INFORMATION TO USERS

This reproduction was made from a copy of a document sent to us for microfilming. While the most advanced technology has been used to photograph and reproduce this document, the quality of the reproduction is heavily dependent upon the quality of the material submitted.

The following explanation of techniques is provided to help clarify markings or notations which may appear on this reproduction.

1. The sign or "target" for pages apparently lacking from the document photographed is "Missing Page(s)". If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting through an image and duplicating adjacent pages to assure complete continuity.

2. When an image on the film is obliterated with a round black mark, it is an indication of either blurred copy because of movement during exposure, duplicate copy, or copyrighted materials that should not have been filmed. For blurred pages, a good image of the page can be found in the adjacent frame. If copyrighted materials were deleted, a target note will appear listing the pages in the adjacent frame.

3. When a map, drawing or chart, etc., is part of the material being photographed, a definite method of "sectioning" the material has been followed. It is customary to begin filming at the upper left hand corner of a large sheet and to continue from left to right in equal sections with small overlaps. If necessary, sectioning is continued again—beginning below the first row and continuing on until complete.

4. For illustrations that cannot be satisfactorily reproduced by xerographic means, photographic prints can be purchased at additional cost and inserted into your xerographic copy. These prints are available upon request from the Dissertations Customer Services Department.

5. Some pages in any document may have indistinct print. In all cases the best available copy has been filmed.

University
Microfilms
International

300 N. Zeeb Road
Ann Arbor, MI 48106

Jin, Yi-Xuan

A HIGH SPEED DIGITAL IMPLEMENTATION OF LPC SPEECH SYNTHESIZER USING THE TMS320

The University of Arizona M.S. 1985

A HIGH SPEED DIGITAL IMPLEMENTATION OF LPC SPEECH SYNTHESIZER USING THE TMS320

by

Yi-Xuan Jin

A Thesis Submitted to the Faculty of the

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

In Partial Fulfillment of the Requirements
For the Degree of

MASTER OF SCIENCE
WITH A MAJOR IN ELECTRICAL ENGINEERING

In the Graduate College

THE UNIVERSITY OF ARIZONA

1 9 8 5

STATEMENT BY AUTHOR

This thesis has been submitted in partial fulfillment of requirements for an advanced degree at The University of Arizona to be made available to borrowers under rules of the Library.

Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledge of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his or her judgment the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

SIGNED: _____

APPROVAL BY THESIS DIRECTOR

This thesis has been approved on the date shown below:

Robin N. Strickland — May 23 1985

R. N. STRICKLAND
Assistant Professor of Electrical and Computer Engineering

Date

ACKNOWLEDGMENTS

I am grateful to Dr. Robin N. Strickland for his guidance and encouragement throughout the course of this research. I would also like to thank Dr. Fredrick J. Hill and Dr. Lawrence P. Huelsman for their time as members of the committee. My appreciation is extended Dr. Gerald R. Peterson, Dr. Hill, and Dr. Donald L. Shirer for their teaching excellence. I also wish to express my appreciation to Ms. Juanita Maier for her help that made my study at the University of Arizona an enjoyable experience.

I also wish to express my deepest thanks to Gerrard M. Carlson and Alen R. Paters for their invaluable suggestions and assistances throughout this research.

Fianlly, I like to express my special thanks to my fiancee, Qiusha, for her understanding, patience, and support, without which this program would not have been possible.

TABLE OF CONTENTS

TABLE OF CONTENTS --Continued

TABLE OF CONTENTS -- Continued

LIST OF ILLUSTRATIONS

LIST OF TABLES

ABSTRACT

In this paper the design, implementation, and evaluation of a fast-time Linear Predictive Coding (LPC) speech synthesis system are examined.

The well-known two-multiplier lattice structure is used to implement the LPC synthesis filter. The synthesis filter, which is defined by the filter coefficients from the speech parameter file, produces the synthetic speech.

The synthesizer is implemented with the TMS320 assembly language that is executed in real-time by a modified Texas Instruments TMS320 evaluation module - EVM320. A Motorola EXORmacs computer is used to generate and store speech parameter files and synthesizer source codes.

The quality of the speech synthesized by the system is evaluated on a quantitative basis. Also, by using the synthesizer as an evaluation tool, several data compression schemes have been implemented and evaluated.

CHAPTER 1

INTRODUCTION

The major contribution of this paper concerns the development, design, and testing of a real-time speech synthesizer implementation on the TMS320 digital signal processing chip. Man-machine speech communication is accomplished by speech analysis and speech synthesis. In this paper, only speech synthesis will be discussed. Among many speech synthesis schemes, the Linear Predictive Coding (LPC) is the most popular one for its accuracy and convenience in speech analysis/synthesis.

Motivation for Using Speech Output

Speech is our everyday, secondary, informal, communication medium. It is secondary because speech by no means can exceed or replace man's most important communication medium-vision. It is also informal because speech is inherently treated as a lower-priority signal (as opposed to visual images) by the human brain. But speech communication has several unique advantages which can be utilized to enhance the possibility of man-computer communication.

First, speech offers an auxiliary communication channel that leaves hands and eyes free for other (usually more important) tasks. One can listen to the radio while writing, reading, or operating a machine. Moreover, speech is omnidirectional, and it does not require a free line of sight. Using this property, speech can be used as a secondary medium to convey some status information like warning messages. Occasional speech signal interruptions do not interfere with other activities, unless they need unusual attention. The speech warning message in a car is the best example; the driver will hear a voice message without diverting his sight to the dashboard.

Second, speech communication can exploit the existing world-wide telephone network. Speech communication can be accomplished with no tools other than a telephone set. This afforts the opportunity of computer-access by telephone from anywhere in the world in seconds. This easy remote man-machine communication by telephone line could bring revolutionary changes in all aspects of our life.

Third, speech is potentially more economical than other means of communication. Today, with the help of the advanced semiconductor technology, a entire speech synthesis system except the loudspeaker can be implemented on a single integrated circuit with a cost as low as $10. Other

computer input/output (I/O) devices at present are tied to either mechanical moving parts or to terminals, which are more expensive and more vulnerable to harsh environmental conditions. This is not to say, however, that speech I/O devices will replace all others in the future. It is clear that I/O devices like printers and visual terminals are indispensable to computer systems. Speech I/O is used only to enrich men-machine communication possibilities.

The advantages of speech mentioned above have been utilized to realize many practical applications in the past decades. The talking calculator and the Speak & Spell are two successful novelties. In contrast, the computer-generated speech wiring list is certainly a practical application. The operator, with the help of speech wiring instructions, no longer needs to look at both the wiring list and the wiring board. The operator will hear the next wiring instruction when he needs it, keeping both his hands and eyes in the task. Probably the most advanced application so far is the Kurzweil reading machine for the blind. This machine can read text at a maximum speed of 225 words per minute - perhaps half the normal human speech rate.

A Survey of Speech Processing Techniques

The first, probably still the most widely used method for storing and replaying audio waveform is the use

of the audio tape recorder. Althougth this method provides high quality output at a low cost, it is difficult for the tape recorder to provide rapid random data access. Second, along with the rapid decline of integrated circuit cost, mechanical systems have become more and more expensive. Third, mechanical systems with moving parts are inherently slower and failure-prone.

The direct storage of digitized audio waveforms has been used in many voice-response systems. Several schemes in this category are Pulse Code Modulation and Delta Modulation. These methods offer extremely high quality reproduction of audio signals and the digitized data are easy to manipulate in a digital computer. Its most obvious limitation is that a very high data rate (about 60-100,000 bps) is required. This limitation, however, is becoming less an overriding factor because of the falling cost of storage systems. A more fundamental limitation is that utterances stored directly cannot be modified sensibly to form a different context since the utterances are treated as just general waveforms.

The synthesis-by-rule method has been used for achieving both maximum flexibility and low data rate. By using this method, a vocabulary of sufficient size is used to yield a complete inventory of phonetic building blocks,

from which any English word can be constructed.Therefore, rather than storing the actual speech waveform, the rules of how to construct the speech by using these building blocks are stored for later synthesis. The synthesis-by-rule method can achieve an extremely low (about 100 bps!) data rate and an infinite-size vocabulary. However, this method is very difficult to realize due to the extreme complex nature of human speech. Thus the synthesized speech is usually of low quality.

Synthesis-by-analysis is the most popular method in today's digital speech processing field. This is because it offers high quality, flexible synthetic speech at a relative low data rate (about 500-4,000 bps). Instead of storing the speech waveform, synthesis-by-analysis stores the analysis results of utterances. The basic idea is to define a model that can be adjusted to produce different speech sounds; then by varying the parameters of the model in a systematic way, a set of parameters can be found to match the original speech with minimum error. When such a match is found, the parameters of the model can be assumed to be the parameters of the speech signal.

The formant analysis method is an interpretation of synthesis-by-analysis in the frequency domain. In formant analysis, a number of filters with different resonant

frequencies and bandwidths are used to model the resonances of the vocal tract by exciting the filters with either a train of periodic impulses or a noise source.

The linear predictive coding speech analysis/synthesis is a time domain method, and is by far the most popular method in speech processing.

Linear Predictive Coding

In this section, the basic principles of linear predictive coding are discussed in a rather intuitive way. The theoretical aspect of linear predictive coding will be treated in Chapter 2.

The basic idea of linear predictive coding is that a speech sample X(n) can be predicted quite closely from the previous samples X(n-1), X(n-2),.... This approximation can be improved by multiplying the previous samples by some coefficients, say A_1, A_2,..., which are adapted on a short-time scale. The prediction error E(n) is therefore

$$E(n)=X(n)-[A_1X(n-1)+A_2X(n-2)+\ldots+A_pX(n-p)] \qquad (1.1)$$

where p is the number of previous samples. The coefficients A_k (k=1,...,p) should be adapted to minimize the error signal E(n). The details of the analysis will be considered in Chapter 2. Since the speech waveform is time-varying, the coefficients A_k need to be re-calculated and

transmitted for every short-time period. It turns out that A_k vary at a much slower rate than the speech waveform in the time domain. Therefore, using A_k to represent a speech signal, a data rate as low as 2,000 bps can be achieved (compared with 64,000 bps in direct digital recording). Figure 1 shows the first five reflection coefficients of the phrase "When will our yellow lion roar?". It is clear that, in the time domain, the coefficients change much slowly than the speech waveform.

It will be shown that the coefficients A_k uniquely define a cascade of second-order polynomials representing the individual formants of the vocal tract. Furthermore, it will be shown that the predictive error E(n), which is minimized by A_k, can be modelled by either an impulse train generator or a white noise generator. Both source models need only two specifications: excitation amplitude, and pitch length. Therefore, in linear predictive analysis/synthesis, the only parameters needed to be stored and transferred are the filter model coefficients and the excitation model parameters. This is how LPC method separates out the excitation properties of the source from that of the vocal tract. The source parameters can be derived from the error E(n), and the vocal tract is modelled by filter coefficients A_k (k=1,2,...,p). Thus, for a modelling filter of

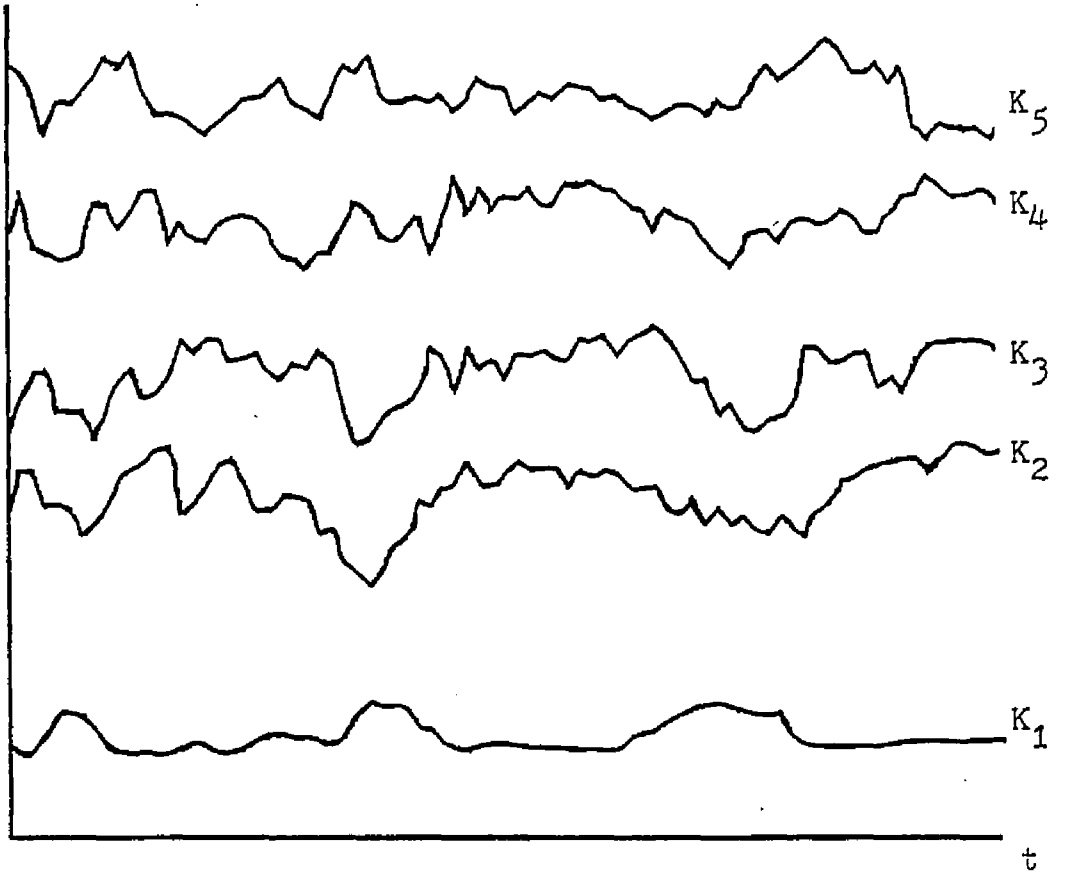


Figure 1. Reflection coefficients plot (K1-K5) for phrase "When will our yellow lion roar?"

order of 10, only 12 parameters need to be transferred for every frames (a frame is a time duration in which the speech waveform can be considered as a time-invariant signal). The block diagram of linear predictive coding scheme is shown in Figure 2. Note again the real speech waveform is not transferred, only the vocal tract modelling and source modelling parameters are.

An LPC Speech Synthesizer Realization on the TMS320

In LPC analysis/synthesis, there are many parameters that affect the quality of the synthesized speech. These parameters are sample rate, filter order, filter word length, and filter type. All these parameters influence the cost of the hardware system and the quality of the synthesized speech in a very complex way.

One of the current research activities in the Digital Signal Processing Laboratory in the Department of Electrical and Computer Engineering at the University of Arizona is to investigate the effects of the parameters on LPC speech quality, and hence derive optimum parameter sets for LPC synthesis. The goal is to provide contributions to the general understanding of LPC speech synthesis, while simultaneously evaluating the effects of the constrains imposed by integrated circuit implementation.

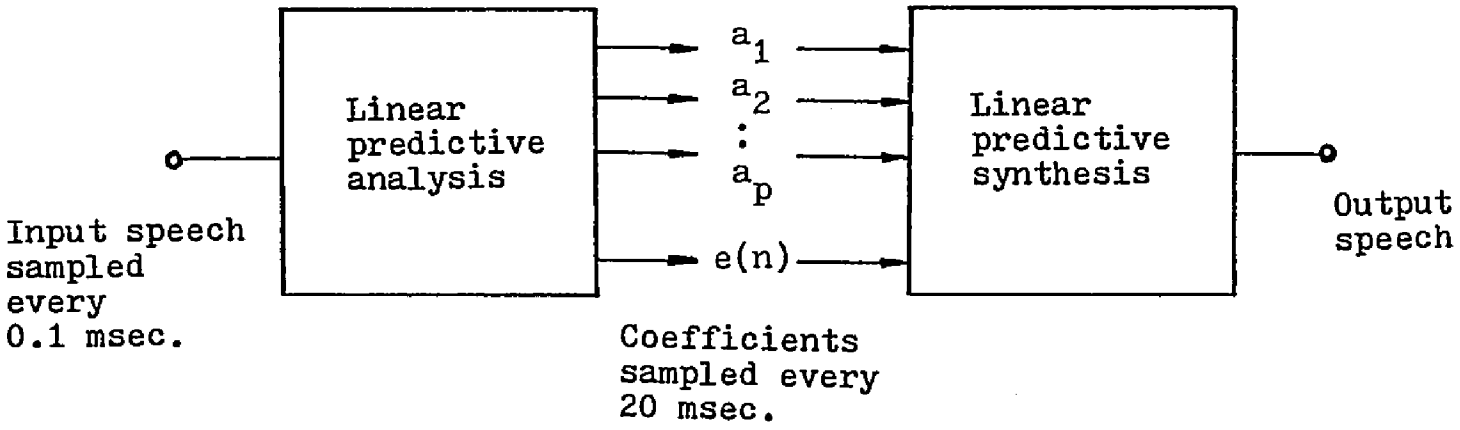


Figure 2. Block diagram of LPC analysis/synthesis

One part of this project is to implement a LPC speech synthesizer on a general-purpose digital signal processing chip - the Texas Instruments TMS320. The objective is to use this speech synthesizer as a convenient tool for evaluating speech synthesis schemes under different conditions in real-time. The synthesizer must meet many requirements. First, it will synthesis speech in real-time (this term could be interpreted rather losely here since a second of delay is of no concern). Second, the quality of the synthetic speech must be good so that any qualitative comparison of different parameter combinations will be meaningful. Third, system parameters can be modified. Three parameters - filter order, word length, and sample rate probably can be easily modified by changing the contents of some data registers. Modification of filter type involves change of algorithms, and this change is only possible on a general-purpose DSP chip. The reason for choosing it in our implementation becomes obvious now.

It turned out that the LPC speech synthesis system developed on a TMS320-based board level system (EVM320) has met most of the requirements. Although only one form of the LPC filters has been implemented, it will be shown that other types also can be implemented on the EVM320.

Chapter 2 will discuss the basic principles of LPC speech analysis/synthesis. In Chapter 3, a survey of IC implementation of speech synthesis is presented. Chapter 4 gives a overview of the TMS320's system architecture and some of its special features. The entire Chapter 5 devotes to the hardware/software implementation of a LPC speech synthesis on the EVM320 system. In Chapter 6, the synthesizer is evaluated. Also in this Chapter, a preliminary investigation on the effect of the linear interpolation of speech data file on the speech quality has been made. Finally, Chapter 7 gives the conclusion of the paper and offers some suggestions for further research.

CHAPTER 2

PRINCIPLES OF LINEAR PREDICTIVE CODING

One of the most powerful techniques in digital signal processing is the method of linear predictive coding (LPC) analysis. This method has become the predominant technique for speech analysis and synthesis. In this chapter, the basic principles of the LPC method are discussed. Also, as a form of the lattice methods, the strightforward two-multiplier lattice structure that is used for our implementation is examined in some detail.

LPC Principles

The LPC method is used here to implement a speech production model which simulates speech production of the human vocal tract. The significant feature of it is that the parameters of the speech production model can be uniquely defined by the LPC analysis of the short-term spectrum of the speech signal. The basic principles of speech production modelling and LPC modelling are briefly discussed in the following.

Speech Analysis and Synthesis Models

A linear speech production model was developed by Fant (1960) as shown in Figure 3. It consists of two sections; the analysis section, and the synthesis section. The original speech S(z) is the input to the analysis model A(z), while the synthesis model H(z) gives the output - the synthesized speech S'(z). The intermediate signal E(z) is called the residual. The assumptions behind the model can be found in Fant (1960) and Flanagan (1972).

The basic principle of the linear speech production model , as shown in Figure 3, is as follows. First, a speech signal S(z) is used to determine the analysis filter A(z) for a specific residual E(z). Second, the residual E(z) is used to excite the synthesis filter 1/A(z) to obtain the synthesized speech S'(z). Therefore, the synthesized speech S'(z) should be the same as the original speech S(z). These two steps can be represented by Eqrs. 2.1,

$$E(z) = A(z)S(z) \qquad (2.1a)$$

$$S'(z) = E(z)/A(z). \qquad (2.1b)$$

The criterion of LPC is to adjust the parameters of A(z) so that the residual E(z) has a minimal mean-square value. Since speech is time-varying in nature, this estima-

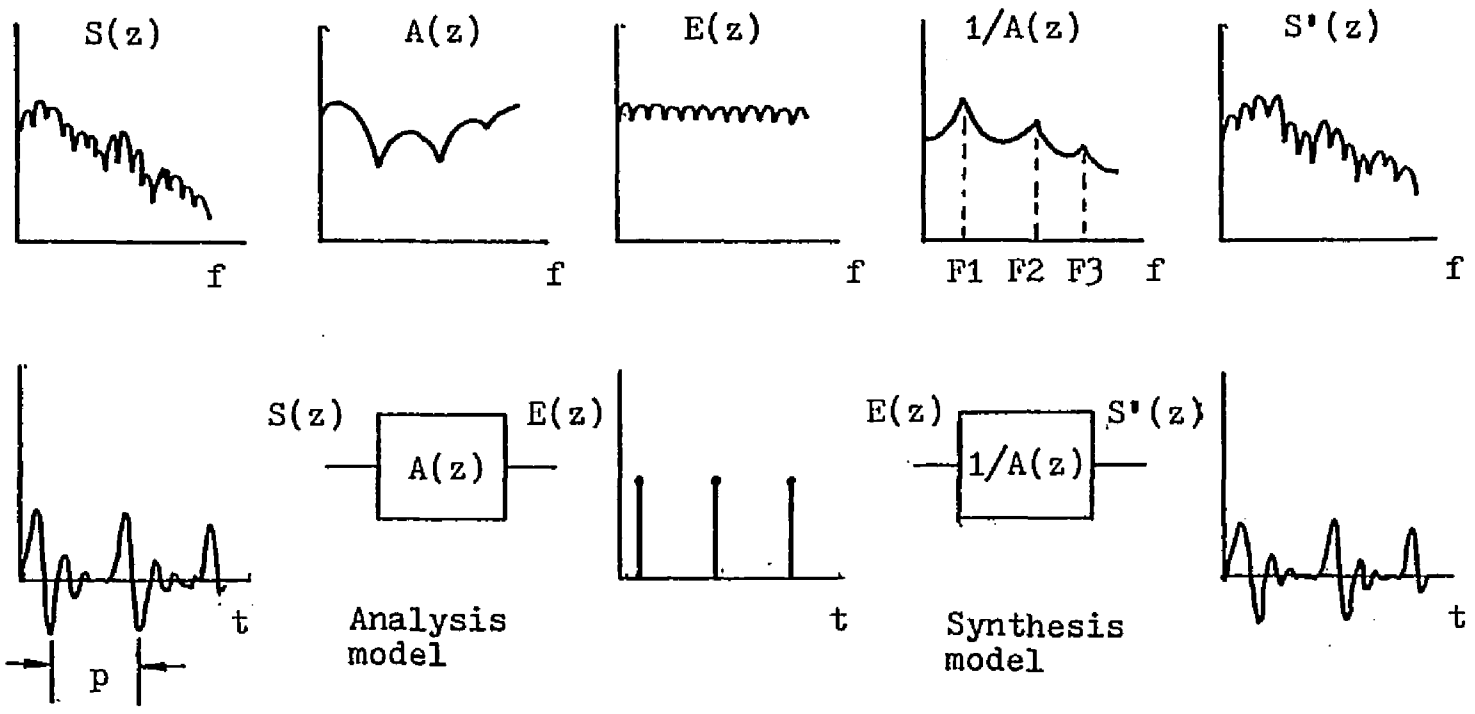


Figure 3. Speech production model (after Markel & Gray)

tion of A(z) must be performed in a short time period within which the speech can be viewed as a time-invariant signal. This time period is usually called a frame. It can be justified theoretically (L.R. Rabiner and R.W. Shafer, 1978) that E(z) has a minimal mean-square value if it is a flat-spectral signal. Therefore, the criterion is modified in order to obtain a set of parameters of A(z) for each frame so that the residual E(z) has a flat spectrum.

The reason why E(z) is constrained to be a flat-spectral signal is now readily seen. The synthesis filter H(z), which is equal to 1/A(z), is uniquely specified by the inverted parameter sets of A(z) obtained from the analysis stage. If the excitation E(z) for the synthesis filter H(z) is made to be a flat spectral signal, the spectrum of the synthesized signal S(z) will have the same shape as H(z). Therefore, the synthesis filter H(z) solely determines the spectrum of the speech output. This leads to an important observation: the filter parameters obtained from speech analysis can uniquely define the spectrum of the synthetic speech.

Therefore, the linear speech production process can be viewed as a coding/decodong method. The speech input is first coded into sets of parameters, which are transmitted or stored. Later , for speech synthesis, these parameters

are decoded to recover the original speech signal. These two tasks are accomplished by the ananlysis filter and systhesis filter, respectively. Since speech waveform is a slow time-varying signal, the rate for changing the filter's parameters is very low. By using this scheme, therefore, a high degree of data compression can be achieved.

All-poleModels

An all-zero model A(z) in Eqrs. 2.1 is defined as

$$A(z) = 1 - \sum_{k=1}^{p} a(k)z^{-k} , \qquad (2.2a)$$

and an all-pole model H(z) = 1/A(z) is defined as

$$H(z) = \frac{1}{1 - \sum_{k=1}^{p} a(k)z^{-k}} , \qquad (2.2b)$$

where [a(k)] 1<=k<=p are known as the predictor coefficients. The residual e(n) in the time domain is given by

$$e(n) = s(n) - \sum_{k=1}^{p} a(k)s(n-k) \qquad (2.3)$$

which is equivalent to

$$s(n) = \sum_{k=1}^{p} a(k)s(n-k) + e(n). \qquad (2.4)$$

If we define

$$s'(n) = \sum_{k=1}^{p} a(k)s(n-k) \qquad (2.5)$$

as the predicted value of s(n) from the previous p samples, then

$$e(n) = s(n) - s'(n) \qquad (2.6)$$

is the prediction error.

Using these notations, the linear speech production criterion can be posed as: find the coefficients [a(k)] that minimize

$$\sum_{n=-\infty}^{+\infty} e(n)^2 = \sum_{n=-\infty}^{+\infty} [\, s(n) - \sum_{k=1}^{p} a(n)s(n-k) \,]^2 \qquad (2.7)$$

for every frame. It has also been known that minimization of Eqr.(2.7) is equivalent to constraining e(n) to be a flat-spectral signal.

The two waveforms with absolutely flat spectra are a single impulse and a white noise. It is at least plausible that an impulsive excitation will have a small mean

power since most of the samples are zero. As mentioned in chapter 1, human speech can be classified into two types: voiced sound and unvoiced sound. The voiced sound is periodic in nature, while the unvoiced sound exhibites a randomness in the time domain. This observation leads to the following approach. For voiced speech, a set of parameters that gives an impulsive E(z) is the desirable parameter set for defining A(z). For unvoiced speech, the desirable set of parameters will be the one which gives a white E(z).

There are three popular methods to determine the parameters of the linear filter A(z) using LPC method: the autocorrelation method, the covariance method, and the lattice method. While the first two methods are widely used in general purpose computer implementation, the lattice method has been the most popular for linear predictive analysis/synthesis for commercial IC chips. This is due to the fact that only the lattice method guarantees system's stability. It will be shown in the next section that the prediction polynomials of a lattice filter, which are obtained from the partial correlation coefficients, satisfy the stability requirement for a digital system by definition. The autocorrelation and covariance methods, on the other hand, require less computation and memory storage than the lattice method (Rabiner & Shafer 1978). However,

the stablity of these methods is more susceptible to the roundoff errors caused by the finite word-length effect of digital sytems. Therefore, in most commercial IC chip implementations, in which the word length is usually short for economical reasons, the lattice method has been adopted for obtaining system stability.

The autocorrelation and covariance methods are not going to be studied here. The lattice method, which is to be used for our LPC algorithm realization, is discussed in some detail. The discussion will, however, be defered until after the lossless acoustic tube model has been introduced.

Source Model

The residual signal E(z) from the analysis filter A(z) has been used as the excitation for the synthesis filter H(z) in our previous discussion. It is very uneconomical since a large data space is required to store the waveform of E(z). In order to achieve any data compression, E(z) obtained from analysis has to be discarded and an ideal flat-spectrum model E(z) is used for synthesis. This source modelling certainly will introduce error in the synthesized speech, since E(z) obtained from analysis cannot have a truly flat spectrum. However, the large data reduction offsets the loss in speech quality for most applications.

The most common form of pulse source is the single impulse response of an all-pass filter. The pulse source produces a sequence of pulses separated by a pitch period to excite the synthesis filter, producing voiced speech. A pseudorandom number generator is generally used as an excitation for unvoiced speech.

The LPC speech synthesis model driven by modelled excitation is shown in Figure 4.

Lattice Filtering

The lattice filter is an important method of linear predictive analysis and synthesis. It guarantees the stability of the linear predictive polynomial even when the computation is performed using finite word length machines. This unique feature makes the lattice filter the most widely used method in LPC digital computer implementation.

Acoustic Tube Modelling

Acoustic tube models of speech production have been studied for many years by a numbers of researchers. Atal (1970) demonstrated that the formant frequencies and the bandwidths are sufficient to uniquely determine the area of an acoustic tube having a finite number of sections. He also showed that a given order transfer function uniquely defines the shape of a acoustic tube that models a human

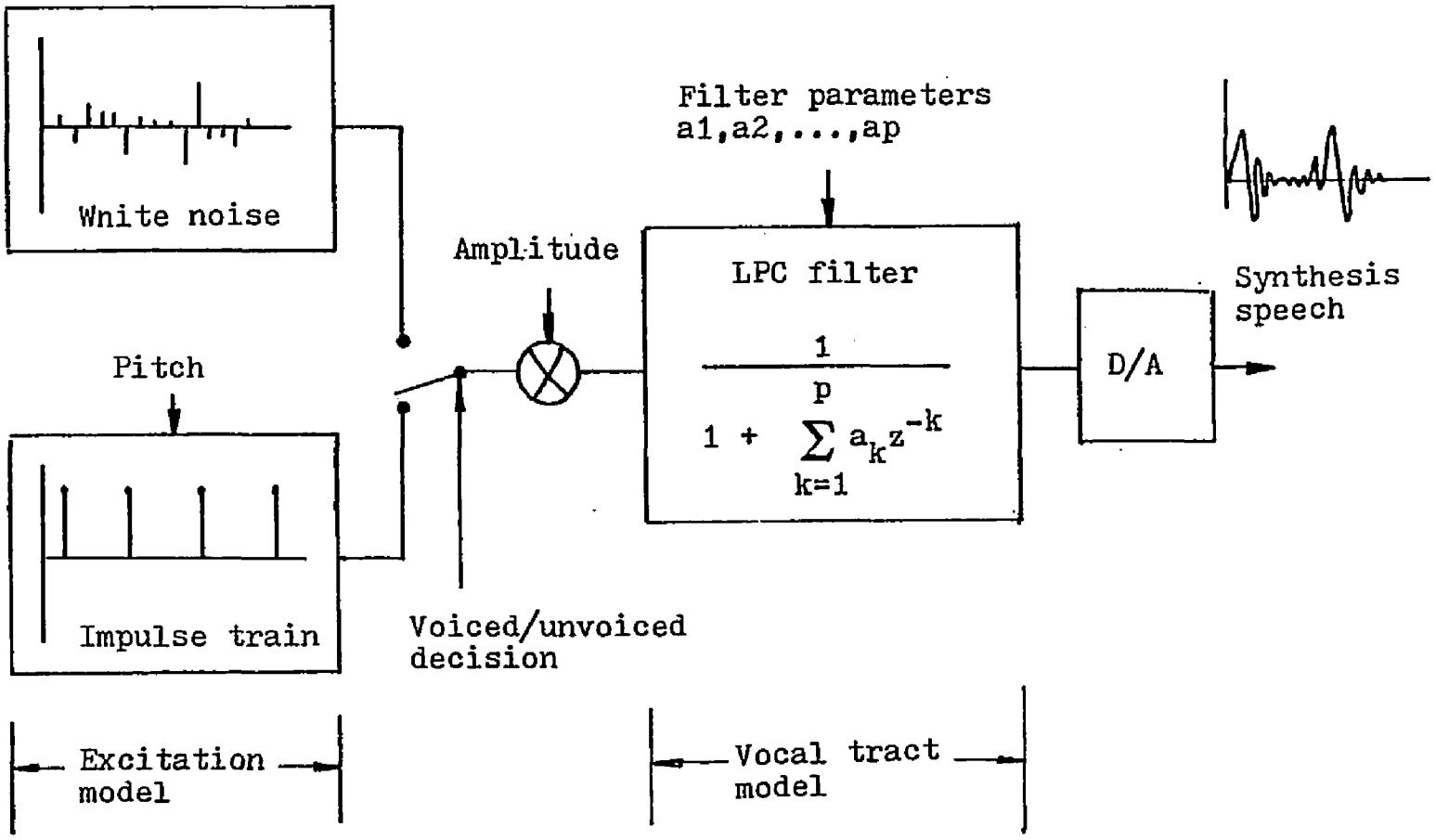


Figure 4. LPC speech synthesis model

vocal tract. Here, only the observations and the conclusions are going to be discussed, with the intention of showing the relationship between acoustic tube modelling and the lattice filter. Pertinent theoretical derivations can be found in the literature, and will be omitted.

The cross-section of a lossless acoustic tube is shown in Figure 5. The tube consists of p equal-length rigid cylindrical sections, each of which has a constant cross-sectional area A. The pressure and volume velocity in an acoustic tube obeys the momentum equation that further gives the Webster horn equation. Solving the Webster equation gives

$$U_m(x,t) = U_m^+(t-x/c) - U_m^-(t+x/c)$$
$$P_m(x,t) = P_m^+(t-x/c) + P_m^-(t+x/c), \qquad (2.8)$$

where superscripts '+' and '-' indicate forward and reverse travelling waves, respectively. The relationships are further shown in Figure 6 for the volume velocity.

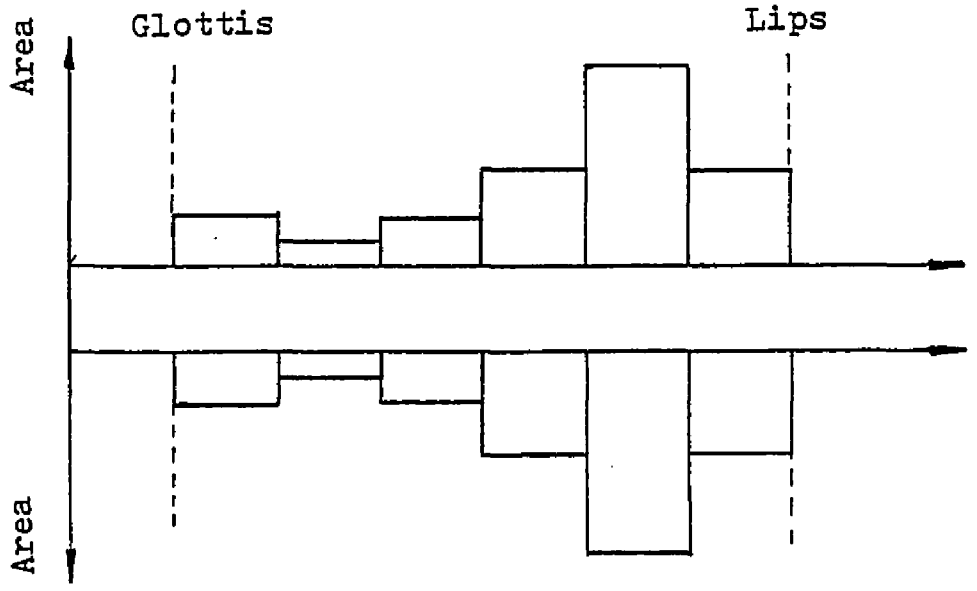


Figure 5. Acoustic Tube model (after Markel & Gray)

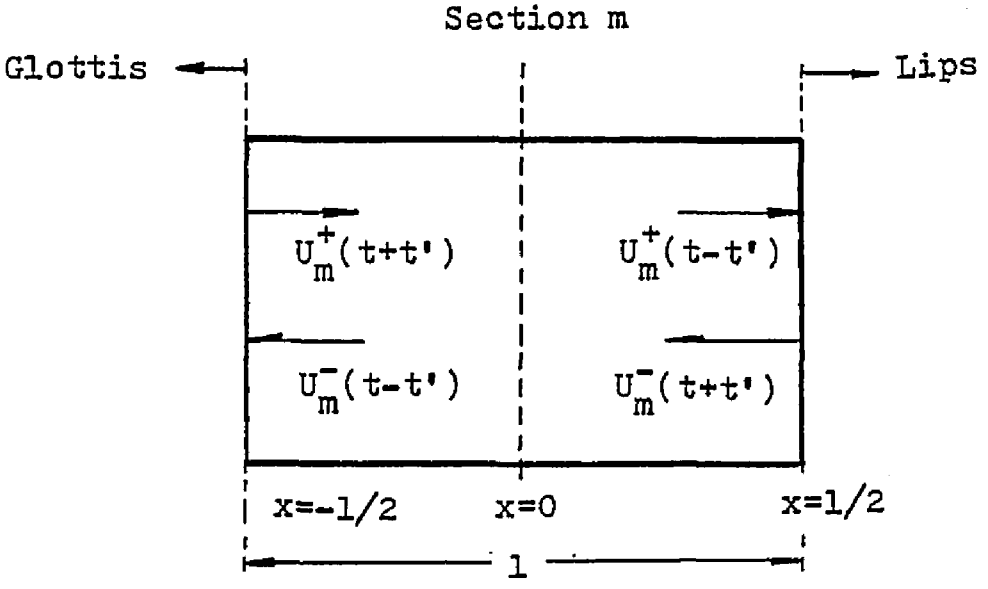


Figure 6. Volume velocity in section m of Acoustic Tube (after Markel & Gray)

Applying the continuity condition to Eqr.(2.8) results in

$$U_m(t-t') = \frac{U_{m-1}^{+}(t+t') - k_m U_{m-1}^{-}(t-t')}{1 + k_m} \qquad (2.9a)$$

$$U_m(t+t') = \frac{k_m U_{m-1}^{+}(t+t') + U_{m-1}^{-}(t-t')}{1 + k_m} \qquad (2.9b)$$

where

$$k_m = \frac{A_{m-1} - A_m}{A_{m-1} + A_m} \qquad (2.10)$$

is defined as the reflection coefficient.

Applying boundary conditions to Eqr.(2.8) gives

$$U_l(t) = 2U_0^{+}(t-t') \qquad (2.11a)$$

$$U_g(t) = \frac{2U_{m-1}^{+}(t+t') - 2k_m U_{m-1}^{-}(t-t')}{1 + k_m} \qquad (2.11b)$$

for lips termination and glottal termination respectively. If a new variable Y_m is introduced to offset the denominators in (2.9) , Eqrs(2.9) can be rewritten as

$$Y_m^+(t) = Y_{m-1}^+(t) + k_m Y_{m-1}^-(t) \qquad (2.12a)$$

$$Y_m^-(t+T) = k_m Y_{m-1}^+(t) + Y_{m-1}^-(t). \qquad (2.12b)$$

If the terms in Eqrs(2.12) are sampled at t=nT, then z-transforms of (2.12) are obtained as

$$Y_m^+(z) = Y_{m-1}^+(z) + k_m Y_{m-1}^-(z) \qquad (2.13a)$$

$$zY_m^-(z) = k_m Y_{m-1}^+(z) + Y_{m-1}^-(z) \qquad (2.13b)$$

for $m = p,\ldots 2,1$

Eqrs.(2.13) are the speech production equations using acoustic tube modelling. The acoustic tube models the human vocal tract by characterizing a set of time-varying reflection coefficients. Therefore, as far as the system output is concerned, any digital implementation that satifies Eqrs. (2.13) should produce the same speech in response to a given input. This leads to the lattice filter method, which is the most direct digital realization of the acoustic tube model.

Two-multiplier Lattice Structure

A two-multiplier lattice structure is shown in Figure 7. The signal flow equation in Fig. 2.5 can be readily obtained as

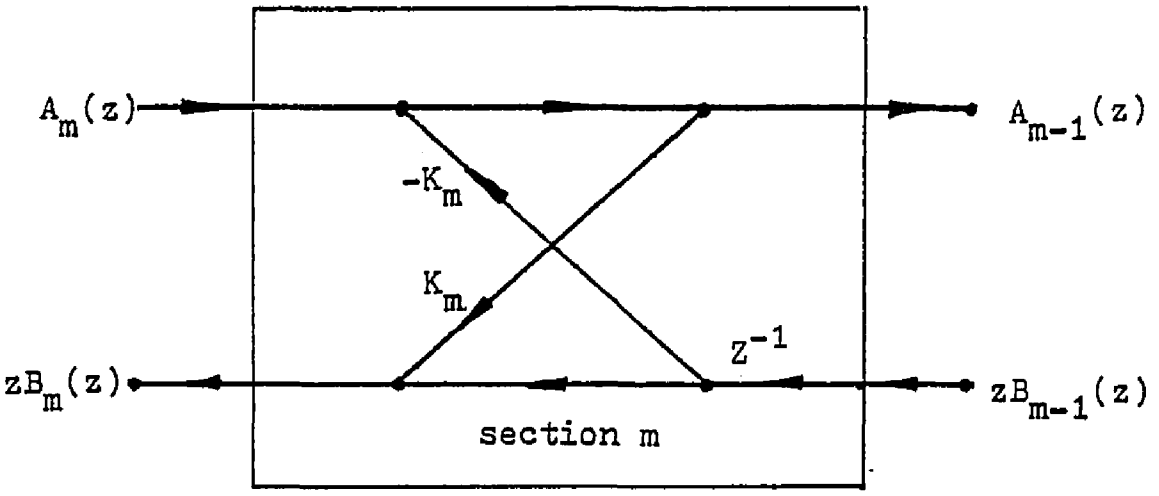


Figure 7. Two-multiplier lattice structure, section m (after Markel & Gray)

$$A_m(z) = A_{m-1}(z) + k_m B_{m-1}(z) \qquad (2.14a)$$

$$zB_m(z) = k_m A_{m-1}(z) + B_{m-1}(z) \ , \qquad (2.14b)$$

which are found to be equivalent to (2.13). The two-multiplier lattice structure is used in our speech synthesizer since it is rather simple to implement and very stable. Other commonly used lattice structures are the one-multiplier or the four-multiplier structures. There are some differences between these structures. The one-multiplier structure has obvious advantages over the other two if the structure is implemented in a system in which multiplication is time-consuming. However, the four-multiplier structure behaves substantially better in finite word-length implementations. The four-multiplier structure can achieve maximum dynamic range of a digital system since the energy at each stage of the lattice are preserved (Markel & Gray 1975). The two-multiplier structure is the most straightforward and simplest implementation of any of the lattice configurations. Compared to the four-multiplier structure, another advantage of using the two-multiplier structure is that it is less susceptible to the overflow problems in fix-point arithmetic realizations (Carlson 1985). The details of the one-multiplier and the four-multiplier structure are not going to be discussed in here.

Recall from the previous discussion that an all-pole synthesis filter can be defined in the frequency domain as

$$H(z) = \frac{1}{A(z)} = \frac{1}{1 - \sum a(k) z^{-k}} . \qquad (2.15)$$

It can be shown that lattice equation defined by Eqrs.(2.14) meets the general definition of a synthesis filter given by Eqr.(2.15). In particular, the first-order lattice equation

$$\frac{Xout}{Xin} = \frac{1}{1 + k_1 z^{-1}} \qquad (2.16)$$

is exactly the same as the 1st-order form in Eqr.(2.15) if k_1 is replaced by a(1). The 2nd-order lattice equation

$$\frac{Xout}{Xin} = \frac{1}{1 + k_1[k_2-1]z^{-1} - k_2 z^{-2}} \qquad (2.17)$$

also satisfies Eqr.(2.15), and so on. It is fairly obvious that a p-th order lattice filter will uniquely define a p-th order all-pole synthesis filter H(z). Thus, the two-multiplier lattice structure can be used to implement a synthesis filter for speech production.

Reflection Coefficient Estimation

Recall that the key problem in linear predictive coding is to choose coefficients [a(m)] for the analysis filter so that the residual output e(t) will have a minimal mean-square value. From Eqrs.(2.16) and (2.17), it is shown that [a(m)] can be represented by arithmetic combinations of reflection coefficients k_m of a lattice filter. Therefore, the criterion for choosing [a(m)] can equally be applied to choosing k_m That is, for each sections of the lattice filter, the reflection coefficients k_m (m=p,...1) are determined such that they will minimize the residual (prediction error) in that section. One remarkable feature about this scheme is that the calculation of k_m (m=p,...1) can be done locally at each section. Thus it avoids the complicated global optimization process.

Witten (1982) showed that taking the expected values of Eqrs.(2.14) and setting the derivative with respect to m to zero leads to

$$k_m = \frac{E[A_m(z)\,zB_m(z)]}{E[A_m(z)^2 + zB_m(z)^2]/2}, \quad m = 1,..p \qquad (2.18)$$

Note that Eqr.(2.18) guarantees that k_m is always less or equal to unity, which ensures that the all-pole synthesis filter will be always stable. This is an invaluable property in finite word-length digital machine implementation.

CHAPTER 3

IMPLEMENTATION OF LPC ON INTEGRATED CIRCUITS

The recent advancement of digital signal processing (DSP) is largely due to the breakthrough in the field of integrated circuits. Until less than a dacade ago, the manufacturing of VSLI devices in high volume and low cost made many DSP products such as the 'talking chip' technically as well as commercially feasible. LPC speech synthesis systems can be implemented in various IC chips with different levels of dedication - the general-purpose DSP chip, the semi-dedicated chip, and the dedicated chip. The characteristics of these chips and basic application requirements will be examined . Some special constraints for DSP implementation will be discussed as well.

LPC on Various Integrated Circuits

A general-purpose chip DSP chip is the most versatile, most powerful user-programmable single-chip signal processing IC. A semi-dedicated chip is usually a digital filter with user-controllable parameters. A dedicated LPC chip can be used for a specific LPC synthesis realization, leaving no room for user's modification. These three cate-

gories of IC chips will be briefly introduced, with examples of each of them.

LPC Using Digital Signal Processors

In recent years, a succession of DSP chips have appeared in the marketplace. These are essentially small number-crunching computers. As with ordinary microprocessors, they are programmable digital systems. Several special features, however, make them distinctively different from microprocessors. These features are: a fast hardware multiplier, a Harvard architecture, instruction pipelining, and a high speed on-chip RAM.

One advantage of using a DSP chip is that it allows the user maximum flexibility. The user develops his own algorithm, specifies data rate, I/O port format, and so on. The DSP's are especially appropriate in cases where a specific algorithm, not available in standard synthesizers, needs to be implemented. This allows the user to realize the system with a selectable quality/data-rate trade-off. The DSP chips are powerful devices, but they are relatively expensive and need supporting chips to accomplish tasks such as I/O, D/A, A/D, etc..

Example 3.1 The S28211 DSP Chip

Chapter 5 is devoted to discussion of the TMS320 DSP chip. Here we briefly look at some special features provided by the American Microcircuit Inc. S28211.

The S28211 has a microprocessor-like structure. A single-cycle parallel multiplier, pipelined with the ALU/accumulator, yields a high device throughput. The S28211 has many efficiency-enhancement features. Among them are a 256 word data RAM, two single-instruction executable I/O ports, and two index registers that can be incremented simultaneously. A special software feature worth-mentioning is that every instruction has two operators; one exclusively reserved for ALU operations. This design maximizes the possibility of parallel instruction execution. All instructions are executed in a single 250 nsec cycle.

LPC Using Digtal Filter Chips

The LPC all pole synthesis filter, which models the human vocal tract spectrum, has a smoothed curve with a number of formant peaks in the frequency domain. If we partition this frequency spectrum into separate formants and describe each formant by coefficients of a second-order filter, then LPC can be realizied by a cascade of these second-order filters. There are several advantages inherent in this implementation. First, the filter

coefficients for each second-order section are simply related to the center frequency and bandwidth of that resonance. Each of the sections could be thought of as an independent filter. Lattice filters, on the other hand, have no simple relationship between the filter coefficient and the resonance frequency. Second, only those filters whose parameters change drastically on frame boundaries need be updated since all filters are independent. This approach lowers data transfer rate.

Example 3.2 The SP025 Chip

The main features of the SP025 by General Instruments is shown in Figure 8. In LPC applications, the filter is energized by either an on-chip impulse generator or a pseudorandom noise generator. The magnitude and the rate of the impulses are adjustable. The parameter input is double-buffered so that up to two frames of data could be requested and stored by the SP025; thus increasing the throughput. The 16-bit data word is truncated to 7 bits for the A/D converter.

LPC Using LPC Synthesis Chips

Implementing LPC on a dedicated chip is probably the most straightforward and economical approach if the LPC chip fits the specific needs. An LPC chip is a direct

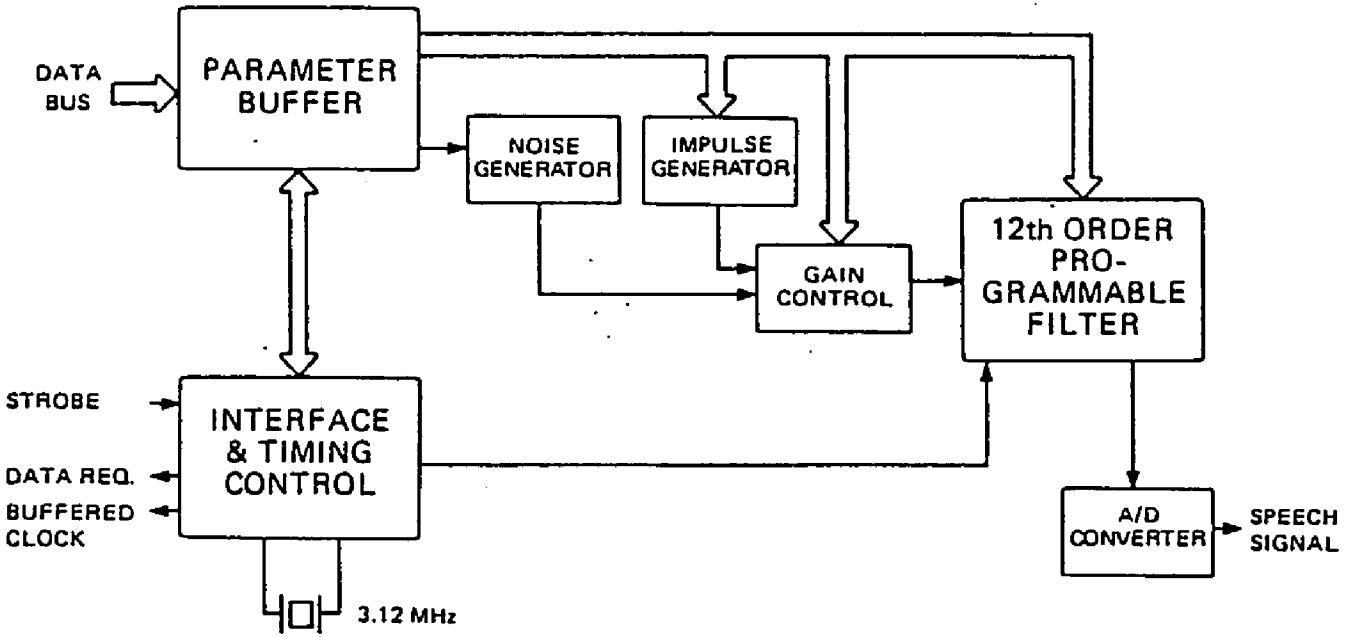


Figure 8. SP025 synthesizer block diagram

implementation of the LPC principles on a single IC. Of several configurations, the two-multiplier lattice version has been most widely used.

Example 3.3 The TMS0280 LPC Chip

The TMS0280 LPC, the heart of the 'Speak and Spell' by Texas Instruments, is a simple and powerful chip, as shown in Figure 9. The data coming in from outside are stored in a control register which decides whether they are control commands or synthesis information. Control data include such as 'repeat once', or 'zero energy', and they are used in special situations such as an occurrence of a continuation of two similar frames, or a silent frame. By using this scheme, the data rate is reduced from originally 2400 bps to 1200 bps (Morgan 1984). Furthermore, the coded data are stored in the RAM so that less space is needed. A multi-stage pipelining is used to reduce multiplication time to 5 microseconds, a big achievement for the inherently slow PMOS chips. To keep data rate low without losing significant speech quality, a linear interpolation is performed on each frame of parameters.

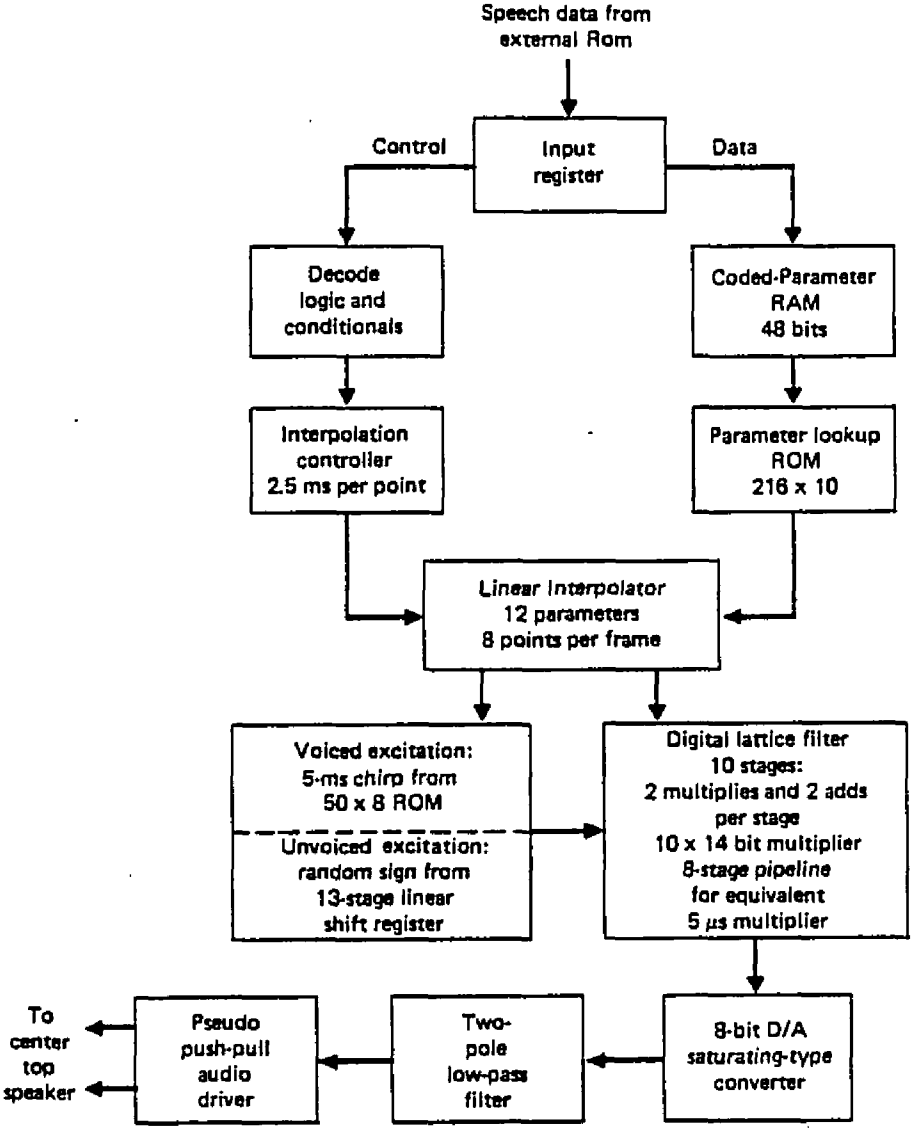


Figure 9. TMS0280 Speak & Spell™ chip block diagram

Requirements and Constraints for Implementing LPC on IC's

The user must be aware of the requirements and limitations for implementing LPC on digital integrated circuits. The requirements serve as the criterion for choosing IC chips for effective LPC realization. Unfortunately, not all the requirements can be met in most cases. The IC realization of LPC, or any other form of DSP, puts constraints on all aspects of system performance. Thus the user must make the compromises to guarantee that the realization satisfies the most critical requirements.

Requirements for IC Implementation

The most important hardware requirement for a DSP chip is that it must have a fast hardware multiplier. The number of multiplications for calculating each LPC sample is enormous. For a 10-pole two-multiplier lattice filter, at least 200,000 multiplications need to be performed every second for a synthesized sample output. Clearly a software multiplier routine taking many instruction cycles is not adequate for real-time application. Therefore, most DSP chips have hardware that does multiplication in a single instruction cycle.

To achieve high speed, given the limitations of a given IC technology, an efficient architectural design is crucial. One way to increase chip speed is the use of a

pipelining scheme. Pipelining refers to an interleaving operation of several instructions. Using pipelining, the three steps of a microprocessor-like DSP chip - instruction fetch, decode, and execution - can often be performed simultaneously. The TMS320 DSP chip to be discussed in Chapter 4 serves as a good example of pipelining.

DSP chips may also require some features such as a large memory, large word length, fast I/O ports, convenient software development, and so on.

Constraints for IC Implementation

In IC realization of DSP, compromises have to be made due to the many constraints that IC chips suffer from. The most important constraint is due to the effect of finite word length (FWL) of IC chips. The FWL effect influences the performance of a DSP chip in the following two ways.

First, FWL will result in truncation and rounding error. The word length of speech data in a DSP chip realization is bounded by the hardware design of the chip. The result of data processing, however, generally leads to variables that require additional word length. Multiplication of two b-bits long words for example, requires 2b-bits to store the product. The product must be quantized back to b-bits to avoid generally word length escalation. The two

common methods, truncation and rounding, are generally used.

Truncation of a number is done by discarding all bits less significant than the least significant bit that is retained. For two's-complement representation, the error in truncating a positive number satisfies the inequality: $0 >= X_t - X > -2^{-b}$, where b is the number of retained bits, and X_t is the truncated value of X. Rounding of a number is done by choosing b-bit number closest to the original quantity. For fixed-point arithmetic, the error resulting from rounding satisfies: $|X_t - X| <= 2^{-b-1}$. Both truncation and rounding cause a loss of data accuracy which becomes more severe when recursive algorithms are used. For a 10-pole LPC filter, synthesis of one speech sample needs about 200,000 truncations or roundings. Since the process is recursive, the accumulated error may be large enough to be detected by the human ear. Rounding is usually preferred in most recursive structures since the error produced has a zero mean value. Rabiner and Gold (1975) give a detailed discussion on this topic.

The second, more serious, result caused by the FWL effect is the arithmetic overflow problem. The difficulty is that overflow should be foreseen and prevented at the design stage; rather than modififying it when it occurs.

The system should be designed so that overflow is unlikely to occur under most normal operating conditions. For fixed-point arithmetic, overflow never occurs in multiplication if the operators are correctly scaled. Addition and subtraction, on the other hand, will cause overflow if the result exceeds the correct maximal FWL expression. Therefore, to prevent overflow, the signal levels at certain points in the calculation need to be checked and properly scaled. In the LPC realization, signal levels obtained from any addition or subtraction need to be checked. Usually, a number of magnitude adjustment factors are used to adjust the amplitude of the excitation signal. This, however, is achieved at the price of limiting the dynamic range of the excitation function of the digital filter.

Another constraint of IC implementation is the speed limitation of IC chips. This speed limitation can directly affect the complexity of the algorithm. To synthesize speech at a given sampling rate, the number of maximum allowable instructions is equal to the time between two sample outputs divided by the time for DSP-execution of one instruction. This limitation has a significant impact on the order of the LPC filter, since it consumes most synthesis time. For state-of-the-art DSP chips, an instruction cycle of 200-300 nsec can be achieved. This technology has

yielded many LPC speech implementations with a 10-12 pole, 10-12 KHz sampling rate compromise.

Recall from Chapter 1 that our LPC speech synthesizer is going to be used as an evaluation tool, and it must be versatile enough so that the result of various parameter combinations of a synthesizer can be easily tested by using this tool. Therefore, it is obvious that the synthesizer has to be built on a DSP chip since only DSP chips afford such capability.

One excellent candidate for our synthesizer implementation is the Texas Instruments TMS320. The TMS320 is a general-purpose DPS chip. It is fast, powerful, and versatile. It has a 200 nsec machine cycle - the shortest among today's DSP chips. It is a very powerful real-time numerical computation capability. Futhermore, its instruction set is close to the microprocessors', thus programming is relatively easy to the user who is familiar with microprocessors.

In the next chapter, the system architecture, the special features of the TMS320 will discussed in detail. It will be shown there that the TMS320 is indeed a suitable IC to meet our design goals.

CHAPTER 4

THE TMS320 DIGITAL SIGNAL PROCESSING CHIP

The architecture of the Texas Instruments' TMS320 general-purpose digital signal processor is discussed in this chapter. Also, some of the TMS320's special features favorable for speech processing will be examined in detail. A complete TMS320 instruction set is given in Appendix A.

Architecture Overview

The TMS320 DSP chip utilizes a modified Harvard architecture for both speed and flexibility. This architecture allows data transfer between program memory and data memory, thereby increasing the flexibility of the device. A hardware-intensive approach in system design has been taken to achieve a very high executing speed.

System Architecture

As shown in Figure 10, the TMS320 has a modified Harvard structure in which the program and data memories lie in two separate spaces. This provides a full overlap of instruction/data fetch and execution. Program memory can reside on both on-chip and off-chip RAM. The maximum addressable space is 4K 16-bit words. An on-chip 114x16 RAM

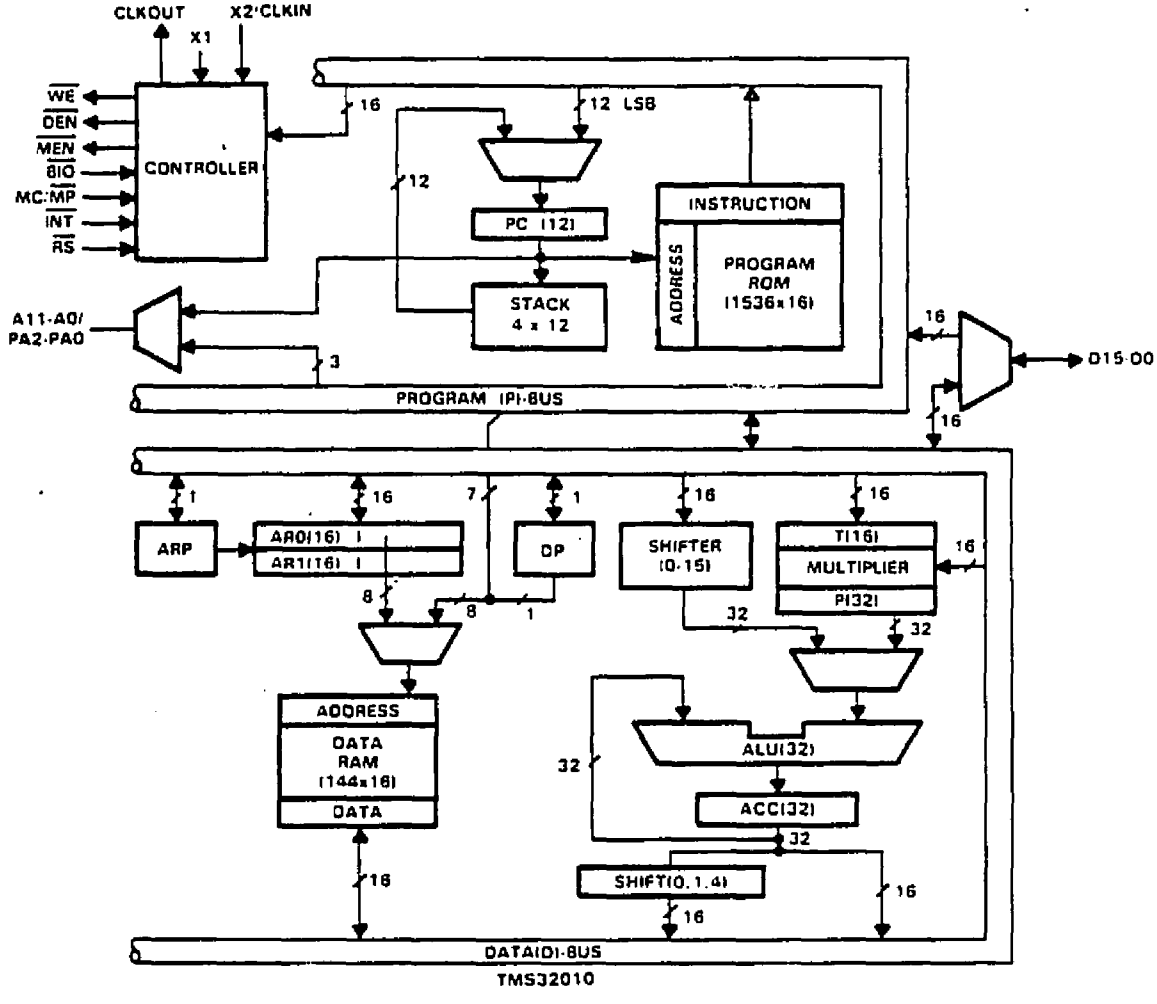


Figure 10. TMS320 architecture

serves as a data memory. Although, instruction operands are fetched from the data RAM only, data can be read into the data RAM from a peripheral port using an "IN" (input) instruction, or from the program memory using a "TBLR" (table read) instruction. The reverse data flow can be done by using "OUT" (output) and "TBLW" (table write) for sending data to the peripheral port and program memory, respectively.

The TMS320's instruction pipelining is shown in Figure 11. The instruction fetch and instruction execution are overlapped as follows: on the falling edge of CLKOUT, the program counter (PC) is loaded with the instruction (load PC2) for the prefetch while the current instruction (execute 1) is decoded and starts to be executed. Then the next instruction is fetched (fetch 2) while the current instruction is executed (execute 1), and so on.

Arithmetic Elements

There are four basic arithmetic elements; the arithmetic-logic unit (ALU), the multiplier, the shifters, and the accumulator. All operations are performed using two's-complement arithmetic. Most arithmetic instructions will access a word in data RAM, and pass it through a 0-15 bit shifter to the ALU. After calculation, the shifted result will be stored in data RAM. Accumulator (ACC) condi-

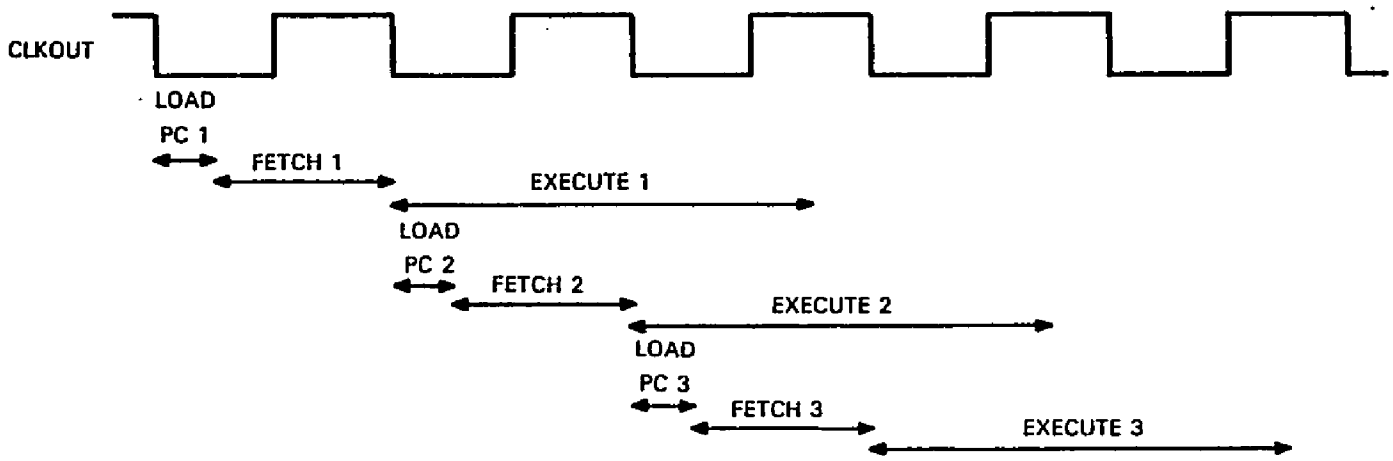


Figure 11. TMS320 instruction overlapping

tions can be tested by the branching instructions, as in most microprocessors.

If the overflow mode register is set (SOVM), the largest or smallest value will be loaded to the ACC when overflow occurs. This is a useful feature for handling overflows.

The multiplier consists of three registers; T register, P register, and Accumulator. To multiply, the multiplicand is first loaded to the T register, followed by execution of the "MPY" instruction. It is important to note that the product in the P register needs to be transferred to the accumulator immediately after "MPY". This is to ensure that an incoming interrupt call will not destroy the results.

Memories and Registers

Besides the on-chip 1536x16 program RAM, the 12-bit address line allows a total of 4kx16 program memory space. If the TMS320 is operating under the microprocessor mode, which is the only mode available for our model, the whole 4k program memory is external to the TMS320.

Data memory consists of 144x16 words of on-chip RAM. All non-immediate operands reside in this RAM. As mentioned before, two pairs of instructions; "IN/OUT", "TBLR/TBLW", allow data transfers between peripherals

(PORT) and data memory, and between program memory and data memory, respectively. Three memory addressing modes; immediate, direct and indirect addressing, are provided. In the immediate mode, the operand is fetched from the program memory, while the operand is fetched from the data memory in the other two modes.

There are two 16-bit auxiliary registers that can be used as loop counters or address pointers. The auxiliary registers can be auto-incremented/decremented.

A four-layer-deep stack is pushed or popped by "PUSH" or "POP" instructions. Although there is no stack overflow check provided, some instructions allow data transfers between the stack and data memory. This feature provides the user a convenient way of expanding the stack if necessary.

Special Features

The TMS320 has meny special features that make it a unique DSP chip. These features are favorable for high-speed implementations such as speech analysis and synthesis.

The Modified Harvard Architecture

A strict Harvard architecture is a machine structure in which separate address spaces are used for program

and data memory. The TMS320, howwver, uses a modified Harvard architecture. The modified Harvard structure has several advantages for DSP implementation.

First, instructions and data can be fetched simultaneously by using separate buses for each. This data/program parallelism ,which is a typical approach for DSP chips, is the key to the machine speed increase.

Second, data crossover between the data memory and the program memory is possible by using some special instructions. This feature offsets the drawback of TMS320's relatively small data RAM, since data can be transferred to data RAM at execution time. This gives the user maximum flexibility in program/data space allocation. For instance, tables of constants can be stored inside the program memory, and later retrieved to the data memory during execution.

Hardware Intensive Approach

Since the TMS320 was designed largely for real-time signal processing, efforts were made to increase its execution speed. A hardware-intensive approach had been taken to achieve this goal.

The 16x16-bit hardware multiplier consists of a P register, T register, and the multiplier array. A multiplication can be accomplished in 400 nsec.

Two shifters are used for manipulating data: a barrel shifter for shifting data from the data RAM into the ALU and a parallel shifter for shifting the accumulator into the data RAM. These hardware shifters will shift data simultaneously with data transfer between the data memory and the accumulator. Therefore, no additional shift is needed after data have been transferrd.

A few instructions can perform several arithmetic tasks within a single instruction cycle. Instruction "LTD", for example, will load the T register, add the P register to the accumulator, and move the word in data memory to the next higher location. Here, the TMS320's data processing ability is clearly seen. The "LTD" instruction is designed for performing discrete convolutions; it will accomplish one convolution calculation in just one instruction cycle.

Another noteworthy feature is that fast 32-bit arithmetical operations can easily be performed. Addition and subtraction can be done on either the upper or lower half (16-bit) of the accumulator. The operation on the lower half of the accumulator (ACCL) can be sign-suppressed for 32-bit arithmetic. Thus the unsigned value in ACCL, concatenated with the signed value in ACCH, readily expresses a 32-bit word.

Overflow Control

In digital signal processing, overflow can create special problems. Overflow causes data swing between very large and very small numbers, thus resulting in erratic values. The TMS320 has been designed with a special overflow control mode to compensate for this problem. When the overflow mode is set by the user, the accumulator will be automatically loaded with the most positive or negative number if an overlow or an underflow does occur. This feature, which models the saturation inherent in analog systems, eliminates erratic data swings.

The overflow status in the accumulator can be also checked by using a "BV" (branch on overflow) instruction. This feature is particularly useful if the user needs to check the correctness of the processed data at some critical points.

The TMS320 Instruction Set

The TMS320 has a relatively rich instruction set. The TMS320 instruction set supports both numeric-intensive operations such as speech analysis and synthesis, and general-purpose operations such as high-speed control. The instruction set is shown in Appendix A. Most instructions are single-cycle, single-word; executable at a rate of 5 million instructions/sec. Only infrequently used branching,

subroutine call/return, and I/O instructions take two instruction cycles. As mentioned before, data shift is done along the data transfers in a single cycle. Therefore, two operands are needed for these instructions; one for specifying the number bit to shift, the other pointing to the data memory space.

Data operands are all 16-bit words. The program counter (PC) is 12-bits long, addressing a total of 4k program memory space. Two auxiliary registers (AR1,AR2) are also 16-bit long. Only the lower 8 bits of the registers are meaningful for indirect address usage, since an 8-bit value is sufficint to address the 144x16 word data memory.

CHAPTER 5

IMPLEMENTATION OF LPC ON THE TMS320

The LPC speech synthesizer is implemented on the EVM320 board. The EVM320 (EVM) is a digital signal processor evaluation module. Connected with a host computer, the TMS320 provides the user with an economical and efficient way to develop and test his program. The speech synthesizer is written in TMS320 assembly language, which is assembled by an EVM on-board assembler, and executed by the debug monitor.

The Hardware System

The EVM320 is connected with the Analog Interface Board (AIB), which works as an I/O device. The EXORmacs computer system, by MOTOROLA Inc., in Room 214 of the Engineering Building was used as a host computer. Program and data files are created and stored in the EXORmacs, which down-loads files to the EVM/AIB for execution.

The EVM Board

The EVM is a single board development system for the TMS320 DSP chip. The EVM can stand alone as a single system, using an on-board text editor for TMS320 assembly

file creation. Alternatively, it can accept files from a host computer through one of the EIA ports. In both situations, the resident assembler will convert the incoming text into executable object codes in one pass. The object codes are stored in a 4k-word memory area - the maximum TMS320-addressable program memory space.

The EVM operating system resides in four EPROMs and consists of four sections: debug monitor, assembler/reverse assembler, text editor, and TMS2764 EPROM utility.

The operation of the TMS320 on the EVM is controlled by a TMS9995 microprocessor that uses a system of logic to control the various sections of the EVM. The TMS9995 has access to 32k-bytes of EPROM (4 TMS2764s), that contains the firmware. An 8k-byte RAM is accessible by both the TMS320 or the TMS9995, so that object codes are loaded into it and then executed by the TMS320. Additional RAM exist for program execution initialization and breakpoint settings. Figure 12 shows the EVM memory map.

The EVM is designed so that the TMS320 is always executing codes, either Non-Operations (NOP) or a user program on a target system. A user program is executed by first loading the program into the TMS320 program memory 4k-word RAM, setting a start point on the program counter (PC) and breakpoints (BP), and then executing by "EX".

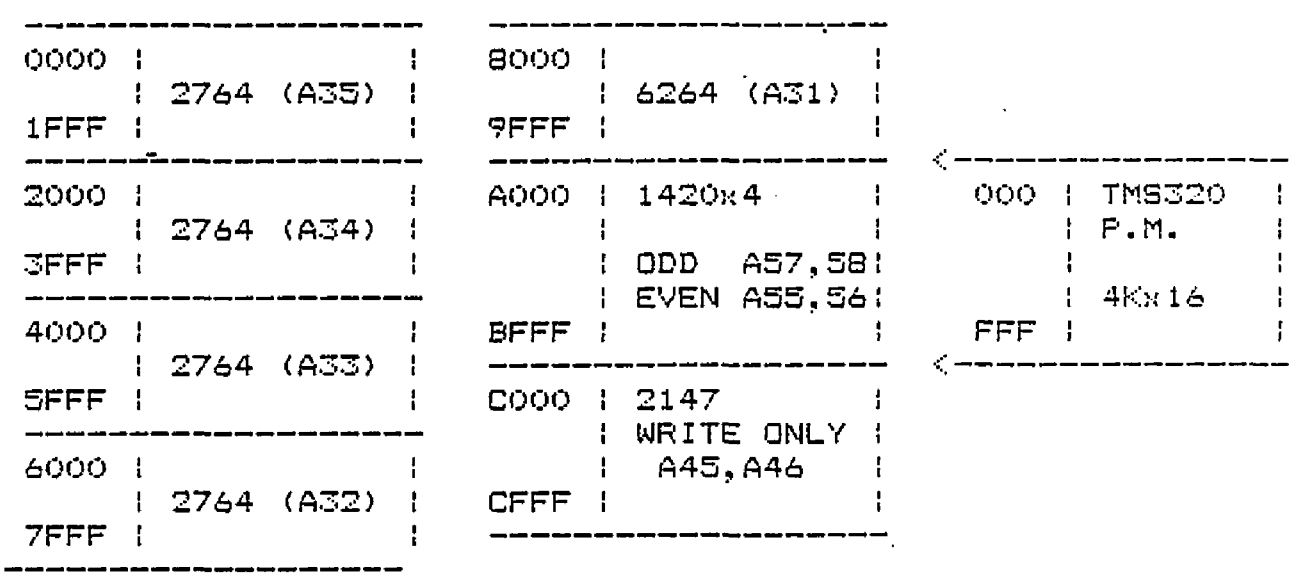


Figure 12 EVM Memory Configuration

The Analog Interface Board

The Analog Interface Board (AIB) is connected to the EVM as a target control system. The AIB consists of a 12-bit A/D, a 12-bit D/A, several data buffers, two low-pass filters, and a sampling clock. The AIB performs I/O operations under full control of the EVM. The TMS320 senses the status of the AIB by two input lines: ©INT, and ©BIO. They represent an interrupt call and an I/O ready, respectively.

The AIB's operation mode and sampling rate are user-controllable. The user can select desired sampling rate and one of the three AIB's operation modes by appropriately initializing the mode and the clock registers through the EVM output ports, PORT1 and PORT2.

Transferring data between the EVM and the AIB is done by using instructions "IN" or "OUT". The three least significant bits of the memory address bus are multiplexed to address up to eight I/O ports. Executions of these instructions by the TMS320 will cause the ©DEN and ©WE to be active, thus initiating the AIB's I/O operation.

The AIB has been modified to suit our specific implementation needs, and the modification is explained in detail in Appendix B.

Interface between EVM and EXORmacs

As mentioned before, the EVM board can assemble program files from one of three sources: an on-board text editor, a host computer, and an audio tape recorder. Assembling a program from the on-board editor is laborious since the text has to be entered line-by-line by hand. Entering data by the audio tape recorder has proven to be very unreliable. For our speech synthesis application, it is most convenient to use a host computer to create and store programs and data files. An RS232 cable connects port 2 of the EVM and one of the terminal ports, #CN13, of the EXORmacs computer. Port #CN13 is the only slow baud-rate port on the EXORmacs. It is found that the EVM's on-board assembler can not assemble incoming codes properly if the data rate is higher than 2400 bps.

The EVM provides a transparency mode operation that allows communication between the host computer and the EVM by using only one terminal. By toggling a toggle character (^C), the terminal switches its connection from one operating system to the other. When the EVM is in the transparency mode, it will transfer any ASCII code from the keyboard to the host as if by direct connection. Unfortunately, it will not work for the EXORmacs. The EXORmacs uses a "Break" key as a log-on command that cannot be transferred

by EVM, since "Break" is not a regular ASCII code. One way to solve this problem is to use a direct path to temporarily connect the EXORmacs to the terminal during log-on, leaving the other path regularly connected to the two through the EVM. Figure 13 shows the EXORmacs-EVM connection schematic.

A data path transferring synthesis parameters needs to be established between the EVM/AIB and its host. Input parameters such as speech pitch length, gain factor, and reflection coefficients are stored in the data file in the EXORmacs. During program execution, the parameters are sent to the EVM/AIB data buffer from the printer port #PR2 of the EXORmacs by a 36-pin Centronix cable also shown in Figure 13.

The EVM's operating system firmware was modified to fit the EVM - EXORmacs interfacing requirements since the two systems have a different data format and baud-rate. The modification was accomplished by using the system EPROM utility. Appendix B provides the modified firmware codes and their locations in the system EPROM.

Software Development

The LPC speech synthesizer is a piece of software written in the TMS320 assembly language. The software development includes LPC system modelling, flowcharting, and

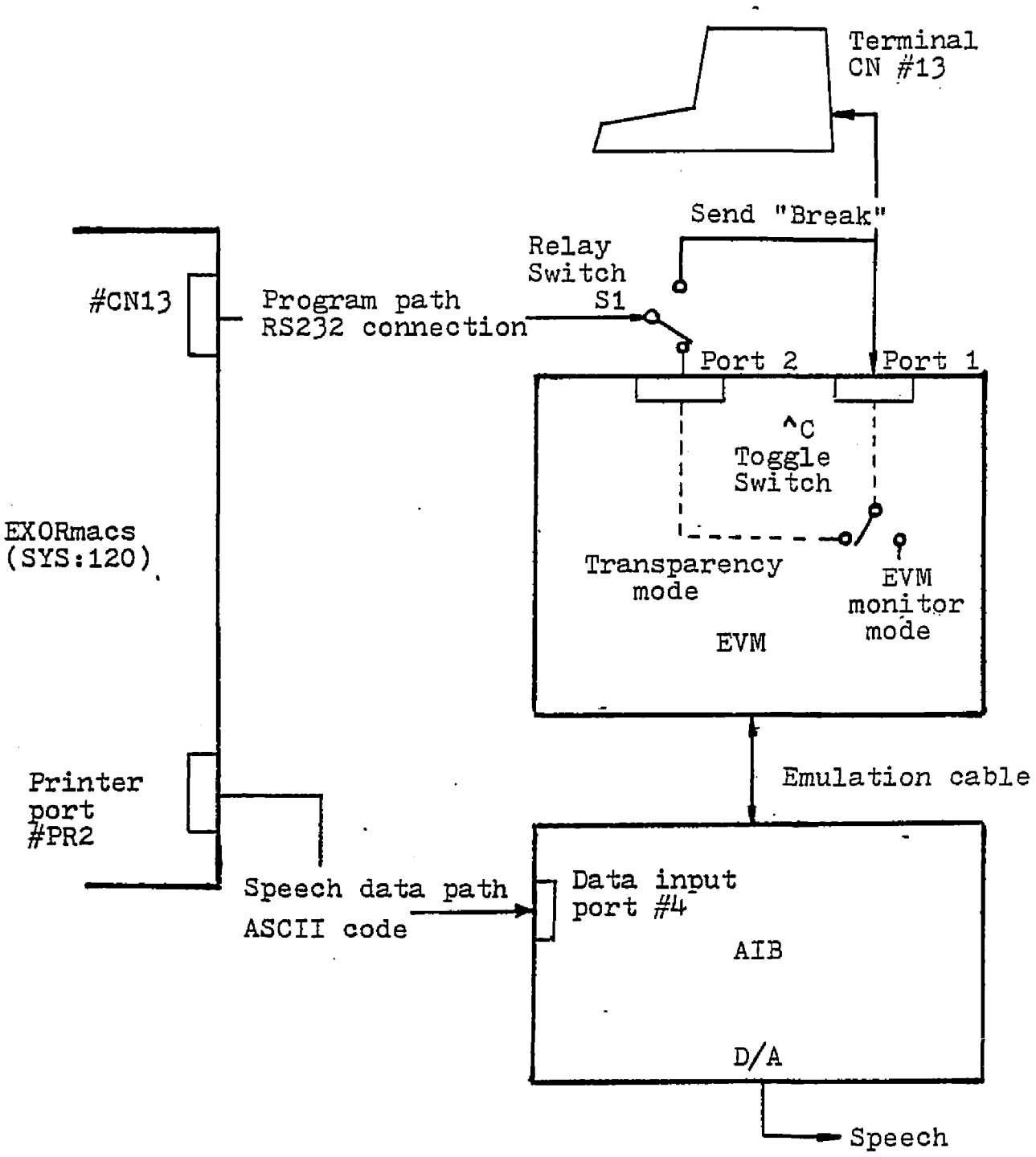


Figure 13. EVM/AIB - EXORmacs connection diagram

finally, assembly language coding. The LPC synthesizer source code is down-loaded into the EVM for assembly into machine codes for execution.

LPC Speech Synthesis Modeling

As discussed in Chapter 2, the LPC speech synthesizer models the human vocal tract with an all-pole filter that is excited by either a sequence of impulses for voiced speech or a sequence of pseudorandom numbers for unvoiced speech (see Figure 4). The filter is defined by a set of reflection coefficients K_p, (p = 1,2,...,M), while the excitation source is defined by the gain G and the pitch period P. All the parameters are updated for each frame.

For voiced speech, G and P define the excitation gain and pitch period, respectively. For unvoiced speech, only G is used to determine the gain of the excitation since the period for an unvoiced pitch is calculated by the program (to be discussed later). The voiced/unvoiced decision switch is incorporated into the period P: a negative P indicates an unvoiced frame. Similarly, a negative gain indicates the end of the parameter file, thus terminating the process.

The LPC all-pole filter is realized by the two-multiplier lattice structure, which is uniquely defined by a set of reflection coefficients K_p. The K_p's are 16-bit

words, and scaled between -1 and +1. Figure 14 shows the format of a set of parameters with a system order of 10.

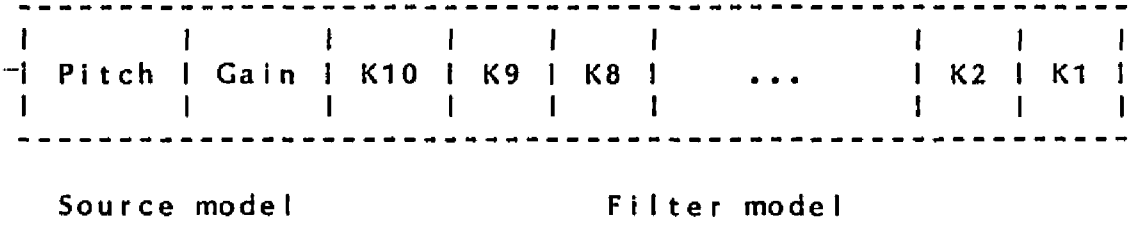


Figure 14. LPC parameter set format

There are three user-definable gain control adjustment factors: EAF, NAF, and GAF. They are used to adjust the gains of different sections of the synthesizer separately. This process is used to achieve maximum system dynamic range while avoiding arithmetic overflow. EAF (excitation adjustment factor) adjusts the gain of the excitation source at the very beginning of the process, NAF (noise ratio factor) adjusts the magnitude ratio of the voiced/unvoiced excitations, and GAF is the total system gain adjustment factor.

In order to accomplish better results, an asynchronous impulse excitation scheme has been used. Using

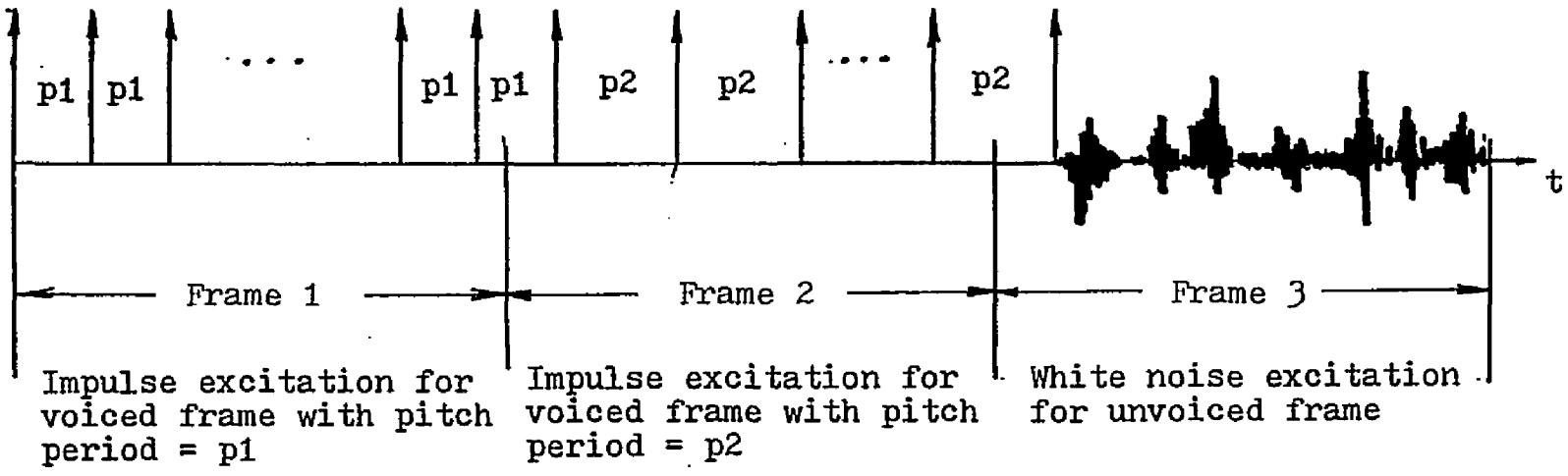


Figure 15. Asynchronous impulse/white noise excitation

this scheme, the impulses will have the same pitch length until after it crosses the frame boundary. Thus the asynchronous scheme avoids sudden pitch changes at the frame boundary. Figure 15 shows how impulse/white excitations vary in speech frames.

Synthesizer Flow Diagram

The system flow diagram for the speech synthesizer is shown in Figure 16. Appendix C provides the program listing of the synthesizer written in the TMS320 assembly language. The functions of the blocks in Figure 16 are discussed in the program flow order.

First, the system data memory must be initialized. This process includes data buffer declaration, data/program memory pointer initialization, and I/O port, system clock specification. Furthermore, a few user-definable values are loaded into the data memory by the user.

The subroutine 'Input data from EXORmacs' inputs an entire phrase of speech from the EXORmacs into the EVM's program memory. Note that the data input process will be done before the speech synthesis process begins. This design eliminates interleaving of the two processes. Although this design causes a small time delay, it seems to be a reasonable solution for the given hardware conditions (more on this in the next section).

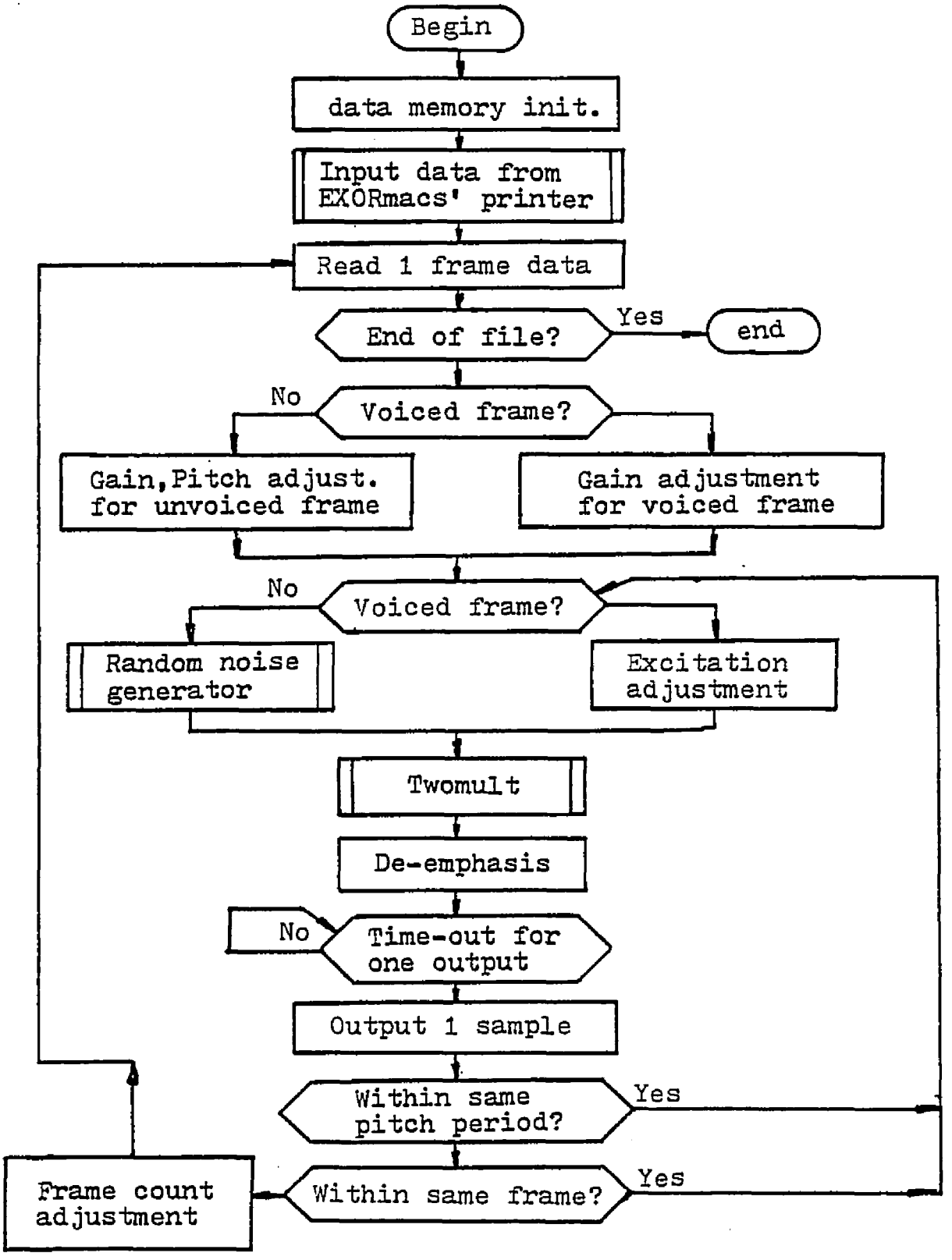


Figure 16. LPC speech synthesis flow diagram

After the entire speech data has been stored in the program memory, the program fetches the first set of data, as depicted in Figure 14, into the data memory. G and P are checked for either the end of the file or voiced/unvoiced conditions.

If the frame is voiced, a train of impulses with magnitude G and pitch period P will be used to excite the filter. For each pitch period P, there is one impulse of magnitude G and P impulses of magnitude -G/P. The arithmetic sum of the excitations over a pitch period, therefore, equals G + (-G/P)P = 0, thus eliminating the DC bias of the excitation source.

If the frame is unvoiced, a train of pseudorandom numbers of G*NAF magnitude will be used to excite the filter. The generator produces pseudorandom numbers by multiplying two 16-bits numbers, and taking the upper 16-bit number from the 32-bit product. Overflows/underflows are intentionally permitted for maximum unpredictability of the result. The 16-bit result is stored as the seed for generating the next random number.

The two-multiplier lattice filter subroutine is the heart of the synthesizer. It consumes about 3/4 of the synthesis time. The subroutine will perform one addition, one subtraction, and two multiplications at every section,

and repeat p times for a pth-order filter. A buffer of size p+1 is needed for storing the backward residuals B_p of Eqs. 2.14. This occurs because from Eqs. 2.14 it is clear that B_p's of the previous sample need to be stored for computing B_p's of the present sample.

After the sample has been calculated, it will go through a de-emphasis filter that low-passes the sample in order to reduce the high-freqency noise.

The system synchronization is achieved by using a loop that senses the AIB's counter by the ©BIO line. A time-up signal will let the program break the loop and produce a synchronized output.

If the sample just being synthesized is neither the last one in a pitch period nor in a frame boundary, the filter will be excited by another impulse; either from the random number generator or from the impulse generator.

Finally, if the sample is the last one in the frame, the number of samples past the frame boundary is recorded, a new set of synthesis parameters is fetched, and the entire process is repeated.

The TMS320 Assembly Language Implementation

The process of implementing the synthesizer using the TMS320 assembly language is more than a direct code translation from the flow chart. Problems associated with real-time programming have to be solved and compromises have to be made.

The most critical issue of any real-time signal processing is the system speed. For speech processing, the sampling rate of the speech signal and instruction cycle of the system determine the number of instructions allowed to be executed between two samples. As mentioned before, the TMS320's instruction cycle is 200 nsec. This time period allows execution of a maximum of 500 instructions for processing speech signals of 10 kHz. Therefore, the number of instructions needs to be minimized to ensure that the time constraint is met.

Another issue deserving special attention is the system I/O design. It was found that the analog interface board (AIB) needs modifications to meet our needs. The system I/O design for the modified AIB is discussed below.

The system output can be easily accomplished by enabling the AIB's synchronous output mode. Two registers in the AIB need to be initialized to the desired value before operation. One is the system clock rate reduction

ratio, while the other is the output operation mode. For our particular application of synthesizing 10 kHz speech, the timing register needs to be set to a value such that the system ©BIO line will be activated at a 10 kHz rate. The program will wait until the ©BIO becomes active, thus a synchronized output is obtained. This is illustrated as follows:

```
                ---                         One output sample
                ---                         is ready at this point.
WAIT        BIOZ        OUTPUT              Wait here if the time
                                            is not up yet.
            B           WAIT
OUTPUT      ---                             Now ready to output
            ---                             one sample.
```

The system parameter input is a more complicated process. This is partly due to the fact that the input data from the host computer are ASCII codes. The ASCII codes need to be unpacked into 16-bit hexa-decimal words by the interrupt-driven input routine. The original design was to interleave the main routine with the input service routine, assuming there is enough time for the service routine to complete the data input process in a 0.1 msec interval. Unfortunately, the printer port of the EXORmacs system has a very low data rate (specification is unavailable). The tests showed that the time delay due to the input process caused intermissions in speech output.

Exploiting the fact that the EVM has a program memory large enough to accommodate both the synthesizer machine codes and the data file codes, a somewhat less elegant approach has been adopted to input data from the host. Using this approach, the speech data file will be received, converted into HX16 word, and stored in the EVM's program memory by a "TBLW" command before the main routine is executed. After the data have been stored, the main routine can retrieve them by using a "TBLR" command. This is illustrated in Fig. 16. An obvious drawback to this approach is that a one-time short delay is inevitable.

Since a fixed-point algorithm is used for implementing the speech synthesis routine, the problems of arithmetic overflow/underflow deserve a special attention. During the test stage, values at every point immediately after additions or subtractions need to be checked for overflow or underflow. This can be most easily done by using the "BV" (branch on accumulator overflow/underflow) instruction.

In a fixed-point arithmetic implementation, it is important to maintain an uniformity of the locations of the decimal points of the processed data. Furthermore, in order to obtain a numerical expression with maximum precision, the user should have some knowledge of the magnitude range

of the incoming data prior to the operation. If the range of incoming data is known to be between -1 and +1, then the data can be expressed with an optimum precision in a Q1 format (Q1 is a digit format with no integer bit, thus expressing a magnitude range of -1.0000 to 0.99999..). For a 16-bit machine, 15 bits are used to express a fraction in Q1 format, while only 14 bits represent a fraction in a Q2 format. However, the decimal point of the product will be left-shifted one bit after every multiplication. It is of vital importance not to neglect to re-adjust the decimal point back to the right place before any further processing.

CHAPTER 6

SYSTEM PERFORMANCE EVALUATION

The EVM digital signal processing system was evaluated for its performance in terms of meeting the system speed and the memory space requirements for our LPC synthesis implementation. Furthermore, the quality of its synthetic speech was evaluated on a comparative basis. In this chapter, it will be shown that the EVM320 system has met the above system requirements, and the speech quality is satisfactory. Also effort has been made to evaluate a data reduction method - linear speech parameter interpolation. By using the synthesizer as an evaluation tool, several speech parameter-interpolation schemes have been tested and the results are presented.

EVM System Evaluation

The EVM digital signal processing system evaluation was performed to examine whether the system meets the speed and memory space requirements of the implementation. The LPC speech synthesizer is a fairly complicated program of more than 400 assembly codes executed at a very high speed. The hardware system, from the evaluation, showed its

adequacy for meeting both the speed and the space implementation requirements.

It was shown previously that the system clock and the sample rate of the signal put a constraint on the number of instructions executed for synthesizing each sample. In our speech synthesis, a speech sample rate of 10,000 samples/sec will be normally used since it represents a reasonable data rate and pole-order compromise. The time for the TMS320 executing one instruction is 200 nsec. Thus a maximum of 500 instructions are allowed in a two-sample interval. The synthesizer is found to execute 310 instructions for every sample, and 392 instructions when across the frame boundary. In the latter case, an extra 82 instructions are used to input a new set of data, to reset address pointers, and to generate another random number. Therefore, the system allows an overhead of about 100 instructions between frames. This time slot is usually large enough for calculations such as data interpolation to reduce the data transfer rate (to be discussed in the next section).

A 3-4 seconds delay will occur when the system synthesizes a speech file during the first time. Once the data are loaded, the synthesis program will run in real-time. The actual time delay depends on the length of the

data file. This time delay will cause no great concern since the system need not be a strict-sense real-time tool for practical purposes.

The EVM's program memory was found to be large enough to store both the program file and the speech parameter file. The program memory is divided into two sections. From location HX0 to HX150 reside the synthesizer machine codes. The input parameter file can be stored in the rest of the program space of size 3,759 bytes from HX151 to HXFFF. In this way, the EVM can synthesize a 10-pole, 16-bit LPC speech parameter file of maximum length of 6.25 seconds (313 frames) at once, which is sufficient for most of our LPC system evaluation applications.

Most of the TMS320's small data memory have been used by the synthesizer. The synthesizer uses 112 bytes of the system's 144-byte data memory for data registers, residual buffer pointers, and so on. The data memory could be expanded by storing values from the data memory to the program memory, at a cost of executing extra instructions.

A 4200 bps data rate could be achieved by our LPC speech synthesis system. The speech parameters G, P, and K_p are transferred from the EXORmacs as 7-bit ASCII codes. Therefore, a frame of 10-pole speech (0.02 second) can be represented by 7x12 = 84 bits, and one second of speech by

4200 bits.

Table 1 summarizes the EVM system evaluation and performance characteristic of the LPC synthesis system.

Table 1 Performance characteristics of the LPC speech synthesizer

ITEM	SPECIFICATION
Number of Instr. per sample(500max.)	310 within frame 392 across frame bound.
First-round execution time delay	2-4 seconds
Longest speech phrase to be played once	6.24 seconds
Data memory occupancy (144 byte max.)	112 byte
Data baud rate	4200 bps
System order (max. 16)	10-16
Sample rate	8-10,000 sample/sec.
Number of samples per frame	200
Number of frames per second	50
System word length	16-bit
System structure	two-multiplier lattice

Synthetic Speech Quality

Five 10,000 sampling rate, 10-pole reflection coefficient female-voicing speech files stored in the EXORmacs computer system have been utilized to test the speech quality of the synthesizer. Since the female speech files are more sensitive to system noise and distortions, they were used as the input files of the synthesizer for more accurate measurements.

The output samples are converted to the synthetic analog signal by an on-board 12-bit D/A convertor, filtered by a low-pass filter, amplified by an audio amplifier, and finally played through a loadspeaker. Better results can be obtained if the speech is played through a high quality cassette deck. Judgements of synthetic speech quality were based entirely on objective opinion. This is due to the limitation that the EXORmacs system which acts as the host computer will not accept data from its devices. Therefore, the synthetic samples cannot be fed back to the host for a quantitative comparison of the synthetic and original signals. This drawback will be overcome when a new operating system is installed on the EXORmacs system.

The five synthetic speech phrases used in performance tests were intelligible and clear. They were compared with the synthetic speech synthesized under similar condi-

tions by the University of Arizona LPC Speech Analysis Programs. The latter is a non-real-time LPC two-multiplier speech synthesis system implemented on the EXORmacs. The result demonstrated that the quality of the TMS320 synthesizer is as good as, if not better, than the latter.

Use of the TMS320 Speech Synthesizer in Data Compression Experiments

One type of data reduction scheme is to selectively transfer parts of the parameters to the synthesizer. The synthesizer, by using the parameters, calculates the non-transferred parameters. Given the existance of a synthesizer with fixed system parameters such as word length, sample rate, filter order, etc., this scheme provides a possibility of further data reduction.

Linear interpolation and prior data duplication are two easy ways to compute the non-transferred values. If the parameters to be interpolated are the reflection coefficients, the stability of the synthesizer is guaranteed. This is so since the interpolated values are still less than unity, which is the condition for a stable lattice filter. This property demonstrates another superb advantage of using the LPC lattice structure.

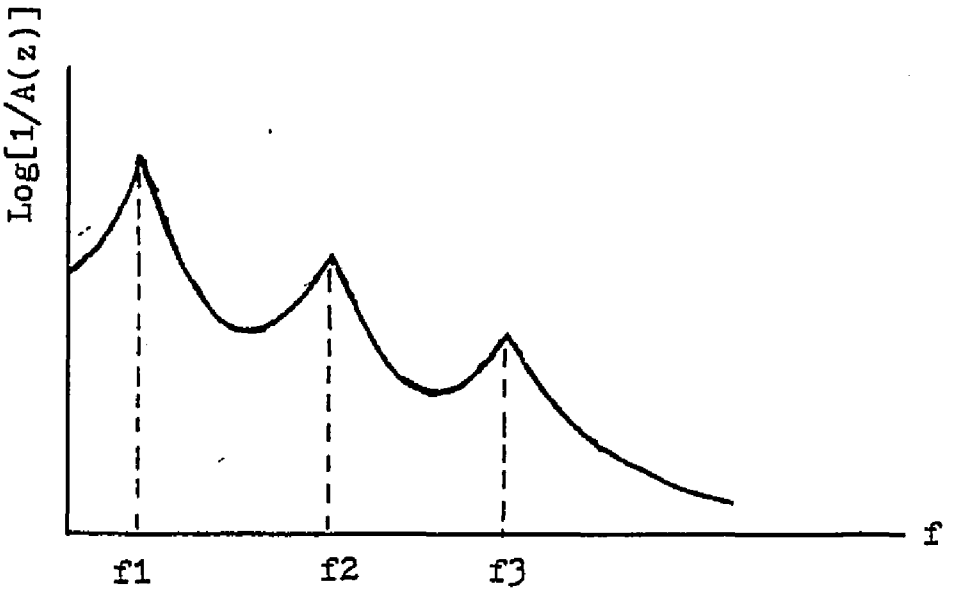
The EVM-based speech synthesizer was used to conveniently evaluate these data rate reduction schemes. The

evaluation is solely based on comparisons of the qualities of synthetic speech obtained from different compression schemes. The evaluation result of three schemes - partial interpolation, entire phrase interpolation/duplication, and moving-average windowing of speech parameter is presented below.

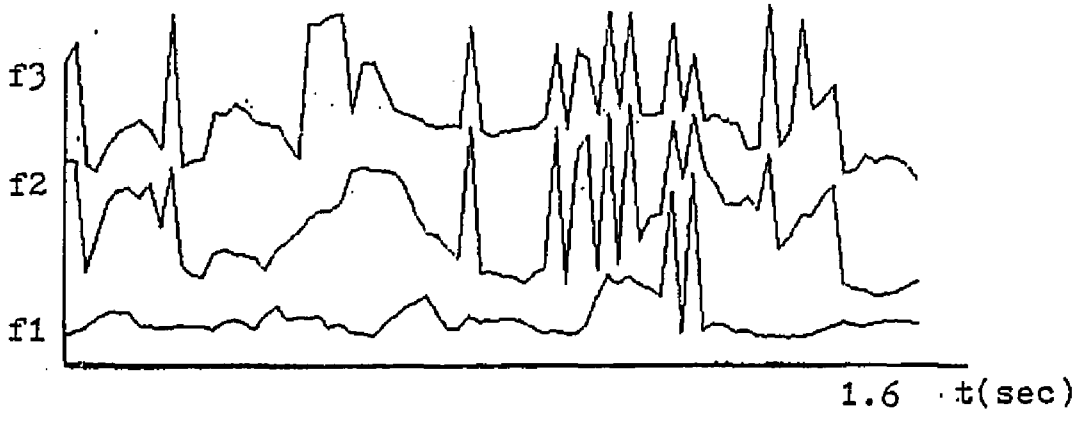
Partial Linear Interpolation

It was shown that the inverse synthesis filter $1/A(z)$ contains the envelope of the power spectrum of the input speech. A number of peaks in the spectrum, called formants, represent the complex roots of the function $1/A(z)$.

Figure 17 shows the spectral plot of $1/[A[\exp(jw)]]^2$ and its first four formants. If these formants are plotted for every frame of speech, the so-called formant trajectory will be obtained as shown in Figure 17(a). Solving the roots of $1/A(z)$ guarantees that all possible formant frequency and bandwidth candidates will be extracted. The disadvantage of root-solving is that it is very computationally intensive, and model-precision dependent. Another method, which is often used for its simplicity, is spectrum peak-picking. The peak-picking method was used in our experiment. The algorithm is due to Markel and Gray (1976).



(a)



(b)

Figure 17. (a) 1/A(z) spectrum plot (single frame)
(b) Formant trajectory plot (80 frames)

After the formant trajectory has been obtained, a smoothing routine was used to smooth out the formant plot. This process is based on the hypothesis that the formant tract is unlikely to change drastically from one frame to another (0.02 seconds) due to the physical limitations of the human vocal tract. The re-defined frames were marked and interpolated by a linear interpolation routine. Results showed that an average of 1/4 of the frames were interpolated, resulting in a 12.5% reduction of data rate.

The result of this selective interpolation on the reflection coefficients was quite surprising: the interpolated speech sounded the same as the non-interpolated one. This observation demonstrates the possibility of reducing data transfer rate by interpolating reflection coefficient data files based on formant trajectory smoothing.

Linear Interpolation and Prior Data Duplication

A simpler way to reduce data rate is to duplicate or interpolate every other frame of parameters of the entire speech, thus achieving a 50% data rate reduction.

In the experiment, linear interpolation or duplication are performed in the EXORmacs, and the compressed parameters are then transferrred to the EVM. In this way, the effects of various data compression schemes are evaluated by the TMS320 synthesizer. However, the actual data

rate remains the same. After the prelimilary evaluation, one can implement the same data compression algorithm to the EVM. The previous discussion showed that about 100 instructions are allowed for overhead computation.

Results showed that all reflection coefficient K-parameters except K1 were insensitive to linear interpolation. Interpolation performed on K1 caused small high-frequency distortions at the beginnings of some syllables. The distortion is most apparent when the syllable is a fricative voice. Interpolation performed on K2-K10 in every other frame does not result in any significant (intelligible) degradation of sound quality.

The prior frame duplication that copies every other data set from its previous one, on the other hand, had a much more severe effect on speech quality. Duplicating K1 or K2 will result in clearly distorted sounds, while duplication of K3 had a much smaller effect. Operation on the higher-order K values exhibited non-detectable degradations.

The above findings demonstrate that LPC filters' lower-order reflection coefficients are more sensitive to data compression than the higher-order ones. Therefore, for a reasonable speech quality and data rate compromise, reflection coefficients K1-K3 must be transferred at the

original data rate, while the rest of the K values can be compressed. The results also show that linear interpolation causes less speech degradation than data duplication does. Linear interpolation requires only one extra addition and one division (or, for faster execution, one multiplication by the inversed divisor) than duplication. Therefore, linear interpolation is preferred among the two.

Moving-Average Windowing

A moving-average windowing is actually a variable-length linear interpolation scheme. The interpolated value X' is given by

$$X_i' = (X_{i-n}+X_{i-n-1}+ \cdots +X_{i+n-1}+X_{i+n})/(2n+1).$$

Moving-average windowing has the advantage of being able to vary the window length so that different parameters can be smoothed by windows of varying length. Figure 18 shows the original and the smoothed reflection coefficients as a function of time.

The result of applying a fixed-length moving-average window to the reflection coefficients was found to be consistent with the result from the linear interpolation. The lower-order K-parameters were more sensitive to windowing than the higher-order ones. The window length also affects the speech quality: a longer window produces more

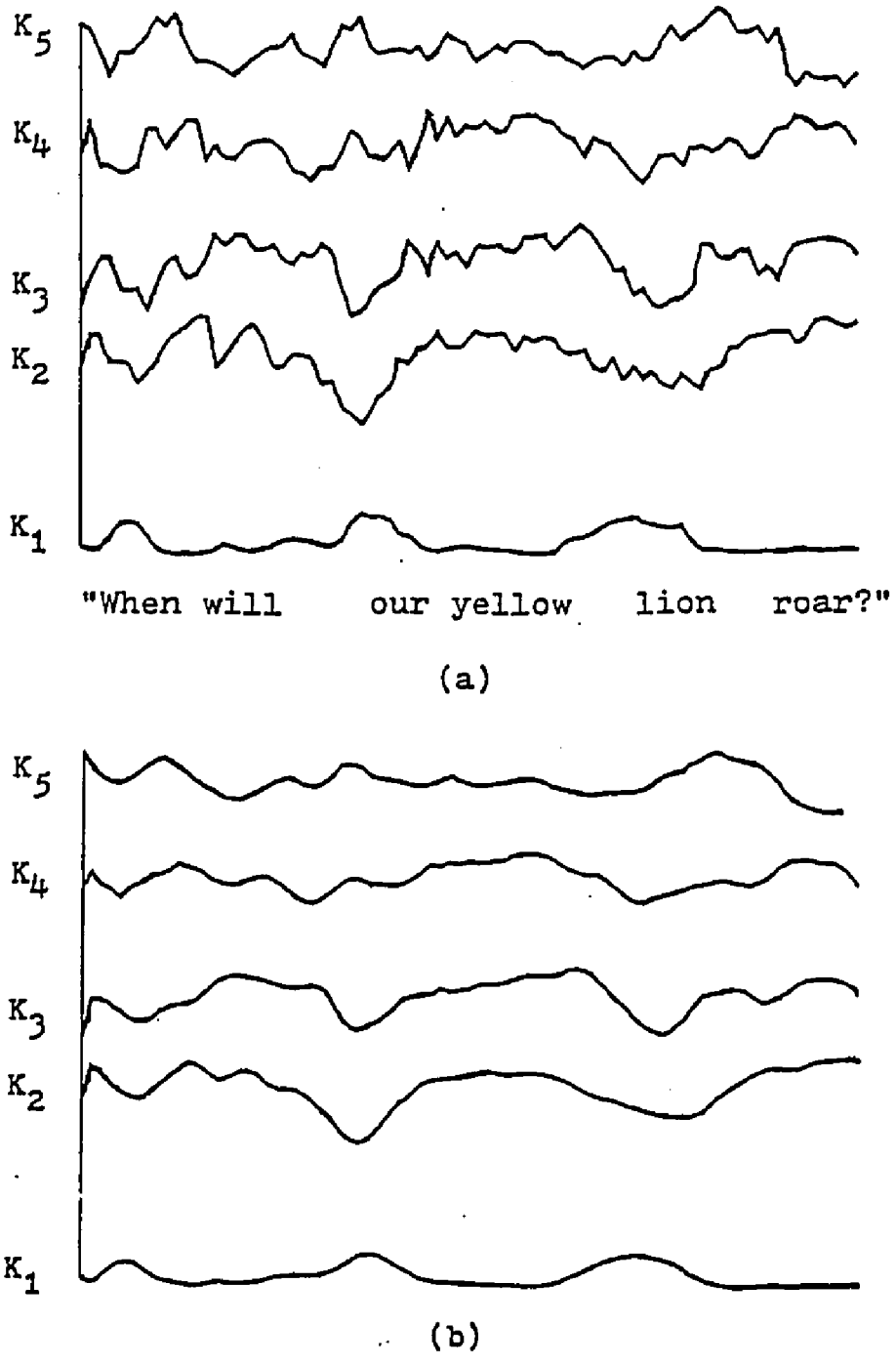


Figure 18. Reflection coefficients plot (a) original (b) original smoothed by a 5-point window

distinctively distorted sound. Therefore, a longer window can be applied to the higher-order K-parameters, while only a short window is allowed to the lower-order ones. Specifically, K1 is sensitive to a window of any length - this is consistant with linear interpolation. K2, K3 generally can be averaged by windows with lengths of 5-9 points. The maximum allowable length depends on the type of speech.

The distortion caused by moving-average windowing has a similar sounding and occurence pattern as the one in the interpolation experiment. The high-frequency distortion occurs at the beginnings of fricative syllables. It is not clear whether the distortion occurs only at fricative syllables. Although distortion could also exist at vowel syllables, it is less discernible to the human ears.

Table 2 summarizes the distortions caused by the three data compression methods applied to the K-parameters.

Table 2 Distortion comparison of three data compression methods

K's	Interpolation	Duplication	Windowing*
K1	Detectable	Large	3 pt.
K2	Slight detectable	Detectable	5 pt.
K3	Non-detectable	Slight	11 pt.
K4	Non-detectable	Non-detectable	17 pt.
K5-10	Non-detectable	Non-detectable	arbitr. length

* Maximum window length before dicernible distortion is apparent.

Conclusion and Suggestion

From the above experiment, it is quite obvious that, under certain conditions, data compression by linear interpolation is possible, and the quality of the synthetic speech output will not be effected by a large amount.

In our preliminary study, a synthesizer with 10-pole and 16-bit word length has been used as the evaluation tool. A different synthesis system might show quantitatively different results. Nevertheless, it is reasonable to conclude that our results are of practical value, which will probably lead to future research efforts.

One possible future experiment could be to interpolate the compressed reflection coefficient file in the EVM system, although some overhead computations would be required. From the previous section, it was demonstrated that the EVM-based synthesizer would provide enough time and space for the extra computation.

CHAPTER 7

CONCLUSION

By using the EVM320, a single-board development system for digital signal processing evaluation and testing, it was found that the TMS320 is a suitable DSP chip for a real-time speech synthesis routine implementation.

The Harvard structure approach and hardware-intensive design make the TSM320 a superb tool for real-time computational intensive signal processing. Equipped with a rich but compact instruction set, the TMS320 chip is very convenient to program. The system input and output can be designed rather easily. The execution speed and memory requirements for the LPC speech synthesizer have been met.

The speech synthesizer uses a two-multiplier lattice linear predictive coding filter. The algorithm is primarily based on Markel and Gray (1976, p. 243) with many modifications. A 16-bit word fixed-point arithmetic has been used to reduce the complexity of the routine. The experiment can be considered a success, for the quality of the synthetic speech output is very satisfactory. The synthetic speech is intelligible, and clear. Its quality is compatible with synthetic speech operated by other systems

using the same synthesis scheme and conditions.

Using the LPC synthesizer as an evaluation tool, efforts also have been made to investigate the effect of linear interpolation on speech parameter files in terms of speech quality. It has been found that, in LPC speech synthesis, linear interpolation of every other frame of speech on either autocorrelation or reflection coefficient files will not result in a noticeable degradation. Several interpolation methods have been studied. It seems that interpolation based on speech formant trajectory smoothing is the most reasonable approach with the least discernible effect on speech quality. The entire-phrase interpolation simply reduces data rate by half, and it only results in detectable effect when the lowest-order parameter is interpolated. Therefore, for a specific LPC speech analysis/synthesis system, data transfer rate reduction between the two stages can be achieved by linearly interpolating the speech parameter.

Future research activities in this project could include making some quantitative measurements of the synthetic speech. The operating system of the host computer EXORmacs needs to be updated in order for it to process the data from the EVM. The printer port speed, which currently puts the limitation on the EVM data input, may also improve under the new operating system.

APPENDIX A

TMS320 INSTRUCTION SET

Accumulator Instructions

ABS --- Absolute value of accumulator

ADD --- Add to accumulator with shift

ADDH -- Add to high-order accumulator bits

ADDS -- Add to accumulator with sign suppressed

AND --- AND with accumulator

LAC --- Load accumulator with sign extension

LACK -- Load accumulator immediate

OR ---- OR with accumulator

SACH -- Store high-order accumulator bits with shift

SACL -- Store low-order accumulator bits

SUB --- Subtract from accumulator with shift

SUBC -- Conditional subtract

SUBH -- Subtract from high-order accumulator bits

SUBS -- Subtract from accum. with no sign extension

XOR --- Exclusive OR with accumulator

ZAC --- Zero accumulator

ZALH -- Zero accumulator and load high-order bit

ZALS-- Zero accum. & load low-order bits with no sign ext.

Auxiliary Register and Data Page Pointer Instructions

LAR --- Load auxiliary register

LARK -- Load auxiliary register immediate

LARP -- Load auxiliary register pointer immediate

LDP --- Load data memory page pointer

LDPK -- Load data memory page pointer immediate

MAR --- Modify auxiliary register and pointer

SAR --- Store auxiliary register

Branch Instructions

B ----- Branch unconditionally

BANZ -- Branch on auxiliary register not zero

BGEZ -- Branch if accumulator >= 0

BGZ --- Branch if accumulator > 0

BIOZ -- Branch if $\overline{BIO}$ = 0

BLEZ -- Branch if accumulator <= 0

BLZ --- Branch if accumulator < 0

BNZ --- Branch if accumulator not zero

BV ---- Branch on overflow

BZ ---- Branch if accumulator = 0

CALA -- Call subroutine from accumulator

CALL -- Call subroutine immediatelly

RET --- Return from subroutine

T Register, P Register, and Multiply Instructions

APAC -- Add P to accumulator

LT ---- Load T

LTA --- Combine LT and APAC into one instruction

LTD --- Combine LTA and DMOV into one instruction

MPY --- Multiply with T, store product in P

MPYK -- Multiply T with immediate operator, store in P

PAC --- Load accumulator from P

SPAC -- Subtract P from accumulator

Control Instructions

DINT -- Disable interrupt

EINT -- Enable interrupt

LSP --- Load status register

POP --- Pop stack to accumulator

PUSH -- Push stack from accumulator

ROVM -- Reset overflow mode

SOVM -- Set overflow mode

SST --- Store status register

NOP --- No operation

I/O and Data Memory Operations

DMOV -- Copy contents of data memory into next location

IN ---- Input data from port

OUT --- Output data to port

TBLR -- Table read from program memory to data RAM

TBLW -- Table write from data RAM to program memory

APPENDIX B

SPECIAL EVM HARDWARE MODIFICATIONS

Both the EVM/AIB system firmware and hardware were modified for implementing the speech synthesizer.

System Firmware Modification

The firmware modification made the EVM's communication port compatable to that of the EXORmacs'. Using the on-board EPROM utility, one system EPROM (U58) is re-programed. The changes are shown as follows.

Addr.	Function	From	To	Description
008A,8B	XON/XOFF	>30FF	>FFFF	enable XON
008E,8F	portformat	>4242	>C3C3	8-bit data,1 s-bit
0090,91	baud rate	>2580	>0965	use 2400 bps
009E,9F	time-out	>0019	>0032	double time-out

AIB Hardware Modification

The AIB was modified so that it can activate both an I/O and an interrupt signal simultaneously. The original design prohibits from using both ©BIO and ©INT for simultaneous I/O operation. In the speech synthesis implementa-

tion, it is necessary to use both of them to control system input and output separately.

The speech data are transmitted from the EXORmacs to the EVM over a 36-pin Centronix cable. Two control lines ©DSTB and ©DTACK, perform a two-line handshaking between the host and the EVM. When ready for the data transfer, the EXORmacs will put an ASCII code on the data line and pull the ©DSTB active, signaling that data are ready to be taken. The ©DSTB, having taken an interrupt call by the EVM, will activate the interrupt handling routine that takes data and sends a ©DTACK to the EXORmacs, indicating that data have been taken. This activates the next ASCII code transfer.

As shown in Figure 19, ©DSTB triggers the flip-flop (F1), whose output connects ©BIO to the EVM. F1 is necessary for holding the status of ©DSTB since the TMS320 requires an interrupt request to be at least one-cycle long. The data-enable (©DEN) from the EVM, ANDed with Y4, serves as a "data acknowledge" signal. Thus, executing a command "IN MEM, PR4" on the interrupt service routine results in a low on ©DTACK, indicating that the data have been taken.

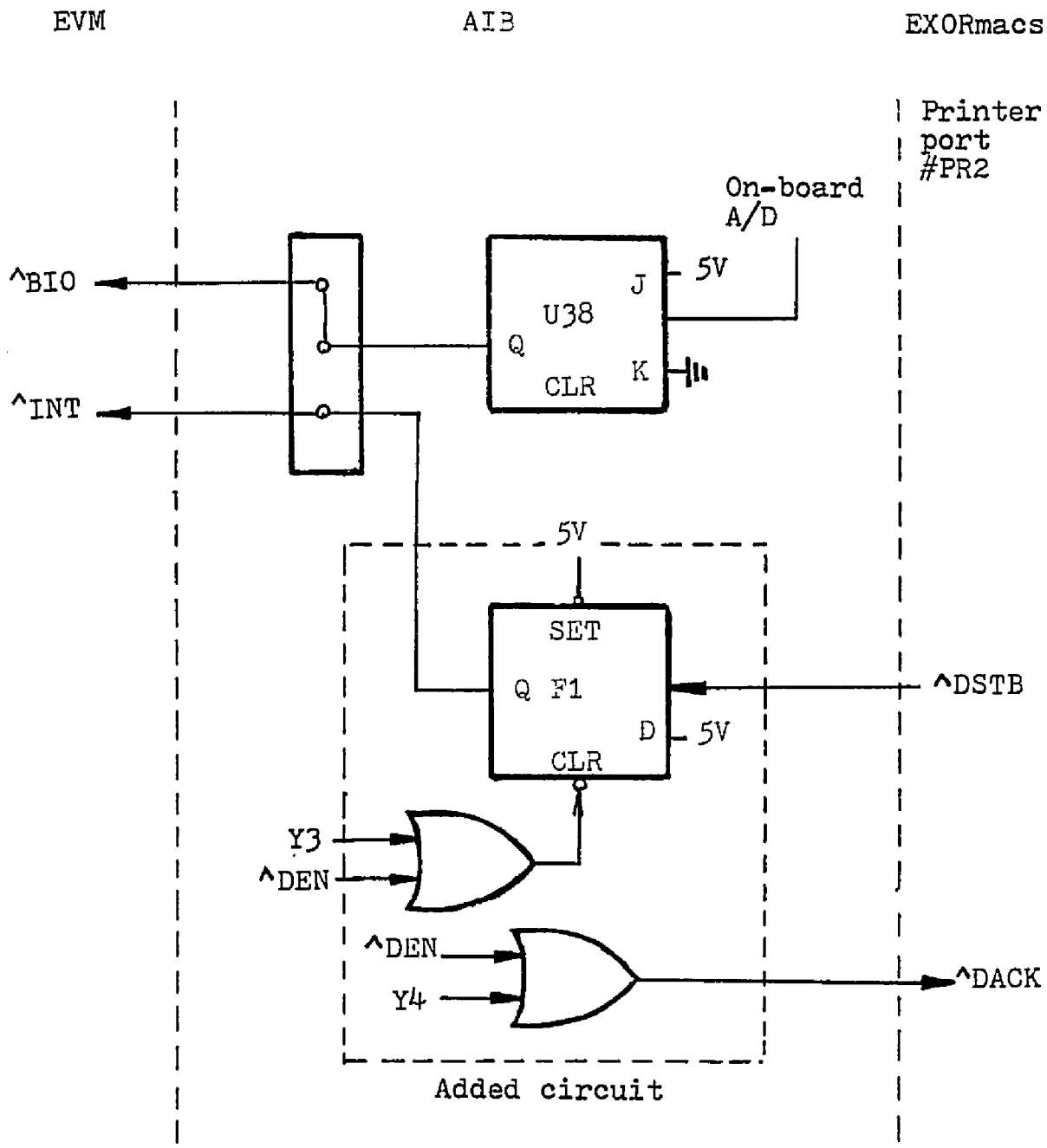


Figure 19. Modified AIB circuit

APPENDIX C

SPEECH SYNTHESIZER ASSEMBLY CODES

```
*
*         SYN.SA --- plays whole section of female speech.
*                --- many gain-related mistakes has been corrected
*                --- reverse K-parameter's order
*                --- Keep all values in Q15 format
*                --- revised 2/5/85
*                --- final version!
*                --- speech data are sent to P.M. area by TBLW, and
*                    then extracted by the synthesizer frame per frame
*                    using TBLR instr.
*                --- speech data file is stored in SYS:100 under the name
*                    of S#.RCF.SF, which will be converted to a HX16-file
*                    TMSDATA.SF by a formatting routine FORMT320.LD
*
*       data area declaration and its intitialization values
*
ONE     EQU     >00             ONE=1
MINUS   EQU     >01             MINUS=-1
RCBUF   EQU     >02             RCBUF(17)=0
IV      EQU     >13             voicing flag, IV=0
IPC     EQU     >14             pitch period counter, IPC=1
IFC     EQU     >15             frame counter, IFC=1
PE      EQU     >16             post-emphasis PE=.95(29572)
YPREV   EQU     >17             output YOUT accumulator, YPREV=0
DRV     EQU     >18             DRV=0
EXSA    EQU     >19             excit. func. for the rest impulse ,=0
YOUT    EQU     >1A             YOUT=0
*
OPA     EQU     >1B             seeds for R.M.G., OPA=2000(SD)
OPB     EQU     >1C                              OPB=3000(SD)
OPC     EQU     >1D                              OPC=1
OPD     EQU     >1E                              OPD=3533(SD)
*
*       Set sampling rate clock, output mode register
*
CNTRL   EQU     >1F             MODE=10
CLKR    EQU     >20             M=524, for 10KHz sampling rate
M       EQU     >21             M = 9 for tenth-order filter
MASK    EQU     >22             MASK = >7FFF for output inversion
FLGTH   EQU     >23             FLGTH=200 spl/frm for 10KHz spl rate
*
*       Initialize 9 variables used by interrupt service routine
*
MASK1   EQU     >24             MASK1 = >000F
NUMDIC  EQU     >25             NUMDIC = >0010
NUMALP  EQU     >26             NUMALP = >0040
TWO     EQU     >27             TWO = 2
NINE    EQU     >28             NINE = 9
FLAG    EQU     >29             FLAG = 0
CFLAG   EQU     >2A             CFLAG = 1
MPSIZE  EQU     >2B             MPSIZE = >7FFF size of P.M. area
PTR     EQU     >2C             start addr. for data in P.M.(>0150)
*
*       three user defnedable parameters
*
EAF     EQU     >30             excitation adjustment factor
NAF     EQU     >31             noise-impulse ratio adjustment factor
GAF     EQU     >32             gain adjustment factor
*
*       Analysis parameters are updated for each analysis frame.
*
PITCH   EQU     >33             RC(16) reflection coeff.
GAIN    EQU     >34             gain
```

```
[illegible]  EQU   [illegible]          patch target
*
TEMP1H  EQU   >50             temp. storage for R.N.G.
TEMP1L  EQU   >51
TEMP2H  EQU   >52
TEMP2L  EQU   >53
*
AR4     EQU   >54             temp. storages for TWOMUL
ARSCL   EQU   >55
ARSC    EQU   >56
TEMP    EQU   >57
*
NUMERA  EQU   >58             temp. storages for DIV1
DENOM   EQU   >59
QUOT    EQU   >5A
TEMSGN  EQU   >5B
EAPT    EQU   >5C
*
*       temp. storage for INTSR
*
TEMP1   EQU   >5D             input ASCII store
TEMP2   EQU   >5E             converted ASCII store
VALUE   EQU   >6F             converted HX16 store
BUFFTR  EQU   >70             P.M. data area pointer
ACH     EQU   >71             ACCH store
ACL     EQU   >72             ACCL store
*
        AORG  >02
*
        B     INTSR
*
INIT    DATA  1,-1,0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0
        DATA  0,1,1,29572,0,0,0,0,2000,3000,1,3533
        DATA  0,524,9,>7FFF,200
        DATA  >F,>10,>40,2,9,0,1,>FFF,>150
*
*       Initialize data memory.
*
        LACK  INIT
        LARK  0,44
        LARK  1,ONE
CONT1   LARP  1
        TBLR  *+
        ADD   ONE
        LARP  0
        BANZ  CONT1
*
*       Input speech data from EXORMAX printer port, store them into
*       program memory area pointed by BUFFTR. This is accomplished by
*       an interrupt-driven service routine.
*
        LAC   PTR
        SACL  BUFPTR
        EINT                  enable interrupt
GETPAR: LAC   FLAG            wait until a HX16 word is packed
        BZ    GETPAR
*
        ZAC
        SACL  FLAG
*
        LAC   BUFPTR          transfer the word from D.M. to P.M.
        TBLW  VALUE
        ADD   ONE
```

```
        SACL    BUFPTR          BUFPTR points to next P.M. area
        SUB     PMSIZE          P.M. size exceed?
        BZ      EOF
        ZALS    VALUE           end of file?
        XOR     MINUS
        BZ      EOF
        EINT
        B       GETPAR
*
*       Upon the completion of parameter input, speech synthesis begins.
*
EOF     SOVM                    set overflow mode
*
        OUT     CLCK,1          initialize clock
        OUT     CNTRL,0         initialize output mode register
*
*       Initialize pointer to data in P.M.
*
        LAC     PTR             BUFPTR = PTR
        SACL    BUFPTR
*
*       Transfer 1 frame of data from P.M. into D.M. started at PITCH
*
GETFRM  LAC     BUFPTR
        LARK    0,11
        LARK    1,PITCH
CONT2   LARP    1
        TBLR    *+
        ADD     ONE
        LARP    0
        BANZ    CONT2
*
        SACL    BUFPTR          store parameter pointer for next frame
*
*       Begin a new frame analysis.
*
        LAC     GAIN            end of analysis?
        BLZ     DONE            (a negative GAIN terminates analysis)
*
*       Start a new pitch period.
*
        LAC     PITCH           voiced ?
        BLZ     UV              no, goto UV
        LAC     ONE             yes, IV = 1
        SACL    IV
*
        LAC     GAIN
        BZ      SILENT
*
        LAC     EAF             DRV = EAF
        SACL    DRV
        LAC     MINUS           DRVN = -[EAF/(PITCH)]
        SACL    NUMERA
        LAC     PITCH
        SACL    DENOM
        CALL    DIV1
        LT      QUOT
        MPY     EAF
        PAC
        SACH    DRVN,1
        B       MORE
*
UV      ZAC                     unvoiced or silent, IV = 0
```

```
        SACL    IV
        LT      EAF             EAFT = EAF*MAF
        MPY     MAF
        PAC
        SACH    EAFT,1
        LAC     FLGTH           PITCH = FLGTH-IPC+1
        SUB     IPC
        ADD     ONE
        SACL    PITCH
*
MODE    LAC     GAIN
        BZ      SILENT
        LAC     IV
        BGZ     SIMUL           voiced?
        CALL    NOISE           for unvoiced frame DRV = (R.N.)*EAF
        LT      TEMP1H
        MPY     EAFT
        PAC
        SACH    DRV,1
        B       SIMUL
*
SILENT  ZAC
        SACL    DRV
*
SIMUL   CALL    TWOMUL          two-multiplier lattice filter
*
        LT      GAIN            YOUTH = GAIN*YOUTH
        MPY     YOUT
        PAC
        SACH    YOUT,1
*
        LT      YOUT            YOUTH-1 = GAF*YOUTH
        MPY     GAF
        PAC
        SACH    YOUT,1
*
        LT      PE              de-emphysis filter
        MPY     YPREV           YOUT = YOUT+PE*YPREV
        PAC
        SACH    YPREV,1
        LAC     YPREV
        ADD     YOUT
        SACL    YPREV
*
WAIT1   BIOZ    OUTPUT          fixed output rate(8K sample/sec)
        B       WAIT1
*
OUTPUT  LAC     YOUT            invert output signal
        XOR     MASK
        SACL    YOUT
        OUT     YOUT,2          output
*
        LAC     DRVM            DRV=DRVM
        SACL    DRV
*
        LAC     IFC             IFC=IFC+1
        ADD     ONE
        SACL    IFC
*
        LAC     IPC             IPC=IPC+1
        ADD     ONE
        SACL    IPC
*
```

```
        SUB     PITCH           within the pitch period ?
        BLZ     MORE            yes
        LAC     ONE             no, start next pitch
        SACL    IPC             IPC=1
*
        LAC     EAF             DRV=EAF
        SACL    DRV
*
        LAC     IFC
        SUB     FLGTH           within the frame boundry ?
        BLZ     MORE            yes
        SACL    IFC             no, IFC = IFC-FLGTH, start next frame
*
        B       GETFRM          get the next-frame data
*
*       Main routine ends.
*
*       subrotine DIV1 for NUMERA/DENOM assuming NUMERA<DENOM
*
DIV1    LARP    0
        LT      NUMERA
        MPY     DENOM
        PAC
        SACH    TEMSGN
        LAC     DENOM
        ABS
        SACL    DENOM
        ZALH    NUMERA
        ABS
        LARK    0,14
*
XPDIVC  SUBC    DENOM
        BANZ    XPDIVC
*
        SACL    QUOT
        LAC     TEMSGN
        BGEZ    DONE1
*
        ZAC
        SUB     QUOT
        SACL    QUOT
DONE1   RET
*
*       random number generator --- actual r.n. in address TEMP1H
*
NOISE   ROVM
        LT      OPA
        MPY     OPB
        PAC
        SACH    TEMP1H
        SACL    TEMP1L
*
        LT      OPB
        MPY     OPC
        PAC
        SACH    TEMP2H
        SACL    TEMP2L
*
        ZALS    TEMP1L
        ADDS    TEMP2L
        SACL    TEMP1L
*
        LT      OPB
```

```
        MPY     OPB
        PAC
        SACL    OPB
        SACH    TEMP2H
*
        ZALS    TEMP1L
        ADD     TEMP2H
        SACL    OPA
        ZALH    OPA
        SACH    TEMP1H,1
        SOVM
        RET
*
*
* Two multiplier synthsis lattice filter
*
TWOMUL  LAC     M               COPY order M into ARM
        SACL    ARM
*
        LACK    3               AR0 points to next-to-last RCBUF
        SACL    ARBUF
        LAR     0,ARBUF
*
        LACK    RC              AR1 points to last RC
        SACL    ARRC
        LAR     1,ARRC
*
        LARP    0
*
LOOP    LT      *
        LARP    1
        MPY     *               TEMP = RCBUF*RC
        PAC
        SACH    TEMP,1
*
        LAC     DRV
        SUB     TEMP            DRV (= DRV-RCBUF*RC
        SACL    DRV
*
        LT      DRV             TEMP = DRV*RC
        MPY     *+,0            AR1--
        PAC
        SACH    TEMP,1
        LAC     TEMP
        ADD     *-              RCBUF (= RCBUF+DRV*RC
        SACL    *+
*
        MAR     *+              bring ARBUF to next RCBUFH position
        SAR     0,ARBUF         STORE ARC TO ARBUF
        LAR     0,ARM           AR0 CONTAINS ARM
        BANZ    CNTNU
        B       LAST
CNTNU   SAR     0,ARM           AR0 TO ARM
        LAR     0,ARBUF         restore RCBUF in AR0 for next looping
        B       LOOP
*
LAST    LAC     DRV             RCBUF(0) = DRV(0)
        LAR     0,ARBUF
        MAR     *-
        SACL    *
*
        SACL    YOUT            YOUT = DRV(0)
*
```

```
        RET
*
*       Interrupt service routine:
*       * 1 HX16 word is packed from 4 ASCII codes
*       * It is enabled only at the begin. of the program where the data
*         transfer process takes place. It will be disabled afterwards.
*
INTSR   SACH    ACH             save ACC
        SACL    ACL
        IN      TEMP1,3         intrpt. service routine
        LAC     TEMP1
        AND     MASK1           MASK1 = 000F
        SACL    TEMP2           TEMP2 contains LSB 4 bits of TEMP1
*
TRY1:   ZALS    TEMP1
        AND     NUMDIG          ASCII code is a digit
        BZ      TRY2
        ZAC
        SACL    CFLAG           CFLAG = 0
        LAC     VALUE,4
        ADD     TEMP2
        SACL    VALUE           VALUE = VALUE(shf.L.4)+TEMP2
        B       NEXT1
*
TRY2    ZALS    TEMP1           ASCII code is a alphabet number
        AND     NUMALP
        BZ      TRY3
        ZAC
        SACL    CFLAG
        LAC     VALUE,4
        ADD     TEMP2
        ADD     NINE
        SACL    VALUE           VALUE = VALUE(shf.L.4)+TEMP2
        B       NEXT1
*
TRY3    LAC     CFLAG           ASCII must be a (CR),(SP), or (LF)
        BGZ     NEXT1           VALUE has been transfered, do nothing
*
        LACK    1               disable further transfer till a new
        SACL    CFLAG           HX16 word is assmbled
        SACL    FLAG
        IN      >7F,4
        IN      >7F,4
        B       FULL
*
NEXT1   IN      >7F,4           out handshake signal as ACK to sender
        IN      >7F,4
*
        ZALH    ACH             restore ACC
        ADDS    ACL
        EINT                    enable interrupt
FULL    RET
*
DONE    B       EOF
        END
```

APPENDIX D

OPERATIONOFTHE TMS320 SPEECH SYNTHESIS SYSTEM

The source codes of speech synthesizer reside in the EXORmacs computer. The EVM assembles the source codes to the executable machine codes during code down-loading througth a RS232 connection. The speech data files are created and stored in the EXORmacs, and transferred to the EVM for speech synthesis througth a 36-bit CENTRONIX printer cable.

Operation of the TMS320 Speech Synthesizer

The first step to operate the synthesizer is to reset the EVM by pressing a reset switch located on the corner of the EVM. This will prompt the EVM to the system monitor.

Then the user needs to log the terminal on to the EXORmacs by pressing a toggle key ©C, which brings the EVM to the transparency mode. In order to log on to the EXORmacs, a red relay switch and a "Break" key are pressed simultaneously. After both systems are up, the following

user/system interaction will load the synthesizer from the EXORmacs to the EVM program memory

KEYBOARD INPUT	SCREEN	NOTE
ASM 2 <CR>	Line number?	Activate Assembler
<CR>	* ASSEMBLER *	No line number
©C	=	Toggle EVMtotrans. mode
Copysyn.sa, #cn13 ©C	Codelisting	Assembling code "SYN.SA", back to EVM monitor mode

After the EVM has assembled the source code and loaded it into the executing area, a few things need to be done before running the program. First, the clock mode needs to be set to "External", and the program counter set to the beginning of the program (HX31). Then breakpoint(s) need to be set at HX51 for repeated execution until an <escape> key is hit. The three user-changeable gain control values listed as follows:

D. M. Addr.	Value(SD)	Description
HX30	4-6000	Excitation amplitude adjust.
HX31	14-16000	Voiced/unvoiced gain ratio adjust.
HX32	20-22000	System overall gain adjustment

These three values are stored in the memory by using the "MDM" command , and re-adjusted to obtain an optimum result. At this point, the synthesizer program is ready to run under the full control of EVM's system debugger.

Now the input data file is sent to the EVM through the EXORmacs' printer cable. Entering "COPY file.TMS.SF #PR2" from one of other two terminals will initiate speech parameter file transfer.

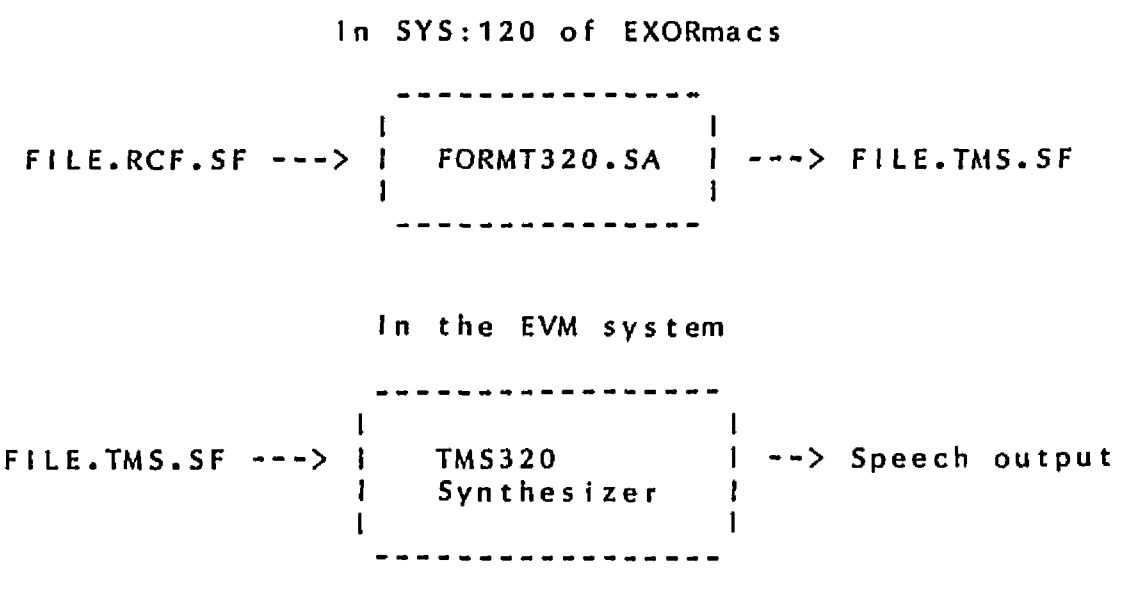
The synthesizer now can be executed by entering a "RUN" command.

Speech Parameter File Generation

The speech parameter files are generated by the Universith of Arizona LPC Speech Synthesis System on the EXORmacs computer system. The speech files contain about 60-100 set of parameters, and each set corresponds to a frame of speech. The parameters needed for the TMS320 synthesizer are the excitation gain, pitch, and the reflection coefficients K-parameters.

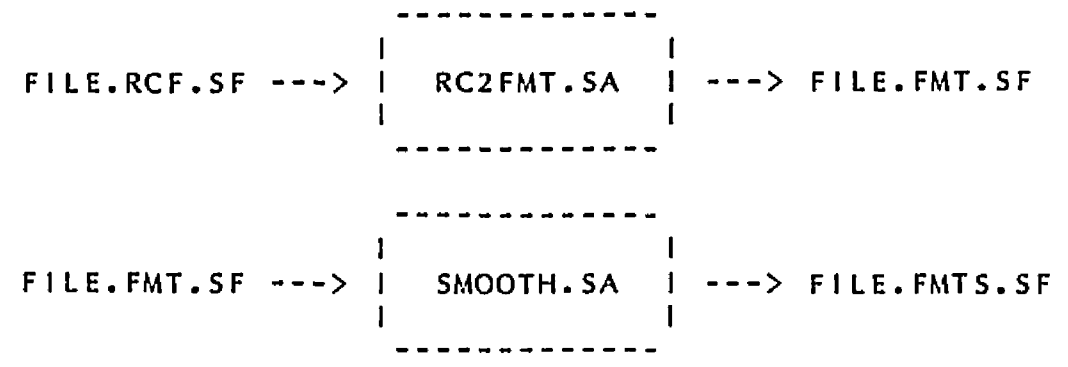
A program, FORMT320.SA, is used to extract these wanted parameters from the speech file. It also converts the parameters to a hexa-decimal form that is acceptable to the TMS320. The FORMT320.SA's input file is filename.RCF.SF ,which is the original speech file, and its output file is

filename.TMS.SF, which is the input of the TMS320 synthesizer. This is shown as



Operation of Interpolation Based on Trajectory Smoothing

The operation procedure linear interpolation of K-parameters of a speech file based on formant trajectory smoothing is shown as follows.



Note:

.FMT.SF - Formant trajectory file

.FMTS.SF - Formant trajectory file, modified

.RCF.SF - Reflection coefficient file

.RCFI.SF - Reflection coefficient file, interpolated

Operation of Entire Phrase Data Compression

The following demonstrates the operation procedures of the entire speech phrase data compression by interpolation, duplication, or moving-average windowing.

Interpolation:

Duplication:

Windowing:

After .RCFI.SF files have been obtained, the FORMT320.SF program will create the .TMS.SF executable by the TMS320 synthesizer, as indicated before.

LIST OF REFERENCES

Atal, B. S., "Determination of the Vocal Tract Shape Directly from the Speech Wave," Journal of Acoustic Soc. of Am., vol. 47, pp. 65-66, 1970.

Bristow, G., Electronic Speech Synthesis, Granada Publising Ltd, London, 1984.

Carlson, Gerrard, M., "The Quality of Synthesized Speech Using Linear Predictive Coding on Finite Wordlength Integrated Circuits," Ph. D. Dissertation, University of Arizona, 1985.

Fant, G. C. M., Acoustic Theory of Speech Production, Mouton and Co.,'s-Gravenhage, The Netherlands, 1960.

Flanagan, J. L., Speech Analysis Synthesis and Perception, Springer-Verlag, Berlin, Heidelberg, New York, 1972.

Markel, J. D., and A. H. Gray, Linear Prediction of Speech, Springer-Verlag Berlin Heiderlberg, 1976.

Markel, J. D., "Digital Inverse Filtering - A New Tool for Formant Trajectory Estimation," IEEE Trans., Audio and Electroacoustics, vol. AU-20, No. 2, June 1972, pp. 129-137.

Markel, J. D., "Linear Prediction: A Tutotial Review," IEEE Proceedings, vol. 63, No. 4, April 1975, pp. 561-580.

McAulay, R. J., and T. F. Quatieri, "Magnitude-Only Reconstruction Using a Sinusoidal Speech Model," IEEE Proceedings, ICASSP, vol. 2, March 1984, pp. 27.6.1-27.6.4

McCandless, S. S., "An Algorithm for Automatic Formant Extraction Using Linear Prediction Spectra," IEEE Trans. ASSP-22, No. 2, April 1974, pp. 135-141.

Morgan, N., Talking Chips, McGraw-Hill Book Company, New York, 1984.

Rabiner, L., and B. Gold, Theory and Application of Digital Signal Processing, Prentice-Hall, Inc., Eaglewood Cliffs, New Jersey, 1975.

Rabiner, L., and R. W. Shafer, Digital Processing of Speech Signals, Prentice-Hall, Inc., Eaglewood Cliffs, New Jersey, 1978.

Texas Instruments, TMS320 Evaluation Module, 1983.

Texas Instruments, TMS320 User Guide, 1983.

Witten, I. H., Principles of Computer Speech, Academic Press, New York, 1982.