A SIMULATION OF A MICROCOMPUTER-BASED INTRUSION DETECTION SYSTEM

by

JOHN WARREN BARTHOLOMEW

B. S., Kansas State University, 1982

A MASTER'S THESIS

submitted in partial fulfillment of the

requirements for the degree

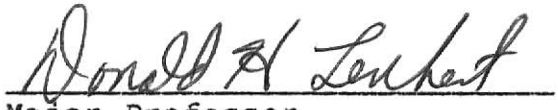
MASTER OF SCIENCE

Department of Electrical Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1983

Approved by:



Major Professor

TABLE OF CONTENTS

LIST OF FIGURES

1.0 INTRODUCTION

In order to determine how the market will react to a new product, a marketing engineer might make a mathematical model of the market and simulate the system on a computer. Several factors could be varied, and the results obtained from a computer simulation. The simulation is useful in providing answers to the "what if" questions -- "what if" the inflation rate decreases by 2%, "what if" there were 10% fewer buyers, etc. In the same way, electrical circuits can be mathematically modeled, and their performance evaluated from the results of a computer simulation. When digital signal processing is implemented on a microprocessor, the effects of using a finite word length can be readily obtained from a computer simulation. By using a computer simulation, one can easily change and evaluate the effects of such factors as the word length, methods of calculating filters, and the amount of noise in the system.

This paper deals with the simulation of a microcomputer-based intrusion detection system and the effects of using a finite word length in the calculation of the digital filters. The research was funded and directed by Sandia National Laboratories, Albuquerque, New Mexico. The initial design

for the system was done at Sandia, while the simulation and coding of the microcomputer implementation was done at Kansas State University. The filters were first checked for overflows by using a maximum input signal. One filter in the low band needed a reduction in gain, and two filters in each band -- the high pass filter in the quadrature phase just before the cross multiply, and the first filter after the cross multiply -- were increased in gain. The modified filters were then checked for underflows by using a minimum input signal. The last filter in both the high band and the mid band had significant underflows and should be removed or modified to retain the signal present in the previous stage. The modifications to the filters should improve the system's performance. Some subjects relating to this are covered as background material first: digital signal processing, intrusion detection systems, and the effects of using finite word length as related to digital filters. Then the filter algorithm is discussed, and the design of the filters is covered. Finally, the simulation itself is presented. A description of the microcomputer implementation is also included.

1.1 DIGITAL SIGNAL PROCESSING

For many reasons, digital signal processing has replaced analog signal processing in certain areas. Besides the obvious cases where no analog equivalent of a digital system exists, (such as predictive filters) the advantages of using

digital filters can outweigh the advantages of using an analog system.

Where a microprocessor is used in the implementation, changing the characteristics of a filter is a simple matter of changing the coefficients or the form of the filter. In the analog case, circuit components may need to be changed or entire sections of the circuit rebuilt. The digital filter will not change its characteristics over time, and doesn't require adjustment from system to system as an analog filter would. Finally, a very complex set of digital filters can be housed in a much smaller package than the analog equivalent.

The drawbacks related to using digital filters exist since the signals must be represented as discrete samples and the calculations required to do the digital filters are usually done in fixed point precision. The speed of the A/D converter taking the samples, or the sheer number of calculations required to implement the system (and thus the speed of the microprocessor) may affect the bandwidth of the system. This paper deals more with the effects produced by using a finite word length than with the problems with using discrete sampling.

1.2 INTRUSION DETECTION SYSTEMS

In cases where a piece of property is to be secured, a system that detects an intruder -- someone or something that is not supposed to be in the area surrounding the piece of property -- is normally used. Usually, digital signal

processing is chosen, and the system is based on a microprocessor. An algorithm is devised that picks out signals that represent intruders, causing an alarm condition. Much work has been done to reduce the number of "nuisance" alarms -- an alarm where no intruder is present.

In this system, the simulation being done is part of an iterative process. The intrusion detection system was designed and a decision was made to implement it on a microprocessor using 16 bit data (with 32 bit partial sums). Once this decision was made, the simulation was done to see what effect this particular word length and number representation would have when the input signal was either a maximum or minimum magnitude, and how the overall performance of the system was affected. In general, the programs were set up to simulate any word length from 1 - 56 bits. Results from the simulation are used to make changes to the system to improve its performance.

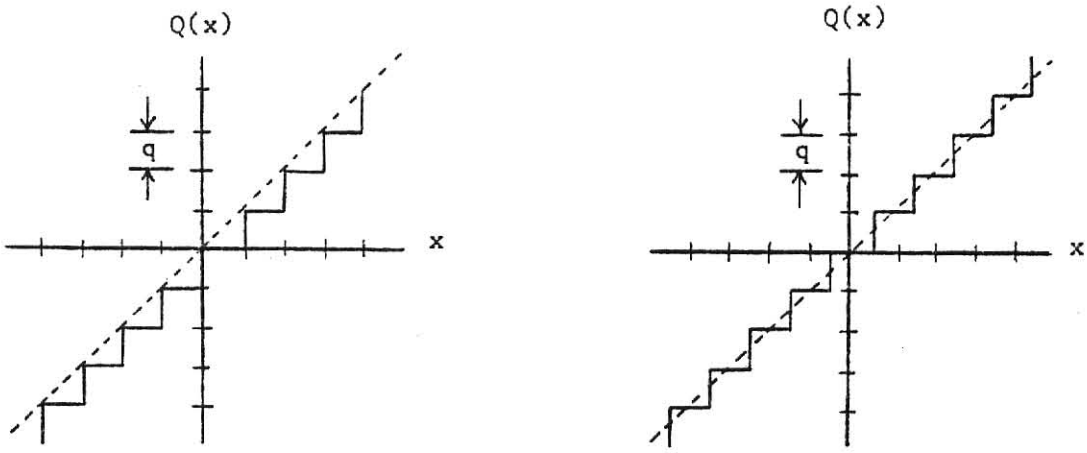
2.0 EFFECTS OF FINITE WORD LENGTH

When a system using digital filters, such as an intrusion detection system, is implemented on a microcomputer, the need for high speed usually means that the word length (which is based on the word length of the microprocessor) is necessarily short. The form in which the numbers are represented is covered first, then the various effects of using a finite word length are covered : coefficient quantization, input quantization, product quantization, scaling considerations,

deadbands, and limit cycles.

2.1 NUMBER REPRESENTATION

In this implementation, all numbers are represented by two's complement fixed point numbers. Coefficients and signals are 16 bits long with one sign bit, one integer bit, and 14 fraction bits. Partial sums are kept as 32 bit numbers. All results are truncated to the 16 bit format. The effect of quantizing the data by truncation as opposed to rounding is shown in figure 1. The maximum absolute error of rounding is smaller than that of truncation.



truncation rounding

Figure 1

2.2 COEFFICIENT QUANTIZATION

Due to the finite word length of the microcomputer implementation, the coefficients of the filters must be

quantized. Since this quantization is done only once, there is no restriction on how fast it is done; thus rounding, which has a smaller maximum absolute error but requires more calculations and thus more time, can be chosen over truncation. In our implementation the coefficient word length is the same as the data word length, 16 bits.

The most pronounced effect of quantizing the filter coefficients is the movement of the zeros and poles of the filter. [1,2,3,4,6] If these zeros or poles lie close to unit circle, the filter may become unstable when quantized. [1,3,4] The actual movement of the zeros or poles can be calculated. [3,4] There are also methods for estimating the required coefficient word length for a desired error, either directly or statistically. [2]

Kaiser has shown that for filters higher than 1st or 2nd order, the direct form is less accurate, and the cascade form with each section a 1st or 2nd order system is preferred. [7]

2.3 INPUT QUANTIZATION

The quantizing of the input to each filter can be looked upon as producing a noise input to the filter. [1,3,4,5] The effect of using truncation as opposed to rounding is different in that the statistical mean of the noise produced by the quantization of the input to the filter is not zero for truncation. The error produced by the A/D can also be considered as an input quantization error, producing a similar additive noise term. [1,4,5,6] As long as the input is of

sufficient magnitude, the noise produced by the quantization will stay below the SNR required by the system. If needed, the word length required to keep a certain SNR can be calculated directly. A method for calculating the required word length is given by Oppenheim and Schafer. [5]

2.4 PRODUCT QUANTIZATION

The quantization of the result of any arithmetic calculation can also be viewed as producing additive noise. [1,2,3,4,5] Since the order in which the calculations are made may make a difference in the final result, the form in which the filter is realized may affect the accuracy of the filter. There are no simple guidelines for choosing the most accurate form of a filter; one may have to resort to simulating the filter on a larger machine and studying the accuracy of the various forms. [6] Better precision can be obtained by retaining a word length of 2N bits after N by N bit multiplies, keeping a sum of 2N bits, then rounding or truncating the final result. It has also been shown that the zeros of a filter tend to attenuate the noise produced by the product and input quantization. [3]

2.5 SCALING CONSIDERATIONS

To keep the signal within the dynamic range of the fixed word length, some signal scaling -- multiplying the input to the filter by a constant -- may be required. One of the problems encountered with digital filters is internal

overflows in the partial sums. Two methods for ensuring that no overflow occurs both use a scaling factor just before the input to the filter. [1,2] The first method calculates the scaling factor by summing the magnitudes of the inputs to a particular summing node in the form of the filter that is used. [2] The second method forces the instantaneous energy in the output of the summing node to be less than the energy in the input to the filter. The scaling factor obtained in both methods is an absolute minimum; in practice, a larger value can normally be used.

Scaling can also be used to boost the magnitude of particular signal up to a workable value. This form of scaling also improves the SNR of the filter by using more of the dynamic range.

2.6 DEADBAND AND LIMIT CYCLES

Two types of limit cycles exist. The first type occurs for zero or near zero inputs and the second type occurs when an overflow in a summing node causes a sign change. The range of input values around zero that produce a limit cycle is referred to as a deadband.

The zero or near zero input limit cycles are attributed to the quantization of values to a finite word length. [1,2,3,5] Since this type of error occurs only in a recursive digital filter (Infinite Impulse Response or IIR filters), Finite Impulse Response (FIR) filters are not affected. If the specifications state that the output of a filter with zero

input must be below some magnitude, there are methods of determining the bounds of the limit cycle, and thus the required word length. [2,5]

The second type of limit cycle, overflow oscillation, can be corrected in two ways. Scaling, as discussed before, can be used to eliminate the internal overflows. Since the oscillation is due to the sum changing sign when an overflow occurs, the oscillation can also be eliminated by setting the sum to the maximum allowable value and letting the filter saturate. [1] Though this may produce an undesirable distortion of the waveform, it may be better than large oscillations.

3.0 FILTER DESIGN

Now that we know some of the quantization errors that might occur in the implementation of digital filters in a fixed length format, the specific filters used in the algorithm can be designed. The simulation deals with this part of the overall iterative system design by showing needed changes to the initial design of the filters to keep the signals within the dynamic range of the fixed point representation. Besides eliminating erroneous results from filters that have overflows, the results from the simulation allow other filters to have increased gain, thus improving the SNR of the system. Since the performance of the system is measured in part by the SNR and the number of nuisance alarms, the changes made will improve the performance of the system.

Several ideas were used to reduce the number of calculations required to compute some filters. Since only the lower frequencies are of interest, and because the Nyquist criterion for sampling states that the sampling rate need be only twice the maximum frequency of interest, we can just save every n'th sample, provided that we use a low pass filter to eliminate the frequencies above one-half the sampling rate. The FIR filters (chosen because they are not recursive and thus need not be calculated every time) thus eliminate the higher frequencies, reduce the sampling rate, and reduce the number of calculations required to compute filters that follow it.

The specifications for the filters will be given first, then the initial design of the filters will be covered. The entire algorithm will be described next, then the overall design of the system will be discussed. Finally, the filters will be checked for errors due to quantization.

3.1 FILTER SPECIFICATIONS

The first step in designing the filters is to specify their characteristics. The filter specifications given by Sandia Laboratories were specifications for certain bandwidths, dB roll-off, and percent ripple in the passband. These specifications were used to design the types of filters and calculate the coefficients for the filters.

3.2 INITIAL DESIGN

We determined the coefficients for the IIR filters by using LPDES, HPDES, and BPDES -- utility programs written at Kansas State on a NOVA 4X computer. (see Appendix 6.2 of Ahmed and Natarajan for listings of the programs) [1] The coefficients sent to us from Sandia match those that were generated at Kansas State. The FIR filters were redesigned to reduce the number of calculations and we determined the coefficients using FIRFILT, a program written at Kansas State on a NOVA 4X computer. (see Appendix 7.1 of Ahmed and Natarajan for a listing of the program) [1]

All of the filters were designed for unity gain, and were generally implemented in the direct form. In order to prevent overflows and keep the coefficients within the range of the fixed word length representation, the higher order IIR filters were implemented as two cascaded filters.

The design for the complex predictor and associated signal strength equalizer was done by Dr. NasirAhmed at Kansas State University.

The type of filter, the sampling rate, the break frequencies, and the coefficients for the filter are listed in appendix B. Also, the amplitude vs. frequency and phase vs. frequency plots are shown in the appendix B.

3.3 ALGORITHM DESCRIPTION

The algorithm can be divided into three sections : high band channel, mid band channel, and low band channel, as shown

in appendix A. The function of each of the filters will be discussed for each channel. Infinite Impulse Response (IIR) filters are labelled IIRn (where n takes on the values 1 through 15) and Finite Impulse Response (FIR) filters are labelled FIRn (where n takes on the values A, B, C, D, E, and G).

3.3.1 High Band Channel The 6 pole bandpass filter IIR1 limits the range of frequencies for this channel. The next block allowed for a variable gain to be introduced. The 1 pole lowpass filter IIR2 reduces the amplitude of the frequencies above the passband of the channel while the 1 pole highpass filter IIR3 provides a linear relationship between the amplitude and the frequency within the passband. The output of the IIR2 filter is limited to +-2.8 V. Since the two sides of this channel are 90 degrees out of phase, multiplying the two together produces I x dQ/dt. The signal at this point now includes a DC value. The 10 weight lowpass filter FIRA and 23 weight lowpass filter FIRB pass the lower frequencies but are mainly used to reduce the sampling rate from 240 samples per second (sps) to 10 sps. Two FIR filters were used instead of just one since the two required fewer calculations and less storage than just one FIR filter. The 3 pole lowpass filter IIR4 passes only the lower frequencies. The output of the IIR4 filter is limited to +-1.0 V and the 1 pole lowpass filter IIR5 passes only the near DC frequencies. The effect at this point is to produce the DC component of the I x dQ/dt

signal. The final step is to detect DC levels above and below a certain level.

3.3.2 Mid Band Channel The 22 weight lowpass filter FIRG mainly serves to reduce the sampling rate from 240 sps to 40 sps. The 8 pole bandpass filter IIR6 limits the range of frequencies for this channel. The next block allowed a variable gain to be introduced. The 1 pole lowpass filter IIR7 reduces the amplitude of the frequencies above the passband of the channel while the 1 pole highpass filter IIR8 provides a linear relationship between the amplitude and the frequency within the passband. The output of the IIR7 filter is limited to +-2.8 V. Since the two sides of this channel are 90 degrees out of phase, multiplying the two together produces I x dQ/dt. The signal at this point now includes a DC value. The 16 weight lowpass filter FIRC passes the lower frequencies but is mainly used to reduce the sampling rate from 40 sps to 5 sps. The 3 pole lowpass filter IIR9 passes only the lower frequencies. After resampling to reduce the sampling rate from 5 sps to 2.5 sps, the signal is limited to +-1.0 V, and the 1 pole lowpass filter IIR10 passes only the near DC frequencies. The effect at this point is to produce the DC component of the I x dQ/dt signal. The final step is to detect DC levels above and below a certain level.

3.3.3 Low Band Channel The 16 weight lowpass filter FIRD mainly serves to reduce the sampling rate from 40 sps to 10 sps. The 8 pole bandpass filter IIR11 limits the range of frequencies for this channel. The next block allowed a

variable gain to be introduced. The signal strength equalizer is used to keep the convergence rate of the complex predictor uniform since signals of larger amplitude will converge faster than signals of lower amplitude. Since the clutter in this band of frequencies is more sinusoidal than the signal produced by the intruder, the complex predictor is useful in eliminating the sinusoidal clutter. The 1 pole lowpass filter IIR12 reduces the amplitude of the frequencies above the passband of the channel while the 1 pole highpass filter IIR13 provides a linear relationship between the amplitude and the frequency within the passband. The output of the IIR12 filter is limited to +-2.8 V. Since the two sides of this channel are 90 degrees out of phase, multiplying the two together produces I x dQ/dt. The signal at this point now includes a DC value. The 9 weight lowpass filter FIRE passes the lower frequencies but is mainly used to reduce the sampling rate from 10 sps to 2.5 sps. The 2 pole lowpass filter IIR14 passes only the lower frequencies. After resampling to reduce the sampling rate from 2.5 sps to 1.25 sps, the 1 pole lowpass filter IIR15 passes only the near DC frequencies. The effect at this point is to produce the DC component of the I x dQ/dt signal. The final step is to detect DC levels above and below a certain level.

3.4 OVERALL DESIGN

The effect of combining the filters in the entire system may not always result in a unity gain output. For this

reason, amplitude vs. frequency plots were made for the output after each filter to determine maximum amplitude, as shown in appendix C. The coefficients for the filters were then adjusted to obtain a unity gain wherever possible. Since the filter coefficients must be representable in the fixed point format, the gain of a filter can not always be arbitrarily set, but may be limited to some maximum value. The following changes to the coefficients were made:

High Band: Multiply IIR3 numerator by 7.4225190
Multiply FIRA by 2.0

Mid Band: Multiply IIR8 numerator by 2.8180921
Multiply FIRC by 2.0

Low Band: Multiply IIR13 numerator by 3.3534541
Multiply FIRE by 2.0

The only point requiring further adjustment was the output of the complex predictor, which was boosted to unity gain with a single external multiply in both the inphase and quadrature signals.

3.5 CHECK FOR QUANTIZATION ERRORS

All of the filters were checked for errors produced from quantization of the coefficients and for other possible errors. Since the filters were implemented mostly in direct form, the quantization of the coefficients could affect the location of the poles and zeroes of the filter, and thus the performance of the filter. The only noticeable overflow that

showed up during the simulation was an internal overflow in IIR11, which was corrected by reducing the gain of the filter. Both IIR5 and IIR10 had significant underflows with a minimum signal, and should either be eliminated or modified to retain the signal present in the previous stage.

4.0 OUTPUT SIGNALS FROM ALGORITHM

Now that the entire system has been designed, the simulation will provide some measure of the performance of the system. A comparison is made between the outputs at various points of the double precision version and the same output points of the fixed point version.

First, the program used to implement the algorithm will be described, then the input test signals will be covered. Finally, the output signals will discussed.

4.1 PROGRAM DESCRIPTION

The programs used to do the simulation are given in appendix D. Basically, the main program consists of calls to general IIR and FIR filter subroutines and provides the proper flow for the signals in the algorithm. The simulation program also includes calls to routines to do the arithmetic operations in a specified fixed point format. The fixed point format is simulated by multiplying the number by 2 ** n (where n is the number of bits to be simulated), performing either truncation or rounding, and dividing by 2 ** n. The program is general in that it allows the user to specify the

input data, the filter coefficients, the format (for the simulation), and the output. In all, there are 6 programs, one program for the double precision version and one program for the fixed point version for each of the three bands.

The programs were first run on a NOVA 4X, but as soon as it was realized that the number of records in some of the input files was too large to be represented by the 16-bit integers on a NOVA 4X, and that some program runs were taking 60 hours to complete, the work was moved over to a VAX 11/750. The longest program still took 12 1/2 hours to complete, but most of the programs take only a few minutes.

4.2 GENERATING INPUT TEST SIGNALS

Four types of input test signals were generated since actual test data could not be obtained from Sandia Laboratories:

1) 1V maximum sine wave
   - test maximum magnitude for overflows, see that dynamic range is used to its full extent

2) 0.05V minimum sine wave
   - test minimum magnitude for loss of signal due to fixed word length

3) 1V maximum sine wave with noise, 20dB SNR

4) 0.05V minimum sine wave with noise, 20dB SNR
   - check both maximum and minimum magnitudes for response to added noise

The original specification for a minimum signal was 0.01, but the following calculations show that the minimum signal should be 0.05.

HIBAND: Within the pass band, the lower frequencies

passed by IIR3 are reduced by a factor of 10. When the I x dQ is formed, the signal is further reduced by the multiplication. The magnitude of the final signal is given by:

A = Amin * 0.1 * Amin

Since the magnitude of the final signal (A) is the largest value of a minimum sine wave, 2 bits were chosen to represent the dynamic range of the minimum sine wave, making the final signal (A) equal to 2 ** -12. Solving for Amin yields:

Amin = (10 * 2 ** -12) ** 0.5

Amin = 0.0494

Thus, 0.05 is aproximately the minimum signal allowable for the high band.

MIDBAND: Within the pass band, the lower frequencies passed by IIR8 are reduced by a factor of 4. When the I x dQ is formed, the signal is further reduced by the multiplication. The magnitude of the final signal is given by:

A = Amin * 0.25 * Amin

Choosing A = 2 ** -12 and solving for Amin yields:

Amin = (4 * 2 ** -12) ** 0.5

Amin = 0.03125

Thus, 0.03 is aproximately the minimum signal allowable for the mid band.

Since the low band contains a signal strength equalizer, and the magnitude of the signal would depend upon the quantization effects, no easy determination could be made

about the minimum signal allowable. A decision was made to use the 0.05 as the minimum signal throughout the algorithm.

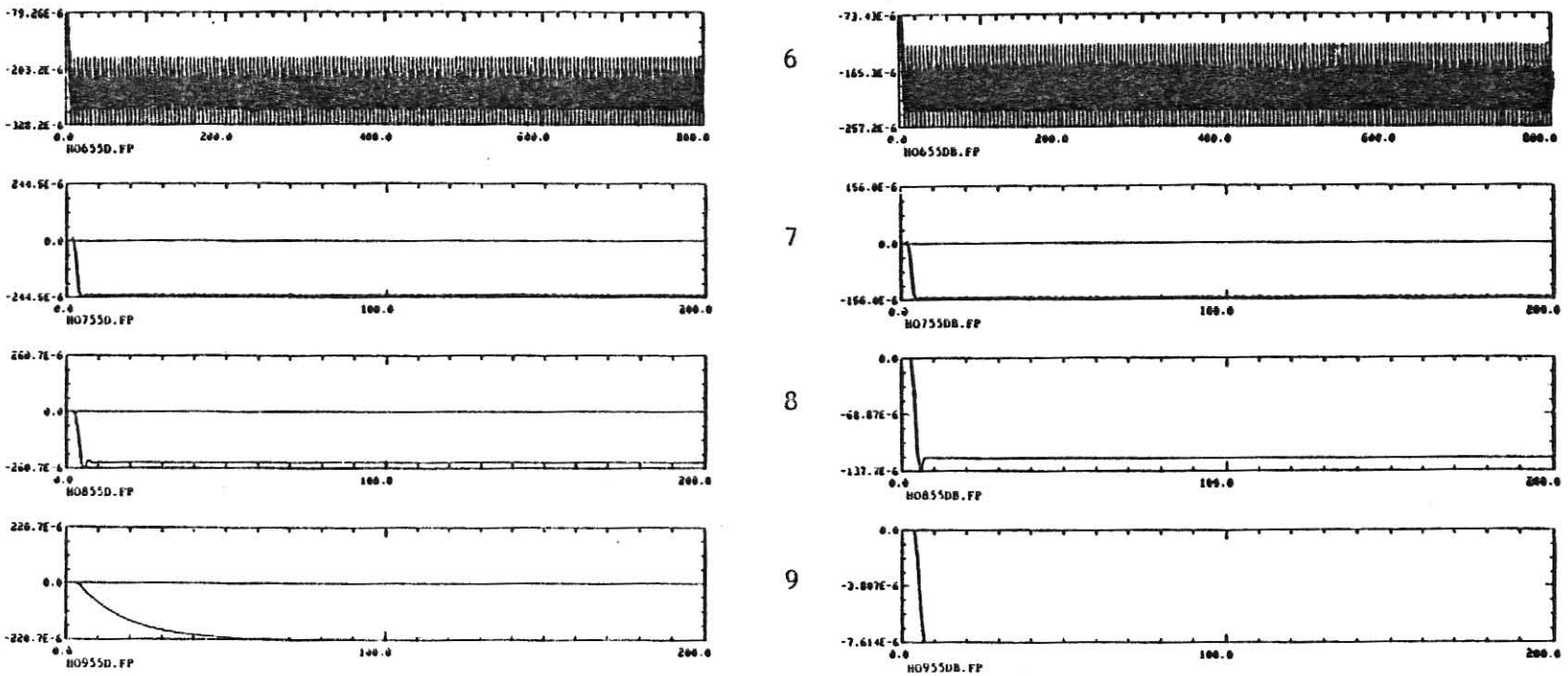
The plots of the amplitude vs. frequency for the outputs of the filters as shown in appendix C were again consulted to determine the frequencies at which the maximum and minimum amplitudes occurred. The lengths of the input files were chosen so that the steady state response of the system could be shown. A utility program called DSINEGEN produced the sine waves, and another utility program called DATGEN produced the white noise. Another program called CHWHITE was written to allow the user to manipulate the input files and combine them into the desired input file. The input signal for the low band is different in that the main sine wave represents the clutter and a short block of a different frequency sine wave represents the intruder. The specifications for each of the input files and which outputs used them can be found in appendix E.

4.3 DESCRIPTION OF OUTPUT SIGNALS

Plots of all the output signals are given in appendix F. The numbers on the left hand side are the magnitude of the signal, and the numbers along the bottom of the plot are the sample number. The filename below the lower left corner is the file from which the plot was obtained. A description of the format of the filename is given in appendix F. One should note that the plots of the fixed point precision outputs represent the 32 bit format and the output will be truncated

to the 16 bit format before entering the next filter. Only a few of the plots are repeated here to show the effects of finite word lengths, and the effects of noise on both the double precision and fixed point versions. Refer to the filter block diagram (appendix A) and the first page of appendix F for a description of the output points and the labelling of the output signals respectively. One should keep in mind that the smallest representable value in this format is about 0.0000061 (61 * 10 ** -6).

The first set of plots shown in figure 2 displays the outputs of the last 4 stages of the high band without noise. Figure 2a shows the double precision outputs for +-0.05 V input while figure 2b shows the fixed point outputs for +-0.05 V input. The plots in figure 3 are the outputs of the last 4 stages of the high band with noise. Figures 3a and 3b show the double precision and fixed point outputs, respectively, for +-0.05 V input with 20 dB Signal to Noise Ratio (SNR). What is most evident in the noise-free case of figures 2a and 2b is that the fixed point version has a much smaller magnitude, due to quantization errors. In fact, the final filter will produce a zero output if the output is truncated to 16 bits. Even with all 32 bits saved, the internal calculations in the last filter are not accurately representable in the present format. The final filter should be removed or modified to retain the information that is present in the previous filter. In the noisy case of figures 3a and 3b, as with the noise-free case, the fixed point



2a. Double precision

2b. Fixed point

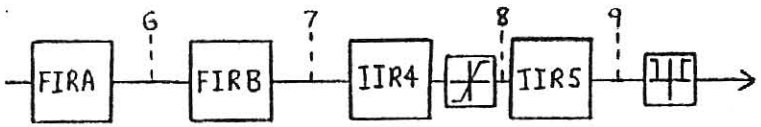
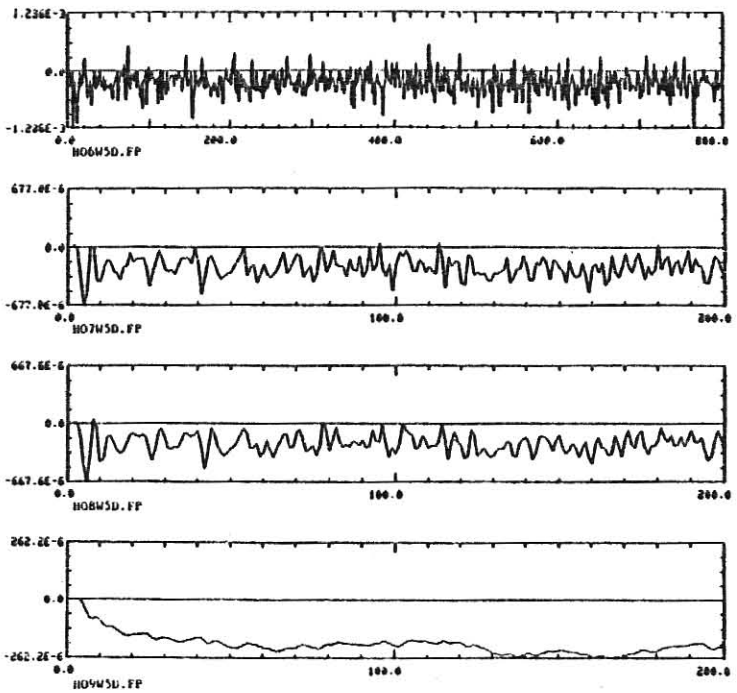
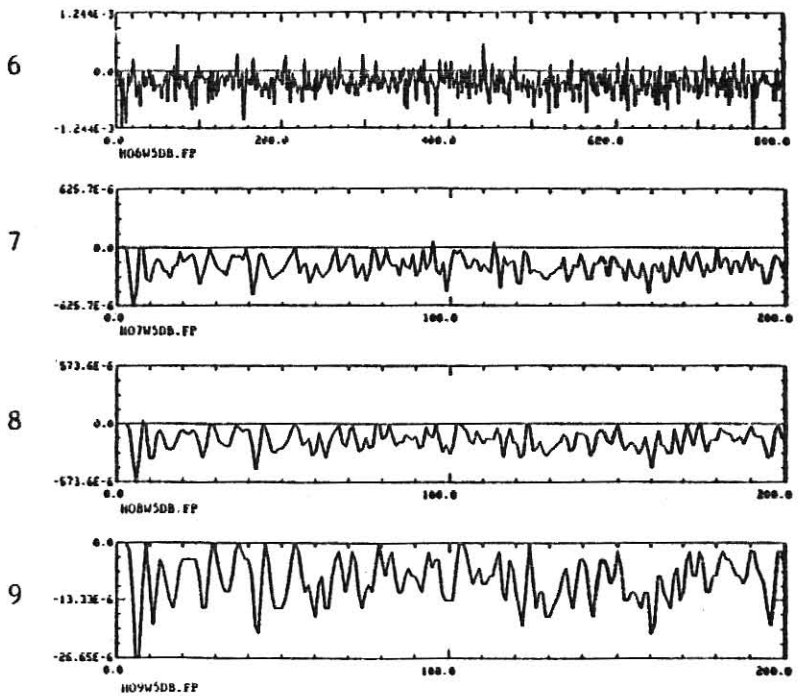


Figure 2. HIGH BAND: Outputs of last 4 stages with +-0.05 input without noise



3a. Double precision



3b. Fixed point

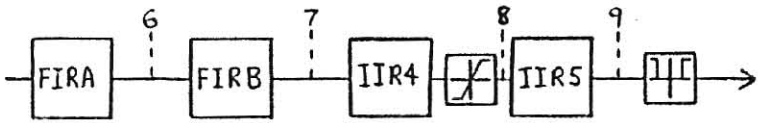
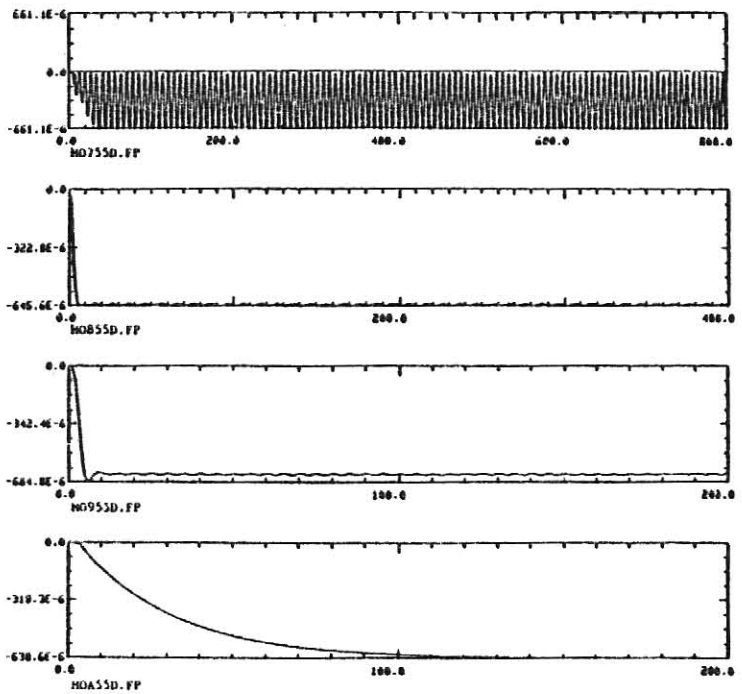


Figure 3. HIGH BAND: Outputs of last 4 stages with +-0.05 input with noise

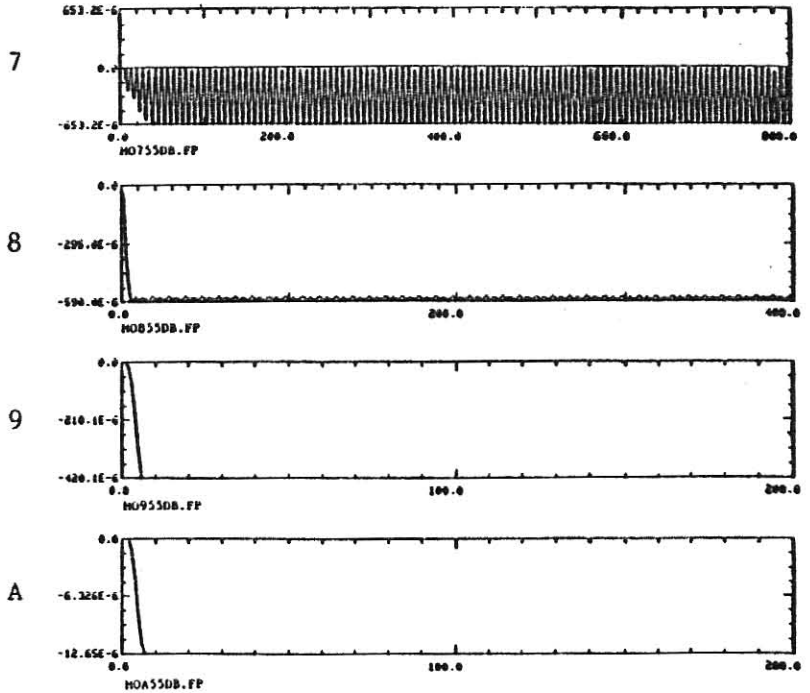
version has a smaller magnitude than the double precision version, but again, the final filter would produce zero output if the output was truncated to 16 bits.

The next set of plots shown in figure 4 displays the outputs of the last 4 stages of the mid band without noise. Figure 4a shows the double precision outputs for +-0.05 V input while figure 4b shows the fixed point outputs for +-0.05 V input. The plots in figure 5 are of the last 4 stages of the mid band with noise. Figures 5a and 5b show the double precision and fixed point outputs, respectively, for +-0.05 V input with 20 dB SNR. What is most evident in the noise-free case of figures 4a and 4b is that the fixed point version has a slightly smaller magnitude, due to quantization errors. As with the high band, the final filter would produce zero output if the output were truncated to 16 bits. The final filter should be removed or modified to retain the information still present in the previous filter. In the noisy case of figures 5a and 5b, as with the noise free case, the fixed point version has a smaller magnitude than the double precision version. Quantization effects in the next to the last output signal are quite evident, but should not effect the detection of an intruder. Again, the final filter would produce zero output if the output were truncated to 16 bits.

The next set of plots shown in figure 6 displays the outputs from the complex predictor, IIR12, and IIR13 in the low band without noise. Figure 6a shows the double precision outputs for +-0.05 V input while figure 6b shows the fixed



4a. Double precision



4b. Fixed point

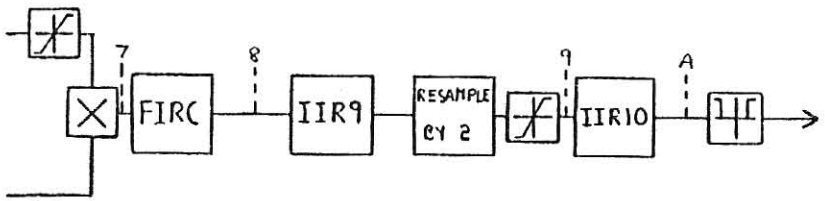
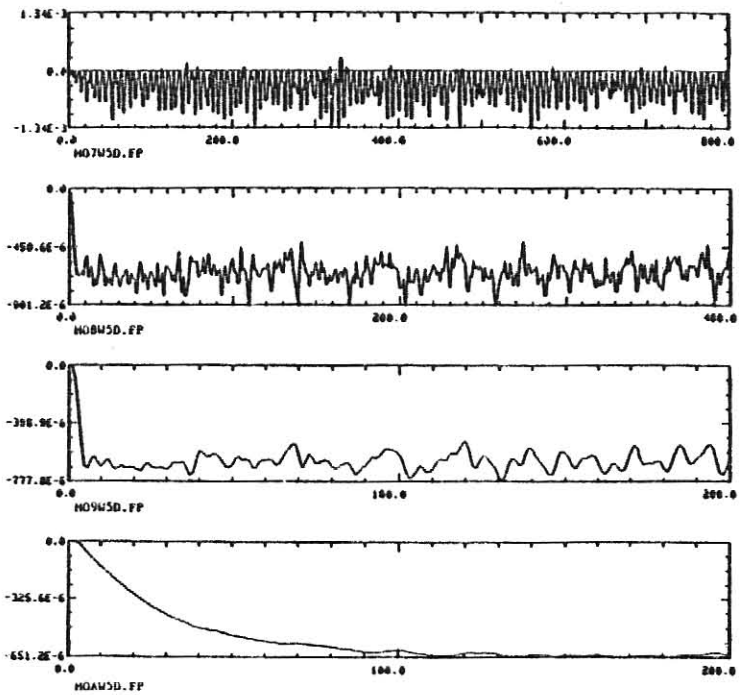
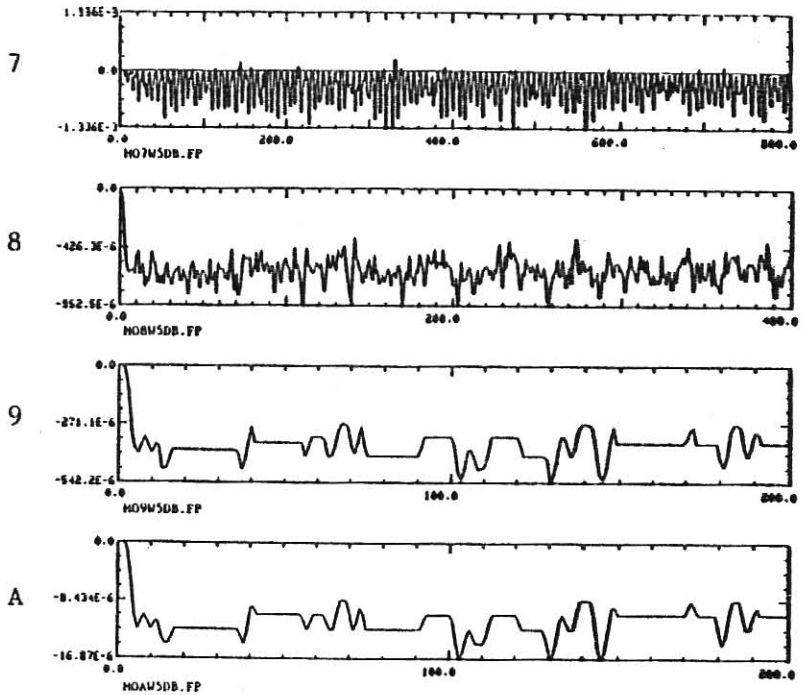


Figure 4. MIDBAND: Outputs of last 4 stages with +-0.05 input without noise





5a. Double precision

5b. Fixed point

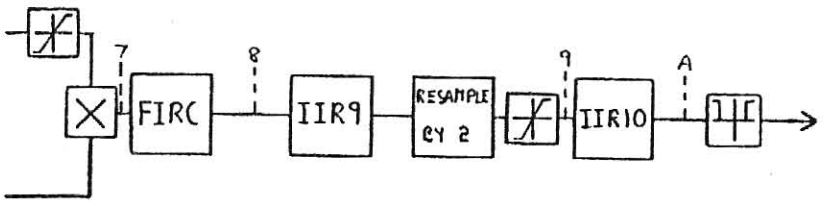
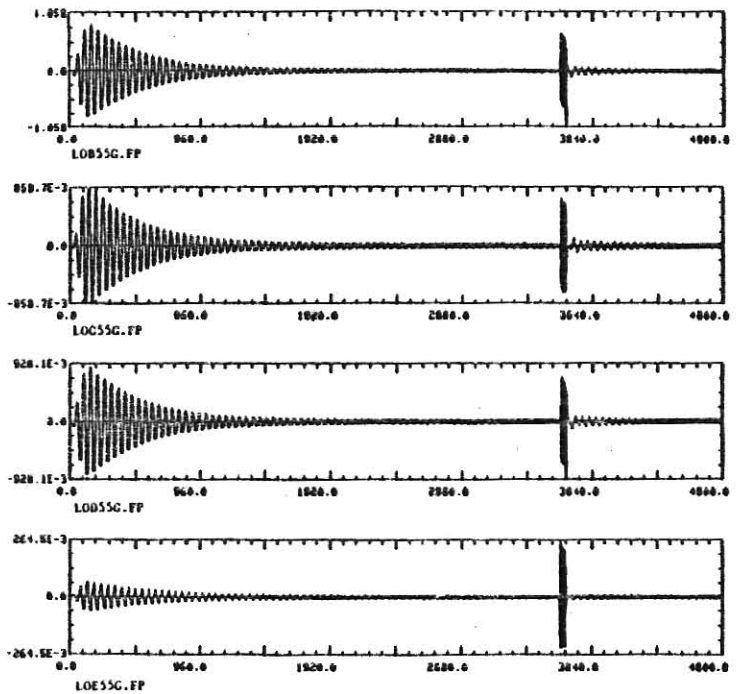


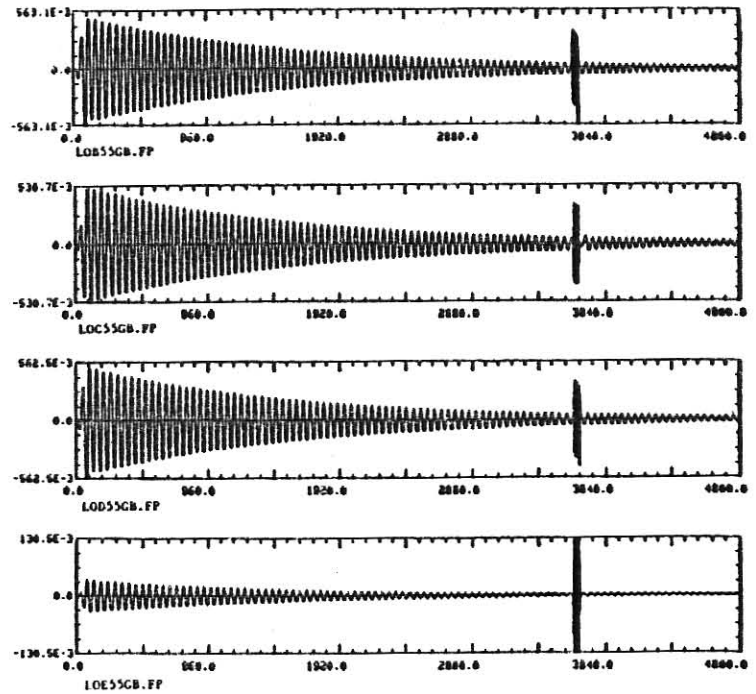
Figure 5. MIDBAND: Outputs of last 4 stages with +-0.05 input with noise

point outputs for +-0.05 V input. The plots in figure 7 are the outputs of the complex predictor, IIR12, and IIR13 in the low band with noise. Figures 7a and 7b show the double precision and fixed point outputs, respectively, for +-0.05 V input with 20 dB SNR. What is most evident in both the noise-free case of figures 6a and 6b and the noisy case of figures 7a and 7b is that the fixed point version has a much smaller magnitude, due to quantization errors. Most of these quantization errors occur in the signal strength equalizer, which uses very small values. One can also see that the larger magnitude signal converges faster than the smaller magnitude signal.

The next set of plots shown in figure 8 displays the outputs of the last 4 stages of the low band without noise. Figure 8a shows the double precision outputs for +-0.05 V input while figure 8b shows the fixed point outputs for +-0.05 V input. The last set of plots in figure 9 displays the last 4 stages of the low band with noise. Figures 9a and 9b show the double precision and fixed point outputs, respectively, for +-0.05 V input with 20 dB SNR. What is most evident in both the noise-free case of figures 8a and 8b and the noisy case of figures 9a and 9b is that the fixed point version has a much smaller magnitude, due to quantization errors. However, an intruder at this signal level could still be detected.



6a. Double precision



6b. Fixed point

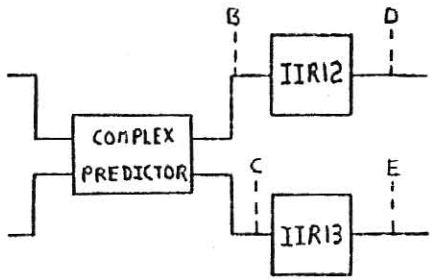
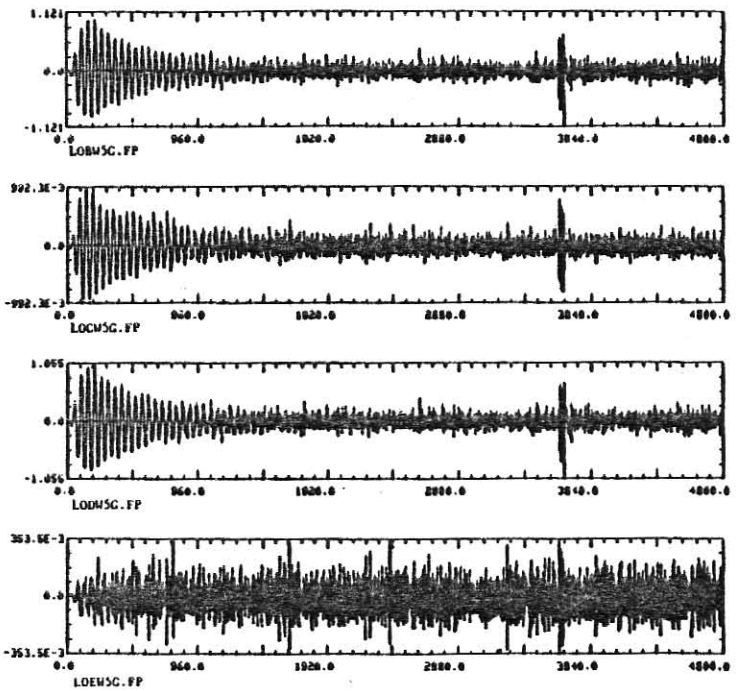
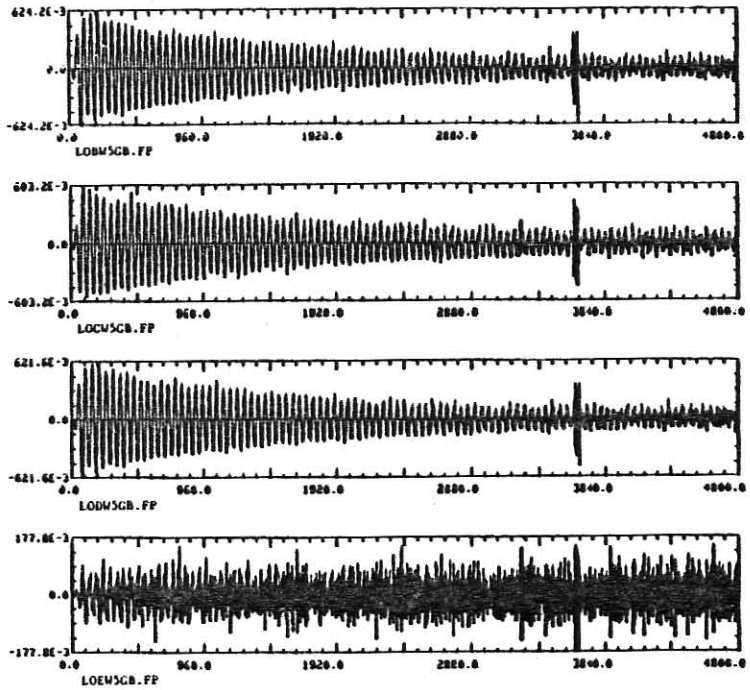


Figure 6. LOW BAND: Outputs after complex predictor with +-0.05 input without noise



7a. Double precision



7b. Fixed point

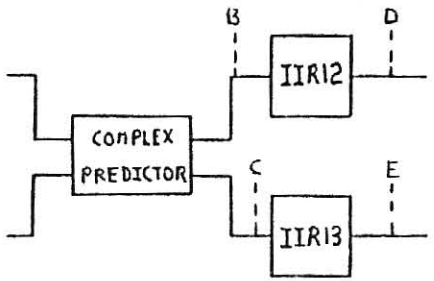
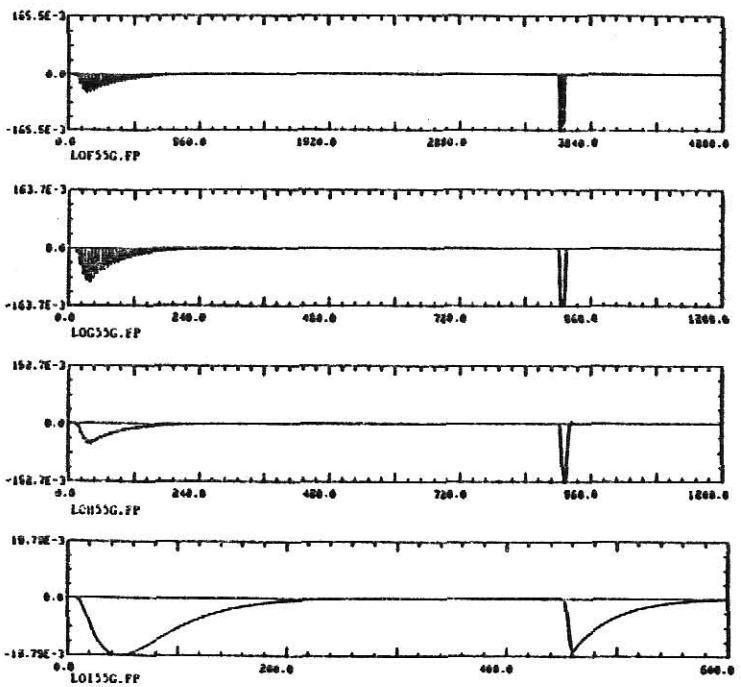
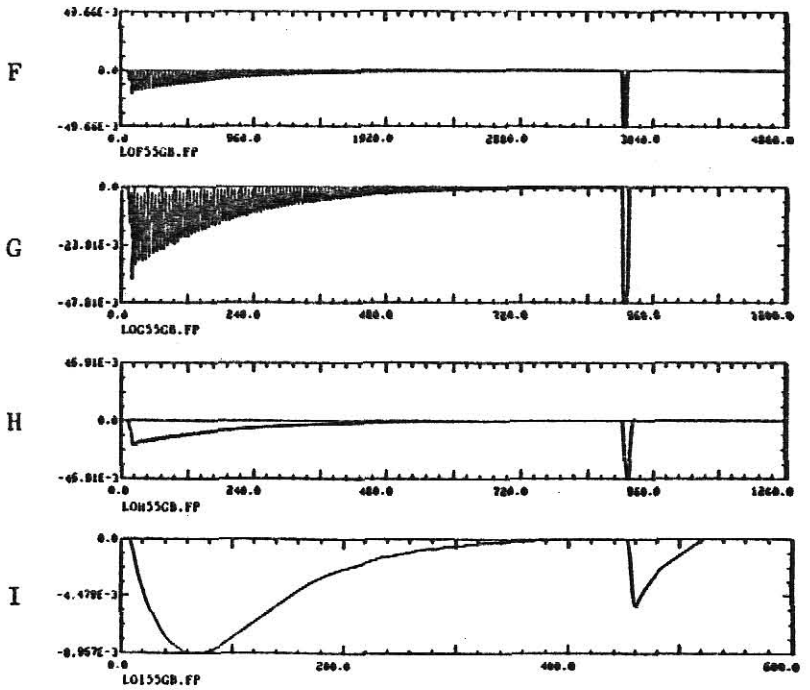


Figure 7. LOW BAND: Outputs after complex predictor with +-0.05 input with noise



8a. Double precision



8b. Fixed point

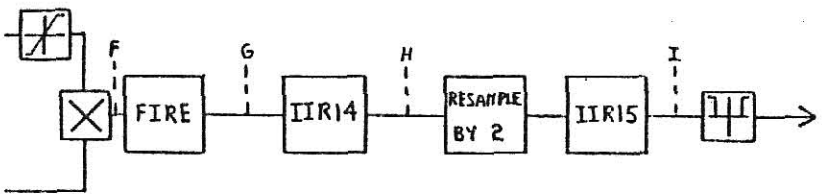
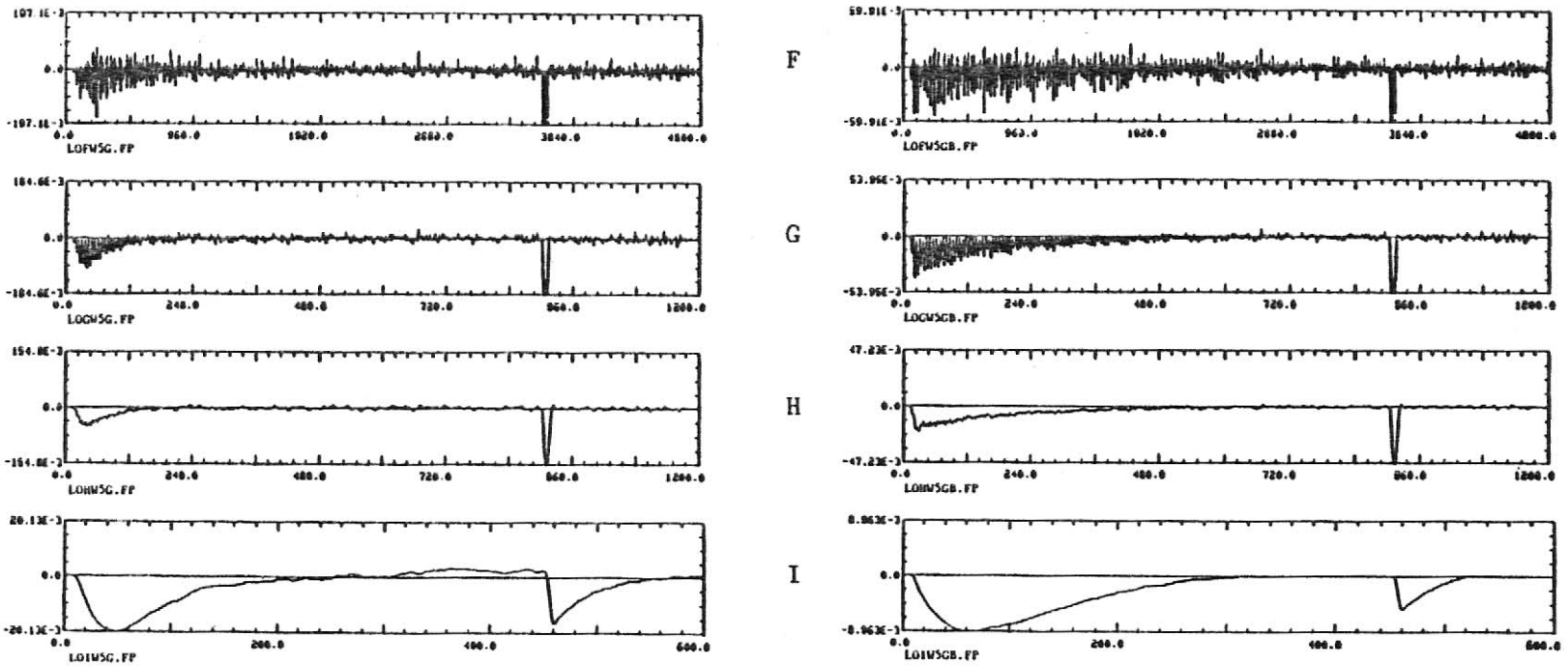


Figure 8. LOW BAND: Outputs of last 4 stages with +-0.05 input without noise



9a. Double precision

9b. Fixed point

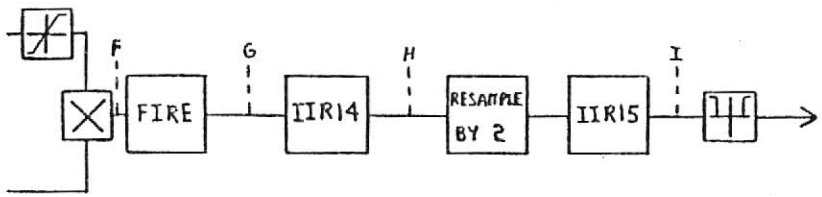


Figure 9. LOW BAND: Outputs of last 4 stages with +-0.05 input with noise

5.0 MICROCOMPUTER IMPLEMENTATION

The resulting modified system is briefly presented here. Two aspects of the microcomputer implementation will be covered -- the general system and the structure of the program.

5.1 SYSTEM DESCRIPTION

The processor chosen for implementing the algorithm was National Semiconductor's NSC800, a CMOS Z80 look-alike. The combination of this processor's high speed and low power with a low power hardware multiplier was necessary to do the large number of computations in the required time. The hardware multiplier does a 16 bit by 16 bit multiply yielding a 32 bit result. A 32 bit accumulator keeps a running sum of the results. Because of the complexity of the algorithm, the ROM part of the memory is rather large, and there is a fair amount of RAM for storage of variables for the filters.

5.2 GENERAL PROGRAM DESCRIPTION

Several methods of coding the filters were looked at, but to achieve the required sampling rate, and since memory is relatively cheap, straight in-line code was used. The coefficients for the filters are stored in the in-line code as immediate values to load to a register pair. The program sends the two operands to the hardware multiplier, and lets the 32 bit accumulator in the hardware multiplier section do the addition to perform the multiply and add sequence present

in all of the digital filters. One obtains the final result by loading in the 16 most significant bits from the 32 bit accumulator, thus truncating the result and producing the quantizing errors related to truncation as discussed earlier. The last n inputs for both IIR and FIR filters and the last n outputs for IIR filters are stored in push-down stacks in RAM. The FIR filters do the push-down of the previous inputs as the current output is calculated.

Since the sampling rate is not constant for all of the filters throughout the algorithm, counters are kept to determine when a certain filter should be done. To get all of the processing done before the next sample is taken, a scheme was devised to determine which filters should be done during a given block of time. Thus, the filters that have a higher sampling rate must be done every sample, but ones with a lower sampling rate can be done when time is available. The code that was sent to Sandia Laboratories consisted of all the filters for the algorithm and the timing scheme for when to do the filters.

6.0 CONCLUSIONS

This paper dealt with the simulation of a microcomputer-based intrusion detection system and the effects of using a finite word length in the calculation of the digital filters. The filters were first checked for overflows with a maximum input signal, and the modified filters were then checked for underflows with a minimum input signal. The modifications to

the filters should improve the system's performance.

Since the original design of the system was done at Sandia, the purpose of the variable gain blocks and the reason why +-2.8 V was chosen for some of the limiters when the present number representation covers only +-1.999939 V is not known. The variable gain was left at unity gain, and the limiters did not effect the signals.

Because of the problems that can occur from using a finite word length, and since the filters in a given algorithm may produce unexpected results when they interact, I developed the following procedure for designing and implementing a system:

1) Design the overall block diagram or algorithm for the specific problem

2) Design the specific blocks or filters according to specifications

   a) If possible, choose the form of filter that produces the smallest errors due to quantization

   b) Check for movement of the zeros and poles

   c) Check for internal overflows and deadbands

3) Design the combination of filters by checking the amplitude after each filter

4) Simulate the system on a larger computer and compare double precision outputs with fixed point outputs

   a) Using a maximum input signal, check for overflows and adjust filters for maximum signal

   b) Using a minimum input signal and the adjusted filters, check for zero outputs or loss of signal

As a result of using the above procedure, the following filters were changed:

High Band: Multiply IIR3 numerator by 7.4225190
Multiply FIRA by 2.0
Eliminate or modify IIR5 to retain information present in previous block

Mid Band: Multiply IIR8 numerator by 2.8180921
Multiply FIRC by 2.0
Eliminate or modify IIR10 to retain information present in previous block

Low Band: Multiply IIR11 numerator by 0.85
Multiplyoutputs of complex predictor by 6.6666666
Multiply IIR13 numerator by 3.3534541
Multiply FIRE by 2.0

The changes that were made in this design served to eliminate an internal overflow in the IIR11 filter, and to increase the SNR throughout the system by increasing the gain of some filters to make better use of the dynamic range of the system. Both the IIR5 and IIR10 filters should be eliminated or modified to maintain the signal present in the previous stage.

As was noted in an earlier discussion of the effects of product quantization, one may need to resort to simulating the algorithm on a larger machine to choose the best form of the filter. Generally, it seems that a simulation provides a much better refinement of the entire system than do theoretical calculations.

ACKNOWLEDGEMENTS

This work was sponsored and funded by the Systems Engineering Division, Organization 5238, Sandia National Laboratories, Kirtland Air Force Base, Albuquerque, New Mexico.

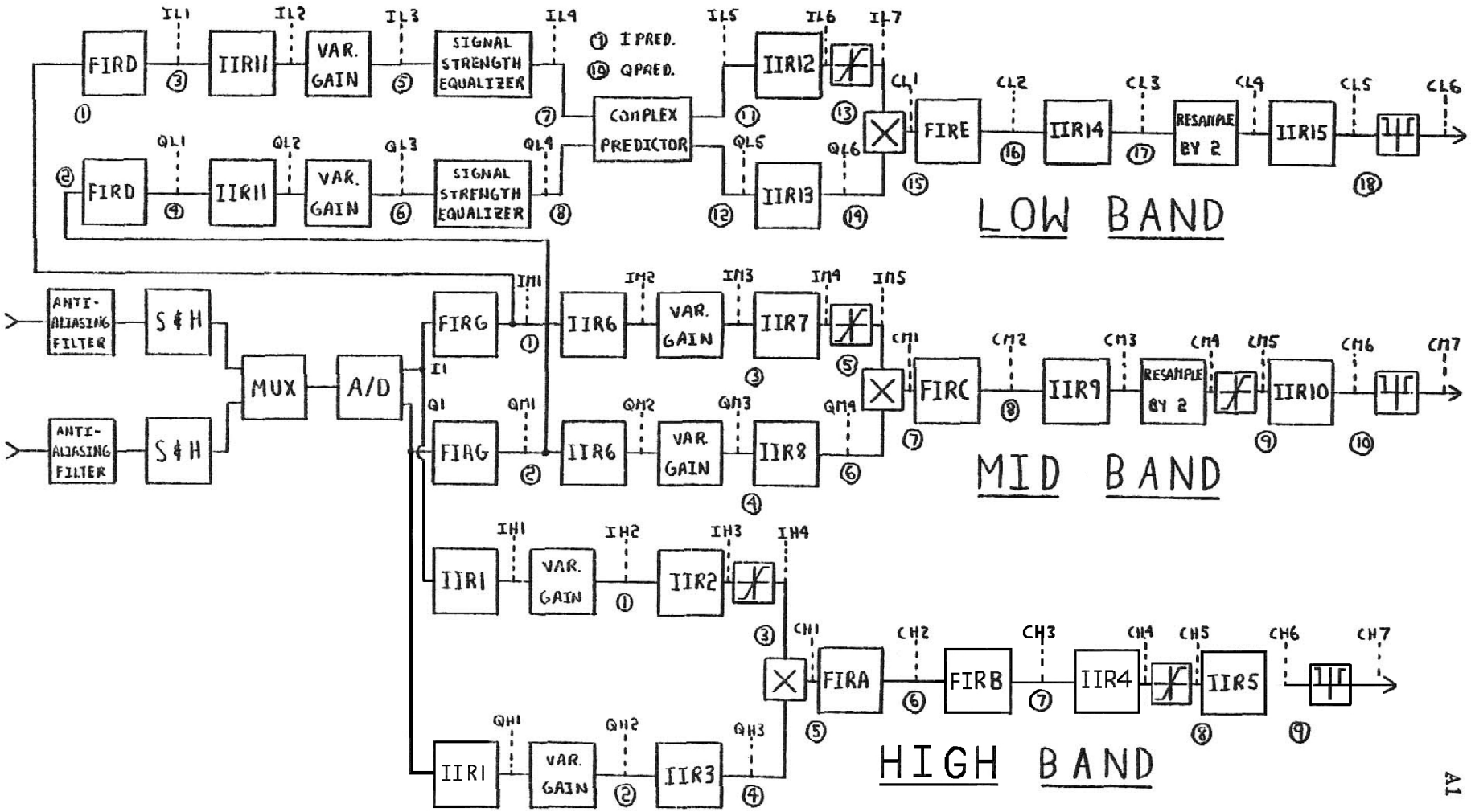
I would like to express my sincere appreciation for the help provided by Dr.D.H.Lenhert and Dr.M.S. P. Lucas during the course of my work. I also thank Sandia Laboratories for funding this research work, and providing needed information.

Special thanks go to Dan Schowengert, who wrote the initial versions of the programs, and to the users of the VAX who put up with all my tape drive problems and ever increasing use of disk space.

APPENDIX A

The following sheet is the filter block diagram of the intrusion detection system. Each block is labelled as to the function it performs; the filters are discussed in appendix B. The output points corresponding to the number used by the program and the naming of the output file are enclosed in circles. The output of every block in the diagram has been labelled according to its position from the original signal input. The first letter is a "I" for Inphase signal, "Q" for Quadphase signal, or "C" for Combined signal. The second letter is a "H" for High band, "M" for Mid band, or "L" for Low band. The final digit indicates the position of the output in that particular section.

FILTER BLOCK DIAGRAM



APPENDIX B

This appendix contains the specifications for the filters used in the algorithm, and plots of their amplitude and phase response.

IIR1:

6 pole band pass Butterworth filter
FS = 240.000 ; F1 = 9.000 ; F2 = 82.500

Stage #1 with two sets of complex conjugate poles

$$H(Z) = \frac{A(K)\ (\ 1 - 2*Z^{**-2} + Z^{**-4}\ )}{1+B(K)Z^{**-1} + C(K)Z^{**-2} + D(K)Z^{**-3} + E(K)Z^{**-4}}$$

Stage #1:

A(1) = 4.58144844E-01 B(1) = -9.74493027E-01
C(1) = -1.09509118E-01 D(1) = -1.60353869E-01
E(1) = 3.61381799E-01

Stage #2 with one set of complex conjugate poles

$$H(Z) = \frac{A((N+1)/2)\ (\ 1 - Z^{**-2}\ )}{1 + B((N+1)/2)Z^{**-1} + C((N+1)/2)Z^{**-2}}$$

Stage #2:

A(2) = 5.89288652E-01 B(2) = -5/23593187E-01
C(2) = -1.78577363E-01

IIR2:

1 pole low pass Butterworth filter
FS = 240.000 ; FC = 112.500

Stage #1 with one pole at -PI

$$H(Z) = \frac{A((N+1)/2)\ (\ 1 + Z^{**-1}\ )}{1 + B((N+1)/2)Z^{**-1}}$$

Stage #1:

A(1) = 9.10339415E-01 B(1) = 8.20678830E-01

IIR3:

1 pole high pass Butterworth filter
FS = 240.000 ; FC = 112.500

Stage #1 with one pole at -PI

$$H(Z) = \frac{A((N+1)/2)\ (\ 1 - Z^{**-1}\ )}{1 + B((N+1)/2)Z^{**-1}}$$

Stage #1:
A(1) = 8.96605924E-02 B(1) = 8.20678830E-01

IIR4:

3 pole low pass Butterworth filter
FS = 10.000 ; FC = 3.300

Stage #1 with two complex conjugate poles

$$H(Z) = \frac{A(K)\ (\ 1 + 2Z^{**-1} + Z^{**-2}\ )}{1 + B(K)Z^{**-1} + C(K)Z^{**-2}}$$

Stage #1:
A(1) = 5.15158534E-01 B(1) = 6.69961870E-01
C(1) = 3.90672445E-01

Stage #2 with one pole at -PI

$$H(Z) = \frac{A((N+1)/2)\ (\ 1 + Z^{**-1}\ )}{1 + B((N+1)/2)Z^{**-1}}$$

Stage #2:
A(2) = 6.28378272E-01 B(2) = 2.56756514E-01

IIR5:

1 pole low pass Butterworth filter
FS = 10.000 ; FC = 0.109

Stage #1 with one pole at -PI

$$H(Z) = \frac{A((N+1)/2)\ (\ 1 + Z^{**-1}\ )}{1 + B((N+1)/2)Z^{**-1}}$$

Stage #1:
A(1) = 3.31220999E-02 B(1) = -9.33755875E-01

IIR6:

8 pole band pass Butterworth filter
FS = 40.000 ; F1 = 2.250 ; F2 = 11.000

Stage #1 with two sets of complex conjugate poles

$$H(Z) = \frac{A(K)\ (\ 1 - 2Z^{**-2} + Z^{**-4}\ )}{1 + B(K)Z^{**-1} + C(K)Z^{**-2} + D(K)Z^{**-3} + E(K)Z^{**-4}}$$

Stage #1:

A(1) = 2.92624205E-01 B(1) = -1.49385989E+00
C(1) = 1.02734399E+00 D(1) = -7.79795229E-01
E(1) = 4.54196155E-01

Stage #2 with two sets of complex conjugate poles

$$H(Z) = \frac{A(K)\ (\ 1 - 2Z^{**-2} + Z^{**-4}\ )}{1 + B(K)Z^{**-1} + C(K)Z^{**-2} + D(K)Z^{**-3} + E(K)Z^{**-4}}$$

Stage #2:

A(2) = 2.11137474E-01 B(2) = -1.44218183E+00
C(2) = 7.41260588E-01 D(2) = -1.98330969E-01
E(2) = 4.92477380E-02

IIR7:

1 pole low pass Butterworth filter
FS = 40.000 ; FC = 15.000

Stage #1 with one pole at -PI

$$H(Z) = \frac{A((N+1)/2)\ (\ 1 + Z^{**-1}\ )}{1 + B((N+1)/2)Z^{**-1}}$$

Stage #1:

A(1) = 7.07106769E-01 B(1) = 4.14213538E-01

IIR8:

1 pole high pass Butterworth filter
FS = 40.000 ; FC = 15.000

Stage #1 with one pole at -PI

$$H(Z) = \frac{A((N+1)/2)\ (\ 1 - Z^{**-1}\ )}{1 + B((N+1)/2)Z^{**-1}}$$

Stage #1:
A(1) = 2.92893231E-01 B(1) = 4.14213538-01

IIR9:

3 pole low pass Butterworth filter
FS = 5.000 ; FC = 0.440

Stage #1 with two complex conjugate poles

$$H(Z) = \frac{A(K)\ (\ 1 + 2Z^{**-1} + Z^{**-2}\ )}{1 + B(K)Z^{**-1} + C(K)Z^{**-2}}$$

Stage #1:
A(1) = 5.90080023E-02 B(1) = -1.34801686E+00
C(1) = 5.84048927E-01

Stage #2 with one pole at -PI

$$H(Z) = \frac{A((N+1)/2)\ (\ 1 + Z^{**-1}\ )}{1 + B((N+1)/2)Z^{**-1}}$$

Stage #2:
A(2) = 2.21017361E-01 B(2) = -5.57965338E-01

IIR10:

1 pole low pass Butterworth filter
FS = 2.500 ; FC = 0.014

Stage #1 with one pole at -PI

$$H(Z) = \frac{A((N+1)/2)\ (\ 1 + Z^{**-1}\ )}{1 + B((N+1)/2)Z^{**-1}}$$

Stage #1:
A(1) = 1.72905121E-02 B(1) = -9.65418994E-01

IIR11:

8 pole band pass Butterworth filter
FS = 10.000 ; F1 = 0.100 ; F2 = 3.000

Stage #1 with two sets of complex conjugate poles

$$H(Z) = \frac{A(K)\ (\ 1 - 2Z^{**-2} + Z^{**-4}\ )}{1 + B(K)Z^{**-1} + C(K)Z^{**-2} + C(K)Z^{**-3} + E(K)Z^{**-4}}$$

Stage #1:

A(1) = 4.55506563E-01 B(1) = -1.50136971E+00
C(1) = 5.59125006E-01 D(1) = -5.09367466E-01
E(1) = 4.59150285E-01

Stage #2 with two sets of complex conjugate poles

$$H(Z) = \frac{A(K)\ (\ 1 - 2Z^{**-2} + Z^{**-4}\ )}{1 + B(K)Z^{**-1} + C(K)Z^{**-2} + C(K)Z^{**-3} + E(K)Z^{**-4}}$$

Stage #2:

A(2) = 3.29494983E-01 B(2) = -1.59343183E+00
C(2) = 4.04448390E-01 D(2) = 1.38945937E-01
E(2) = 5.54901883E-02

IIR12:

1 pole low pass Butterworth filter
FS = 10.000 ; FC = 4.000

Stage #1 with one pole at -PI

$$H(Z) = \frac{A((N+1)/2)\ (\ 1 + Z^{**-1}\ )}{1 + B((N+1)/2)Z^{**-1}}$$

Stage #1:

A(1) = 7.54762769E-01 B(1) = 5.09525537E-01

IIR13:

1 pole high pass Butterworth filter
FS = 10.000 ; FC = 4.000

Stage #1 with one pole at -PI

$$H(Z) = \frac{A((N+1)/2)\ (\ 1 - Z^{**-1}\ )}{1 + B((N+1)/2)Z^{**-1}}$$

Stage #1:
A(1) = 2.45237201E-01 B(1) = 5.09525537E-01

IIR14:

2 pole low pass Butterworth filter
FS = 2.500 ; FC = 0.150

Stage #1 with two complex conjugate poles

$$H(Z) = \frac{A(K)\ (\ 1 + 2Z^{**-1} + Z^{**-2}\ )}{1 + B(K)Z^{**-1} + C(K)Z^{**-2}}$$

Stage #1:
A(1) = 2.78597660E-02 B(1) = -1.47548032E+00
C(1) = 5.86919427E-01

IIR15:

1 pole low pass Butterworth filter
FS = 1.250 ; FC = 0.005

Stage #1 with one pole at -PI

$$H(Z) = \frac{A((N+1)/2)\ (\ 1 + Z^{**-1}\ )}{1 + B((N+1)/2)Z^{**-1}}$$

Stage #1:
A(1) = 1.24110617E-02 B(1) = -9.75177884E-01

FIRA:

10 pole low pass decimation filter

H(Z) = A(1)Z + A(2)Z**-1 + A(3)Z**-2 + ... + A(10)Z**-9

A(1) =	3.42925005E-02	A(2) =	6.23697005E-02
A(3) =	1.00152999E-01	A(4) =	1.33774996E-01
A(5) =	1.53612003E-01	A(6) =	1.53612003E-01
A(7) =	1.33774996E-01	A(8) =	1.00152999E-01
A(9) =	6.23697005E-02	A(10)=	3.42925005E-02

FIRB:

23 pole low pass decimation filter

H(Z) = A(1)Z + A(2)Z**-1 + A(3)Z**-2 + ... + A(23)Z**-22

A(1) =	1.86330993E-02	A(2) =	3.56806000E-03
A(3) =	-7.86815956E-03	A(4) =	-2.45447997E-02
A(5) =	-3.85877006E-02	A(6) =	-3.94657999E-02
A(7) =	-1.85441002E-02	A(8) =	2.62908991E-02
A(9) =	8.81720036E-02	A(10)=	1.52307004E-01
A(11)=	2.00696006E-01	A(12)=	2.18698993E-01
A(13)=	2.00696006E-01	A(14)=	1.52307004E-01
A(15)=	8.81720036E-02	A(16)=	2.62908991E-02
A(17)=	-1.85441002E-02	A(18)=	-3.94657999E-02
A(19)=	-3.85877006E-02	A(20)=	-2.45447997E-02
A(21)=	-7.86815956E-03	A(22)=	3.56806000E-03
A(23)=	1.86330993E-02		

FIRC:

16 pole low pass decimation filter

H(Z) = A(1)Z + A(2)Z**-1 + A(3)Z**-2 + ... + A(16)Z**-15

A(1) =	2.64819991E-02	A(2) =	2.92818006E-02
A(3) =	4.22337018E-02	A(4) =	5.59437014E-02
A(5) =	6.91791028E-02	A(6) =	8.05597976E-02
A(7) =	8.89628008E-02	A(8) =	9.34130028E-02
A(9) =	9.34130028E-02	A(10)=	8.89628008E-02
A(11)=	8.05597976E-02	A(12)=	6.91791028E-02
A(13)=	5.59437014E-02	A(14)=	4.22337018E-02
A(15)=	2.92818006E-02	A(16)=	2.64819991E-02

FIRD:

16 pole low pass decimation filter

H(Z) = A(1)Z + A(2)Z**-1 + A(3)Z**-2 + ... + A(16)Z**-15

A(1) =	-2.98360996E-02	A(2) =	-3.04717999E-02
A(3) =	-2.41525006E-02	A(4) =	5.10158995E-03
A(5) =	5.72319999E-02	A(6) =	1.22233003E-01
A(7) =	1.82209998E-01	A(8) =	2.18251005E-01
A(9) =	2.18251005E-01	A(10)=	1.82209998E-01
A(11)=	1.22233003E-01	A(12)=	5.72319999E-02
A(13)=	5.10158995E-03	A(14)=	-2.41525006E-02
A(15)=	-3.04717999E-02	A(16)=	-2.98360996E-02

FIRE:

9 pole low pass decimation filter

H(Z) = A(1)Z + A(2)Z**-1 + A(3)Z**-2 + ... + A(9)Z**-8

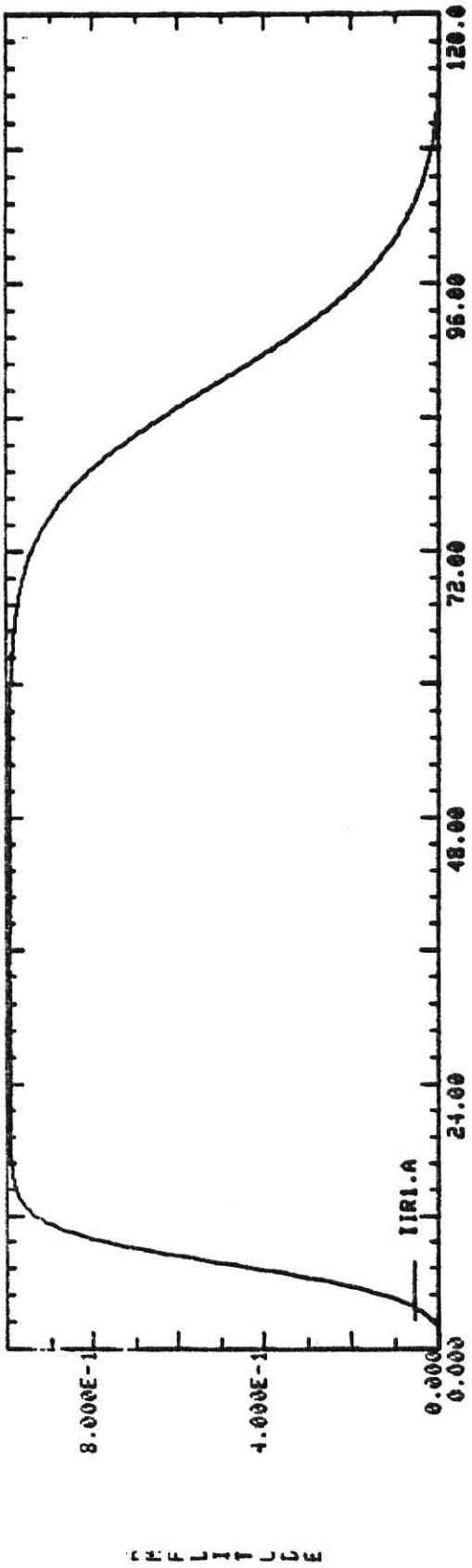
A(1) =	3.86609994E-02	A(2) =	7.59138986E-02
A(3) =	1.22501001E-01	A(4) =	1.59609005E-01
A(5) =	1.73759997E-01	A(6) =	1.59609005E-01
A(7) =	1.22501001E-01	A(8) =	7.59138986E-02
A(9) =	3.86609994E-02		

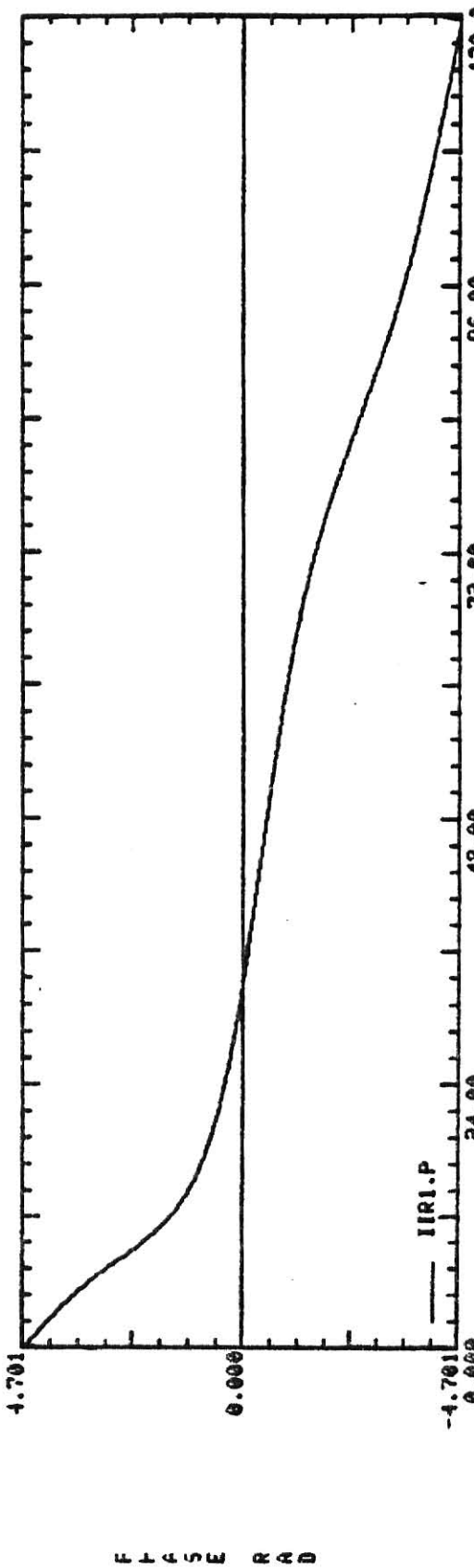
FIRG:

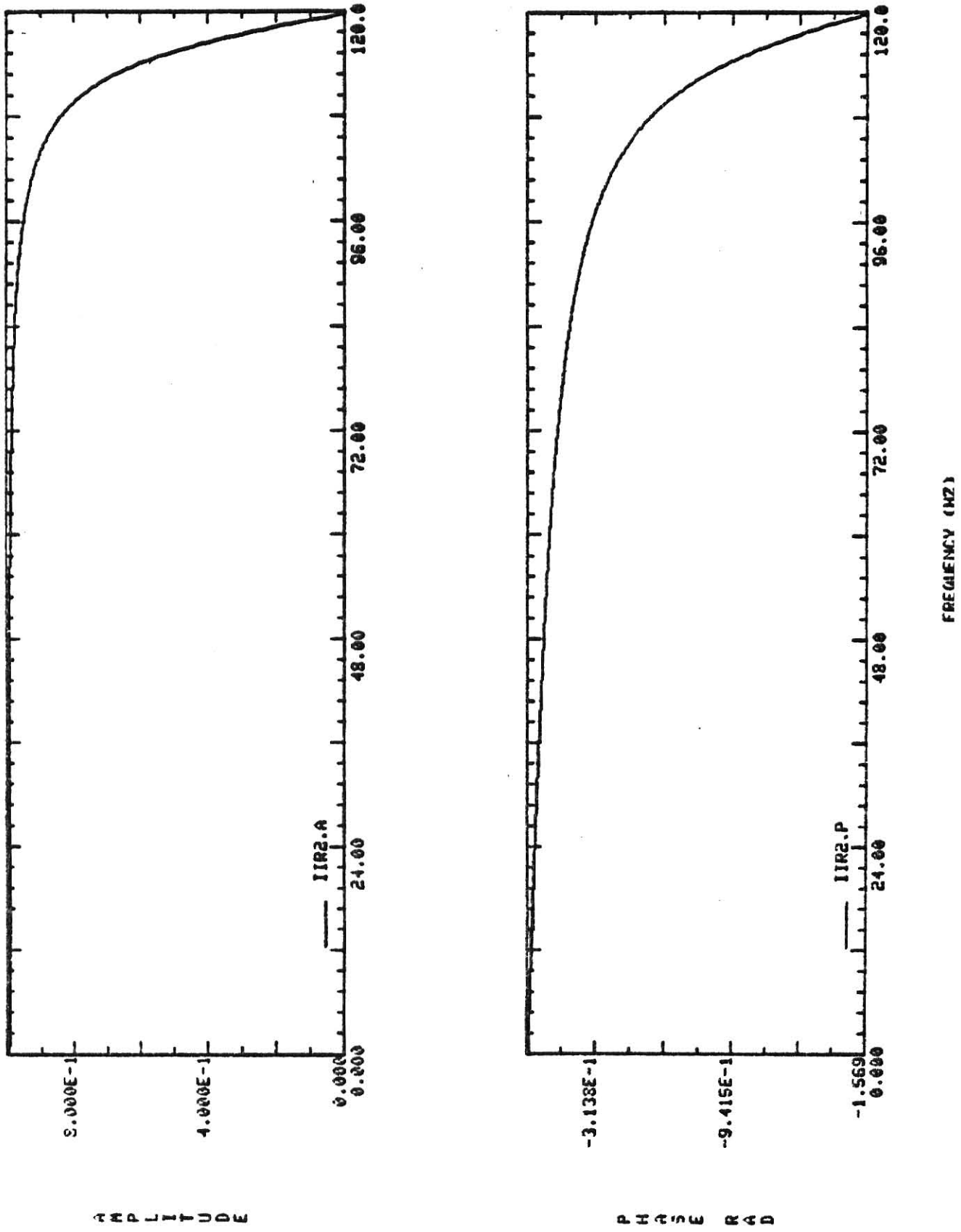
22 pole low pass decimation filter

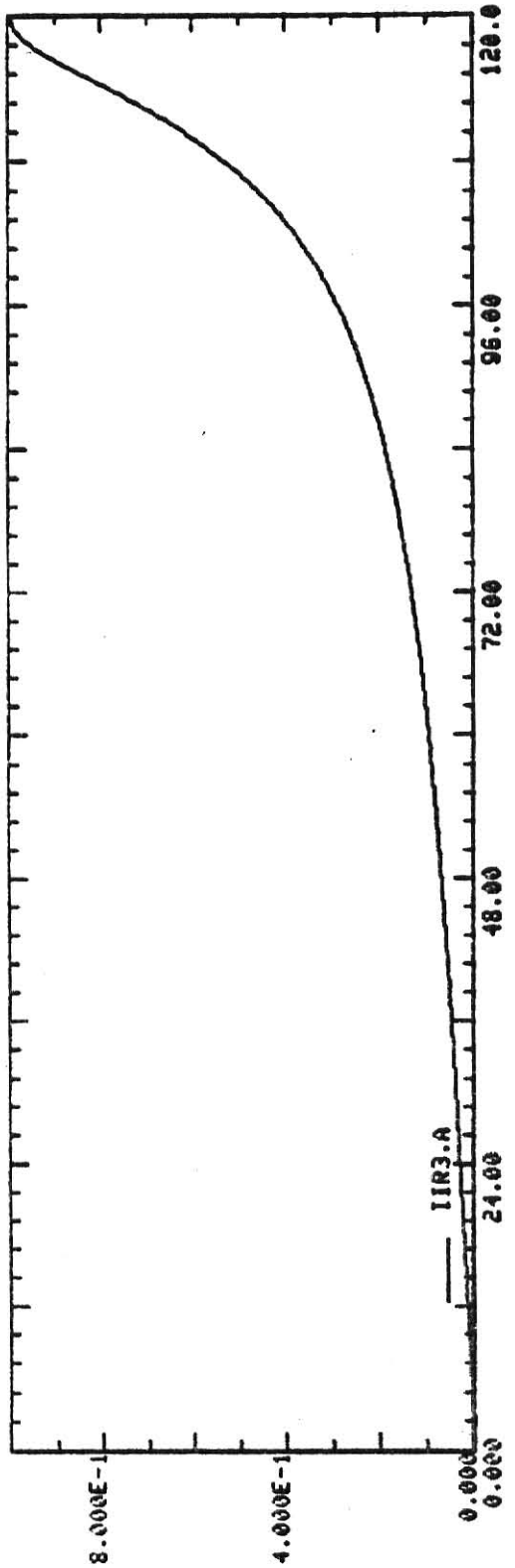
H(Z) = A(1)Z + A(2)Z**-1 + A(3)Z**-2 + ... + A(22)Z**-21

A(1) =	-2.28914991E-02	A(2) =	-1.52225001E-02
A(3) =	-1.38199003E-02	A(4) =	-6.11437019E-03
A(5) =	8.97826999E-03	A(6) =	3.11846007E-02
A(7) =	5.86810000E-02	A(8) =	8.81808028E-02
A(9) =	1.15640000E-01	A(10)=	1.36868000E-01
A(11)=	1.48442000E-01	A(12)=	1.48442000E-01
A(13)=	1.36868000E-01	A(14)=	1.15640000E-01
A(15)=	8.81808028E-02	A(16)=	5.86810000E-02
A(17)=	3.11846007E-02	A(18)=	8.97826999E-03
A(19)=	-6.11437019E-03	A(20)=	-1.38199003E-02
A(21)=	-1.52225001E-02	A(22)=	-2.28914991E-02

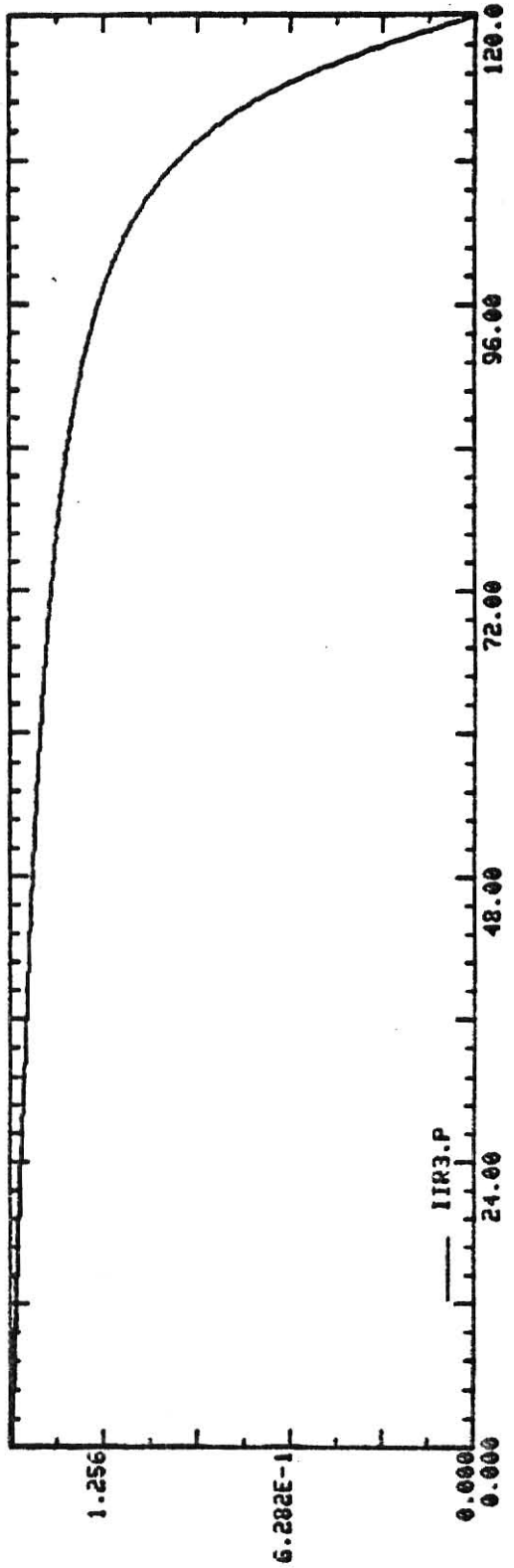

8.000E-1
4.000E-1
0.000
0.000
24.00
48.00
72.00
96.00
120.0
IIR1.A


4.701
0.000
-4.701
0.000
24.00
48.00
72.00
96.00
120.0
IIR1.P
FREQUENCY (HZ)


AMPLITUDE
8.000E-1
4.000E-1
0.000
0.000
24.00
48.00
72.00
96.00
120.0
IIR2.A
PHASE RAD
-3.138E-1
-9.415E-1
-1.569
0.000
24.00
48.00
72.00
96.00
120.0
IIR2.P
FREQUENCY (HZ)

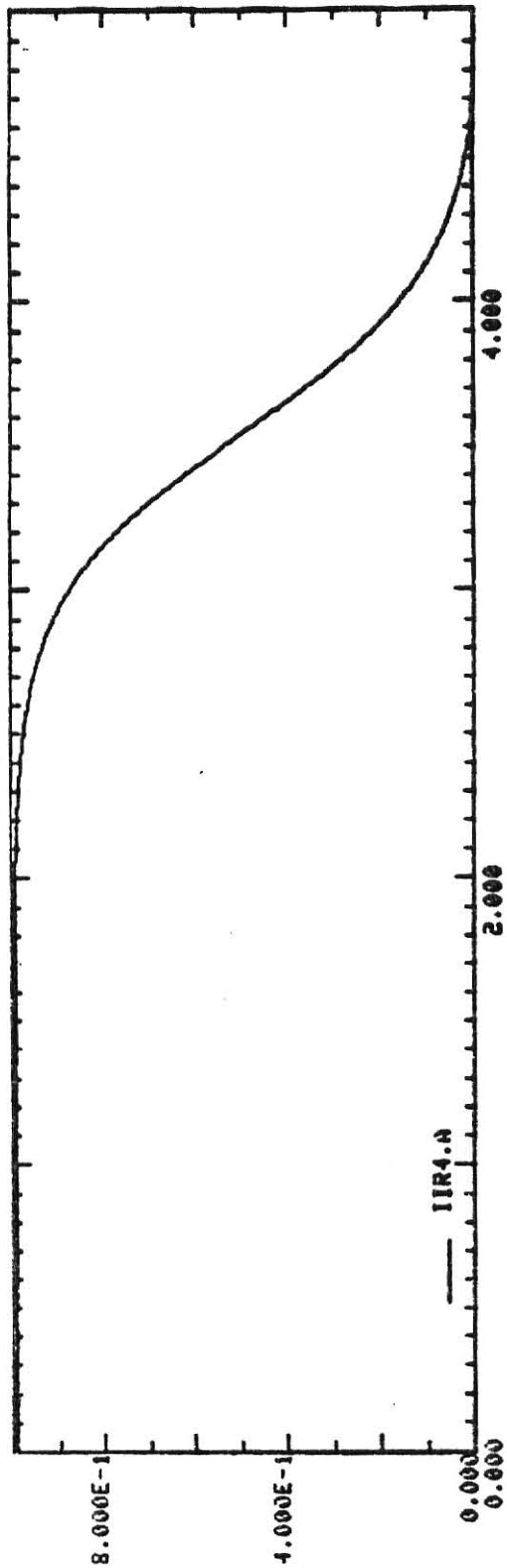


AMPLITUDE

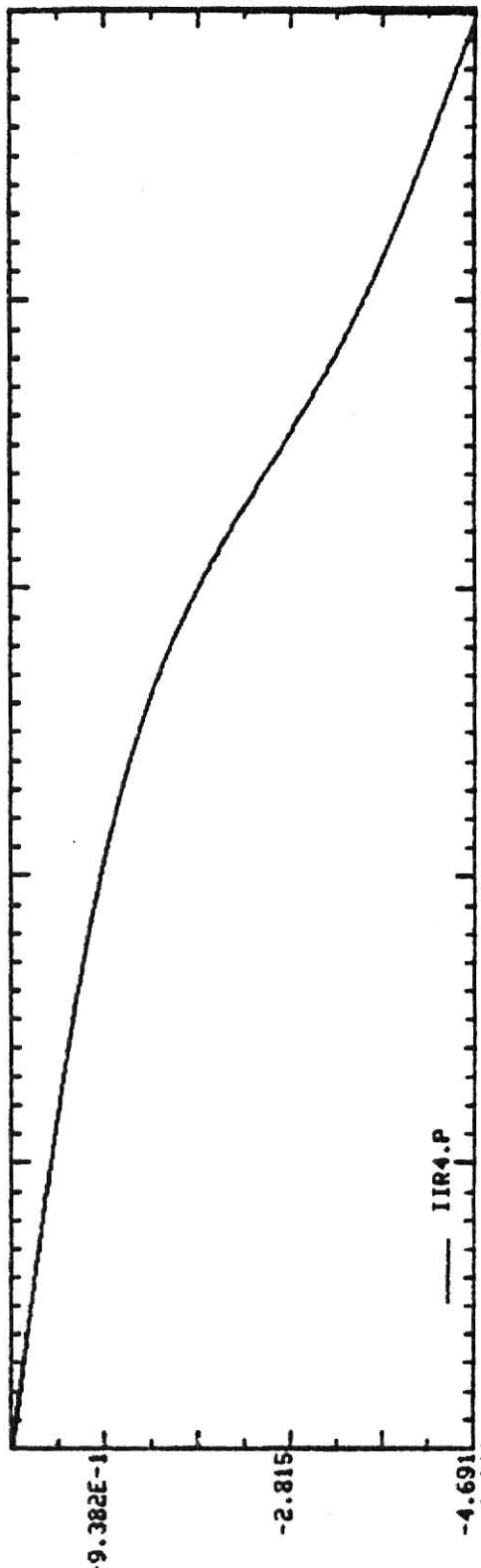


PHASE RAD

FREQUENCY (HZ)

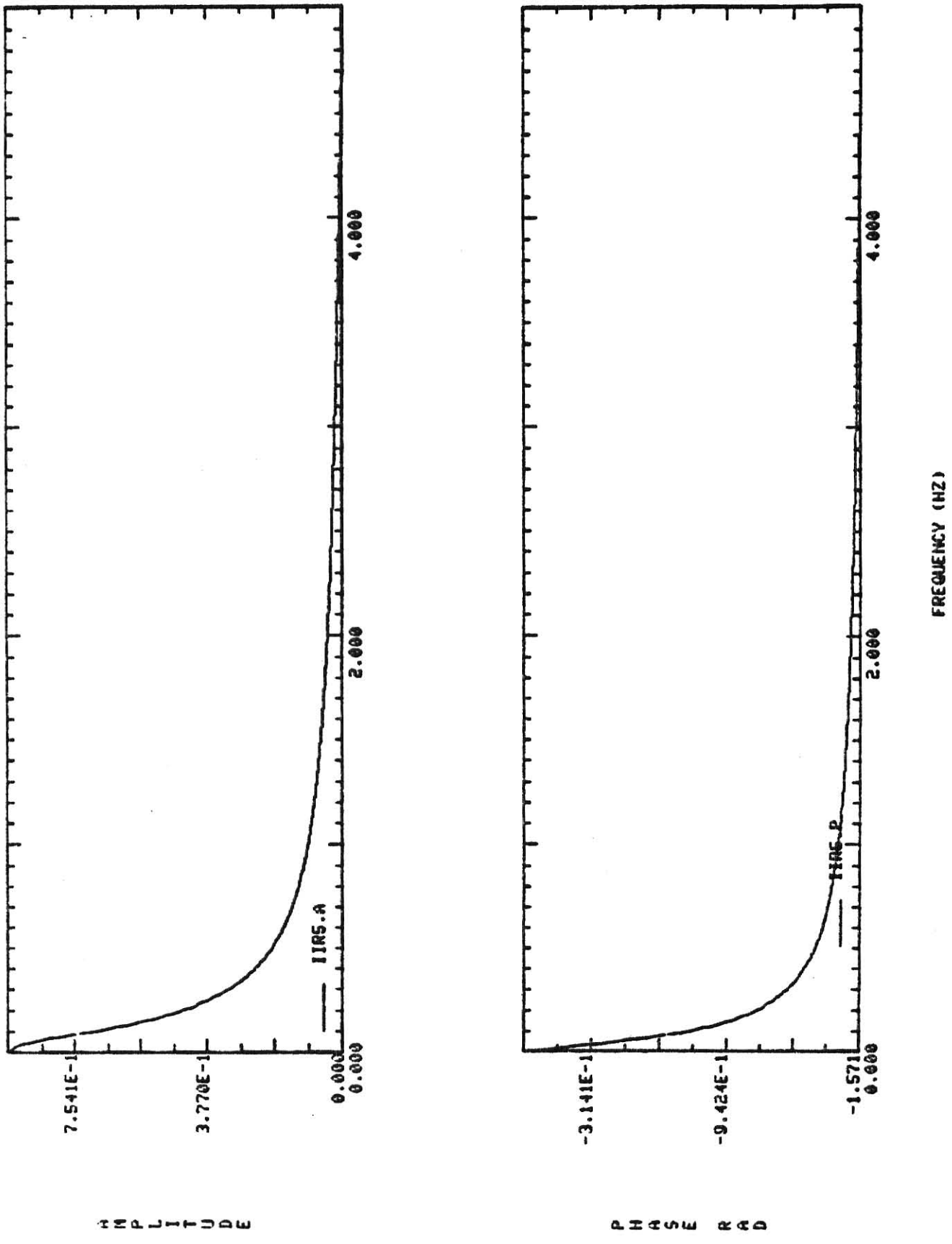


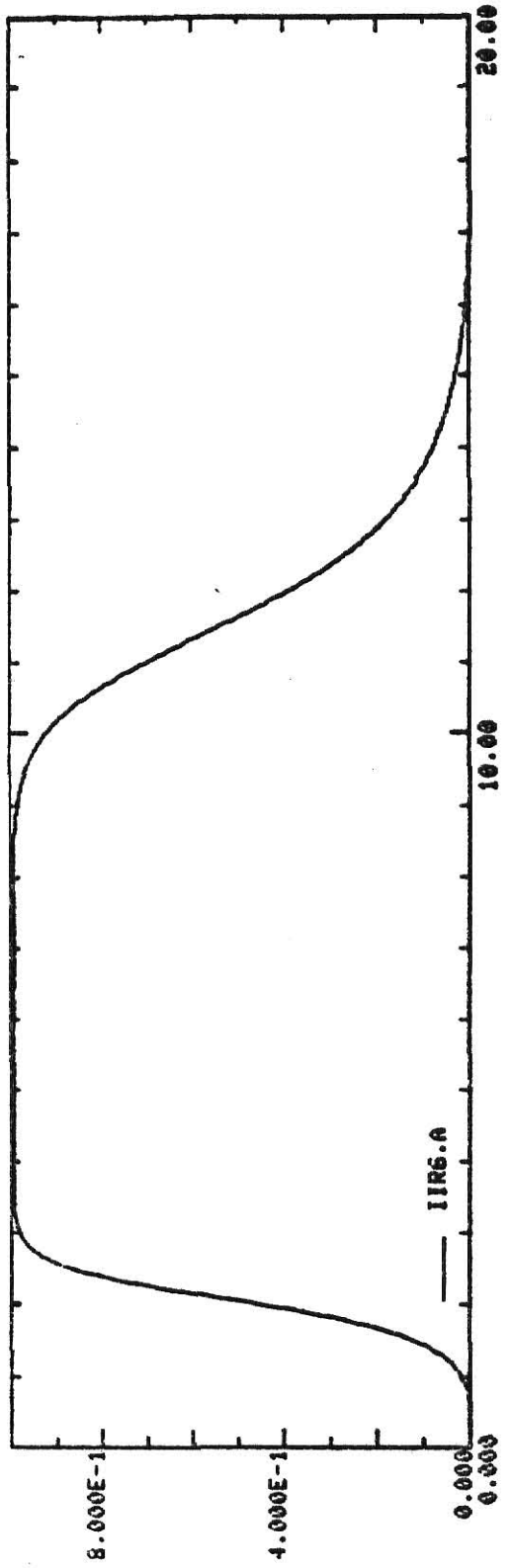
AMPLITUDE

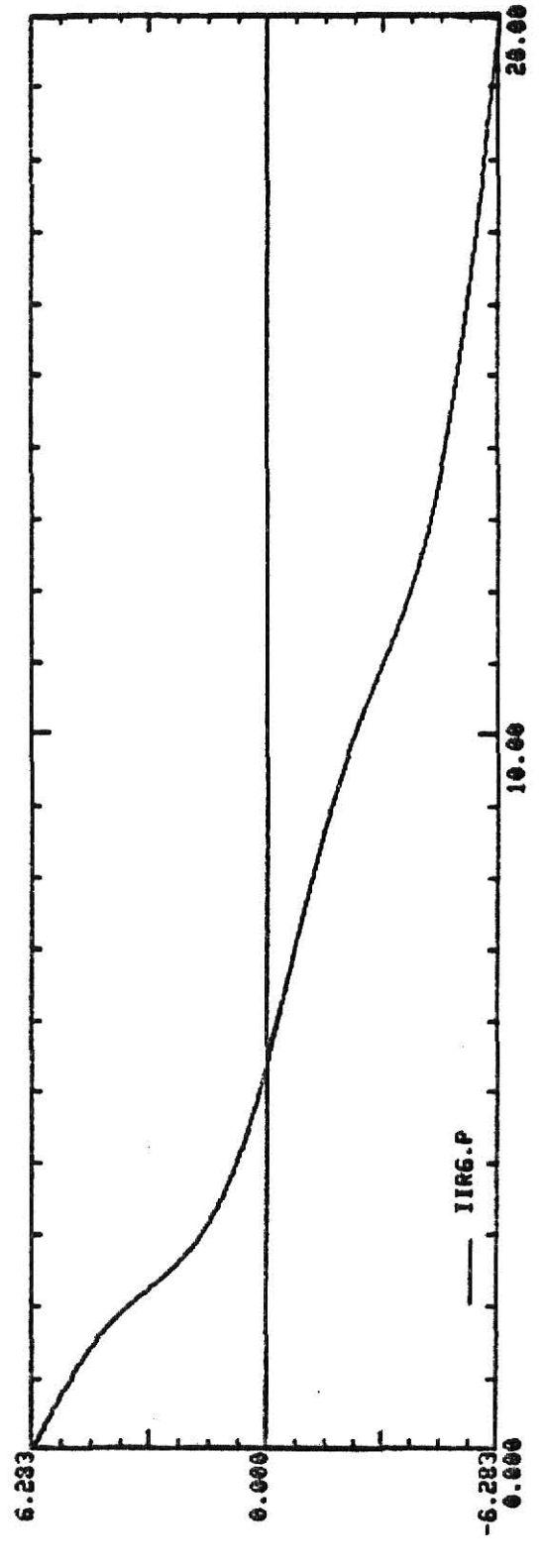


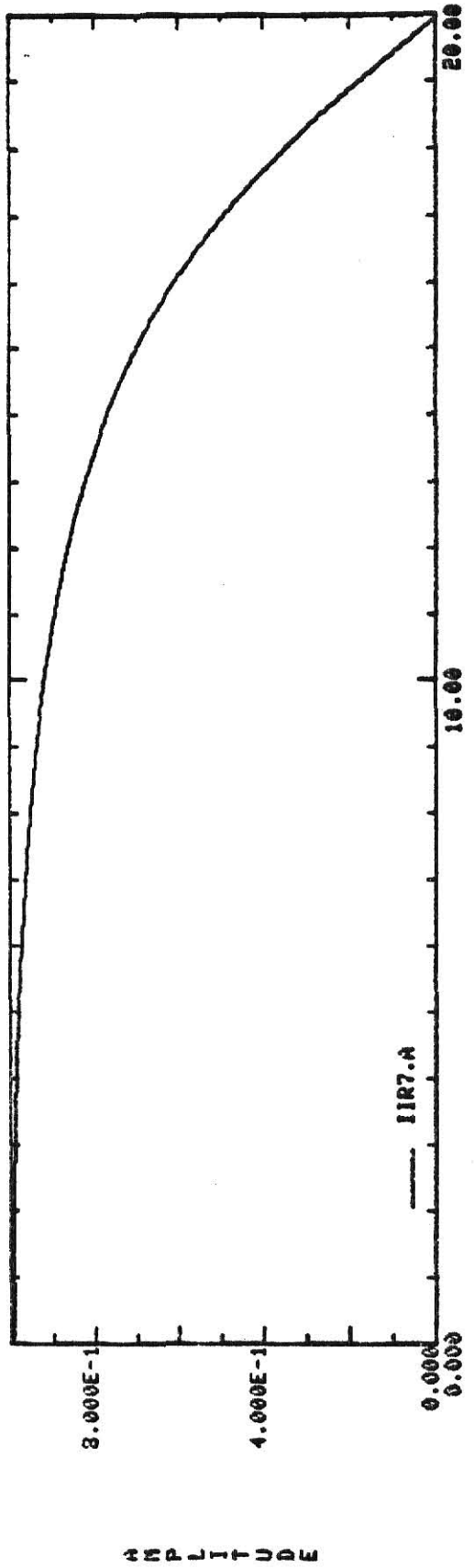
PHASE RAD

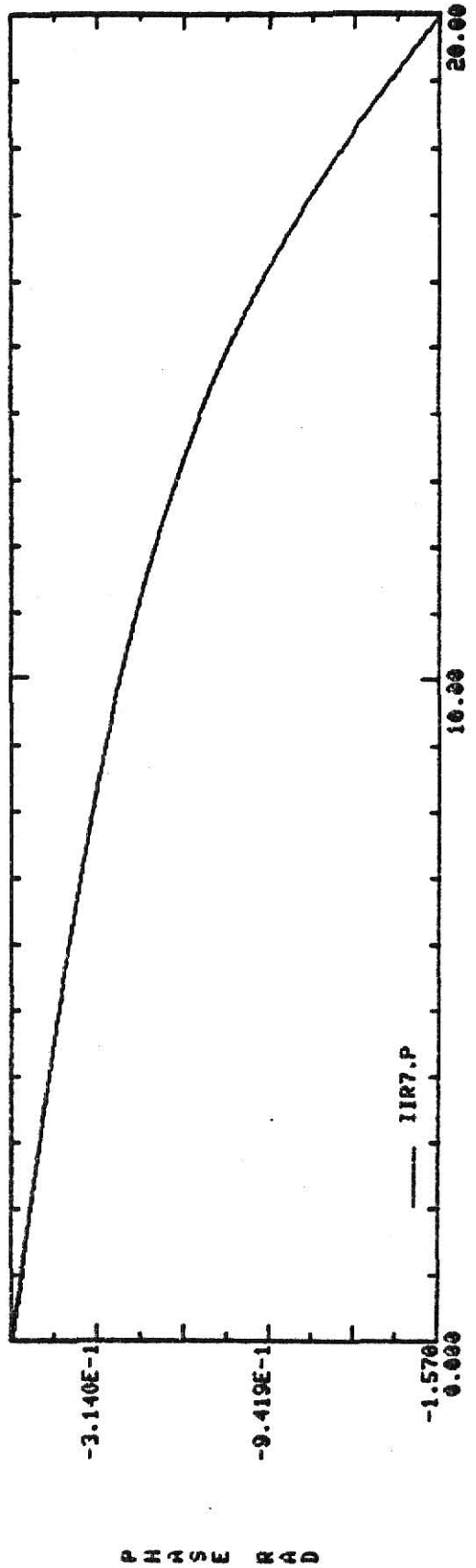
FREQUENCY (HZ)

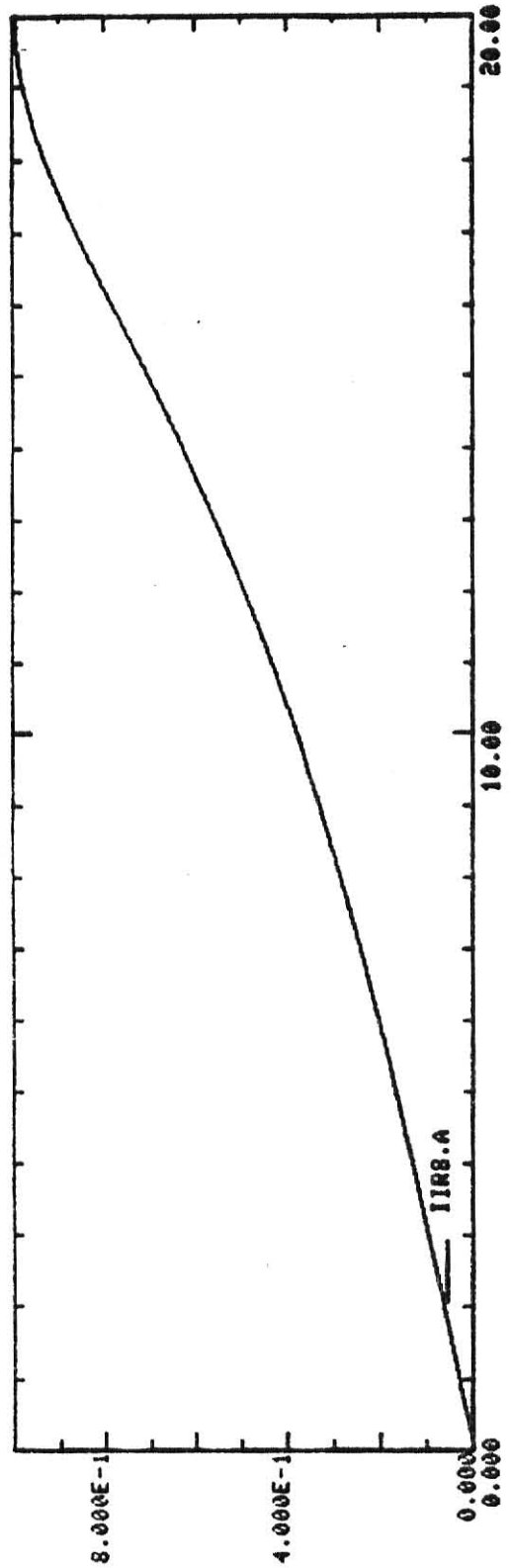

AMPLITUDE
7.541E-1
3.770E-1
0.000
0.000
2.000
4.000
IIR5.A
PHASE RAD
-3.141E-1
-9.424E-1
-1.571
0.000
2.000
4.000
IIR5.P
FREQUENCY (HZ)

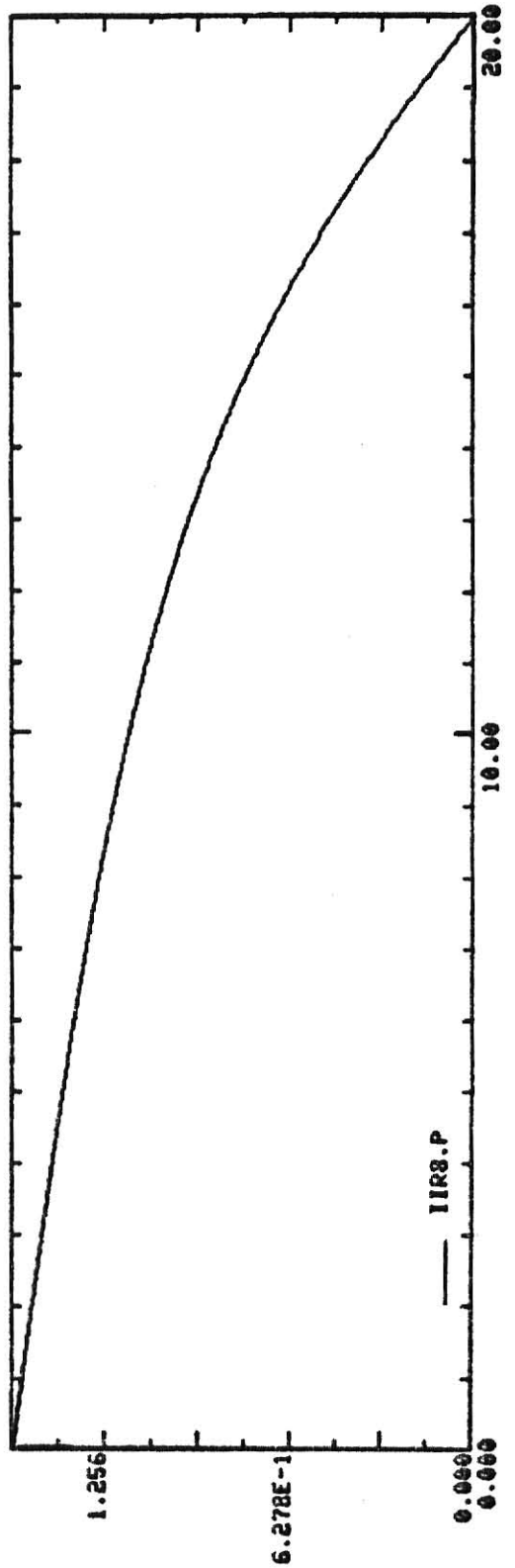

AMPLITUDE
8.000E-1
4.000E-1
0.000
0.000
10.00
20.00
IIR6.A

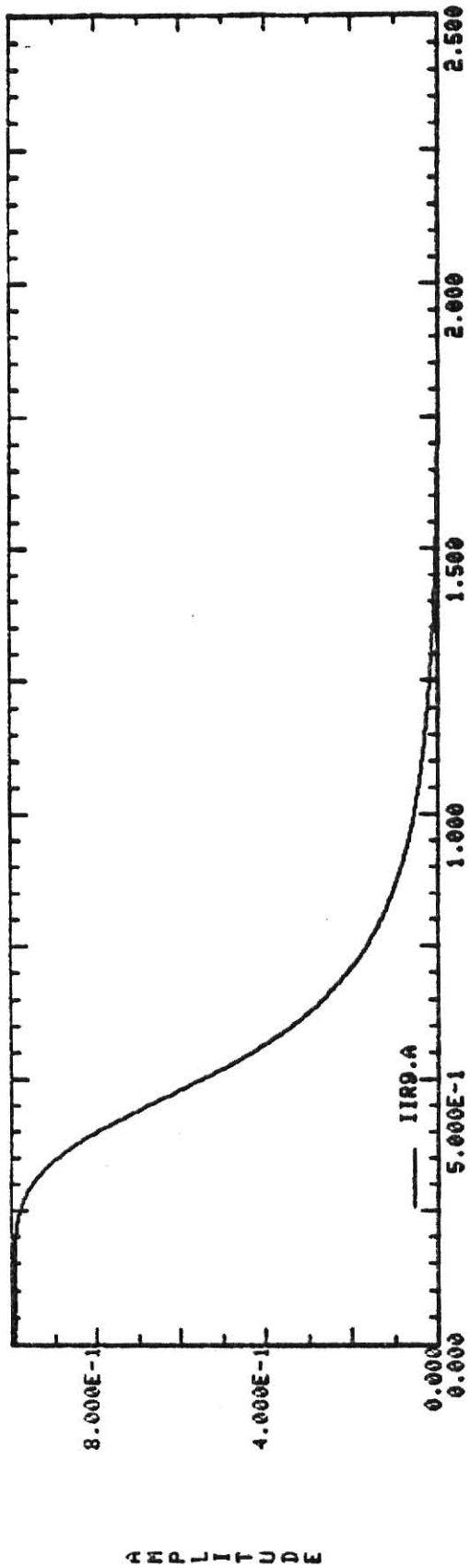

PHASE RAD
6.283
0.000
-6.283
0.000
10.00
20.00
IIR6.P
FREQUENCY (HZ)

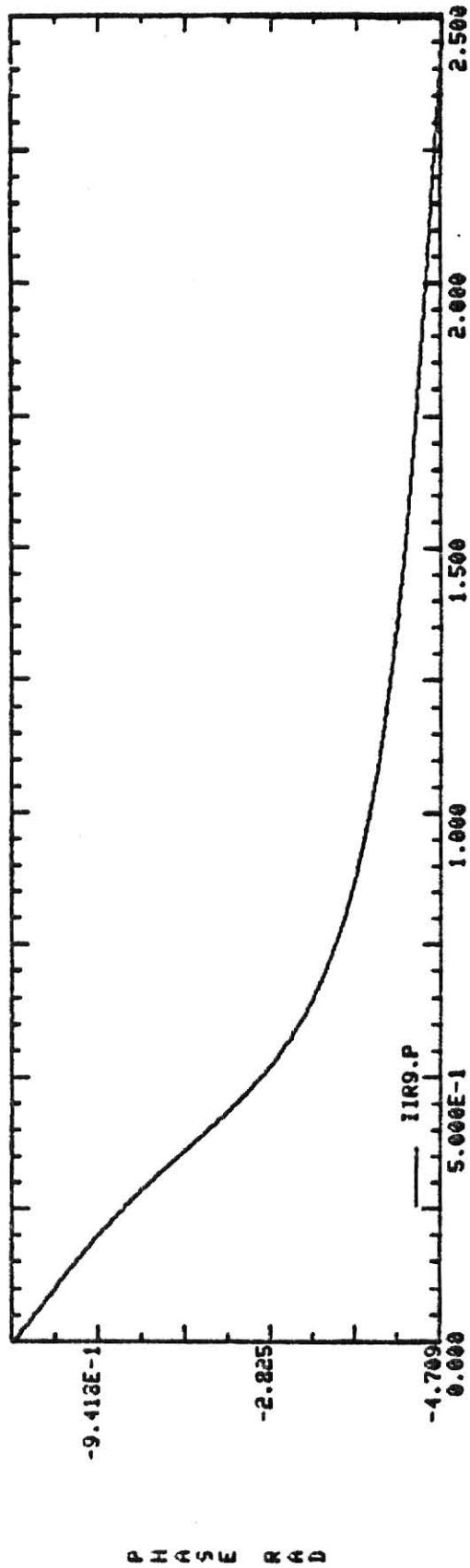

8.000E-1
4.000E-1
0.000
0.000
10.00
20.00
IIR7.A
AMPLITUDE

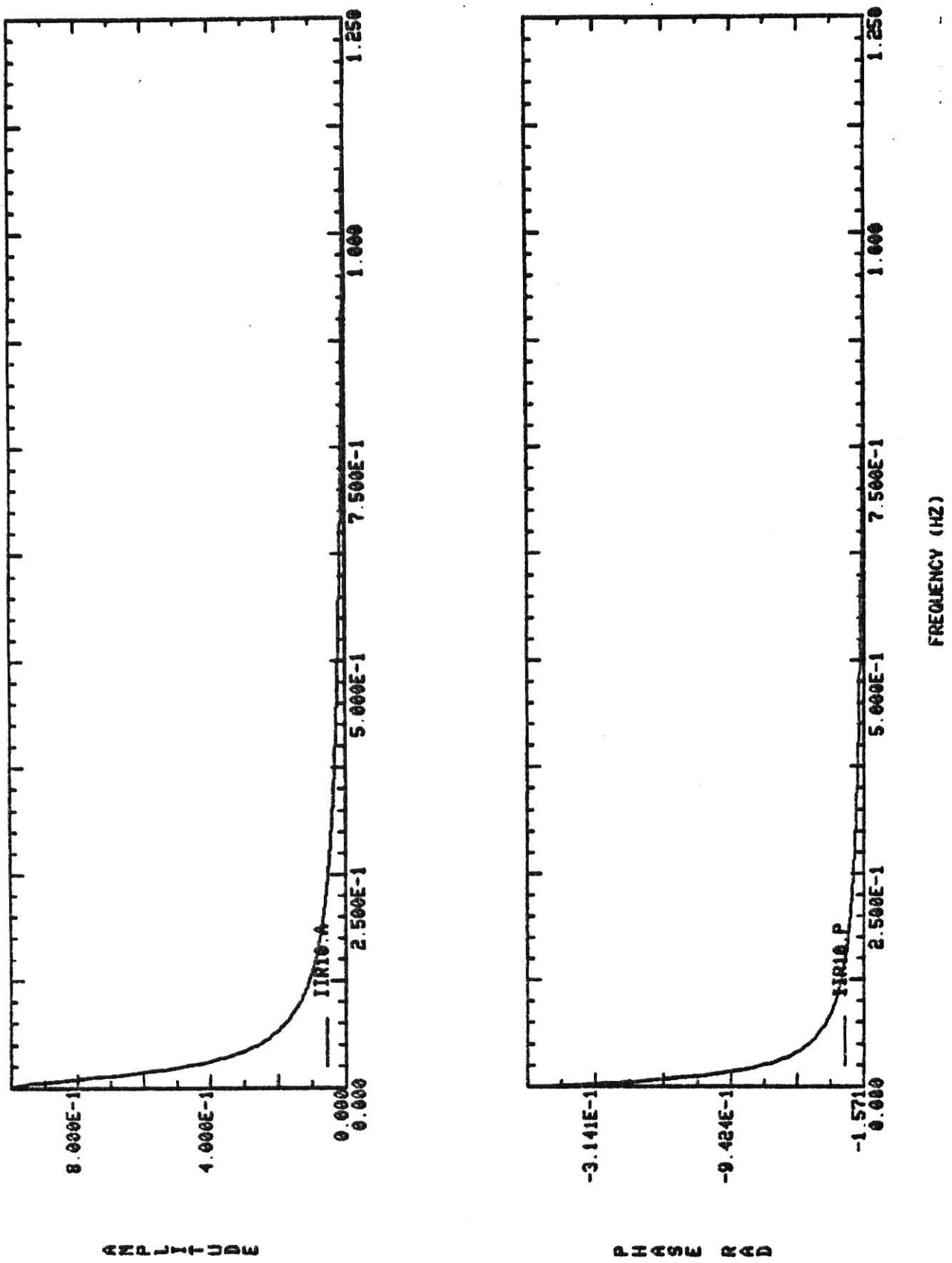

-3.140E-1
-9.419E-1
-1.570
0.000
10.00
20.00
IIR7.P
PHASE RAD
FREQUENCY (HZ)

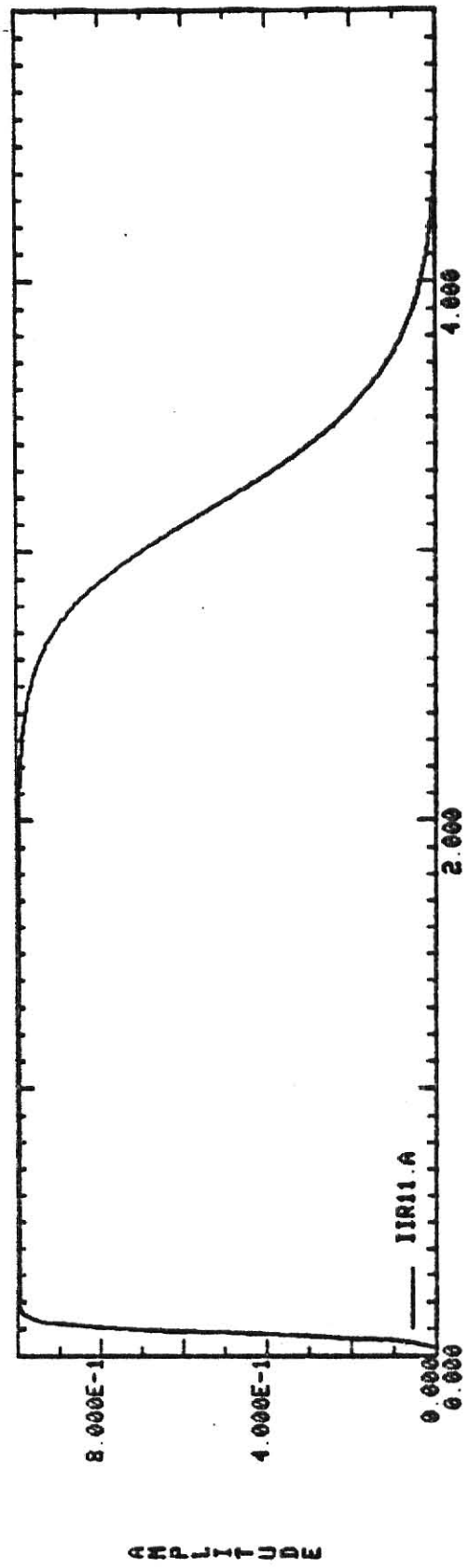

AMPLITUDE
8.000E-1
4.000E-1
0.000
0.000
10.00
20.00
IIR8.A

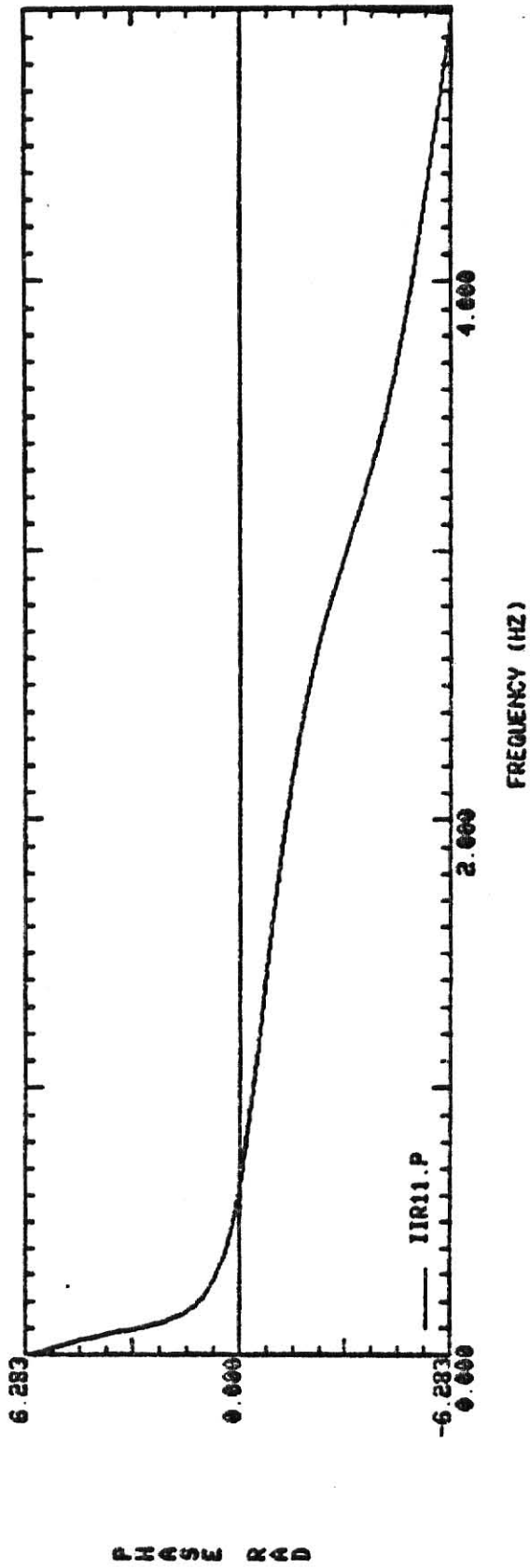

PHASE RAD
1.256
6.278E-1
0.000
0.000
10.00
20.00
IIR8.P
FREQUENCY (HZ)

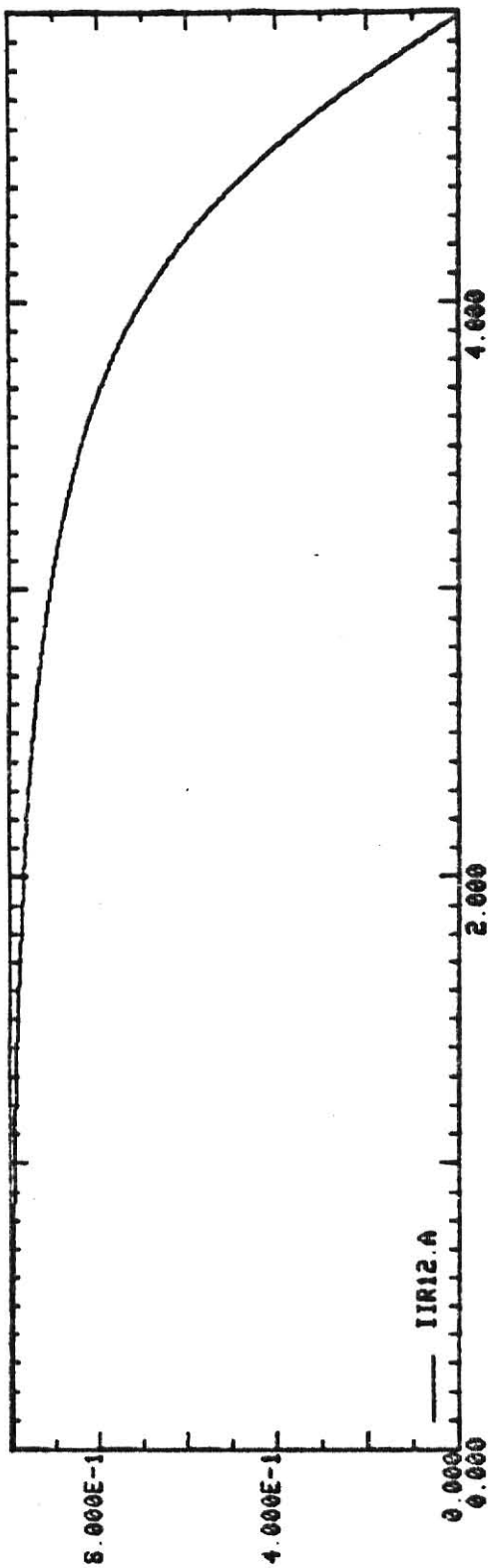

AMPLITUDE
8.000E-1
4.000E-1
0.000
0.000
5.000E-1
1.000
1.500
2.000
2.500
IIR9.A


PHASE RAD
-9.413E-1
-2.825
-4.709
0.000
5.000E-1
1.000
1.500
2.000
2.500
IIR9.P
FREQUENCY (HZ)

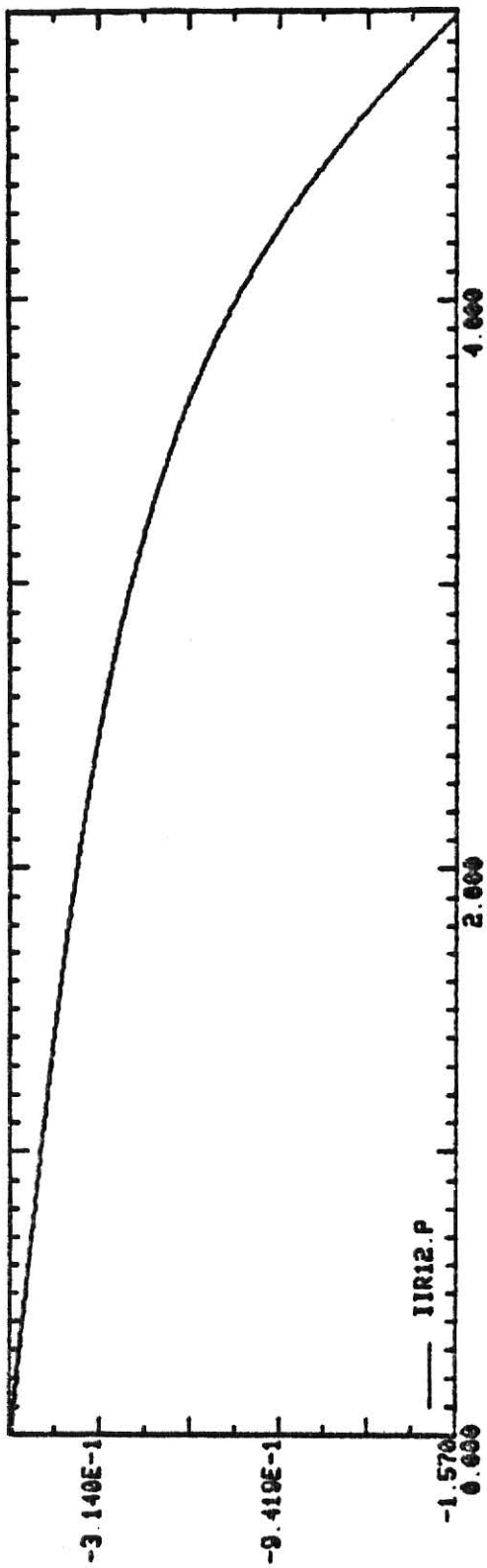

AMPLITUDE
8.000E-1
4.000E-1
0.000
0.000
2.500E-1
5.000E-1
7.500E-1
1.000
1.250
IIR10.A
PHASE RAD
-3.141E-1
-9.424E-1
-1.571
0.000
2.500E-1
5.000E-1
7.500E-1
1.000
1.250
IIR10.P
FREQUENCY (HZ)


8.000E-1
4.000E-1
0.000
0.000
2.000
4.000
IIR11.A
AMPLITUDE


6.283
0.000
-6.283
0.000
2.000
4.000
IIR11.P
PHASE RAD
FREQUENCY (HZ)

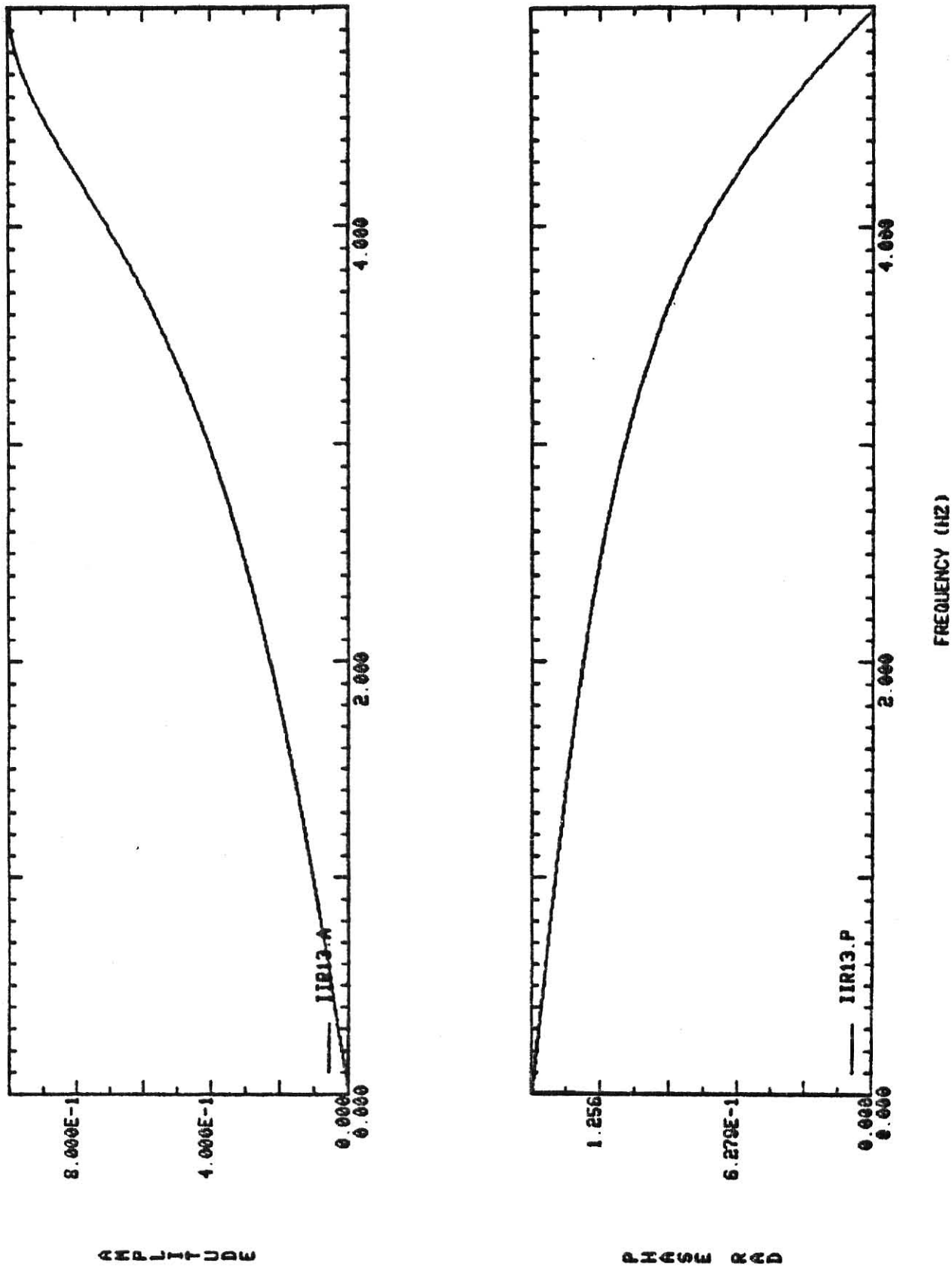


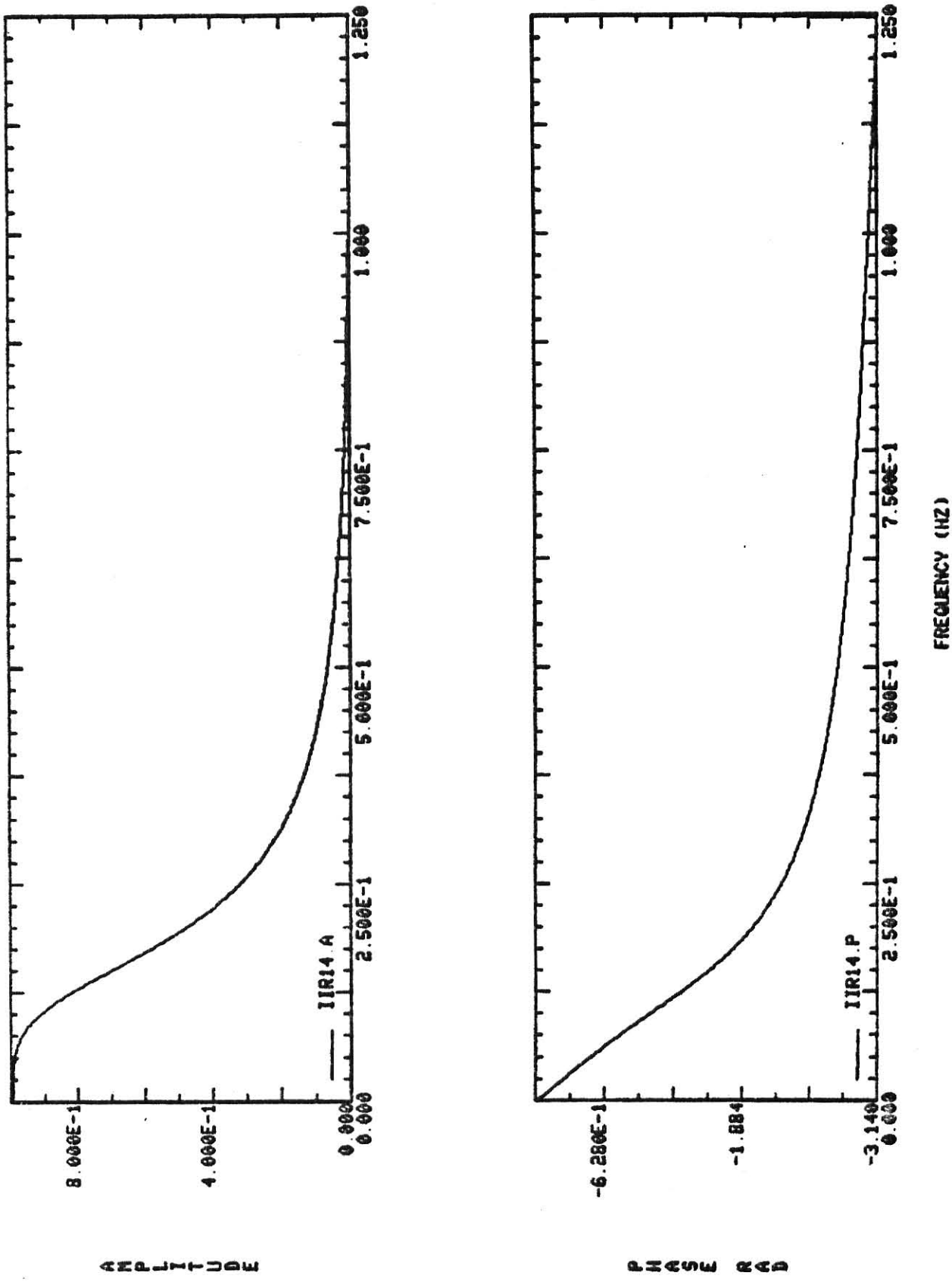
AMPLITUDE

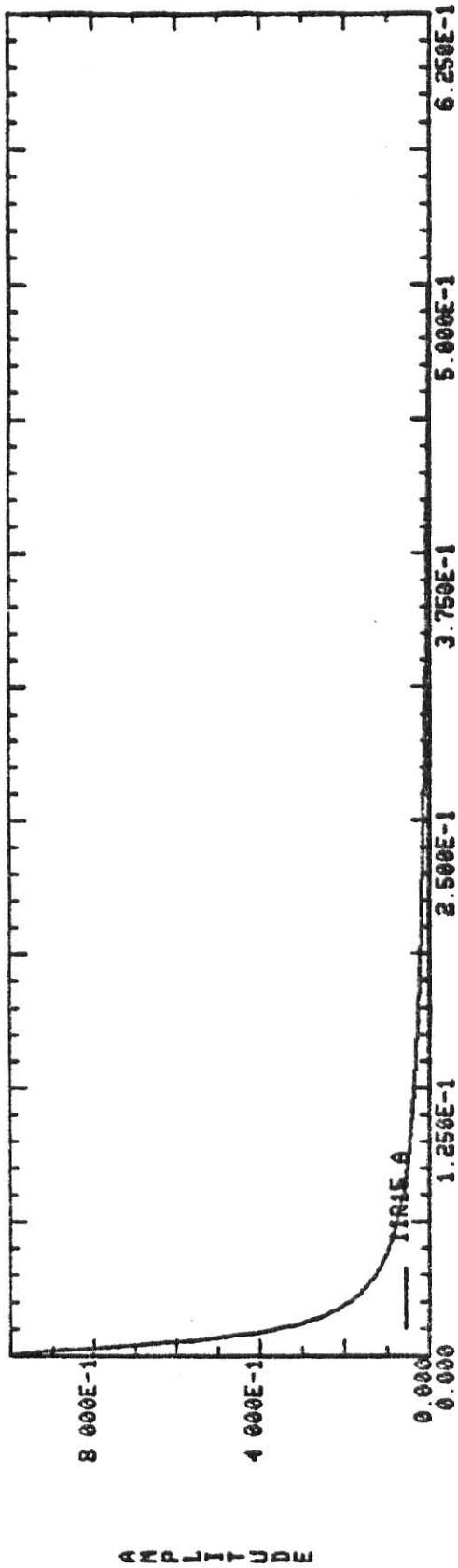


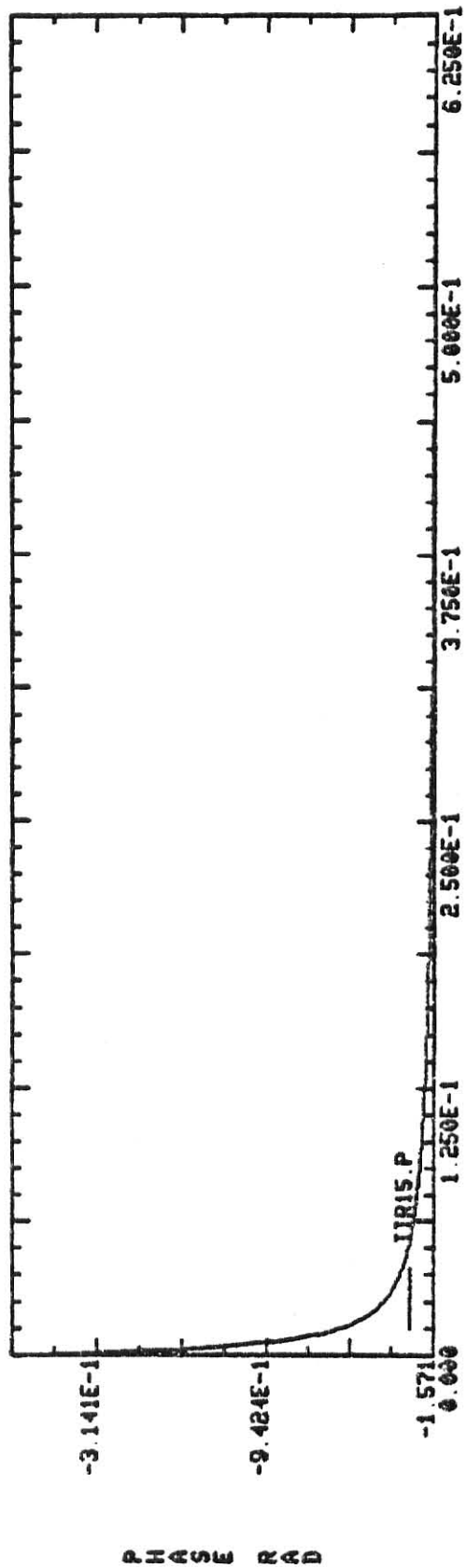
PHASE RAD

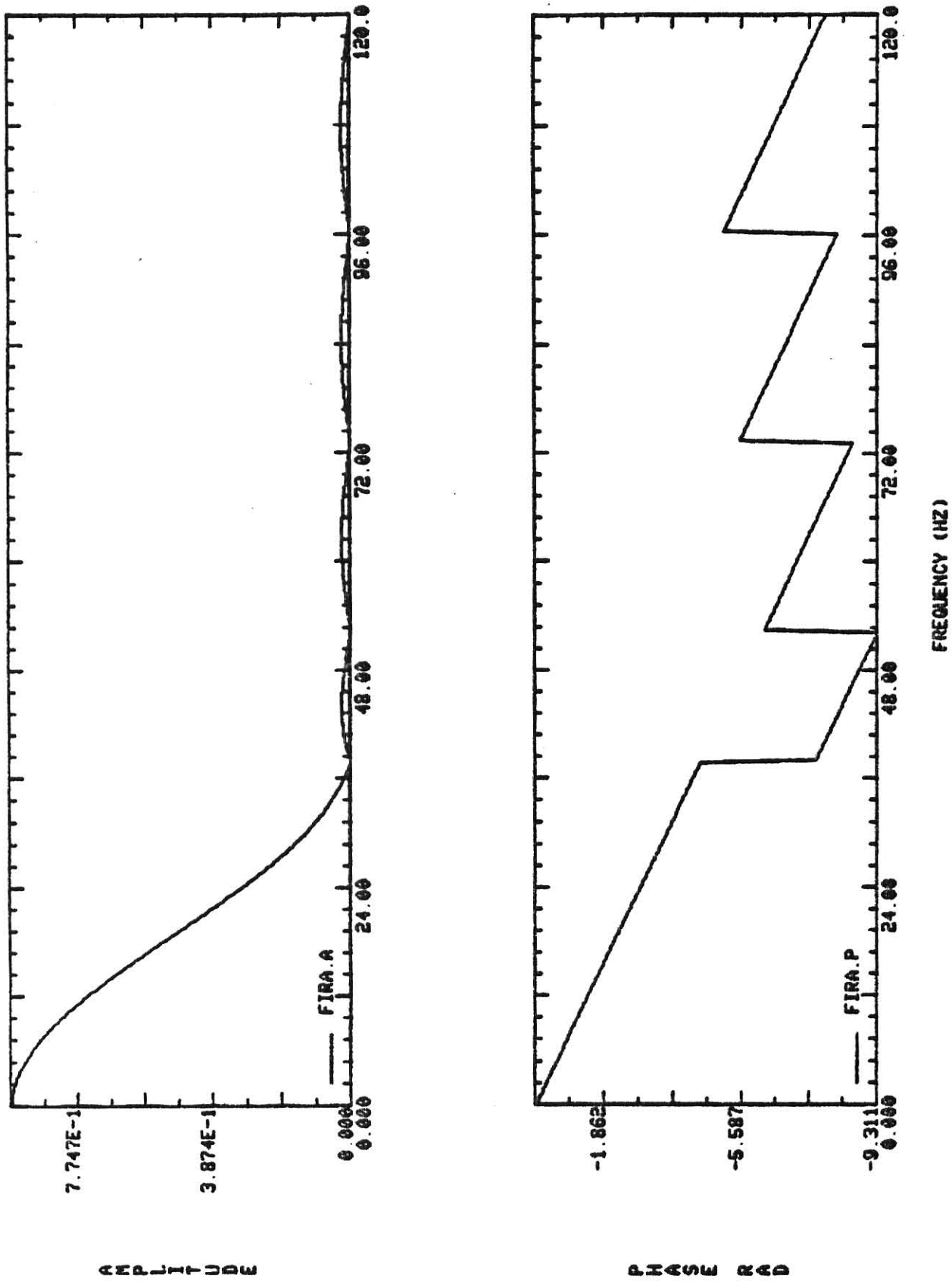
FREQUENCY (HZ)

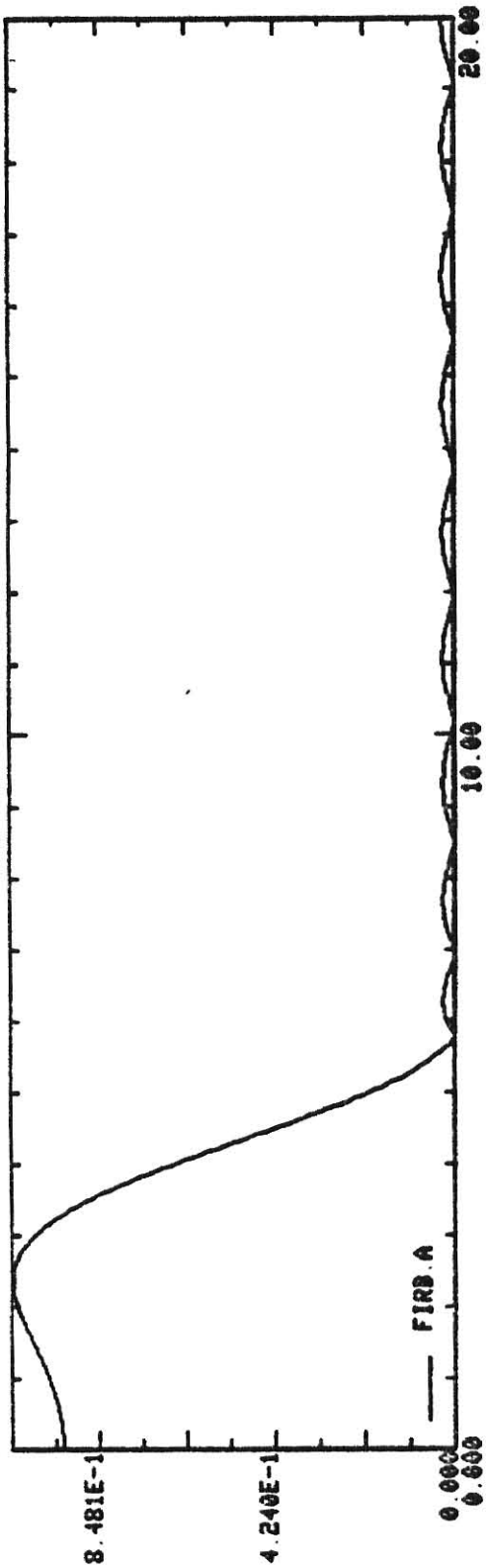

AMPLITUDE
8.000E-1
4.000E-1
0.000
0.000
2.000
4.000
IIR13.A
PHASE RAD
1.256
6.278E-1
0.000
0.000
2.000
4.000
IIR13.P
FREQUENCY (HZ)

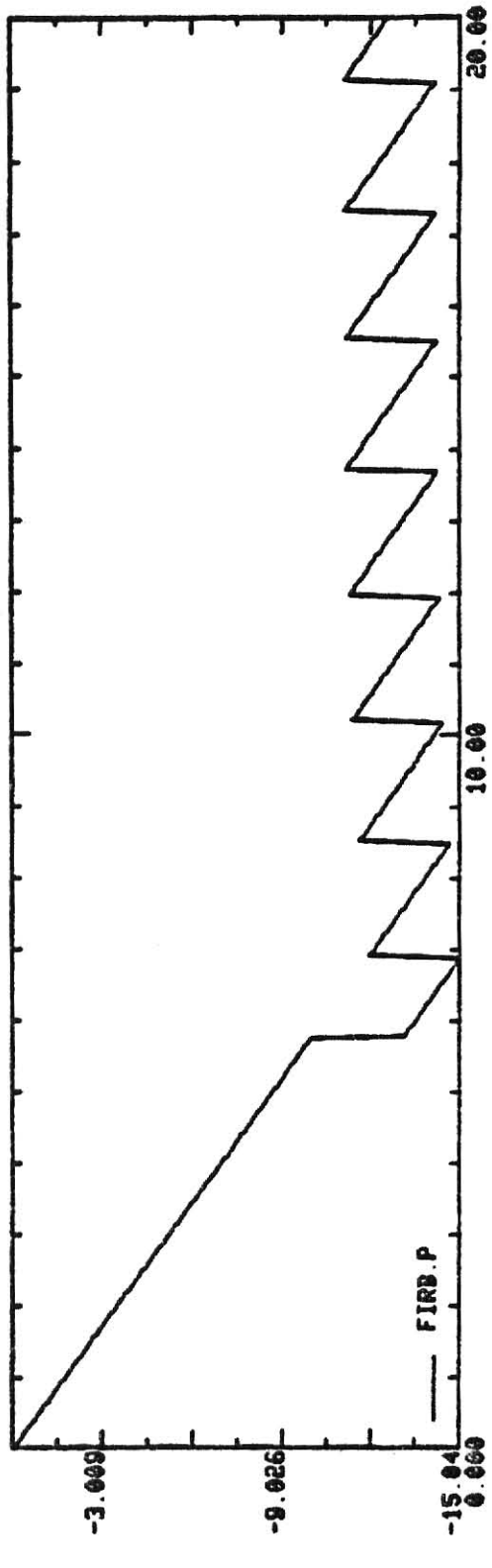

AMPLITUDE
8.000E-1
4.000E-1
0.000
0.000
2.500E-1
5.000E-1
7.500E-1
1.000
1.250
IIR14 A
PHASE RAD
-6.280E-1
-1.884
-3.140
0.000
2.500E-1
5.000E-1
7.500E-1
1.000
1.250
IIR14.P
FREQUENCY (HZ)

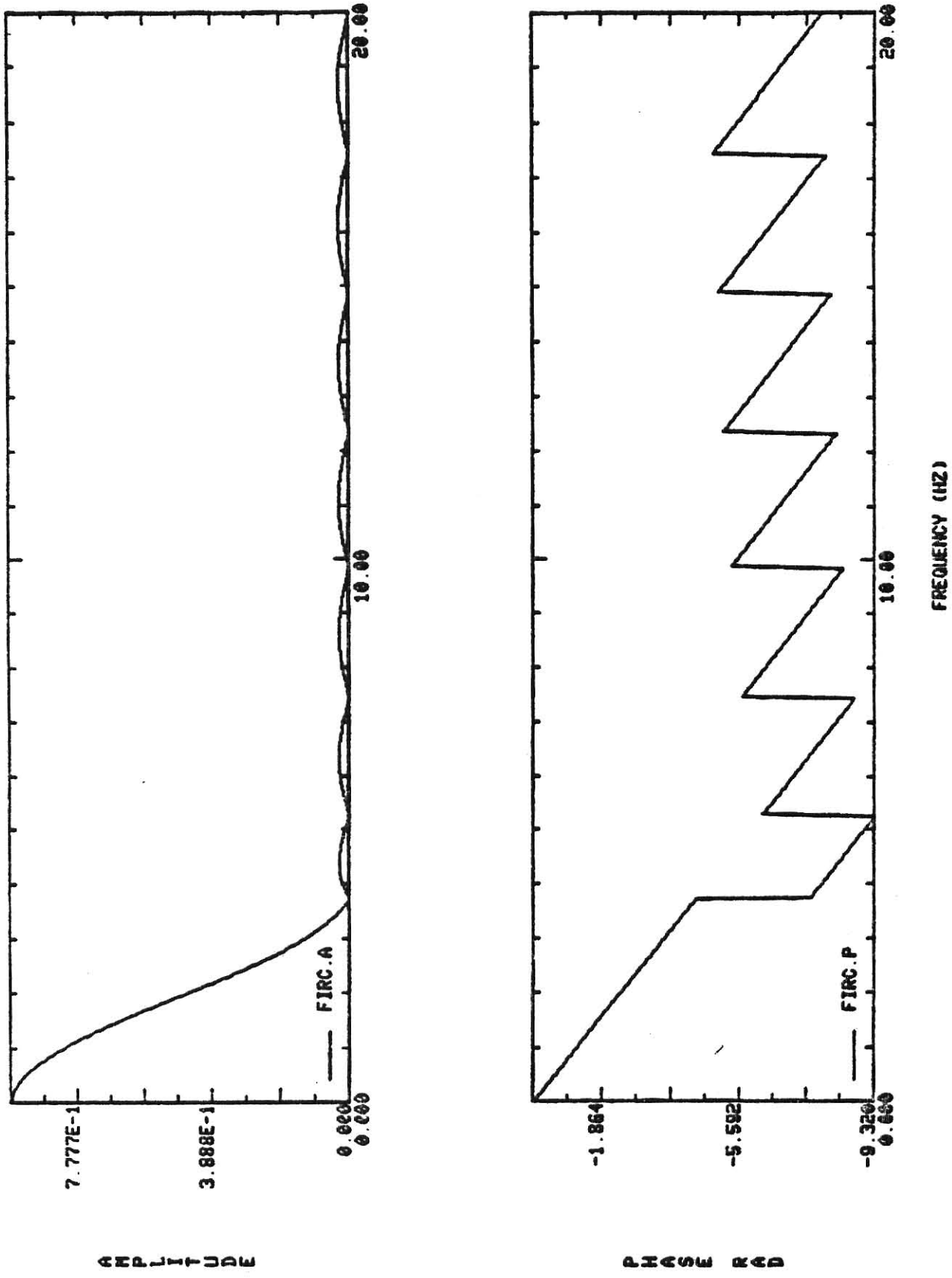

AMPLITUDE
8.000E-1
4.000E-1
0.000
0.000
1.250E-1
2.500E-1
3.750E-1
5.000E-1
6.250E-1
IIR15.A

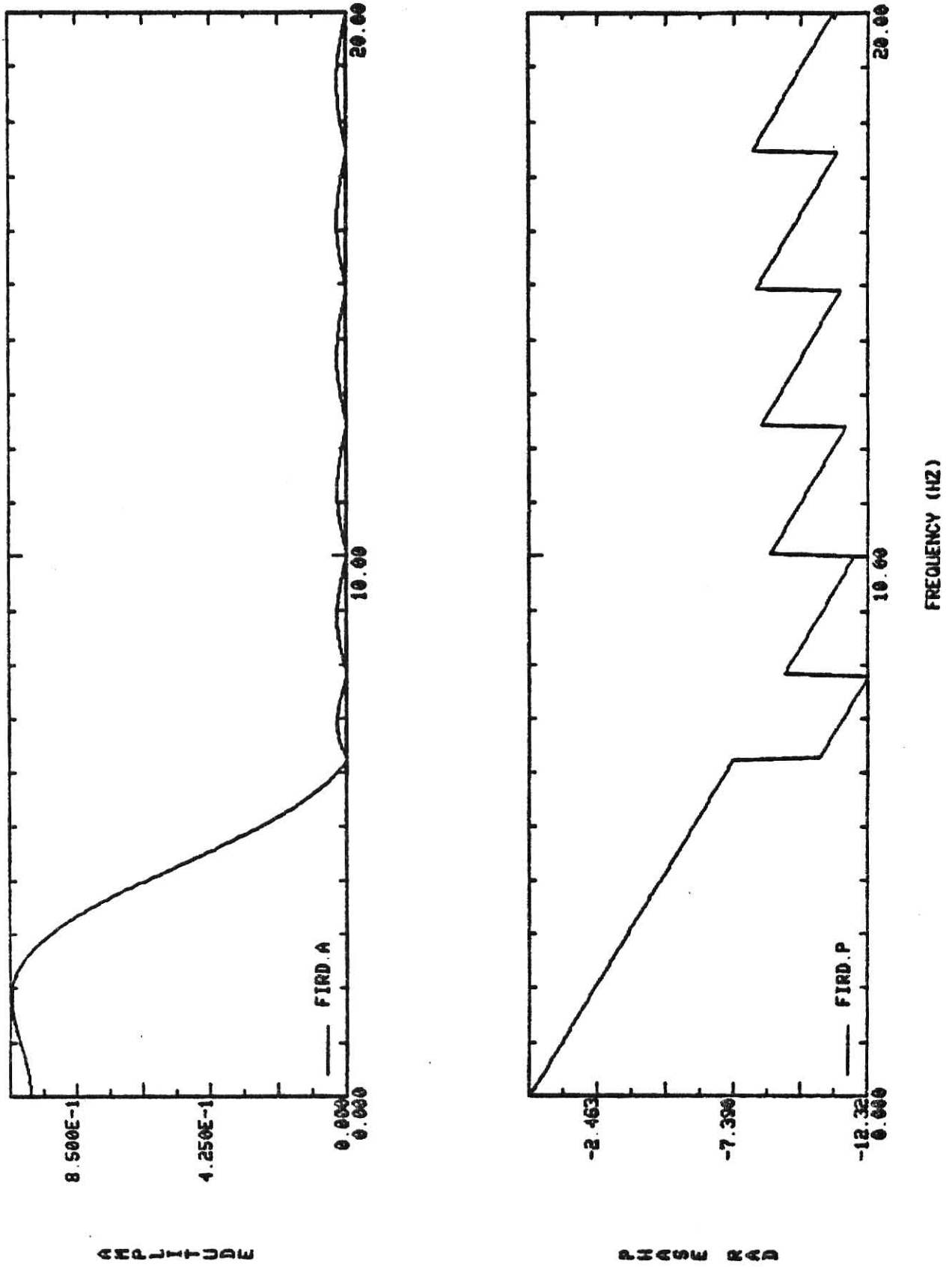

PHASE RAD
-3.141E-1
-9.424E-1
-1.571
0.000
1.250E-1
2.500E-1
3.750E-1
5.000E-1
6.250E-1
IIR15.P
FREQUENCY (HZ)

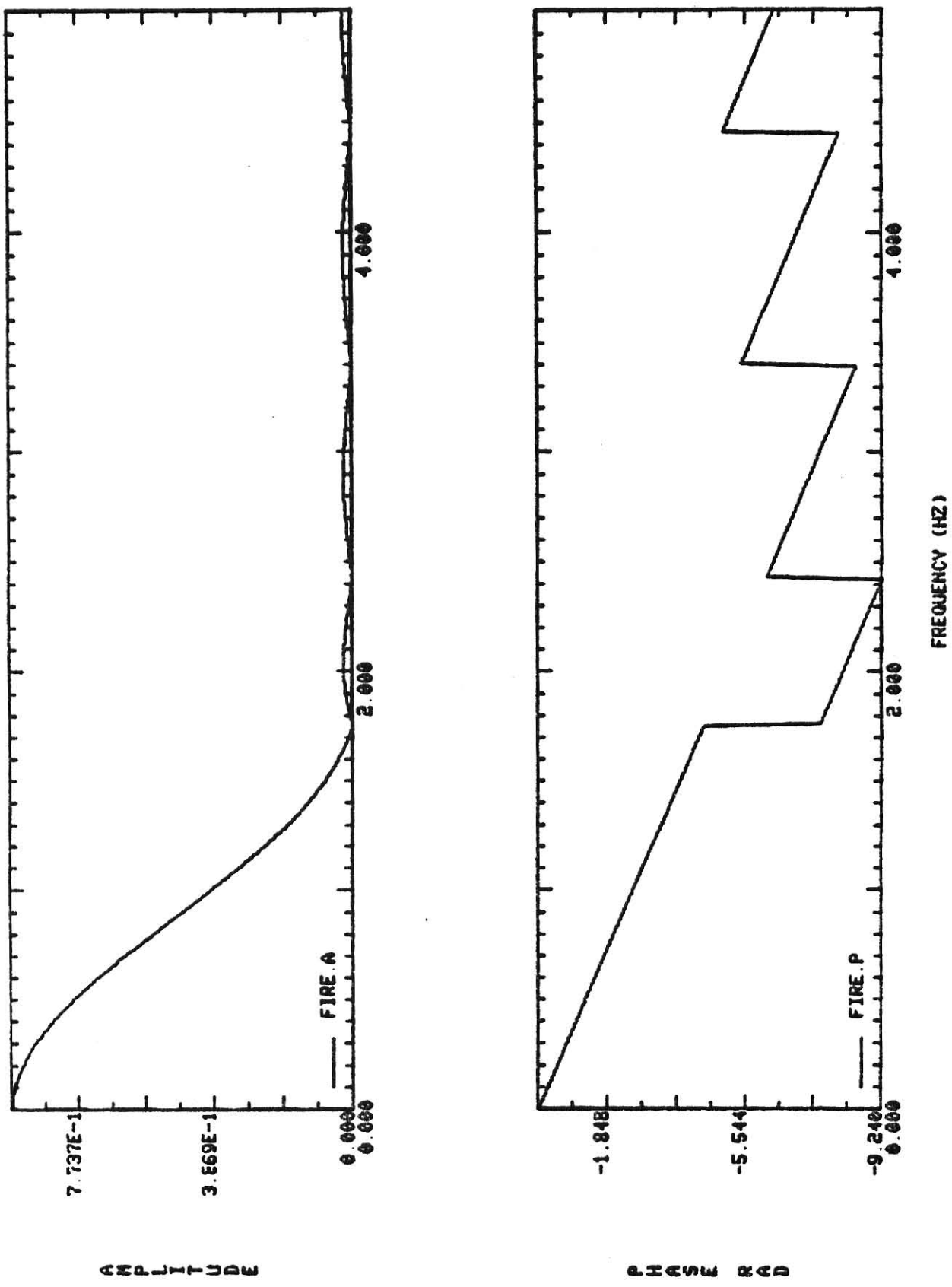

AMPLITUDE
7.747E-1
3.874E-1
0.000
0.000
24.00
48.00
72.00
96.00
120.0
FIRA.A
PHASE RAD
-1.862
-5.587
-9.311
0.000
24.00
48.00
72.00
96.00
120.0
FIRA.P
FREQUENCY (HZ)

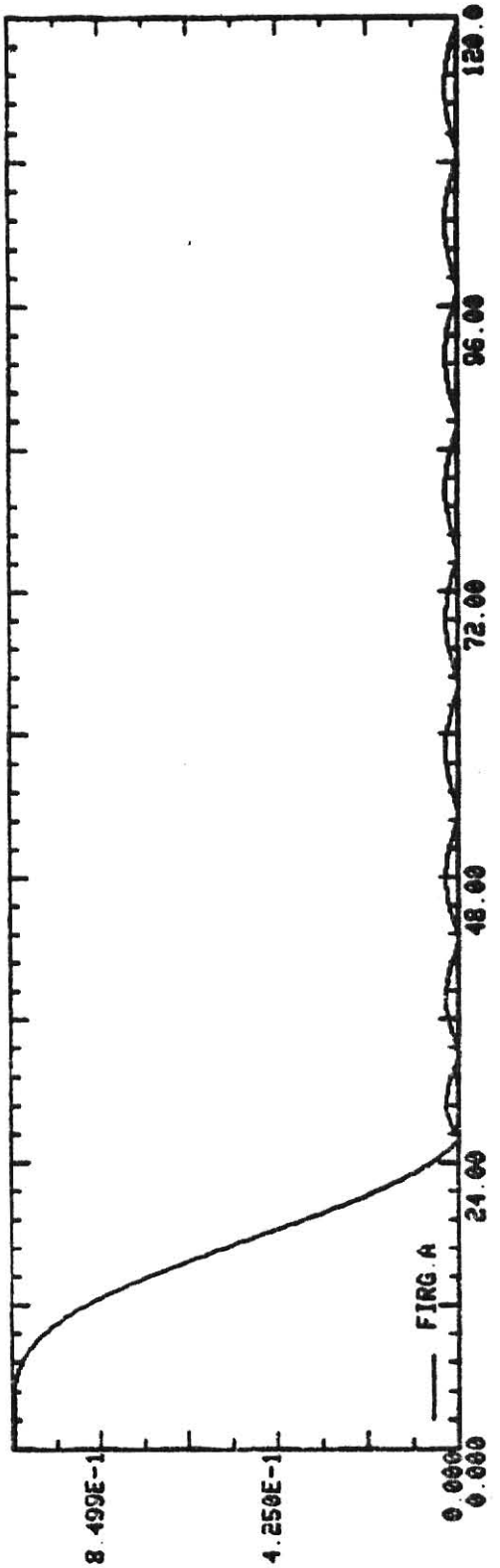

AMPLITUDE
8.481E-1
4.240E-1
0.000
0.000
10.00
20.00
FIRB.A

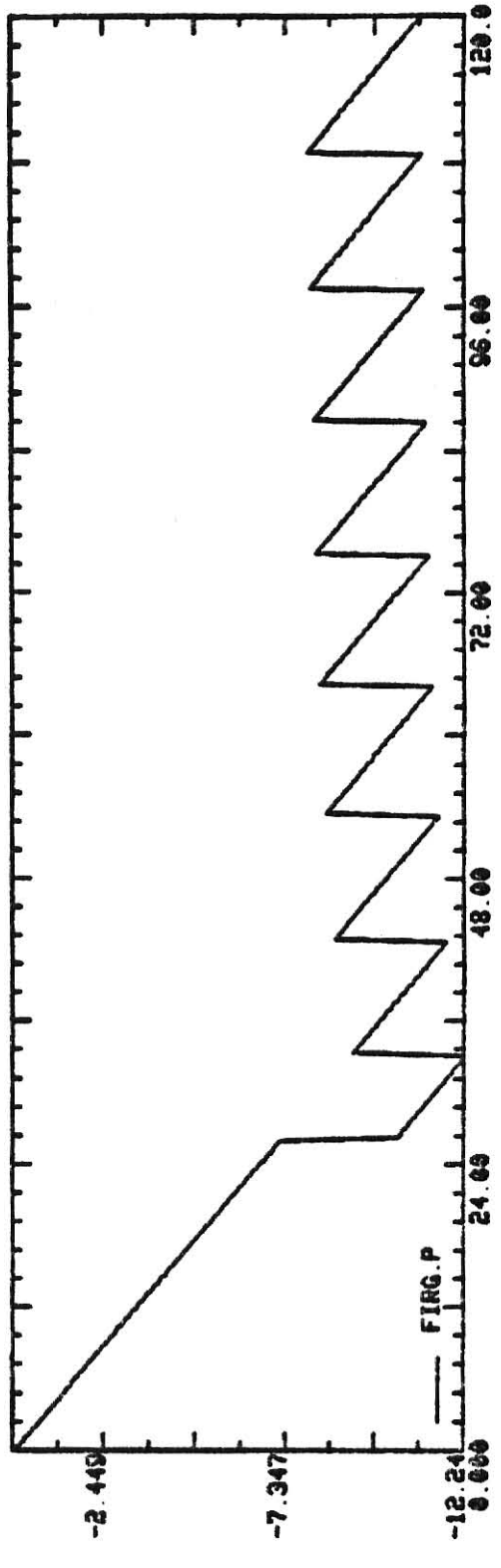

PHASE RAD
-3.000
-9.026
-15.04
0.000
10.00
20.00
FREQUENCY (HZ)
FIRB.P


AMPLITUDE
7.777E-1
3.888E-1
0.000
0.000
10.00
20.00
FIRC.A
PHASE RAD
-1.864
-5.592
-9.320
0.000
10.00
20.00
FIRC.P
FREQUENCY (HZ)


AMPLITUDE
8.500E-1
4.250E-1
0.000
0.000
10.00
20.00
FIRD.A
PHASE RAD
-2.463
-7.396
-12.32
0.000
10.00
20.00
FIRD.P
FREQUENCY (HZ)


AMPLITUDE
7.737E-1
3.869E-1
0.000
0.000
2.000
4.000
FIRE.A
PHASE RAD
-1.848
-5.544
-9.240
0.000
2.000
4.000
FIRE.P
FREQUENCY (HZ)

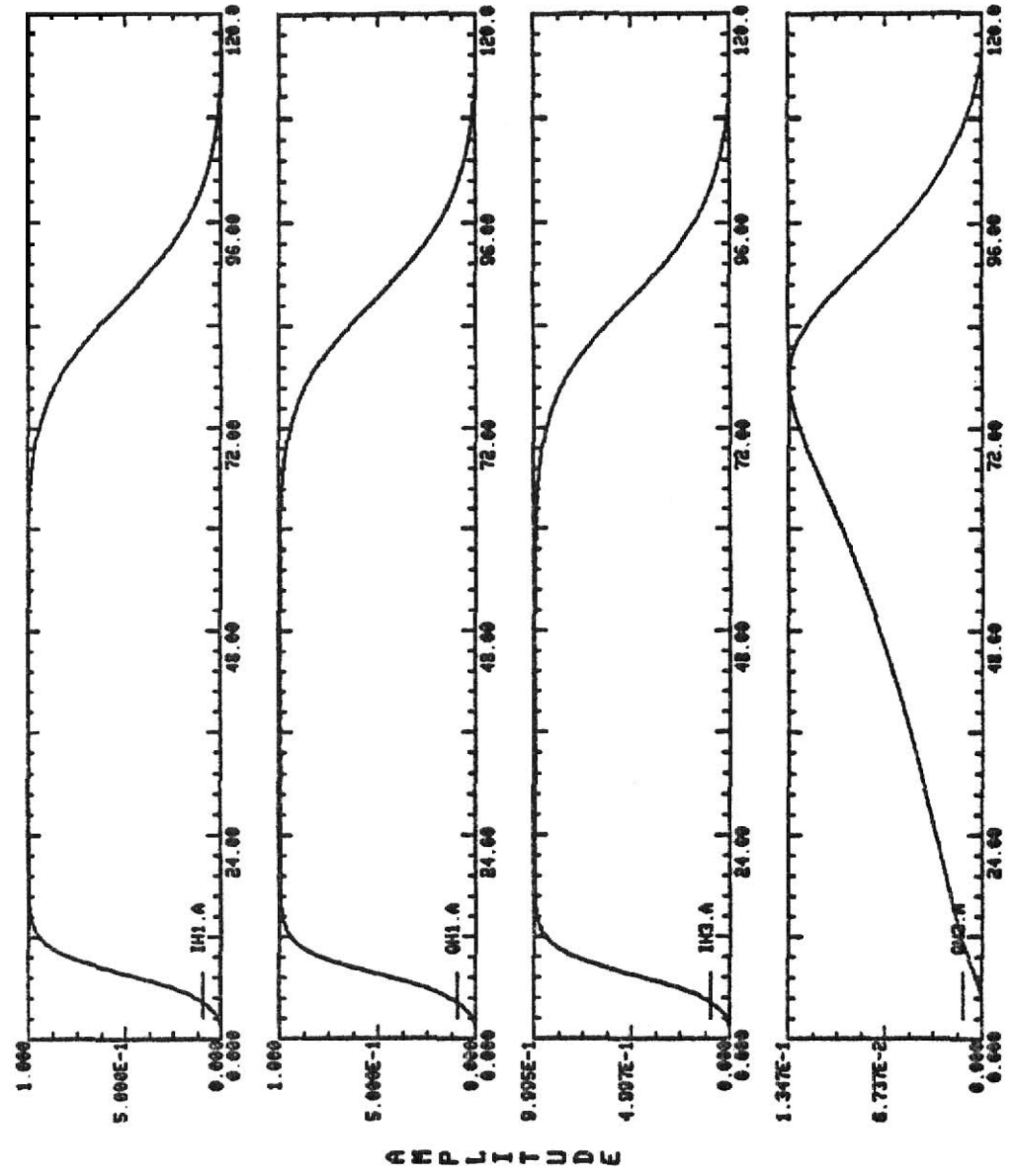

AMPLITUDE
8.499E-1
4.250E-1
0.000
0.000
24.00
48.00
72.00
96.00
120.0
FIRG.A


PHASE RAD
-2.449
-7.347
-12.24
0.000
24.00
48.00
72.00
96.00
120.0
FIRG.P
FREQUENCY (HZ)

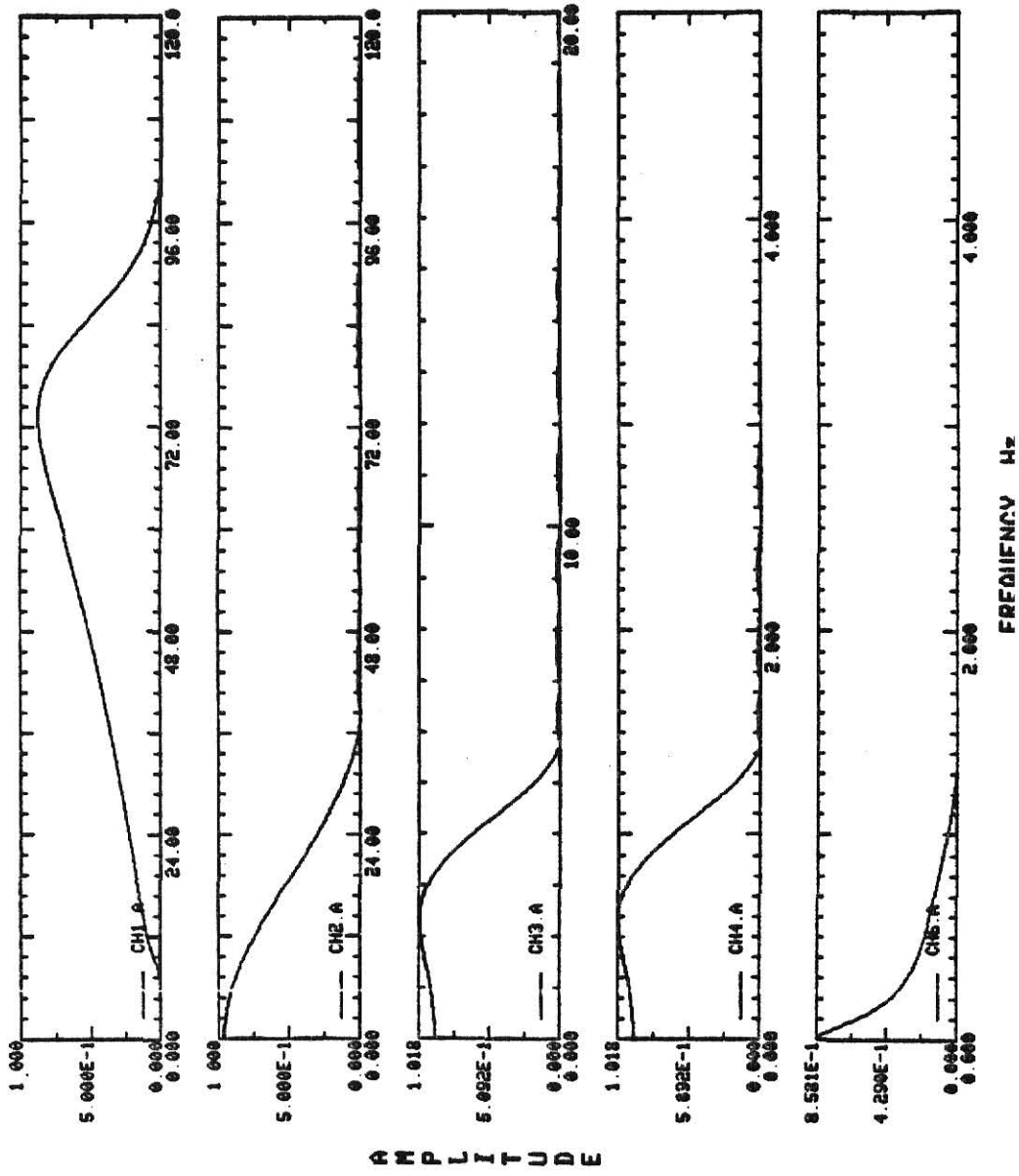
APPENDIX C

This appendix contains the amplitude versus frequency plots of the various output points of the simulation. These plots were used to determine the gain for some of the filters and to determine which input frequency would produce a maximum output and which frequency would produce a minimum output. The filenames in the lower left corner of each plot correspond to the output point as labelled in the filter block diagram in appendix A.

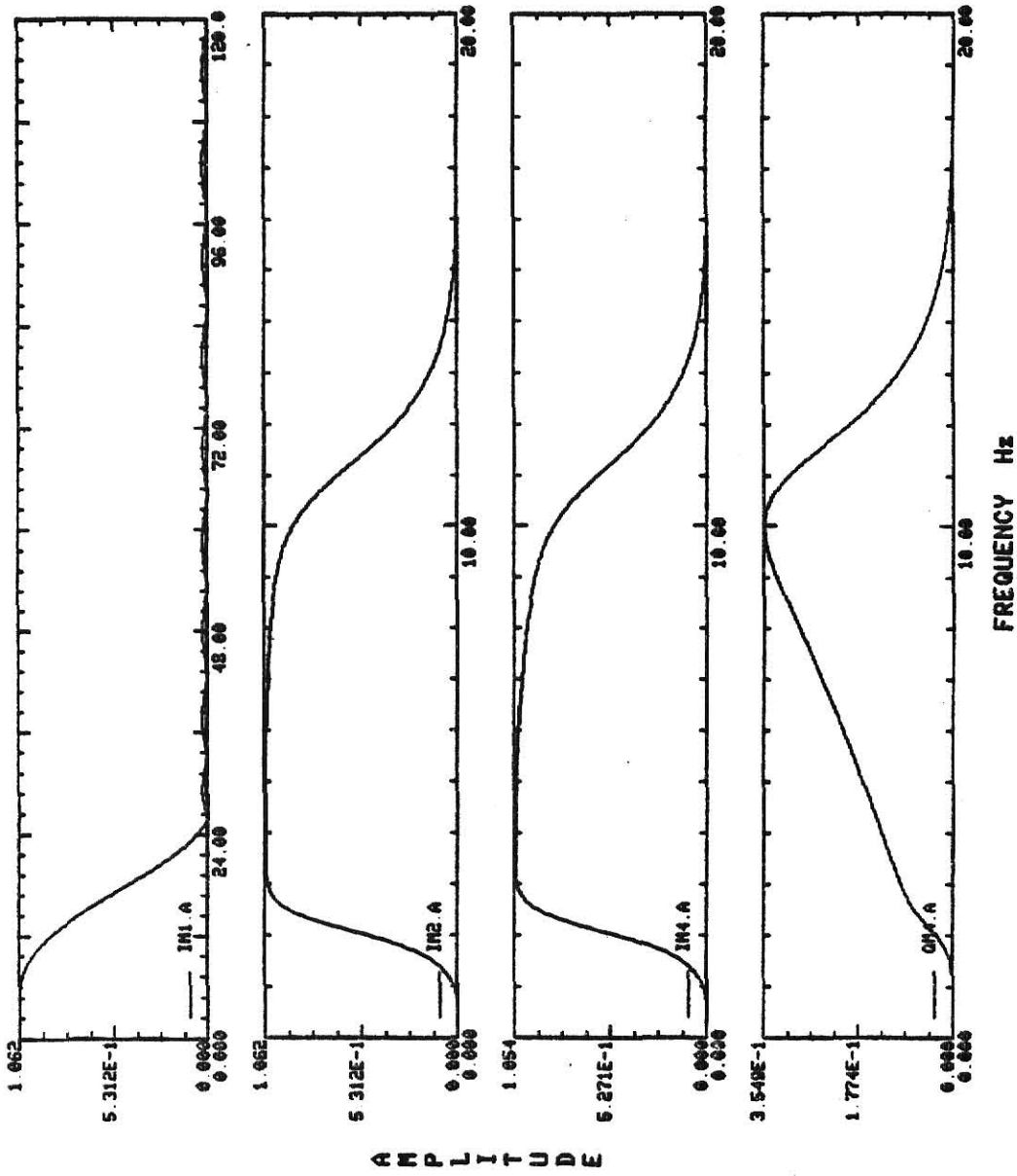
HIGH BAND


AMPLITUDE
FREQUENCY Hz
IH1.A
OH1.A
IH3.A
OH3.A
1.000
5.000E-1
0.000
1.000E-1
1.997E-1
1.347E-1
6.737E-2
24.00
48.00
72.00
96.00
120.0

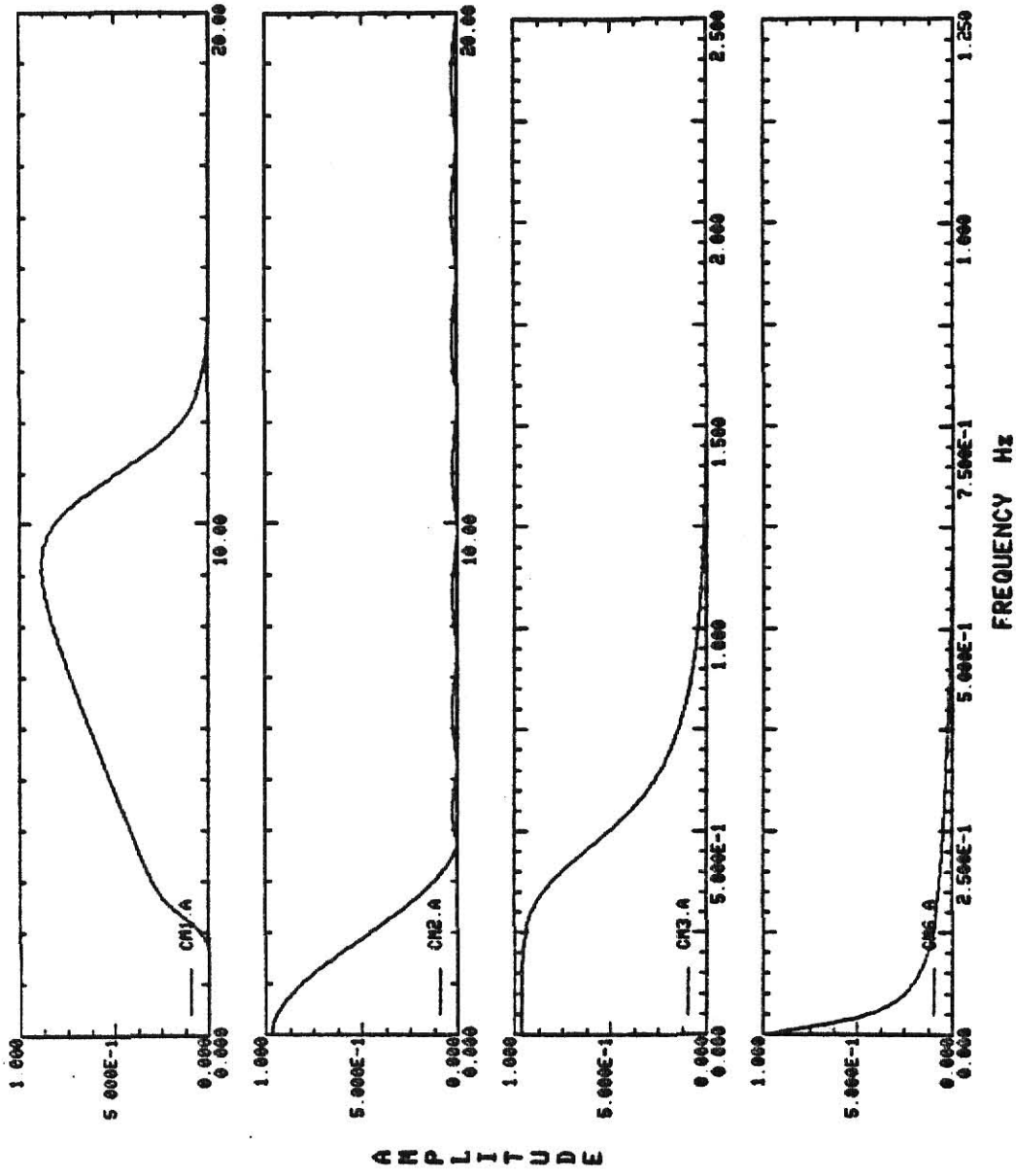
HIGH BAND



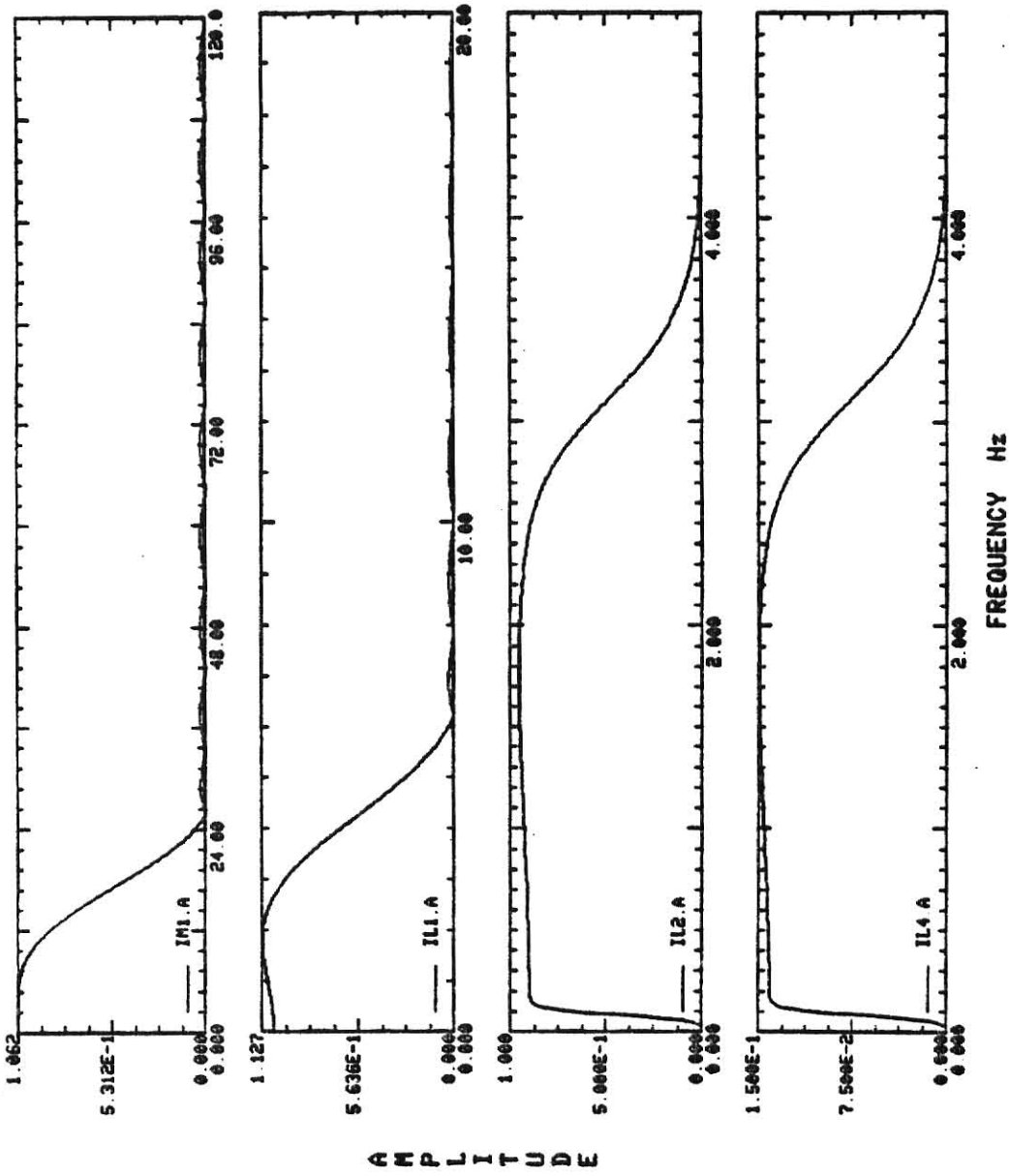
MID BAND



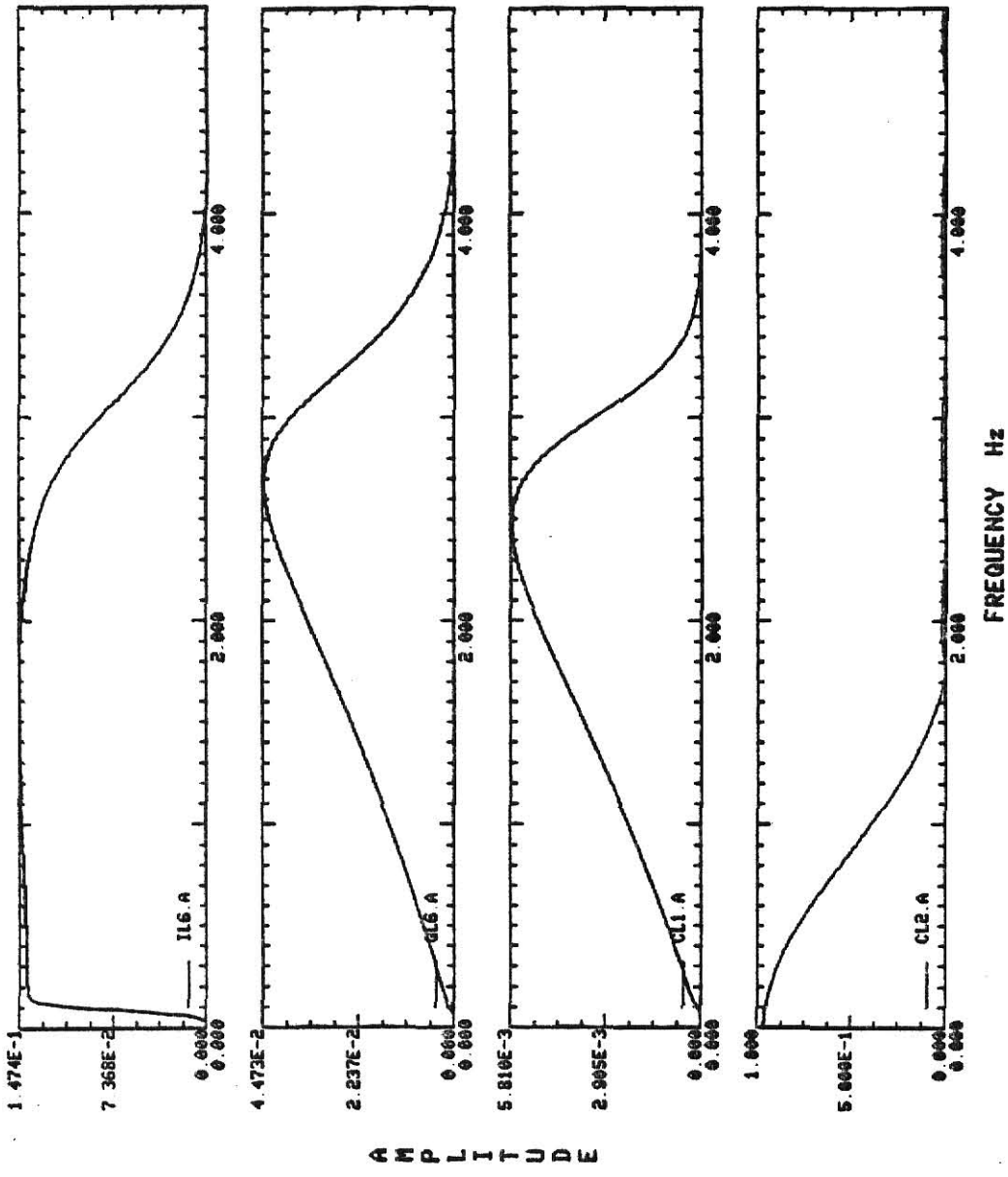
MID BAND



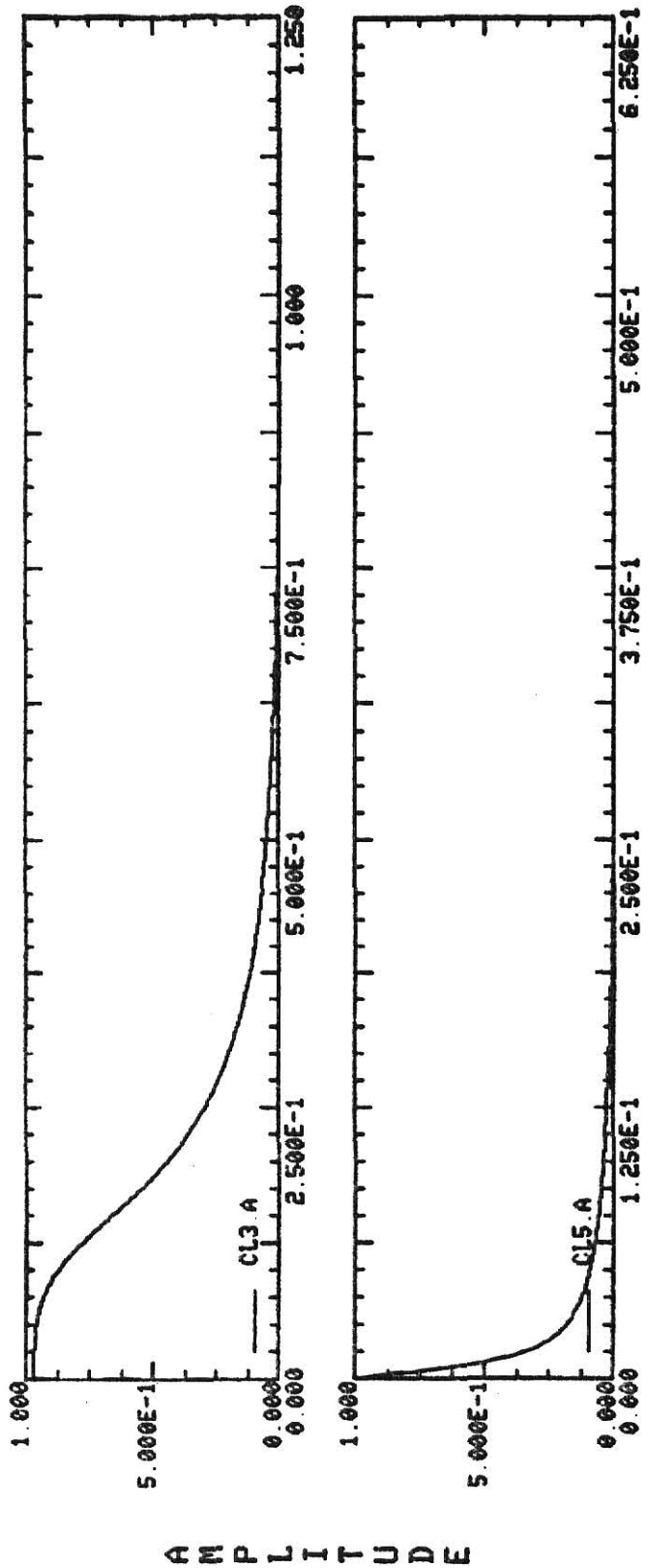
LOW BAND


1.062
5.312E-1
0.000
0.000
IM1.A
24.00
48.00
72.00
96.00
120.0
1.127
5.636E-1
0.000
0.000
IL1.A
10.00
20.00
1.000
5.000E-1
0.000
0.000
IL2.A
2.000
4.000
1.500E-1
7.500E-2
0.000
0.000
IL4.A
2.000
4.000
AMPLITUDE
FREQUENCY Hz

LOW BAND


1.474E-1
7.368E-2
0.000
0.000
2.000
4.000
IL6.A
4.473E-2
2.237E-2
0.000
0.000
2.000
4.000
GL6.A
5.810E-3
2.905E-3
0.000
0.000
2.000
4.000
CL1.A
1.000
5.000E-1
0.000
0.000
2.000
4.000
CL2.A
AMPLITUDE
FREQUENCY Hz

LOW BAND


1.000
5.000E-1
0.000
0.000
2.500E-1
5.000E-1
7.500E-1
1.000
1.250
CL3 A
1.000
5.000E-1
0.000
0.000
1.250E-1
2.500E-1
3.750E-1
5.000E-1
6.250E-1
CL5 A
AMPLITUDE
FREQUENCY Hz

APPENDIX D

This appendix contains listings of the programs used to do the simulation, and contains listings of support programs that were used. All of the programs except for one of the support programs are written in FORTRAN 77 and run on a VAX 11/750. The programs contain sufficient comments to explain what they are doing and how to use them. The following is a list of the programs contained in this appendix:

Simulation programs:

Support programs:

```
C*******************************************************************************
C
C       HIB
C
C       VAX-11 FORTRAN SOURCE FILENAME:         HIB.FOR
C
C       DEPARTMENT OF ELECTRICAL ENGINEERING    KANSAS STATE UNIVERSITY
C
C       REVISION        DATE                    PROGRAMMER
C       --------        ----                    ----------
C       00.0            NOV 24, 1982            DANIEL B. SCHOWENGERDT
C       01.0            MAY 26, 1983            JOHN W. BARTHOLOMEW
C
C*******************************************************************************
C
C       CALLING SEQUENCE
C
C               RUN HIB
C
C       PURPOSE
C               The purpose of this program is to perform, using double
C               precision arithmetic, the high frequency channel of the
C               LAMBAKINS algorithm.  Double precision arithmetic
C               is used so that comparisons can be made to other
C               versions of this program that use fixed point
C               arithmetic.
C
C       ROUTINE(S) CALLED BY THIS PROGRAM
C
C               FIRF - This subroutine performs FIR filtering.
C
C               IIRF - This function subroutine performs IIR filtering.
C
C       DATA REQUIRED FROM USER
C
C
C               'HIBATCH.DAT' - Data file containing the name of the
C                  file containing the required inputs to the program.
C
C               IN PHASE DATA FILE (SAMPLING RATE 240 HZ)
C
C               QUADRATURE DATA FILE (SAMPLING RATE 240 HZ)
C
C               FILTER COEFFICIENT FILE - The order of the coefficients
C                 in this file is:
C                 1) IIR1 coefficients - a four pole + two pole
C                    a) first four pole coefficients - 9 weight
C                    b) the two pole - 5 weight.
C                 2) IIR2 coefficients - 3 weight.
C                 3) IIR3 coefficients - 3 weight.
C                 4) FIRA coefficients - 10 weight.
C                 5) FIRB - coefficients - 23 weight.
C                 6) IIR4 coefficients - a 2 pole + 1 pole
```

```
C                   a) 2 pole coefficients - 5 weight.
C                   b) 1 pole - 3 weight.
C                 7) IIR5 coefficients - 3 weight.
C
C               DBGAIN - The variable gain applied to the algorithm.
C
C       INFORMATION RETURNED TO USER
C
C               The data generated in the program selected from
C               any of 9 different outspots.
C                1) Inphase IIR1 * gain output.
C                2) Quadrature IIR1 * gain output.
C                3) Inphase limited IIR2 output.
C                4) Quadrature IIR3 output.
C                5) Inphase * Quadrature output.
C                6) FIRA output.
C                7) FIRB output.
C                8) Limited IIR4 output.
C                9) IIR5 output.
C
C*************************************************************************
C
        PARAMETER MAX_ARRAY_SIZE=200000
        REAL SCRATCH1(MAX_ARRAY_SIZE)
        REAL SCRATCH2(MAX_ARRAY_SIZE)
        REAL SCRATCH3(MAX_ARRAY_SIZE)
        INTEGER OUTCOUNT/0/
C
        CHARACTER*80 FNAM
        CHARACTER*80 INFNAM
        CHARACTER*80 QUFNAM
        CHARACTER*80 COFNAM
        CHARACTER*80 OUTPUT
C
        DOUBLE PRECISION DI1IA(9)/9*0.0/,DI1IB(5)/5*0.0/,
     *                   DI1QA(9)/9*0.0/,DI1QB(5)/5*0.0/,
     *                   DI2I(3)/3*0.0/,DI3Q(3)/3*0.0/,
     *                   DF1(10)/10*0.0/,DF3(23)/23*0.0/,
     *                   DI4A(5)/5*0.0/,DI4B(3)/3*0.0/,
     *                   DI5(3)/3*0.0/
C
        DOUBLE PRECISION CI1A(9),CI1B(5),CI2I(3),CI3Q(3),
     *                   CF1(10),CF3(23),CI4A(5),CI4B(3),
     *                   CI5(3),C(64)
C
        DOUBLE PRECISION GAIN,I,Q,IIRF,INTOUT
C
        REAL DBGAIN,SIZE,OUT(9)
        REAL COEF(64)
C
        INTEGER DID,FOUR/4/,ICNT1/0/,ICNT2/0/,ICNT3/0/,IER,I00/0/,
     *          I01/1/,I02/2/,IOUT,ISIZE,NC10/10/,NC23/23/,
     *          NCNT1/0/,NCNT2/0/,NDEC4/4/,NDEC6/6/,NR/0/,
```

```
     *          ONE/1/,TWO/2/,J
C
      EQUIVALENCE (C(1),CI1A(1)),(C(10),CI1B(1)),(C(15),CI2I(1)),
     *            (C(18),CI3Q(1)),(C(21),CF1(1)),(C(31),CF3(1)),
     *            (C(54),CI4A(1)),(C(59),CI4B(1)),(C(62),CI5(1))
C
C
 10     FORMAT (A)
 11     FORMAT (F10.6)
 12     FORMAT (I4)
C
        FNAM = 'HIBATCH.DAT'
        OPEN (UNIT=4,FILE=FNAM,STATUS='OLD')
        READ (4,10) FNAM
C
        OPEN (UNIT=3,FILE=FNAM,STATUS='OLD')
C
 85     CONTINUE
C
        DO 86 J=1,9
                DI1IA(J) = 0.0D0
                DI1QA(J) = 0.0D0
 86     CONTINUE
        DO 87 J=1,5
                DI1IB(J) = 0.0D0
                DI1QB(J) = 0.0D0
                DI4A(J) = 0.0D0
 87     CONTINUE
        DO 88 J=1,3
                DI2I(J) = 0.0D0
                DI3Q(J) = 0.0D0
                DI4B(J) = 0.0D0
                DI5(J) = 0.0D0
 88     CONTINUE
        DO 89 J=1,10
                DF1(J) = 0.0D0
 89     CONTINUE
        DO 90 J=1,23
                DF3(J) = 0.0D0
 90     CONTINUE
        ICNT1 = 0
        ICNT2 = 0
        ICNT3 = 0
C
        READ (3,10,END=3000) INFNAM
        READ (3,10,END=3000) QUFNAM
        READ (3,10,END=3000) CDFNAM
        READ (3,12,END=3000) IOUT
        READ (3,11,END=3000) DBGAIN
        READ (3,10,END=3000) OUTPUT
C
        CALL SGOPEN(IO0,'READ','NOPROMPT',INFNAM,'REAL',ISIZE0)
        CALL SGOPEN(IO1,'READ','NOPROMPT',QUFNAM,'REAL',ISIZE1)
```

```
      CALL SGOPEN(IO2,'READ','NOPROMPT',COFNAM,'REAL',ISIZE2)
C
      CALL SGTRAN(IO2,'READ','REAL',COEF,ISIZE2)
      DO 100 I1=1,ISIZE2
            C(I1) = DBLE(COEF(I1))
100   CONTINUE
C
      GAIN = DBLE(10.0 ** (DBGAIN/20.0))
C
C
C     READ THE TWO INPUT FILES
C
C
      CALL SGTRAN(IO0,'READ','REAL',SCRATCH1,ISIZE0)
      CALL SGTRAN(IO1,'READ','REAL',SCRATCH2,ISIZE1)
C
      DO 500 NR=1,MIN0(ISIZE0,ISIZE1)
C
      DI1IA(1) = DBLE( SCRATCH1(NR) )
      DI1QA(1) = DBLE( SCRATCH2(NR) )
C
C     PERFORM 6 POLE BAND PASS FILTER 9.0-82.5 HZ.
C        REQUIRES DOUBLE PRECISION DI1IA(9),DI1IB(5),
C        DI1QA(9),DI1QB(5),CI1A(9),CI1B(5).
C
C     DO INPHASE CHANNEL
C
      DI1IB(1) = IIRF(DI1IA,CI1A,FOUR)
      DI2I(1) = IIRF(DI1IB,CI1B,TWO)
C
C     DO QUADRATURE CHANNEL
C
      DI1QB(1) = IIRF(DI1QA,CI1A,FOUR)
      DI3Q(1) = IIRF(DI1QB,CI1B,TWO)
C
C     MULTIPLY IN GAIN
C
      DI2I(1) = DI2I(1) * GAIN
      DI3Q(1) = DI3Q(1) * GAIN
C
C     DO ONE POLE LOW PASS FILTER IIR2  112.5 HZ
C        REQUIRES DI2I(3),CI2(3)
C
      I = IIRF(DI2I,CI2I,ONE)
C
C     DO ONE POLE HIGH PASS FILTER IIR3 112.5 HZ
C        REQUIRES DI3Q(3),CI3(3)
C
      Q = IIRF(DI3Q,CI3Q,ONE)
C
C     LIMIT INPHASE SIGNAL TO +-2.8
C
      IF (DABS(I).GT.2.8D0) I = DSIGN(2.8D0,I)
```

```
C
C       FIND PRODUCT OF INPHASE AND QUADRATURE SIGNAL
C
        DF1(1) = I * Q
C
C       OUTPUT DESIRED VALUES
C
        IF (IOUT.GT.5) GO TO 310
        ICNT1 = ICNT1 + 1
        OUT(1) = DI2I(2)
        OUT(2) = DI3Q(2)
        OUT(3) = I
        OUT(4) = Q
        OUT(5) = DF1(1)
        OUTCOUNT = OUTCOUNT + 1
        SCRATCH3( OUTCOUNT ) = OUT( IOUT )
        GO TO 500
 310    CONTINUE
C
C
C       DECIMATE SIGNAL BY 6 WITH FIR1
C          REQUIRES DF1(10),CF1(10)
C
        CALL FIRF(DF1,CF1,NC10,NDEC6,NCNT1,DF3(1),DID)
        IF (DID.EQ.0) GO TO 500
C
C       OUTPUT THIS VALUE?
C
        IF (IOUT.NE.6) GO TO 320
        ICNT2 = ICNT2 + 1
        OUT(6) = DF3(1)
        OUTCOUNT = OUTCOUNT + 1
        SCRATCH3( OUTCOUNT ) = OUT ( IOUT )
        GO TO 500
 320    CONTINUE
C
C       DECIMATE SIGNAL BY 4 WITH FIR3
C          REQUIRES DF3(23),CF3(23)
C
        CALL FIRF(DF3,CF3,NC23,NDEC4,NCNT2,DI4A(1),DID)
        IF (DID.EQ.0) GO TO 500
C
C       PERFORM 3 POLE LOW PASS FILTER IIR4  3.3 HZ
C         REQUIRES DI4A(5),CI4A(5),DI4B(3),CI4B(3)
C
        DI4B(1) = IIRF(DI4A,CI4A,TWO)
        DI5(1) = IIRF(DI4B,CI4B,ONE)
C
C       LIMIT SIGNAL TO +- 1.0
C
        IF (DABS(DI5(1)).GT.1.0D0) DI5(1) = DSIGN(1.0D0,DI5(1))
C
C       PERFORM ONE POLE LOW PASS FILTER IIR5 0.109 HZ
```

```
C          REQUIRES DI5(3),CI5(3)
C
        INTOUT = IIRF(DI5,CI5,ONE)
C
C       OUTPUT ANY VALUES?
C
        IF (IOUT.LT.7) GO TO 500
        ICNT3 = ICNT3 + 1
        OUT(7) = DI4A(2)
        OUT(8) = DI5(2)
        OUT(9) = INTOUT
        OUTCOUNT = OUTCOUNT + 1
        SCRATCH3( OUTCOUNT ) = OUT( IOUT )
C
 500    CONTINUE
        CALL SGOPEN(IO2,'WRITE','NOPROMPT',OUTPUT,'REAL',OUTCOUNT)
        CALL SGTRAN(IO2,'WRITE','REAL',SCRATCH3,OUTCOUNT)
        OUTCOUNT = 0
C
        GOTO 85
C
3000    CONTINUE
        END
```

```
C*************************************************************************
C
C       HIBAND
C
C       VAX-11 FORTRAN SOURCE FILENAME:         HIBAND.FOR
C
C       DEPARTMENT OF ELECTRICAL ENGINEERING    KANSAS STATE UNIVERSITY
C
C       REVISION        DATE                    PROGRAMMER
C       --------        ----                    ----------
C       00.0            NOV 29, 1982            DANIEL B. SCHOWENGERDT
C       01.0            MAY 26, 1983            JOHN W. BARTHOLOMEW
C
C*************************************************************************
C
C       CALLING SEQUENCE
C
C               RUN HIBAND
C
C       PURPOSE
C               The purpose of this program is to perform, using fixed
C               point arithmetic and data storage, the high
C               frequency channel of the LAMBAKINS algorithm.  The
C               attempt in this program is to simulate the limitations
C               that would occur if this algorithm were executed by
C               a microprocessor.  The simulated limitations are in the
C               precision of data storage and math operations.
C
C       ROUTINE(S) CALLED BY THIS PROGRAM
C
C               IIRFILT A function subroutine that performs an IIR
C                       filter.
C
C               FIRFILT A subroutine that performs an FIR filter.
C
C               MULT    A function subroutine that performs a fixed
C                       point multiply of two operands.
C
C       DATA REQUIRED FROM USER
C
C               'HIFBATCH.DAT' - Data file containing the name of the
C                  file containing the required inputs to the program.
C
C               INPHASE DATA FILE
C
C               QUADRATURE DATA FILE
C
C               FILTER COEFFICIENT FILE - The order of the coefficients
C                 in this file are:
C                 1) IIR1 coefficients - a four pole + a two pole
C                   a) the four pole - 9 weights.
C                   b) the two pole - 5 weights.
C                 2) IIR2 - 1 pole - 3 weights.
```

```
C                3) IIR3 - 1 pole - 3 weights.
C                4) FIRA coefficients - 10 weights.
C                5) FIRB coefficients - 23 weights.
C                6) IIR4 coefficients - a two pole + a one pole.
C                  a) the two pole - 5 weights.
C                  b) the one pole - 3 weights.
C                7) IIR5 coefficients - 1 pole - 3 weights.
C
C              DATA FORMAT FILE - Contains the fixed point format
C                specifiers for all data storage and math
C                operations in the program.  The order of formats is:
C                1) IIR1 formats.
C                  a) Coefficient format.
C                  b) Data format.
C                  c) Accumulator format.
C                2) GAIN x Data (inphase and quad).
C                  a) Format for GAIN value.
C                  b) Data format.
C                3) IIR2 and IIR3 formats.
C                  a) Coefficient format.
C                  b) Data format.
C                  c) Accumulator format.
C                4) Inphase x Quad data formats.
C                  a) Inphase data format.
C                  b) Quadrature data format.
C                5) FIRA formats.
C                  a) Coefficient format.
C                  b) Data format.
C                  c) Accumulator format.
C                6) FIRB formats.
C                  a) Coefficient format.
C                  b) Data format.
C                  c) Accumulator format.
C                7) IIR4 formats.
C                  a) Coefficient format.
C                  b) Data format.
C                  c) Accumulator format.
C                8) IIR5 formats.
C                  a) Coefficient format.
C                  b) Data format.
C                  c) Accumulator format.
C
C              DBGAIN - The variable gain to be applied to algorithm.
C
C       INFORMATION RETURNED TO USER
C
C              The data generated in the program selected from
C              any of 9 different spots.
C               1) Inphase IIR1 * GAIN output.
C               2) Quadrature IIR1 * GAIN output.
C               3) Inphase limited IIR2 output.
C               4) Quadrature IIR3 output.
C               5) Inphase * Quadrature output.
```

```
C                6) FIRA output.
C                7) FIRB output.
C                8) Limited IIR4 output.
C                9) IIR5 output.
C
C*************************************************************************
C
      PARAMETER MAX_ARRAY_SIZE=200000
      REAL SCRATCH1(MAX_ARRAY_SIZE)
      REAL SCRATCH2(MAX_ARRAY_SIZE)
      REAL SCRATCH3(MAX_ARRAY_SIZE)
      INTEGER OUTCOUNT/0/
C
      CHARACTER*80 FNAM
      CHARACTER*80 INFNAM
      CHARACTER*80 QUFNAM
      CHARACTER*80 COFNAM
      CHARACTER*80 FOFNAM
      CHARACTER*80 OUTPUT
C
      DOUBLE PRECISION DI1IA(9)/9*0.0/,DI1IB(5)/5*0.0/,
     *                 DI1QA(9)/9*0.0/,DI1QB(5)/5*0.0/,
     *                 DI2I(3)/3*0.0/,DI3Q(3)/3*0.0/,
     *                 DF1(10)/10*0.0/,DF3(23)/23*0.0/,
     *                 DI4A(5)/5*0.0/,DI4B(3)/3*0.0/,
     *                 DI5(3)/3*0.0/
C
      DOUBLE PRECISION CI1A(9),CI1B(5),CI2I(3),CI3Q(3),
     *                 CF1(10),CF3(23),CI4A(5),CI4B(3),
     *                 CI5(3),C(64)
C
      DOUBLE PRECISION GAIN,I,Q,IIRFILT,INTOUT,MULT
C
      REAL DBGAIN,SIZE,OUT(9),F(50)/50*0.0/
      REAL COEF(64)
C
      INTEGER DID,FOUR/4/,FPTR,ICNT1/0/,ICNT2/0/,ICNT3/0/,IER,
     *        I00/0/,I01/1/,I02/2/,IOUT,ISIZE,NC10/10/,NC23/23/,
     *        NCNT1/0/,NCNT2/0/,NDEC4/4/,NDEC6/6/,NR/0/,
     *        ONE/1/,TWO/2/,J
C
      EQUIVALENCE (C(1),CI1A(1)),(C(10),CI1B(1)),(C(15),CI2I(1)),
     *            (C(18),CI3Q(1)),(C(21),CF1(1)),(C(31),CF3(1)),
     *            (C(54),CI4A(1)),(C(59),CI4B(1)),(C(62),CI5(1))
C
      COMMON FPTR,F
C
 10   FORMAT (A)
 11   FORMAT (F10.6)
 12   FORMAT (I4)
C
      FNAM = 'HIFBATCH.DAT'
      OPEN (UNIT=4,FILE=FNAM,STATUS='OLD')
```

```
      READ (4,10) FNAM
C
      OPEN (UNIT=3,FILE=FNAM,STATUS='OLD')
C
 85   CONTINUE
C
      DO 86 J=1,9
            DI1IA(J) = 0.0D0
            DI1QA(J) = 0.0D0
 86   CONTINUE
      DO 87 J=1,5
            DI1IB(J) = 0.0D0
            DI1QB(J) = 0.0D0
            DI4A(J) = 0.0D0
 87   CONTINUE
      DO 88 J=1,3
            DI2I(J) = 0.0D0
            DI3Q(J) = 0.0D0
            DI4B(J) = 0.0D0
            DI5(J) = 0.0D0
 88   CONTINUE
      DO 89 J=1,10
            DF1(J) = 0.0D0
 89   CONTINUE
      DO 90 J=1,23
            DF3(J) = 0.0D0
 90   CONTINUE
      ICNT1 = 0
      ICNT2 = 0
      ICNT3 = 0
C
      READ (3,10,END=3000) INFNAM
      READ (3,10,END=3000) QUFNAM
      READ (3,10,END=3000) COFNAM
      READ (3,10,END=3000) FOFNAM
      READ (3,12,END=3000) IOUT
      READ (3,11,END=3000) DBGAIN
      READ (3,10,END=3000) OUTPUT
C
      CALL SGOPEN(IO0,'READ','NOPROMPT',INFNAM,'REAL',ISIZE0)
      CALL SGOPEN(IO1,'READ','NOPROMPT',QUFNAM,'REAL',ISIZE1)
      CALL SGOPEN(IO2,'READ','NOPROMPT',COFNAM,'REAL',ISIZE2)
C
      CALL SGTRAN(IO2,'READ','REAL',COEF,ISIZE2)
      DO 100 I1=1,ISIZE2
            C(I1) = DBLE(COEF(I1))
 100  CONTINUE
C
      CALL SGOPEN(IO2,'READ','NOPROMPT',FOFNAM,'REAL',ISIZE2)
C
      CALL SGTRAN(IO2,'READ','REAL',F,ISIZE2)
C
      GAIN = DBLE(10.0 ** (DBGAIN/20.0))
```

```
C
C
C       READ THE TWO INPUT FILES
C
C
        CALL SGTRAN(IO0,'READ','REAL',SCRATCH1,ISIZE0)
        CALL SGTRAN(IO1,'READ','REAL',SCRATCH2,ISIZE1)
C
        DO 500 NR=1,MIN0(ISIZE0,ISIZE1)
C
        DI1IA(1) = DBLE( SCRATCH1(NR) )
        DI1QA(1) = DBLE( SCRATCH2(NR) )
C
C       PERFORM 6 POLE BAND PASS FILTER 9.0-82.5 HZ.
C          REQUIRES DOUBLE PRECISION DI1IA(9),DI1IB(5),
C          DI1QA(9),DI1QB(5),CI1A(9),CI1B(5).
C          REQUIRES 3 FORMATS, SAME FOR BOTH STAGES.
C
C       DO INPHASE CHANNEL
C
C       SET FORMAT POINTER
        FPTR = 1
        DI1IB(1) = IIRFILT(DI1IA,CI1A,FOUR)
C       RESET FORMAT POINTER
        FPTR =1
        DI2I(1) = IIRFILT(DI1IB,CI1B,TWO)
C
C       DO QUADRATURE CHANNEL
C
C       RESET FORMAT POINTER
        FPTR = 1
        DI1QB(1) = IIRFILT(DI1QA,CI1A,FOUR)
C       RESET FORMAT POINTER
        FPTR = 1
        DI3Q(1) = IIRFILT(DI1QB,CI1B,TWO)
C
C       MULTIPLY IN GAIN
C          REQUIRES TWO FORMATS, ONE FOR EACH OPERAND
C
        DI2I(1) = MULT(GAIN,DI2I(1),ONE)
C       RESET FORMAT POINTER
        FPTR = FPTR - 2
        DI3Q(1) = MULT(GAIN,DI3Q(1),ONE)
C
C       DO ONE POLE LOW PASS FILTER IIR2  112.5 HZ
C          REQUIRES DI2I(3),CI2(3)
C          REQUIRES 3 FORMATS,SAME USED IN INPHASE
C          AND QUAD CHANNEL.
C
        I = IIRFILT(DI2I,CI2I,ONE)
C
C       DO ONE POLE HIGH PASS FILTER IIR3 112.5 HZ
C          REQUIRES DI3Q(3),CI3(3)
```

```
C          SAME FORMATS AS FOR INPHASE CHANNEL
C
C       RESET FORMAT POINTER
        FPTR = FPTR - 3
        Q = IIRFILT(DI3Q,CI3Q,ONE)
C
C       LIMIT INPHASE SIGNAL TO +-2.8
C
        IF (DABS(I).GT.2.8D0) I = DSIGN(2.8D0,I)
C
C       FIND PRODUCT OF INPHASE AND QUADRATURE SIGNAL
C       REQUIRES TWO FORMATS ONE FOR EACH OPERAND.
C
        DF1(1) = MULT(I,Q,ONE)
C
C       OUTPUT DESIRED VALUES
C
        IF (IOUT.GT.5) GO TO 310
        ICNT1 = ICNT1 + 1
        OUT(1) = DI2I(2)
        OUT(2) = DI3Q(2)
        OUT(3) = I
        OUT(4) = Q
        OUT(5) = DF1(1)
        OUTCOUNT = OUTCOUNT + 1
        SCRATCH3( OUTCOUNT ) = OUT( IOUT )
        GOTO 500
 310    CONTINUE
C
C       DECIMATE SIGNAL BY 6 WITH FIRA
C          REQUIRES DF1(10),CF1(10)
C          REQUIRES THREE FORMATS.
C
        CALL FIRFILT(DF1,CF1,NC10,NDEC6,NCNT1,DF3(1),DID)
        IF (DID.EQ.0) GO TO 500
C
C       OUTPUT THIS VALUE?
C
        IF (IOUT.NE.6) GO TO 320
        ICNT2 = ICNT2 + 1
        OUT(6) = DF3(1)
        OUTCOUNT = OUTCOUNT + 1
        SCRATCH3( OUTCOUNT ) = OUT( IOUT )
        GOTO 500
 320    CONTINUE
C
C       DECIMATE SIGNAL BY 4 WITH FIRB
C          REQUIRES DF3(23),CF3(23)
C          REQUIRES THREE FORMATS
C
        CALL FIRFILT(DF3,CF3,NC23,NDEC4,NCNT2,DI4A(1),DID)
        IF (DID.EQ.0) GO TO 500
C
```

```
C       PERFORM 3 POLE LOW PASS FILTER IIR4  3.3 HZ
C         REQUIRES DI4A(5),CI4A(5),DI4B(3),CI4B(3)
C         REQUIRES 3 FORMATS, SAME USED FOR A AND B
C         STAGES.
C
        DI4B(1) = IIRFILT(DI4A,CI4A,TWO)
C       RESET FORMAT POINTER
        FPTR = FPTR - 3
        DI5(1) = IIRFILT(DI4B,CI4B,ONE)
C
C       LIMIT SIGNAL TO +- 1.0
C
        IF (DABS(DI5(1)).GT.1.0D0) DI5(1) = DSIGN(1.0D0,DI5(1))
C
C       PERFORM ONE POLE LOW PASS FILTER IIR5 0.109 HZ
C          REQUIRES DI5(3),CI5(3)
C          REQUIRES THREE FORMATS.
C
        INTOUT = IIRFILT(DI5,CI5,ONE)
C
C       OUTPUT ANY VALUES?
C
        IF (IOUT.LT.7) GO TO 500
        ICNT3 = ICNT3 + 1
        OUT(7) = DI4A(2)
        OUT(8) = DI5(2)
        OUT(9) = INTOUT
        OUTCOUNT = OUTCOUNT + 1
        SCRATCH3( OUTCOUNT ) = OUT( IOUT )
C
 500    CONTINUE
        CALL SGOPEN(IO2,'WRITE','NOPROMPT',OUTPUT,'REAL',OUTCOUNT)
        CALL SGTRAN(IO2,'WRITE','REAL',SCRATCH3,OUTCOUNT)
        OUTCOUNT = 0
C
        GOTO 85
C
 3000   CONTINUE
        END
```

```
C*******************************************************************************
C
C       MIDB
C
C       VAX-11 FORTRAN SOURCE FILENAME:         MIDB.FOR
C
C       DEPARTMENT OF ELECTRICAL ENGINEERING    KANSAS STATE UNIVERSITY
C
C       REVISION        DATE                    PROGRAMMER
C       --------        ----                    ----------
C       00.0            NOV 14, 1982            DANIEL B. SCHOWENGERDT
C       01.0            NOV 15, 1982            DANIEL B. SCHOWENGERDT
C       02.0            MAY 26, 1983            JOHN W. BARTHOLOMEW
C
C*******************************************************************************
C
C
C       CALLING SEQUENCE
C
C               RUN MIDB
C
C       PURPOSE
C               The purpose of this program is to perform, using double
C               precision arithmetic, the medium frequency channel of
C               the LAMBAKINS algorithm.  Double precision arithmetic
C               is used so that comparisons can be made to other
C               versions of this program that use fixed point
C               arithmetic.
C
C       ROUTINE(S) CALLED BY THIS PROGRAM
C
C               FIRF - This subroutine performs FIR filtering.
C
C               IIRF - This function subroutine performs IIR filtering
C
C       DATA REQUIRED FROM USER
C
C               'MIBATCH.DAT' - Data file containing the name of the
C                  file containing the required inputs to the program.
C
C               IN PHASE DATA FILE
C
C               QUADRATURE DATA FILE
C
C               FILTER COEFFICIENT FILE - The order of the coefficients
C                 in this file are:
C                 1) FIR6 coefficients - 22 weights
C                 2) IIR6 coefficients - divided into two four pole
C                                        filters
C                   a) first four pole coefficients - 9 weights
C                   b) second four pole coefficients - 9 weights
C                 3) IIR7 coefficients - 3 weights
C                 4) IIR8 coefficients - 3 weights
```

```
C                5) FIRC coefficients - 16 weights
C                6) IIR9 coefficients - a 2 pole + a 1 pole filter
C                  a) the 2 pole coefficients - 5 weights
C                  b) the 1 pole coefficients - 3 weights
C                7) IIR10coefficients - 3 weights
C
C       INFORMATION RETURNED TO USER
C
C               The data generated in the program selected from
C               any of 10 different spots.
C
C*************************************************************************
C
        PARAMETER MAX_ARRAY_SIZE=200000
        REAL SCRATCH1(MAX_ARRAY_SIZE)
        REAL SCRATCH2(MAX_ARRAY_SIZE)
        REAL SCRATCH3(MAX_ARRAY_SIZE)
        INTEGER OUTCOUNT/0/
C
        CHARACTER*80 FNAM
        CHARACTER*80 INFNAM
        CHARACTER*80 QUFNAM
        CHARACTER*80 COFNAM
        CHARACTER*80 OUTPUT
C
        DOUBLE PRECISION DF5I(22)/22*0.0/,DF5Q(22)/22*0.0/,
     *                   DI6IA(9)/9*0.0/,DI6IB(9)/9*0.0/,
     *                   DI6QA(9)/9*0.0/,DI6QB(9)/9*0.0/,
     *                   DI7I(3)/3*0.0/,DI8Q(3)/3*0.0/,
     *                   DF7(16)/16*0.0/,
     *                   DI9A(5)/5*0.0/,DI9B(3)/3*0.0/,
     *                   DI10(3)/3*0.0/
C
        DOUBLE PRECISION CF5(22),CI6A(9),CI6B(9),CI7(3),
     *                   CI8(3),CF7(16),CI9A(5),CI9B(3),
     *                   CI10(3),C(73)
C
        DOUBLE PRECISION GAIN,I,Q,IIRF,INTOUT
C
        REAL DBGAIN,SIZE,OUT(13)
        REAL COEF(73)
C
        INTEGER IOUT,ICNT1/0/,ICNT2/0/,ICNT3/0/,NCNT5/0/,
     *          NCNT7/0/,NCNT2/0/,NR/0/,DID,IER,ISIZE,
     *          I00/0/,I01/1/,I02/2/,ONE/1/,FOUR/4/,
     *          NC22/22/,NC16/16/,NDEC6/6/,NDEC8/8/,
     *          NCNT5A/0/,TWO/2/,J
C
        EQUIVALENCE (C(1),CF5(1)),(C(23),CI6A(1)),(C(32),CI6B(1)),
     *              (C(41),CI7(1)),(C(44),CI8(1)),(C(47),CF7(1)),
     *              (C(63),CI9A(1)),(C(68),CI9B(1)),(C(71),CI10(1))
C
 10     FORMAT (A)
```

```
 11     FORMAT (F10.6)
 12     FORMAT (I4)
C
       FNAM = 'MIBATCH.DAT'
       OPEN (UNIT=4,FILE=FNAM,STATUS='OLD')
       READ (4,10) FNAM
C
       OPEN (UNIT=3,FILE=FNAM,STATUS='OLD')
C
 85    CONTINUE
C
       DO 86 J=1,22
              DF5I(J) = 0.0D0
              DF5Q(J) = 0.0D0
 86    CONTINUE
       DO 87 J=1,9
              DI6IA(J) = 0.0D0
              DI6QA(J) = 0.0D0
              DI6IB(J) = 0.0D0
              DI6QB(J) = 0.0D0
 87    CONTINUE
       DO 88 J=1,3
              DI7I(J) = 0.0D0
              DI8Q(J) = 0.0D0
              DI9B(J) = 0.0D0
              DI10(J) = 0.0D0
 88    CONTINUE
       DO 89 J=1,16
              DF7(J) = 0.0D0
 89    CONTINUE
       DO 90 J=1,5
              DI9A(J) = 0.0D0
 90    CONTINUE
       ICNT1 = 0
       ICNT2 = 0
       ICNT3 = 0
       NCNT5 = 0
       NCNT7 = 0
       NCNT2 = 0
       NCNT5A = 0
C
       READ (3,10,END=3000) INFNAM
       READ (3,10,END=3000) QUFNAM
       READ (3,10,END=3000) COFNAM
       READ (3,12,END=3000) IOUT
       READ (3,11,END=3000) DBGAIN
       READ (3,10,END=3000) OUTPUT
C
       CALL SGOPEN(IO0,'READ','NOPROMPT',INFNAM,'REAL',ISIZE0)
       CALL SGOPEN(IO1,'READ','NOPROMPT',QUFNAM,'REAL',ISIZE1)
       CALL SGOPEN(IO2,'READ','NOPROMPT',COFNAM,'REAL',ISIZE2)
C
       CALL SGTRAN(IO2,'READ','REAL',COEF,ISIZE2)
```

```
      DO 100 I1=1,ISIZE2
             C(I1) = DBLE(COEF(I1))
 100  CONTINUE
C
      GAIN = DBLE(10.0 ** (DBGAIN/20.0))
C
C
C     READ THE TWO INPUT FILES
C
C
      CALL SGTRAN(IO0,'READ','REAL',SCRATCH1,ISIZE0)
      CALL SGTRAN(IO1,'READ','REAL',SCRATCH2,ISIZE1)
C
      DO 500 NR=1,MINO(ISIZE0,ISIZE1)
C
      DF5I(1) = DBLE( SCRATCH1(NR) )
      DF5Q(1) = DBLE( SCRATCH2(NR) )
C
C     DECIMATE SIGNAL BY 6 WITH FIR FILTER FIRG
C        REQUIRES DOUBLE PRECISION DF5I(22),DF5Q(22),CF5(22)
C          THIS PROGRAM USES SAME THREE FOR EACH FILTER
C
C     DECIMATE BY 6 INPHASE CHANNEL
      CALL FIRF(DF5I,CF5,NC22,NDEC6,NCNT5,DI6IA(1),DID)
C
C     DECIMATE BY 6 QUADRATURE CHANNEL
      CALL FIRF(DF5Q,CF5,NC22,NDEC6,NCNT5A,DI6QA(1),DID)
C
C     IF NO OUTPUT FFOM FILTER GO GET NEXT DATAPOINT
C
      IF (DID.EQ.0) GO TO 500
C
C     DO 8 POLE BAND PASS IIR FILTER IIR6   (2.25-11 HZ)
C       REQUIRES DI6IA(9),DI6IB(9),CI6A(9),CI6B(9)
C                DI6QA(9),DI6QB(9)
C
      DI6IB(1) = IIRF(DI6IA,CI6A,FOUR)
      DI7I(1) = IIRF(DI6IB,CI6B,FOUR)
C
      DI6QB(1) = IIRF(DI6QA,CI6A,FOUR)
      DI8Q(1) = IIRF(DI6QB,CI6B,FOUR)
C
C     APPLY GAIN
C
      DI7I(1) = DI7I(1)*GAIN
      DI8Q(1) = DI8Q(1)*GAIN
C
C     DO 1 POLE FILTER IIR7 AND IIR8
C        REQUIRES DI7I(3),CI7(3)
C                 DI8Q(3),CI8(3)
C
      I = IIRF(DI7I,CI7,ONE)
      Q = IIRF(DI8Q,CI8,ONE)
```

```
C
C       LIMIT INPHASE SIGNAL +-2.8
C
        IF (DABS(I).GT.2.8D0) I = DSIGN(2.8D0,I)
C
C       FORM PRODUCT OF INPHASE AND QUAD CHANNEL
        DF7(1) = I*Q
C
C       OUTPUT ANY CALCULATED VALUES NOW?
        IF(IOUT.GT.7) GO TO 310
        ICNT1 = ICNT1 + 1
        OUT(1) = DI6IA(2)
        OUT(2) = DI6QA(2)
        OUT(3) = DI7I(2)
        OUT(4) = DI8Q(2)
        OUT(5) = I
        OUT(6) = Q
        OUT(7) = DF7(1)
        OUTCOUNT = OUTCOUNT + 1
        SCRATCH3( OUTCOUNT ) = OUT( IOUT )
        GO TO 500
 310    CONTINUE
C
C       DECIMATE BY 8 WITH FIR FILTER FIRC
C         REQUIRES DF7(16),CF7(16)
C
        CALL FIRF(DF7,CF7,NC16,NDEC8,NCNT7,DI9A(1),DID)
        IF (DID.EQ.0) GO TO 500
C
C       OUTPUT THIS RESULT?
C
        IF (IOUT.NE.8) GO TO 320
        ICNT2 = ICNT2 + 1
        OUTCOUNT = OUTCOUNT + 1
        SCRATCH3( OUTCOUNT ) = DI9A(1)
        GO TO 500
 320    CONTINUE
C
C       DO 3 POLE IIR FILTER IIR9
C         REQUIRES DI9A(5),CI9A(5)
C                  DI9B(3),CI9B(3)
C
        DI9B(1) = IIRF(DI9A,CI9A,TWO)
        DI10(1) = IIRF(DI9B,CI9B,ONE)
C
C       RESAMPLE /2
C
        NCNT2 = NCNT2 + 1
        IF(NCNT2.LT.2) GO TO 500
        NCNT2 = 0
C
C       LIMIT VALUE +- 1.0
C
```

```
      IF (DABS(DI10(1)).GT.1.0D0) DI10(1) = DSIGN(1.0D0,DI10(1))
C
C     DO 1 POLE IIR FILTER IIR10
C       REQUIRES DI10(3),CI10(3)
C
      INTOUT = IIRF(DI10,CI10,1)
C
C     OUTPUT ANY OF THESE CALCULATED VALUES?
      IF (IOUT.LT.9) GO TO 500
      ICNT3 = ICNT3 + 1
      OUT(9) = DI10(2)
      OUT(10) = INTOUT
      OUTCOUNT = OUTCOUNT + 1
      SCRATCH3( OUTCOUNT ) = OUT( IOUT )
C
 500  CONTINUE
      CALL SGOPEN(IO2,'WRITE','NOPROMPT',OUTPUT,'REAL',OUTCOUNT)
      CALL SGTRAN(IO2,'WRITE','REAL',SCRATCH3,OUTCOUNT)
      OUTCOUNT = 0
C
      GOTO 85
C
 3000 CONTINUE
      END
```

```
C*******************************************************************************
C
C       MIDBAND
C
C       VAX-11 FORTRAN SOURCE FILENAME:         MIDBAND.FOR
C
C       DEPARTMENT OF ELECTRICAL ENGINEERING    KANSAS STATE UNIVERSITY
C
C       REVISION        DATE                    PROGRAMMER
C       --------        ----                    ----------
C       00.0            NOV 14, 1982            DANIEL B. SCHOWENGERDT
C       01.0            MAY 27, 1983            JOHN W. BARTHOLOMEW
C
C*******************************************************************************
C
C       CALLING SEQUENCE
C
C               RUN MIDBAND
C
C       PURPOSE
C               The purpose of this program is to perform, using fixed
C               point arithmetic and data storage, the medium
C               frequency channel of the LAMBAKINS algorithm.  The
C               attempt in this program is to simulate the limitations
C               that would occur if this algorithm were executed by
C               a microprocessor.  The simulated limitations are in the
C               precision of data storage and math operations.
C
C       ROUTINE(S) CALLED BY THIS PROGRAM
C
C               IIRFILT A function subroutine that performs an IIR
C                       filter.
C
C               FIRFILT A subroutine that performs an FIR filter.
C
C               MULT    A function subroutine that performs a
C                       fixed point multiply.
C
C       DATA REQUIRED FROM USER
C
C               'MIFBATCH.DAT' - Data file containing the name of the
C                  file containing the required inputs to the program.
C
C               INPHASE DATA FILE
C
C               QUADRATURE DATA FILE
C
C               FILTER COEFFICIENT FILE - The order of the coefficients
C                 in this file are:
C                 1) FIR6 coefficients - 22 weights
C                 2) IIR6 coefficients - divided into two four pole
C                    filters.
C                   a) first four pole coefficients - 9 weights
```

```
C                   b) second four pole coefficients - 9 weights
C                 3) IIR7 coefficients - 3 weights
C                 4) IIR8 coefficients - 3 weights
C                 5) FIRC coefficients - 16 weights
C                 6) IIR9 coefficients - a 2 pole + 1 pole filter.
C                   a) the 2 pole coefficients - 5 weights
C                   b) the 1 pole coefficients - 3 weights
C                 7) IIR10 coefficients - 3 weights
C
C               DATA FORMAT FILE - Contains the fixed point format
C                 specifiers for all data storage and math operations
C                 in the program.  The order of formats is;
C                 a) FIRG formats.
C                   1) Coefficient format.
C                   2) Data format.
C                   3) Sum format.
C                 b) IIR6 formats.
C                   1) Coefficient format.
C                   2) Data format.
C                   3) Sum format.
C                 c) GAIN x Data (inphase and quad).
C                   1) format for GAIN value.
C                   2) Data format.
C                 d) IIR7 and IIR8 formats.
C                   1) Coefficient format.
C                   2) Data format.
C                   3) Sum format.
C                 e) Inphase x quad formats.
C                   1) Inphase data format.
C                   2) Quadrature data format.
C                 f) FIRC formats.
C                   1) Coefficient format.
C                   2) Data format.
C                   3) Sum format.
C                 g) IIR9 formats.
C                   1) Coefficient format.
C                   2) Data format.
C                   3) Sum format.
C                 h) IIR10 formats.
C                   1) Coefficient format.
C                   2) Data format.
C                   3) Sum format.
C
C       INFORMATION RETURNED TO USER
C
C               The data generated in the program seclected from
C               any of 10 different spots.
C
C*******************************************************************************
C
        PARAMETER MAX_ARRAY_SIZE=200000
        REAL SCRATCH1(MAX_ARRAY_SIZE)
        REAL SCRATCH2(MAX_ARRAY_SIZE)
```

```
      REAL SCRATCH3(MAX_ARRAY_SIZE)
      INTEGER OUTCOUNT/0/
C
      CHARACTER*80 FNAM
      CHARACTER*80 INFNAM
      CHARACTER*80 QUFNAM
      CHARACTER*80 COFNAM
      CHARACTER*80 FOFNAM
      CHARACTER*80 OUTPUT
C
      DOUBLE PRECISION DF5I(22)/22*0.0/,DF5Q(22)/22*0.0/,
     *                 DI6IA(9)/9*0.0/,DI6IB(9)/9*0.0/,
     *                 DI6QA(9)/9*0.0/,DI6QB(9)/9*0.0/,
     *                 DI7I(3)/3*0.0/,DI8Q(3)/3*0.0/,
     *                 DF7(16)/16*0.0/,
     *                 DI9A(5)/5*0.0/,DI9B(3)/3*0.0/,
     *                 DI10(3)/3*0.0/
C
      DOUBLE PRECISION CF5(22),CI6A(9),CI6B(9),CI7(3),
     *                 CI8(3),CF7(16),CI9A(5),CI9B(3),
     *                 CI10(3),C(73)
C
      DOUBLE PRECISION GAIN,I,Q,IIRFILT,MULT,INTOUT
C
      REAL DBGAIN,SIZE,OUT(13),F(50)/50*0.0/
      REAL COEF(73)
C
      INTEGER DID,FOUR/4/,FPTR,ICNT1/0/,ICNT2/0/,ICNT3/0/,IER,
     *        I00/0/,I01/1/,I02/2/,IOUT,ISIZE,NC16/16/,NC22/22/,
     *        NCNT2/0/,NCNT5/0/,NCNT5A/0/,NCNT7/0/,NDEC6/6/,
     *        NDEC8/8/,NR/0/,ONE/1/,TWO/2/,J
C
      EQUIVALENCE (C(1),CF5(1)),(C(23),CI6A(1)),(C(32),CI6B(1)),
     *            (C(41),CI7(1)),(C(44),CI8(1)),(C(47),CF7(1)),
     *            (C(63),CI9A(1)),(C(68),CI9B(1)),(C(71),CI10(1))
      COMMON FPTR,F
C
 10   FORMAT (A)
 11   FORMAT (F10.6)
 12   FORMAT (I4)
C
      FNAM = 'MIFBATCH.DAT'
      OPEN (UNIT=4,FILE=FNAM,STATUS='OLD')
      READ (4,10) FNAM
C
      OPEN (UNIT=3,FILE=FNAM,STATUS='OLD')
C
 85   CONTINUE
C
      DO 86 J=1,22
            DF5I(J) = 0.0D0
            DF5Q(J) = 0.0D0
 86   CONTINUE
```

```
      DO 87 J=1,9
              DI6IA(J) = 0.0D0
              DI6QA(J) = 0.0D0
              DI6IB(J) = 0.0D0
              DI6QB(J) = 0.0D0
 87   CONTINUE
      DO 88 J=1,3
              DI7I(J) = 0.0D0
              DI8Q(J) = 0.0D0
              DI9B(J) = 0.0D0
              DI10(J) = 0.0D0
 88   CONTINUE
      DO 89 J=1,16
              DF7(J) = 0.0D0
 89   CONTINUE
      DO 90 J=1,5
              DI9A(J) = 0.0D0
 90   CONTINUE
      ICNT1 = 0
      ICNT2 = 0
      ICNT3 = 0
      NCNT5 = 0
      NCNT7 = 0
      NCNT2 = 0
      NCNT5A = 0
C
      READ (3,10,END=3000) INFNAM
      READ (3,10,END=3000) QUFNAM
      READ (3,10,END=3000) COFNAM
      READ (3,10,END=3000) FOFNAM
      READ (3,12,END=3000) IOUT
      READ (3,11,END=3000) DBGAIN
      READ (3,10,END=3000) OUTPUT
C
      CALL SGOPEN(IO0,'READ','NOPROMPT',INFNAM,'REAL',ISIZE0)
      CALL SGOPEN(IO1,'READ','NOPROMPT',QUFNAM,'REAL',ISIZE1)
      CALL SGOPEN(IO2,'READ','NOPROMPT',COFNAM,'REAL',ISIZE2)
C
      CALL SGTRAN(IO2,'READ','REAL',COEF,ISIZE2)
      DO 100 I1=1,ISIZE2
              C(I1) = DBLE(COEF(I1))
 100  CONTINUE
C
      CALL SGOPEN(IO2,'READ','NOPROMPT',FOFNAM,'REAL',ISIZE2)
C
      CALL SGTRAN(IO2,'READ','REAL',F,ISIZE2)
C
      GAIN = DBLE(10.0 ** (DBGAIN/20.0))
C
C
C     READ THE TWO INPUT FILES
C
C
```

```
      CALL SGTRAN(IO0,'READ','REAL',SCRATCH1,ISIZE0)
      CALL SGTRAN(IO1,'READ','REAL',SCRATCH2,ISIZE1)
C
      DO 500 NR=1,MIN(ISIZE0,ISIZE1)
C
      DF5I(1) = DBLE( SCRATCH1(NR) )
      DF5Q(1) = DBLE( SCRATCH2(NR) )
C
C     DECIMATE SIGNAL BY 6 WITH FIR FILTER FIRG
C        REQUIRES DOUBLE PRECISION DF5I(22),DF5Q(22),CF5(22)
C        REQUIRES 3 FORMATS FOR EACH FILTER
C          THIS PROGRAM USES SAME THREE FOR EACH FILTER
      FPTR = 1
C
C     DECIMATE BY 6 INPHASE CHANNEL
      CALL FIRFILT(DF5I,CF5,NC22,NDEC6,NCNT5,DI6IA(1),DID)
C
C     RESET FORMAT POINTER TO USE SAME FORMATS
      IF(DID.NE.0) FPTR = FPTR - 3
C
C     DECIMATE BY 6 QUADRATURE CHANNEL
      CALL FIRFILT(DF5Q,CF5,NC22,NDEC6,NCNT5A,DI6QA(1),DID)
C
C     IF NO OUTPUT FFOM FILTER GO GET NEXT DATAPOINT
C
      IF (DID.EQ.0) GO TO 500
C
C     DO 8 POLE BAND PASS IIR FILTER IIR6   (2.25-11 HZ)
C       REQUIRES DI6IA(9),DI6IB(9),CI6A(9),CI6B(9)
C                DI6QA(9),DI6QB(9)
C        REQUIRES 3 FORMATS FOR EACH FILTER
C         WILL USE THE SAME 3 FORMATS FOR EACH FILTER
C
      DI6IB(1) = IIRFILT(DI6IA,CI6A,FOUR)
C     RESET FORMAT POINTER
      FPTR = FPTR - 3
      DI7I(1) = IIRFILT(DI6IB,CI6B,FOUR)
C
C     RESET FORMAT POINTER
      FPTR = FPTR - 3
      DI6QB(1) = IIRFILT(DI6QA,CI6A,FOUR)
C     RESET FORMAT POINTER
      FPTR = FPTR - 3
      DI8Q(1) = IIRFILT(DI6QB,CI6B,FOUR)
C
C     APPLY GAIN
C       REQUIRES TWO FORMATS, ONE FOR EACH OPERAND
C         THIS PROGRAM USES SAME FORMATS FOR EACH MULT.
C
      DI7I(1) = MULT(DI7I(1),GAIN,1)
C     RESET FORMAT POINTER
      FPTR = FPTR - 2
      DI8Q(1) = MULT(DI8Q(1),GAIN,ONE)
```

```
C
C       DO 1 POLE FILTER IIR7 AND IIR8
C          REQUIRES DI7I(3),CI7(3)
C                   DI8Q(3),CI8(3)
C          REQUIRES 3 FORMAT PER IIRFILT
C            THIS PROGRAM USES SAME FOR EACH
C
        I = IIRFILT(DI7I,CI7,ONE)
C       RESET FORMAT POINTER
        FPTR = FPTR - 3
        Q = IIRFILT(DI8Q,CI8,ONE)
C
C       LIMIT INPHASE SIGNAL +-2.8
C
        IF (DABS(I).GT.2.8D0) I = DSIGN(2.8D0,I)
C
C       FORM PRODUCT OF INPHASE AND QUAD CHANNEL
C         REQUIRES 2 FORMATS
        DF7(1) = MULT(I,Q,ONE)
C
C       OUTPUT ANY CALCULATED VALUES NOW?
        IF(IOUT.GT.7) GO TO 310
        ICNT1 = ICNT1 + 1
        OUT(1) = DI6IA(2)
        OUT(2) = DI6QA(2)
        OUT(3) = DI7I(2)
        OUT(4) = DI8Q(2)
        OUT(5) = I
        OUT(6) = Q
        OUT(7) = DF7(1)
        OUTCOUNT = OUTCOUNT + 1
        SCRATCH3( OUTCOUNT ) = OUT( IOUT )
        GO TO 500
 310    CONTINUE
C
C       DECIMATE BY 8 WITH FIR FILTER FIRC
C         REQUIRES DF7(16),CF7(16)
C         REQUIRES 3 FORMATS
C
        CALL FIRFILT(DF7,CF7,NC16,NDEC8,NCNT7,DI9A(1),DID)
        IF (DID.EQ.0) GO TO 500
C
C       OUTPUT THIS RESULT?
C
        IF (IOUT.NE.8) GO TO 320
        ICNT2 = ICNT2 + 1
        OUTCOUNT = OUTCOUNT + 1
        SCRATCH3( OUTCOUNT ) = DI9A(1)
        GO TO 500
 320    CONTINUE
C
C       DO 3 POLE IIR FILTER IIR9
C         REQUIRES DI9A(5),CI9A(5)
```

```
C                 DI9B(3),CI9B(3)
C          REQUIRES 3 FORMATS FOR EACH IIRFILT
C            THIS PROGRAM USES SAME FORMATS FOR EACH
C
        DI9B(1) = IIRFILT(DI9A,CI9A,TWO)
C       RESET FORMAT POINTER
        FPTR = FPTR - 3
        DI10(1) = IIRFILT(DI9B,CI9B,ONE)
C
C       RESAMPLE /2
C
        NCNT2 = NCNT2 + 1
        IF(NCNT2.LT.2) GO TO 500
        NCNT2 = 0
C
C       LIMIT VALUE +- 1.0
C
        IF (DABS(DI10(1)).GT.1.0D0) DI10(1) = DSIGN(1.0D0,DI10(1))
C
C       DO 1 POLE IIR FILTER IIR10
C         REQUIRES DI10(3),CI10(3)
C         REQUIRES 3 FORMATS
C
        INTOUT = IIRFILT(DI10,CI10,ONE)
C
C       OUTPUT ANY OF THESE CALCULATED VALUES?
        IF (IOUT.LT.9) GOTO 500
        ICNT3 = ICNT3 + 1
        OUT(9) = DI10(2)
        OUT(10) = INTOUT
        OUTCOUNT = OUTCOUNT + 1
        SCRATCH3( OUTCOUNT ) = OUT( IOUT )
C
 500    CONTINUE
        CALL SGOPEN(IO2,'WRITE','NOPROMPT',OUTPUT,'REAL',OUTCOUNT)
        CALL SGTRAN(IO2,'WRITE','REAL',SCRATCH3,OUTCOUNT)
        OUTCOUNT = 0
C
        GOTO 85
C
 3000   CONTINUE
        END
```

```
C*************************************************************************
C
C       LOB
C
C       VAX-11 FORTRAN SOURCE FILENAME:         LOB.FOR
C
C       DEPARTMENT OF ELECTRICAL ENGINEERING    KANSAS STATE UNIVERSITY
C
C       REVISION        DATE                    PROGRAMMER
C       --------        ----                    ----------
C       00.0            DEC 31, 1982            DANIEL B. SCHOWENGERDT
C       00.1            FEB 22, 1983            JOHN W. BARTHOLOMEW
C       01.0            MAY 19, 1983            MARCUS K. JUNOD
C       02.0            MAY 20, 1983            JOHN W. BARTHOLOMEW
C       02.1            MAY 24, 1983            JOHN W. BARTHOLOMEW
C
C*************************************************************************
C
C       CALLING SEQUENCE
C
C               RUN LOB
C
C       PURPOSE
C               The purpose of this program is to perform the low
C               frequency channel of the LAMBAKINS algorithm.
C
C       ROUTINE(S) CALLED BY THIS PROGRAM
C
C               IIRF    A function subroutine that performs an IIR
C                       filter.
C
C               FIRF    A subroutine that performs an FIR filter.
C
C               COMPR   A subroutine which performs a complex
C                       predictor filter.
C
C               SIGE    Function subroutine which performs a signal
C                       strength equalizer routine.
C
C       DATA REQUIRED FROM USER
C
C               'LOBATCH.DAT' - Data file containing the name of the
C                  file containing the required inputs to the program.
C
C               INPHASE DATA FILE
C
C               QUADRATURE DATA FILE
C
C               FILTER COEFFICIENT FILE - The order of the coefficients
C                 in this file are:
C                 1) FIR8 coefficients - 22 weights
C                 2) FIR4 coefficients - 16 weights.
C                 3) IIR11 coefficients - two four pole filters.
```

```
C                  a) First four pole - 9 weights.
C                  b) Second four pole - 9 weights.
C                4) IIR12 coefficients - 1 pole - 3 weights.
C                5) IIR13 coefficients - 1 pole - 3 weights.
C                6) FIRE coefficients - 9 weights.
C                7) IIR14 coefficients - 5 weights - 2 pole.
C                8) IIR15 coefficients - 3 weights - 1 pole.
C
C              DBGAIN  Variable gain in DB.
C
C       INFORMATION RETURNED TO USER
C
C              The data generated in the program selected from
C              any of 18 different spots.
C               1) Inphase FIRG output.
C               2) Quadrature FIRG output.
C               3) Inphase FIRD output.
C               4) Quadrature FIRD output.
C               5) Inphase IIR11 * GAIN output.
C               6) Quadrature IIR11 * GAIN output.
C               7) Inphase SSE output.
C               8) Quadrature SSE output.
C               9) Inphase complex pred. output.
C              10) Quadrature complex pred. output.
C              11) Inphase complex pred. error output.
C              12) Quadrature complex pred. error output.
C              13) Inphase limited IIR12 output.
C              14) Quadrature IIR13 output.
C              15) I x Q output.
C              16) FIRE output.
C              17) IIR14 output.
C              18) IIR15 output.
C
C*************************************************************************
C
      PARAMETER MAX_ARRAY_SIZE=200000
      REAL SCRATCH1(MAX_ARRAY_SIZE)
      REAL SCRATCH2(MAX_ARRAY_SIZE)
      REAL SCRATCH3(MAX_ARRAY_SIZE)
      INTEGER OUTCOUNT/0/
C
      CHARACTER*80 FNAM
      CHARACTER*80 INFNAM
      CHARACTER*80 QUFNAM
      CHARACTER*80 COFNAM
      CHARACTER*80 OUTPUT
      DOUBLE PRECISION DFGI(22)/22*0.0/,DFGQ(22)/22*0.0/,
     *                 DFDI(16)/16*0.0/,DFDQ(16)/16*0.0/,
     *                 DI11IA(9)/9*0.0/,DI11IB(9)/9*0.0/,
     *                 DI11QA(9)/9*0.0/,DI11QB(9)/9*0.0/,
     *                 DI12(3)/3*0.0/,DI13(3)/3*0.0/,
     *                 DFE(9)/9*0.0/,DI14(5)/5*0.0/,
     *                 DI15(3)
```

```
C
      DOUBLE PRECISION CFG(22),CFD(16),CI11A(9),CI11B(9),CI12(3),
     *                 CI13(3),CFE(9),CI14(5),CI15(3),C(79)
C
      DOUBLE PRECISION GAIN,I,Q,IIRF,SIGE,
     *         SSI,FIN,PR,PI,SSQ,TEMP,SSIO,SSQO,SQ,SI
C
      REAL DBGAIN,SIZE,OUT(18)
      REAL COEF(79)
C
      INTEGER DID,FOUR/4/,ICNT1/0/,ICNT2/0/,ICNT3/0/,ICNT4/0/,
     *         ICNT5/0/,IER,I00/0/,I01/1/,I02/2/,IOUT,ISIZE,NC9/9/,
     *         NC16/16/,NC22/22/,NCNTD/0/,NCNTDA/0/,NCNTE/0/,
     *         NCNTG/0/,NCNTGA/0/,NDEC4/4/,NDEC6/6/,NR/0/,ONE/1/,
     *         TWO/2/,J
C
      LOGICAL FIRSTCOMPR/.TRUE./,FIRSTSIGE/.TRUE./
C
      EQUIVALENCE (C(1),CFG(1)),(C(23),CFD(1)),(C(39),CI11A(1)),
     *            (C(48),CI11B(1)),(C(57),CI12(1)),(C(60),CI13(1)),
     *            (C(63),CFE(1)),(C(72),CI14(1)),(C(77),CI15(1))
C
 10   FORMAT (A)
 11   FORMAT (F10.6)
 12   FORMAT (I4)
C
      FNAM = 'LOBATCH.DAT'
      OPEN (UNIT=4,FILE=FNAM,STATUS='OLD')
      READ (4,10) FNAM
C
      OPEN (UNIT=3,FILE=FNAM,STATUS='OLD')
C
 85   CONTINUE
C
      DO 86 J=1,22
            DFGI(J) = 0.0D0
            DFGQ(J) = 0.0D0
 86   CONTINUE
      DO 87 J=1,16
            DFDI(J) = 0.0D0
            DFDQ(J) = 0.0D0
 87   CONTINUE
      DO 88 J=1,9
            DI11IA(J) = 0.0D0
            DI11IB(J) = 0.0D0
            DI11QA(J) = 0.0D0
            DI11QB(J) = 0.0D0
            DFE(J) = 0.0D0
 88   CONTINUE
      DO 89 J=1,3
            DI12(J) = 0.0D0
            DI13(J) = 0.0D0
 89   CONTINUE
```

```
      DO 90 J=1,5
            DI14(J) = 0.0D0
 90   CONTINUE
      ICNT1 = 0
      ICNT2 = 0
      ICNT3 = 0
      ICNT4 = 0
      ICNT5 = 0
C
      READ (3,10,END=3000) INFNAM
      READ (3,10,END=3000) QUFNAM
      READ (3,10,END=3000) COFNAM
      READ (3,12,END=3000) IOUT
      READ (3,11,END=3000) DBGAIN
      READ (3,10,END=3000) OUTPUT
C
      CALL SGOPEN(IO0,'READ','NOPROMPT',INFNAM,'REAL',ISIZE0)
      CALL SGOPEN(IO1,'READ','NOPROMPT',QUFNAM,'REAL',ISIZE1)
      CALL SGOPEN(IO2,'READ','NOPROMPT',COFNAM,'REAL',ISIZE2)
C
      CALL SGTRAN(IO2,'READ','REAL',COEF,ISIZE2)
      DO 100 I1=1,ISIZE2
         C(I1) = DBLE(COEF(I1))
 100  CONTINUE
C
      GAIN = DBLE(10.0 ** (DBGAIN/20.0))
C
C
C     READ THE TWO INPUT FILES
C
C
      CALL SGTRAN(IO0,'READ','REAL',SCRATCH1,ISIZE0)
      CALL SGTRAN(IO1,'READ','REAL',SCRATCH2,ISIZE1)
C
C
C
      DO 500 NR=1,MINO(ISIZE0,ISIZE1)
C
      DFGI(1) = DBLE( SCRATCH1(NR) )
      DFGQ(1) = DBLE( SCRATCH2(NR) )
C
C
C     DECIMATE SIGNAL BY 6 WITH FIR FILTER FIRG
C        REQUIRES DOUBLE PRECISION DFGI(22),DFGQ(22),CFG(22)
C
C     DECIMATE BY 6 INPHASE CHANNEL
      CALL FIRF(DFGI,CFG,NC22,NDEC6,NCNTG,DFDI(1),DID)
C
C
C     DECIMATE BY 6 QUADRATURE CHANNEL
      CALL FIRF(DFGQ,CFG,NC22,NDEC6,NCNTGA,DFDQ(1),DID)
C
C     IF NO OUTPUT FFOM FILTER GO GET NEXT DATAPOINT
```

```
C
      IF (DID.EQ.0) GOTO 500
C
C     OUTPUT ANY VALUE?
C
      IF (IOUT.GT.2) GO TO 310
      ICNT1 = ICNT1 + 1
      OUT(1) = DFDI(1)
      OUT(2) = DFDQ(1)
      OUTCOUNT = OUTCOUNT + 1
      SCRATCH3( OUTCOUNT ) = OUT( IOUT )
      GOTO 500
  310 CONTINUE
C
C     DECIMATE SIGNAL BY 4 WITH FIR FILTER FIRD.
C       REQUIRES DOUBLE PRECISION DFDI(16),DFDQ(16),CFD(16)
C
C     DECIMATE BY 4 THE INPHASE CHANNEL
      CALL FIRF(DFDI,CFD,NC16,NDEC4,NCNTD,DI11IA(1),DID)
C
C     DECIMATE BY 4 THE QUADRATURE CHANNEL
      CALL FIRF(DFDQ,CFD,NC16,NDEC4,NCNTDA,DI11QA(1),DID)
C
C     IF NO OUTPUT FROM FILTER GO GET NEXT DATA POINT.
C
      IF (DID.EQ.0) GOTO 500
C
C     DO 8 POLE BAND PASS IIR FILTER IIR11   (0.1 - 3 HZ)
C       REQUIRES DI11IA(9),DI11IB(9),CI11A(9),CI11B(9)
C                DI11QA(9),DI11QB(9)
C
      DI11IB(1) = IIRF(DI11IA,CI11A,FOUR)
      SSI = IIRF(DI11IB,CI11B,FOUR)
C
      DI11QB(1) = IIRF(DI11QA,CI11A,FOUR)
      SSQ = IIRF(DI11QB,CI11B,FOUR)
C
C     APPLY GAIN
C
      SSI = GAIN * SSI
      SSQ = GAIN * SSQ
C
C     SIGNAL STRENGTH EQUALIZATION SECTION
C       SMOOTHING PARAMETER: BETA = 0.95
C       DESIRED SIGNAL STRENGTH: D = 0.1
C       MINIMUN SIGNAL STRENGTH: SIMIN = SQMIN = 0.001
C
      SSIO = SIGE(SSI,ONE,FIRSTSIGE)
      SSQO = SIGE(SSQ,TWO,FIRSTSIGE)
C
C     COMPLEX PREDICTOR SECTION
C
      CALL COMPR(SSIO,SSQO,TWO,PR,PI,DI12(1),DI13(1),FIRSTCOMPR)
```

```
C
C       BOOST SIGNAL BACK UP TO MAX OF 1.0
C
        DI12(1) = DI12(1) / 0.15D0
        DI13(1) = DI13(1) / 0.15D0
C
C       DO 1 POLE IIR FILTERS IIR12,IIR13
C         REQUIRES ER,CI12(3), AND
C                  EI,CI13(3)
        I = IIRF(DI12,CI12,ONE)
        Q = IIRF(DI13,CI13,ONE)
C
C       LIMIT INPHASE SIGNAL
C
        IF(I.GT.2.8D0) I=DSIGN(2.8D0,I)
C
C       FIND PRODUCT OF INPHASE AND QUADRATURE SIGNAL
C
        DFE(1) = I * Q
C
C       OUTPUT ANY VALUES SO FAR ?
C
        IF(IOUT.GT.15) GO TO 320
        ICNT2 = ICNT2 + 1
        OUT(3) = DI11IA(2)
        OUT(4) = DI11QA(2)
        OUT(5) = SSI
        OUT(6) = SSQ
        OUT(7) = SSIO
        OUT(8) = SSQO
        OUT(9) = PR
        OUT(10) = PI
        OUT(11) = DI12(2)
        OUT(12) = DI13(2)
        OUT(13) = I
        OUT(14) = Q
        OUT(15) = DFE(1)
        OUTCOUNT = OUTCOUNT + 1
        SCRATCH3( OUTCOUNT ) = OUT( IOUT )
        GOTO 500
 320    CONTINUE
C
C       DECIMATE SIGNAL BY 4
C         REQUIRES DFE(9),CFE(9)
C
        CALL FIRF(DFE,CFE,NC9,NDEC4,NCNTE,DI14(1),DID)
C
C       IF NO OUTPUT FROM FILTER GO GET NEXT DATA POINT.
C
        IF (DID.EQ.0) GOTO 500
C
C       DO 2 POLE IIR FILTER IIR14  LP = 0.15 HZ
C         REQUIRES DI14(5),CI14(5)
```

```
C
      DI15(1) = IIRF(DI14,CI14,TWO)
C
C     OUTPUT ANY VALUES ?
C
      IF(IOUT.GT.17) GO TO 330
      ICNT3 = ICNT3 + 1
      OUT(16) = DI14(2)
      OUT(17) = DI15(1)
      OUTCOUNT = OUTCOUNT + 1
      SCRATCH3( OUTCOUNT ) = OUT( IOUT )
      GOTO 500
 330  CONTINUE
C
C     RESAMPLE SIGNAL BY 2
C
      ICNT4 = ICNT4 + 1
      IF(ICNT4.NE.2) GOTO 500
      ICNT4 = 0
C
C     DO 1 POLE FILTER IIR15
C       REQUIRES DI15(3),CI15(3)
C
      FIN = IIRF(DI15,CI15,ONE)
      OUT(18) = FIN
      ICNT5 = ICNT5 + 1
      OUTCOUNT = OUTCOUNT + 1
      SCRATCH3( OUTCOUNT ) = OUT( 18 )
 500  CONTINUE
C
      CALL SGOPEN(IO2,'WRITE','NOPROMPT',OUTPUT,'REAL',OUTCOUNT)
C
      CALL SGTRAN(IO2,'WRITE','REAL',SCRATCH3,OUTCOUNT)
C
      OUTCOUNT = 0
      FIRSTCOMPR = .TRUE.
      FIRSTSIGE = .TRUE.
C
      GOTO 85
C
 3000 CONTINUE
      END
```

```
C*******************************************************************************
C
C       LOBAND
C
C       VAX-11 FORTRAN SOURCE FILENAME:         LOBAND.FOR
C
C       DEPARTMENT OF ELECTRICAL ENGINEERING    KANSAS STATE UNIVERSITY
C
C       REVISION        DATE                    PROGRAMMER
C       --------        ----                    ----------
C       00.0            DEC 13, 1982            DANIEL B. SCHOWENGERDT
C       01.0            MAY 19, 1983            JOHN W. BARTHOLOMEW
C       02.0            MAY 20, 1983            JOHN W. BARTHOLOMEW
C       02.1            MAY 24, 1983            JOHN W. BARTHOLOMEW
C
C*******************************************************************************
C
C       CALLING SEQUENCE
C
C               RUN LOBAND
C
C       PURPOSE
C               The purpose of this program is to perform, using fixed
C               point arithmetic and data storage, the low
C               frequency channel of the LAMBAKINS algorithm.  The
C               attempt in this program is to simulate the limitations
C               that would occur if this algorithm were executed by
C               a microprocessor.  The simulated limitations are in the
C               precision of data storage and math operations.
C
C       ROUTINE(S) CALLED BY THIS PROGRAM
C
C               IIRFILT A function subroutine that performs an IIR
C                       filter.
C
C               FIRFILT A subroutine that performs an FIR filter.
C
C               MULT    A function subroutine that performs a
C                       fixed point multiply or divide.
C
C               ADD     A function subroutine that performs a
C                       fixed point add or subtract.
C
C               COMPRED A subroutine which performs a complex
C                       predictor filter using fixed point
C                       arithmetic.
C
C               SIGEQU  Function subroutine which performs a signal
C                       strength equalizer routine.
C
C       DATA REQUIRED FROM USER
C
C               'LOFBATCH.DAT' - Data file containing the name of the
```

```
C               file containing the required inputs to the program.
C
C             INPHASE DATA FILE
C
C             QUADRATURE DATA FILE
C
C             FILTER COEFFICIENT FILE - The order of the coefficients
C               in this file are:
C               1) FIRG coefficients - 22 weights
C               2) FIRD coefficients - 16 weights.
C               3) IIR11 coefficients - two four pole filters.
C                  a) First four pole - 9 weights.
C                  b) Second four pole - 9 weights.
C               4) IIR12 coefficients - 1 pole - 3 weights.
C               5) IIR13 coefficients - 1 pole - 3 weights.
C               6) FIRE coefficients - 9 weights.
C               7) IIR14 coefficients - 5 weights - 2 pole.
C               8) IIR15 coefficients - 3 weights - 1 pole.
C
C             DATA FORMAT FILE - Contains the fixed point format
C               specifiers for all data storage and math operations
C               in the program.  The order of formats is:
C               a) FIRG formats.
C                  1) Coefficient format.
C                  2) Data format.
C                  3) Sum format.
C               b) FIRD formats.
C                  1) Coefficient format.
C                  2) Data format.
C                  3) Sum format.
C               c) IIR11 formats.
C                  1) Coefficient format.
C                  2) Data format.
C                  3) Sum format.
C               d) GAIN x Data (inphase and quad).
C                  1) Format for GAIN value.
C                  2) Data format.
C               e) Signal Strength Equalizer,
C                    Sx(n) = BETA*Sx(n-1) + BETA*ABS(SSx)
C                    SSxO = SSx * D / (Sx(n) + SxMIN)
C                  1) Format for BETA, BETA1, D and
C                     (D / (Sx(n) + SxMIN)).
C                  2) Format for ABS(SSx), SX(n), (Sx(n) + SxMIN),
C                     and SSx.
C                  3) Format for both operands of any add.
C               f) Complex predictor.
C                  1) Format for real and imaginary part of weights,
C                     error and scalar value v.
C                  2) Format for real and imaginary part of input
C                     value and (e(n)*x(n-?)).
C                  3) Format for both operands of and add.
C               g) Formats for IIR12 and IIR13 filters.
C                  1) Coefficint format.
```

```
C                   2) Data format.
C                   3) Sum format.
C                 h) Inphase x quadrature data formats.
C                   1) Format for inphase data.
C                   2) Format for quadrature data.
C                 i) FIRE formats.
C                   1) Coefficient format.
C                   2) Data format.
C                   3) Sum format.
C                 j) IIR14 formats.
C                   1) Coefficient format.
C                   2) Data format.
C                   3) Sum format.
C                 k) IIR15 formats.
C                   1) Coefficient format.
C                   2) Data format.
C                   3) Sum format.
C
C       INFORMATION RETURNED TO USER
C
C               The data generated in the program selected from
C               any of 18 different spots.
C                1) Inphase FIR6 output.
C                2) Quadrature FIR6 output.
C                3) Inphase FIR0 output.
C                4) Quadrature FIR0 output.
C                5) Inphase IIR11 * GAIN output.
C                6) Quadrature IIR11 * GAIN output.
C                7) Inphase SSE output.
C                8) Quadrature SSE output.
C                9) Inphase complex pred. error output.
C               10) Quadrature complex pred. error output.
C               11) Inphase complex pred. output.
C               12) Quadrature complex pred. output.
C               13) Inphase limited IIR12 output.
C               14) Quadrature IIR13 output.
C               15) I x Q output.
C               16) FIRE output.
C               17) IIR14 output.
C               18) IIR15 output.
C
C*******************************************************************************
C
        PARAMETER MAX_ARRAY_SIZE=200000
        REAL SCRATCH1(MAX_ARRAY_SIZE)
        REAL SCRATCH2(MAX_ARRAY_SIZE)
        REAL SCRATCH3(MAX_ARRAY_SIZE)
        INTEGER OUTCOUNT/0/
C
        CHARACTER*80 FNAM
        CHARACTER*80 INFNAM
        CHARACTER*80 QUFNAM
        CHARACTER*80 COFNAM
```

```
      CHARACTER*80 FOFNAM
      CHARACTER*80 OUTPUT
C
      DOUBLE PRECISION DFGI(22)/22*0.0/,DFGQ(22)/22*0.0/,
     *                 DFDI(16)/16*0.0/,DFDQ(16)/16*0.0/,
     *                 DI11IA(9)/9*0.0/,DI11IB(9)/9*0.0/,
     *                 DI11QA(9)/9*0.0/,DI11QB(9)/9*0.0/,
     *                 DI12(3)/3*0.0/,DI13(3)/3*0.0/,
     *                 DFE(9)/9*0.0/,DI14(5)/5*0.0/,
     *                 DI15(3)
C
      DOUBLE PRECISION CFG(22),CFD(16),CI11A(9),CI11B(9),CI12(3),
     *                 CI13(3),CFE(9),CI14(5),CI15(3),C(79)
C
C     VARIABLES - DOUBLE PRECISION
      DOUBLE PRECISION FIN,PR,PI,GAIN,I,Q,SI,SQ,SSI,SSQ,SSIO,SSQO
C     FUNCTION SUBROUTINES - DOUBLE PRECISION
      DOUBLE PRECISION IIRFILT,MULT,ADD,SIGEQU
C
      REAL DBGAIN,SIZE,OUT(16),F(50)/50*0.0/
      REAL COEF(79)
C
      INTEGER DID,FOUR/4/,FPTR,ICNT1/0/,ICNT2/0/,ICNT3/0/,ICNT4/0/,
     *        ICNT5/0/,IER,IO0/0/,IO1/1/,IO2/2/,IOUT,ISIZE,NC9/9/,
     *        NC16/16/,NC22/22/,NCNTD/0/,NCNTDA/0/,NCNTE/0/,
     *        NCNTG/0/,NCNTGA/0/,NDEC4/4/,NDEC6/6/,NR/0/,ONE/1/,
     *        TWO/2/,J
C
      LOGICAL FIRSTCOMPR/.TRUE./,FIRSTSIGE/.TRUE./
C
      EQUIVALENCE (C(1),CFG(1)),(C(23),CFD(1)),(C(39),CI11A(1)),
     *            (C(48),CI11B(1)),(C(57),CI12(1)),(C(60),CI13(1)),
     *            (C(63),CFE(1)),(C(72),CI14(1)),(C(77),CI15(1))
C
      COMMON FPTR,F
C
 10   FORMAT (A)
 11   FORMAT (F10.6)
 12   FORMAT (I4)
C
      FNAM = 'LOFBATCH.DAT'
      OPEN (UNIT=4,FILE=FNAM,STATUS='OLD')
      READ (4,10) FNAM
C
      OPEN (UNIT=3,FILE=FNAM,STATUS='OLD')
C
 85   CONTINUE
C
      DO 86 J=1,22
            DFGI(J) = 0.0D0
            DFGQ(J) = 0.0D0
 86   CONTINUE
      DO 87 J=1,16
```

```
                DFDI(J) = 0.0D0
                DFDQ(J) = 0.0D0
 87     CONTINUE
        DO 88 J=1,9
                DI11IA(J) = 0.0D0
                DI11IB(J) = 0.0D0
                DI11QA(J) = 0.0D0
                DI11QB(J) = 0.0D0
                DFE(J) = 0.0D0
 88     CONTINUE
        DO 89 J=1,3
                DI12(J) = 0.0D0
                DI13(J) = 0.0D0
 89     CONTINUE
        DO 90 J=1,5
                DI14(J) = 0.0D0
 90     CONTINUE
        DO 91 J=1,50
                F(J) = 0.0
 91     CONTINUE
        ICNT1 = 0
        ICNT2 = 0
        ICNT3 = 0
        ICNT4 = 0
        ICNT5 = 0
C
        READ (3,10,END=3000) INFNAM
        READ (3,10,END=3000) QUFNAM
        READ (3,10,END=3000) COFNAM
        READ (3,10,END=3000) FOFNAM
        READ (3,12,END=3000) IOUT
        READ (3,11,END=3000) DBGAIN
        READ (3,10,END=3000) OUTPUT
C
        CALL SGOPEN(IO0,'READ','NOPROMPT',INFNAM,'REAL',ISIZE0)
        CALL SGOPEN(IO1,'READ','NOPROMPT',QUFNAM,'REAL',ISIZE1)
        CALL SGOPEN(IO2,'READ','NOPROMPT',COFNAM,'REAL',ISIZE2)
C
        CALL SGTRAN(IO2,'READ','REAL',COEF,ISIZE2)
        DO 100 I1=1,ISIZE2
                C(I1) = DBLE(COEF(I1))
 100    CONTINUE
C
        CALL SGOPEN(IO2,'READ','NOPROMPT',FOFNAM,'REAL',ISIZE2)
        CALL SGTRAN(IO2,'READ','REAL',F,ISIZE2)
C
        GAIN = DBLE(10.0 ** (DBGAIN/20.0))
C
C
C       READ THE TWO INPUT FILES
C
C
        CALL SGTRAN(IO0,'READ','REAL',SCRATCH1,ISIZE0)
```

```
      CALL SGTRAN(IO1,'READ','REAL',SCRATCH2,ISIZE1)
C
C
      DO 500 NR=1,MINO(ISIZE0,ISIZE1)
C
      DFGI(1) = DBLE( SCRATCH1(NR) )
      DFGQ(1) = DBLE( SCRATCH2(NR) )
C
C
C     DECIMATE SIGNAL BY 6 WITH FIR FILTER FIRG
C        REQUIRES DOUBLE PRECISION DFGI(22),DFGQ(22),CFG(22)
C        REQUIRES 3 FORMATS FOR EACH FILTER
C          THIS PROGRAM USES SAME THREE FOR EACH FILTER
      FPTR = 1
C
C     DECIMATE BY 6 INPHASE CHANNEL
      CALL FIRFILT(DFGI,CFG,NC22,NDEC6,NCNTG,DFDI(1),DID)
C
C     RESET FORMAT POINTER TO USE SAME FORMATS
      IF(DID.NE.0) FPTR = FPTR - 3
C
C     DECIMATE BY 6 QUADRATURE CHANNEL
      CALL FIRFILT(DFGQ,CFG,NC22,NDEC6,NCNTGA,DFDQ(1),DID)
C
C     IF NO OUTPUT FROM FILTER GO GET NEXT DATAPOINT
C
      IF (DID.EQ.0) GO TO 500
C
C     OUTPUT ANY VALUE?
C
      IF (IOUT.GT.2) GO TO 310
      ICNT1 = ICNT1 + 1
      OUT(1) = DFDI(1)
      OUT(2) = DFDQ(1)
      OUTCOUNT = OUTCOUNT + 1
      SCRATCH3( OUTCOUNT ) = OUT( IOUT )
      GOTO 500
 310  CONTINUE
C
C     DECIMATE SIGNAL BY 4 WITH FIR FILTER FIRD.
C       REQUIRES DOUBLE PRECISION DFDI(16),DFDQ(16),CFD(16)
C       REQUIRES 3 FORMATS FOR EACH FILTER.
C         SAME 3 FORMATS USED FOR EACH FILTER.
C
C     DECIMATE BY 4 THE INPHASE CHANNEL
      CALL FIRFILT(DFDI,CFD,NC16,NDEC4,NCNTD,DI11IA(1),DID)
C
C     RESET FORMAT POINTER TO USE SAME FORMATS
      IF(DID.NE.0) FPTR = FPTR - 3
C
C     DECIMATE BY 4 THE QUADRATURE CHANNEL
      CALL FIRFILT(DFDQ,CFD,NC16,NDEC4,NCNTDA,DI11QA(1),DID)
C
```

```
C       IF NO OUTPUT FROM FILTER GO GET NEXT DATA POINT.
C
        IF (DID.EQ.0) GO TO 500
C
C       DO 8 POLE BAND PASS IIR FILTER IIR11   (0.1 - 3 HZ)
C         REQUIRES DI11IA(9),DI11IB(9),CI11A(9),CI11B(9)
C                  DI11QA(9),DI11QB(9)
C          REQUIRES 3 FORMATS FOR EACH FILTER
C           SAME 3 FORMATS USED FOR EACH FILTER
C
        DI11IB(1) = IIRFILT(DI11IA,CI11A,FOUR)
C       RESET FORMAT POINTER
        FPTR = FPTR - 3
        SSI = IIRFILT(DI11IB,CI11B,FOUR)
C
C       RESET FORMAT POINTER
        FPTR = FPTR - 3
        DI11QB(1) = IIRFILT(DI11QA,CI11A,FOUR)
C       RESET FORMAT POINTER
        FPTR = FPTR - 3
        SSQ = IIRFILT(DI11QB,CI11B,FOUR)
C
C       APPLY GAIN
C         REQUIRES TWO FORMATS, ONE FOR EACH OPERAND
C           THIS PROGRAM USES SAME FORMATS FOR EACH MULT.
C
        SSI = MULT(GAIN,SSI,ONE)
C       RESET FORMAT POINTER
        FPTR = FPTR - 2
        SSQ = MULT(GAIN,SSQ,ONE)
C
C       SIGNAL STRENGTH EQUALIZATION SECTION
C         SMOOTHING PARAMETER: BETA = 0.95
C         DESIRED SIGNAL STRENGTH: D = 0.1
C         MINIMUM SIGNAL STRENGTH: SIMIN = SQMIN = 0.001
C
        SSIO = SIGEQU(SSI,ONE,FIRSTSIGE)
C       RESET FORMAT POINTER
        FPTR = FPTR - 2
        SSQO = SIGEQU(SSQ,TWO,FIRSTSIGE)
C       RESET FORMAT POINTER
        FPTR = FPTR + 1
C
C       COMPLEX PREDICTOR SECTION
C
        CALL COMPRED(SSIO,SSQO,TWO,PR,PI,DI12(1),DI13(1),FIRSTCOMPR)
C
C       BOOST VALUES BACK TO 1.0
C
        DI12(1) = DI12(1) / 0.15D0
        DI13(1) = DI13(1) / 0.15D0
C
C
```

```
C
C       DO 1 POLE IIR FILTERS IIR12,IIR13
C         REQUIRES DI12(3),CI12(3), AND
C                  DI13(3),CI13(3)
C         REQUIRES 3 FORMATS PER IIRFILT.
C           THIS PROGRAM USES SAME FOR EACH.
        I = IIRFILT(DI12,CI12,ONE)
C       RESET FORMAT POINTER
        FPTR = FPTR - 3
        Q = IIRFILT(DI13,CI13,ONE)
C
C       LIMIT INPHASE SIGNAL
C
        IF(I.GT.2.8D0) I=DSIGN(2.8D0,I)
C
C       FIND PRODUCT OF INPHASE AND QUADRATURE SIGNAL
C
        DFE(1) = MULT(I,Q,ONE)
C
C       OUTPUT ANY VALUES SO FAR ?
C
        IF(IOUT.GT.15) GO TO 320
        ICNT2 = ICNT2 + 1
        OUT(3) = DI11IA(2)
        OUT(4) = DI11QA(2)
        OUT(5) = SSI
        OUT(6) = SSQ
        OUT(7) = SSIO
        OUT(8) = SSQO
        OUT(9) = PR
        OUT(10) = PI
        OUT(11) = DI12(2)
        OUT(12) = DI13(2)
        OUT(13) = I
        OUT(14) = Q
        OUT(15) = DFE(1)
        OUTCOUNT = OUTCOUNT + 1
        SCRATCH3( OUTCOUNT ) = OUT( IOUT )
        GO TO 500
 320    CONTINUE
C
C       DECIMATE SIGNAL BY 4
C         REQUIRES DFE(9),CFE(9)
C         REQUIRES 3 FORMATS FOR EACH FILTER.
C
        CALL FIRFILT(DFE,CFE,NC9,NDEC4,NCNTE,DI14(1),DID)
C
C       IF NO OUTPUT FROM FILTER GO GET NEXT DATA POINT.
C
        IF (DID.EQ.0) GO TO 500
C
C       DO 2 POLE IIR FILTER IIR14  LP = 0.15 HZ
C         REQUIRES DI14(5),CI14(5)
```

```
C         REQUIRES 3 FORMATS.
C
        DI15(1) = IIRFILT(DI14,CI14,TWO)
C
C       OUTPUT ANY VALUES ?
C
        IF(IOUT.GT.17) GO TO 330
        ICNT3 = ICNT3 + 1
        OUT(16) = DI14(2)
        OUT(17) = DI15(1)
        OUTCOUNT = OUTCOUNT + 1
        SCRATCH3( OUTCOUNT ) = OUT( IOUT )
        GO TO 500
 330    CONTINUE
C
C       RESAMPLE SIGNAL BY 2
C
        ICNT4 = ICNT4 + 1
        IF(ICNT4.NE.2) GO TO 500
        ICNT4 = 0
C
C       DO 1 POLE FILTER IIR15
C         REQUIRES DI15(3),CI15(3)
C         REQUIRES 3 FORMATS.
C
        FIN = IIRFILT(DI15,CI15,ONE)
        OUT(18) = FIN
        ICNT5 = ICNT5 + 1
        OUTCOUNT = OUTCOUNT + 1
        SCRATCH3( OUTCOUNT ) = OUT( 18 )
 500    CONTINUE
C
        CALL SGOPEN(IO2,'WRITE','NOPROMPT',OUTPUT,'REAL',OUTCOUNT)
C
        CALL SGTRAN(IO2,'WRITE','REAL',SCRATCH3,OUTCOUNT)
C
        OUTCOUNT = 0
        FIRSTCOMPR = .TRUE.
        FIRSTSIGE = .TRUE.
C
        GOTO 85
C
 3000   CONTINUE
        END
```

```
C*****************************************************************************
C
C       FIRF
C
C       VAX-11 FORTRAN SOURCE FILENAME:         FIRF.FOR
C
C       DEPARTMENT OF ELECTRICAL ENGINEERING    KANSAS STATE UNIVERSITY
C
C       REVISION        DATE                    PROGRAMMER
C       --------        ----                    ----------
C       00.0            NOV 14, 1982            DANIEL B. SCHOWENGERDT
C       01.0            NOV 15, 1982            DANIEL B. SCHOWENGERDT
C       02.0            MAY 19, 1983            JOHN W. BARTHOLOMEW
C
C*****************************************************************************
C
C       CALLING SEQUENCE
C
C               CALL FIRF(DATA,COEF,NCOEF,NDECX,NCNTX,SUM,DID)
C
C       PURPOSE
C               The purpose of this subroutine is to execute a disital
C               FIR filter.  The maximum number of FIR coefficients
C               is 50.  The filter type (low pass, band pass, band stop,
C               high pass) is chosen by the filter coefficients in the
C               array COEF.  The generalized filter for an N coefficient
C               filter is:
C
C                FIRFILT = DATA(1)*COEF(1) + ... + DATA(N)*COEF(N).
C
C       ROUTINE(S) CALLED BY THIS PROGRAM
C
C               NONE
C
C       ARGUMENTS REQUIRED FROM CALLING ROUTINE
C
C               DATA    Array containing current and past input values.
C
C               COEF    Array of FIR coefficients for the filter.
C
C               NCOEF   Value equal to the number of FIR coefficients.
C
C               NDEC    Value equal to the desired decimation value.
C
C               NCNTX   Count of number of input values.
C
C
C
C       ARGUMENT(S) SUPPLIED TO THE CALLING ROUTINE
C
C               SUM     Contains current output of filter.
C
C               DID     If DID = 0 the subroutine stored current input
```

```
C                       data point but did not perform the filter.
C                       If DID not 0, the subroutine performed the
C                       filter.
C
C*************************************************************************
C
      SUBROUTINE FIRF(DATA,COEF,NCOEF,NDECX,NCNTX,SUM,DID)
C
      INTEGER NCOEF,NDECX,NCNTX,DID
      DOUBLE PRECISION DATA(50),COEF(50),SUM
C
C     INCREMENT THE INPUT DATA COUNT
C
      NCNTX = NCNTX + 1
      DID = 0
C
C     ENOUGH DATA POINTS TO DO FILTER?
C
      IF (MOD(NCNTX,NDECX).NE.0.0) GO TO 200
C
C     IF YES THEN DO FILTER BUT FIRST RESET COUNT.
C
      NCNTX = 0
C
      SUM = 0.0D0
      DO 100 I1=1,NCOEF
         SUM = SUM + COEF(I1)*DATA(I1)
 100  CONTINUE
C
C     RETURN DID = 1
C
      DID = 1
C
 200  CONTINUE
C
C     SHIFT DATA VALUES
C
      DO 300 I3=NCOEF,2,-1
         DATA(I3) = DATA(I3-1)
 300  CONTINUE
C
      RETURN
      END
```

```
C*************************************************************************
C
C       IIRF
C
C       VAX-11 FORTRAN SOURCE FILENAME:         IIRF.FOR
C
C       DEPARTMENT OF ELECTRICAL ENGINEERING    KANSAS STATE UNIVERSITY
C
C       REVISION        DATE                    PROGRAMMER
C       --------        ----                    ----------
C       00.0            ?                       SANDIA
C       01.0            NOV 08, 1982            DANIEL B. SCHOWENGERDT
C       02.0            NOV 12, 1982            DANIEL B. SCHOWENGERDT
C       03.0            MAY 19, 1983            JOHN W. BARTHOLOMEW
C
C*************************************************************************
C
C       CALLING SEQUENCE
C
C               IIRVALUE =  IIRF(FG,A,NPOLE)
C
C       PURPOSE
C               The purpose of this subroutine is to execute a disital
C               IIR filter.  The number of poles is variable from 1
C               to a 10 pole maximum.  The filter type (low pass,
C               band pass, band stop, high pass) is chosen by the
C               filter coefficients in the array 'A'.  The generalized
C               filter is:
C
C                         G(Z)   A(1) + A(2)*Z**-1 + ... + A(N+1)*Z**-N
C                  H(Z) = ---- = ---------------------------------------
C                         F(Z)     1 + A(4)*Z**-1 + ... A(2N+1)*Z**-N.
C
C               Solution of this equation for G(Z) yields:
C
C                    G(1) = A(1)*F(1) + A(2)*F(2) +
C                              A(3)*F(3) + ... + A(N+1)*F(N+1)  -
C                              A(N+2)*G(2) - ... - A(2N+1)*G(N+1)
C
C       ROUTINE(S) CALLED BY THIS PROGRAM
C
C               NONE
C
C
C       ARGUMENTS REQUIRED FROM CALLING ROUTINE
C
C               FG      Array of current and past input values as well
C                       as past computed filter values.
C
C               A       Array of 2*(# of poles)+1 filter coefficients.
C                       The order of the coefficients for an n pole
C                       filter is:
C                        1) Numerator coefficients;
```

```
C                          a) A(1)
C                          b) A(2) of A(2)*Z**-1
C                          c) A(3) of A(3)*Z**-2
C                          d) to A(n+1) of A(n+1)*Z**-n
C                       2) Denominator coefficients;
C                          a) B(1) of B(1)*Z**-1
C                          b) B(2) of B(2)*Z**-2
C                          c) to B(n) of B(n)*Z**-n
C                       3) Note that the very first denominator
C                          coefficient is assumed to be a 1; so
C                          it is not included in this list.
C
C               NPOLE   Number of poles in the IIR filter.
C
C
C       ARGUMENT(S) SUPPLIED TO THE CALLING ROUTINE
C
C               IIRFILT Contains current output of filter.
C
C*******************************************************************************
C
        DOUBLE PRECISION FUNCTION IIRF(FG,A,NPOLE,FIRST)
C
        INTEGER NPOLE,NCOEF
        DOUBLE PRECISION FG(21),A(21)
C
C
C       THE FOLLOWING INSTRUCTIONS EXECUTE THIS EQUATION:
C
C       G(1) = A(1)*F(1) + ... + A(*F(N+1)
C                - A(4)*G(2) - ... - A(2N+1)*G(N+1)
C
        NCOEF = 2*NPOLE + 1
C
C
        IIRF = 0.0D0
        DO 100  I1=1,NCOEF
           IIRF = IIRF + A(I1)*FG(I1)
 100    CONTINUE
C
C       SHIFT ARRAY VALUES (STORE NEGATIVE RESULT)
C
        FG(NPOLE+1) = 0.0D0 - IIRF
        DO 200 I2=NCOEF,2,-1
           FG(I2) = FG(I2-1)
 200    CONTINUE
C
        RETURN
        END
```

```
C*************************************************************************
C
C       SIGE
C
C       VAX-11 FORTRAN SOURCE FILENAME:         SIGE.FOR
C
C       DEPARTMENT OF ELECTRICAL ENGINEERING    KANSAS STATE UNIVERSITY
C
C       REVISION        DATE                    PROGRAMMER
C       --------        ----                    ----------
C       00.0            DEC 31, 1982            DANIEL B. SCHOWENGERDT
C       01.0            JAN  3, 1983            DANIEL B. SCHOWENGERDT
C       02.0            MAY 19, 1983            JOHN W. BARTHOLOMEW
C       02.1            MAY 24, 1983            JOHN W. BARTHOLOMEW
C
C*************************************************************************
C
C       CALLING SEQUENCE
C
C               VALUE = SIGE(SSI,ID,FIRST)
C
C       PURPOSE
C               This is a signal strength equalizer subroutine.  The
C               working equations are:
C                 i(n) = current input value
C                 s(n) = beta*s(n) + (1-beta)*ABS(i(n))
C                 i'(n) = i(n) * d / (s(n) + smin)
C                 i'(n) = current output value
C               Beta is the smoothing parameter, d is the desired
C               signal strength and smin is the minimum
C               signal strength.  The program is designed to do up
C               to 10 different signals each with different parameters.
C               Each signal and its associated parameters is
C               identified by a control value.
C
C       ROUTINE(S) CALLED BY THIS PROGRAM
C
C               NONE
C
C       ARGUMENTS REQUIRED FROM CALLING ROUTINE.
C
C               SSI     Input signal value.
C
C               ID      Integer value identifying smoothing
C                       parameter, desired signal strength and
C                       minimum signal strength for input signal.
C
C               FIRST   Flag that is true the first time that the
C                       routine is called, used to initialize certain
C                       variables.
C
C
C       ARGUMENT(S) SUPPLIED TO THE CALLING ROUTINE
```

```
C
C               SIGEQ   Result of signal strength equalizing
C                       input signal.
C
C               FIRST   This flag is always set to false upon returning.
C
C*************************************************************************
C
        DOUBLE PRECISION FUNCTION SIGE(SSI,ID,FIRST)
C
        DOUBLE PRECISION SSI,SI,
     *          BETA(10)/2*0.95,8*0.0/,
     *          BETA1(10)/2*0.05,8*0.0/,
     *          D(10)/2*0.1,8*0.0/,
C
C       *** NOTE SA IS INITIALIZED TO 1.0 ***
     *          SA(10)/10*1.0/,
     *          SMIN(10)/2*0.001,8*0.0/
C
        LOGICAL FIRST
        INTEGER ID,J
C
C       CHECK FOR THE FIRST TIME THROUGH
C
        IF (FIRST) THEN
                DO 55 J=3,10
                        BETA(J) = 0.0D0
                        BETA1(J) = 0.0D0
                        D(J) = 0.0D0
                        SA(J) = 1.0D0
                        SMIN(J) = 0.0D0
 55             CONTINUE
                BETA(1) = 0.95D0
                BETA(2) = 0.95D0
                BETA1(1) = 0.05D0
                BETA1(2) = 0.05D0
                D(1) = 0.1D0
                D(2) = 0.1D0
                SA(1) = 1.0D0
                SA(2) = 1.0D0
                SMIN(1) = 0.001D0
                SMIN(2) = 0.001D0
        ENDIF
C
C       CHECK FOR CORRECT RANGE OF ID
        IF(ID.GE.1.AND.ID.LE.10) GO TO 100
        TYPE *,' ****** ',ID,' ILLEGAL ID FOR SIGE *******'
        STOP
 100    CONTINUE
C
C       DO SA = BETA * SA + (1-BETA) * DABS(SI)
C
        SI = SSI
```

```
      SA(ID) = BETA(ID)*SA(ID) + BETA1(ID)*DABS(SI)
C
C     DO SIGE = SSI * D / (SA + SMIN)
C
      SIGE = SI * (D(ID) / (SA(ID) + SMIN(ID)))
C
      FIRST = .FALSE.
C
      RETURN
      END
```

```
C*****************************************************************************
C
C       COMPR
C
C       VAX-11 FORTRAN SOURCE FILENAME:         COMPR.FOR
C
C       DEPARTMENT OF ELECTRICAL ENGINEERING    KANSAS STATE UNIVERSITY
C
C       REVISION        DATE                    PROGRAMMER
C       --------        ----                    ----------
C       00.0            DEC 31, 1982            DANIEL B. SCHOWENGERDT
C       01.0            MAY 19, 1983            JOHN W. BARTHOLOMEW
C       01.1            MAY 24, 1983            JOHN W. BARTHOLOMEW
C
C*****************************************************************************
C
C       CALLING SEQUENCE
C
C               CALL COMPR(RIN,IIN,NW,XR,XI,ER,EI,FIRST)
C
C       PURPOSE
C               The purpose of this subroutine is to execute a complex
C               predictor filter.  The number of weights of the filter
C               can range from 1 to 10.  The equations for this
C               complex predictor are:
C                1) x(n) = i(n) + J q(n)
C                2) x'(n) = W1(n)*x(n-1) + W2(n)*x(n-2) +
C                   ... + WN(n)*x(n-N)
C                3) e(n) = x(n) - x'(n)
C                4) W1(n+1) = W1(n) + v*e(n)*(conjugate of x(n-1))
C                   W2(n+1) = W2(n) + v*e(n)*(conjugate of x(n-2))
C                   and so on.
C                5) Output is
C                  a) i'(n) = Re[x'(n)]
C                  b) q'(n) = Im[x'(n)]
C
C       ROUTINE(S) CALLED BY THIS PROGRAM
C
C               NONE
C
C       ARGUMENTS REQUIRED FROM CALLING ROUTINE
C
C               RIN     Real part of the input value.
C
C               IIN     Imaginary part of the input value.
C
C               NW      Integer number of weights to be used in the
C                       filter.
C
C               FIRST   Flag that is true the first time that the
C                       routine is called, used to initialize certain
C                       variables.
C
```

```
C       ARGUMENT(S) RETURNED TO THE CALLING ROUTINE
C
C               XR      Real part of complex predicted value
C
C               XI      Imaginary part of complex predicted value.
C
C               ER      Real part of complex error between input value
C                       and predicted value.
C
C               EI      Imaginary part of complex error between input
C                       value and predicted value.
C
C               FIRST   This flag is always set to false upon returning.
C
C*************************************************************************
C
        SUBROUTINE COMPR(RIN,IIN,NW,XR,XI,ER,EI,FIRST)
C
        DOUBLE PRECISION RIN,IIN,XR,XI,ER,EI,
     *                   WR(10)/10*0.0/,WI(10)/10*0.0/,
     *                   R(11)/11*0.0/,I(11)/11*0.0/
C
        LOGICAL FIRST
        INTEGER J
C
C       DESCRIPTION OF DATA STORAGE ARRAYS
C          R(10) STORES REAL PART OF INPUT VALUE.
C          I(10) STORES IMAGINARY PART OF INPUT VALUE.
C          WR(11) STORES REAL PART OF WEIGHTS.
C          WI(11) STORES IMAGINARY PART OF WEIGHTS.
C          XR STORES REAL PART OF PREDICTED VALUE.
C          XI STORES IMAGINARY PART OF PREDICTED VALUE.
C          ER STORES REAL PART OF ERROR.
C          EI STORES IMAGINARY PART OF ERROR.
C
C
C       *** CONVERGENCE PARAMETER SET HERE! ***
C
        DOUBLE PRECISION V/0.05/
C
        INTEGER NW,ZERO/0/,ONE/1/
C
C
C       CHECK FOR FIRST TIME THROUGH
C
        IF (FIRST) THEN
                DO 55 J=1,10
                        WR(J) = 0.0D0
                        WI(J) = 0.0D0
                        R(J) = 0.0D0
                        I(J) = 0.0D0
 55             CONTINUE
                R(11) = 0.0D0
```

```
                I(11) = 0.0D0
        ENDIF
C
        R(1) = RIN
        I(1) = IIN
C
C       GENERATE THE PREDICTED VALUE X'
C         EQUATION IS
C           X'(N) = W1(N)*X(N-1) + W2(N)*X(N-2)
C         XR IS Re[X']
C         XI IS Im[X']
C
        XI = 0.0D0
        XR = 0.0D0
C
        DO 100 I1=1,NW
           XR = XR + (WR(I1)*R(I1+1)) - (WI(I1)*I(I1+1))
           XI = XI + (WR(I1)*I(I1+1)) + (WI(I1)*R(I1+1))
 100    CONTINUE
C
C       GENERATE THE ERROR
C         E(N) = X(N) - X'(N)
C         ER = Re[ E ]
C         EI = Im[ E ]
C
        ER = R(1) - XR
        EI = I(1) - XI
C
C       UPDATE WEIGHTS
C         W1(N+1) = W1(N) + v*E(N)*X}(N-1)
C         W2(N+1) = W2(N) + v*E(N)*X}(N-2)
C         AND X} IS COMPLEX CONJUGATE OF X
C
        DO 200 I2=1,NW
           WR(I2) = WR(I2) + (V * ((ER*R(I2+1)) + (EI*I(I2+1))))
           WI(I2) = WI(I2) + (V * ((EI*R(I2+1)) - (ER*I(I2+1))))
 200    CONTINUE
C
C       PUSH DOWN PAST INPUT VALUES
C
        DO 300 I3 = NW,1,-1
           R(I3+1) = R(I3)
           I(I3+1) = I(I3)
 300    CONTINUE
C
        FIRST = .FALSE.
C
        RETURN
        END
```

```
C*****************************************************************************
C
C       FIRFILT
C
C       VAX-11 FORTRAN SOURCE FILENAME:         FIRFILT.FOR
C
C       DEPARTMENT OF ELECTRICAL ENGINEERING    KANSAS STATE UNIVERSITY
C
C       REVISION        DATE                    PROGRAMMER
C       --------        ----                    ----------
C       00.0            NOV 14, 1982            DANIEL B. SCHOWENGERDT
C       01.0            MAY 19, 1983            JOHN W. BARTHOLOMEW
C
C*****************************************************************************
C
C       CALLING SEQUENCE
C
C               CALL FIRFILT(DATA,COEF,NCOEF,NDECX,NCNTX,SUM,DID)
C
C       PURPOSE
C               The purpose of this subroutine is to execute a digital
C               FIR filter in a fixed point mode that will simulate its
C               execution on a microprocessor.  The maximum number of
C               FIR coefficients is 50.  The filter type (low pass,
C               band pass, band stop, high pass) is chosen by the
C               filter coefficients in the array COEF.  The generalized
C               filter for an N coefficient filter is:
C
C                FIRFILT = DATA(1)*COEF(1) + ... + DATA(N)*COEF(N).
C
C       ROUTINE(S) CALLED BY THIS PROGRAM
C
C               ADD     Add or subtract two fixed point formated
C                       numbers.
C
C               MULT    Multiply two fixed point formated numbers.
C
C
C       ARGUMENTS REQUIRED FROM CALLING ROUTINE
C
C               DATA    Array containing current and past input values.
C
C               COEF    Array of FIR coefficients for the filter.
C
C               NCOEF   Value equal to the number of FIR coefficients.
C
C               NDECX   Value equal to the desired decimation value.
C
C               NCNTX   Count of number of input values.
C
C
C       DATA SHARED IN COMMON BLOCK WITH MAIN PROGRAM.
C
```

```
C               FPTR    Pointer to fixed point format specifiers
C                       for each value used in the filter.
C                       In this filter each multiply requires two fixed
C                       point formats, one for each operand.  Each add
C                       requires one format.  For example,
C                       the formats required for the following two
C                       weight filter:
C                        SUM = COEF(1)*DATA(1) + COEF(2)*DATA(2)
C                       are:
C                        1) Format for COEF(1) in COEF(1)*DATA(1)
C                        2) Format for DATA(1) in COEF(1)*DATA(1)
C                        3) Format for COEF(2) in COEF(2)*DATA(2)
C                        4) Format for DATA(2) in COEF(2)*DATA(2)
C                        5) Format for the addition COEF(1)*DATA(1) +
C                           COEF(2)*F(2).
C
C               ***************  NOTICE ***********************
C
C                       In this program the same fixed point formats
C                       will be used for each multiply and each add.
C                       When this program is called, FPTR must be
C                       pointing to the first of only three formats
C                       which it will repeatedly use.
C                       These formats are, in order of use:
C                        1) Format for filter coeff. in multiply.
C                        2) Format for data value in multiply.
C                        3) Format for addition of products.
C
C       ARGUMENT(S) SUPPLIED TO THE CALLING ROUTINE
C
C               SUM     Contains current output of filter.
C
C               DID     If DID = 0 the subroutine stored current input
C                       data point but did not perform the filter.
C                       If DID not 0, the subroutine performed the
C                       filter.
C
C*********************************************************************
C
      SUBROUTINE FIRFILT(DATA,COEF,NCOEF,NDECX,NCNTX,SUM,DID)
C
      INTEGER NCOEF,NDECX,NCNTX,DID,FPTR
      DOUBLE PRECISION ADD,MULT
      DOUBLE PRECISION DATA(50),COEF(50),SUM,TEMP
      COMMON FPTR
C
C     INCREMENT THE INPUT DATA COUNT
C
      NCNTX = NCNTX + 1
      DID = 0
C
C     ENOUGH DATA POINTS TO DO FILTER?
C
```

```
      IF (NCNTX.LT.NDECX) GO TO 200
C
C     IF YES THEN DO FILTER BUT FIRST RESET COUNT.
C
      NCNTX = 0
C
      SUM = 0.0D0
      DO 100 I1=1,NCOEF
         TEMP = MULT(COEF(I1),DATA(I1),1)
         SUM = ADD(SUM,TEMP,1)
C        RESET FORMAT POINTER
         FPTR = FPTR - 3
  100 CONTINUE
C     SET FORMAT POINTER TO NEXT SET
      FPTR = FPTR + 3
C
C     RETURN DID = 1
      DID = 1
C
  200 CONTINUE
C
C     SHIFT DATA VALUES
C
      DO 300 I3=NCOEF,2,-1
         DATA(I3) = DATA(I3-1)
  300 CONTINUE
C
      RETURN
      END
```

```
C*************************************************************************
C
C       IIRFILT
C
C       VAX-11 FORTRAN SOURCE FILENAME:          IIRFILT.FOR
C
C       DEPARTMENT OF ELECTRICAL ENGINEERING     KANSAS STATE UNIVERSITY
C
C       REVISION        DATE                     PROGRAMMER
C       --------        ----                     ----------
C       00.0            ?                        SANDIA
C       01.0            NOV 08, 1982             DANIEL B. SCHOWENGERDT
C       02.0            NOV 12, 1982             DANIEL B. SCHOWENGERDT
C       03.0            MAY 19, 1983             JOHN W. BARTHOLOMEW
C
C*************************************************************************
C
C       CALLING SEQUENCE
C
C               IIRVALUE =  IIRFILT(FG,A,NPOLE)
C
C       PURPOSE
C               The purpose of this subroutine is to execute a digital
C               IIR filter in a fixed point mode that will simulate its
C               execution on a microprocessor.  The number of poles is
C               variable from 1 to a 10 pole maximum.  The filter type
C               (low pass, band pass, band stop, high pass) is chosen
C               by the filter coefficients in the array 'A'.  The
C               generalized filter is:
C
C                                 G(Z)   A(1) + A(2)*Z**-1 + ... + A(N+1)*Z**-N
C                          H(Z) = ---- = ---------------------------------------
C                                 F(Z)     1 + A(4)*Z**-1 + ... A(2N+1)*Z**-N.
C
C               Solution of this equation for G(Z) yields:
C
C                    G(1) = A(1)*F(1) + A(2)*F(2) +
C                              A(3)*F(3) + ... + A(N+1)*F(N+1)  -
C                              A(N+2)*G(2) - ... - A(2N+1)*G(N+1)
C
C       ROUTINE(S) CALLED BY THIS PROGRAM
C
C               ADD     Add or subtract two fixed point formated
C                       numbers.
C
C               MULT    Multiply two fixed point formated numbers.
C
C
C       ARGUMENTS REQUIRED FROM CALLING ROUTINE
C
C               FG      Array of current and past input values as well
C                       as past computed filter values.
C
```

```
C               A       Array of 2*(# of poles)+1 filter coefficients.
C                       The order of the coefficients for an n pole
C                       filter is:
C                        1) Numerator coefficients;
C                          a) A(1)
C                          b) A(2) of A(2)*Z**-1
C                          c) A(3) of A(3)*Z**-2
C                          d) to A(n+1) of A(n+1)*Z**-n.
C                        2) Denominator coefficients;
C                          a) B(1) of B(1)*Z**-1
C                          b) B(2) of B(2)*Z**-1
C                          c) to B(n) of B(n)*Z**-n.
C                        3) Note that the very first denominator
C                           coefficient is assumed to be 1.0, so
C                           it is not included in this list.
C
C               NPOLE   Number of poles in the IIR filter.
C
C       DATA SHARED IN COMMON BLOCK WITH MAIN PROGRAM.
C
C               FPTR    Pointer to fixed point format specifiers
C                       for each value used in the filter.
C                       In this filter each multiply requires two fixed
C                       point formats, one for each operand.  Each add
C                       or subtract requires one format.  For example,
C                       the formats required for the following two
C                       pole filter:
C                       G(1) = A(1)*F(1) + A(2)*F(2) + A(3)*F(3) -
C                              A(4)*G(2) - A(5)*G(3)
C                       are:
C                        1) Format for A(1) in A(1)*F(1) multiply.
C                        2) Format for F(1) in A(1)*F(1) multiply.
C                        3) Format for A(2) in A(2)*F(2) multiply.
C                        4) Format for F(2) in A(2)*F(2) multiply.
C                        5) Format for the addition
C                              A(1)*F(1) + A(2)*F(2).
C                        6) Format for A(3) in A(3)*F(3) multiply.
C                        7) Format for F(3) in A(3)*F(3) multiply.
C                        8) Format for addition (SUM) + A(3)*F(3).
C                        9) Format for A(4) in A(4)*G(2) multiply.
C                        10) Format for G(2) in A(4)*G(2) multiply.
C                        11) Format for subtraction (SUM) - A(4)*G(2).
C                        12) Format for A(5) in A(5)*G(3) multiply.
C                        13) Format for G(3) in A(5)*G(3) multiply.
C                        14) Format for subtraction (SUM) - A(5)*G(3).
C
C               ************** NOTICE **********************
C
C                       In this program the same fixed point formats
C                       will be used for each multiply and each add.
C                       When this program is called, FPTR must be
C                       pointing to the first of only three formats
C                       which it will repeatedly use.
```

```
C                       These formats are, in order of use:
C                        1) Format for filter coeff. in multiply.
C                        2) Format for data value in multiply.
C                        3) Format for addition of products.
C
C       ARGUMENT(S) SUPPLIED TO THE CALLING ROUTINE
C
C               IIRFILT Contains current output of filter.
C
C*************************************************************************
C
        DOUBLE PRECISION FUNCTION IIRFILT(FG,A,NPOLE)
C
        INTEGER NPOLE,FPTR,NCOEF
        DOUBLE PRECISION ADD,MULT
        DOUBLE PRECISION FG(21),A(21),TEMP
        COMMON FPTR
C
C
C       THE FOLLOWING INSTRUCTIONS EXECUTE THIS EQUATION:
C
C       G(1) = A(1)*F(1) + ... + A(*F(N+1)
C              - A(4)*G(2) - ... - A(2N+1)*G(N+1)
C
C       IN FIXED POINT FORMAT.
C
        NCOEF = 2*NPOLE + 1
C
        IIRFILT = MULT(A(1),FG(1),1)
C       RESET FORMAT POINTER
        FPTR = FPTR + 1
C
        DO 100  I1=2,NCOEF
C          RESET FORMAT POINTER
           FPTR = FPTR - 3
           TEMP = MULT(A(I1),FG(I1),1)
           IIRFILT = ADD(IIRFILT,TEMP,1)
 100    CONTINUE
C
C       SHIFT ARRAY VALUES (STORE NEGATIVE RESULT)
C
        FG(NPOLE+1) = 0.0D0 - IIRFILT
        DO 200 I2=NCOEF,2,-1
           FG(I2) = FG(I2-1)
 200    CONTINUE
C
        RETURN
        END
```

```
C*************************************************************************
C
C       SIGEQU
C
C       VAX-11 FORTRAN SOURCE FILENAME:         SIGEQU.FOR
C
C       DEPARTMENT OF ELECTRICAL ENGINEERING    KANSAS STATE UNIVERSITY
C
C       REVISION        DATE                    PROGRAMMER
C       --------        ----                    ----------
C       00.0            DEC 30, 1982            DANIEL B. SCHOWENGERDT
C       01.0            MAY 19, 1983            JOHN W. BARTHOLOMEW
C       01.1            MAY 24, 1983            JOHN W. BARTHOLOMEW
C
C*************************************************************************
C
C       CALLING SEQUENCE
C
C               VALUE = SIGEQU(SSI,ID,FIRST)
C
C       PURPOSE
C               This is a signal strength equalizer program that
C               works using fixed point arithmetic.  The
C               working equations are:
C                 i(n) = current input value
C                 s(n) = beta*s(n) + (1-beta)*ABS(i(n))
C                 i'(n) = i(n) * d / (s(n) + smin)
C                 i'(n) = current output value
C               Beta is the smoothing parameter, d is the desired
C               signal strength and smin is the minimum
C               signal strength.  The program is designed to do up
C               to 10 different signals each with different parameters.
C               Each signal and its associated parameters is
C               identified by a control value.
C
C       ROUTINE(S) CALLED BY THIS PROGRAM
C
C               ADD     A function subroutine that performs a
C                       fixed point add or subtract.
C
C               MULT    A function subroutine that performs a
C                       fixed point multiply or divide.
C
C       ARGUMENTS REQUIRED FROM CALLING ROUTINE.
C
C               SSI     Input signal value.
C
C               ID      Integer value identifying smoothing
C                       parameter, desired signal strength and
C                       minimum signal strength for input signal.
C
C               FIRST   Flag that is true the first time that the
C                       routine is called, used to initialize
```

```
C                       certain variables.
C
C       DATA SHARED IN COMMON BLOCK WITH MAIN PROGRAM
C
C               FPTR    Pointer to correct format specifier for
C                       operands used in fixed point
C                       math operations in this subroutine.
C                       Three format specifiers are used.  They are:
C                        1) Format for BETA, BETA1, D, and (D/(SA+SMIN).
C                        2) FORMAT for ABS(SI), SA, (SA+SMIN), and SI.
C                        3) Format for both operands in every add.
C
C       ARGUMENT(S) SUPPLIED TO THE CALLING ROUTINE
C
C               SIGEQU  Result of signal strength equalizing
C                       input signal.
C
C               FIRST   This flag is always set to false upon returning.
C
C*************************************************************************
C
        DOUBLE PRECISION FUNCTION SIGEQU(SSI,ID,FIRST)
C
        DOUBLE PRECISION SSI,SI,TEMP,ADD,MULT,
     *          BETA(10)/2*0.95,8*0.0/,
     *          BETA1(10)/2*0.05,8*0.0/,
     *          D(10)/2*0.1,8*0.0/,
     *          SA(10)/10*1.0/,
     *          SMIN(10)/2*0.001,8*0.0/
C
        LOGICAL FIRST
        INTEGER ID,FPTR,ZERO/0/,ONE/1/
        COMMON FPTR
C
C       CHECK FOR FIRST TIME THROUGH
C
        IF (FIRST) THEN
                DO 55 J=3,10
                        BETA(J) = 0.0D0
                        BETA1(J) = 0.0D0
                        D(J) = 0.0D0
                        SA(J) = 0.0D0
                        SMIN(J) = 0.0D0
 55             CONTINUE
                BETA(1) = 0.95D0
                BETA(2) = 0.95D0
                BETA1(1) = 0.05D0
                BETA1(2) = 0.05D0
                D(1) = 0.1D0
                D(2) = 0.1D0
                SA(1) = 1.0D0
                SA(2) = 1.0D0
                SMIN(1) = 0.001D0
```

```
                SMIN(2) = 0.001D0
        ENDIF
C
C       CHECK FOR CORRECT RANGE OF ID
        IF(ID.GE.1.AND.ID.LE.10) GO TO 100
        TYPE *,' ****** ',ID,' ILLEGAL ID FOR SIGEQU *******'
        STOP
 100    CONTINUE
C
C       DO SA = BETA * SA + (1-BETA) * ABS(SI)
C
        SI = SSI
        TEMP = MULT(BETA1(ID),DABS(SI),ONE)
C       SET FORMAT POINTER
        FPTR = FPTR - 2
        SA(ID) = MULT(BETA(ID),SA(ID),ONE)
        SA(ID) = ADD(SA(ID),TEMP,ONE)
C       SET FORMAT POINTER
        FPTR = FPTR - 1
C
C       DO SIGEQU = SSI * D / (SA + SMIN)
C
        TEMP = ADD(SA(ID),SMIN(ID),ONE)
C       SET FORMAT POINTER
        FPTR = FPTR - 3
        TEMP = MULT(D(ID),TEMP,ZERO)
C       SET FORMAT POINTER
        FPTR = FPTR - 2
        SIGEQU = MULT(TEMP,SI,ONE)
C
        FIRST = .FALSE.
C
        RETURN
        END
```

```
C*******************************************************************************
C
C       COMPRED
C
C       VAX-11 FORTRAN SOURCE FILENAME:         COMPRED.FOR
C
C       DEPARTMENT OF ELECTRICAL ENGINEERING    KANSAS STATE UNIVERSITY
C
C       REVISION        DATE                    PROGRAMMER
C       --------        ----                    ----------
C       00.0            DEC 15, 1982            DANIEL B. SCHOWENGERDT
C       01.0            MAY 19, 1983            JOHN W. BARTHOLOMEW
C       01.1            MAY 24, 1983            JOHN W. BARTHOLOMEW
C
C*******************************************************************************
C
C       CALLING SEQUENCE
C
C               CALL COMPRED(RIN,IIN,NW,XR,XI,ER,EI,FIRST)
C
C       PURPOSE
C               The purpose of this subroutine is to execute a complex
C               predictor filter in a fixed point mode that will
C               simulate its execution on a microprocessor.  The number
C               of weights of the filter can range from 1 to 10.
C               The equations for this predictor are:
C                1) x(n) = i(n) + J q(n)
C                2) x'(n) = W1(n)*x(n-1) + W2(n)*x(n-2)
C                   + ... + WN(n)*x(n-N)
C                3) e(n) = x(n) - x'(n)
C                4) W1(n+1) = W1(n) + v*e(n)*(conjugate of x(n-1))
C                   W2(n+1) = W2(n) + v*e(n)*(conjugate of x(n-2))
C                   and so on.
C                5) Output is
C                  a) i'(n) = Re[x'(n)]
C                  b) q'(n) = Im[x'(n)]
C
C       ROUTINE(S) CALLED BY THIS PROGRAM
C
C               ADD     This subroutine adds or subtracts two numbers
C                       which it has just fix point formatted.
C
C               MULT    This subroutine multiplies or divides two
C                       numbers which it has just fix point formatted.
C
C       ARGUMENTS REQUIRED FROM CALLING ROUTINE
C
C               RIN     Real part of the input value.
C
C               IIN     Imaginary part of the input value.
C
C               NW      Integer number of weights to be used in the
C                       filter.
```

```
C
C               FIRST   Flag that is true the first time that the
C                       routine is called, used to initialize certain
C                       variables.
C
C       ARGUMENT(S) RETURNED TO THE CALLING ROUTINE
C
C               XR      Real part of complex predicted value
C
C               XI      Imaginary part of complex predicted value.
C
C               ER      Real part of complex error between input value
C                       and predicted value.
C
C               EI      Imaginary part of complex error between input
C                       value and predicted value.
C
C               FIRST   This flag is always set to false upon returning.
C
C       DATA SHARED IN COMMON WITH MAIN PORGRAM.
C
C               FPTR    Pointer to fixed point format specifiers
C                       for each value used in the filter.  In this
C                       filter each multiply requires two fixed
C                       point formats, one for each operand.  Each
C                       add or subtract requires one format.
C
C               ********************NOTICE ********************
C
C                       In this program all first operands in all
C                       multiplies done in this program will use the
C                       same format specifier.  All second operands in
C                       all multiplies done in this program will use
C                       the same format specifier which may be
C                       different than the one for the first operand.
C                       Also, both operands in all adds or subtracts
C                       done in this program will use the same format
C                       specifier.
C
C*************************************************************************
C
      SUBROUTINE COMPRED(RIN,IIN,NW,XR,XI,ER,EI,FIRST)
C
      DOUBLE PRECISION RIN,IIN,XR,XI,ER,EI,MULT,ADD,
     *                 WR(10)/10*0.0/,WI(10)/10*0.0/,WRT,WIT,
     *                 R(11)/11*0.0/,I(11)/11*0.0/,TEMP
C
      LOGICAL FIRST
      INTEGER J
C
C     DESCRIPTION OF DATA STORAGE ARRAYS
C       R(10) STORES REAL PART OF INPUT VALUE.
C       I(10) STORES IMAGINARY PART OF INPUT VALUE.
```

```
C          WR(11) STORES REAL PART OF WEIGHTS.
C          WI(11) STORES IMAGINARY PART OF WEIGHTS.
C          XR STORES REAL PART OF PREDICTED VALUE.
C          XI STORES IMAGINARY PART OF PREDICTED VALUE.
C          ER STORES REAL PART OF ERROR.
C          EI STORES IMAGINARY PART OF ERROR.
C          WRT,WIT ,TEMP ARE TEMPORARY STORAGE.
C
C
C       **** CONVERGENCE PARAMETER SET HERE! ****
C
        DOUBLE PRECISION V/0.05/
C
        INTEGER NW,ZERO/0/,ONE/1/,FPTR
C
        COMMON FPTR
C
C
C       CHECK FOR FIRST TIME THROUGH
C
        IF (FIRST) THEN
                DO 55 J=1,10
                        WR(J) = 0.0D0
                        WI(J) = 0.0D0
                        R(J) = 0.0D0
                        I(J) = 0.0D0
 55             CONTINUE
                R(11) = 0.0D0
                I(11) = 0.0D0
        ENDIF
C
        R(1) = RIN
        I(1) = IIN
C
C       GENERATE THE PREDICTED VALUE X'
C         EQUATION IS
C           X'(N) = W1(N)*X(N-1) + W2(N)*X(N-2)
C         XR IS Re[X']
C         XI IS Im[X']
C
        XI = 0.0D0
        XR = 0.0D0
C       INITIALIZE FORMAT POINTER
        FPTR = FPTR + 3
C
        DO 100 I1=1,NW
C          RESET FORMAT POINTER
           FPTR = FPTR - 3
C       ESTIMATE REAL VALUE XR
           TEMP = MULT(WR(I1),R(I1+1),ONE)
           XR = ADD(XR,TEMP,ONE)
C          RESET FORMAT POINTER
           FPTR = FPTR - 3
```

```
            TEMP = MULT(WI(I1),I(I1+1),ONE)
            XR = ADD(XR,TEMP,ZERO)
C           RESET FORMAT POINTER
            FPTR = FPTR - 3
C         ESTIMATE IMAGINARY VALUE XI
            TEMP = MULT(WR(I1),I(I1+1),ONE)
            XI = ADD(XI,TEMP,ONE)
C           RESET FORMAT POINTER
            FPTR = FPTR - 3
            TEMP = MULT(WI(I1),R(I1+1),ONE)
            XI = ADD(XI,TEMP,ONE)
 100      CONTINUE
C
C         GENERATE THE ERROR
C           E(N) = X(N) - X'(N)
C           ER = Re[ E ]
C           EI = Im[ E ]
C
C         RESET FORMAT POINTER
          FPTR = FPTR - 1
          ER = ADD(R(1),XR,ZERO)
C         RESET FORMAT POINTER
          FPTR = FPTR - 1
          EI = ADD(I(1),XI,ZERO)
C
C         UPDATE WEIGHTS
C           W1(N+1) = W1(N) + v*E(N)*X}(N-1)
C           W2(N+1) = W2(N) + v*E(N)*X}(N-2)
C           AND X} IS COMPLEX CONJUGATE OF X
C
          DO 200 I2=1,NW
C           RESET FORMAT POINTER
            FPTR = FPTR - 3
            WRT = MULT(ER,R(I2+1),ONE)
C           RESET FORMAT POINTER
            FPTR = FPTR - 2
            TEMP = MULT(EI,I(I2+1),ONE)
            WRT = ADD(WRT,TEMP,ONE)
C           RESET FORMAT POINTER
            FPTR = FPTR - 3
            WIT = MULT(EI,R(I2+1),ONE)
C           RESET FORMAT POINTER
            FPTR = FPTR - 2
            TEMP = MULT(ER,I(I2+1),ONE)
            WIT = ADD(WIT,TEMP,ZERO)
C           RESET FORMAT POINTER
            FPTR = FPTR - 3
            WRT = MULT(V,WRT,ONE)
            WR(I2) = ADD(WR(I2),WRT,ONE)
C           RESET FORMAT POINTER
            FPTR = FPTR - 3
            WIT = MULT(V,WIT,ONE)
            WI(I2) = ADD(WI(I2),WIT,ONE)
```

```
 200    CONTINUE
C
C       PUSH DOWN PAST INPUT VALUES
C
        DO 300 I3 = NW,1,-1
          R(I3+1) = R(I3)
          I(I3+1) = I(I3)
 300    CONTINUE
C
        FIRST = .FALSE.
C
        RETURN
        END
```

```
C*************************************************************************
C
C       MULT
C
C       VAX-11 FORTRAN SOURCE FILENAME:         MULT.FOR
C
C       DEPARTMENT OF ELECTRICAL ENGINEERING    KANSAS STATE UNIVERSITY
C
C       REVISION        DATE                    PROGRAMMER
C       --------        ----                    ----------
C       00.0            OCT 24, 1982            DANIEL B. SCHOWENGERDT
C       01.0            NOV 11, 1982            DANIEL B. SCHOWENGERDT
C       02.0            MAY 19, 1983            JOHN W. BARTHOLOMEW
C
C*************************************************************************
C
C       CALLING SEQUENCE
C
C               PRODUCT = MULT(M1,M2,OPT)
C
C       PURPOSE
C               This program performs a fixed point multiply or
C               divide of two double precision numbers.  It does
C               this by first formatting each of the two numbers to
C               the desired number of bits using the FXPT
C               subroutine and then multiplying or dividing.
C
C       ROUTINE(S) CALLED BY THIS PROGRAM
C
C               FXPT    Fixed point formatting subroutine.
C
C       ARGUMENTS REQUIRED FROM CALLING ROUTINE
C
C               M1      One of the two operands to be multiplied.
C
C               M2      The other of the two operands to be multiplied.
C
C               OPT     If OPT = 0 then divide (M1 / M2)
C                       If OPT not 0 then multiply (M1 * M2)
C
C       ARGUMENT(S) SUPPLIED TO THE CALLING ROUTINE
C
C               MULT    Double precision product or dividend of the
C                       formated M1 and M2.
C
C*************************************************************************
C
        DOUBLE PRECISION FUNCTION MULT(M1,M2,OPT)
C
        DOUBLE PRECISION M1,M1A,M1FX,M2,M2A,M2FX,FXPT
        INTEGER OPT
C
C       FORMAT THE TWO OPERANDS
```

```
C
      M1A = M1
      M2A = M2
      M1FX = FXPT(M1A)
      M2FX = FXPT(M2A)
C
C     MULTIPLY THE FORMATTED OPERANDS.
C
      IF (OPT.EQ.0) MULT = M1FX / M2FX
      IF (OPT.NE.0) MULT = M1FX * M2FX
C
      RETURN
      END
```

```
C*******************************************************************************
C
C       ADD
C
C       VAX-11 FORTRAN SOURCE FILENAME:         ADD.FOR
C
C       DEPARTMENT OF ELECTRICAL ENGINEERING    KANSAS STATE UNIVERSITY
C
C       REVISION        DATE                    PROGRAMMER
C       --------        ----                    ----------
C       00.0            OCT 24, 1982            DANIEL B. SCHOWENGERDT
C       01.0            NOV 11, 1982            DANIEL B. SCHOWENGERDT
C       02.0            MAY 19, 1983            JOHN W. BARTHOLOMEW
C
C*******************************************************************************
C
C       CALLING SEQUENCE
C
C               SUM = ADD(A1,A2,OPT)
C
C       PURPOSE
C               This program performs a fixed point add or subtract
C               of two double precision numbers.  It does this by
C               first formatting each of the two numbers to the
C               desired number of bits using the FXPT subroutine
C               and then adding or subtracting them.
C
C       ROUTINE(S) CALLED BY THIS PROGRAM
C
C               FXPT    Fixed point formatting subroutine.
C
C       ARGUMENTS REQUIRED FROM CALLING ROUTINE
C
C               A1      One of the two operands to be added.
C
C               A2      The other of the two operands to be added.
C
C               OPT     If OPT = 0 then subtract (A1 - A2)
C                       If OPT not 0 then add (A1 + A2)
C
C       DATA SHARED IN COMMON BLOCK WITH MAIN PROGRAM
C
C               FPTR    Pointer to correct format specifier for
C                       each of the operands of the add/subtract.
C                       Since the same format is used for each
C                       operand after it is used for the first
C                       operand it is reset to point to the
C                       same format for the second operand.
C
C       ARGUMENT(S) SUPPLIED TO THE CALLING ROUTINE
C
C               ADD     Result of (A1 + A2) or (A1 - A2).
C
```

```
C
C*******************************************************************************
C
      DOUBLE PRECISION FUNCTION ADD(A1,A2,OPT)
C
      DOUBLE PRECISION A1,A1A,A1FX,A2,A2A,A2FX,FXPT
      INTEGER OPT,FPTR
      COMMON FPTR
C
C     FORMAT THE TWO OPERANDS
C
      A1A = A1
      A1FX = FXPT(A1A)
C
C     RESET FPTR TO POINT TO THE SAME
C     FORMAT AS FOR THE FIRST OPERAND
C
      FPTR = FPTR - 1
      A2A = A2
      A2FX = FXPT(A2A)
C
C     ADD THE FORMATTED OPERANDS TOGETHER
C
      IF (OPT.EQ.0) ADD = A1FX - A2FX
      IF (OPT.NE.0) ADD = A1FX + A2FX
C
      RETURN
      END
```

```
C*************************************************************************
C
C       FXPT
C
C       VAX-11 FORTRAN SOURCE FILENAME:         FXPT.FOR
C
C       DEPARTMENT OF ELECTRICAL ENGINEERING    KANSAS STATE UNIVERSITY
C
C       REVISION        DATE                    PROGRAMMER
C       --------        ----                    ----------
C       00.0            JUL 28, 1982            DANIEL B. SCHOWENGERDT
C       01.0            OCT 24, 1982            DANIEL B. SCHOWENGERDT
C       02.0            NOV 10, 1982            DANIEL B. SCHOWENGERDT
C       03.0            JAN 21, 1983            JOHN W. BARTHOLOMEW
C       04.0            MAY 19, 1983            JOHN W. BARTHOLOMEW
C
C*************************************************************************
C
C       CALLING SEQUENCE
C
C               FORMATTEDD =  FXPT(D)
C
C       PURPOSE
C               The purpose of this function subroutine is to
C               limit the number of bits that represent a certain
C               number.  For example 16 bit data can be truncated
C               or rounded to 8 bit data.  The number is also
C               limited to some desired magnitude.  Double precision
C               data is used so that the maximum number of
C               bits is 56.
C
C       ROUTINE(S) CALLED BY THIS PROGRAM
C
C               None
C
C       ARGUMENTS REQUIRED FROM CALLING ROUTINE
C
C               D       Double precision data to be modified.
C
C       DATA SHARED IN COMMON BLOCK WITH MAIN PROGRAM
C
C               F       Array containing format specifiers for
C                       each time FXPT is called.  Each element in
C                       F is coded as a five digit number ABCDE.
C                       Digits AB yield the total number of bits.
C                       Digits CD yield the number of fraction bits
C                       and if D = 0 then round off data, if D = 1
C                       then truncate data.
C
C               FPTR    Pointer to correct format specifiers in
C                       the array F.
C
C       ARGUMENT(S) SUPPLIED TO THE CALLING ROUTINE
```

```
C
C               FXPT    Modified D.
C
C*************************************************************************
C
C       NOTICE :
C               By conditionally compiling the lines with a D in the
C               first column, all overflows, underflows, and zero
C               errors will be printed.  A normal compilation treats
C               these lines as comments.
C
C*************************************************************************
C
        DOUBLE PRECISION FUNCTION FXPT(D)
C
        REAL SHFT,MAX,MIN,F(50)
        INTEGER FPTR,TBIT,FBIT,TRUN
        COMMON FPTR,F
        DOUBLE PRECISION D
C
        TBIT = AINT(F(FPTR)/1000.0)
        FBIT = AINT(F(FPTR)/10 - 100.0*TBIT)
        TRUN = F(FPTR) - 10.0*AINNT(F(FPTR)/10.0)
C
        SHFT = 2.0**FBIT
C
C       MAKE ALL FRACTION DATA BITS INTEGER BITS
C
        FXPT = D * SHFT
C
C       ROUND OFF OR TRUNCATE REMAINING FRACTION BITS
C
        IF (TRUN.EQ.0) FXPT = DNINT(FXPT)
        IF (TRUN.EQ.1) FXPT = DINT(FXPT)
C
C       RESTORE THE FRACTION BITS
C
        FXPT = FXPT / SHFT
C
C       CHECK FOR TOO LARGE A VALUE
C
        IBIT = TBIT - FBIT - 1
        MAX = 2.0**IBIT
        MIN = 0.0 - MAX
        IF (FXPT.LT.MAX) GO TO 10
D          TYPE *,' OVERFLOW! ',FXPT,' FORMAT CODE = ',FPTR
           FXPT = MAX - (1/2**FBIT)
C
 10     CONTINUE
        IF (FXPT.GE.MIN) GO TO 20
D          TYPE *,' UNDERFLOW! ',FXPT,' FORMAT CODE = ',FPTR
           FXPT = MIN
C
```

```
 20     CONTINUE
        IF (.NOT.((FXPT.EQ.0.0).AND.(D.NE.0.0))) GO TO 30
D          TYPE *,' ZERO ERROR! ',D,' FORMAT CODE = ',FPTR
C
 30     CONTINUE
        FPTR = FPTR + 1
        RETURN
        END
```

```
C*************************************************************************
C
C       SINEWAVE GENERATOR
C
C       VAX-11 FORTRAN SOURCE FILENAME:         DSINEGEN.FOR
C
C       DEPARTMENT OF ELECTRICAL ENGINEERING    KANSAS STATE UNIVERSITY
C
C       REVISION        DATE                    PROGRAMMER(S)
C       --------        ----                    -------------
C       00.0            APR 20, 1983            KENT SCARBROUGH
C       01.0            MAY 19, 1983            JOHN W. BARTHOLOMEW
C
C*************************************************************************
C
C       PURPOSE
C
C               This routine computes a sequence of sinewaves
C               or sums of sinewaves.  The amplitudes, frequencies,
C               and phases of the sinewave(s) may be changed
C               for each block of points.
C
C*************************************************************************
C
C       NOTE :  The number of points per block may vary; however,
C               the total number of points to be generated is
C               limited by the array size of 250000 points.
C
C       NOTE :  The program also asks for a minimum absolute value to
C               truncate to 0.0D0 so that zero-crossings are exact.
C
C*************************************************************************
C
        DOUBLE PRECISION  SINE(250000), AMP(10,10)
        DOUBLE PRECISION  FRQ(10,10), PHI(10,10),TPI
        DOUBLE PRECISION  FS,TIND,ANG,DMIN
        REAL SINEOUT(250000)
        INTEGER  NWAV(10), NPTS(10), TPTS, NBLK
C
C  ASSIGN CONSTANTS
C
        IND = 0
        TPTS = 0
        TPI = 8.0D0 * DATAN(1.0D0)
    1   FORMAT(A)
C
C*************************************************
C
C  READ INPUT PARAMETERS
C
C
        TYPE   *, '*** SINEWAVE GENERATOR ***'
        TYPE   *
```

```
      TYPE   1, '$SAMPLING FREQUENCY = '
      ACCEPT *, FS
      TYPE   1, '$NUMBER OF BLOCKS(10 MAX) = '
      ACCEPT *, NBLK
C
C  ACCEPT SINEWAVE PARAMETERS
C
      DO IBLK = 1, NBLK
          TYPE   *
          TYPE   *, 'BLOCK # ', IBLK
          TYPE   *
C
          TYPE   1, '$  NUMBER OF POINTS = '
          ACCEPT *, NPTS(IBLK)
C
          TPTS = TPTS + NPTS(IBLK)
          IF (TPTS.GE.250000)  STOP 'MAX ARRAY SIZE(250000) EXCEEDED'
C
          TYPE   1, '$  NUMBER OF SINEWAVES(10 MAX) = '
          ACCEPT *, NWAV(IBLK)
          DO IWAV = 1, NWAV(IBLK)
              TYPE   *
              TYPE   *, '  SINEWAVE # ',IWAV
              TYPE   *
C
              TYPE   1, '$    AMPLITUDE = '
              ACCEPT *, AMP(IWAV,IBLK)
C
              TYPE   1, '$    FREQUENCY = '
              ACCEPT *, FRQ(IWAV,IBLK)
C
              TYPE   1, '$    PHASE(DEG) = '
              ACCEPT *, PHI(IWAV,IBLK)
C
              TYPE   1, '$    MINIMUM VALUE TRUNCATED TO ZERO = '
              ACCEPT *, DMIN
C
              PHI(IWAV,IBLK) = TPI * PHI(IWAV,IBLK) / 360.0D0
         END DO
      END DO
C
C***********************************************
C
C  OPEN OUTPUT FILE
C
      TYPE *
      CALL SGOPEN(0,'WRITE','OUTPUT FILENAME ? ','NONAME','REAL',TPTS)
C
C***********************************************
C
C  GENERATE SINEWAVE(S)
C
      DO  IBLK = 1, NBLK
```

```
            DO  IPT = 1, NPTS(IBLK)
                IND = IND + 1
                SINE(IND) = 0.0D0
                TIND = DBLE(IPT-1) * TPI / FS
                DO  IWAV = 1, NWAV(IBLK)
                    ANG = TIND * FRQ(IWAV,IBLK) + PHI(IWAV,IBLK)
                    SINE(IND) = SINE(IND) + AMP(IWAV,IBLK) * DSIN(ANG)
                    IF (DABS(SINE(IND)).LE.DMIN) SINE(IND) = 0.0D0
                END DO
            END DO
        END DO
C
C*************************************************
C
C  CONVERT DOUBLE PRECISION ARRAY TO REAL
C
        DO INDEX = 1, TPTS
                SINEOUT(INDEX) = SNGL(SINE(INDEX))
        END DO
C
C*************************************************
C
C  WRITE ARRAY OF POINTS TO DISK
C
        CALL SGTRAN(0,'WRITE','REAL',SINEOUT,TPTS)
C
        STOP
        END
```

```
C*************************************************************************
C
C       CHWHITE
C
C       VAX-11 FORTRAN SOURCE FILENAME:         CHWHITE.FOR
C
C       DEPARTMENT OF ELECTRICAL ENGINEERING    KANSAS STATE UNIVERSITY
C
C       REVISION        DATE                    PROGRAMMER
C       --------        ----                    ----------
C       00.0            JUNE 9, 1983            JOHN W. BARTHOLOMEW
C
C*************************************************************************
C
C       CALLING SEQUENCE
C
C               RUN CHWHITE
C
C       PURPOSE
C
C               This program allows the user to alter the contents of
C               data file (presumably Gaussian White noise) in the
C               following ways :
C
C                       1) Rescale any part of the file
C
C                       2) Append another file to an existing file
C
C                       3) Reorder the file by blocks
C
C                       4) Add the file to another file
C
C                       5) Limit the file to some min and max value
C
C                       6) Multiply the file by another file
C
C                       9) Output any part of the file to a new file
C
C                       0) Exit the program
C
C*************************************************************************
C
C
      REAL INPUT(125000),OUTPUT(125000),RESCALE,OFFSET
      REAL SCALE,UPPER,LOWER
      INTEGER I,LOW,HIGH,ISIZEIN,ISIZEOUT,ISIZE1,CODE
      INTEGER BLOCKSIZE,BLOCKS,ORDER(500),IIN,IOUT,J
      CHARACTER*80 FNAME,FNAME1
      LOGICAL FIRST/.TRUE./
C
C*************************************************************************
C
 2    FORMAT (A)
```

```
        TYPE *,' '
        TYPE *,' CHWHITE VERSION 0.0'
        TYPE *,' '
 10     CONTINUE
        TYPE *,' SELECT OPERATION : '
        TYPE *,'        1) RESCALE PART OF FILE'
        TYPE *,'        2) APPEND PART OF FILE TO ANOTHER FILE'
        TYPE *,'        3) REORDER FILE BY BLOCKS'
        TYPE *,'        4) ADD FILE TO ANOTHER FILE'
        TYPE *,'        5) LIMIT FILE TO SOME MAX AND MIN'
        TYPE *,'        6) MULTIPY FILE BY ANOTHER FILE'
        TYPE *,'        9) SAVE PART OF FILE'
        TYPE *,'        0) END PROGRAM'
        TYPE *,' '
        TYPE 2,'$ ENTER SELECTION(1,2,3,4,5,6,9,0) '
        ACCEPT *,CODE
        IF (CODE.EQ.1) GOTO 100
        IF (CODE.EQ.2) GOTO 200
        IF (CODE.EQ.3) GOTO 300
        IF (CODE.EQ.4) GOTO 400
        IF (CODE.EQ.5) GOTO 500
        IF (CODE.EQ.6) GOTO 600
        IF (CODE.EQ.9) GOTO 900
        IF (CODE.EQ.0) GOTO 1000
        TYPE *,' INVALID CODE'
        GOTO 10
C
C*************************************************************************
C
 100    CONTINUE
C
        IF (FIRST) THEN
                TYPE 2,'$ INPUT FILENAME ? '
                ACCEPT 2,FNAME
                CALL SGOPEN(0,'READ','NOPROMPT',FNAME,'REAL',ISIZEIN)
                CALL SGTRAN(0,'READ','REAL',INPUT,ISIZEIN)
        ENDIF
 105    CONTINUE
        TYPE *,' FILE LENGTH = ',ISIZEIN
        TYPE 2,'$ RANGE OF VALUES TO BE RESCALED ? '
        ACCEPT *,LOW,HIGH
        IF ((LOW.LE.0).OR.(HIGH.GT.ISIZEIN)) THEN
                TYPE *,' NOT IN RANGE, TRY AGAIN'
                GOTO 105
        ENDIF
        TYPE 2,'$ RESCALE FACTOR ? '
        ACCEPT *,RESCALE
        TYPE 2,'$ OFFSET FACTOR ? '
        ACCEPT *,OFFSET
        DO I=LOW,HIGH
                INPUT(I) = ( INPUT(I) * RESCALE ) + OFFSET
        ENDDO
C
```

```
        FIRST = .FALSE.
        GOTO 10
C
C*******************************************************************************
C
 200    CONTINUE
C
        IF (FIRST) THEN
                TYPE 2,'$ FIRST FILENAME ? '
                ACCEPT 2,FNAME
                CALL SGOPEN(0,'READ','NOPROMPT',FNAME,'REAL',ISIZEIN)
                CALL SGTRAN(0,'READ','REAL',INPUT,ISIZEIN)
        ENDIF
        TYPE 2,'$ SECOND FILENAME ? '
        ACCEPT 2,FNAME1
        CALL SGOPEN(1,'READ','NOPROMPT',FNAME1,'REAL',ISIZE1)
        ISIZEOUT = ISIZEIN + ISIZE1
        IF (ISIZEOUT.GT.125000) THEN
                TYPE *,' APPENDED FILE IS TOO LONG'
                GOTO 1000
        ENDIF
        CALL SGTRAN(1,'READ','REAL',OUTPUT,ISIZE1)
        DO I=1,ISIZE1
                INPUT(I+ISIZEIN) = OUTPUT(I)
        ENDDO
        ISIZEIN = ISIZEOUT
C
        FIRST = .FALSE.
        GOTO 10
C
C*******************************************************************************
C
 300    CONTINUE
C
        IF (FIRST) THEN
                TYPE 2,'$ INPUT FILENAME ? '
                ACCEPT 2,FNAME
                CALL SGOPEN(0,'READ','NOPROMPT',FNAME,'REAL',ISIZEIN)
                CALL SGTRAN(0,'READ','REAL',INPUT,ISIZEIN)
        ENDIF
        TYPE *,' FILE LENGTH = ',ISIZEIN
        TYPE 2,'$ SIZE OF BLOCKS ? '
        ACCEPT *,BLOCKSIZE
        BLOCKS = ISIZEIN / BLOCKSIZE
        TYPE *,' ENTER THE NEW ORDER OF TTHE BLOCKS'
        DO I=1,BLOCKS
 310            CONTINUE
                TYPE 3,'$ BLOCK #',I,' ? '
 3              FORMAT(A9,I5,A3)
                ACCEPT *,ORDER(I)
                IF (ORDER(I).GT.BLOCKS) THEN
                        TYPE *,' BLOCK NUMBER TOO LARGE'
                        GOTO 310
```

```
                ENDIF
        ENDDO
        DO I=1,BLOCKS
                DO J=1,BLOCKSIZE
                        IOUT = (I-1) * BLOCKSIZE + J
                        IIN = (ORDER(I)-1) * BLOCKSIZE + J
                        OUTPUT(IOUT) = INPUT(IIN)
                        ISIZEOUT = IOUT
                ENDDO
        ENDDO
        DO I=1,ISIZEOUT
                INPUT(I) = OUTPUT(I)
        ENDDO
        ISIZEIN = ISIZEOUT
C
        FIRST = .FALSE.
        GOTO 10
C
C*******************************************************************************
C
 400    CONTINUE
C
        IF (FIRST) THEN
                TYPE 2,'$ FIRST FILENAME ? '
                ACCEPT 2,FNAME
                CALL SGOPEN(0,'READ','NOPROMPT',FNAME,'REAL',ISIZEIN)
                CALL SGTRAN(0,'READ','REAL',INPUT,ISIZEIN)
        ENDIF
        TYPE 2,'$ SECOND FILENAME ? '
        ACCEPT 2,FNAME1
        CALL SGOPEN(1,'READ','NOPROMPT',FNAME1,'REAL',ISIZEOUT)
        CALL SGTRAN(1,'READ','REAL',OUTPUT,ISIZEOUT)
        TYPE 2,'$ SCALE FACTOR FOR SECOND FILE ? '
        ACCEPT *,SCALE
        DO I=1,MINO(ISIZEIN,ISIZEOUT)
                INPUT(I) = INPUT(I) + SCALE * OUTPUT(I)
        ENDDO
        ISIZEIN = MINO(ISIZEIN,ISIZEOUT)
C
        FIRST = .FALSE.
        GOTO 10
C
C*******************************************************************************
C
 500    CONTINUE
C
        IF (FIRST) THEN
                TYPE 2,'$ INPUT FILENAME ?'
                ACCEPT 2,FNAME
                CALL SGOPEN(0,'READ','NOPROMPT',FNAME,'REAL',ISIZEIN)
                CALL SGTRAN(0,'READ','REAL',INPUT,ISIZEIN)
        ENDIF
        TYPE 2,'$ UPPER AND LOWER LIMITS ? '
```

```
        ACCEPT *,UPPER,LOWER
        DO I=1,ISIZEIN
                IF (INPUT(I).GT.UPPER) THEN
                        INPUT(I) = UPPER
                ENDIF
                IF (INPUT(I).LT.LOWER) THEN
                        INPUT(I) = LOWER
                ENDIF
        ENDDO
C
        FIRST = .FALSE.
        GOTO 10
C
C*******************************************************************************
C
 600    CONTINUE
C
        IF (FIRST) THEN
                TYPE 2,'$ FIRST FILENAME ? '
                ACCEPT 2,FNAME
                CALL SGOPEN(0,'READ','NOPROMPT',FNAME,'REAL',ISIZEIN)
                CALL SGTRAN(0,'READ','REAL',INPUT,ISIZEIN)
        ENDIF
        TYPE 2,'$ SECOND FILENAME ? '
        ACCEPT 2,FNAME1
        CALL SGOPEN(1,'READ','NOPROMPT',FNAME1,'REAL',ISIZEOUT)
        CALL SGTRAN(1,'READ','REAL',OUTPUT,ISIZEOUT)
        TYPE 2,'$ SCALE FACTOR FOR SECOND FILE ? '
        ACCEPT *,SCALE
        DO I=1,MIN0(ISIZEIN,ISIZEOUT)
                INPUT(I) = INPUT(I) * SCALE * OUTPUT(I)
        ENDDO
        ISIZEIN = MIN0(ISIZEIN,ISIZEOUT)
C
        FIRST = .FALSE.
        GOTO 10
C
C*******************************************************************************
C
 900    CONTINUE
C
        IF (FIRST) THEN
                TYPE 2,'$ INPUT FILENAME ? '
                ACCEPT 2,FNAME
                CALL SGOPEN(0,'READ','NOPROMPT',FNAME,'REAL',ISIZEIN)
                CALL SGTRAN(0,'READ','REAL',INPUT,ISIZEIN)
        ENDIF
 905    CONTINUE
        TYPE *,' FILE LENGTH = ',ISIZEIN
        TYPE 2,'$ RANGE OF VALUES TO BE SAVED ? '
        ACCEPT *,LOW,HIGH
        IF ((LOW.LE.0).OR.(HIGH.GT.ISIZEIN)) THEN
                TYPE *,' NOT IN RANGE'
```

```
                GOTO 905
        ENDIF
        ISIZEOUT = HIGH - LOW + 1
        DO I=LOW,HIGH
                OUTPUT(I-LOW+1) = INPUT(I)
        ENDDO
        CALL SGOPEN(1,'WRITE',' OUTPUT FILENAME ? ',
       1                'NONAME','REAL',ISIZEOUT)
        CALL SGTRAN(1,'WRITE','REAL',OUTPUT,ISIZEOUT)
C
        FIRST = .FALSE.
        GOTO 10
C
C*******************************************************************************
C
 1000   CONTINUE
        STOP
        END
```

```
C*****************************************************************************
C
C       SGOPEN SUBROUTINE                           (UTILITY LIBRARY)
C
C       VAX-11 FORTRAN SOURCE FILENAME:             SGOPEN.FOR
C
C       DEPARTMENT OF ELECTRICAL ENGINEERING        KANSAS STATE UNIVERSITY
C
C       REVISION        DATE                        PROGRAMMER(S)
C       --------        ----                        -------------
C       00.0            MAR 31, 1983                DEVORE/JUNOD/RATCLIFFE
C
C*****************************************************************************
C
C       CALLING SEQUENCE
C
C               CALL SGOPEN (UNIT, OPER, PROMPT, NAME, TYPE, NUMBER)
C
C       PURPOSE
C
C               This routine may request the terminal operator to supply
C               a filename from which to read or write data.  The
C               file is opened after a valid filename has been supplied.
C
C       ROUTINE(S) ACCESSED OR CALLED BY THIS ROUTINE
C
C               SGLENG          utility library routine
C
C       ARGUMENT(S) REQUIRED FROM THE CALLING ROUTINE
C
C               NAME    filename (character constant, variable or
C                       string of ASCII characters inclosed in quotes)
C                       (see NOTES 2 thru 4 below)
C
C               NUMBER  how many data elements to be written
C                       (1 or greater)  (see NOTE 5 below)
C
C               OPER    = 'READ'        open file for reading
C                       = 'WRITE'       open file for writing
C
C               PROMPT  question (character constant, variable or
C                       string of ASCII characters inclosed in quotes)
C                       (see NOTES 2 thru 4 below)
C
C               TYPE    = 'BYTE'
C                       = 'COMPLEX'
C                       = 'DOUBLE PRECISION'
C                       = 'INTEGER'
C                       = 'INTEGER*2'
C                       = 'REAL'
C                       data type that file should or will contain
C
C               UNIT    logical unit specifier (0-99)
```

```
C
C       ARGUMENT(S) SUPPLIED  TO  THE CALLING ROUTINE
C
C               NAME    filename (character constant, variable or
C                       string of ASCII characters inclosed in quotes)
C                       (see NOTES 2 thru 4 below)
C
C               NUMBER  how many data elements that can be read
C                       (see NOTE 5 below)
C
C*************************************************************************
C
C       NOTE 1: The argument(s) supplied by this routine are guaranteed
C               to be correct only if the disk file was written as
C               A) a single FORTRAN unformatted sequential record of
C                  any length or
C               B) a set of FORTRAN unformatted sequential records.
C                  Each record (with the exception of the last one)
C                  must have the same size.  An individual record
C                  must be smaller than 2042 bytes.
C               Any data disk file written using SGTRAN will
C               automatically satisfy these conditions.
C
C       NOTE 2: The terminal operator is not querried for a filename
C               when PROMPT = 'NOPROMPT'.  The value of NAME passed
C               from the calling routine is used for the filename.
C
C       NOTE 3: A filename supplied by the operator will not be returned
C               to the calling routine when NAME = 'NONAME'.
C
C       NOTE 4: The combination of NAME = 'NONAME' and
C               PROMPT = 'NOPROMPT' is not allowed.
C
C       NOTE 5: The calling routine supplies NUMBER when OPER = 'WRITE'.
C               This routine returns NUMBER when OPER = 'READ'.
C
C*************************************************************************
C
        SUBROUTINE SGOPEN (UNIT, OPER, PROMPT, NAME, TYPE, NUMBER)
C
        CHARACTER*(*)  OPER, PROMPT, NAME, TYPE
        INTEGER  UNIT, NUMBER
        CHARACTER*80  FNAME
        LOGICAL  VALID
C
1       FORMAT (A)
C
C***************************************
C
C       VERIFY THAT A VALID DATA TYPE IS SPECIFIED
C
        IF (TYPE.EQ.'REAL')  THEN
                NBYTE = 4
```

```
      ELSE    IF (TYPE.EQ.'INTEGER')  THEN
              NBYTE = 4
      ELSE    IF (TYPE.EQ.'COMPLEX')  THEN
              NBYTE = 8
      ELSE    IF (TYPE.EQ.'DOUBLE PRECISION')  THEN
              NBYTE = 8
      ELSE    IF (TYPE.EQ.'INTEGER*2')  THEN
              NBYTE = 2
      ELSE    IF (TYPE.EQ.'BYTE')  THEN
              NBYTE = 1
      ELSE
              TYPE *
              TYPE *, 'argument TYPE invalid'
              STOP 'FATAL ERROR -> ROUTINE SGOPEN <-'
      END IF
C
C*****************************************
C
C       VERIFY THAT REMAINING ARGUMENT(S) ARE VALID
C
      IF (NAME.EQ.'NONAME'.AND.PROMPT.EQ.'NOPROMPT')  THEN
              TYPE *
              TYPE *, 'arguments NAME and PROMPT inconsistent'
              STOP 'FATAL ERROR -> ROUTINE SGOPEN <-'
      END IF
      IF (NUMBER.LE.0.AND.OPER.EQ.'WRITE')  THEN
              TYPE *
              TYPE *, 'argument NUMBER not greater than zero'
              TYPE *, 'when file is to be opened for writing'
              STOP 'FATAL ERROR -> ROUTINE SGOPEN <-'
      END IF
      IF (OPER.NE.'READ'.AND.OPER.NE.'WRITE')  THEN
              TYPE *
              TYPE *, 'argument OPER invalid'
              STOP 'FATAL ERROR -> ROUTINE SGOPEN <-'
      END IF
C
C*****************************************
C
C       CLOSE FILE IF OPENED
C
      INQUIRE (UNIT, OPENED = VALID)
      IF (VALID)  CLOSE (UNIT)
C
C*****************************************
C
C       HERE TO OPEN FILE FOR READING
C
      IF (OPER.EQ.'READ')  THEN
              IF (PROMPT.EQ.'NOPROMPT')  THEN
                      CALL SGLENG (NAME, NBYTE, NPOINT, NUMBER, VALID)
                      IF (.NOT.VALID)  THEN
                              TYPE *
```

```
                        TYPE *, 'wrong combination of arguments'
                        STOP 'FATAL ERROR -> ROUTINE SGOPEN <-'
                END IF
                FNAME = NAME
        ELSE
                VALID = .FALSE.
                DO  WHILE (.NOT.VALID)
                        TYPE 1, '$'//PROMPT
                        READ 1, FNAME
                        CALL SGLENG (FNAME, NBYTE, NPOINT,
     &                                  NUMBER, VALID)
                        IF (.NOT.VALID)  THEN
                                TYPE *
                                TYPE *, 'something went wrong'
                                TYPE *, 'PLEASE TRY AGAIN'
                                TYPE *
                        END IF
                END DO
                IF (NAME.NE.'NONAME')  NAME = FNAME
        END IF
        OPEN (UNIT, FILE = FNAME, READONLY, STATUS = 'OLD',
     &       FORM = 'UNFORMATTED')
   END IF
C
C****************************************
C
C       HERE TO OPEN FILE FOR WRITING
C
        IF (OPER.EQ.'WRITE')  THEN
                LENGTH = NUMBER * NBYTE
                NRECORD = (LENGTH + 1023) / 1024
                ISIZE = (LENGTH + NRECORD * 4 + 511) / 512
                IF (PROMPT.EQ.'NOPROMPT')  THEN
                        OPEN (UNIT, FILE = NAME, STATUS = 'NEW',
     &                       SHARED, INITIALSIZE = ISIZE, FORM
     &                       = 'UNFORMATTED', IOSTAT = IERROR)
                        IF (IERROR.NE.0)  THEN
                                TYPE *
                                TYPE *, 'error code  = ', IERROR
                                STOP 'FATAL ERROR -> ROUTINE SGOPEN <-'
                        END IF
                ELSE
                        VALID = .FALSE.
                        DO  WHILE (.NOT.VALID)
                                TYPE 1, '$'//PROMPT
                                READ 1, FNAME
                                OPEN (UNIT, FILE = FNAME, STATUS = 'NEW',
     &                                 SHARED, INITIALSIZE = ISIZE, FORM
     &                                 = 'UNFORMATTED', IOSTAT = IERROR)
                                IF (IERROR.EQ.0)  VALID = .TRUE.
                                IF (.NOT.VALID)  THEN
                                        TYPE *
                                        TYPE *, 'error code  = ', IERROR
```

```
                                TYPE *, 'PLEASE TRY AGAIN'
                                TYPE *
                            END IF
                        END DO
                        IF (NAME.NE.'NONAME')  NAME = FNAME
                END IF
        END IF
        RETURN
        END
```

```
C*****************************************************************************
C
C       SGTRAN SUBROUTINE                           (UTILITY LIBRARY)
C
C       VAX-11 FORTRAN SOURCE FILENAME:             SGTRAN.FOR
C
C       DEPARTMENT OF ELECTRICAL ENGINEERING        KANSAS STATE UNIVERSITY
C
C       REVISION        DATE                        PROGRAMMER(S)
C       --------        ----                        -------------
C       00.0            MAR 31, 1983                DEVORE/JUNOD/RATCLIFFE
C
C*****************************************************************************
C
C       CALLING SEQUENCE
C
C               CALL SGTRAN (UNIT, OPER, TYPE, ARRAY, NUMBER)
C
C       PURPOSE
C
C               This routine reads or writes a data disk file.
C
C       ROUTINE(S) ACCESSED OR CALLED BY THIS ROUTINE
C
C               SGLENG          utiltiy library routine
C
C       ARGUMENT(S) REQUIRED FROM THE CALLING ROUTINE
C
C               ARRAY   vector containing data elements to be written
C                       (must be dimensioned NUMBER or greater by the
C                       calling routine)
C
C               NUMBER  how many data elements to be read or written
C                       (1 or greater)
C
C               OPER    = 'READ'        read from file
C                       = 'WRITE'       write to file
C
C               TYPE    = 'BYTE'
C                       = 'COMPLEX'
C                       = 'DOUBLE PRECISION'
C                       = 'INTEGER'
C                       = 'INTEGER*2'
C                       = 'REAL'
C                       data type that file should or will contain
C
C               UNIT    logical unit specifier (0-99)
C
C       ARGUMENT(S) SUPPLIED  TO  THE CALLING ROUTINE
C
C               ARRAY   vector containing data elements read
C                       (must be dimensioned NUMBER or greater by the
C                       calling routine)
```

```
C
C*******************************************************************************
C
C       NOTE 1: The argument(s) supplied by this routine are guaranteed
C               to be correct only if the disk file was written as
C               A) a single FORTRAN unformatted sequential record of
C                  any length or
C               B) a set of FORTRAN unformatted sequential records.
C                  Each record (with the exception of the last one)
C                  must have the same size.  An individual record
C                  must be smaller than 2042 bytes.
C               Any data disk file written using SGTRAN will
C               automatically satisfy these conditions.
C
C*******************************************************************************
C
        SUBROUTINE SGTRAN (UNIT, OPER, TYPE, ARRAY, NUMBER)
C
        BYTE  ARRAY(*)
        CHARACTER*(*)  OPER, TYPE
        INTEGER  UNIT, NUMBER
        CHARACTER*80  FNAME
        LOGICAL  VALID
C
C****************************************
C
C       VERIFY THAT A VALID DATA TYPE IS SPEICIFIED
C
        IF (TYPE.EQ.'REAL')  THEN
                NBYTE = 4
        ELSE    IF (TYPE.EQ.'INTEGER')  THEN
                NBYTE = 4
        ELSE    IF (TYPE.EQ.'COMPLEX')  THEN
                NBYTE = 8
        ELSE    IF (TYPE.EQ.'DOUBLE PRECISION')  THEN
                NBYTE = 8
        ELSE    IF (TYPE.EQ.'INTEGER*2')  THEN
                NBYTE = 2
        ELSE    IF (TYPE.EQ.'BYTE')  THEN
                NBYTE = 1
        ELSE
                TYPE *
                TYPE *, 'argument TYPE invalid'
                STOP 'FATAL ERROR -> ROUTINE SGTRAN <-'
        END IF
C
C****************************************
C
C       VERIFY THAT REMAINING ARGUMENT(S) ARE VALID
C
        INQUIRE (UNIT, NAME = FNAME, OPENED = VALID)
        IF (.NOT.VALID)  THEN
                TYPE *
```

```
                TYPE *, 'file specified by argument UNIT not open'
                STOP 'FATAL ERROR -> ROUTINE SGTRAN <-'
        END IF
        IF (NUMBER.LE.0)  THEN
                TYPE *
                TYPE *, 'argument NUMBER not greater than zero'
                STOP 'FATAL ERROR -> ROUTINE SGTRAN <-'
        END IF
        IF (OPER.NE.'READ'.AND.OPER.NE.'WRITE')  THEN
                TYPE *
                TYPE *, 'argument OPER invalid'
                STOP 'FATAL ERROR -> ROUTINE SGTRAN <-'
        END IF
C
C*****************************************
C
C       HERE FOR READ OPERATION
C
        IF (OPER.EQ.'READ')  THEN
                CALL SGLENG (FNAME, NBYTE, NPOINT, LENGTH, VALID)
                IF (.NOT.VALID)  THEN
                        TYPE *
                        TYPE *, 'wrong combination of arguments'
                        STOP 'FATAL ERROR -> ROUTINE SGTRAN <-'
                END IF
                IF (NUMBER.GT.LENGTH)  THEN
                        TYPE *
                        TYPE *, 'argument NUMBER too large for file'
                        STOP 'FATAL ERROR -> ROUTINE SGTRAN <-'
                END IF
                IEND = NBYTE * NUMBER
                ISTEP = NBYTE * NPOINT
                DO  I = 1, IEND, ISTEP
                        JEND = MIN (I + ISTEP - 1, IEND)
                        READ (UNIT)  (ARRAY(J), J = I, JEND)
                END DO
                REWIND (UNIT)
        END IF
C
C*****************************************
C
C       HERE FOR WRITE OPERATION
C
        IF (OPER.EQ.'WRITE')  THEN
                IEND = NBYTE * NUMBER
                ISTEP = 1024
                DO  I = 1, IEND, ISTEP
                        JEND = MIN (I + ISTEP - 1, IEND)
                        WRITE (UNIT)  (ARRAY(J), J = I, JEND)
                END DO
                REWIND (UNIT)
        END IF
        RETURN
        END
```

```
;*****************************************************************************
;
;       SGLENG SUBROUTINE                               (PRIMATIVE LIBRARY)
;
;       VAX-11 MACRO ASSEMBLER SOURCE FILENAME: SGLENG.MAR
;
;       DEPARTMENT OF ELECTRICAL ENGINEERING    KANSAS STATE UNIVERSITY
;
;       REVISION        DATE                    PROGRAMMER(S)
;       --------        ----                    -------------
;       00.0            MAR 31, 1983            DEVORE/JUNOD/RATCLIFFE
;
;*****************************************************************************
;
;       CALLING SEQUENCE
;
;               CALL SGLENG (NAME, NBYTE, NPOINT, NUMBER, STATUS)
;
;       PURPOSE
;
;               This routine determines the size of a data disk file.
;
;       ROUTINE(S) ACCESSED OR CALLED BY THIS ROUTINE
;
;               $CLOSE          VAX-11 RMS macro call
;               $FAB            VAX-11 RMS macro call
;               $FAB_STORE      VAX-11 RMS macro call
;               $OPEN           VAX-11 RMS macro call
;               $XABFHC         VAX-11 RMS macro call
;
;       ARGUMENT(S) REQUIRED FROM THE CALLING ROUTINE
;
;               NAME    filename (character constant, variable or
;                       string of ASCII characters inclosed in quotes)
;
;               NBYTE   number of bytes needed to store one data element
;
;       ARGUMENT(S) SUPPLIED  TO  THE CALLING ROUTINE
;
;               NPOINT  number of data elements per FORTRAN
;                       unformatted sequential record
;                       (always equals 0 if STATUS = .FALSE.)
;
;               NUMBER  how many data elements in the file
;                       (always equals 0 if STATUS = .FALSE.)
;
;               STATUS  = .FALSE.       an error has occurred
;                       = .TRUE.        no error has occurred
;                                       (see NOTE 2 below)
;
;*****************************************************************************
;
;       NOTE 1: The argument(s) supplied by this routine are guaranteed
```

```
;               to be correct only if the disk file was written as
;               A) a single FORTRAN unformatted sequential record of
;                  any length or
;               B) a set of FORTRAN unformatted sequential records.
;                  Each record (with the exception of the last one)
;                  must have the same size.  An individual record
;                  must be smaller than 2042 bytes.
;               Any data disk file written using SGTRAN will
;               automatically satisfy these conditions.
;
;       NOTE 2: If trouble developed while accessing the file then
;               STATUS will equal the VAX-11 Record Management Services
;               (RMS) completion status code.  This completion status
;               code can be decoded by a FORTRAN program as follows:
;
;               IF (.NOT.STATUS)  CALL LIB$STOP (STATUS)
;
;*******************************************************************************
;
                .TITLE  SGLENG
                .PSECT  RWDATA,WRT,NOEXE

FAB:            $FAB    XAB=FHC,SHR=<GET,PUT,DEL,UPD,UPI,MSE>
FHC:            $XABFHC

                .PSECT  CODE,EXE,NOWRT

                .ENTRY  SGLENG,^M<R2,R3>

                MOVL    4(AP),R1                ;GET STRING DESC IN R1
                $FAB_STORE FAB=FAB,FNA=@4(R1),FNS=(R1)
                $OPEN   FAB = FAB
                BLBS    R0,1$                   ;CONT IF NO ERROR
                BRW     EXIT                    ;ELSE EXIT
;
;       OPEN WORKED - NOW VERIFY FILE IS PROPER TYPE
;
        1$:     CMPB    FHC+XAB$B_ATR,#0        ;VERIFY RAT IS NULL
                BNEQ    2$
                CMPB    FHC+XAB$B_RFO,#FAB$C_VAR ;VERIFY RFM=VAR & ORG=SEQ
                BEQL    3$
        2$:     BRW     EXITC
;
;       ASSUME TYPE LENGTH IS 1, 2, 4, 8, OR 16 FOR THIS INTERNAL ROUTINE
;
;
;       CALCULATE TOTAL BYTES IN FILE
;
        3$:     MOVL    FHC+XAB$L_EBK,R0        ;END OF FILE BLOCK
                DECL    R0                      ;NUMB OF FULL BLOCKS
                MULL2   #512,R0                 ;NUMB OF BYTES
                MOVZWL  FHC+XAB$W_FFB,R1        ;GET FIRST FREE BYTE (0 OR
G)
                ADDL2   R1,R0
```

```
;
;       PICK UP THE MAX RECORD LENGTH IN FILE
;
                MOVZWL  FHC+XAB$W_LRL,R1
;
;       CALCULATE THE NUMBER OF DATA BYTES IN FILE
;
                ADDL3   R0,R1,R2                ;PREPARE TO ROUND UP
                ADDL3   #2,R1,R3                ;LEN OF REC + LEN BYTES
                DIVL2   R3,R2                   ;R2 CONTAINS # OF RECORDS
                ROTL    #2,R2,R2                ;NUMB OF CONTROL BYTES IN
FILE
                SUBL2   R2,R0                   ;LESS CONTROL LONGWORDS =
DATA
;
;       CALCULATE NUMB OF ITEMS OF DATA & VERIFY REMAINDER EQUAL ZERO
;
                CLRL    R1
                EDIV    @8(AP),R0,@16(AP),R3
                TSTL    R3
                BNEQ    EXITC
;
;       SEE IF FORTRAN SET THE BLOCKING SIZE
;
                MOVZWL  FHC+XAB$W_LRL,R0
                CMPL    #2044,R0
                BLEQ    4$                      ;FORTRAN BLOCKING
;
;       CALCULATE USER BLOCKING USED
;
                SUBL2   #2,R0
                EDIV    @8(AP),R0,@12(AP),R3
                TSTL    R3
                BNEQ    EXITC
                BRB     5$
;
;       RETURN LENGTH AS BLOCKING
;
        4$:     MOVL    @16(AP),@12(AP)         ;ASSUME ALL ONE BIG RECORD

        5$:     $CLOSE  FAB = FAB

RET:            MOVL    R0,@20(AP)              ;PUT STATUS IN FIFTH ARG
                RET

EXITC:          $CLOSE  FAB = FAB
                CLRL    R0
EXIT:           CLRL    @12(AP)
                CLRL    @16(AP)
                BRB     RET

                .END
```

APPENDIX E

This appendix contains a description of all of the input test signals, and which outputs used them.

A white noise file named WHITE1.FP of 24800 samples was formed on the NOVA using the DATGEN and WHITE programs written by SANDIA and transfered to the VAX 11/750.

HIBAND :

NAME	AMPLITUDE	FREQUENCY	OUTPUT POINTS
SINH1.FP COSH1.FP	1.0	40 Hz	1,2,3
SINH2.FP COSH2.FP	1.0	78 Hz	4
SINH3.FP COSH3.FP	1.0	73.2 Hz	5,6,7,8,9
SINH4.FP COSH4.FP	0.05	40 Hz	1,2,3
SINH5.FP COSH5.FP	0.05	12 Hz	4,5,6,7,8,9

The following noisy input files were formed:

WHITEHI.FP	Rescaled WHITE1.FP * 2 , points 1,4800
WHITEHQ.FP	Rescaled WHITE1.FP * 2 , points 4801,9600

For n = 1,2,3

SINWHn.FP	SINHn.FP + WHITEHI.FP, limited to +-1.0
COSWHn.FP	COSHn.FP + WHITEHQ.FP, limited to +-1.0

For n = 4,5

SINWHn.FP	SINHn.FP + rescaled WHITEHI.FP * 0.05
COSWHn.FP	COSHn.FP + rescaled WHITEHQ.FP * 0.05

MIDBAND :

NAME	AMPLITUDE	FREQUENCY	OUTPUT POINTS
SINM1.FP COSM1.FP	1.0	4 Hz	1,2,3,4,5
SINM2.FP COSM2.FP	1.0	9.8 Hz	6
SINM3.FP COSM3.FP	1.0	9.1 Hz	7,8,9,10
SINM4.FP COSM4.FP	0.05	4 Hz	1,2,3,4,5
SINM5.FP COSM5.FP	0.05	2.75 Hz	6,7,8,9,10

The following noisy input files were formed:

WHITEMI.FP	Rescaled WHITE1.FP * 2 , points 1,19200
WHITEMQ.FP	Rescaled WHITE1.FP * 2 , reordered by blocks blocklength = 960 (20 blocks) order: 20,19,17,14,10,5,1,18,16,13,9,4,2, 15,12,8,3,11,7,6,21

For n = 1,2,3

SINWMn.FP	SINMn.FP + WHITEMI.FP, limited to +-1.0
COSWMn.FP	COSMn.FP + WHITEMQ.FP, limited to +-1.0

For n = 4,5

SINWMn.FP	SINMn.FP + rescaled WHITEMI.FP * 0.05
COSWMn.FP	COSMn.FP + rescaled WHITEMQ.FP * 0.05

LOBAND :

A 1000 sample long, 1.0 Hz signal with the same amplitude as the rest of the input is inserted in all of the following input signal at points 86401 - 87400.

NAME	AMPLITUDE	FREQUENCY	OUTPUT POINTS
SINL1.FP COSL1.FP	1.0	1.7 Hz	1 - 13
SINL2.FP COSL2.FP	1.0	2.7 Hz	14
SINL3.FP COSL3.FP	1.0	2.5 Hz	15,16,17,18
SINL4.FP COSL4.FP	0.05	1.7 Hz	1,2
SINL5.FP COSL5.FP	0.05	0.2 Hz	3 - 18

The following noisy input files were formed:

WHITELI.FP	Append WHITEl.FP 6 times, rescaled * 2 , points 1,115200
WHITELQ.FP	Reordered WHITELI.FP by blocks, blocklength = 2560 (45 blocks) order: 8,7,6,5,4,3,2,1,16,14,12,10,15,13, 11,9,23,24,21,22,19,20,17,18,27,28, 29,30,31,32,25,26,37,38,39,40,33,34, 35,36,44,43,42,41,45

For n = 1,2,3

SINWLn.FP	SINLn.FP + WHITELI.FP, limited to +-1.0
COSWLn.FP	COSLn.FP + WHITELQ.FP, limited to +-1.0

For n = 4,5

SINWLn.FP	SINLn.FP + rescaled WHITELI.FP * 0.05
COSWLn.FP	COSLn.FP + rescaled WHITELQ.FP * 0.05

APPENDIX F

This appendix contains the entire set of output plots generated by the simulation. The plots are organized in the following manner:

1. High Band
2. Mid Band
3. Low Band

Within each band, the plots are organized according to type of input being tested:

1. +-1.0 maximum signal
2. +-0.05 minimum signal
3. +-1.0 maximum signal with 20DB SNR
4. +-0.05 minimum signal with 20DB SNR

Within each type of input signal being tested, the complete double precision version is shown first, followed by the fixed point version. The filenames describe which output point is selected and what inputs were used to generate the output. The following codes were used:

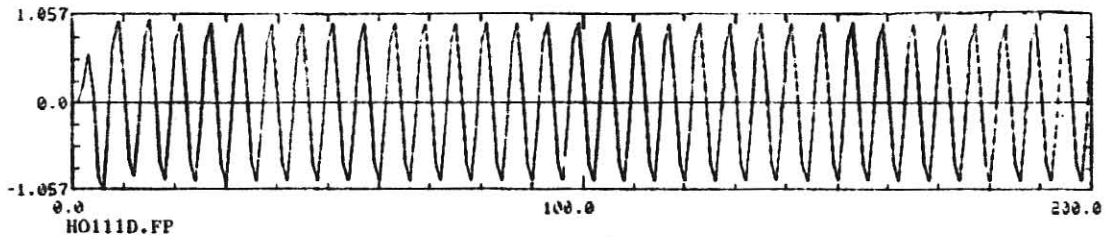
MO455D.FP

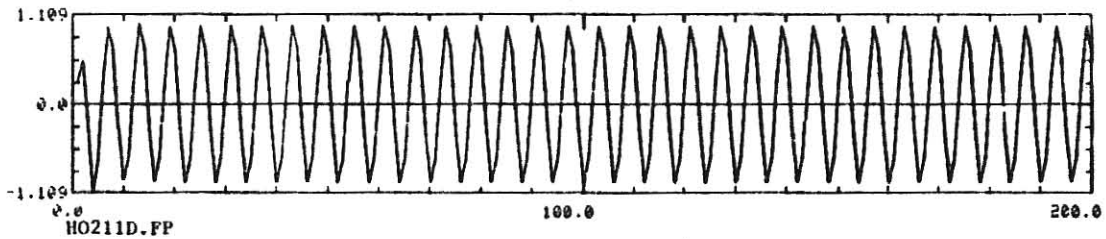
```
MO : Mid band Output
4  : output point number 4 (see filter block diagram in
     appendix A or refer to the program listing)
55 : input signal SINM5.FP & COSM5.FP
D  : coefficient file MCD.FP
   : absence of another letter indicates double
     precision output
FP : Floating Point file
```

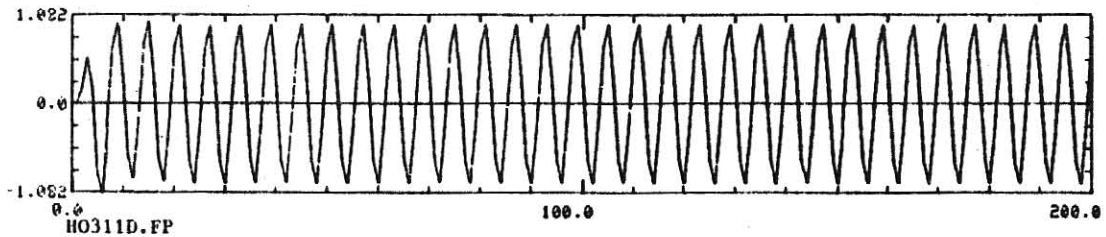
LO1W1GB.FP

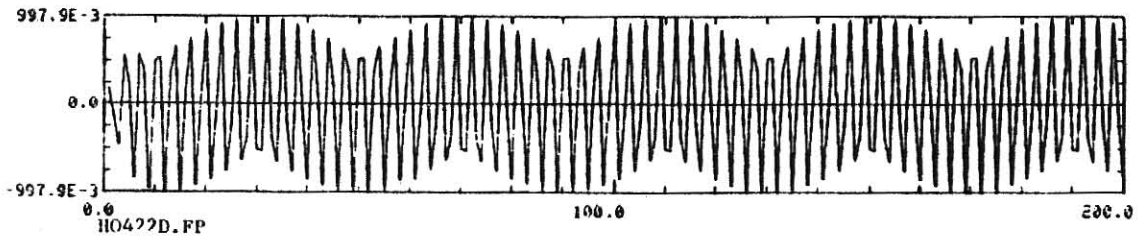
```
LO : Low band Output
1  : output point number 1
W1 : input signal SINWL1.FP & COSWL1.FP
G  : coefficient file LCG.FP
B  : format file LFB.FP (16 bit)
FP : Floating Point file
```

HIGH BAND: +-1.0 input without noise, double precision

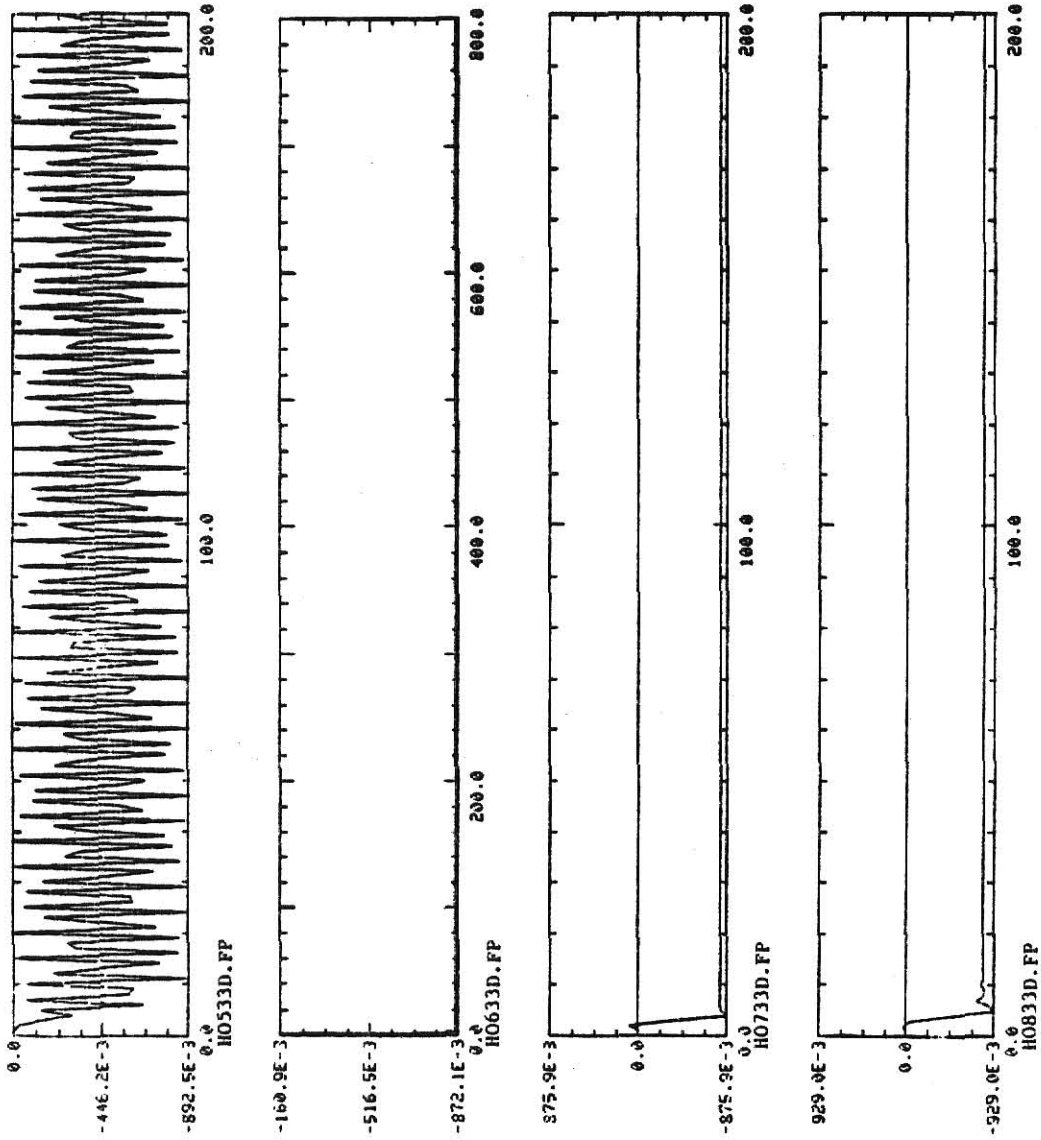




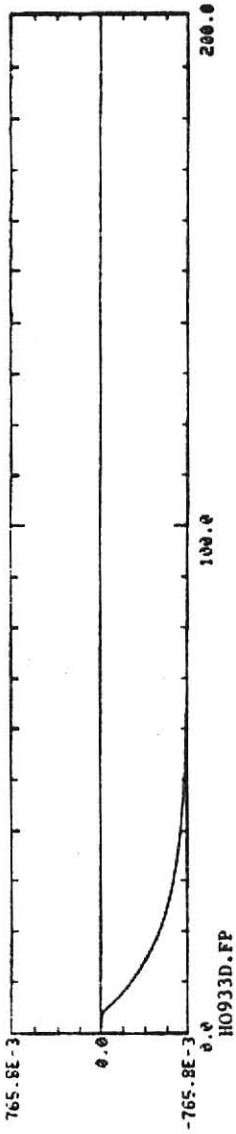




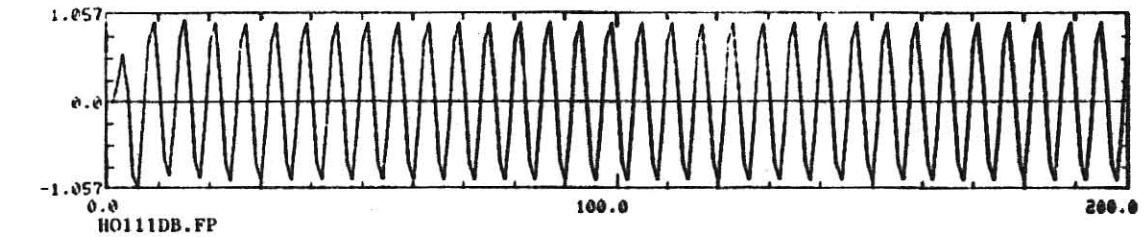
HIGH BAND: +-1.0 input without noise, double precision

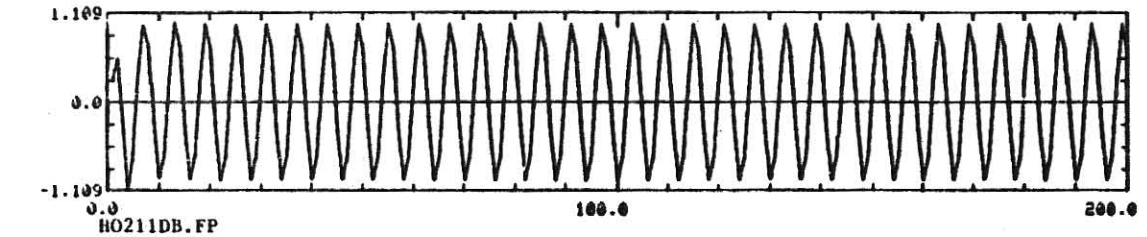


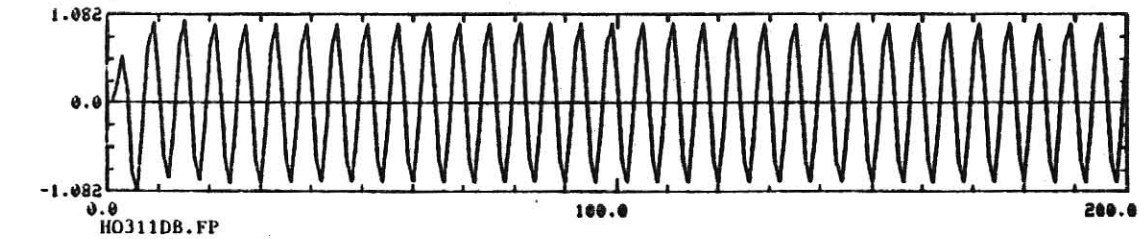
HIGH BAND: +-1.0 input without noise, double precision

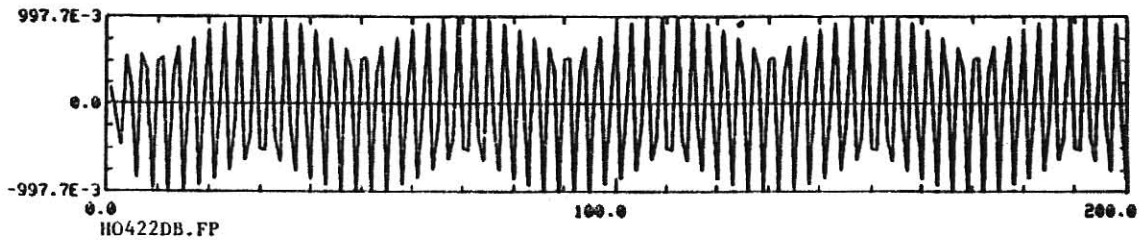


HIGH BAND: +-1.0 input without noise, fixed point

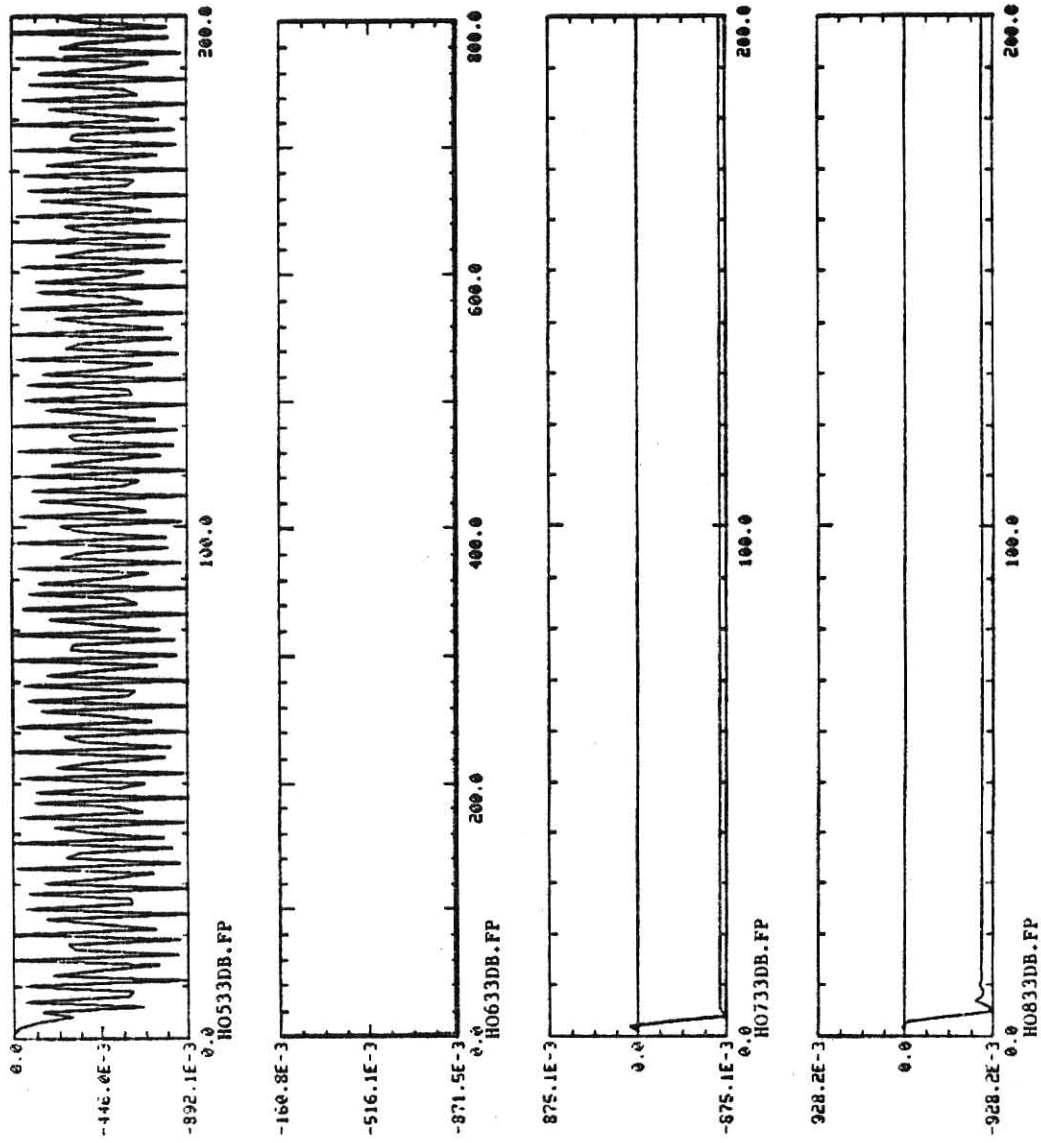




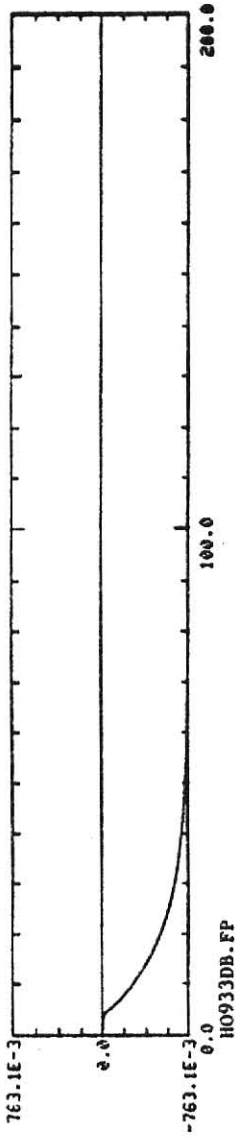




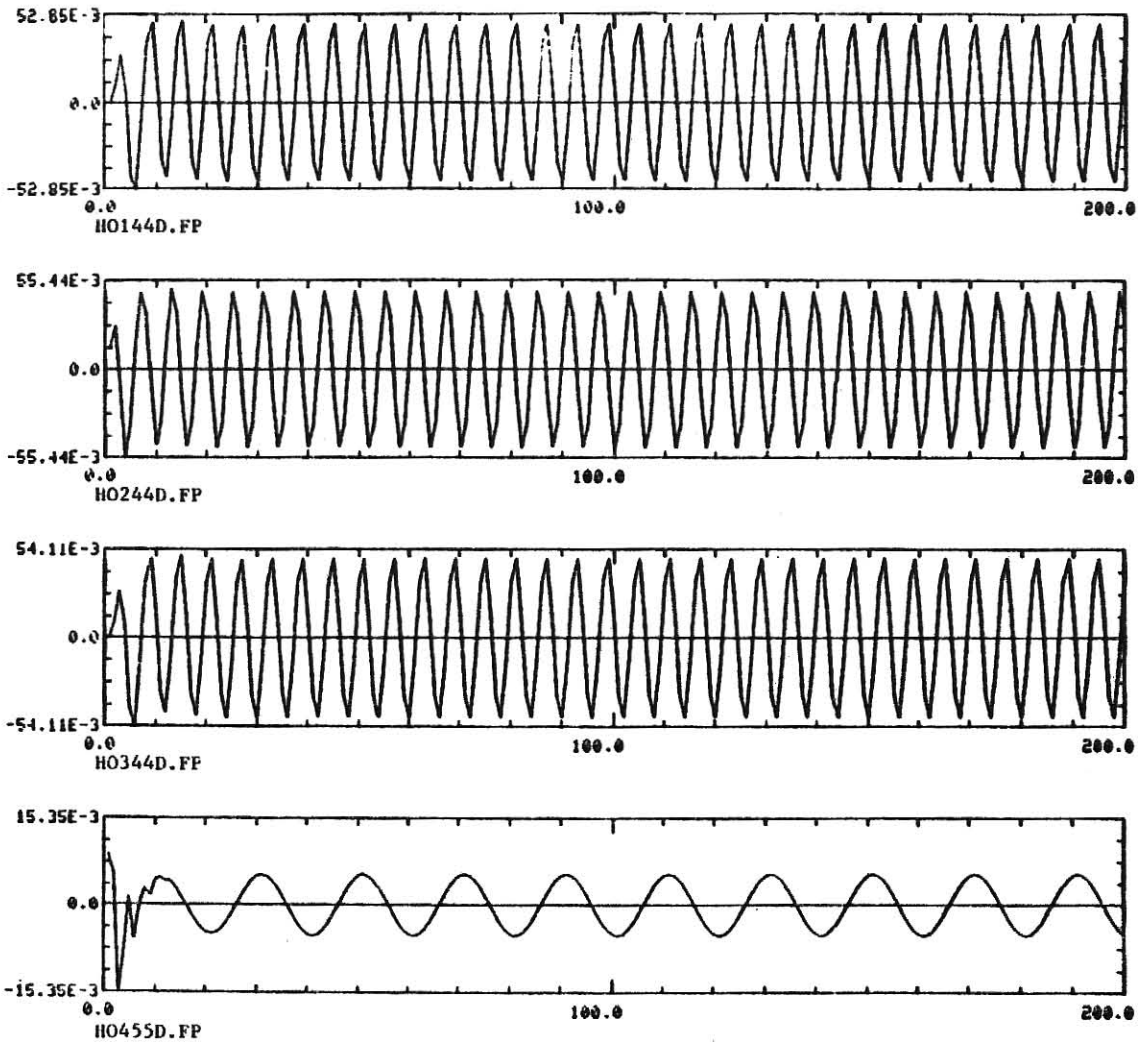
HIGH BAND: +-1.0 input without noise, fixed point



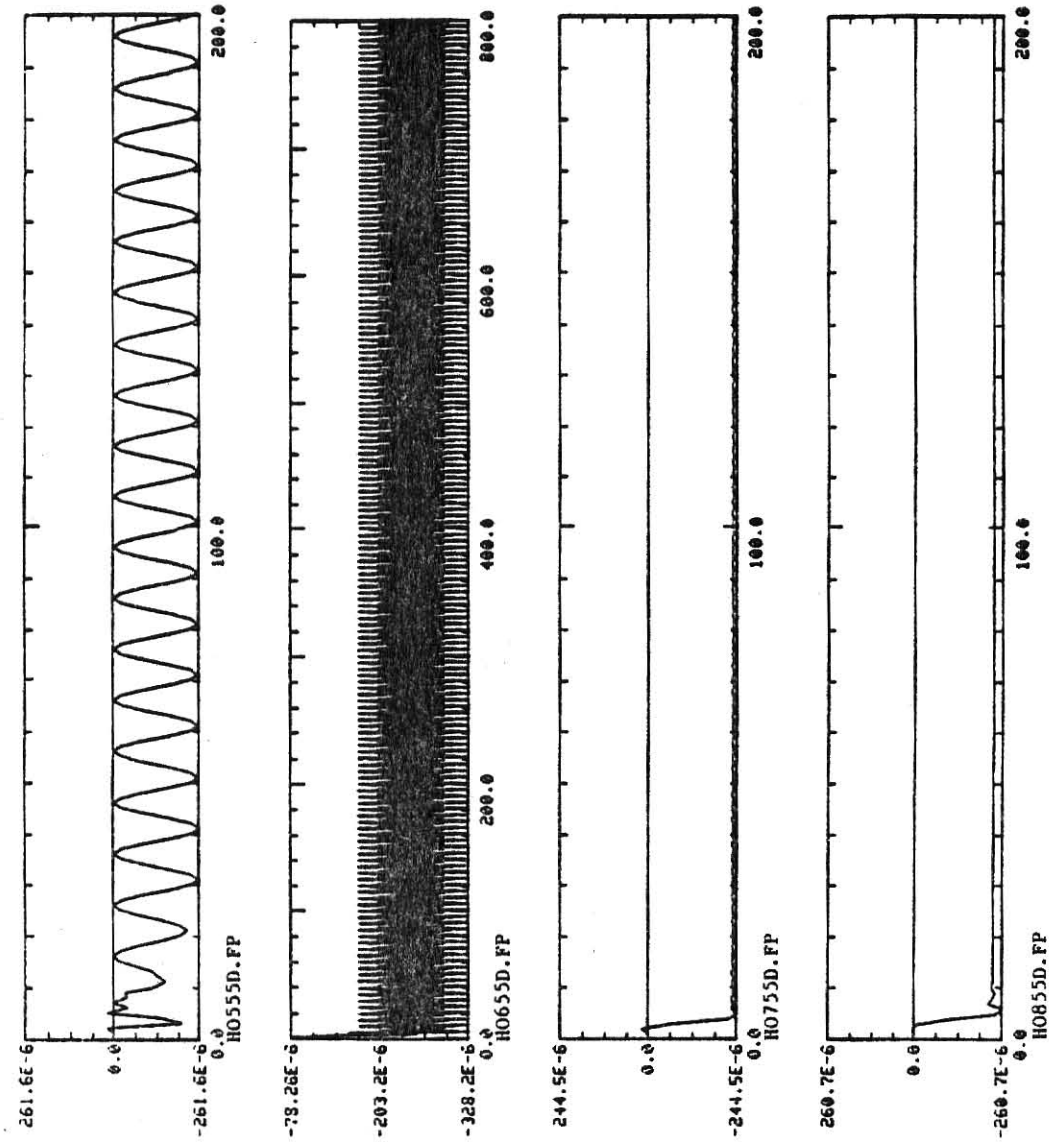
HIGH BAND: +-1.0 input without noise, fixed point



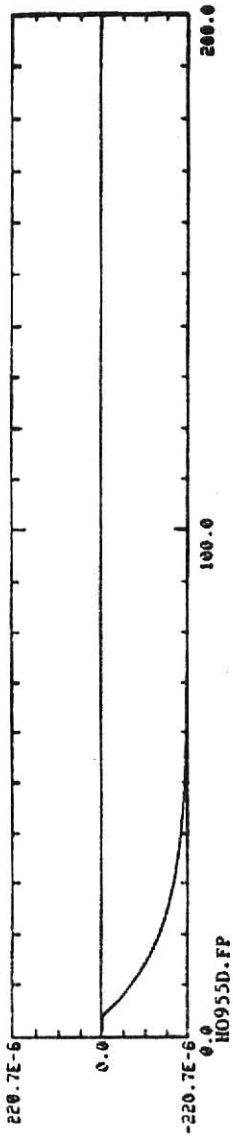
HIGH BAND: +-0.05 input without noise, double precision



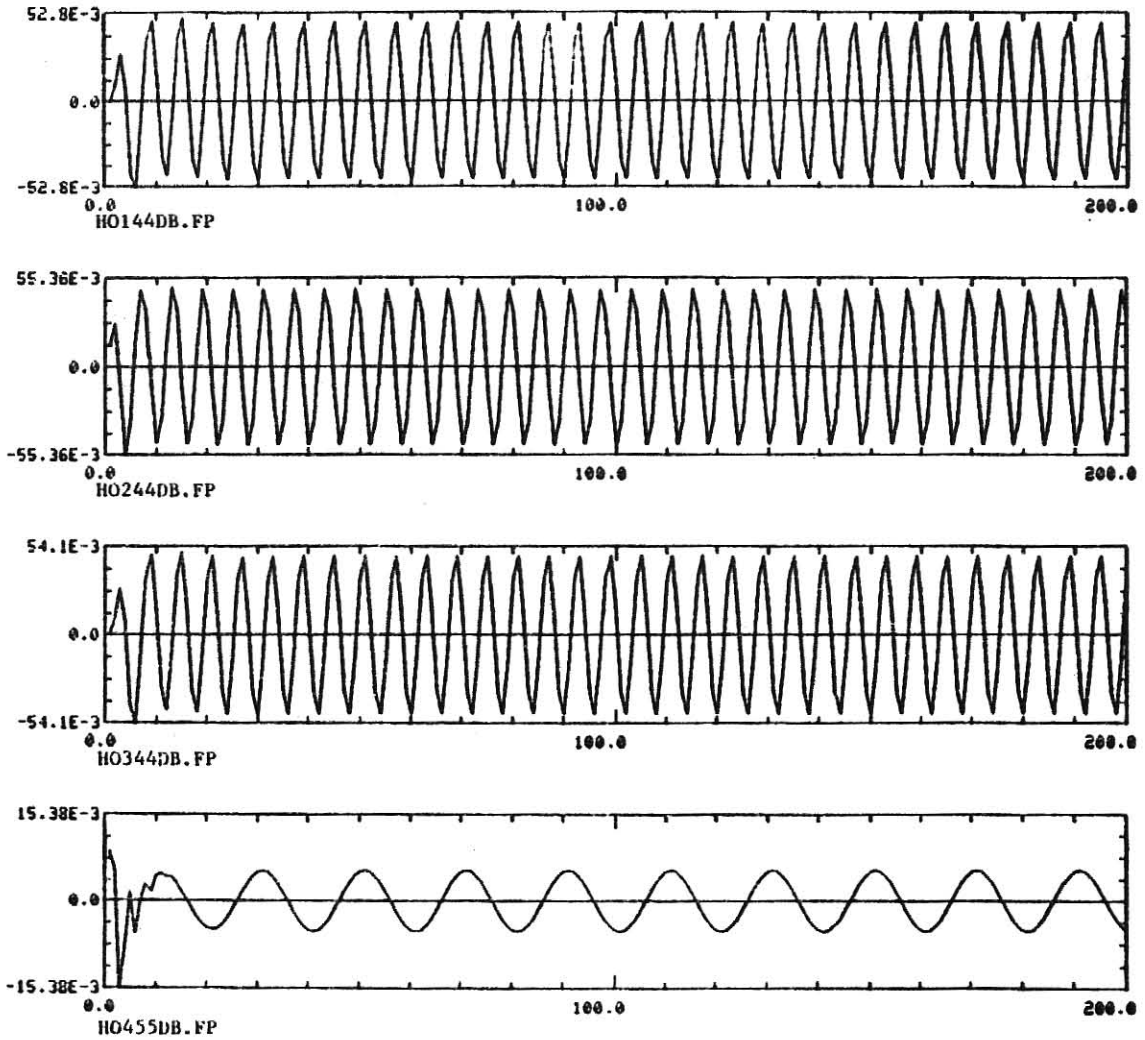
HIGH BAND: +-0.05 input without noise, double precision



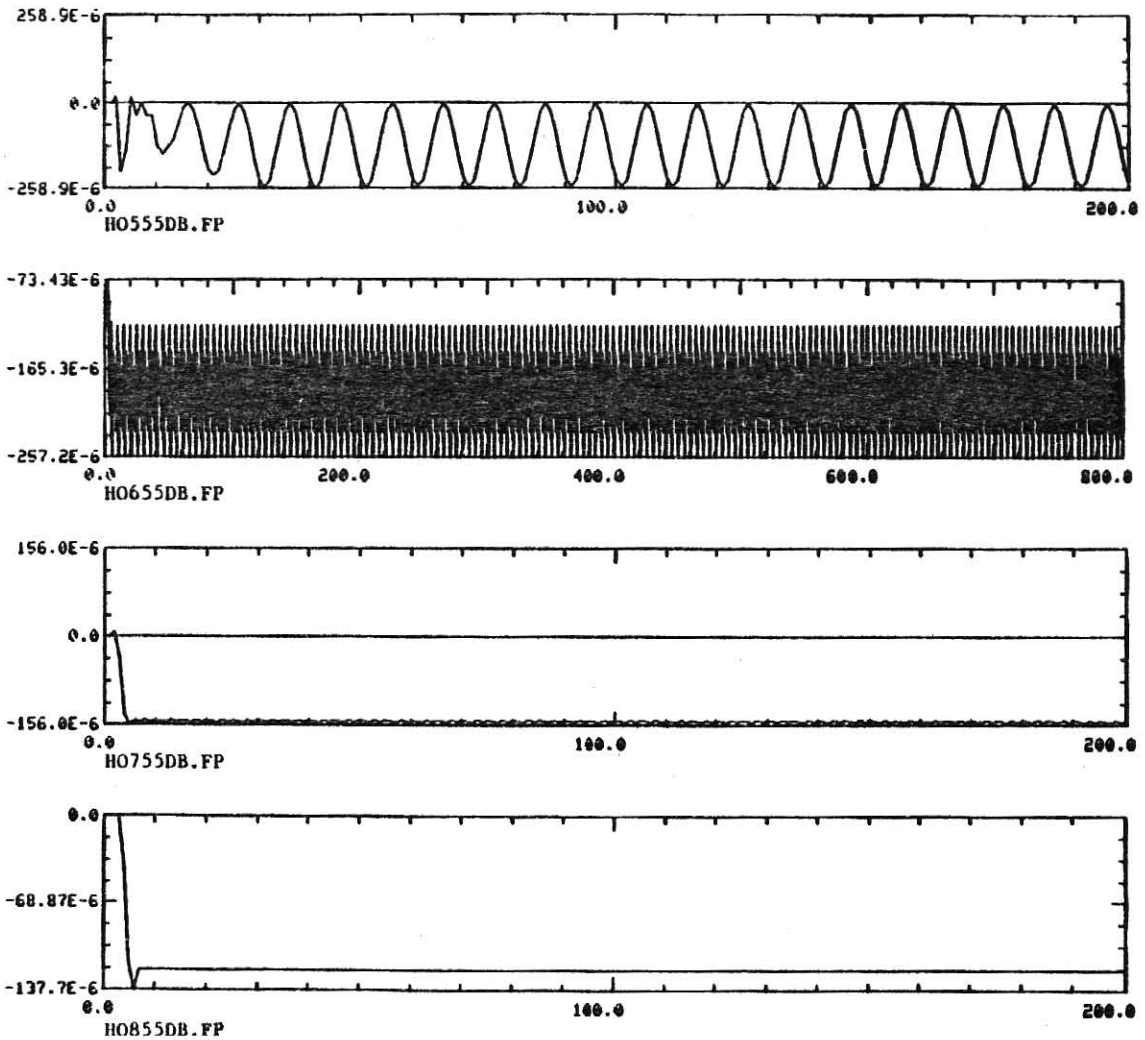
HIGH BAND: +-0.05 input without noise, double precision



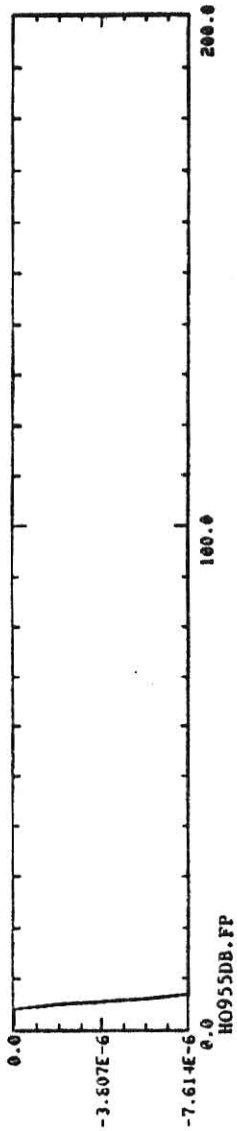
HIGH BAND: +-0.05 input without noise, fixed point



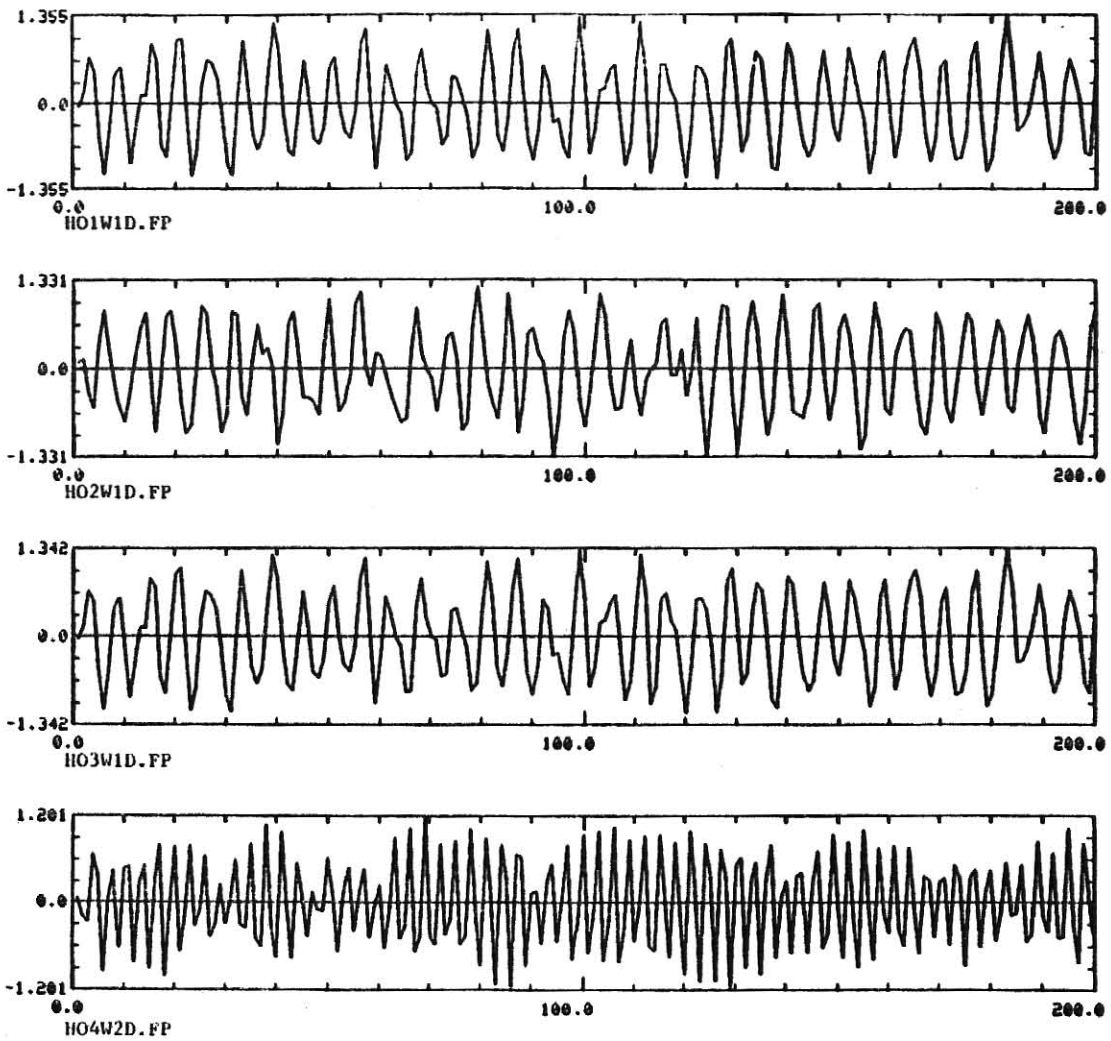
HIGH BAND: +-0.05 input without noise, fixed point



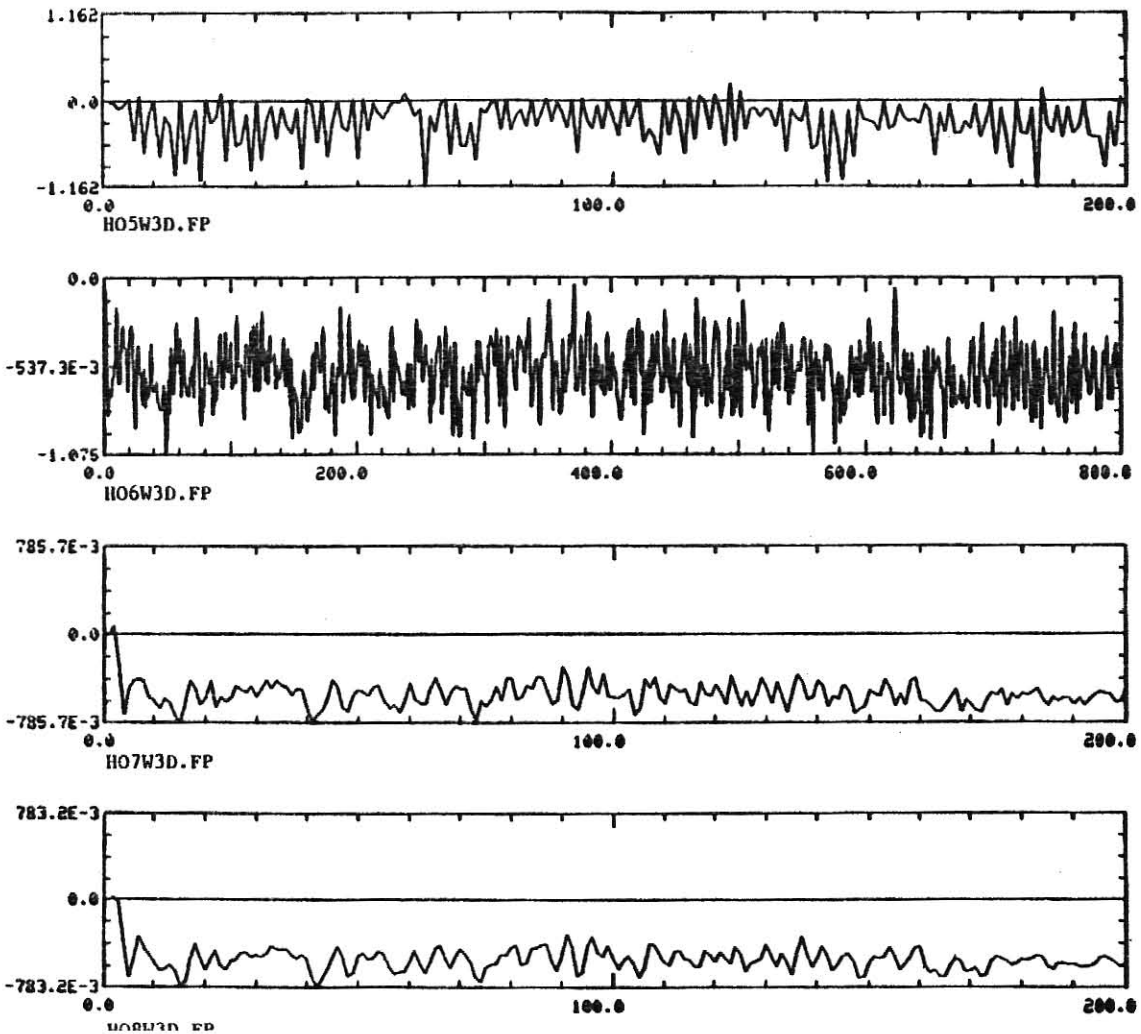
HIGH BAND: +-0.05 input without noise, fixed point



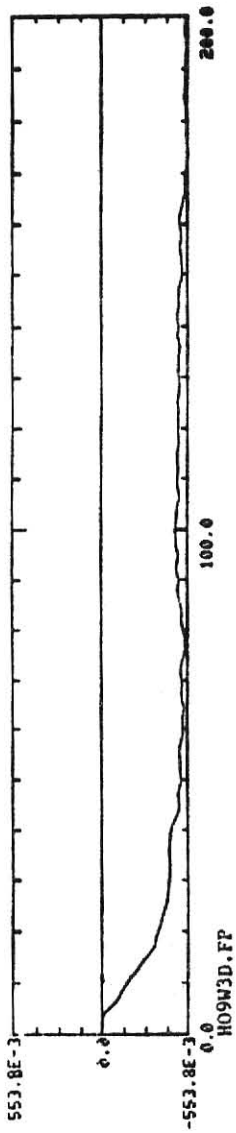
HIGH BAND: +-1.0 input with noise, double precision



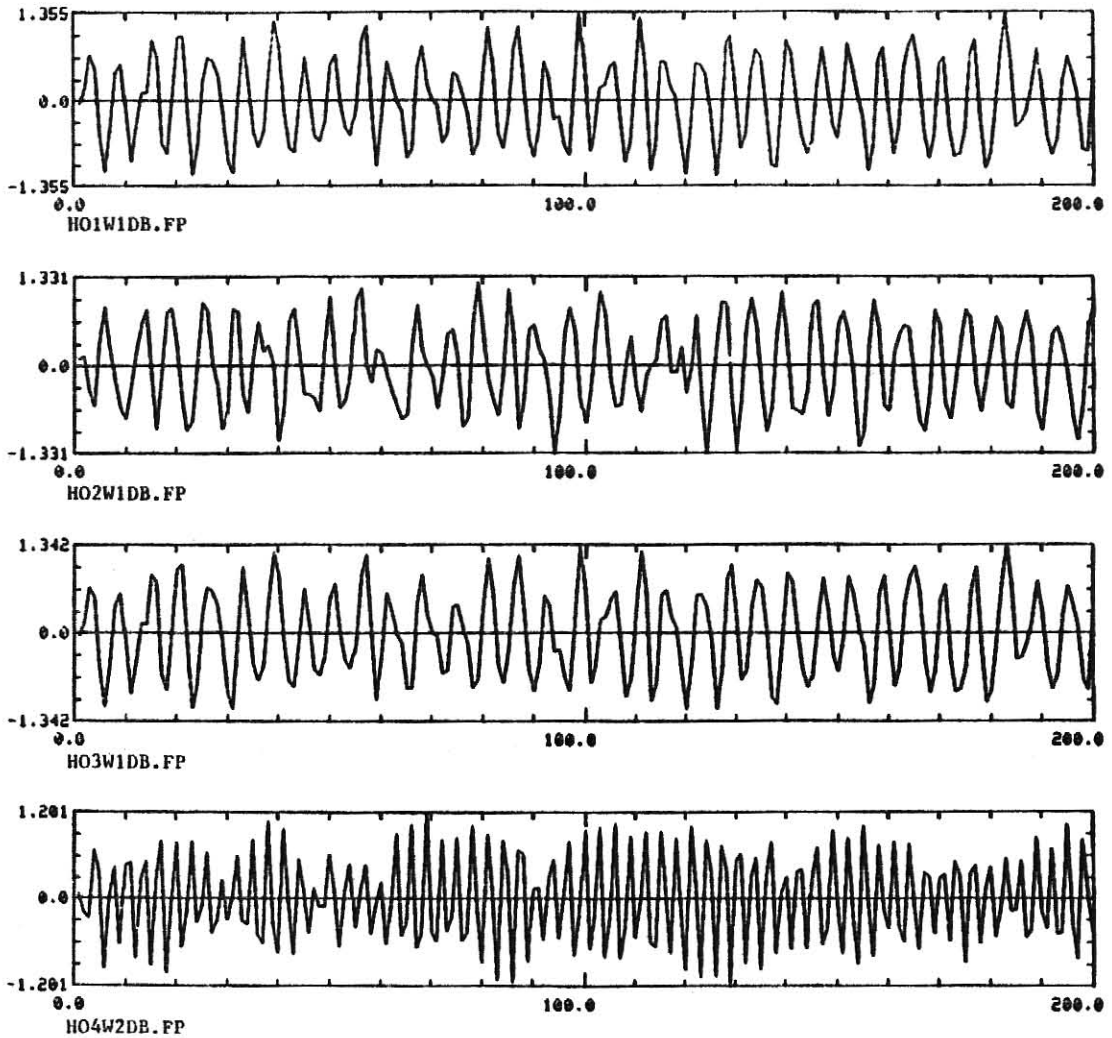
HIGH BAND: +-1.0 input with noise, double precision



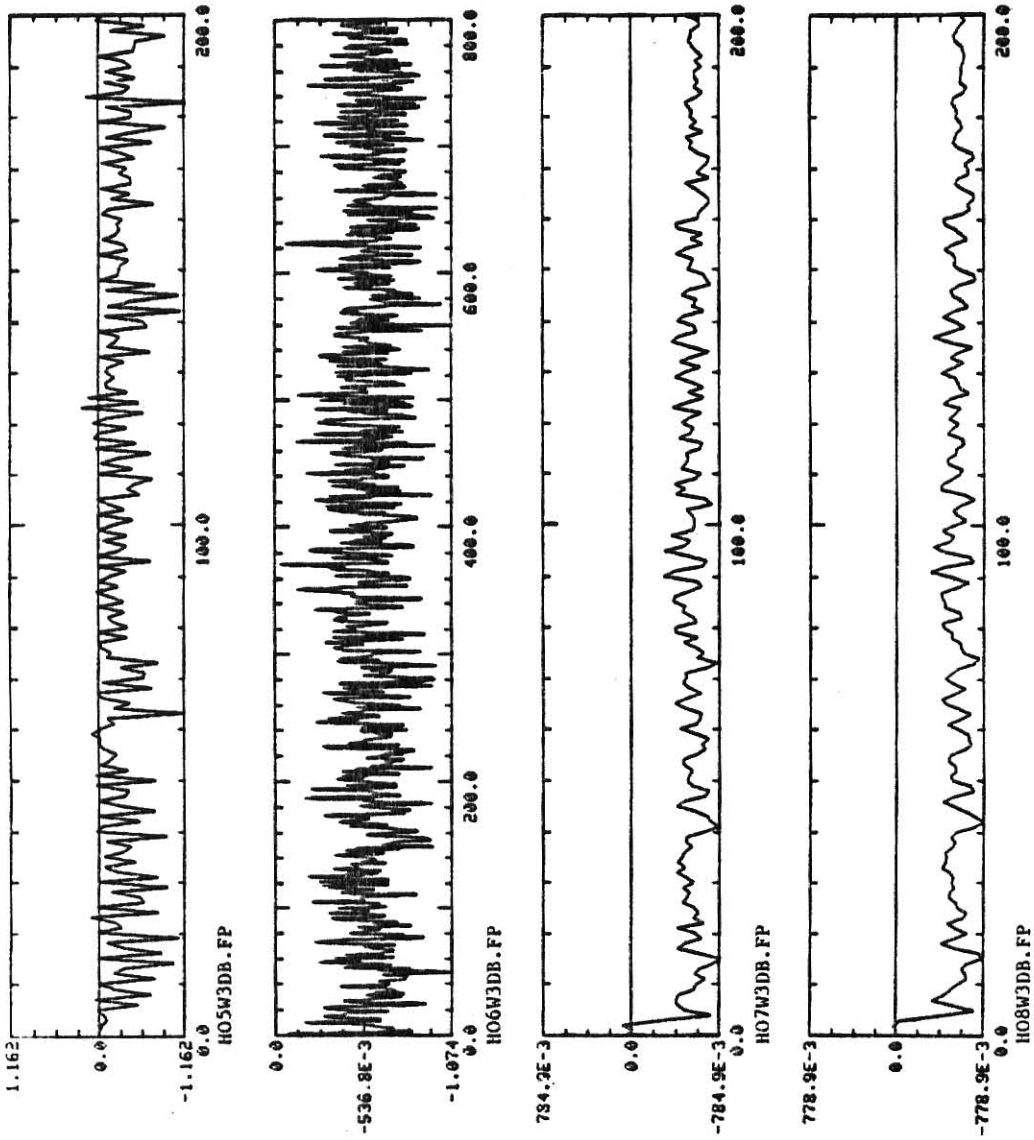
HIGH BAND: +-1.0 input with noise, double precision



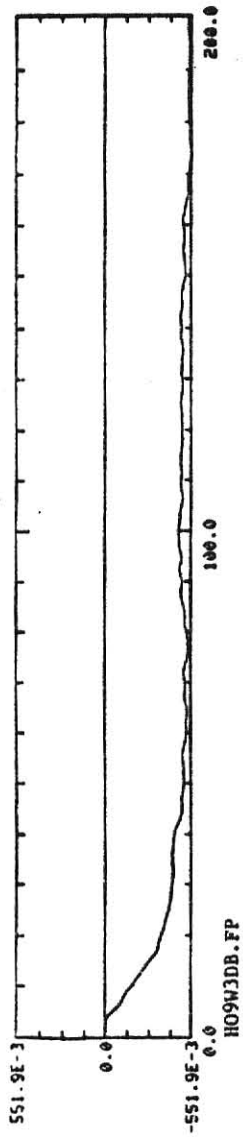
HIGH BAND: +-1.0 input with noise, fixed point



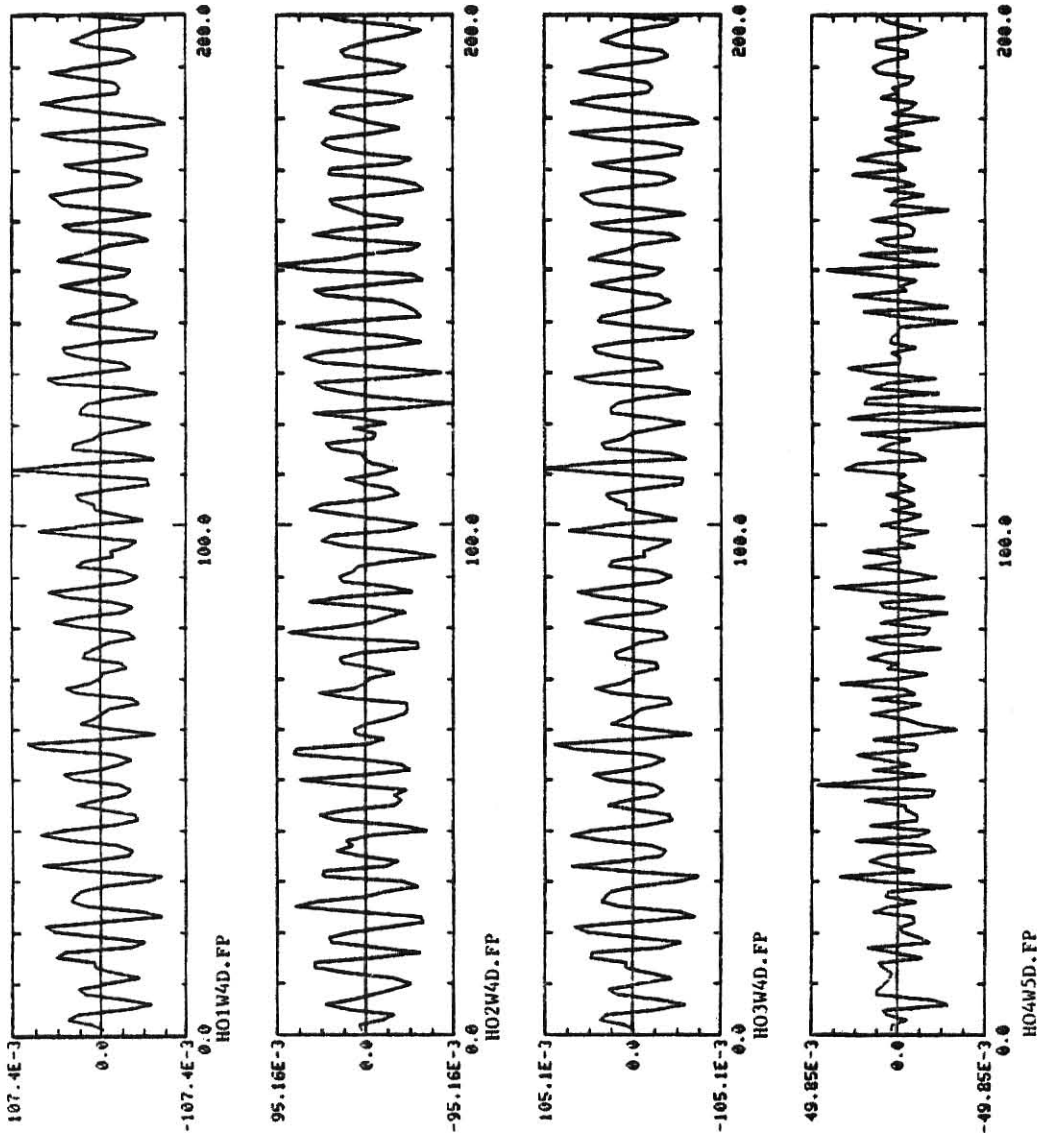
HIGH BAND: +-1.0 input with noise, fixed point



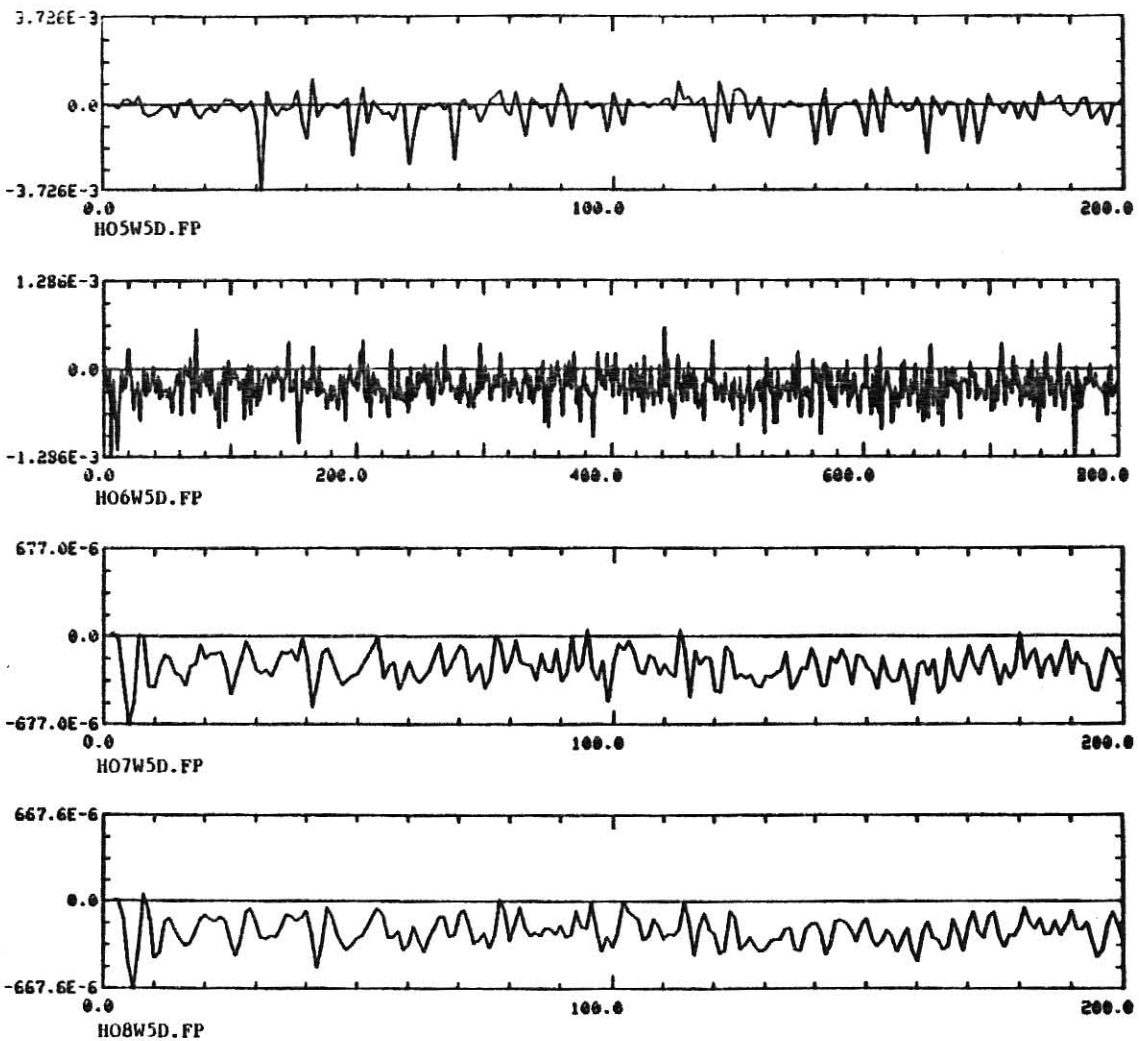
HIGH BAND: +-1.0 input with noise, fixed point



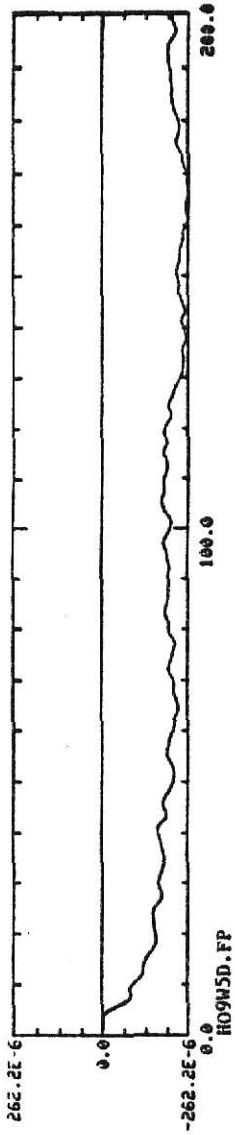
HIGH BAND: +-0.05 input with noise, double precision



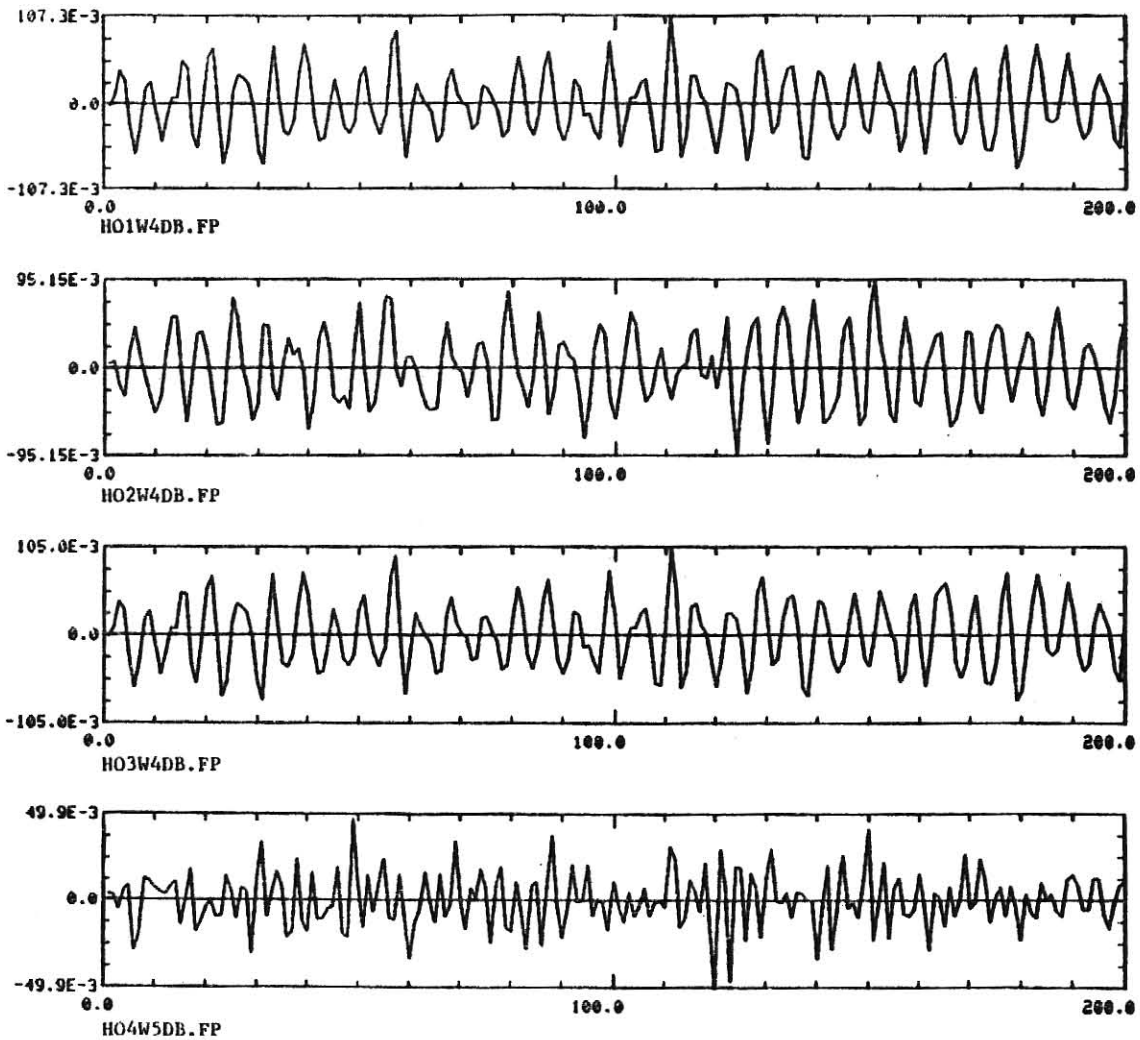
HIGH BAND: +-0.05 input with noise, double precision



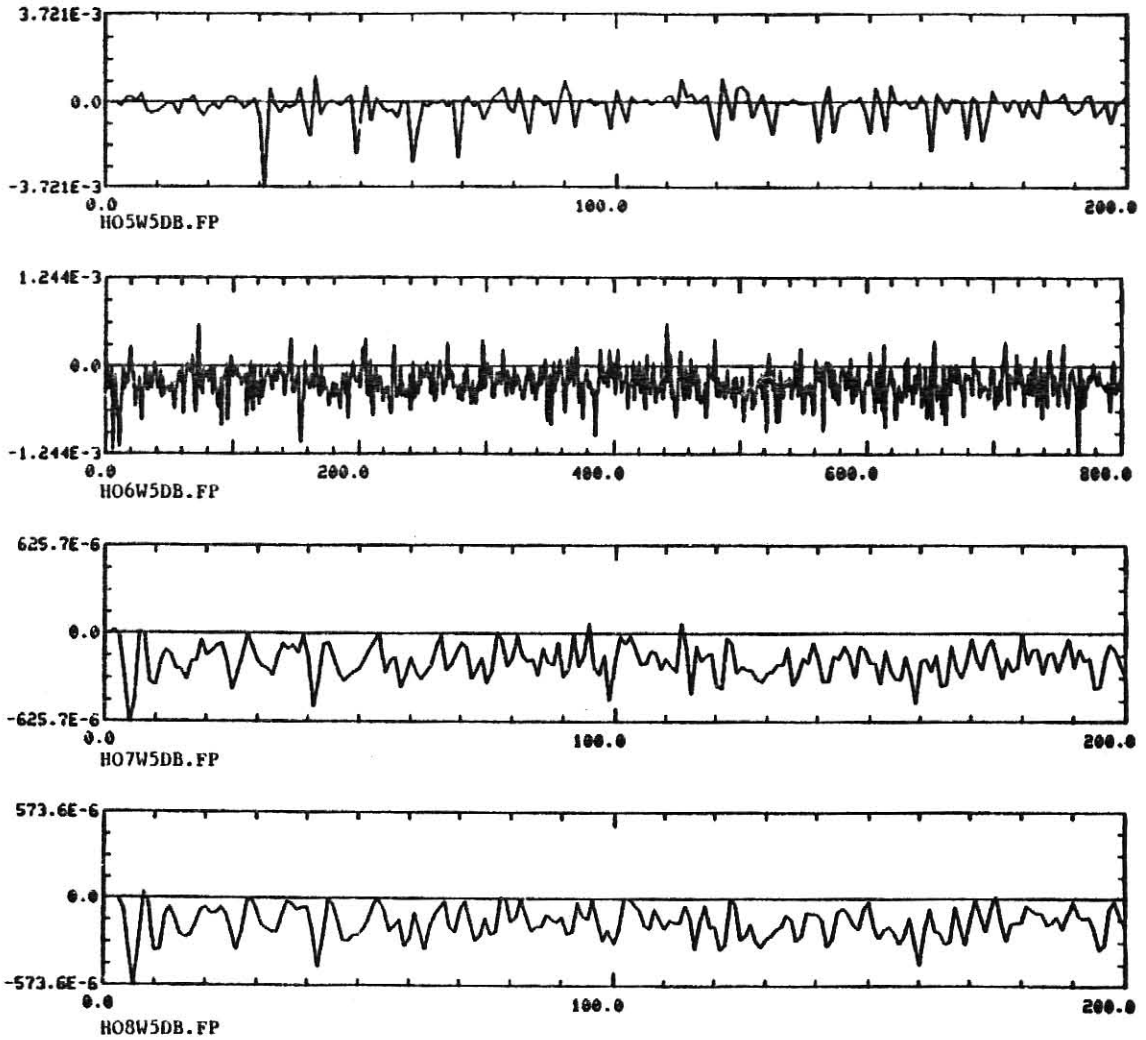
HIGH BAND: +-0.05 input with noise, double precision



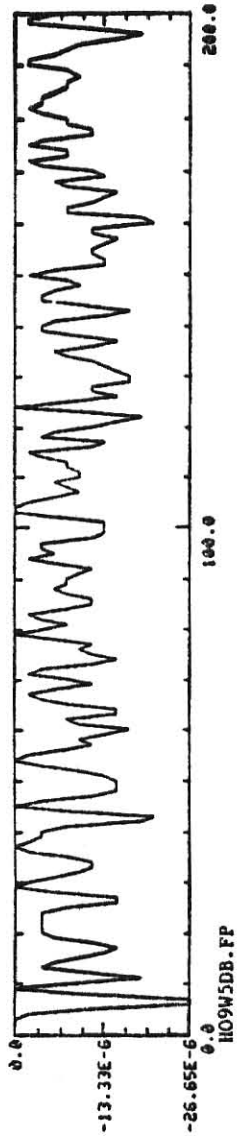
HIGH BAND: +-0.05 input with noise, fixed point



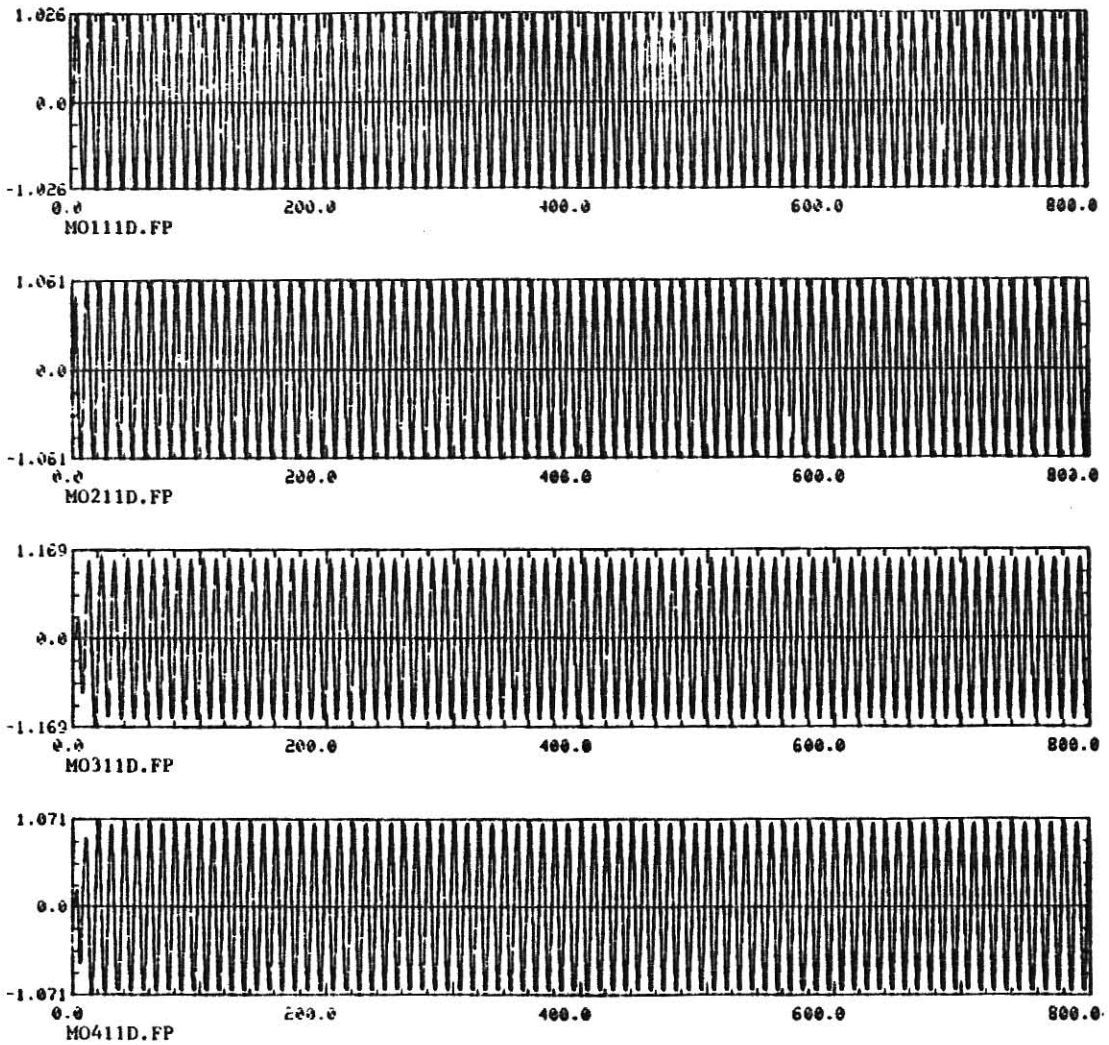
HIGH BAND: +-0.05 input with noise, fixed point



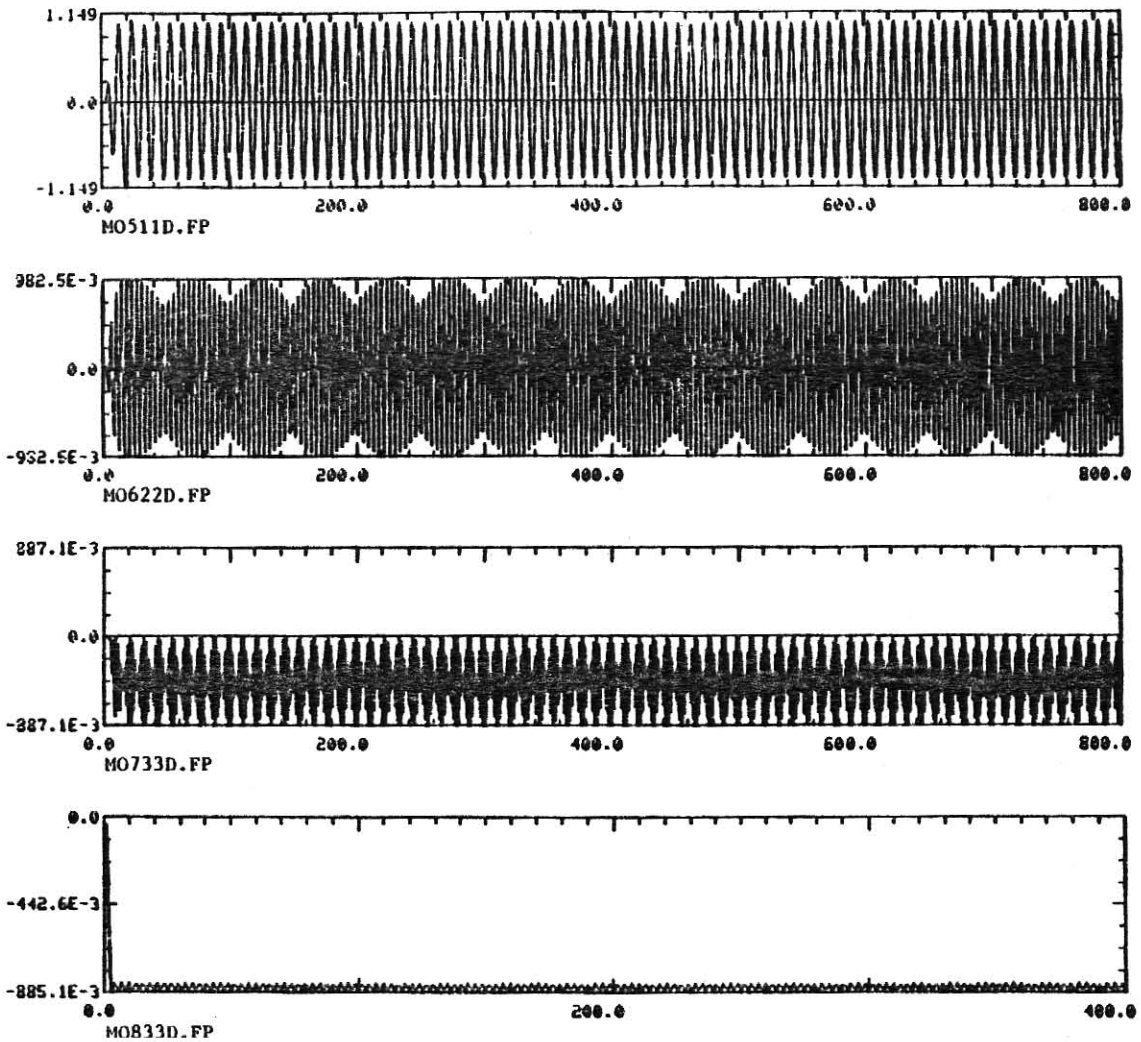
HIGH BAND: +-0.05 input with noise, fixed point



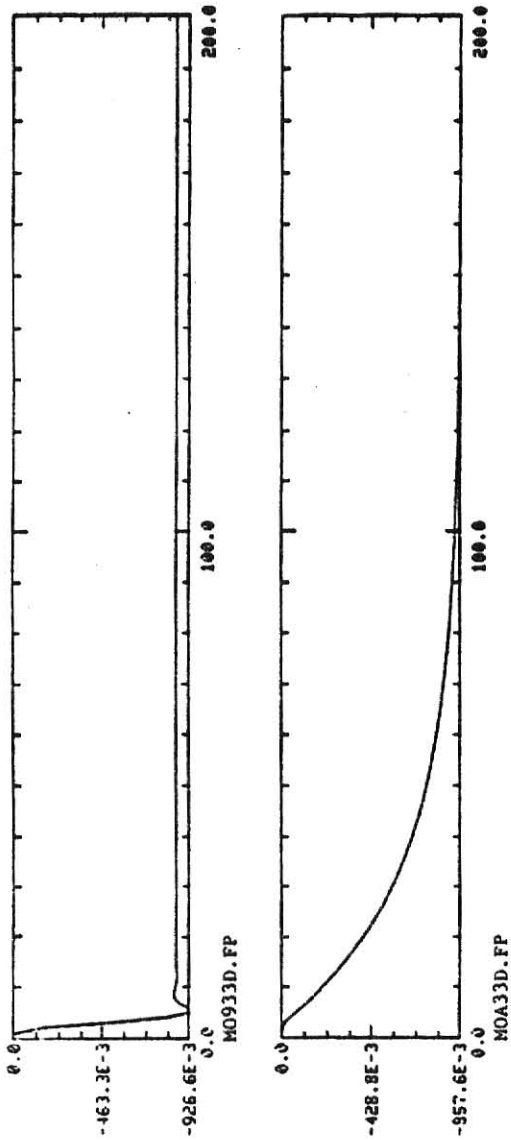
MID BAND: +-1.0 input without noise, double precision



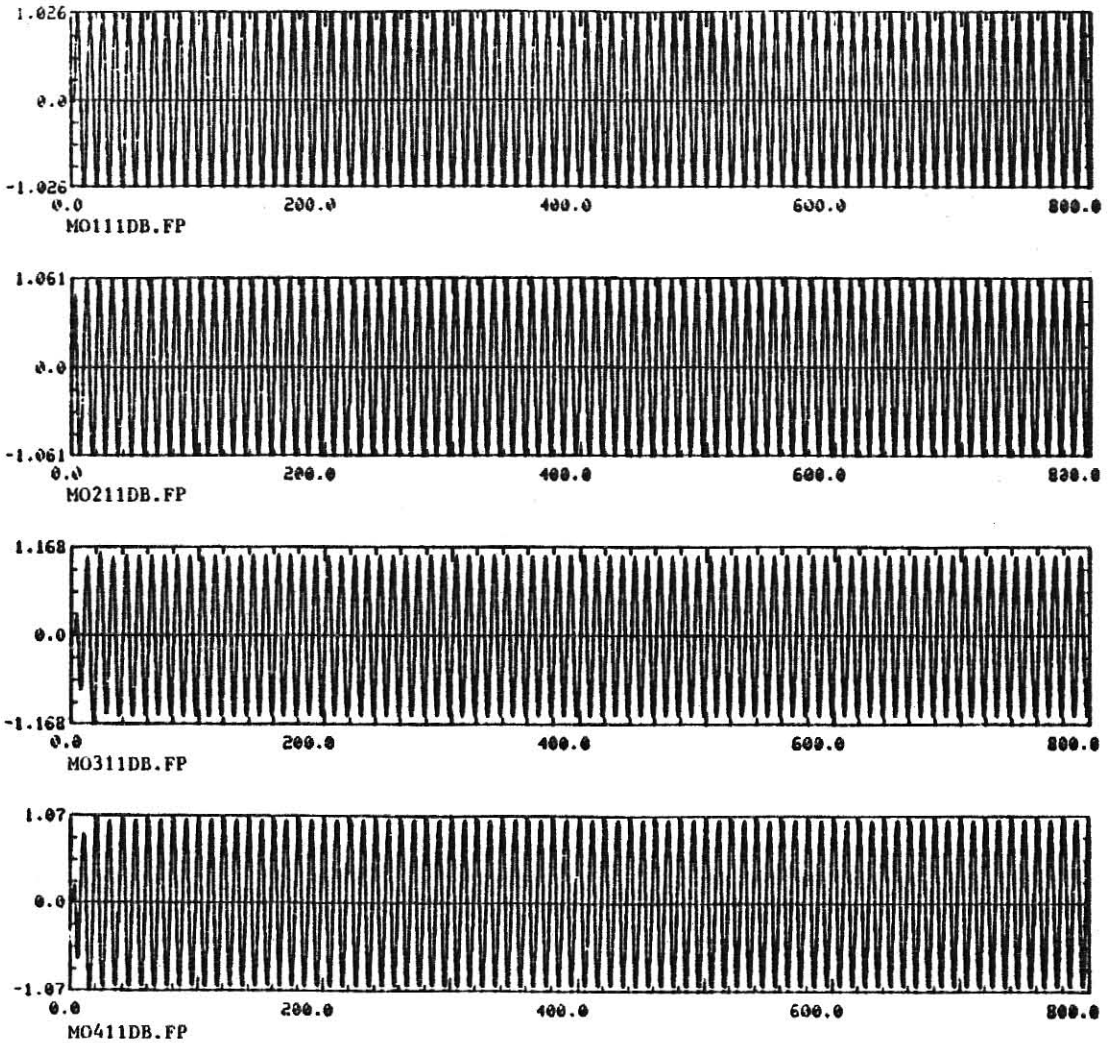
MID BAND: +-1.0 input without noise, double precision



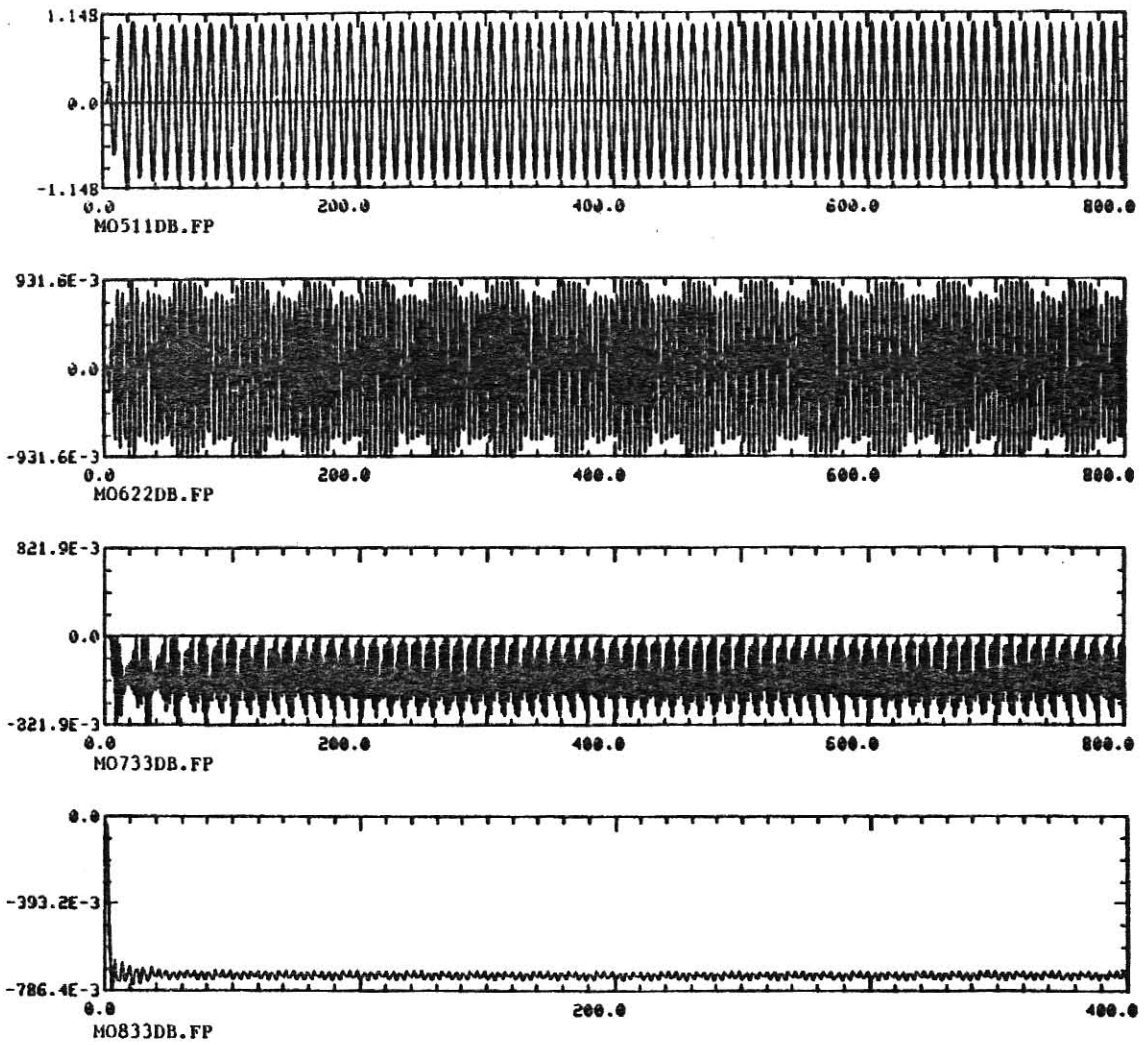
MID BAND: +-1.0 input without noise, double precision



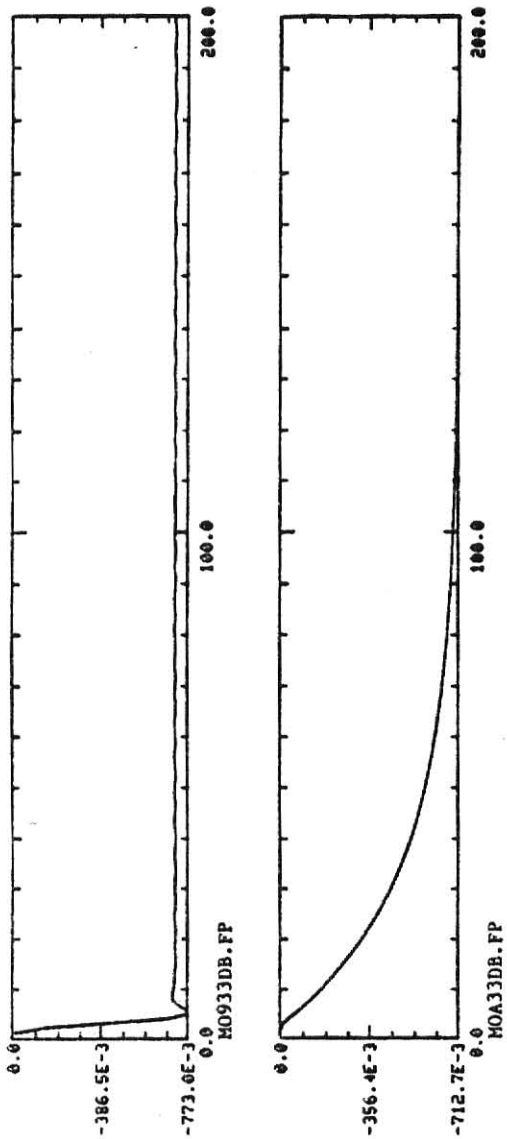
MID BAND: +-1.0 input without noise, fixed point



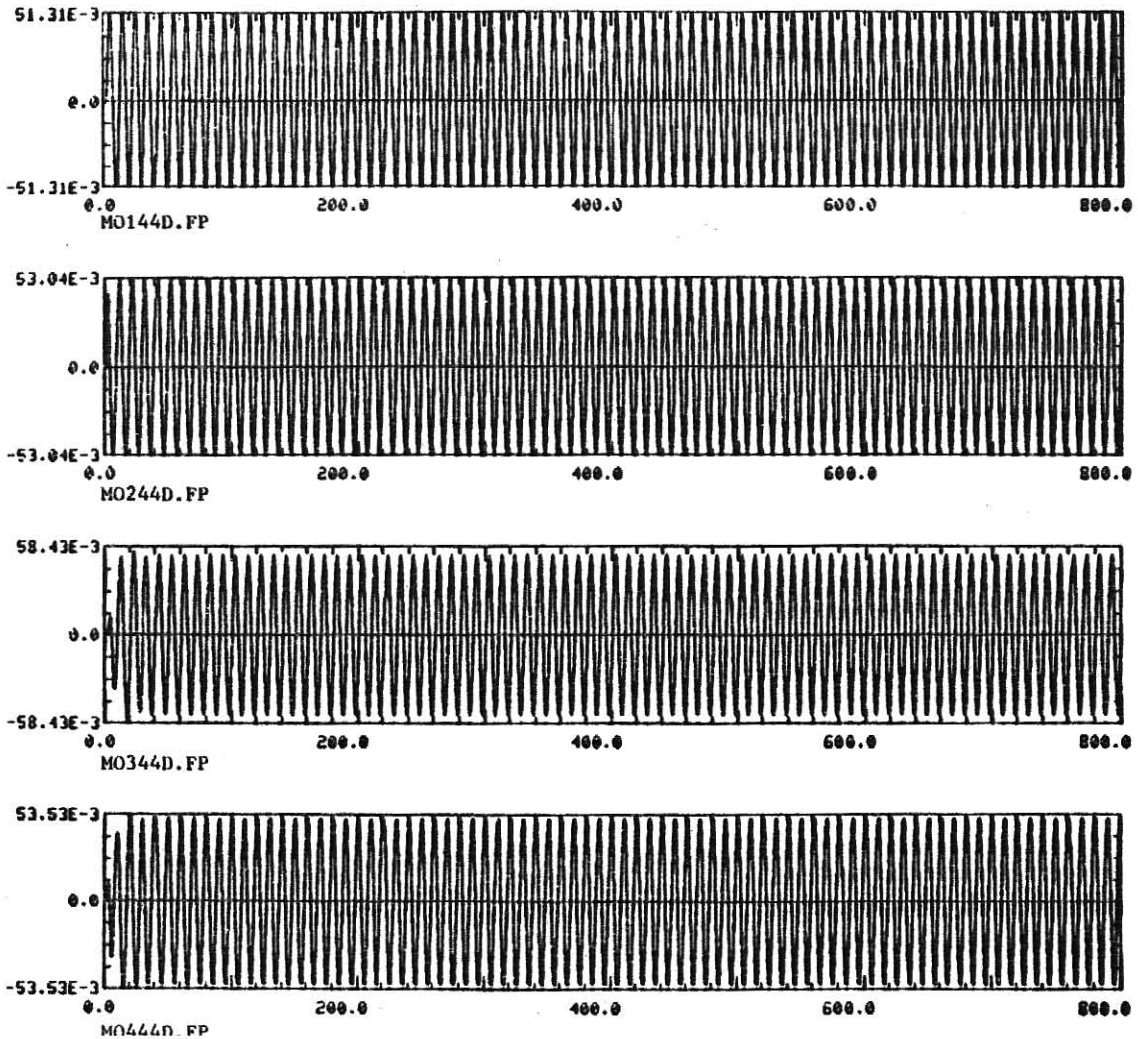
MID BAND: +-1.0 input without noise, fixed point



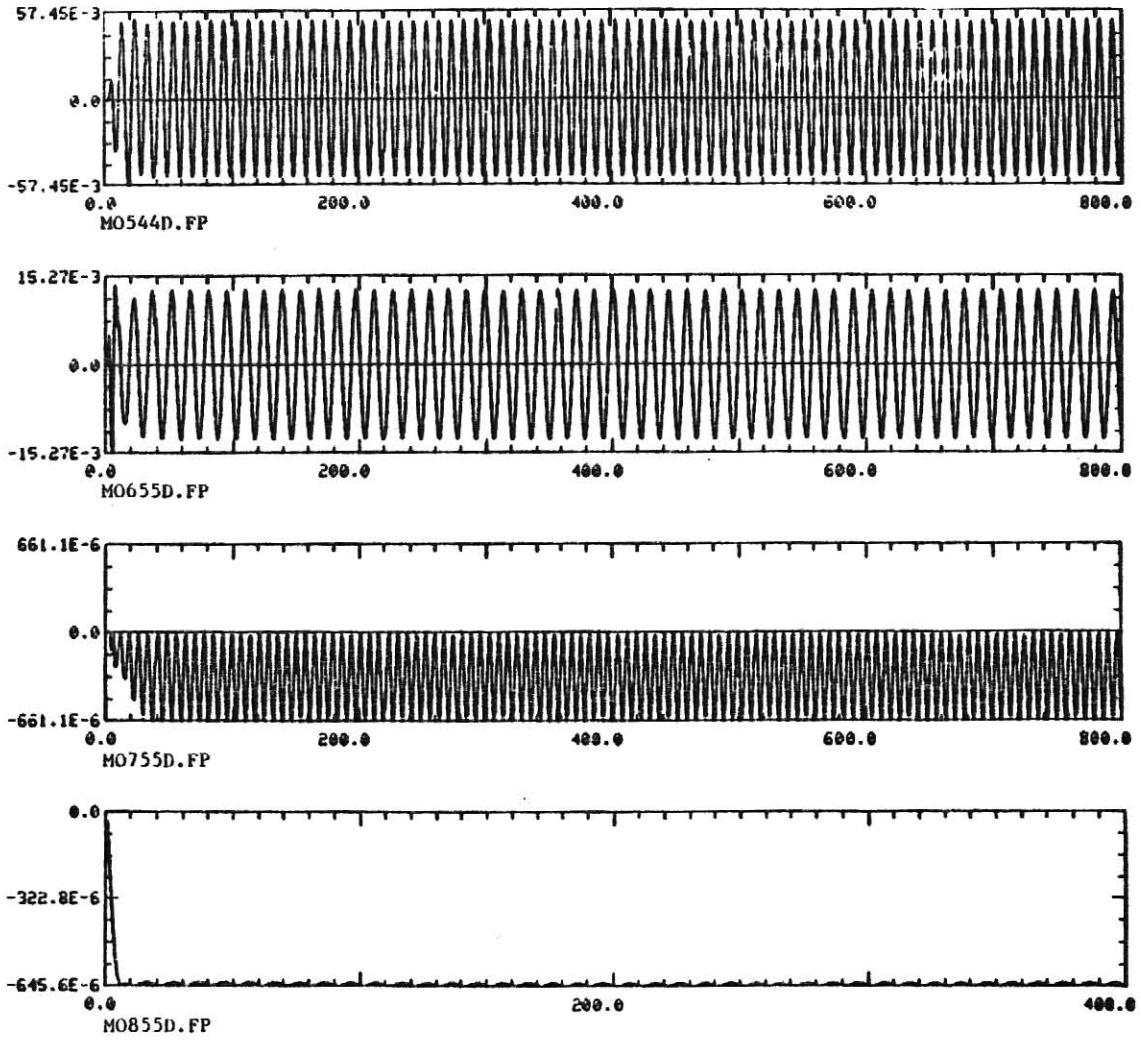
MID BAND: +-1.0 input without noise, fixed point



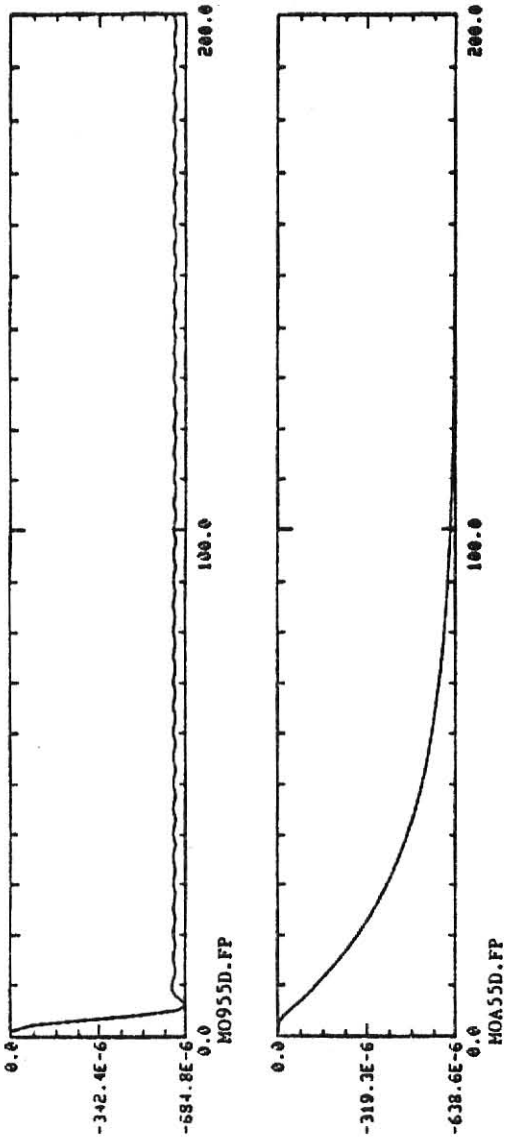
MID BAND: +-0.05 input without noise, double precision



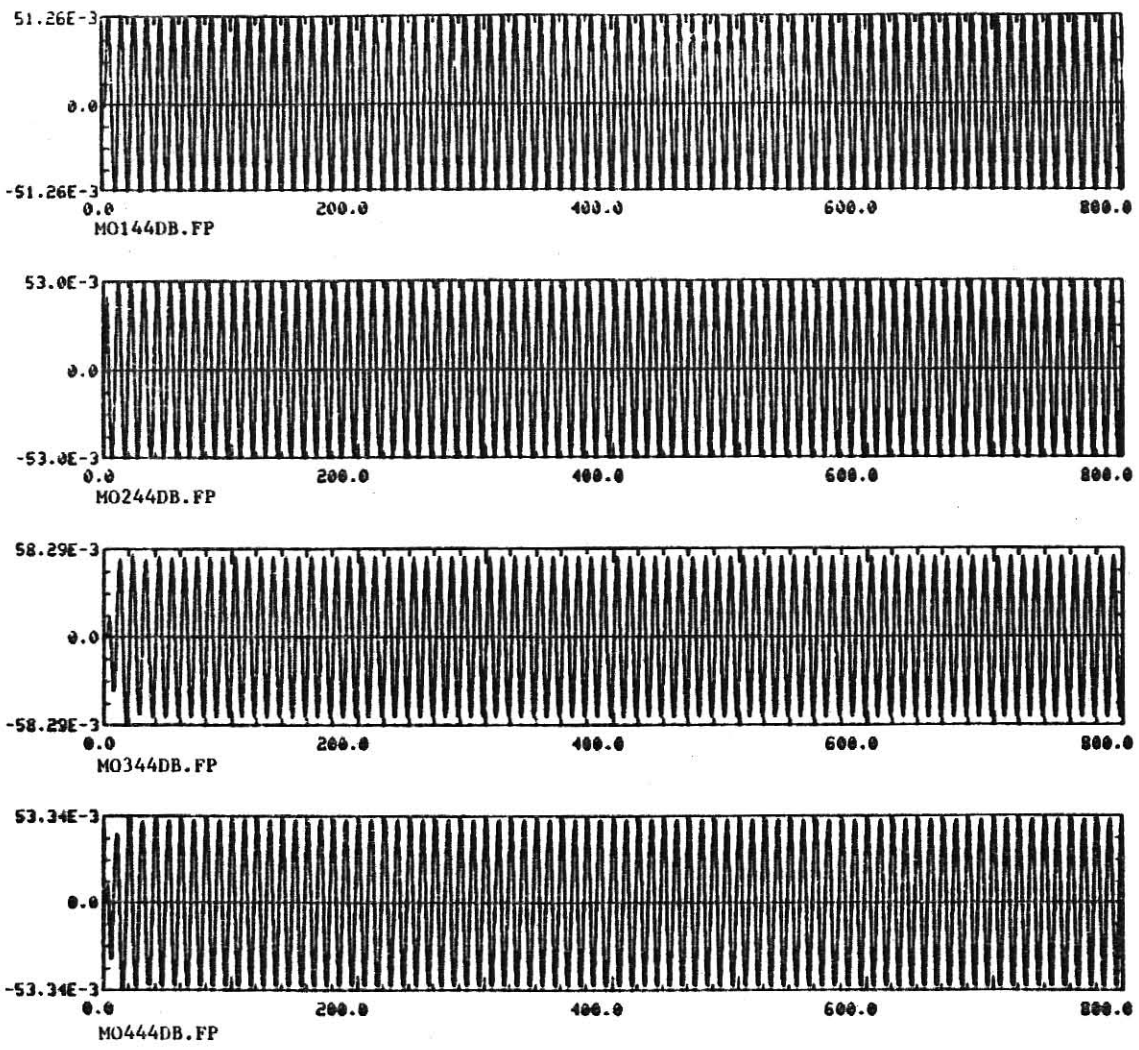
MID BAND: +-0.05 input without noise, double precision



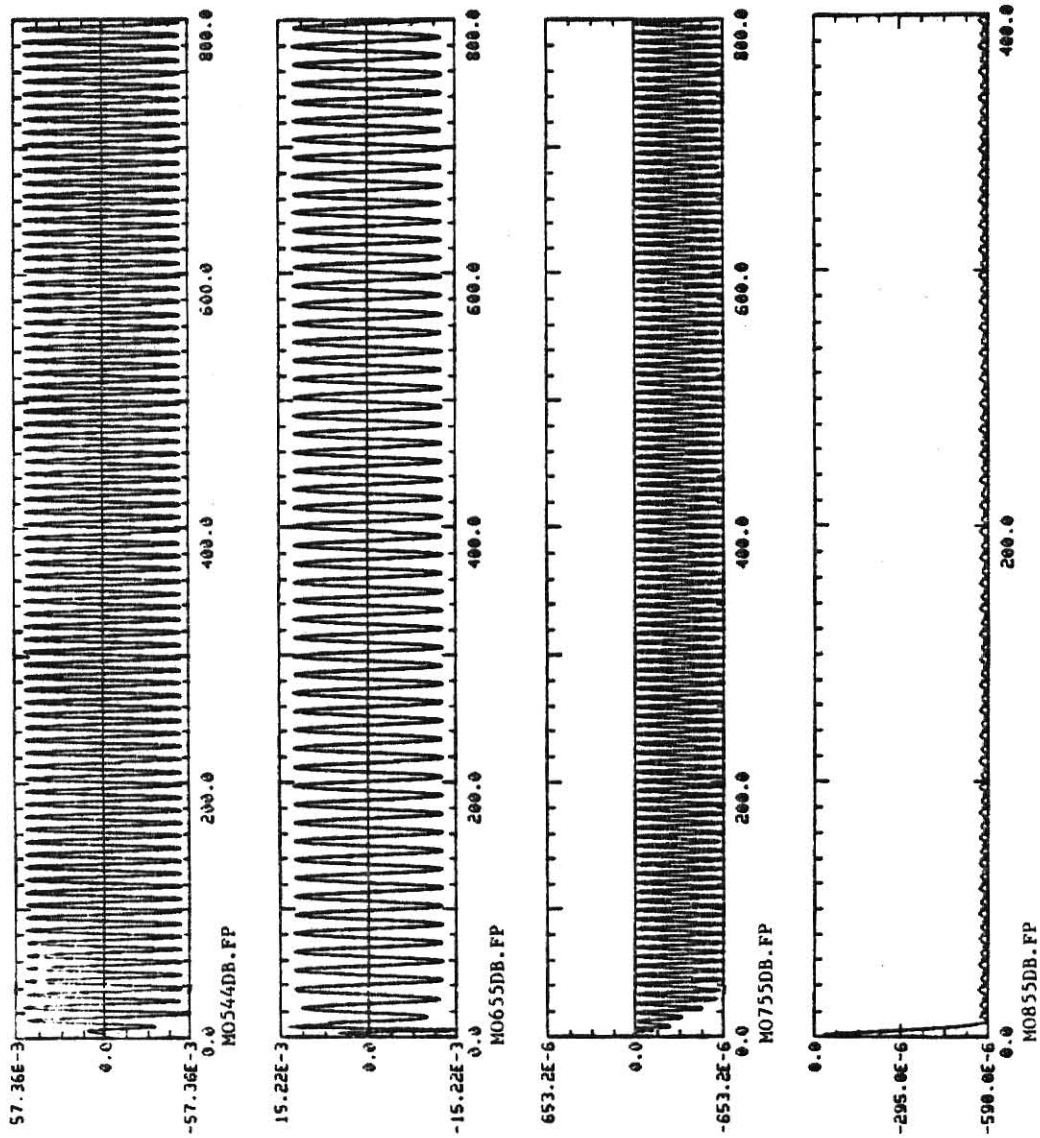
MID BAND: +-0.05 input without noise, double precision



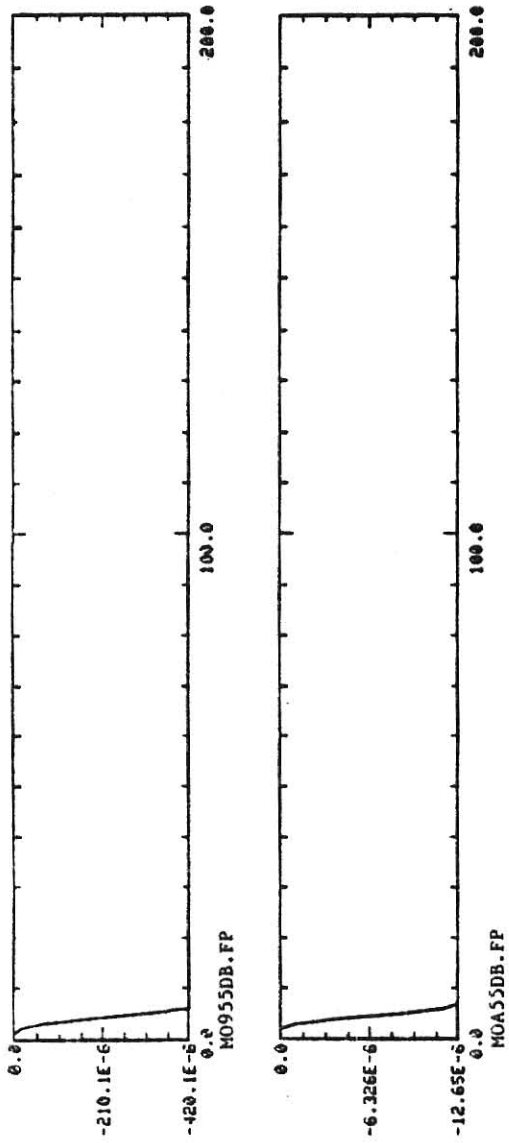
MID BAND: +-0.05 input without noise, fixed point



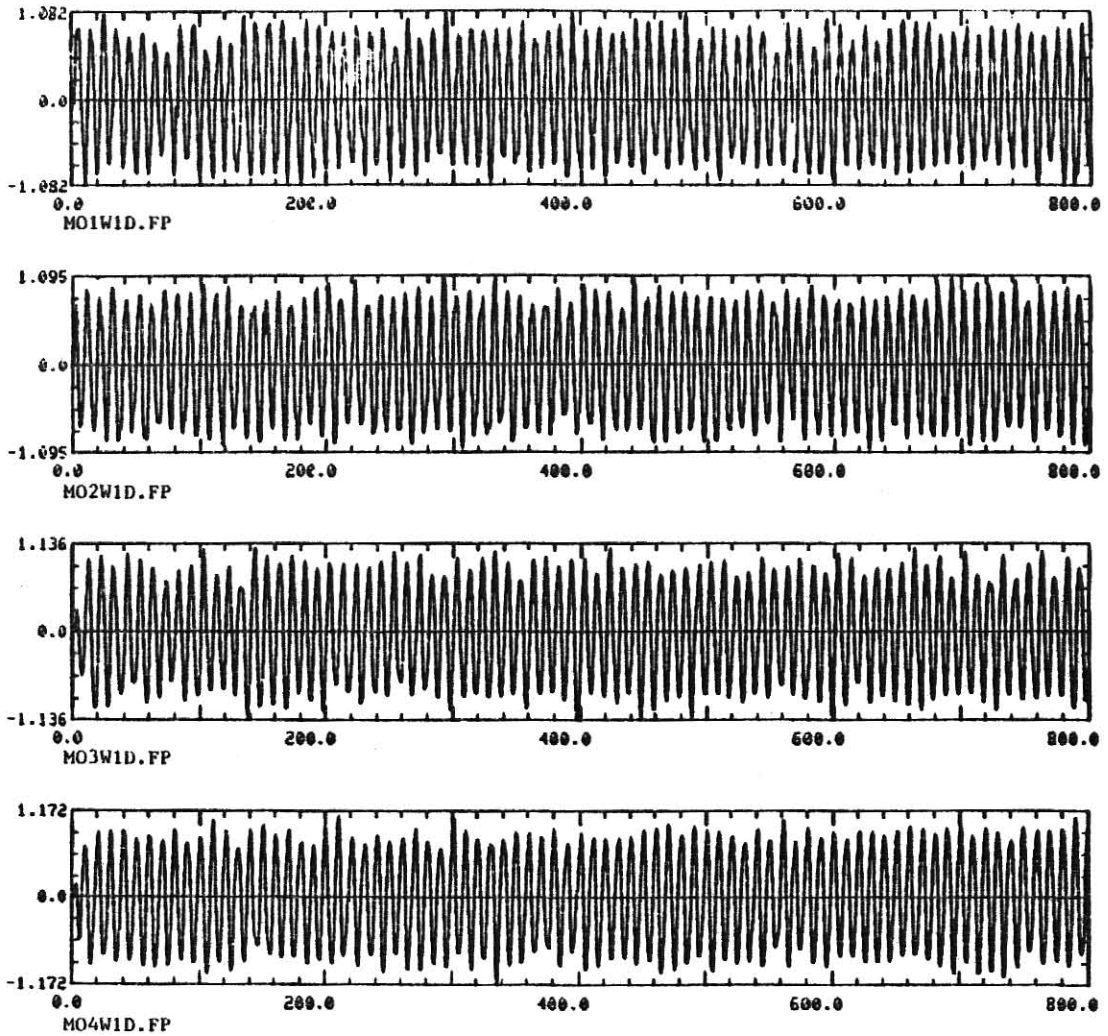
MID BAND: +-0.05 input without noise, fixed point



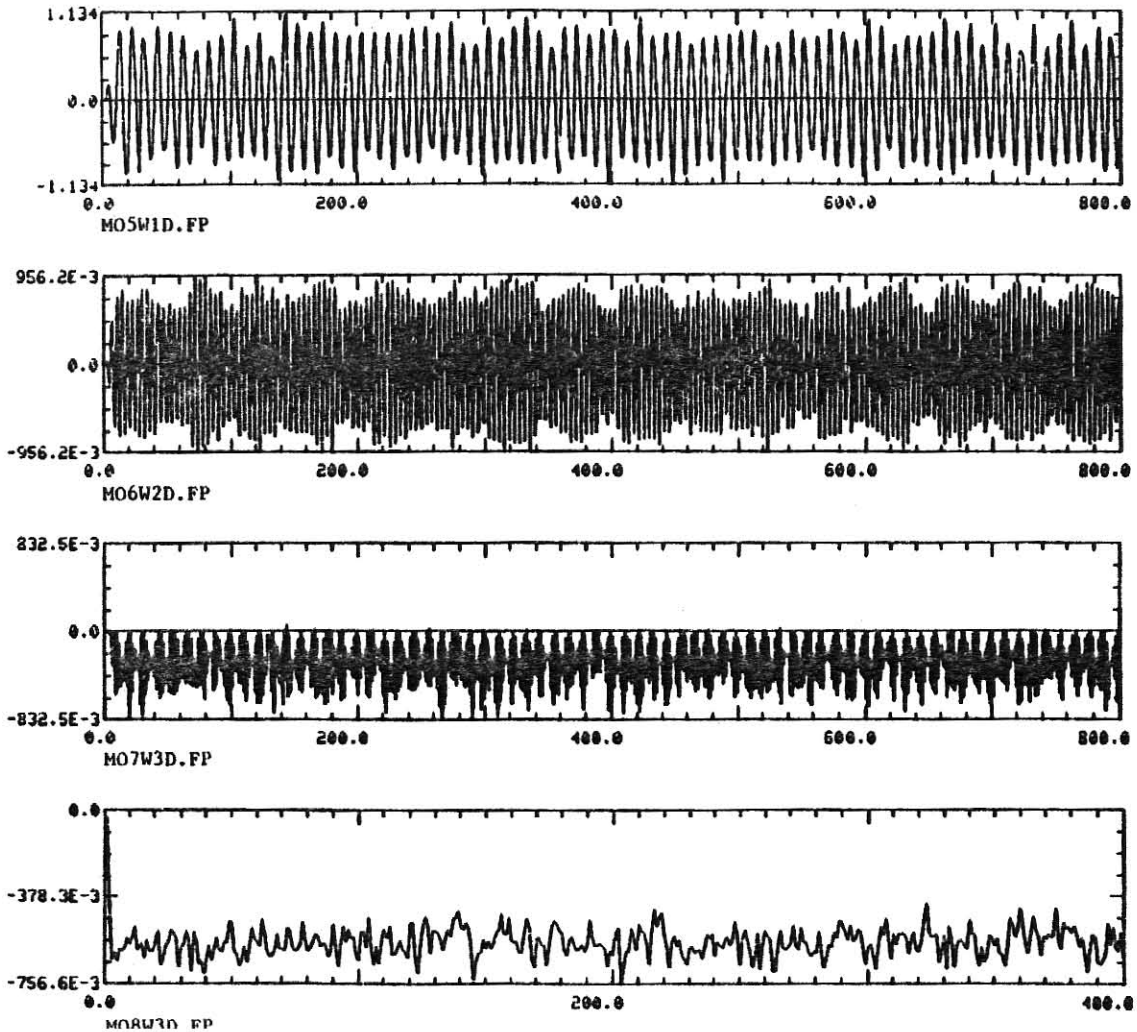
MID BAND: +-0.05 input without noise, fixed point



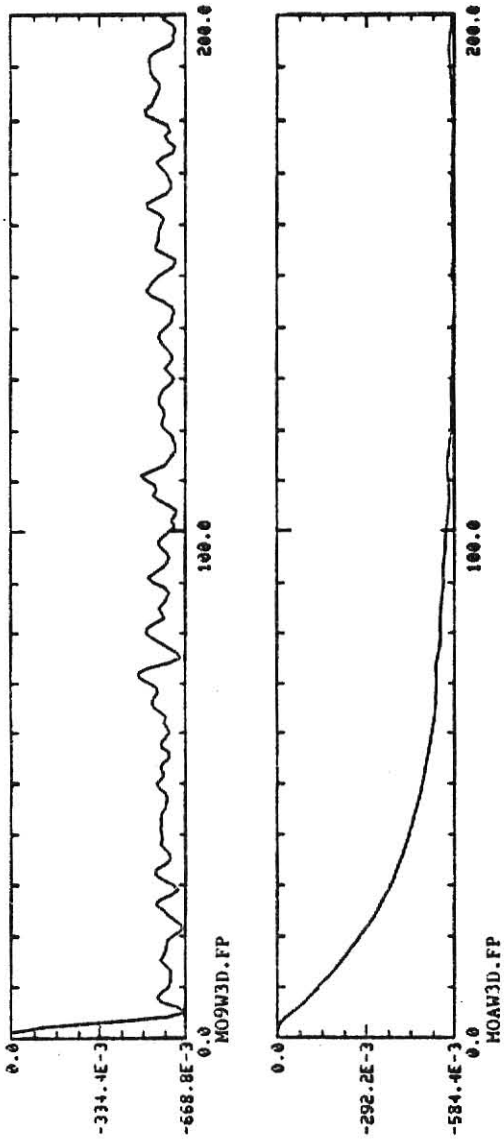
MID BAND: +-1.0 input with noise, double precision



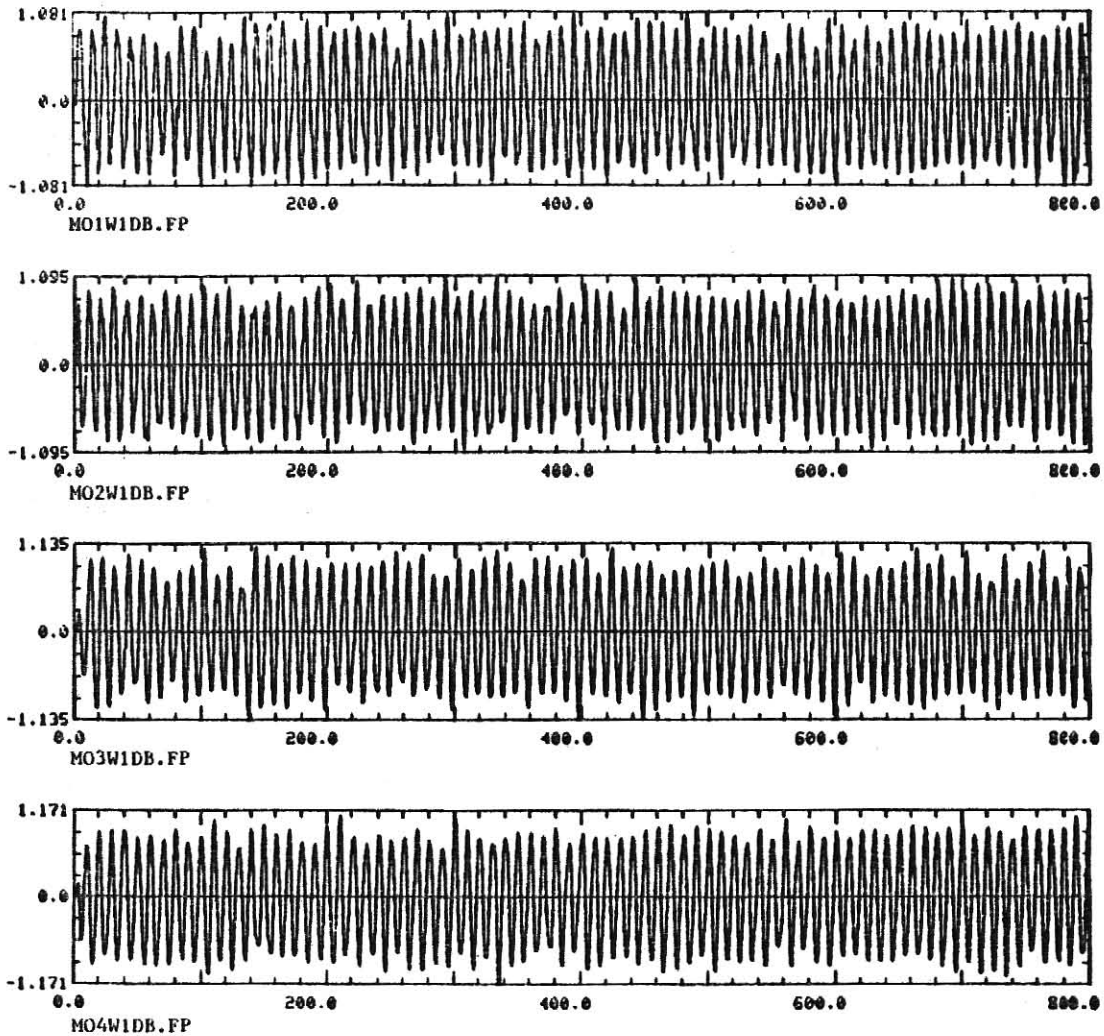
MID BAND: +-1.0 input with noise, double precision



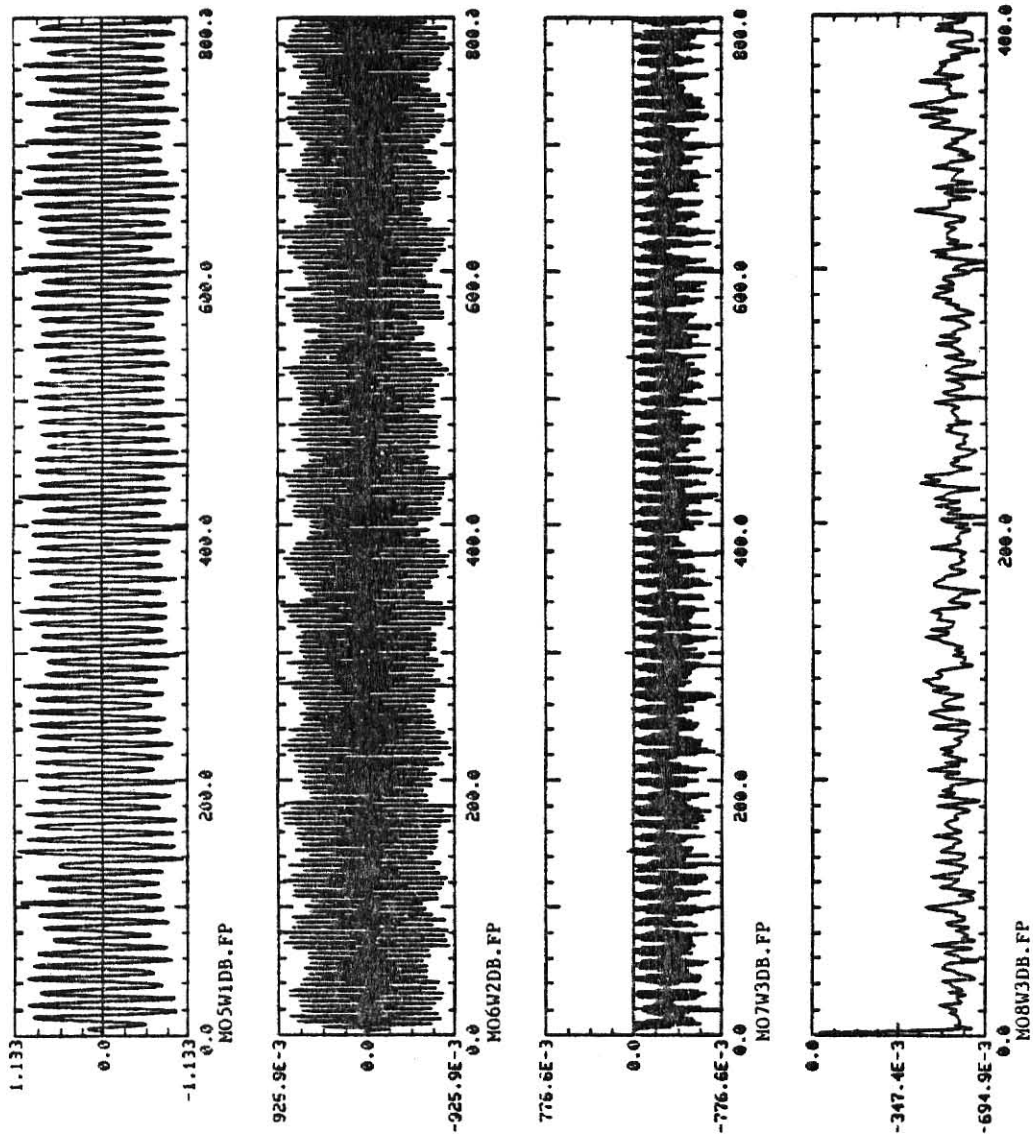
MID BAND: +-1.0 input with noise, double precision



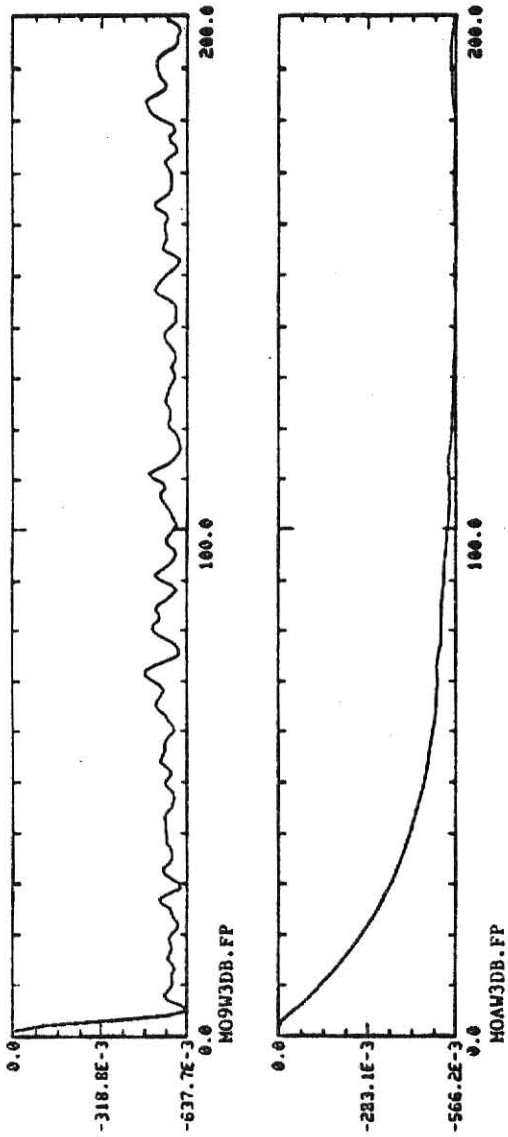
MID BAND: +-1.0 input with noise, fixed point



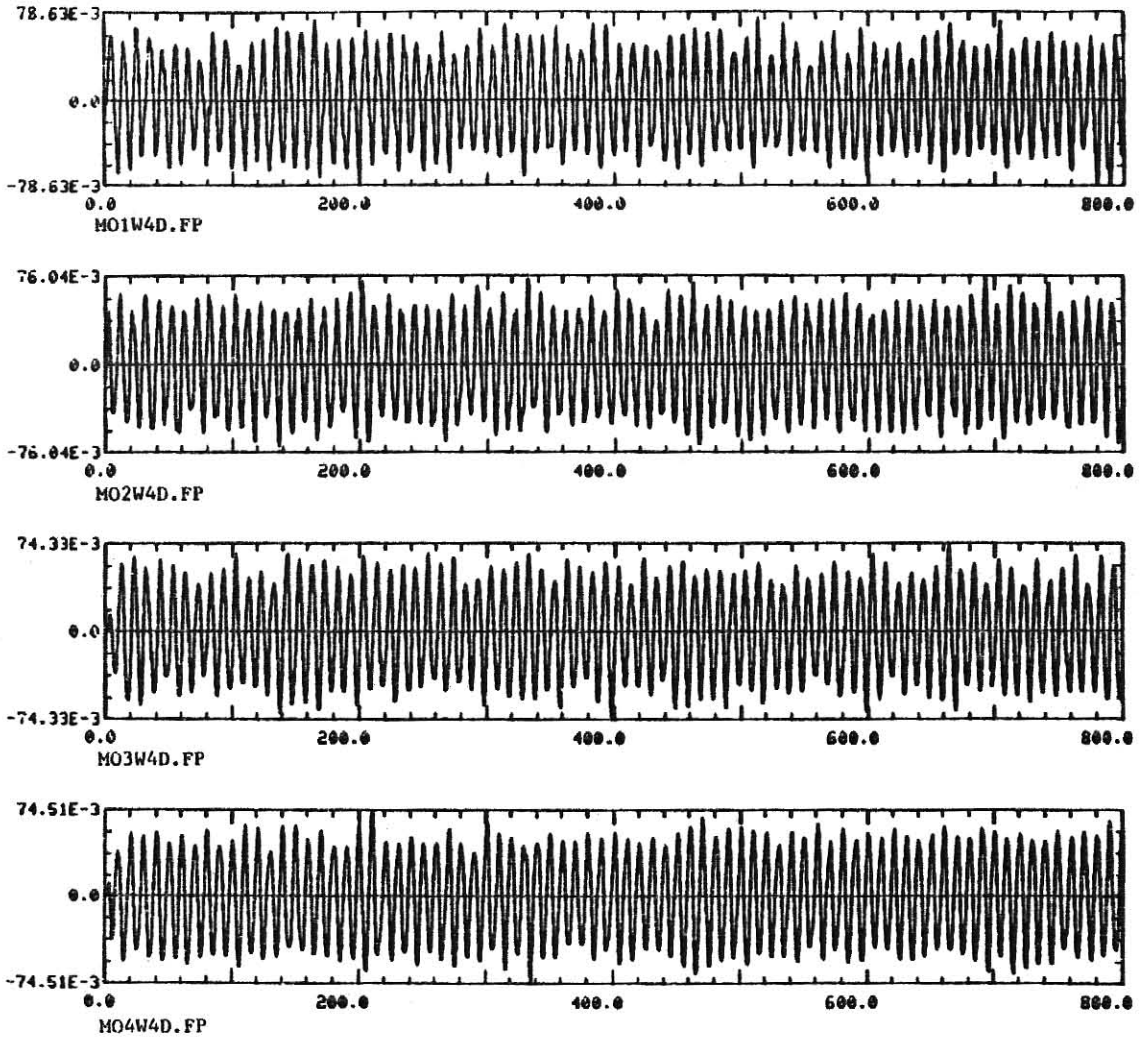
MID BAND: +-1.0 input with noise, fixed point



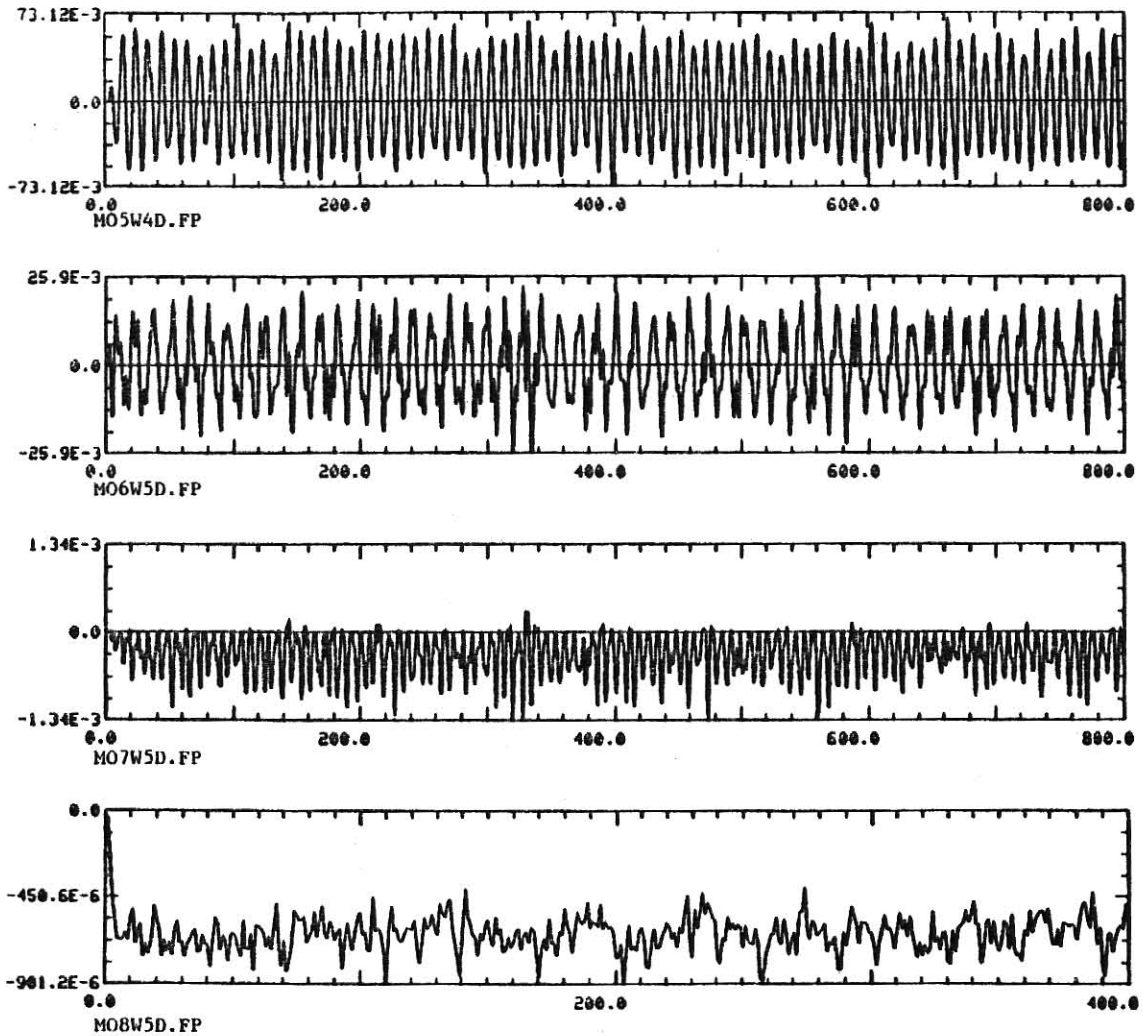
MID BAND: +-1.0 input with noise, fixed point



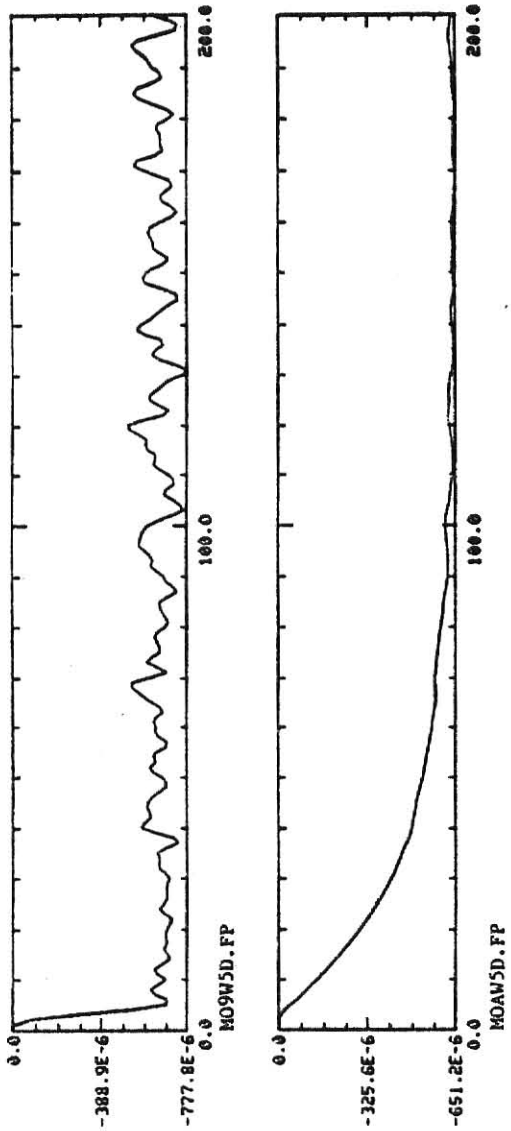
MID BAND: +-0.05 input with noise, double precision



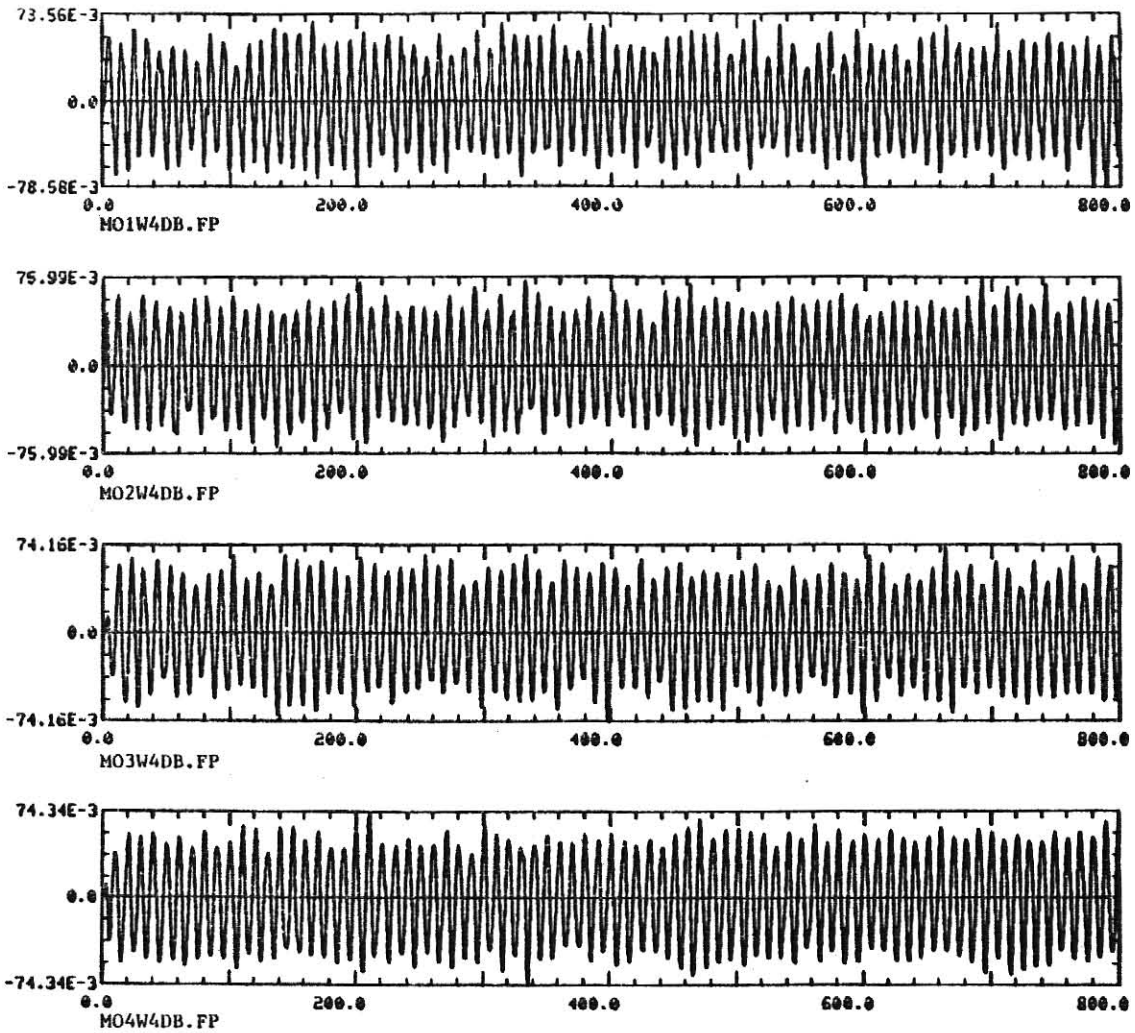
MID BAND: +-0.05 input with noise, double precision



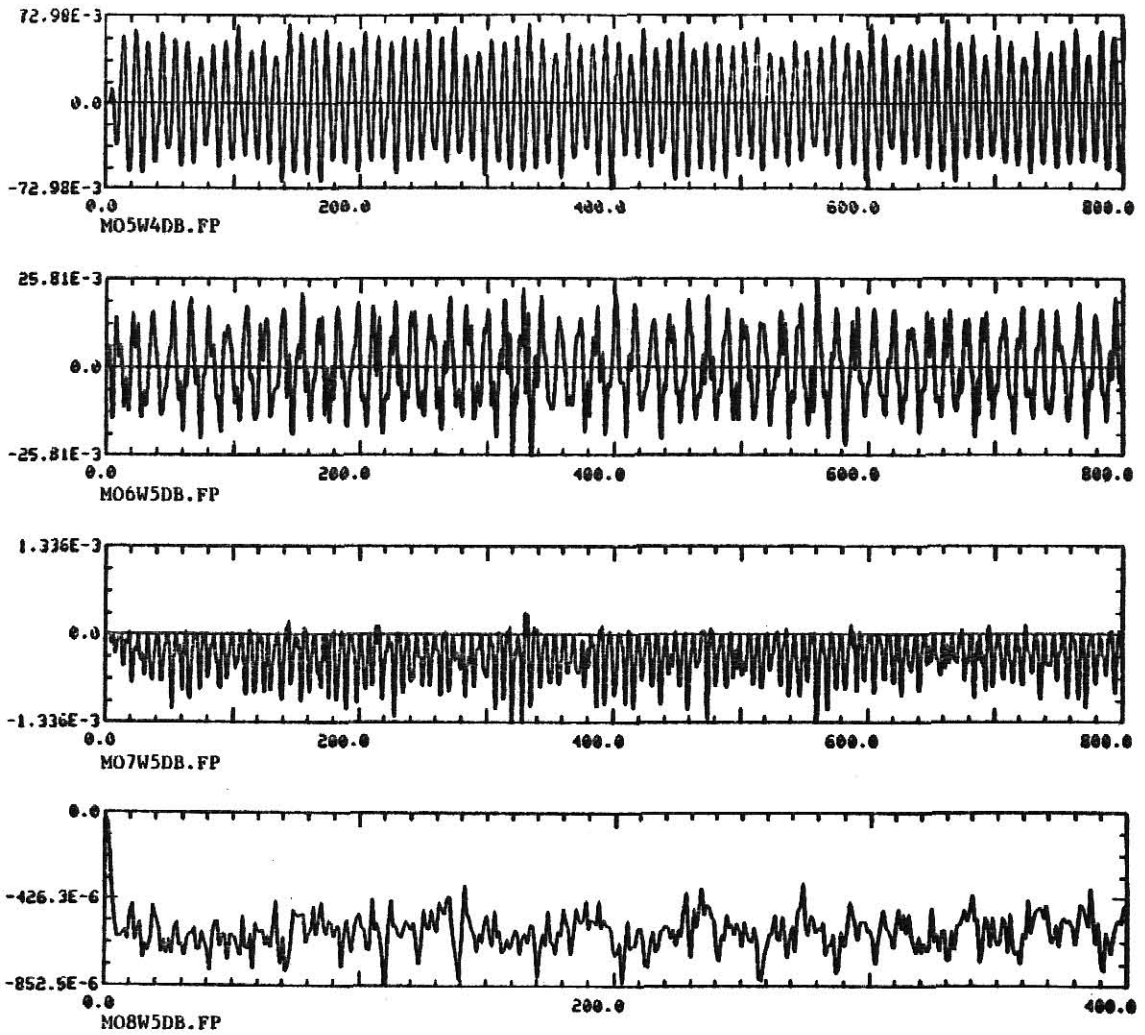
MID BAND: +-0.05 input with noise, double precision



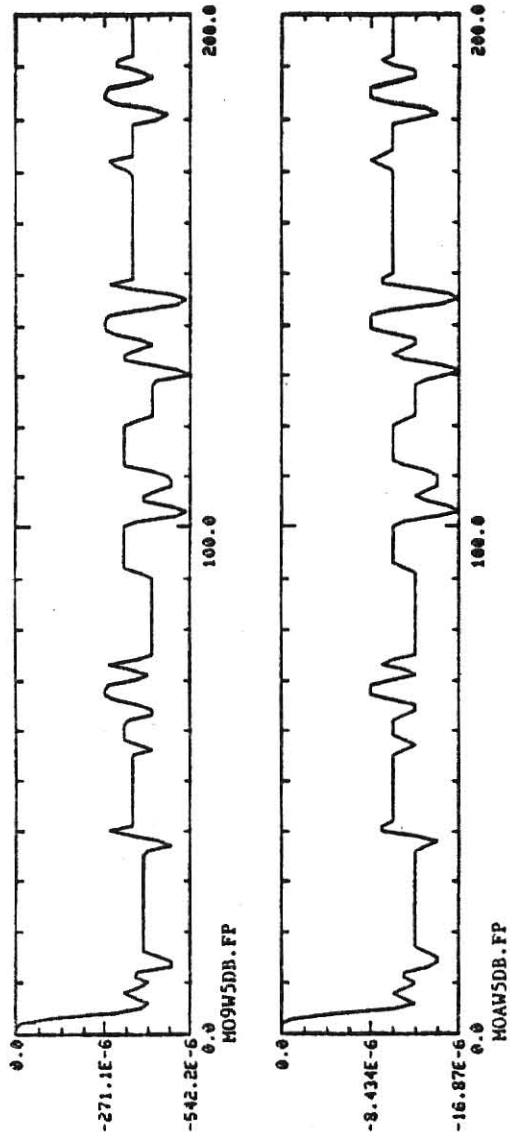
MID BAND: +-0.05 input with noise, fixed point



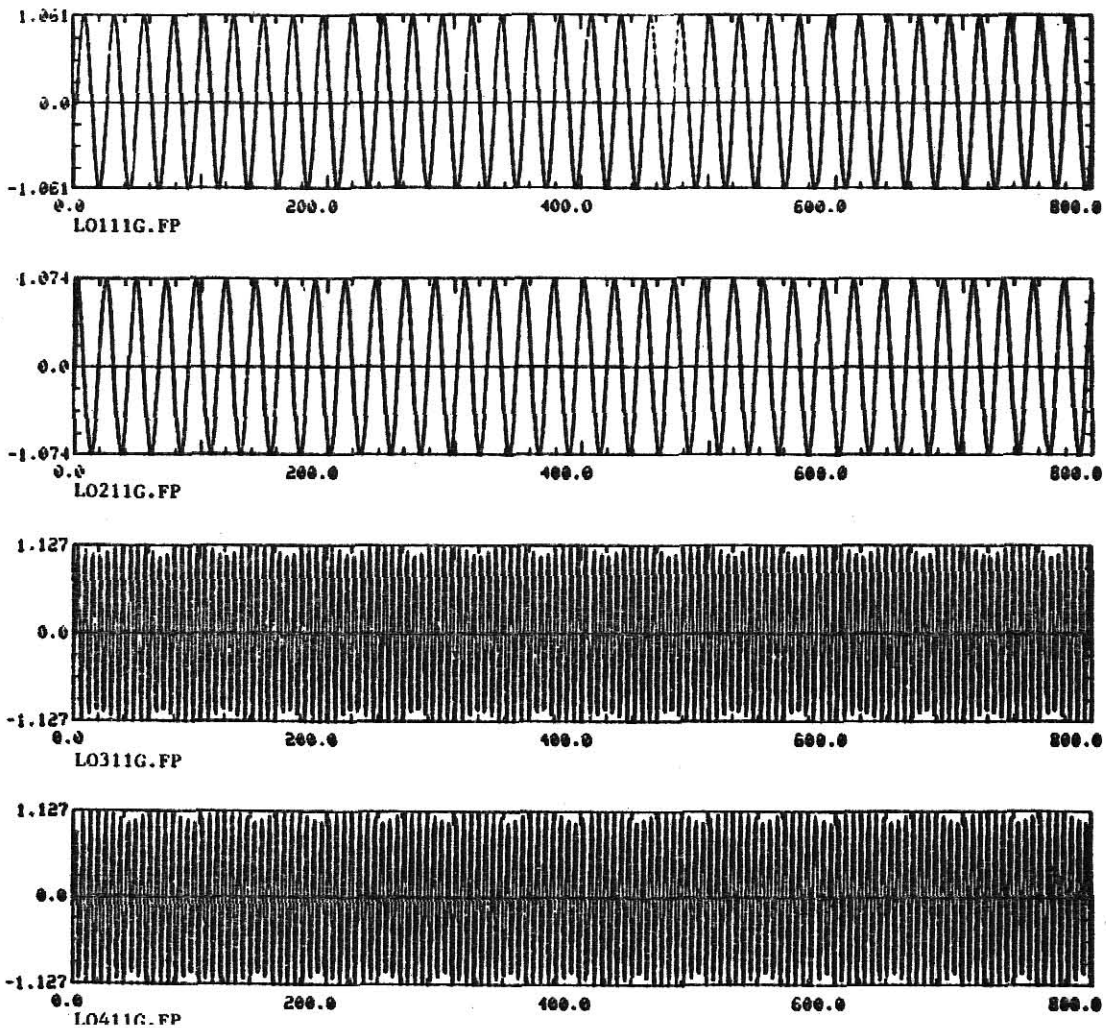
MID BAND: +-0.05 input with noise, fixed point



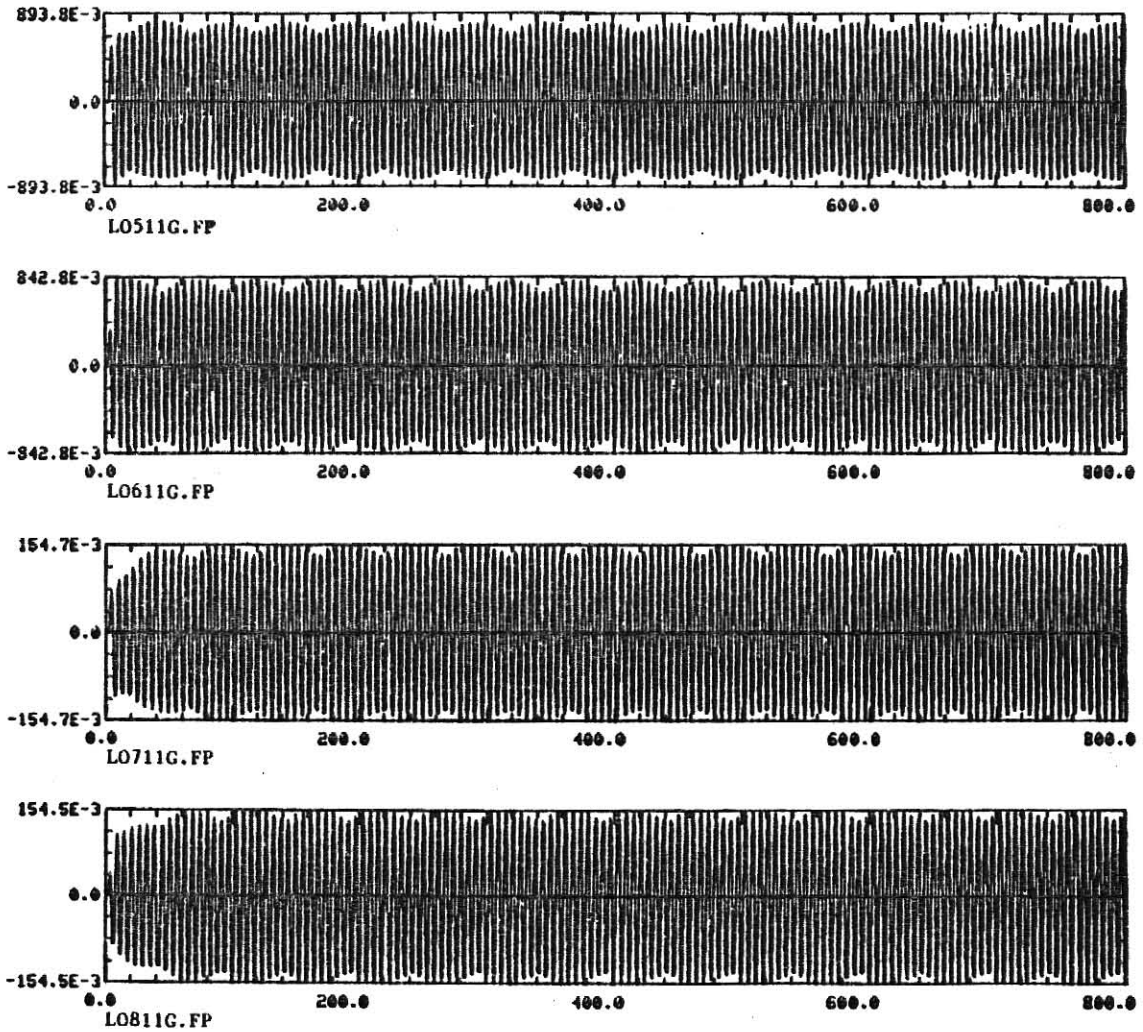
MID BAND: +-0.05 with noise, fixed point



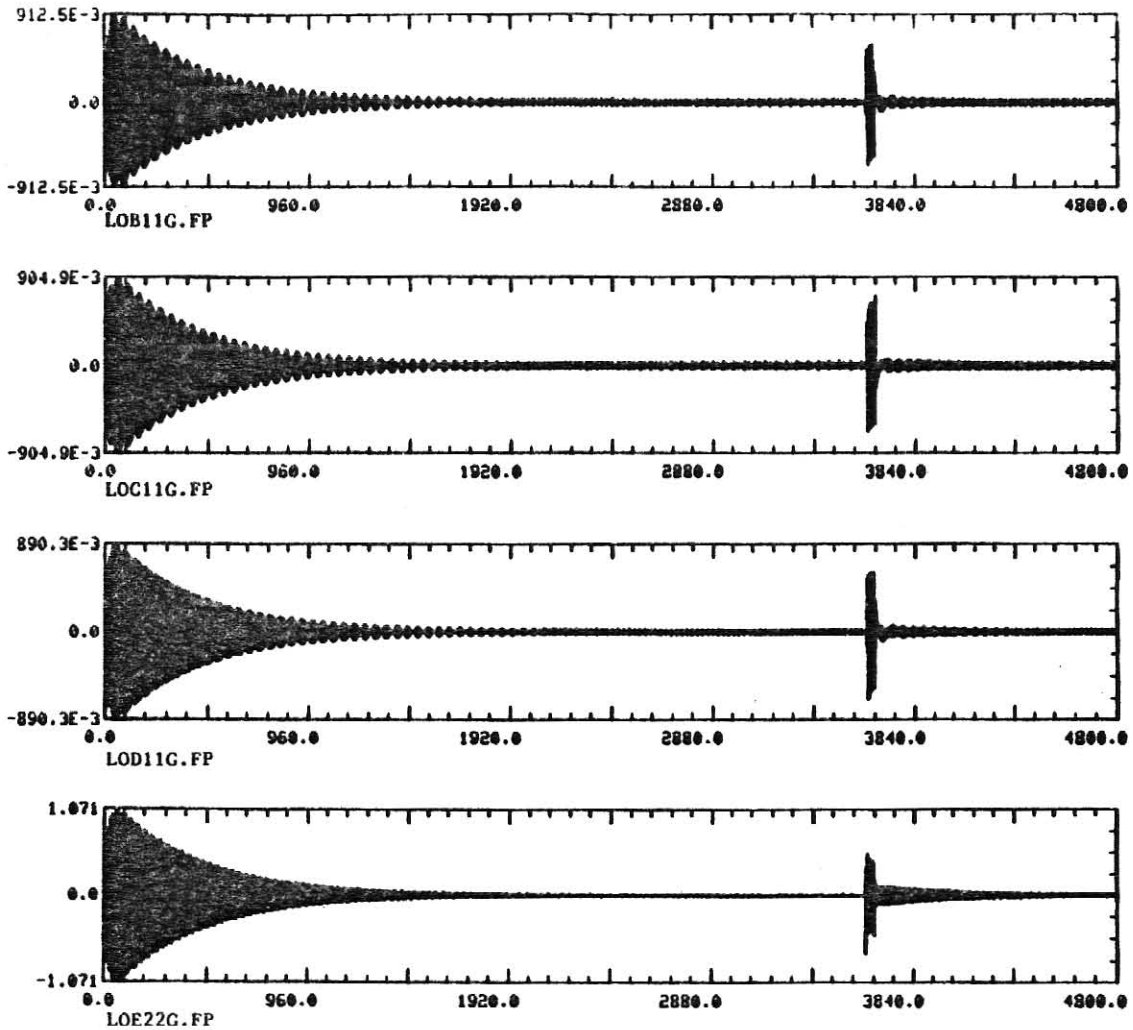
LOW BAND: +-1.0 input without noise, double precision



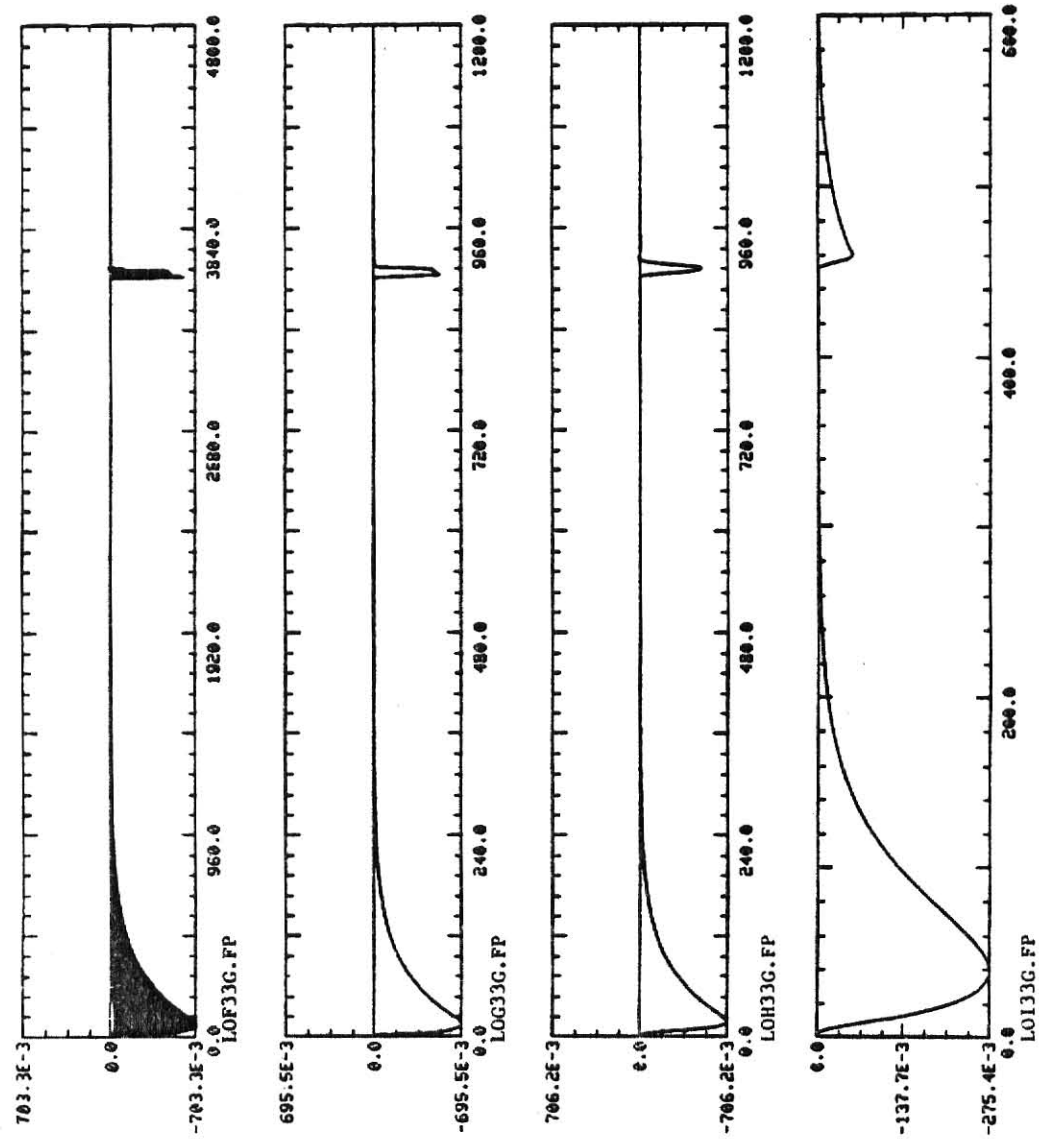
LOW BAND: +-1.0 input without noise, double precision



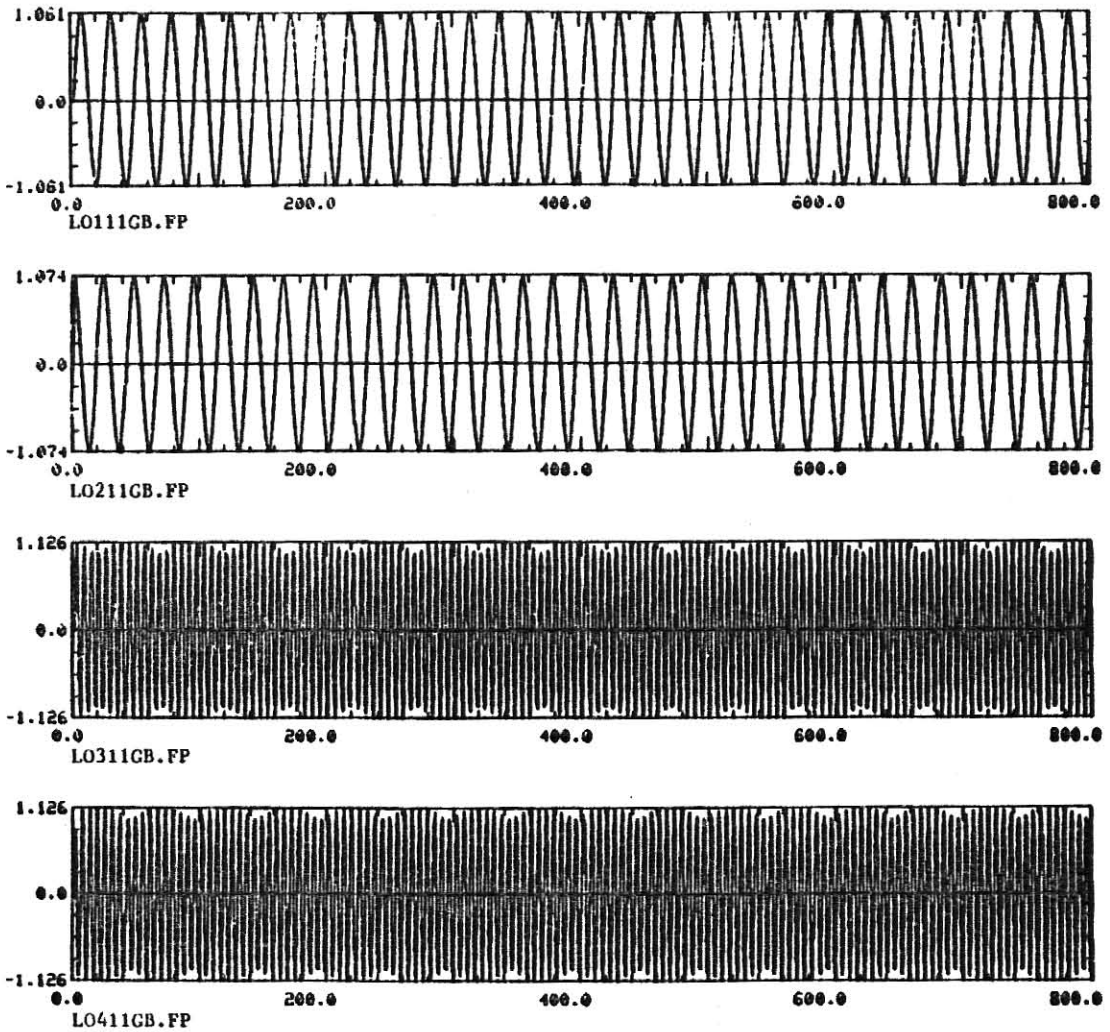
LOW BAND: +-1.0 input without noise, double precision



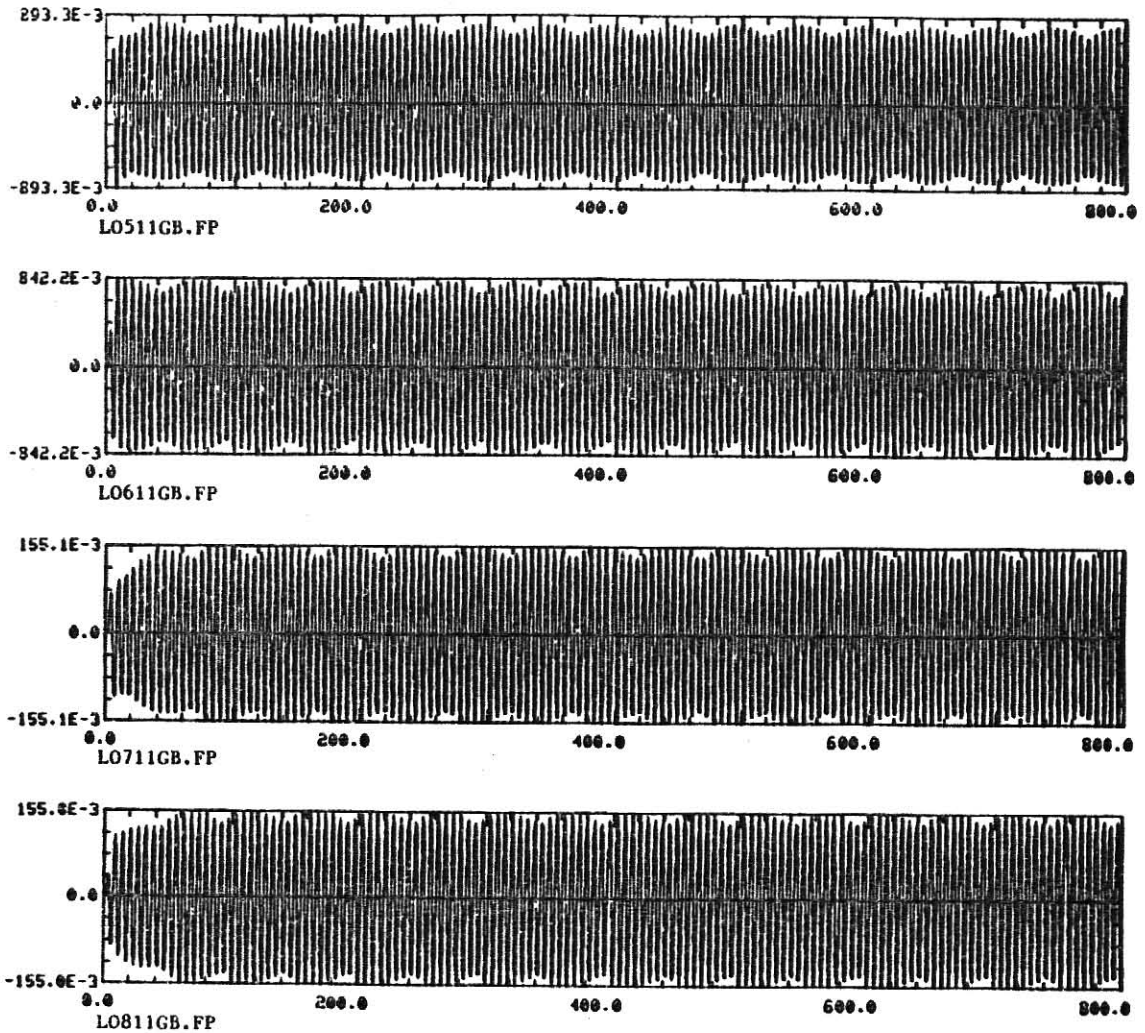
LOW BAND: +-1.0 input without noise, double precision



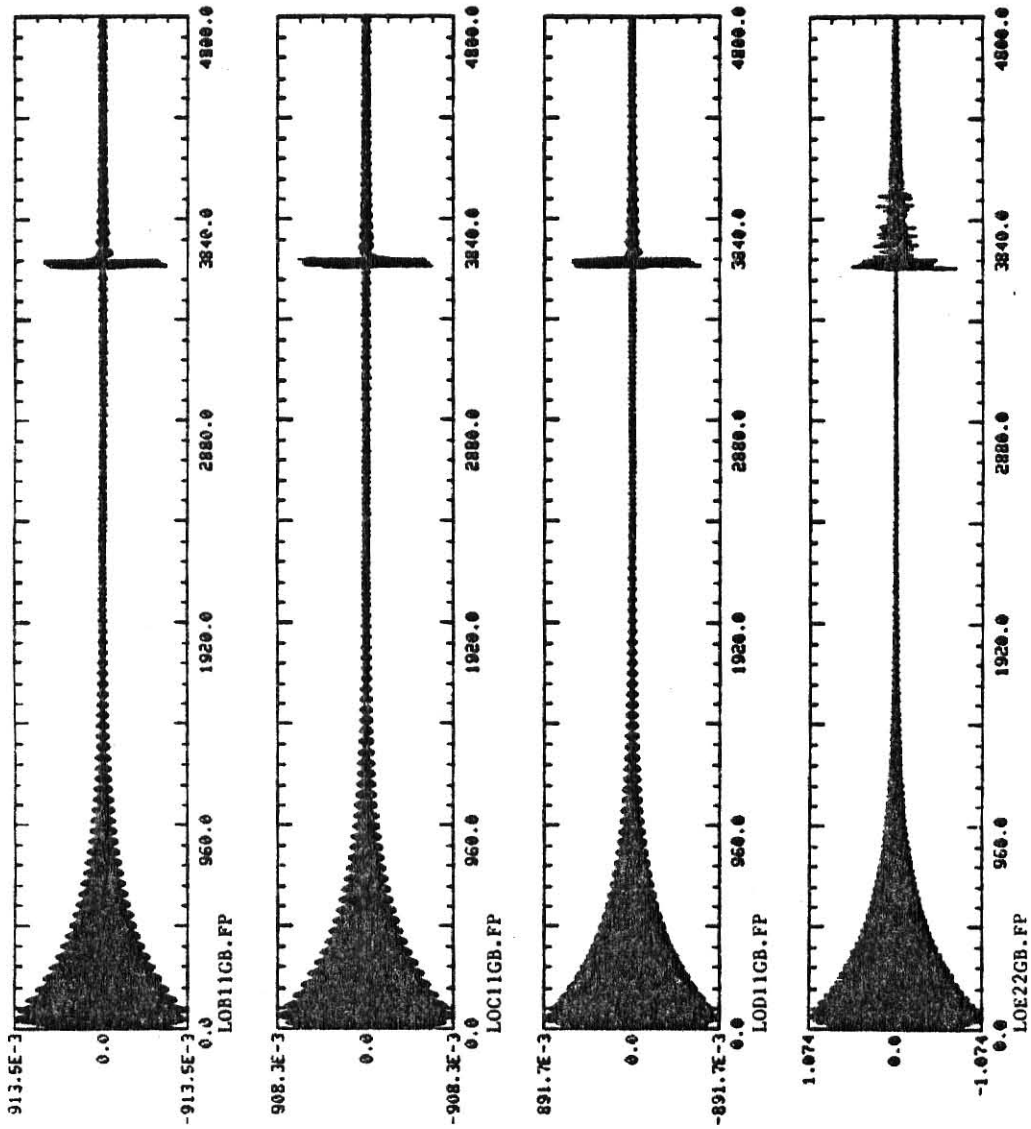
LOW BAND: +-1.0 input without noise, fixed point



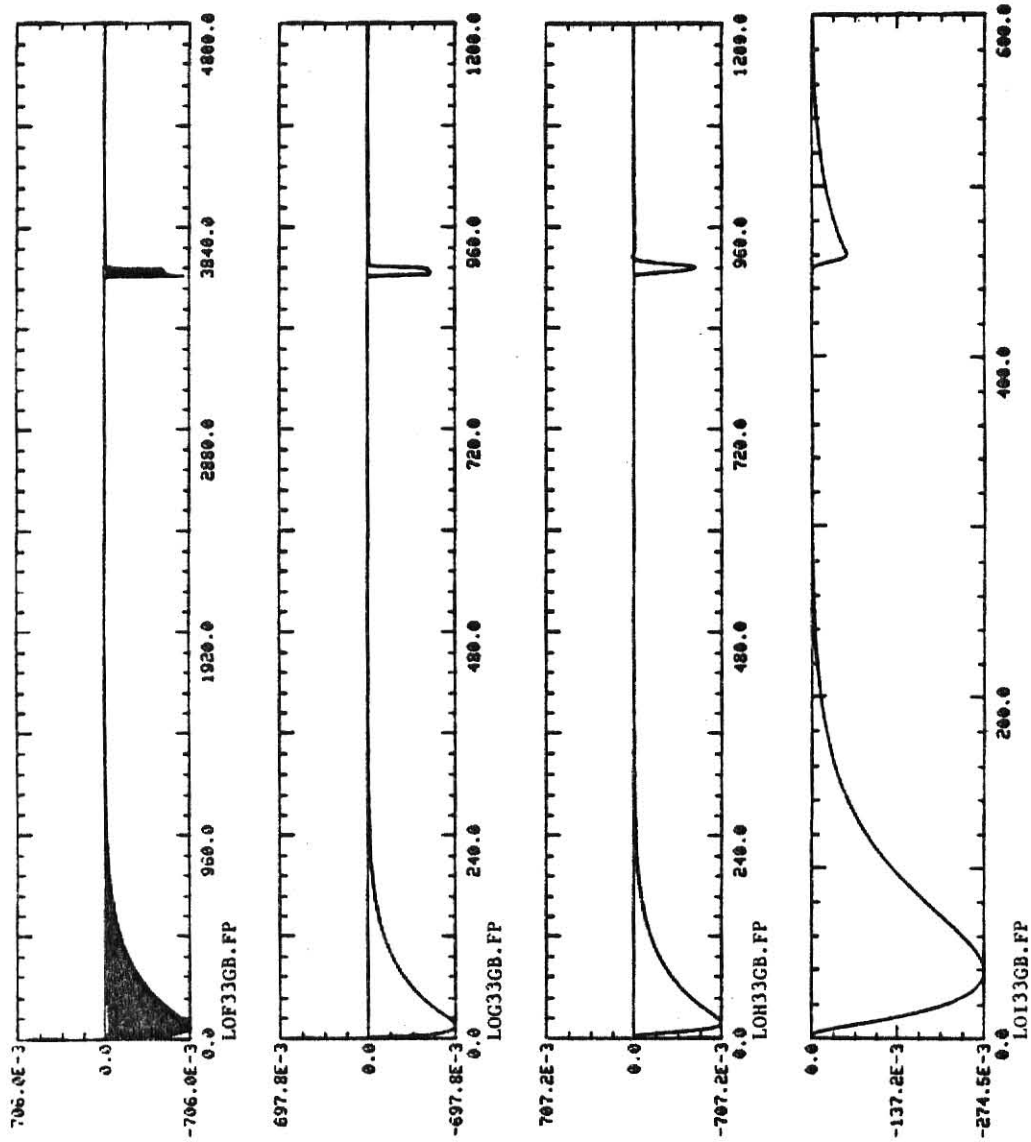
LOW BAND: +-1.0 without noise, fixed point



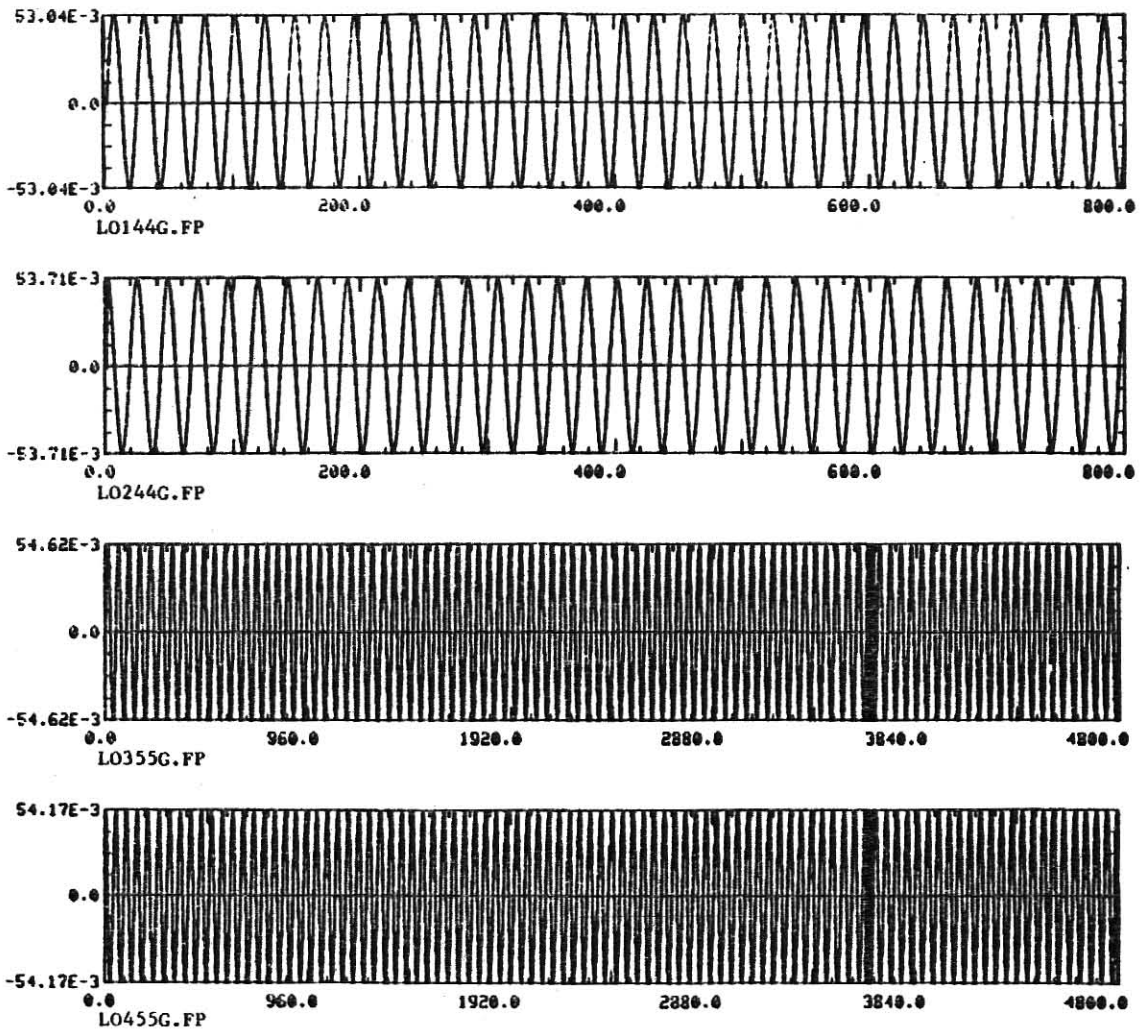
LOW BAND: +-1.0 input without noise, fixed point



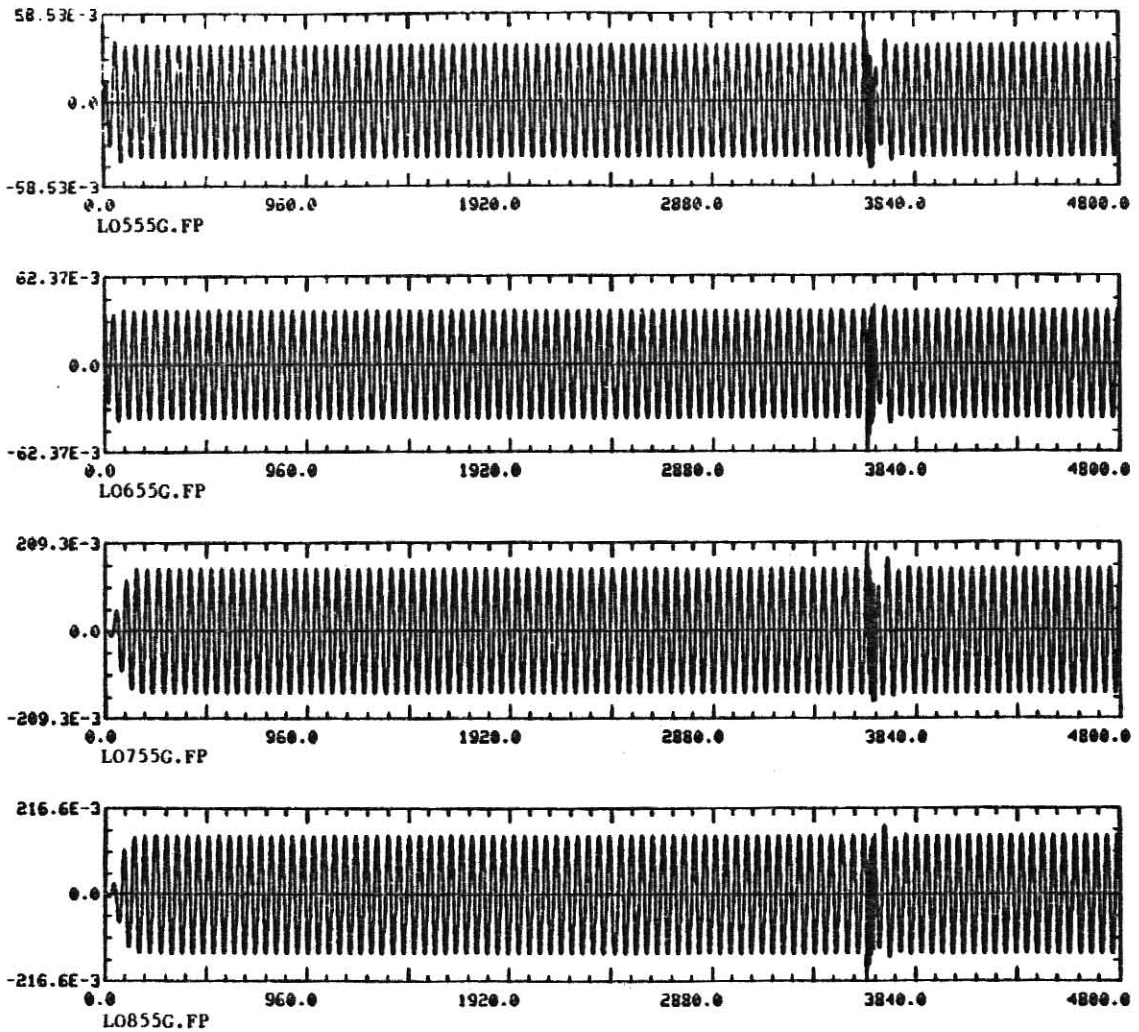
LOW BAND: +-1.0 input without noise, fixed point



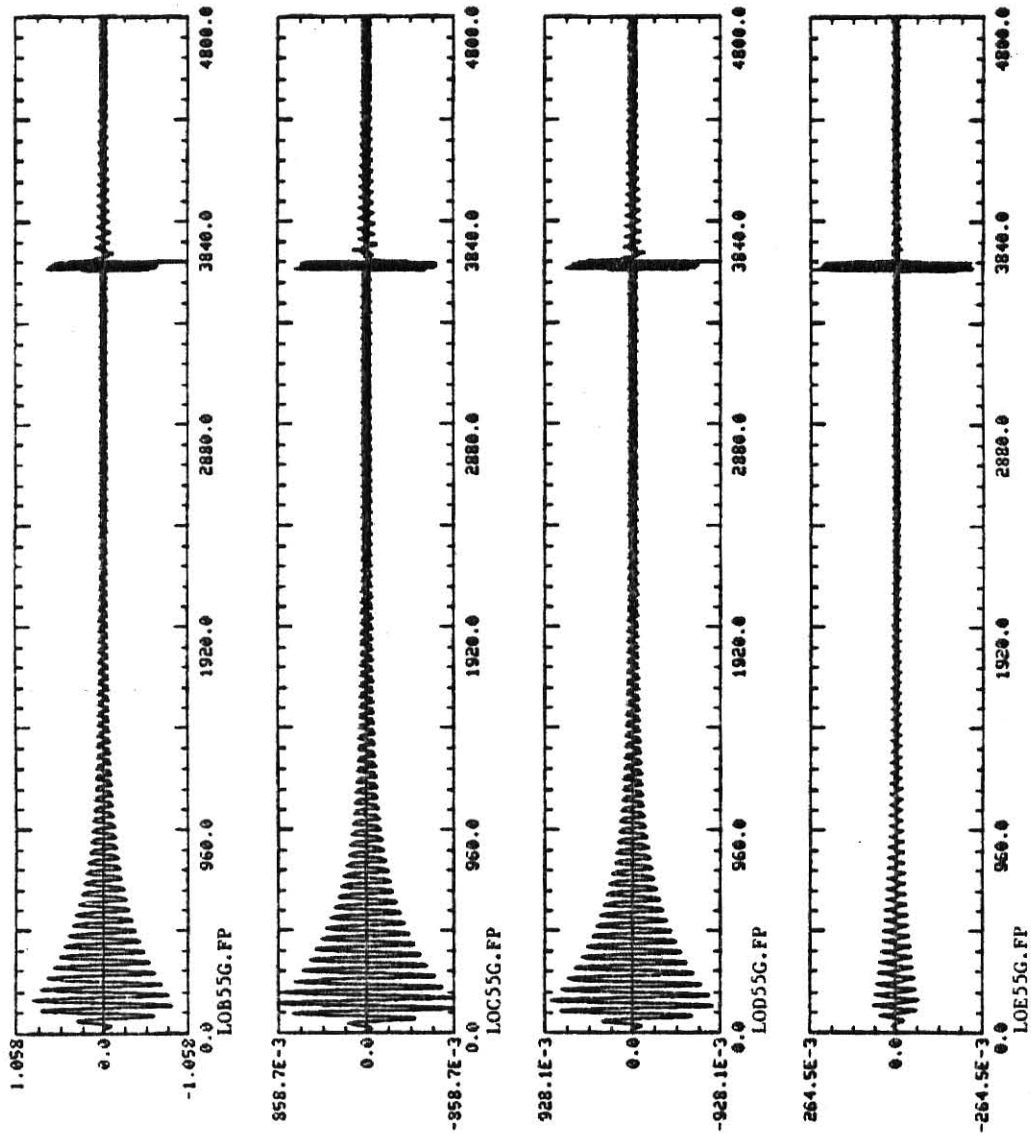
LOW BAND: +-0.05 input without noise, double precision



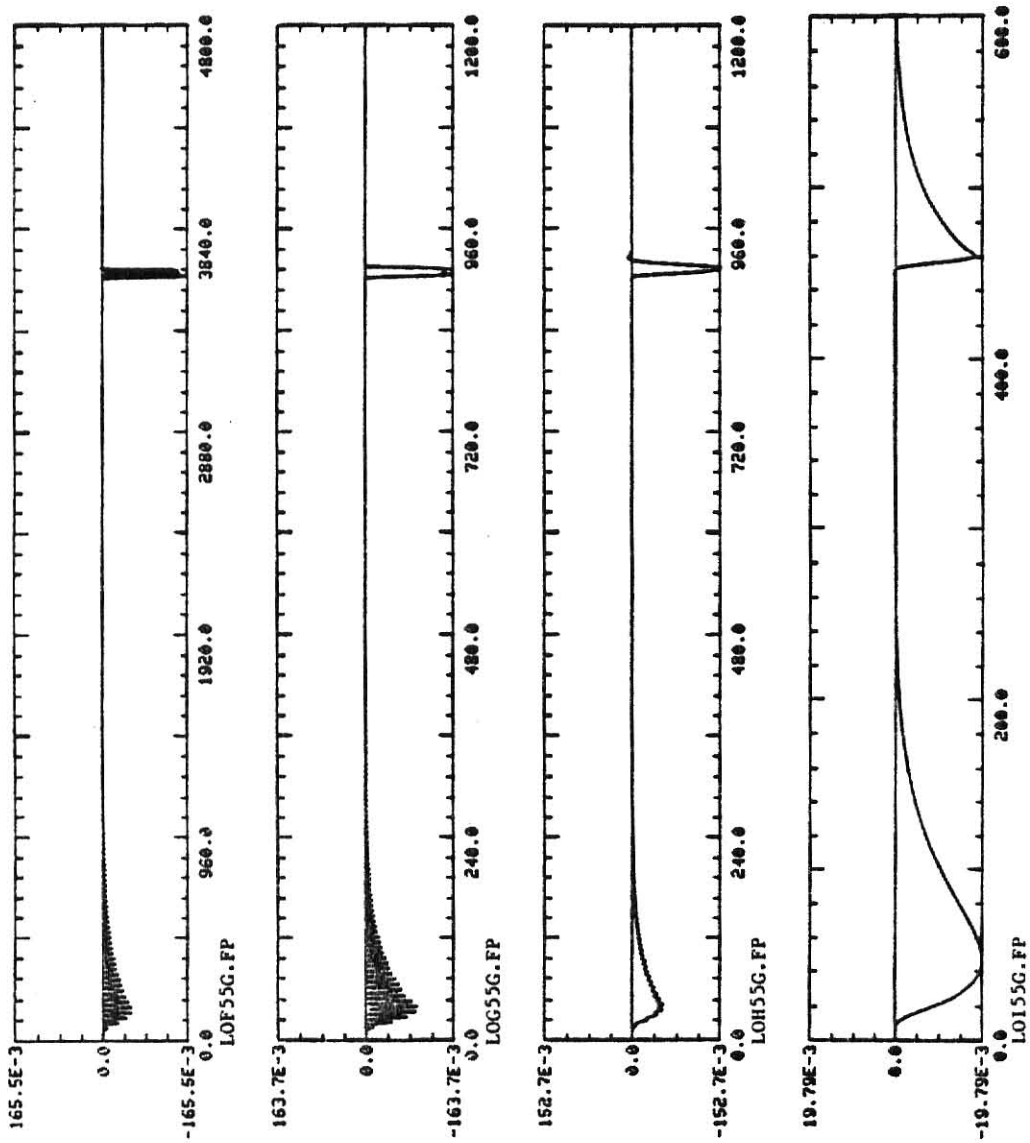
LOW BAND: +-0.05 input without noise, double precision



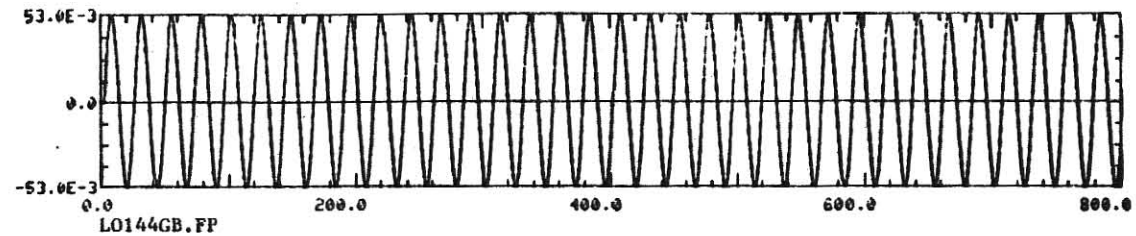
LOW BAND: +-0.05 input without noise, double precision

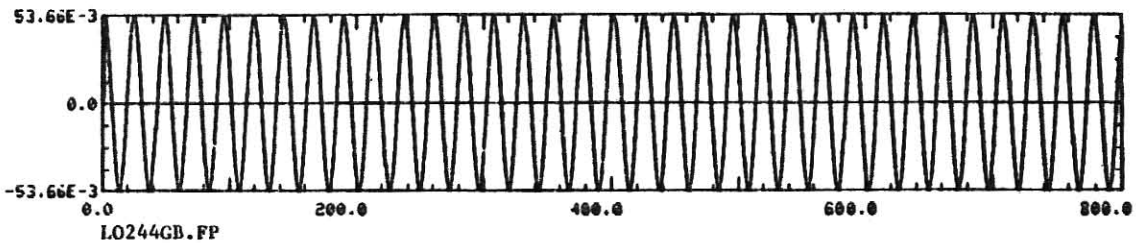


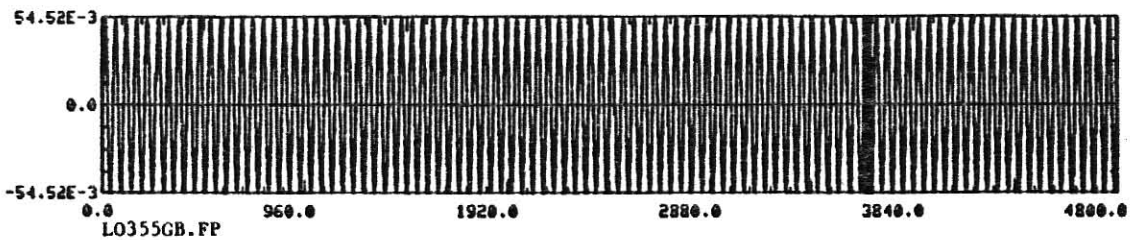
LOW BAND: +-0.05 input without noise, double precision

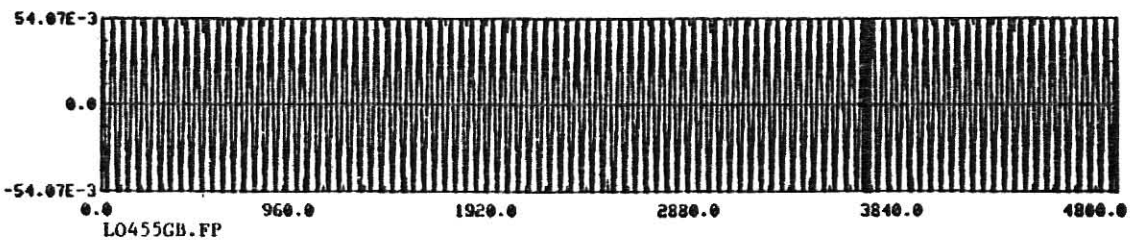


LOW BAND: +-0.05 input without noise, fixed point

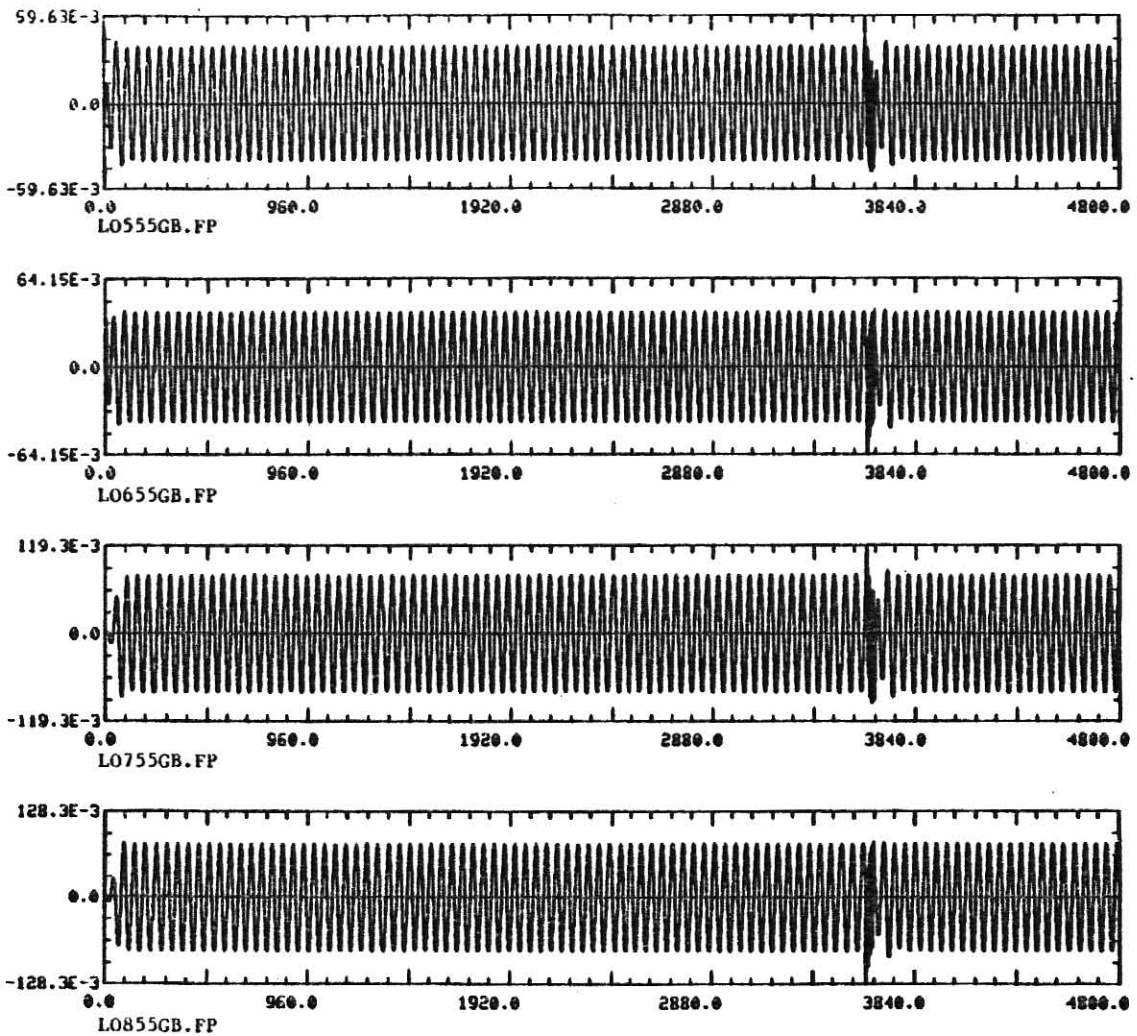




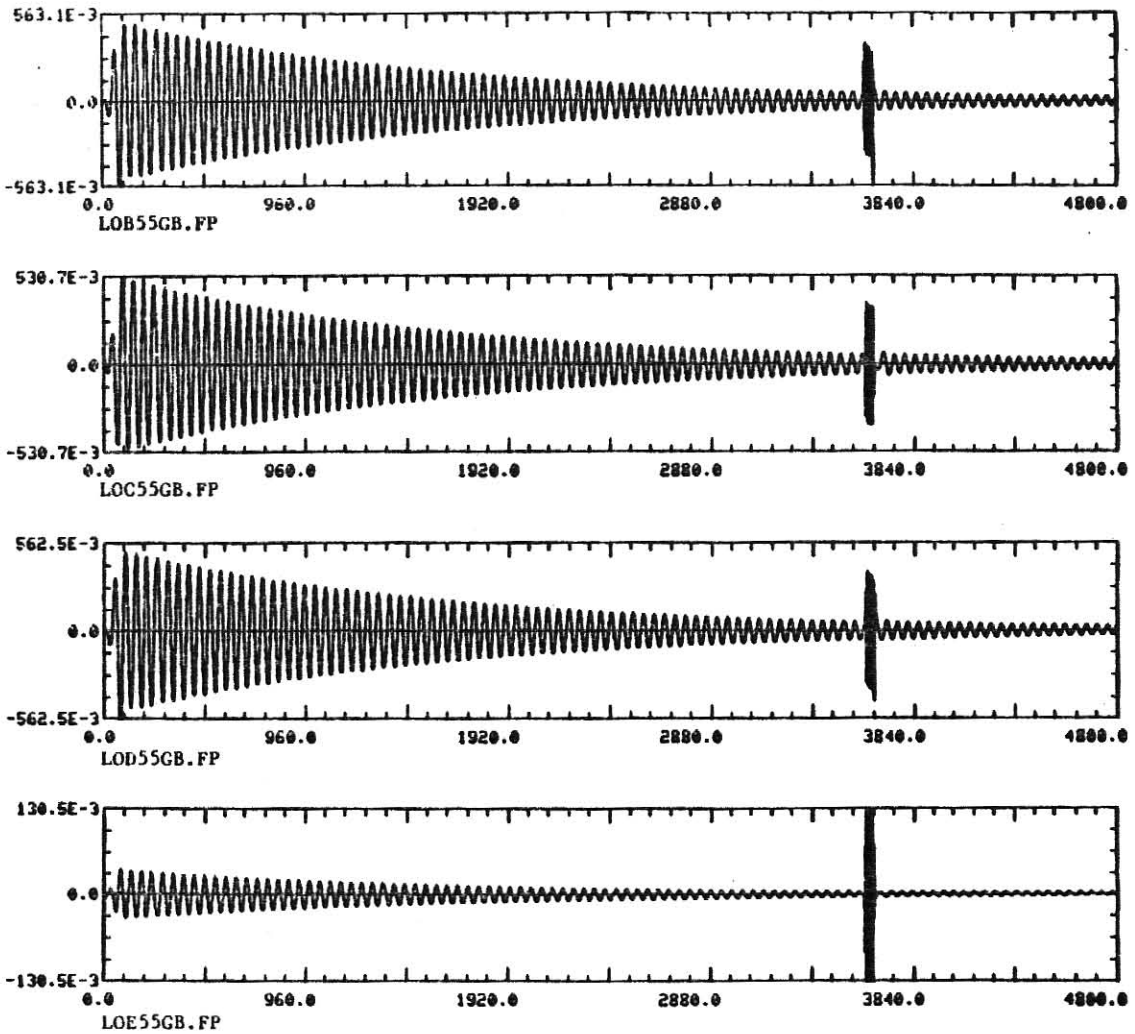




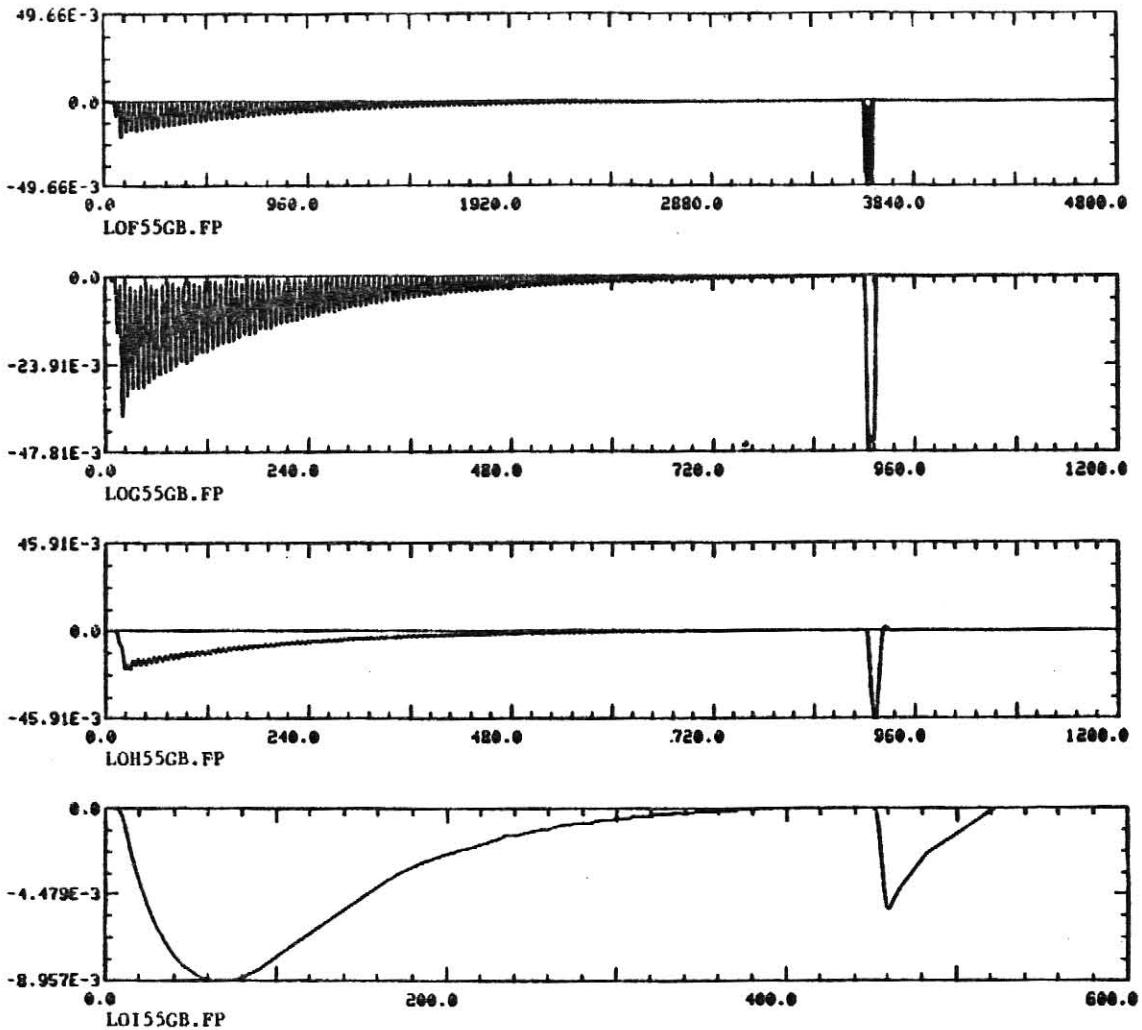
LOW BAND: +-0.05 input without noise, fixed point



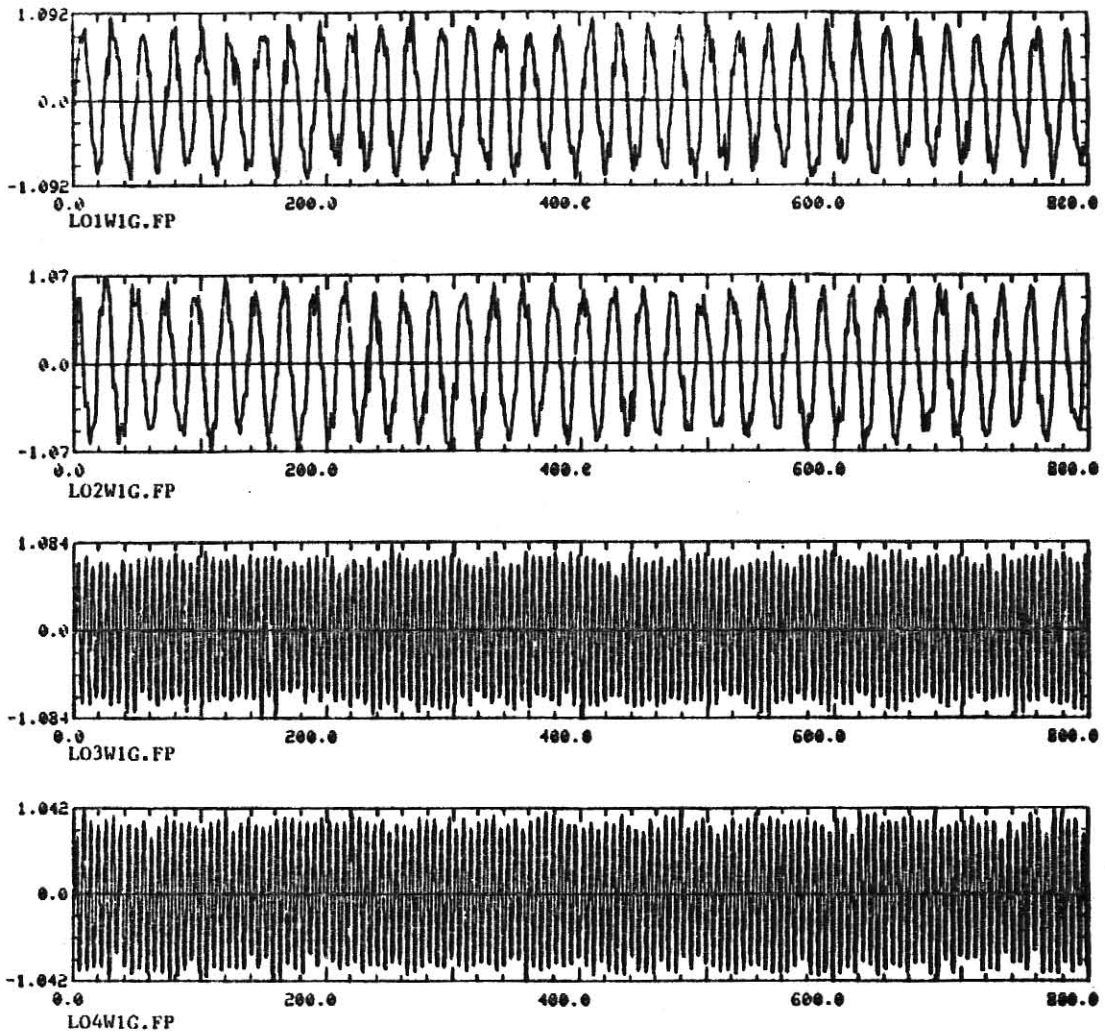
LOW BAND: +-0.05 input without noise, fixed point



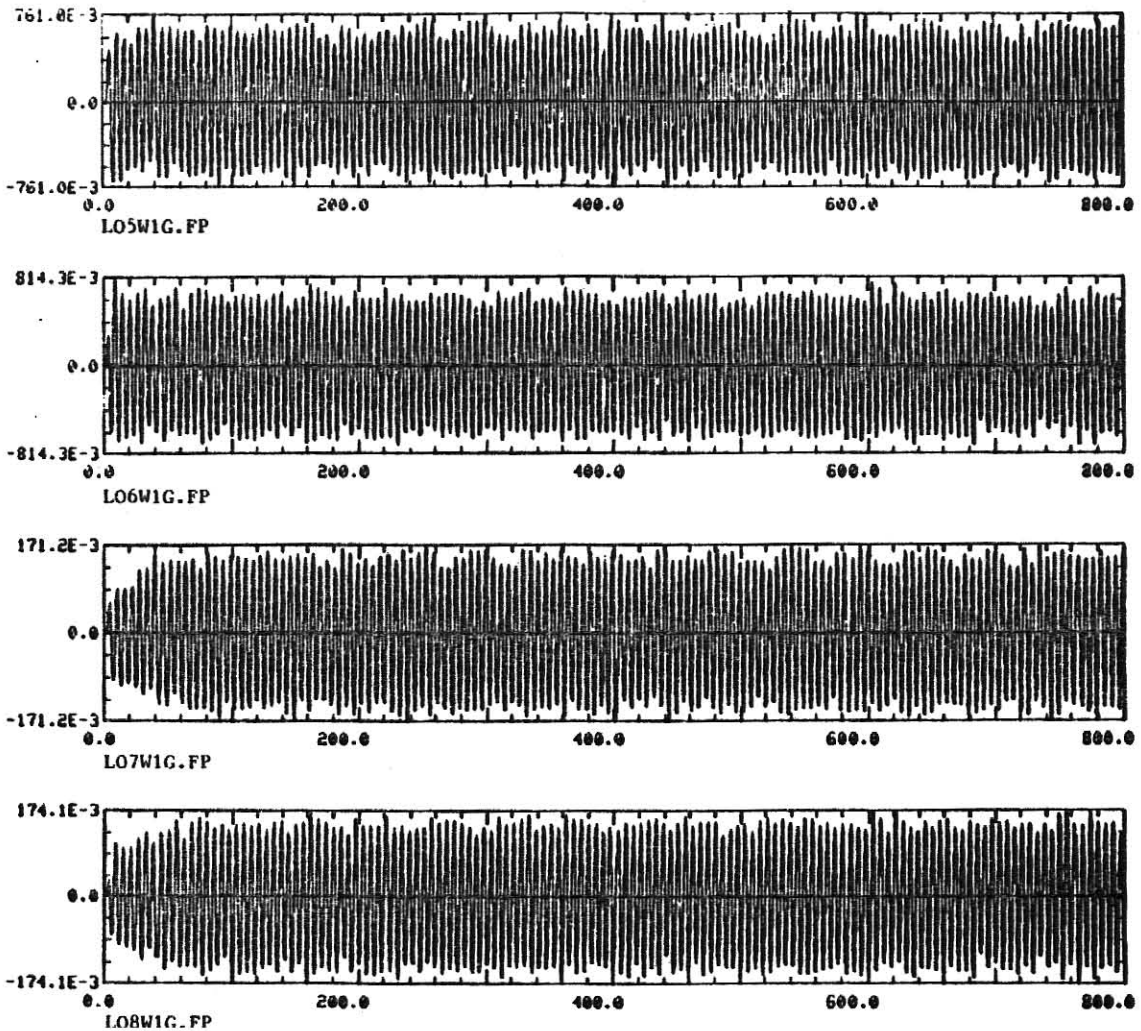
LOW BAND: +-0.05 input without noise, fixed point



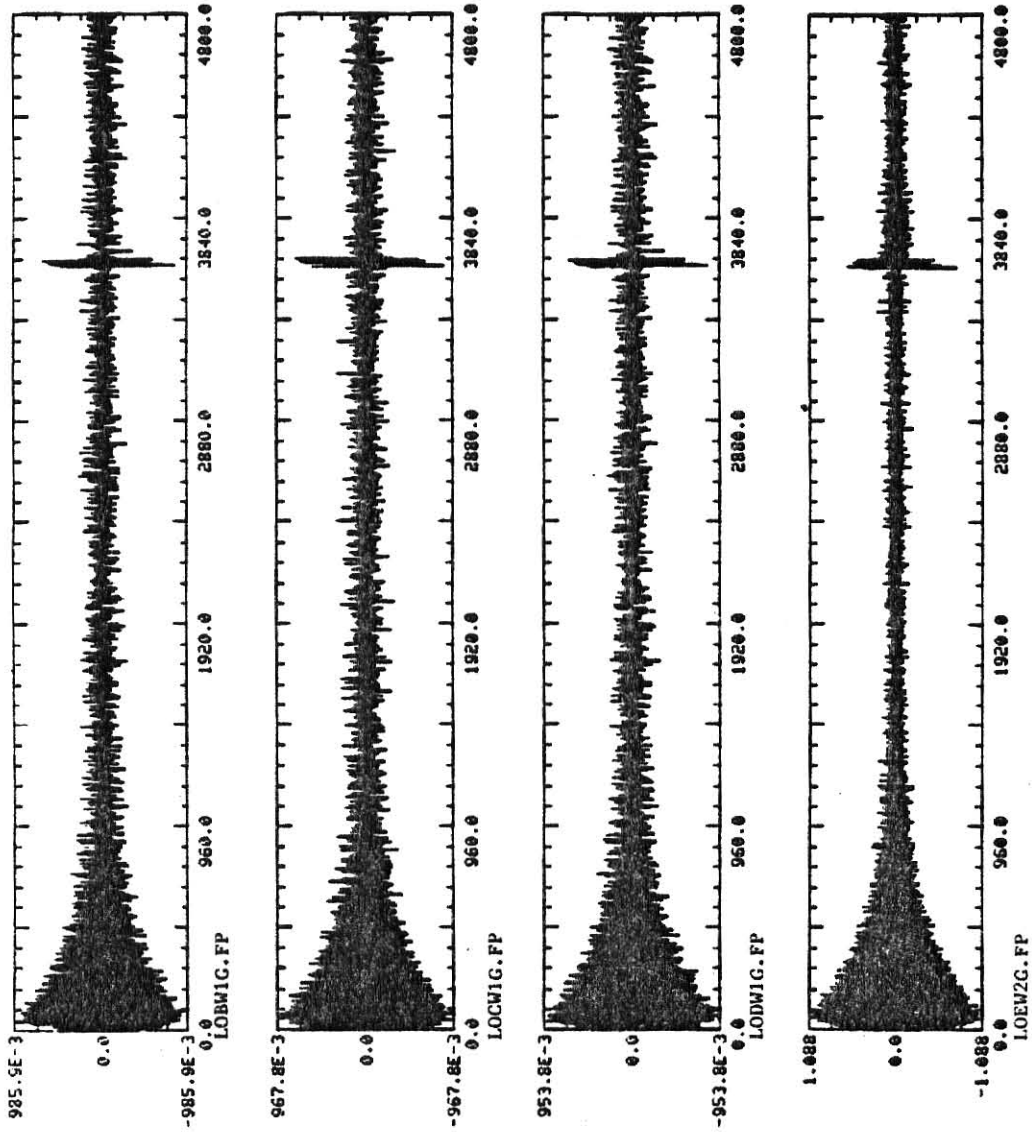
LOW BAND: +-1.0 input with noise, double precision



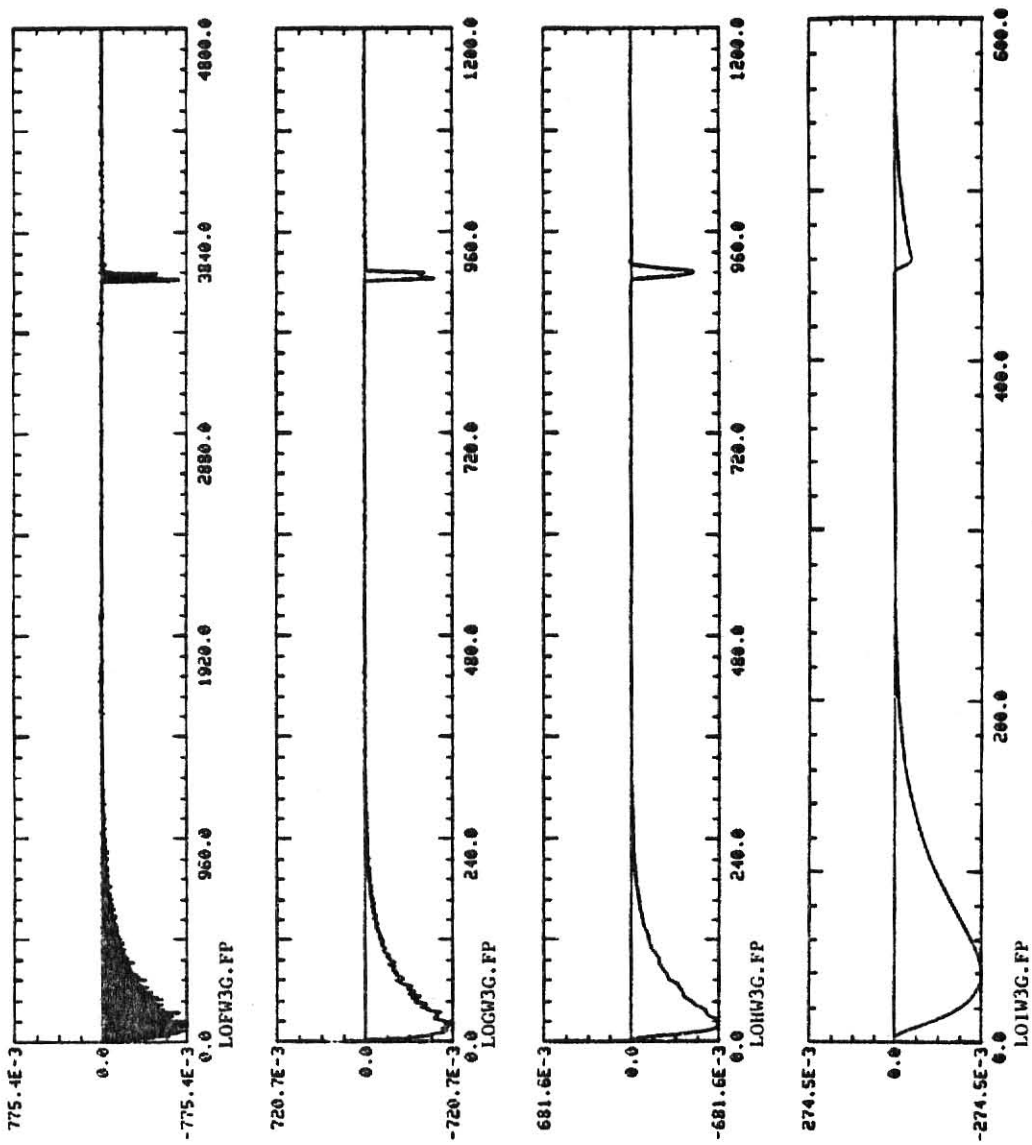
LOW BAND: +-1.0 input with noise, double precision



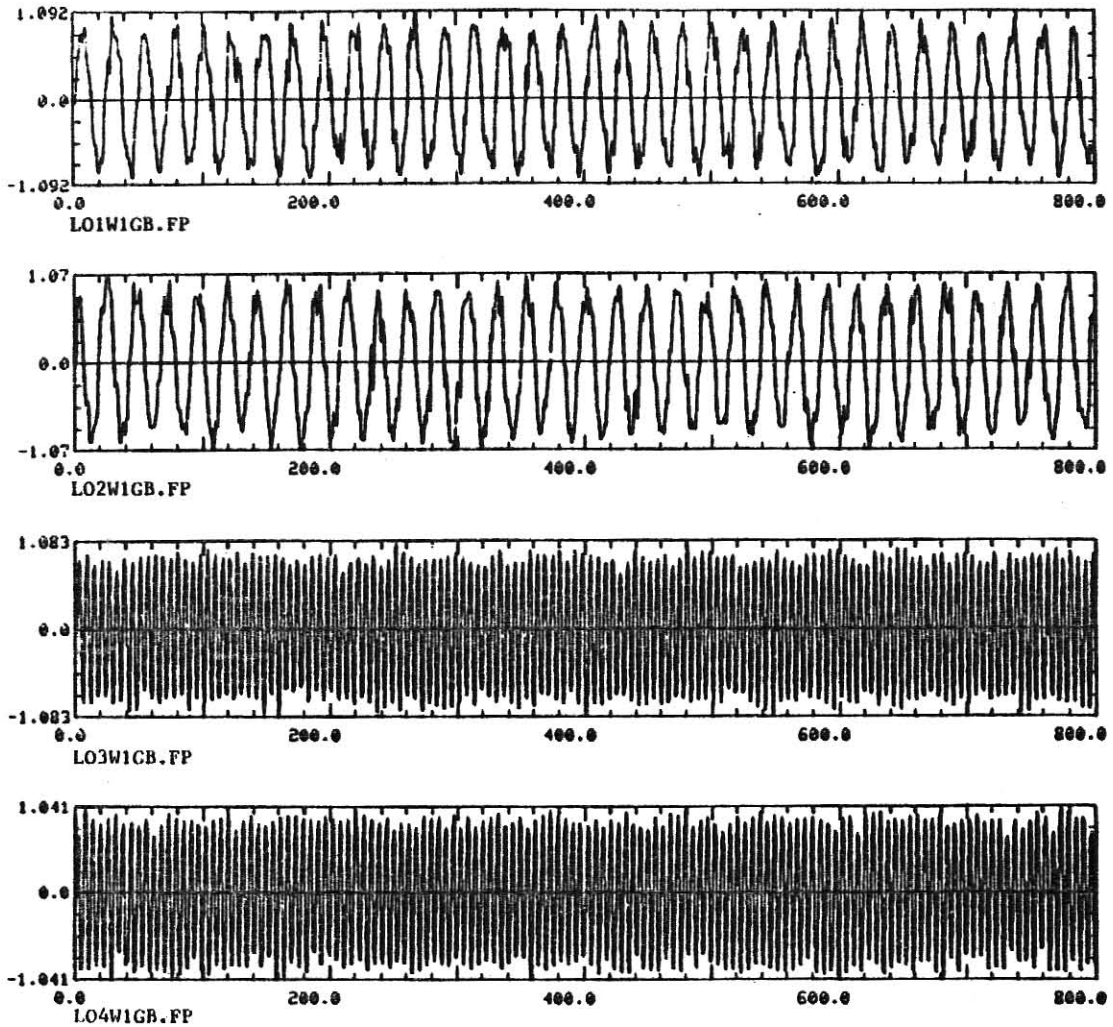
LOW BAND: +-1.0 input with noise, double precision



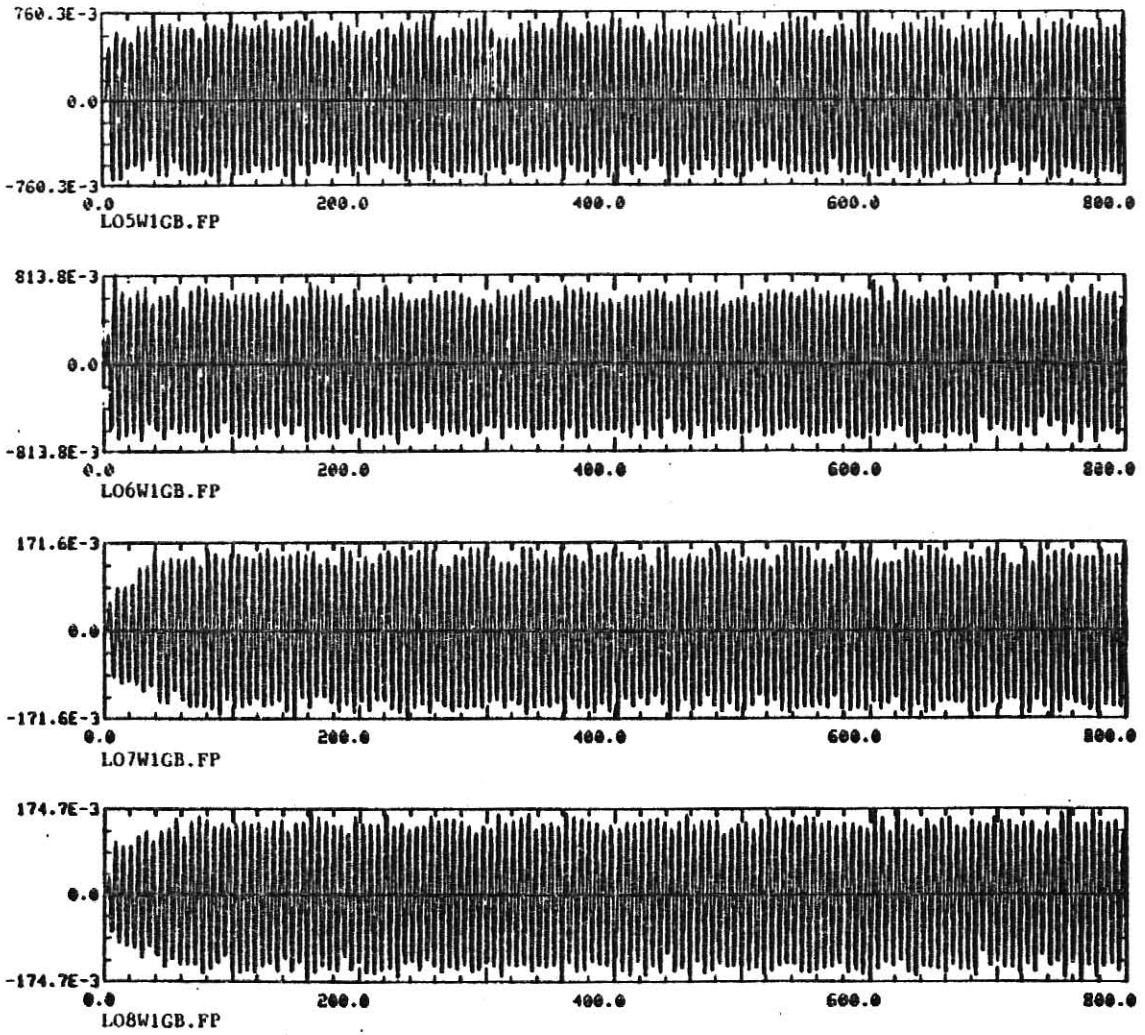
LOW BAND: +-1.0 input with noise, double precision



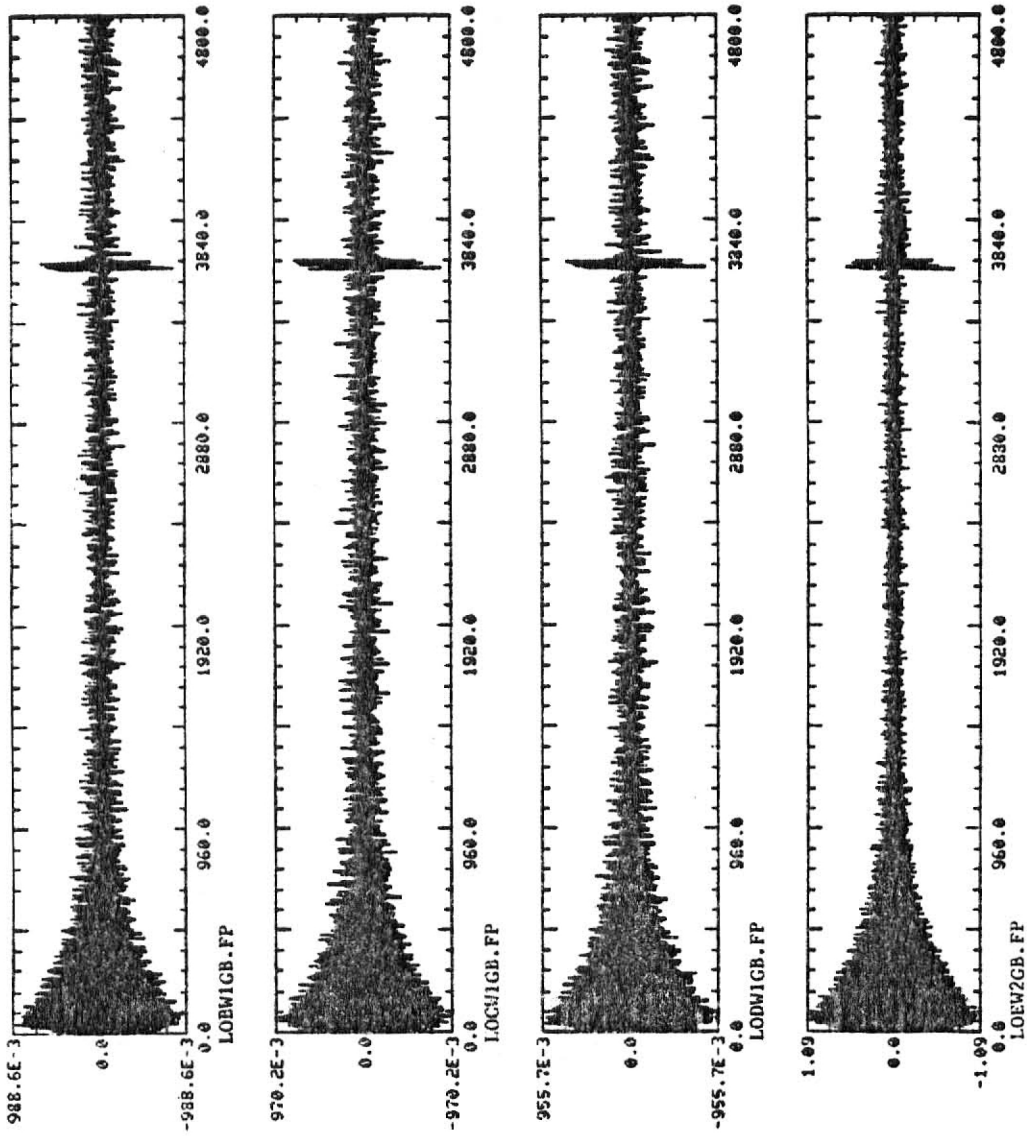
LOW BAND: +-1.0 input with noise, fixed point



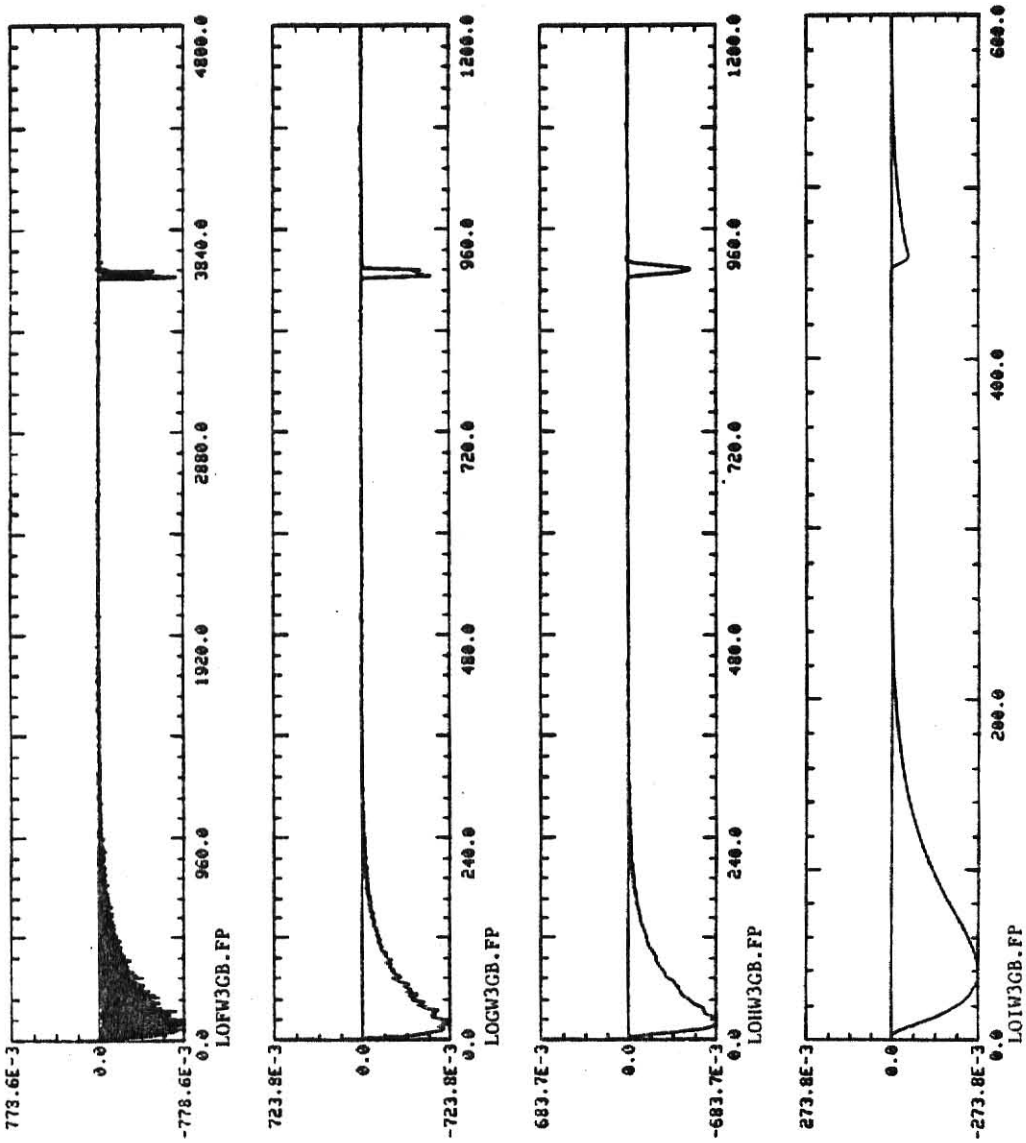
LOW BAND: +-1.0 input with noise, fixed point



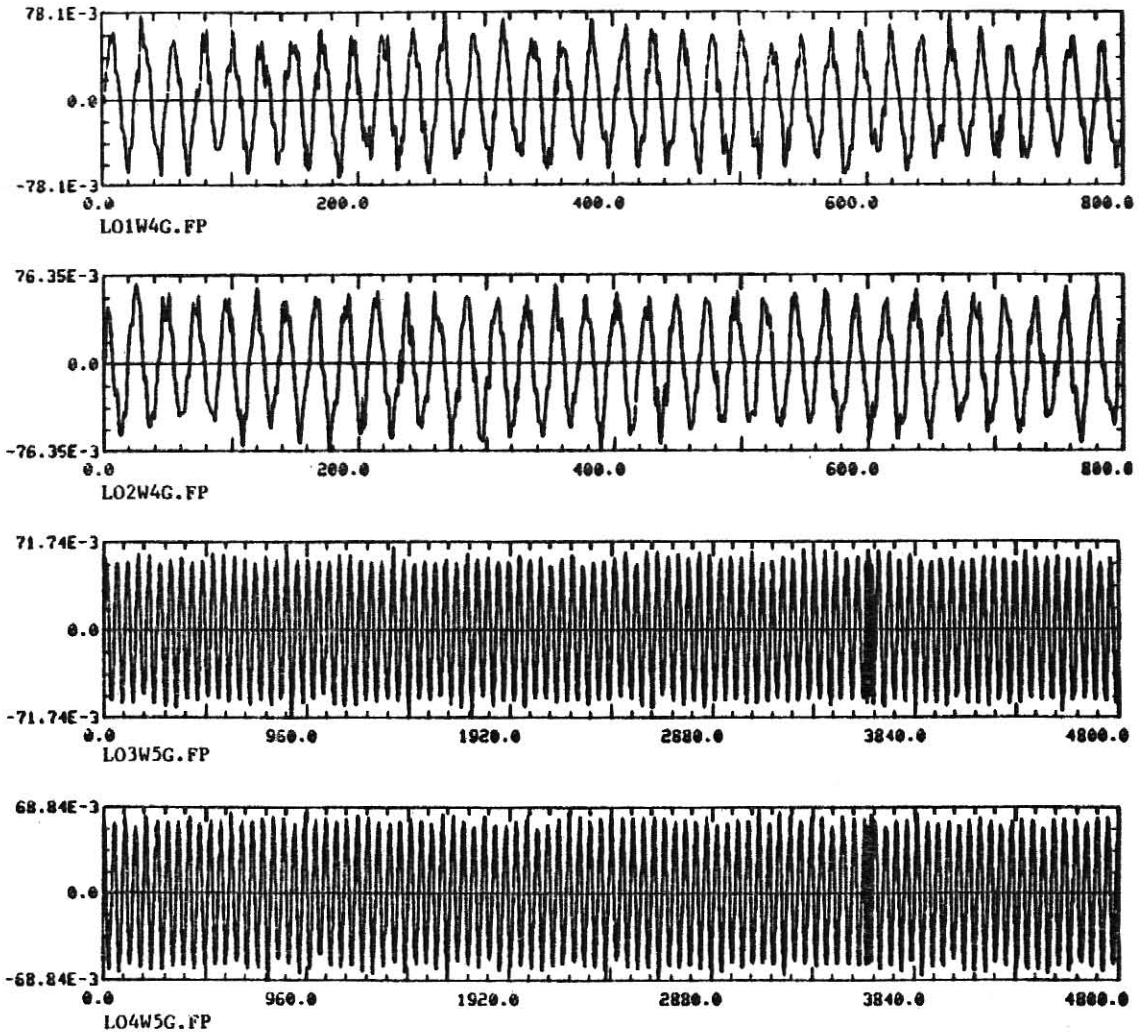
LOW BAND: +-1.0 input with noise, fixed point



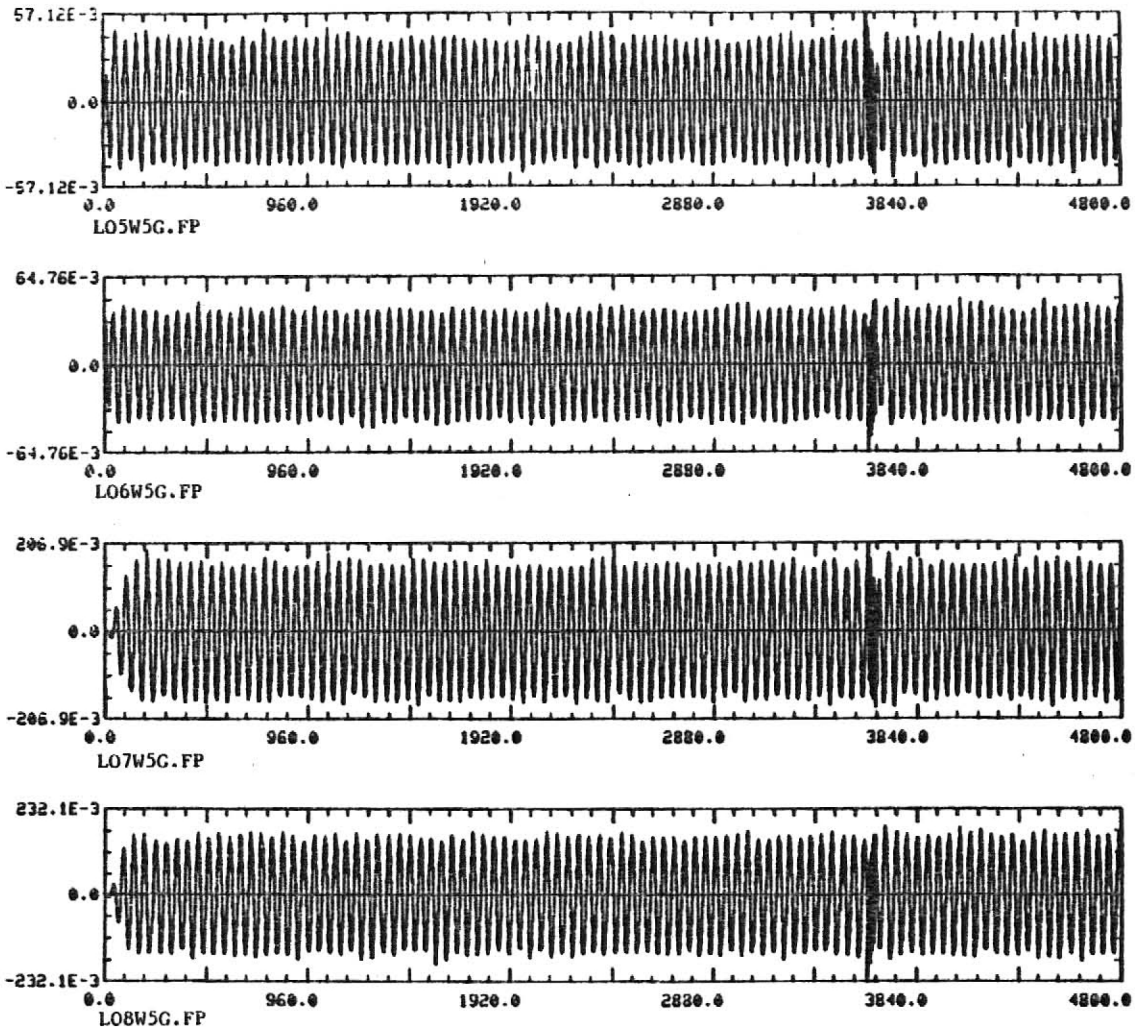
LOW BAND: +-1.0 input with noise, fixed point



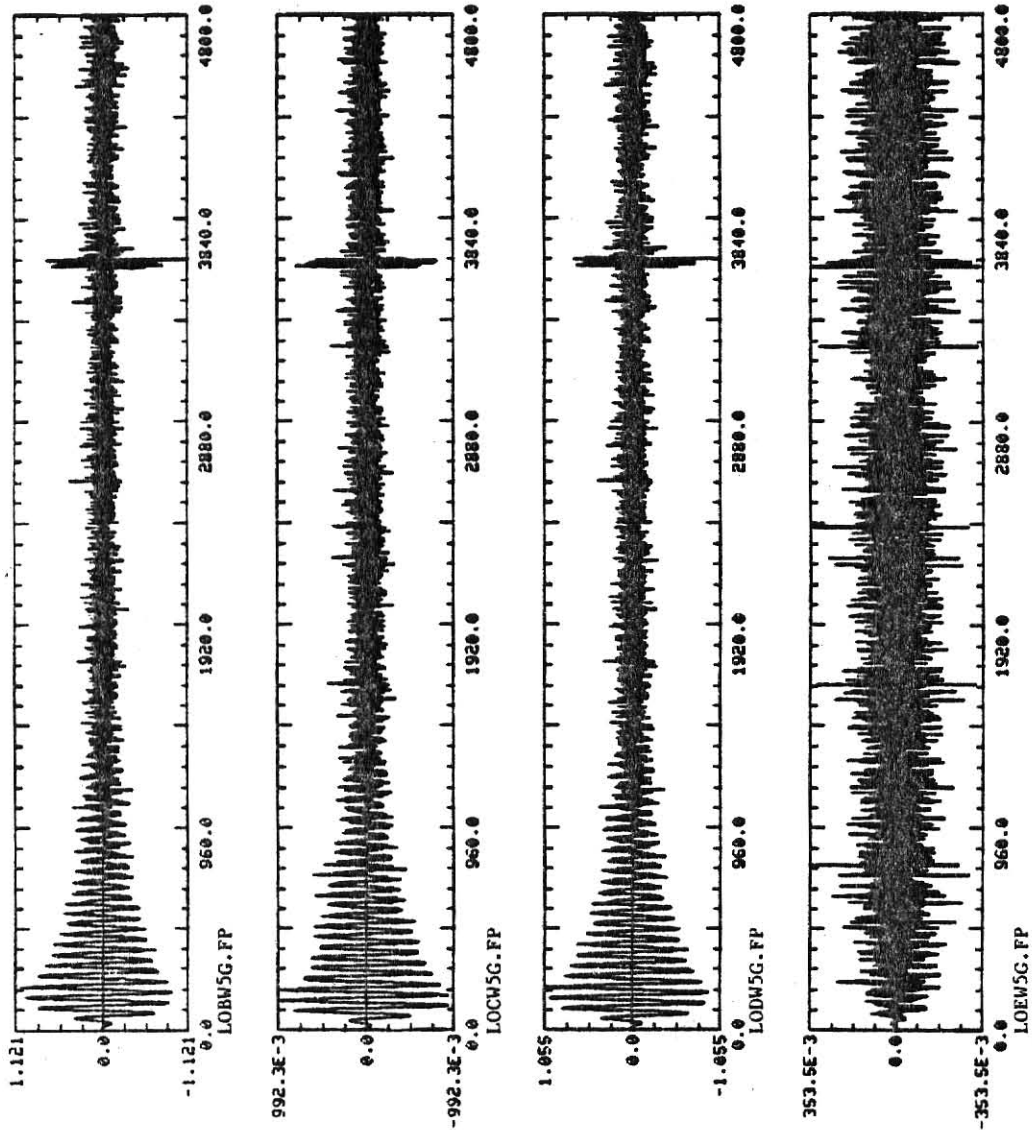
LOW BAND: +-0.05 input with noise, double precision



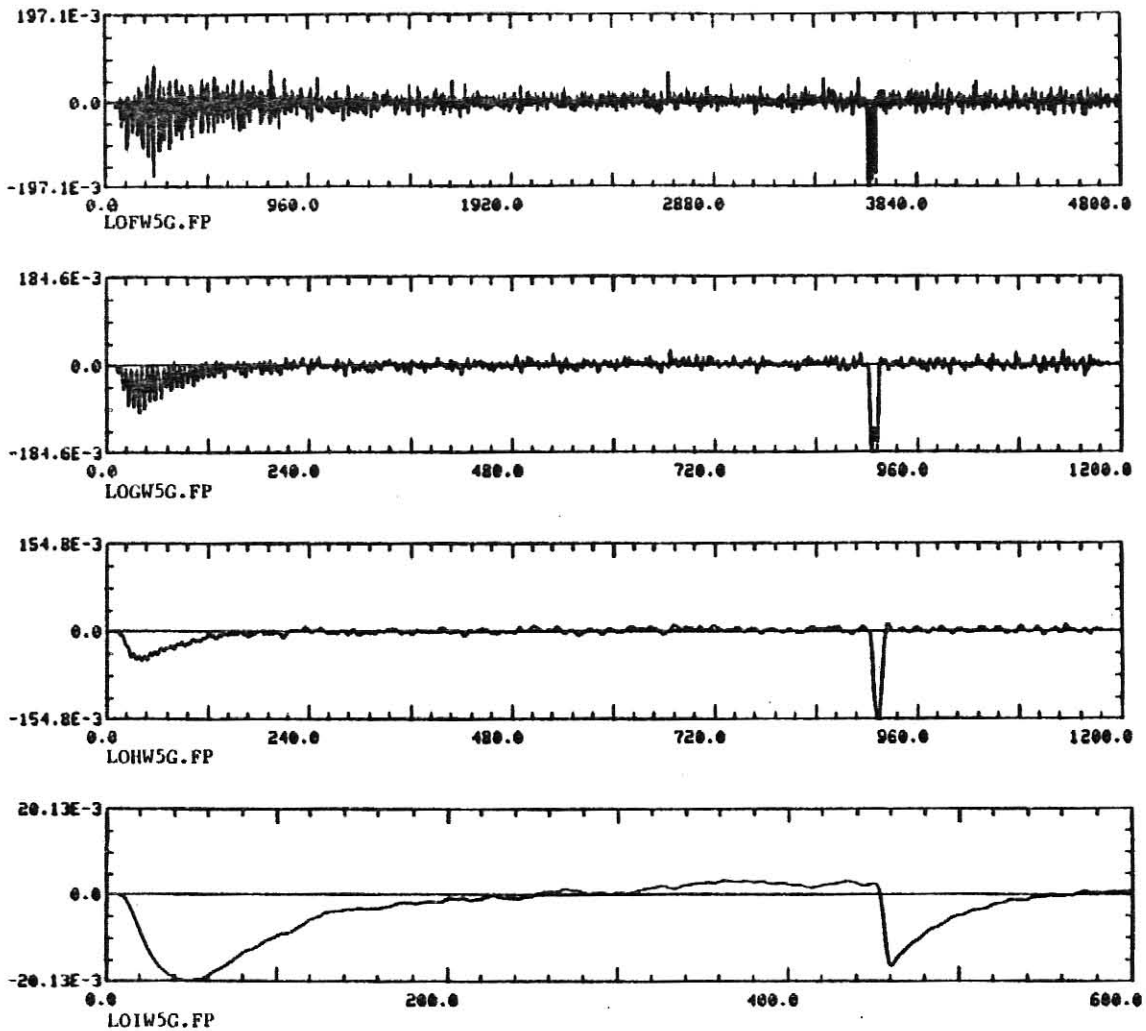
LOW BAND: +-0.05 input with noise, double precision



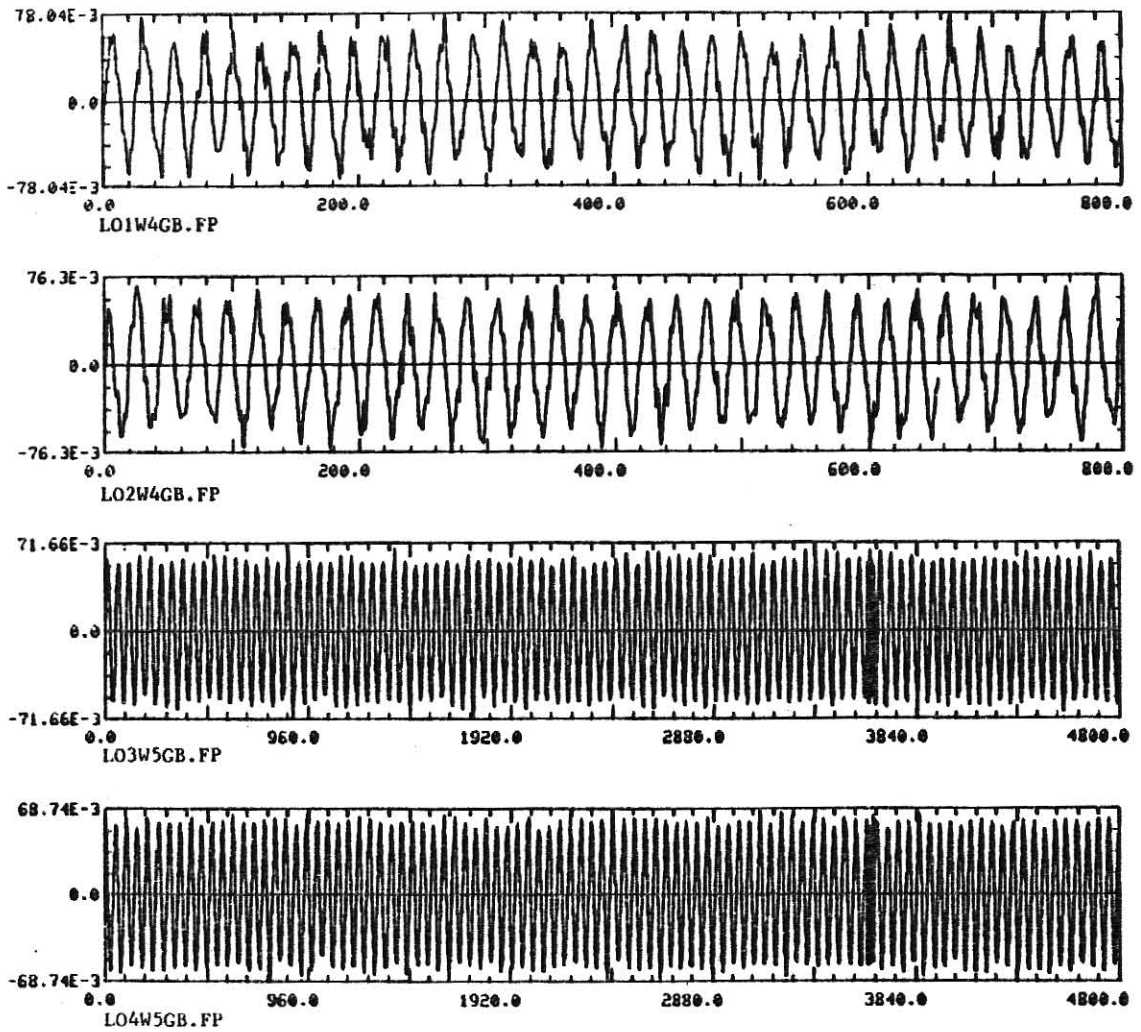
LOW BAND: +-0.05 input with noise, double precision



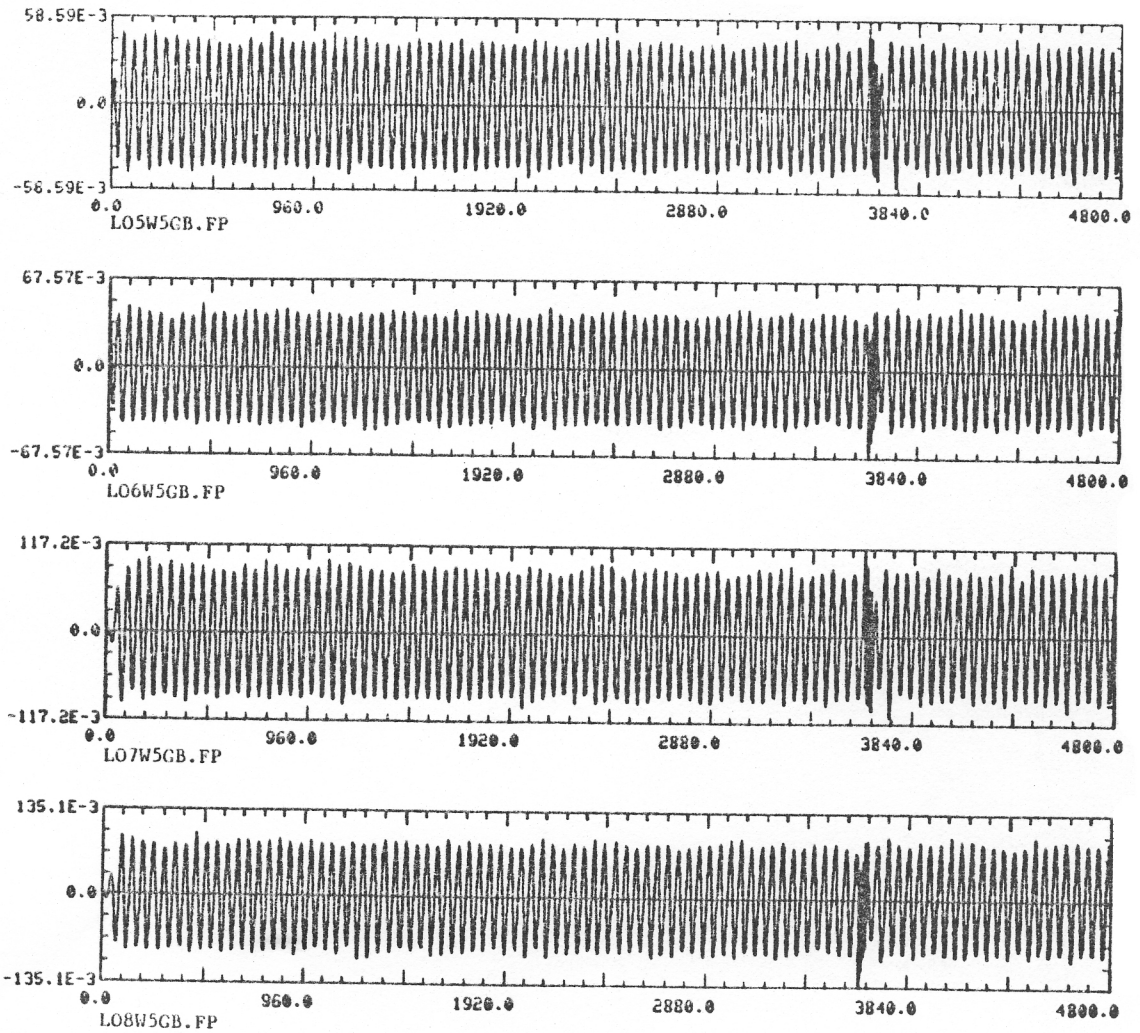
LOW BAND: +-0.05 input with noise, double precision



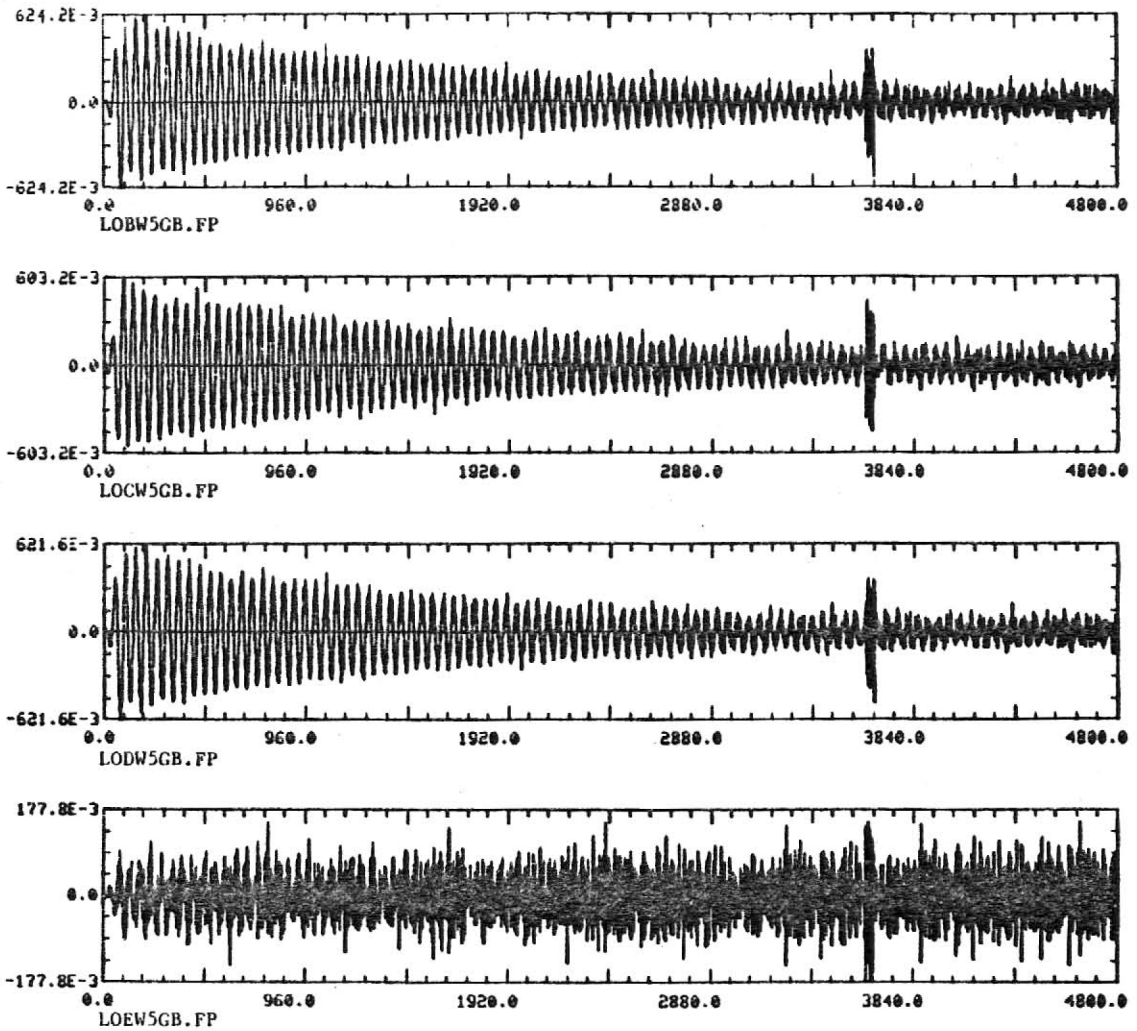
LOW BAND: +-0.05 input with noise, fixed point



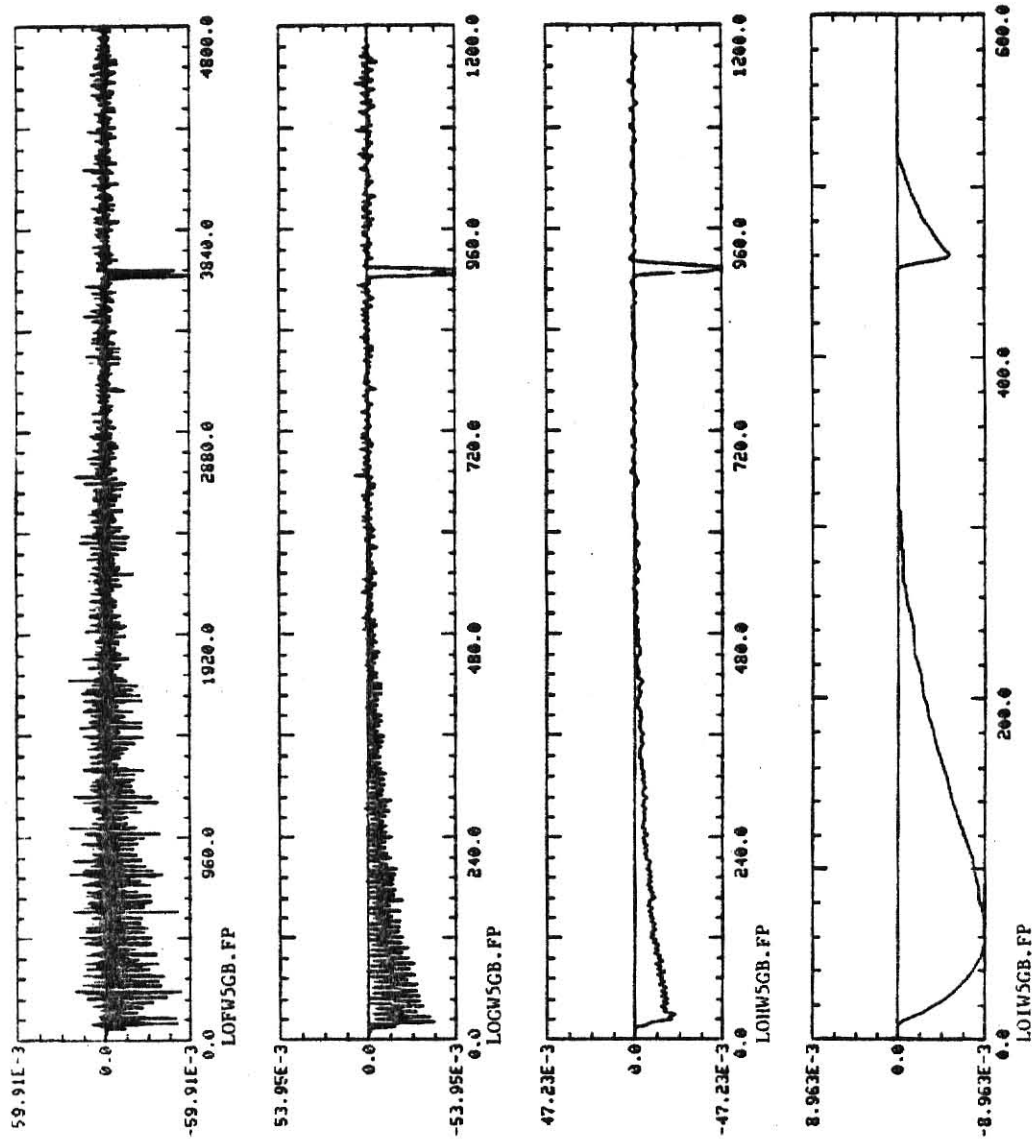
LOW BAND: +-0.05 input with noise, fixed point



LOW BAND: +-0.05 input with noise, fixed point



LOW BAND: +-0.05 input with noise, fixed point



REFERENCES

1. Ahmed, N. and Natarajan, T., Discrete-Time Signals And Systems, Reston Publishing Co., Inc., 1983, pp. 313-387.

2. Antoniou, A., Digital Filters: Analysis And Design, McGraw-Hill Book Co., 1979, pp. 274-305.

3. Bogner, R. and Constantinides, A., Introduction To Digital Filtering, John Wiley & Sons, 1975, pp. 157-184.

4. Liu, B., "Effect of Finite Word Length on the Accuracy of Digital Filters -- A Review," IEEE Trans. On Circuit Theory, Vol. CT-18, No. 6, Nov. 1971.

5. Oppenheim, A. and Schafer, R., Digital Signal Processing, Prentice-Hall, Inc., 1975, pp. 404-464.

6. Stearns, S., Digital Signal Analysis, Hayden Book Co., Inc., 1975, pp. 216-222.

7. Kaiser, J., "Some Practical Considerations in the Realization of Linear Digital Filters," Proc. 3rd Annual Alterton Conf., Oct. 1965.

A SIMULATION OF A MICROCOMPUTER-BASED INTRUSION DETECTION SYSTEM

by

JOHN WARREN BARTHOLOMEW

B. S., Kansas State University, 1982

AN ABSTRACT OF A MASTER'S THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Electrical Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1983

Computer simulations are useful in obtaining information about a system when input parameters are changed. This paper deals with the simulation of a microcomputer-based intrusion detection system and the effects of using a finite word length in the calculation of digital filters. The modifications made in the system as a result of the simulation improve the performance of the system.

An intrusion detection system is usually implemented on a microcomputer using digital filters when speed and low power are necessary. The system may be designed on a larger computer, and the digital filters optimized to reduce quantization errors due to using the finite word length on the microcomputer, but the effects of implementing the system on a microcomputer with a finite word length are best seen by doing a computer simulation.

This system was simulated in order keep the signal within the dynamic range of the fixed word length, and to make the best use of that dynamic range to reduce the signal to noise ratio and thus improve the performance of the system. A general procedure for designing and implementing a digital signal processing system was devised and relies heavily on the use of computer simulation to revise the system.