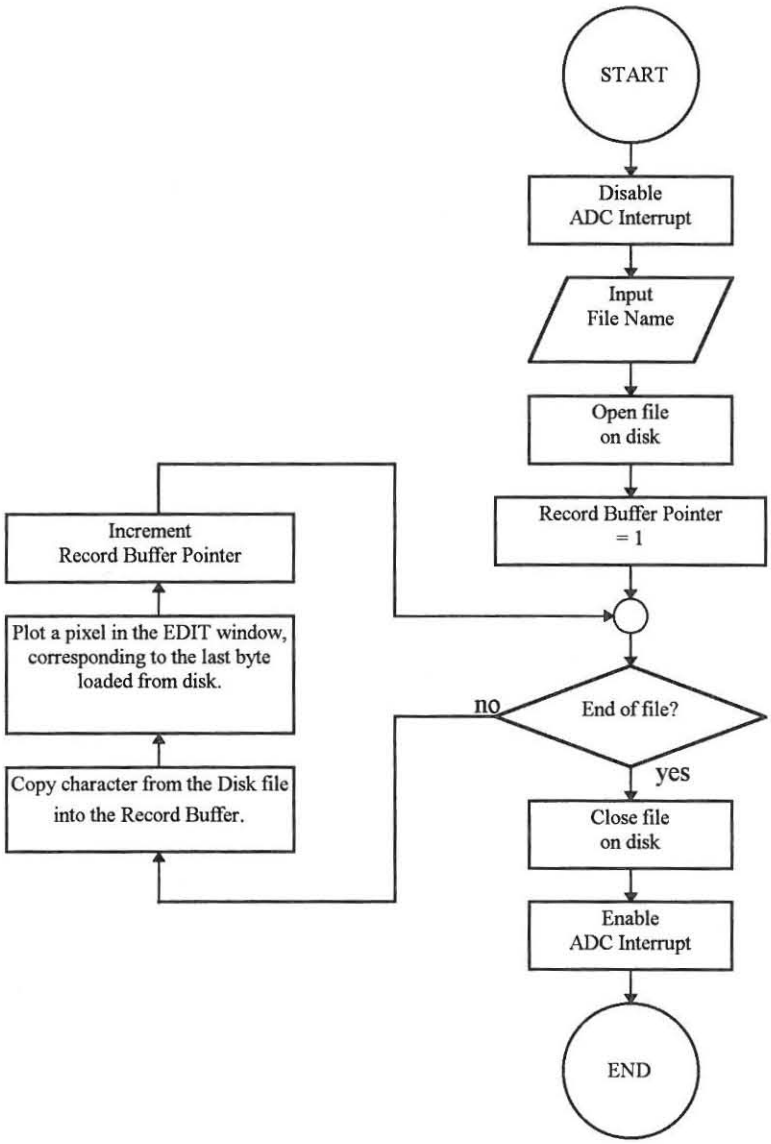


Figure 5.7 LOAD procedure flowchart.

5.2.2.10 PATH

The path button prompts the user to enter a path to where files are to be saved to or loaded from.

5.2.2.11 WriteE

When the WriteE button is 'pressed', the signal part marked by Limit1 and Limit2 in the EDIT window, is written to the EPROM in the EPROM programmer socket. This may either be an empty EPROM, or an EPROM to which a message is to be added. Figure 5.8 represents a flowchart of this procedure.

The first thing the procedure does is to verify the presence of an EPROM in the programmer socket. It does this by reading the byte at address '0'. If this byte is a '0' it indicates the presence of a previously used EPROM, and the procedure searches through the EPROM for enough empty space to hold the proposed message (Empty space is indicated by hex FF). If enough empty space is found, the routine commences with the writing of the message into the EPROM. Otherwise a warning message is flagged.

If the byte at address '0' is not a '0', it either indicates the absence of an EPROM or the presence of an empty EPROM in the programmer socket. Which of the two is determined by attempting to write a '0' into memory location '0'. If this is possible, it indicates the presence of an empty EPROM and writing of the message into the EPROM may commence.

If it is not possible to write a '0' into the EPROM, there is no EPROM present in the programmer socket, or the EPROM is faulty. A warning message is flagged to the user.

Once enough space has been located in the EPROM, the user is prompted for an ID number for the message to be programmed (Any value between 1 and 15). The procedure then copies that part of the Record Buffer indicated by Limit1 and Limit2 into the EPROM. The space in the EPROM between the address of the last byte programmed and the address of the first subsequent multiple of 128 address, is filled with value 127, which represents silence[35].

[35] see paragraph 4.2.4

All that remains is to notify the user that the message has been successfully programmed into the EPROM.

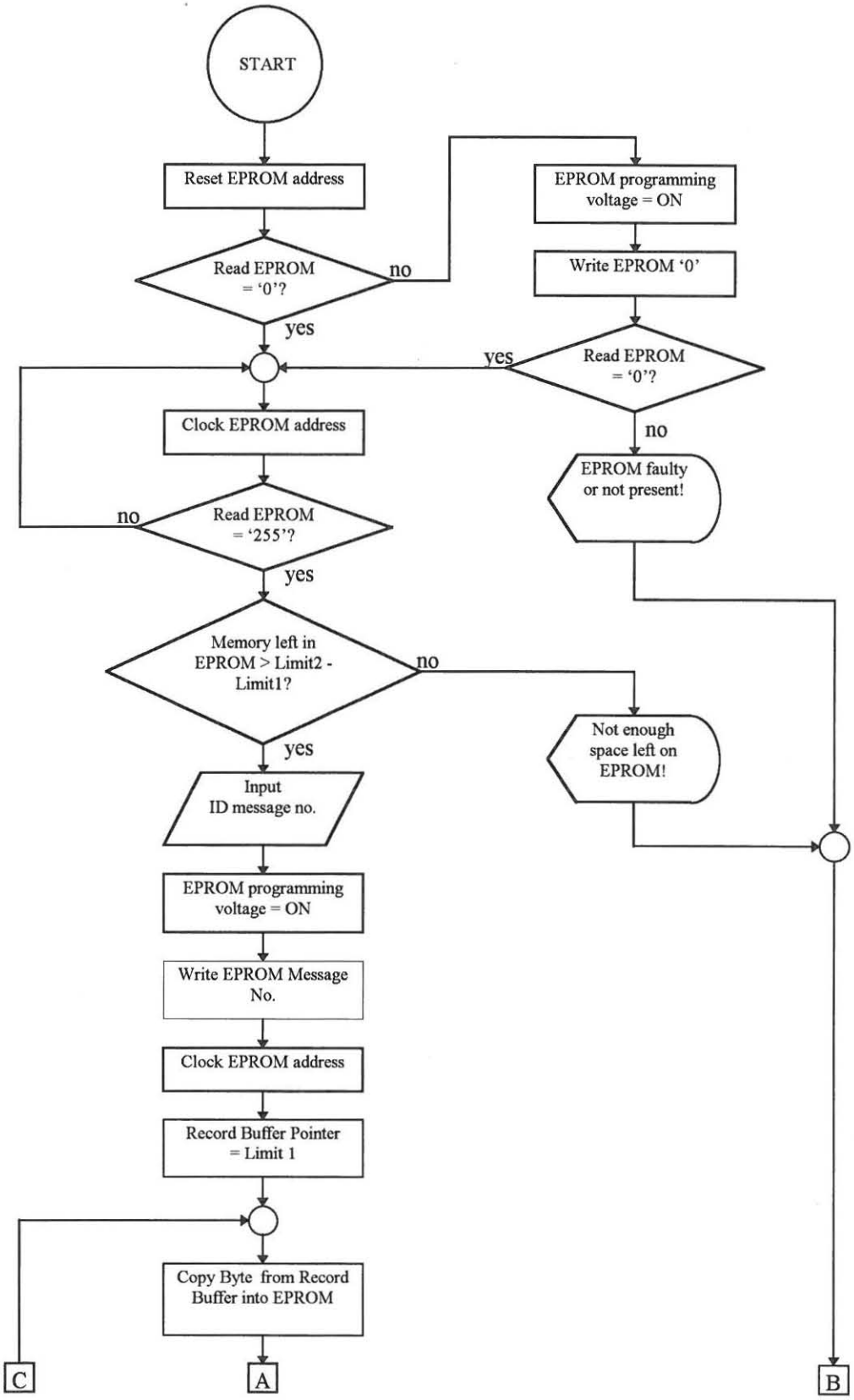


Figure 5.8 WriteE procedure (continued on next page)

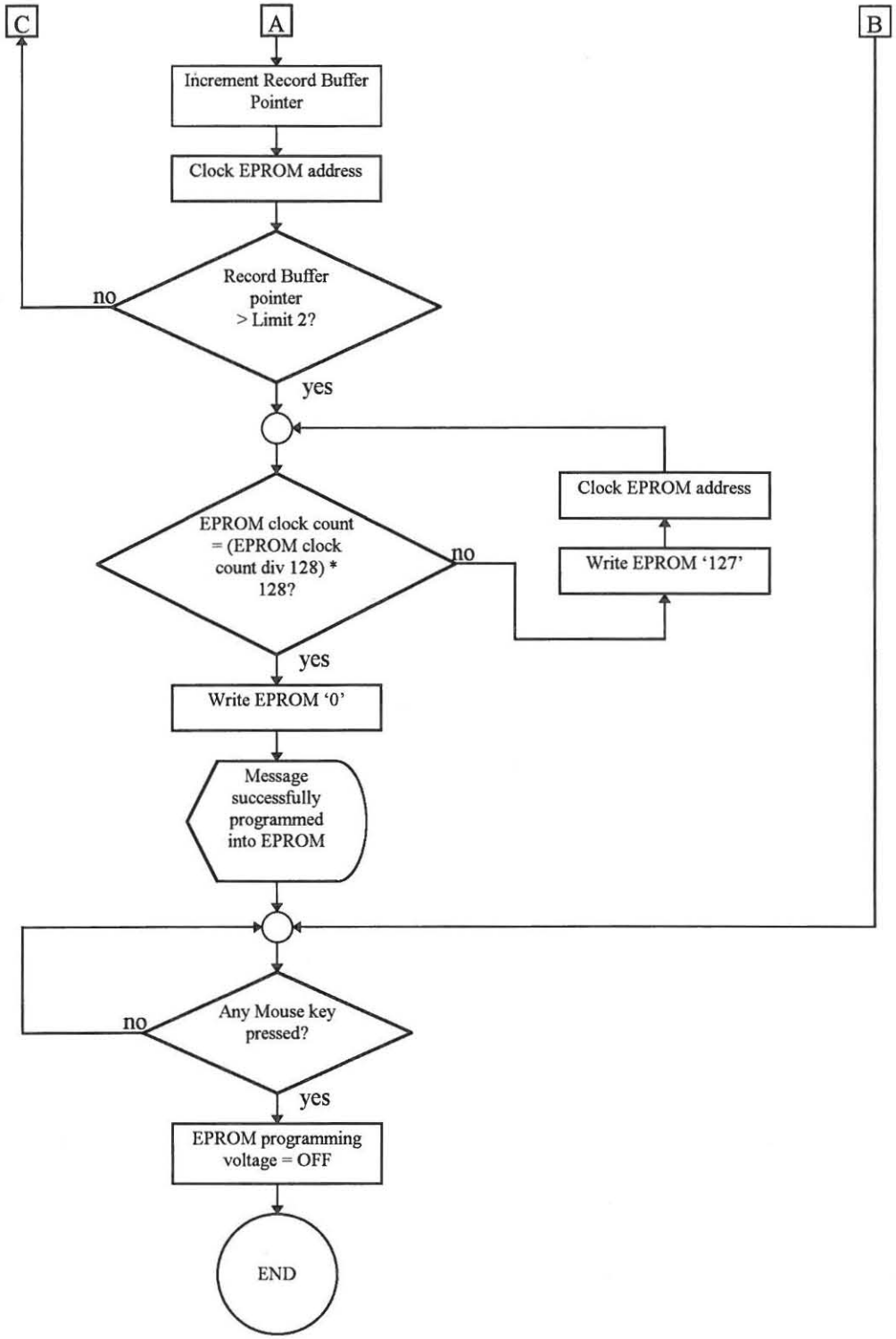


Figure 5.8 WriteE procedure flowchart. (continued)

5.2.2.12 *ReadE*

This function is used to read programmed messages back from an EPROM inserted in the EPROM-programmer socket. The flowchart of the routine is represented by Figure 5.9. Similar to the WriteE procedure, this procedure also verifies the presence of an EPROM in the programmer socket. If an EPROM is found to be present, the user is prompted for the required message ID number. The system then searches through the EPROM for the corresponding message. As soon as it is found, it is loaded into the Record Buffer and displayed as a waveform in the EDIT window.

5.2.2.13 *ListE*

This function graphically gives an indication of the contents of an EPROM inserted into the EPROM programmer socket and display the ID numbers of all the messages present in an EPROM.

5.2.2.14 *HELP*

'Pressing' this button displays topic related help files on the use of the system.

5.2.2.15 *128K/256K*

This allows the user to toggle between two possible EPROM sizes, namely 128k bytes and 256k Bytes.

5.2.2.16 *QUIT*

'Pressing' this button aborts the programme.

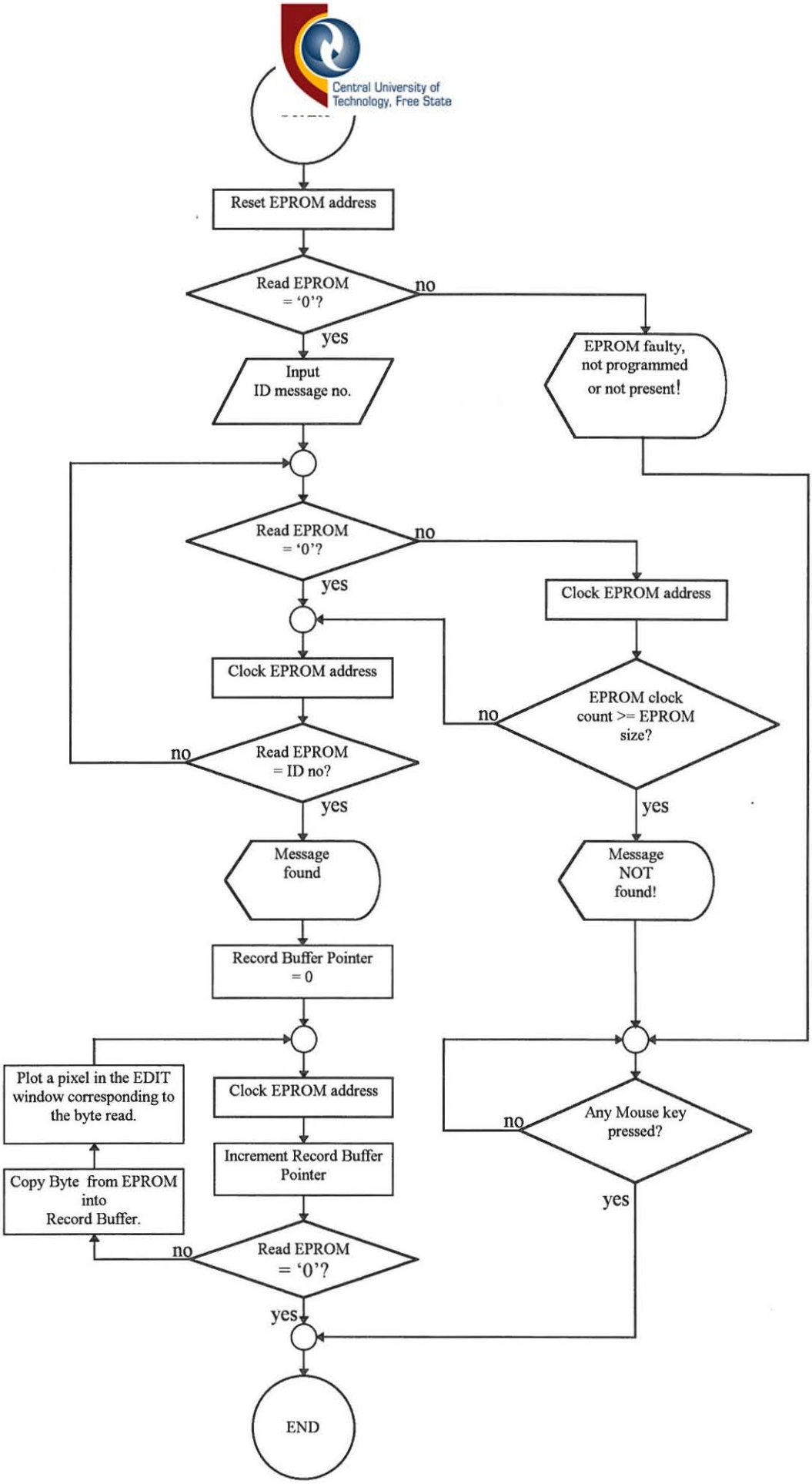


Figure 5.9 ReadE procedure flowchart.

5.3 Conclusion

The software provides the user with a graphical interface that is both efficient and easy to use.

No specialized programming skills or previous knowledge of the system is necessary to operate it as all functions are mouse driven.

CHAPTER 6

EVALUATION

6.1 Introduction

In order to evaluate the system, a prototype had to be constructed, consisting of:

- Computer interface.
- Record/Playback/Programmer module.
- Speech circuit.
- Software.

6.1.1 Computer interface

The computer interface is based on a PC-35 prototyping board. The prototyping board consists of an interface section and a prototyping section. For the purpose of this project, the prototyping section has been separated from the interface section. This was done in order to simplify construction, and to isolate the analogue circuits from possible sources of interference from within the computer. The original PC-35 interface was slightly modified to allow the use of hardware generated interrupts to the host computer. Figure 6.1 is a photograph of the computer interface.

6.1.2 Record/Playback/Programmer module

The Record/Playback/Programmer module is constructed on the prototyping area of the original PC-35 prototyping board, as depicted in Figure 6.2.

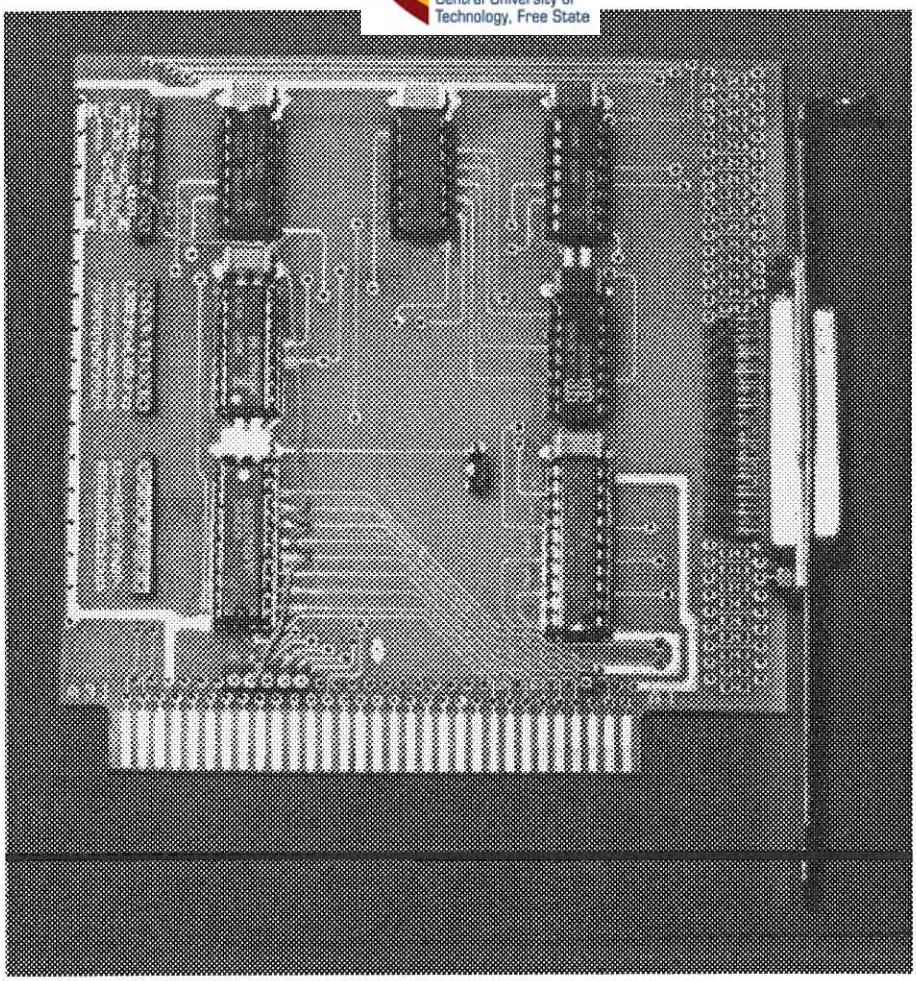


Figure 6.1 Computer Interface Card

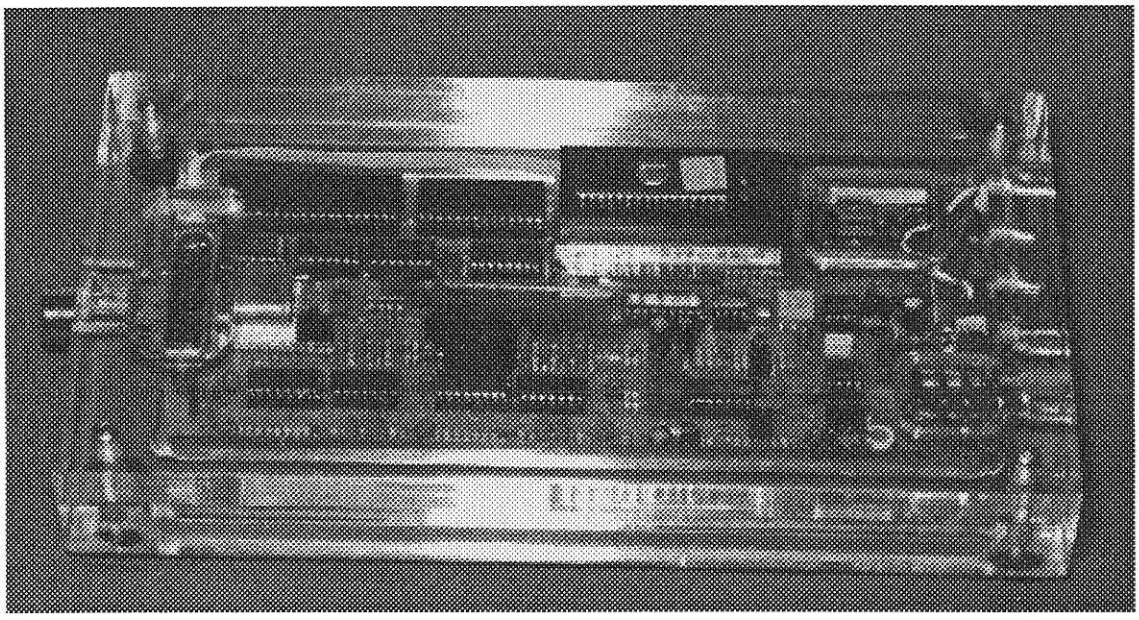


Figure 6.2 Record/Playback/Programmer module

The fragile nature of the construction method used on the protoboard made it necessary to encase the board in a perspex enclosure for protection.

6.1.3 Speech circuit

The Speech circuit is constructed on a printed circuit board specifically designed for this purpose. Figure 6.3 is a photograph of the Speech circuit together with a memory expansion module.

6.1.4 Software

The software is written in Turbo Pascal version 6.0. A copy of the software is provided as a diskette at the back of the paper, and the source code is provided as appendix C. The software automatically switches to a demo mode when the hardware associated with the project is not detected. This allows an evaluation of the software when the system's hardware is not available. Figure 6.4 is a photograph of the software as it appears on a computer screen.

6.2 Using the system

6.2.1 Recording, playback and programming.

Using the system proved to be very easy and users who had no former experience with the system was soon able to use it. Programming a message into an EPROM may be as simple as:

- Press the RECORD button and say the message into the microphone.
- Use the PLAY function together with COPY, MOVE, DELETE and Limit1 and Limit2 to edit the recording.
- Insert an EPROM into the programmer socket and press the WriteE button.
- When prompted, enter an ID number for the message.

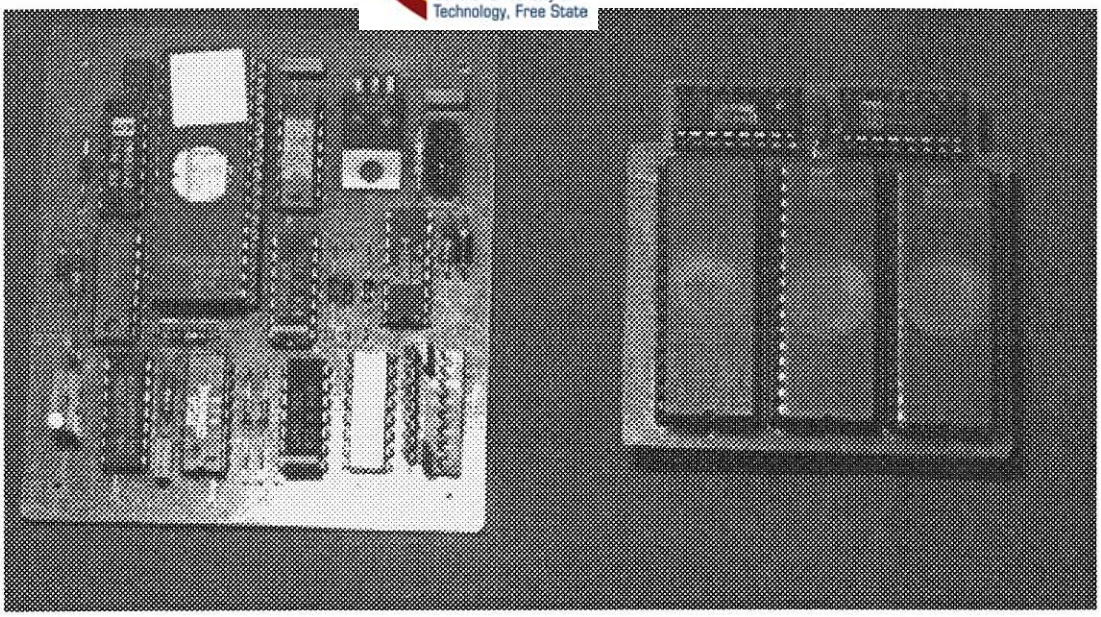


Figure 6.3 Speech circuit together with memory expansion unit

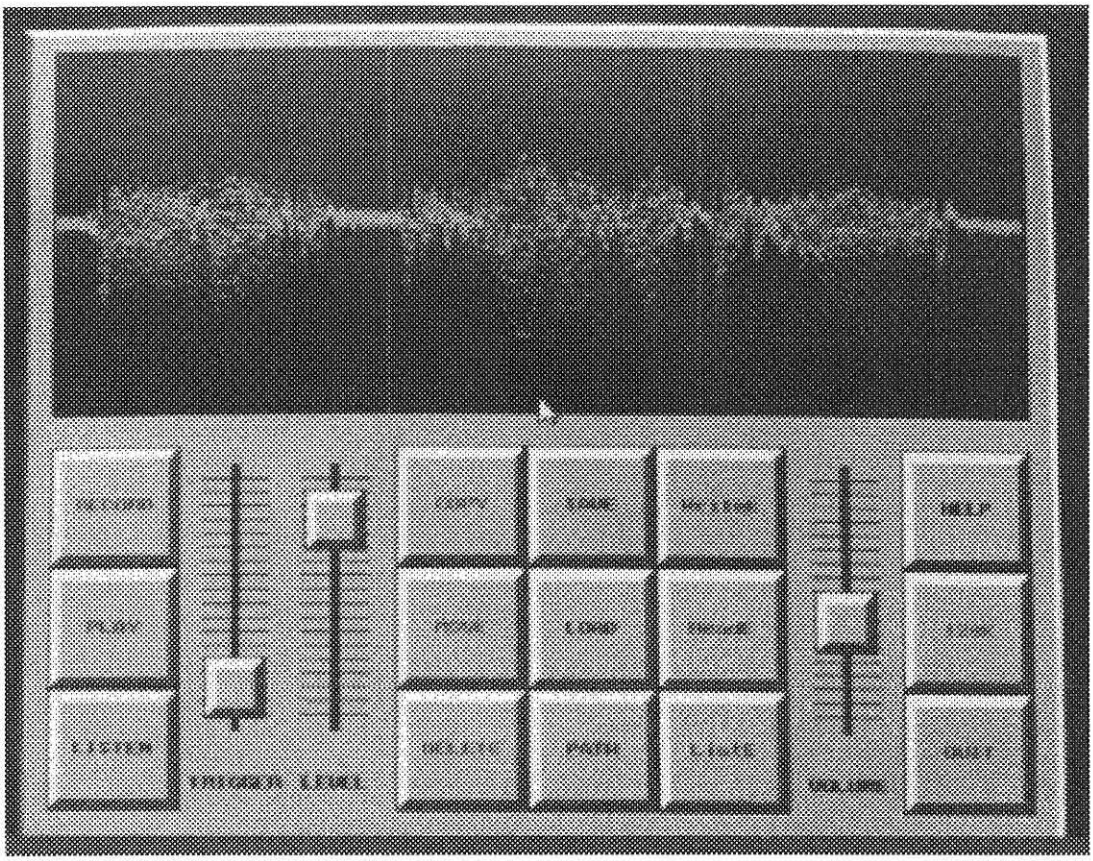


Figure 6.4 Software as seen on a computer monitor

- When the programming sequence is complete, insert the programmed EPROM back into the speech circuit.

Figure 6.5 is a photograph of the setup required to allow recording, playback and programming of messages.

6.2.2 Speech circuit in use

The speech circuit is easy to use as it requires only a start pulse and a 4-bit address to trigger any specific message. The 4-bit address may be generated by a logic circuit or simply by a set of switches. The output of the speech circuit need to be amplified to be audible. No filtering or amplitude expansion has been provided at the output of the speech circuit. In situations where speech quality is important, the user may add an active filter and an expander similar to that used in the Record/Playback/Programmer module. However, the unit proved to be quite intelligible when used with a simple passive filter network between the output of the speech circuit and an audio amplifier. Figure 6.6 is a photograph of the speech circuit connected for an aural evaluation test.

6.3 *Evaluating the system*

6.3.1 Measurements

To evaluate the system, a sine wave at various frequencies was sampled and played back. Measurements were made with a Tektronix FG 501 function generator, Tektronix 2230 oscilloscope, and a PC-30 Analog I/O card with STATUS-30 software.

The Oscilloscope was used for all time domain measurements, and the PC-30/STATUS-30 for all frequency domain measurements. Measurements were made at 300Hz and 3400Hz — the extremes

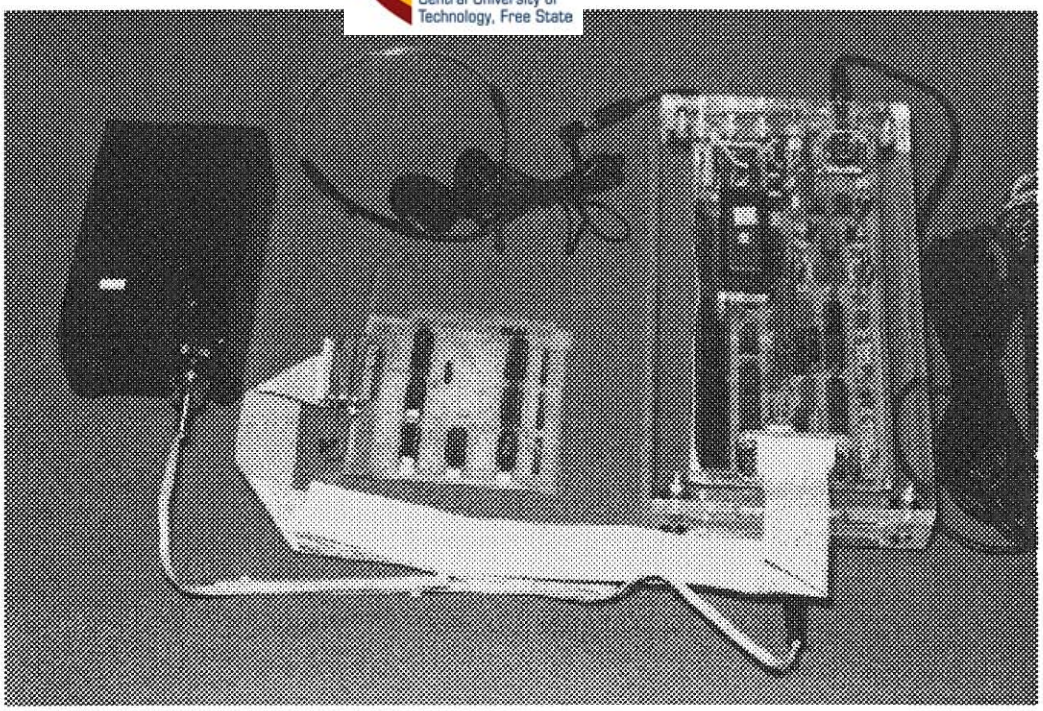


Figure 6.5 Setup of complete system

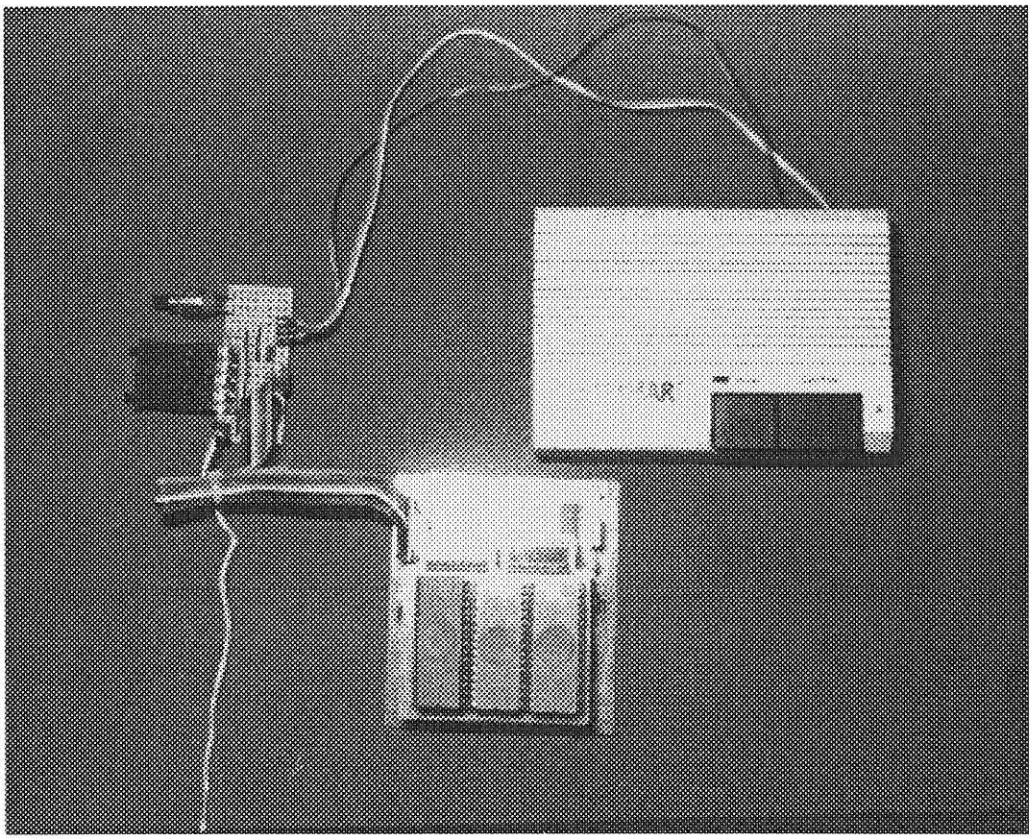


Figure 6.6 Speech circuit under aural evaluation

of normal speech [36] — and at 1k[illegible] frequency for audio applications. Figures 6.7 to 6.9 show the results of the 1kHz test, while the results of the 300Hz and 3400Hz tests appear as appendix B at the end of this manuscript. The results comprise of a time domain representation on the upper half of the page and a frequency domain representation on the lower.

Measurements were taken at three points on the signal path through the system:

- **Figure 6.7:** Measured at the output of the signal generator to serve as a reference.
- **Figure 6.8:** Measured at the output of the digital-to-analog converter during playback. This is the point where the effects of the digitization process is most noticeable.
- **Figure 6.9:** Measured at the output of the output low pass filter. The low pass filter smoothes out all the rough edges caused by the digitization process, delivering a signal which closely resembles the original.

The readings taken at 300Hz and 3400Hz were done in a similar manner as the above. As can be seen from the results depicted in appendix B, there is great similarity between the input signal and the filtered output signal. As the input frequency approaches half the sampling frequency, distortion of the recovered signal becomes noticeable in the 3400Hz test. Because a sampling frequency of 6944Hz is used[37], it implicates a maximum possible input frequency of

6944Hz / 2 = 3472Hz

Because the cutoff frequency of the low pass filter is set at 2844Hz[38], the 3400Hz test signal is partly suppressed.

---

[36] see paragraph 4.2.1

[37] see paragraph 4.3.1.5

[38] see paragraphs 4.3.1.3 and 4.3.2.2

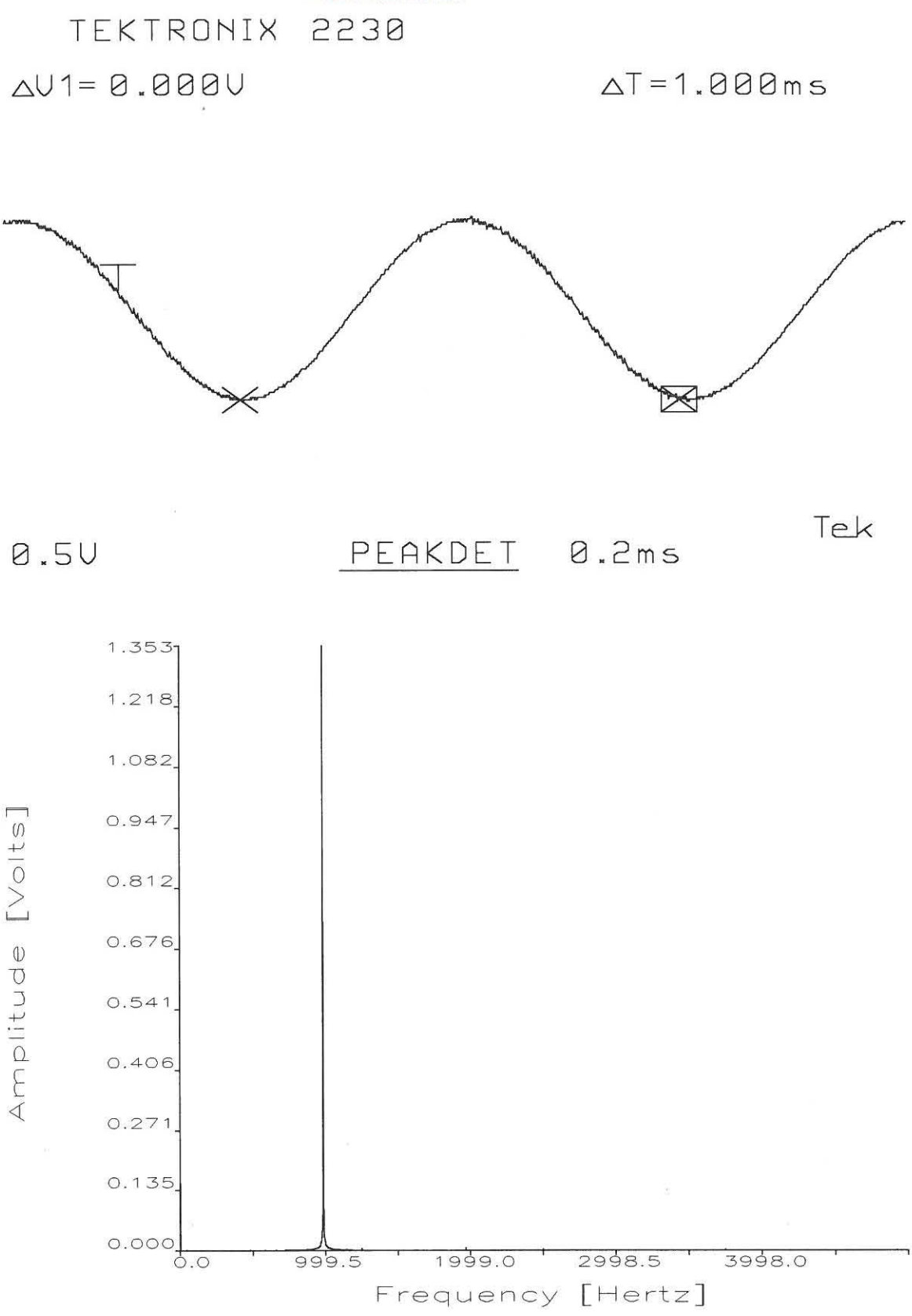


Figure 6.7 1kHz reference input signal.

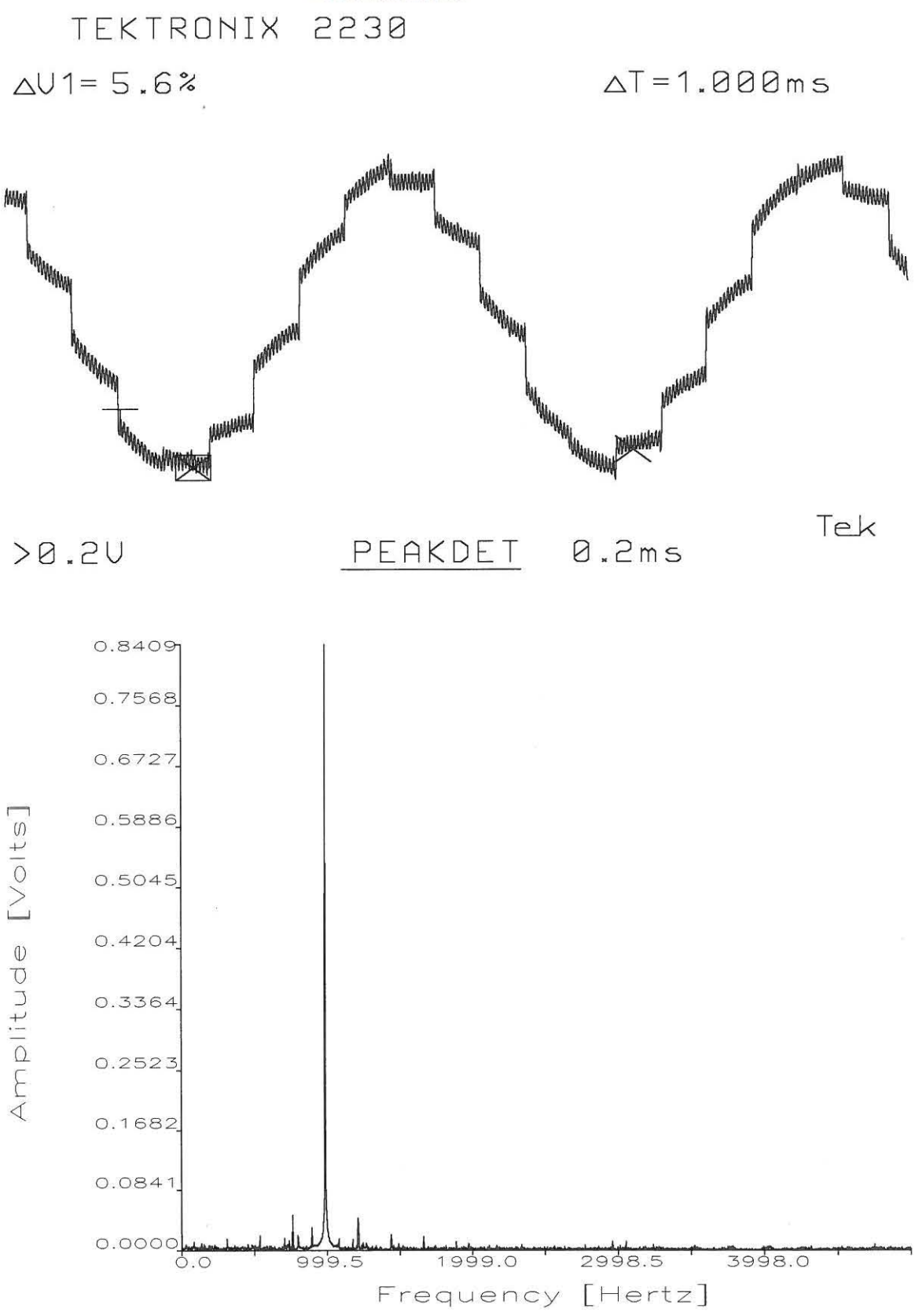


Figure 6.8 1kHz signal recovered from the DAC.

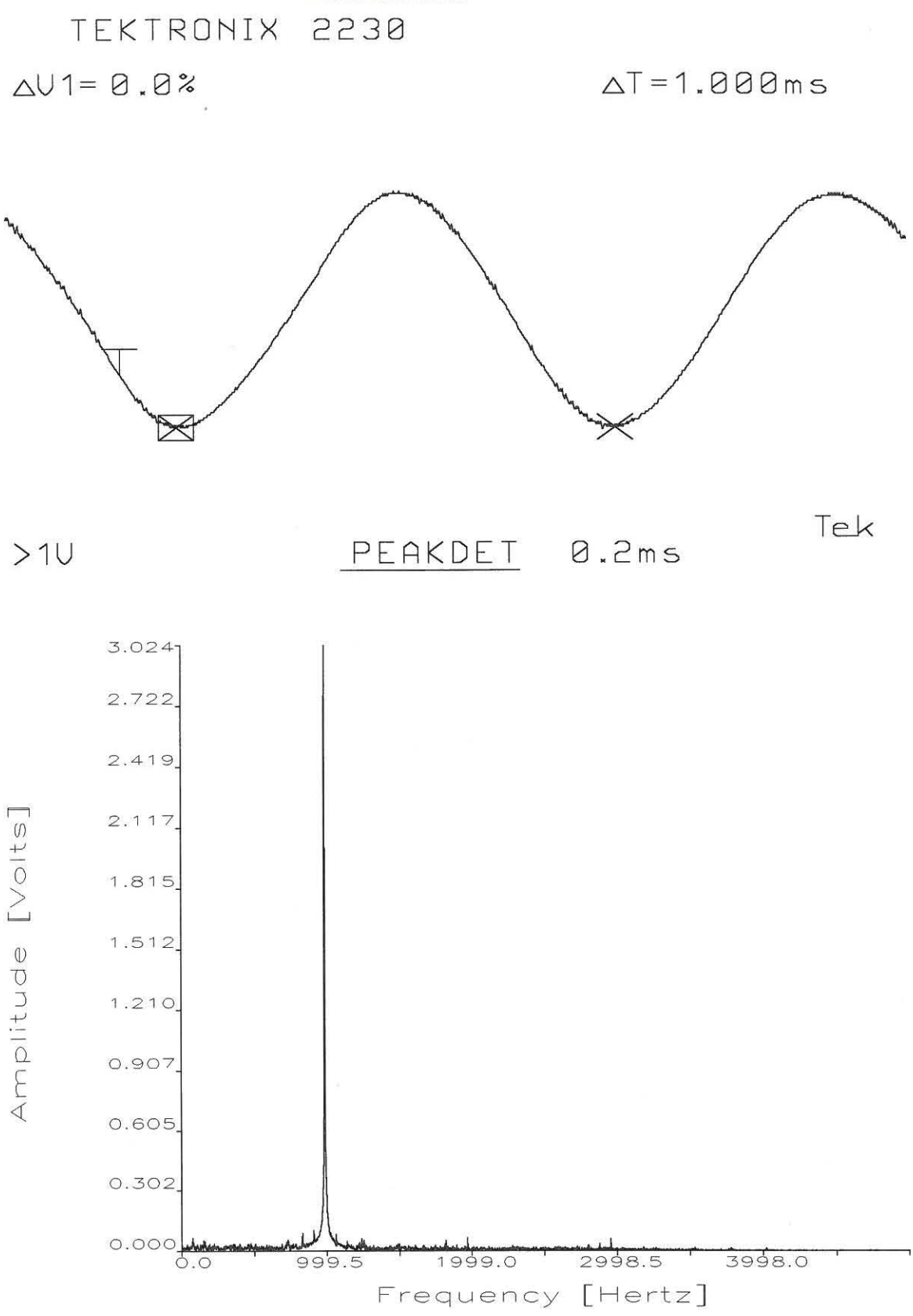


Figure 6.9 1kHz signal at output of system.

Beyond this frequency, recovery of the original signal would no longer be possible, leading to alias distortion[39]. Frequencies exceeding half the sampling frequency is prevented from entering the system by the low pass filter[40] — the measured frequency response of which is represented by Figure 6.10.

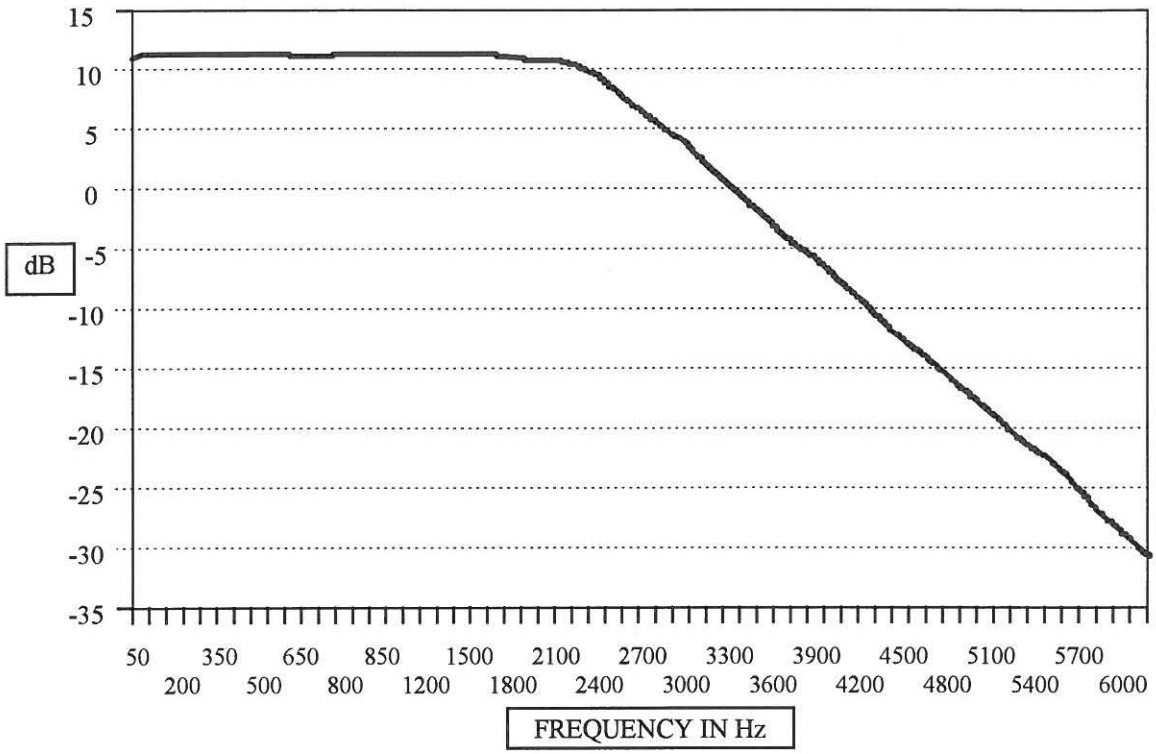


Figure 6.10 Frequency response of input/output low pass filter.

6.3.2 Aural tests

Aural tests were conducted by sampling various signal sources and listening to results. These included spoken messages recorded by means of a microphone and directly sampling audio material from a compact disk player by means of the auxiliary input of the system.

Although there was an inevitable loss of high frequency content, as could be expected by the limited bandwidth of the system, speech was always clear and natural sounding.

---

[39] see paragraph 3.2.1

[40] see circuit diagram 6, appendix A

The compander reduced quantization noise to an almost unnoticeable level, as long as care was taken in keeping the input signal near the overload point of the system without exceeding it[41] .

6.4 CONCLUSION

This study proves that a speech synthesis system based on the replay of messages stored in EPROM, without the use of any data compression techniques, may in many cases prove to be an attractive alternative over other existing speech synthesis systems.

The following benefits can be accredited to this system:

- Very easy to program. No specialized skills are necessary.
- Clear natural speech in any language is possible.
- The programmed speech units does not need a computer or other complicated circuits to control it.
- Recent drops in EPROM prices make the system relatively inexpensive as a whole.

By completion of this project, the researcher has gained invaluable knowledge and experience in the following fields:

- Speech synthesis
  - Origins
  - Different speech synthesis techniques
  - Problems associated with synthetic speech
  - Using PCM for synthetic speech

[41] see paragraph 3.2.2

- Digital circuitry
    - How to use analogue-to-digital and digital-to-analogue converters.
    - How to use and program EPROM's.
    - The influence of invalid data conditions on digital circuits and how to avoid it.

- Analogue circuitry
    - Design of active analogue filters
    - The use and working principles of a compander

- Pulse Code Modulation
    - Quantization noise and methods of reducing it
    - Alias distortion and avoiding it

- IBM compatible personal computers
    - Working principles of the computer bus
    - Using and programming of the 8259 interrupt handler

- Software: How to program in Turbo Pascal

    In particular:

    - Programming mouse routines
    - How to use and handle hardware generated interrupts
    - Writing to and reading data from disk
    - Designing practical software layout.
    - Data manipulation.

CHAPTER 7

SUMMARY

Chapter 1 gives a short description of everyday speech synthesis applications. Some of the disadvantages associated with present speech synthesis techniques are highlighted. The purpose of this study, the method of research and the problems encountered are briefly discussed.

Chapter 2 gives a simplified description of human speech production, followed by a short account of early attempts at speech synthesis. The main speech synthesis techniques in present use are discussed.

Chapter 3 investigates pulse code modulation as recording technique. Attention is paid to causes of waveform degradation and possible ways of avoiding or lessening the effects of the degradation.

Chapter 4 discusses the hardware associated with the project. Operation of the various circuits is described in conjunction with the circuit diagrams included in appendix A.

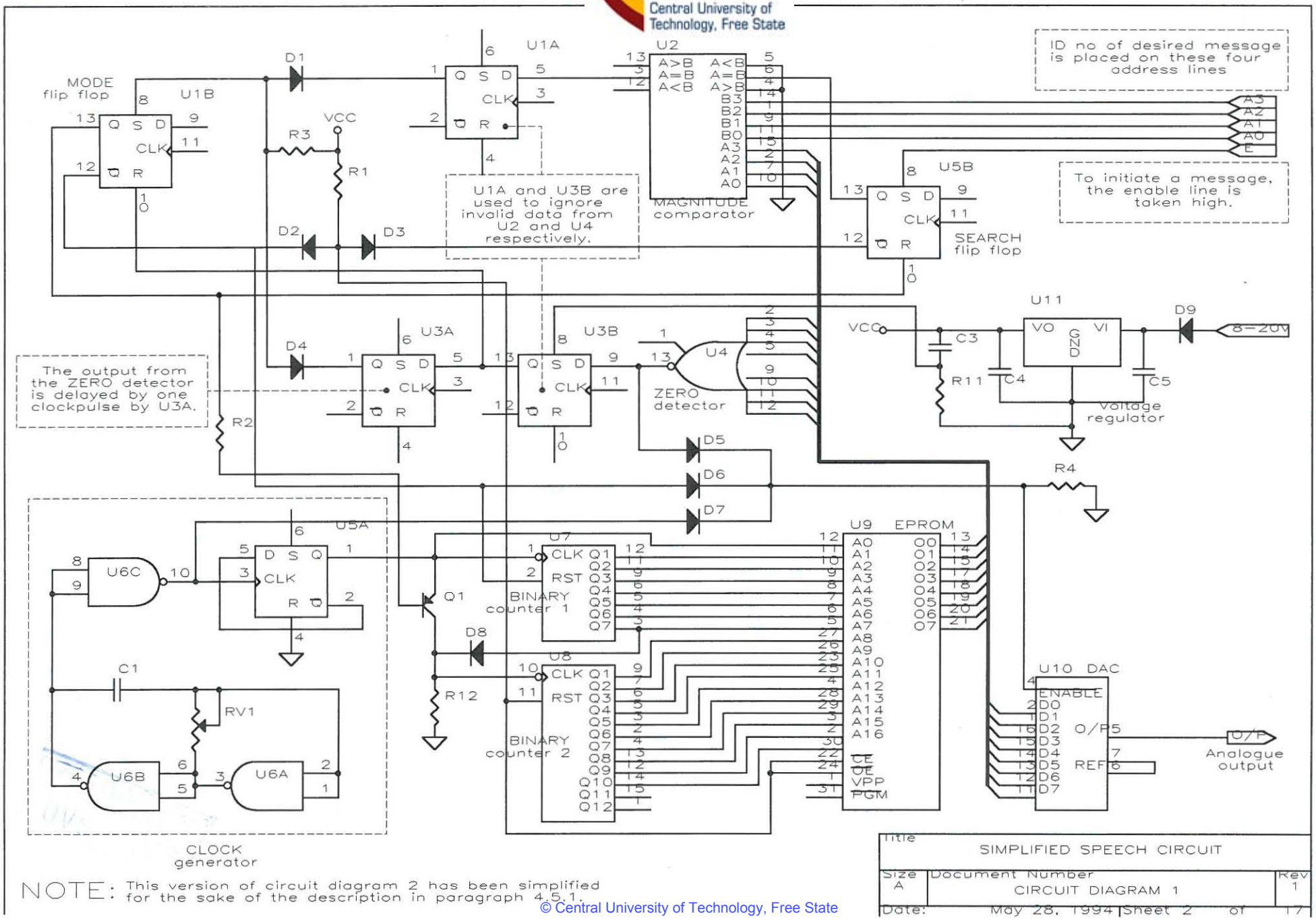
Chapter 5 describes the function and operation of the various software subroutines used in the system. Where necessary, flowcharts are included to clarify the operation of routines.

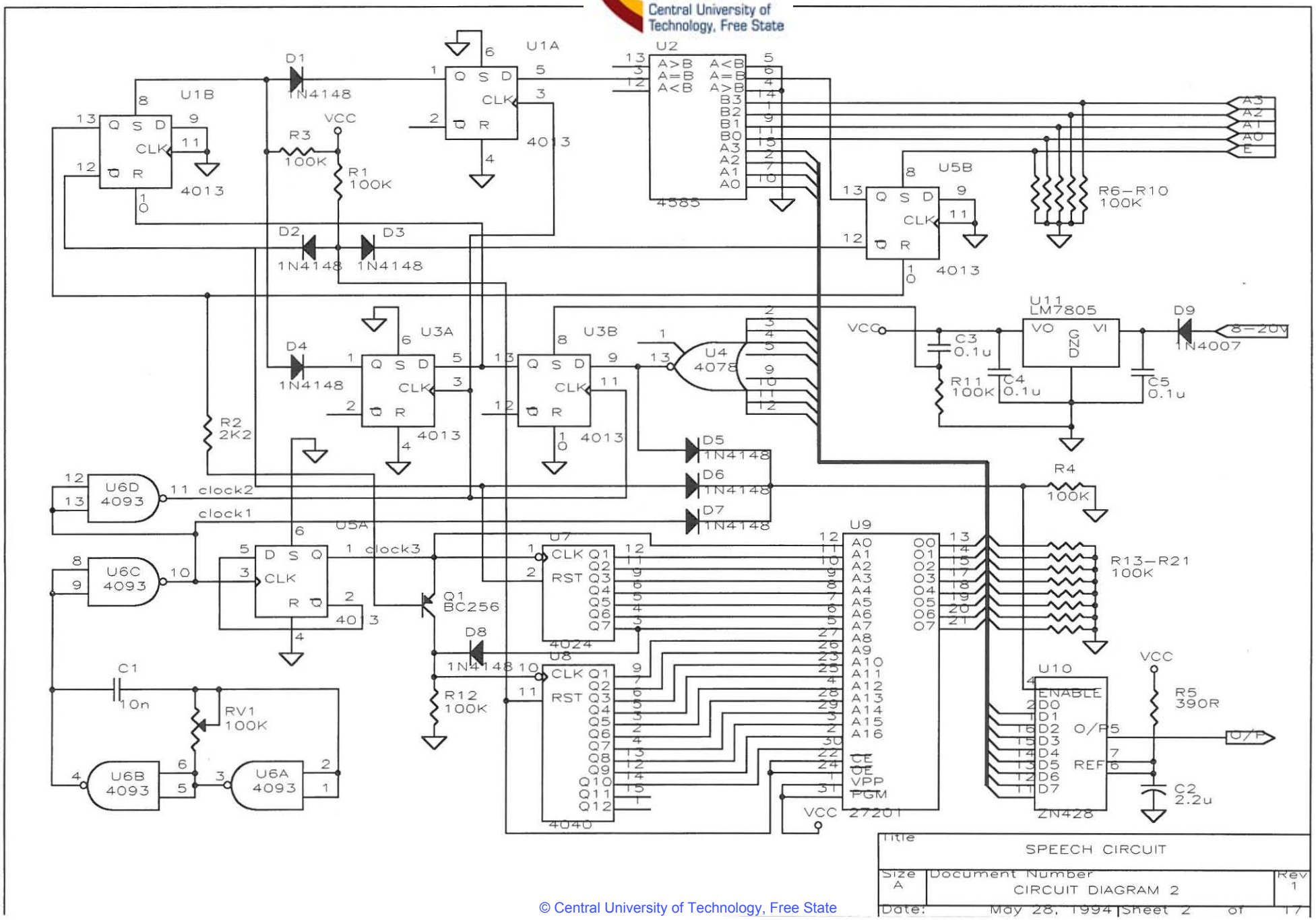
Chapter 6 evaluates the project as a whole.

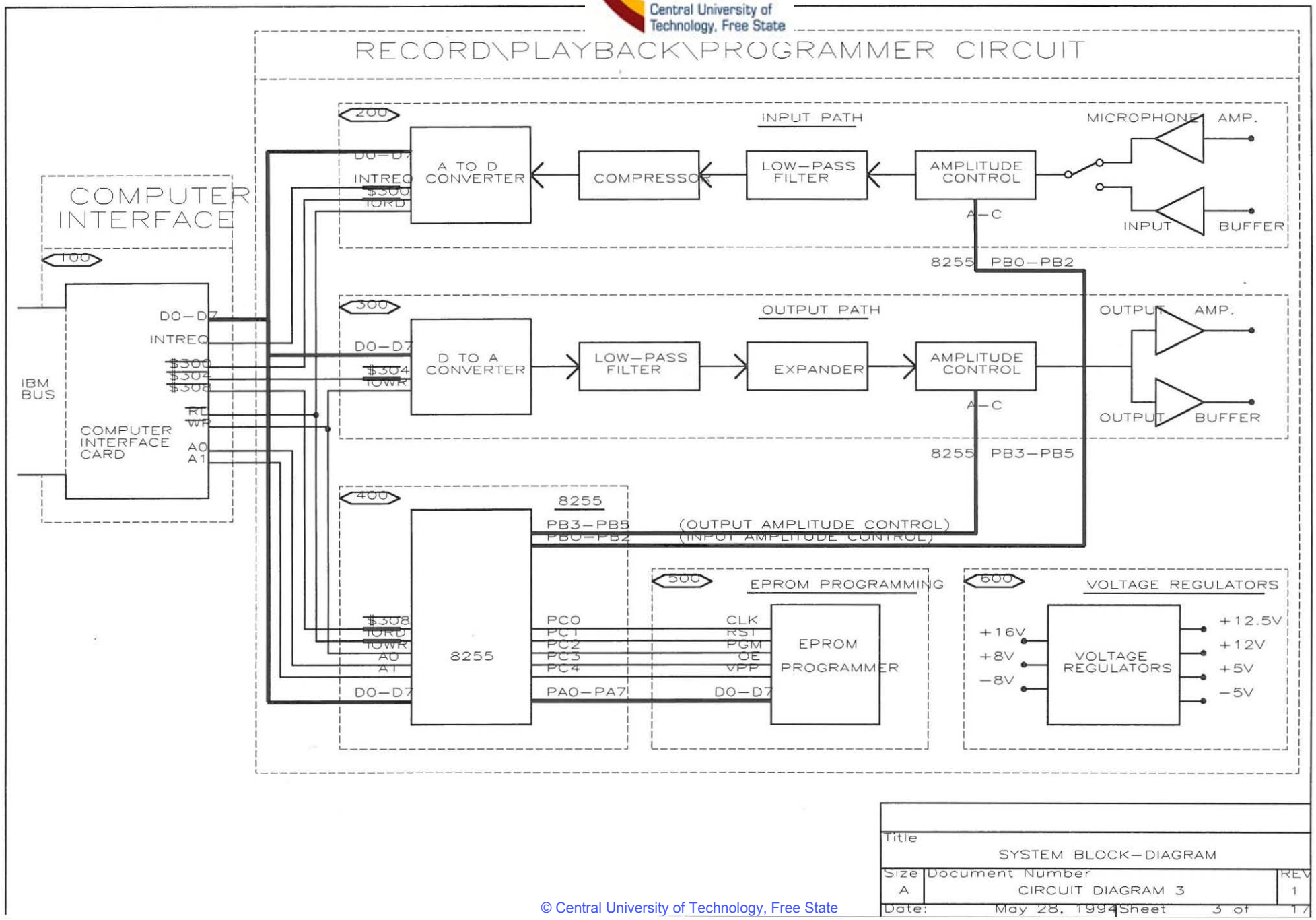
Central University of Technology, Free State

APPENDIX A

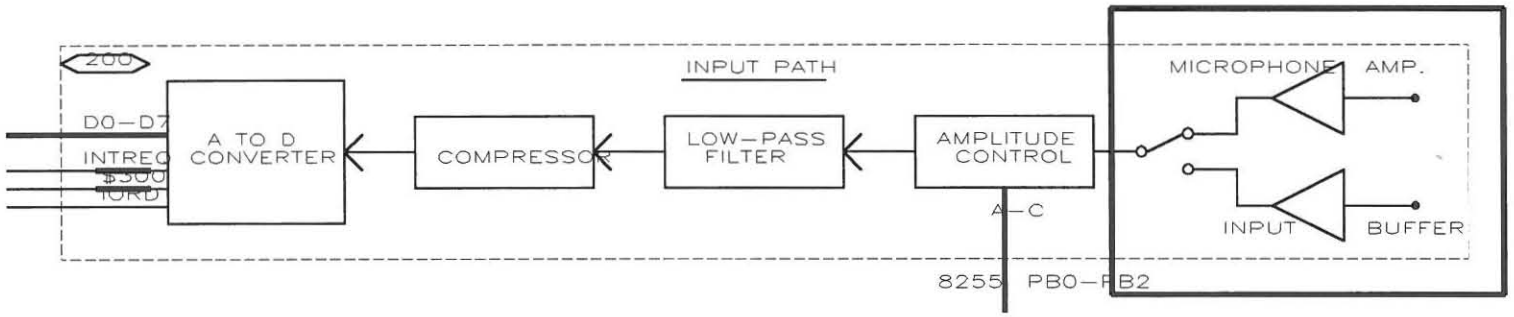
CIRCUIT DIAGRAMS

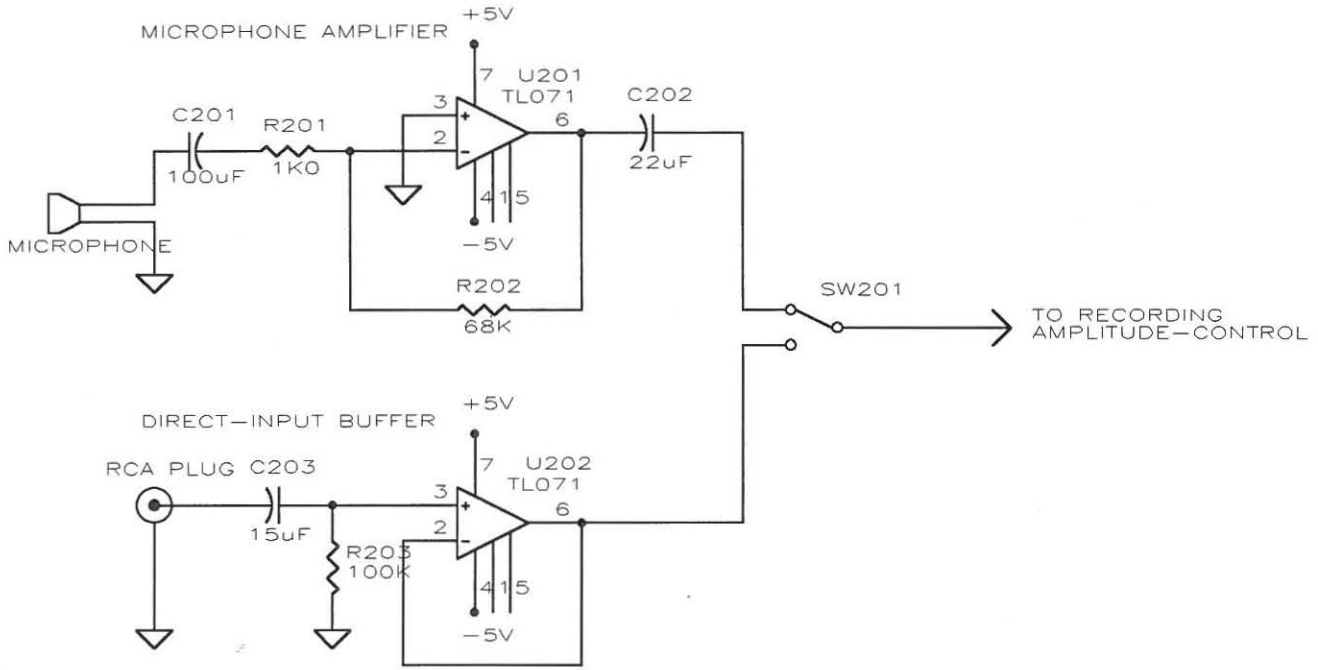




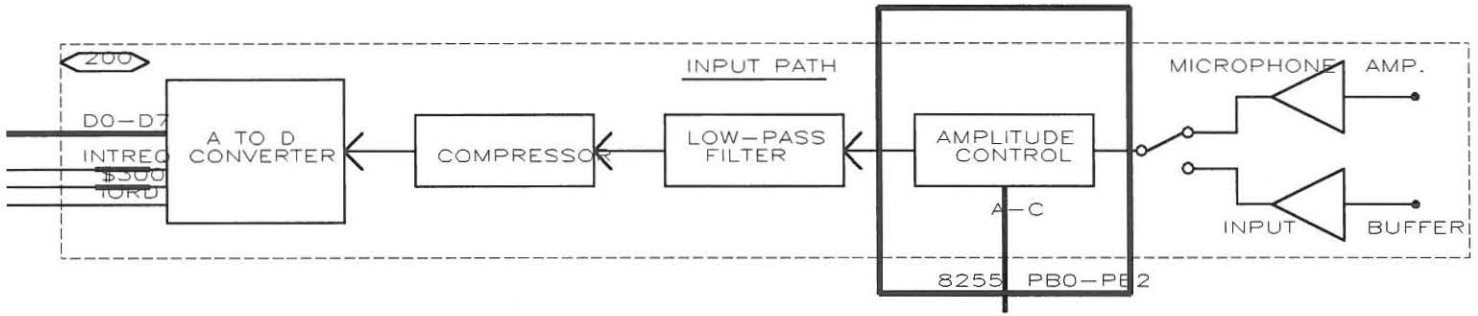


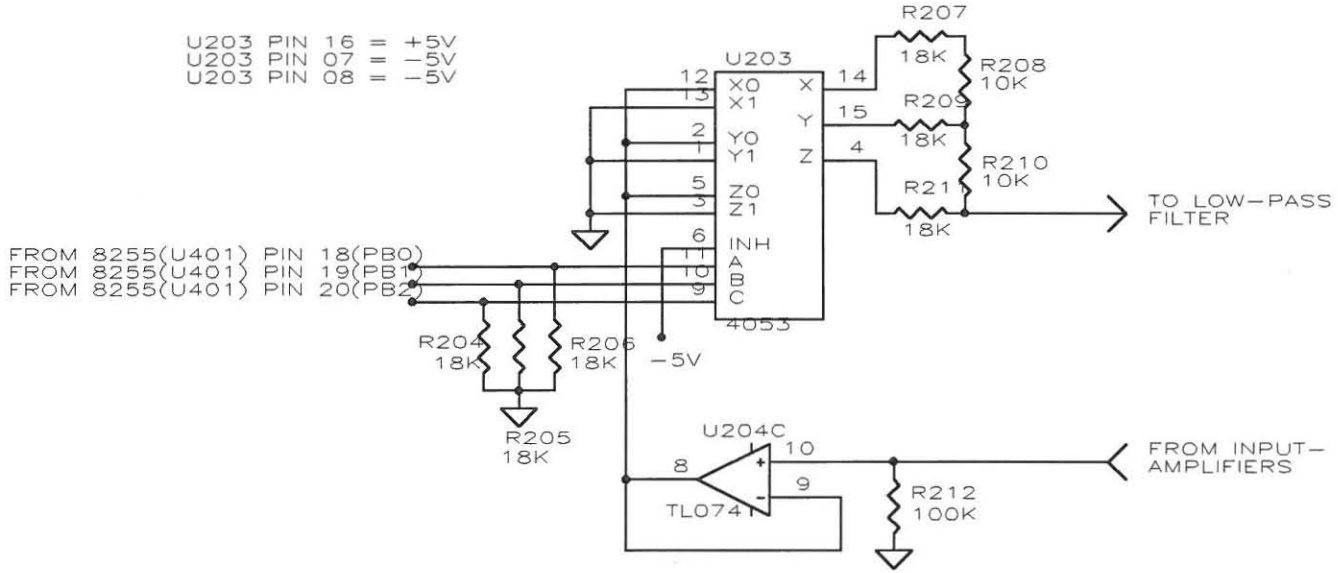
Title	SYSTEM BLOCK–DIAGRAM	
Size A	Document Number CIRCUIT DIAGRAM 3	REV 1
Date: May 28, 1994	Sheet 3 of 17	



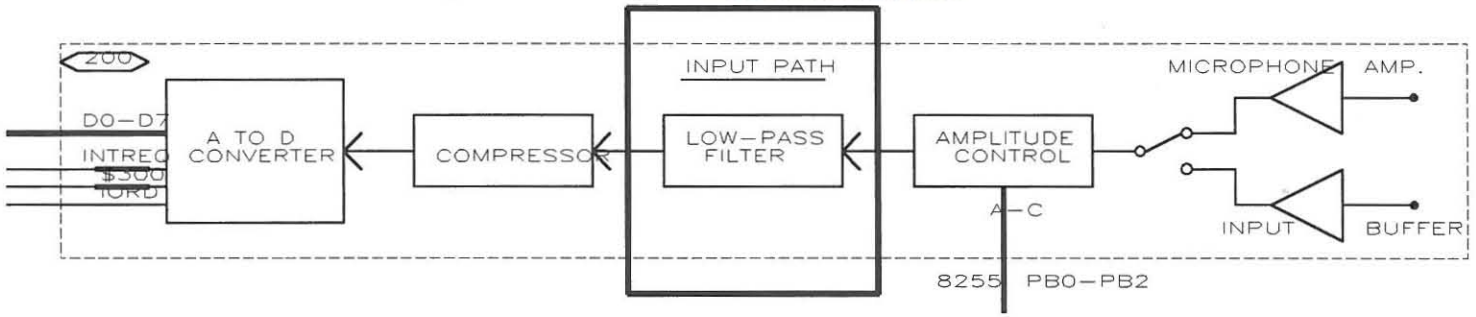


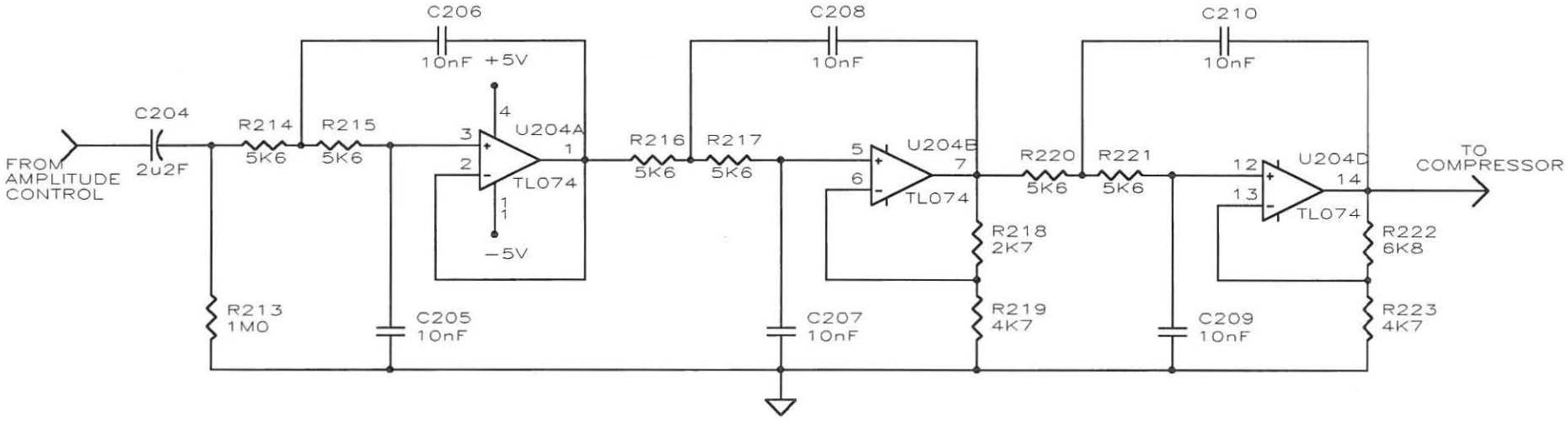
Title: INPUT AMPLIFIERS		
Size: A	Document Number: CIRCUIT DIAGRAM 4	REV
Date: December 30, 1994	Sheet 4 of 17	



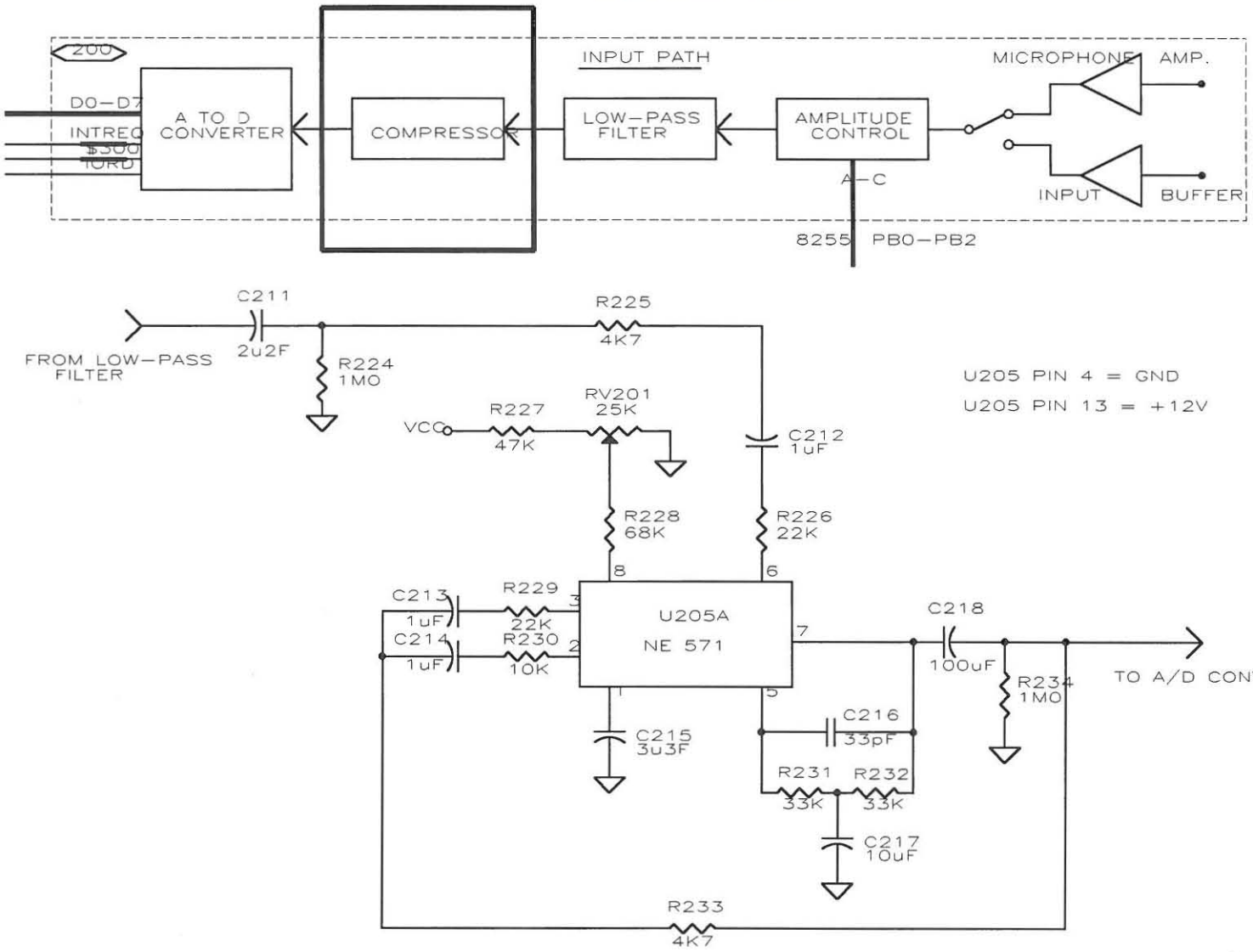


Title: INPUT AMPLITUDE CONTROL		
Size: A	Document Number: CIRCUIT DIAGRAM 5	REV: 1
Date: May 28, 1994	Sheet 5 of 17	

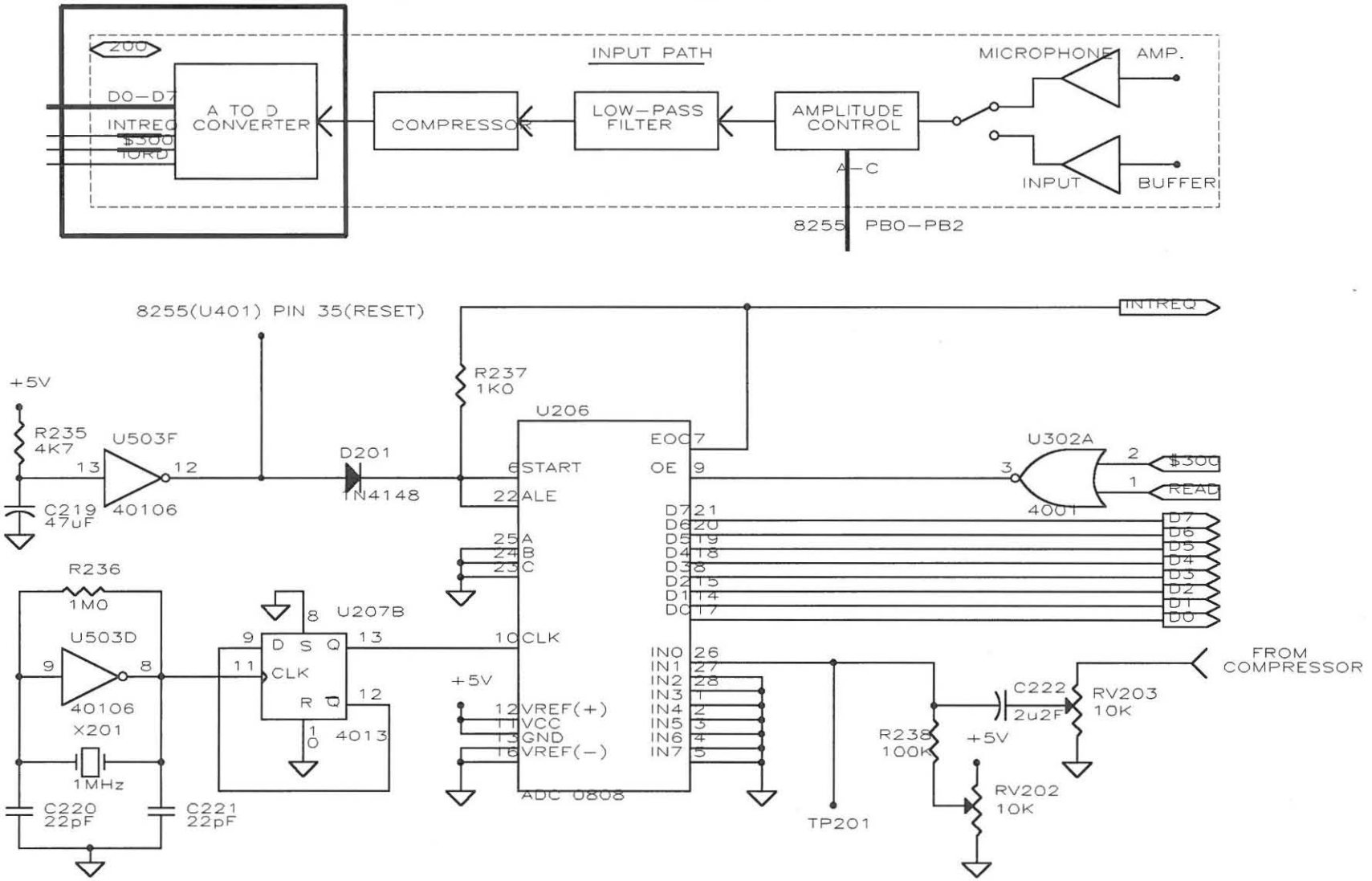


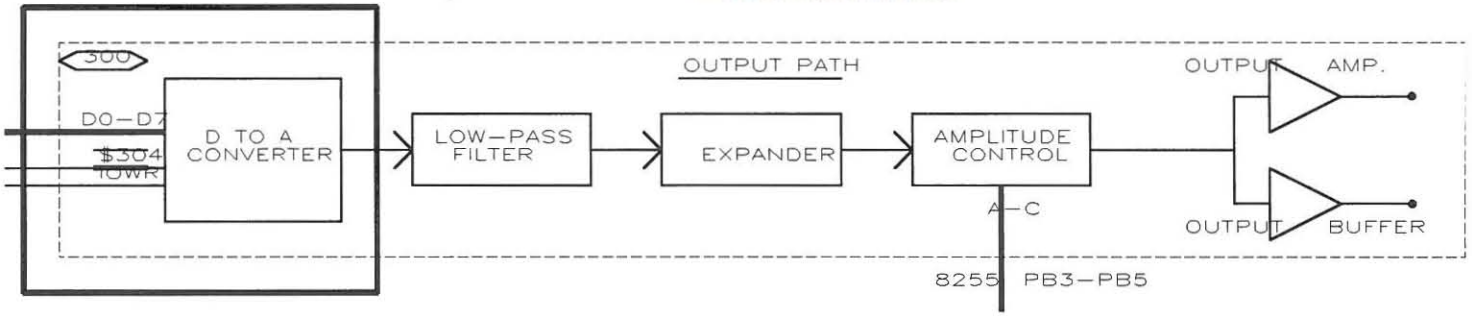


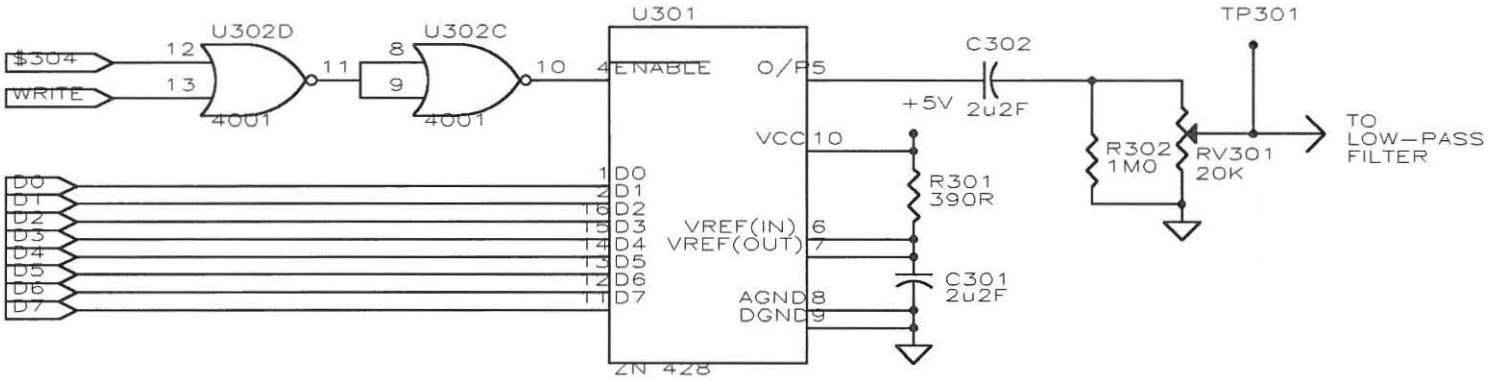
Title: INPUT LOW–PASS FILTER		
Size: A	Document Number: CIRCUIT DIAGRAM 6	REV: 1
Date: May 28, 1994	Sheet 6 of 17	



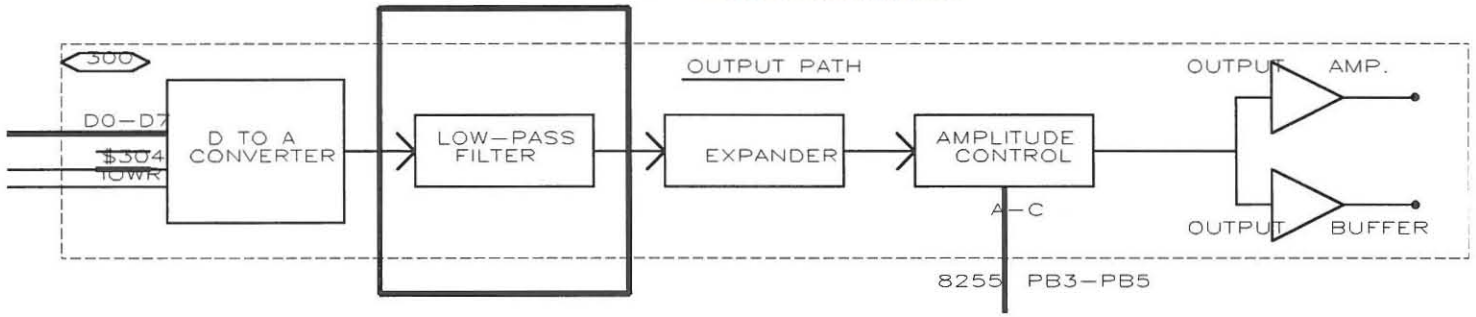
Title: COMPRESSOR		
Size: A	Document Number: CIRCUIT DIAGRAM 7	REV: 1
Date: May 28, 1994	Sheet 7 of 17	

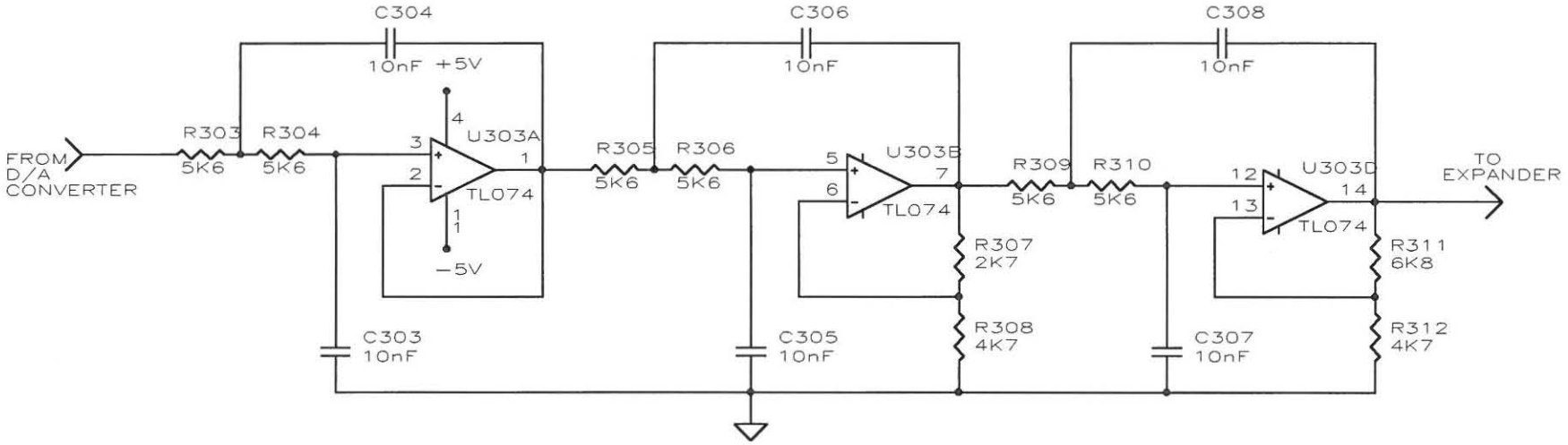




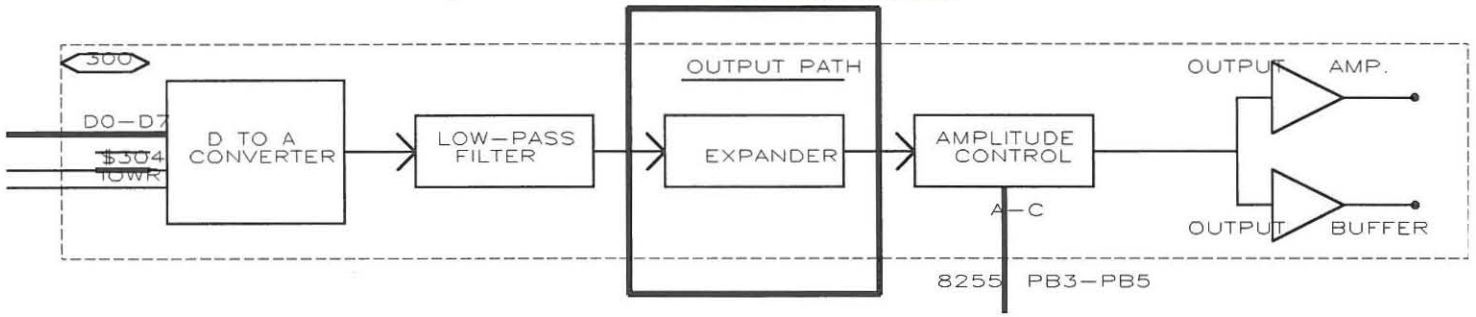


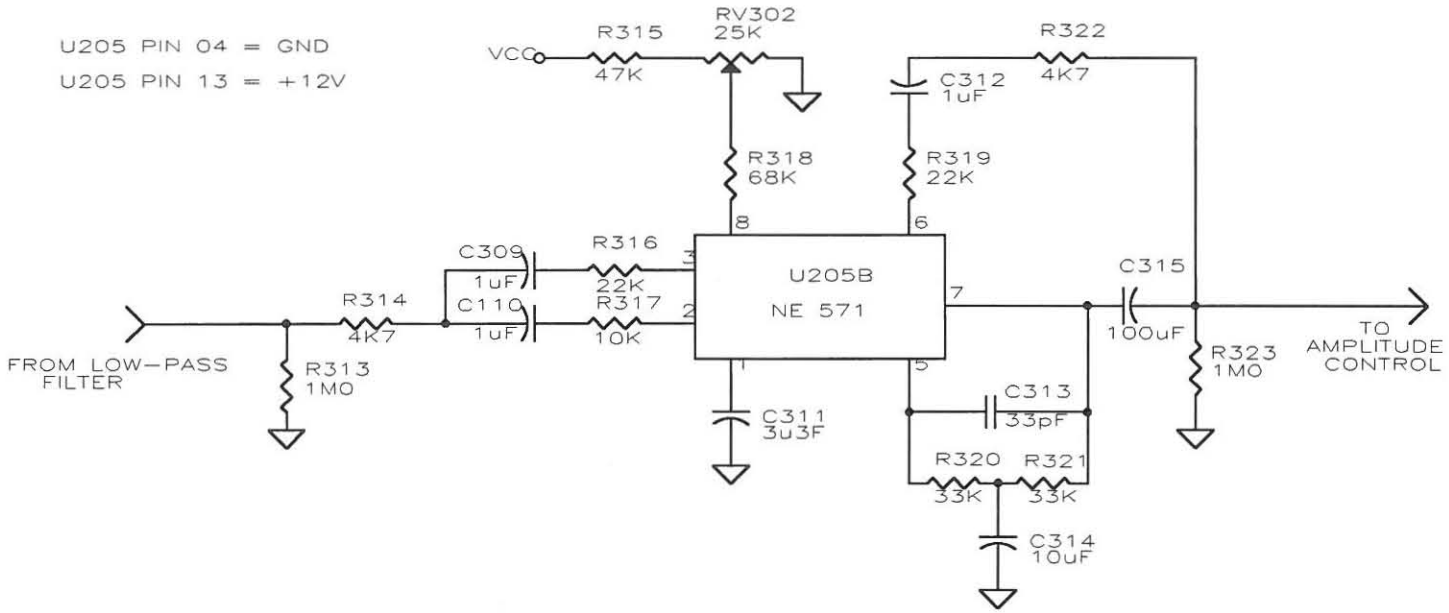
Title: DIGITAL TO ANALOG CONVERTER		
Size: A	Document Number: CIRCUIT DIAGRAM 9	REV: 1
Date: May 28, 1994	Sheet 9 of 17	



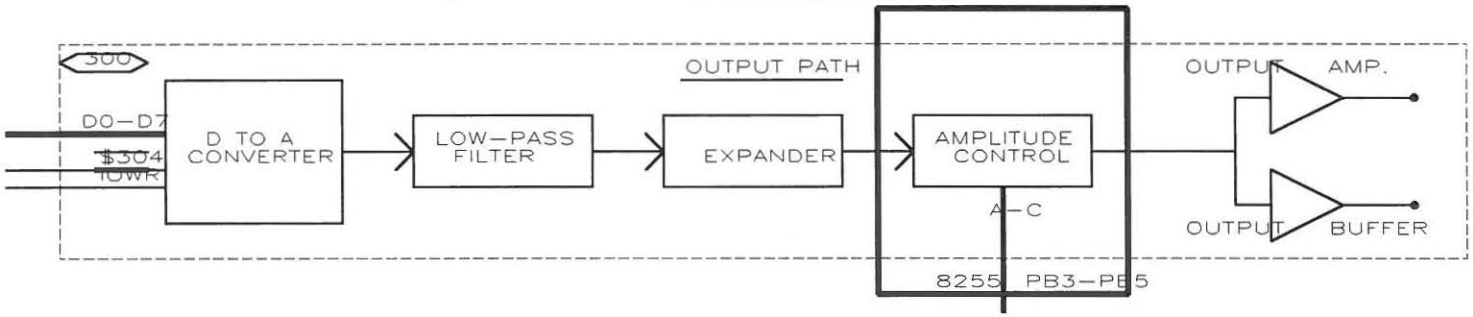


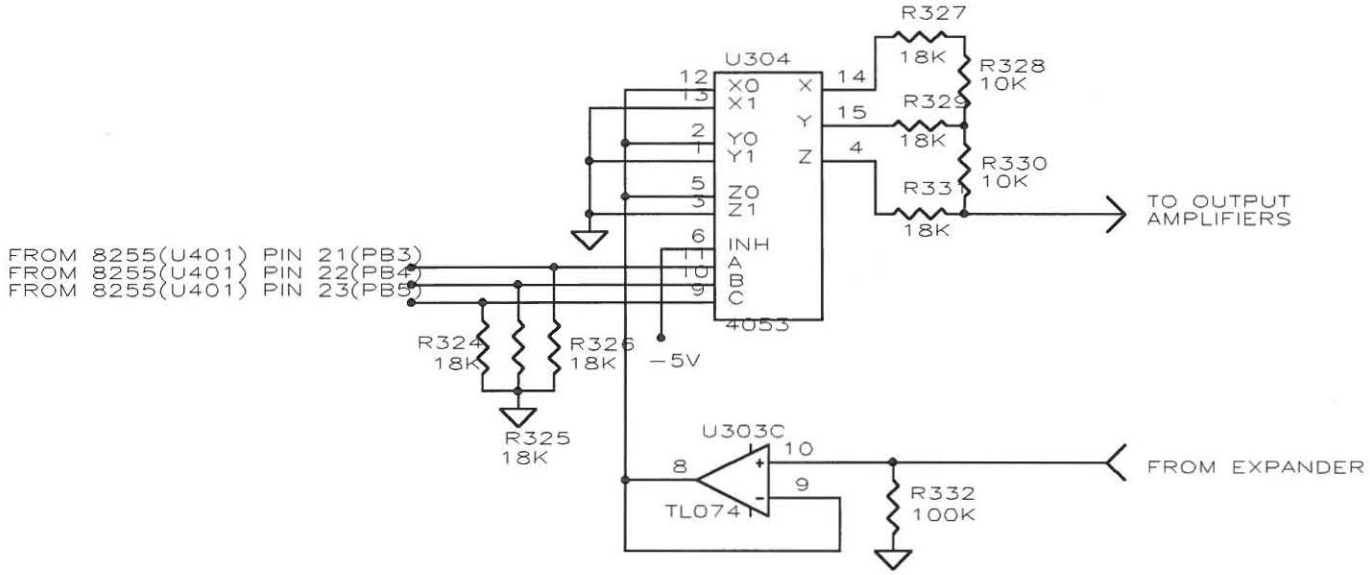
Title: OUTPUT LOW–PASS FILTER		
Size: A	Document Number: CIRCUIT DIAGRAM 10	REV: 1
Date: May 28, 1994	Sheet 10 of 17	



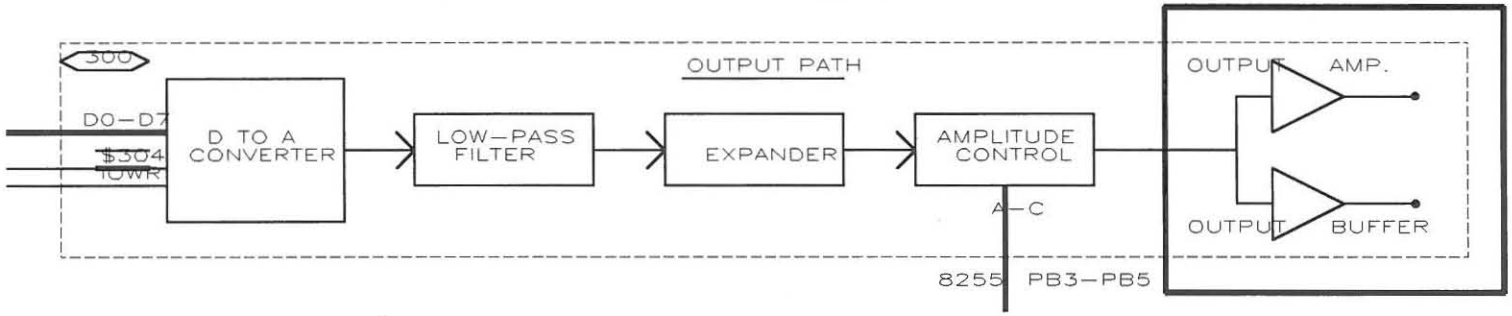


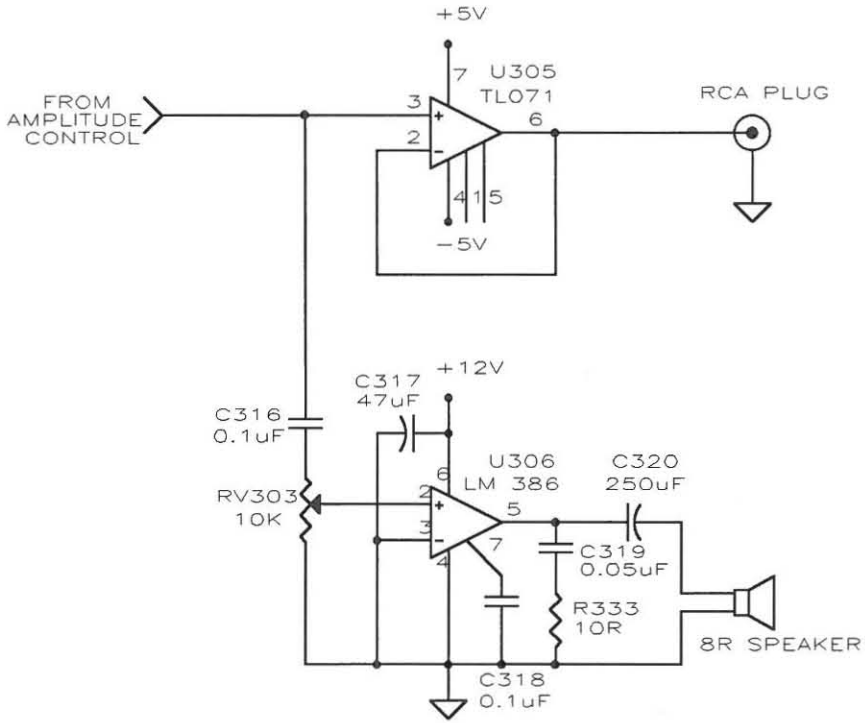
Title: EXPANDER		
Size: A	Document Number: CIRCUIT DIAGRAM 11	REV: 1
Date: May 28, 1994	Sheet 11 of 17	



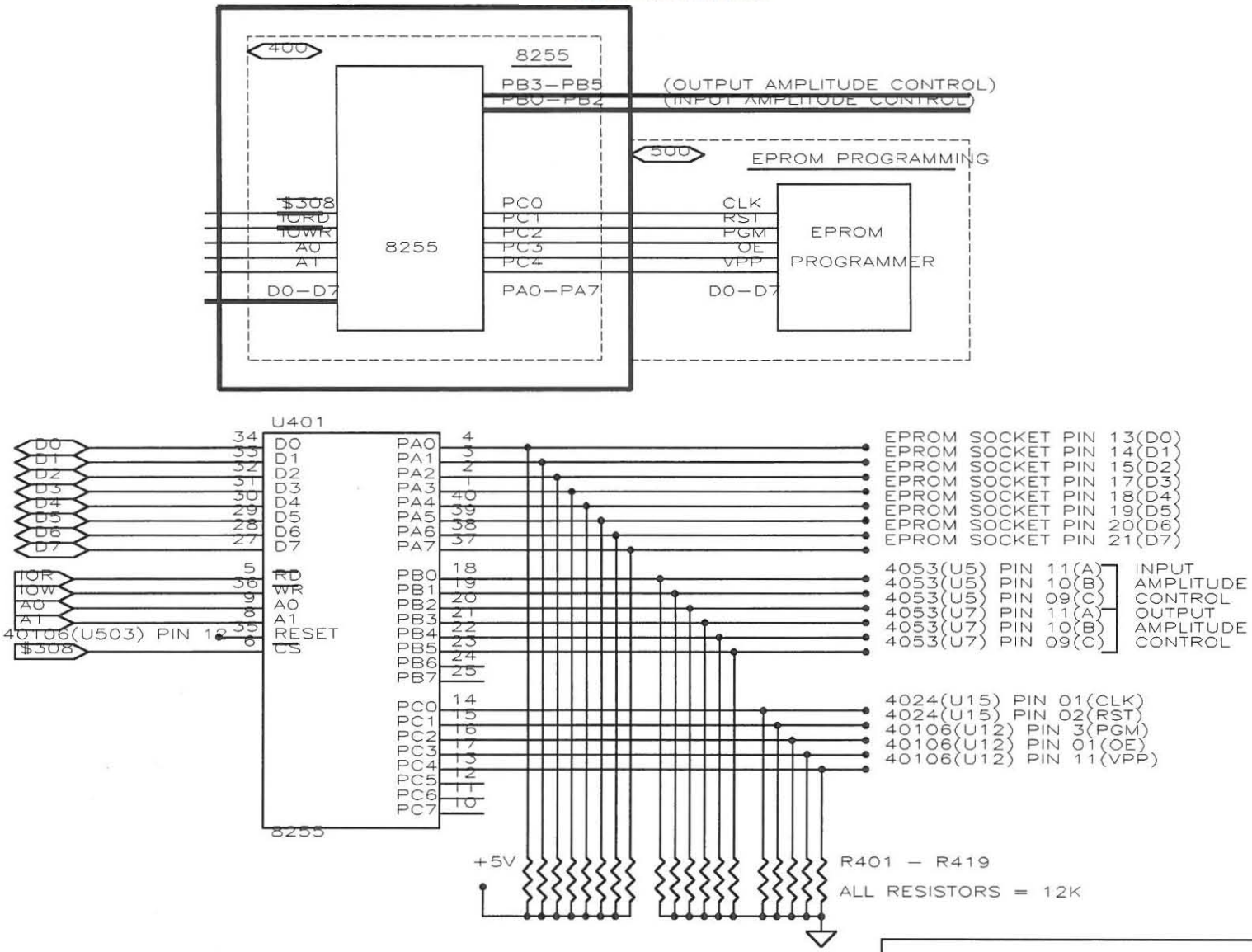


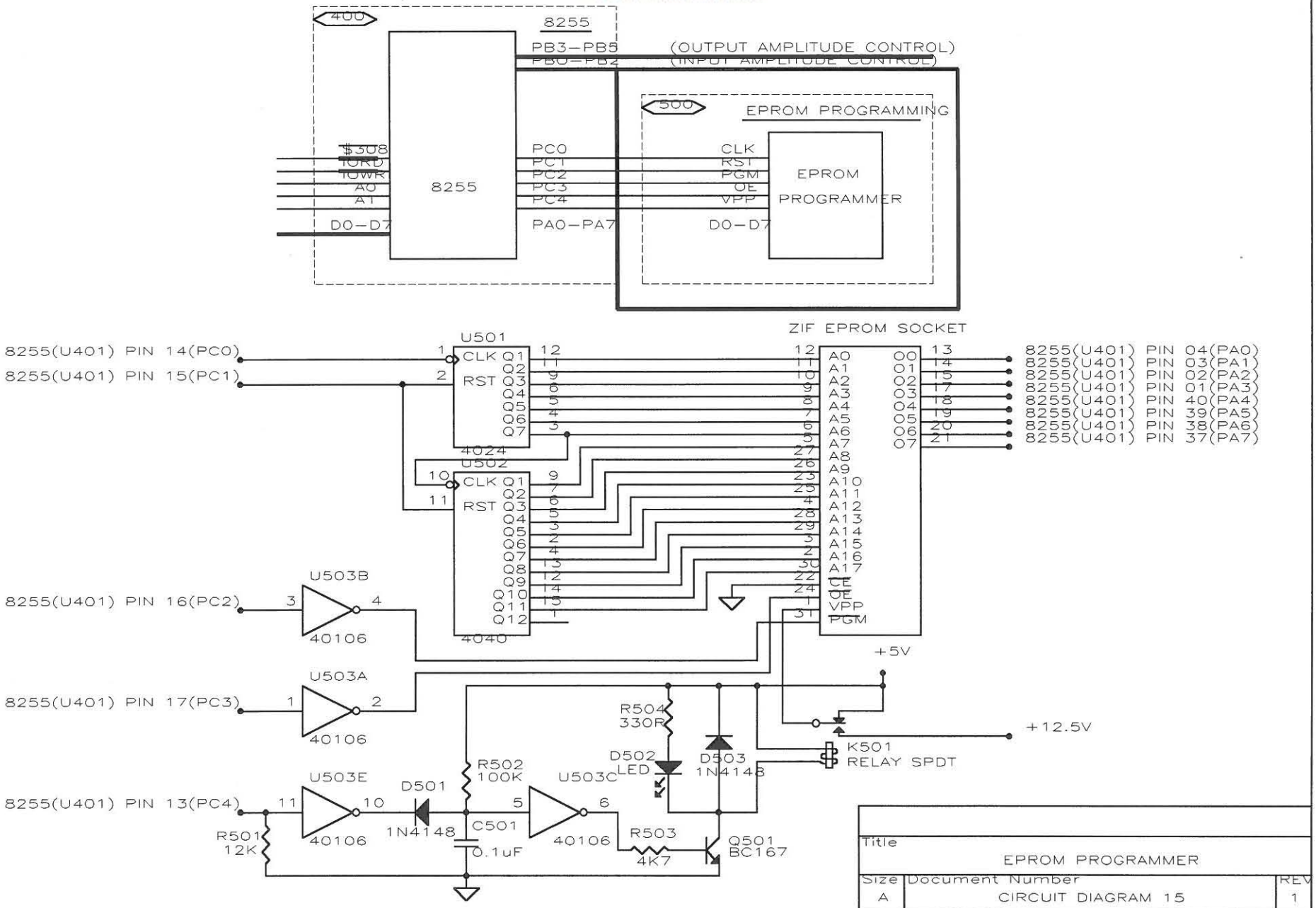
Title: OUTPUT AMPLITUDE CONTROL		
Size: A	Document Number: CIRCUIT DIAGRAM 12	REV: 1
Date: May 28, 1994	Sheet 12 of 17	

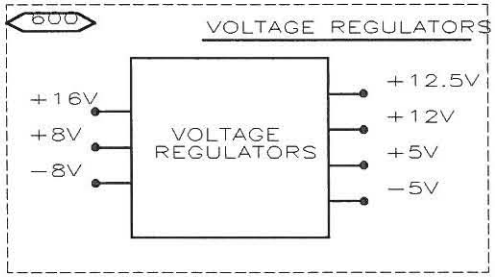


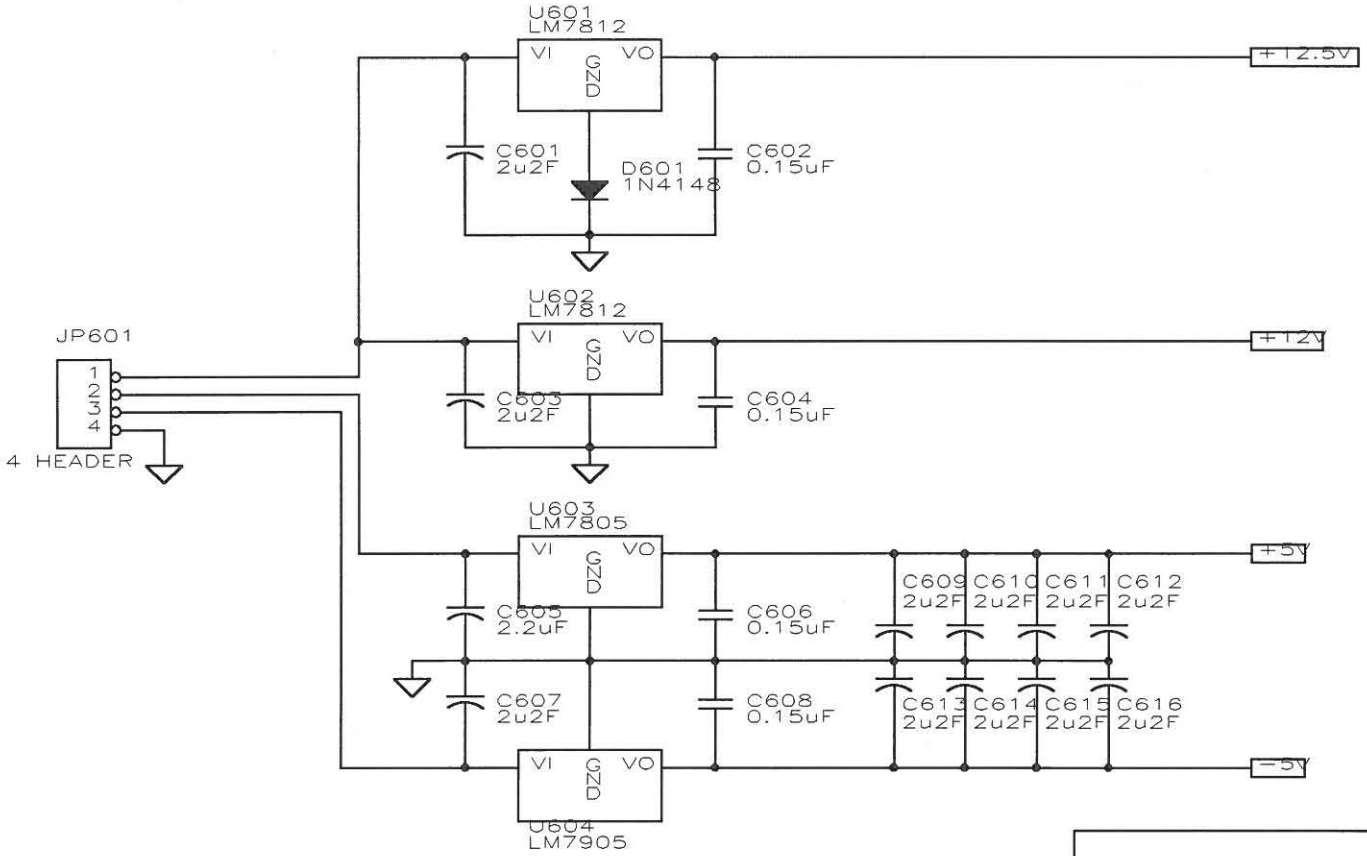


Title: OUTPUT BUFFER AND AMPLIFIER		
Size: A	Document Number: CIRCUIT DIAGRAM 13	REV: 1
Date: May 28, 1994	Sheet 13 of 17	

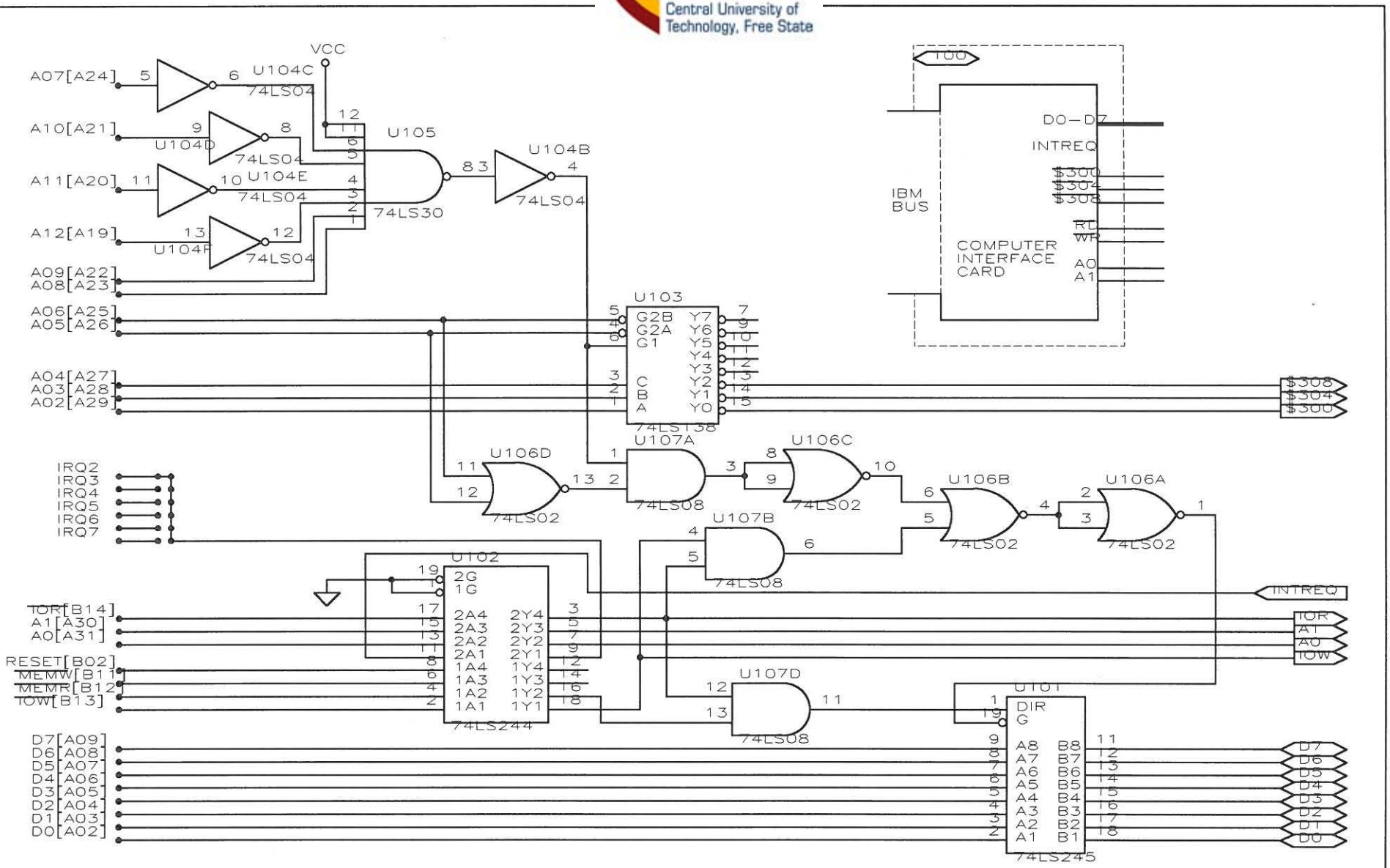








Title: VOLTAGE REGULATORS		
Size: A	Document Number: CIRCUIT DIAGRAM 16	REV: 1
Date: December 29, 1995	Sheet 16 of 17	

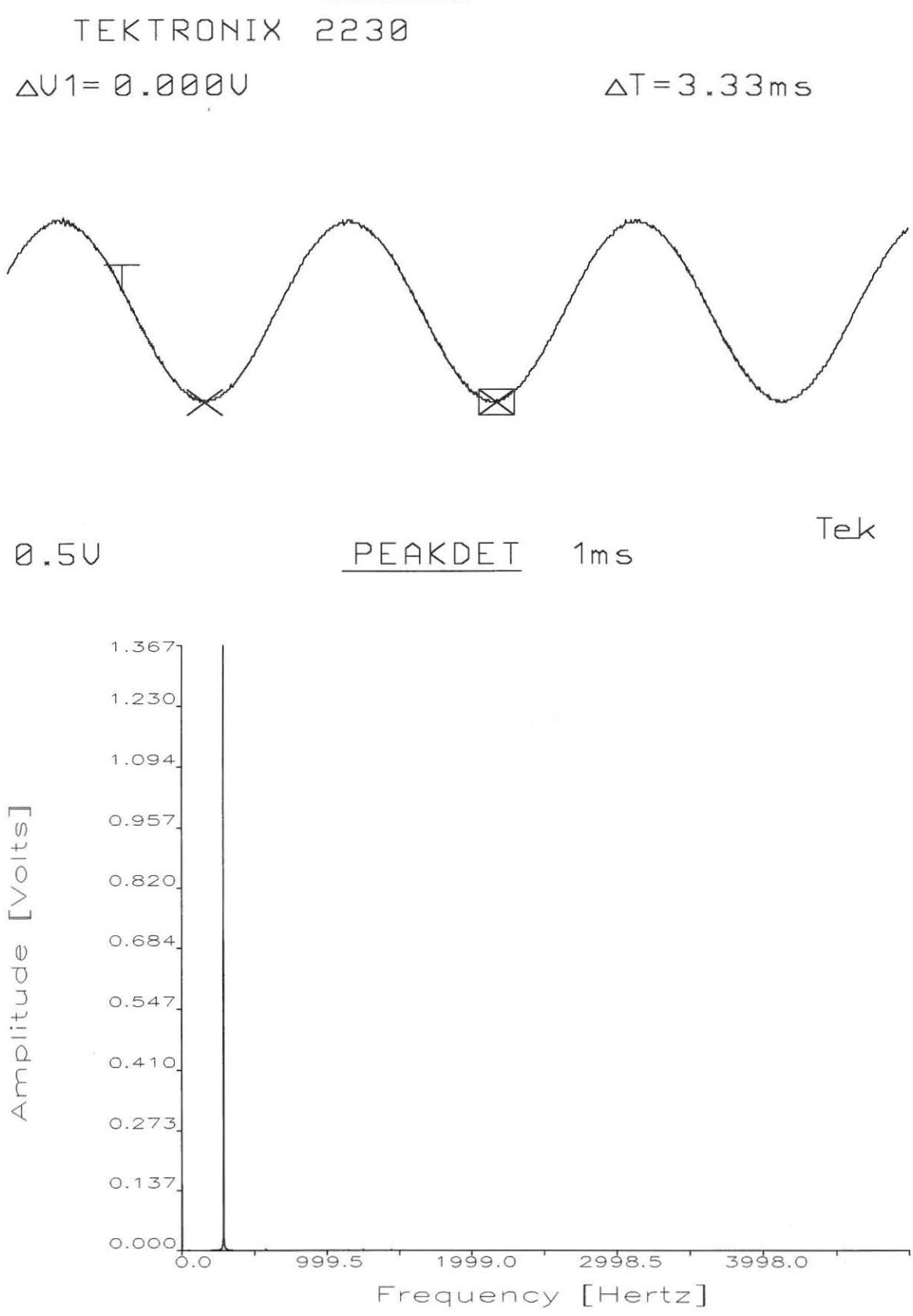


Title	COMPUTER INTERFACE CARD	
Size A	Document Number CIRCUIT DIAGRAM 17	REV 1
Date: May 28, 1994	Sheet 17 of 17	

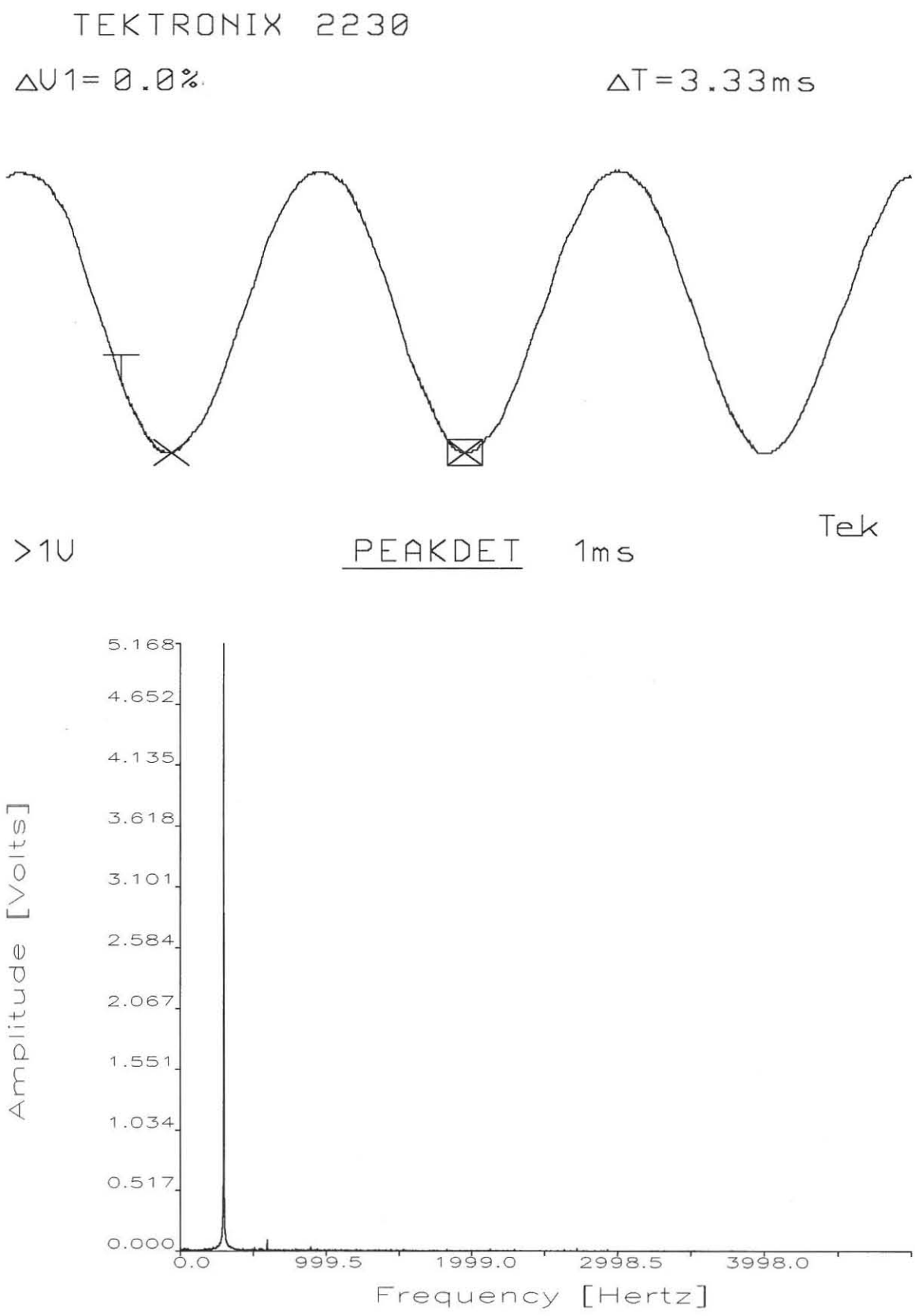
Central University of Technology, Free State

APPENDIX B

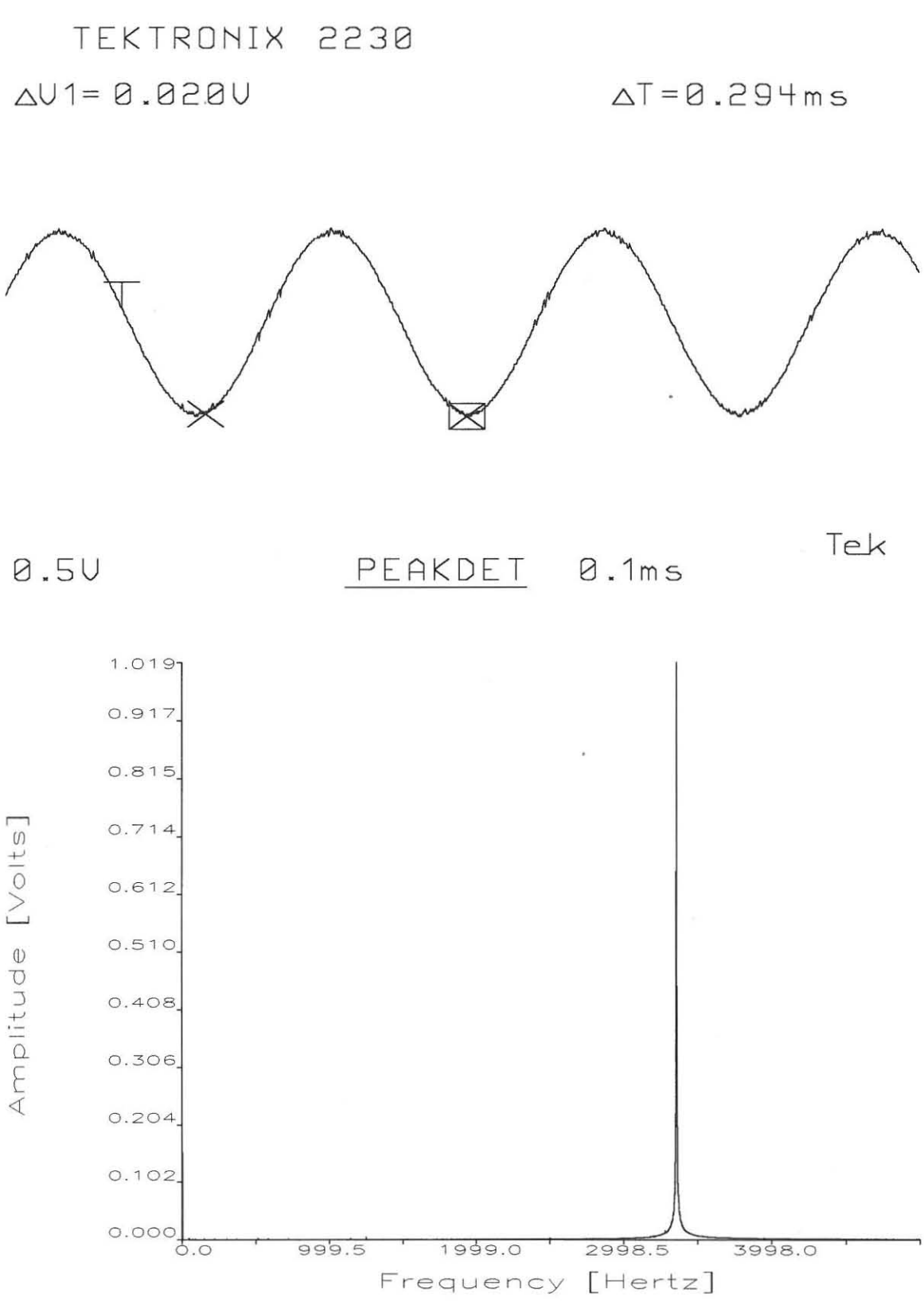
MEASUREMENT RESULTS



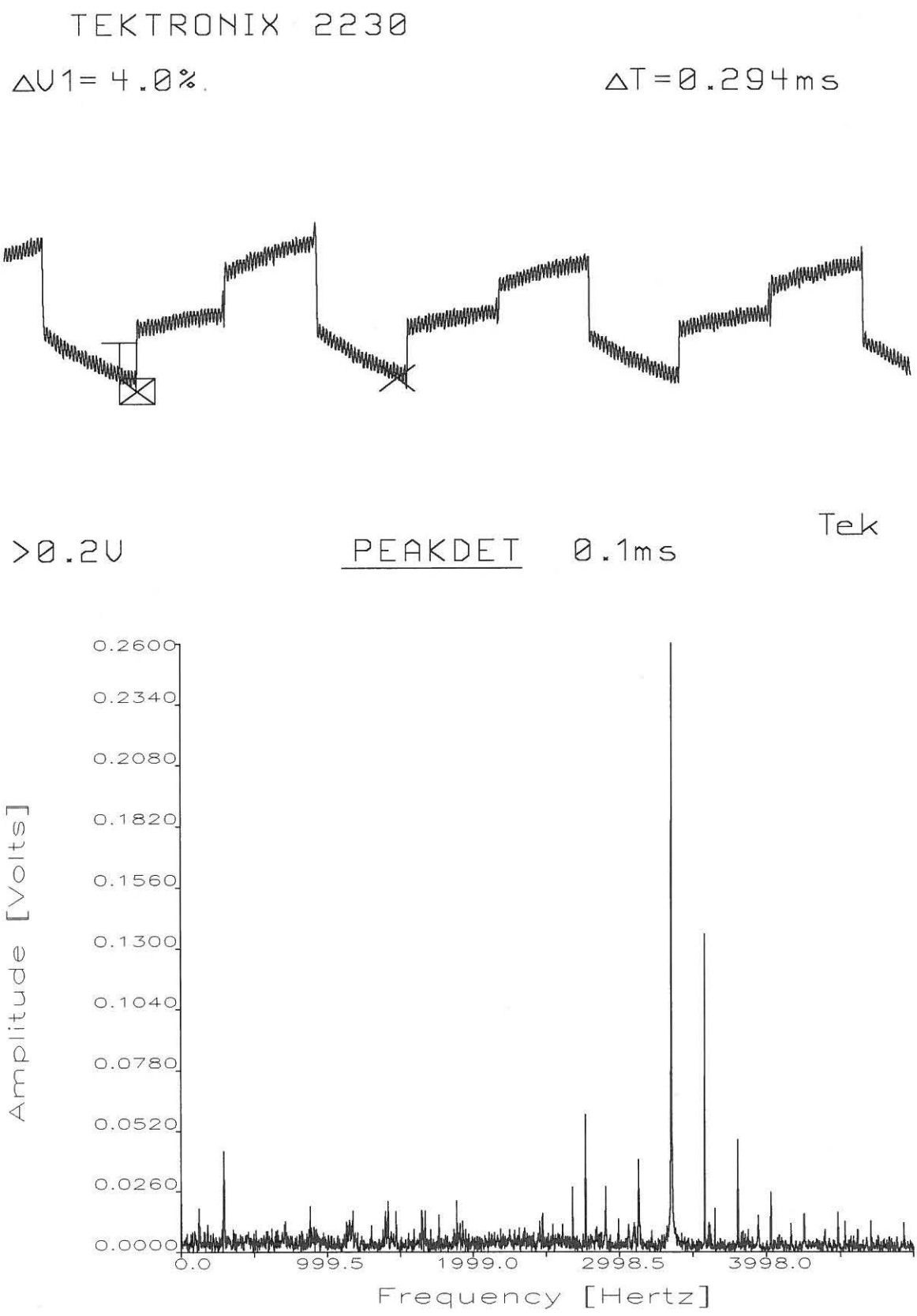
300Hz reference input signal.



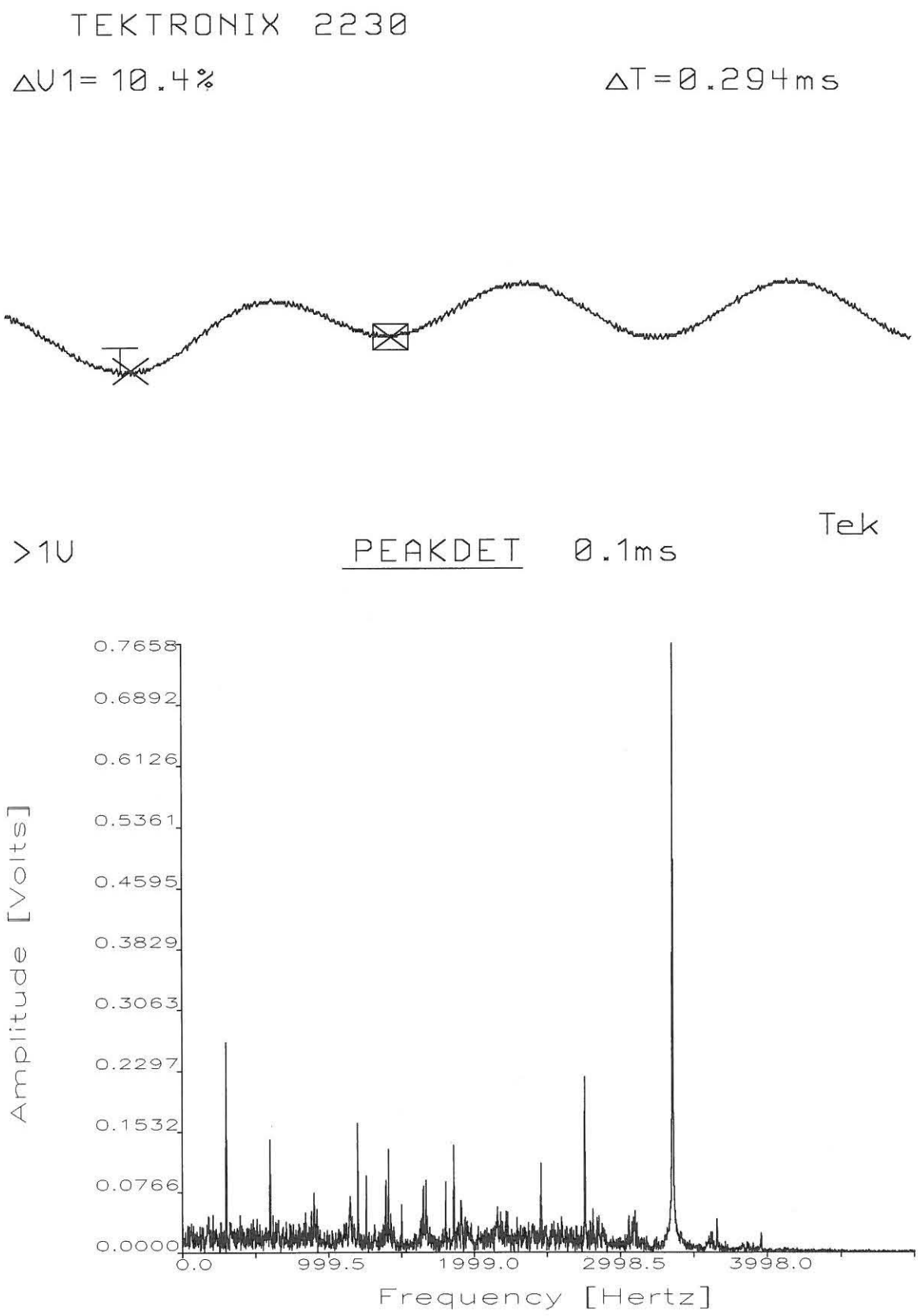
300Hz signal at output of system.



3400Hz reference input signal



3400Hz signal recovered from the DAC.



3400Hz signal at output of system.

APPENDIX C

SOFTWARE LISTINGS

Unit Variable;

```
(*************************************************************************)
Interface
(*************************************************************************)

Uses gKeybrd, DOS, Graph;

const
   ScreenX      = 20;   ScreenY      =  15;
   ScreenSizeX  = 600;  ScreenSizeY  =  216;

   ButtonSizeX  =  79;  ButtonSizeY  =  69;

   SampleX      =  20;  SampleY      = 250;
   TalkX        =  20;  TalkY        = 320;
   MonitorX     =  20;  MonitorY     = 390;
   CopyX        = 230;  CopyY        = 250;
   MoveX        = 230;  MoveY        = 320;
   DeleteX      = 230;  DeleteY      = 390;
   SaveX        = 310;  SaveY        = 250;
   LoadX        = 310;  LoadY        = 320;
   DirectoryX   = 310;  DirectoryY   = 390;
   WriteEX      = 390;  WriteEY      = 250;
   ReadEX       = 390;  ReadEY       = 320;
   listEX       = 390;  ListEY       = 390;
   HelpX        = 540;  HelpY        = 250;
   EpromSizeX   = 540;  EpromSizeY   = 320;
   QuitX        = 540;  QuitY        = 390;

   MessageBoxSizeX = 300;  MessageBoxSizeY = 100;

   PotSizeX     =  36;  PotSizeY     = 153;
   SliderSizeX  =  36;  SliderSizeY  = 35;

   TriggerX     = 115;  TriggerY     = 260;
   SensitivityX = 175;  SensitivityY = 260;
   VolumeX      = 487;  VolumeY      = 260;

   AdcPort = $300;
   DacPort = $304;
   IntCtlrAddr = $21; {Port for 8259 Interrupt controller.}
   VolPort = $309;
   InterruptNo = 5;

   ProposedRecordBufferSize = 60000;

   DirCol = 5;
   DirRow = 10;

   PotColor = Black;

   ButtonBorder      = 5;
   Backgroundborder  = 3;
   ScreenBorder      = 8;

   cBackground       = LightGray;
   cBackgroundShade1 = DarkGray;
   cBackgroundShade2 = White;
   cBackgroundShade3 = Black;

   cScreenShade1     = DarkGray;
   cScreenShade2     = White;
   cScreenShade3     = LightGray;
   cScreenBackGround = Black;
   cScreenForeGround = Green;

   cButton           = LightGray;
   cButtonShade1     = DarkGray;
   cButtonShade2     = White;
   cButtonShade3     = Black;
   cButtonLimits     = Red;
   cPots             = Black;

   cRecord           = Red;
```

```
    cPlay           = Blue;
    cMute           = Black;

    cTrigger        = Black;
    cSensitivity    = Black;

    cCopy           = Magenta;
    cMove           = Magenta;
    cDelete         = Magenta;
    cSave           = Blue;
    cLoad           = Blue;
    cPath           = Blue;
    cReadE          = Red;
    cWriteE         = Red;
    cListE          = Red;

    cVolume         = Black;

    cHelp           = Black;
    c256K           = Brown;
    cQuit           = Red;

    cLimits         = LightRed;

    cIntroText      = Green;

    cEditWindowDisplay1  = Brown;
    cEditWindowDisplay2  = Red;
    cEditWindowDisplay3  = Cyan;
    cEditWindowDisplay4  = Yellow;
    cEditWindowDisplay5  = Green;
    cEditWindowDisplay6  = Blue;
    cEditWindowDisplay7  = Magenta;
    cEditWindowDisplay8  = DarkGray;
    cEditWindowDisplay9  = White;

    cEditWindowDisplay12 = LightRed;
    cEditWindowDisplay13 = LightCyan;
    cEditWindowDisplay14 = Yellow;
    cEditWindowDisplay15 = LightGreen;
    cEditWindowDisplay16 = LightBlue;
    cEditWindowDisplay17 = LightMagenta;
    cEditWindowDisplay18 = Lightgray;
    cEditWindowDisplay19 = White;

    FileExtension    = '.TLK';
    FileListFilter   = '*.TLK';

    Pitch = -48;

type
  TalkStatType = (Rec, Play, Idle);

  ByteArray_ = Array[1..ProposedRecordBufferSize] OF BYTE; {RecordBuffer}
  ByteArrayPtr_ = ^ByteArray_;

  ByteArray10 = ARRAY[0..0] OF BYTE;  {Slider}
  ByteArrayPtr10 =^ByteArray10;

  ByteArrayWholeScreen    = ARRAY[0..0] OF BYTE; {GetScreen & PutScreen (help)}
  ByteArrayWholeScreenPtr = ^ByteArrayWholeScreen;

  ByteArrayScreenPart     = ARRAY[0..0] OF BYTE; {GetScreenPart & PutScreenPart}
  ByteArrayScreenPartPtr  = ^ByteArrayScreenPart;

  ByteArrayLimits = ARRAY[0..0] OF BYTE; {Limit1 & Limit2}
  ByteArrayPtrLimits =^ByteArrayLimits;

  ByteArray6 = ARRAY[0..0] OF BYTE; {GetScreen & PutScreen (escape)}
  ByteArrayPtr6 =^ByteArray6;

  DecisionType = (Wait,Place,Escape,Continue_KeyBrd,Continue_Mouse);

  zet = Record
      x,y,xx,yy : Integer;
      END;

  MonitorType = (On,Off);
```

```
var
  DemoMode,
  xDemoMode,
  EpromEmpty,
  fk,
  TalkCardPresent,
  FileToBeSaved      : Boolean;

  LastRecByte,
  EpromError,
  IntEnableNo,
  IntDisablNo,
  OffsetIntNo        : Byte;

  aaa,
  bbb,
  k,
  ax,
  aa,
  bb,
  d1,
  d2,
  d3,
  d4,
  xx,
  Size               : Word;

  _trigger,
  trigger,
  sensitivity,
  volume,
  SensitivitySlider,
  SL1,
  SL2,
  ButtonStatus,
  IoError,
  TalkSpeed,
  b,
  c,
  s,
  l,
  m,
  n,
  o,
  p,
  GraphDriver,
  GraphMode,
  ErrorCode,
  ScreenMarker,
  RecVol,PlayVol     : Integer;

  zxc,
  RecordBufferSize,
  f,
  g,
  EpromCount,
  EpromSize,
  dPointer,
  dPointerStop,
  Limit1,
  Limit2             : LongInt;

  y                  : real;

  ch                 : char;

  Path,
  ErrorMessage       : String[25];

  OldVector,
  imagePotButton,
  imagePot,
  imageLimits,
  imageMessageBox,
  imageScreenpart,
  imageWholeScreen,
  image6,
```

```
  Heap,
  Pp              : Pointer;

  Monitor         : MonitorType;

  TalkStat        : TalkStatType;

  GlobalKey       : KeyType;

  DosReg          : Registers;

  d               : ByteArrayPtr_;

  DirArray        : Array[1..DirCol, 1..DirRow] OF String[12];

  Decision        : DecisionType;

(*****************************************************************************)
Implementation
(*****************************************************************************)

BEGIN
   DemoMode  := True;
   xDemoMode := True;

   Trigger    := 0;
   Sensitivity := 100;
   Volume     := 64;

   Path := '';

   LeftMouseKeyWasPressed  := FALSE;
   RightMouseKeyWasPressed := FALSE;

   dPointer := 1;
   TalkStat := Idle;

   IntEnableNo   := $FF - Trunc(Exp(InterruptNo * Ln(2)));
   IntDisablNo   := Trunc(Exp(InterruptNo * Ln(2)));
   OffsetIntNo   := InterruptNo + 8;

   RecVol := 0;
   PlayVol := 16;

   FOR g := 1 to DirRow DO BEGIN
      FOR f := 1 TO DirCol DO BEGIN
         DirArray[f,g] := '...........';
      END;
   END;
END.
```

Program Talk;

```
uses
   MousU, Dos,graph,crt,Variable, Video, BinU, gKeyBrd, Disp, Disk, Eprom, iReq, Helper;

Label Retry;
Label EndlessLoop;

Function MouseTouchArea (d1, d2, d3, d4 : Integer) : Boolean;
      BEGIN
         IF (d1 < MouseX) AND (MouseX < d1 + d3) AND
           (d2 < MouseY) AND (MouseY < d2 + d4)
         THEN MouseTouchArea := True
         ELSE MouseTouchArea := False;
      END;

procedure DetectTalkHardware;
      Var  d1,d2,d3,h,m,s,hun : Word;
      BEGIN
         WriteLn('Looking for Talk hardware');
         InstallAdcInt;
         Monitor := Off;
         dPointer := 1;
         TalkStat := Rec;
         Delay(10);
         TalkStat := Idle;
         UnInstallAdcInt;
         WriteLn('Pointer value = ',dPointer);

         IF (dPointer >= 130) AND (dPointer <= 140) THEN BEGIN
           WriteLn('Talk hardware detected OK');
           DemoMode := FALSE;
           xDemoMode := TRUE;
         END
         ELSE BEGIN
            WriteLn('No Talk hardware detected - switching to demo mode');
            DemoMode := TRUE;
            xDemoMode := TRUE;
         END;
      END;

procedure Beep;
      BEGIN
         Sound(1000);
         Delay(100);
         NoSound;
      END;


procedure SetpotTo(d1 : Integer);
      BEGIN
         IF TalkCardPresent THEN
         BEGIN
            IF Port[VolPort] < d1 THEN REPEAT Port[VolPort] := Port[VolPort] + 1 UNTIL Port[VolPort] = d1;
            IF Port[VolPort] > d1 THEN REPEAT Port[VolPort] := Port[VolPort] - 1 UNTIL Port[VolPort] = d1;
         END;
      END;

Function Chance (Percentage : Real) : Boolean;
      BEGIN
         IF Random(10000) <= Percentage * 100 THEN Chance := TRUE
         ELSE Chance := FALSE;
      END;

Procedure _Record;
      var x, dd : integer;
         mk : Boolean;
         yy ,yyy : Real;
      BEGIN
         LastRecByte := 127;
         d2 := 1;
         dd := 1;
         d3 := 0;
         mk := FALSE;
         ax := 0;
         b  := 0;
```

```
Button (SampleX, SampleY, ButtonSizeX, ButtonSizeY, 'RECORD', cRecord, on);

Port[VolPort] := (((Volume div 20) * 8) + (Sensitivity div 20));

KeepMouseOffEditWindow;

IF FileToBeSaved THEN BEGIN
  MessageBox('WARNING',
         '',
         'edits will be lost',
         '',
         'left = yes / right = no');
  IF LeftMouseKeyWasPressed THEN FileToBeSaved := FALSE;
END;

IF NOT FileToBeSaved THEN BEGIN

  ClearScreen(0);
  FillChar(d^, SizeOf(d^), 127);
  SL1 := ScreenX + 1;
  SL2 := ScreenX + ScreenSizeX - 1;
  Limit1 := SL1 - ScreenX;
  Limit2 := SL2 - ScreenX;

  x := ScreenY + ((ScreenSizeY - 64) div 2);
  Moveto(ScreenX, x);

  IF NOT DemoMode THEN BEGIN
    WriteBottomScreen(' Waiting for input signal...', cEditWindowDisplay2 + blink);
    REPEAT
        IF RightMouseKeyPressed THEN mk:=TRUE;
    UNTIL (ABS(port[ADCport] - 127) > Trigger) OR (mk) {OR (DemoMode)};
  END;

     IF NOT mk THEN BEGIN


      IF DemoMode THEN BEGIN
        Randomize;
        y := Random(127);

WriteBottomScreen('████████████████████████████████████████████████████████████
███████', 0);
        WriteBottomScreen('NOTE: This is a simulation of a signal being recorded.', cEditWindowDisplay2);
      END
      ELSE BEGIN

WriteBottomScreen('████████████████████████████████████████████████████████████
███████', 0);
         WriteBottomScreen(' RECORDING... ', cEditWindowDisplay2 + blink);
      END;

      dPointer := 1;
      TalkStat := Rec;
      RightMouseKeyWasPressed := FALSE;

      REPEAT
          IF NOT DemoMode THEN PutPixel(dPointer div 100 + ScreenX,
            (LastRecByte div 2 - 64 + (ScreenY + ScreenSizeY div 2)), cEditWindowDisplay5);

          IF DemoMode THEN BEGIN
            {slope}
            IF Chance({10}10) THEN
              IF yy < 0 THEN yy := Random(200) / 1000 * (-1)
              ELSE yy := Random(200) / 1000;

            {direction}

            IF Chance(1) THEN
              {IF Chance(10) THEN} yy := yy * (-1);

            y := y + yy;
            IF y <= 0 THEN y := 0;
            IF y >= 127 THEN y := 127;

            LastRecByte := Random(Round(y));
```

```
        LastRecByte := 127 -
        IF Chance(50) THEN LastRecByte := 255 - LastRecByte;
        IF (dPointer div 5) * 5 = dPointer THEN
          PutPixel(dPointer div 100 + ScreenX, (LastRecByte div 2{4}-64 +
                (ScreenY + ScreenSizeY div 2)), cEditWindowDisplay5);

        d^[dPointer] := LastRecByte;

        Inc(dPointer);

      END;

      IF RightMouseKeyPressed THEN RightMouseKeyWasPressed := TRUE;

    UNTIL (dPointer >= RecordBufferSize) OR (TalkStat <> Rec) OR (RightMouseKeyWasPressed) ;
    IF RightMouseKeyWasPressed THEN BEGIN
      SetColor(cEditWindowDisplay5);
      Line(dPointer div 100 + ScreenX, ScreenY + ScreenSizeY div 2, RecordBufferSize div 100 + ScreenX,
         ScreenY + ScreenSizeY div 2);

      SL2 := ScreenX + dPointer div 100;
      Limit2 := SL2 - ScreenX;
    END;
   END;
  SetViewPort(0, 0, GetMaxX, GetMaxY, ClipOn);

WriteBottomScreen('■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■■■■■', 0);
  LimitsOnOff;
 END;
 FreeMouse;
 Button (SampleX    ,SampleY    ,ButtonSizeX,ButtonSizeY,'RECORD',cRecord,off);
END;

Procedure _Play;
var d1 : Byte;

BEGIN
  Button (TalkX, TalkY, ButtonSizeX, ButtonSizeY, 'PLAY', cPlay, on);
  WriteBottomScreen('Playing signal...', cEditWindowDisplay4);
  IF NOT DemoMode THEN BEGIN

    Port[VolPort] := (((Volume div 20) * 8) + (Sensitivity div 20));

    dPointer := (Limit1) * 100;
    dPointerStop := (Limit2 {+ 2}) * 100;

    TalkStat := Play;

    ScreenMarker := (Limit1 + ScreenX + 1);
    ScreenMarkerOnOff;
    REPEAT
        DrawScreenMarker(dPointer div 100 + ScreenX);
    UNTIL {(dPointer >= (Limit2 + 2) * 100) OR}(TalkStat <> Play) OR (RightMouseKeyPressed);
    ScreenMarkerOnOff;

    IF TalkStat <> Idle THEN TalkStat := Idle;
  END
  ELSE BEGIN
     dPointer := (Limit1) * 100;

     ScreenMarker := (Limit1 + ScreenX + 1);
     ScreenMarkerOnOff;
     REPEAT
         DrawScreenMarker(dPointer div 100 + ScreenX);
         Inc(dPointer, 300);
     UNTIL (dPointer >= (Limit2 + 2) * 100) OR (RightmousekeyPressed);
     ScreenMarkerOnOff;
  END;
  LimitsOnOff;

WriteBottomScreen('■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■■■■■', 0);
 LimitsOnOff;
  Button (TalkX, TalkY, ButtonSizeX, ButtonSizeY, 'PLAY', cPlay, off);
END;

Procedure _Mute;
```

```
    BEGIN
      IF monitor = On  THEN Button (MonitorX, MonitorY, ButtonSizeX, ButtonSizeY, 'LISTEN', Blue    , on);
      IF monitor = Off THEN Button (MonitorX, MonitorY, ButtonSizeX, ButtonSizeY, '-MUTE-', DarkGray, on);
      REPEAT UNTIL NOT(AnyMouseKeyPressed);
      IF monitor = On  THEN Monitor := Off ELSE Monitor := On;
      IF monitor = On  THEN Button (MonitorX, MonitorY, ButtonSizeX, ButtonSizeY, 'LISTEN', Blue    , off);
      IF monitor = Off THEN Button (MonitorX, MonitorY, ButtonSizeX, ButtonSizeY, '-MUTE-', DarkGray, off);
    END;

Procedure _Path;
    Var d1 : String;
      d2 : Integer;
      DirInfo: SearchRec;
      WhatToLookFor : String;
    Label Retry;
    BEGIN
      Button (DirectoryX, DirectoryY, ButtonSizeX, ButtonSizeY, 'PATH', cPath, on);

      Retry:

      WriteBottomScreen(' Path: ', cEditWindowDisplay4);
      d1 := Path;
      InputString(d1, ScreenX + 70, ScreenY + ScreenSizeY - 17, 30, 4, 0,
              [EscapeKey, CarriageReturn, RightMouseKey, LeftMouseKey]);

      IF (Key = CarriageReturn) OR (Key = LeftMouseKey) THEN BEGIN
        {$i-};
            ChDir(d1);
            IoError:=IoResult;
        {$i+};
        IF IoError <> 0 THEN BEGIN
          Error;
          Goto Retry;
        END
        ELSE Path := d1;
      END;
      LimitsOnOff;

WriteBottomScreen('■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■■■■■', 0);
LimitsOnOff;
      Button (DirectoryX, DirectoryY, ButtonSizeX, ButtonSizeY, 'PATH', cPath, off);
    END;

Procedure _Copy;
    var NewSL1,x,d3:Integer;
      Key       : KeyType;
    BEGIN
      Button (CopyX, CopyY, ButtonSizeX, ButtonSizeY, 'COPY', cCopy, on);
      NewSL1 := SL1 {+ 1};
      Decision := Wait;

      HideMouseCursor;

      GetWholeScreen;
      GetScreenPart(SL1 + 1, ScreenY + ScreenSizeY div 2 - 64, SL2 - SL1 - 2, 128{ScreenSizeY div 2 + 64});
      PutScreenPart(SL1 + 1, ScreenY + ScreenSizeY div 2 - 64, NotPut);

      ShowMouseCursor;
      IF Ch='c' THEN MoveMouseTo(SL1 + (SL2 - SL1) div 2, ScreenY + (ScreenY + ScreenSizeY) div 2);
      REPEAT
          IF (RightMouseKeyPressed) THEN Decision := Escape;
          IF (LeftMouseKeyPressed) AND (MouseTouchArea (SL1, ScreenY, SL2 - SL1, ScreenSizeY))
          THEN Decision := Continue_Mouse;
      UNTIL Decision <> Wait;
      IF Decision <> Escape THEN BEGIN
        REPEAT
            IF NOT LeftMouseKeyPressed THEN Decision := Place;
            x := MouseX - (SL2 - SL1) DIV 2;
            IF x <= ScreenX THEN x := ScreenX;
            IF x + (SL2 - SL1) >= ScreenX + ScreenSizeX
            THEN x := NewSL1;
            IF x <> NewSL1 THEN BEGIN
              HideMouseCursor;
              PutScreenPart(NewSL1, ScreenY + ScreenSizeY div 2 - 64, XorPut);
              PutScreenPart( x    , ScreenY + ScreenSizeY div 2 - 64, XorPut);
              ShowMouseCursor;
              NewSL1 := x;
```

```
            END;
          UNTIL (Decision = Place) OR (Decision = Escape);
        END {if decision <> escape}
        ELSE BEGIN
          PutWholeScreen;
          ShowMouseCursor;
          Button (CopyX, CopyY, ButtonSizeX, ButtonSizeY, 'COPY', cCopy, off);
          Exit;
        END;
        IF (Decision <> Escape) THEN BEGIN
          FileToBeSaved := TRUE;
          xx := 0;
          d1 := (Limit1 + 1) * 100;
          d2 := (Limit2 - 1) * 100;
          d3 := (NewSL1 - ScreenX) * 100;
          d4 := ((NewSL1 - ScreenX) + (SL2 - SL1 - 2)) * 100;
          IF d1 = 0 THEN d1 := 1;
          IF d3 <= d1 THEN BEGIN
            xx := 0;
            REPEAT
              d^[d3 + xx] := {d^[d3 + xx] +} d^[d1 + xx] {- 128};
              xx := xx + 1;
            UNTIL (xx + d1) >= (d2);
          END{if d3 <= 0}
          ELSE BEGIN
            REPEAT
              d^[d4 - xx] := d^[d2 - xx];
              INC(xx);
            UNTIL d1 > (d2 - xx);
          END;
        END;
        HideMouseCursor;
        LimitsOnOff;
        PutScreenPart(SL1 + 1, ScreenY + ScreenSizeY div 2 - 64, NormalPut);

        SetViewPort(NewSL1, ScreenY + ScreenSizeY div 2 - 64, NewSl1 + SL2 - SL1, 128                , ClipOn);
        ClearViewPort;
        SetViewPort(0, 0, GetMaxX, GetMaxY, ClipOn);

        PutScreenPart(NewSL1, ScreenY + ScreenSizeY div 2 - 64, OrPut);
        LimitsOnOff;
        ShowMouseCursor;
        REPEAT UNTIL NOT AnyMouseKeyPressed;
        Button (CopyX, CopyY, ButtonSizeX, ButtonSizeY, 'COPY', cCopy, off);

      END;

Procedure _Move;
      var NewSL1,x,xl,d3:Integer;
        Key        : KeyType;
      BEGIN
        Button (MoveX, MoveY, ButtonSizeX, ButtonSizeY, 'MOVE', cMove, on);
        Decision := Wait;
        HideMouseCursor;

        GetWholeScreen;
        GetScreenPart(SL1 + 1, ScreenY + ScreenSizeY div 2 - 64, SL2 - SL1 - 2, 128);
        PutScreenPart(SL1 + 1, ScreenY + ScreenSizeY div 2 - 64, NotPut);

        ShowMouseCursor;
        IF Ch='m' THEN MoveMouseTo(SL1 + (SL2 - SL1) div 2, ScreenY + (ScreenY + ScreenSizeY) div 2);

        REPEAT
          IF (RightMouseKeyPressed) THEN Decision := Escape;
          IF (LeftMouseKeyPressed) AND (MouseTouchArea(SL1,ScreenY,SL2-SL1,ScreenSizeY))
          THEN Decision := Continue_Mouse;
        UNTIL Decision <> Wait;
        IF Decision <> Escape THEN BEGIN
          HideMouseCursor;
          PutScreenPart(SL1 + 1, ScreenY + ScreenSizeY div 2 - 64, OrPut);
          ShowMouseCursor;
          REPEAT
            IF NOT LeftMouseKeyPressed THEN Decision := Place;
            x := MouseX - (SL2 - SL1) div 2;
            IF x <= ScreenX THEN x := ScreenX;
            IF x + (SL2 - SL1) >= ScreenX + ScreenSizeX
            THEN  x := NewSL1;
            IF x <> NewSL1 THEN BEGIN
```

```
            HideMouseCursor;
            PutScreenPart(NewSL1, ScreenY + ScreenSizeY div 2 - 64, XorPut);
            PutScreenPart( x  , ScreenY + ScreenSizeY div 2 - 64, XorPut);
            ShowMouseCursor;
            NewSL1 := x;
          END;{if x <> NewSL1}
        UNTIL (Decision = Place) OR (Decision = Escape);
      END {if decision <> Escape}
      ELSE BEGIN
        PutWholeScreen;
        ShowMouseCursor;
        Button (MoveX, MoveY, ButtonSizeX, ButtonSizeY, 'MOVE', cMove, off);
        Exit;
      END;

      IF Decision <> Escape THEN BEGIN
        FileToBeSaved := TRUE;
        BEGIN
          xx := 0;
          d1 := Limit1 * 100;
          d2 := Limit2 * 100;
          d3 := (NewSL1 - ScreenX) * 100;
          d4 := ((NewSL1 - ScreenX) + (SL2 - SL1)) * 100;
          IF d1 = 0 THEN d1 := 1;
          IF d3 <= d1 THEN BEGIN
            xx := 0;
            REPEAT
              d^[d3 + xx] := d^[d1 + xx];
              d^[d1 + xx] := 127;
              xx := xx + 1;
            UNTIL (xx + d1) >= d2;
          END {if d3 <= d1}
          ELSE BEGIN
            REPEAT
              d^[d4 - xx] := d^[d2 - xx];
              d^[d2 - xx] := 127;
              INC(xx);
            UNTIL d1 > (d2 - xx);
          END;
        END;
        LimitsOnOff;
        HideMouseCursor;
        SetViewPort(SL1 + 1, ScreenY + ScreenSizeY div 2 - 64, SL2 - 1, ScreenY + ScreenSizeY div 2 + 64, ClipOn);
        ClearViewPort;
        SetViewPort(0, 0, GetMaxX, GetMaxY, ClipOn);
        x := ScreenY + ScreenSizeY div 2;
        SetColor(cEditWindowDisplay5);
        Line(SL1, x, SL2, x);
        PutScreenPart(NewSL1 + 1, ScreenY + ScreenSizeY div 2 - 64, NormalPut);
        LimitsOnOff;
        ShowMouseCursor;
      END {if decision <> Escape}
      ELSE BEGIN
        PutWholeScreen;
        ShowMouseCursor;
      END;
      REPEAT UNTIL NOT AnyMouseKeyPressed;
      Button (MoveX, MoveY, ButtonSizeX, ButtonSizeY, 'MOVE', cMove, off);
    END;

Procedure _Delete;
    var x:Integer;
    BEGIN
      Button (DeleteX, DeleteY, ButtonSizeX, ButtonSizeY, 'DELETE', cDelete, on);

      GetScreenPart(SL1, ScreenY + ScreenSizeY div 2 - 64, SL2 - SL1, 128);
      PutScreenPart(SL1, ScreenY + ScreenSizeY div 2 - 64, NotPut);
      MessageBox('DELETE',",'are you sure?',", 'left = yes / right = no');
      IF LeftMouseKeyWasPressed THEN BEGIN
        FileToBeSaved := TRUE;
        LeftMouseKeyWasPressed:= FALSE;
        LimitsOnOff;
        HideMouseCursor;
        SetViewPort(SL1, ScreenY + ScreenSizeY div 2 - 64, SL2, ScreenY + ScreenSizeY div 2 + 64, ClipOn);
        ClearViewPort;
        SetViewPort(0, 0, GetMaxX, GetMaxY, ClipOn);
        x := ScreenY + ScreenSizeY DIV 2;
        SetColor(cEditwindowDisplay5);
```

```
        Line(SL1, x, SL2, x);
        ShowMouseCursor;
        xx := 0;
        d1 := Limit1 * 100;
        d2 := Limit2 * 100;
        BEGIN
          REPEAT
            d^[d1 + xx] := 127;
            INC(xx);
          UNTIL (xx + d1) >= d2;
        END;
        LimitsOnOff;
      END
      ELSE PutScreenPart(SL1, ScreenY + ScreenSizeY div 2 - 64, NormalPut);
      REPEAT UNTIL NOT AnyMouseKeyPressed;
      Button (DeleteX, DeleteY, ButtonSizeX, ButtonSizeY, 'DELETE', cDelete, off);
    END;

Procedure _WriteE;
    Const  yOffset    = 37;
           yScaling   = 5;
           yTextoffset = 78;
    Var dv1, dv2, dv3 : LongInt;
      Code    : Integer;
      dv0     : LongInt;
      IDstr   : String;
      IDno    : Word;
      EpromOK : Boolean;
      EpromCountMarker : LongInt;
    label exit;

    BEGIN
      Button (WriteEX, WriteEY, ButtonSizeX, ButtonSizeY, 'WriteE', cWriteE, on);

      SetMouseCursorPos(WriteEX + ButtonSizeX div 2, WriteEY + ButtonSizeY div 2);
      KeepMouseOffEditWindow;

      GetWholeScreen;

      EpromCountMarker := 0;
      EpromOK := FALSE;
      EpromError := 0;
      SetTextJustify(LeftText, BottomText);
      SetViewPort(ScreenX, ScreenY, ScreenX +
            ScreenSizeX, ScreenY + ScreenSizeY, ClipOn);
      ClearViewPort;
      SetViewPort(0, 0, GetMaxX, GetMaxY, ClipOn);
      WriteBottomScreen('Processing...', cEditWindowDisplay2);


      SetColor(cEditWindowDisplay6);
      SetTextStyle(SmallFont, HorizDir, 4);
      Line(ScreenX + ScreenSizeX div 2, ScreenY + 1, ScreenX + ScreenSizeX div 2, ScreenY + 86);
      SetLineStyle(DottedLn, 0, NormWidth);
      Line(ScreenX + 1, ScreenY + 86, ScreenX + ScreenSizeX - 1, ScreenY + 86);
      SetLineStyle(SolidLn, 0, NormWidth);
      SetTextJustify(LeftText, TopText);
      OutTextXY(ScreenX + 1, 106, '0K');
      SetTextJustify(CenterText, TopText);
      OutTextXY(ScreenX + ScreenSizeX div 2, 106, '128K');
      SetTextJustify(RightText, TopText);
      OutTextXY(ScreenX + ScreenSizeX - 1, 106, '256K');
      SetTextStyle(DefaultFont, HorizDir, 1);

      FOR dv0 := (Limit1 * 25) TO (Limit2 * 25) DO BEGIN
        PutPixel((dv0 - Limit1 * 25) div 25 + ScreenX, (d^[dv0*4] div 4) + ScreenY + 115,
              cEditWindowDisplay5);
        IF RightMouseKeyPressed THEN GOTO Exit;
      END; {for}

      IF NOT DemoMode THEN BEGIN
        EpromOK := TRUE;
      END
      ELSE BEGIN
        EpromOK := FALSE;
        MessageBox('NOTE',
              '',
              'WriteE not available in demo mode',
```

```
                '',
                'any mouse key = exit');
       Goto Exit;
    END;

    ResetEprom;
    IF ReadEprom <> 0 THEN BEGIN
      EpromVppOn;
      WriteEprom(0);
      IF EpromError = 0 THEN EpromOK := TRUE;
      IF EpromError = 1 THEN BEGIN
        EpromVppOff;
        EpromOK := FALSE;
        MessageBox('ERROR!',
                'EPROM not present',
                'or EPROM faulty.',
                '',
                'Any mouse key = continue');
      END;
    END;

    IF EpromOK THEN BEGIN
      SetColor(cEditWindowDisplay2);

WriteBottomScreen('■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■■■■■', 0);
WriteBottomScreen(' Please wait while looking for EPROM space...', cEditWindowDisplay2);
      SetTextJustify(CenterText, CenterText);
      SetTextStyle(SmallFont, VertDir, 4);
      SetColor(cEditWindowDisplay4);
      REPEAT
          dv1 := ReadEprom;
          IF dv1 = 0 THEN BEGIN
            ClockEprom;
            dv1 := ReadEprom;
            IF dv1 <> 255 THEN BEGIN
              str(dv1 : 3, IDstr);
              OutTextXY(EpromCount div 430 + ScreenX + 5, ScreenY + yTextOffset, IDstr);
            END;
            {IF dv1 <> 255 THEN ClockEprom;}
            IF dv1 <> 255 THEN FOR f := 1 TO 31 DO ClockEPROM;
          END
          ELSE IF dv1 <> 255 THEN FOR f := 1 TO 32 DO ClockEPROM;
            PutPixel(EpromCount div 430 + ScreenX, (dv1 div yScaling + ScreenY) +
                  yOffset - (256 div yScaling div 2), cEditWindowDisplay5);
          IF RightMouseKeyPressed THEN GOTO EXIT;
      UNTIL (dv1 = 255) OR (EpromCount > EpromSize);
      SetTextJustify(LeftText, BottomText);
      SetTextStyle(DefaultFont, HorizDir, 1);
      SetColor(cEditWindowDisplay4);

      IF (Limit2 * 100 - Limit1 * 100) > (EpromSize - EpromCount) THEN BEGIN
        MessageBox('ERROR!',
                'Not enough space',
                'left on EPROM to',
                'hold the message.',
                'Any mouse key = continue');
      END
      ELSE BEGIN
         {ClearBottomScreen;}

WriteBottomScreen('■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■■■■■', 0);
WriteBottomScreen(' Space found. Message ID no?', cEditWindowDisplay2);

         IDstr := '';
         REPEAT
             InputString(IDstr, ScreenX + 260, ScreenY + ScreenSizeY - 17, 2, 3, 0,
                   [EscapeKey, CarriageReturn, LeftMouseKey, RightMouseKey]);
             IF RightMousekeyPressed THEN GOTO Exit;
             VAL(IDstr, IDno, code);
         UNTIL ((Code = 0) AND (IDno > 0) AND (IDno < 16)) OR (Key = EscapeKey) OR (Key = RightMouseKey);;


         IF Key = CarriageReturn THEN BEGIN
           {ClearBottomScreen;}
```

```
WriteBottomScreen('■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■■■■■■', 0);
WriteBottomScreen(' Programming message into EPROM ', cEditWindowDisplay2 + blink);

            SetTextJustify(LeftText, CenterText);
            SetTextStyle(SmallFont, VertDir, 4);
            SetColor(cEditWindowDisplay4);
            WriteEprom(IDno);


            IF EpromError = 0 THEN BEGIN
              str(IDno:3, IDstr);
              OutTextXY(EpromCount div 430 + ScreenX + 5, ScreenY + {25}yTextOffset, IDstr);
              ClockEprom;

              FOR f := Limit1 * 100 TO Limit2 * 100 DO BEGIN
                IF d^[f] = $FF THEN d^[f] := $FE;
                IF d^[f] = $0  THEN d^[f] := $1;
                WriteEprom(d^[f]);
                PutPixel(EpromCount div 430 + ScreenX, (ReadEprom div yScaling + ScreenY) +
                    yOffset - (256 div yScaling div 2), cEditWindowDisplay7);
                PutPixel((f - Limit1 * 100) div 100 + ScreenX, (d^[f] div 4) + ScreenY + 115,
                    cEditWindowDisplay7);
                ClockEprom;
              END;

              WHILE EpromCount <> ((EpromCount div 256) * 256)  DO BEGIN
                 WriteEprom(127);
                 PutPixel(EpromCount div 430 + ScreenX + 5,(ReadEprom div yScaling + ScreenY) +
                     yOffset - (256 div yScaling div 2), cEditWindowDisplay5);
                 ClockEprom;
              END;
              WriteEprom(0);
            END;
            SetTextJustify(LeftText, BottomText);
            SetTextStyle(DefaultFont, HorizDir,1);
            SetColor(cEditWindowDisplay4);
            EpromVppOff;
            {ClearBottomScreen;}

WriteBottomScreen('■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■■■■■■', 0);

            IF EpromError = 0 THEN BEGIN
              MessageBox('EPROM successfully programmed.',
                   'No errors were encountered',
                   'while transferring data from the',
                   'Record Buffer to the EPROM.',
                   'Any mouse key = continue');

            END;

            IF EpromError = 1 THEN BEGIN
              MessageBox('ERROR!',
                   'An error ocurred',
                   'during programming',
                   'of the EPROM.',
                   'Any mouse key = continue');
            END;
          END;
        END;
      END;

      EXIT:
      SetTextJustify(LeftText, BottomText);
      SetTextStyle(DefaultFont, HorizDir,1);
      SetColor(cEditWindowDisplay4);
      EpromVppOff;
      PutWholeScreen;
      Port[$30B] := 128;
      LeftMouseKeyWasPressed := FALSE;
      RightMouseKeyWasPressed := FALSE;
      FreeMouse;

      Button (WriteEX   ,WriteEY   ,ButtonSizeX,ButtonSizeY,'WriteE',cWriteE,off);
    END;

procedure _ReadE;
```

```
Var dv1, dv2, dv3 : LongInt;
  dv4 : byte;
  Code : Integer;
  dv0 : LongInt;
  IDstr : String;
  Divider, IDno : Word;
  EpromOK : Boolean;
  s, ss : string;
  EpromCountMarker : LongInt;

  Label exit;


BEGIN
  Button (ReadEX, ReadEY, ButtonSizeX, ButtonSizeY, 'ReadE', cReadE, on);

  SetMouseCursorPos(ReadEX + ButtonSizeX div 2, ReadEY + ButtonSizeY div 2);
  KeepMouseOffEditWindow;

  EpromCountMarker := 0;

  IF NOT DemoMode THEN BEGIN
    EpromOK := TRUE;
  END
  ELSE BEGIN
     EpromOK := FALSE;

MessageBox('NOTE',
           '',
           'ReadE not available in demo mode',
           '',
           'any key = exit');
    Goto exit;
  END;

  GetWholeScreen;
  EpromOK := FALSE;
  EpromError := 0;
  SetTextJustify(LeftText, BottomText);
  ClearScreen(0);

  EpromIcon(ScreenX + 305, ScreenY + ScreenSizeY div 2 - 50, 4);
  Str(EpromSize div 1000, s);
  s := Concat(s, 'K');
  SetTextStyle(DefaultFont, VertDir, 2);
  SetTextJustify(CenterText,CenterText);
  SetColor(LightGray);
  OutTextXY(ScreenX + (ScreenSizeX div 8) * 7, ScreenY + ScreenSizeY div 2, s);
  SetTextStyle(DefaultFont, HorizDir, 1);


  ResetEprom;

  IF ReadEprom <> 0 THEN BEGIN
    WriteEprom(0);
    IF EpromError = 0 THEN EpromOK := TRUE;
    IF EpromError = 1 THEN BEGIN
      EpromVppOff;
      EpromOK := FALSE;
      MessageBox('ERROR!',
           'EPROM not present',
           'or',
           'EPROM faulty.',
           'Any mouse key = continue');

    END;
  END
  ELSE EpromOK := TRUE;

  IF EpromOK THEN BEGIN
    ClockEprom;
    IF ReadEprom = 255 THEN BEGIN
      MessageBox('NOTE',
           '',
           'The EPROM is empty.',
           '',
           'Any mouse key = continue');
      EpromOK := FALSE;
```

```
            END;
            ResetEprom;
          END;

          IF EpromOK THEN BEGIN

WriteBottomScreen('■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■■■■■', 0);
WriteBottomScreen('ID no of message to read?', cEditWindowDisplay2);
            IdStr := '';
            REPEAT
               InputString(IDstr, ScreenX + 235, ScreenY + ScreenSizeY - 17, 2, 3, 0,
                      [EscapeKey, CarriageReturn, LeftMouseKey, RightMouseKey]);
               VAL(IDstr, IDno, code);
            UNTIL ((Code = 0) AND (IDno > 0) AND (IDno < 16{255})) OR (Key = EscapeKey) OR (Key = RightMouseKey);

            IF (Key = CarriageReturn) OR (Key = LeftMouseKey) THEN BEGIN

WriteBottomScreen('■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■■■■■', 0);
 SetTextStyle(DefaultFont, HorizDir, 3);
              SetTextJustify(CenterText,CenterText);
              SetColor(cEditWindowDisplay2);
              OutTextXY(ScreenX + ScreenSizeX div 4, ScreenY + ScreenSizeY div 2 - 15, 'SCANNING');
              OutTextXY(ScreenX + ScreenSizeX div 4, ScreenY + ScreenSizeY div 2 + 15, 'EPROM');
              SetTextStyle(DefaultFont, HorizDir, 1);

              s := ('');

              IF EpromSize = 128000 THEN Divider := 1938 div 4;
              IF EpromSize = 256000 THEN Divider := 1938 div 2;
              ResetEprom;
              ScreenMarker := (ScreenX + 300 + EpromCount div 1938);
              ScreenMarkerOnOff;

              REPEAT
                 WHILE (ReadEprom <> 0) AND (EpromCount < EpromSize) DO BEGIN

                   IF RightMouseKeyPressed THEN BEGIN
                     PutWholeScreen;
                     Goto Exit;
                   END;

                   IF EpromCount >= EpromCountMarker THEN BEGIN
                     ScreenMarkerOnOff;
                     ScreenMarker := (ScreenX + 300 + EpromCount div Divider);
                     ScreenMarkerOnOff;
                     Inc(EpromCountMarker, 1024 * 5);
                   END;

                   IF EpromCount + 256 < EpromSize THEN FOR f := 1 TO 256 DO ClockEprom
                   ELSE ClockEprom;

                 END;
                 ClockEprom;
                 IF ReadEprom <> IDno THEN BEGIN
                   IF EpromCount + 256 < EpromSize THEN FOR f := 1 TO 255 DO ClockEprom;

                 END;
              UNTIL (EpromCount >= EpromSize) OR (ReadEprom = IDno);
              ScreenMarkerOnOff;

              IF ReadEprom = IDno THEN BEGIN
                {
                SetMouseCursorPos(ReadEX + ButtonSizeX div 2, ReadEY + ButtonSizeY div 2);
                KeepMouseOffEditWindow;
                }
                dv1 := ScreenY + ScreenSizeY div 2 {((ScreenSizeY - 64) div 2)};
                ClearScreen(0);

                WriteBottomScreen(' Message found.- Busy reading.', cEditWindowDisplay2 + blink);
                dv2 := 0;

                FillChar(d^, SizeOf(d^), 127);
                REPEAT
                   inc(dv2);
                   ClockEprom;
                   dv4 := ReadEprom;
```

```
          d^[dv2] := dv4;
          IF dv4 <> 0 THEN
          PutPixel((dv2 div 100 + ScreenX),(dv4 div 2 + dv1 - 64), cEditWindowDisplay5);
          IF RightMouseKeyPressed THEN BEGIN
            PutWholeScreen;
            FreeMouse;
            Goto Exit;
          END;
        UNTIL (dv4 = 0) OR (EpromCount >= EpromSize);
        FreeMouse;

        SL1 := ScreenX+1;
        Limit1 := SL1 - ScreenX;
        SL2 := ScreenX + dv2 div 100;
        Limit2 := SL2 - ScreenX;

        SetColor(cEditWindowDisplay5);
        Line(dv2 div 100 + ScreenX, ({31 +} dv1), RecordBufferSize div 100 + ScreenX, dv1);

        IF EpromCount >= EpromSize THEN BEGIN
          MessageBox('ERROR!',
                'Could not find',
                'the end of the',
                'message.',
                'Any mouse key = continue');

        END;
      END
      ELSE BEGIN
          MessageBox('ERROR!',
                '',
                'Message not found',
                '',
                'Any mouse key = continue');

          PutWholeScreen;
          {LimitsOnOff;}
          Goto Exit;
      END;
    END
    ELSE BEGIN
        PutWholeScreen;
        Goto Exit;
    END;
  END
  ELSE BEGIN
      PutWholeScreen;
      Goto Exit;
  END;

WriteBottomScreen('■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■■■■■■', 0);
LimitsOnOff;
  ClearEpromIcon(ScreenX + 300, ScreenY + ScreenSizeY - 34, 1);

  Exit:
  FreeMouse;
  LeftMouseKeyWasPressed := FALSE;
  RightMouseKeyWasPressed := FALSE;
  Button (ReadEX, ReadEY, ButtonSizeX, ButtonSizeY, 'ReadE', cReadE, off);
END;

procedure _ListE;
    Var dv1,dv2,dv3:LongInt;
      Code : Integer;
      dv0:LongInt;
      IDstr:String;
      IDno:Word;
      EpromOK:Boolean;
    Label Exit;

    Procedure WriteBottomScreen( s : String; c : Byte );
        BEGIN
          HideMouseCursor;
          SetColor(c);
          SetTextJustify(LeftText,BottomText);
          OutTextXY(ScreenX + 5, ScreenY + ScreenSizeY - 5, s);
          ShowMouseCursor;
```

```
        END;

    Procedure ClearBottomScreen;
        BEGIN
          HideMouseCursor;
          LimitsOnOff;
          SetColor(0);
          SetTextJustify(LeftText,BottomText);
          OutTextXY(ScreenX + 5, ScreenY + ScreenSizeY - 5,
'██████████████████████████████████████████████████████████████████████', 0);

          LimitsOnOff;
          ShowMouseCursor;
        END;


    BEGIN
      Button (ListEX, ListEY, ButtonSizeX, ButtonSizeY, 'ListE', cListE, on);

      GetWholeScreen;
      EpromOK := FALSE;
      EpromError := 0;
      SetTextJustify(LeftText, BottomText);
      SetViewPort(ScreenX, ScreenY, ScreenX +
              ScreenSizeX, ScreenY + ScreenSizeY, ClipOn);
      ClearViewPort;
      SetViewPort(0, 0, GetMaxX, GetMaxY, ClipOn);

      SetTextStyle(SmallFont, HorizDir, 4);

      SetMouseCursorPos(ListEX + ButtonSizeX div 2, ListEY + ButtonSizeY div 2);
      KeepMouseOffEditWindow;

      SetLineStyle(DottedLn, 0, NormWidth);
      SetColor(cEditWindowDisplay6);
      Line(ScreenX + 1, ScreenY + 90, ScreenX + ScreenSizeX - 1, ScreenY + 090);
      Line(ScreenX + 1, ScreenY + 180, ScreenX + ScreenSizeX - 1, ScreenY + 180);
      SetLineStyle(SolidLn, 0, NormWidth);

      SetTextJustify(LeftText, TopText);
      OutTextXY( ScreenX + 1, 110, '0K');
      OutTextXY( ScreenX + 1, 200, '128K');

      SetTextJustify(CenterText, TopText);
      OutTextXY(ScreenX + ScreenSizeX div 2, 110, '64K');
      OutTextXY(ScreenX + ScreenSizeX div 2, 200, '192K');

      SetTextJustify(RightText, TopText);
      OutTextXY(ScreenX + ScreenSizeX, 110, '128K');
      OutTextXY(ScreenX + ScreenSizeX, 200, '256K');
      SetTextStyle(DefaultFont,HorizDir,1);

      WriteBottomScreen('Processing...', cEditWindowDisplay2);

      IF NOT DemoMode THEN BEGIN
        EpromOK:=TRUE;
      END
      ELSE BEGIN
         EpromOK := FALSE;
         MessageBox('NOTE',
                 '',
                 'ListE not available in demo mode',
                 '',
                 'any key = exit');
         PutWholeScreen;
         Goto Exit;
      END;

      ResetEprom;
      IF ReadEprom <> 0 THEN BEGIN
        WriteEprom(0);
        IF EpromError = 0 THEN EpromOK := TRUE;
        IF EpromError = 1 THEN BEGIN
          EpromVppOff;
          EpromOK := FALSE;
          MessageBox('ERROR!',
                 'EPROM not present',
```

```
                    'or EPROM empty',
                    'or EPROM faulty.',
                    'Any mouse key = continue');

          END;
        END;

        IF EpromOK THEN BEGIN
          {ClearBottomScreen;}
WriteBottomScreen('■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■■■■■■', 0);
WriteBottomScreen(' Listing EPROM...', cEditWindowDisplay2 + blink);
          SetTextJustify(CenterText, CenterText);
          SetTextStyle(SmallFont, VertDir, 4);
          SetColor(cEditWindowDisplay5);

          REPEAT
            dv1 := ReadEprom;
            IF dv1 = 0 THEN BEGIN
              ClockEprom;
              dv1 := ReadEprom;
              IF dv1 <> 255 THEN BEGIN
                SetColor(cEditWindowDisplay4);
                str(dv1 : 3, IDstr);
                IF EpromCount < 128000 THEN
                  OutTextXY(  EpromCount div 214 + ScreenX {+ 5}, ScreenY + 83, IDstr)
                ELSE OutTextXY((EpromCount - 128000) div 214 + ScreenX + 5, ScreenY + 173, IDstr);
                SetColor(cEditWindowDisplay5);
              END;
              {IF dv1<>255 THEN ClockEprom;}
              FOR f := 1 TO 31 DO ClockEPROM;
            END
            ELSE FOR f := 1 TO 32 DO ClockEPROM;

            IF EpromCount < 128000 THEN
              PutPixel(EpromCount div 214 + ScreenX {+ 5},(dv1 div 5 + ScreenY) + 40 - 25
                    , cEditWindowDisplay5)
            ELSE PutPixel((EpromCount - 214 * 600) div 214 + ScreenX {+ 5}, (dv1 div 5 + ScreenY) + 130 - 25
                    , cEditWindowDisplay5);

            IF RightMouseKeyPressed THEN GOTO Exit;
          UNTIL {(dv1 = 255) OR} (EpromCount > EpromSize);



          SetTextJustify(LeftText, BottomText);
          SetTextStyle(DefaultFont, HorizDir, 1);

          {ClearBottomScreen;}
WriteBottomScreen('■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■■■■■■', 0);
WriteBottomScreen(' Press any key to continue. ', cEditWindowDisplay2);
          REPEAT UNTIL AnyMousekeyPressed;

        END;

        LeftMouseKeyWasPressed := FALSE;
        RightMouseKeyWasPressed := FALSE;

        Exit:
        SetTextJustify(LeftText, BottomText);
        SetTextStyle(DefaultFont, HorizDir,1);

        PutWholeScreen;

        FreeMouse;
        LeftMouseKeyWasPressed := FALSE;
        RightMouseKeyWasPressed := FALSE;

        Button (ListEX, ListEY, ButtonSizeX, ButtonSizeY, 'ListE', cListE, off);
      END;

Procedure _Help;
      BEGIN
        Button (HelpX    ,HelpY    ,ButtonSizeX,ButtonSizeY,'HELP',cHelp,on);
        REPEAT UNTIL NOT(AnyMouseKeyPressed);
        Button (HelpX    ,HelpY    ,ButtonSizeX,ButtonSizeY,'HELP',cHelp,off);
```

```
    END;

Procedure _256K;
    BEGIN
      IF EpromSize = 128000 THEN Button (EpromSizeX, EpromSizeY, ButtonSizeX, ButtonSizeY, '128K', Brown  , on);
      IF EpromSize = 256000 THEN Button (EpromSizeX, EpromSizeY, ButtonSizeX, ButtonSizeY, '256K', Magenta, on);
      REPEAT UNTIL NOT(AnyMouseKeyPressed);
      IF EpromSize = 256000 THEN EpromSize := 128000 ELSE EpromSize := 256000;
      IF EpromSize = 128000 THEN Button (EpromSizeX, EpromSizeY, ButtonSizeX, ButtonSizeY, '128K', Brown  , off);
      IF EpromSize = 256000 THEN Button (EpromSizeX, EpromSizeY, ButtonSizeX, ButtonSizeY, '256K', Magenta, off);
    END;

Procedure _Quit;
    BEGIN
      Button (QuitX   ,QuitY   ,ButtonSizeX,ButtonSizeY,'QUIT',cQuit,on);
      REPEAT UNTIL NOT(AnyMouseKeyPressed);
      Button (QuitX   ,QuitY   ,ButtonSizeX,ButtonSizeY,'QUIT',cQuit,off);

      MessageBox('QUIT',
            '',
            'Are you sure?',
            '',
            'left = yes / right = no');
      IF LeftMouseKeyWasPressed THEN BEGIN
        IF NOT DemoMode THEN UnInstallAdcInt;
        SetTextJustify(LeftText,BottomText);
        CloseGraph;
        ClrScr;
        Release(Heap);
        Halt;
      END;
      SetTextJustify(LeftText,BottomText);
      ;
    END;

procedure MouseStuff;
    BEGIN

      IF LeftMouseKeyWasPressed THEN
      BEGIN

        IF (LeftMouseKeyWasPressed) AND ((MouseTouchArea(ScreenX, ScreenY, ScreenSizeX, ScreenSizeY)))
        THEN DrawLimit1(0);


        IF (LeftMouseKeyWasPressed) AND (MouseTouchArea(SampleX  ,SampleY ,ButtonSizeX,ButtonSizeY)) THEN
_Record;
        IF (LeftMouseKeyWasPressed) AND (MouseTouchArea(TalkX    ,TalkY   ,ButtonSizeX,ButtonSizeY)) THEN _Play ;
        IF (LeftMouseKeyWasPressed) AND (MouseTouchArea(MonitorX ,monitorY,ButtonSizeX,ButtonSizeY)) THEN
_Mute;
        IF (LeftMouseKeyWasPressed) AND (MouseTouchArea(CopyX    ,CopyY   ,ButtonSizeX,ButtonSizeY)) THEN
_Copy;
        IF (LeftMouseKeyWasPressed) AND (MouseTouchArea(MoveX    ,MoveY   ,ButtonSizeX,ButtonSizeY)) THEN
_Move;
        IF (LeftMouseKeyWasPressed) AND (MouseTouchArea(DeleteX  ,DeleteY ,ButtonSizeX,ButtonSizeY)) THEN
_Delete;
        IF (LeftMouseKeyWasPressed) AND (MouseTouchArea(SaveX    ,SaveY   ,ButtonSizeX,ButtonSizeY)) THEN  Save;
        IF (LeftMouseKeyWasPressed) AND (MouseTouchArea(LoadX    ,LoadY   ,ButtonSizeX,ButtonSizeY)) THEN  Load;
        IF (LeftMouseKeyWasPressed) AND (MouseTouchArea(DirectoryX,DirectoryY,ButtonSizeX,ButtonSizeY)) THEN
_Path;


        IF (LeftMouseKeyWasPressed) AND (MouseTouchArea(WriteEX  ,WriteEY ,ButtonSizeX,ButtonSizeY)) THEN
_WriteE;
        IF (LeftMouseKeyWasPressed) AND (MouseTouchArea(ReadEX   ,ReadEY  ,ButtonSizeX,ButtonSizeY)) THEN
_ReadE;
        IF (LeftMouseKeyWasPressed) AND (MouseTouchArea(ListEX   ,ListEY  ,ButtonSizeX,ButtonSizeY)) THEN _ListE;
        IF (LeftMouseKeyWasPressed) AND (MouseTouchArea(HelpX    ,HelpY   ,ButtonSizeX,ButtonSizeY)) THEN Help ;
        IF (LeftMouseKeyWasPressed) AND (MouseTouchArea(EpromSizeX,EpromSizeY,ButtonSizeX,ButtonSizeY)) THEN
_256K;

        IF (LeftMouseKeyWasPressed) AND (MouseTouchArea(QuitX,QuitY,ButtonSizeX,ButtonSizeY)) THEN        _Quit;


        IF (LeftMouseKeyWasPressed) AND (MouseTouchArea(TriggerX,TriggerY,SliderSizeX,PotSizeY)) THEN BEGIN
          _Trigger:=DrawSlider(TriggerX,TriggerY,_Trigger);
          Trigger := _Trigger - 8;
```

```
            IF Trigger > 127 THEN Tri
            Trigger := 127 - Trigger;
            {
            Str(Trigger, ErrorMessage);
            SetColor(Black);
            OutTextXY(130, 242, 'ÛÛÛ');
            SetColor(LightBlue);
            OutTextXY(130, 242, ErrorMessage);
            }
          END;


          IF (LeftMouseKeyWasPressed) AND (MouseTouchArea(SensitivityX,SensitivityY,SliderSizeX,PotSizeY)) THEN
BEGIN
            Sensitivity := DrawSlider(SensitivityX, SensitivityY, Sensitivity);
            IF NOT DemoMode THEN Port[VolPort] := (((Volume div 20) * 8) + (Sensitivity div 20));
            {
            Str((port[VolPort]), ErrorMessage);
            SetColor(Black);
            OutTextXY(190, 242, 'ÛÛÛ');
            SetColor(LightBlue);
            OutTextXY(190, 242, ErrorMessage);
            }
          END;

          IF (LeftMouseKeyWasPressed) AND (MouseTouchArea(volumeX,volumeY,SliderSizeX,PotSizeY)) THEN BEGIN
            volume := DrawSlider(volumeX, volumeY, volume);
            IF NOT DemoMode THEN Port[VolPort] := (((Volume div 20) * 8) + (Sensitivity div 20));
            {
            Str((port[VolPort]), ErrorMessage);
            SetColor(Black);
            OutTextXY(502, 242, 'ÛÛÛ');
            SetColor(LightBlue);
            OutTextXY(502, 242, ErrorMessage);
            }
          END;
        END;

        IF RightMouseKeyWasPressed THEN
        BEGIN
          IF (RightMouseKeyWasPressed) AND ((MouseTouchArea(ScreenX, ScreenY, ScreenSizeX, ScreenSizeY)))
          {AND (MouseX > SL1)} THEN DrawLimit2(0);
        END;

      END;

procedure KeyBrdStuff;
      var start, stop : Word;
        d1  : Byte;
        Key : KeyType;
      BEGIN
        InKey(Ch,fk,key);
        b := Ord(Ch) + Pitch;
        IF (b >= 0) AND (NOT DemoMode) THEN BEGIN
          DisAbleAdcInt;

          Port[VolPort] := (((Volume div 20) * 8) + (Sensitivity div 20));

          start := Limit1 * 100;
          stop  := Limit2 * 100;

          REPEAT
            IF NOT DemoMode THEN Port[DacPort] := d^[start];
            for s:=1 to b do;
            INC(start);
          UNTIL (start >= stop) OR (KeyPressed);

          EnableAdcInt;
        END;
      END;

{*********************************************************************}
BEGIN

  ResetMouse;
  IF Not MousePresent THEN BEGIN
    RestoreCRTmode;
    TextColor(7); TextBackGround(0);
```

```
  ClrScr;
  WriteLn('   ************************************************************************');
  WriteLn('   *                                                                      *');
  WriteLn('   *                  Mouse driver not present.                           *');
  WriteLn('   *                                                                      *');
  WriteLn('   *                                                                      *');
  WriteLn('   *                  Program terminated.                                 *');
  WriteLn('   *                                                                      *');
  WriteLn('   *                                                                      *');
  WriteLn('   ************************************************************************');
  Halt;
END
ELSE WriteLn('Mouse detected OK');


Retry:

Mark(Heap);

DetectTalkHardWare;

Monitor := Off;
EpromSize := 256000;

SetGraphicsCursor(StandardShapeCurs);


GraphDriver := Detect;
InitGraph(GraphDriver,GraphMode,'');
IF NOT (GraphDriver IN [VGA] ) THEN BEGIN
  TextColor(7); TextBackGround(0);
  RestoreCRTmode;
  ClrScr;
  WriteLn('   ************************************************************************');
  WriteLn('   *                                                                      *');
  WriteLn('   *                  No VGA Hardware present.                            *');
  WriteLn('   *                                                                      *');
  WriteLn('   *                                                                      *');
  WriteLn('   *                  Program terminated.                                 *');
  WriteLn('   *                                                                      *');
  WriteLn('   *                                                                      *');
  WriteLn('   ************************************************************************');
  HALT;
END
ELSE WriteLn('VGA detected OK');


GetMem(imageMessageBox, ImageSize(0, 0, MessageBoxSizeX, MessageBoxSizeY));
IF MemAvail < 60000 THEN BEGIN
  MessageBox('WARNING','', 'not enough memory','','any mouse key = continue');
  _Quit;
END;

GetMem(imageScreenPart,ImageSize( ScreenX {+ 1}, ScreenY + 1, ScreenX + ScreenSizeX {- 1}, ScreenY + ScreenSizeY - 1));

GetMem(imageWholeScreen,ImageSize(ScreenX {+ 1}, ScreenY + 1, ScreenX + ScreenSizeX {- 1}, ScreenY + ScreenSizeY -
1));

GetMem(imagePotButton, ImageSize(30,30,30+SliderSizeX,30+SliderSizeY+10));

GetMem(imageLimits, ImageSize(ScreenX, ScreenY, ScreenX, ScreenY+ ScreenSizeY));

GetMem(ImagePot,ImageSize(TriggerX,TriggerY-SliderSizeY div 2,TriggerX+PotSizeX,TriggerY+PotSizeY+SliderSizeY div
2));

RecordBufferSize := MemAvail;
IF RecordBufferSize > ProposedRecordBufferSize THEN RecordBufferSize := ProposedRecordBufferSize;
GetMem(d, RecordBufferSize);

FillChar(d^, SizeOf(d^), 127);

IF NOT DemoMode THEN Port[$30B] := 128; {a,b&c out}

SetGraphicsCursor(StandardShapeCurs);
HideMouseCursor;

SL1:=ScreenX+1;
```

```
  SL2:=ScreenX+ScreenSizeX-1;
  Limit1:=SL1-ScreenX;
  Limit2:=SL2-ScreenX;

  SetTextStyle(0,0,1);

  ClearDevice;

  {Get pot button image}
  HideMouseCursor;
  Button (30,30,SliderSizeX,SliderSizeY ,",blue,off);
  GetImage(30,30,SliderSizeX+30,SliderSizeY+30,imagePotButton^);
  ClearDevice;

  {Get Limit1 and Limit2 line}
  HideMouseCursor;
  SetColor(cLimits);
  Line(ScreenX, ScreenY + 1, ScreenX, ScreenY + ScreenSizeY - 1);
  GetImage(ScreenX, ScreenY, ScreenX, ScreenY + ScreenSizeY, imageLimits^);

  Screen;
  HideMouseCursor;

  Button (SampleX   ,SampleY   ,ButtonSizeX,ButtonSizeY,'RECORD',cRecord,off);
  Button (TalkX     ,TalkY     ,ButtonSizeX,ButtonSizeY,'PLAY',cPlay,off);
  _Mute;

  HideMouseCursor;
  DrawPot(TriggerX,TriggerY,'TRIGGER',cTrigger);

  {Get pot Image}
  GetImage(TriggerX,TriggerY-SliderSizeY div 2,TriggerX+PotSizeX,TriggerY+PotSizeY+SliderSizeY div 2,imagePot^);

  SetMouseCursorPos(TriggerX,TriggerY + (PotSizeY - Trigger));
  _Trigger:=DrawSlider(TriggerX,TriggerY,_Trigger);
  Trigger := _Trigger - 8;
  IF Trigger > 127 THEN Trigger := 127;
  Trigger := 127 - Trigger;

  DrawPot(SensitivityX, SensitivityY, 'LEVEL', cSensitivity);
  SetMouseCursorPos(SensitivityX, SensitivityY + (PotSizeY - Sensitivity));
  Sensitivity := DrawSlider(SensitivityX, SensitivityY, Sensitivity);

  Button (CopyX     ,CopyY     , ButtonSizeX, ButtonSizeY, 'COPY'  , cCopy  , off);
  Button (MoveX     ,MoveY     , ButtonSizeX, ButtonSizeY, 'MOVE'  , cMove  , off);
  Button (DeleteX   ,DeleteY   , ButtonSizeX, ButtonSizeY, 'DELETE', cDelete, off);
  Button (SaveX     ,SaveY     , ButtonSizeX, ButtonSizeY, 'SAVE'  , cSave  , off);
  Button (LoadX     ,LoadY     , ButtonSizeX, ButtonSizeY, 'LOAD'  , cLoad  , off);
  Button (DirectoryX ,DirectoryY , ButtonSizeX, ButtonSizeY, 'PATH'  , cPath  , off);
  Button (WriteEX   ,WriteEY   , ButtonSizeX, ButtonSizeY, 'WriteE', cWriteE, off);
  Button (ReadEX    ,ReadEY    , ButtonSizeX, ButtonSizeY, 'ReadE' , cReadE , off);
  Button (ListEX    ,ListEY    , ButtonSizeX, ButtonSizeY, 'ListE' , cListE , off);

  DrawPot(volumeX, volumeY, 'VOLUME', cVolume);
  SetMouseCursorPos(VolumeX, VolumeY + (PotSizeY - Volume));
  Volume := DrawSlider(VolumeX, VolumeY, Volume);

  Button (HelpX     ,HelpY     ,ButtonSizeX , ButtonSizeY, 'HELP'  ,cHelp   , off);
  _256K;
  Button (QuitX     ,QuitY     ,ButtonSizeX , ButtonSizeY, 'QUIT'  ,cQuit   , off);

  SL1:=ScreenX;
  SL2:=ScreenX + ScreenSizeX;
  FileToBeSaved := FALSE;
  LimitsOnOff;
  SetTextJustify(CenterText,BottomText);

  {INTRO TEXT}
{
  zxc := MemAvail;
  Str(zxc, errormessage);
}
  SetColor(cIntroText);
  SetTextJustify(CenterText, CenterText);
  SetTextStyle(Defaultfont,HorizDir,1);
  OutTextXY(ScreenX + ScreenSizeX div 2, ScreenY + ScreenSizeY div 2 - 30, 'DESIGN OF A PROGRAMMABLE TIME-
DOMAIN');
  OutTextXY(ScreenX + ScreenSizeX div 2, ScreenY + ScreenSizeY div 2 - 15, 'SPEECH-SYNTHESIS SYSTEM.');
```

```
    OutTextXY(ScreenX + ScreenSizeX di                    div 2 + 0,  'K. N. van der WALT');
    OutTextXY(ScreenX + ScreenSizeX div 2, ScreenY + ScreenSizeY div 2 + 15, 'January 1995');
{
    OutTextXY(ScreenX + ScreenSizeX div 2, ScreenY + ScreenSizeY div 2 + 30,  ErrorMessage);
}
    SetTextStyle(DefaultFont,HorizDir,1);

    SetMouseCursorPos(GetMaxX div 2, GetMaxY div 2);
    ShowMouseCursor;
    d1 := MouseX; d2 := MouseY;
    Delay(500);
    ClearScreen(0);
    LimitsOnOff;

    IF NOT DemoMode THEN InstallAdcInt;

    IF DemoMode THEN BEGIN
      MessageBox('DEMO MODE',
              '',
              'No interrupt signal detected.',
              '',
              'left = demo \ right = retry');
      IF RightMouseKeyWasPressed THEN BEGIN
        RightMouseKeyWasPressed := FALSE;
        IF NOT DemoMode THEN UnInstallAdcInt;
        SetTextJustify(LeftText,BottomText);
        CloseGraph;
        ClrScr;
        ResetMouse;

        Release(Heap);

        Goto Retry;
      END;

      REPEAT UNTIL NOT AnyMouseKeyPressed;
    END;

    EndLessLoop:
        LeftMouseKeyWasPressed := FALSE;
        RightMouseKeyWasPressed := FALSE;
        REPEAT
            IF LeftMouseKeyPressed  THEN LeftMouseKeyWasPressed  := TRUE;
            IF RightMouseKeyPressed THEN RightMouseKeyWasPressed := TRUE;
        UNTIL (KeyPressed) OR (LeftMouseKeyWasPressed) OR (RightMouseKeyWasPressed);

        IF KeyPressed THEN KeyBrdStuff ELSE MouseStuff;

    GOTO EndLessLoop;
END.
```

Unit Disp;

```
(*************************************************************************)
Interface
(*************************************************************************)

Uses MousU, Graph, Variable,{} CRT, gKeybrd, DOS;

procedure Beep;
Procedure D_SlopedSquare(x,         {x coordinate}
                         y,         {y coordinate}
                         xx,        {x size}
                         yy,        {y size}
                         cf,        {face color}
                         cn,        {color of northern slope}
                         ce,        {color of eastern slope}
                         cw,        {color of western slope}
                         cs,        {color of southern slope}
                         cb,        {color of border}
                         sw:Integer;  {width of slope}
                         logo:string; {logo to write on square}
                         cl:Byte;     {logo color}
                         lo:Byte);    {logo offset}
Procedure KeepMouseOffEditWindow;
procedure GetScreenPart (x1,y1,x2,y2 : Integer);
procedure PutScreenPart (x1,y1 : Integer;PutMode:Word);
procedure GetWholeScreen;
procedure PutWholeScreen;
procedure WriteTopScreen( s : String; c : Byte);
procedure WriteBottomScreen( s : String; c : Byte);
procedure LimitsOnOff;
procedure DrawScreenMarker(d1:LongInt);
procedure ScreenMarkerOnOff;
procedure DrawLimit2(d1:LongInt);
procedure DrawLimit1(d1:LongInt);
procedure MessageBox (s1, s2a, s2b, s2c, s3 : String);
Function  DrawSlider(PotX,PotY,Value:Integer):Integer;
procedure Button (x,y,xx,yy:Integer;Logo:String;LogoColor:Byte;Status:Monitortype);
procedure DrawPot(PotX,PotY:Integer;Logo: String; LogoColor : Byte);
procedure Screen;
Procedure ClearScreen(ReductionY : Byte);
procedure Error;
procedure MoveMouseTo(d1,d2:Integer);
Function  DirFileName : String;
Procedure EpromIcon(Xoffset, YOffset, Magnification : Word);
Procedure ClearEpromIcon(Xoffset, YOffset, Magnification : Word);
procedure ClearTextWindows;

(*************************************************************************)
Implementation
(*************************************************************************)

procedure Beep;
      BEGIN
         Sound(1000);
         Delay(100);
         NoSound;
      END;

Procedure KeepMouseOffEditWindow;
      BEGIN
         SetMinMaxVertCursPos(ScreenY + ScreenSizeY, GetMaxY);
      END;

procedure GetScreenPart (x1,y1,x2,y2 : Integer);
      BEGIN
         HideMouseCursor;
         GetImage(x1, y1, x1 + x2, y1 + y2, imageScreenPart^);
         ShowMouseCursor;
      END;

procedure PutScreenPart (x1, y1 : Integer; PutMode : Word);
      BEGIN
         HideMouseCursor;
         PutImage(x1, y1, imageScreenPart^, PutMode);
         ShowMouseCursor;
      END;
```

```
procedure GetWholeScreen;
    BEGIN
      HideMouseCursor;
      GetImage(ScreenX {+ 1}, ScreenY + 1, ScreenX + ScreenSizeX {- 1}, ScreenY + ScreenSizeY - 1, imageWholeScreen^);
      ShowMouseCursor;
    END;

procedure PutWholeScreen;
    BEGIN
      HideMouseCursor;
      PutImage(ScreenX {+ 1}, ScreenY + 1, imageWholeScreen^, NormalPut);
      ShowMouseCursor;
    END;

procedure WriteTopScreen( s : String; c : Byte);
    BEGIN
      SetColor(c);
      SetTextJustify(LeftText,TopText);
      HideMouseCursor;
      OutTextXY(ScreenX + {10}17, ScreenY + 18, s);
      ShowMouseCursor;
    END;

procedure WriteBottomScreen( s : String; c : Byte );
    BEGIN
      SetColor(c);
      SetTextJustify(LeftText,BottomText);
      HideMouseCursor;
      OutTextXY(ScreenX + {10}17, ScreenY + ScreenSizeY - 17,s);
      ShowMouseCursor;
    END;

procedure LimitsOnOff;
    BEGIN
    HideMouseCursor;
    PutImage(SL1,ScreenY,imageLimits^,xorPut);
    PutImage(SL2,ScreenY,imageLimits^,xorPut);
    ShowmouseCursor;
    END;

procedure DrawScreenMarker(d1:LongInt);
    BEGIN
      IF d1 <> ScreenMarker THEN
      BEGIN
        HideMouseCursor;
        PutImage(ScreenMarker,ScreenY,imagelimits^,xorPut);
        PutImage(d1,ScreenY,imagelimits^,xorPut);
        ShowMouseCursor;
        ScreenMarker := d1;
      END;
    END;

procedure ScreenMarkerOnOff;
    BEGIN
      HideMouseCursor;
      PutImage(ScreenMarker,ScreenY,imagelimits^,xorPut);
      ShowMouseCursor;
    END;

procedure DrawLimit1(d1:LongInt{Integer});
    Const PlayRange = 5;
    Var  x : Word{Integer};
    BEGIN
      REPEAT
        IF d1 = 0 THEN x := MouseX ELSE x := {SL1+}d1;
        IF x < ScreenX then x := ScreenX;
        IF (x > SL2 - 5) AND (d1 =  0) THEN BEGIN DrawLimit2( x + 5); x := SL2 - 5; END;
        IF (x > SL2 - 5) AND (d1 <> 0) THEN BEGIN DrawLimit2(    1); x := SL2 - 5; END;

        IF x < ScreenX THEN x := ScreenX;
        IF x > ScreenX + ScreenSizeX THEN x := ScreenX + ScreenSizeX;

        IF x <> SL1 THEN BEGIN
          HideMouseCursor;
          PutImage(SL1, ScreenY, imageLimits^, xorPut);
          PutImage(x, ScreenY, imageLimits^, xorPut);
          ShowMouseCursor;
```

```
            SL1 := x;
          END;

          IF (MouseY < (ScreenY + (ScreenSizeY div 2))) AND (NOT DemoMode) THEN BEGIN
            {For f := ((SL1-ScreenX-2)*100) TO ((SL1-ScreenX+2)*100) DO}
            dPointer := ((SL1 - ScreenX - {2}PlayRange) * 100);
            dPointerStop := ((SL1 - ScreenX + {2}PlayRange) * 100);
            TalkStat := Play;

            REPEAT
            UNTIL TalkStat <> Play;
          END;
        UNTIL NOT (LeftMouseKeyPressed) OR (d1 <> 0);
        Limit1 := SL1 - ScreenX;
      END;

procedure DrawLimit2(d1:LongInt{Integer});
      Const PlayRange = 5;
      Var x : Word{Integer};
      BEGIN
        REPEAT
          IF d1 = 0 THEN x := MouseX ELSE x := {SL2+}d1;
          IF x < ScreenX then x := ScreenX;
          IF (x < SL1 + 5) AND (d1 =  0) THEN BEGIN DrawLimit1( x - 5); x := SL1 + 5; END;
          IF (x < SL1 + 5) AND (d1 <> 0) THEN BEGIN DrawLimit1(  - 1); x := SL1 + 5; END;
          IF x>ScreenX+ScreenSizeX THEN x:=ScreenX+ScreenSizeX;

          IF x < ScreenX THEN x := ScreenX;
          IF x > ScreenX + ScreenSizeX THEN x := ScreenX + ScreenSizeX;

          IF x <> SL2 THEN BEGIN
            HideMouseCursor;
            PutImage(SL2,ScreenY, imageLimits^, xorPut);
            PutImage(x,  ScreenY, imageLimits^, xorPut);
            ShowMouseCursor;
            SL2 := x;
          END;

          IF (MouseY < (ScreenY + (ScreenSizeY div 2))) AND (NOT DemoMode) THEN BEGIN
            dPointer := ((SL2 - ScreenX - {2}PlayRange) * 100);
            dPointerStop := ((SL2 - ScreenX + {2}PlayRange) * 100);
            TalkStat := Play;
            REPEAT
            UNTIL TalkStat <> Play;
            {TalkStat := Idle;}
          END;
        UNTIL NOT (RightMouseKeyPressed) OR (d1 <> 0);
        Limit2 := SL2 - ScreenX;
      END;

procedure MessageBox (s1, s2a, s2b, s2c, s3 : String);
      Var MessageBoxX, MessageBoxY : Integer;
      BEGIN
        Beep;
        LeftMouseKeyWasPressed  := FALSE;
        RightMouseKeyWasPressed := FALSE;
        HideMouseCursor;
        MessageBoxX := ScreenX + (ScreenSizeX - MessageBoxSizeX) div 2;
        MessageBoxY := ScreenY + (ScreenSizeY - MessageBoxSizeY) div 2;
        GetImage(MessageBoxX, MessageBoxY, MessageBoxX + MessageBoxSizeX,
              MessageBoxY + MessageBoxSizeY, imageMessageBox^);
        SetViewPort(MessageBoxX, MessageBoxY, MessageBoxX + MessageBoxSizeX,
               MessageBoxY + MessageBoxSizeY, ClipOn);
        ClearViewPort;
        SetViewPort(0, 0, GetMaxX, GetMaxY, ClipOn);
        SetColor(cEditWindowDisplay8);
        RecTangle(MessageBoxX, MessageBoxY, MessageBoxX + MessageBoxSizeX,
              MessageBoxY + MessageBoxSizeY);

        SetColor(LightRed);
        SetTextJustify(CenterText, TopText);
        OutTextXY(MessageBoxX + MessageBoxSizeX div 2, MessageBoxY + 10, s1);

        SetColor(cEditWindowDisplay3);
        SetTextJustify(CenterText, CenterText);
        OutTextXY(MessageBoxX + MessageBoxSizeX div 2 , MessageBoxY + MessageBoxSizeY div 2 - 10, s2a);
        OutTextXY(MessageBoxX + MessageBoxSizeX div 2 , MessageBoxY + MessageBoxSizeY div 2     , s2b);
        OutTextXY(MessageBoxX + MessageBoxSizeX div 2 , MessageBoxY + MessageBoxSizeY div 2 + 10, s2c);
```

```
        SetColor(cEditWindowDisplay7);
        SetTextJustify(CenterText, BottomText);
        OutTextXY(MessageBoxX + MessageBoxSizeX div 2, MessageBoxY + MessageBoxSizeY - 10, s3);

        ShowMouseCursor;
        REPEAT UNTIL NOT AnyMouseKeyPressed;
        REPEAT
            IF LeftMouseKeyPressed  THEN LeftMouseKeyWasPressed  := TRUE;
            IF RightMouseKeyPressed THEN RightMouseKeyWasPressed := TRUE;
        UNTIL (LeftMouseKeyWasPressed) OR (RightMouseKeyWasPressed);
        HideMouseCursor;
        PutImage(MessageBoxX, MessageBoxY, imageMessageBox^, NormalPut);
        ShowMouseCursor;
      END;

Function xDrawSlider(PotX,PotY,Value,dir:Integer):Integer;
     Var d1,d2,d3:Integer;
     BEGIN
        d1 := SliderSizeY div 2;
        d2 := Value + PotY;
        IF dir = 0  THEN d3 := MouseY;
        IF dir <> 0 THEN d3 := d2 + dir;
        IF d3 < PotY THEN d3 := PotY;
        IF d3 > PotY + PotSizeY THEN d3 := PotY + PotSizeY;
        IF d3 <> d2 THEN

        REPEAT
            HideMouseCursor;
            PutImage(PotX, d3 - d1, ImagePotButton^, normalput);
            ShowMouseCursor;
            d2 := d3;
        UNTIL NOT LeftMouseKeyPressed;
        xDrawSlider := d2 - PotY;


       SetMinMaxHorzCursPos(0, GetMaxX);
       SetMinMaxVertCursPos(0, GetMaxY);

     END;

Function DrawSlider(PotX,PotY,Value:Integer):Integer;
      Var d1,d2,d3:Integer;
      BEGIN
        d1 := SliderSizeY Div 2;
        d2 := Value+PotY;
        SetMinMaxHorzCursPos(MouseX,MouseX);
        SetMinMaxVertCursPos(VolumeY,VolumeY+PotSizeY);
        REPEAT
            d3 := MouseY;
            IF d3 <  PotY + SliderSizeY div 4 THEN d3 := PotY + SliderSizeY div 4;
            IF d3 >  PotY + PotSizeY - SliderSizeY div 4 THEN d3 := PotY + PotSizeY - SliderSizeY div 4;
            IF d3 <> d2 THEN BEGIN
              HideMouseCursor;
              PutImage(PotX, PotY - d1, imagePot^, NormalPut);
              PutImage(PotX, d3 - d1, imagePotButton^, NormalPut);
              ShowMouseCursor;
              d2 := d3;
            END;
        UNTIL NOT LeftMouseKeyPressed;
        DrawSlider := d2 - PotY;

        SetMinMaxHorzCursPos(0, GetMaxX);
        SetMinMaxVertCursPos(0, GetMaxY);
      END;

Procedure D_SlopedSquare(x,         {x coordinate}
                         y,         {y coordinate}
                         xx,        {x size}
                         yy,        {y size}
                         cf,        {face color}
                         cn,        {color of northern slope}
                         ce,        {color of eastern slope}
                         cw,        {color of western slope}
                         cs,        {color of southern slope}
                         cb,        {color of border}
                         sw:Integer;  {width of slope}
                         logo:string; {logo to write on square}
```

```
            cl:Byte;    {logo color}
            lo:Byte);   {logo offset}
    BEGIN
      HideMouseCursor;

      SetFillStyle(1,cf);
      SetColor(cf);
      BAR(x,y,x+xx,y+yy);
      SetColor(cb);
      Rectangle(x,y,x+xx,y+yy);

      {northern slope}
      SetColor(cn);
      MoveTo(x+1,y+1);
      LineTo(x+xx-1,y+1);
      LineTo(x+xx-1-sw,y+1+sw);
      LineTo(x+1+sw,y+1+sw);
      LineTo(x+1,y+1);
      SetFillStyle(1,cn);
      FloodFill(x+(xx div 2),y+(sw div 2),cn);

      {eastern slope}
      SetColor(ce);
      MoveTo(x+1,y+1);
      LineTo(x+1,y+yy-1);
      LineTo(x+1+sw,y+yy-1-sw);
      LineTo(x+1+sw,y+1+sw);
      LineTo(x+1,y+1);
      SetFillStyle(1,ce);
      FloodFill(x+(sw div 2),y+(yy div 2),ce);

      {western slope}
      SetColor(cw);
      MoveTo(x+xx-1,y+yy-1);
      LineTo(x+xx-1,y+1);
      LineTo(x+xx-1-sw,y+1+sw);
      LineTo(x+xx-1-sw,y+yy-1-sw);
      LineTo(x+xx-1,y+yy-1);
      SetFillStyle(1,cw);
      FloodFill(x+xx-(sw div 2),y+(yy div 2),cw);

      {southern slope}
      SetColor(cs);
      MoveTo(x+xx-1,y+yy-1);
      LineTo(x+1,y+yy-1);
      LineTo(x+1+sw,y+yy-1-sw);
      LineTo(x+xx-1-sw,y+yy-1-sw);
      LineTo(x+xx-1,y+yy-1);
      SetFillStyle(1,cs);
      FloodFill(x+(xx div 2),y+yy-(sw div 2),cs);

      {write logo}
      SetTextJustify(CenterText,CenterText);
      SetColor(cl);
      OutTextXY(x + xx div 2 + lo, y + yy div 2 + lo,Logo);

      ShowMouseCursor;
    END;

Procedure Button (x,y,xx,yy:Integer;Logo:String;LogoColor:Byte;Status:Monitortype);
    BEGIN

    IF Status = off THEN BEGIN
      D_SlopedSquare(x,              {x coordinate}
                y,              {y coordinate}
                xx,             {x size}
                yy,             {y size}
                cButton,        {face color}
                cButtonShade2,  {color of northern slope}
                cbuttonShade2,  {color of eastern slope}
                cButtonShade1,  {color of western slope}
                cButtonShade1,  {color of southern slope}
                cButtonShade3,  {color of border}
                ButtonBorder,   {width of slope}
                Logo,           {logo to write on square}
                LogoColor,      {Logo color}
                0);             {logo offset}
    END;
```

```
    IF Status = on THEN BEGIN
      D_SlopedSquare(x,              {x coordinate}
                y,              {y coordinate}
                xx,             {x size}
                yy,             {y size}
                cButtonShade2,     {face color}
                cButtonShade1,          {color of northern slope}
                cbuttonShade1,          {color of eastern slope}
                cButton,          {color of western slope}
                cButton,          {color of southern slope}
                cButtonShade3,     {color of border}
                ButtonBorder,      {width of slope}
                Logo,           {logo to write on square}
                LogoColor,         {Logo color}
                2);             {logo offset}
    END;
    END;

procedure DrawPot(PotX,PotY:Integer;Logo: String; LogoColor : Byte);

    Const   StripeLength = PotSizeX DIV 2;
            StripeCount  = PotSizeY DIV 8;
            SlotWidth    = {PotSizeX DIV 3}16;

    Var   d1,d2,x,y:Integer;

    BEGIN
      HideMouseCursor;
      SetColor(PotColor);
      d2:=PotSizeY DIV StripeCount;
      Rectangle(PotX + SlotWidth, PotY, PotX + PotSizeX - SlotWidth, PotY + PotSizeY);
      SetFillStyle(1, PotColor);
      FloodFill(PotX + PotSizeX div 2, PotY + PotY div 2, PotColor);

      FOR  d1 := 1 TO StripeCount - 1 DO
        BEGIN
          Line(PotX + SlotWidth - StripeLength, PotY + d1 * d2, PotX + SlotWidth, PotY + d1 * d2);
          Line(PotX + PotSizeX - SlotWidth, PotY + d1 * d2, PotX + PotSizeX - SlotWidth + StripeLength,
            PotY + d1 * d2);
        END;
      SetTextJustify(CenterText, BottomText);
      SetColor(LogoColor);
      OutTextXY(PotX + PotSizeX div 2, PotY + PotSizeY + SliderSizeY, Logo);
      ShowMouseCursor;
    END;

procedure Screen;
    Var  x,y,xx,yy : Integer;
    BEGIN
      HideMouseCursor;

      {BACKGROUND}

      x:=0;
      y:=0;
      xx:=GetmaxX;
      yy:=GetMaxY;

      SetFillStyle(1,cBackGround);
      SetColor(cBackGround);
      BAR(x,y,x+xx,y+yy);
      SetColor(cBackgroundShade3);
      Rectangle(x,y,x+xx,y+yy);

      SetColor(cBackgroundShade1);
      Line(x+1,y+1,x+1,y+yy-1);
      Line(x+1,y+yy-1,x+xx-1,y+yy-1);
      Line(x+xx-1,y+yy-1,x+xx-1-BackgroundBorder,y+yy-1-BackgroundBorder);
      Line(x+xx-1-BackgroundBorder,y+yy-1-BackgroundBorder,x+1+BackgroundBorder,y+yy-1-BackgroundBorder);
      Line(x+1+BackgroundBorder,y+yy-1-BackgroundBorder,x+1+BackgroundBorder,y+1+BackgroundBorder);
      Line(x+1+BackgroundBorder,y+1+BackgroundBorder,x+1,Y+1);
      SetFillStyle(1,cBackGroundShade1);
      FloodFill(x+2,y+yy-2,cBackGroundShade1);

      SetColor(cBackgroundShade2);
      Line(x+1,y+1,x+xx-1,y+1);
      Line(x+xx-1,y+1,x+xx-1,y+yy-1);
```

```
        Line(x+xx-1,y+yy-1,x+xx-1-Bac                groundborder);
        Line(x+xx-1-BackgroundBorder,y+yy-1-Backgroundborder,x+xx-1-Backgroundborder,y+1+Backgroundborder);
        Line(x+xx-1-Backgroundborder,y+1+Backgroundborder,x+1+Backgroundborder,y+1+Backgroundborder);
        Line(x+1+Backgroundborder,y+1+Backgroundborder,x+1,Y+1);
        SetFillStyle(1,cBackgroundShade2);
        FloodFill(x+xx-2,y+2,cBackgroundShade2);

        ShowMouseCursor;

        ClearScreen(0);

    END;

Procedure ClearScreen(ReductionY : Byte);
    BEGIN
      HideMouseCursor;
      {LimitsOnOff;}
      SetViewPort(ScreenX, ScreenY + ReductionY, ScreenX + ScreenSizeX, ScreenY + ScreenSizeY - ReductionY, ClipOn);
      ClearViewPort;
      SetViewPort(0,0,GetMaxX,GetMaxY,ClipOn);
      SetLineStyle(DottedLn,0,NormWidth);
      SetColor(cEditWindowDisplay8);
      Line(ScreenX, (ScreenY + ScreenSizeY div 2) - (256 div 2 -  63),
         ScreenX + ScreenSizeX, (ScreenY + ScreenSizeY div 2) - (256 div 2 -  63));
      Line(ScreenX, (ScreenY + ScreenSizeY div 2) + (256 div 2 -  63),
         ScreenX + ScreenSizeX, (ScreenY + ScreenSizeY div 2) + (256 div 2 -  63));
      SetLineStyle(SolidLn,0,NormWidth);

      IF DemoMode THEN BEGIN
        SetColor(LightRed);
        SetTextJustify(CenterText, TopText);
        OutTextXY(ScreenX + ScreenSizeX div 2, ScreenY + 5, 'DEMO MODE');
      END;

      ShowMouseCursor;
    END;

procedure Error;
    BEGIN
      {GetScreen;}
      CASE IoError OF

        {DOS errors}
         2  : ErrorMessage := 'File not found.';
         3  : ErrorMessage := 'Path not found.';
         4  : ErrorMessage := 'Too many open files.';
         5  : ErrorMessage := 'File access denied.';
         6  : ErrorMessage := 'Invalid file handle.';
         8  : ErrorMessage := 'Not enough memory.';
        10  : ErrorMessage := 'Invalid environment.';
        11  : ErrorMessage := 'Invalid format.';
        12  : ErrorMessage := 'Invalid file access code';
        15  : ErrorMessage := 'Invalid drive number.';
        16  : ErrorMessage := 'Cannot remove current directory.';
        17  : ErrorMessage := 'Cannot rename across drives.';
        18  : ErrorMessage := 'No more files.';

        {I/O errors}
        100  : ErrorMessage := 'Disk read error.';
        101  : ErrorMessage := 'Disk write error.';
        102  : ErrorMessage := 'File not assigned.';
        103  : ErrorMessage := 'File not open.';
        104  : ErrorMessage := 'File not open for input.';
        105  : ErrorMessage := 'File not open for output.';
        106  : ErrorMessage := 'Invalid numeric format.';

        {Critical errors}
        150  : ErrorMessage := 'Disk is write-protected.';
        151  : ErrorMessage := 'Unknown unit.';
        152  : ErrorMessage := 'Drive not ready.';
        153  : ErrorMessage := 'Unknown command.';
        154  : ErrorMessage := 'CRC error in data.';
        155  : ErrorMessage := 'Bad drive request structure length';
        156  : ErrorMessage := 'Disk seek error.';
        157  : ErrorMessage := 'Unknown media type.';
        158  : ErrorMessage := 'Sector not found.';
        159  : ErrorMessage := 'Printer out of paper.';
        160  : ErrorMessage := 'Device write fault.';
```

```
            161  : ErrorMessage := 'Devic
            162  : ErrorMessage := 'Hardware failure.';

            {Fatal errors}
            200  : ErrorMessage := 'Division by zero.';
            201  : ErrorMessage := 'Range check error.';
            202  : ErrorMessage := 'Stack overflow error.';
            203  : ErrorMessage := 'Heap overflow error.';
            204  : ErrorMessage := 'Invalid pointer operation.';
            205  : ErrorMessage := 'Floating point overflow.';
            206  : ErrorMessage := 'Floating point underflow.';
            207  : ErrorMessage := 'Invalid floating point operation.';
            208  : ErrorMessage := 'Overlay manager not installed.';
            209  : ErrorMessage := 'Overlay file read error.';

          ELSE
                ErrorMessage := 'This IoError has not been defined.';
          END;
          MessageBox('ERROR!','', ErrorMessage,'', 'any mouse key = continue');
          {OutTextXY(220,46,ErrorMessage);
          REPEAT UNTIL (LeftMouseKeyPressed) OR (RightMouseKeyPressed)
                            OR (KeyPressed);
          InKey(Ch,fk,key);
          PutScreen;}
        END;

procedure MoveMouseTo(d1,d2:Integer);
    Var d3,d4:Integer;
    BEGIN
      d3:=MouseX;
      d4:=MouseY;
      REPEAT
      IF d3 < d1 THEN d3 := d3 + ((d1-d3) Div 16+1);
      IF d3 > d1 THEN d3 := d3 - ((d3-d1) Div 16+1);
      IF d4 < d2 THEN d4 := d4 + ((d2-d4) Div 32+1);
      IF d4 > d2 THEN d4 := d4 - ((d4-d2) Div 32+1);
      SetMouseCursorPos(d3,d4);
      HideMouseCursor;ShowMouseCursor;
      UNTIL (d3=d1) AND (d4=d2);
    END;

procedure ClearTextWindows;
        BEGIN
          HideMouseCursor;
          LimitsOnOff;
          SetColor(0);
          SetTextJustify(LeftText,TopText);
          OutTextXY(ScreenX + 10, ScreenY + 18,
'■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■');
SetTextJustify(LeftText,BottomText);
          OutTextXY(ScreenX + 10, ScreenY + ScreenSizeY - 17,

'■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■');
LimitsOnOff;
          ShowMouseCursor;
        END;

procedure _Directory;
        var
          DirInfo: SearchRec;
          Path:String;
        Label Exit;
        BEGIN
          Button (DirectoryX, DirectoryY, ButtonSizeX, ButtonSizeY, 'PATH', cPath, on);
          SetTextJustify(LeftText, BottomText);
          GetWholeScreen;
          SetViewPort(ScreenX, ScreenY, ScreenX +
                ScreenSizeX, ScreenY + ScreenSizeY, ClipOn);
          ClearViewPort;
          SetColor(cEditWindowDisplay5);
          {$i-};
          FindFirst(FileListFilter, Archive, DirInfo);
          IoError:=IoResult;
          {$i+};
          IF IoError <>0 THEN Error;
          ClearViewPort;
          OutTextXY(10,10,'DIRECTORY:');
          MoveTo(10, 30);
```

```
    WHILE DosError = 0 DO
    BEGIN
      IF GetX > 500 THEN MoveTo(10,GetY+10);
      OutText(DirInfo.Name);
      OutText('  ');
      FindNext(DirInfo);
      IF GetY >= {70}168 THEN
      BEGIN
        OutTextXY(0,ScreenSizeY,' Press any key for more...');
        REPEAT UNTIL (LeftMouseKeyPressed) OR (RightMouseKeyPressed);
        IF RightMouseKeyPressed THEN GOTO Exit;
        ClearViewPort;
        OutTextXY(10,10,'DIRECTORY:');
        MoveTo(10,30);
      END;
    END;
    SetViewPort(0,0,GetMaxX,GetMaxY,ClipOn);
    WriteBottomScreen(' Press any key to continue...', cEditWindowDisplay2);
    REPEAT UNTIL (KeyPressed) OR (LeftMouseKeyPressed)
                OR (RightMouseKeyPressed);
    Exit:
    SetViewPort(0,0,GetMaxX,GetMaxY,ClipOn);
    PutWholeScreen;
    Button (DirectoryX, DirectoryY, ButtonSizeX, ButtonSizeY, 'PATH', cPath, off);
  END;

Function DirFileName : String;
  Const
    DirOffsetX  = 10;
    DirOffsetY  = 40;
    DirSpacingX = 120;
    DirSpacingY = 15;
  var
    DirInfo: SearchRec;
    d1, dx, dy, xdx, xdy : Integer;
    WhatToLookFor, s : String;
    Wipe : Boolean;
  Label Exit;
  BEGIN
    Wipe := FALSE;
    SetTextJustify(LeftText, BottomText);
    GetWholeScreen;
    SetViewPort(ScreenX, ScreenY, ScreenX +
          ScreenSizeX, ScreenY + ScreenSizeY, ClipOn);
    ClearViewPort;
    SetColor(cEditWindowDisplay5);

    WhatToLookFor := FileListFilter;

    {$i-};
      FindFirst(WhatToLookFor, Archive, DirInfo);
      IoError:=IoResult;
    {$i+};
    IF IoError <>0 THEN Error;
    ClearViewPort;

    SetColor(cEditWindowDisplay4);
    OutTextXY(10, 15, 'DIRECTORY : ');
    GetDir(0, s);
    s := Concat(s, ' ', WhatToLookFor);
    OutTextXY(120, 15, s);


    WHILE DosError = 0 DO BEGIN
      SetColor(cEditWindowDisplay8);
      FOR g:= 1 TO DirRow DO BEGIN
        FOR f:= 1 TO DirCol DO BEGIN
          IF DosError = 0 THEN BEGIN
            DirArray[f, g] := DirInfo.Name;
            FindNext(DirInfo);
          END
          ELSE DirArray[f, g] := '--------.---';
          HideMouseCursor;
          OutTextXY((f - 1) * DirSpacingX + DirOffsetX, (g - 1) * DirSpacingY + DirOffsetY,
                DirArray[f, g]);
          ShowMouseCursor;
        END;
      END;
```

```
  IF DosError = 0 THEN BEGIN
    HideMouseCursor;
    SetColor(cEditWindowDisplay5);
    SetTextJustify(CenterText, CenterText);
    OutTextXY(ScreenX + ScreenSizeX div 2, ScreenSizeY - 15, ' CLICK HERE FOR MORE FILES');
    SetTextJustify(LeftText, BottomText);
    SetColor(cEditWindowDisplay8);
    Rectangle(5, ScreenSizeY - 5, ScreenSizeX - 5, ScreenSizeY - 5 - 20);
    ShowMouseCursor;
  END;

  LeftMouseKeyWasPressed  := FALSE;
  RightMouseKeyWasPressed := FALSE;

  REPEAT
     IF LeftMouseKeyPressed  THEN LeftMouseKeyWasPressed  := TRUE;
     IF RightMouseKeyPressed THEN RightMouseKeyWasPressed := TRUE;

     dx :=  MouseX;
     dy :=  MouseY - 5;
     dx :=  (dx - DirOffsetX - 10) div DirSpacingX;
     dy :=  (dy - DirOffsetY) div DirSpacingY;

     IF (xdx <> dx) OR (xdy <> dy) OR (NOT Wipe) THEN BEGIN

       IF Wipe THEN BEGIN
         HideMouseCursor;
         SetColor(cEditWindowDisplay8);
         OutTextXY(xdx * DirSpacingX + DirOffsetX, xdy * DirSpacingY + DirOffsetY, DirArray[xdx + 1,
               xdy + 1]);
         ShowMouseCursor;
         Wipe := FALSE;
       END;

       IF (dx >= 0) AND (dx < DirCol) AND (dy >= 0) AND (dy < DirRow) THEN BEGIN
         HideMouseCursor;
         SetColor(cEditWindowDisplay12);
         OutTextXY(dx * DirSpacingX + DirOffsetX, dy * DirSpacingY + DirOffsetY, DirArray[dx + 1,
               dy + 1]);
         ShowMouseCursor;
         Wipe := TRUE;
         xdx := dx;
         xdy := dy;
         DirFileName := DirArray[dx + 1, dy + 1];

       END
       ELSE DirFileName := '--------.---';
     END;

     IF RightMouseKeyWasPressed THEN BEGIN
       DirFileName := '--------.---';
       GOTO Exit;
     END;

  UNTIL (LeftMouseKeyWasPressed) OR (RightMouseKeyWasPressed);

  IF (MouseX > ScreenX + 5) AND (MouseX < ScreenX + ScreenSizeX - 5) AND
    (MouseY < ScreenY + ScreenSizeY - 5) AND (MouseY > ScreenY + ScreenSizeY - 5 - 20) THEN BEGIN
    SetColor(cEditWindowDisplay5);
    Rectangle(5, ScreenSizeY - 5, ScreenSizeX - 5, ScreenSizeY - 5 - 20);
    Delay(100);
    HideMouseCursor;
    ClearViewPort;
    SetColor(cEditWindowDisplay4);
    OutTextXY(10, 15,'DIRECTORY:');
    OutTextXY(120, 15, WhatToLookFor);
    ShowMouseCursor;
  END
  ELSE GOTO Exit;

END;

Exit:
SetViewPort(0,0,GetMaxX,GetMaxY,ClipOn);
HideMouseCursor;
PutWholeScreen;
ShowMouseCursor;
```

```
    END;

Procedure EpromIcon(Xoffset, Yoffset, Magnification : Word);
    Const PinWidth    = 2;
          PinLength   = 1;
          PinSpacing  = 2;
          EpromWidth  = 23;
          EpromLength = 66;
          EpromWindow = 15;
          EpromKey    = 1;
    Var m : word;
    BEGIN
      m := Magnification;
      HideMouseCursor;
      SetFillStyle(1, LightGray);
      FOR f := 0 to 15 DO
        Bar(Xoffset + f * (PinWidth * m) + f * (PinSpacing * m),
          Yoffset,
          Xoffset + f * (PinWidth * m) + f * (PinSpacing * m) + (PinWidth * m),
          Yoffset + (PinLength * m) * 2 + (EpromWidth * m));

      SetFillStyle(1, DarkGray);
      Bar(Xoffset - (PinWidth * m),
        YOffset + (PinLength * m),
        Xoffset - (PinWidth * m) + (EpromLength * m),
        Yoffset + (PinLength * m) + (EpromWidth * m));

      SetColor(LightGray);
      FOR f := 1 TO (EpromWindow * m) div 2 DO
      Circle(Xoffset - (PinWidth * m) + (EpromLength * m) div 2,
          Yoffset + (PinLength * m) + (EpromWidth * m) div 2,
          f);

      SetColor(Black);
      FOR f := 1 TO (EpromKey * m) DO
        Circle(Xoffset - (PinWidth * m),
            YOffset + (PinLength * m) + (EpromWidth * m) div 2,
            f);
      ShowMouseCursor;
    END;

Procedure ClearEpromIcon(Xoffset, YOffset, Magnification : Word);
    Const
      PinWidth    = 2;
      PinLength   = 1;
      PinSpacing  = 2;
      EpromWidth  = 23;
      EpromLength = 66;
      EpromWindow = 15;
      EpromKey    = 1;
    Var m : Word;
    BEGIN
      m := Magnification;
      HideMouseCursor;
      SetViewPort(Xoffset - (PinWidth * m),
            Yoffset,
            Xoffset - (PinWidth * m) + (EpromLength * m),
            YOffset + (EpromWidth * m) + (PinLength * m) * 2,
            ClipOn);
      ClearViewPort;
      SetViewPort(0, 0, GetMaxX, GetMaxY, ClipOn);
      ShowMouseCursor;
    END;
END.
```

Unit Disk;

```
(*****************************************************************************)
Interface
(*****************************************************************************)

Uses Variable, MousU, Disp, Graph, gKeybrd, CRT, iReq, DOS;

Procedure Save;
Procedure Load;

(*****************************************************************************)
Implementation
(*****************************************************************************)

procedure Save;
      var FileVar : FILE OF BYTE;
        start, stop, d2 : word;
        d1 : Byte;
        SaveFile, s : String;
        FileExists : Boolean;
        fSize, bSize, dFree : Real;
      Label Rename;
      BEGIN
        Button (SaveX, SaveY, ButtonSizeX, ButtonSizeY, 'SAVE', cSave, on);
        IF NOT DemoMode THEN DisableAdcInt;
        Rename:
        GetDir(0, s);
        s := Concat('Name of file to SAVE:  ', s);
        WriteBottomScreen(s , cEditWindowDisplay2);
        SetTextStyle(DefaultFont,HorizDir,1);
        SaveFile := '';
        d2 := Length(s) * 8 + ScreenX + 30;

        InputString(SaveFile, d2, ScreenY + ScreenSizeY - 17, {12}8, cEditWindowDisplay4, 0,
              [EscapeKey, CarriageReturn, RightMouseKey]);

        FOR d1 := Length(SaveFile) Downto 1 DO BEGIN
          IF (SaveFile[d1] = '\') OR (SaveFile[d1] = ':') THEN Delete(SaveFile, 1, d1);
        END;

        FOR d1 := 1 TO Length(SaveFile) DO BEGIN
          IF SaveFile[d1] = '.' THEN Delete(SaveFile, d1, Length(SaveFile) - d1 + 1);
        END;

        SaveFile := Concat({Path,} SaveFile, FileExtension);

        IF Key = CarriageReturn THEN BEGIN
          Assign(FileVar, SaveFile);

          {$i-};
            ReSet(FileVar);
            fSize := FileSize(FileVar);
            Close(FileVar);
          {$i+};
          IoError := IoResult;
          IF IoError = 0 THEN BEGIN
            {Close (FileVar);}
            FileExists := TRUE;
            GetDir(0, s);
            s := Concat (' Saving File ', s, ' ');
            s := Concat(s,  SaveFile);
            MessageBox('WARNING',
                  SaveFile,
                  'Already exists.',
                  '',
                  'left = overwrite \ right = rename');

            IF RightMouseKeyWasPressed THEN Goto Rename;
          END
          ELSE BEGIN
            FileExists := FALSE;
            fSize := 0;
          END;

          dFree := DiskFree(0);
          IF FileExists THEN dFree := dFree + fSize;
```

```
            bSize := (Limit2 - Limit1) * 100
            IF dFree > bSize THEN BEGIN
               {$i-};
                  ReWrite(FileVar);
                  IoError:=IoResult;
               {$i+};
               IF IoError <>0 THEN Error
               ELSE BEGIN
                  GetDir(0, s);
                  s := Concat (' Saving File ', s, ' ');
                  s := Concat(s,  SaveFile);
                  WriteTopScreen(s, cEditWindowDisplay4);
                  start := Limit1 * 100; stop := Limit2 * 100;
                  ScreenMarker := (start div 100 + ScreenX + 1);
                  ScreenMarkerOnOff;
                  {$i-}
                     REPEAT
                        d1 := d^[start];
                        Write(FileVar, d1);
                        IoError := IoResult;
                        DrawScreenMarker(start div 100 + ScreenX);
                        INC(start);
                     UNTIL (start >= stop) OR (IoError <> 0);
                  {$i+}
                  ScreenMarkerOnOff;

                  Close (FileVar);

                  IF IoError <> 0 THEN Error;

                  FileToBeSaved := FALSE;
               END;
            END
            ELSE BEGIN
               GetDir(0 ,s);
               MessageBox('WARNING',
                     'Not enough space in path',
                     s,
                     'to hold message.',
                     'Any mouse key = continue');
            END;
         END;
         ClearTextWindows;
         IF Not DemoMode THEN EnableAdcInt;
         SetViewPort(0, 0, GetMaxX, GetMaxY, ClipOn);
         Button (SaveX, SaveY, ButtonSizeX, ButtonSizeY, 'SAVE', cSave, off);
      END;

procedure Load;
      var FileVar : FILE OF BYTE;
         a,d1,d2 : word;
         LoadFile, s : String;
         d0 : Byte;
      BEGIN
         Button (LoadX, LoadY, ButtonSizeX, ButtonSizeY, 'LOAD', cSave, on);
         IF NOT DemoMode THEN DisableAdcInt;
         IF FileToBeSaved THEN BEGIN
           GetWholeScreen;
           MessageBox('WARNING!',
                 '',
                 'edits will be lost',
                 '',
                 'any mouse key = continue');
           IF LeftMouseKeyWasPressed THEN FileToBeSaved := FALSE;
           PutWholeScreen;
         END;

         IF NOT FileToBeSaved THEN BEGIN
           d2 := 0;
           LoadFile := DirFileName;

           IF LoadFile <> '--------.---' THEN BEGIN
             LimitsOnOff;
             Assign(FileVar,LoadFile);
             {$i-};
                ReSet(FileVar);
                IoError := IoResult;
             {$i+};
```

```
            IF IoError <>0 THEN Error
            ELSE BEGIN
               SetMouseCursorPos(LoadX + ButtonSizeX div 2, LoadY + ButtonSizeY div 2);
               KeepMouseOffEditWindow;
               ClearScreen({4}0);
               WriteTopScreen(' Loading File .....', cEditWindowDisplay4 + blink);
               GetDir(0, s);
               s := Concat(s, '', LoadFile);
               WriteBottomScreen ({LoadFile}s, cEditWindowDisplay5);
               d2 := 0;

               d1 := ScreenY + ScreenSizeY div 2 - 64;
               a := 1;

               RightMouseKeyWasPressed := FALSE;

               FillChar(d^, SizeOf(d^), 127);
               {$i-}
                  WHILE (NOT Eof(FileVar)) AND (IoError = 0) AND (NOT RightMouseKeyWasPressed) DO BEGIN
                     FOR f := 1 To 4 DO BEGIN
                        INC(a);
                        IF IoError = 0 THEN BEGIN
                          Read(FileVar, d0);
                          IoError := IoResult;
                        END;
                        d^[a] := d0;
                     END;
                     INC(d2);
                     PutPixel(d2 div 25 + ScreenX, (d^[a] div 2 + d1), cEditWindowDisplay5);
                     IF RightMouseKeyPressed THEN RightMouseKeyWasPressed := TRUE;
                  END;
               {$i+}

               Close(FileVar);

               IF IoError <> 0 THEN Error;

               SL1    := ScreenX + 1;
               Limit1 := SL1 - ScreenX;
               SL2    := ScreenX + a div 100;
               Limit2 := SL2 - ScreenX;

               SetColor(cEditWindowDisplay5);
               Line(d2 div 25 + ScreenX, ScreenY + ScreenSizeY div 2, RecordBufferSize div 100 + ScreenX,
                  ScreenY + ScreenSizeY div 2);

               FreeMouse;
            END;
            LimitsOnOff;
          END ELSE FileToBeSaved := TRUE;
        END;
        ClearTextWindows;
        IF NOT DemoMode THEN EnableAdcInt;
        SetViewPort(0, 0, GetMaxX, GetMaxY, ClipOn);
        Button (LoadX, LoadY, ButtonSizeX, ButtonSizeY, 'LOAD', cSave, off);
      END;
END.
```

Unit EPROM;

```
(*************************************************************************)
Interface
(*************************************************************************)
Uses CRT, Variable;

Procedure Pause;
Procedure EpromVppOn;
Procedure EpromVppOff;
Procedure ResetEprom;
Procedure ClockEprom;
Procedure PgmEprom;
Procedure EpromOutputEnable;
Procedure EpromOutputDisable;
Function ReadEprom : Byte;
Procedure WriteEprom(x:byte);
Procedure VerifyEpromEmpty;
Procedure ProgramEprom(MessageNumber : Byte);
Procedure List;

(*************************************************************************)
Implementation
(*************************************************************************)

Procedure Pause;
      BEGIN
         WriteLn('press any key to continue');
         REPEAT UNTIL KeyPressed;
         ch:=ReadKey;
      END;

Procedure SetPortC_Bit (d1:Byte);
      BEGIN
         Port[$30B] := (d1*2+1);
      END;

Procedure ResetPortC_Bit (d1:Byte);
      BEGIN
         Port[$30B] := (d1*2+0);
      END;

Procedure EpromVppOn;
      Var d1 : Byte;
      BEGIN
         SetPortC_Bit(4);
      END;

Procedure EpromVppOff;
      Var d1 : Byte;
      BEGIN
         ResetPortC_Bit(4);
      END;

Procedure ResetEprom;
      Var d1 : Byte;
      BEGIN
         SetPortC_Bit(1);
         {Delay(1);}
         ResetPortC_Bit(1);
         EpromCount:=0;
      END;

Procedure ClockEprom;
      var d1 : integer;
      BEGIN
         SetPortC_Bit(0);
         {Delay(1);}
         ResetPortC_Bit(0);
         Inc(EpromCount);
      END;

Procedure PgmEprom;
     Var d1 : Byte;
     BEGIN
         SetPortC_Bit(2);
         {Delay(1);}
```

```
        ResetPortC_Bit(2);
    END;

Procedure EpromOutputEnable;
    Var d1 : Byte;
    BEGIN
        SetPortC_Bit(3);
    END;

Procedure EpromOutputDisable;
    Var d1 : Byte;
    BEGIN
        ResetPortC_Bit(3);
    END;

Function ReadEprom : Byte;
    BEGIN
        Port[$30B]:= 144;   {setup for b, c out, a in}
        EpromOutputEnable;
        ReadEprom := Port[$308];
        EpromOutputDisable;
    END;

Procedure WriteEprom(x:byte);
    Var d1 : Integer;
      check:byte;
    BEGIN
        check:=ReadEprom;
        d1 := 0;
        EpromOutputDisable;
        REPEAT
            {Port[$30B] := 128 + 2;} {port a&c out}
            Port[$30B] := 128; {port a,b&c out}
            EpromVppOn;
            Port[$308] := x;
            PgmEprom;
            Port[$308] := $FF;
            check:=ReadEprom;
            EpromVppOn;
            Inc(d1);
            {GotoXY(1,12);WriteLn(x,' ',d1,' ',f);}
        UNTIL (check = x) OR (d1 > 100);
        IF d1 > 100 THEN EpromError:=1;
    END;

Procedure VerifyEpromEmpty;
    BEGIN
        ResetEprom;
        EpromOutputEnable;
        f:=1;
        EpromEmpty := TRUE;
        REPEAT
            GotoXY(1,4);WriteLn('checking : ',f,' ',ReadEprom);
            IF ReadEprom <> $FF THEN BEGIN GotoXY(1,5);WriteLn('EPROM not EMPTY  :',f);
                                    EpromEmpty := False
                                END;
            inc(f);
            ClockEprom;
        UNTIL (f = 1280) ;{OR (NOT EpromEmpty);}
        IF EpromEmpty THEN BEGIN GotoXY(1,5);WriteLn('EPROM EMPTY         ');END;
        EpromOutputDisable;
    END;

Procedure ProgramEprom(MessageNumber : Byte);
    BEGIN;
        EpromVppOn;
        WriteEprom(0);
        ClockEprom;
        WriteEprom(MessageNumber);
        ClockEprom;

        FOR f:= 1 TO Limit2 DO
        BEGIN
            IF d^[f]=$FF THEN d^[f] := $FE;
            WriteEprom(d^[f]);
            ClockEprom;
        END;
    WHILE EpromCount<>((EpromCount div 128)*128)  DO
```

```
            BEGIN
               WriteEprom(127);
               ClockEprom;
            END;
            EpromVppOff;
         END;

Procedure List;
         var   d1 : integer;
         BEGIN
            f:=0;
            ResetEprom;
            REPEAT
               IF (ReadEprom=0) THEN BEGIN
                                  WriteLn(f,' ',ReadEprom);
                                  INC(f);
                                  ClockEprom;
                                  WriteLn(f,' ',ReadEprom);
                                  INC(f);
                                  ClockEprom;
                                  WriteLn(f,' ',ReadEprom);
                               END;
               ClockEprom;
               INC(f);
            UNTIL f=70000;
            WriteLn('listing finished');
            ReadLn;
         END;

END.
```

Unit iReq;

```
(*************************************************************************)
Interface
(*************************************************************************)

Uses Variable, DOS, crt;

Procedure AdcInt; Interrupt;
Procedure EnableAdcInt;
Procedure DisableAdcInt;
Procedure InstallAdcInt;
Procedure UnInstallAdcInt;

(*************************************************************************)
Implementation
(*************************************************************************)

Procedure AdcInt;
      BEGIN
         Inline($FB);     {STI}

         CASE TalkStat OF
            Rec : BEGIN
                     LastRecByte := Port[AdcPort];
                     d^[dPointer] := LastRecByte;
                     Inc(dPointer);
                     IF dPointer >= RecordBufferSize THEN TalkStat := Idle;
                     IF Monitor = On THEN Port[DacPort] := LastRecByte;
                  END;
            Play: BEGIN
                     IF (dPointer <= RecordBufferSize) THEN
                       Port[DacPort]:= d^[dPointer];
                     Inc(dPointer);
                     IF (dPointer >= dPointerStop)
                       OR (dPOinter >= RecordBufferSize)
                       THEN TalkStat := Idle;
                  END;
            Idle: IF Monitor = On THEN Port[DacPort] := Port[AdcPort];

         END;

         {DemoMode := False;}
         InLine($FA);     {CLI}
         Port[$20] := $20;
      END;

Procedure EnableAdcInt;
      Var d1:Byte;
      BEGIN
         d1 := Port[IntCtlrAddr];
         d1 := d1 AND IntEnableNo;
         Port[IntCtlrAddr] := d1;
         Port[$20] := $20;
      END;

Procedure DisableAdcInt;
      Var d1:Byte;
      BEGIN
         d1 := Port[IntCtlrAddr];
         d1 := d1 OR IntDisablNo;
         Port[IntCtlrAddr] := d1;
         Port[$20] := $20;
      END;

Procedure InstallAdcInt;
      Var d1:Byte;
      BEGIN
         d1 := InterruptNo + 8;
         GetIntVec(d1,OldVector);
         SetIntVec(d1,@AdcInt);
         EnableAdcInt;
      END;

Procedure UnInstallAdcInt;
      Var d1 : Byte;
      BEGIN
```

```
    d1 := InterruptNo + 8;
    DisableAdcInt;
    SetIntVec(d1,OldVector);
  END;

END.
```

Unit Helper;

```
(*************************************************************************)
Interface
(*************************************************************************)

Uses Variable, Disp, Graph, MousU, Gkeybrd;

procedure  Help;

(*************************************************************************)
Implementation
(*************************************************************************)

Procedure CheckMouseKeys;
      BEGIN
         LeftMouseKeyWasPressed  := FALSE;
         RightMouseKeyWasPressed := FALSE;
         REPEAT UNTIL NOT AnyMouseKeyPressed;
         REPEAT
            IF LeftMouseKeyPressed  THEN LeftMouseKeyWasPressed  := TRUE;
            IF RightMouseKeyPressed THEN RightMouseKeyWasPressed := TRUE;
         UNTIL (LeftMouseKeyWasPressed) OR (RightMouseKeyWasPressed);
      END;


Procedure HelpEditWindow;
      BEGIN
         ClearScreen(0);
         {LimitsOnOff;}
         SetColor(cEditWindowDisplay4);
         SetTextStyle(DefaultFont, HorizDir, 2);
         WriteTopScreen('HELP: Edit Window',cEditWindowDisplay5);
         SetColor(cEditWindowDisplay3);
         SetTextStyle(SmallFont,HorizDir,4);
         HideMouseCursor;

OutTextXY(30,  62, 'All messages, whether recorded or loaded from disk or EPROM, are displayed as a waveform in the    ');
OutTextXY(30,  72, 'Edit window.                                                                                       ');
OutTextXY(30,  82, '                                                                                                   ');
OutTextXY(30,  92, 'The maximum and minimum values possible for a waveform is indicated by two grey dotted lines. When ');
OutTextXY(30, 102, 'recording, it is important to try and keep the amplitude of the input signal at such a level that  ');
OutTextXY(30, 112, 'the signal peaks touch the grey lines without exceeding it. This minimises Quantization noise.    ');
OutTextXY(30, 122, 'Excessively exceeding the grey lines leed to a type of distortion called clipping.                ');
OutTextXY(30, 132, '                                                                                                   ');
OutTextXY(30, 142, 'Specific parts of the waveform may be marked for Copying, Moving, Deleting or Saving. This is done ');
OutTextXY(30, 152, 'by means of Limit markers 1 and 2. Limit1 indicates the start of a signal block and Limit2 the     ');
OutTextXY(30, 162, 'end. Limit1 is positioned by moving the mouse cursor to the desired position and then pressing the ');
OutTextXY(30, 172, 'left mouse key. Limit2 is moved in a similar manner by pressing the right mouse key.              ');

         ShowMouseCursor;
         SetTextStyle(DefaultFont, HorizDir, 1);
         WriteBottomScreen('Press the right mouse key to return to the Help base page.', cEditWindowDisplay8);
         CheckMouseKeys;
      END;

Procedure HelpRecord;
      BEGIN
         ClearScreen(0);
         {LimitsOnOff;}
         SetColor(cEditWindowDisplay4);
         SetTextStyle(DefaultFont, HorizDir, 2);
         WriteTopScreen('HELP: Record', cEditWindowDisplay5);
         SetColor(cEditWindowDisplay3);
         SetTextStyle(SmallFont,HorizDir,4);
         HideMouseCursor;

OutTextXY(30,  62, 'This function allows the user to record new messages into the Record buffer by either using a     ');
OutTextXY(30,  72, 'mirophone or directly by using the auxiliary input.                                               ');
OutTextXY(30,  82, '                                                                                                  ');
OutTextXY(30,  92, 'Record procedure:                                                                                 ');
OutTextXY(30, 102, ' 1) Press the Record button. The recording level is controlled by the level potentiometer.       ');
OutTextXY(30, 112, ' 2) Recording is delayed until the input signal exceeds the minimum level defined by the Trigger  ');
OutTextXY(30, 122, '    potentiometer. This function may be disabled by setting Trigger to its minimum level.        ');
OutTextXY(30, 132, '                                                                                                  ');
OutTextXY(30, 142, 'Note: It is not possible to adjust potentiometers during the Recording process.                  ');
OutTextXY(30, 152, '                                                                                                  ');
```

```
IF DemoMode THEN BEGIN
  SetColor(cEditWindowDisplay4);
  OutTextXY(30, 162, 'In Demo mode a simulation of a recorded signal is generated for demonstration puposes. This       ');
  OutTextXY(30, 172, 'simulated signal is stored in the Record buffer and it is thus possible to Edit and Save.        ');
END;

        ShowMouseCursor;
        SetTextStyle(DefaultFont, HorizDir, 1);
        WriteBottomScreen('Press the right mouse key to return to the Help base page.', cEditWindowDisplay8);
        CheckMouseKeys;
      END;

Procedure HelpPlay;
      BEGIN
        ClearScreen(0);
        {LimitsOnOff;}
        SetColor(cEditWindowDisplay4);
        SetTextStyle(DefaultFont, HorizDir, 2);
        WriteTopScreen('HELP: Play', cEditWindowDisplay5);
        SetColor(cEditWindowDisplay3);
        SetTextStyle(SmallFont,HorizDir,4);
        HideMouseCursor;

OutTextXY(30,  62, 'This function plays back the waveform segment between Limit1 and Limit2.                        ');
OutTextXY(30,  72, '                                                                                                ');
OutTextXY(30,  82, 'Play procedure:                                                                                 ');
OutTextXY(30,  92, ' 1) Press the Play button. The playback level is controlled by means of the volume potentiometer. ');
OutTextXY(30, 102, '                                                                                                ');
OutTextXY(30, 112, '                                                                                                ');
OutTextXY(30, 122, 'Note: It is not possible to adjust potentiometers during the Playback process.                  ');
OutTextXY(30, 132, '                                                                                                ');
OutTextXY(30, 142, '                                                                                                ');
OutTextXY(30, 152, '                                                                                                ');
IF DemoMode THEN BEGIN
  SetColor(cEditWindowDisplay4);
  OutTextXY(30, 162, 'Pressing the Play button in Demo mode has no real effect.                                     ');
  OutTextXY(30, 172, '                                                                                              ');
END;

        ShowMouseCursor;
        SetTextStyle(DefaultFont, HorizDir, 1);
        WriteBottomScreen('Press the right mouse key to return to the Help base page.', cEditWindowDisplay8);
        CheckMouseKeys;
      END;

Procedure HelpMute;
      BEGIN
        ClearScreen(0);
        {LimitsOnOff;}
        SetColor(cEditWindowDisplay4);
        SetTextStyle(DefaultFont, HorizDir, 2);
        WriteTopScreen('HELP: Mute / Listen', cEditWindowDisplay5);
        SetColor(cEditWindowDisplay3);
        SetTextStyle(SmallFont,HorizDir,4);
        HideMouseCursor;

OutTextXY(30,  62, 'When in use, the system continuously convert the input signal to digital and back to analog in    ');
OutTextXY(30,  72, 'realtime. Monitoring the input signal this way, the user gets an indication of what influence the ');
OutTextXY(30,  82, 'system would have on the input signal, once recorded.                                            ');
OutTextXY(30,  92, '                                                                                                ');
OutTextXY(30, 102, 'This function may be disabled by pressing the Listen / Mute button, toggling between Listen and  ');
OutTextXY(30, 112, 'Mute.                                                                                           ');
OutTextXY(30, 122, '                                                                                                ');
OutTextXY(30, 132, '(When monitoring the input with a loudspeaker rather than headphones, undesireable feedback may  ');
OutTextXY(30, 142, ' result. This can be avoided by Muting the system.)                                             ');
OutTextXY(30, 152, '                                                                                                ');
IF DemoMode THEN BEGIN
  SetColor(cEditWindowDisplay4);
  OutTextXY(30, 162, 'Pressing the Mute / Listen button in Demo mode has no effect.                                 ');
  OutTextXY(30, 172, '                                                                                              ');
END;

        ShowMouseCursor;
        SetTextStyle(DefaultFont, HorizDir, 1);
        WriteBottomScreen('Press the right mouse key to return to the Help base page.', cEditWindowDisplay8);
        CheckMouseKeys;
      END;
```

```
Procedure HelpTrigger;
      BEGIN
         ClearScreen(0);
         {LimitsOnOff;}
         SetColor(cEditWindowDisplay4);
         SetTextStyle(DefaultFont, HorizDir, 2);
         WriteTopScreen('HELP: Trigger', cEditWindowDisplay5);
         SetColor(cEditWindowDisplay3);
         SetTextStyle(SmallFont,HorizDir,4);
         HideMouseCursor;

OutTextXY(30,  62, 'The Tigger potentiometer is used to set a level which the input signal must exceed before recording');
OutTextXY(30,  72, 'commences. This function is overridden by setting the Trigger poteniometer to its minimum level.  ');
OutTextXY(30,  82, '                                                                                                   ');
OutTextXY(30,  92, 'The Trigger function is achieved by software.                                                      ');
OutTextXY(30, 102, '                                                                                                   ');
OutTextXY(30, 112, 'The potentiometer is set by clicking the left mouse key with the mouse cursor at the desired      ');
OutTextXY(30, 122, 'poteniometer position.                                                                             ');
OutTextXY(30, 132, '                                                                                                   ');
OutTextXY(30, 142, '                                                                                                   ');
OutTextXY(30, 152, '                                                                                                   ');
IF DemoMode THEN BEGIN
  SetColor(cEditWindowDisplay4);
  OutTextXY(30, 162, 'The Trigger potentiometer has no effect when in demo mode.                                       ');
  OutTextXY(30, 172, '                                                                                                 ');
END;

         ShowMouseCursor;
         SetTextStyle(DefaultFont, HorizDir, 1);
         WriteBottomScreen('Press the right mouse key to return to the Help base page.', cEditWindowDisplay8);
         CheckMouseKeys;
      END;

Procedure HelpLevel;
      BEGIN
         ClearScreen(0);
         {LimitsOnOff;}
         SetColor(cEditWindowDisplay4);
         SetTextStyle(DefaultFont, HorizDir, 2);
         WriteTopScreen('HELP: Level', cEditWindowDisplay5);
         SetColor(cEditWindowDisplay3);
         SetTextStyle(SmallFont,HorizDir,4);
         HideMouseCursor;

OutTextXY(30,  62, 'The Level potentiometer is used to control the level of the input signal when recording.         ');
OutTextXY(30,  72, '                                                                                                 ');
OutTextXY(30,  82, 'The Level function is achieved by the software controlling an eight-step programmable attenuator ');
OutTextXY(30,  92, 'in the Record/Playback/Programmer module.                                                        ');
OutTextXY(30, 102, '                                                                                                 ');
OutTextXY(30, 112, 'The potentiometer is set by clicking the left mouse key with the mouse cursor at the desired    ');
OutTextXY(30, 122, 'potentiometer position.                                                                          ');
OutTextXY(30, 132, '                                                                                                 ');
OutTextXY(30, 142, '                                                                                                 ');
OutTextXY(30, 152, '                                                                                                 ');
IF DemoMode THEN BEGIN
  SetColor(cEditWindowDisplay4);
  OutTextXY(30, 162, 'The Level potentiometer has no effect when in demo mode.                                       ');
  OutTextXY(30, 172, '                                                                                                ');
END;

         ShowMouseCursor;
         SetTextStyle(DefaultFont, HorizDir, 1);
         WriteBottomScreen('Press the right mouse key to return to the Help base page.', cEditWindowDisplay8);
         CheckMouseKeys;
      END;

Procedure HelpCopy;
      BEGIN
         ClearScreen(0);
         {LimitsOnOff;}
         SetColor(cEditWindowDisplay4);
         SetTextStyle(DefaultFont, HorizDir, 2);
         WriteTopScreen('HELP: Copy', cEditWindowDisplay5);
         SetColor(cEditWindowDisplay3);
         SetTextStyle(SmallFont,HorizDir,4);
         HideMouseCursor;

OutTextXY(30,  62, 'This function allows the user to make a copy of the waveform segment between Limit1 and Limit2.  ');
```

```
OutTextXY(30,  72, '                                                                    ');
OutTextXY(30,  82, 'Copy procedure:                                                     ');
OutTextXY(30,  92, ' 1) By pressing the Copy button, the signal segment between Limit1 and Limit2 is highlighted.    ');
OutTextXY(30, 102, ' 2) Place the mouse cursor over the marked signal segment.          ');
OutTextXY(30, 112, ' 3) While holding down the left mouse key, drag the copy to a new location by moving the mouse.  ');
OutTextXY(30, 122, ' 4) The copy is placed as soon as the left mouse key is released.   ');
OutTextXY(30, 132, '                                                                    ');
OutTextXY(30, 142, 'A corresponding copy of data is made within the Record buffer.      ');
OutTextXY(30, 152, '                                                                    ');
OutTextXY(30, 162, 'The Copy routine is aborted by pressing the right mouse key.        ');
OutTextXY(30, 172, '                                                                    ');

        ShowMouseCursor;
        SetTextStyle(DefaultFont, HorizDir, 1);
        WriteBottomScreen('Press the right mouse key to return to the Help base page.', cEditWindowDisplay8);
        CheckMouseKeys;
    END;

Procedure HelpMove;
    BEGIN
        ClearScreen(0);
        {LimitsOnOff;}
        SetColor(cEditWindowDisplay4);
        SetTextStyle(DefaultFont, HorizDir, 2);
        WriteTopScreen('HELP: Move', cEditWindowDisplay5);
        SetColor(cEditWindowDisplay3);
        SetTextStyle(SmallFont,HorizDir,4);
        HideMouseCursor;

OutTextXY(30,  62, 'This function allows the user to move the waveform segment between Limit1 and Limit2 to a new  ');
OutTextXY(30,  72, 'location.                                                           ');
OutTextXY(30,  82, '                                                                    ');
OutTextXY(30,  92, 'Move procedure:                                                     ');
OutTextXY(30, 102, ' 1) By pressing the Move button, the signal segment between Limit1 and Limit2 is highlighted.    ');
OutTextXY(30, 112, ' 2) Place the mouse cursor over the marked signal part.             ');
OutTextXY(30, 122, ' 3) While holding down the left key, move the signal part to a new location by moving the mouse.  ');
OutTextXY(30, 132, ' 4) The move is completed as soon as the left mouse key is released.  ');
OutTextXY(30, 142, '                                                                    ');
OutTextXY(30, 152, 'A corresponding move of data is made within the Record buffer.      ');
OutTextXY(30, 162, '                                                                    ');
OutTextXY(30, 172, 'The Move routine is aborted by pressing the right mouse key.        ');

        ShowMouseCursor;
        SetTextStyle(DefaultFont, HorizDir, 1);
        WriteBottomScreen('Press the right mouse key to return to the Help base page.', cEditWindowDisplay8);
        CheckMouseKeys;
    END;

Procedure HelpDelete;
    BEGIN
        ClearScreen(0);
        {LimitsOnOff;}
        SetColor(cEditWindowDisplay4);
        SetTextStyle(DefaultFont, HorizDir, 2);
        WriteTopScreen('HELP: Delete', cEditWindowDisplay5);
        SetColor(cEditWindowDisplay3);
        SetTextStyle(SmallFont,HorizDir,4);
        HideMouseCursor;

OutTextXY(30,  62, 'This function allows the user to delete the waveform segment between Limit1 and Limit2.    ');
OutTextXY(30,  72, '                                                                    ');
OutTextXY(30,  82, 'Delete procedure:                                                   ');
OutTextXY(30,  92, ' 1) By pressing the Delete button, the signal segment between Limit1 and Limit2 is highlighted.  ');
OutTextXY(30, 102, ' 2) The user is prompted again for confirmation.                    ');
OutTextXY(30, 112, ' 3) Press the left mouse key to complete the delete routine.        ');
OutTextXY(30, 122, '                                                                    ');
OutTextXY(30, 132, 'A corresponding deletion of data is carried out within the Record buffer.  ');
OutTextXY(30, 142, '                                                                    ');
OutTextXY(30, 152, 'The Delete routine is aborted by pressing the right mouse key.      ');

        ShowMouseCursor;
        SetTextStyle(DefaultFont, HorizDir, 1);
        WriteBottomScreen('Press the right mouse key to return to the Help base page.', cEditWindowDisplay8);
        CheckMouseKeys;
    END;

Procedure HelpLoad;
    BEGIN
```

```
        ClearScreen(0);
        {LimitsOnOff;}
        SetColor(cEditWindowDisplay4);
        SetTextStyle(DefaultFont, HorizDir, 2);
        WriteTopScreen('HELP: Load', cEditWindowDisplay5);
        SetColor(cEditWindowDisplay3);
        SetTextStyle(SmallFont,HorizDir,4);
        HideMouseCursor;

OutTextXY(30,  62, 'This function allows the user to Load messages previously saved to disk by means of the Save      ');
OutTextXY(30,  72, 'command.                                                                                          ');
OutTextXY(30,  82, '                                                                                                  ');
OutTextXY(30,  92, 'Load procedure:                                                                                   ');
OutTextXY(30, 102, ' 1) By pressing the Load button, all files in the current directory with a TLK extension is listed.');
OutTextXY(30, 112, ' 2) The user selects a file by highliting the specific file by means of the mouse cursor.        ');
OutTextXY(30, 122, ' 3) The user confirms his choice by pressing the left mouse key.                                  ');
OutTextXY(30, 132, ' 4) The file is loaded into the Record Buffer and is displayed as a waveform in the Edit window.  ');
OutTextXY(30, 142, '                                                                                                  ');
OutTextXY(30, 152, 'The Load routine is aborted by pressing the right mouse key.                                      ');

        ShowMouseCursor;
        SetTextStyle(DefaultFont, HorizDir, 1);
        WriteBottomScreen('Press the right mouse key to return to the Help base page.', cEditWindowDisplay8);
        CheckMouseKeys;
    END;

Procedure HelpSave;
    BEGIN
        ClearScreen(0);
        {LimitsOnOff;}
        SetColor(cEditWindowDisplay4);
        SetTextStyle(DefaultFont, HorizDir, 2);
        WriteTopScreen('HELP: Save', cEditWindowDisplay5);
        SetColor(cEditWindowDisplay3);
        SetTextStyle(SmallFont,HorizDir,4);
        HideMouseCursor;

OutTextXY(30,  62, 'This function allows the user to Save messages to disk.                                          ');
OutTextXY(30,  72, '                                                                                                  ');
OutTextXY(30,  82, 'Save procedure:                                                                                   ');
OutTextXY(30,  92, ' 1) Press the Save button. The user is prompted for a filename.                                   ');
OutTextXY(30, 102, ' 2) By entering a valid file name, the signal segment between Limit1 and Limit2 is saved to disk. ');
OutTextXY(30, 112, '    function.                                                                                     ');
OutTextXY(30, 122, '                                                                                                  ');
OutTextXY(30, 132, 'Note: The file is saved to the path entered by the Path function, with a TLK extension. If the    ');
OutTextXY(30, 142, '      entered file name contains another path or extension, it is replaced with the above.        ');
OutTextXY(30, 152, '                                                                                                  ');
OutTextXY(30, 162, 'The Save routine is aborted by pressing the right mouse key.                                      ');

        ShowMouseCursor;
        SetTextStyle(DefaultFont, HorizDir, 1);
        WriteBottomScreen('Press the right mouse key to return to the Help base page.', cEditWindowDisplay8);
        CheckMouseKeys;
    END;

Procedure HelpPath;
    BEGIN
        ClearScreen(0);
        {LimitsOnOff;}
        SetColor(cEditWindowDisplay4);
        SetTextStyle(DefaultFont, HorizDir, 2);
        WriteTopScreen('HELP: Path', cEditWindowDisplay5);
        SetColor(cEditWindowDisplay3);
        SetTextStyle(SmallFont,HorizDir,4);
        HideMouseCursor;

OutTextXY(30,  62, 'By pressing the Path button, the user is prompted for a path to which future files will be Saved  ');
OutTextXY(30,  72, 'to and Loaded from.                                                                               ');
OutTextXY(30,  82, '                                                                                                  ');
OutTextXY(30,  92, 'The Path routine is aborted by pressing the right mouse key.                                      ');
OutTextXY(30, 102, '                                                                                                  ');
OutTextXY(30, 112, '                                                                                                  ');
OutTextXY(30, 122, '                                                                                                  ');
OutTextXY(30, 132, '                                                                                                  ');
OutTextXY(30, 142, '                                                                                                  ');
OutTextXY(30, 152, '                                                                                                  ');
OutTextXY(30, 162, '                                                                                                  ');
```

```
        ShowMouseCursor;
        SetTextStyle(DefaultFont, HorizDir, 1);
        WriteBottomScreen('Press the right mouse key to return to the Help base page.', cEditWindowDisplay8);
        CheckMouseKeys;
    END;

Procedure HelpWriteE;
    BEGIN
        ClearScreen(0);
        {LimitsOnOff;}
        SetColor(cEditWindowDisplay4);
        SetTextStyle(DefaultFont, HorizDir, 2);
        WriteTopScreen('HELP: WriteE', cEditWindowDisplay5);
        SetColor(cEditWindowDisplay3);
        SetTextStyle(SmallFont,HorizDir,4);
        HideMouseCursor;

OutTextXY(30,  62, 'This function allows the user to write the waveform segment between Limit1 and Limit2 into an     ');
OutTextXY(30,  72, 'EPROM, placed in the EPROM programmer socket integral to the Record/Playback/Programmer module.   ');
OutTextXY(30,  82, '                                                                                                  ');
OutTextXY(30,  92, 'WriteE procedure:                                                                                 ');
OutTextXY(30, 102, ' 1) Press the WriteE button. A reduced size version of the signal segment between Limit1 and Limit2');
OutTextXY(30, 112, '    is displayed. The routine starts looking for enough space to hold the message inside the EPROM.');
OutTextXY(30, 122, ' 2) As soon as enough space is found, the user is prompted for an ID number for the message.      ');
OutTextXY(30, 132, ' 3) The signal segment between Limit1 and Limit2 is written together with the ID no. into the EPROM.');
OutTextXY(30, 142, '                                                                                                  ');
OutTextXY(30, 152, 'The WriteE routine is aborted by pressing the right mouse key.                                    ');
OutTextXY(30, 162, '                                                                                                  ');
IF DemoMode THEN BEGIN
  SetColor(cEditWindowDisplay4);
  OutTextXY(30, 172, 'The WriteE function is not available in Demo mode.                                             ');
END;

        ShowMouseCursor;
        SetTextStyle(DefaultFont, HorizDir, 1);
        WriteBottomScreen('Press the right mouse key to return to the Help base page.', cEditWindowDisplay8);
        CheckMouseKeys;
    END;

Procedure HelpReadE;
    BEGIN
        ClearScreen(0);
        {LimitsOnOff;}
        SetColor(cEditWindowDisplay4);
        SetTextStyle(DefaultFont, HorizDir, 2);
        WriteTopScreen('HELP: ReadE',cEditWindowDisplay5);
        SetColor(cEditWindowDisplay3);
        SetTextStyle(SmallFont,HorizDir,4);
        HideMouseCursor;

OutTextXY(30,  62, 'This function allows the user to read messages, stored in an EPROM iserted into the EPROM       ');
OutTextXY(30,  72, 'programmer socket integral to the Record/Playback/Programmer module, back into the Record buffer. ');
OutTextXY(30,  82, '                                                                                                  ');
OutTextXY(30,  92, 'ReadE procedure:                                                                                  ');
OutTextXY(30, 102, ' 1) Press the ReadE button. The user is prompted for the ID number of the requested message.     ');
OutTextXY(30, 112, ' 2) The routine searches for the specific message inside the EPROM.                               ');
OutTextXY(30, 122, ' 3) As soon as the message is found, it is loaded into the Record buffer, while being displayed as ');
OutTextXY(30, 132, '    a waveform in the Edit window.                                                                ');
OutTextXY(30, 142, '                                                                                                  ');
OutTextXY(30, 152, 'The ReadE routine is aborted by pressing the right mouse key.                                     ');
OutTextXY(30, 162, '                                                                                                  ');
IF DemoMode THEN BEGIN
  SetColor(cEditWindowDisplay4);
  OutTextXY(30, 172, 'The ReadE function is not available in Demo mode.                                              ');
END;

        ShowMouseCursor;
        SetTextStyle(DefaultFont, HorizDir, 1);
        WriteBottomScreen('Press the right mouse key to return to the Help base page.', cEditWindowDisplay8);
        CheckMouseKeys;
    END;

Procedure HelpListE;
    BEGIN
        ClearScreen(0);
        {LimitsOnOff;}
        SetColor(cEditWindowDisplay4);
        SetTextStyle(DefaultFont, HorizDir, 2);
```

```
        WriteTopScreen('HELP: ListE', c
        SetColor(cEditWindowDisplay3);
        SetTextStyle(SmallFont,HorizDir,4);
        HideMouseCursor;

OutTextXY(30,  62, 'This function allows the user to List all messages, stored in an EPROM iserted into the EPROM    ');
OutTextXY(30,  72, 'programmer socket integral to the Record/Playback/Programmer module as waveforms, together      ');
OutTextXY(30,  82, 'with their ID numbers, in the Edit window.                                                       ');
OutTextXY(30,  92, '                                                                                                 ');
OutTextXY(30, 102, 'ListE procedure:                                                                                 ');
OutTextXY(30, 112, ' 1) Press the ListE button.                                                                      ');
OutTextXY(30, 122, '                                                                                                 ');
OutTextXY(30, 132, '                                                                                                 ');
OutTextXY(30, 142, 'The ListE routine is aborted by pressing the right mouse key.                                    ');
OutTextXY(30, 152, '                                                                                                 ');
OutTextXY(30, 162, '                                                                                                 ');
IF DemoMode THEN BEGIN
  SetColor(cEditWindowDisplay4);
  OutTextXY(30, 172, 'The ListE function is not available in Demo mode.                                             ');
END;

        ShowMouseCursor;
        SetTextStyle(DefaultFont, HorizDir, 1);
        WriteBottomScreen('Press the right mouse key to return to the Help base page.', cEditWindowDisplay8);
        CheckMouseKeys;
      END;

Procedure HelpVolume;
      BEGIN
        ClearScreen(0);
        {LimitsOnOff;}
        SetColor(cEditWindowDisplay4);
        SetTextStyle(DefaultFont, HorizDir, 2);
        WriteTopScreen('HELP: Volume', cEditWindowDisplay5);
        SetColor(cEditWindowDisplay3);
        SetTextStyle(SmallFont,HorizDir,4);
        HideMouseCursor;

OutTextXY(30,  62, 'The Volume poteniometer is used to control the level of the output signal when playing back a    ');
OutTextXY(30,  72, 'message.                                                                                         ');
OutTextXY(30,  82, '                                                                                                 ');
OutTextXY(30,  92, 'The Volume function is achieved by the software controlling an eight-step programmable attenuator ');
OutTextXY(30, 102, 'in the Record/Playback/Programmer module.                                                        ');
OutTextXY(30, 112, '                                                                                                 ');
OutTextXY(30, 122, 'The poteniometer is set by clicking the left mouse key with the mouse cursor at the desired     ');
OutTextXY(30, 132, 'poteniometer position.                                                                           ');
OutTextXY(30, 142, '                                                                                                 ');
OutTextXY(30, 152, '                                                                                                 ');
IF DemoMode THEN BEGIN
  SetColor(cEditWindowDisplay4);
  OutTextXY(30, 162, 'The Volume poteniometer has no effect when in demo mode.                                      ');
  OutTextXY(30, 172, '                                                                                               ');
END;

        ShowMouseCursor;
        SetTextStyle(DefaultFont, HorizDir, 1);
        WriteBottomScreen('Press the right mouse key to return to the Help base page.', cEditWindowDisplay8);
        CheckMouseKeys;
      END;


Procedure Help256K;
      BEGIN
        ClearScreen(0);
        {LimitsOnOff;}
        SetColor(cEditWindowDisplay4);
        SetTextStyle(DefaultFont, HorizDir, 2);
        WriteTopScreen('HELP: 128K / 256K',cEditWindowDisplay5);
        SetColor(cEditWindowDisplay3);
        SetTextStyle(SmallFont,HorizDir,4);
        HideMouseCursor;

OutTextXY(30,  62, 'The system allows for the use of either 128K byte EPROMs, or 256K byte EPROMs. The 128K / 256K    ');
OutTextXY(30,  72, 'function is used to indicate to the system which of the two types of EPROM is in use.           ');
OutTextXY(30,  82, '                                                                                                 ');
OutTextXY(30,  92, '                                                                                                 ');
OutTextXY(30, 102, 'Pressing the 128K / 256K button toggles between the two possibilities.                          ');
```

```
OutTextXY(30, 112, '                                                    ');
OutTextXY(30, 122, '                                                    ');
OutTextXY(30, 132, '                                                    ');
OutTextXY(30, 142, '                                                    ');
OutTextXY(30, 152, '                                                    ');
IF DemoMode THEN BEGIN
  SetColor(cEditWindowDisplay4);
  OutTextXY(30, 162, 'Pressing the 128K / 256K button in Demo mode has no effect.                ');
  OutTextXY(30, 172, '                                                  ');
END;

        ShowMouseCursor;
        SetTextStyle(DefaultFont, HorizDir, 1);
        WriteBottomScreen('Press the right mouse key to return to the Help base page.', cEditWindowDisplay8);
        CheckMouseKeys;
      END;

Procedure HelpQuit;
      BEGIN
        ClearScreen(0);
        {LimitsOnOff;}
        SetColor(cEditWindowDisplay4);
        SetTextStyle(DefaultFont, HorizDir, 2);
        WriteTopScreen('HELP: Quit',cEditWindowDisplay5);
        SetColor(cEditWindowDisplay3);
        SetTextStyle(SmallFont,HorizDir,4);
        HideMouseCursor;

OutTextXY(30,  62, 'The Quit routine, aborts the programme.                    ');
OutTextXY(30,  72, '                                                  ');
OutTextXY(30,  82, 'Quit procedure:                                   ');
OutTextXY(30,  92, ' 1) Press the Quit button.                        ');
OutTextXY(30, 102, ' 2) The user is prompted for confirmation. Press the left mouse key to abort the program.  ');
OutTextXY(30, 112, '                                                  ');
OutTextXY(30, 122, '                                                  ');
OutTextXY(30, 132, 'Pressing the right mouse key resumes the programme.        ');
OutTextXY(30, 142, '                                                  ');
OutTextXY(30, 152, '                                                  ');
OutTextXY(30, 162, '                                                  ');
OutTextXY(30, 172, '                                                  ');

        ShowMouseCursor;
        SetTextStyle(DefaultFont, HorizDir, 1);
        WriteBottomScreen('Press the right mouse key to return to the Help base page.', cEditWindowDisplay8);
        CheckMouseKeys;
      END;



Function MouseTouchArea (d1, d2, d3, d4 : Integer) : Boolean;
     BEGIN
       IF (d1 < MouseX) AND (MouseX < d1 + d3) AND
         (d2 < MouseY) AND (MouseY < d2 + d4)
       THEN MouseTouchArea := True
       ELSE MouseTouchArea := False;
     END;


procedure Help;

      Label BasePage;
      BEGIN
        Button (HelpX   ,HelpY   ,ButtonSizeX,ButtonSizeY,'HELP',cHelp,on);
        SetTextJustify(LeftText,BottomText);
        GetWholeScreen;

        BasePage:

        ClearScreen(0);
        {LimitsOnOff;}
        SetColor(cEditWindowDisplay4);
        SetTextStyle(DefaultFont, HorizDir, 2);
        WriteTopScreen('HELP Base Page',cEditWindowDisplay5);
        SetColor(cEditWindowDisplay3);
        SetTextStyle(SmallFont,HorizDir,4);
        HideMouseCursor;

OutTextXY(30,  62, 'Help briefly describe the various program functions.       ');
```

```
OutTextXY(30,  72, '                                                    ');
OutTextXY(30,  82, 'Help procedure:                                     ');
OutTextXY(30,  92, ' 1) Move the mouse cursor to the button or object more information is needed about.  ');
OutTextXY(30, 102, ' 2) Press the left mouse key.                       ');
OutTextXY(30, 112, '                                                    ');
OutTextXY(30, 122, 'Repeat this process to gain information about other buttons and objects as well.  ');
OutTextXY(30, 132, '                                                    ');
OutTextXY(30, 142, 'Press the right mouse key or select the Help button to return to the Help Base Page.  ');
OutTextXY(30, 152, '                                                    ');
OutTextXY(30, 162, 'To Exit the Help function, press the right mouse key while in the Help Base Page.  ');
OutTextXY(30, 172, '                                                    ');

        ShowMouseCursor;
        SetTextStyle(DefaultFont, HorizDir, 1);
        WriteBottomScreen('Select a button or object by clicking it using the left mouse key.',
                cEditWindowDisplay8);

        LeftMouseKeyWasPressed  := FALSE;
        RightMouseKeyWasPressed := FALSE;

        REPEAT
            LeftMouseKeyWasPressed  := FALSE;
            RightMouseKeyWasPressed := FALSE;
            REPEAT
                IF LeftMouseKeyPressed  THEN LeftMouseKeyWasPressed  := TRUE;
                IF RightMouseKeyPressed THEN RightMouseKeyWasPressed := TRUE;
            UNTIL (LeftMouseKeyWasPressed) OR (RightMouseKeyWasPressed);

            IF RightMouseKeyWasPressed THEN BEGIN
              SetTextStyle(DefaultFont,HorizDir,1);
              {SetMouseCursorPos(HelpX + ButtonSizeX div 2, HelpY + ButtonSizeY div 2);}
              PutWholeScreen;
              REPEAT UNTIL NOT AnyMouseKeyPressed;
              LeftMouseKeyWasPressed  := FALSE;
              RightMouseKeyWasPressed := FALSE;
              Button (HelpX   ,HelpY   ,ButtonSizeX,ButtonSizeY,'HELP',cHelp,off);
              Exit;
            END;

            IF LeftMouseKeyWasPressed THEN BEGIN

              IF (MouseTouchArea(ScreenX, ScreenY, ScreenSizeX, ScreenSizeY))THEN HelpEditWindow;

        IF (LeftMouseKeyWasPressed) AND (MouseTouchArea(SampleX  ,SampleY  ,ButtonSizeX,ButtonSizeY)) THEN
HelpRecord;
        IF (LeftMouseKeyWasPressed) AND (MouseTouchArea(TalkX    ,TalkY    ,ButtonSizeX,ButtonSizeY)) THEN HelpPlay
;
        IF (LeftMouseKeyWasPressed) AND (MouseTouchArea(MonitorX ,monitorY ,ButtonSizeX,ButtonSizeY)) THEN
HelpMute;
        IF (LeftMouseKeyWasPressed) AND (MouseTouchArea(CopyX    ,CopyY    ,ButtonSizeX,ButtonSizeY)) THEN
HelpCopy;
        IF (LeftMouseKeyWasPressed) AND (MouseTouchArea(MoveX    ,MoveY    ,ButtonSizeX,ButtonSizeY)) THEN
HelpMove;
        IF (LeftMouseKeyWasPressed) AND (MouseTouchArea(DeleteX  ,DeleteY  ,ButtonSizeX,ButtonSizeY)) THEN
HelpDelete;
        IF (LeftMouseKeyWasPressed) AND (MouseTouchArea(SaveX    ,SaveY    ,ButtonSizeX,ButtonSizeY)) THEN
HelpSave;
        IF (LeftMouseKeyWasPressed) AND (MouseTouchArea(LoadX    ,LoadY    ,ButtonSizeX,ButtonSizeY)) THEN
HelpLoad;
        IF (LeftMouseKeyWasPressed) AND (MouseTouchArea(DirectoryX,DirectoryY,ButtonSizeX,ButtonSizeY)) THEN
HelpPath;


        IF (LeftMouseKeyWasPressed) AND (MouseTouchArea(WriteEX  ,WriteEY  ,ButtonSizeX,ButtonSizeY)) THEN
HelpWriteE;
        IF (LeftMouseKeyWasPressed) AND (MouseTouchArea(ReadEX   ,ReadEY   ,ButtonSizeX,ButtonSizeY)) THEN
HelpReadE;
        IF (LeftMouseKeyWasPressed) AND (MouseTouchArea(ListEX   ,ListEY   ,ButtonSizeX,ButtonSizeY)) THEN
HelpListE;
        IF (LeftMouseKeyWasPressed) AND (MouseTouchArea(HelpX    ,HelpY  ,ButtonSizeX,ButtonSizeY)) THEN Goto
BasePage;
        IF (LeftMouseKeyWasPressed) AND (MouseTouchArea(EpromSizeX,EpromSizeY,ButtonSizeX,ButtonSizeY)) THEN
Help256K;
        IF (LeftMouseKeyWasPressed) AND (MouseTouchArea(QuitX,QuitY,ButtonSizeX,ButtonSizeY)) THEN HelpQuit;
        IF (LeftMouseKeyWasPressed) AND (MouseTouchArea(TriggerX,TriggerY,SliderSizeX,PotSizeY)) THEN HelpTrigger;
        IF (LeftMouseKeyWasPressed) AND (MouseTouchArea(SensitivityX,SensitivityY,SliderSizeX,PotSizeY)) THEN
HelpLevel;
        IF (LeftMouseKeyWasPressed) AND (MouseTouchArea(volumeX,volumeY,SliderSizeX,PotSizeY)) THEN HelpVolume;
```

```
END;

        UNTIL {(LeftMouseKeyWasPressed) OR} (RightMouseKeyWasPressed);
        GOTO BasePage;
    END;

END.
```

Unit BINU;

```
(*************************************************************************)
Interface
(*************************************************************************)

Function BinToWord(s : String) : Word;

Function WordToBin(w : Word) : String;

(*************************************************************************)
Implementation
(*************************************************************************)

Function BinToWord(s : String) : Word;
Var
  w,i,j : Word;
Begin
w := 0;
j := 1;
For i := Length(s) DownTo 1 Do
   Begin
   w := w + (Ord(s[i])-48) * j;
   j := j * 2;
   End;
BinToWord := w;
End;

(*************************************************************************)

Function WordToBin(w : Word) : String;
Var
  s : String;
  i,j,k : Word;
Begin
k := 1;
s := '';
For i :=$0 To $F Do
   Begin
   If (W and k) > 0 Then
     s := '1' + s
   Else
     s := '0' + s;
   k := k * 2;
   End;
WordToBin := s;
End;

(*************************************************************************)

end.
```

Unit VIDEO;

```
(*************************************************************************)
Interface
(*************************************************************************)

Uses DOS,
   CRT,
   Graph;

Type
  ScreenChars = Record
    ch : Char;
    at : Byte;
    End;

  screens = Record
    Position : Array[1..25, 1..80] Of ScreenChars;
    x, y : Byte;
    End;

  ScreenType =(Mono, Color);
  Var
    Stype : ScreenType;
    VidSeg : Word;

Procedure ShowScreen(Var Source, Video; Length: Word);

Procedure GetScreen(Var Video, Source; Length: Word);

Procedure XYstring(x, y : Byte;
                   s: String;
                   fg,
                   bg: Byte);

Procedure ReadScr(Var s);

Procedure WriteScr(Var s);

Procedure HorStr(x, y, Len : Byte;
                 fg, bg : Byte;
                 Ch : Char);

Procedure VerStr(x, y, Len : Byte;
                 fg, bg : Byte;
                 Ch : Char);

Procedure Box(x1,y1,x2,y2 : Byte;
              fg, bg : Byte);

Procedure Center(y : Byte;
                 st : String;
                 fg,
                 bg : Byte);

Procedure BoxString(y : Byte;
                    st : String;
                    fg, bg : Byte);

Procedure FillScreen(Var sc : screens;
                     s : String;
                     x, y : Byte;
                     fg, bg : Byte);

Procedure CursorOff;

Procedure CursorSmall;

Procedure CursorBig;

Function ReadCharOnScreen (Col,Row:Integer):Char;

Procedure WriteCharOnScreen(Col,Row,Character:Integer);

Procedure WriteCharOnScreenA(Col,Row,Character,Attribute:Integer);
```

```
(*************************************************************************)
Implementation
(*************************************************************************)

Var
  Regs : Registers;
  Vid : Pointer;


(*************************************************************************)

Procedure ShowScreen(Var Source, Video;
                     Length: Word);
{
This Procedure writes directly to video memory without snow.
}
Begin

If Stype = Color Then
  Inline($90/$90/$90/$90/
        $1E/$55/$BA/$DA/$03/$C5/$B6/ Source /$C4/$BE/ Video /
        $8B/$8E/ Length /$FC/$AD/$89/$C5/$B4/$09/$EC/$D0/$D8/
        $72/$FB/$FA/$EC/$20/$E0/$74/$FB/$89/$E8/$AB/$FB/$E2/
        $EA/$5D/$1F)

Else
   Begin
   Length := Length * 2;
   Move(Source,Video,Length);
   End;

End;

(*************************************************************************)

Procedure GetScreen(Var Video, Source;
                    Length: Word);
{
This procedure reads directly from video memory without snow.
}
Begin

If Stype = Color Then
  Inline ($1E/$55/$BA/$DA/$03/$C5/$B6/ Video /$C4/$BE/ Source /
        $8B/$8E/Length /$FC/$EC/$D0/$D8/$72/$FB/$FA/$EC/$D0/
        $D8/$73/$FB/$AD/$FB/$AB/$E2/$F0/$5D/$1F)

Else
   Begin
   Length := Length * 2;
   Move(Video,Source,Length);
   End;

End;

(*************************************************************************)

Procedure XYstring(x, y : Byte;
                   s: String;
                   fg,
                   bg: Byte);
{
This procedure writes string s at coordinates x:y using
foreground color fg and background color bg.
}
Var
  SA : Array [1..255] Of Record
     Ch : Char;
     at : Byte;
     End;

  b, i : Byte;
  Offset : Word;

Begin (* XYstring *)
If (Length(s) = 0) Or
```

```
  (x > 80) Or
  (x < 1) Or
  (y > 25) Or
  (y < 1) Then Exit;

(* create a single attribute byte. *)
b := (Ord(bg) Shl 4) Or Ord(fg);

(* move the string characters and *)
(* attribute byte into SA.       *)

FillChar(SA,SizeOf(SA),b);
For i := 1 To length(s) Do
   SA[i].ch := s[i];

(* calculate the offset into video display memory. *)
Offset := (((y-1)*80)+(x-1))*2;

Vid := Ptr(VidSeg,Offset);

(* write the string to video display memory. *)
ShowScreen (SA, Vid^,Length(s));
End; (* XYstring *)

(*************************************************************************)

Procedure ReadScr(Var s);
{
This procedure reads an entire screen display into s.
}
Begin
Vid := Ptr(VIdSeg,0);
GetScreen(Vid^,s,2000);
End;

(*************************************************************************)

Procedure WriteScr(Var s);
{
This screen writes an entire screen display from s.
}
Begin
Vid := Ptr(VidSeg,0);
ShowScreen(s, Vid^,2000);
End;

(*************************************************************************)

Procedure HorStr(x, y, Len : Byte;
                 fg, bg : Byte;
                 Ch : Char);
{
This Procedure draws a horizontal line.
}
Var
  i : Byte;
Begin
For i := 1 To Len Do
   Begin
   XYstring(x,y,ch,fg,bg);
   x := x + 1;
   End;
End;

(*************************************************************************)

Procedure VerStr(x, y, Len : Byte;
                 fg, bg : Byte;
                 Ch : Char);
{
This procedure draws a vertical line.
}
Var
  i : Byte;
Begin
For i := 1 to Len Do
   Begin
   XYstring(x,y,ch,fg,bg);
```

```
   y := y + 1;
   End;
End;

(*****************************************************************************)

Procedure Box(x1,y1,x2,y2 : Byte;
          fg, bg : Byte);
{
This procedure draws a box with upper left corner at x1:y1 and
lower right corner at x2:y2 with foreground color fg and
background color bg.
}
Begin
If (x1 < 1) Or
  (x2 > 80) Or
  (y1 < 1) Or
  (y2 > 25) Or
  ((x2-x1) < 2) Or
  ((y2-y1) < 2) Then Exit;

HorStr(x1, y1, 1, fg, bg, #201);
HorStr(x2, y1, 1, fg, bg, #187);
HorStr(x1, y2, 1, fg, bg, #200);
HorStr(x2, y2, 1, fg, bg, #188);
VerStr(x1, y1+1, y2-y1-1, fg, bg, #186);
VerStr(x2, y1+1, y2-y1-1, fg, bg, #186);
HorStr(x1+1, y1, x2-x1-1, fg, bg, #205);
HorStr(x1+1, y2, x2-x1-1, fg, bg, #205);
End;

(*****************************************************************************)

Procedure Center(y : Byte;
            st : String;
            fg,
            bg : Byte);
{
This procedure displays string st centered on line y using foreground
color fg and background color bg,
}
Var
  x : Byte;

Begin (* center *)
x := (40-(Length(st) Div 2));
XYstring(x, y, st, fg, bg);
End; (* center *)

(*****************************************************************************)

Procedure BoxString(y : Byte;
              st : String;
              fg, bg : Byte);
{
This procedure displays string st centered on line y surrounded
by a box, all displayed in foreground color fg and background
color bg.
}
Var
  x1 ,y1, x2, y2 : Byte;
Begin
Center(y, st, fg, bg);
x1 := 40-(Length(st) Div 2)-2;
x2 := x1+(Length(st)+3);
y1 := y-1;
y2 := y+1;
Box(x1 , y1, x2, y2, fg, bg);
End;

(*****************************************************************************)

Procedure FillScreen(Var sc : screens;
              s : String;
              x, y : Byte;
              fg, bg : Byte);
{
FillScreen writes string s to logical screen sc at coordinates
```

```
x:y in foreground color fg and background
}
Var
  i,atx : Byte;
Begin
atx := fg Or (bg Shl 4);

For  i := 1 To Length(s) Do
   Begin
   sc.position[y, x].ch := s[i];
   sc.position[y, x].at := atx;
   x := x+1;
   If x > 80 Then
     Begin
     x := 1;
     y := y+1;
     If y > 25 Then
       Exit;
     End;
   End;
End;

(*************************************************************************)

Procedure CursorOff;
{
Turns cursor off.
}
Begin
FillChar(Regs, sizeof(Regs), 0);

With Regs Do
   Begin
   AH := $01;
   CH := $20;
   CL := $20;
   End;

Intr($10, Regs);
End;

(*************************************************************************)


Procedure CursorSmall;
{
Turns cursor on in small size.
}
Begin
FillChar(Regs, sizeof(Regs), 0);

Regs.AH := $01;

      Case stype OF
      Mono :
         Begin
         With Regs Do
            Begin
            CH := 12;
            CL := 13;
            End;
         End;

      Color :
         Begin
         With Regs Do
            Begin
            CH := 6;
            CL := 7;
            End;
         End;
      End;   (* Of Case *)

Intr($10, Regs);
End;

(*************************************************************************)
```

```
Procedure CursorBig;
{
Turns cursor on in large size.
}
Begin
FillChar(Regs,sizeof(Regs),0);
Regs.AH := 1;
Regs.CH := 0;

     Case stype Of
     Mono : Regs.CL := 13;
     Color : Regs.CL := 7;
     End;

Intr($10, Regs);
END;

(*************************************************************************)

Function ReadCharOnScreen (Col,Row:Integer):Char;
{
Reads the character on the screen at position col, row.
}
     Var ScreenAddress : Word;
       Offset      : Integer;
     BEGIN
       IF (Mem[0000:1040] AND 48) <> 48 THEN
         ScreenAddress:=$B800
       ELSE ScreenAddress:= $B000;
       Offset:=((Row-1)*160)+((Col-1)*2);
       ReadCharOnScreen:=Chr(Mem[ScreenAddress:Offset]);
     END;

(*************************************************************************)

PROCEDURE WriteCharOnScreen(Col,Row,Character:Integer);
{
Writes a character on the screen at position col, row.
}

     VAR
       x,y : Integer;
       ScrAddr : Word;

     BEGIN
       IF (Mem[0000:1040] AND 48) <> 48 THEN
         ScrAddr:=$B800
       ELSE ScrAddr:= $B000;

        x:=((Col-1)*2);
        y:=((Row-1)*160);
        Mem[ScrAddr:x+y]:=Character;
     END;

(*************************************************************************)

PROCEDURE WriteCharOnScreenA(Col,Row,Character,Attribute:Integer);
{
Writes a character on the screen at position col, row with attribute.
}

     VAR
       x,y : Integer;
       ScrAddr : Word;

     BEGIN
       IF (Mem[0000:1040] AND 48) <> 48 THEN
         ScrAddr:=$B800
       ELSE ScrAddr:= $B000;

        x:=((Col-1)*2);
        y:=((Row-1)*160);
        Mem[ScrAddr:x+y]:=Character;
        Mem[ScrAddr:x+y+1]:=Attribute;
     END;

(*************************************************************************)
```

```
Begin
{
Initialization part of unit. Determines the type of screen
in use and set stype and VidSeg appropriately.
}
FillChar(Regs,sizeof(Regs),0);
Regs.AH := $0F;

Intr($10,Regs);
If Regs.AL = 7 Then
  Begin
  Stype := Mono;
  VidSeg := $B000;
  End
Else
   Begin
   Stype := Color;
   VidSeg := $B800;
   End;
End.  (* Of Unit *)
```

Unit gKEYBRD;

```
(*************************************************************************)
Interface
(*************************************************************************)

Uses Graph,CRT,MousU;
Type
  KeyType = (NullKey, F1, F2, F3, F4, F5, F6, F7, F8, F9, F10,
       CarriageReturn, TabKey, BackSpaceKey, UpArrow,
       DownArrow, RightArrow, LeftArrow, DelKey, InsertKey,
       HomeKey, EndKey, TextKey, NumberKey, SpaceKey,
       PgUp, PgDn, EscapeKey, RightMouseKey, LeftMouseKey);

  KeySetType = Set OF KeyType;

Var
  Key : KeyType;
  InsertOn : Boolean;
  LeftMouseKeyWasPressed,
  RightMouseKeyWasPressed : Boolean;


Procedure InKey(Var Ch : Char;
          Var fk : Boolean;
          Var Key : KeyType);

Procedure InputString(Var S : String;
            x,y,l : Word;
            fg,bg : Byte;
            KeySet : KeySetType);

(*************************************************************************)
Implementation
(*************************************************************************)

Procedure InKey(Var Ch : Char;
          Var fk : Boolean;
          Var Key : KeyType);
{
Gets a key from the user and returns the character. If the key
was a function key, fk is set to true. This procedure also sets
key equal to the type of key pressed.
}

    Begin

      LeftMouseKeyWasPressed  := FALSE;
      RightMouseKeyWasPressed := FALSE;

      REPEAT UNTIL NOT AnyMouseKeyPressed;

      REPEAT
        IF LeftMouseKeyPressed  THEN LeftMouseKeyWasPressed  := TRUE;
        IF RightMouseKeyPressed THEN RightMouseKeyWasPressed := TRUE;
      UNTIL (KeyPressed) OR (LeftMouseKeyWasPressed) OR (RightMouseKeyWasPressed);

      fk := False;


      IF LeftMouseKeyWasPressed  THEN BEGIN
        Key := LeftMouseKey;
        Exit;
      END;

      IF RightMouseKeyWasPressed THEN BEGIN
        Key := RightMouseKey;
        Exit;
      END;


      ch := ReadKey;
      fk := False;
      If ch = #0 Then Begin
        fk := True;
        ch := ReadKey;
```

```
        End;

        If fk Then
          Case ch Of
             #72 : key := UpArrow;          (* up arrow *)
             #80 : key := DownArrow;        (* down arrow *)
             #82 : key := InsertKey;        (* insert key *)
             #75 : key := LeftArrow;        (* let arrow *)
             #77 : key := RightArrow;       (* right arrow *)
             #73 : key := PgUp;             (* pge up *)
             #81 : key := PgDn;             (* pge down *)
             #71 : key := HomeKey;          (* home *)
             #79 : key := EndKey;           (* end key *)
             #83 : key := DelKey;           (* delete *)
             #82 : key := InsertKey;        (* Insert *)
             #59 : key := F1;               (* F1 *)
             #60 : key := F2;               (* F2 *)
             #61 : key := F3;               (* F3 *)
             #62 : key := F4;               (* F4 *)
             #63 : key := F5;               (* F5 *)
             #64 : key := F6;               (* F6 *)
             #65 : key := F7;               (* F7 *)
             #66 : key := F8;               (* F8 *)
             #67 : key := F9;               (* F9 *)
             #68 : key := F10;              (* F10 *)
          End  (* Of Case *)
        Else
          Case ch Of
             #8 : key := BackSpaceKey;      (* back space key *)
             #9 : key := TabKey;            (* tab key *)
             #13 : key := CarriageReturn;   (* carriage return *)
             #27 : key := EscapeKey;        (* escape *)
             #32 : key := SpaceKey;         (* space bar *)

             #33..#47,
             #58..#255 : key := TextKey;    (* text character *)

             #48..#57  : key := NumberKey;  (* number character *)
          End;
    End;

(*****************************************************************************)

Procedure InputString(Var S : String;
                      x,y,l : Word;
                      fg,bg : Byte;
                      KeySet : KeySetType);
{
This procerure allows the user to input string S at coordinates
x:y with a maximum length of l. The string is displayed in foreground
color fg and background color bg. KeySet defines the set of keys that
can be used to terminate the procedure.
}
    Const
       Fill : Char = #176;

    Var
      p : Byte;
      i,j : Word;
      ch,GraphicsCursor : Char;
      fk : Boolean;
      d1 : Byte;
      d2 : String;

    Begin

       i := Length(s) +1;
       If i > l Then s := Copy(s,1,l)
       Else Begin
          For j := i To l Do
          s[j] := Fill;
          s[0] := Chr(l);
       End;

       {
       For j := 1 To l Do s[j] := Fill;
       }
       s[0] := Chr(l);
```

```
    p:=1;

    Repeat
      If InsertOn Then GraphicsCursor := #95
      Else GraphicsCursor := #177;
      SetColor(0);

      d2 := '';
      FOR d1 := 1 to l DO BEGIN
        Insert('Û', d2, 1);
      END;

      OutTextXY(x, y, d2);
      SetColor(fg{1});
      OutTextXY(x+p*8-8,y,GraphicsCursor);
      OutTextXY(x,y,s);

      Inkey(ch,fk,key);

      Case Key Of
        TextKey, NumberKey, SpaceKey  : Begin
                                  If InsertOn Then Begin
                                    Insert(ch,s,p);
                                    s[0] := Chr(l);
                                    If p < l Then p := p + 1;
                                  End
                                  Else Begin
                                     s[p] := ch;
                                     If p < l Then p := p + 1;
                                  End;
                                End;

        InsertKey : Begin
                      InsertOn := Not InsertOn;
                    End;


        DelKey : Begin
                   Delete(s,p,1);
                   s := s + #176;
                 End;

        LeftArrow : Begin
                      If p > 1 Then
                      p := p - 1;
                    End;

        RightArrow : Begin
                       If (Pos(#176,s)>0) Then Begin
                         If (p < Pos(#176,s)) Then p := p + 1
                       End
                       Else If (p < l) Then p := p + 1;
                     End;

        BackSpaceKey : Begin
                         If p > 1 Then Begin
                           p := p - 1;
                           Delete(s,p,1);
                           s := s + #176;
                         End;
                       End;
      End; { Of Case }
    Until Key in KeySet;
    REPEAT UNTIL NOT AnyMouseKeyPressed;
    i := Pos(#176,s);
    If i > 0 Then s := Copy(s,1,i-1);
  End;

(*****************************************************************************)

Begin
InsertOn := true;
Key := NullKey;
End.
```

Unit MOUSU;

```
(*************************************************************************)
Interface
(*************************************************************************)

Uses CRT,DOS,VIDEO,BINU, Graph;

Type
   GraphCursMaskType = Record
     Mask : Array [0..1,0..15] of Word;
     HorzHotSpot,
     VertHotSpot : Integer;
     End;
Var
  m_a,m_l,m_r,m_b:Boolean;
  StandardShapeCurs,
  UpArrowCurs,
  LeftArrowCurs,
  CheckMarkCurs,
  PointingHandCurs,
  DiagonalCrossCurs,
  RectangularCrossCurs,
  HourGlassCurs : GraphCursMaskType;

  MouseX,
  MouseY,
  ButtonPressCount,
  ButtonReleaseCount,
  TextScrMask,
  TextCursMask : Word;

  NumMouseKeys : Byte;

  MousePresent : Boolean;

  MKey : (None,Left,Right,Both);

Procedure Resetmouse;

Procedure VirtualScreenSize(Var MaxX,
                            MaxY,
                            CellSizeX,
                            CellSizeY : Word);

Procedure ShowMouseCursor;

Procedure HideMouseCursor;

Procedure GetButtonStatus;

Procedure SetMouseCursorPos(x,y : Word);

Procedure GetButtonPressInfo(KeyNum : Word);

Procedure GetButtonReleaseInfo(KeyNum : Word);

Procedure SetMinMaxHorzCursPos(Min,Max : Word);

Procedure SetMinMaxVertCursPos(Min,Max : Word);

Procedure SetGraphicsCursor(Var Mask : GraphCursMaskType);

Procedure SetSoftTextCursor(TextScrMask,TextCursMask : Word);

Procedure SetHardTextCursorBig;

Procedure SetHardTextCursorSmall;

Procedure ReadMouseMotionCounters(Var Hcount, Vcount : Word);

Procedure LightPenEmulOn;

Procedure LightPenEmulOff;
```

```
Procedure SetMickeyToPixels(HRatio,VR

Procedure ConditionalOff(x1,y1,x2,y2 : Word);

Procedure SetDoubleSpeed(Speed : Word);

Procedure SaveMouseDriverState;

Procedure RestoreMouseDriverState;

Procedure SetCrtPageNumber(Page : Word);

Procedure GetCrtPageNumber(Var Page : Word);

Function AnyMouseKeyPressed : Boolean;

Function LeftMouseKeyPressed : Boolean;

Function RightMouseKeyPressed : Boolean;

Function BothMouseKeysPressed : Boolean;

Function AnyMouseKeyClicked : Boolean;

Function LeftMouseKeyClicked : Boolean;

Function RightMouseKeyClicked : Boolean;

Function BothMouseKeysClicked : Boolean;

Function MousX : Word;

Function MousY : Word;

Procedure FreezeMouse;

Procedure FreeMouse;


(*************************************************************************)
Implementation
(*************************************************************************)

Const
   MouseDelay = 250;
Var
  MouseSaveBuffer : Pointer;
  MKP,
  MouseVisible : Boolean;
  MouseBufferSize : Word;

  Regs : Registers;

Procedure DefineStandardShape;
Begin
With StandardShapeCurs DO
   Begin
   Mask[0,0]  := BinToWord('0001111111111111');
   Mask[0,1]  := BinToWord('0000111111111111');
   Mask[0,2]  := BinToWord('0000011111111111');
   Mask[0,3]  := BinToWord('0000001111111111');
   Mask[0,4]  := BinToWord('0000000111111111');
   Mask[0,5]  := BinToWord('0000000011111111');
   Mask[0,6]  := BinToWord('0000000001111111');
   Mask[0,7]  := BinToWord('0000000000111111');
   Mask[0,8]  := BinToWord('0000000000011111');
   Mask[0,9]  := BinToWord('0000000000001111');
   Mask[0,10] := BinToWord('0000000000000111');
   Mask[0,11] := BinToWord('0000000000000111');
   Mask[0,12] := BinToWord('0000000000111111');
   Mask[0,13] := BinToWord('0000100000011111');
   Mask[0,14] := BinToWord('0001111000011111');
   Mask[0,15] := BinToWord('1111111000011111');

   Mask[1,0]  := BinToWord('0000000000000000');
   Mask[1,1]  := BinToWord('0100000000000000');
   Mask[1,2]  := BinToWord('0110000000000000');
   Mask[1,3]  := BinToWord('0111000000000000');
```

```
    Mask[1,4]  := BinToWord('0111100000
    Mask[1,5]  := BinToWord('0111110000000000');
    Mask[1,6]  := BinToWord('0111111000000000');
    Mask[1,7]  := BinToWord('0111111100000000');
    Mask[1,8]  := BinToWord('0111111110000000');
    Mask[1,9]  := BinToWord('0111111111000000');
    Mask[1,10] := BinToWord('0111111111100000');
    Mask[1,11] := BinToWord('0111011000000000');
    Mask[1,12] := BinToWord('0110001100000000');
    Mask[1,13] := BinToWord('0100000110000000');
    Mask[1,14] := BinToWord('0000000011000000');
    Mask[1,15] := BinToWord('0000000000000000');

    HorzHotSpot := 0;
    VertHotSpot := -1;
    End;
End;

(*************************************************************************)

Procedure DefineUpArrow;
Begin
With UpArrowCurs DO
    Begin
    Mask[0,0]  := BinToWord('1111100111111111');
    Mask[0,1]  := BinToWord('1111000011111111');
    Mask[0,2]  := BinToWord('1110000001111111');
    Mask[0,3]  := BinToWord('1110000001111111');
    Mask[0,4]  := BinToWord('1100000000111111');
    Mask[0,5]  := BinToWord('1100000000111111');
    Mask[0,6]  := BinToWord('1000000000011111');
    Mask[0,7]  := BinToWord('1000000000011111');
    Mask[0,8]  := BinToWord('0000000000001111');
    Mask[0,9]  := BinToWord('0000000000001111');
    Mask[0,10] := BinToWord('1111000011111111');
    Mask[0,11] := BinToWord('1111000011111111');
    Mask[0,12] := BinToWord('1111000011111111');
    Mask[0,13] := BinToWord('1111000011111111');
    Mask[0,14] := BinToWord('1111000011111111');
    Mask[0,15] := BinToWord('1111000011111111');

    Mask[1,0]  := BinToWord('0000000000000000');
    Mask[1,1]  := BinToWord('0000011000000000');
    Mask[1,2]  := BinToWord('0000111100000000');
    Mask[1,3]  := BinToWord('0000111100000000');
    Mask[1,4]  := BinToWord('0001111110000000');
    Mask[1,5]  := BinToWord('0001111110000000');
    Mask[1,6]  := BinToWord('0011111111000000');
    Mask[1,7]  := BinToWord('0011111111000000');
    Mask[1,8]  := BinToWord('0111111111100000');
    Mask[1,9]  := BinToWord('0111111111100000');
    Mask[1,10] := BinToWord('0000011000000000');
    Mask[1,11] := BinToWord('0000011000000000');
    Mask[1,12] := BinToWord('0000011000000000');
    Mask[1,13] := BinToWord('0000011000000000');
    Mask[1,14] := BinToWord('0000011000000000');
    Mask[1,15] := BinToWord('0000000000000000');

    HorzHotSpot := 5;
    VertHotSpot := 0;
    End;
End;

(*************************************************************************)

Procedure DefineLeftArrow;
Begin
With LeftArrowCurs DO
    Begin
    Mask[0,0]  := BinToWord('1111111000011111');
    Mask[0,1]  := BinToWord('1111000000011111');
    Mask[0,2]  := BinToWord('0000000000000000');
    Mask[0,3]  := BinToWord('0000000000000000');
    Mask[0,4]  := BinToWord('0000000000000000');
    Mask[0,5]  := BinToWord('1111000000011111');
    Mask[0,6]  := BinToWord('1111111000011111');
    Mask[0,7]  := BinToWord('1111111111111111');
    Mask[0,8]  := BinToWord('1111111111111111');
```

```
    Mask[0,9]  := BinToWord('111111111
    Mask[0,10] := BinToWord('1111111111111111');
    Mask[0,11] := BinToWord('1111111111111111');
    Mask[0,12] := BinToWord('1111111111111111');
    Mask[0,13] := BinToWord('1111111111111111');
    Mask[0,14] := BinToWord('1111111111111111');
    Mask[0,15] := BinToWord('1111111111111111');

    Mask[1,0]  := BinToWord('0000000000000000');
    Mask[1,1]  := BinToWord('0000000011000000');
    Mask[1,2]  := BinToWord('0000011111000000');
    Mask[1,3]  := BinToWord('0111111111111110');
    Mask[1,4]  := BinToWord('0000011111000000');
    Mask[1,5]  := BinToWord('0000000011000000');
    Mask[1,6]  := BinToWord('0000000000000000');
    Mask[1,7]  := BinToWord('0000000000000000');
    Mask[1,8]  := BinToWord('0000000000000000');
    Mask[1,9]  := BinToWord('0000000000000000');
    Mask[1,10] := BinToWord('0000000000000000');
    Mask[1,11] := BinToWord('0000000000000000');
    Mask[1,12] := BinToWord('0000000000000000');
    Mask[1,13] := BinToWord('0000000000000000');
    Mask[1,14] := BinToWord('0000000000000000');
    Mask[1,15] := BinToWord('0000000000000000');

    HorzHotSpot := 0;
    VertHotSpot := 3;
    End;
End;

(*****************************************************************************)

Procedure DefineCheckMark;
Begin
With CheckMarkCurs DO
    Begin
    Mask[0,0]  := BinToWord('1111111111110000');
    Mask[0,1]  := BinToWord('1111111111100000');
    Mask[0,2]  := BinToWord('1111111111000000');
    Mask[0,3]  := BinToWord('1111111110000001');
    Mask[0,4]  := BinToWord('1111111100000011');
    Mask[0,5]  := BinToWord('0000011000000111');
    Mask[0,6]  := BinToWord('0000000000001111');
    Mask[0,7]  := BinToWord('0000000000011111');
    Mask[0,8]  := BinToWord('1100000000111111');
    Mask[0,9]  := BinToWord('1111000001111111');
    Mask[0,10] := BinToWord('1111111111111111');
    Mask[0,11] := BinToWord('1111111111111111');
    Mask[0,12] := BinToWord('1111111111111111');
    Mask[0,13] := BinToWord('1111111111111111');
    Mask[0,14] := BinToWord('1111111111111111');
    Mask[0,15] := BinToWord('1111111111111111');

    Mask[1,0]  := BinToWord('0000000000000000');
    Mask[1,1]  := BinToWord('0000000000000110');
    Mask[1,2]  := BinToWord('0000000000001100');
    Mask[1,3]  := BinToWord('0000000000011000');
    Mask[1,4]  := BinToWord('0000000000110000');
    Mask[1,5]  := BinToWord('0000000001100000');
    Mask[1,6]  := BinToWord('0111000011000000');
    Mask[1,7]  := BinToWord('0001110110000000');
    Mask[1,8]  := BinToWord('0000011100000000');
    Mask[1,9]  := BinToWord('0000000000000000');
    Mask[1,10] := BinToWord('0000000000000000');
    Mask[1,11] := BinToWord('0000000000000000');
    Mask[1,12] := BinToWord('0000000000000000');
    Mask[1,13] := BinToWord('0000000000000000');
    Mask[1,14] := BinToWord('0000000000000000');
    Mask[1,15] := BinToWord('0000000000000000');

    HorzHotSpot := 6;
    VertHotSpot := 7;
    End;
End;

(*****************************************************************************)

Procedure DefinePointingHand;
```

```
Begin
With PointingHandCurs DO
   Begin
   Mask[0,0]  := BinToWord('1110001111111111');
   Mask[0,1]  := BinToWord('1100000111111111');
   Mask[0,2]  := BinToWord('1100000111111111');
   Mask[0,3]  := BinToWord('1100000111111111');
   Mask[0,4]  := BinToWord('1100000000001111');
   Mask[0,5]  := BinToWord('1100000000000111');
   Mask[0,6]  := BinToWord('1100000000000111');
   Mask[0,7]  := BinToWord('0000000000000111');
   Mask[0,8]  := BinToWord('0000000000000011');
   Mask[0,9]  := BinToWord('0000000000000011');
   Mask[0,10] := BinToWord('0000000000000011');
   Mask[0,11] := BinToWord('0000000000000011');
   Mask[0,12] := BinToWord('0000000000000011');
   Mask[0,13] := BinToWord('1000000000000011');
   Mask[0,14] := BinToWord('1100000000000111');
   Mask[0,15] := BinToWord('1100000000000111');

   Mask[1,0]  := BinToWord('0000000000000000');
   Mask[1,1]  := BinToWord('0000100000000000');
   Mask[1,2]  := BinToWord('0001010000000000');
   Mask[1,3]  := BinToWord('0001010000000000');
   Mask[1,4]  := BinToWord('0001010000000000');
   Mask[1,5]  := BinToWord('0001011010100000');
   Mask[1,6]  := BinToWord('0001010101010000');
   Mask[1,7]  := BinToWord('0001010101010000');
   Mask[1,8]  := BinToWord('0111010101010000');
   Mask[1,9]  := BinToWord('0101000000001000');
   Mask[1,10] := BinToWord('0101000000001000');
   Mask[1,11] := BinToWord('0100000000001000');
   Mask[1,12] := BinToWord('0010000000001000');
   Mask[1,13] := BinToWord('0001000000010000');
   Mask[1,14] := BinToWord('0001111111110000');
   Mask[1,15] := BinToWord('0000000000000000');

   HorzHotSpot := 4;
   VertHotSpot := 0;
   End;
End;

(*****************************************************************************)

Procedure DefineDiagonalCross;
Begin
With DiagonalCrossCurs DO
   Begin
   Mask[0,0]  := BinToWord('0000011111100000');
   Mask[0,1]  := BinToWord('0000000110000000');
   Mask[0,2]  := BinToWord('0000000000000000');
   Mask[0,3]  := BinToWord('1100000000000011');
   Mask[0,4]  := BinToWord('1111000000001111');
   Mask[0,5]  := BinToWord('1100000000000011');
   Mask[0,6]  := BinToWord('0000000000000000');
   Mask[0,7]  := BinToWord('0000000110000000');
   Mask[0,8]  := BinToWord('0000001111000000');
   Mask[0,9]  := BinToWord('1111111111111111');
   Mask[0,10] := BinToWord('1111111111111111');
   Mask[0,11] := BinToWord('1111111111111111');
   Mask[0,12] := BinToWord('1111111111111111');
   Mask[0,13] := BinToWord('1111111111111111');
   Mask[0,14] := BinToWord('1111111111111111');
   Mask[0,15] := BinToWord('1111111111111111');

   Mask[1,0]  := BinToWord('0000000000000000');
   Mask[1,1]  := BinToWord('0111000000001110');
   Mask[1,2]  := BinToWord('0001110000111000');
   Mask[1,3]  := BinToWord('0000011001100000');
   Mask[1,4]  := BinToWord('0000001111000000');
   Mask[1,5]  := BinToWord('0000011001100000');
   Mask[1,6]  := BinToWord('0001110000111000');
   Mask[1,7]  := BinToWord('0111000000001110');
   Mask[1,8]  := BinToWord('0000000000000000');
   Mask[1,9]  := BinToWord('0000000000000000');
   Mask[1,10] := BinToWord('0000000000000000');
   Mask[1,11] := BinToWord('0000000000000000');
   Mask[1,12] := BinToWord('0000000000000000');
```

```
    Mask[1,13] := BinToWord('000000000
    Mask[1,14] := BinToWord('0000000000000000');
    Mask[1,15] := BinToWord('0000000000000000');

    HorzHotSpot := 7;
    VertHotSpot := 4;
    End;
End;

(*****************************************************************************)

Procedure DefinerectangularCross;
Begin
With RectangularCrossCurs DO
    Begin
    Mask[0,0]  := BinToWord('1111110000111111');
    Mask[0,1]  := BinToWord('1111110000111111');
    Mask[0,2]  := BinToWord('1111110000111111');
    Mask[0,3]  := BinToWord('0000000000000000');
    Mask[0,4]  := BinToWord('0000000000000000');
    Mask[0,5]  := BinToWord('0000000000000000');
    Mask[0,6]  := BinToWord('1111110000111111');
    Mask[0,7]  := BinToWord('1111110000111111');
    Mask[0,8]  := BinToWord('1111110000111111');
    Mask[0,9]  := BinToWord('1111111111111111');
    Mask[0,10] := BinToWord('1111111111111111');
    Mask[0,11] := BinToWord('1111111111111111');
    Mask[0,12] := BinToWord('1111111111111111');
    Mask[0,13] := BinToWord('1111111111111111');
    Mask[0,14] := BinToWord('1111111111111111');
    Mask[0,15] := BinToWord('1111111111111111');

    Mask[1,0]  := BinToWord('0000000000000000');
    Mask[1,1]  := BinToWord('0000000110000000');
    Mask[1,2]  := BinToWord('0000000110000000');
    Mask[1,3]  := BinToWord('0000000110000000');
    Mask[1,4]  := BinToWord('1111111111111111');
    Mask[1,5]  := BinToWord('0000000110000000');
    Mask[1,6]  := BinToWord('0000000110000000');
    Mask[1,7]  := BinToWord('0000000110000000');
    Mask[1,8]  := BinToWord('0000000000000000');
    Mask[1,9]  := BinToWord('0000000000000000');
    Mask[1,10] := BinToWord('0000000000000000');
    Mask[1,11] := BinToWord('0000000000000000');
    Mask[1,12] := BinToWord('0000000000000000');
    Mask[1,13] := BinToWord('0000000000000000');
    Mask[1,14] := BinToWord('0000000000000000');
    Mask[1,15] := BinToWord('0000000000000000');

    HorzHotSpot := 7;
    VertHotSpot := 4;
    End;
End;

(*****************************************************************************)

Procedure DefineHourGlass;
Begin
With HourGlassCurs DO
    Begin
    Mask[0,0]  := BinToWord('0000000000000000');
    Mask[0,1]  := BinToWord('0000000000000000');
    Mask[0,2]  := BinToWord('0000000000000000');
    Mask[0,3]  := BinToWord('0000000000000000');
    Mask[0,4]  := BinToWord('0000000000000000');
    Mask[0,5]  := BinToWord('1000000000000001');
    Mask[0,6]  := BinToWord('1110000000000111');
    Mask[0,7]  := BinToWord('1111100000011111');
    Mask[0,8]  := BinToWord('1111100000011111');
    Mask[0,9]  := BinToWord('1110000000000111');
    Mask[0,10] := BinToWord('1000000000000001');
    Mask[0,11] := BinToWord('0000000000000000');
    Mask[0,12] := BinToWord('0000000000000000');
    Mask[0,13] := BinToWord('0000000000000000');
    Mask[0,14] := BinToWord('0000000000000000');
    Mask[0,15] := BinToWord('0000000000000000');

    Mask[1,0]  := BinToWord('0000000000000000');
```

```
    Mask[1,1]  := BinToWord('0111111111
    Mask[1,2]  := BinToWord('0000000000000000');
    Mask[1,3]  := BinToWord('0111111111111110');
    Mask[1,4]  := BinToWord('0011010101010100');
    Mask[1,5]  := BinToWord('0000101010110000');
    Mask[1,6]  := BinToWord('0000001101000000');
    Mask[1,7]  := BinToWord('0000000110000000');
    Mask[1,8]  := BinToWord('0000000110000000');
    Mask[1,9]  := BinToWord('0000001011000000');
    Mask[1,10] := BinToWord('0000111010110000');
    Mask[1,11] := BinToWord('0011010101011100');
    Mask[1,12] := BinToWord('0110101010101010');
    Mask[1,13] := BinToWord('0000000000000000');
    Mask[1,14] := BinToWord('0111111111111110');
    Mask[1,15] := BinToWord('0000000000000000');

   HorzHotSpot := 7;
    VertHotSpot := 7;
    End;
End;

(*************************************************************************)

Procedure SetKeyStatus(Mstatus : Word);
Begin
    Case Mstatus of
    0: Mkey := None;
    1: Mkey := Left;
    2: Mkey := Right;
    3: Mkey := Both;
    End;
End;

(*************************************************************************)

Procedure ResetMouse;
Begin
Regs.AX := 0;
Intr($33,Regs);
With Regs Do
    Begin
    MousePresent := AX > 0;
    if MousePresent Then
      NummouseKeys := BX
    Else
       NumMouseKeys := 0;
    End;
MouseVisible := False;
End;

(*************************************************************************)

Procedure VirtualScreenSize(Var MaxX,
                            MaxY,
                            CellSizeX,
                            CellSizeY : Word);
Begin
(* Determine The Video Mode *)
Regs.AH := $0F;
Intr($10,Regs);

 Case Regs.AL of

 0,1:
    Begin
    MaxX     := 640;
    MaxY     := 200;
    CellSizeX := 16;
    CellSizeY := 8;
    End;

 2,3,7:
    Begin
    MaxX     := 640;
    MaxY     := 200;
    CellSizeX := 8;
    CellSizeY := 8;
    End;
```

```
 4,5:
    Begin
    MaxX     := 640;
    MaxY     := 200;
    CellSizeX := 2;
    CellSizeY := 1;
    End;

 6:
    Begin
    MaxX     := 640;
    MaxY     := 200;
    CellSizeX := 1;
    CellSizeY := 1;
    End;

 13:
    Begin
    MaxX     := 640;
    MaxY     := 200;
    CellSizeX := 16;
    CellSizeY := 8;
    End;

 14,15:
    Begin
    MaxX     := 640;
    MaxY     := 350;
    CellSizeX := 1;
    CellSizeY := 1;
    End;
 End;
End;

(*****************************************************************************)

Procedure ShowMouseCursor;
Begin
If not MouseVisible Then
  Begin
  Regs.AX := 1;
  Intr($33,Regs);
  MouseVisible := True;
  End;
End;

(*****************************************************************************)

Procedure HideMouseCursor;
Begin
If MouseVisible Then
  Begin
  Regs.AX := 2;
  Intr($33,Regs);
  MouseVisible := False;
  End;
End;

(*****************************************************************************)

Procedure GetButtonStatus;
Begin
Regs.AX := 3;
Intr($33,Regs);
With Regs Do
    Begin
    SetKeyStatus(BX);
    MouseX := CX;
    MouseY := DX;
    End;
End;

(*****************************************************************************)

Procedure SetMouseCursorPos(x,y : Word);
Begin
With Regs Do
```

```
    Begin
    AX := 4;
    CX := x;
    dx := y;
    End;

Intr($33,Regs);
MouseX := x;
MouseY := y;
End;

(*****************************************************************************)

Procedure GetButtonPressInfo(KeyNum : Word);
Begin
With Regs Do
    Begin
    AX := 5;
    BX := KeyNum - 1;
    End;
Intr($33,Regs);
With Regs Do
    Begin
    SetKeyStatus(AX);
    ButtonPressCount := BX;
    MouseX := CX;
    MouseY := DX;
    End;
End;

(*****************************************************************************)

Procedure GetButtonReleaseInfo(KeyNum : Word);
Begin
With Regs Do
    Begin
    AX := 6;
    BX := KeyNum - 1;
    End;
Intr($33,Regs);
With Regs Do
    Begin
    SetKeyStatus(AX);
    ButtonPressCount := BX;
    MouseX := CX;
    MouseY := DX;
    End;
End;

(*****************************************************************************)

Procedure SetMinMaxHorzCursPos(Min,Max: Word);
Begin
With Regs Do
    Begin
    AX := 7;
    CX := Min;
    DX := Max;
    End;
Intr($33,Regs);
End;

(*****************************************************************************)

Procedure SetMinMaxVertCursPos(Min,Max: Word);
Begin
With Regs Do
    Begin
    AX := 8;
    CX := Min;
    DX := Max;
    End;
Intr($33,Regs);
End;

(*****************************************************************************)

Procedure SetGraphicsCursor(Var Mask: GraphCursMaskType);
```

```
Begin
With Regs Do
   Begin
   AX := 9;
   BX := Word(Mask.HorzHotSpot);
   CX := Word(Mask.VertHotSpot);
   DX := Ofs(Mask);
   ES := Seg(Mask);
   End;
Intr($33,Regs);
End;

(*****************************************************************************)

Procedure SetSoftTextCursor(TextScrMask,
                  TextCursMask : Word);
(* TextScrMask defines which character and color attributes are *)
(*    after the sotware mouse cursor is moved,              *)
(* TextCursMask defines how the character and colors are      *)
(*    changed in order to display the software mouse cursor   *)
Begin
With Regs Do
   Begin
   AX := 10;
   BX := 0;
   CX := TextScrMask;
   DX := TextCursMask;
   End;
Intr($33,Regs);
End;

(*****************************************************************************)

Procedure SetHardTextCursorBig;
Begin
With Regs Do
   Begin
   AX := 10;
   BX := 1;
   If Stype = Mono Then
     Begin
     CX := 0;
     DX := 13;
     End
   Else
      Begin
      CX := 0;
      DX := 7;
      End;
   End;
Intr($33,Regs);
End;

(*****************************************************************************)

Procedure SetHardTextCursorSmall;
Begin
With Regs Do
   Begin
   AX := 10;
   BX := 1;
   If Stype = Mono Then
     Begin
     CX := 12;
     DX := 13;
     End
   Else
      Begin
      CX := 6;
      DX := 7;
      End;
   End;
Intr($33,Regs);
End;

(*****************************************************************************)

Procedure ReadMouseMotionCounters(Var Hcount,
```

```
                    Vcount : Word);
Begin
Regs.AX := 11;
Intr($33,Regs);
With Regs Do
   Begin
   Hcount := CX;
   Vcount := DX;
   End;
End;

(*****************************************************************************)

Procedure LightPenEmulOn;
Begin
Regs.AX := 13;
Intr($33,Regs);
End;

(*****************************************************************************)

Procedure LightPenEmulOff;
Begin
Regs.AX := 14;
Intr($33,Regs);
End;

(*****************************************************************************)

Procedure SetMickeyToPixels(HRatio,VRatio : Word);
Begin
With Regs Do
Begin
   AX := 15;
   CX := HRatio;
   DX := VRatio;
   End;
Intr($33,Regs);
End;

(*****************************************************************************)

Procedure ConditionalOff(x1,y1,x2,y2 : Word);
Begin
With Regs Do
Begin
   AX := 16;
   CX := x1;
   DX := y1;
   SI := x2;
   DI := y2;
   End;
Intr($33,Regs);
End;

(*****************************************************************************)

Procedure SetDoubleSpeed(Speed : Word);
Begin
With Regs Do
Begin
   AX := 19;
   DX := Speed;
   End;
Intr($33,Regs);
End;

(*****************************************************************************)

Procedure GetMouseSaveSize;
Begin
Regs.AX := 21;
Intr($33,Regs);
MouseBufferSize := Regs.BX;
End;

(*****************************************************************************)
```

```
Procedure SaveMouseDriverState;
Begin
GetMem(MouseSaveBuffer,MouseBufferSize);
With Regs Do
Begin
   AX := 22;
   DX := Ofs(MouseSaveBuffer^);
   ES := Seg(MouseSaveBuffer^);
   End;
Intr($33,Regs);
End;

(*****************************************************************************)

Procedure RestoreMouseDriverState;
Begin
With Regs Do
Begin
   AX := 23;
   DX := Ofs(MouseSaveBuffer^);
   ES := Seg(MouseSaveBuffer^);
   End;
Intr($33,Regs);
FreeMem(MouseSaveBuffer,MouseBufferSize);
End;

(*****************************************************************************)

Procedure SetCrtPageNumber(Page : Word);
Begin
With Regs Do
Begin
   AX := 29;
   BX := Page;
   End;
Intr($33,Regs);
End;

(*****************************************************************************)

Procedure GetCrtPageNumber(Var Page : Word);
Begin
Regs.AX := 23;
Intr($33,Regs);
Page := Regs.BX;
End;

(*****************************************************************************)

Function AnyMouseKeyPressed : Boolean;
Begin
GetButtonStatus;
MKP := Mkey <> None;
AnyMouseKeyPressed := MKP;
End;

(*****************************************************************************)

Function LeftMouseKeyPressed : Boolean;
Begin
GetButtonStatus;
MKP := Mkey = Left;
LeftMouseKeyPressed := MKP;
End;

(*****************************************************************************)

Function RightMouseKeyPressed : Boolean;
Begin
GetButtonStatus;
MKP := Mkey = Right;
RightMouseKeyPressed := MKP;
End;

(*****************************************************************************)

Function BothMouseKeysPressed : Boolean;
Begin
```

```
{If MKP Then}
GetButtonStatus;
MKP := Mkey = Both;
BothMouseKeysPressed := MKP;
End;

(*****************************************************************************)

Function AnyMouseKeyClicked : Boolean;
Begin
GetButtonStatus;
MKP := Mkey <> None;
IF (NOT m_a) AND (MKP) THEN AnyMouseKeyClicked := TRUE
ELSE AnyMouseKeyClicked := FALSE;
m_a:=MKP;
End;

(*****************************************************************************)

Function LeftMouseKeyClicked : Boolean;
Begin
GetButtonStatus;
MKP := Mkey = Left;
IF (NOT m_l) AND (MKP) THEN LeftMouseKeyClicked := TRUE
ELSE LeftMouseKeyClicked := FALSE;
m_l:=MKP;
End;

(*****************************************************************************)

Function RightMouseKeyClicked : Boolean;
Begin
GetButtonStatus;
MKP := Mkey = Right;
IF (NOT m_r) AND (MKP) THEN RightMouseKeyClicked := TRUE
ELSE RightMouseKeyClicked := FALSE;
m_r:=MKP;
End;

(*****************************************************************************)

Function BothMouseKeysClicked : Boolean;
Begin
{If MKP Then}
GetButtonStatus;
MKP := Mkey = Both;
IF (NOT m_b) AND (MKP) THEN BothMouseKeysClicked := TRUE
ELSE BothMouseKeysClicked := FALSE;
m_b:=MKP;
End;

(*****************************************************************************)

Function MousX : Word;
    BEGIN
      MousX:=MouseX div 8 + 1;
    END;

(*****************************************************************************)

Function MousY : Word;
    BEGIN
      MousY:=MouseY div 8 + 1;
    END;

(*****************************************************************************)

Procedure FreezeMouse;
    BEGIN
      SetMinMaxHorzCursPos(MouseX,MouseX);
      SetMinMaxVertCursPos(MouseY{-8},MouseY{+8});
    END;

(*****************************************************************************)

Procedure FreeMouse;
    BEGIN
      SetMinMaxHorzCursPos(0, GetMaxX){(0,79*8)};
```

```
    SetMinMaxVertCursPos(0, GetMax
  END;

(*****************************************************************************)


Begin
ResetMouse;
DefineStandardShape;
DefineUpArrow;
DefineLeftArrow;
DefineCheckMark;
DefinePointingHand;
DefineDiagonalCross;
DefineRectangularCross;
DefineHourGlass;
MKP := False;
End.
```

REFERENCES

ANONYMOUS, 1987: Signal processing, : *Practical electronics* Vol 23 No. 8: 19 - 23. August 1987.

BAERT &THEUNISSEN & VERGULT 1992. *Digital Audio and Compact Disc Technology.* Butterworth-Heinemann Ltd, Oxford.

BAUMANN, HP. 1981: Talk to computers : *Elektor* Issue No. 73: 5.17 - 5.19. May 1981.

BRISTOW, G. 1986. *Electronic Speech Recognition.* Collins, London.

CHAPPELL, P. 1987: The sample life : *Electronics Digest* Vol 8 No 2: 10 - 14 September 1987.

FURUI, S. & SONDHI, MM. 1992. *Advances in Speech Signal Processing,* Marcel Dekker, New York.

FURUI, S. 1989. *Digital Speech Processing, Synthesis and Recognition.* Marcel Dekker, New York.

HAYKIN, S. 1988. *Digital Communications*, John Wiley & Sons Inc, New York.

HOLMES, JN. 1988. *Speech synthesis and recognition*. Van Nostrand Reinhold, Berkshire.

JIMENEZ, R. 1991. *Designing with speech processing chips.* Academic press, San Diego.

KAUFMAN, M & SEIDMAN, AH, 1988. *Electronics Sourcebook for Technicians and Engineers.* McGraw-Hill, New York.

KENNEDY, G. 1984. *Electroni ems*. McGraw-Hill, New York.

KING, W. 1977: Noise reduction systems : *Electronics today international* Vol. 6 No. 10: 16- 19. November 1977.

MORGAN, N. 1984. *Talking Chips*. McGraw-Hill,New York.

NOLL, AM. 1988. *Introduction to Telecommunication Electronics.* Artech House, Boston.

OWENS, FJ. 1993. *Signal Processing of Speech*, The MacMillan Press, London.

PETERSEN, D. 1992. *Audio, Video and Data Telecommunications.* McGraw-Hill, New York.

RAMSEY, F. & WATKINSON, J. 1993. *The Digital Interface Handbook.* Butterworth-Heinemann, Oxford.

REESE, R and KELLER, S. 1982: Speech board makes anything talk : *Computers and electronics* Vol. 20 No. 12: 47 - 62. December 1982.

RODDY, D & COOLEN, J. 1984. *Electronic Comunications*. Prentice Hall, Virginia.

RUMSEY, F. 1990. *Tapeless sound recording*. Focal press, London.

SAVON, K. 1982: Speech Synthesis Techniques : *Radio- Electronics* Vol. 53 No. 2: 62 - 65. February 1982.

SCHAERER, T. 1982: A/D and D/A conversion. *Elektor* Issue No. 83: 3.42 - 3.46. March 1982.

SOKOLOWSKI, S. 1990. *The talking telephone*. McGraw-Hill, New York.

TAKASAKI, Y. 1991. *Digital Transmission Design and Jitter Analysis.* Artech House, Boston.

WALKER, P. 1984:. Programmable speech board, *Electronics today international* Vol 13 No. 2: 20 - 26. February 1984.

WATKINSON, J. 1991. *The art of digital audio*. Butterworth-Heinemann, Oxford.

WHEDDON, C. & LINGGARD, R. 1990. *Speech and Language Processing*. Chapman and Hall, London.

WILLIAMS, T. 1994: New Software techniques boost the IQs of embedded systems. *Computer Design* September 1994 67-76

References used but not directly referred to:

ANALOG DEVICES. 1986. *Analog - Digital conversion handbook*. Prentice Hall, London.

BIGNELL, JW. & DONOVAN, RL. 1994.*Digital Electronics.* Delmar publishers, New York.

HSU, HP. 1993. *Analog and Digital Communications*. McGraw-Hill, New York.

JONES, LD. 1986. *Principals and Applications of Digital Electronics.* MacMillan Publishing, New York.

JONES, NB & WATSON, JD. 1990. *Digital Signal Processing*. Peter Peregrinus Ltd, London.

LEE, BGI. & KANG, M. & LEE, J. 1993. *Broadband Telecommunications Technology.* Artech house, Boston.

References consulted in developing the software:

Bishop, J & Bishop, N. 1990. *Pascal Precisely for Engineers and Scientists.* Addison - Wesley, Wokingham.

Dutton, F. 1989. *Turbo Pascal Toolbox.* Sybex, San Francisco.

Ezzell, B. 1989 *Object- Oriented programming in Turbo Pascal 5.5.* Addison - Wesley, Reading.

Mallozzi, JS. 1988 *Turbo Pascal for program design.* Mc Graw - Hill, New York.

O' Brien, SK. 1988. *Turbo Pascal Advanced Programmers Guide.* Osborne Mc Graw - Hill, Berkeley.

O' Brien, SK. 1989. *Turbo Pascal 5.5 The Complete Reference.* Osborne Mc Graw - Hill, Berkeley.

Ohlsen, C. & Stoker, G. 1989. *Turbo Pascal Advanced Techniques.* QUE corporation, Carmel.

Rought, E. & Hoops, T. 1988 *Turbo Pascal 4.0 developers library.* Howard W. Sams, Indianapolis.

Swan, T. 1988. *Mastering Turbo Pascal 4.0.*Howard W Sams , Indianapolis.

Swan, T. 1987. *Mastering Turbo Pascal Files.* Howard W Sams , Indianapolis.