NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

THIS DISSERTATION HAS BEEN MICROFILMED EXACTLY AS RECEIVED.

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne qui protège le droit d'auteur. SRC 1970, c. C-30. Veuillez prendre connaissance des formulaires d'autorisation qui accompagnent cette thèse.

LA THÈSE A ÉTÉ MICROFILMÉE TELLE QUE NOUS L'AVONS REÇUE.
AID OF AUTHOR/NOM DE L'AUTEUR: TIMOTHY C. GREEN

TITLE OF THESIS/TITRE DE LA THÈSE: A Design Study of a Digital Speaker Monitor

UNIVERSITY/UNIVERSITÉ: C.C.U.

DEGREE FOR WHICH THESIS WAS PRESENTED/GRADE POUR LEQUEL CETTE THÈSE FUT PRÉSENTÉE: M.Eng.

EAR THIS DEGREE CONFERRED/ANNÉE D'OBTENTION DE CE GRADÉ: 1984

NAME OF SUPERVISOR/NOU DU DIRECTEUR DE THÈSE: Dr. L. Robins, M.I.E.

Permission is hereby granted to the NATIONAL LIBRARY OF CANADA to microfilm this thesis and to lend or sell copies of the film. The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

DATED/DATE: 21 SEPTEMBER 89 SIGNED/SIGNÉ: [Signature]

PERMANENT ADDRESS/RÉSIDENCE FIXE: 407 1405 Prince of Wales Dr.

OHawa, Colombo, E2L 3J4
A DESIGN STUDY FOR
A DFT-BASED SPEECH MODIFICATION SYSTEM
ON A DIF MICROCHIP

by

Timothy G. Green

A thesis submitted to the
Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for
the degree of
Master of Engineering

Carleton University
Ottawa, Ontario

August 1984
LICENCE TO CARLETON UNIVERSITY

In the interests of facilitating research by others at this institution and elsewhere, I hereby grant a licence to:

CARLETON UNIVERSITY

to make copies of my thesis:

A.D.R. for a Def. Code Spint Control System on a DSP

or substantial parts thereof, the copyright which is invested in me, provided that the licence is subject to the following conditions:

Only single copies shall be made or authorized to be made at any one time, and only in response to a written request from the library of any university or similar institution on its own behalf or on behalf of one of its users.

This licence shall continue for the full term of the copyright, or for so long as may be legally permitted.

The Universal Copyright Notice shall appear on the title page of all copies made under the authority of this licence.

This licence does not permit the sale of authorized copies at a profit, but does permit the collection by the institution or institutions concerned of charges covering actual costs.

All copies under the authority of this licence shall bear a statement to the effect that the copy in question "is being made available in this form by the authority of the copyright owner solely for the purpose of private study and research and may not be copies or reproduced except as permitted by the copyright laws without written authority from the copyright owner."

The foregoing shall in no way preclude the granting by the author of a licence to the National Library of Canada to reproduce the thesis and to lend or to sell copies of the same. For this purpose it shall also be permissible for Carleton University to submit the abovementioned thesis to the National Library of Canada.

Signature of Witness

Signature of Student

Date

Degree

Department of Student
The undersigned recommend to the Faculty of Graduate Studies and Research acceptance of the thesis

A DESIGN STUDY FOR
A DFT-BASED SPEECH MODIFICATION SYSTEM
ON A DSP MICROCHIP

submitted by Timothy C. Green, C.Eng., P.Eng., B.Eng., in partial fulfillment of the requirements for the degree of Master of Engineering.

[Signature]
Thesis Supervisor

[Signature]
Chairman
Department of Systems and Computer Engineering

Carleton University
September 1984
ABSTRACT

The short time spectrum of a voice signal and its time scale are closely related. This is obvious to anyone who has played a phonograph record at the wrong speed. If the turntable rotates too quickly, the time scale is compressed so that the length of time to play the record is reduced. However, the pitch appears to increase proportionately. Slowing down the record expands the time scale and drops the pitch.

The phase vocoder is based on the principle of short time Fourier analysis and requires no pitch estimation. In this thesis, algorithms and programs based on the phase vocoder are developed to drop the pitch of voiced signals by a factor of two. Recording the output and playing it back at twice the speed restores the pitch to its original value but cuts the time scale in half. This is useful in reducing playback time for talking books for the blind, and in reducing the storage requirements for taped minutes of meetings or conferences.

The principles of operation of the system are straightforward. Overlapping segments of the input signal are converted to the frequency domain where the spectrum is compressed to half its original width. The result is then converted back to the time domain.

The concepts developed in this thesis are evaluated for real time implementation on the TMS 320 digital signal microprocessor. The same principles may also be used in the design of systems to perform real time spectrum modifications other than the compression by a factor of two.
ACKNOWLEDGMENTS

I would like to acknowledge the Canadian Armed Forces, especially the Directorate of Communications Engineering and Maintenance, for providing the time and support required to carry out this study. I would also like to thank Dr. L. Robert Morris for suggesting the project and for sharing his expertise with me in all facets of this thesis. Further I would like to acknowledge the invaluable advice and encouragement given to me by Diane Dodds in the preparation and printing of this work.

Timothy C. Green
Ottawa, Ontario
August, 1984
# TABLE OF CONTENTS

## CHAPTER 1: INTRODUCTION

1.1 Background ................................................. 1  
1.2 Purpose .................................................. 4  
1.3 Contributions ............................................. 5  
1.4 Outline .................................................. 5

## CHAPTER 2: GENERAL ALGORITHM DETAILS

2.1 Introduction ................................................. 7  
2.2 Analysis 

  2.2.1 Definition of the STFT ................................. 7  
  2.2.2 The STFT as a DFT ................................. 8  
  2.2.3 The STFT as a Linear Filter ...................... 10  
  2.2.4 The Analysis Algorithm ......................... 10  
2.3 Spectrum Modification 

  2.3.1 Spectrum Compression ................................. 17  
  2.3.2 Phase Multiplication ............................... 19  
  2.3.3 Principles of Phase Unwrapping .................. 22  
  2.3.4 Phase Unwrapping in the Phase Vocoder ........ 23  
2.4 Synthesis ............................................... 27

## CHAPTER 3: THE PHASE Vocoder IN FLOATING POINT

3.1 Introduction ................................................. 31  
3.2 Parameter Selection ...................................... 31  
3.3 PHVOC7-POX .............................................. 34  
3.4 Testing and Performance ................................ 36  
3.5 Conclusion ............................................... 38

## CHAPTER 4: THE INTEGER VERSION OF THE PHASE Vocoder

4.1 Introduction ................................................. 39
Table of contents (continued)

4.2 Interrupt Service Routine ........................................... 41
4.3 Windowing and Transfer to FFT Workspace ....................... 42
4.4 FFT ........................................................................ 45
4.5 Spectrum Modification ............................................... 48
4.6 Inverse FFT ............................................................ 49
4.7 Overlap Add ............................................................. 52
4.8 Performance ............................................................. 52
4.9 Conclusion ............................................................... 54

CHAPTER 5: THE PHASE Vocoder ON THE TMS 320

5.1 Introduction ............................................................ 55
5.2 Program Changes ....................................................... 55
5.3 Time and Space ......................................................... 59

CHAPTER 6: CONCLUSION

6.1 Introduction ............................................................ 64
6.2 Distortion Causes and Solutions ................................... 64
6.3 Overflow Causes and Solutions .................................... 66
6.4 Conclusion ............................................................. 67
6.5 The Future ............................................................. 67

REFERENCES ...................................................................... 69
**LIST OF FIGURES**

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1.</td>
<td>STFA sliding window.</td>
<td>9</td>
</tr>
<tr>
<td>Figure 2a.</td>
<td>Linear filtering interpretation of STFA (complex output).</td>
<td>11</td>
</tr>
<tr>
<td>Figure 2b.</td>
<td>Linear filtering interpretation of STFA (real and imaginary output separated)</td>
<td>11</td>
</tr>
<tr>
<td>Figure 3.</td>
<td>Windowing of (x(m-R-N/2)).</td>
<td>15</td>
</tr>
<tr>
<td>Figure 4.</td>
<td>STFA procedure.</td>
<td>16</td>
</tr>
<tr>
<td>Figure 5a.</td>
<td>Magnitude spectrum of typical voiced speech segment.</td>
<td>18</td>
</tr>
<tr>
<td>Figure 5b.</td>
<td>Compressed voiced speech spectrum.</td>
<td>18</td>
</tr>
<tr>
<td>Figure 6.</td>
<td>Unwrapped phase for one harmonic.</td>
<td>21</td>
</tr>
<tr>
<td>Figure 7.</td>
<td>One sample windowed by different portions of the window.</td>
<td>28</td>
</tr>
<tr>
<td>Figure 8.</td>
<td>Phase vocoder speech modification system.</td>
<td>30</td>
</tr>
<tr>
<td>Figure 9a.</td>
<td>Analysis window time response.</td>
<td>33</td>
</tr>
<tr>
<td>Figure 9b.</td>
<td>Window magnitude spectrum.</td>
<td>33</td>
</tr>
<tr>
<td>Figure 10.</td>
<td>PV10 Input pointer sequences.</td>
<td>43</td>
</tr>
<tr>
<td>Figure 11.</td>
<td>PV10 Window alignment.</td>
<td>44</td>
</tr>
<tr>
<td>Figure 12.</td>
<td>Output pointer sequences.</td>
<td>53</td>
</tr>
<tr>
<td>Figure 13.</td>
<td>PV12 Input pointer sequences.</td>
<td>56</td>
</tr>
<tr>
<td>Figure 14a.</td>
<td>PV12 Window alignment showing full window.</td>
<td>58</td>
</tr>
<tr>
<td>Figure 14b.</td>
<td>PV12 window alignment showing only the 129 points of window actually stored</td>
<td>58</td>
</tr>
<tr>
<td>Figure 15.</td>
<td>Frame execution time and space requirements for TMS 320.</td>
<td>60</td>
</tr>
</tbody>
</table>
LIST OF ANNEXES

Annex A. PHVOC7.FOR program listing
Annex B. PV10.FOR program listing
Annex C. Support program listings
Annex D. PV12.FOR program listing
Annex E. TMS 320 program listings
Chapter 1 INTRODUCTION

1.1 BACKGROUND

When Cooley and Tukey published their important paper on the FFT in 1965 [1], the second generation digital computer had already become a standard fixture in research centres and universities. The new field of digital signal processing (DSP) was born. Areas and techniques that were before only of mathematical interest became computationally practical. New applications developed as improved algorithms and hardware permitted more complicated processing in less time. This trend continues today. Much work is being done in the area of real time processing of speech.

One general class of speech processing is the modification of spectrum qualities. When a hard-hat diver, for example, breathes a partial helium atmosphere, the spectrum of his voice changes in a complex manner, making it difficult to understand him [2]. A technique to render his voice more comprehensible would be useful, especially if it worked in real time. Other applications where spectrum modification is required include spectrum shifting for people with high frequency hearing difficulties [3], and spectrum compression for bandwidth reduction in a communications system.
A second class of speech processing is the modification of the time scale, slowing the signal down or speeding it up. The normal human speaking rate is between 110 and 180 words per minute, but the comprehension rate is two or three times this value [4]. Time scale compression could be used to decrease the archival storage requirements for tape recordings of meetings or lectures. It could also be used to speed up recorded books for the visually handicapped.

Note that by its very nature, time scaling cannot be done continuously in real time. It is, in the case of time scale expansion, impossible to get 16 seconds of speech out of a system for only 8 seconds of input, unless an internal buffer is used. Finite buffering precludes continuous processing. Likewise for time scale compression, the output cannot appear in real time faster than the input.

Spectrum manipulation and time scale modification are closely linked. This is obvious to anyone who has played a phonograph record at the wrong speed. If the turntable rotates too quickly, the time scale is compressed in that the length of time to play the complete record is reduced. However the pitch appears to increase proportionately. Slowing down the record expands the time scale and drops the pitch.

If in some manner, the pitch of a voice signal could be changed without affecting the time scale, then such a technique would fall into the first general class of speech modification mentioned above. It does, in fact encompass the second class as well. If the pitch of a recording
is halved, then playing it back at twice the speed will restore the pitch to its original value and compress the time scale. Thus changing the spectrum (and specifically the pitch) and modifying the time scale are two aspects of the same problem.

Linear predictive coding (LPC) characterizes a speech signal by estimates of vocal tract parameters and the period of the pitch. One would expect that given such a representation, the pitch parameter could be modified prior to synthesis to produce desired spectrum/time scale changes. Pitch estimation, however, is a difficult and heuristic process for a variety of reasons [5]. As well, the distortion caused by simply speeding up the playback of a recording is not just due to the change in pitch but also to the scaling of the spectral envelope and shift in formant frequencies [6].

Flanagan and Golden introduced the phase vocoder in 1966 [7] primarily as a tool for bandwidth reduction in an analogue communications system. The basic concept is simple, even if the details are not: periodically compute the spectrum of the signal, perform the required modifications directly on that spectrum, and convert the result back to the time domain. This method is appealing because no pitch estimation is required as in LPC or other vocoding methods. The phase vocoder was one of the first applications of the short term Fourier transform (STFT) to a practical problem.

Schafer and Rabiner [8] demonstrated how the FFT could be used in
an STFT analysis-synthesis system. This was computationally simplified by Portnoff [9] who showed that in the absence of spectrum modifications, his system was in fact an identity system, as opposed to pitch-excited LPC which is not. In 1977, Allen [10] introduced the overlap-add method of synthesis which was much less complex than the previously-used method of filter bank summation. Crookham [11] illustrated in 1980 how a short term Fourier analysis system with spectrum modification could be implemented using the overlap-add synthesis method. Recent papers, however, [3,4,6,12,13] have not fully exploited the simplicity and full computational savings afforded by these methods.

1.2 PURPOSE

The aim of this thesis is to investigate the digital phase vocoder as a means for real time speech processing. Attention is restricted to the specific case of lowering the pitch by a factor of two without affecting the time scale. As discussed, this is analogous to the problem of compressing the time scale by two without altering the pitch. This has specific applications in reducing the magnetic tape storage requirements of spoken conversation, but the algorithms involved may be adapted to other areas.

The concepts are developed while trying to minimize complexity and maximize computational speed with the view to eventual implementation in
real time on a specialized DSP microprocessor such as the Texas Instruments TMS 320. In fact, this device seems to be the first DSP microchip capable of a real-time implementation of such a system.

1.3 CONTRIBUTIONS

This thesis contains several original contributions to the field of speech processing. A number of disparate theoretical concepts have been combined for the first time in the design of a practical economical real-time system. Flanagan's phase vocoder [4] has been used as the basic structure and avoids the complicated phase unwrapping algorithms used in other studies. Crochiere's overlap-add method of short term Fourier analysis and synthesis [11] reduces the computational burden necessary in previous systems. A scheme of rotating buffers and pointers has been developed to handle the necessary circular and linear shifts of samples in an indirect but rapid manner. The algorithms developed have been adapted in a modular form for easy implementation on DSP microprocessors. The detailed structure for a TMS 320 version has been developed based on these algorithms and may serve as a basis for other real-time systems using the principles presented.

1.4 OUTLINE

Chapter 2 is a discussion of the digital phase vocoder using short
term Fourier analysis and synthesis with an intervening spectrum modification. Chapter 3 describes how the algorithms developed in Chapter 2 were implemented in floating point using FORTRAN on a PDP 11/55. Chapter 4 considers the factors involved in an integer version for a processor such as the TMS 320. In Chapter 5 the actual time and space requirements of a TMS 320 implementation are detailed. Chapter 6 concludes the paper with an identification of problem areas and recommendations for future research.

Listings of all programs mentioned in this study are included as annexes.
Chapter 2  GENERAL ALGORITHM DETAILS

2.1 INTRODUCTION

The proposed speech processing system is based on the digital phase vocoder. Since voice signals are assumed to be stationary only over short periods, techniques for short term Fourier analysis (STFA) and synthesis are used. This chapter is made up of three parts, explaining the analysis, spectrum modification and synthesis algorithms.

2.2 ANALYSIS

2.2.1 Definition of the STFT

The discrete short term Fourier transform (STFT) of a signal \( x(m) \) may be defined \([11]\) as:

\[
X_k(n) = \sum_{m=0}^{M-1} w(n-m) x(m) W^{-nk}_M
\]

\( W_M = e^{j2\pi/n} \)  

where \( m \) is the discrete time index, \( k \) is the discrete frequency index, \( M \)
is the number of frequency points in the transform and \( w(m) \) is the analysis window. The SIFT converts the one dimensional signal \( x(m) \), a function of time, into \( X_k(n) \), a two dimensional function of time and frequency.

### 2.2.2 The SIFT as a DFT

At a specific time \( m \), \( X_k \) may be interpreted as the conventional discrete Fourier transform of the product sequence \( w(n-m) \cdot x(m) \) for \(-N < m < N\). For successive times \( m+1, m+2 \ldots \) the analysis window can be thought of as 'sliding' along the signal sequence \( x(m) \). This sliding action is shown in Figure 1. The calculation of \( X_k \) for each \( m \) implies moving the analysis window one point relative to the input sequence, multiplying point by point, and evaluating the Fourier transform of the result. This gives the SIFT relative to the sliding time origin \( m \). To refer the SIFT to a fixed origin, the phase of the spectrum calculated in successive frames must be multiplied by a linearly increasing amount. A simplified procedure to do this is described in section 2.2.4 below.

Using this Fourier transform interpretation of (1), several required characteristics of the analysis window may be deduced. First, the window should be of finite length to ensure the tractability of the actual calculation and to guarantee that the summation in (1) is finite. Second, the window should be long enough to include sufficient pitch periods of voiced speech so that the short term periodicity may be
Figure 1. STFA sliding window

Note that the window is discrete and not continuous as the window envelope may imply.
analyzed. Third, the window must be short enough so that the segment of
\( x(m) \) it covers may be considered approximately stationary.

### 2.2.3 The DFT as a Linear Filter

Equation (1) can be interpreted in another manner. It may be
considered to be a function of time for a fixed frequency value \( k \). As
such, \( X_k \) is the convolution of \( w(n) \) with \( x(n) \) \( w^{-nk} \) and may be
visualized as a linear filtering operation as shown in Figure 2.

In this case, the spectrum of \( x(n) \) at a certain frequency index \( k \)
is shifted down to zero. If \( w(n) \) is chosen such that it is the impulse
response of a lowpass filter whose bandwidth is narrow with respect to
the spectrum of \( x(n) \), then the output \( X_k(n) \) will depend essentially only
on \( x(n) \). The effect is a bank of \( N \) identical bandpass filters. They are
spaced \( f_s/N \) apart, centred on harmonics \( k=0,1,2,...N-1 \). Here \( f_s \) is the
sampling frequency. In order to avoid overlap of the filters, the cutoff
frequency of \( w(n) \) should be \( f_s/2N \).

### 2.2.4 The Analysis Algorithm

In most applications, the window may be moved more than one point
relative to the time sequence between evaluations of the DFT. Doing so
cuts down on the computational load. Following the development in [11],
equation (1) may be more generally written as:
Figure 2a. Linear filtering interpretation of STFA (complex output)

Figure 2b. Linear filtering interpretation of STFA (real and imaginary outputs separated)
\[ X_k(sR) = \sum_{m=-\infty}^{\infty} v(sR-m) x(m) W_{n}^{-mk} \tag{2} \]

where \( R \) denotes the distance in points that the window is moved between successive DFT's (one frame), and \( s \) is the frame index, equal to 0, 1, 2, ... By using the substitution \( r = m - sR \), (2) becomes:

\[ X_k(sR) = \sum_{r=-\infty}^{\infty} v(-r) x(r+sR) W_{n}^{-(sR+r)k} \tag{3} \]

As before, \( X_k(sR) \) is the STFT referred to the fixed time origin \( s=0 \), \( m=0 \). Note that if the analysis window is symmetric around \( w(0) \), \( w(-r) = w(r) \). Factoring (3) yields:

\[ X_k(sR) = \left( \sum_{r=-\infty}^{\infty} v(-r) x(r+sR) W_{n}^{-rk} \right) W_{n}^{-sRk} \tag{4} \]

\[ = X_k^{-}(sR) W_{n}^{-sRk} \tag{5} \]

It can be seen that \( X_k^{-}(sR) \) is also an STFT, but instead of having a fixed reference as does \( X_k(sR) \), it is based on the point \( m=sR \).
corresponding to the zero point in the analysis window. This reference
'slides' between frames with successive values of \( \alpha \). Note that the two
SIFT's differ only in the phase factor of \( \tilde{V}_M^{-\alpha Rk} \).

Using the substitution \( r = pM + m \), the sliding reference SIFT can be
written as a DFT:

\[
X_k^{\sim}(sR) = \sum_{m=0}^{N-1} \left( \sum_{p=-\infty}^{\infty} w(-pM-m) x(sR+pM+m) \right) \tilde{V}_M^{-\alpha k}
\]

(6)

\[
= \sum_{m=0}^{N-1} x_{-m}^{\sim}(sR) \tilde{V}_M^{-\alpha k}
\]

(7)

If \( x_{-m}^{\sim}(sR) \) is circularly shifted by \( sR \) points, then \( X_k^{\sim}(sR) \) will
undergo a phase shift of \( \tilde{V}_M^{-sRk} \):

\[
\sum_{m=0}^{N-1} x_{-(m-sR) \text{mod } N}^{\sim}(sR) \tilde{V}_M^{-\alpha k} = X_k^{\sim}(sR) \tilde{V}_M^{-sRk}
\]

(8)

\[
= X_k^{\sim}(sR)
\]

(9)
Time \( X_k(sR) \), the desired fixed time reference STFT can be evaluated by taking the time aliased windowed signal \( x_{m}(sR) \), circularly shifting it by \( sR \) points, and taking the DFT of the result. In practice, \( N \) input samples \( x(m+sR-N/2) \) where \( m=0,1,2,... N-1 \), are windowed by \( w(-m) \) as shown in Figure 3. Here \( w(0) \) is aligned with the sample at \( m-N/2 \). To make the centre sample of the result correspond to the first sample of the transform, an additional circular shift of \( N/2 \) points is required. This implies a multiplication of \( w_{N/2} = (-1)^k \) in the frequency domain.

The resulting procedure for calculating the STFT \( X_k(sR) \) is shown in Figure 4 and is repeated every frame. Briefly the operation is:

1. shift the time samples by \( s \) points (discard the \( s \) oldest ones and pick up \( s \) new ones);

2. multiply the last \( N \) time samples by window \( w(-m) \);

3. circularly shift the result by \( (N/2 + sR) \mod N \) points, where \( s = 0, 1, 2 \ldots \) in successive frames; and

4. take the \( N \) point DFT.
Figure 3. Windowing of $x(m+sR-M/2)$, $s=0,1,2 \ldots M-1$

Note that the window is discrete and not continuous as the window envelope may imply.
Figure 4. STFA procedure
2.3 SPECTRUM MODIFICATION

2.3.1 Spectrum Compression

As stated in Chapter 1, the proposed system is designed to lower the pitch of voiced sounds by a factor of two. Such an effect may be accomplished by compressing the short time spectrum to half its initial width.

The magnitude of the STFT of a segment of voiced speech typically appears as shown in Figure 5a. The three peaks in the magnitude envelope at F1, F2, and F3 represent the locations of the formant frequencies. The ripple on the envelope is due to the pitch of the voiced segment. If the magnitude spectrum is compressed by a factor of two, i.e., to half its bandwidth, it would appear as shown in Figure 5b. In this case, it is obvious that the pitch ripple has been reduced to half its former value, the formant peaks are half as far apart and they appear at half the frequency they did before. In other words, all frequencies have been scaled downwards by a factor of two.

If just the magnitude spectrum were to be compressed, this could be accomplished by simply downsampling the spectrum in frequency. This means replacing the harmonic at index $k$ with the harmonic at $2k$ for all harmonics $0 \leq k \leq M/4$ where $M$ is the number of points in the discrete transform. Since the magnitude spectrum of a real signal is even, a symmetric compression must be performed on the top half of the spectrum.

-17-
Figure 5a. Magnitude spectrum of typical voiced speech segment

Figure 5b. Compressed voiced speech spectrum
2.3.2 Phase Multiplication

When spectrum compression as just described is performed, the phase spectrum must be modified as well as the magnitude spectrum, in a manner which is not quite so simple. This is called phase multiplication and the need for it is explained below.

As shown in Figure 2, the STFA process can be thought of as a linear filtering process. For a given value of \( k \), the output \( X_k(n) \) is a discrete complex function of time. The real and imaginary parts of \( X_k(n) \) may be separated as shown in Figure 2b. Letting \( \phi_k = 2\pi k/N \), consider a simple input signal \( x(n) = \cos(\phi_o n) \). As a function of discrete time, then the real part of \( X_k(n) \) before compression will be:

\[
s_k(n) = [\cos(\phi_k n) \cos(\phi_o n)] * w(n)
\]

\[
= 1/2 \{ \cos[(\phi_k - \phi_o)n] + \cos[(\phi_k + \phi_o)n] \} * w(n) \tag{10}
\]

Similarly, the imaginary part is:

\[
b_k(n) = -1/2 \{ \sin[(\phi_k - \phi_o)n] + \sin[(\phi_k + \phi_o)n] \} * w(n) \tag{11}
\]

If the lowpass window \( w(n) \) is chosen so that it has unity gain in its passband, and so that \( \phi_k + \phi_o \) is outside that passband and \( \phi_k - \phi_o \) inside, then \( a_k \) and \( b_k \) become:
\[ a_k(n) = \frac{1}{2} \cos \left( \Theta_k - \Theta_o \right)n \]  
(12)

\[ b_k(n) = -\frac{1}{2} \sin \left( \Theta_k - \Theta_o \right)n \]  
(13)

The magnitude of \( X_k(n) \) is a constant, and its phase is:

\[ \phi_k(n) = -\tan^{-1} \left( \frac{b_k(n)}{a_k(n)} \right) \]

\[ = \left( \Theta_k - \Theta_o \right)n \]  
(14)

This \( \phi_k(n) \) is the phase of the one harmonic at frequency index \( k \) for a constant input. It is a linear function of the time index \( n \) and has a slope equal to the frequency of the harmonic present in that section of the filter bank as shown in Figure 6. Note that \( \phi_k(n) \) is the unwrapped phase constantly increasing in value and not its principal value.

Voiced speech signals are much more complex than the simple signal illustrated and can be considered to be made up of a collection of sine waves spaced apart at the pitch frequency. If the bandwidth of the filter \( w \) is such that it is narrower than the pitch frequency, at most only one such sine wave will be in its passband and the above development holds.
Figure 6. Unwrapped phase for one harmonic
If the spectrum is to be compressed to half its original value, two factors change. First, each analysis frequency in Figure 2 will be shifted downwards to half its original value. Thus $\theta_k$ becomes $\theta_k/2$. Second, the effective value of $\theta_0$, the input signal frequency at the harmonic $k$ is reduced to half its original value. These combine so that for the compressed spectrum, (14) becomes:

$$d_{k/2(n)}^{\text{compressed}} = 1/2 \ d_k(n) = 1/2 \ (\theta_k - \theta_0)n$$  \hspace{1cm} (15)

### 2.3.3 Principles of Phase Unwrapping

The most straightforward method of accomplishing this phase multiplication by $1/2$ is described in [12]. At each time $n$, the real and imaginary parts of the harmonic $k$ are converted to polar coordinates. The phase angle thus obtained is the principal value and is in the range $\pm \pi$. The unwrapped value of this phase is estimated by adding $\theta_k$ to the value unwrapped at time $n-1$. The actual unwrapped value is then calculated by adding successive multiples of $2\pi$ to the principal value until it is within $\pi$ of the estimate. The first backwards difference in time of the unwrapped phase is multiplied by $1/2$ and added to the accumulated value of the modified phase. The magnitude and modified phase are then converted back to rectangular coordinates. This amounts to discrete integration of a modified discrete differential, effectively reducing the slope of the line in Figure 6 to half its original value. If at time $n-1$, the modified phase $d_{k/2(n-1)}$ were known, then
\[ \dot{d}_{k/2}^{(n)} \text{compressed} = \dot{d}_{k/2}^{(n-1)} \text{compressed} + (1/2) \frac{d}{dn} \] (16)

While the principle of the above technique is sound, its implementation on a microprocessor presents problems. Conversion to polar coordinates requires either the arctangent function or the iterative process described in [12]. In both cases the question of the quadrant of the result must be resolved, and this is not an intractable problem. However, since the constantly increasing unwrapped phase values are maintained, they would sooner or later cause an overflow. This would happen sooner rather than later in an integer implementation. The heuristic unwrapping method is non-deterministic, a characteristic one tries to avoid in real-time programming.

2.3.4 Phase Unwrapping in the Phase Vocoder

The unwrapped value of the unmodified phase is only required to determine its time derivative. If in some way, the derivative could be calculated otherwise, then (16) could be implemented without the problems described above. Only the principal value, and not the unwrapped value of the modified phase is required since it is converted to rectangular coordinates anyway.
Taking the derivative of (14) yields:

\[
\frac{d\phi_k}{dn} = - \frac{d}{dn} \left( \tan^{-1} \left( \frac{b_k(n)}{a_k(n)} \right) \right)
\]

\[
= b_k(n) \frac{d}{dn} \left[ a_k(n) \right] - a_k(n) \frac{d}{dn} \left[ b_k(n) \right] \]
\]

\[
\frac{a_k(n)^2 + b_k(n)^2}{(a_k(n)^2 + b_k(n)^2)}
\]

(17)

For discrete time, the derivatives of \(a_k\) and \(b_k\) may be approximated by backwards differences as:

\[
\frac{d}{dn} \left[ a_k(n) \right] = \frac{1}{T} \left[ a_k(n) - a_k(n-1) \right]
\]

(18)

\[
\frac{d}{dn} \left[ b_k(n) \right] = \frac{1}{T} \left[ b_k(n) - b_k(n-1) \right]
\]

(19)

where \(T\) is the period in discrete time \(n\) across which these differences are evaluated and is equal to unity. Using these approximations, equation (17) becomes:
\[ \frac{d\phi(n)_{\text{est}}}{dn} = \frac{b_k(n) a_k(n) - b_k(n) a_k(n-1)}{a_k(n)^2 + b_k(n)^2} - \frac{a_k(n) b_k(n) + a_k(n) b_k(n-1)}{a_k(n)^2 + b_k(n)^2} \]

Substituting the values from (12) and (13) into (20) show that this approximation for the phase derivative is equal to \( \sin(\theta_k - \Theta_0) \), whereas the desired value is \( \Theta_k - \Theta_0 \). For small values of \( \Theta_k - \Theta_0 \), the error in estimation will be low since for small values, the sine function is approximately equal to its argument. Because the phase derivative is calculated every \( R \) sample times, or every \( R/f_s \) seconds, and due to the lowpass nature of the analysis filter, the maximum value that \( \Theta_k - \Theta_0 \) will take on is:

\[ \Theta_{\text{max}} = 2\pi \Theta_0 (R/f_s) \]  

(radians)  

(21)
where $\theta_c$ is the cutoff frequency in Hz of the filter $v$. The maximum percentage error in using (20) to evaluate the phase derivative is thus:

$$\text{Err}_\text{max} = 100 \left[ \frac{\theta_{\text{max}} - \sin(\theta_{\text{max}})}{\theta_{\text{max}}} \right]$$

For $\theta_c = f_s/2R$, $\text{Err}_\text{max}$ is less than 10%.

Equation (20) is a conservative estimate for the phase derivative; that is to say that the estimate it returns is always less in absolute value than the actual phase derivative. Since voice signals are not stationary over the long term, the error in using the estimated phase derivative in equation (16) will sometimes be positive and sometimes negative. In integration, the error will thus average out to zero. Equation (20) uses no trigonometric functions, making it well suited for use on a real-time, fixed point microprocessor based system such as the TMS 320.

For algorithm start up, an initial condition of zero for $\phi_k$, $(a_k=1, b_k=0)$ may be used. If the phase derivative were known exactly, the desired phase could be calculated exactly with one application of (16). Even using the estimator of (20), the integrated phase seems in practice to converge quickly to the desired value.
2.4 SYNTHESIS

In the analysis section, the STFT was evaluated by windowing and circularly rotating overlapping input sections and then taking the DFT of the result, as shown in Figure 4. For synthesis from the STFT, this process must be reversed. The effect of the analysis DFT will be reversed by an inverse DFT. The circular shift in the analysis section may be reversed by a similar shift in the opposite direction. The remaining problem is to undo the effect of the window.

For the purpose of illustration, consider the case where \(M/R = 4\). In the STFA procedure described, any particular sample \(x(m)\) will be windowed in \(M/R = 4\) successive frames, each time by a different portion of the window as shown in Figure 7. These portions of the window will be separated by \(R\) points. The problem then, is to recover \(x(m)\) given the four windowed samples.

It has been shown [10] using the Poisson summation formula, that if the cutoff frequency of the filter \(w(n)\) is at most \(f_s/2R\) where \(f_s\) is the sampling frequency, then

\[
\sum_{k} w(mR-k) = \text{constant}
\]  

(23)

independent of \(k\). More precisely, if \(W(0) = R\), then the summation of
Figure 7. One sample windowed by different portions of the window in successive frames.
(23) is equal to unity. Using this fact, the 'unwinding' process then amounts to a simple overlap-add procedure since for the example in Figure 7:

\[ x(n) = x(n) \left[ w(-k) + w(-k) + w(k) + w(2k) \right] \quad (24) \]

The synthesis process is illustrated in the lower portion of Figure 8 as part of the complete system.
Figure 8. Phase vocoder speech modification system
Chapter 3 THE PHASE VOCODER IN FLOATING-POINT

3.1 INTRODUCTION

The algorithms described in Chapter 2 for the phase vocoder speech modification system were programmed in FORTRAN on a PDP-11/55. Floating-point was used throughout. The program was made as general as possible so that the effects of different parameter values could be observed. Speed or efficiency were not goals at this stage.

Section 3.2 discusses the values chosen for the variable parameters discussed in Chapter 2. Section 3.3 describes the floating point program PHVOC7.FOR which is a direct implementation of Figure 8. Section 3.4 discusses the program's testing and performance. Section 3.5 concludes with a summary of the principles demonstrated by the program.

3.2 PARAMETER SELECTION

As a starting point for determining the parameter values for the system, a sampling rate of $f_s = 8$ KHz was assumed. This is standard for many voice systems. To avoid aliasing, the input signal must be band-limited to less than 4 KHz. This limiting process may be done
either by a simple analogue or digital filter external to the phase vocoder and was not further considered as part of the system.

A transform size of \( M = 256 \) points was chosen. This is a power of two and thus permitted a standard FFT program to be used. As well, for the 8 KHz sampling rate chosen, 256 points represent 32 msec of speech. It was felt that for voiced speech, such a segment would be approximately stationary and contain three to four pitch periods.

For reasons of flexibility, a Kaiser lowpass function [14] was used for the analysis window. The characteristics of such a function may be changed by varying a single parameter \( \alpha \). A length of 255 points was chosen for the window. This means that it will be of linear phase and symmetric about its centre point, and thus be uniquely specified by 128 points. This may be important in a DSP microprocessor implementation where storage space is at premium. Using a value of \( \alpha = 2.0 \) and a cutoff frequency of \( f_c/2M = 15,625 \) Hz gives the function whose impulse and frequency responses are shown in Figure 9.

The other parameter left to determine was \( R \), the number of points by which the window should slide between frames. Applying the Nyquist criterion to the filter bank analogy of STFA means that the output of each filter bank section should be sampled at least at twice the rate of its highest frequency. If the highest frequency passed is taken to be that at which the main lobe is equal to the highest side lobe, then from Figure 9, this is \( 1.2/256 \) of the sampling frequency, or 37.5 Hz. The
Kaiser window:
cutoff freq = 0.001953 f_s
α = 2.0
length = 255
gain = 256.0

Figure 9a. Analysis window time response (only half shown, symmetric about n=0)

Figure 9b. Window magnitude spectrum around zero frequency
frame rate should be twice this or 75.0 Hz. This corresponds to one frame every 107 sample times, and is a maximum value for \( K \). A conservative value of \( K = 64 \) was chosen.

It can be seen that the cutoff frequency of the window is less than \( \frac{f_s}{2K} \) for the values chosen. As explained in Chapter 2, this is sufficient to permit the overlap add procedure to be used in the synthesis section.

3.3 PHVOC7.FOR

The floating point phase vocoder program PHVOC7.FOR is found in Annex A along with its subroutines. It does not operate in real time. Rather, it reads input data one block at a time from an external device, performs the processing directly as shown in Figure 8 and writes one block of data to an external device, repeating the processing until all input data has been read. The first two thirds of the program are devoted to setting the parameters to control the analysis, spectrum modification and synthesis sections which are in the last third of the program. PHVOC7.FOR simply forms a framework that calls different subroutines. This provides flexibility in that individual portions may be modified without disturbing the rest of the program.

PHVOC7 prompts the user to enter a value \( N \) for the transform size. This is assumed to be a power of two and is limited to being less than

-34-
or equal to 512. The program prompts for input and output overlap parameters R1 and R2. These are the values by which the input and output shift registers of Figure 8 are shifted each frame, and in general need not be the same [8]. For the present system, R1 = R2 = R.

The analysis window can be chosen either as rectangular or as a Kaiser lowpass function. If a Kaiser window is selected, the user is prompted for the window length, the cutoff frequency and the parameter \( a \). The cutoff frequency is specified as a number between 0.0 and 1.0 where the sampling frequency \( f_s = 1.0 \). The user is also prompted for a scaling factor SCALE which is applied to all elements of the window. For a window whose centre point is to be 1.0, this scaling factor should be specified as half the reciprocal of the cutoff frequency. This analysis window scaling may be regarded as a useful heuristic for controlling the input gain to the system. In a similar manner, a synthesis window may be specified to be either rectangular (effectively no window) or a Kaiser lowpass function.

Spectrum modification may or may not be selected. If it is not, synthesis is performed on the unmodified short time spectrum to produce an output equivalent to the input signal, i.e., an identity system.

As the last part of the parameter specification, the user is prompted to specify the input and output data files. The input data is expected as a sequential file of NSAMI + 1 unformatted integers, the first of which is NSAMI and indicates the number of integers that
follows. The program assumes these integers came from a 10 (sign + 9) bit A/D converter and writes the output file in the same format as the input.

Once the parameters have been set, the analysis – spectrum compression – synthesis cycle occurs in a block by block fashion as shown in Figure 8 until the complete input file has been read. The analysis, compression and synthesis operations are performed by subroutines ANLSYS, SPCMOD, and SMTH respectively. They directly implement the algorithms discussed in Chapter 2.

The number of samples output will be less than the number of samples input. This is because the transients generated during algorithm start up are not sent to the output.

3.4 TESTING AND PERFORMANCE

PHVOC7 was tested using a group of programs specially designed for this purpose. The program WOVERE was used to generate synthetic phoneme sequences of about 17000 samples to use as input. DECIMS was designed to downsample sequences by an integer factor. SEQSAT was used to play sequences back through the D/A. Listings for these programs are included in Annex C.

The output of PHVOC7 was compared on a sample by sample basis to
the input for the case where spectrum modification was not selected. As expected with an identity system, within a scale factor the input and output were identical. This is in contrast to pitch-excited LPC which, even in theory before any implementation compromises, does not have the property of being an identity system.

Using a value of SCALE = 256 for the analysis window resulted in the output signal exceeding the 10 bit D/A range. This was solved by reducing SCALE to 192. Once this was done, the output of PHVOC7 when spectrum modification was selected was indeed the same as the input, except at half the pitch. Downsampling this by two resulted in a sequence very close to the original phoneme sequence, at the same pitch but lasting only half as long. There was a very slight 'rain barrel' reverberation present in the output. This is discussed in Chapter 6.

Since the input and output shift registers are initialized to all zeros, valid output samples are not obtained until the start of the fourth frame, i.e. starting with the sample 193. At 8 kHz, the 192 sample start-up time represents 24 mssec. Actually PHVOC7 suppresses the first 192 available samples on starting in order to give only valid output.
3.5 CONCLUSION

From PHVOC7, the following conclusions may be drawn:

1. Without any spectrum modification, the phase vocoder analysis-synthesis procedure is an identity and thus starts out being perfect.

2. The phase vocoder with the spectrum modification carried out as described accomplishes the task of reducing the pitch of a voiced signal by a factor of two.
Chapter 4 THE INTEGER VERSION OF THE PHASE VOCODER

4.1 INTRODUCTION

As mentioned in Chapter 3, the FORTRAN program PHVOC7.FOR was not designed for speed or efficiency. Once the algorithm of Figure 3 was tested and confirmed to be workable, however, a second program PV10.FOR was written. In view of eventual implementation on the TMS 320 microprocessor, PV10.FOR was designed to simulate as much as possible the actual integer operations that would have to carried out on the TMS 320, taking up a minimum of memory space. While FORTRAN floating point was used, integer operations were simulated by truncating real numbers to their integer parts and checking to make sure no overflow had occurred.

Since the program in this chapter is designed with implementation on the TMS 320 in mind, it is important to understand the characteristics of this microprocessor designed specifically for DSP. The TMS 320 is a reduced instruction set computer (RISC) with a 16 bit word size. It has a single 32 bit accumulator, two 16 bit index pointers, and an integral 16 x 16 multiplier that yields a 32 bit result. Two's complement arithmetic is used and most operations including multiplication take 200 ns. The TMS 320 has a Harvard memory
architecture with separate 4096 word program and 144 word data spaces; however provision is made for data to cross from one memory space to another. Integral barrel shifters permit multiplications by powers of two in moving data to and from the accumulator. These multiplications are implicit in the instructions and add nothing to the execution time or size of a program that uses them.

The optimized program PV10.FOR discussed in this chapter is found in Annex B. It uses the same FFT and FILTER subroutines included with PHVOC7.FOR. The first portion of the program is devoted to generating the window and initializing memory locations. The actual start of the phase vocoder section is clearly identified and this section is repeated once per frame. This breaks down into six logical steps again clearly identified in the program listing and are described individually in sections of the chapter. These are:

1. interrupt service routine (input and output processing);
2. windowing and transfer to the FFT workspace;
3. FFT;
4. spectrum modification;
5. inverse FFT; and

6. overlap-add for output.

The chapter is concluded by a discussion of the performance of the program. Throughout, the values for the variable parameters determined in Chapter 3 are used.

4.2 INTERRUPT SERVICE ROUTINE (ISR)

In actual TMS 320 implementation, processing would be interrupt driven, with interrupts occurring at a rate of 8 kHz. At each interrupt, one input sample would be shifted into the input shift register, and one output sample shifted out of the output register as shown in Figure 8. The processing of one frame would be completed during the period of 64 interrupts.

To avoid samples being moved in and out while they are being operated on by the frame processing section of the program, the input and output shift registers, instead of being the required 256 elements, were set to 320 elements long, or five blocks of 64 elements each. In this way, 256 elements are available for processing in the current frame, and the other 64 are being filled by input samples, or emptied of output samples as appropriate by the ISR.
The start of each 64 element block destined for block processing is indicated by a pointer: IIP(1)...IIP(4) for input, IOP(1)...IOP(4) for output. Another pointer IPIN points to the start of the block that receives the next 64 input samples and IOUT points to the start of the block that provides the next 64 output samples.

Between frames, samples in both shift registers are shifted by exactly 64 elements. This is accomplished without moving the samples themselves but simply by updating the pointers. This logical shift register effect on the input is shown for three successive frames in Figure 10. Note how IIP(1) always points to the oldest set of 64 points, and IIP(4) to the newest. IPIN is moved in such a manner that the input samples overwrite what were the oldest points in the last frame. A similar logical shift register effect for the output is described in Section 4.7 below.

4.3 WINDOWING AND TRANSFER TO FFT WORKSPACE

The analysis window is 255 points long, aligned with the 256 point input segment as shown in Figure 11. The centre point of the window about which it is symmetric is the 128th, and it lines up with the 129th point of the input. The first point of the input is not covered by the window and is thus multiplied by zero. Since the window is symmetric, only 128 points of it need be stored. Using the same 64 point portion of the window, blocks 1 and 4 from the input shift register may be
Figure 18. PV10 input pointer sequences
Figure 11. PV10 window alignment
windowed, followed by blocks 2 and 3 using the other 64 point portion of the window. This is shown in the PV10.FOR listing.

As shown in Figure 8, after 256 samples have been windowed, they undergo a variable circular shift which is always a multiple of 64 points. Rather than actually shifting the points, a set of four pointers IFTP(1) ... IFTP(4) is used to shift the blocks logically. IFTP(n) points to the start of the FFT workspace section that receives the result of windowing with the nth section of the window. The changing sequence of these pointers is shown in Figure 10. Note that although only 128 points of the window are actually used, the window is shown logically as being 255 points long.

4.4 FFT

In order to conserve memory space and due to the fact a 128 point FFT program would be available for the TMS 320 [15], it was decided to perform the DFT of 256 real points using one 128 point complex FFT. This method is described in [16]. For the 256 point real sequence x(n), this involves creating a new 128 point complex sequence c(n) whose real and imaginary parts are made up of the even and odd points respectively of x(n). Half of X(k), the DFT of x(n), may be recovered from C(k), the DFT of c(n), by an unraveling process of the spectrum. Only half is required since if x(n) is real, X(k) will be conjugate even. This unraveling process involves the evaluation of:
\[
I(k) = \frac{1}{4} \left\{ \left[ C^* (128-k) + C(k) \right] + \sqrt{256} \left[ C^* (128-k) - C(k) \right] \right\} 
\] (25)

\[
I(128-k) = \frac{1}{4} \text{CONJ} \left\{ \left[ C^* (128-k) + C(k) \right] \right. \\
\left. + \sqrt{256} \left[ C^* (128-k) - C(k) \right] \right\} 
\] (26)

for \( k = 0, 1, 2 \ldots 64 \), where * and CONJ both denote complex conjugation and 
\( \sqrt{256} = e^{(j2\pi/256)} \).

Expanding (25) and (26) shows that, using the same variable names as in FV10.FOR, the following need be calculated:

\[
V_1 = \cos \frac{2\pi(64+k)}{256} 
\] (27)

\[
V_2 = \sin \frac{2\pi(64+k)}{256} 
\] (28)

\[
V_3 = \text{Re}( C^* (128-k) ) + \text{Re}( C(k) ) 
\] (29)

\[
V_4 = \text{Im}( C^* (128-k) ) + \text{Im}( C(k) ) 
\] (30)

\[
V_5 = \text{Re}( C^* (128-k) ) - \text{Re}( C(k) ) 
\] (31)

\[
V_6 = \text{Im}( C^* (128-k) ) - \text{Im}( C(k) ) 
\] (32)

Then using the following combinations:
\[ V7 = V1 \cdot V5 \]  
(33)

\[ V8 = V2 \cdot V4 \]  
(34)

\[ V9 = V1 \cdot V4 \]  
(35)

\[ V10 = V2 \cdot V5 \]  
(36)

\[ V11 = V7 + V8 \]  
(37)

\[ V12 = V10 - V9 \]  
(38)

Half the unraveled spectrum may be calculated by:

\[ X(k) = \frac{1}{4} \left[ (V3+V11) + j(V12-V6) \right] \]  
(39)

\[ X(128-k) = \frac{1}{4} \left[ (V3-V11) + j(V12+V6) \right] \]  
(40)

The values of cosine and sine in (28) and (29) come from a pre-calculated table stored in memory. In the TMS 320, this table would also used by the FFT.

As mentioned, the TMS 320 has a 16 x 16 bit multiplier that gives a full result in the 32 bit accumulator. It is convenient to consider both
inputs to the multiplier as numbers between 0 and 1, multiplied by $32767 = 2^{15} - 1$. As such the product will never overflow and the 15 most significant bits plus the sign may be obtained by dividing by $2^{15}$ as described in [17]. Divisions by $2^{15}$ and $2^{12}$ when storing the accumulator may be included at no execution time penalty. The scaling used in PV10.FOR uses these facts.

The TMS 320 FFT program that is simulated in PV10.FOR divides all inputs by $128 = 2^7$ to avoid internal overflow. If the inputs are considered as numbers between 0 and 1, multiplied by a scale factor of $2^{15}$, then at the output of the FFT the effective scale factor is reduced to $2^{(15-7)} = 2^8$. During the unraveling process, each harmonic is multiplied by $2^4$ to help minimize the effects of roundoff errors.

Note that explicit reording of odd and even real samples into real and imaginary parts for the input to the FFT is not required since the FFT used assumes that real and imaginary parts are in successive memory locations. Note as well that only every second harmonic is unraveled since the intervening ones will be over-written in the spectrum compression of the next stage.

4.5 SPECTRUM MODIFICATION

The spectrum modification stage of PV10.FOR is made up of two parts. The first is the spectrum compression which is implied in the way
the harmonics are indexed: harmonics 0,2,4,...126 are manipulated and
stored back into the places of harmonics 0,1,2,...63. The upper portion
of the spectrum normally would be set to zero after the compression.
This effect is accomplished indirectly in the inverse FFT section.

The second part is the phase multiplication by 0.5. For every second
harmonic, arrays A and B hold the real and imaginary part of the
spectrum from the previous frame and PHI holds the value of the
accumulated phase. V13 is the magnitude of one harmonic, and V12 is the
square of that magnitude. V15 is the phase derivative evaluated directly
as shown in (20) and V17 is the updated modified phase, calculated
according to (16).

This section of the program calls four subroutines to perform
functions that are not part of the instruction set of a microprocessor
such as the TMS 320. These are WNSSRT (square root), TNSSDIV (division),
TNSSCOS (cosine of an arbitrary angle) and TNSSSIN (sine of an arbitrary
angle).

4.6 INVERSE FFT

Since a 128 point complex FFT is used to perform a 256 point real
inverse DFT, the spectrum must be revealed beforehand as described in
[16]. This revealing amounts to changing the half spectrum X(k) into the
complex sequence C(k) such that:
\[ C(k) = \frac{1}{4} \left\{ X(k) + X^*(128-k) \right\} + \sqrt{\frac{64-k}{256}} \left\{ X(k) - X^*(128-k) \right\} \] \hspace{1cm} (41) \\
\[ C(128-k) = \frac{1}{4} \text{CONJ} \left\{ X(k) + X^*(128-k) \right\} - \sqrt{\frac{64-k}{256}} \left\{ X(k) - X^*(128-k) \right\} \] \hspace{1cm} (42) 

After the spectrum was compressed in the previous section, the upper portion should have been set to all zeros. This need not be done explicitly, but may be accomplished by letting \( X^*(128-k) = 0 \) for \( k=0,1,...,63 \) when evaluating (41) and (42). Since an FFT is used to calculate an inverse DFT, the inputs (and the outputs after the FFT) must be replaced by their complex conjugates. Using these facts, then in the notation of PV10.FOR, the following must be calculated:

\[ V_{20} = \cos \left( \frac{2\pi(64-k)}{256} \right) \] \hspace{1cm} (43)  \\
\[ V_{21} = \sin \left( \frac{2\pi(64-k)}{256} \right) \] \hspace{1cm} (44)  \\
\[ V_{22} = \text{Re} \left( X(k) \right) \cdot V_{20} \] \hspace{1cm} (45)  \\
\[ V_{23} = \text{Im} \left( X(k) \right) \cdot V_{20} \] \hspace{1cm} (46)  \\
\[ V_{24} = \text{Re} \left( X(k) \right) \cdot V_{21} \] \hspace{1cm} (47)  \\
\[ V_{25} = \text{Im} \left( X(k) \right) \cdot V_{21} \] \hspace{1cm} (48)
\[ V_{26} = V_{22} - V_{25} \]  \hspace{1cm} (49) \\
\[ V_{27} = V_{23} + V_{24} \]  \hspace{1cm} (50)

Then the revealed harmonics may be calculated by:

\[ C(k) = \frac{1}{2} \left[ \text{Re} \left( X(k) \right) + V_{26} \right] - \frac{j}{2} \left[ \text{Im} \left( X(k) \right) + V_{27} \right] \hspace{1cm} (51) \]

\[ C(128-k) = \frac{1}{2} \left[ \text{Re} \left( X(k) \right) - V_{26} \right] + \frac{j}{2} \left[ \text{Im} \left( X(k) \right) - V_{27} \right] \hspace{1cm} (52) \]

Once this complex sequence \( C(k) \) is calculated using (51) and (52) for \( k=0,1...64 \), the FFT is performed. It automatically divides the inputs by 128 as required for an inverse DFT. The result is a 128 point complex sequence of which the real parts represent the even points in the real 256 point output sequence, and the imaginary parts represent the negative of the odd points. This sign disparity results from using the FFT to calculate the inverse DFT. The complex inputs were conjugated and the odd output points are restored to their proper signs in the overlap-add section below.

As a final step, all points output by the FFT are divided by 8 to compensate for a multiplication of 64 on input, a division of 128 in the forward DFT and a multiplication of 16 in the post-FFT unraveling. The result is scaled properly for a 9+1 bit D/A.
4.7 OVERLAP-ADD

Since an FFT was used to compute the inverse DFT, even points are correct, but odd points have the wrong sign. Even points are thus overlapped and added to the output shift register, while the odd points are subtracted to correct for this difference. As well, instead of explicitly shifting 64 zeros into the shift register as shown in Figure 8, the samples from the FFT workspace destined for that block overwrite the former contents instead of being added to it.

Actually as previously mentioned, the output shift register is made up of 320 elements, or five blocks of 64. The circular shift required between the FFT workspace and the output shift register is effected by appropriately manipulating the pointers IOP(1) ... IOP(4) as shown in Figure 12. IPUT points to the start of the block that will provide the next 64 points.

4.8 PERFORMANCE

PV10 was tested with the same programs and sequences as PHVOC7. As with PHVOC7, SCALE was set to 192 to reduce saturation of the output. PV10 did produce an output whose pitch was half that of the input, but the 'rain barrel' reverberation was much more pronounced than in the floating point version and this carried over into downsampled sequences.
Figure 12. Output pointer sequences
During the post-DFT unraveling, all spectrum coefficients were multiplied by 16 to help reduce the effect of roundoff errors. This opens the possibility of a 16 bit overflow. For sequences composed of changing phonemes, overflow rarely happens, and when it does, it is very small. However since a sustained sequence such as a single phoneme of constant pitch is highly periodic, for such an input some of the unraveled spectrum components are quite large and cause repeated overflows. Solutions to these problems are discussed in Chapter 6.

Since the size of the input and output shift registers is greater in PV10 than it is in PHVOC7, the start-up time is increased. Valid output samples will not appear until the start of the fifth frame. At 8 kHz, this means a delay of 32 msec.

4.9 CONCLUSION

The phase vocoder implementation in simulated fixed point works, although with some distortion which is discussed in Chapter 6. This guarantees that the algorithms developed would work when implemented 'identically' on the TMS 320 or other fixed point machine with the same precision.
Chapter 5 THE PHASE VOCODER ON THE TMS 320

5.1 INTRODUCTION

Chapter 4 showed that the algorithm developed for the phase vocoder in PV10.FOR would work on the TMS 320, although with some distortion. It was discovered that changes to the program structure were required when carrying out the actual implementation.

The changes to the program are described in section 5.2 and embodied in program PV12.FOR found in Annex D. Section 5.3 contains the time and space requirements in actually implementing the algorithm on the TMS 320.

5.2 PROGRAM CHANGES

In PV10, the alignment of the input shift register and FFT workspace both vary with respect to the fixed window. Thus for example IIIP(1) and IFTP(1) point to the source and destination respectively of data passing through the first section of the window. These pointers
Figure 13. PV12 input pointer sequences
change from frame to frame, but the reference is always the window. This scheme could have been implemented directly on the TMS 320.

The crucial time-critical portion of the phase vocoder is the FFT which is called twice per frame. PV10 was developed in isolation from the TMS 320 version of the FFT used [15] and when the two were combined, it was discovered that the input pointer scheme could be modified to achieve a greater economy in execution time.

Instead of changing the input shift register and FFT workspace alignments with respect to the window, it was decided to 'move' the input shift register and window while keeping the FFT workspace as a fixed reference. Thus the pointers were redefined so that IIP(n) points to that section of the input shift register that would end up in the n-th section of the FFT workspace after passing through the section of window pointed at by IWP(n). This is illustrated in Figure 13. Note that in comparison with Figure 10, the flow of data is no different; only the pointers have changed. This allows windowed input data to be placed in the FFT workspace in a pre-determined fixed order, rather than a constantly changing sequence as in PV10. This reduces the number of data transfers in and out of the limited data memory that the FFT must make to get the data it needs in the order it needs it.

With the new input pointer scheme, only one of the IIP pointers changes at the end of each frame. A new 'pointer to a pointer' called IX was introduced to indicate which one. Since the output shift register
Figure 14a. PV12 window alignment showing full window.

Figure 14b. PV12 window alignment showing only the 129 points of window actually stored.
pointers IOP change in the same manner. IX shows which of these to update as well. The input and output sample pointers IPIN and IOUTP remain unchanged.

Another simplification was made by increasing the number of stored window values by one to 129. Explicitly storing a zero as the first value permits the windowing of all sections to be done in a direct and symmetric manner rather than making exceptions for the end points. Figure 14a shows how the windowing would be carried out if a whole 256 point window were stored. Figure 14b shows how the 129 points actually stored are applied to the four sections of the input sequence.

5.3 TIME AND SPACE

In real-time programming, program execution time can often be reduced at the expense of program storage space. The reverse is also true. Programming the TMS 320 is no exception to this, and for complicated programs such as the phase vocoder, the limits in time (125 frames per second) and space (4096 words) are quickly reached. Thus considerable juggling must be done in order implement the phase vocoder algorithms developed in this thesis on the TMS 320 in real time.

Notwithstanding the modifications and simplifications explained above, the program could still not be implemented so as to fit in the
<table>
<thead>
<tr>
<th>Description</th>
<th>Time (msec)</th>
<th>Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program initialization (done only once, outside of frame)</td>
<td>(est) 75</td>
<td></td>
</tr>
<tr>
<td>Data stored in program memory</td>
<td></td>
<td>1414</td>
</tr>
<tr>
<td>Interrupt service routine (one sample in, one sample out) x 64</td>
<td>0.371</td>
<td>22</td>
</tr>
<tr>
<td>Window and transfer to FFT workspace</td>
<td>1.145</td>
<td>140</td>
</tr>
<tr>
<td>Forward FFT</td>
<td>2.400</td>
<td>2249</td>
</tr>
<tr>
<td>Post-FFT unraveling and spectrum compression</td>
<td>3.323</td>
<td>296</td>
</tr>
<tr>
<td>Pre-inverse FFT unraveling</td>
<td>0.855</td>
<td>88</td>
</tr>
<tr>
<td>Inverse FFT</td>
<td>2.400</td>
<td></td>
</tr>
<tr>
<td>Overlap-add</td>
<td>0.812</td>
<td>103</td>
</tr>
<tr>
<td>End of frame processing</td>
<td>0.013</td>
<td>58</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td><strong>11.319</strong></td>
<td><strong>4445</strong></td>
</tr>
</tbody>
</table>

Figure 15. Frame execution time and space requirements for TMS 320
available instruction space and operate in real time. More changes were made and the resulting TMS 320 code appears as Annex E. Note that this does not constitute a complete tested and working program. Rather it is an optimized and detailed outline of the main part of the phase vocoder program designed to give an accurate estimate of the time and space required. These values are summarized in Figure 15.

Chapter 4 stated that since a table of stored trigonometric values was used by the FFT, the same table would be used to provide cosine and sine in the raveling and unraveling sections. The final version of the FFT developed separately from the phase vocoder did not in fact use such a table. The raveling and unraveling sections were changed in the TMS 320 version to calculate cosine and sine recursively from the relations:

\[
\cos(A+B) = \cos(A) \cos(B) - \sin(A) \sin(B) \tag{53}
\]

\[
\sin(A+B) = \sin(A) \cos(B) + \cos(A) \sin(B) \tag{54}
\]

Since values of cosine and sine at regularly spaced intervals around the unit circle are required, the interval may be interpreted as the constant B in the above equations, and thus \cos(B) and \sin(B) may be pre-calculated and stored as constants. Given \cos(A) and \sin(A), then \cos(A+B) and \sin(A+B) result by direct application of (53) and (54) and further values are found recursively.
Another improvement avoids the need for explicit evaluation of the square root in the spectrum compression section. The piece-wise magnitude approximation described in [18] is easily implemented on the TMS 320 as shown in Annex E. This takes less time than Newton's method and avoids the inaccuracies introduced by a power series expansion.

When converting from polar to rectangular coordinates after the spectrum compression, the sine and cosine of an arbitrary angle must be calculated. This is accomplished in Annex E using a recently developed method [19] well suited to integer arithmetic.

The two separate steps of post-FFT unraveling and spectrum compression cannot be combined into a single operation, since compression of the top half of the spectrum would overwrite data needed for further unraveling. In Annex E, however, the compression of the bottom half of the spectrum is carried out as harmonics are unraveled. Compression of the top half occurs in a separate step only after all harmonics have been unraveled. This saves movement of data back and forth between data RAM and program space.

At a sampling rate of 8 KHz, one frame which lasts 64 sample times, is 8.0 msec long. Figure 15 shows that even with all the optimization techniques used, the frame time on the TMS 320 is still somewhat greater than this. Figure 15 also illustrates that the maximum available program space of 4096 words is exceeded. Coupled with a shortage of data memory in some sections of Annex E, this implies that the phase vocoder as
described is too complicated for the TMS 320. Even if it could be implemented, it still wouldn't operate in real time. However, with the next generation of DSP chips (the expected TMS 32020, for example) such an implementation should be possible. With a decreased cycle time and an increase in data and program memory spaces, the FFT section would take half as long. The other sections would also be much faster, and more complicated spectrum manipulations would be possible.
Chapter 6: CONCLUSION

6.1 INTRODUCTION

Programs PHVOC7 and PV10 show that the principle of using the digital phase vocoder in time scale compression is sound. There is room, however, for improvement in the systems developed and this provides material for future research.

6.2 DISTORTION CAUSES AND SOLUTIONS

The slight distortion in the output of PHVOC7 is possibly due to the fact that the initial condition for the phase integration of each harmonic is zero. If the actual initial phase is near zero, (20) and (16) will give a good estimate of the desired modified phase. If, however, the actual initial phase is a large angle, these equations will not yield an accurate estimate, and subsequent estimates may be off as well. For a constant input signal close to a given harmonic, the phase estimated using (20) and (17) will converge to the desired phase regardless of the initial conditions. More study needs to be done on this aspect of the phase estimation for inputs such as voice signals which are not quite so well-behaved. The problem may be resolved at the
start of the algorithm by calculating the actual initial phase for each
harmonic given the real and imaginary parts. The four quadrant
arctangent function is built into FORTRAN, but is not a trivial matter
in a microprocessor.

Another possible cause of the 'rainbarrel' effect is time domain
aliasing introduced by not having a dense enough spectrum for the
inverse DFT in the synthesis section. This effect and its solutions are
discussed in [10].

The difference in performance between PHVOC7 and PV10 may be
attributed to the accumulation of errors caused by truncation of
intermediate results to 16 bit integer equivalents. One manner in which
this may be mitigated to a certain extent would be to store all spectrum
values as double precision and perform double precision arithmetic on
them. Memory space limits in current DSP microprocessors may preclude
this, however. It is anticipated that future members of the TMS 320
family will have sufficient data memory to enable this strategy to be
applied. A second solution would be to combine the post-DFT unraveling,
spectrum modification and pre-IDFT raveling into a single block. The
harmonics would be manipulated individually in double precision, instead
of in pairs, converting to single precision only for the IDFT. Such a
scheme could not be accomplished 'in place.' A third approach to
reducing the effect of roundoff errors would be to keep spectrum values
in double precision only internally to each of the unraveling,
modification and raveling sections, passing them on to the following
section in single precision. This would take up very little extra storage space but not be as effective as using double precision throughout.

6.3 OVERFLOW CAUSES AND SOLUTIONS

The problems of overflow in PV10 can be addressed in three ways. First, the amplitude of the input signal could be reduced either explicitly by attenuating the analogue input to the A/D, or implicitly by reducing the value of SCALE and thus modifying the analysis window. Either method would reduce the input by any desired fractional amount, but also reduce the signal to roundoff noise ratio.

A second method would be to clamp any 16 bit over or underflows to +32767 or -32768 as appropriate. This is easily done on the TMS 320 using the saturation mode of operation and would give only a slight distortion for occasional small overflows such as those caused by a varying voiced input. It would not be acceptable for sustained inputs which cause larger overflows. The third way of preventing overflow would be to reduce the multiplication by 16 included in the post-DFT unraveling by some multiple of two. Tests indicate that a value of 8 would probably prevent most overflows, even for sustained input sequences, and those that did occur could be clamped. Changing this scaling factor also requires a change by the same amount in the gain adjustment that follows the IDFT.
6.4 CONCLUSION

Many applications in speech processing involve the manipulation of the STFT. The phase vocoder provides a means to do this but has been avoided until now for use in real-time systems since it is computationally intensive. As shown in this thesis, it will be possible in the near future to use the phase vocoder structure in a real-time speech modification system based on a DSP microchip. The introduction of faster, more powerful chips expected will make the phase vocoder truly a cost-effective technique in many areas of real-time digital signal processing.

6.5 THE FUTURE

In any frame, three quarters of the points transformed by the forward DFT were also transformed in the previous frame. There are ways [20, 21] to exploit this overlap, perhaps to reduce computation time. The implication of this for the inverse DFT is not clear. In any case, however, standard FFT's could probably not be used.

The system is not limited to changing the pitch or time scale by a factor of two. If the unraveling and unraveling in PV10 are changed to include all harmonics, then any arbitrary spectrum modification may be accomplished in between. Benett [3, 12] describes some interesting
applications and the spectrum modifications required for each. The applications mentioned in [2] and [13] might also be well served by using the phase vocoder structure presented.
REFERENCES


-69-


-70-


[19] L. Robert Morris, personal communication

This annex contains the source listing for the program PHVOC7 as well as for all its required subroutines. PHVOC7 is the floating point version of the phase vocoder speech modification system and is described in Chapter 3.

---

PHVOC7 FOR

---

C """"""""""""""""""""
C PHVOC7 -- SHORT TERM SPECTRAL ANALYSIS, MODIFICATION AND RESTYNTHESIS
C
C THIS PROGRAM STARTS WITH A ZERO INITIAL CONDITION FOR THE PHASE UNWRAPPING INTEGRATION.
C
C LINK THIS PROGRAM WITH:
C SUBS.OBJ -- includes FILTER,FFT and SHIFTS
C ANSYN.OBJ -- includes ANLSTS and SNTH
C SPCMOD.OBJ -- spectrum compression
C
C FIXED INITIALIZATION:
C
C REAL X(512),XN(512),H(512),F(513)
C INTEGER INPUT(256),OUTPUT(256),NAME(7)
C REAL AO(512),BO(512),PHI(512)
C INTEGER R1,R2,S,T,RMAX
C COMPLEX XX(512)
C
C DATA NAME /2H ,2H ,2HSE,2HQU,2H ,2H.D,2HAT/
C
C PARAMETER MAXIMA IMPOSED BY ARRAY SIZES ABOVE:
C RMAX=512
C RMAX=256
C
C CLEAR ARRAYS:
C DO 14 J=1,512
C AO(J) = 1.0
C BO(J) = 0.0
C PHI(J) = 0.0
C X(J) = 0.0
C XN(J) = 0.0
C 14 H(J) = 0.0
C
C ---

-A1-
C VARIABLE INITIALIZATION:

C WRITE(7,99)
99  FORMAT('O', 'PHASE VOCODER SPEECH MODIFICATION, VERSION 7')
C SET UP ANALYSIS-SYNTHESIS PARAMETERS:

C WRITE(7,100)
100  FORMAT(' ',60('=',) )
WRITE(7,105)
105  FORMAT(' ', 'PARAMETER SPECIFICATION:')
12  WRITE(7,110) MMAX
110  FORMAT(' ',5X,'FFT SIZE M, MAXIMUM = ',I4)
READ(7,111) M
111  FORMAT(I3)
IF (M GT. MMAX) GO TO 12
C
13  WRITE(7,115) RMAX
115  FORMAT(' ',5X,'INPUT, OUTPUT OVERLAP PARAMETERS MAX= ',I4)
READ(7,116) R1,R2
116  FORMAT(2(I4))
IF ((R1 GT. RMAX) .OR. (R2 GT. RMAX)) GO TO 13
WRITE(7,100)
C ANALYSIS WINDOW SPECIFICATION:

C WRITE(7,120)
120  FORMAT(' ', 'ANALYSIS WINDOW:')
WRITE(7,125)
125  FORMAT(' ',5X,'0=RECTANGULAR 1=KAISER LOWPASS')
READ(7,141) IPRMPT
IF(IPRMPT .EQ. 0) GO TO 5
C KAISER ANALYSIS WINDOW GENERATION:
2  WRITE(7,130) M
130  FORMAT(' ',5X,'WINDOW LENGTH, MUST BE ODD AND LESS THAN ',I4)
READ(7,131) LENGTH
131  FORMAT(I3)
IF (LENGTH .GE. M) GO TO 2
WRITE(7,135)
135  FORMAT(' ',5X,'CUTOFF FREQUENCY, ALPHA, SCALE')
READ(7,136) FB,ALPHA,SCALE
136  FORMAT(E12.4,F6.4,F6.4)
CALL FILTER(H,LENGTH,0.0,FB,ALPHA,SCALE)
C CENTRE FILTER TO ALIGN IT WITH INPUT SEGMENT:
MOVE=(M-LENGTH-1)/2
IF(MOVE .EQ. 0) GO TO 7
DO 15 J=1,MOVE
DO 4 L=M,2,-1
4   H(L)=H(L-1)
15   H(1)=0.0
GO TO 7.
C RECTANGULAR ANALYSIS WINDOW GENERATION:
5 DO 6 J=1,M
6 H(J)=1.0
C
C SYNTHESIS WINDOW SPECIFICATION:
7 WRITE(7,140)
140 FORMAT(' ', 'SYNTHESIS WINDOW:')
WRITE(7,125)
READ(7,141)IPRMPT
141 FORMAT(I1)
IF(IPRMPT .EQ. 0) GO TO 8
C
C KAISER SYNTHESIS WINDOW GENERATION:
WRITE(7,145)M+1
145 FORMAT(' ',5X,'WINDOW LENGTH SET AT ',I4)
WRITE(7,135)
READ(7,136)FB,ALPHA,SCALE
CALL FILTER(F,M+1,0.0,FB,ALPHA,SCALE)
GO TO 10
C
C RECTANGULAR SYNTHESIS WINDOW GENERATION:
8 DO 9 J=1,M+1
9 P(J)=1.0
C
C SPECTRUM MODIFICATION:
C
10 WRITE(7,190)
190 FORMAT(' ', 'SPECTRUM MODIFICATION:')
WRITE(7,192)
192 FORMAT(' ',5X, 'MODIFY? 0=NO 1=YES')
READ(7,193)IMOD
193 FORMAT(I1)
C
WRITE(7,100)
C
INPUT/OUTPUT CONTROL:
C
20 WRITE(7,200)
200 FORMAT(' ', 'INPUT/OUTPUT:')
WRITE(7,210)
210 FORMAT(' ',5X, 'READ INPUT FROM WHAT DEVICE? ( e.g. RKO: )')
READ(7,220)NAME(1),NAME(2)
220 FORMAT(2A2)
WRITE(7,230)
230 FORMAT(' ',5X, 'File to be read is SEQUxx.DAT. What is xx?')
READ(7,220)NAME(5)
C
CALL ASSIGN(8,NAME(1),18)
C
... source (input) file name
C
READ(8)NSAMI
C
... NSAMI is number of samples in
-A3-
WRITE(7,240) NAME,NSAMI
240 FORMAT(' ',...,'TA2,'contains ',I5,'samples...')

WRITE(7,250)
250 FORMAT('0',5X,'WRITE OUTPUT TO WHAT DEVICE? (e.g. RE0: )')
READ(7,220) NAME(1),NAME(2)
WRITE(7,260)
260 FORMAT(' ',5X,'File to be written is SEQUxx.DAT. What is xx?')
READ(7,220) NAME(5)

CALL ASSIGN(9,NAME(T),14)
...destination (output) file name

NSAMO=INT((FLOAT(R2)/FLOAT(R1))\*FLOAT(NSAMI-M))\*R2
...NSAMO is number of sample out

WRITE(9) NSAMO

---

PHASE VOCODER:

T = 0
...T is the discrete time index (integer)

S = 0
...S is the frame index (integer)

11 CONTINUE

READ IN ONE BLOCK OF R1 SAMPLES:
DO 30 J=1,R1
T=T+1
INPUT(J)=0
IF(T.LE.NSAMO) READ(8) INPUT(J)
30 CONTINUE

DO ANALYSIS:
CALL ANLSYS(X,XX,H,M,R1,S,INPUT)

DO SPECTRUM MODIFICATION:
IF(IMOD .NE. 0) CALL SPCMOD(XX,A0,B0,PHI,M)

DO SYNTHESIS:
CALL SMTX(XX,XN,OUTPUT,F,M,M+1,R1,S)

SUPPRESS STARTUP TRANSIENTS:
IF(T .LE. M-R1) GO TO 997
C WRITE BLOCK OF R2 OUTPUT SAMPLES:
  DO 50 J=1,R2
  KNSAM=KNSAM+1
  IF(KNSAM .GT. NSAM0)GO TO 998
  50 WRITE(9) OUTPUT(J)
C
C PROGRESS REPORT:
  997 WRITE(7,280) KNSAM,NSAMO
  280 FORMAT( ' ', '...Written ',I5, ' / ',I5)
C
  GO TO 11
C
C=================================================================================================
C END OF PROGRAM:
C
  998 ENDFILE 8
  END FILE 9
  WRITE(7,270) NSAMO,NAME
  270 FORMAT( ' ', I5, ' samples written to ',7A2)
C
C STOP
END
C=================================================================================================
C FAST FOURIER TRANSFORM (DECIMATION IN TIME)
C
SUBROUTINE FFT(X,N,M,MODE)
C
X IS THE COMPLEX ARRAY TO BE TRANSFORMED IN PLACE
C
NUMBER OF POINTS N = 2**M
C
MODE = 1 FOR FFT
C
= -1 FOR INVERSE FFT
C
COMPLEX X(N),U,W,T
PI=3.1415926
C
C DIVIDE EACH POINT BY N IF DOING INVERSE:
C IF(MODE .EQ. 1) GO TO 5
C
  DO 4 L=1,N
  4  X(L)=X(L)/N

C BIT REVERSAL:
  5 NV2=N/2
  NM1=N-1
  J=1
  DO 30 I=1,NM1
  IF (I .GE. J) GO TO 25
  T=X(J)
  X(J)=X(I)
  X(I)=T
  25 K=NV2
  30 CONTINUE

IF (K GE J) GO TO 30
J = J - K
K = K / 2
GO TO 26

30 J = J + K

M PASSES FOR N = 2^M POINTS
DO 20 L = 1, M
LE = 2^M
LE1 = LE / 2
U = (1.0, 0.0)
W = CMPLX(COS(PI / LE1), MODE * SIN(PI / LE1))
DO 20 J = 1, LE1
DO 10 I = J, N, LE

20 U = U * W
RETURN
END

=================================================================================
C THIS SUBROUTINE GENERATES A GENERAL BANDPASS FILTER AS THE
C PRODUCT OF A DIFFERENCE OF SINC FUNCTIONS AND A KAISER
C WINDOW. THIS FOLLOW THE GENERAL TECHNIQUE OF J.F.
C KAISER'S PAPER 'NONRECURSIVE FILTER DESIGN USING THE
C IO-SINH WINDOW FUNCTION' IN PROC 1974 IEEE INT STMP ON
C CIRCUITS AND SYSTEMS, APR 22-25 1974, PP 20-23
C
SUBROUTINE FILTER(H, N, FA, FB, ALPHA, SCALE)
C
H: THE N-POINT ARRAY CONTAINING THE FILTER IMPULSE
C RESPONSE ON EXIT FROM THE SUBROUTINE
C N: NUMBER OF POINTS (ASSUMED ODD) IN IMPULSE RESPONSE
C FA: LOWER FREQUENCY OF PASSBAND NORMALIZED TO RANGE
C 0 - 0.5 WHERE SAMPLING FREQUENCY = 1.0
C (FA = 0.0 FOR LOWPASS FILTER)
C FB: UPPER FREQUENCY OF PASSBAND, AS FOR FA ABOVE
C (FB = 0.0 FOR HIGHPASS FILTER)
C ALPHA: KAISER WINDOW VARIABLE
C SCALE: MULTIPlicative SCALE FACTOR APPLIED TO ALL
C ELEMENTS OF THE IMPULSE RESPONSE
C
SUBROUTINES REQUIRED:
1. SINC(A, X) : RETURNS (SIN(A * X)) / X
2. IO(X) : MOD BESSEL PCN OF FIRST KIND, ORDER 0
C
LOCAL VARIABLES:
OFFSET: CENTRE POINT OF WINDOW/FILTER
AH: DIFFERENCE OF SINC FUNCTIONS TERM
WKAIS: KAISER WINDOW

-A6-
REAL IO, N02, H(N)

PI=3.1415926
OFFSET=(N-1)/2+1
FA2=FA*2
FB2=FB*2
N02=(OFFSET-1)#*2
DENOM=IO(ALPHA)
DO 1 J=1,N
ARG1=J-OFFSET
ARG2=PI*ARG1
AM=SINC(FB2, ARG2) - SINC(FA2, ARG2)
WKAISR=IO(ALPHA*SQRT(1.0-(ARG1)**2/N02))
WKAISR=WKAISR/DENOM
1 H(J)=AM*WKAISR*SCALE
RETURN
END

C ===================================================================================================
C THIS SUBROUTINE CALCULATES MODIFIED BESSEL
C FUNCTIONS OF THE FIRST KIND, ORDER ZERO
C
FUNCTION IO(I)

REAL IO, I01
I0=1.0
K=1
I01=(I/2)**2
1 IF ((I01/I0) .LE. 1.0E-9) GO TO 2
I0=I0+I01
K=K+1
I01=I01*(I/2/K)**2
GO TO 1
2 RETURN
END

C ===================================================================================================
C THIS FUNCTION RETURNS ( SIN(A*X) ) / X
C
FUNCTION SINC(A, X)

SINC=0.0
1 IF(A .EQ. 0.0) GO TO 1
SINC=A
IF(X .EQ. 0.0) GO TO 1
SINC=SINC(A*X)/X
RETURN
END
C ***************************************************************
C VOCODER ANALYSIS ROUTINE (VERSION 6)
C
C SUBROUTINE ANALYS(X,XX,H,M,R1,S,INPUT)
C
C X -- M-POINT REAL INPUT BUFFER
C XX -- M-POINT COMPLEX WORKING ARRAY, CONTAINS THE
C RESULT OF THE ANALYSIS
C H -- M-POINT ANALYSIS FILTER, MUST BE DEFINED BEFORE
C THIS ROUTINE IS CALLED
C M -- SIZE OF FFT, AND OTHER ARRAYS
C R1 -- INTEGER. THE ANALYSIS IS DONE IN BLOCKS OF
C OF R1 SAMPLES
C S -- S=0,1,2... THIS SHOULD BE ZERO ON FIRST CALL.
C IT IS UPDATED BY THE SYNTHESIS ROUTINE
C INPUT-- INTEGER ARRAY HOLDING THE NEXT R1 INPUT VALUES
C
C INTEGER R1,S,SR,T,INPUT(R1)
C REAL X(M), H(M)
C COMPLEX XX(M)
C
C SHIFT INPUT BUFFER LEFT BY R1 POINTS AND INPUT R1 NEW ONES:
C CALL RLSH(X,M,R1)
C DO 1 J=1,R1
C 1 X(M-R1+J)=FLOAT(INPUT(J))
C
C APPLY WINDOW AND PUT RESULT IN REAL PART OF COMPLEX ARRAY:
C DO 2 J=1,M
C 2 XX(J)=CMPLX( X(J)*H(J) , 0:0)
C
C APPLY CIRCULAR SHIFT TO XX SO THAT RESULT OF DFT WILL HAVE
C FIXED TIME REFERENCE:
C SR=(S*R1) - (S*R1/M)*H
C ISHIFT=M-(M/2+SR)
C IF(ISHIFT .LT. 0) ISHIFT=ISHIFT+M
C CALL CCLSH(XX,M,ISHIFT)
C
C DO DFT:
C MM=INT( ALOG(FLOAT(M)) / ALOG(2.0) )
C CALL FFT(XX,M,MM,1)
C
C RETURN
C END
SUBROUTINE SUMTH(XX, XM, OUTPUT, F, M, MF, R2, S)

XX -- COMPLEX M-POINT ARRAY CONTAINING PARAMETERS PASSED
     FROM ANALYSIS
XM -- M-POINT REAL OUTPUT BUFFER
OUTPUT-- R2-POINT INTEGER ARRAY CONTAINING THE R2 POINTS
     SHIFTED OUT OF XM ON EACH CALL
F -- M+1-POINT REAL ARRAY CONTAINING THE SYNTHESIS
     FILTER IMPULSE RESPONSE
M -- SIZE OF INVERSE DFT, SIZE OF OTHER ARRAYS
MF -- SIZE OF F, EQUAL TO M+1
R2 -- SYNTHESIS IS CARRIED OUT IN BLOCKS OF R2 POINTS
S -- AS FOR THE ANALYSIS ROUTINE

REAL XM(M), F(MF)
COMPLEX XX(M)
INTEGER R2, S, SR, OUTPUT(R2)

DO INVERSE FFT:
   N=INT(ALOG(FLOAT(M)) / ALOG(2.0))
   CALL FFT(XX, M, N, -1)

APPLY CIRCULAR SHIFT:
   SR=(S*R2) - (S*R2/M)*M
   ISHIFT=M/2+SR
   IF(ISHIFT .GE. M) ISHIFT=ISHIFT-M
   CALL CCLSH(XX, M, ISHIFT)
   S=S+1
   IF(S .GE. M) S=S-M

WINDOW AND OVERLAP INTO OUTPUT BUFFER:
   DO 2 J=1, M
      2    XM(J)=XM(J)+REAL(XX(J))*F(J+1)

OUTPUT R2 VALUES AND SHIFT OUTPUT BUFFER:
   DO 1 J=1, R2
      1    OUTPUT(J)=INT(XM(J))
   CALL RLSH(XM, M, R2)
RETURN
END

COMPLEX CIRCULAR LEFT-SHIFT SUBROUTINE (VERSION 7)

SUBROUTINE CCLSH(XX, M, ISHIFT)

THIS SUBROUTINE DOES A CIRCULAR LEFT SHIFT OF THE
M-POINT COMPLEX ARRAY XX. DISTANCE OF THE SHIFT
IS ISHIFT. MAXIMUM M IS EQUAL TO DIMENSION OF
TEMP
C

COMPLEX XK(M), TEMP(512)

C

IF (ISHIFT .EQ. 0) GO TO 3
DO 1 J = 1, M
INDX = J + ISHIFT
IF (INDX .GT. M) INDX = INDX - M
1 TEMP(J) = XK(INDX)
DO 2 J = 1, M
2 XK(J) = TEMP(J)
3 RETURN
END

C

=======================================================================

C REAL LEFT-SHIFT SUBROUTINE

C

SUBROUTINE RLSH(X,M,R)

C

THIS SUBROUTINE Shifts THE M-POINT ARRAY X TO THE
C LEFT (DOWNWARDS) BY R POINTS, FILLING IN THE TOP
C R ELEMENTS WITH ZEROES.

C

INTEGER R
REAL X(M)

C

DO 1 J = 1, M - R
1 X(J) = X(J + R)
DO 2 J = M - R + 1, M
2 X(J) = 0.0
RETURN
END

C

=======================================================================

C SPECTRUM COMPRESSION SUBROUTINE

C

This subroutine compresses the spectrum by a
C factor of two. It multiplies the phase of the
C harmonics by 0.5.

C

SUBROUTINE SPCMOD(XK, A0, BO, PHI, M)

C

XX -- M-POINT COMPLEX ARRAY CONTAINING
C SPECTRUM TO BE MODIFIED
C A0 -- M-POINT REAL ARRAY CONTAINING THE
C REAL PART OF THE SPECTRUM FROM THE
C PREVIOUS CALL
C BO -- M-POINT REAL ARRAY CONTAINING THE
C IMAGINARY PART OF THE SPECTRUM FROM
C THE PREVIOUS CALL
C PHI -- M-POINT REAL ARRAY CONTAINING THE
C ACCUMULATED UNWRAPPED PHASE FOR
C THE MODIFIED SPECTRUM

-A10-
COMPLEX XX(M)
REAL AO(M),BO(M),PHI(M),MAG,MAGSQR
C
BETA=0.5
TWOPI=6.2831853
C
DO 1 J=2,M/2
C
A1=REAL(XX(J))
B1=AIMAG(XX(J))
MAGSQR=A1*A1+B1*B1
MAG=SQRT(MAGSQR)
DPHI=(B1*AO(J)-A1*BO(J))/MAGSQR
DPHI=BETA*DPHI
PHI(J)=PHI(J)+DPHI
IF(PHI(J) .GT. TWOPI) PHI(J)=PHI(J)-TWOPI
IF(PHI(J) .LT.-TWOPI) PHI(J)=PHI(J)+TWOPI
C
AO(J)=A1
BO(J)=B1
C
XX(J)=CMPLX(MAG*COS(PHI(J)),MAG*SIN(PHI(J)))
XX(M+2-J)=CONJG(XX(J))
1 CONTINUE
C
M4=M/4
C
C REMAP SPECTRUM (COMPRESS BY FACTOR OF TWO):
DO 2 J=2,M4
XX(J)=XX(2*J-1)
2 XX(M+2-J)=CONJG(XX(J))
C
C ZERO THE UPPER HALF:
DO 3 J=M4+1,M4+M4
3 XX(J)=CMPLX(0.0,0.0)
C
RETURN
END
ANNEX B

PV10.FOR

This annex contains the source listing for PV10.FOR, the simulated integer implementation of the phase vocoder voice modification system. It is explained in Chapter 4. All other subroutines called by PV10 are included in this annex except for FILTER and FFT which are in Annex A.

C=================================================================
C
C PV10 PHASE VOCODER SPEECH MODIFICATION SYSTEM Ver 10
C
C LINK with FILTER.OBJ and FFT.OBJ
C
C=================================================================

C ARRAY SPECIFICATION (indices are unity based):
C
C Arrays for sake of convenience in this FORTRAN program only:
C
REAL W(256)  /* used since FILTER subroutine generates full window
INTEGER NAME(7)
DATA NAME /2H, 2H, 2HSE, 2HST, 2H, 2H,D, 2HAT/  /* used for controlling input/output files

C Arrays that must be implemented on the microprocessor:
C
REAL SHREGI(320), SHREGO(320)
...input and output shift registers
REAL Z(256)
...DFT workspace
REAL WINDOW(128)
...window coefficients
REAL A(63), B(63)
...real and imag parts of prev frame compressed spectrum
REAL PHI(63)
...accumulated phase for compressed spectrum
REAL TRIG(63)
...trig function lookup table
INTEGER IIP(4), IFTP(4), IOP(4)
...input and output pointers

C=================================================================

C CONSTANT DEFINITION:
TWO12 = 4096.0
TWO15 = 32768.0
T15M1 = TWO15 - 1.0
PI = 12868.0
C 3.1415 * 2^12
TWOPI = 2.0 * PI
C

-B1-
C
WRITE(7,100)
100 FORMAT('0',60('x'))
WRITE(7,105)
105 FORMAT('0','PHASE VOCODER (Version 10) ')
WRITE(7,100)

C
C Window specification:
C
WRITE(7,110)
110 FORMAT('0','ANALYSIS WINDOW: ')
10 WRITE(7,115)
115 FORMAT(' ',5X,'Length?')
READ(7,120)LENGTH
120 FORMAT(I4)
IF( LENGTH .GE. 256 ) GO TO 10
WRITE(7,125)
125 FORMAT('+',5X,'Normalized cutoff frequency?')
READ(7,130)FB
130 FORMAT(F14.8)
WRITE(7,135)
135 FORMAT('+',5X,'Kaiser parameter ALPHA?')
READ(7,130)ALPHA
WRITE(7,140)
140 FORMAT('+',5X,'Scale factor?')
READ(7,130)SCALE

C
C Generate window:
C CALL FILTER(W,LENGTH,0.0,FB,ALPHA,SCALE)

C
C Centre window on 256 points:
C MOVE = (255-LENGTH)/2
DO 12 J=1,MOVE
DO 14 L=256,2,-1
14 W(L) = W(L-1)
12 W(1) = 0.0

C
C Scale window coefficients and place first 129 into array that will be
C used:
DO 20 J=1,128
20 WINDOW(J) = AINT( W(J) * T15M1 )
C INPUT AND OUTPUT FILE CONTROL:
C
WRITE(7,150)
150 FORMA'T('0', 'INPUT/OUTPUT:')
WRITE(7,155)
155 FORMAT(' ',5X,'Read input from what device? ( e.g. RKO: )')
READ(7,160) NAME(1), NAME(2)
160 FORMAT(2A2)
WRITE(7,165).
165 FORMAT('+',5X,'Read from file SEQUxx.DAT. What is xx?')
READ(7,160) NAME(5)
CALL ASSIGN(8,NAME(1),14)
READ(8) NSAMi
WRITE(7,1170) NAME, NSAMi
170 FORMAT('+',10X,'...;',7A2,' contains ',I5,' samples...')
WRITE(7,175)
175 FORMAT('0',5X,'Write output to what device? ( e.g. RKO: )')
READ(7,160) NAME(1), NAME(2)
WRITE(7,180)
180 FORMAT('+',5X,'Write to file SEQUxx.DAT. What is xx?')
READ(7,160) NAME(5)
CALL ASSIGN(9,NAME(1),14)
NSAM0=NSAMi
WRITE(9) NSAM0
INSAM = 0
INSTM = 0

C TRIG FUNCTION LOOKUP TABLE CALCULATION:
C
DO 25 J=1,63
25 TRIG(J) = AIMT( 15M1 * COS(J*0.0245437) )
       ...cos(J*2pi/256) scaled by 2**15-1

C POINTER INITIALIZATION:
C
IIP(n) points to the start of the input shift register section
that will be windowed by block n of the window.
IIP(1) = 1
   ...and 65,129,193,257,1... in successive frames
IIP(2) = 65
   ...and 129,193,257,1,65...
IIP(3) = 129
   ...and 193,257,1,65,129...
IIP(4) = 193
   ...and 257,1,65,129,193...

IPIM points to the start of the input shift register sect that
will receive the input samples.
IPIM = 257
   ...and 1,65,129,193,257... in successive frames
C IFTP(n) points to the start of the FFT workspace sect that will
receive the result of windowing with block n of window.
IFTP(1) = 65
...and 129,193,1,65... in successive frames
IFTP(2) = 129
...and 193,1,65,129...
IFTP(3) = 193
...and 1,65,129,193...
IFTP(4) = 1
...and 65,129,193,1...

C IOP(n) points to the start of the output shift register section
that will be overlap-added by block n of FFT workspace.
IOP(1) = 193
...and 129,65,1,257,193... not nec changing each frame
IOP(2) = 257
...and 193,129,65,1,257...
IOP(3) = 65
...and 1,257,193,129,65...
IOP(4) = 129
...and 65,1,257,193,129...

C IOUTP points to the start of the output shift register section
that will provide the output
IOUTP = 1
...and 65,129,193,257,1... in successive frames

C ZERO SHIFT REGISTERS:
DO 27 J=1,320
SHREGI(J) = 0.0
27 SHREGO(J) = 0.0

C ZERO A,B, AND PHI:
DO 29 J=1,63
A(J) = TM012
B(J) = 0.0
29 PHI(J) = 0.0

C START OF VOCODER LOOP (repeated once per frame)

C 30 CONTINUE
C ...loop back to this point
C INTERRUPT SERVICE ROUTINE:
C
C This section simulates 64 interrupts. At each interrupt, once
C sample is read into the input shift register and one is written
C out of the output shift register. Once over 64 interrupts, (once
C per frame), the pointers are updated.
C
C Get 64 samples into input shift register segment:
DO 32 J=0,63
ITEMP = 0
INSAM = INSAM + 1
IF( INSAM .LE. NSAMI ) READ(8) ITEMPIF( INSAM .LE. NSAMI ) READ(8) ITEMPIF( INSAM .LE. NSAMI ) READ(8) ITEMPIF( INSAM .LE. NSAMI ) READ(8) ITEMPIF( INSAM .LE. NSAMI ) READ(8) ITEMP
SHREGI(IPIN+J) = 64.0 * FLOAT(ITEMP)
SHREGI(IPIN+J) = 64.0 * FLOAT(ITEMP)
SHREGI(IPIN+J) = 64.0 * FLOAT(ITEMP)
SHREGI(IPIN+J) = 64.0 * FLOAT(ITEMP)
SHREGI(IPIN+J) = 64.0 * FLOAT(ITEMP)
Note that since a 9+1 bit A/D is assumed, the input
Note that since a 9+1 bit A/D is assumed, the input
Note that since a 9+1 bit A/D is assumed, the input
Note that since a 9+1 bit A/D is assumed, the input
Note that since a 9+1 bit A/D is assumed, the input
samples are multiplied by 64 to left justify them in
samples are multiplied by 64 to left justify them in
samples are multiplied by 64 to left justify them in
samples are multiplied by 64 to left justify them in
samples are multiplied by 64 to left justify them in
a 16 bit word.

C Write 64 samples from output shift register segment:
DO 34 J=0,63
KNSAM = KNSAM + 1
KNSAM = KNSAM + 1
KNSAM = KNSAM + 1
KNSAM = KNSAM + 1
KNSAM = KNSAM + 1
IF( KNSAM .GT. NSAMO ) GO TO 998
IF( KNSAM .GT. NSAMO ) GO TO 998
IF( KNSAM .GT. NSAMO ) GO TO 998
IF( KNSAM .GT. NSAMO ) GO TO 998
IF( KNSAM .GT. NSAMO ) GO TO 998

34 WRITE(9) INT( SHREGO(IOOTP+J) )
WRITE(9) INT( SHREGO(IOOTP+J) )
WRITE(9) INT( SHREGO(IOOTP+J) )
WRITE(9) INT( SHREGO(IOOTP+J) )
WRITE(9) INT( SHREGO(IOOTP+J) )

C Update pointers:
IPIN = IPIN + 64
IPIN = IPIN + 64
IPIN = IPIN + 64
IPIN = IPIN + 64
IPIN = IPIN + 64
IF( IPIN .GT. 257 ) IPIN = IPIN - 320
IF( IPIN .GT. 257 ) IPIN = IPIN - 320
IF( IPIN .GT. 257 ) IPIN = IPIN - 320
IF( IPIN .GT. 257 ) IPIN = IPIN - 320
IF( IPIN .GT. 257 ) IPIN = IPIN - 320

IOOTP = IOOTP + 64
IOOTP = IOOTP + 64
IOOTP = IOOTP + 64
IOOTP = IOOTP + 64
IOOTP = IOOTP + 64
IF( IOOTP .GT. 257 ) IOOTP = IOOTP - 320
IF( IOOTP .GT. 257 ) IOOTP = IOOTP - 320
IF( IOOTP .GT. 257 ) IOOTP = IOOTP - 320
IF( IOOTP .GT. 257 ) IOOTP = IOOTP - 320
IF( IOOTP .GT. 257 ) IOOTP = IOOTP - 320

DO 36 J=1,4
IIP(J) = IIP(J) + 64
IIP(J) = IIP(J) + 64
IIP(J) = IIP(J) + 64
IIP(J) = IIP(J) + 64
IIP(J) = IIP(J) + 64
IF( IIP(J) .GT. 257 ) IIP(J) = IIP(J) - 320
IF( IIP(J) .GT. 257 ) IIP(J) = IIP(J) - 320
IF( IIP(J) .GT. 257 ) IIP(J) = IIP(J) - 320
IF( IIP(J) .GT. 257 ) IIP(J) = IIP(J) - 320
IF( IIP(J) .GT. 257 ) IIP(J) = IIP(J) - 320

IFTP(J) = IFTP(J) + 64
IFTP(J) = IFTP(J) + 64
IFTP(J) = IFTP(J) + 64
IFTP(J) = IFTP(J) + 64
IFTP(J) = IFTP(J) + 64
IF( IFTP(J) .GT. 193 ) IFTP(J) = IFTP(J) - 256
IF( IFTP(J) .GT. 193 ) IFTP(J) = IFTP(J) - 256
IF( IFTP(J) .GT. 193 ) IFTP(J) = IFTP(J) - 256
IF( IFTP(J) .GT. 193 ) IFTP(J) = IFTP(J) - 256
IF( IFTP(J) .GT. 193 ) IFTP(J) = IFTP(J) - 256

C Progress report:
WRITE(7,185) KNSAM,NSAMO
WRITE(7,185) KNSAM,NSAMO
WRITE(7,185) KNSAM,NSAMO
WRITE(7,185) KNSAM,NSAMO
WRITE(7,185) KNSAM,NSAMO
185 FORMAT('*','...written ',I5,' / ',I5)
185 FORMAT('*','...written ',I5,' / ',I5)
185 FORMAT('*','...written ',I5,' / ',I5)
185 FORMAT('*','...written ',I5,' / ',I5)
185 FORMAT('*','...written ',I5,' / ',I5)

C ------- end of ISR------------------------
WINDOW SAMPLES AND PUT THEM IN THE FFT WORKSPACE:

Do blocks 1 and 4 which are multiplied by symmetric portions of the window:

\[
\begin{align*}
Z(\text{IFTP}(1)) &= 0.0 \\
Z(\text{IFTP}(4)+63) &= \text{AINT}( \text{SHREGI}(\text{IIP}(4)+63) \times \text{WINDOW}(1) ) / \text{TWO15} \\
\text{DO 40 J=1,63} \\
Z(\text{IFTP}(1)+J) &= \text{AINT}( \text{SHREGI}(\text{IIP}(1)+J) \times \text{WINDOW}(J) ) / \text{TWO15} \\
\text{40} \\
Z(\text{IFTP}(4)+J-1) &= \text{AINT}( \text{SHREGI}(\text{IIP}(4)+J-1) \times \text{WINDOW}(65-J) ) / \text{TWO15} \\
\end{align*}
\]

Do blocks 2 and 3 which are multiplied by symmetric portions of the window:

\[
\begin{align*}
\text{DO 42 J=0,63} \\
Z(\text{IFTP}(2)+J) &= \text{AINT}( \text{SHREGI}(\text{IIP}(2)+J) \times \text{WINDOW}(J+64) ) / \text{TWO15} \\
\text{42} \\
Z(\text{IFTP}(3)+J) &= \text{AINT}( \text{SHREGI}(\text{IIP}(3)+J) \times \text{WINDOW}(128-J) ) / \text{TWO15} \\
\end{align*}
\]

***TEST POINT 1:

CALL OFLOW(Z,256,1,256,1,1)

---------------------------

DO 128 POINT COMPLEX FFT, USING EVEN AND ODD SAMPLES AS REAL AND IMAG PARTS.

CALL TMSPFFT(Z)

Post DFT unravelling:

Since a 128 point complex FFT is used to calculate the DFT of 256 real points, unravelling of the resulting spectrum is required. At the input to the FFT all points were scaled by \(2^{815}\) but underwent a division by 128 internal to the FFT to prevent overflow. Thus at the entry to this unravelling section, points are scaled by \(2^{88}(15-7) = 2^{88}\). During the unravelling process, all points are multiplied by \(2^{88}\) so that at the output, the effective scaling factor is \(2^{8812}\). All functions of angles are scaled by \(2^{8815}\).

Note that only every second harmonic is unravelled since the rest will be lost anyway in the spectrum compression that follows.

Special cases for harmonics 0 and 64 (Re:Im X=1:2 and X=129:130)

Should divide by 2, but will multiply by 8 instead, effecting the increase in scaling factor by \(2^{84}\).

\[
\begin{align*}
Z(1) &= 8.0 \times (Z(1) + Z(2)) \\
Z(2) &= 0.0 \\
Z(129) &= 8.0 \times Z(129) \\
Z(130) &= 8.0 \times Z(130) \\
\end{align*}
\]
Unravel every second harmonic in pairs: 2,126,4,124 ... 62,66
(Re:Im K= 5:6,253:254 9:10,2:9:2:250 ... 125:126,133:134)
The order in which the unravelling is done is irrelevant.

DO 45 K=5,125,4
   V1 = TRIG( 6A-(K-1)/2 )
   V2 = TRIG( (K-1)/2 )
   ...((K-1)/2 = 2,4,6 ... 62
   V3 = Z(258-K) + Z(K)
   V4 = Z(259-K) + Z(K+1)
   V5 = Z(258-K) - Z(K)
   V6 = Z(259-K) - Z(K+1)
   ...V3,V4,V5,V6 still scaled by 2**8
   V7 = AINT( V1*V5/TW012 )
   V8 = AINT( V2*V4/TW012 )
   V9 = AINT( V1*V4/TW012 )
   V10= AINT( V2*V5/TW012 )
   ...V7,V8,V9,V10 now scaled by 2**8(8+15-12) = 2**11
   V11 = V7 + V8
   V12 = V10- V9
   ...V11,V12 are also scaled by 2**11
   In the next section, should divide by 4. Instead will divide only
   by 2, effectively multiplying by 2. The end result is overall
   scaling by 2**12.
   Z(K) = AINT( (V3*8.0 + V11)/2.0 )
   Z(K+1)= AINT( (V12 - V6*8.0)/2.0 )
   Z(258-K) = AINT( (V3*8.0 - V11)/2.0 )
   Z(259-K) = AINT( (V12 + V6*8.0)/2.0 )
45 CONTINUE

***TEST POINT 2:
   CALL OFLOW(Z,256,5,125,4,2)
   CALL OFLOW(Z,256,6,126,4,2)

SPECTRUM COMPRESSION AND PHASE MULTIPLICATION:
   On entry to this section, all Z values are scaled by 2**12.
   Angles and functions of angles are scaled by 2**12.
   Compress harmonics 2,4,6...126 into 1,2,3...63
   (Re:Im K=5:6, 9:10 ... 253:254 into 3:4, 5:6 ... 127:128)
   Harmonic 0 is the DC component and is not affected.
DO 50 K=3,127,2
   K= 3,5,7,...127
   A1 = Z( 2^K-1 )
   B1 = Z( 2^K )
   A0 = A( (K-1)/2 )
   B0 = B( (K-1)/2 )
   ...real part of same harmonic from last frame
   ...imaginary part of same harmonic from last frame
   ...((K-1)/2 = 1,2,3,...63

   ...32 bit accumulation
   IF( V12 .EQ. 0 ) V12 = 1.0
   ...to avoid zero divided by zero later
   V13 = TMSSRT(V12)
   ...TMSSRT is square root routine
   V14 = ALN( B1*A0 - A1*B0 )
   ...32 bit accumulation
   V15 = TMSSDIV( V14,V12 )
   ...V15 = TO012 * (V14/V12)
   V16 = ALN(V15/2.0)
   ...phase multiplication by 0.5
   V17 = V16 + PHI( (K-1)/2 )
   ...phase accumulation (integration)
   IF( V17 .LT.-PI ) V17 = V17 + TOPI
   IF( V17 .GT. PI ) V17 = V17 - TOPI
   ...principal value of phase
   V18 = ALN( V13 * TMSCOS(V17) / TO012 )
   ...V18 = V13 * COS(V17/TO012)
   V19 = ALN( V13 * TMSSIN(V17) / TO012 )
   ...V19 = V13 * SIN(V17/TO012)

Z(K) = V18
Z(K+1) = V19
   ...modified spectrum (real,imag)

PHI( (K-1)/2 ) = V17
A( (K-1)/2 ) = A1
B( (K-1)/2 ) = B1
   ...for next frame

50 CONTINUE

***TEST POINT 3:
   CALL OPLOM(Z,256,1,128,1,3)

-Z8-
INVERSE FFT:

Pre-IDFT unraveling:

Since a 128 point complex FFT is used to calculate the inverse DFT to obtain a 256 point real output, unraveling of the spectrum is required before the FFT. Inputs to the FFT are replaced by their complex conjugates as are the outputs (in the overlap add section) thus effecting the inverse DFT.

On entry, all Z values are scaled by $2^{12}$.
Functions of angles are scaled by $2^{15}$.

Special case for harmonics 0 and 64 (Re:Im K=1:2 and 129:130):

$Z(1) = Z(1)$
$Z(2) = -Z(2)$
$Z(129) = 0.0$
$Z(130) = 0.0$

Ravel for harmonics in pairs 1,127, 2,126,... 63,65
(Re:Im K=3:8,255:256 5:6,253:254 ... 127:128,131:132)

DO 55 K=3,127,2

$V_{20} = \text{TRIG}(64-(K-1)/2)$
$V_{21} = \text{TRIG}( (K-1)/2 )$

\( \ldots (K-1)/2 = 1,2,3,\ldots,63 \)

$V_{22} = \text{AINT}(Z(K) \times V_{20} / \text{TW015})$
$V_{23} = \text{AINT}(Z(K+1) \times V_{20} / \text{TW015})$
$V_{24} = \text{AINT}(Z(K) \times V_{21} / \text{TW015})$
$V_{25} = \text{AINT}(Z(K+1) \times V_{21} / \text{TW015})$

$V_{26} = V_{22} - V_{25}$
$V_{27} = V_{23} + V_{24}$

$Z(K) = \text{AINT}( (Z(K) + V_{26}) / 2.0 )$
$Z(K+1) = \text{AINT}( (-Z(K+1) - V_{27}) / 2.0 )$
$Z(258-K) = \text{AINT}( (Z(K) - V_{26}) / 2.0 )$
$Z(259-K) = \text{AINT}( (Z(K+1) - V_{27}) / 2.0 )$

55 CONTINUE

TEST POINT 4:

CALL OFLOW(Z,256,1,256,1,4)

(INVERSE) FFT:

CALL TMSFFT(Z)
C ADJUST GAIN:
C
Since a 10 bit A/D was assumed, input samples were multiplied by 64. In the FFT, they were divided by 128. In the post-FFT unravelling, they were further multiplied by 16. The overall effect of this felt at the output of the inverse DFT is a magnification by 8. Therefore before going to a 10 bit D/A, the output samples are all divided by 8.
C
DO 57 J=1,256
57 Z(J) = ALNT( Z(J) / 8.0 )
C
C OVERLAP ADD INTO OUTPUT SHIFT REGISTER:
C
Blocks 1, 2 and 3 of the FFT workspace are added to shift register sections indicated by IOP(1), IOP(2) and IOP(3). Block 4 of the FFT workspace is written over the section indicated by IOP(4) rather than being added. The effect of this is the same as shifting zeroes into that section of the output shift register and overlapping the fourth section from the FFT workspace.
C
Every second point is subtracted rather than being added. This is necessary since the forward FFT program is used to effect the inverse FFT. The input was conjugated, so then must be the output, since every second point may be considered to be the imaginary part of the 128 point FFT.
C
DO 60 J=0,62,2
C
SHREGO( IOP(1)+J ) = SHREGO( IOP(1)+J ) + Z( J+1 )
SHREGO( IOP(1)+J+1 ) = SHREGO( IOP(1)+J+1 ) - Z( J+2 )
C
SHREGO( IOP(2)+J ) = SHREGO( IOP(2)+J ) + Z( J+65 )
SHREGO( IOP(2)+J+1 ) = SHREGO( IOP(2)+J+1 ) - Z( J+66 )
C
SHREGO( IOP(3)+J ) = SHREGO( IOP(3)+J ) + Z( J+129 )
SHREGO( IOP(3)+J+1 ) = SHREGO( IOP(3)+J+1 ) - Z( J+130 )
C
SHREGO( IOP(4)+J ) = Z( J+193 )
SHREGO( IOP(4)+J+1 ) = - Z( J+194 )
C
60 CONTINUE
C
GO TO 30
C
END OF VOCODER LOOP (repeated once per frame)
C
998  ENUDFILE 9
    WRITE(7,190) NSAMO,NAMK
190  FORMAT('0',15,' samples written to ',7A2)
    STOP
    END

C ================================================================

FUNCTION TMSJOS(X)

This function subroutine expects as an argument, an angle in
radians which has been multiplied by 4096. It returns the
sine of that angle, also multiplied by 4096.

TWO12 = 4096.0
TMSJOS = AJNT( TWO12 * SIN( X/TWO12 ) )
RETURN
END

C ================================================================

FUNCTION TMSJOS(X)

This function has the same input and output scaling factors as
TMSJOS but returns the cosine of the angle rather than the cosine.

TWO12 = 4096.0
TMSJOS = AJNT( TWO12 * COS( X/TWO12 ) )
RETURN
END

C ================================================================

SUBROUTINE OFLOW(X,N,J,K,L,IPT)

This subroutine checks the N point real array X for possible over-
flows where floating point is used to simulate a 16 bit machine.
It checks every Lth element from the Jth to the Kth. IPT is the
test point at which this routine is called.

REAL X(N)
RMAX = 0.0
RMIN = 0.0
DO 1 M=J,K
IF( X(M) .GT. RMAX ) RMAX = X(M)
IF( X(M) .LT. RMIN ) RMIN = X(M)
1   CONTINUE
IF( RMIN .LT. -32768.0 ) GO TO 2
IF( RMAX .LT. 32767.0 ) GO TO 9
2   WRITE(7,900) IPT,RMIN,RMAX
990  FORMAT(' +25X,'"UNDER/OVERFLOW AT POINT",I3,2(F10.1) )
9   RETURN
    END

-B11-
C ***********************************************************************
C
C SUBROUTINE TMSFFT(X)
C
C This subroutine simulates the DSPS 128 point complex FFT for the
C TMS320 microprocessor. At the input, all points are divided by
C 128 to prevent internal overflow.
C
C REAL X(256)
C COMPLEX ZZZ(128)
C DO 1 J=1,128
C RE = AINT( X(2*J-1) / 128.0 )
C IM= AINT( X(2*J ) / 128.0 )
C ZZZ(J) = CMPLX( RE,IM )
C 1
C CALL FFT(ZZZ,128,7,1)
C
C DO 2 J=1,128
C X( 2*J-1 ) = AINT( REAL( ZZZ(J) ) )
C X( 2*J ) = AINT( AIMAG( ZZZ(J) ) )
C 2
C RETURN
C END
C
C ***********************************************************************
C
C FUNCTION TMSDIV(X,Y)
C
C This function subroutine returns \( 4096 \times \frac{X}{Y} \).
C
C TMSDIV = AINT( 4096.0 \times (X/Y) )
C RETURN
C END
C
C ***********************************************************************
C
C FUNCTION TMSSRT(X)
C
C This function returns the integer square root of its argument.
C
C TMSSRT = AINT( SQRT(X) )
C RETURN
C END
ANNEX C

SUPPORT PROGRAMS

This annex contains the source listings of three programs used in testing the phase vocoder speech modification system. These programs are:

1. WOEE.FOR -- generates phoneme sequences and stores them on disk

2. SEQSAY.FOR -- plays phoneme sequences through the D/A

3. DECIM8.FOR -- downsamples a phoneme sequence by an integer factor

C ==============================================================
C W O E E E                                               Timothy C. Green
C
C GENERAL:
C
C WOEE is a vowel phoneme generator that can be used to create integer time sequences for use in testing voice DSP algorithms. A series of three phonemes at specified pitches are generated and the program's output consists of a time sequence that glides at a uniform rate from the first to the second phoneme/pitch and then to the third. The number of samples generated is always less than MAXSAM (set at 18000) and varies depending on the values specified for the pitch.

C ASSEMBLING AND LINKING:
C
C The main program is WOEE.FOR and it must be compiled and linked with assembled versions of GENSAM.MAC and TALK.MAC. The following sequence does this:

.R FORTRA
*WOEE=WOEE C
.* R MACRO
*GENSAM=GENSAM
*TALK=TALK C
.*
.* R LINK
*WOEE=WOEE/F,GENSAM,TALK
"C
.*RUN WOEE (to run the program)

C
The program is computation intensive and the use of floating point hardware is highly recommended.

-C1-
The program asks first for the phoneme glide sequence. This is entered as three phoneme numbers 1-10 corresponding to the formant frequencies given in the FREQ data statements below. Example: 1,3,1 <Return>

The program then requires the pitch period sequence in milliseconds. Limits are checked to ensure the pitch period lies between PITMIN and PITMAX. Example: 2.0,5.3,2.5 <Return>

The program then calculates the samples, displaying its progress through the algorithm. A self-explanatory menu then appears.

DISK FILE FORMAT:

When samples are written to disk, the resulting file is NSAM+1 integers long, written in unformatted binary form. The first integer is NSAM itself, which indicates the number of integers that follow.

VARIABLES:

FREQ - array containing the formant freqs in Hz for all the phonemes
F - array containing the actual formant freqs used to generate the samples
PIT - array containing the pitch period values
OUTPUT - array containing the calculated time samples
IGL - array containing the phoneme numbers input
NSAM - actual number of samples generated
INDX - index of array OUTPUT at which successive blocks of samples are stored by GENSAM

SUBROUTINES CALLED:

COEF1 - the computational kernel of the program, calculates the appropriate coefficients, included with WOeree.FOR
GENSAM - generates the samples and places them in array OUTPUT
TALK - sends array OUTPUT to D/A one at a time

REAL FREQ(12),FREQ(10,3),F(3,3),PIT(3),SUMRY(3,5)
INTEGER NP(10),OUTPUT(18000),NC(13),IGL(3),ISUMRY(3),NAME(7)
COMMON FREQ,GAIN,IMULT,LRSS
COMMON /XX/NC

-C2-
DATA FREQ(1,1), FREQ(1,2), FREQ(1,3)/270, 2290, 3010/
DATA FREQ(2,1), FREQ(2,2), FREQ(2,3)/309, 1990, 2550/
DATA FREQ(3,1), FREQ(3,2), FREQ(3,3)/530, 1840, 2480/
DATA FREQ(4,1), FREQ(4,2), FREQ(4,3)/660, 1720, 2410/
DATA FREQ(5,1), FREQ(5,2), FREQ(5,3)/640, 1190, 2390/
DATA FREQ(6,1), FREQ(6,2), FREQ(6,3)/730, 1090, 2440/
DATA FREQ(7,1), FREQ(7,2), FREQ(7,3)/570, 840, 2410/
DATA FREQ(8,1), FREQ(8,2), FREQ(8,3)/440, 1020, 2240/
DATA FREQ(9,1), FREQ(9,2), FREQ(9,3)/300, 870, 2240/
DATA FREQ(10,1), FREQ(10,2), FREQ(10,3)/490, 1350, 1690/

C
DATA NAME /2H ,2H ,2HSE ,2HQU ,2H ,2H.D ,2HAT/
C
C MAXIMA:
  IGLMAX=10
  MAXSAM=18000
  PITMIN=1.0
  PITMAX=10.0
  IGMAX=2
C
C
WRITE(7,500)
500 FORMAT('0', '*************** W O W E E ***************')
C
JUNIT=8
C
C INITIALIZATION:
100 MSAM=0
INDEX=1
C
5 WRITE(7,503)MAXSAM=1
503 FORMAT('0', 'MAXIMUM NUMBER OF SAMPLES TO GENERATE? ( <', 'I5, ')')
READ(7,504)MAXTMP
504 FORMAT(15)
IF(MAXTMP.GT. MAXSAM) GO TO 5
MAXSAM=MAXTMP
C
WRITE(7,505)
505 FORMAT('0', 'PHONEME GLIDE SEQUENCE? ( e.g. 1,4,1 )')
READ(7,510) IGL(1), IGL(2), IGL(3)
510 FORMAT(3(I2))
C
C INPUT ERROR TRAPPING:
DO 10 J=1,3
  IGL(J)=IABS(IGL(J))
  IF((IGL(J).LT.IGL(J)) .OR. (IGL(J).EQ.0)) GO TO 100
  ISUM=I=1,3
15 SUM=10*FREQ(IGL(J),K)
10 CONTINUE
105 WRITE(7,515)
515 FORMAT(' ', 'PITCH PERIOD SEQUENCE? (e.g. 2.0, 5.3, 2.0)')
READ(7,520)PIT(1),PIT(2),PIT(3)
520 FORMAT(3(F4.1))
C
C INPUT ERROR TRAPPING:
   DO 25 J=1,3
   PIT(J)=ABS(PIT(J))
   IF ( (PIT(J) .LT. PITMIN) .OR. (PIT(J) .GT. PITMAX) ) GO TO 105
   SUMRT(J,5)=PIT(J)
   PIT(J)=11.765*PIT(J)
25 CONTINUE
C
   WRITE(7,521)
521 FORMAT(' ')
C
C LOOP ONCE FOR EACH GLIDE SEQUENCE:
   DO 80 IG=1,2
C
   NPASS=MAXSAM/(PIT(1)+PIT(2))
   PITINC=(PIT(2)-PIT(1))/(NPASS-1)
C
C SET UP GLIDE SEQUENCE FREQUENCIES:
   DO 20 J=1,2
   DO 20 K=1,3
20   F(J,K)=FPREC(IGL(J),K)
   DO 30 J=1,3
30   F(3,J)=(F(2,J)-F(1,J))/NPASS
C
C SAMPLE GENERATION:
   DO 40 IPASS=1,NPSS
   WRITE(7,522)IG,IGMAX,IPASS,NPSS
522 FORMAT(' ',5X,'GLIDE: ',I1,' / ',I1,5X,'PASS: ',I3,' / ',I3)
C
   CALL CORRF1( F(1,1), F(1,2), F(1,3) )
C
   NGAIN=GAIN
   DO 50 NN=1,10
50   NP(NN)=PRED(NN)*MULT
C
C INPUT=INT(PIT(1))
   DO 60 NS=1,10
60   NC(NS)=NP(NS)
   NC(11)=INPUT
   NC(12)=NGAIN
   NC(13)=RSS
C
C PUT SAMPLES INTO ARRAY:
   CALL GENSAM( OUTPUT(INDX) )
C
-C4-
C PREPARE FOR NEXT ROUND OF SAMPLE GENERATION:
    INDEX = INDEX + INPUT
    NSAM = NSAM + INPUT - 1
    PIT(1) = PIT(1) + PITINC
    DO 70 J = 1, 3
    70 F(1, J) = F(1, J) + F(3, J)
C
    CONTINUE
C
C PREPARE FOR NEXT GLIDE:
    PIT(1) = PIT(2)
    PIT(2) = PIT(3)
    IGL(1) = IGL(2)
    IGL(2) = IGL(3)
C
    CONTINUE
C END OF SAMPLE GENERATION
C
C
130 WRITE(7, 525) NSAM
525 FORMAT('0', IT, ' SAMPLES GENERATED')
    WRITE(7, 527)
527 FORMAT('0', '==================================')
110 WRITE(7, 530)
530 FORMAT('0', 5X, '1. OUTPUT TO D/A')
    WRITE(7, 535)
535 FORMAT(' ', 5X, '2. WRITE TO DISK')
    WRITE(7, 540)
540 FORMAT(' ', 5X, '3. SUMMARY TO SCREEN')
    WRITE(7, 545)
545 FORMAT(' ', 5X, '4. CALCULATE NEW SEQUENCE')
    WRITE(7, 555)
555 FORMAT('0', 5X, '9. EXIT PROGRAM')
    READ(7, 510) IWHAT
    GO TO (120, 125, 135, 100, 110, 110, 110, 110, 999), IWHAT
    GO TO 110
C
C
C OUTPUT TO D/A:
C
120 WRITE(7, 560)
560 FORMAT('0', 'DO HOW MANY REPETITIONS?')
    READ(7, 510) IREP
    DO 90 J = 1, IREP
    90 CALL TALK( OUTPUT(1), NSAM )
    GO TO 110
C WRITE TO DISK:
C
125 WRITE(7,580)
580 FORMAT( 'O', 'DISK DEVICE NAME? ( e.g. DI1: )')
READ(7,585)NAME(1),NAME(2)
585 FORMAT(2A2)
WRITE(7,590)
590 FORMAT( ' ', 'File name will be SEQUxx.DAT . What is xx?')
READ(7,585)NAME(5)
C
CALL ASSIGN(IUNIT,NAME(1),1)
C
WRITE(7,600)NSAM,NAME
600 FORMAT( 'O', '...Writing ',I5,' samples to ',7A2)
WRITE(IUNIT,ERR=97)NSAM
DO 85 J=1,NSAM
ITEMP=OUTPUT(J)-512
85 WRITE(IUNIT,ERR=97)ITEMP
C
ENDFILE IUNIT
C
IUNIT=IUNIT+1
C
GO TO 110
C
C WRITE ERROR:
97 WRITE(7,700)
700 FORMAT( 'O', '*** WRITE ERROR. DISK FULL? ***')
GO TO 110
C
C PRINT SUMMARY:
C
135 WRITE(7,565)
565 FORMAT( 'O', '================== SUMMARY : ===============')
WRITE(7,570)
570 FORMAT( ' ', 'Phoneme #',7X,'Formant Freqs(Hz)',7X,'Pitch pd(ms)')
DO 95 J=1,3
95 WRITE(7,575)ISUMRY(J),SUMRY(J,M),M=1,5
575 FORMAT( ' ',3X,12,5X,4(F5.0,3X),3X,F4.1)
GO TO 130
C
C EXIT PROGRAM:
C
999 STOP
END
SUBROUTINE COEFF1(FF1, FF2, FF3)

REAL COEFF(4,3), ALPHA(4), F(4), PRED(12), XPCH(100)
REAL C(13), CT(13)
COMMON PRED, GAIN, XMULT, LRSS

C INITIALIZATION:
F(1)=FF1
F(2)=FF2
F(3)=FF3
F(4)=3500.
TT=1./8000.
GAIN=1.
ALPHA(1)=30.
ALPHA(2)=50.
ALPHA(3)=60.
ALPHA(4)=87.5
PI=3.1415926

DO 1 J=1,11
   C(J)=0.0
   CT(J)=0.0

1  CONTINUE

DO 2 NFORM=1,4
   COEFF(NFORM,1)=1.0
   X=-2.*EXP(-2.*PI*ALPHA(NFORM)*TT)
   COEFF(NFORM,2)=X*COS(2.*PI*F(NFORM)*TT)
   COEFF(NFORM,3)=EXP(-4.*PI*ALPHA(NFORM)*TT)

2  CONTINUE

CT(1)=1.
CT(2)=EXP(-400.*PI*TT)+EXP(-5000.*PI*TT)
CT(3)=EXP(-400.*PI*TT)*EXP(-5000.*PI*TT)
NPOWER=2
NPW1=NPW+1

DO 5 NFORM=1,4

5  CONTINUE

DO 6 K=1,NPW1
   DO 7 NCOEF=1,3
      NDEG=NCOEF+K-2
      NSLOT=NDEG+1
      X=C(NSLOT)

7   CONTINUE

   C(NSLOT)=X*COEFF(NFORM,NCOEF)*CT(K)
6   CONTINUE

   NPOWER=NPW+2
   NPW1=NPW+1

7   CONTINUE

DO 8 J=1,NPW1
   CT(J)=C(J)
8   CONTINUE

C(J)=0.0
5 CONTINUE

-C7-
C
C
DO 10 K=1,10
10 PRED(K) = CT(K+1)
C
XPCH(1) = 1.
NPITCH = 80
C
DO 11 N=2,NPITCH
M=N
IF(N .LE. 10) M=N-1
XPCH(N) = 0.0
DO 12 K=1,M
WX=N-K
XPCH(N) = XPCH(N) + PRED(K) * XPCH(WX)
12 CONTINUE
11 CONTINUE
C
RMS = 0.0
C
DO 13 K=1,NPITCH
RMS = RMS + XPCH(K)**2
13 CONTINUE
C
GAIN = 800. * SQRT(1./RMS)
YMAX = ABS(PRED(1))
DO 14 K=2,10
14 IF(ABS(PRED(K)) .GT. YMAX) YMAX = ABS(PRED(K))
DO 15 H=1,4
M=M+1
THRESH = 2.**H
IF(YMAX .GE. THRESH) GO TO 15
M=MULT = 2.**((15-M)
LRSS=M+1
GO TO 99
15 CONTINUE
99 CONTINUE
RETURN
END
SUBROUTINE GEMSAM (called by WOEVER):

.GLOBAL GEMSAM
.CSECT XX

;THIS IS A SUBROUTINE TO GENERATE A SERIES OF
;SAMPLES AND PLACE THEM IN AN ARRAY MAM. IT
;IS PATTERNED AFTER REALMA.MAC

;FORTRAN CALL:   CALL GEMSAM( MAM(N) )

; WHERE MAM(N) IS THE ELEMENT WHERE THE FIRST
; SAMPLE WILL GO.

NC=. ;COMMON SECTION
.CSECT

.IRFP I,0123456
R'X=I'I
.ENDM
SP=%6
PC=%7 ;REGISTER NAMES

GENSAM:   MOV   2(R5),R3
       MOV   %NC,R4
       MOV   20.(R4),PITCH ;PITCH
       MOV   22.(R4),R2 ;IMPULSE
       MOV   24.(R4),LSS ;SHIFT FACTOR
       MOV   %SAMPL,R1
       MOV   %NC,R0

LOOP2:   .MACRO MULT
       MOV   (R1)+,R4
       MUL   (R0)+,R4
       ADD   R4,R2
       .ENDM

       MULT
       MULT
       MULT
       MULT
       MULT
       MULT
       MULT
       MULT
       MULT
       MULT
       MULT

-C9-
ADD  #1000,R2
MOV  R2,(R3)+ ;PLACE IN ARRAY
SUB  #1000,R2
ASH  LRSS,R2
SUB  #22.,R1
MOV  R2,(R1)
SUB  #20.,R0
CLR  R2
DEC  PITCH
BNE  LOOP2
;
RTS  PC ;**EXIT

LRSS: 0
PITCH: 0
=*500.
SAMPL: 0,0,0,0,0,0,0,0
;
.END

SUBROUTINE TALK (called by WOEEE and SEQSAV):
;
.TITLE TALK
.GLOBAL TALK
;
.IRPC I, 012345 ;REGISTER NAMES
R'S=143'I
.EDM
SP=*6
PC=*7
;
DA=*170412 ;D/A ADDRESSES AND VECTORS
CLKV1=*344
CLKV2=*346
CLKSR=*170404
CLKBR=*170406
CLK=*177546
;
TALK: MOV 2(R5),R0
MOV @4(R5),R1
MOV #SERV,#CLKV1
MOV #340,#CLKV2
MOV 0-125,#CLKSR
MOV #503,#CLKSR
WAIT: TST  R1
BNE  WAIT CLR  @CLKSR
RTS  PC
;
SERV: MOV  (R0)+,#DA ;SERVICE ROUTINE TO SEND SAMPLE TO D/A
DEC  R1
RTI
;
.END

-C10-
C # *******************************  
C        SEQSA Y             Timothy C. Green  
C  SEQSA Y is a program that sends a phoneme sequence created by  
C WO WEE to the D/A. To run the program, SEQSA Y.FOR must first  
C compiled and linked with the assembled version of TALK.MAC in  
C the following manner:  
C  
C .R FORTRA 
C *SEQSA Y=SEQSA Y 
C *C 
C .R MACRO 
C *TAL K=TALK 
C *C 
C .R LI NE 
C *SEQSA Y=SEQSA Y/F,TALK 
C *C 
C .RUN SEQSA Y          (to run the program)  
C  
C SEQSA Y reads the sequence from a disk file created by WO WEE.  
C  
C *****************************  
C  
C INTEGER NAME(7),OUTPUT(18000)  
C  
C DATA NAME /2H ,2H ,2HSE,2HQU,2H ,2H.D,2HAT/  
C  
C IUNIT=8  
C  
C WRITE(7,100)  
C 100 FORMAT('0','*************** SEQSA Y ******************')  
C 30 WRITE(7,110)  
C 110 FORMAT('0','SEQUENCE TO BE READ FROM WHAT DEVICE? (e.g. DI1:):')  
C READ(7,120)NAME(1),NAME(2)  
C 120 FORMAT(2A2)  
C WRITE(7,130)  
C 130 FORMAT('0','File to be read is SEQUxx.DAT. What is xx?')  
C READ(7,120)NAME(5)  
C  
C CALL ASSIGN(IUNIT,NAME(1),14)  
C  
C READ(IUNIT)NSAM  
C  
C WRITE(7,140)NSAM,NAME  
C 140 FORMAT('0','...Reading ',IS,' samples from ',7A2)  
C  
C DO 10 J=1,NSAM  
C READ(IUNIT)OUTPUT(J)  
C 10 OUTPUT(J)=OUTPUT(J)+512  
C  
C ENDFILE IUNIT
C IUNIT=IUNIT+1
C
40  WRITE(7,150)
150  FORMAT( '0',5X,'1. OUTPUT TO D/A')
    WRITE(7,160)
160  FORMAT( ',5X,'2. GET ANOTHER SEQUENCE')
    WRITE(7,170)
170  FORMAT( ',5X,'3. NAME OF LAST FILE READ')
    WRITE(7,200)
200  FORMAT( '0',5X,'9. EXIT PROGRAM')
C
READ(7,210)IWHAT
210  FORMAT(I1)
    GO TO(20,30,70,40,40,40,40,40,40,40),IWHAT
    GO TO 40
C
C OUTPUT TO D/A:
C
20  WRITE(7,220)
220  FORMAT( '0', 'HOW MANY REPETITIONS?')
    READ(7,230)NREP
230  FORMAT(I2)
    WRITE(7,250)
250  FORMAT( ', ' 'START AT SAMPLE NUMBER n. n=?')
    READ(7,260)ISTART
260  FORMAT(15)
    IF( (ISTART.LT.1). OR. (ISTART.GT.NSAM) )ISTART=1
    DO 60 J=1,NREP
60  CALL TALK( OUTPUT(ISTART),NSAM+1-ISTART )
    GO TO 40
C
C LAST FILE READ:
C
70  WRITE(7,240)NAME
240  FORMAT( '0', 'Last file read was ',7A2)
    GO TO 40
C
C EXIT PROGRAM:
50  STOP
END
C ************************************************************************
C
C          D E C I M 8
C
C THIS PROGRAM READS IN ONE INTEGER FILE AND WRITES EVERY N'TH
C INTEGER TO THE SPECIFIED OUTPUT FILE.  THE FACTOR N (this is
called INTFAC in the program) MUST BE AN INTEGER.  THIS
C PROGRAM IS TYPICALLY USED TO DECIMATE A SERIES OF SAMPLES
C BY AN INTEGER AMOUNT.
C
C THE INPUT AND OUTPUT FILES ARE FORMATTED AS FOR THE PROGRAM
C WORKE.FOR.  THAT IS, THE FIRST INTEGER IN THE FILE INDICATES THE NUMBER OF SAMPLES THAT FOLLOW.
C
C SUBROUTINES REQUIRED:  NONE
C
C
INTEGER NAME(7)
DATA NAME /2H",12H,2HSE,2HQU,2H",2H.D,2HAT/

WRITE(7,100)
100 FORMAT('0','D E C I M 8 -- DECIMATION BY AN INTEGER FACTOR')
WRITE(7,105)
105 FORMAT(' ',60('=',"'))
WRITE(7,110)
110 FORMAT('0','WHAT IS DECIMATION FACTOR?')
READ(7,115)INTFAC
115 FORMAT(12)
WRITE(7,120)
120 FORMAT(' ', 'READ FROM WHAT DEVICE? (e.g. RKO:)' )
READ(7,125)NAME(1),NAME(2)
125 FORMAT(2A2)
WRITE(7,130)
130 FORMAT(' ', 'FILE TO BE READ IS SEQUxx.DAT. WHAT IS xx?')
READ(7,125)NAME(5)

CALL ASSIGN(8,NAME(1),14)

READ(8)NIN
WRITE(7,135)
135 FORMAT(' ', 'WRITE TO WHAT DEVICE? (e.g. RKO:)' )
READ(7,125)NAME(1),NAME(2)
WRITE(7,140)
140 FORMAT(' ', 'FILE TO BE WRITTEN IS SEQUxx.DAT. WHAT IS xx?')
READ(7,125)NAME(5)

OUT=NIN/INTFAC
WRITE(7,145)NIN,OUT
145 FORMAT(' ',I5,' samples in, ',I5,' samples out...')
CALL ASSIGN(9,NAME(1),14)

WRITE(9)NOUT

DO 10 J=1,NOUT
    DO 20 K=1,INTFAC
        READ(8)ITEMP
        WRITE(9)ITEMP
    20
10

ENDFILE 8
ENDFILE 9

STOP
END
ANNEX D

PV12.FOR

This annex contains the source listing for PV12.FOR, the simulated integer implementation of the phase vocoder voice modification system adapted to the structure of the TMS 320. It is explained in Chapter 5. It uses the same subroutines as PV10 with the exception of WFT which is included in this annex.

C=====================================================================
C P V 1 2 PHASE VOCODER SPEECH MODIFICATION SYSTEM Ver 12
C LINK with FILTER.OBJ and FFT.OBJ
C Differences from PV12: window increased to 129 points
C windowing technique
C input shift register indexing technique
C=====================================================================
C ARRAY SPECIFICATION (indices are unity based):
C Arrays for sake of convenience in this FORTRAN program only:
C
C REAL W(256)
C ...used since FILTER subroutine generates full window
C INTEGER NAME(7)
C DATA NAME /2H ,2H ,2RSE, 2BQU, 2H ,2H.D, 2HAT/
C ...used for controlling input/output files
C
C Arrays that must be implemented on the microprocessor:
C
C REAL SHREGI(320), SHREGO(320)
C ...input and output shift registers
C REAL Z(256)
C ...DFT workspace
C REAL WINDOW(129)
C ...window coefficients
C REAL A(63), B(63)
C ...real and imag parts of prev frame compressed spectrum
C REAL PHI(63)
C ...accumulated phase for compressed spectrum
C REAL TRIG(63)
C ...trig function lookup table
C INTEGER IIP(4), IMP(4), IOP(4), IX
C ...input and output pointers
C
-D1-
C CONSTANT DEFINITION:
    TWO12 = 4096.0
    TWO15 = 32768.0
    T15M1 = TWO15 - 1.0
    PI = 12868.0
    TWOPI = 2.0 * PI
    IWMAX = 129

C -----------------------------------------------
    WRITE(7,100)
    100 FORMAT('0',60('=', ));
    WRITE(7,105)
    105 FORMAT('0',PHASE VOCODER (Version 12) ')
    WRITE(7,100)

C C WINDOW:
C
C Window specification:
    WRITE(7,110)
    110 FORMAT('0','ANALYSIS WINDOW:
    10 WRITE(7,115)
    115 FORMAT(' ','5X','Length?
    READ(7,120)LENGTH
    120 FORMAT(I4)
    IF( LENGTH .GE. 256 ) GO TO 10
    WRITE(7,125)
    125 FORMAT('+','5X','Normalized cutoff frequency?
    READ(7,130)FB
    130 FORMAT(F14.8)
    WRITE(7,135)
    135 FORMAT('+','5X','Kaiser parameter ALPHA?
    READ(7,130)ALPHA
    140 FORMAT('+','5X','Scale factor?
    READ(7,130)SCALE

C C Generate window:
    CALL FILTER(W,LENGTH,0.0,FB,ALPHA,SCALE)

C C Centre window on 256 points:
    MOVE = (255-LENGTH)/2
    DO 12 J=1,MOVE
        DO 14 L=256,2,-1
            14 W(L) = W(L-1)
    12 W(1) = 0.0

C C Scale window coefficients and place first 129 into array that will be
C used:
    WINDOW(1) = 0.0
    DO 20 J=1,128
        20 WINDOW(J+1) = AINT( W(J) * T15M1 )

C -----------------------------------------------
INPUT AND OUTPUT FILE CONTROL:

WRITE(7,150)
WRITE(7,155)
FORMAT(')',X,5X,'Read input from what device? ( e.g. RKO: )'
READ(7,160) NAME(1), NAME(2)
WRITE(7,165)
FORMAT('+',5X,'Read from file SEQUxx.DAT. What is xx ?'
READ(7,160) NAME(5)
CALL ASSIGN(8,NAME(1),14)
READ(8) NSAM1
WRITE(7,170) NAME, NSAM1
FORMAT('+',10X,'...,'TA2,' contains ',IS,,' samples...')
WRITE(7,175)
FORMAT(')',5X,'Write output to what device? ( e.g. RKO: )'
READ(7,160) NAME(1), NAME(2)
WRITE(7,180)
FORMAT('+',5X,'Write to file SEQUxx.DAT. What is xx ?'
READ(7,160) NAME(5)
CALL ASSIGN(9,NAME(1),14)
NSAMO=NSAM1
WRITE(9) NSAMO
INSAM = 0
KNSAM = 0

TRIG FUNCTION LOOKUP TABLE CALCULATION:

DO 25 J=1,63
TRIG(J) = AMT( T15M1 * COS(J*0.0245437) )
...cos(J*2pi/256) scaled by 2**15-1

POINTER INITIALIZATION:

IIP(n) points to the start of the input shift register section
that will go to the nth block of the FFT workspace after
being windowed.

IIP(1) = 129
...and 2(129), 4(65), 4(1), 4(257), 4(193), 4(129)...
IIP(2) = 193
...and 3(193), 4(129), 4(65), 4(1), 4(257), 4(193)...
IIP(3) = 1
...and 4(257), 4(193), 4(129), 4(65), 4(1)...
IIP(4) = 65
...and 65, 4(1), 4(257), 4(193), 4(129), 4(65)...
IIP points to the start of the input shift register sect that
will receive the input samples.
IIPN = 257
...and 1,65,129,193,257... in successive frames.
IOP(n) points to the start of the output shift register section that will be overlap-added by block n of FFT workspace.

IOP(1) = 193
...and 129,65,1,257,193... not nec. changing each frame

IOP(2) = 257
...and 193,129,65,1,257...

IOP(3) = 65
...and 1,257,193,129,65...

IOP(4) = 129
...and 65,1,257,193,129...

IOUTP points to the start of the output shift register section that will provide the output

IOUTP = 1
...and 65,129,193,257,1... in successive frames

IX is a pointer to a pointer. It indicates which of IIP() and IOP() should be changed at each frame.
IX = 3
...and 4,1,2,3,4... in successive frames

ZERO SHIFT REGISTERS:
DO 27 J=1,320
SHREGI(J) = 0.0
27 SHREGO(J) = 0.0

ZERO A, B, AND PHI:
DO 29 J=1,63
A(J) = 0.0
B(J) = 0.0
29 PHI(J) = 0.0
START OF VOCODER LOOP (repeated once per frame)

30 CONTINUE

...loop back to this point

INTERRUPT SERVICE ROUTINE:

This section simulates 64 interrupts. At each interrupt, once
sample is read into the input shift register and one is written
out of the output shift register. Once over 64 interrupts, (once
per frame), the pointers are updated.

Get 64 samples into input shift register segment:

DO 32 J=0,63
   ITEMP = 0
   INSAM = INSAM + 1
   IF( INSAM .LE. NSAMI ) READ(8) ITEMP
   SHREG(IPIN+J) = 64.0 * FLOAT(ITEMP)
   NOTE THAT SINCE A 9+1 BIT A/D IS ASSUMED, THE INPUT
   SAMPLES ARE MULTIPLIED BY 64 TO LEFT JUSTIFY THEM IN
   A 16 BIT WORD.

Write 64 samples from output shift register segment:

DO 32 J=0,63
   KNSAM = KNSAM + 1
   IF( KNSAM .GT. NSAMO ) GO TO 998
   WRITE(9) INT( SHREGO(IOUTF+J) )

Update pointers:

   IPIN = IPIN + 64
   IF( IPIN .GT. 257 ) IPIN = IPIN - 320

   IOUTF = IOUTF + 64
   IF( IOUTF .GT. 257 ) IOUTF = IOUTF - 320

   IIP(IX) = IIP(IX) - 64
   IF( IIP(IX) .LE. 0 ) IIP(IX) = IIP(IX) + 320
   IOP(IX) = IOP(IX) - 64
   IF( IOP(IX) .LE. 0 ) IOP(IX) = IOP(IX) + 320

DO 36 J=1,4
   IWP(J) = IWP(J) - 64
   IF( IWP(J) .LE. 0 ) IWP(J) = IWP(J) + 192
36 CONTINUE

IX = IX + 1
IF( IX .GT. 4 ) IX = IX - 4

Progress report:

WRITE(7,185) KNSAM,NSAMO
185 FORMAT( '...written ',3I5, ' / ',5I5)

-D5-
C WINDOW SAMPLES AND PUT THEM IN THE FFT WORKSPACE:
C
C BLOCK 1:
IZ = 1
CALL WFT(Z,IZ,WINDOW,IFP(1),SHREGI,IIP(1),IWMAX)
C BLOCK 2:
CALL WFT(Z,IZ,WINDOW,IFP(2),SHREGI,IIP(2),IWMAX)
C BLOCK 3:
CALL WFT(Z,IZ,WINDOW,IFP(3),SHREGI,IIP(3),IWMAX)
C BLOCK 4:
CALL WFT(Z,IZ,WINDOW,IFP(4),SHREGI,IIP(4),IWMAX)
C
C ***TEST POINT 1:
CALL OFLOW(Z,256,1,256,1,1)
C
C DO 128 POINT COMPLEX FFT, USING EVEN AND ODD SAMPLES AS REAL AND IMAG
C PARTS.
C
CALL TMSPFT(Z)
C
C Post DFT unravelling:
C
Since a 128 point complex FFT is used to calculate the DFT of 256
real points, unravelling of the resulting spectrum is required.
At the input to the FFT all points were scaled by 2**15 but under-
went a division by 128 internal to the FFT to prevent overflow.
Thus at the entry to this unravelling section, points are scaled
by 2**(15-7) = 2**8. During the unravelling process, all points
are multiplied by 2**8 so that at the output, the effective scaling
factor is 2**12. All functions of angles are scaled by 2**15.
C
Note that only every second harmonic is unravelled since the rest
will be lost anyway in the spectrum compression that follows..
C
Special cases for harmonics 0 and 64 (Re:Im K=1:2 and K=129:130)
C
Should divide by 2, but will multiply by 8 instead, effecting
increase in scaling factor by 2**4.
Z(1) = 8.0 * ( Z(1) + Z(2) )
Z(2) = 0.0
Z(129) = 8.0 * Z(129)
Z(130) = 8.0 * Z(130)
C
C Unravel every second harmonic in pairs: 2,126 4,124 ... 62,66
C (Re:Im K=5:6,253:254 9:10,249:250 ... 125:126,133:134)
C
The order in which the unraveling is done is irrelevant.
C
DO 45 K=5,125,4
C
V1 = - TRIG( 64*K- (K-1)/2 )
V2 = TRIG( (K-1)/2 )
C ...((K-1)/2 = 2,4,6 ... 62

-D6-
V3 = Z(258-K) + Z(K)
V4 = Z(259-K) + Z(K+1)
V5 = Z(258-K) - Z(K)
V6 = Z(259-K) - Z(K+1)

...V3,V4,V5,V6 still scaled by 2**8

V7 = AINT( V1*V5/TW012 )
V8 = AINT( V2*V4/TW012 )
V9 = AINT( V1*V4/TW012 )
V10= AINT( V2*V5/TW012 )

...V7,V8,V9,V10 now scaled by 2**(8+15-12) = 2**11

V11 = V7 + V8
V12 = V10 - V9

...V11,V12 are also scaled by 2**11

In the next section, should divide by 4. Instead will divide only by 2, effectively multiplying by 2. The end result is overall scaling by 2**12.

Z(K) = AINT( (V3*8.0 + V11)/2.0 )
Z(K+1)= AINT( (V12 - V6*8.0)/2.0 )

Z(258-K) = AINT( (V3*8.0 - V11)/2.0 )
Z(259-K) = AINT( (V12 + V6*8.0)/2.0 )

45 CONTINUE

***TEST POINT 2:
CALL OFLOW(Z,256,5,125,4,2)
CALL OFLOW(Z,256,6,126,4,2)

SPECTRUM COMPRESSION AND PHASE MULTIPLICATION:

On entry to this section, all Z values are scaled by 2**12. Angles and functions of angles are scaled by 2**12.

Compress harmonics 2,4,6...126 into 1,2,3..63
(Re:Im K=5:6, 9:10 ... 253:254 into 3:4, 5:6 ... 127:128)
Harmonic 0 is the DC component and is not affected.

DO 50 K=3,127,2
   ... K= 3,5,7,9...127
   A1 = Z( 2**K-1)
   ... 2**K-1= 5,9,13,17...253 (real part of harmonic)
   B1 = Z( 2**K )
   ... 2**K= 6,10,14,18...254 (imaginary part of harmonic)
   A0 = A( (K-1)/2 )
   ...real part of same harmonic from last frame
   B0 = B( (K-1)/2 )
   ...imaginary part of same harmonic from last frame
   ...(K-1)/2 = 1,2,3...63

-D7-
C
    ...32 bit accumulation
IF( V12 .EQ. 0) V12 = 1.0
    ...to avoid zero divided by zero later
V13 = TMSSRT(V12)
    ...TMSSRT is square root routine
V14 = A*M( B1*A0 - A1*B0)
    ...32 bit accumulation
V15 = TMSDIV(V14, V12)
    ...V15 = TWO12 * (V14/V12)
V16 = A*M(V15/2.0)
    ...phase multiplication by 0.5
V17 = V16 + PHI( (K-1)/2)
    ...phase accumulation (integration)
IF( V17 .LT.-PI ) V17 = V17 + TWOPI
IF( V17 .GT. PI ) V17 = V17 - TWOPI
    ...principal value of phase
V18 = A*M( V13 * TMSCOS(V17) / TWO12)
    ...V18 = V13 * COS(V17/TWO12)
V19 = A*M( V13 * TMSSIN(V17) / TWO12)
    ...V19 = V13 * SIN(V17/TWO12)
Z(K) = V18
Z(K+1) = V19
    ...modified spectrum (real, imag)
PHI( (K-1)/2 ) = V17
A( (K-1)/2 ) = A1
B( (K-1)/2 ) = B1
    ...for next frame
50 CONTINUE
C
***TEST POINT 3:
   CALL OFLOW(Z,256,1,128,1,3)
C
-----------------------------
C
C INVERSE FFT:
C
C Pre-IDFT ravelling:
C
Since a 128 point complex FFT is used to calculate the inverse
C DFT to obtain a 256 point real output, ravelling of the spectrum
C is required before the FFT. Inputs to the FFT are replaced by
C their complex conjugates as are the outputs (in the overlap add
C section) thus affecting the inverse DFT.
C
On entry, all Z values are scaled by 2**12.
C
Functions of angles are scaled by 2**15.

-D8-
C Special case for harmonics 0 and 64 (Re:Im K=1:2 and 129:130):
Z(1) = Z(1)
Z(2) = -Z(2)
Z(129) = 0.0
Z(130) = 0.0

C Ravel for harmonics in pairs 1,127 2,126 ... 63,65
C
DO 55 K=3,127,2
C
V20 = TRIG( 64-(K-1)/2 )
V21 = TRIG( (K-1)/2 )
C ...(K-1)/2 = 1,2,3 ... 63
C
V22 = AINT( Z(K) * V20 / TWO15 )
V23 = AINT( Z(K+1) * V20 / TWO15 )
V24 = AINT( Z(K) * V21 / TWO15 )
V25 = AINT( Z(K+1) * V21 / TWO15 )
C
V26 = V22 - V25
V27 = V23 + V24
C
Z(K) = AINT( ( Z(K) + V26 ) / 2.0 )
Z(K+1) = AINT( ( -Z(K+1) - V27 ) / 2.0 )
Z(258-K) = AINT( ( Z(K) - V26 ) / 2.0 )
Z(259-K) = AINT( ( Z(K+1) - V27 ) / 2.0 )
C
55 CONTINUE
C
C ***TEST POINT 4:
CALL OFLOW(Z,256,1,256,1,4)
C
C (INVERSE) FFT:
C
CALL TMSFPT(Z)
C
C ADJUST GAIN:
C
Since a 10 bit A/D was assumed, input samples were multiplied by
64. In the FFT, they were divided by 128. In the post-FFT unravelling, they were further multiplied by 16. The overall ef-
cfect of this felt at the output of the inverse DFT is a multi-
cation by $8$. Therefore before going to a 10 bit D/A, the output
samples are all divided by $8$.
C
DO 57 J=1,256
57 Z(J) = AINT( Z(J) / 8.0 )
OVERLAP ADD INTO OUTPUT SHIFT REGISTER:

Blocks 1, 2 and 3 of the FFT workspace are added to shift register sections indicated by IOP(1), IOP(2) and IOP(3). Block 4 of the FFT workspace is written over the section indicated by IOP(4) rather than being added. The effect of this is the same as shifting zeroes into that section of the output shift register and overlapping the fourth section from the FFT workspace.

Every second point is subtracted rather than being added. This is necessary since the forward FFT program is used to effect the inverse FFT. The input was conjugated, so then must be the output, since every second point may be considered to be the imaginary part of the 128 point FFT.

```
DO 60 J=0,62,2
  SHREGO( IOP(1)+J  ) = SHREGO( IOP(1)+J  ) + Z( J+1 )
  SHREGO( IOP(1)+J+1 ) = SHREGO( IOP(1)+J+1 ) - Z( J+2 )

  SHREGO( IOP(2)+J  ) = SHREGO( IOP(2)+J  ) + Z( J+65 )
  SHREGO( IOP(2)+J+1 ) = SHREGO( IOP(2)+J+1 ) - Z( J+66 )

  SHREGO( IOP(3)+J  ) = SHREGO( IOP(3)+J  ) + Z( J+129 )
  SHREGO( IOP(3)+J+1 ) = SHREGO( IOP(3)+J+1 ) - Z( J+130 )

  SHREGO( IOP(4)+J  ) = Z( J+193 )
  SHREGO( IOP(4)+J+1 ) = - Z( J+194 )
```

60 CONTINUE

GO TO 30

END OF VOCODER LOOP (repeated once per frame)

ENDFILE 9
WRITE(7,190) NSAMO,NAM
190 FORMAT('0',15,' samples written to ',7A2)
STOP
END
SUBROUTINE WFT(Z,IZ,W,IW,SHREG,IIP,IWMAX)

This function does the actual windowing of the input points and transfers them to the FFT workspace.

Z - FFT workspace array
IZ - location in Z where first of 64 points is to be placed
W - window
IW - points to first logical point in window to use
SHREG - input shift register
IIP - input pointer
IWMAX - maximum value that real (as opposed to logical) window pointers can have

REAL Z(256), W(129), SHREG(320)

TWO15 = 32768.0
INDX = IW
INCR = 1
IF( (IW-IWMAX) .LE. 0 ) GO TO 1
INDX = IWMAX + 128 - IW +1
INCR = -1
CONTINUE

DO 2 J=0,63
  Z(IZ+J) = AIMT( W(INDX) * SHREG(IIP+J) / TWO15 )
  INDX = INDX + INCR
2

IZ = IZ + 64

RETURN

END
ANNEX E

TMS 320 PROGRAM LISTINGS

This annex contains sections of TMS 320 source code that when integrated would implement the phase vocoder algorithms discussed in the main part of the thesis. The sections in themselves do not form a working program, but are used to estimate the time and space requirements for such a program. See comments on this program in Chapter 5.

;*******************************************************************************
;
; PHASE VOCODER:
;
; Use of program RAM for data:
;
; SHREGI 320  input shift register
; SHREGO 320  output shift register
; A  63  RE part of harmonics in previous frame
; B  63  IM part of harmonics in previous frame
; PHI  63  phase accumulated up to previous frame
; Z  256  FFT workspace
; WINDOW 128  also used by FFT
; TRIG  129  analysis window
; IIP1...IIP4 4  input pointers
; IOF1...IOF4 4  output pointers
; IX  1  pointer to pointer that needs to be updated
; misc  22  other constants and pointers
;
; ---
; 1414 TOTAL
;
;*******************************************************************************

; INTERRUPT SERVICE ROUTINE VERSION 13

; This routine handles the input of samples to the input shift register
; and the output from the output shift register.

; DRAM LOCATIONS, PAGE 0:
; not used
; DRAM LOCATIONS, PAGE 1:
; locations for
; SAVST  processor status storage
; SAVAH  ACC high 16 bits storage
; SAVA  ACC low 16 bits storage
; TIPIN  IPIN input pointer storage, points to location where
;        last input sample went in array SHREGI
; TIOUT  IOUT output pointer storage, points to location where
;        last output sample came from in array SHREGO

-81-
; TEMP temporary storage
; NDONE reset to -63 at start of every frame and incremented
; once at each interrupt. A value of zero here indicates
; to the main program that 64 samples have been input and
; output since start of last frame.

;CONSTANT DEFINITION:
PORTI = 0 ; input port
PORTO = 0 ; output port

;ISR:
SST SAVST ; save processor status (automatically p. 1)
LDPK 1 ; use page 1 for others
SACH SAVAH
SACL SAVAL ; save accumulator

LACK 1
ADD TIPIN ; point to PRAM loc for this input sample
IN TEMP,PORTI ; TEMP <- input sample
TBLW TEMP ; SHREG(TIPIN) <- input sample
SACL TIPIN ; for next time

LACK 1
ADD TIOUT ; point to PRAM loc to get this output sample
TBLR TEMP ; TEMP <- SHREG(TIOUT)
OUT TEMP,PORTO ; output <- TEMP
SACL IOUT ; for next time

LACK 1
ADD NDONE ; increment towards 0
SACL NDONE

ZALH SAVAH ; restore accumulator
ADD SAVAL
LST SAVST ; restore processor status (LDPK 0 implicit)
EINT ; ready for next interrupt
RET ; back to main program

;=============================================================================

;ISR SUMMARY:
; PROGRAM WORDS: 22 words
; FRAME TIME: 1856 cycles (64 ints X 29 cycles/int)
; FRAME TIME 0.371 usec

;=============================================================================

-E2-
; WINDOW VERSION 13:

; NOTE:
; This section windows the input points, taking into account the relative
; movements of the window and shift register with respect to the FFT
; workspace. The top half of the result is stored in the top half of the
; FFT workspace in PRAM. The bottom half is passed to the FFT stored in
; page 0 of DRAM.

; DRAM LOCATIONS, PAGE 0:
; locations 0-127 used to pass half the windowed FFT coeffe to the FFT

; DRAM LOCATIONS, PAGE 1:
; locations for:
; ONE assume pre-initialized to 1
; IIPN holds IIPn, n=1,2,3 or 4
; IWPN holds IWPn, n=1,2,3 or 4
; PTRSTO holds pointer to pointers IIP and IWP in PRAM
; FPTPTR points to next point in FFT workspace to be written
; TEMP
; TEMP2
; WINCR set equal to +/-1 to control movement along window
; WBASE lowest address of window in PRAM
; IBASE lowest address of SHRREGI in PRAM
; IX pointer to input pointer to update

;-------------------------------------------------------------------------------

; LDPK 1 ; use page 1 for all temporary data

; Read in constants and pointers:
LACKA WPTR
TBLR FPTPTR ; FPTPTR <- address of last pt in FFT workspace
ADD ONE
TBLR PTRSTO ; PTRSTO <- address of IIP4 in PRAM
ADD ONE
TBLR WBASE
ADD ONE
TBLR IBASE
ADD ONE
TBLR IX

; Window 64 points: SHRREGI(IIP4-J) * WINDOW(IWP4-J) for J=0,1,2...63

; LACK PTRSTO ; Acc points to IIP4 (which is followed by IWP4,
; IIP3, IWP3, IIP2, IWP2, IIP1, IWP1)
TBLR IIPN ; IIPN <- IIP4
ADD ONE ; Acc points to IWP4
TBLR IWPN ; IWPN <- IWP4
SACL PTRSTO ; store the pointer to pointers

CALL WSETUP ; get ready for windowing, update IWP4
CALL WWDW1 ; window 64 points and write them to FFT
; workspace in PRAM

- E3 -
; Window 64 points: SHREGI(IIP3-J) • WINDOW(IWP3-J) for J=0,1,2...63

LAC PTRSTO ;Acc points to IWP4 from previous block
ADD ONE ;Acc now points to IIP3
TBLR IIPN ;IIPN <- IIP3
ADD ONE ;Acc now points to IWP3
TBLR IWPN ;IWPN <- IWP3
SACL PTRSTO ;store the pointer to pointers

CALL WSETUP ;get ready for windowing, update IWP3
CALL WNDW1 ;window 64 points and write them to FFT workspace in PRAM

; Window 64 points: SHREGI(IIP2-J) • WINDOW(IWP2-J) for J=0,1,2...63

LARK AR1,127 ;leave windowed samples in DRAM starting at loc 127

LAC PTRSTO ;Acc points to IWP3 from previous block
ADD ONE ;Acc now points to IIP2
TBLR IIPN ;IIPN <- IIP2
ADD ONE ;Acc now points to IWP2
TBLR IWPN ;IWPN <- IWP2
SACL PTRSTO ;store the pointer to pointers

CALL WSETUP ;get ready for windowing, update IWP2
CALL WNDW2 ;window 64 points but leave them in page 0 of DRAM

; Window 64 points: SHREGI(IIP1-J) • WINDOW(IWP1-J) for J=0,1,2...63

LAC PTRSTO ;Acc points to IWP2 from previous block
ADD ONE ;Acc now points to IIP1
TBLR IIPN ;IIPN <- IIP1
ADD ONE ;Acc now points to IWP1
TBLR IWPN ;IWPN <- IWP1
SUB ONE ;
SACL PTRSTO ;PTRSTO points to back to IIP1

CALL WSETUP ;get ready for windowing, update IWP1
CALL WNDW2 ;window 64 points but leave them in page 0 of DRAM

; Update one of IIP1, IIP2, IIP3 or IIP4 as indicated by IX:
; (IX is equal to 0,1,2 or 3)

LAC PTRSTO ;Acc points to IIP1
SUB IX,S2 ;Acc now points to which IIP to update
SACL PTRSTO ;store the pointer to pointer
TBLR TEMP ;TEMP <- pointer to update
LAC TEMP ;Acc <- pointer to update

-84-
SUB ONE,S6  ;pointer is updated (-64), but must do limit
SUB IBASE  ;IBASE is minimum addr of SHREGI
BGZ UPDT  ;branch if updated pointer was in fact within
           ;limits
ADD ONE,S8
ADD ONE,S6  ;add 320
UPDT:
ADD IBASE
SACL TEMP
LAC PTRSTO
TBLW TEMP  ;write updated pointer back to PRAM

; Update IX:

LACK 3  ;mask
SACL TEMP
LAC IX
ADD ONE  ;update pointer
AND IX  ; modulo 4
SACL IX
LAC TEMP2  ;Acc <- PRAM loc of IX
TBLW IX  ;write IX back to PRAM

;----END OF WINDOW SECTION----

---------------------------------------------------------------------

Subroutine WSETUP

This subroutine sets up the pointers so that windowing can take place.
Since only the lower half of the window is stored, sections that
logically require the top half must use the reflected half.
Pointer IVPN is updated.

WSETUP: LAC ONE
         SACL WINCR  ;normally will step backwards through window
                     ;(SUB ONE)
         LAC IVPN  ;Acc <- IVPN (points to last point in the
                     ;LOGICAL section of the window
         SUB WBASE  ;WBASE is the addr of the first point of the
                     ;window in PRAM
         SACL TEMP  ;TEMP <- IVPN - WBASE
         SUB ONE,S6  ;Acc <- IVPN - WBASE - 64 (pointer updated)
         BGE UPDT2  ;branch if no need for modulo 256 correction
         ADD ONE,S8  ;add 256

UPDT2: ADD WBASE  ;restore to actual address
         SACL TEMP2
         LAC PTRSTO  ;Acc <- location of IWP to update
         TBLW TEMP2  ;write IWP back to PRAM
         LAC TEMP  ;Acc <- IVPN - WBASE
         SUB ONE,S7  ;Acc <- IVPN - WBASE - 128
         BLZ FWDHO  ;branch if logical portion of window corresponds
                     ;to portion of window actually in memory
         SUB ONE,S7  ;Acc <- IVPN - WBASE - 256
         SUB ONE,S2  ;Acc <- IVPN - WBASE - 256 - 2

-25-
SA CL TEMP
ZAC TEMP ;Acc <- 256 - IWPN + WBASE + 2
SUB WBASE ;Acc <- 256 - IWPN + 2
SA CL IWPN ;pointer reflected to half of window actually
; stored
ZAC TEMP
SUB ONE
SA CL WINC ;to step forward through window (SUB -ONE)

Subroutine WNDW1

;This subroutine windows 64 points and writes the result back to the FFT
;workspace area in PRAM. It takes the points pointed at by IIPN and
;windows them by points pointed at by IWPN and places the results in the
;FFT workspace starting at FFTPTR.

WNDW1: LARK AR0,64 ;do 64 points
WW1: LAC IIPN ;Acc points to SHIREG(IIPn)
TBLR TEMP ;TEMP <- input point
SUB ONE
SA CL IIPN ;point to next input point
LT TEMP ;T <- input point
LAC IWPN ;Acc points to window point
TBLR TEMP ;TEMP <- window point
SUB WINC ; +/- 1
SA CL IWPN ;point to next window point
MPY TEMP
PAC
SA CL TEMP,X1 ;TEMP <- input point * window point
LAC FFTPTR
TBLW TEMP ;write result back to PRAM
SUB ONE
SA CL FFTPTR ;point to next locin FFT workspace PRAM
BANZ WW1 ;loop up until done 64 points
RET

Subroutine WNDW2

;This subroutine is similar to WNDW1 except that it does not write the
;windowed values back to PRAM. Rather it keeps them in DRAM so they may
;be accessed by the FFT. On entry, AR1 points to the first location in
;DRAM irw which to store values. AR1 is decremented once after each
;point

WNDW2: LARK AR0,64 ;do 64 points
WW2: LARF 1 ;use AR1
LAC IIPN ;Acc points to SHIREG(IIPn)
TBLR TEMP ;TEMP <- input point
SUB ONE
SACL IIPW ;point to next input point
LT TEMP ;T <- input point
LAC IWPW ;Acc points to window point
TBLR TEMP ;TEMP <- window point
SUB WINC ; +/- 1
SACL IWPW ;point to next window point
MPI TEMP
PAC
SACL R1DEC,NEWO,X1 ;TEMP <- input point - window point
BANZ WW2 ;loop up until done 64 points
RET

===============================================================
;WINDOW SUMMARY:
; PROGRAM: 140 words
; FRAME TIME: 5726 cycles
; FRAME TIME: 1.145 msec
===============================================================

-E7-
CALL FFT ;The 128 point complex FFT expects the first
;64 complex points (128 real) in page 0 of DRAM
;and the other 64 in the top half of the FFT
;workspace.

;FFT SUMMARY:
; PROGRAM: 2249 words
; FRAME TIME: 12000 cycles
; FRAME TIME: 2.400 msec

;UNRAVEL, SPECTRUM COMPRESSION

;NOTE:
;This section combines the raveling with the compression in the
;following manner. As harmonics are unraveled in pairs (one high and
;one low) the high ones are written back to PRAM and the low ones are
;compressed immediately. The high harmonics are compressed in a
;separate stage afterwards. This cannot be done all in one step since
;compressing the high ones immediately would over-write the harmonics
;required to unravel the low harmonics.

;DRAM LOCATIONS, PAGE 0:
;used for:
; ONE  assume pre-initialized to 1
; SP1  to point to Z(259-K)
; SP2  to point to Z(K)
; SP3  to point to destination when unraveling
; V1...V19 as in Fortran simulation
; OLDPTT to point to start of A0,B0 which are harmonics from last
; frame
; PHIPTR to point to phase accumulated up to last frame
; TABLA to point to start of stored sin table
; A1,B1 RE,IM part of harmonic from this frame
; A0,B0 RE,IM part of harmonic from last frame
; K32767 to be initialized to 32767
; BS  to be initialized to 32728
; BC  to be initialized to 1608
; IT,IT used in magnitude determination
; SIN32,COS32,COSFR,SINFR,FR used in cos and sin determination
; for converting to rectangular coords from polar
; TEMP  temporary storage

PAGE 1: (not used)

-E8-
; UNRAVELING:
; Read required constants and pointers from PHAM:
LAC A RAVTBL
TBLR K32767
ADD ONE
TBLR OLDPTR
ADD ONE
TBLR PHIPTR
ADD ONE
TBLR TABLA
ADD ONE
TBLR BC
ADD ONE
TBLR BS
ADD ONE
TBLR SP1
ADD ONE
TBLR SP2
ADD ONE
TBLR SP3

; unravel special cases Z(1), Z(2), Z(129), Z(130):
LAC SP2 ; point at Z(1)
TBLR Z0 ; Z0 <- Z(1)
ADD ONE
TBLR Z1 ; Z1 <- Z(2)
LAC Z0, S3
ADD Z1, S3
SA XL Z0 ; Z0 <- 8*( Z(1)+Z(2) )
ZAC
SA XL Z1 ; Z1 <- 0
LAC SP2
TBL W Z0 ; Z(1) <- 8*( Z(1)+Z(2) )
ADD ONE
TBL W Z0 ; Z(2) <- 0
ADD ONE, S7 ; point to Z(130)
TBL R Z1 ; Z1 <- Z(130)
SUB ONE
TBLR Z0 ; Z0 <- Z(129)
SA XL SP2
LAC Z0, S3
SA XL Z0 ; Z0 <- 8 * Z(129)
LAC Z1, S3
SA XL Z1 ; Z1 <- 8 * Z(130)
LAC SP2
TBL W Z0 ; Z(129) <- 8 * Z(129)
ADD ONE
TBL W Z1 ; Z(130) <- 8 * Z(130)
ADD ONE
ADD ONE, S1
SUB ONE, S7 ; point to Z(130+1+2-128) = Z(5)
SA XL SP2

-89-
LARK AR0,31 ;will loop 31 times

;UNLOOP: 
LAC SP1 ;point to Z(259-K)
TBLR Z9 ;read in IM part of high harmonic Z(259-K)
SUB ONE
TBLR Z8 ;read in RE part of high harmonic Z(258-K)

LAC SP2 ;point to Z(K)
TBLR Z0 ;read in RE part of low harmonic Z(K)
ADD ONE
TBLR Z1 ;read in IM part of low harmonic Z(K+1)
ADD ONE
ADD ONE,S1 ;wont pickup next harmonic, since only doing
          ;   every second one
SACL SP2

LAC Z8
ADD Z0
SACL V3 ;V3 <- Z(258-K) + Z(K)
SUB Z0,S1
SACL V5 ;V5 <- Z(258-K) - Z(K)

LAC Z9
ADD Z1
SACL V4 ;V4 <- Z(259-K) + Z(K+1)
SUB Z1,S1
SACL V6 ;V6 <- Z(259-K) - Z(K+1)

LT V1
MPY V5
PAC
SACH V7,X4 ;V7 <- V1 * V5
MPY V4
PAC
SACH V9 ;V9 <- V1 * V4

LT V2
MPY V4
PAC
SACH V8,X4 ;V8 <- V2 * V4
MPY V5
PAC
SACH V10,X4 ;V10 <- V2 * V5

;Update trig functions cos = V1, sin = V2
;Note: The trig functions are not read from a table as in the Fortran
;simulation. Rather they are computed recursively. If cos(theta) is
;known, then:
;   cos(theta+delta) = cos(theta)cos(delta) - sin(theta)sin(delta).
;For a constant interval around the circle (for delta constant)
;sin(delta) and cos(delta) are also constant and stored as such. In a
;similar manner:
;   sin(theta+delta) = sin(theta)sin(delta) + cos(theta)cos(delta).
; T is already V2
MPY BC ; BC = 32767 * COS(2PI/128)
PAC
LT BS ; BS = 32767 * SIN(2PI/128)
MPY V1
LTA V2 ; T <-- sin, P <-- cos * BS
SACH V2 ; V2 <-- V1*BS + V2*BC

; ZAC
MPY BS ; T is already V2
SPAC ; Acc <-- V2 * BS
LT V1 ; T <-- V1
MPY BC
APAC
SACH V1,1 ; V1 <-- V1*BC - V2*BS

; trig functions now updated for next pass, continue on unraveling

LAC V10
SUB V9
SACL V12 ; V12 <-- V10-V9
LAC V7
ADD V8
SACL V11 ; V11 <-- V7+V8

ADD V3,S3
SACL V3
LAC V3,S15
SACH A1 ; A1 <-- (V11 + 8*V3)/2
SUBH V11
SACH Z8 ; Z8 <-- (8*V3 - V11)/2

LAC V12,S11 ; Acc <-- V12 * 2**11
ADD V6,S14 ; Acc <-- (V12 + 8*V6) * 2**11
SACH Z9,S14 ; Z9 <-- (V12 + 8*V6)/2
LAC Z9
SUB V6,S3
SACL B1 ; B1 <-- (V12 - 8*V6)/2

LAC OLDPTR ; points to spectrum values from prev frame
TBLR A0 ; read in RE part of harmonic from prev frame
ADD ONE
TBLR B0 ; read in IM part of harmonic from prev frame

LAC SP1
TBLW Z9 ; write IM part of high harmonic back to PRAM
SUB ONE
TBLW Z8 ; write RE part of high harmonic back to PRAM
SUB ONE,S1
SUB ONE ; subtract three to skip next harmonic and point
SACL SP1 ; to IM part of the one following

-E11-
CALL CMPRSS ; compress low harmonic using A0, A1, B0, B1 as args

; BANZ UNLOOP ; do again

; So far, have unraveled every second harmonic and compressed the bottom half of the resulting spectrum. Now must compress the top half of the spectrum.
LARK AR0,32 ; have 32 harmonics to do

CMP2:
LAC SP2
TBLR A1 ; pick up RE part of desired harmonic
ADD ONE
TBLR B1 ; pick up IM part of desired harmonic
ADD ONE, S1
ADD ONE ; ready for next, skipping odd harmonics
SACL SP2

LAC OLDPTR
TBLR A0 ; pick up RE part of harmonic from prev frame
ADD ONE
TBLR B0 ; pick up IM part of harmonic from prev frame

CALL CMPRSS ; compress high harmonic using A0, A1, B0 and B1 as args

; BANZ CMP2 ; do once for each harmonic

; Finished unraveling and compression

; Subroutine CMPRSS

; The first stage of compression is to find the magnitude of the harmonic in question. This is done with the piece-wise approximation described by Adams and Bailey in MICRO Oct 83, pp 27-31. This method avoids taking the square root.
CMPRSS: LAC A1
ABS
SACL XT
LAC B1
ABS
SACL YI

SUB XT ; Acc <- XT-XT
BLEZ NEG ; branch if XT > YT

; YT > XT, therefore must exchange them
ZALH XT
ADDS YT
SACL XT
SACH YT ; XT and YT exchanged
NEG: LAC XT
   SUB YT,S2 ;Acc <- XT - #YT
   BZ POS

;Case XT < #YT:
   ZALH XT
   SUB XT,S13
   ADD YT,S15 ;Acc <- (0.875*XT + 0.5*YT) * 2**16
   B FINMAG

;Case XT >= #YT:
   POS: ZALH XT
        ADD YT,S13 ;Acc <- (XT + 0.125*YT) * 2**16
        FINMAG: SACH V13 ;V13 <- magnitude
        BNZ GTZERO ;check for zero
        LACK 1
        SACL V13 ;force to 1 to avoid div by 0 later
        ;Compute square of magnitude
        GTZERO: LT V13
        MPY V13
        PAC
        SACH V12,X4
        LT B1
        MPY A0
        PAC
        LT A1
        MPY B0
        SPAC ;Acc <- B1*A0 - B0*A1

;Now divide (B1*A0 - B0*A1) by V12:
;This divide routine is from the book "Digital Signal Processing
;Software" by DSPI Inc. The routine is by B. Bryden and E. Coll.
   SACH DDH ;save high part of dividend
   SACL DLL ;save low part of dividend
   ZAC
   SACL TEMP2 ;constant 0 for now
   LAC ONE,S15 ;Acc <- 8000h
   SUB ONE ;Acc <- 7FFFh
   SACL PLUS1 ;PLUS1 <- 7FFFh
   ;
   LARK AR1,15 ;will do 15 passes
   LAC ONE,S15 ;mask 8000h
   AND DDH ;Acc <- sign (8000h or 0000h)
   BLZ LESSQ ;branch if negative
   ZALH DDH
   ADDS DLL ;32 bit positive dividend
   LOOP1: ADD TEMP2 ;add zero
           SUBC V12 ;conditionally subtract divisor (V12)
           BANZ LOOP1

-E13-
SACH . REM ; remainder
AND PLUS1 ; force high bit of quotient to zero
B KSAVE
;
LESS4: LACK 0
SUBH DDH
SUBS DDL ; force 32 bit negative dividend to be +ve
LOOP2: ADD TEMP2
SUBC V12 ; conditionally subtract divisor (V12)
BANZ LOOP2
;
SACH REM
AND PLUS1 ; force high bit of quotient to zero
SACL TEMP2
ZAC
SUB TEMP2 ; negative quotient
; end of divide
;
KSAVE: SACL V15 ; V15 <- (B1#A0 - B0#A1) / (A1#A1 + B1#B1)
;
LAC PHIPTR ; points to phase accumulated to last frame
TBLR TEMP ; get phase for this harmonic
;
LAC V15,S15
ADDH TEMP
SACH V17 ; V17 <- phase + V15/2
;
; check phase and force inside range 0 to 2 pi
LAC V17
ADD PI ; PI is 3.1415 * 2**12
BLEZ LTP1 ; if (V17+PI) < 0, then V17 < -PI
SUB PI,S1 ; Acc <- (V17+PI)-2PI = V17-PI
BLEZ OK ; if (V17-PI) > 0, then V17 > PI
SUB PI ; Acc <- (V17-PI)-PI = V17-2PI
B STRV17
LTP1: ADD PI ; Acc <- (V17+PI)+PI = V17 + 2PI
;
STRV17: SACL V17 ; V17 now principal value of phase
;
OK: LACK PHIPTR
TBLW V17 ; write phase back to FRAM
ADD ONE
SACL PHIPTR ; save for next time
;
; now convert from polar to rectangular coords:
; the method of determining sin and cos of an arbitrary angle used in
; this section was developed by L.R. Morris.
LT V17 ; T <- phi * 2**12
MPYK 2608 ; 2608 = (32 * 2**9) / (2 * 3.1415)
PAC ; Acc <- INDEX * 2**21
SACH INDEX ; INDEX * 2**5, 0 <= INDEX <= 32
LAC INDEX,S5
SACL INDEX2 ; 0 <= INDEX2 <= 32 as 0 <= phi <= 2pi

-K14-
;33 value table sin + 1/4 cycle (8) = 41
LACK TABLA
ADD INDEX
TBLR SIM32
ADD ONE,S3 ;INDEX + 8 = index for cos
TBLR COS32

; LAC INDEX ;(decimal index) * 2**5
SUB INDEX2,S5 ;(integer index) * 2**5
SACL FR ;0 <= (FR = fraction * 2**5) <= 31

; SIN(A+FR/32) = SIN[A*(1-FR*(\{ 1-COS(2PI/32) \}/32 ))]
; +COSA*[ \{ SIN(2PI/32)/32 \} *FR ]
;
; [32767 - COS(2PI/32) * 32767] /32 = 1024 - 1004 = 20
; SIN(2PI/32) * 32767/32 = 200

; SIN(A+FR/32) = SIN[A*(32767-20*FR)+COSA*[200*FR]
; = SIN[A* COSFR +COSA* SINFR

; LAC K32767 ;32767
LT FR
MPYK 20 ;
SPAC ;32217 = 32767*COS(2PI/32) < 20*FR <= 32767
; where 32767 = COS(0)
SACH COSFR,S1

; MPYK 200 ;
;6400 = 32767 * SIN(2PI/32) >= 200*FR >= 0
; where 0 =SIN(0)
PAC
SACL SINFR,S1

; LT SIM32
MPY COSFR
PAC
LT COS32
MPY SINFR
APAC
SACH V19,S1 ;V19 <- SIN32*COSFR+COS32*SINFR
;COS(A+FR/32) = COSA*COSFR-SIN*SINFR

; MPY COSFR
PAC
LT SIM32
MPY SINFR
SPAC
SACH V18,S1 ;V18 <- COS32*COSFR-SIN*SINFR

; LT V13
MPY V18
PAC
SACH V18,X1 ;V18 <- V18 * V13

-W15-
MPY  V19
PAC
SACH V19,X1

; LAC SP3 ;write RE part back to PRAM
TBLW V18
ADD ONE
TBLW V19 ;write IM part back to PRAM
ADD ONE
SACL SP3

; LAC OLDPTR ;point to area for prev frame harmonic storage
TBLW A1 ;write RE part back to PRAM for next frame
ADD ONE
TBLW B1 ;write IM part back to PRAM for next frame
ADD ONE
SACL OLDPTR

; RET

;=================================================================================================

; UNRAVELING AND COMPRESSION SUMMARY: ;
; PROGRAM: 296 words ;
; FRAME TIME: 16616 cycles ;
; FRAME TIME: 3.323 msec ;
;=================================================================================================

-316-
;PRE-FFT RAVEL

;NOTE:
;This section ravel all the harmonics in pairs, writing the high ones
;back to the FFT workspace in FRAM. The lower half is left in page 0
;of DRAM for the FFT.

;DRAM LOCATIONS, PAGE 0
; locations 0-127 used to pass half the coeffs to FFT

;PAGE 1 DRAM:
; locations for:
; ONE holds 1
; SP1 pointer to bottom half of spectrum in FRAM
; SP2 pointer to top half of spectrum in FRAM
; Z1,Z1,Z8,Z9 holds RE and IM parts of low and high harmonics
; BS,BS as for raveling section
; V21-V27 as for Fortran simulation

LARP 1
LARK AR0,0
LDPE 1
LACK 1
SACL ONE

LACKA RAVPTR
TBLR SP1
ADD ONE
TBLR SP2
ADD ONE
TBLR BC
ADD ONE
TBLR BS
ADD ONE
TBLR V20
ADD ONE
TBLR V21

;special cases Z(1),Z(2),Z(129),Z(130)
LAC SP2
TBLR R1INCR,NEW1 ;loc 0 <- Z(1)
ADD ONE
TBLR Z1 ;Z1 <- Z(2)
SACL SP2
ZAC
SACL Z0 ;Z0 <- 0
SUB Z1 ;Acc <- -Z(2)
SACL R1INCR,NEW1 ;loc 1 <- -Z(2)
LAC SP2
ADD ONE,ST ;point to Z(130)
TBLW Z0 ;Z(130) <- 0
SUB ONE
TBLW Z0 ;Z(129) <- 0

-E17-
RALOOP: LARP 1
LAC SP2
ADD ONE
TBLR Z0 ;read RE part of harmonic from PRAM Z(K)
ADD ONE
TBLR Z1 ;read IM part of harmonic from PRAM Z(K+1)
SACL SP2

LT V20 ;T <- COS
MPY Z0
PAC ;Acc <- Z(K) * V20 + TW015
SACH V22, X1 ;V22 <- Z(K) * V20
MPY Z1
PAC
SACH V23, X1 ;V23 <- Z(K+1) * V20

LT V21
MPY Z0
PAC
SACH V24, X1 ;V24 <- Z(K) * V21
MPY Z1
PAC
SACH V25, S1 ;V25 <- Z(K+1) * V21

;update sin and cos for next pass:
;The sin and cos are calculated as in the unraveling section.
;(V20 is cos, V21 is sin)

MPY BC ;V21 is in T
PAC ;P <- BC * sin
LT BS
MPY V20 ;P <- BS * cos
LTA V21
SACH V21, X1 ;V21 <- BC * V21 + BS * V20

ZAC ;T is V21 from before
MPY BS
SPAC ;Acc <- -BS * V21
LT V20
MPY BC
LTA V21 ;Acc <- BC * V20 - BS * V21
SACH V20, X1

;V20 and V21 updated

LAC V22
SUB V25
SACL V26 ;V26 <- V22 - V25
LAC V23
ADD V24
SACL V27 ;V27 <- V23 + V24

-B18-
LAC Z0,S15
ADD V26,S15
SACH R1INC,NEW1 ; (page 0) <- (Z(K)+V26)/2
SUBH V26
SACH Z8 ; Z8 <- (Z(K)-V26)/2

ZAC
SUB V27,S15
ADD Z1,S15
SACH Z9 ; Z9 <- (Z(K+1)-V27)/2
SUBH Z1
SACH R1INC,NEW0 ; Z1 <- (-Z(K+1)-V27)/2

LAC SP1
TBLW Z9 ; write Z(259-K) to PRAM
SUB ONE
TBLW Z8 ; write Z(258-K) to PRAM
SUB ONE
SACL SP1

BANZ RALOOP

;=============================================================================
;RAVELING SUMMARY:
;PROGRAM 88 words
;FRAME TIME: 4273 cycles
;FRAME TIME: 0.855 msec
;=============================================================================

;FFT (used for inverse DFT)

CALL FFT

;=============================================================================
;FFT SUMMARY:
;PROGRAM: (already counted)
;FRAME TIME: 12000 cycles
;FRAME TIME: 2.400 msec
;=============================================================================

-E19-
; OVERLAP ADD VERSION 13
;
; NOTE:
; All Z values are divided by 4 before overlap adding to compensate for
; gains of 64 on A/D, 1/128 on FFT and 8 in the unraveling section.
;
; DRAM LOCATIONS, PAGE 0
; ONE to contain 1
; TZ1 contains addr of Z(1) in PRAM from prev sect
; ZPTR1 pointer to section 1 of FFT workspace
; ZPTR2 2
; ZPTR3 3
; ZPTR4 4
; TIOP1 holds IOP1
; TIOP2 IOP2
; TIOP3 IOP3
; TIOP4 IOP4
; ZJ1 points to Z(J+1), Z(J+65), Z(J+129), Z(J+193)
; ZJ2 points to Z(J+2), Z(J+66), Z(J+130), Z(J+194)
; SHRJ SHREG0(IOPn+J)
; SHRJ1 SHREG0(IOPn+J+1)
; FTEMP holds location of IOP1
;
; set up pointers:
;
; LDPK 0
; LACK 1
; SAACL ONE
; LACKA TZ1
; SAACL ZPTR1 ; ZPTR1 <- location of Z(1)
; ADD ONE,36 ; add 64
; SAACL ZPTR2 ; ZPTR2 <- location of Z(65)
; ADD ONE,36 ; add 64
; SAACL ZPTR3 ; ZPTR3 <- location of Z(129)
; ADD ONE,36 ; add 64
; SAACL ZPTR4 ; ZPTR4 <- location of Z(193)
;
; LACKA IOP1
; SAACL FTEMP ; for future ref when pointers are updated
; TBLR TIOP1
; ADD ONE
; TBLR TIOP2
; ADD ONE
; TBLR TIOP3
; ADD ONE
; TBLR TIOP4
;
; LARK AR0,32 ; use as counter
;
; Main loop like Fortran DO J=0,63,2
;

=E20= 
; first section:
OLOOP:  LAC  ZPTR1
        TBLR  ZJ1  ;ZJ1 <- Z(J+1).
        ADD  ONE
        TBLR  ZJ2  ;ZJ2 <- Z(J+2)
        ADD  ONE
        SACL  ZPTR1  ;point to Z(J+3) for next pass

        LAC  TIOP1
        TBLR  SHRJ
        ADD  ONE
        TBLR  SHRJ1  ;SHRJ1 <- SHREG0(IOP1+J)

        LAC  SHRJ,S12  ;SHRJ1 <- SHREG0(IOP1+J+1)
        ;won't update TIOP1 since have to store
        ; results back in the same place

        LAC  SHRJ1,S12  ;SHRJ1 <- SHREG0(IOP1+J+1)
        SUB  ZJ2,S10  ;ACC <- TWO12 * SHREG0(IOP1+J+1)
        SACH  SHRJ1,X4  ;SHRJ1 <- SHREG0(IOP1+J+1) - Z(J+2)/4

        LAC  TIOP1
        TBLW  SHRJ
        ADD  ONE
        TBLW  SHRJ1  ;SHREG0(IOP1+J+1) <- SHREG0(IOP1+J) + Z(J+1)/4
        ADD  ONE
        SACL  TIOP1

; section 2:
LAC  ZPTR2
        TBLR  ZJ1  ;ZJ1 <- Z(J+65)
        ADD  ONE
        TBLR  ZJ2
        ADD  ONE
        SACL  ZPTR2  ;point to Z(J+67) for next pass

        LAC  TIOP2
        TBLR  SHRJ
        ADD  ONE
        TBLR  SHRJ1  ;SHRJ1 <- SHREG0(IOP2+J)

        LAC  SHRJ,S12  ;SHRJ1 <- SHREG0(IOP2+J+1)
        ;won't update TIOP2 since have to store
        ; results back in the same place

        LAC  SHRJ1,S12  ;SHRJ1 <- SHREG0(IOP2+J+1)
        SUB  ZJ2,S10  ;ACC <- TWO12 * SHREG0(IOP2+J+1)
        SACH  SHRJ1,X4  ;SHRJ1 <- SHREG0(IOP2+J+1) - Z(J+66)/4

-K21-
LAC  TIOP2
    TBLW  SHRJ
    ADD  ONE
    TBLW  SHRJ1
    ;SHREGO(IOP2+J) <- SHREGO(IOP2+J)+Z(J+65)/4
    ADD  ONE
    TBLW  SHRJ1
    ;SHREGO(IOP2+J+1) <- SHREGO(IOP2+J+1)
      +Z(J+66)/4
    SACL  TIOP2

;section 3:
LAC  ZPTR3
    TBLR  ZJ1
    ;ZJ1 <- Z(J+129)
    ADD  ONE
    TBLR  ZJ2
    ;ZJ2 <- Z(J+130)
    ADD  ONE
    SACL  ZPTR3
    ;point to Z(J+131) for next pass

LAC  TIOP3
    TBLR  SHRJ
    ;SHRJ <- SHREGO(IOP3+J)
    ADD  ONE
    TBLR  SHRJ1
    ;SHRJ1 <- SHREGO(IOP3+J+1)
    ;won't update TIOP3 since have to store results back in the same place
    LAC  SHRJ,S12
    ;ACC <- TWO12 * SHREGO(IOP3+J)
    ADD  ZJ1,S10
    ;ACC <- TWO12 * (SHREGO(IOP3+J)+Z(J+129)/4)
    SACH  SHRJ,X4
    ;SHRJ <- SHREGO(IOP3+J) + Z(J+129)/4

LAC  SHRJ1,S12
    ;ACC <- TWO12 * SHREGO(IOP3+J+1)
    SUB  ZJ2,S10
    ;SHRJ1 <- SHREGO(IOP3+J+1) - Z(J+130)/4
    SACH  SHRJ1,X4
    ;SHRJ1 <- SHREGO(IOP3+J+1) - Z(J+130)/4

LAC  TIOP3
    TBLW  SHRJ
    ;SHREGO(IOP3+J) <- SHREGO(IOP3+J)+Z(J+129)/4
    ADD  ONE
    TBLW  SHRJ1
    ;SHREGO(IOP3+J+1) <- SHREGO(IOP3+J+1)
      +Z(J+130)/4
    ADD  ONE
    SACL  TIOP3

;section 4: (overlap and replacement instead of overlap add)
LAC  ZPTR4
    TBLR  ZJ1
    ;ZJ1 <- Z(J+193)
    ADD  ONE
    TBLR  ZJ2
    ;ZJ2 <- Z(J+194)
    ;AR0 points back at Z193
    ADD  ONE
    SACL  ZPTR4
    ;point to Z(J+195) for next pass

ZAC  SUB  ZJ2
    ;ZJ2 <- -ZJ194
    SACL  Z2
LAC  TIOP4
TBLW  ZJ1  ;SHREG0(IOP4+J) <- Z(J+193)
ADD  ONE
TBLW  ZJ2  ;SHREG0(IOP4+J+1) <- Z(J+194)
SACL  TIOP4

; BANZ OLOOP  ;decrement 8RO used as counter

;OVERLAP SUMMARY:
;  PROGRAM:  103  words
;  FRAME TIME:  4061  cycles
;  FRAME TIME:  0.812  msec

-823-
;END OF FRAME PROCESSING VERSION 13

;DRAM LOCATIONS, PAGE 0:
; locations for:
; ONE still 1 from last section
; M64 to be initialized to -64
; IBASE base address of SHREGI
; IMAX maximum address+1 of SHREGI
; OBASE base address of SHREGO
; OMAX maximum address+1 of SHREGO
; TEMP temporary storage
; IIX holds IX, pointer to pointer
; XIPIN,XIPUT storage for IPIN and IPOUT
; IOPTR to point to IOPT in PRAM

;DRAM LOCATIONS, PAGE 1:
; TIPIN holds input pointer IPIN, as for ISR
; TIPUT holds output pointer IPOUT, as for ISR
; NDONE when = 0 indicates that 64 samples have been clocked in
; and out by ISR

;Read in required constants and pointers from PRAM:
; LACKA EOPPTR
; TBLR IMAX
; ADD ONE
; TBLR OMAX
; ADD ONE
; TBLR IOPTR

; LAC
; SUB ONE,56; ACC <- -64
; SA CL M64
; LAR ARO; ARO <- -64
; LP DK 1
;
; MOTYET: LAC NDONE; check to see if 64 samples done by ISR
; BLZ MOTYET
; SAR ARO,NDONE; reset input output sample counter to -64
;
; Check pointers IPIN and IPOUT: (stored in TIPIN and TIPOUT)
; LAR ARO,TIPIN
; LAR AR1,TIPOUT
; LP DK 0
; SAR ARO,XIPIN
; SAR AR1,XIPOUT

; IPIN and IPOUT have been updated by the ISR, but must see that they
; still point inside their respective shift registers.

; LAC XIPIN
; SUB IMAX
; BLZ TB1; branch if no correction needed
; SUB ONE,58
; SUB ONE,56; sub 320

TB1:
; ADD IMAX
; SA CL XIPIN; IPIN done

-E24-
LAC XIPOUT
SUB OMAI
BLZ TB2 ;branch if no correction needed
SUB OWE,38 ;sub 320
SUB OWE,36
TB2:
ADD OMAI
SACL XIPOUT ;IPOUT done

;Update one of IOP1, IOP2, IOP3 or IOP4 as indicated by IX:
LAC IX
ADD IOPTR
SACL IOPTR ;Acc now points to IOPn to update
TBLR TEMP
LAC TEMP ;Acc <- IOPn
SUB OWE,36 ;Pointer is now updated, but must check that
; it is still within limits
SUB OBASE
BGZ UPDT2 ;branch if pointer was OK
ADD OWE,38
ADD OWE,36 ;add 320
UPDT2:
ADD OBASE
SACL TEMP
LAC IOPTR
TBLW TEMP ;write updated pointer back to PRAM

LAR ARG,XIPIN
LAR AR1,XIPOUT
LDPK 1
SAR ARG,TIPIN ;write input and output pointers back to
SAR AR1,TIPOUT ; their normal places in page 1

;Done one frame. Branch back to top of frame to start again:
B TOF

;================================================================================
;END OF FRAME SUMMARY:
; PROGRAM: 58 words
; FRAME TIME: 65 cycles
; FRAME TIME: 0.013 msec
;================================================================================
END
11-06-86
FIN