RULE BASED SINUSOIDAL ENCODING OF SPEECH

THESIS

Luis M. F. Alenquer, Capt, PAF

AFIT/GE/ENG/90M-1

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GE/ENG/90M-1

RULE BASED SINUSOIDAL ENCODING OF SPEECH

THESIS

Presented to the Faculty of School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Electrical Engineering

Luis M. F. Alenquer, Capt, PAF

March 1990

Acknowledgments

This work is dedicated to those I love; my parents, Luis and Etelvina Alenquer; my wife, Leonor, whose support and patience made this work possible; and my newborn son, Carlos.

Also, special thanks to my thesis advisor, Dr. Matthew Kabrisky, for his suggestions, support, and freedom of work.

Table of Contents

List of Figures

Abstract

A system was developed to investigate the data rate necessary to transmit speech using a rule based sinusoidal model. The system consists of a speech analyzer and a synthesizer. The analyzer outputs discrete frequencies and quantized amplitudes and phases of selected speech spectral components. The synthesizer reconstructs speech from these components based on a sinusoidal model. The selection of spectral components for voiced speech regions is based on the detection of harmonics of the fundamental frequency. To obtain a specific number of spectral components, a variable amplitude threshold is applied to the detected harmonics and their nearest neighbors. For unvoiced regions only the variable amplitude step is applied. The lowest data rate obtained for toll quality speech was about 18 Kbps. This system was implemented in Fortran 77 on a VAX 11/780 computer. Visual analysis of speech was provided by the software package SPIRE (Speech and Phonetics Interactive Research Environment).

RULE BASED SINUSOIDAL ENCODING OF SPEECH

I. Introduction

The importance of digital speech transmission systems has increased over the last few years because they offer two considerable advantages over analog systems. The first is the replacement of major portions of analog circuitry by digital integrated circuits, which increases reliability and stability (13:384). The second is a very low error rate. In long distance communications, several repeater stations are used to compensate for attenuations in the line; analog repeaters have cumulative errors, because they don't compensate exactly for line attenuations that vary with frequency. On the other hand, digital regenerators are immune to cumulative noise (14:78).

These and some other advantages related to digital design flexibility are somewhat offset by the bigger channel bandwidth needed to transmit digital speech. Transmission lines, microwave links, etc, are band limited by nature,

making bandwidth a very important economical factor and the research for smaller bandwidth digital speech channel worthwhile. Theoretically, smaller bandwidths can be obtained by using an efficient code to reduce speech redundancy.

Background

Currently there are several systems suitable for digital speech transmission. These systems can be roughly divided into two categories depending on the encoding scheme used. When they generate an approximation to the input, based on minimizing the distance between signal and approximation, they are usually called waveform encoders (11:649). On the other hand, systems that reconstruct the speech signal based on the magnitudes of the signal short-term spectrum are called analysis/synthesis coders (11:649).

Pulse Code Modulation (PCM) is an example of a waveform encoder that is in use world-wide for speech transmission. The PCM codec (coder-decoder) samples the speech signal at 8000 samples per second, and produces a 7 or 8 bit number per sample, which yields a data rate of 56 or 64 Kbps. To reduce this number, some systems where proposed that use past data to predict the next sample value. These schemes are called n-tap Differential Pulse Code Modulators (DPCM), and use an n-tap Linear Prediction Coding (LPC) filter, where n represents the number of past values used to make the prediction.

The data rate reduction is accomplished by transmitting

the difference between the prediction and the actual sample value. The difference signal has reduced variance when compared to the variance of the original signal, which justifies the smaller data rate obtained (11:625). The bit rate reduction, for a fixed output signal-to-noise ratio (SNR), is usually measured by the prediction gain, defined as the ratio of input to output variances; DPCM coders have prediction gains in the order of 6 to 8 dB (11:633).

Some other techniques are able to improve this gain by adding data dependent adjustments to the coefficients of the LPC filters, for example the International Telegraph and Telephone Consultative Committee (CCITT) has set a 32 Kbps Adaptative Differential Pulse Modulator (ADPCM) as a standard for toll-quality speech (11:639). It is worthwhile noting that the CCITT specifies a 64 Kbps PCM channel for speech in its Integrated Services Digital Network (ISDN) standard, although a 32 Kbps is already possible, and 16 Kbps will be enough by the time ISDN will be completely implemented (14:108).

In the category of analysis/synthesis coders, the most representative examples are the vocoder (voice coder) and the Linear Predictive Coder (LPC). The vocoder models the speech mechanism. A typical vocoder consists of several narrow band filters that estimate the power spectrum of the speech signal, and of an excitation estimator to decide if the signal is periodic - voiced speech, or turbulent - unvoiced speech. The spectral amplitudes and the lower band of

frequencies that contain pitch information are transmitted along with the voiced / unvoiced decision. The spectral amplitudes affect the gain of the filter banks in the receiver (decoder), which are driven by a noise generator in case of unvoiced sounds or by the lower band of voice frequencies (using a non-linear spectral expansion process). The total required to transmit vocoder speech can be reduced to 700 Hz or less (11:652). This implies 11.2 Kbps if the transmitted information is digitized with 8 bits per sample. The well known KY-585 vocoder transmitted intelligible, but low quality speech at 2400 bps.

The Linear Prediction Coder (LPC) makes use of a scheme similar to ADPCM, but instead of transmitting the difference between the predicted and actual value, it transmits the filter coefficients that are used to make the prediction, and the voiced/unvoiced excitation. The rational behind is that if an optimal algorithm is used to predict the values then the difference signal is essentially zero, and need not be transmitted.

In the area of speech enhancement, recent work at AFIT produced an analysis/synthesis coder with a very high quality of output speech (5). Essentially, this system reconstructed speech from selected amplitude and phase spectral components of time slices of the input speech. The number of required components (54 to 85) to obtain high quality reconstructed speech generated the motivation for the following investigation presented in this thesis.

Problem

The purpose of this thesis is to implement a rule based system for speech analysis and synthesis, using processing techniques developed around a sinusoidal speech model and to investigate and minimize the data rate necessary for digital speech transmission. The system should be flexible enough to allow quick changes on the rules for selecting frequency components that characterize speech and should also permit the variation of the number of bits needed to quantize those components.

Scope

The speech is reconstructed from the frequency, quantized amplitude and phase of selected spectral components. The speech spectrum, on which the ruled base selection takes place, was obtained by taking a 512-point Discrete Fourier Transform (DFT) of speech time slices. To obtain a fixed frame length for transmission, a post-selection was done by a variable threshold.

Approach

The overall system can be divided into two subsystems: the speech analyzer and the speech synthesizer. The approach for the speech analyzer is outlined as follows. First, Hamming windows with 50% overlap are applied to the speech time frames; this doubles, by induced redudancy, the total

amount of data processed. Then a 512-point DFT is used to perform the conversion to the frequency domain. In the frequency domain the components that characterize the speech are selected based on a heuristic rule. Finally, the amplitudes and phases of the selected components are quantized and packed into a vector form, and a frame with frequency information about these components is generated.

The synthesizer portion of the system is outlined as follows. First the reduced frames containing the quantized amplitudes and phases are expanded to their original size, by using the frequency vector information. Then, a time domain conversion algorithm is used, and the resultant wave shapes are averaged to obtain the original number of frames. Finally, the time waveform is amplitude normalized to drive a digital-to-analog converter.

Sequence of Presentation

Chapter Two presents the hardware and software environment used to process and analyze speech waveforms.

Chapter Three describes the entire system. The modules are functionally analyzed and details for the algorithms used are given.

Chapter Four presents the results. Input and output time waveforms, and respective spectrograms are showed, and a qualitative evaluation is given for the different data rates obtained by varying the number of selected components, or the number of quantization bits used for the amplitudes

and phases.

Chapter Five provides conclusions and recommendations for upgrading and using this system, and the Appendices provide source code listings, data flow diagrams and additional results.

II. Acoustic Processing Environment

The purpose of this chapter is to describe the software and hardware tools used to implement and test the system. The first section describes the speech digitizing system. Then the programing development environment is described. The last section, discusses SPIRE a speech analysis tool that was used to visualize speech files.

Speech Digitizing System (1)

The Digital Sound Corporation, DSC-200, was the analog-digital converter used to digitize and playback the input and output speech files, respectively. This digitizer has the capability of sampling audio signals at a maximum rate of 5 KHz, and produces a speech file constituted by consecutive 256-point arrays with integer values from -32,768 to 32,767. The sampling rate used was 8 KHz, yielding a speech frame length of 32 msec. The resultant digitized speech was stored in a VAX 11/780 system for further processing.

Software Development

The software was written in Fortran 77 on a VAX 11/780 system, under the VMS operating system. Besides, the VAX 11/780, the program was sometimes executed on a MicroVAX III machine. Structured programing techniques were used to written the source code. Data flow diagrams and source code are presented on Appendices C and D, respectively.

SPIRE (12)

Speech and Phonetics Interactive Research Environment (SPIRE) is a software package running on a 3600 LISP machine. SPIRE allows the user to interactively examine and process speech by generating several bit-mapped displays with resolutions of 1280 by 760 pixels or 1216 by 773 pixels. Among the various displays that SPIRE generates, time waveforms and narrow-band spectrograms were the most frequently used. They helped in system development and provide an illustration of the results obtained. All the displays are synchronized and the narrow-band spectrogram uses a bandwidth of 78 Hz.

III. Speech Processing System

The processing system consists of a speech analyzer and a speech synthesizer whose main functions are described in the following subsections. Meanwhile, the calculation of the data rate as a function of the overall system is presented.

The data rate is determined by the flux of information from the analyzer to the synthesizer. This information is contained in three frames, i.e. arrays, with length set a priori as an input parameter. One frame contains the spectral frequencies, and the other two the spectral amplitudes and phases. The number of bits used per frame slot is fixed for the frequency frame (8 bits), and is determined by the number of quantization levels for the other two frames. We must, also, consider, that these frames need to be transmitted in half the time of the input speech frame, because overlapping the windows in the analyzer has doubled the amount of data to be processed. These considerations yield the following expression to calculate the data rate.

$$\text{Data Rate} = 2L(A+P+8)/T \qquad (3.1)$$

where

L = frame length (# of slots in a frame)

A = number of amplitude quantization bits

B = number of phase quantization bits

T = duration of speech time frame (32 ms)

Speech Analyzer

A simplified block diagram is showed on Figure 3.1. A more detailed representation is provided in the form of data flow diagrams in Appendix C. The analyzer receives its input from a digitized speech file produced by the DSC-200 with the sampling rate set to 8 KHz. This input is Hamming windowed into 256 points frames, each representing 32 ms of speech. Except for the first frame, each time frame is completely processed before another frame is inputed. The exception results from the need to generate overlapping data for processing and has two main consequences: the first is that the total amount of speech frames is approximately doubled; the second is that one frame, or 32 ms, of intrinsic delay is generated between input and output.

After being windowed a speech frame is converted to the frequency domain by a DFT algorithm. The resulting spectral amplitudes, after being processed by a selection routine, are packed into a reduced length frame and quantized. The corresponding phases are also quantized and a frame containing frequency information is generated.

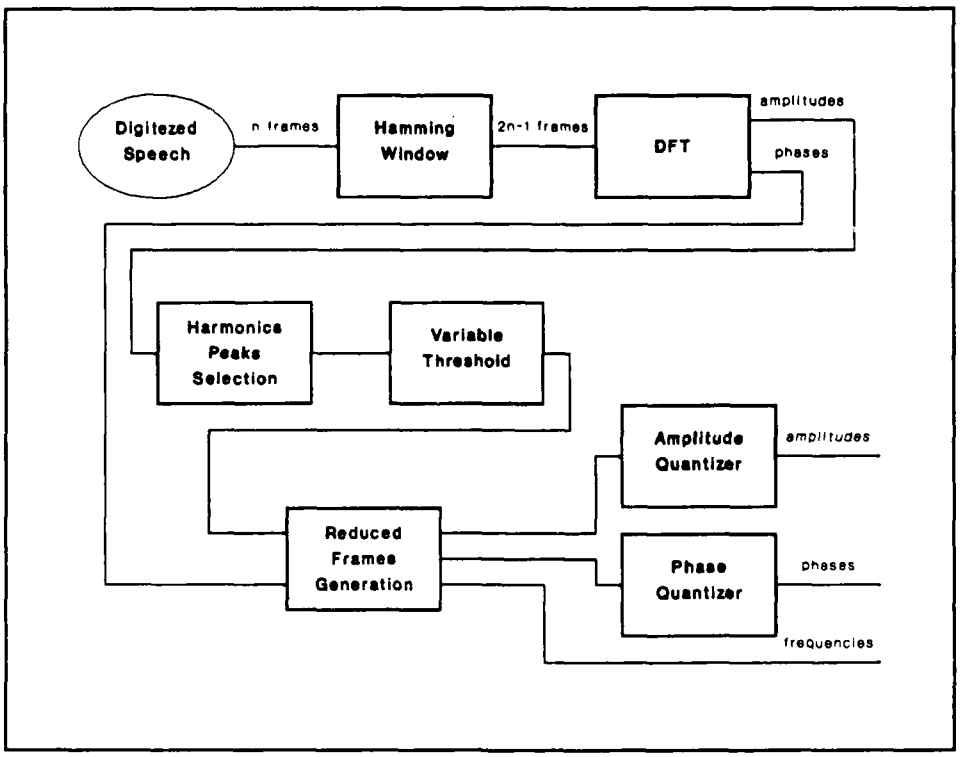


Figure 3.1. Speech Analyzer

Windowing. To reduce the frequency ringing effects near discontinuity points (Gibbs phenomenon) associated with rectangular windows (3:59), a Hamming window is used to sample the input signal. This window offers a good compromise between dynamic range and transition time which allows good resolution of closely spaced frequency tones having great differences in amplitude. The Hamming window is defined by the following equation:

$$W_{Ham}(i) = 0.54 - 0.46\cos[2\pi i/(L-1)] \qquad (3.2)$$

where

L = frame length (256)

i = 0, 1, 2 ... (L-1)

To avoid loss of data of the input time series, due to small values of the Hamming window near the boundaries, a overlapping scheme of 50% is used before the window is applied (3:56). This overlapping, illustrated in Figure 3.2, increases the number of input frames (n) to 2n-1.

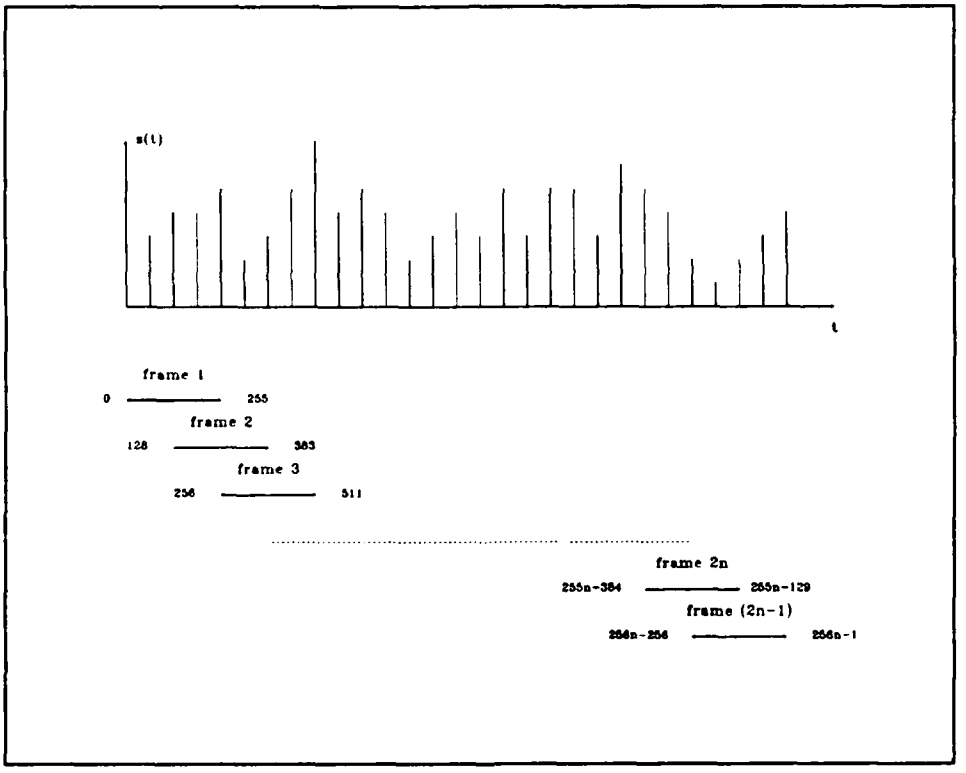


Figure 3.2. Hamming Windows with 50% Overlap

Discrete Fourier Transform. To convert the speech to the frequency domain, a 512-point Fast Fourier Transform (FFT) (6:457) was taken of the 256-point time series packed

with 256 zeros. The real and imaginary components out of the FFT were used to calculate the spectral amplitudes and phases. The result is two 256-point amplitude and phase frames, with a resolution of 15.625 Hz. Figure 3.3 illustrates this process.

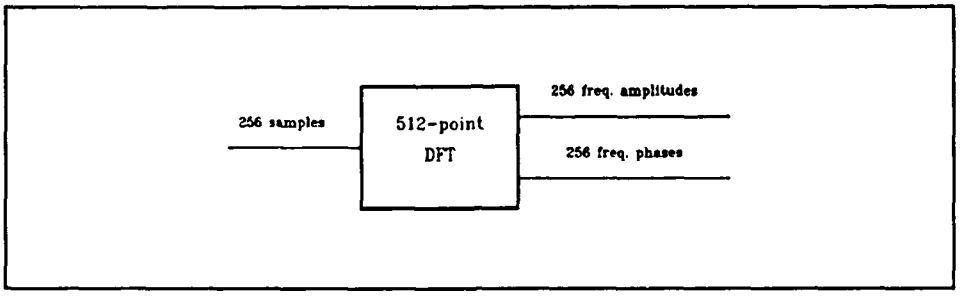


Figure 3.3. Discrete Fourier Transform

Harmonics/Peaks Selection. To reduce the number of spectral components that characterize speech, a rule based selection was carried out on the amplitude frame generated by the DFT. First, the energy of the frame was computed to determine if the frame contained voiced or unvoiced speech (5:6). If the energy was above a certain threshold level, then the frame was considered to contain voiced speech. For the speech files that were processed, this level was empirically set to 10^6. Based on this classification, different selection rules were applied.

The selection rules for voiced speech make use of the fact that voiced sounds are periodic (9:4). This implies that the frequency spectrum is composed of harmonics of the

fundamental frequency of the time waveform, also known as the glottal pitch frequency, and allow the potential of speech reconstruction from these harmonics (7:27.6.2) . A subproduct of selecting harmonics is the elimination of non-harmonic noise.

Because, synthesized speech from the exact harmonics appears to have a musical noise (2:3-6), all the rules developed were based on the selection of the highest amplitude frequency, in a frquency region containing the harmonic and a few of its neighboring frequencies. The different rules tried differ by the way the glottal frequency was determined, the way it was used to search for harmonics and by the number of neighbors used. One of the rules tried was used by Bashir in his master's thesis: the number of neighbors of the harmonic was set to two, the glottal frequency was fixed at 125 Hz (value characteristic of male speech) and the next harmonic was determined by adding the glottal frequency to the frequency of the highest amplitude neighbor. This allowed local and global adjustments to the harmonic frequency.

In some other rules developed and tried, the glottal frequency was not fixed, but was set from frame to frame to the maximum amplitude in a region were the glottal frequency was expected to lie, generally from 125 to 187.5Hz (4). Also tried was harmonic selection in the neighborhood of multiples of the glottal frequency, thus only allowing local variations in the harmonic position, and not global as in Bashir's

thesis.

Subjective auditory tests, determined the rule that was used to produce the results presented in Chapter 4. The number of neighbors was set to four, the glottal pitch frequency was set to maximum amplitude frequency in the region 125 to 187.5 Hz, and this value was used to step through the spectrum in search of the harmonics, thus allowing local and global variations to the harmonics position.

For the unvoiced regions no selection rule was develloped, even though a separate software module was designed to allow a rule selection in the future use of the system. The selection of peaks in the unvoiced frame, is only performed by the variable threshold module, which selects the peaks with amplitudes above the threshold.

Variable Threshold. This module sets up a variable threshold to perform post-selection on the harmonic or peak frame generated by the above process. Its implementation makes use of the fact that the energy in speech falls off at a rate of 6 db per octave after about 625 Hz (10:651); this threshold was approximated by the following equation:

$$\text{Threshold} = a/[1 + (i/42)^2]^{1/2} \qquad (3.3)$$

The number of frequency amplitudes above the threshold depends on the value used for a. A successive approximation

algorithm increases or decreases the value a untill the number of components above the threshold equals the desired value L. All the components below the threshold are set to zero, therefore the resulting number of amplitude frequencies in a frame, after selection and post-selection, equals the input parameter L.

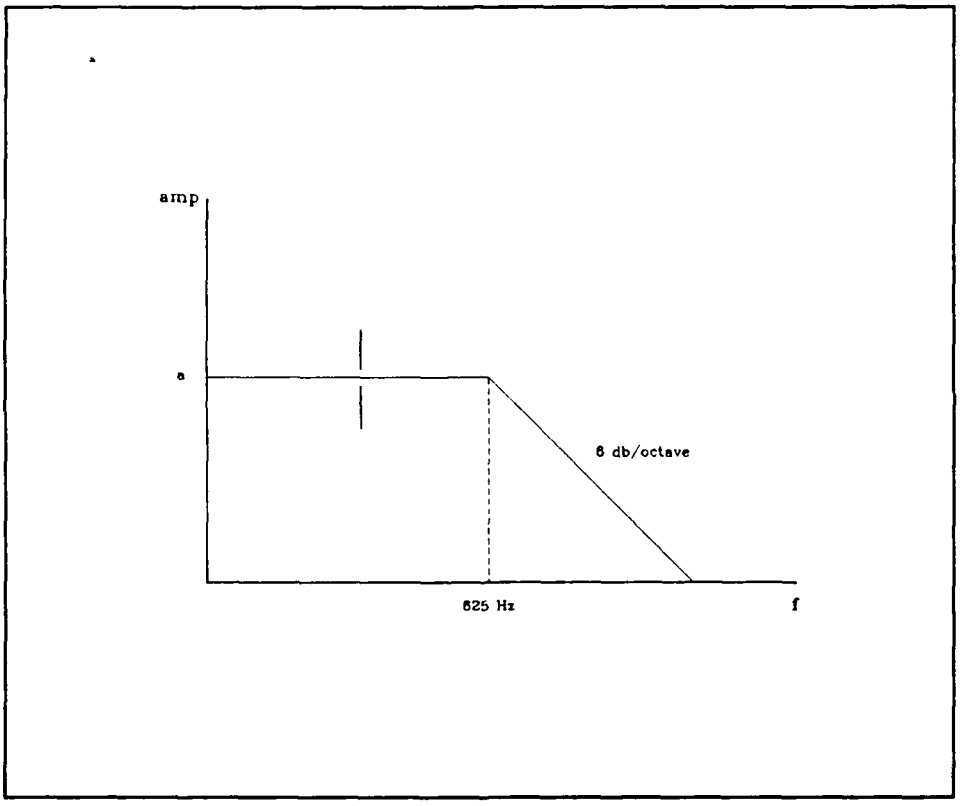


Figure 3.4. Variable Threshold

Amplitude, Frequency and Phase Frame Generation. The selected amplitude components, originally in a 256-point array, are encapsulated in an array of length L. The frequency of each amplitude component is written into another

complementary array with the same length. This frequency frame indexes the phase frame generated by the DFT and the phases corresponding to the selected amplitudes are also written to a reduced frame of length L. The two reduced frames with amplitudes and phases are passed to the respective quantizer modules and the frequency frame is outputed to the synthesizer.

Amplitude Quantizer. The quantizer implemented is uniform with midtread at the origin; it uses quantile intervals determined by the number of bits needed to encode the total number of intervals and by the maximum amplitude level it can handle without saturating. Both the number of bits (A in equation 3-1) and the dynamic range are inputs to the system. The maximum amplitude level was set to $6x10^5$. This value was experimentally determined, for the speech files used, as a compromise between the best use of the dynamic range and less operating time in the saturation region.

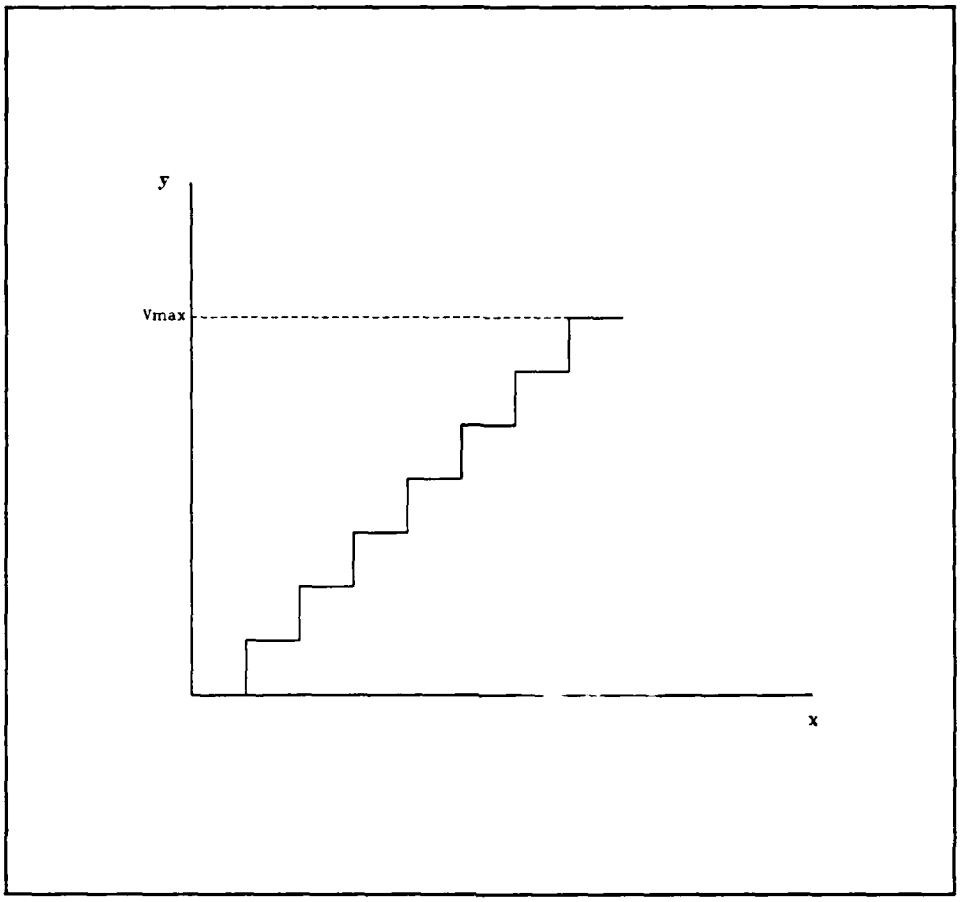


Figure 3.5. Amplitude Quantizer

Phase Quantizer. The phase quantizer is, also, a linear quantizer, with the levels uniformly distributed between π and $-\pi$, and with midtread at the origin. The number of levels is determined by the input parameter P (see equation 3.1).

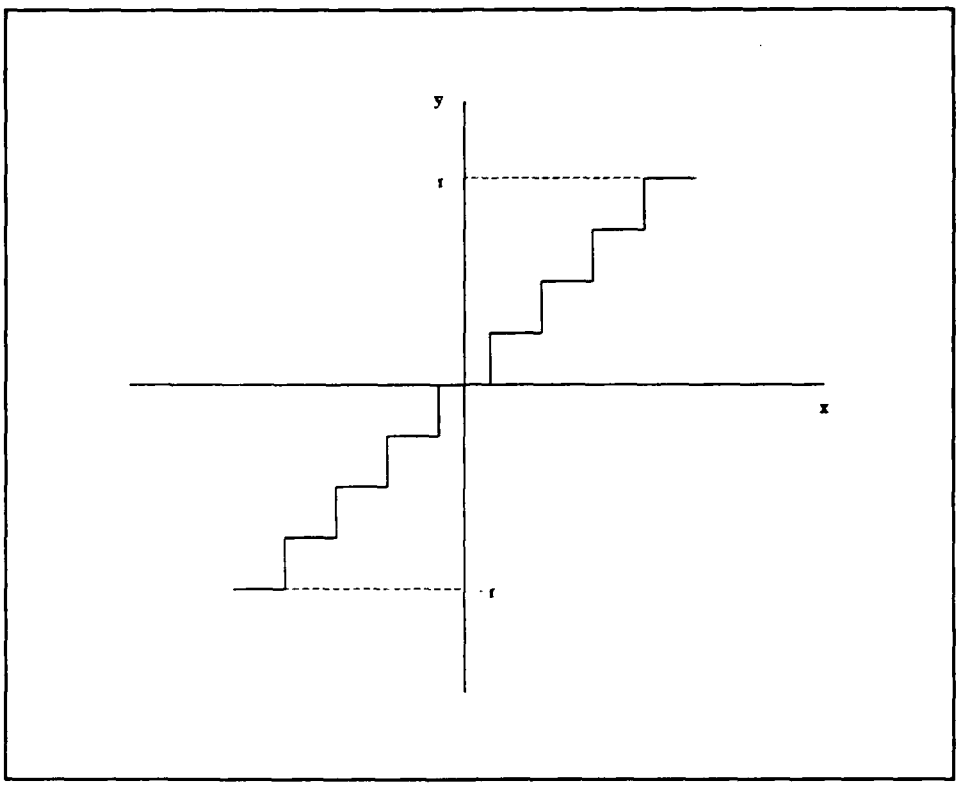


Figure 3.6. Phase Quantizer

Speech synthesizer

This subsystem reconstructs speech from the three frames containing frequencies, and quantized amplitudes and phases of length L generated by the speech analyzer. The reconstruction is based on the fact that speech can be represented as a sum of sinusoidal waveforms (8:489). After conversion to the time domain, a data reduction algorithm is used to restore the original number of frames. Recall that the initial windowing process has expanded the number of speech frames from n to 2n-1. The following discussion analyses the modules that constitute the synthesizer. These modules are depicted in Figure 3.5.

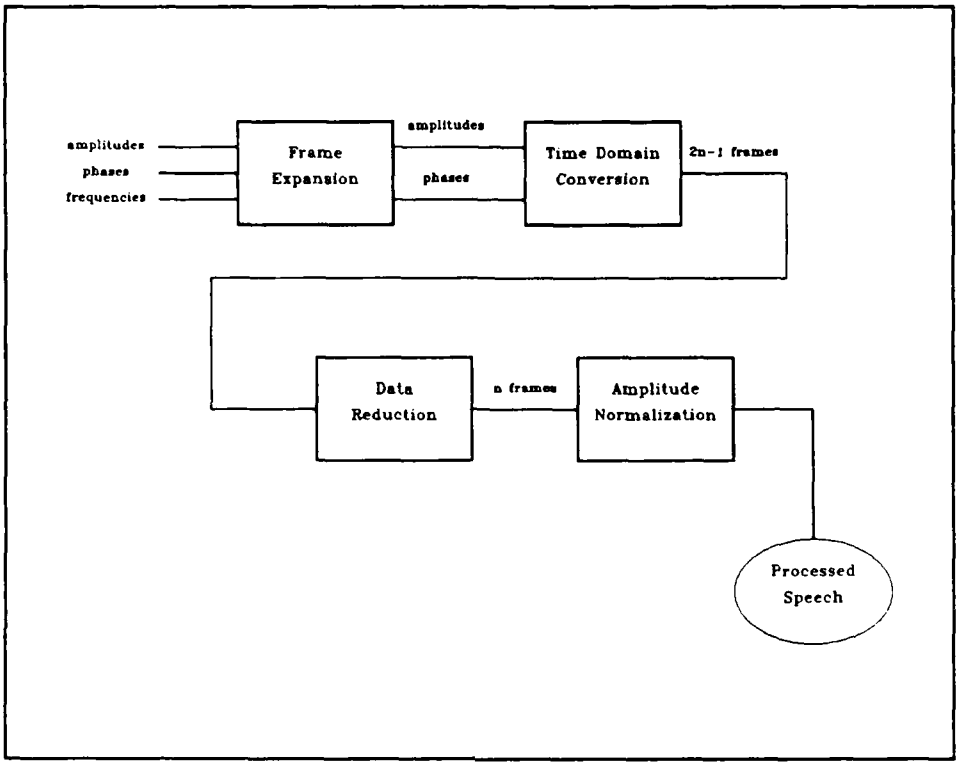


Figure 3.7. Speech synthesizer

Frame expansion. This module expands the reduced amplitude and phase frames to the original size. By making use of the frequency frame, the amplitudes and phases are written in the proper locations of two 256-point arrays.

Time Domain Conversion. The expanded amplitude and phase frame are used to perform the time domain conversion, according to the following expression:

$$s(t) = \Sigma_i \; a_i \cos(2\pi f_i t + \phi_i) \qquad (3.4)$$

where

a_i = value of i^{th} location of amplitude frame

f_i = frequency correspondent to the i^{th} location

ϕ_i = value of the i^{th} location of the phase frame

Data Reduction. This module makes use of local memory, to reduce the number of speech frames, from the value produced by 50% overlapping windows, to the original value n. A Hamming window is applied to all input frames, then if the frame being processed is odd, its first 128 locations are changed to the value of their sum with the last 128 locations of the preceding (and already Hamming windowed) even frame. The resultant frame is stored in local memory, waiting for the next incoming even frame. When this frame arrives, the last 128 positions of the stored frame, are changed to the value of its sum with the first 128 positions of the even incoming frame. Then the stored frame is outputed. The following figure illustrates this process.

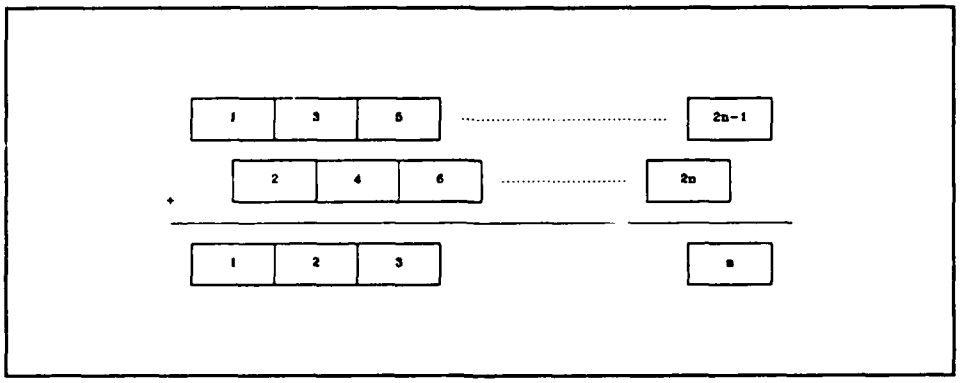


Figure 3.8. Data reduction

Amplitude Normalization. Here the incoming speech frames from the Data Reduction module, are written to a file untill the system as finished processing its input. Then, this file is amplitude normalized to drive the DSC-200 and produce listenable speech. The normalization process is essentially the same as used by Bashir in his mater's thesis. The speech amplitudes are multiplied by 32767, and divided by an estimated maximum amplitude value. This carries out the conversion to integer*2 data type needed to drive the digital/analog converter, and makes the output volume independent of the input level (2:3-14).

IV. Results

The search for the lowest data rate possible with this system began with the determination of the best algorithm to look for harmonics in the voiced regions. Then the effects on the reconstructed waveform of changing either the number of quantization levels, or the frame length were observed. The results obtained by this process were mainly determined by subjective listening tests even though the visualization of narrow band spectograms and time waveforms of reconstructed and original speech had also played a preliminary role in that determination. The system was tested for different input speech files, but the results presented were all obtained with the input file SND. The following sections analyze each step of the above process.

Harmonic Search Algorithm

The different algorithms tested were based on variations of the scanning mechanism to find harmonics, and on the number of selected components in the vicinity of the harmonic. Two different searching methods were used. The first, is based on an estimation of the pitch frequency (150 Hz), and uses this value to jump to the location of the next harmonic in the amplitude frame. The other evaluates the pitch frequency (also, the jump value) in a neighborhood of 150 Hz thus allowing variations on the pitch frequency from

frame to frame. In this case, the pitch value corresponds to the frequency of the highest amplitude in the neighborhood. The neighborhood was set to 2, 4 or 6 neighbors of the harmonic, in both cases. Figures 4-1 to 4-6, present the results obtained by the two mechanisms, using constant values for frame length (64), number of amplitude bits (12), and number of phase bits (12). Figures 4-1 and 4-2, show the original (SND) and reconstructed waveforms using fixed pitch (2NFP, 4NFP, 6NFP). Figure 4-3, shows the corresponding narrow-band spectograms. The reconstructed files obtained with variable pitch (2NVP, 4NVP, 6NVP) are shown in Figures 4-4 to 4-6. The best reconstruction of speech was considered to be done by the fixed pitch algorithm with 4 neighbors. This configuration was then used in the determination of the number of quantization levels and the frame length, that yield the lowest possible data rate for quality speech transmission.

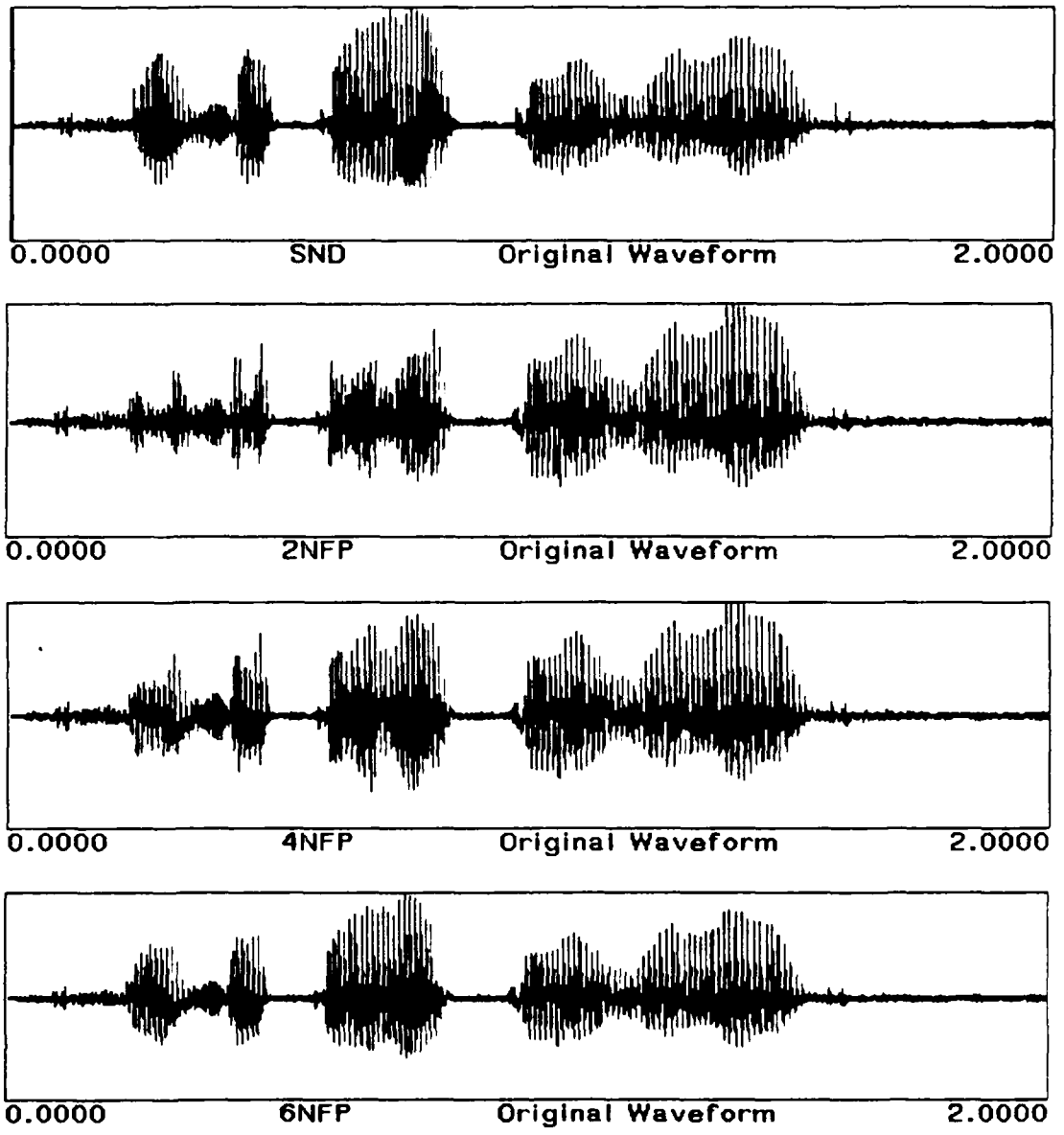


Figure 4.1. Time waveforms. Original waveform SND, and reconstructed waveforms using fixed pitch. The number of neighbors was set to two (2NFP), four (4NFP), and six (6NFP).

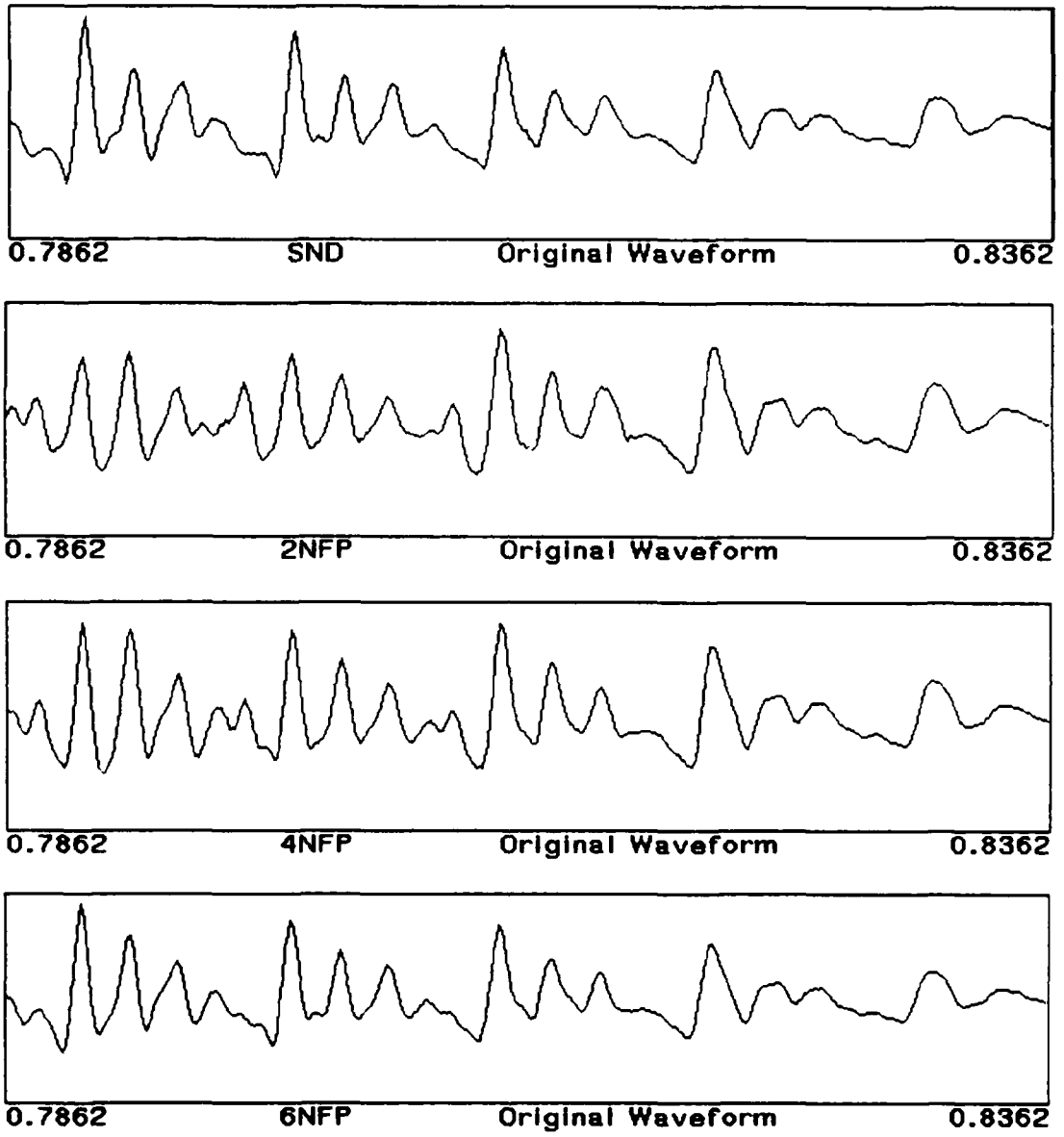


Figure 4.2. Detailed Time Waveforms. Same as last figure, but with increased detail.

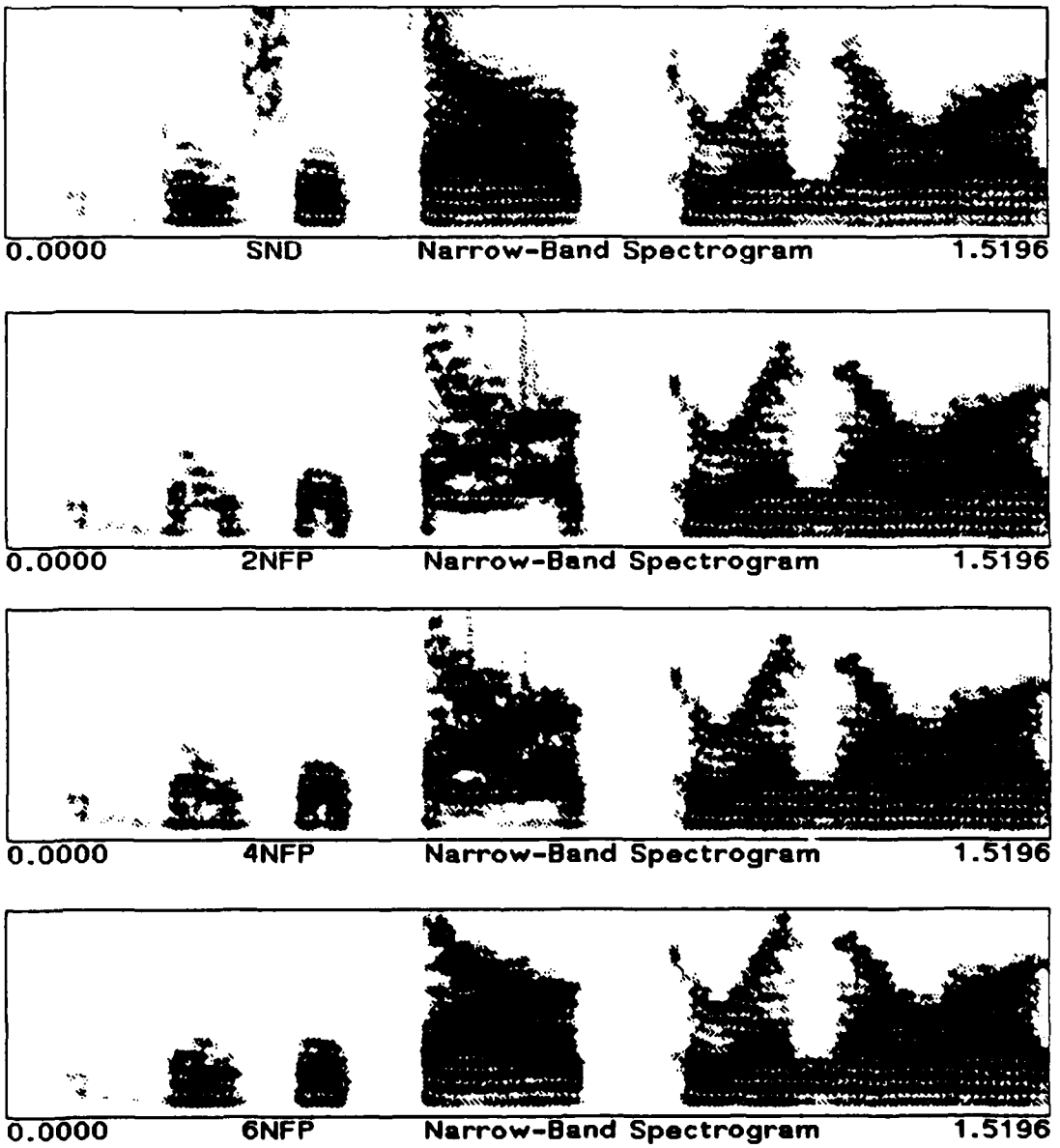


Figure 4.3. Narrow-band Spectrograms.These spectrograms correspond to the waveforms shown in Figure 4.1.

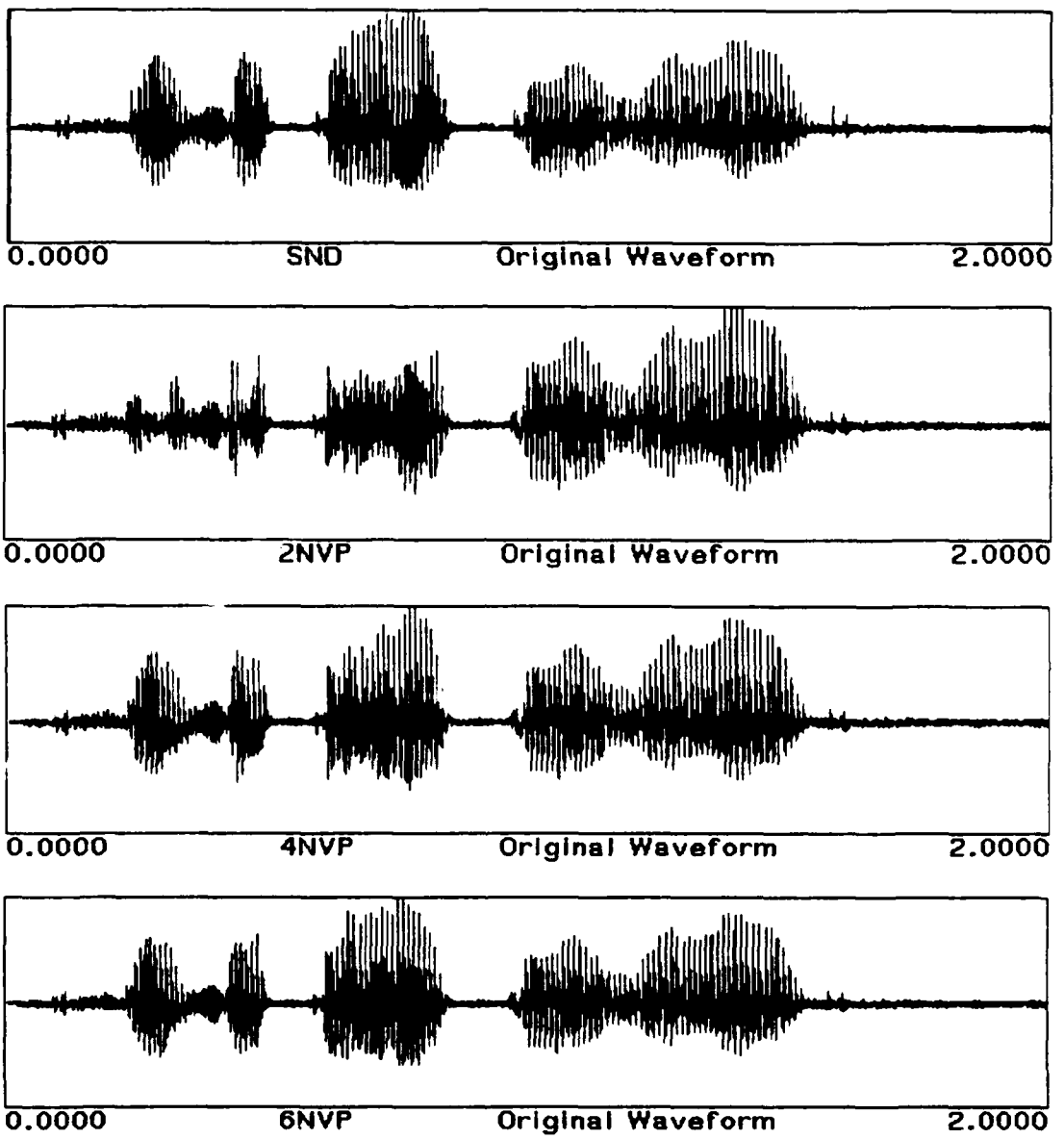


Figure 4.4. Time Waveforms. Original waveform SND, and reconstructed waveforms using variable pitch. The number of neighbors was set to two (2NVP), four (4NVP), and six (6NVP).

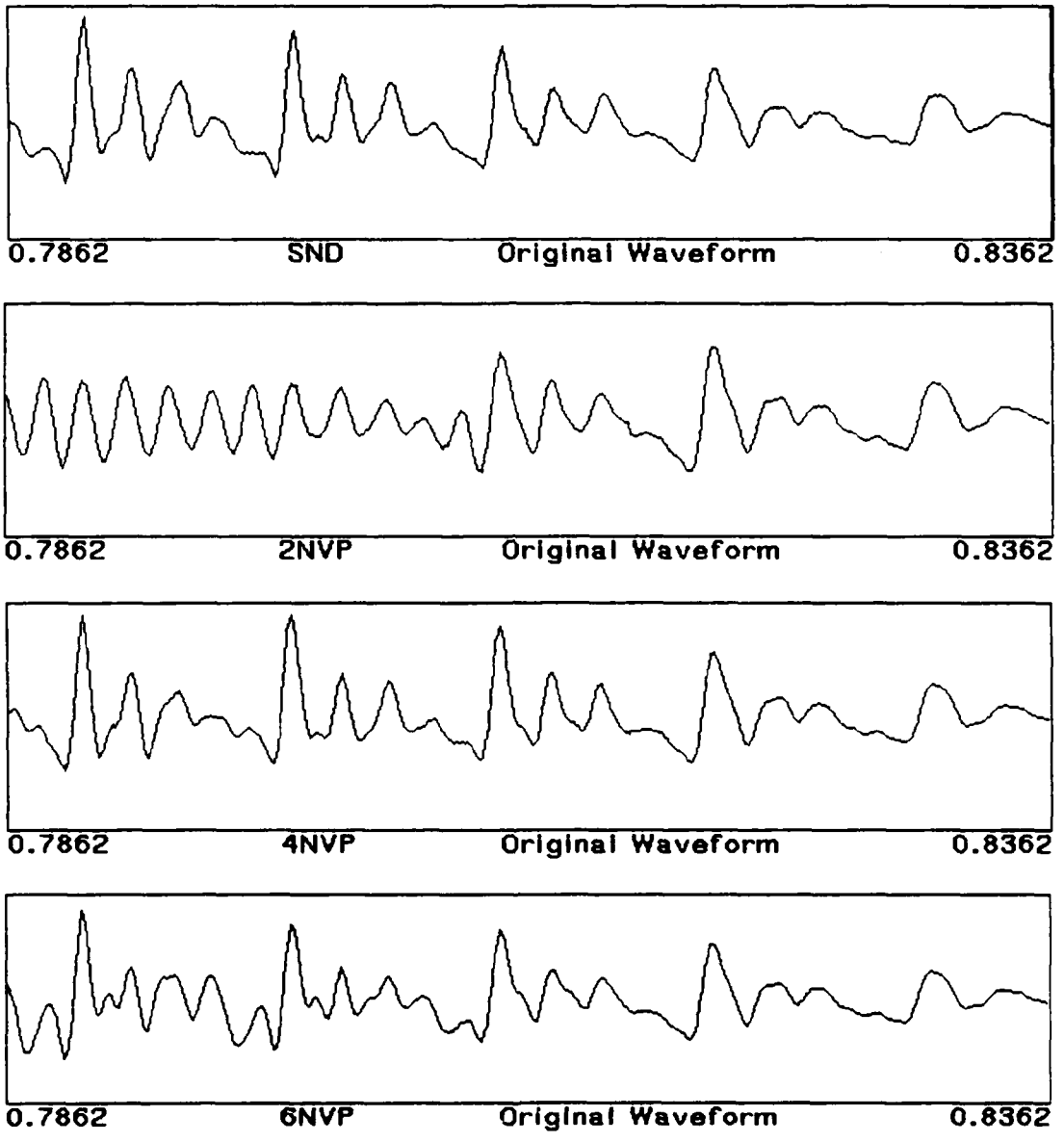


Figure 4.5. Detailed Time Waveforms. Same as last figure but with increased detail.

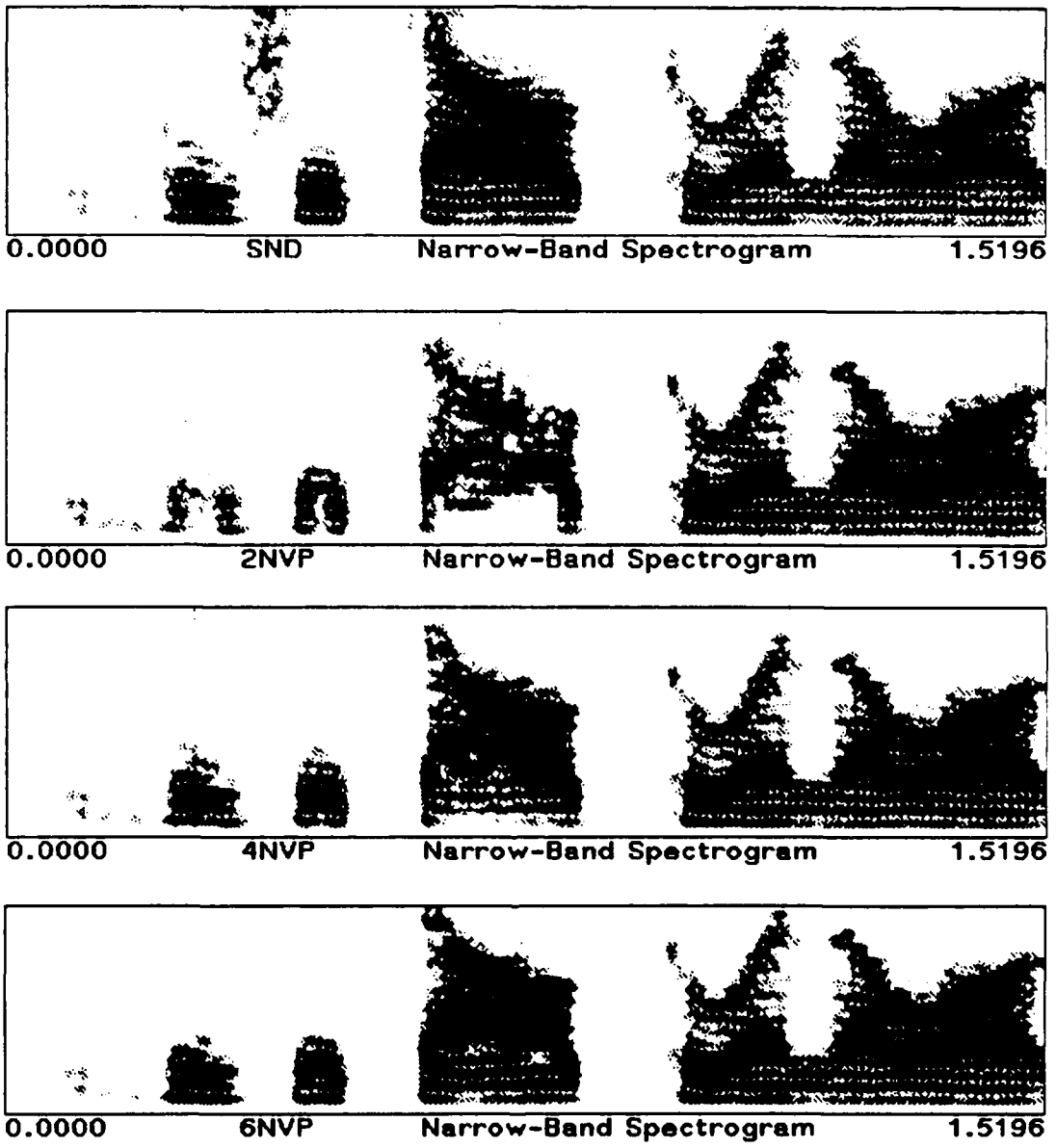


Figure 4.6. Narrow-band Spectrograms.These spectrograms correspond to the waveforms shown in Figure 4.4.

Quantization Levels and Frame Length

The effects of the quantization levels and the frame length on the data rate is expressed in equation 3.1, through the number of amplitude bits (A), phase bits (P), and frame length (L). The minimum data rate, for which the reconstructed waveform produced acceptable listening tests, was obtained with six amplitude bits, two phase bits, and eighteen length frame, yielding a data rate of 18 Kbps. This waveform (4NFP18L6A2P) is shown in comparison with the original waveform (SND) in Figures 4.7 and 4.8. Figures 4.9 to 4.20, show intermediate results, obtained by varying one of the above parameters while maintaining constant the others. When the fixed parameter was A or P its value was set to 12, and to 64 in case of the frame length L. Figures 4.9 to 4.11, present the effect of the number of amplitude quantization bits. Figures 4.12 to 4.14, show the effect of the number of phase bits. Figures 4.15 to 4.20 show the effect of the length of the frame. Appendix A and B expand these results, by showing waveforms and narrow-band spectograms of reconstructed speech files, for different numbers of amplitude bits (8, 6, 4, 2), different numbers of phase bits (6, 4, 2), and different frame lengths (32, 24, 22, 20, 18, 16, 12).

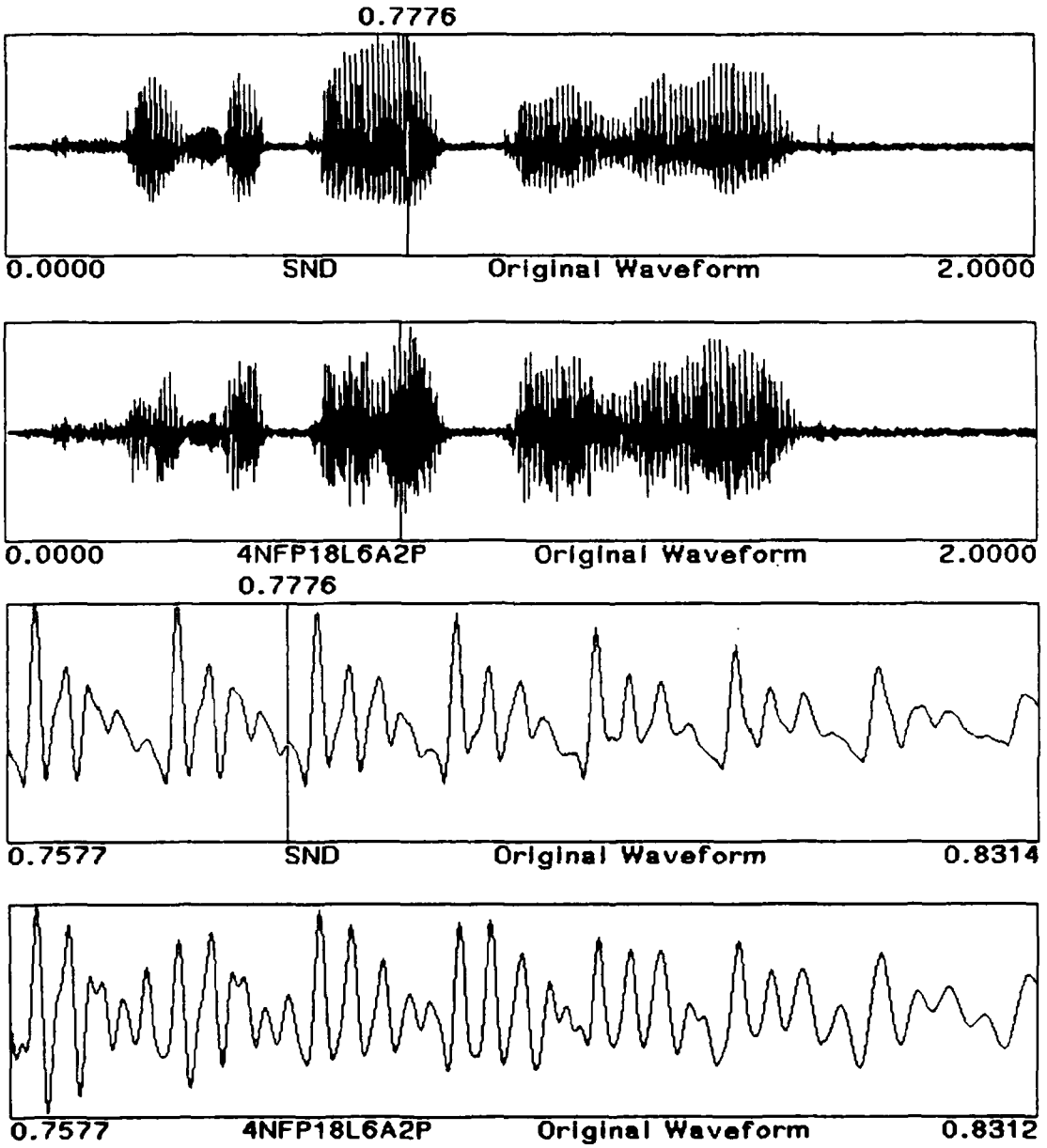


Figure 4.7. Time Waveforms. Comparison between original speech (SND) and reconstructed waveform with L=18, A=6, and P=2 (4NFP18L6A2P). This parameters yield a data rate of 18 Kbps.

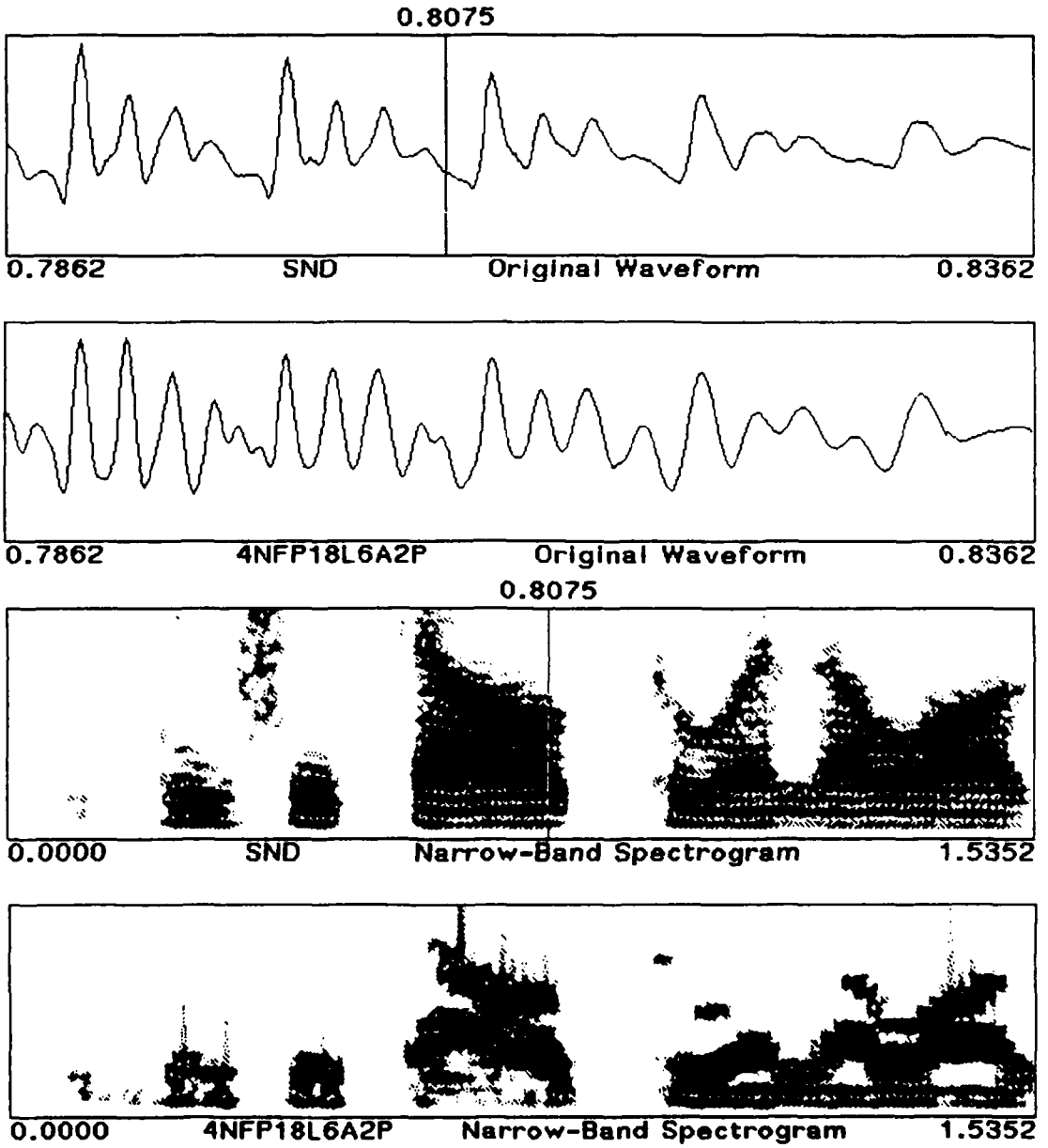


Figure 4.8. Time Waveforms and Narrowband Spectrograms. Same as last figure with another detail of the time waveform, and narrow-band spectrograms.

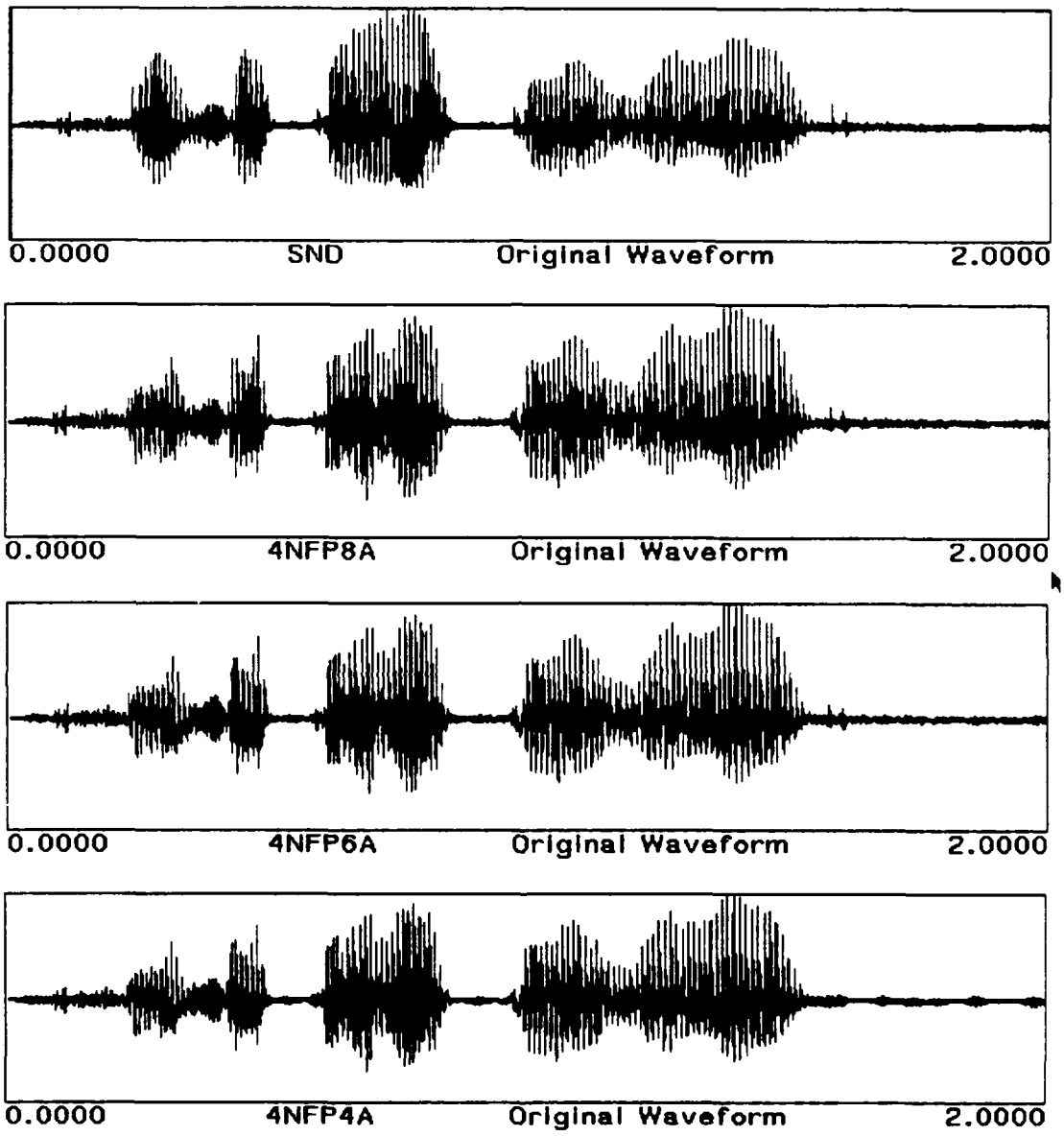


Figure 4.9. Time Waveforms. Comparison between input speech (SND) and reconstructed waveforms with A=8, 6, and 4.

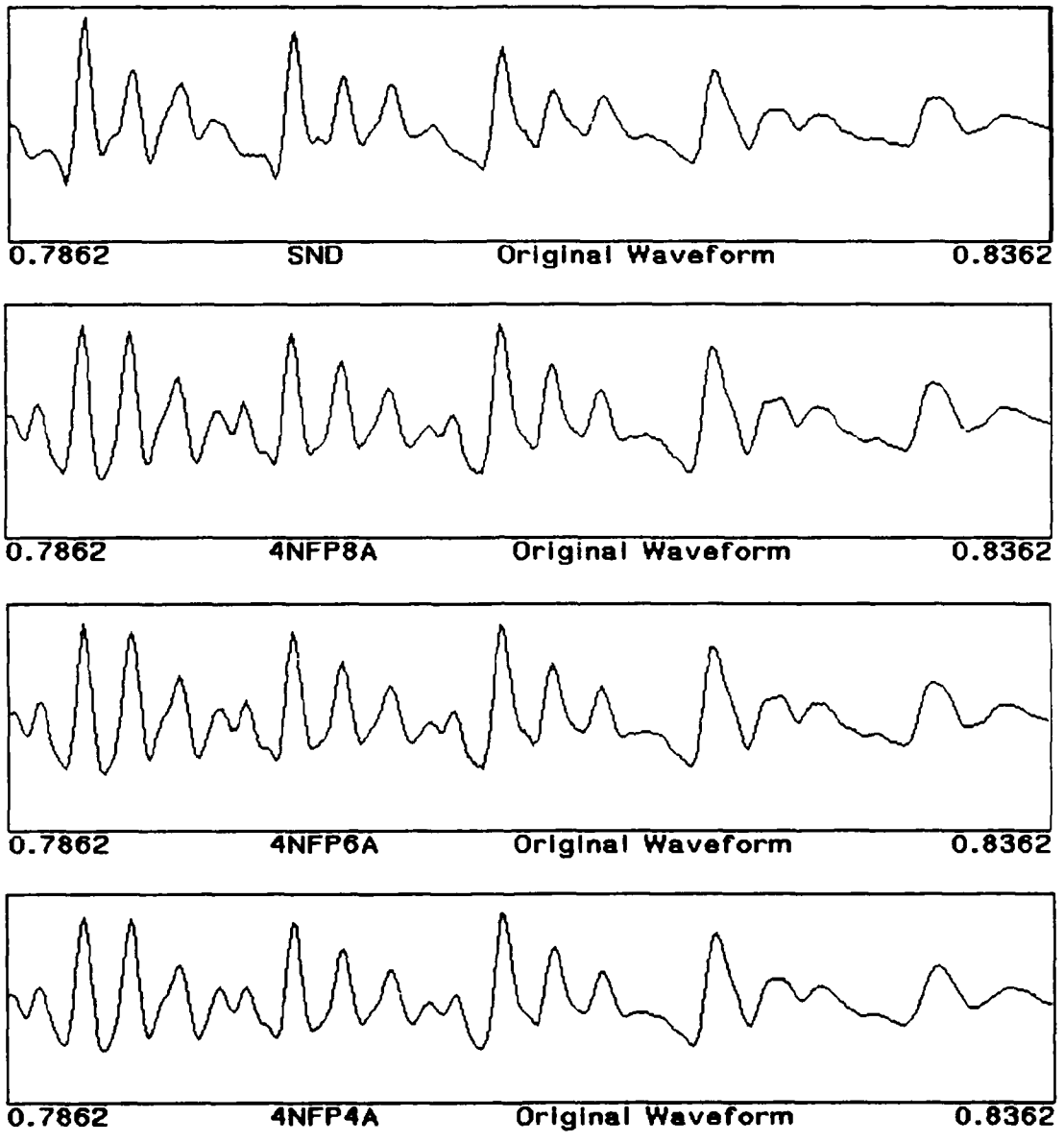


Figure 4.10. Detailed Time Waveforms. Same as last figure with increased detail.

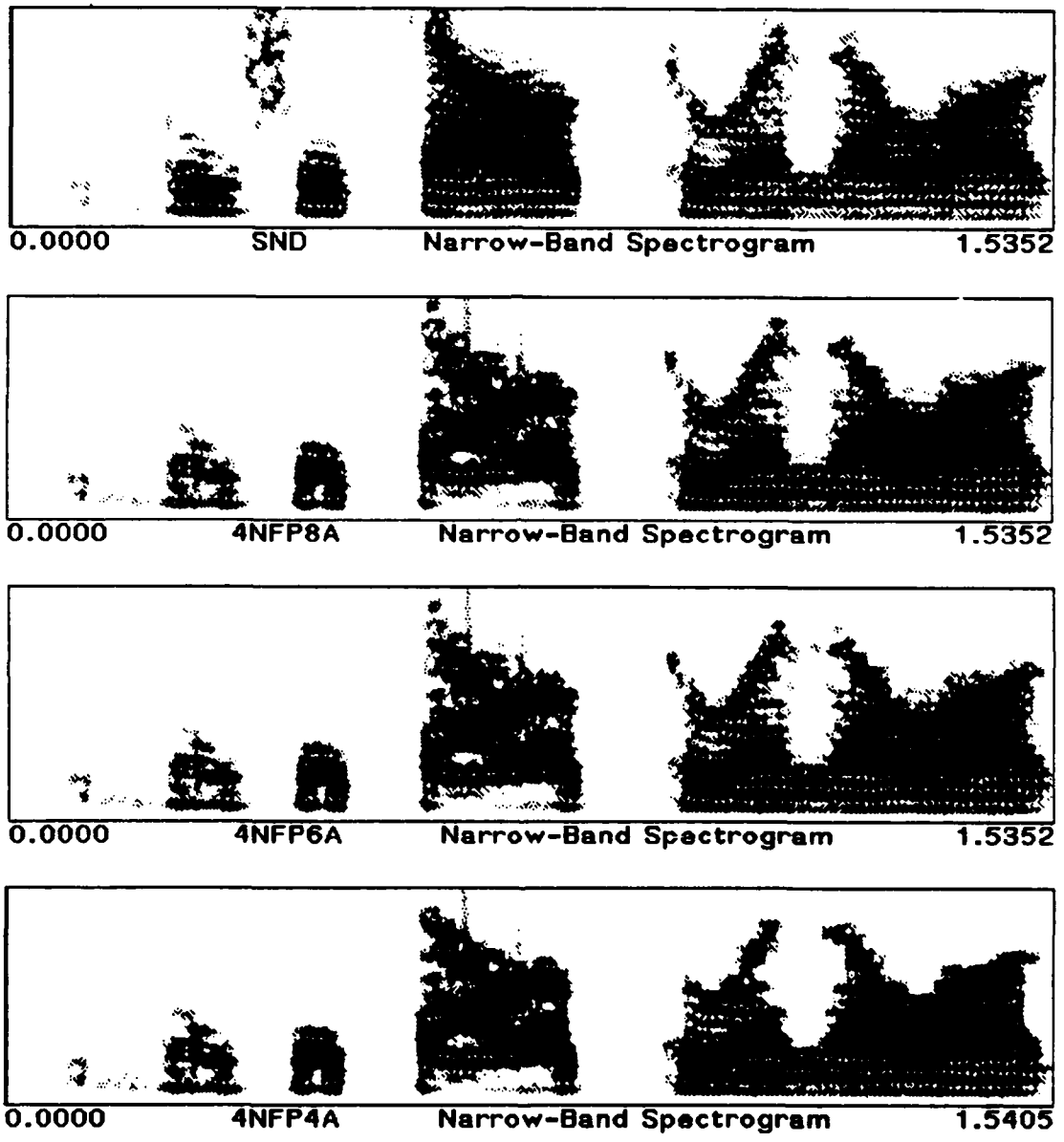


Figure 4.11. Narrow-band Spectrograms. These spectrograms correspond to the waveforms of Figure 4.9.

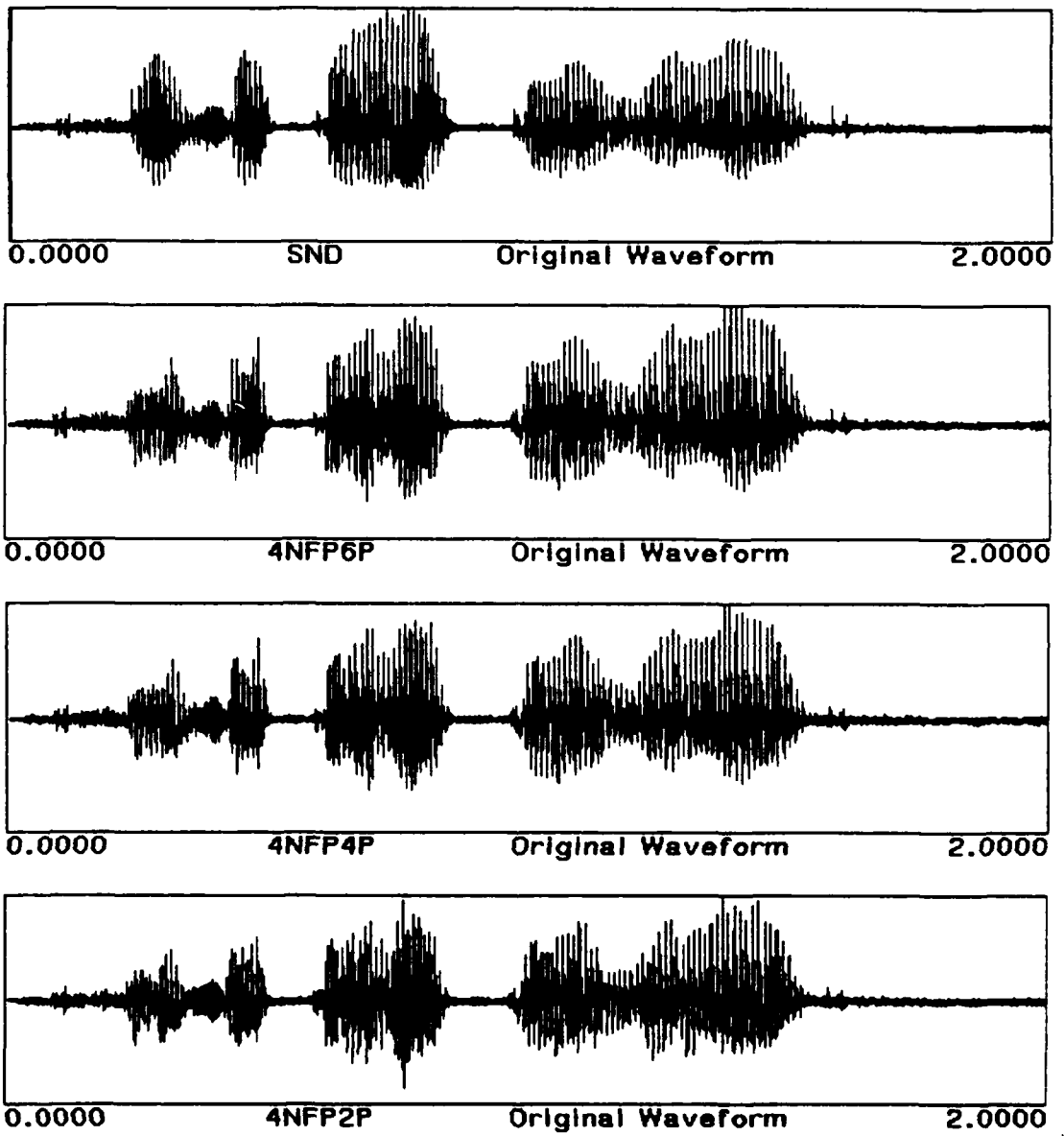


Figure 4.12. Time Waveforms. Comparison between input speech (SND) and reconstructed waveforms using P=6, 4, and 2.

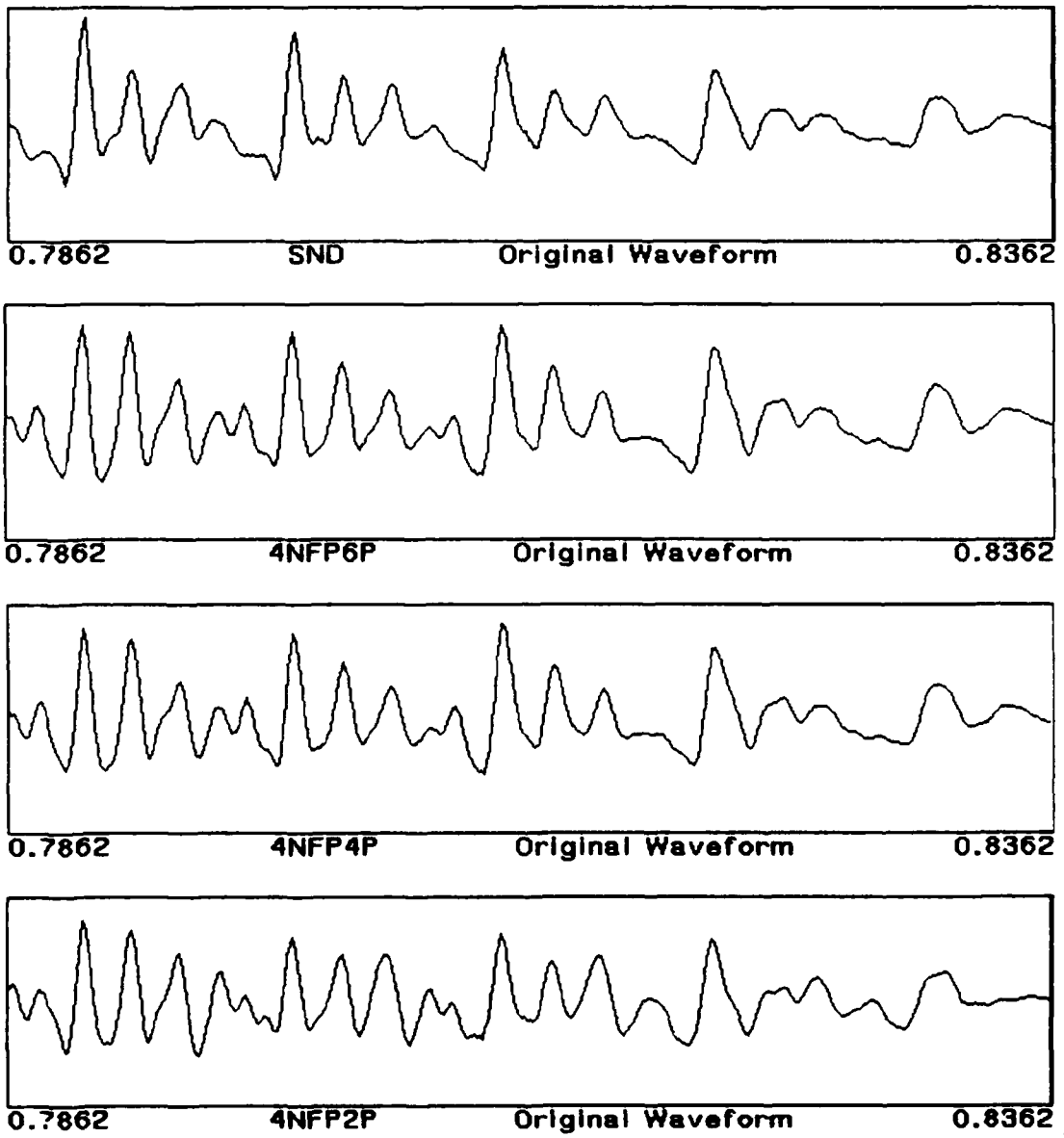


Figure 4.13. Detailed Time Waveforms. Same as last figure with increased detail.

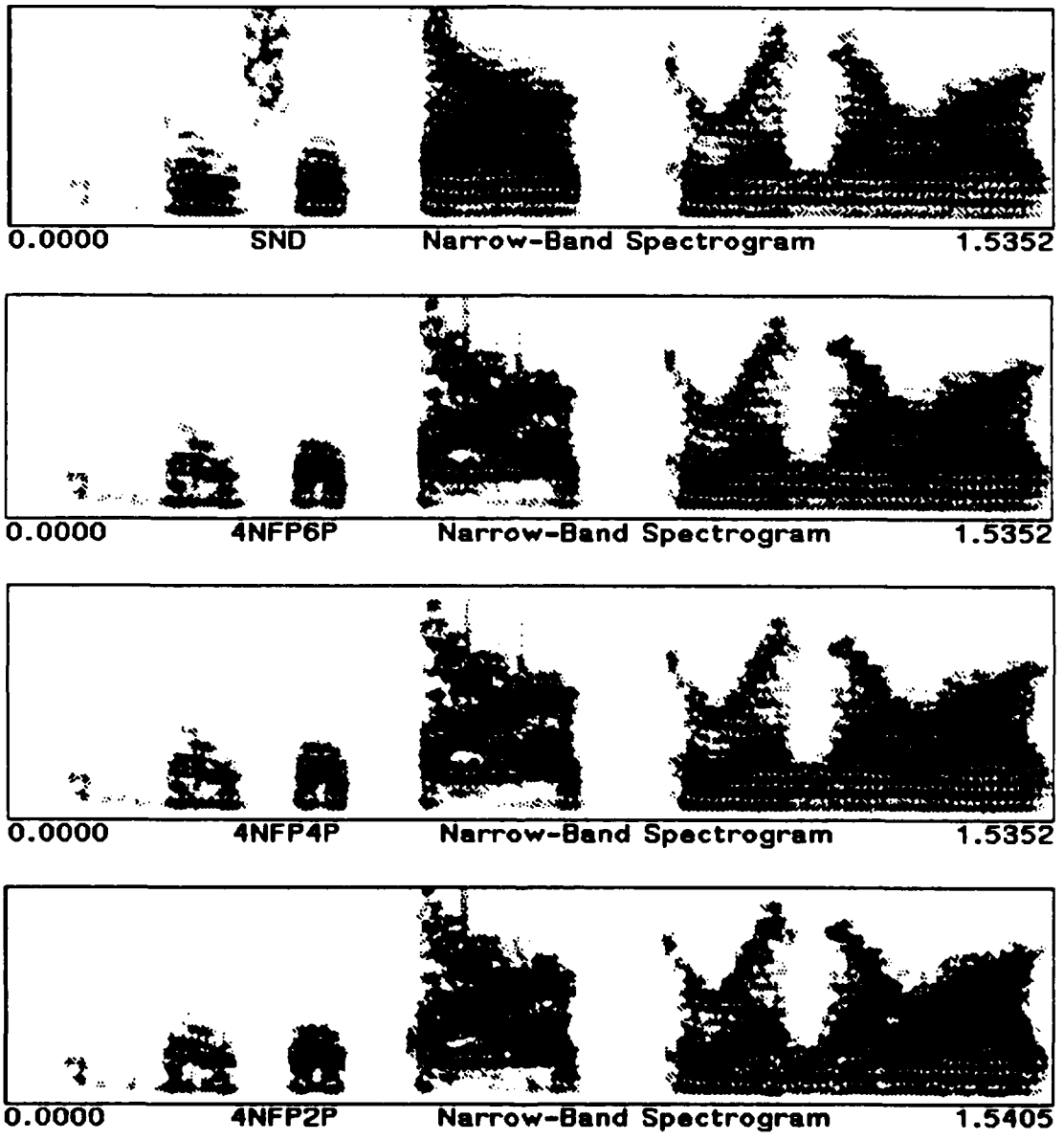


Figure 4.14. Narrow-band Spectrograms. These spectrograms correspond to the waveforms of Figure 4.12.

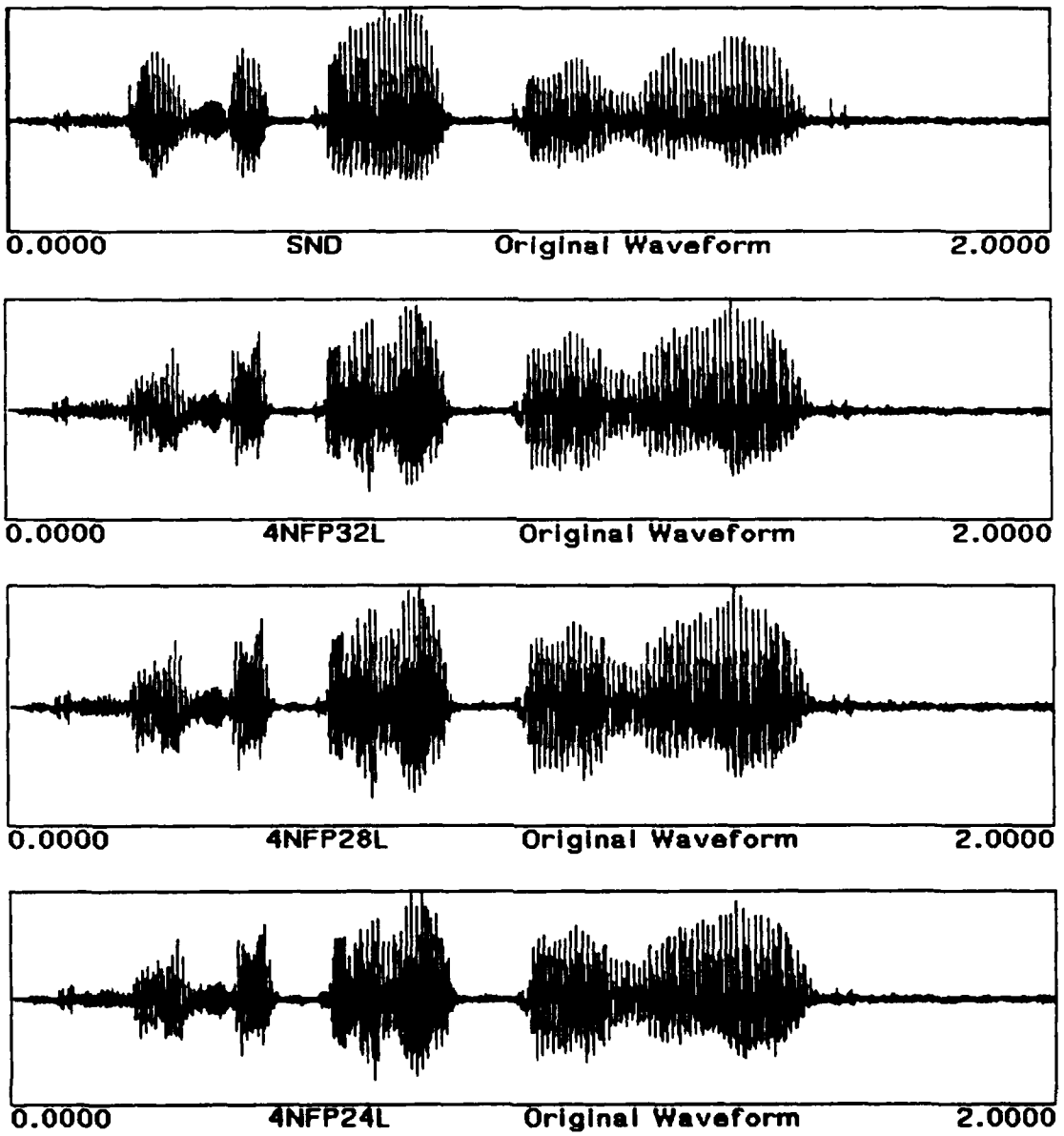


Figure 4.15. Time Waveforms. Comparison with input speech (SND) and reconstructed waveforms with L=32, 28, and 24.

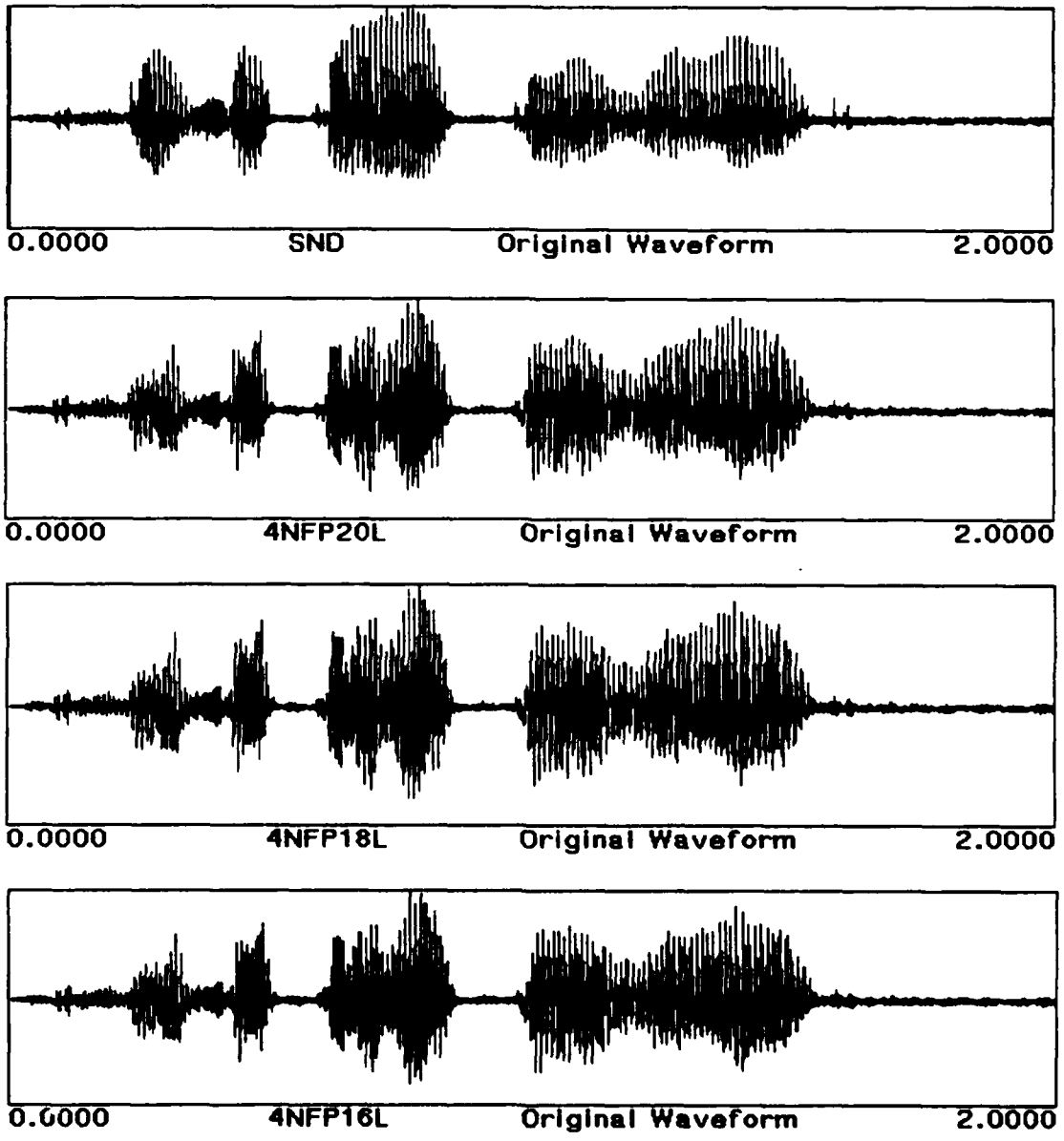


Figure 4.16. Time Wveforms. Same as last figure for L=20, 18, and 16.

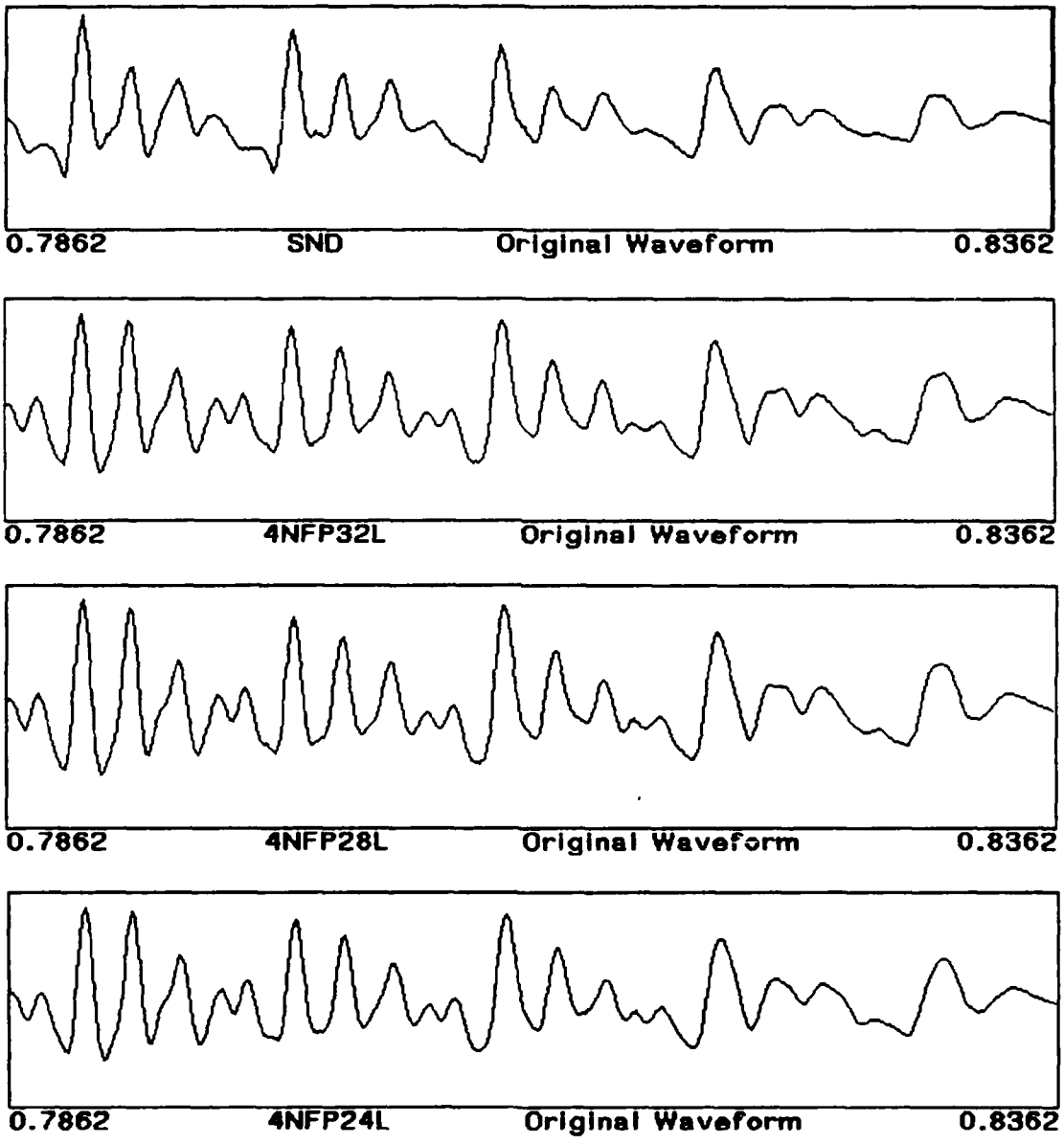


Figure 4.17. Detailed Timewaveforms. Detail of Figure 4.15.

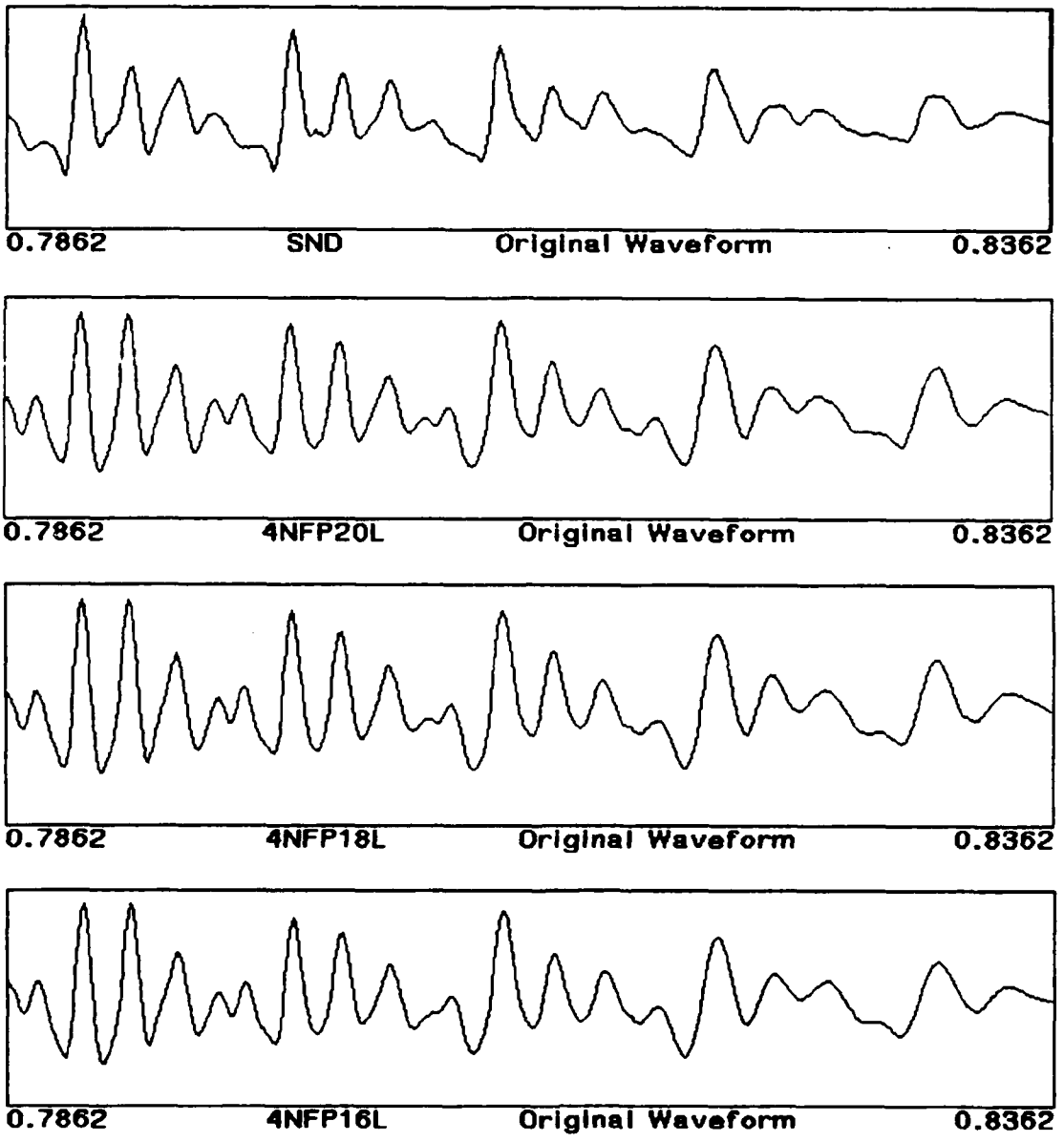


Figure 4.18. Detailed Time Waveforms. Detail of Figure 4.16.

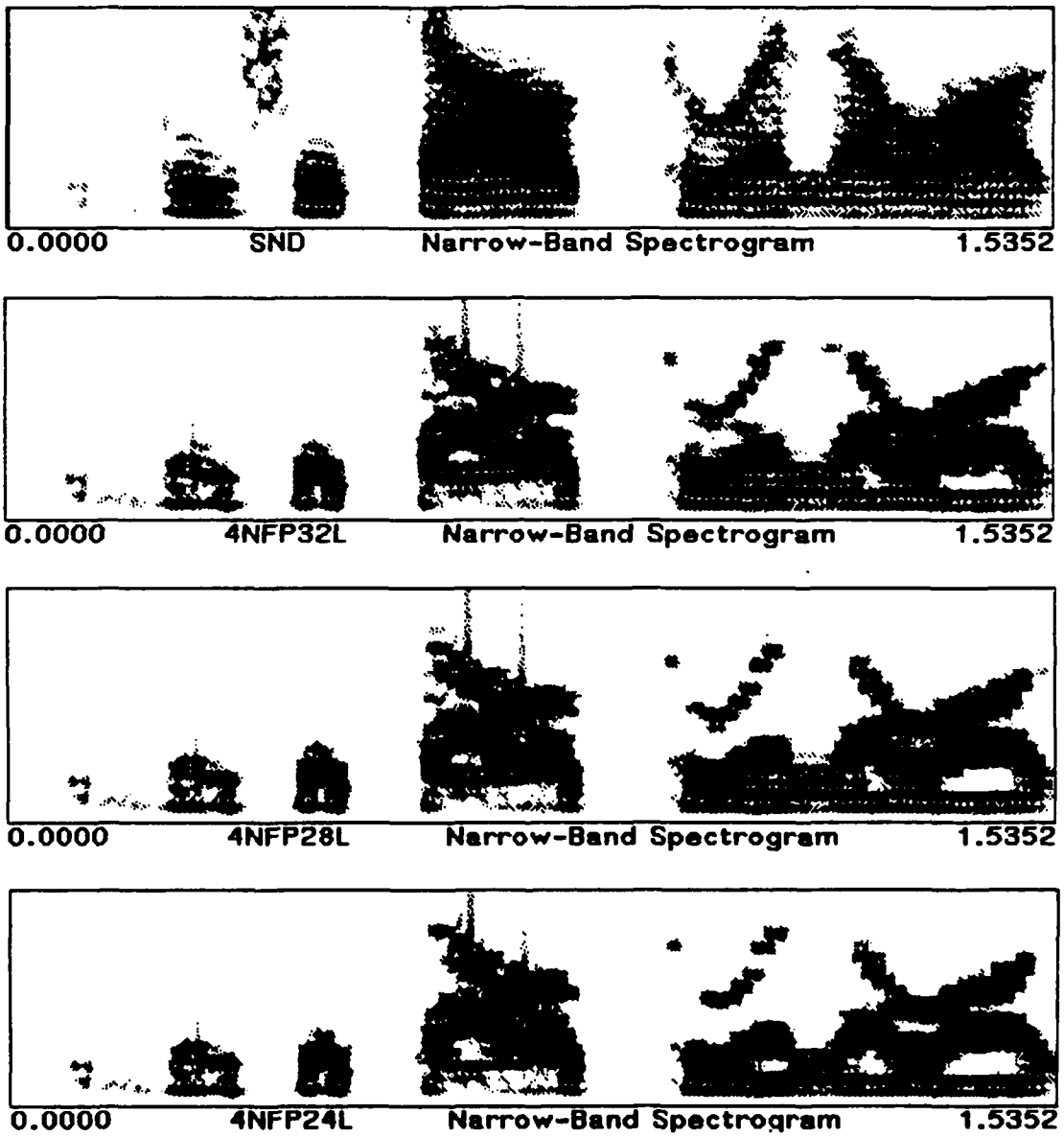


Figure 4.19. Narrow-band spectrograms. These spectrograms correspond to the waveforms of Figure 4.15.

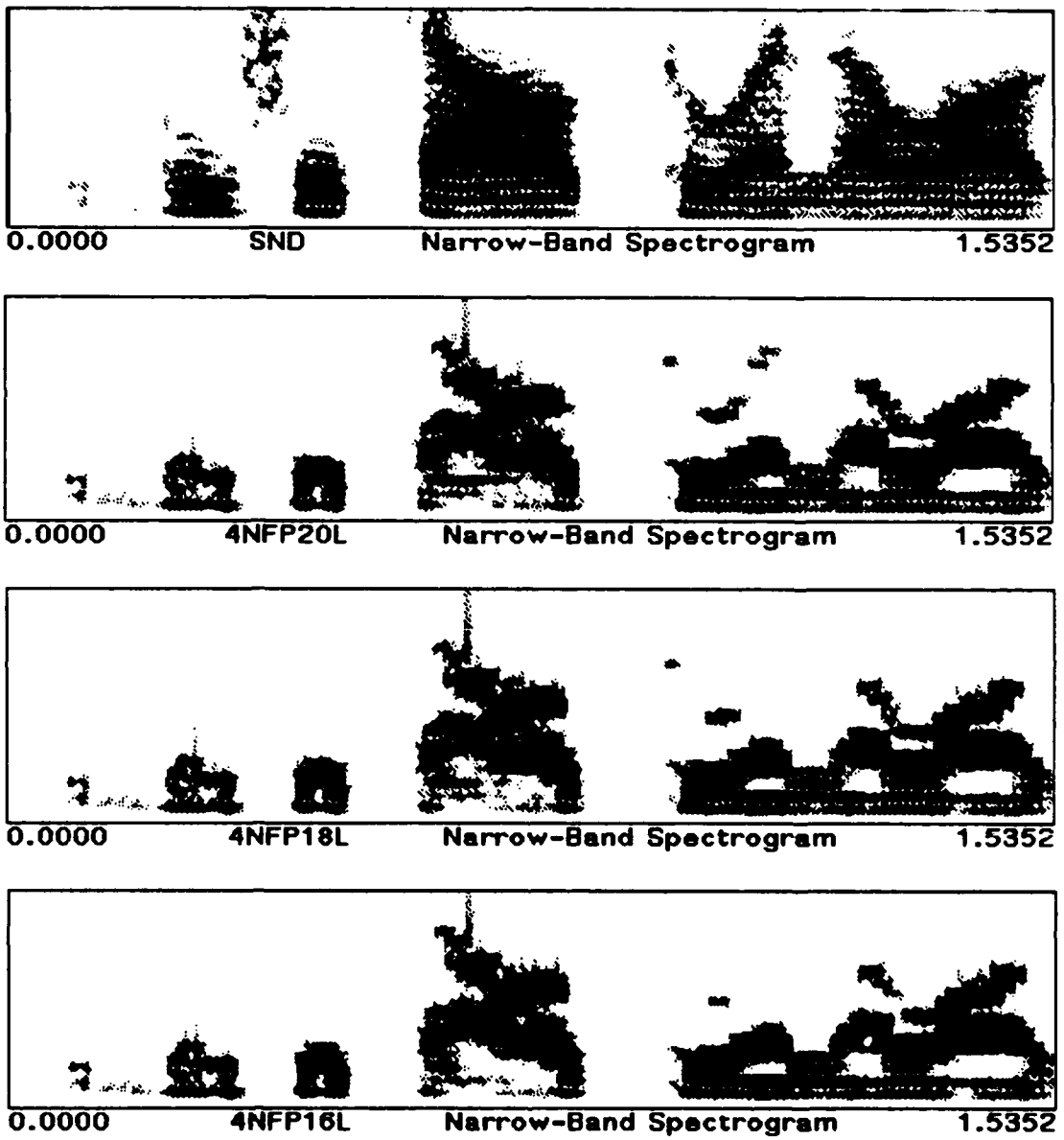


Figure 4.20. Narrow-band Spectrograms. These spectrograms correspond to the waveforms of Figure 4.16.

V. Conclusions and Recommendations

The purpose of this chapter is to draw conclusions about the performance of this system and give recommendations for further research in this area.

Conclusions

This thesis showed that it is possible to use a sinusoidal speech model, with ruled based spectral selection, to obtain relative low data rates (18 Kbps). The minimization of the data rate is accomplished by minimizing the three parameters: number of spectral components that characterize speech (i.e. frame length), number of amplitude quantization bits, and number of phase quantization bits. The parameter with most effect on the data rate is the frame length (see expression 3.1). The use of frame lengths lower than eighteen, generated a "birdie" noise on the reconstructed speech. On the other hand, the output of the system with respect to the other two parameters didn't show audible degradation for amplitude bits down to six, and phase bits down to two.

A side result from this encoding/decoding process is the elimination of non-harmonic noise from speech, because speech is synthesized from a selection of harmonics of the fundamental frequency (5).

Recommendations

The "birdie" noise in the output for small frame lengths, suggests the first recommendation: a process for expanding the spectral content of the received frames, should be investigated. Further work can also be done in the area of the statistics of the speech spectral amplitudes and phases, to improve their respective quantizers.

Appendix A: Amplitude Quantization Bits - Sample Values

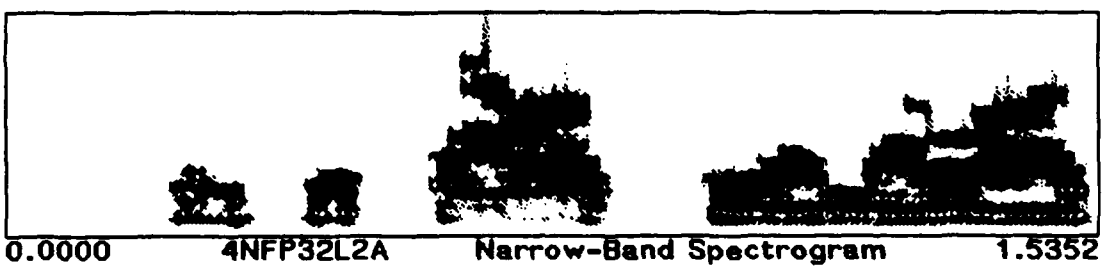
This Appendix shows the effects of the number of amplitude quantization bits for several frame lengths. To suppress the effects of the number of phase quantization bits, this number was set to a high value (12 bits). The code that appears bellow the pictures has the following meaning:

Example: 4NFP18L8A
4N - 4 Neighbors
FP - Fixed Pitch
18L - Frame Length is 18
8A - Number of Amplitude Quantization Bits is 8

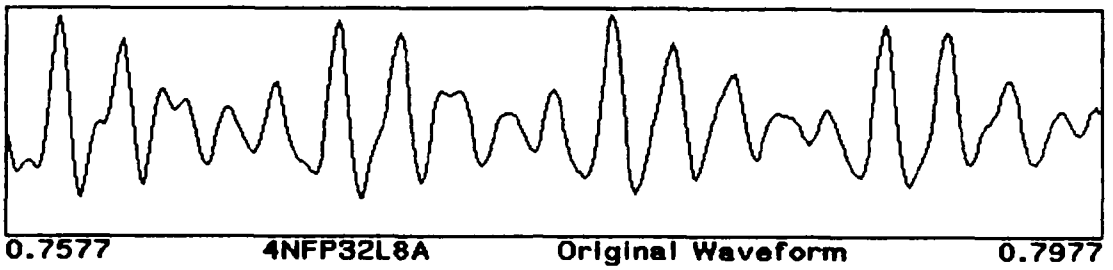
0.0000 4NFP32L8A Narrow-Band Spectrogram 1.5352

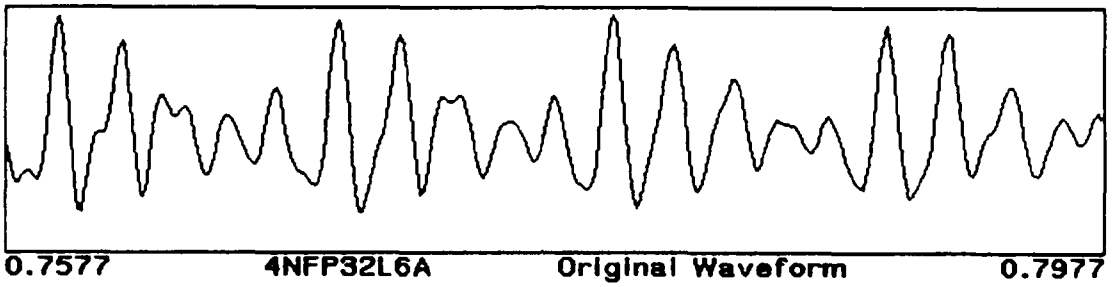
0.0000 4NFP32L6A Narrow-Band Spectrogram 1.5352

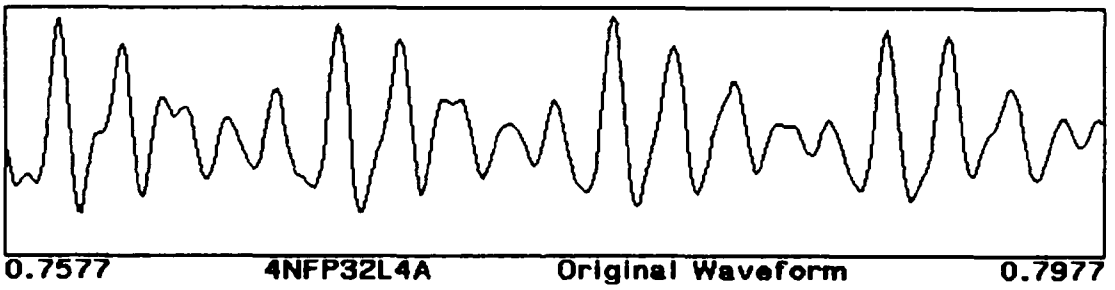
0.0000 4NFP32L4A Narrow-Band Spectrogram 1.5352

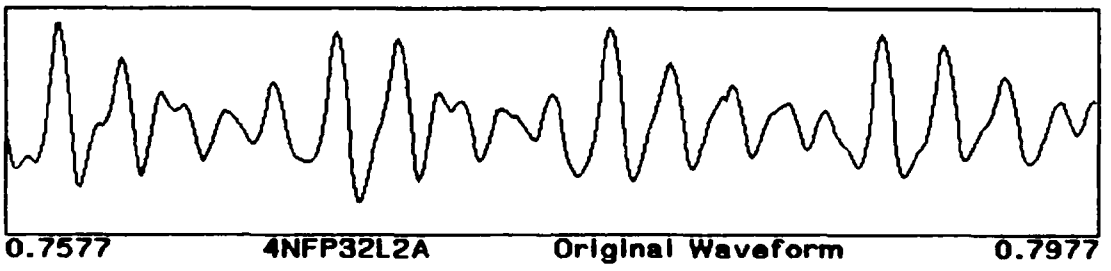


0.0000 4NFP32L2A Narrow-Band Spectrogram 1.5352


0.7577
4NFP32L8A
Original Waveform
0.7977


0.7577
4NFP32L6A
Original Waveform
0.7977

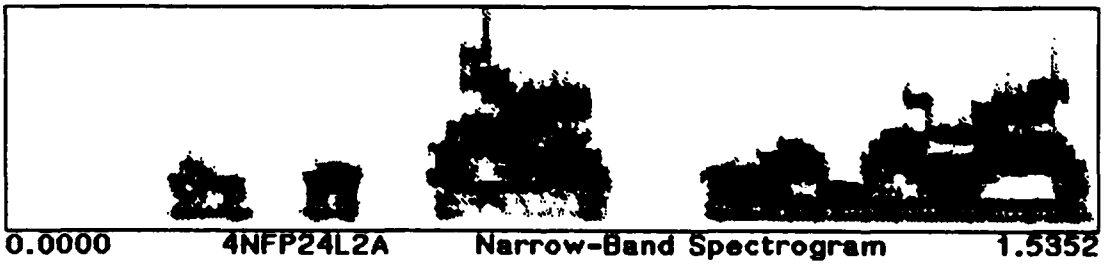

0.7577
4NFP32L4A
Original Waveform
0.7977

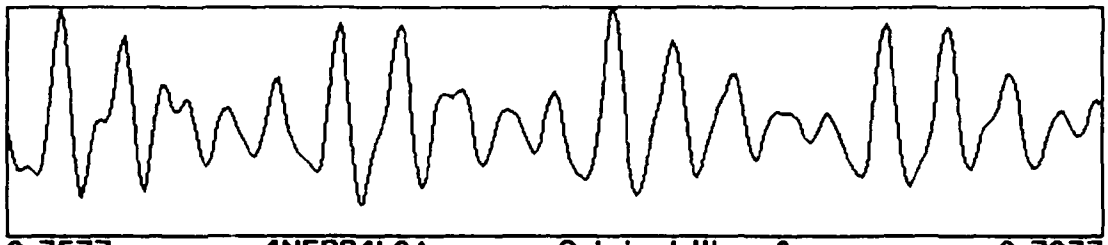

0.7577
4NFP32L2A
Original Waveform
0.7977

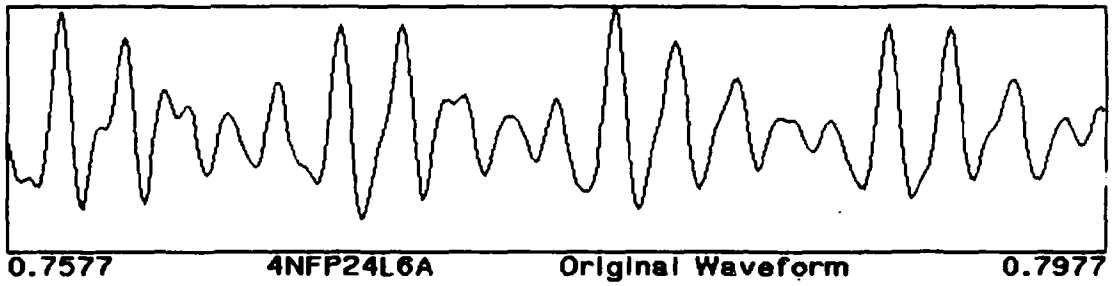

0.0000
4NFP24L8A
Narrow-Band Spectrogram
1.5352

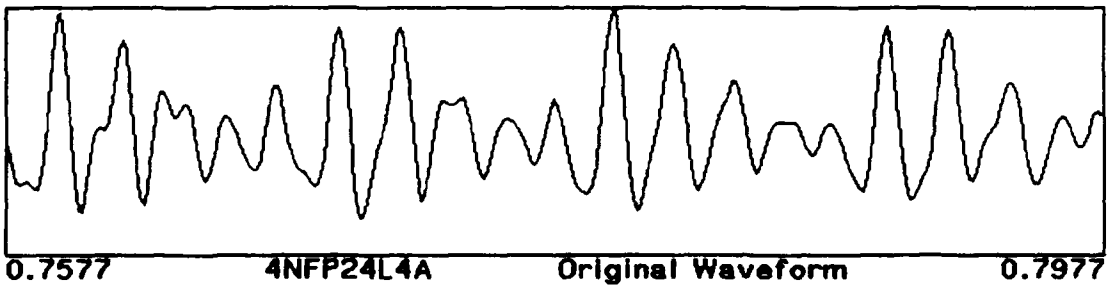

0.0000
4NFP24L6A
Narrow-Band Spectrogram
1.5352

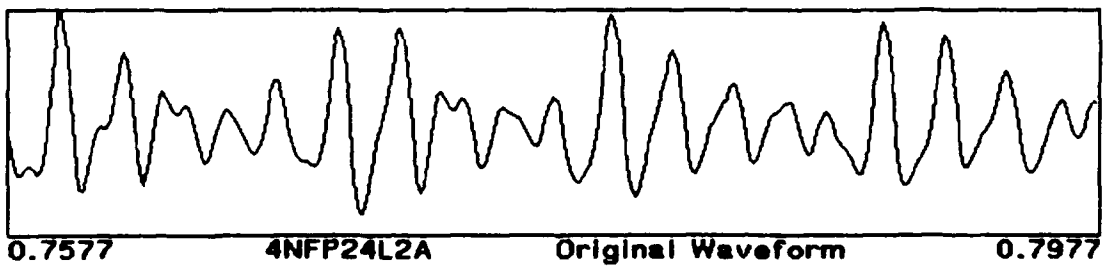

0.0000
4NFP24L4A
Narrow-Band Spectrogram
1.5352


0.0000
4NFP24L2A
Narrow-Band Spectrogram
1.5352

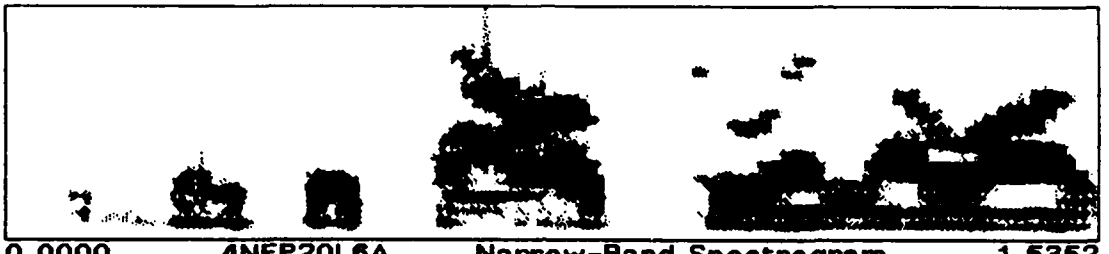

0.7577
4NFP24L8A
Original Waveform
0.7977


0.7577
4NFP24L6A
Original Waveform
0.7977

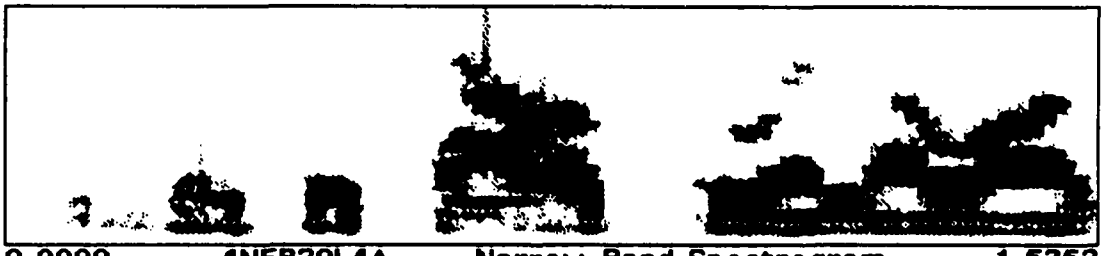

0.7577
4NFP24L4A
Original Waveform
0.7977


0.7577
4NFP24L2A
Original Waveform
0.7977

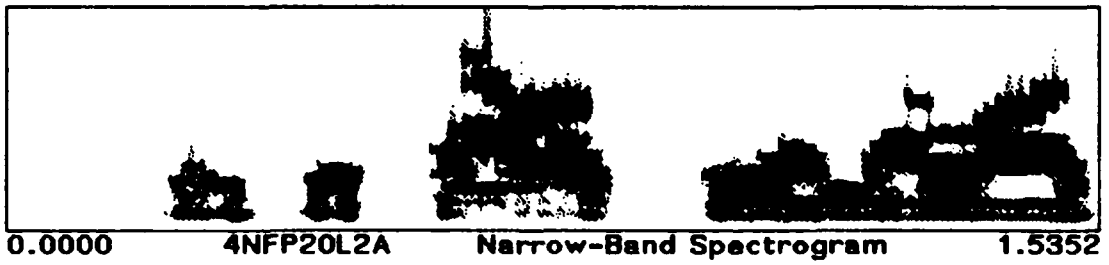
0.0000 4NFP20L8A Narrow-Band Spectrogram 1.5352



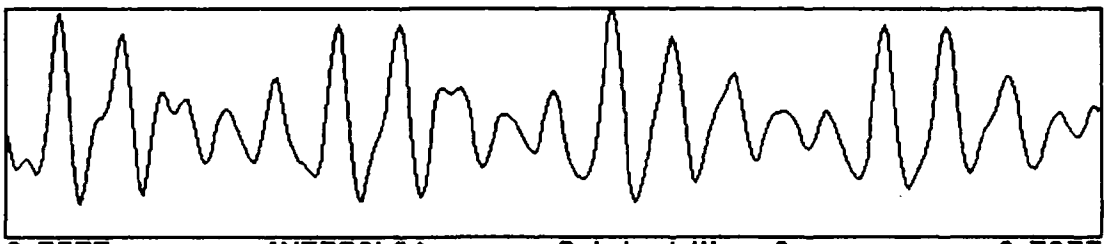
0.0000 4NFP20L6A Narrow-Band Spectrogram 1.5352

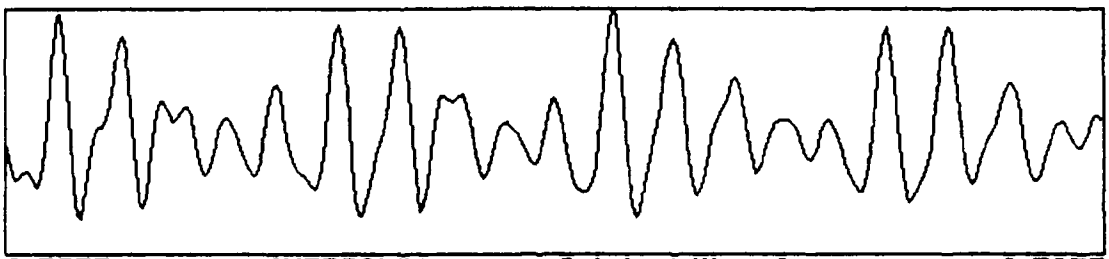


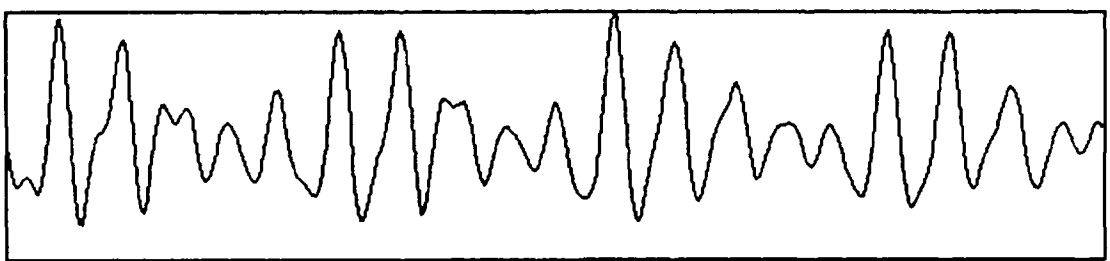
0.0000 4NFP20L4A Narrow-Band Spectrogram 1.5352

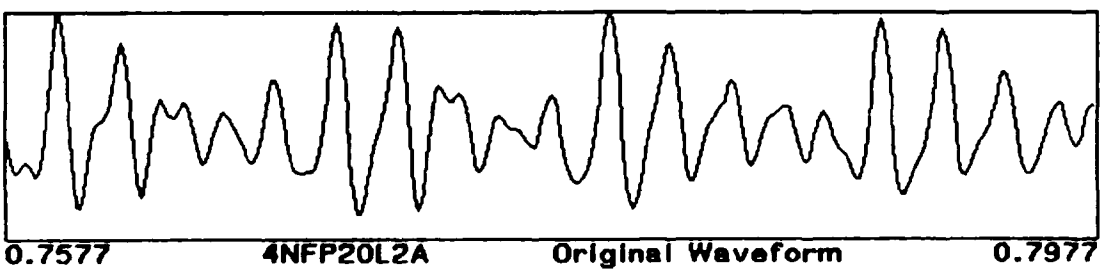


0.0000 4NFP20L2A Narrow-Band Spectrogram 1.5352

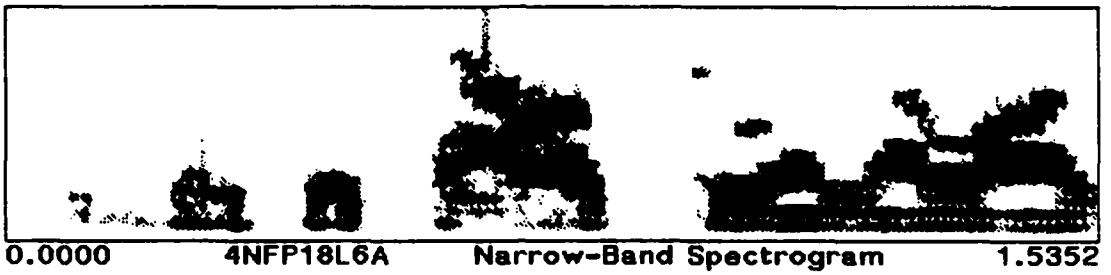

0.7577
4NFP20L8A
Original Waveform
0.7977


0.7577
4NFP20L6A
Original Waveform
0.7977

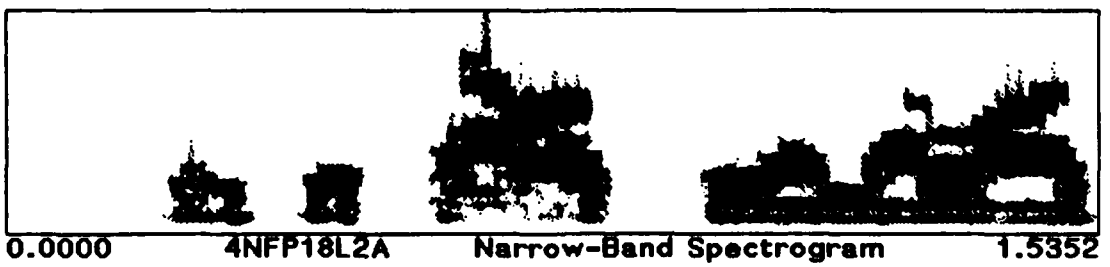

0.7577
4NFP20L4A
Original Waveform
0.7977

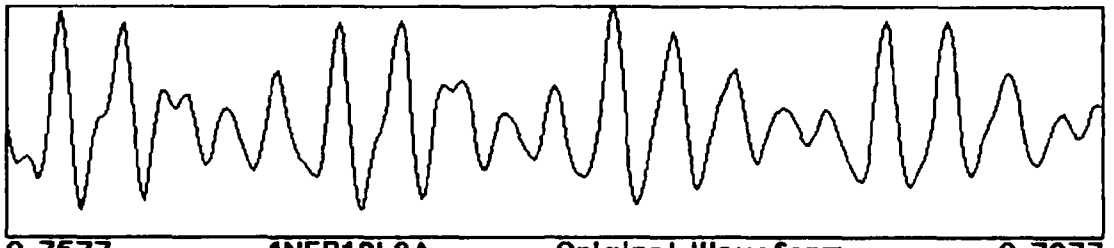

0.7577
4NFP20L2A
Original Waveform
0.7977

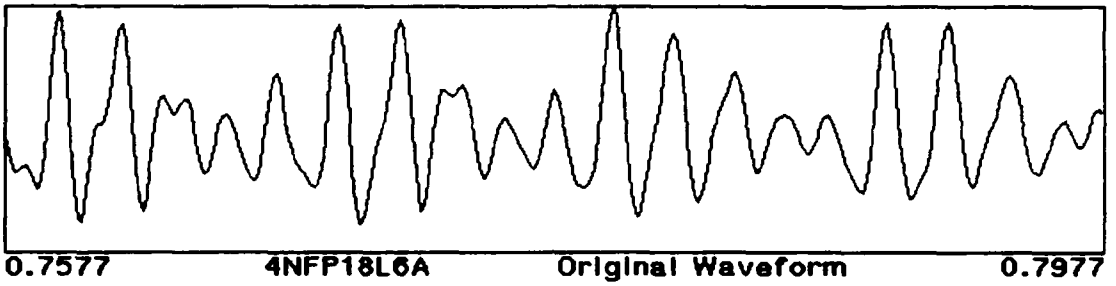

0.0000
4NFP18L8A
Narrow-Band Spectrogram
1.5352

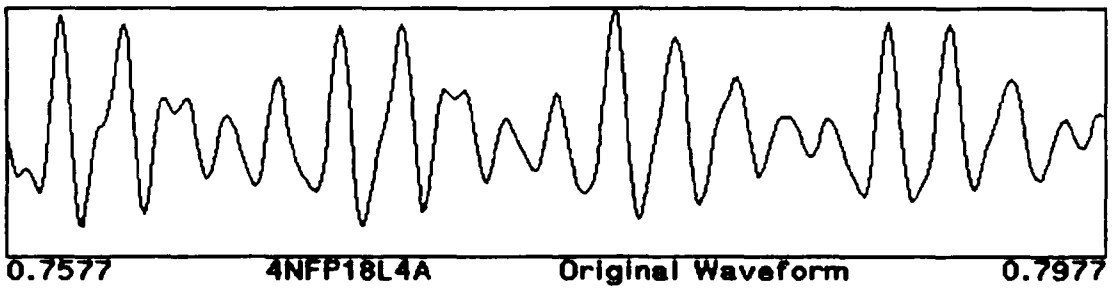

0.0000
4NFP18L6A
Narrow-Band Spectrogram
1.5352

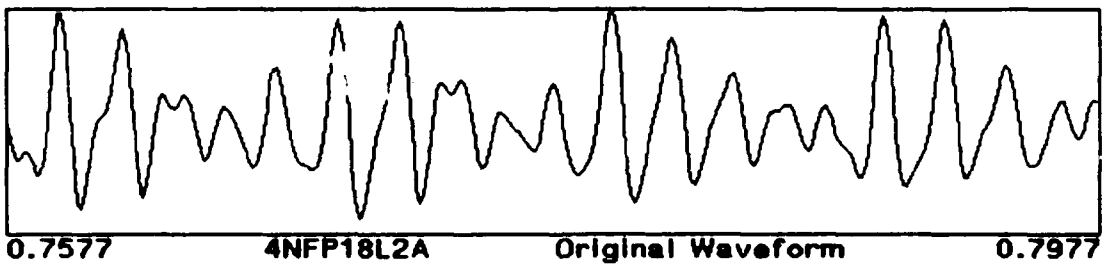

0.0000
4NFP18L4A
Narrow-Band Spectrogram
1.5352


0.0000
4NFP18L2A
Narrow-Band Spectrogram
1.5352

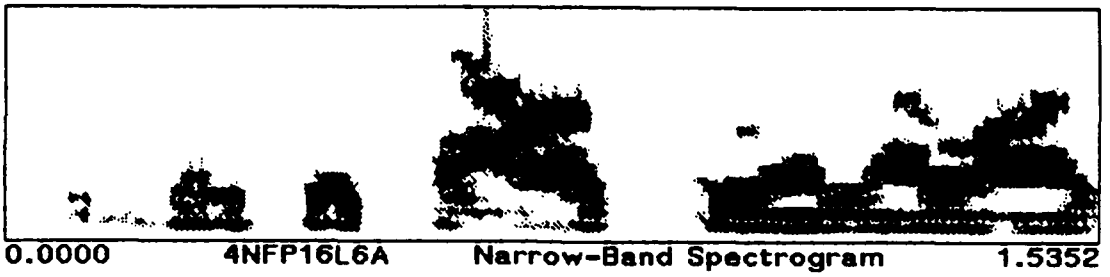

0.7577
4NFP18L8A
Original Waveform
0.7977

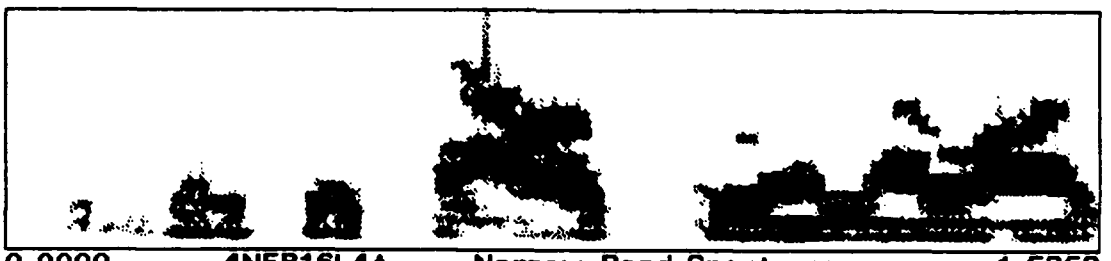

0.7577
4NFP18L6A
Original Waveform
0.7977

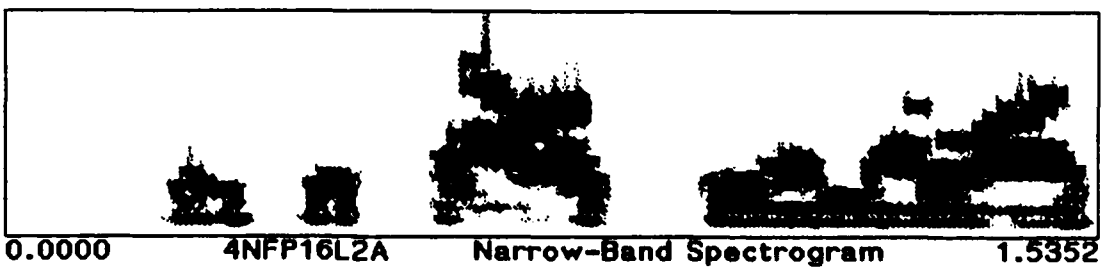

0.7577
4NFP18L4A
Original Waveform
0.7977

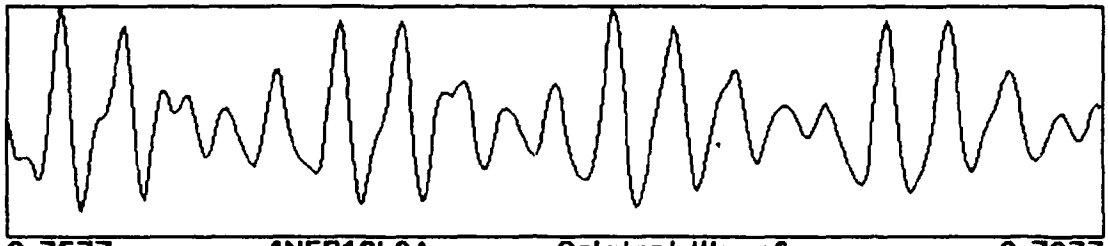

0.7577
4NFP18L2A
Original Waveform
0.7977

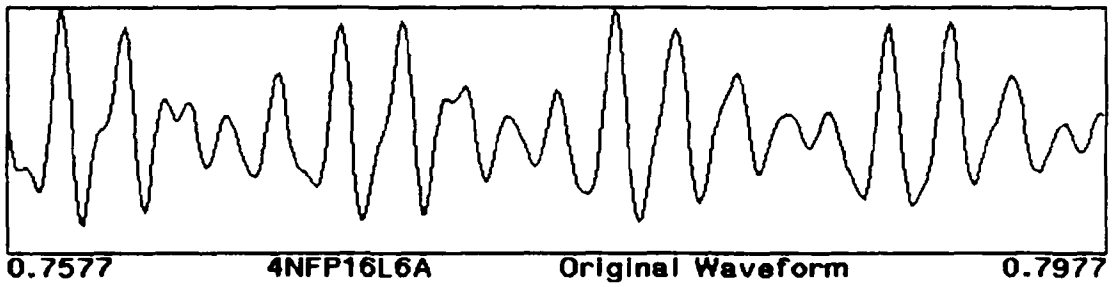

0.0000
4NFP16L8A
Narrow-Band Spectrogram
1.5352

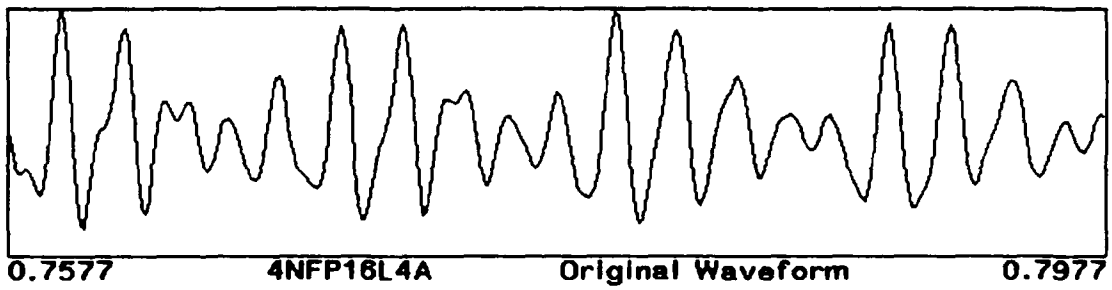

0.0000
4NFP16L6A
Narrow-Band Spectrogram
1.5352

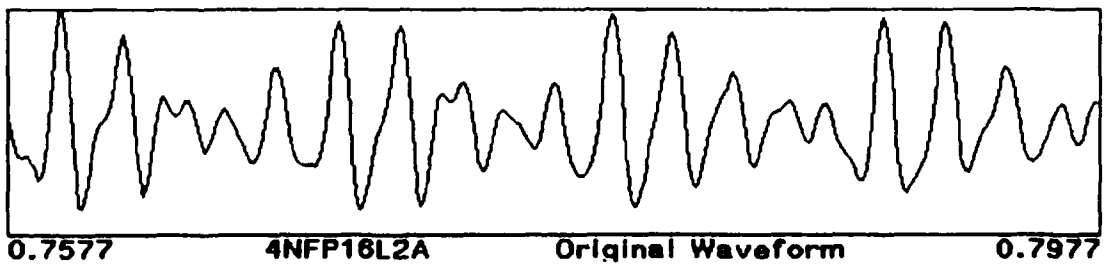

0.0000
4NFP16L4A
Narrow-Band Spectrogram
1.5352

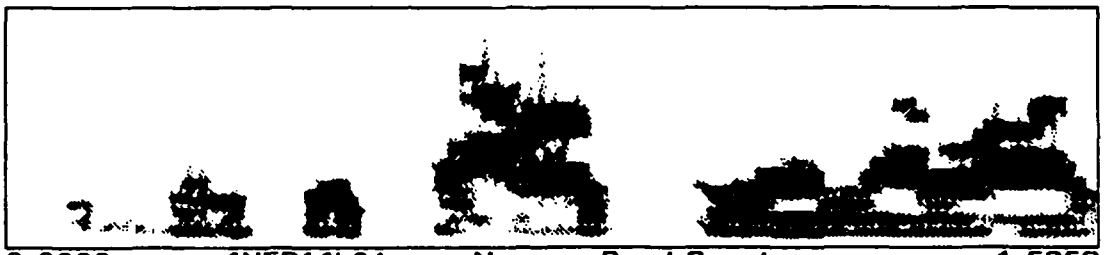

0.0000
4NFP16L2A
Narrow-Band Spectrogram
1.5352


0.7577
4NFP16L8A
Original Waveform
0.7977

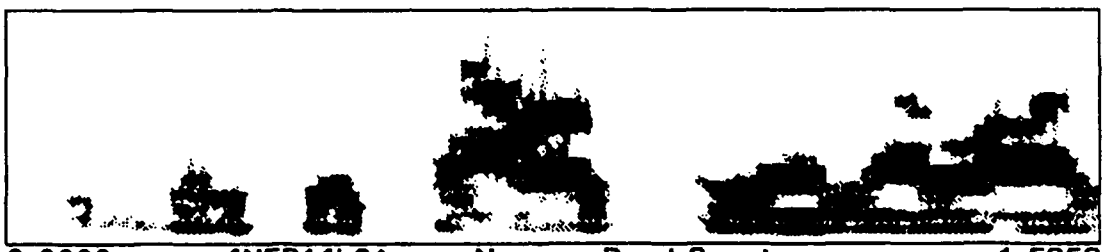

0.7577
4NFP16L6A
Original Waveform
0.7977


0.7577
4NFP16L4A
Original Waveform
0.7977


0.7577
4NFP16L2A
Original Waveform
0.7977

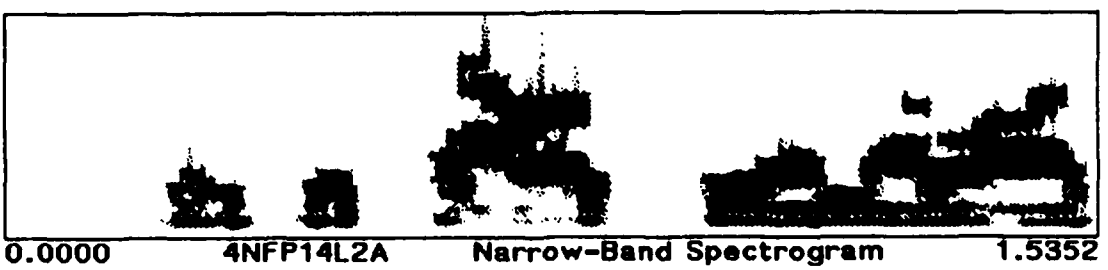


0.0000 4NFP14L8A Narrow-Band Spectrogram 1.5352

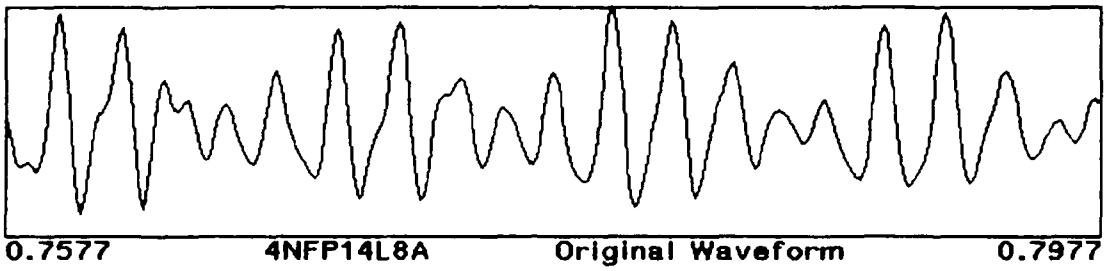


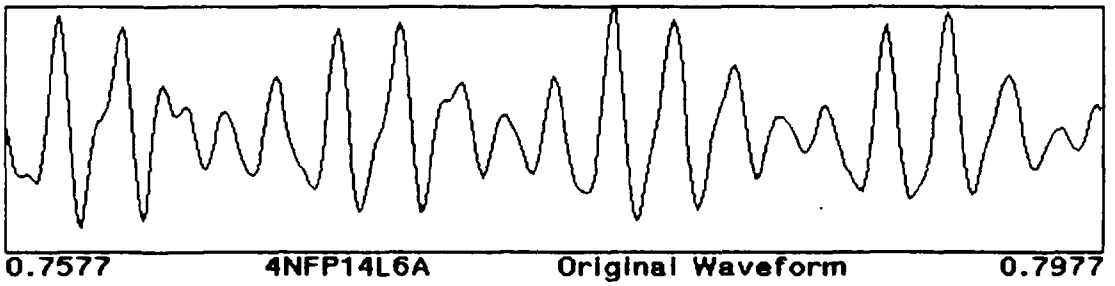
0.0000 4NFP14L6A Narrow-Band Spectrogram 1.5352

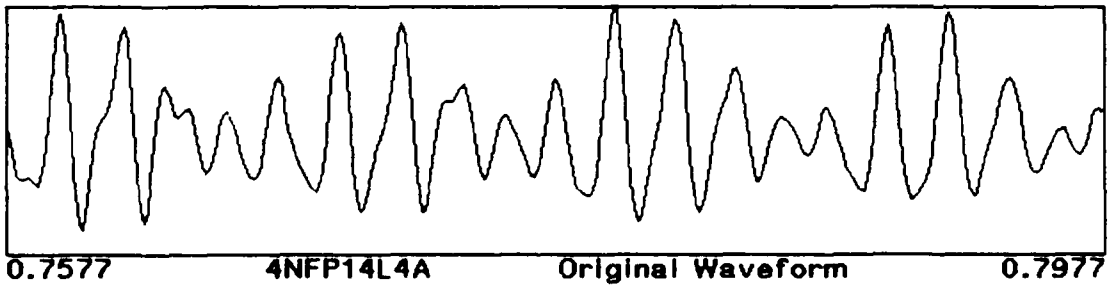
0.0000 4NFP14L4A Narrow-Band Spectrogram 1.5352

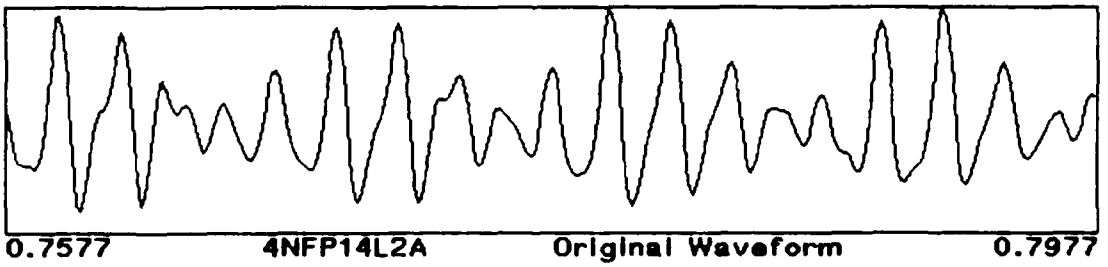


0.0000 4NFP14L2A Narrow-Band Spectrogram 1.5352


0.7577
4NFP14L8A
Original Waveform
0.7977


0.7577
4NFP14L6A
Original Waveform
0.7977

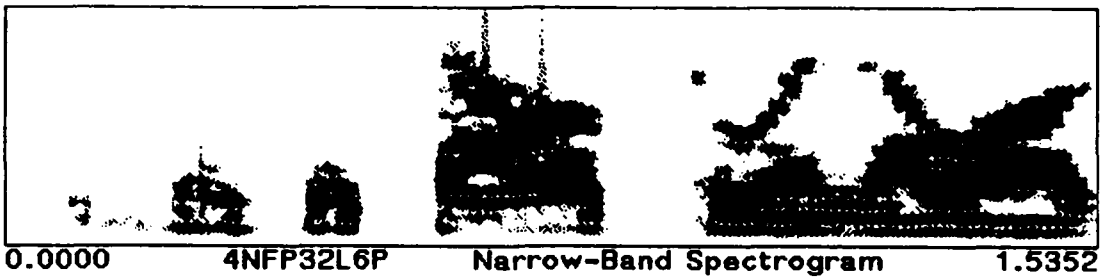

0.7577
4NFP14L4A
Original Waveform
0.7977

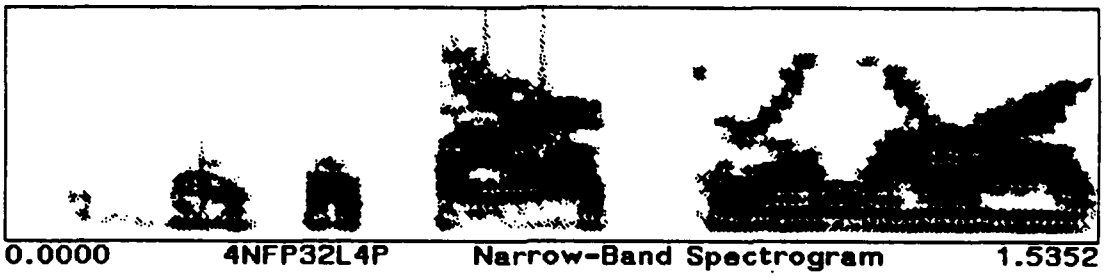

0.7577
4NFP14L2A
Original Waveform
0.7977

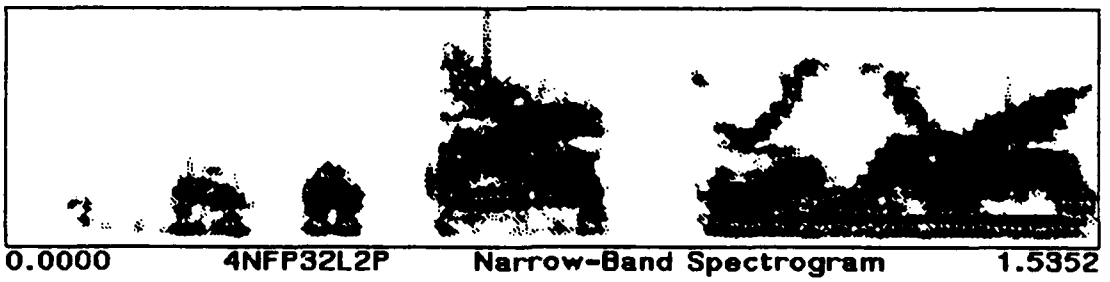
Appendix B: Phase Quantization Bits - Sample Values

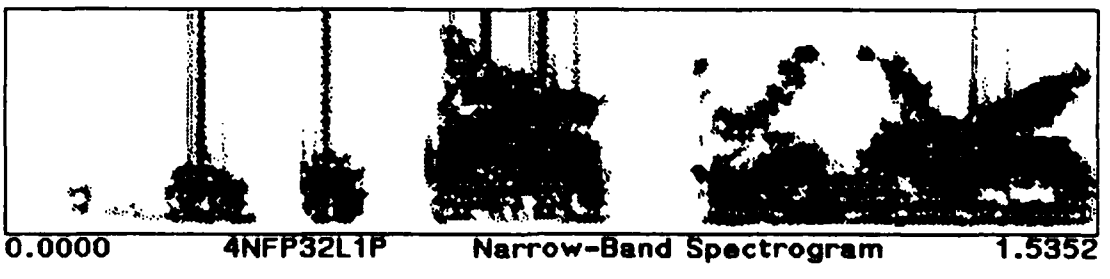
This Appendix shows the effects of the number of phase quantization bits for several different frame lengths. In order to reduce the effects of the number of amplitude quantization bits, this number was set to a high value (12). The code that appears bellow the pictures has the following meaning:

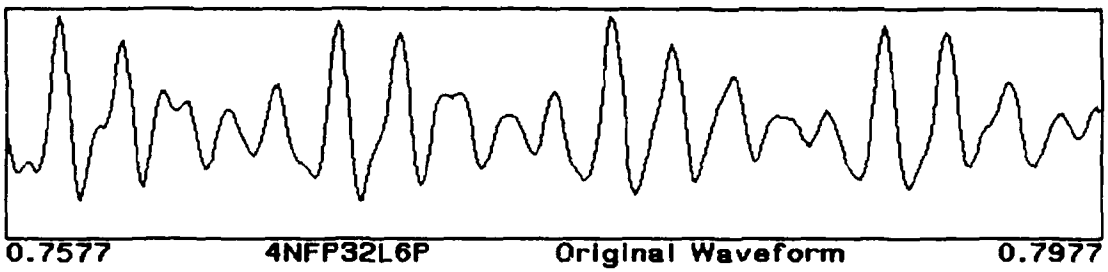
Example: 4NFP22L6P
4N - 4 Neighbors
FP - Fixed Pitch
22L - Frame Length is 22
6P - Number of Phase Quantization Bits is 6

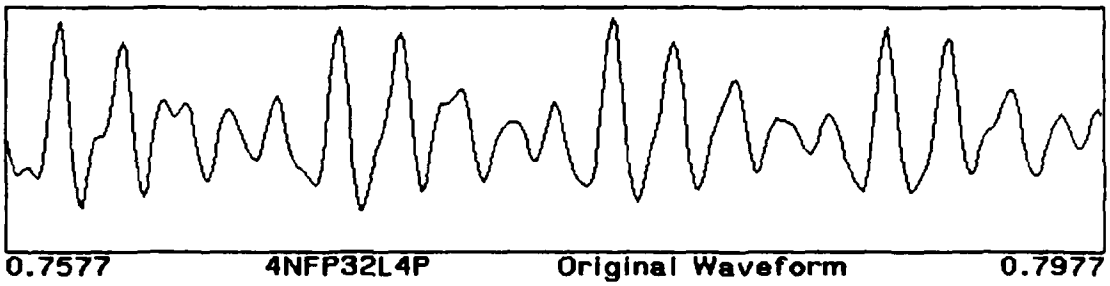

0.0000
4NFP32L6P
Narrow-Band Spectrogram
1.5352

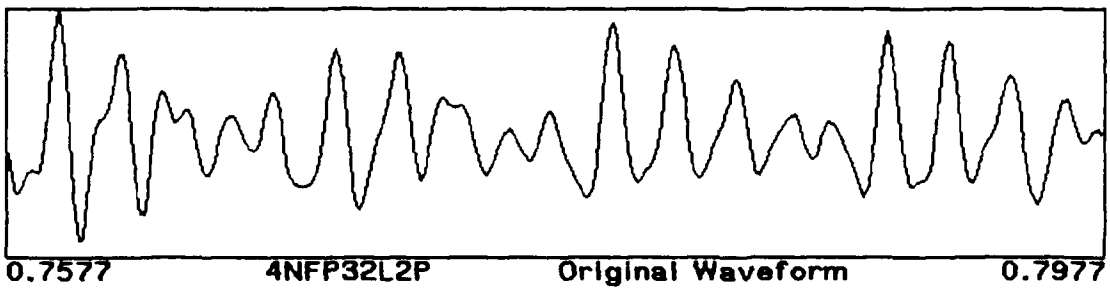

0.0000
4NFP32L4P
Narrow-Band Spectrogram
1.5352

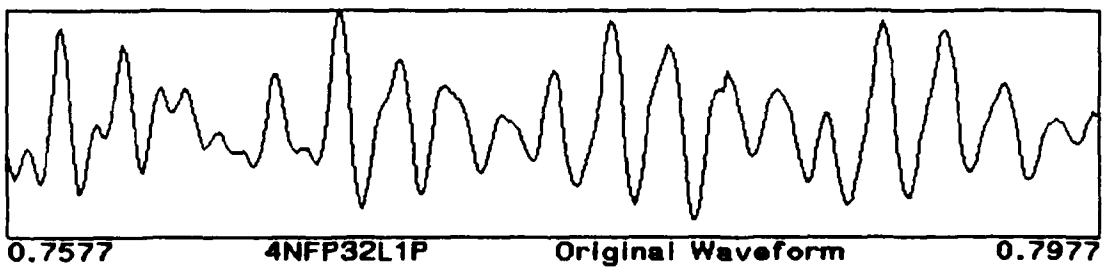

0.0000
4NFP32L2P
Narrow-Band Spectrogram
1.5352

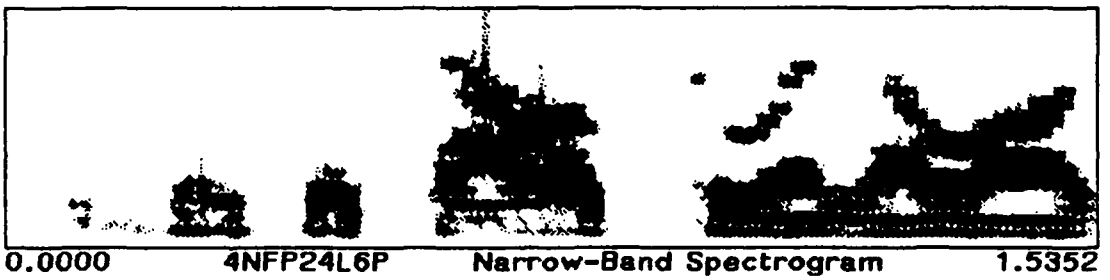

0.0000
4NFP32L1P
Narrow-Band Spectrogram
1.5352

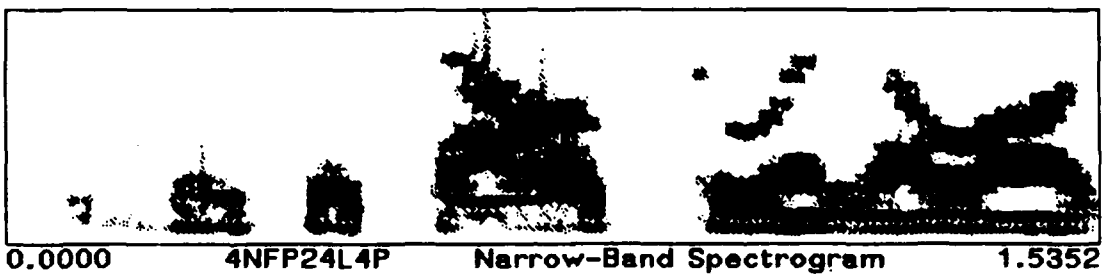

0.7577
4NFP32L6P
Original Waveform
0.7977

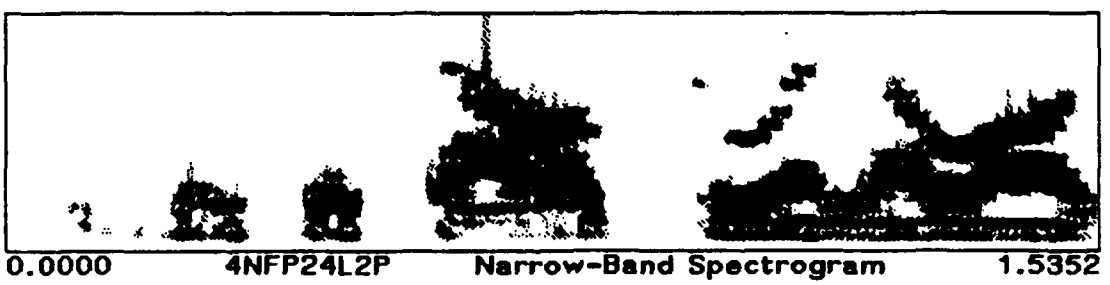

0.7577
4NFP32L4P
Original Waveform
0.7977

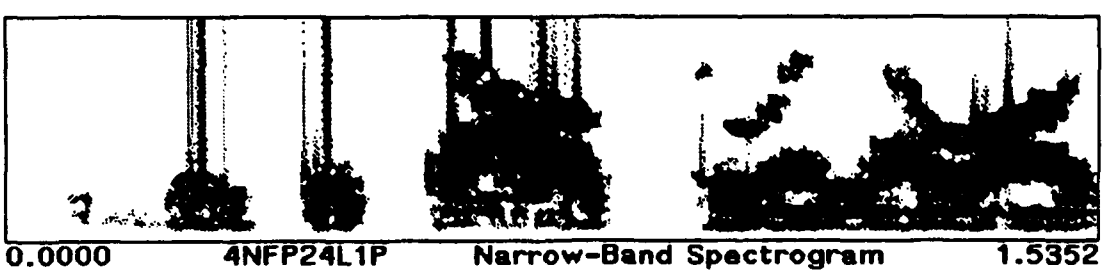

0.7577
4NFP32L2P
Original Waveform
0.7977

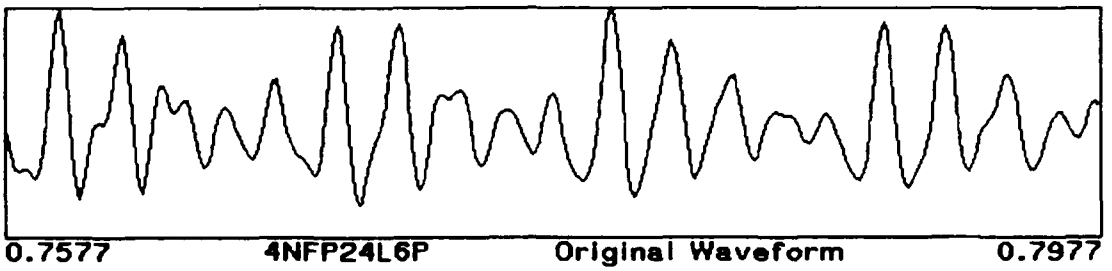

0.7577
4NFP32L1P
Original Waveform
0.7977

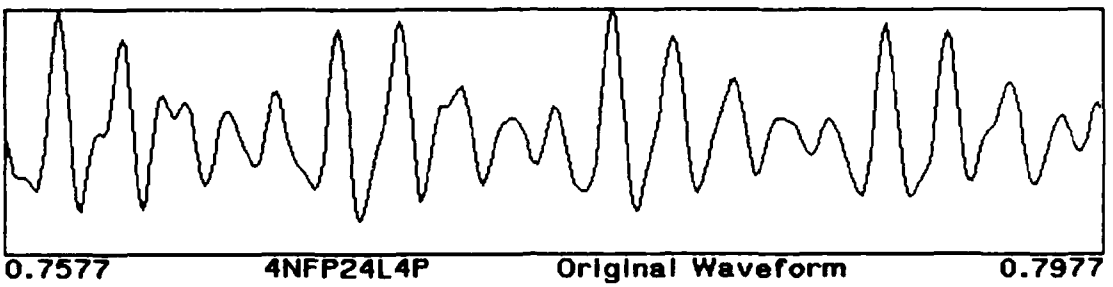

0.0000
4NFP24L6P
Narrow-Band Spectrogram
1.5352

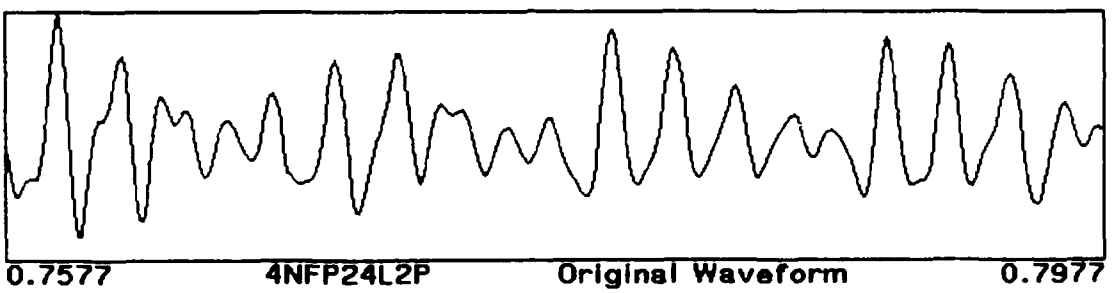

0.0000
4NFP24L4P
Narrow-Band Spectrogram
1.5352

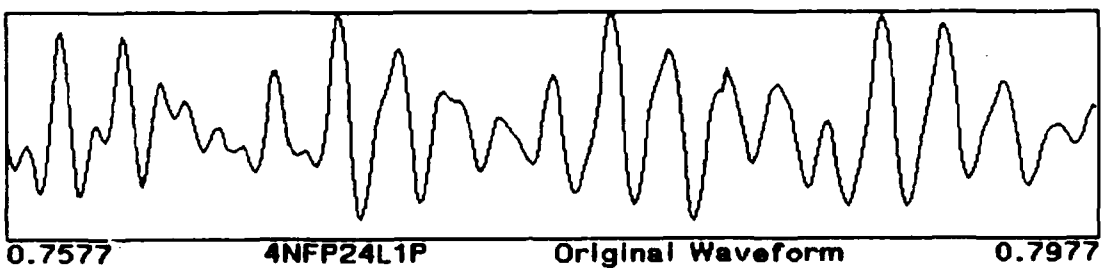

0.0000
4NFP24L2P
Narrow-Band Spectrogram
1.5352

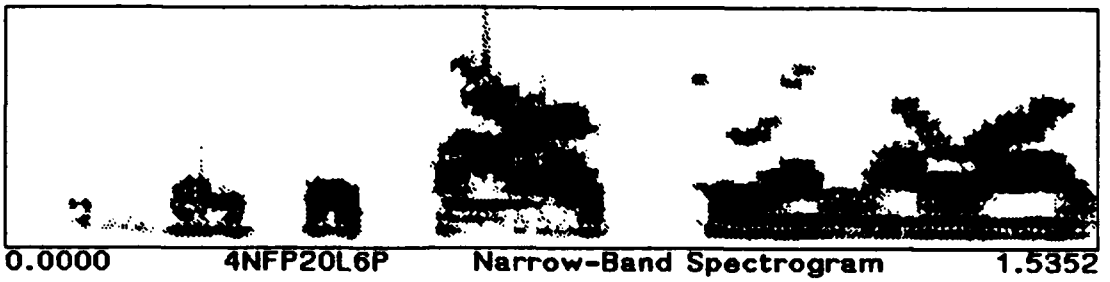

0.0000
4NFP24L1P
Narrow-Band Spectrogram
1.5352

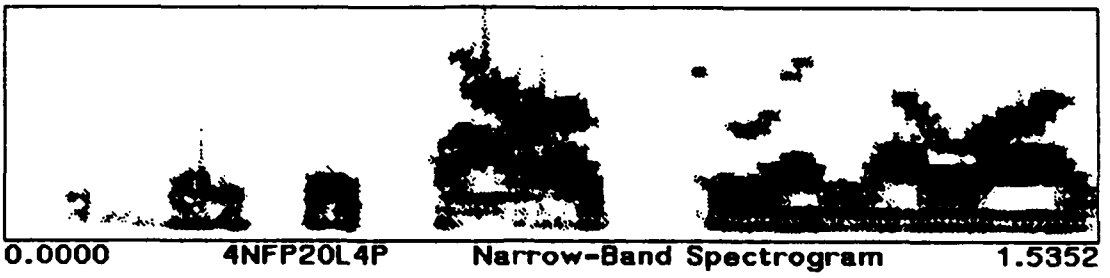

0.7577
4NFP24L6P
Original Waveform
0.7977

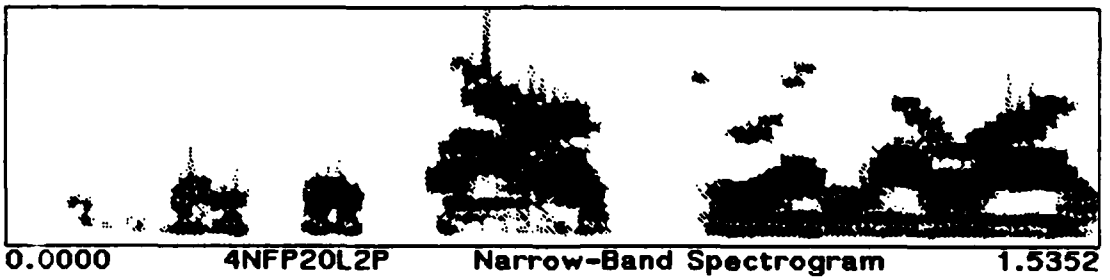

0.7577
4NFP24L4P
Original Waveform
0.7977

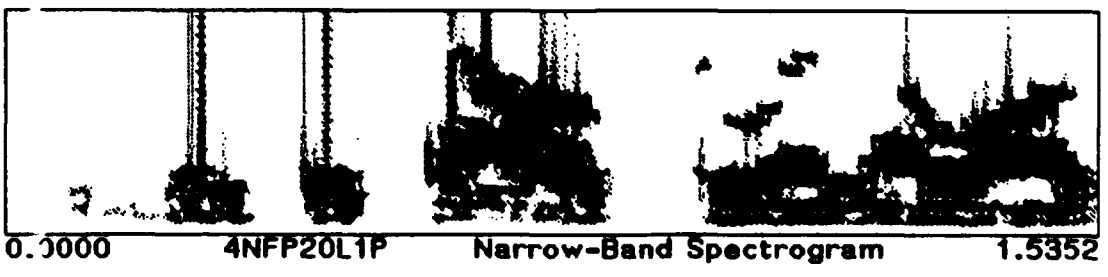

0.7577
4NFP24L2P
Original Waveform
0.7977

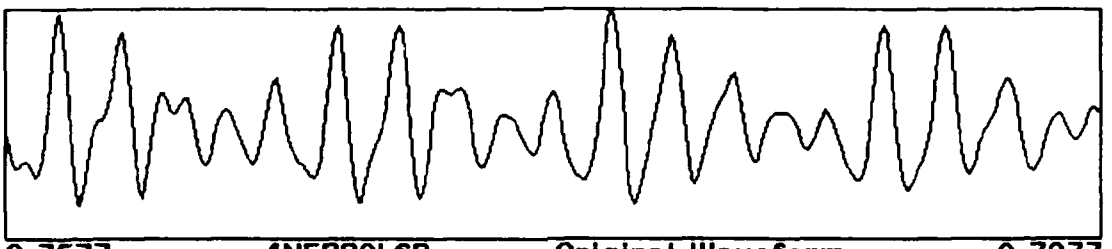

0.7577
4NFP24L1P
Original Waveform
0.7977

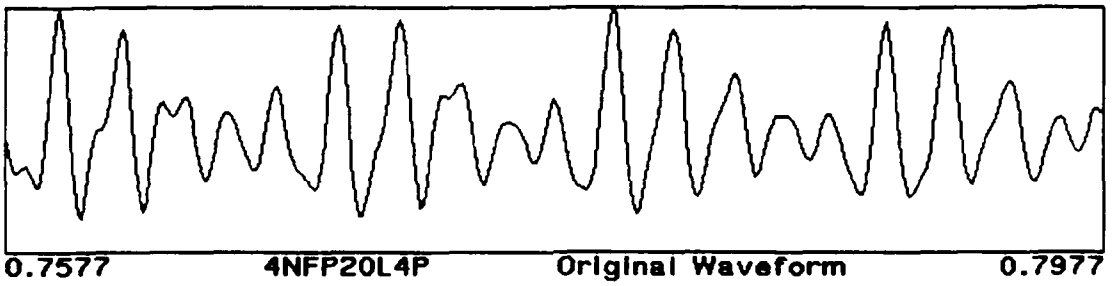

0.0000
4NFP20L6P
Narrow-Band Spectrogram
1.5352

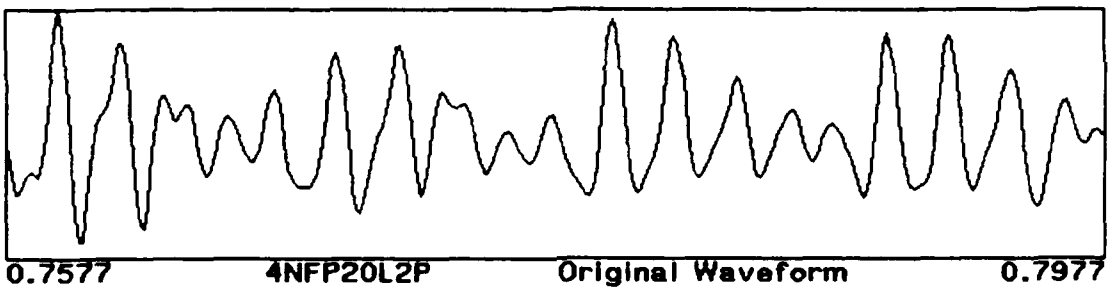

0.0000
4NFP20L4P
Narrow-Band Spectrogram
1.5352

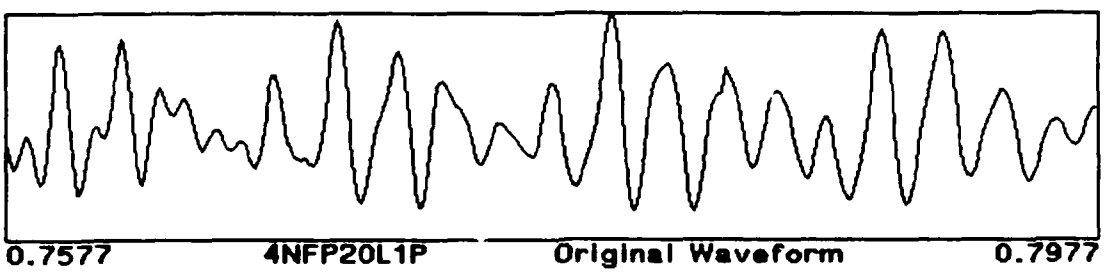

0.0000
4NFP20L2P
Narrow-Band Spectrogram
1.5352


0.0000
4NFP20L1P
Narrow-Band Spectrogram
1.5352


0.7577
4NFP20L6P
Original Waveform
0.7977


0.7577
4NFP20L4P
Original Waveform
0.7977

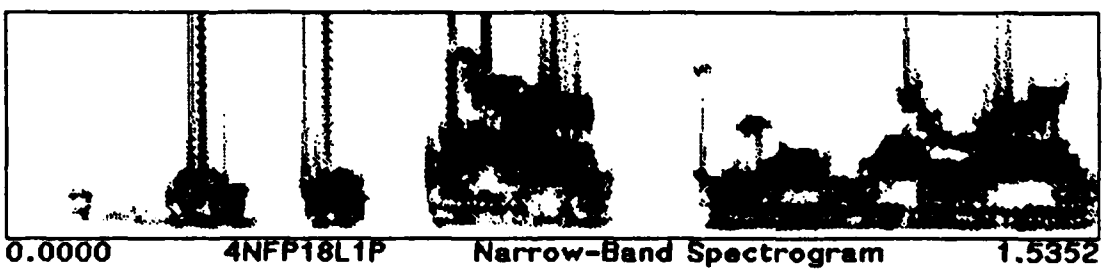

0.7577
4NFP20L2P
Original Waveform
0.7977

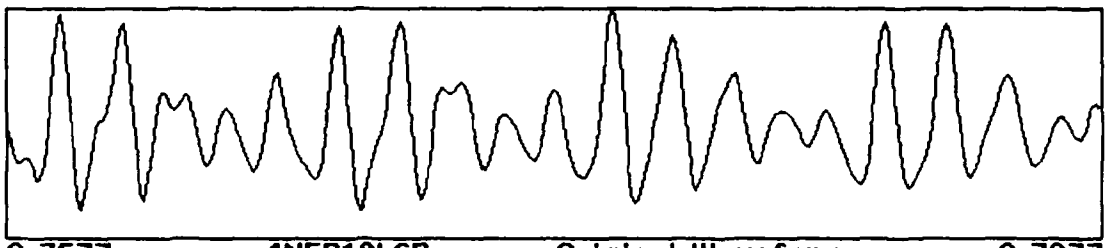

0.7577
4NFP20L1P
Original Waveform
0.7977

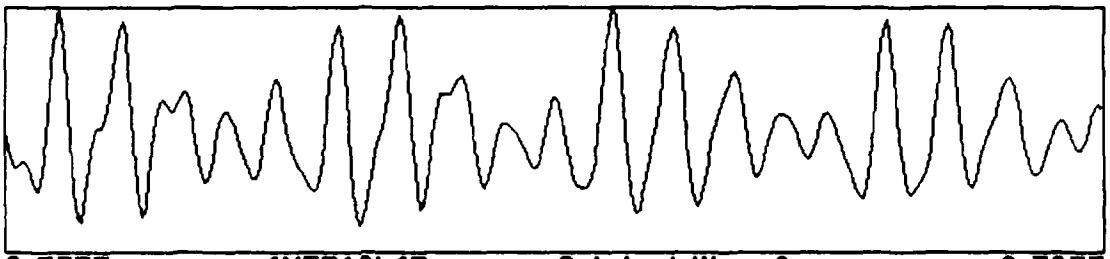

0.0000
4NFP18L6P
Narrow-Band Spectrogram
1.5352

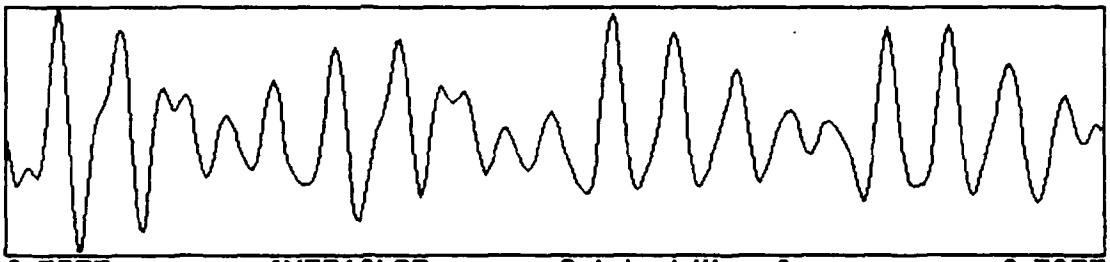

0.0000
4NFP18L4P
Narrow-Band Spectrogram
1.5352

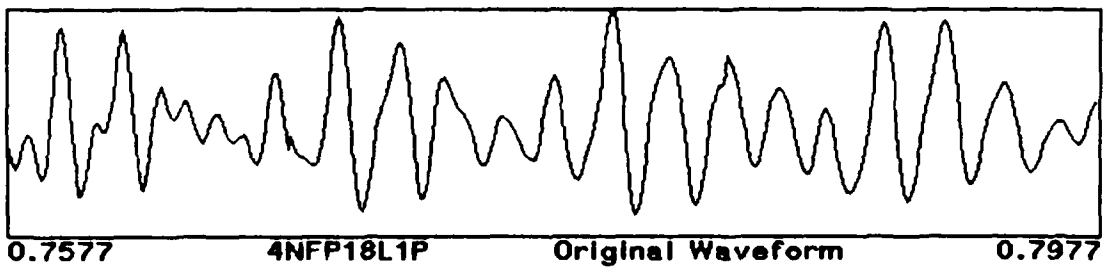

0.0000
4NFP18L2P
Narrow-Band Spectrogram
1.5352


0.0000
4NFP18L1P
Narrow-Band Spectrogram
1.5352


0.7577
4NFP18L6P
Original Waveform
0.7977

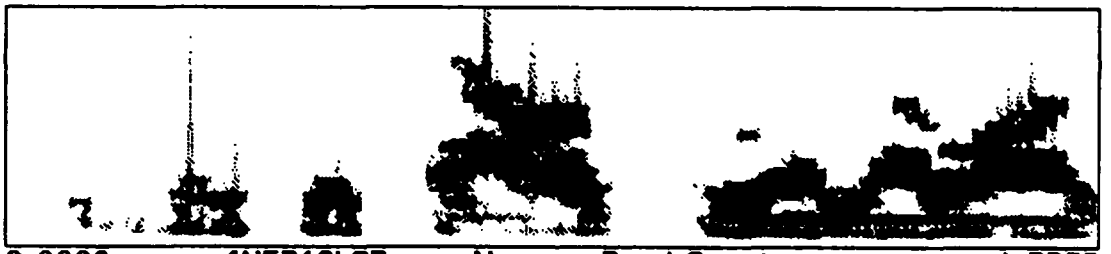

0.7577
4NFP18L4P
Original Waveform
0.7977

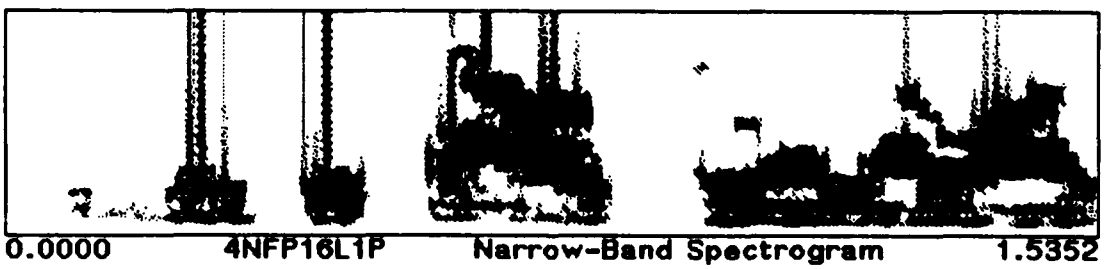

0.7577
4NFP18L2P
Original Waveform
0.7977

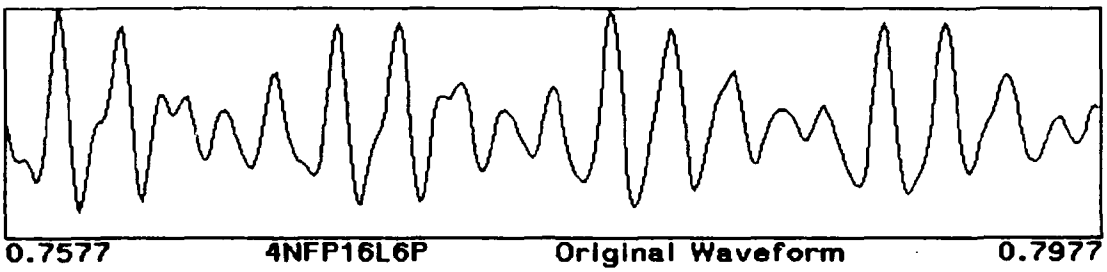

0.7577
4NFP18L1P
Original Waveform
0.7977

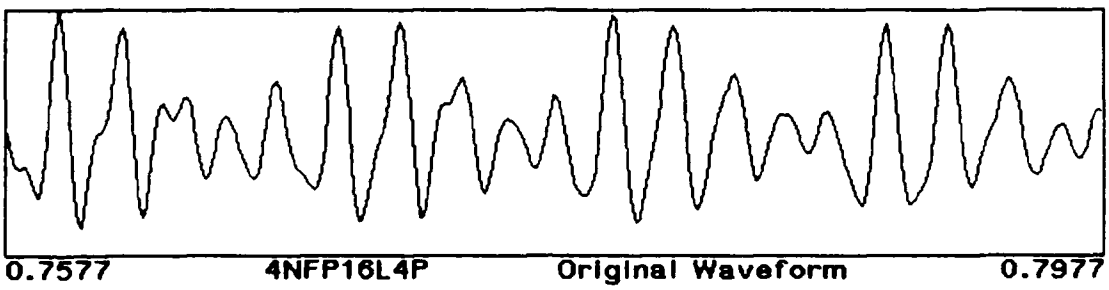

0.0000
4NFP16L6P
Narrow-Band Spectrogram
1.5352

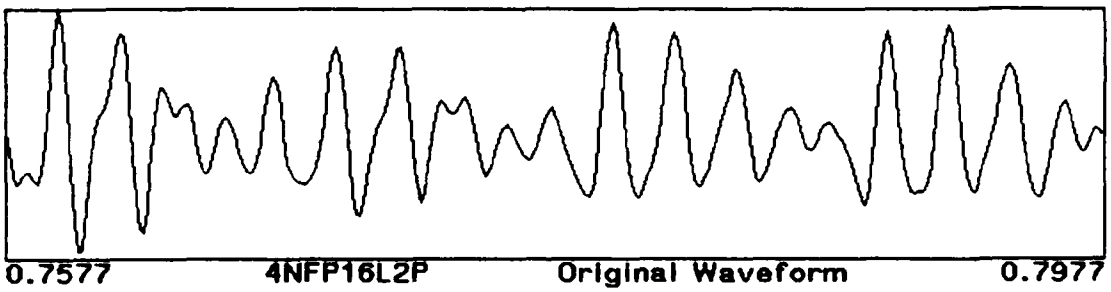

0.0000
4NFP16L4P
Narrow-Band Spectrogram
1.5352

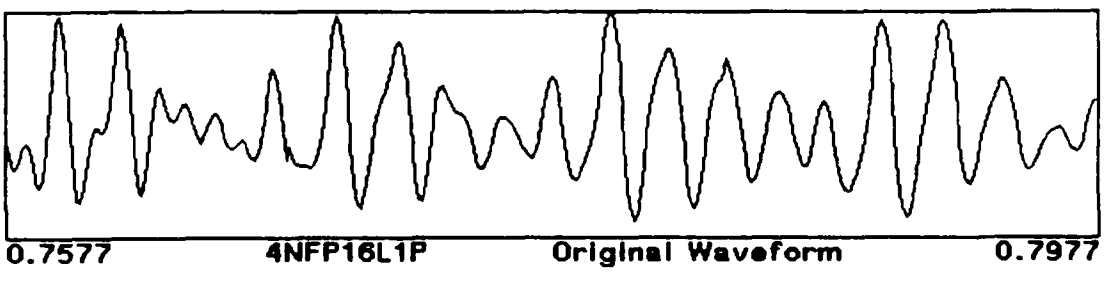

0.0000
4NFP16L2P
Narrow-Band Spectrogram
1.5352

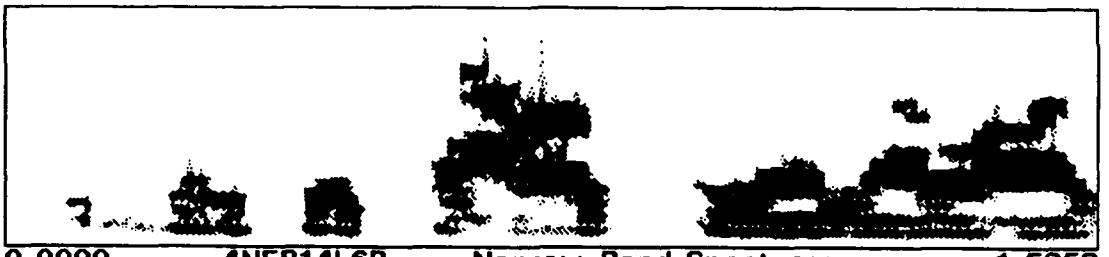

0.0000
4NFP16L1P
Narrow-Band Spectrogram
1.5352


0.7577
4NFP16L6P
Original Waveform
0.7977


0.7577
4NFP16L4P
Original Waveform
0.7977

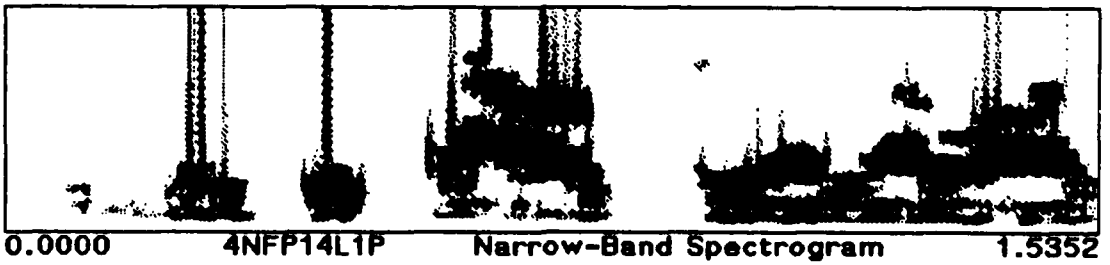

0.7577
4NFP16L2P
Original Waveform
0.7977

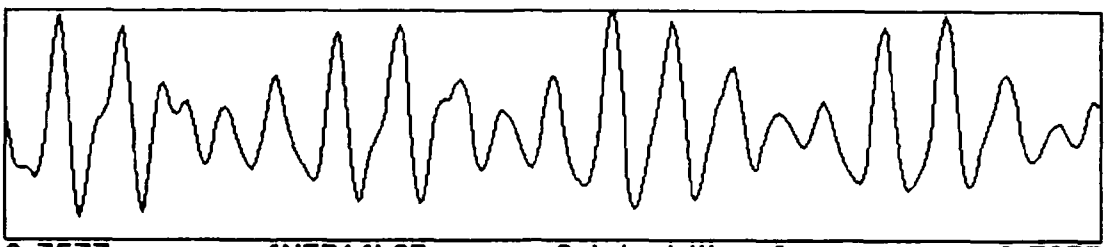

0.7577
4NFP16L1P
Original Waveform
0.7977

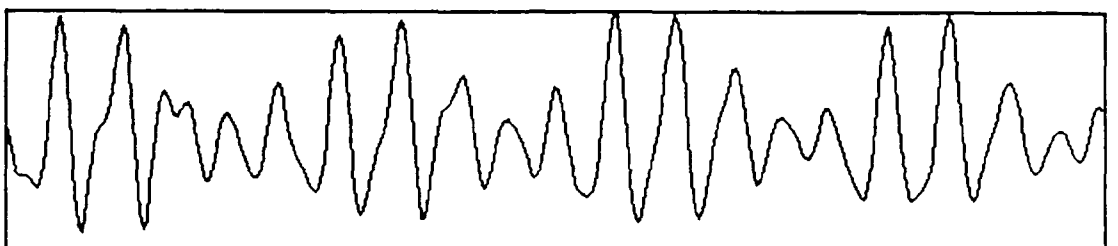

0.0000
4NFP14L6P
Narrow-Band Spectrogram
1.5352

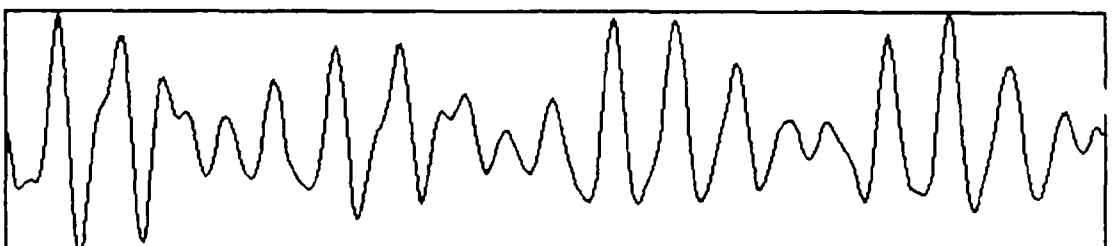

0.0000
4NFP14L4P
Narrow-Band Spectrogram
1.5352

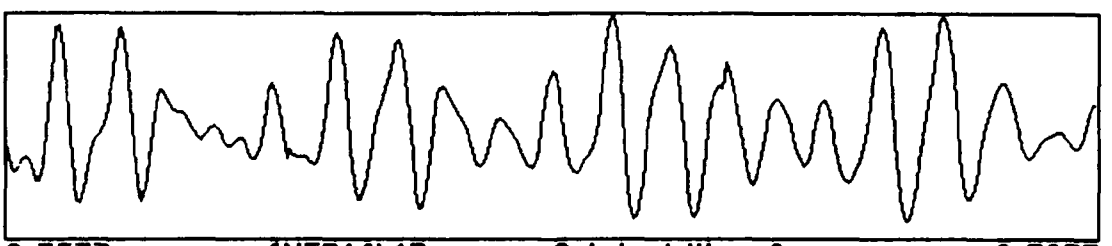

0.0000
4NFP14L2P
Narrow-Band Spectrogram
1.5352


0.0000
4NFP14L1P
Narrow-Band Spectrogram
1.5352

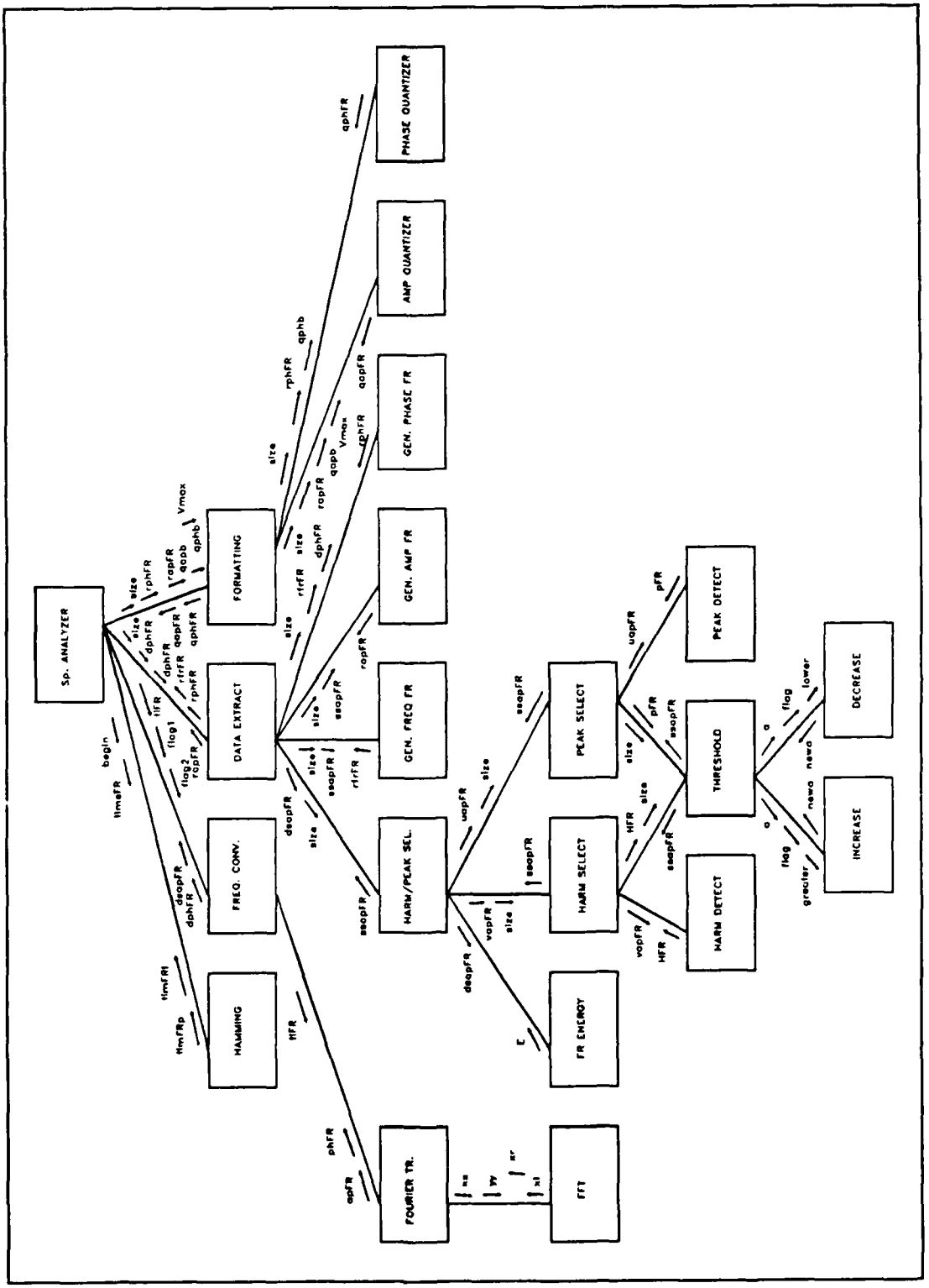

0.7577
4NFP14L6P
Original Waveform
0.7977

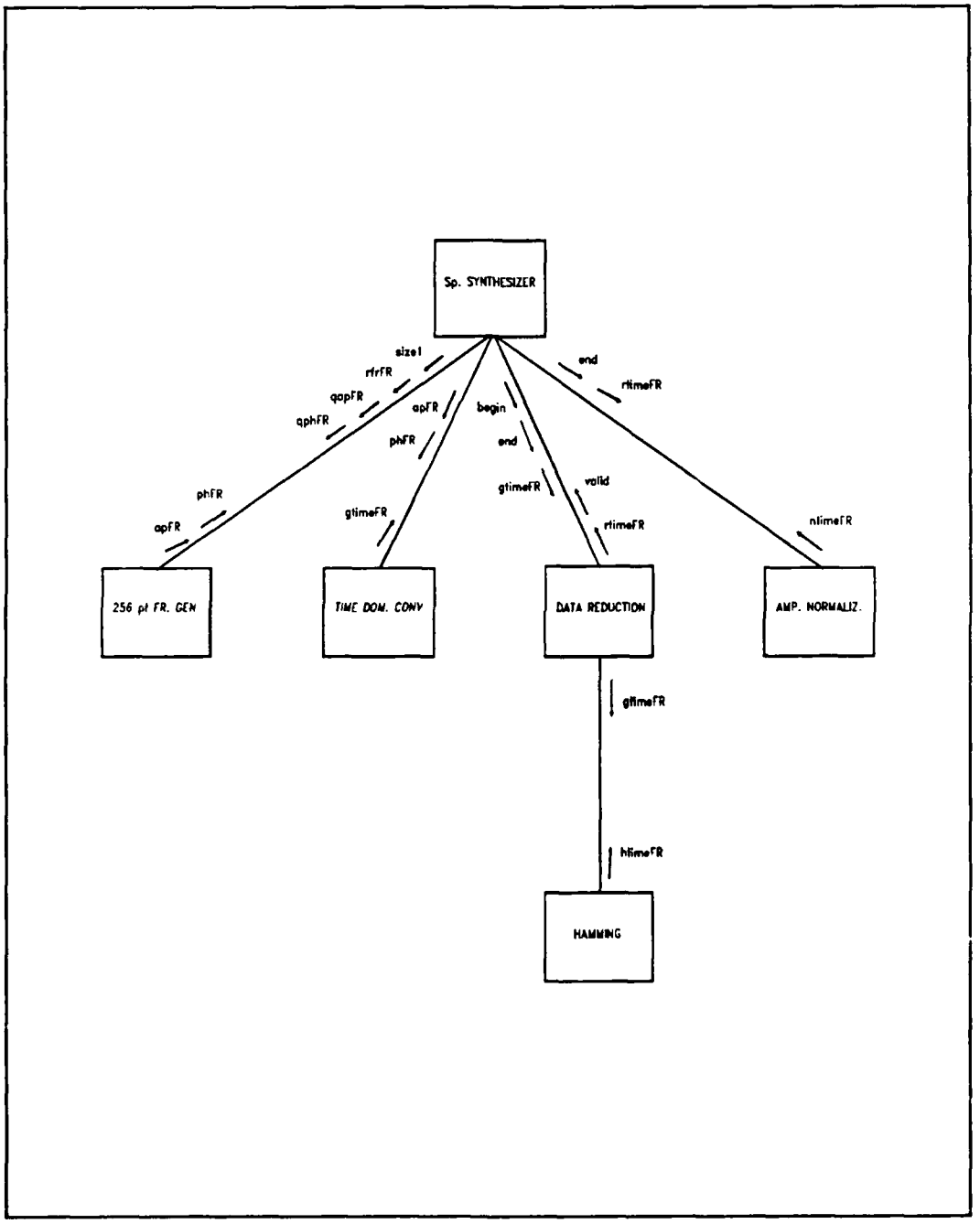

0.7577
4NFP14L4P
Original Waveform
0.7977


0.7577
4NFP14L2P
Original Waveform
0.7977


0.7577
4NFP14L1P
Original Waveform
0.7977

Appendix C: Data Flow Diagrams


Sp. ANALYZER
FORMATTING
DATA EXTRACT
FREQ. CONV.
HAMMING
FOURIER TR.
FFT
HARM/PEAK SEL.
FR ENERGY
GEN. FREQ FR
GEN. AMP FR
GEN. PHASE FR
PHASE QUANTIZER
AMP QUANTIZER
PEAK SELECT
HARM SELECT
PEAK DETECT
THRESHOLD
HARM DETECT
DECREASE
INCREASE
size
qphFR
qphb
Vmax
qapFR
qapb
dphFR
rifFR
ropFR
rpnFR
qfpFR
begin
timeFR
flag1
flag2
dapFR
dfpFR
timFRI
timFRp
HFR
phFR
apFR
ns
N
ar
ai
saapFR
uapFR
vapFR
daapFR
pFR
flag
lower
greater
newa
a
c


Sp. SYNTHESIZER
size1
rfrFR
qapFR
qphFR
apFR
phFR
begin
end
gtimeFR
valid
rtimeFR
end
rtimeFR
ntimeFR
phFR
apFR
gtimeFR
256 pt FR. GEN
TIME DOM. CONV
DATA REDUCTION
AMP. NORMALIZ.
gtimeFR
htimeFR
HAMMING

Appendix D: Source Code

```
************************************************************
*                                                          *
*                                                          *
*   Title    : Main Program - Speech Analysis/Synthesis    *
*   Author   : Luis Alenquer                               *
*   Date     : November 1989                               *
*   Language : Fortran 77                                  *
*   Computer : VAX 11/780                                  *
*                                                          *
*                                                          *
*   This program encodes a speech file into three frames   *
*   containing frequency, amplitude, and phase informa-    *
*   tion. The original speech is reconstructed from        *
*   values. The first part is accomplished in the sub-     *
*   -program "Speech Analyser", and the reconstruction     *
*   is done in the "Speech Sythesizer".                    *
*                                                          *
*                                                          *
************************************************************

      integer*4 hdata(64)
      character*32 in, out
      integer*2 timeFR(256), ntimeFR(256)
      integer n, i, qapb, qphb, size1
      parameter ( size1=18)
      real rfrFR(size1), qapFR(size1), qapFRp(size1),
     + qapFRi(size1)
      real qphFR(size1), qphFRp(size1), qphFRi(size1)
      real rfrFRp(size1), rfrFRi(size1), vmax
      logical begin,end


      begin=.true.
      end=.false.

      write (*,4) '      Enter the name of input file:   '
      read (*,6) in

      write (*,4) '     Enter the name of output file:  '
      read (*,6) out

      write(*,4) '     Enter # of ampl. quantization bits:'
      read (*,*) qapb

      write(*,4) '     Enter # of phase quantization bits:'
      read (*,*) qphb

      write(*,4) '      Enter quantizer max amplitude:   '
      read (*,*) vmax

4     format (a,$)
6     format (a)
```

```
      call datarate(size1, qapb, qphb)
      open (10, file=in, status='old', access='sequential',
     +      form='unformatted', recordtype='fixed',
     +      recl=128)
      open ( unit=99, file=out, status='new',
     +       access='sequential', form='unformatted',
     +       recordtype='fixed', recl=128)
      read(10) hdata
15    write (99) hdata
20    call analyzer(begin, timeFR, size1, qapb, qphb, vmax,
     +              qapFRp, qapFRi, qphFRp, qphFRi, rfrFRp,
                    rfrFRi, end)
      if (.not.end) then
         if (.not.begin) then
            do 30 i=1,size1
                 qapFR(i)=qapFRp(i)
                 qphFR(i)=qphFRp(i)
30               rfrFR(i)=rfrFRp(i)
            call synthesizer(begin, end, size1, qapFR,
     +                       qphFR, rfrFR, ntimeFR)

         endif
         do 40 i=1,size1
              qapFR(i)=qapFRi(i)
              qphFR(i)=qphFRi(i)
40            rfrFR(i)=rfrFRi(i)
      endif
      call synthesizer(begin, end, size1, qapFR, qphFR,
     +                 rfrFR, ntimeFR)
      begin=.false.
      if (.not.end) goto 20

      end



*****************************************************************
*                                                               *
*   Subroutine Speech Analyzer                                  *
*                                                               *
*   Top-level module of the speech encoding process.            *
*   The original speech file is analysed in the fre-            *
*   quency domain to produce three frames containing            *
*   frequency, amplitude and phase information.                 *
*                                                               *
*****************************************************************


      subroutine analyzer(begin, timeFR, size1, qapb, qphb,
     +                    vmax, qapFRp, qapFRi, qphFRp,
                          qphFRi, rfrFRp, rfrFRi, end)
```

```
      integer size, size1
      parameter (size=18)
      integer*2 timeFR(256)
      integer i, qapb, qphb
      real timFRp(256), timFRi(256)
      real dsapFR(256), dphFR(256)
      real rapFR(size), rphFR(size)
      real rfrFR(size), qapFR(size)
      real qapFRp(size1), qapFRi(size1)
      real qphFR(size)
      real qphFRp(size1), qphFRi(size1)
      real rfrFRp(size1), rfrFRi(size1),vmax
      logical flag1, flag2, begin, end

      end=.false.
      if (begin) then
         open (50, access='sequential', form='unformatted',
     +          file='50tmp',status='new')
         open (60, access='sequential', form='unformatted',
     +          file='60tmp',status='new')
      endif

      read(10,end=200) timeFR
      call hamming(begin, timeFR, timFRp, timFRi)
      if (.not.begin) then
         call freqconv(flag1, flag2, timFRp, dsapFR, dphFR)
         call dataextraction(size, dsapFR, dphFR, rapFR,
     +                       rphFR, rfrFR)
         do 5 i=1,size
5           rfrFRp(i)=rfrFR(i)
         call outputformatting(size, qapb, qphb, vmax,
     +                         rapFR, rphFR, qapFR, qphFR)
         do 10 i=1,size
            qapFRp(i)=qapFR(i)
10          qphFRp(i)=qphFR(i)
      endif
      call freqconv(flag1, flag2, timFRi, dsapFR, dphFR)
      call dataextraction(size, dsapFR, dphFR, rapFR, rphFR,
     +                    rfrFR)
      call outputformatting(size,qapb, qphb, vmax, rapFR,
     +                      rphFR, qapFR, qphFR)
      do 20 i=1,size
         qapFRi(i)=qapFR(i)
         qphFRi(i)=qphFR(i)
20       rfrFRi(i)=rfrFR(i)

      return
200   end=.true.

      end
```

```
************************************************************
*                                                          *
*   Subroutine Hamming Windows                             *
*                                                          *
*   Hamming Windows with 50% overlap are applied to        *
*   the incoming speech frames.                            *
*                                                          *
************************************************************

      subroutine hamming(begin, timeFR, timFRp, timFRi)

      real timFRi(256),timFRp(256)
      integer*2 timeFR(256)
      integer j,k
      dimension x(256), z(256),w(256)
      real x, z, pi, w
      logical begin
      save x

      data pi/3.14159265358979324/

      if (.not.begin) goto 120
110   do 115 j=1,256
         x(j)=timeFR(j)
         w(j)=x(j)*(0.54-0.46*cos((2*pi/255)*(j-1)))
115      timFRi(j)=w(j)
      goto 199
120   do 125 j=1,128
         z(j)=x(j+128)
125      z(j+128)=timeFR(j)
      do 130 j=1,256
130       timFRp(j)=z(j)*(0.54-0.46*cos((2*pi/255)*(j-1)))

      goto 110
199   continue

      end


************************************************************
*                                                          *
*   Subroutine Frequency Conversion                        *
*                                                          *
*   An input time frame is converted to the                *
*   frequency domain, resulting two frames with            *
*   spectral amplitudes and phases.                        *
*                                                          *
************************************************************

      subroutine freqconv(flag1, flag2, tiFR, dsapFR, dphFR)

      real apFR(256), tiFR(256), phFR(256), sapFR(256),
     +     apFR1(256), apFR2(256), phase(256), dsapFR(256),
```

```
     +      dphFR(256), apFR3(256)
       logical flag1, flag2
       integer i
*      save apFR1, apFR2, apFR3, phase
       save

       call dft(tiFR, apFR, phFR)
       do 10 i=1,256
          dsapFR(i)=apFR(i)
10        dphFR(i)=phFR(i)

       end


*****************************************************************
*                                                               *
*   Subroutine Discrete Fourier Transform                       *
*                                                               *
*   This modules calls a Fast Fourier Transform                 *
*   subroutine to take a 512-point FFT of an                    *
*   incoming time frame. The output of the FFT is               *
*   converted to amplitudes and phases.                         *
*                                                               *
*****************************************************************

       subroutine dft(tiFR, apFR, phFR)

       real yy(512), xx(512), apFR(256), phFR(256), tiFR(256)
       real a, b
       integer i

       do 10 i=1,512
10        yy(i)=0
       do 20 i=1,256
          xx(i)=tiFR(i)
20        xx(i+256)=0
       call fft(9, xx, yy, 1)
       do 30 i=1,256
          a=xx(i)
          b=yy(i)
          apFR(i)=(sqrt(a**2+b**2)*1.7)
          if (a.eq.0) then
             phFR(i)=pi/2
          else
             phFR(i)=atan2(b,a)
          end if
30     continue

       end
```

```
*****************************************************************
*                                                               *
*   Subroutine Fast Fourier Transform (10:457)                  *
*                                                               *
*****************************************************************


      subroutine fft(log2n, xr, xi, ntype)

      dimension xr(1), xi(1)
      integer log2n, ntype, i, j, k, n, nv2, nm1, l, le,
     +        le1, ip
      real xi, xr, tr, ti, ur, ui, wr, wi, ain, sign, pi
      data pi/3.14159265358979324/

      sign=-1.
      if (ntype.lt.0) sign=1.
      n=2**log2n
      nv2=n/2
      nm1=n-1
      j=1
      do 7 i=1,nm1
         if (i.ge.j) goto 5
         tr=xr(j)
         ti=xi(j)
         xr(j)=xr(i)
         xi(j)=xi(i)
         xr(i)=tr
         xi(i)=ti
5        k=nv2
6        if (k.ge.j) goto 7
         j=j-k
         k=k/2
         goto 6
7        j=j+k
      do 20 l=1,log2n
         le=2**l
         le1=le/2
         ur=1.
         ui=0.
         wr=cos(pi/le1)
         wi=sign*sin(pi/le1)
         do 20 j=1,le1
            do 10 i=j,n,le
               ip=i+le1
               tr=xr(ip)*ur-xi(ip)*ui
               ti=xr(ip)*ui+xi(ip)*ur
               xr(ip)=xr(i)-tr
               xi(ip)=xi(i)-ti
               xr(i)=xr(i)+tr
10             xi(i)=xi(i)+ti
            tr=ur*wr-ui*wi
            ti=ur*wi+ui*wr
```

```
            ur=tr
20          ui=ti
       if (ntype.gt.0) return
       ain=1./n
       do 30 i=1,n
          xr(i)=xr(i)*ain
30        xi(i)=xi(i)*ain
       return

       end


*****************************************************************
*                                                               *
*   Subroutine Data Extraction                                  *
*                                                               *
*   This module controls the encoding process.                  *
*                                                               *
*****************************************************************

       subroutine dataextraction(size, dsapFR, dphFR, rapFR,
     +                           rphFR, rfrFR)

       integer size
       real dsapFR(256), dphFR(256), rapFR(size),
     +      rphFR(size), rfrFR(size), ssapFR(256)

       call harmpeakselect(size, dsapFR, ssapFR)
       call freqFR(size, ssapFR, rfrFR)
       call ampFR(size, ssapFR, rapFR)
       call phaseFR(size, rfrFR, dphFR, rphFR)

       end


*****************************************************************
*                                                               *
*   Subroutine Harmonics/Peaks Selection                        *
*                                                               *
*   Based on the energy of a frame this module chooses          *
*   an harmonic or a peak selection process.                    *
*                                                               *
*****************************************************************

       subroutine harmpeakselect(size, dsapFR, ssapFR)

       integer size, i
       real dsapFR(256), ssapFR(size), energy, vapFR(256),
      +      uapFR(256)
       external energy


       if (energy(dsapFR).gt.9.e+5) then
```

```
            do 10 i=1,256
10             vapFR(i)=dsapFR(i)
            call harmselect(size, vapFR, ssapFR)
         else
            do 20 i=1,256
20             uapFR(i)=dsapFR(i)
            call peakselect(size, uapFR, ssapFR)
         endif


         end



*************************************************************
*                                                           *
*   Function Energy of a Frame                              *
*                                                           *
*   The energy of frame is computed from its spectral       *
*   amplitudes.                                             *
*                                                           *
*************************************************************

         real function energy(dsapFR)

         real dsapFR(256)
         integer i

         energy=0.
         do 10 i=1,256
10          energy=energy+dsapFR(i)**2
            energy=sqrt(energy)
         write(*,*) ' '
         write(*,*) 'energy:'
         write(*,*) energy
         write(*,*) ' '
         end


*************************************************************
*                                                           *
*   Subroutine Harmonic Selection                           *
*                                                           *
*   This is a control module that encapsulates the          *
*   harmonics selection process.                            *
*                                                           *
*************************************************************

       subroutine harmselect(size, vapFR, ssapFR)

       integer size
       real vapFR(256), ssapFR(256), hFR(256)
```

```
      call harmdetect(vapFR, hFR)
      call threshold(size, hFR, ssapFR)

      end

************************************************************
*                                                          *
*   Subroutine Harmonic Detection                          *
*                                                          *
*   Detects harmonics based on a fixed pitch and a         *
*   four neighbors rule.                                   *
*                                                          *
************************************************************

      subroutine harmdetect(vapFR, hFR)

      integer i, j, n
      real vapFR(256), hFR(256), harm, a, b, c, d, e

      do 10 i=1,256
10       hFR(i)=0.0
      n=8
      j=n+2
      do 20 i=n+2,255
         harm=0.0
         if (i.eq.j) then
            a=vapFR(i-1)
            b=vapFR(i)
            c=vapFR(i+1)
            d=vapFR(i-2)
            e=vapFR(i+2)
            harm=max(a,b,c,d,e)
            hFR(i)=b
            hFR(i-1)=a
            hFR(i+1)=c
            hFR(i-2)=d
            hFR(i+2)=e
            if (j.le.12) then n=j
            a=0.0
            b=0.0
            c=0.0
            d=0.0
            e=0.0
            j=j+n
      endif
20    continue

      end
```

```
************************************************************
*                                                          *
*   Subroutine Variable Amplitude Threshold                *
*                                                          *
*   Sets up a variable amplitude threshold to select       *
*   a desired number of amplitudes components.             *
*                                                          *
************************************************************

      subroutine threshold(size, hFR, ssapFR)

      integer size, n
      real hFR(256), ssapFR(256), a, b, x(256), d, c, e, newa
      logical lower, greater, incr, decr, flag

      d=hFR(8)
      c=hFR(9)
      e=hFR(10)
      a=max(d,c,e)/4
      if (a.eq.0.0) then
         a=100.0
      endif
      lower=.false.
      greater=.false.
      incr=.false.
      decr=.false.
      flag=.false.
5     n=256
      do 10 i=1,256
10       x(i)=hFR(i)
      do 20 i=1,256
         b=a/sqrt(1.0+(i/42.0)**2.0)
      if (x(i).le.b) then
         x(i)=0.0
         n=n-1
      endif
20       continue
      if (n.lt.size) then
         if (a.eq.0.0) goto 25
         lower=.true.
         greater=.false.
      endif
      if (n.gt.size) then
         greater=.true.
         lower=.false.
      endif
    if(((.not.decr).and.(greater.or.incr)).and.(n.ne.size))
     +   then
         call increase(flag, greater, a, newa)
         incr=.true.
         a=newa
         goto 5
      endif
```

```
      if((decr.or.(lower.and.(.not.incr))).and.(n.ne.size))
     +   then
          call decrease(flag,lower,a,newa)
          decr=.true.
          a=newa
          goto 5
        endif
        flag=.false.
25      do 30 i=1,256
30         ssapFR(i)=x(i)


        end

*****************************************************************
*                                                               *
*   Subroutine Peak Selection                                   *
*                                                               *
*   This is a control module that encapsulates the              *
*   peaks selection process.                                    *
*                                                               *
*****************************************************************

        subroutine peakselect(size, uapFR, ssapFR)

        integer size
        real uapFR(256), ssapFR(256), pFR(256)

        call peakdetect(uapFR, pFR)
        call threshold(size, pFR, ssapFR)

        end


*****************************************************************
*                                                               *
*   Subroutine Peak Detection                                   *
*                                                               *
*   No peak detection is done. Nevertheless, the module         *
*   is maintained so it can be used in future research.         *
*                                                               *
*****************************************************************

        subroutine peakdetect(uapFR, pFR)

        integer i
        real uapFR(256), pFR(256), a, b, c

        do 30 i=1,256
30          pFR(i)=uapFR(i)

        end
```

```
************************************************************
*                                                          *
*   Subroutine Frequency Frame Generation                  *
*                                                          *
*   The frequency of the selected amplitude components     *
*   is written into an array.                              *
*                                                          *
************************************************************

      subroutine freqFR(size, ssapFR, rfrFR)

      integer size, i, j
      real ssapFR(256), rfrFR(size)

      do 5 i=1,size
5        rfrFR(i)=0.0
      j=0
      do 10 i=1,256
         if (ssapFR(i).ne.0.0) then
            j=j+1
            rfrFR(j)=i
         endif
10       continue

      end


************************************************************
*                                                          *
*   Subroutine Amplitude Frame Generation                  *
*                                                          *
*   The selected amplitude components are written          *
*   into a reduced length frame                            *
*                                                          *
************************************************************

      subroutine ampFR(size, ssapFR, rapFR)

      integer size, i, j
      real ssapFR(256), rapFR(size)

      do 5 i=1,size
5        rapFR(i)=0.0
      j=0
      do 10 i=1,256
         if (ssapFR(i).ne.0.0) then
            j=j+1
            rapFR(j)=ssapFR(i)
         endif
10       continue
      end
```

```
************************************************************
*                                                          *
*   Subroutine Phase Frame Generation                      *
*                                                          *
*   According to the frequency of the selected             *
*   components, the original phase frame is                *
*   rewritten into a smaller size frame.                   *
*                                                          *
************************************************************

      subroutine phaseFR(size, rfrFR, dphFR, rphFR)

      integer size, i, j
      real dphFR(256), rfrFR(size), rphFR(size)

      do 5 i=1,size
5          rphFR(i)=0.0
      do 10 i=1,size
         j=rfrFR(i)
         if (j.ne.0.0) then
            rphFR(i)=dphFR(j)
         endif
10    continue


      end


************************************************************
*                                                          *
*   Subroutine Increase                                    *
*                                                          *
*   Works in conjuction with the Variable Amplitude        *
*   Threshold, by furnishing to this module new            *
*   values for the threshold.                              *
*                                                          *
************************************************************

      subroutine increase(flag, greater, a, newa)

      real a, newa, step, lasta
      logical greater, flag
      save

      if (greater) then
         if (.not.flag) step=100.0
            newa=a+step
            lasta=a
      endif
      if (.not.greater) then
         step=step/10.0
         newa=lasta+step
         flag=.true.
```

```
      endif

      end


*****************************************************************
*                                                               *
*   Subroutine Decrease                                         *
*                                                               *
*   Works in conjuction with the Variable Amplitude             *
*   Threshold module. It generates new values for               *
*   the threshold.                                              *
*                                                               *
*****************************************************************

      subroutine decrease(flag, lower, a, newa)

      real a, newa, step, lasta
      logical lower, flag
      save

      if (lower) then
         if (.not.flag) step=100.0
         newa=a-step
         if (newa.le.0.0) newa=0.0
         lasta=a
      endif
      if (.not.lower) then
         step=step/10.0
         newa=lasta-step
         if (newa.le.0.0) newa=0.0
         flag=.true.
      endif

      end


*****************************************************************
*                                                               *
*   Subroutine Output Formatting                                *
*                                                               *
*   This module encapsulates the quantizing process.            *
*                                                               *
*****************************************************************

      subroutine outputformatting(size, qapb, qphb, vmax,
     +                            rapFR, rphFR, qapFR, qphFR)


      integer size, qapb, qphb
      real rapFR(size), rphFR(size), qapFR(size),
     +     qphFR(size), vmax
```

```
      call ampquantizer(size, qapb, vmax, rapFR, qapFR)
      call phasequantizer(size, qphb, rphFR, qphFR)

      end


*************************************************************
*                                                           *
*   Subroutine Amplitude Quantizer                          *
*                                                           *
*   This module implements a linear quantizer with         *
*   midread at the origin. The quantile interval is         *
*   determined by the number of bits (qapb) and by          *
*   the dynamic range (vmax).                               *
*                                                           *
*************************************************************


      subroutine ampquantizer(size, qapb, vmax, rapFR, qapFR)


      integer i, size, qapb
      real rapFR(size), qapFR(size), q, level, vmax, a,
     +     c, qapl

      qapl=real(2**qapb)
      q=vmax/qapl
      do 20 i=1,size
         c=aint(rapFR(i)/q)
         level=c*q-q
         if (rapFR(i).ge.vmax) then
            qapFR(i)=vmax
            goto 20
         end if
10       level=level+q
         a=level+q/2
         if ((rapFR(i).gt.a).and.
     +      (rapFR(i).lt.vmax)) goto 10
         qapFR(i)=level
20       continue

      end
```

```
*****************************************************************
*                                                               *
*   Subroutine Phase Quantizer                                  *
*                                                               *
*   This module implements a linear quantizer with             *
*   midread at the origin. The quantile interval is             *
*   determined by the number of bits (qphb).                    *
*                                                               *
*                                                               *
*****************************************************************

      subroutine phasequantizer(size, qphb, rphFR, qphFR)

      integer i, size, qphb
      real rphFR(size), qphFR(size), q, level, pi, a, c, qphl
      logical tst
      data pi/3.14159265358979324/

      qphl=real(2**qphb)
      q=2*pi/qphl
      do 20 i=1,size
         c=aint(abs(rphFR(i))/q)
         level=c*q-q
10       level=level+q
         a=level+q/2
         if (abs(rphFR(i)).gt.a) goto 10
         if (rphFR(i).lt.0.0) then
            qphFR(i)=-level
         else
            qphFR(i)=level
         endif
20       continue

      end


*****************************************************************
*                                                               *
*   Subroutine Data Rate Calculation                            *
*                                                               *
*   Computes the data rate based on the length of               *
*   the frame (size1), and on the number of                     *
*   amplitude (qapb) and phase (pphb) quantization              *
*   bits.                                                       *
*                                                               *
*****************************************************************

      subroutine datarate(size1, qapb, qphb)

      integer size1, qapb, qphb
      real rate

      rate=(2*size1*(qapb+qphb+8))/(32)
```

```
      write(*,*) size1
      write(*,*) qapb
      write(*,*) qphb
      write(*,*) '-------------------------------------'
      write(*,*) 'Data Rate [kbit/s]:'
      write(*,*) rate
      write(*,*) '-------------------------------------'

      end


************************************************************
*                                                          *
*   Subroutine Synthesizer                                 *
*                                                          *
*   This module encapsulates and controls the speech       *
*   reconstruction process.                                *
*                                                          *
************************************************************

      subroutine synthesizer(begin, end, size1, qapFR, qphFR,
     +                       rfrFR, ntimeFR)

      integer*2 ntimeFR(256)
      integer size1
      real qapFR(size1), qphFR(size1), rfrFR(size1),      +
   pFR(256), phFR(256),gtimeFR(256),rtimeFR(256)
      logical begin, end, valid

      if (.not.end) then
         call frgeneration(size1, qapFR, qphFR, rfrFR, apFR,
     +                     phFR)
         call tdconvertion(apFR, phFR, gtimeFR)
      endif
      call datareduction(begin, end, gtimeFR, rtimeFR, valid)
      if (valid) then
         call ampnorm(rtimeFR, ntimeFR, end)
      endif

      end
```

```
************************************************************
*                                                          *
*   Subroutine 256-point Frame Generation                  *
*                                                          *
*   Expands the reduced length frames to 256-point         *
*   frames by using the frequency information.             *
*                                                          *
************************************************************

      subroutine frgeneration(size1, qapFr, qphFR, rfrFR,
     +                        apFR, phFR)

      integer size1, i, j
      real qapFR(size1), qphFR(size1), rfrFR(size1),
     +     apFR(256), phFR(256)

      do 10 i=1,256
         apFR(i)=0.0
10       phFR(i)=0.0
      do 20 i=1,size1
         j=rfrFR(i)
         apFR(j)=qapFR(i)
20       phFR(j)=qphFR(i)


      end


************************************************************
*                                                          *
*   Subroutine Time Domain Conversion                      *
*                                                          *
*   Uses a speech sinusoidal model to produce a time       *
*   waveform from the amplitudes and phase values.         *
*                                                          *
************************************************************

      subroutine tdconvertion(apFR, phFR, gtimeFR)

      integer i, j
      real apFR(256), phFR(256), gtimeFR(256), pi
      data pi/3.14159265358979324/

      do 20 j=1,256
         gtimeFR(j)=0.0
         do 10 i=1,256
            if (apFR(i).gt.0.0) then
               gtimeFR(j)=gtimeFR(j)+(apFR(i)*
     +          cos(((2*pi*(i-1)*(j-1))/512)+ phFR(i)))
            endif
10       continue
20       continue
```

```
      end

*****************************************************************
*                                                               *
*   Subroutine Data Reduction                                   *
*                                                               *
*   The number of frames that was doubled by the                *
*   Hamming Windows module,in the Analyser, is reduced          *
*   to its original number.                                     *
*                                                               *
*****************************************************************
      subroutine datareduction(begin, end, gtimeFR, rtimeFR,
     +                         valid)

      integer n, i, k
      real gtimeFR(256), rtimeFR(256), x(256), y(256),
     +     htimeFR(256)
      logical begin,end,valid
      save

      if (.not.end) then
         if (begin) then
            n=1
            call hamm(gtimeFR,htimeFR)
            valid=.false.
            do 10 i=1,256
10             x(i)=htimeFR(i)
            else
            n=n+1
            k=mod(n,2)
            if (k.eq.0) then
               call hamm(gtimeFR,htimeFR)
               do 20 i=1,256
20                y(i)=htimeFR(i)
               do 30 i=1,128
                  rtimeFR(i)=x(i)
30                rtimeFR(i+128)=x(i+128)+y(i)
               valid=.true.
            else
               call hamm(gtimeFR,htimeFR)
               do 40 i=1,256
40                x(i)=htimeFR(i)
               do 50 i=1,128
50                x(i)=x(i)+y(i+128)
               valid=.false.
            endif
         endif
      else
         do 60 i=1,256
60          rtimeFR(i)=x(i)
            valid=.true.
      endif
```

```
      end

*****************************************************************
*                                                               *
*   Subroutine Hamming                                          *
*                                                               *
*   All the incoming time frames are Hamming windowed           *
*   before a data reduction is performed.                       *
*                                                               *
*****************************************************************

      subroutine hamm(gtimeFR, htimeFR)

      integer i
      real gtimeFR(256), htimeFR(256)
      data pi/3.14156265358979324/

      do 10 i=1,256
10       timeFR(i)=gtimeFR(i)*(0.54-0.46*
     +              cos((2*pi/255)*(j-1)))

      end


*****************************************************************
*                                                               *
*   Subroutine Amplitude Normalization                          *
*                                                               *
*   Normalizes the reconstructed speech output.                 *
*   This module was adapted from Bashir thesis (9:B-7).         *
*                                                               *
*****************************************************************

      subroutine ampnorm(rtimeFR, ntimeFR, end)

      integer i
      integer*2 ntimeFR(256), s(256)
      real rtimeFR(256), amax, amax1, x(256), z(256), p
      logical end

      if (.not.end) then
         write(50) rtimeFR
      else
         write(50) rtimeFR
         rewind(50)
800      read(50,end=820) z
         amax=0.0
         do 810 i=1,256
            x(i)=abs(z(i))
810         amax=max(amax,x(i))
         write(60) amax
```

```
          goto 800
820       amax=0.0
          rewind(60)
840       read(60,end=860) amax1
          amax=max(amax,amax1)
          goto 840
860       rewind(50)
          p=32760.0/amax
880       read(50,end=899) z
          do 890 i=1,256
890          s(i)=int(p*z(i))
          write(99) s
          goto 880
899       rewind(99)
          endif

      end
```

Bibliography

1. Audio Data Conversion System. Maintenace Manual. D/N 00200-90151-A. Digital Sound Corporation, Santa Barbara, CA. December 1985.

2. Bashir, Flt. Lt. Nadeem A. Phonem Adjustment in Enhanced Speech. MS Thesis, AFIT/GE/ENG/89M-2. School of Engineering. Air Force Institute of Technology (AU), Wright-Patterson AFB OH. March 1989.

3. Harris, Fredric J. "On the use of Windows for Harmonic Analysis with the Discrete Fourier Transform, "Proceedings of the IEEE Vol.66 No.1 : 51-83 (January 1978).

4. Kabrisky, Mathew, Professor. Personnel Discussion. School of Engineering, AFIT (AU), Wright-Patterson AFB OH. November 1989.

5. Kabrisky, Maj Rogers and Flt Lt Bashir. "Reconstruction of Multilated Speech," IEEE AES Magazine. 40-43 (September 1989).

6. Kuc, Roman. "Introduction to Digital Signal Processing." New York : McGraw-Hill Inc. 1988.

7. McAulay, Robert J. and T.F. Quatieri. "Magnitude-only Reconstruction using a Sinusoidal Model of Speech," IEEE Transactions of Acoustics, Speech, and Signal Processing. 27.6.1-27.6.4 (March 1984).

8. McAulay, Robert J. and T.F. Quatieri. "Speech Transformation Based on a Sinusoidal Representation," IEEE Transactions of Acoustics, Speech, and Signal Processing. : 489-492 (March 1985).

9. Parsons, Thomas W. "Separation of Speech from Interfering Speech by means of Harmonic Selection," The Journal of the Acoustical Society of America, Volume 60 : 911-918 (October 1976).

10. Sivian L.J. "Speech Power and Its Measurement." Bell System Technical Journal, Volume 8 : 646-661 1929.

11. Sklar, Bernard. "Digital Communications Fundamentals and Applications." New Jersey : Prentice Hall, 1988.

12. SPIRE 17.2 Preliminary User's Guide. Speech Communications Group, Research Laboratory of Electronics, Massachussetts Institute of Technology, February 1986.

13. Stremler, Ferrel G. "Introduction to Communication Systems," Addison-Wesley Publishing Company, 1982.

14. Tanenbaum, Andrew S. "Computer Networks." New Jersey: Prentice Hall, 1988.

Vita

Luis M. F. Alenquer was born on 27 October 1953 at Mem Martins, Portugal. He received the degree of Electrical Enginner from Instituto Superior Tecnico (Lisbon) in 1978. In June 1988, he entered the School of Engineering, Air Force Institute of Technology.

Permanent Address: Rua Ilha de Coloane 7-3Esq
2725 Mem Martins
Portugal

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT APROVED FOR PUBLIC RELEASE DISTRIBUTION UNLIMITED
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GE/ENG/90M-1	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION School of Engineering 6b. OFFICE SYMBOL (If applicable) AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) Air Force Institute of Technology Wright-Patterson AFB,OH 45433	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
Rule Based Sinusoidal Encoding of Speech UNCLASSIFIED

12. PERSONAL AUTHOR(S)
Alenquer,Luis M.F. Capt Portuguese Air Force

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
MS Thesis	FROM _____ TO _____	March 90	112

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Speech Processing, Speech Analyzer, Speech Synthesizer, Harmonics/Peaks Selection.
17	02		

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Thesis Chairman : Matthew Kabrisky, PhD

Professor of Electrical Engineering

Cont/-

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code)	22c. OFFICE SYMBOL
Dr.Matthew Kabrisky Professor, GS-15	(513) 255-5276	AFIT/ENG

DD Form 1473, JUN 86 Previous editions are obsolete. SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

Continued from block 19: Abstract

A system was developed to investigate the data rate necessary to transmit speech using a rule based sinusoidal model. The system consists of a speech analyzer and a synthesizer. The analyzer outputs discrete frequencies and quantized amplitudes and phases of selected speech spectral components. The synthesizer reconstructs speech from these components based on a sinusoidal model. The selection of spectral components for voiced speech regions is based on the detection of harmonics of the fundamental frequency. To obtain a specific number of spectral components, a variable amplitude threshold is applied to the detected harmonics and their nearest neighbors. For unvoiced regions only the variable amplitude step is applied. The lowest data rate obtained for toll quality speech was about 18 Kbps. This system was implemented in Fortran 77 on a VAX 11/780 computer. Visual analysis of speech was provided by the software package SPIRE (Speech and Phonetics Interactive Research Environment).