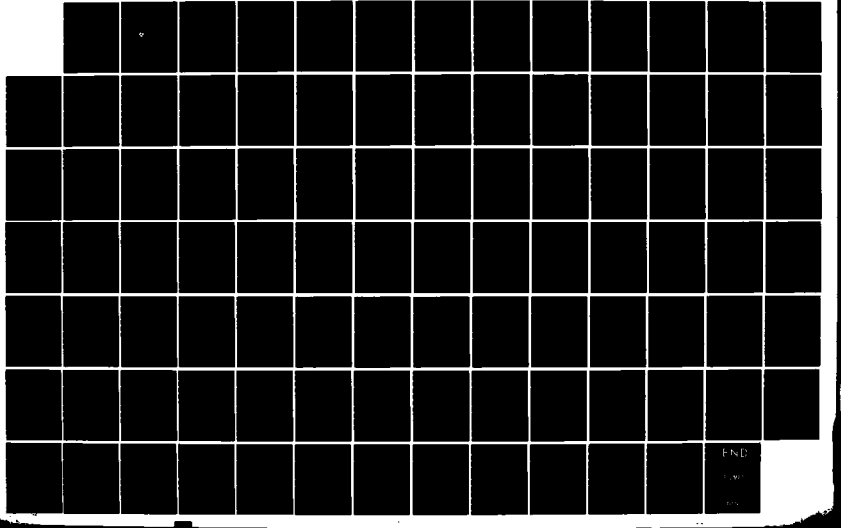
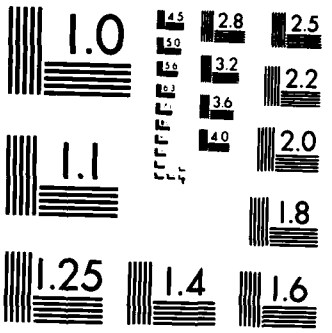
AD-A154 798 APPLICATION OF ADAPTIVE NOISE CANCELLATION TO COAST GUARD VOICE COMMUNICATIONS(U) COAST GUARD WASHINGTON DC OFFICE OF RESEARCH AND DEVELOPMENT B B PETERSON ET AL. 1/1

UNCLASSIFIED MAR 85 USCG-D-5-85 F/G 17/2 NL





Report No. **CG-D-5-85**

Application of Adaptive Noise Cancellation to Coast Guard Voice Communications

Prepared for:

U.S. Department of Transportation
United States Coast Guard

Office of Research and Development
Washington, D.C. 20593

NOTICE

This document is disseminated under the sponsorship of the Department of Transportation in the interest of information exchange. The United States Government assumes no liability for its contents or use thereof.

The contents of this report do not necessarily reflect the official view or policy of the Coast Guard; and they do not constitute a standard, specification, or regulation.

This report, or portions thereof may not be used for advertising or sales promotion purposes. Citation of trade names and manufacturers does not constitute endorsement or approval of such products.

Technical Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
CG-D-5-85		

4. Title and Subtitle	5. Report Date
Application of Adaptive Noise Cancellation to Coast Guard Voice Communications	March 1985
	6. Performing Organization Code
	8. Performing Organization Report No.
7. Author(s)	
B. B. Peterson, K. U. Dykstra, M. D. Sakahara	
9. Performing Organization Name and Address	10. Work Unit No. (TRAIS)
Department of Engineering U. S. Coast Guard Academy New London, CT 06320	11. Contract or Grant No.
	13. Type of Report and Period Covered
12. Sponsoring Agency Name and Address	Final Report
Department of Transportation U. S. Coast Guard Office of Research and Development Washington, DC 20593	14. Sponsoring Agency Code G-DST-3

15. Supplementary Notes

16. Abstract

A variety of approaches to the digital filtering of voice signals corrupted by background engine noise are presented. These approaches include the standard Least Mean Square (LMS) adaptive noise cancellation algorithm, an optimum fixed weight filter, a special type of notch filter and frequency domain adaptive noise cancellation. The filters have been implemented both off-line using FORTRAN programs on an LSI-11/2 microcomputer, and in real time using the Texas Instruments TMS 320 microprocessor and in PDP-11 assembly language using an LSI-11/2.

Frequency domain adaptive filtering was seen to be superior to the LMS time domain algorithm because its much greater computational efficiency allowed the analysis of much longer filters.

A digital filter that exploits the engine noise by having a notch at each harmonic was seen to be more effective than any of the adaptive filters. The digital filter equations are derived, starting with a particular periodic reference input. This adaptive filter is shown to be, in reality, an infinite impulse response, time invariant filter.

17. Key Words	18. Distribution Statement
noise cancellation digital filtering digital signal processing adaptive filters	This document is available to the U. S. public through the National Technical Information Service, Springfield, VA 22161

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
UNCLASSSIFIED	UNCLASSIFIED	90	

Form DOT F 1700.7 (8-72) Reproduction of completed page authorized

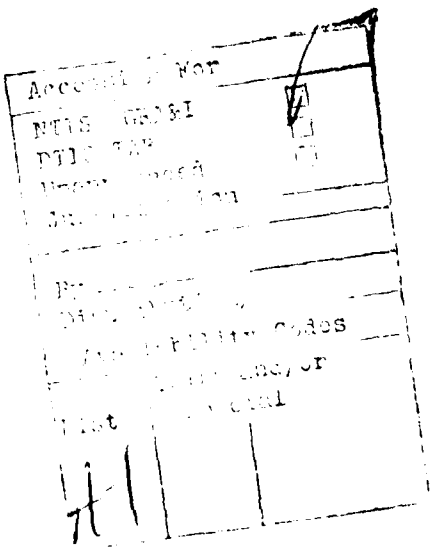
Application of Adaptive Noise Cancellation to Coast Guard Voice Communications

Final Report

March 1985

B. B. PETERSON, K. U. DYKSTRA
and M. D. SAKAHARA

Department of Engineering
U. S. Coast Guard Academy
New London, CT 06320



Abstract

A variety of approaches to the digital filtering of voice signals corrupted by background engine noise are presented. These approaches include the standard Least Mean Squares (LMS) adaptive noise cancellation algorithm, an optimum fixed weight filter, a special type of notch filter, and frequency domain adaptive noise cancellation. The filters have been implemented both off-line using FORTRAN programs on an LSI-11/2 microcomputer and in real time using the Texas Instruments TMS 320 microprocessor and in PDP-11 assembly language using an LSI-11/2.

Frequency domain adaptive filtering was seen to be superior to the LMS time domain algorithm because it's much greater computational efficiency allowed the analysis of much longer filters.

A digital filter that exploits the periodicity of the engine noise by having a notch at each harmonic was seen to be more effective than any of the adaptive filters. The digital filter equations are derived starting with an adaptive, finite impulse response filter with a particular periodic reference input. This adaptive filter is shown to be in reality an infinite impulse response, time invariant filter.

Table of Contents

I. Introduction 1

II. Experimental Methods 1

III. Summary of Approaches and Results 3

a. Analysis of LMS Adaptive Filters in FORTRAN 3

b. Analysis of Optimum Fixed Weight Filters 4

c. Real Time Adaptive Filtering Using the TMS 320 5

d. Notch Filters 6

e. Frequency Domain Adaptive Filtering 7

f. Adaptive Filtering of Automobile Engine Noise 7

IV. Publications and Presentations 11

V. Conclusions and Recommendations for Future Research 12

Appendixes

Appendix A: LMS Filters, Implementation and Performance A1

Appendix B: FORTRAN Program to Calculate Optimum Filter Weights B1

Appendix C: Implementation of an Adaptive Noise Cancellation Algorithm on the TMS 320 C1

Appendix D: Notch Filter Analysis and Results D1

Appendix E: Frequency Domain Adaptive Filtering E1

Appendix F: LMS Adaptive Filter FORTRAN Program where Audio Signal is Added in Software F1

References (Single list of references for both main body of report and appendixes) R1

I. Introduction

Background noise at the source has long been a problem affecting the intelligibility of Coast Guard voice communications. Recent studies of other researchers [7-10] indicate that additive noise will become a much more severe problem as the Coast Guard converts from analog systems to voice privacy systems using speech compression and digital transmission. In particular, Gold and Tierney at MIT's Lincoln Laboratory [7] have found that computer simulated F-15 cockpit noise added to clean speech reduces diagnostic rhyme test (DRT) [11] intelligibility scores to 92.6% without speech compression but to 75.2% when processed through a 2400 bit/second LPC-10 [12] speech compression algorithm.

Our efforts for the past eighteen months have focused on determining if digital filtering in general and adaptive noise cancellation in particular can improve the signal to noise ratio at the input to the system and thus the intelligibility of the received signal. A wide variety of approaches have been attempted with varying degrees of success. A summary of these efforts is contained in the main body of this report. The specific details including program listings are in Appendixes A-F.

II. Experimental Methods

The general technique employed has been to make stereo tape recordings of background engine noise both with and without a voice signal on one of the channels, and then to process and

in the laboratory. Recordings were made
my Luder yawl with the engine (a four
el) running, in the Academy Power
ith a single cylinder, four cycle,
running and most recently in the Power
ith a four cylinder Datsun 710 running.
D reel to reel tape deck donated by CDR
for the recordings. All recordings were
nd. General Radio Model 1952 Universal
pass anti-aliasing and reconstruction

ssing the recorded waveforms were of two
gments (less than one second) were
emory using a Digital Equipment
rocomputer and a Data Translation 2785
ilters were then implemented and the
e using FORTRAN programs.
s to implement the filter in real time
n an oscilloscope or to listen to the
one exception this was done using a
0 Evaluation Module (EVM). The notch
ndix D was implemented in real time on
the TMS 320 EVM. In our proposal [13]
real time implementation were proposed.
W TDC-1010 hardware multiplier/
ed to the LSI-11/2 computer but even with
slow to implement reasonable size

adaptive filters in real time. It was hoped to play the tape deck into the filter at a much slower speed than it was recorded, tape record the filter output and then play the output back at the original speed, but we were unable to obtain a tape deck with this capability. No attempt was made to use the Western Digital WP 3150 Programmable Digital Filter. We were unable to obtain a final data sheet and have concluded the announcement was a trial balloon and that the circuit never went into mass production. An overwhelming conclusion we have reached in the context of our work is that the TMS 320 is by far the best approach to audio frequency digital signal processing. The second generation of the circuit [14] was announced in February 1985 at the IEEE International Solid State Circuits Conference and will make even more sophisticated processing possible.

III. Summary of Approaches and Results

a. Analysis of LMS Adaptive Filters in FORTRAN

Our first approach was an off-line FORTRAN implementation of the standard LMS adaptive noise cancellation algorithm [1]. Several such programs were written, one of which is described in Appendix A. When the inputs were highly correlated signals from laboratory signal generators the filter was seen to be very effective. When the inputs were two microphone recordings of engine noise there was virtually no improvement in signal to noise ratio. It is felt this was due to a combination of two problems. The filter length had to be kept short to insure the weights would converge within the sample size allowed by the 56

Kbyte random access memory of the LSI-11/2. This resulted in filter lengths much shorter than the fundamental period of the engine noise and very poor filtering. When the filter length was increased the weights would not converge within the allowable sample size. The recent acquisition of an LSI-11/23 microcomputer with 248 Kbytes of random access memory from the USCG Electronics Engineering Laboratory will permit the analysis of longer filters in the future.

b. Analysis of Optimum Fixed Weight Filters

It was next attempted to eliminate the slow weight convergence problem by calculating the optimum (in a linear least squares sense) fixed weight filter for the particular data set and then calculating the output using these fixed weights. A description and listing of the FORTRAN program to accomplish this is contained in Appendix B. Improvements in signal to noise ratio of 6-8db were possible when the sampling rate was reduced to 1-2 kHz and the anti-aliasing filters adjusted accordingly. This effectively increases the filter length in time to approximately the period of the noise. Although these low sampling rates are unacceptable for voice communications the intent was to determine if longer filters would be effective at the necessary higher sampling frequencies. It is also believed there were numerical problems in the Gauss elimination subroutine when attempting to analyze longer filters. The subroutine should have used double precision arithmetic [15] but single precision was used due to limited memory and because double precision floating point instructions are not part of the assembly language

instruction set on the LSI-11/2 [18] and would have been slow to execute. The LSI-11/23 with it's larger memory and double precision instructions in assembly language will allow analysis of longer filters.

c. Real Time Adaptive Filtering Using the TMS 320

At this point (November 1983) a TMS 320 Evaluation Module was procured and efforts shifted to real time implementation. The first version was done by then Cadet 1/c now ENS M. D. SAKAHARA as his Systems Design and Synthesis project. His report was included in the July 1984 interim report [16]. His filter had 40 weights and would operate at sampling frequencies of up to 8.5 kHz.

A second version (Appendix C) used BASIC on the Dartmouth Time Sharing System to write much more efficient TMS 320 code and resulted in a 68 weight filter that would operate at sampling rates of up to 10.7 kHz. The essential difference is that all the looping is done in BASIC at the assembly language generation level resulting in straight line code that executes at the maximum benchmark speed of 1.2 microseconds/weight [17]. The first version used conditional branching and executed at 2.6 microseconds/weight.

Extensive tests were done on recorded engine noise and the results are contained in Appendix C. The conclusion reached was that 68 weights are not enough for an 8 kHz sampling frequency. The second generation TMS 32020 [14] has 544 words of on chip data memory compared to 144 for the TMS 32010, efficient access to 64K words of external memory, and has combined some operations

that were two instructions into a single 200 ns instruction. Adaptive filters of at least 120 weights at 8 kHz should be possible.

d. Notch Filters

The highly periodic nature of the engine noise implies that any reference input that contains all the harmonics of the noise in the primary input should suffice. A reference input of unit impulses periodic at the same frequency as the engine noise contains these harmonics and results in very efficient algorithm allowing filters of virtually arbitrary length. Synchronization is obtained by triggering the A/D converter with an optical shaft encoder mounted on the engine.

Further analysis of the algorithm revealed that what started as an adaptive, finite impulse filter was in fact a fixed weight, infinite impulse response filter. Preliminary results of using the filter were contained in the interim report [16] and a detailed analysis is contained in Appendix D.

This filter appears to be the most promising at this stage. It is intended to further evaluate it's performance by making a series of comparative diagnostic rhyme tests (DRT's) [11]. Comparisons of intelligibility will be made among filtered and unfiltered speech using both an omnidirectional microphone and a Shure Model 562 noise canceling microphone [19-20]. Due to the failure of the motor generator set and therefore the lack of DC power it has not been possible to start the engines in the Power Engineering Laboratory since September 1984. We have designed and constructed a high power DC supply and hope to have engines running in the near future.

The LMS filter was implemented through simulation on the LSI-11/2 computer. Since the LSI-11 could not execute the adaptive algorithm fast enough for real time applications, data sets were obtained and the adaptive algorithm applied to the data sets.

A Fortran IV program was written to implement the adaptive filter and used assembly language subroutines to communicate with the real world (see flow chart). The input data set was placed into two input arrays, one for the samples with noise (IN1) and the other for samples containing noise plus the desired signal (IN2).

The program first implements a fixed weight filter on the reference input and subtracts the result from the primary input. The output of the adaptive filter was plotted and displayed on the video terminal. This result was applied to the adaptive algorithm to adjust the weight vector. Once the adaptation cycle is completed, the computer then gets the next input and starts the process all over.

When all of the primary samples are used, the program plots the impulse response of the filter and calculates the frequency spectrum if the weights. Both phase and magnitude were plotted. Since the adaptive cycle was applied after the fixed filter, the plotted weight vector has gone through an extra adaptation cycle.

In the strictest case, the performance of the plotted weights is unknown. However, this dose not pose a significant problem since the weights change very little once they have converged. This is evident in the plots obtained from the program (see Appendix 1). Once the weights converged on the LMS solution, they change very little.

The entry of pertinent constants is executed at the beginning of each run. On the first run, the input channels (NC1 and NC2) were selected and the number of samples (NUMS), delays (NUMD) and weights (NUMW) for the filter entered. The adaptive coefficient (ADAPT) was entered and the weight (WEIGHT) and input arrays were reset as necessary. The data entry segment of the program was placed at the end of the program to help speed execution, however it did not have a significant effect.

Implementation

The least mean square (LMS) algorithm of the adaptive filter was used since it was one of the more simple forms. It has the form of

$$W(k+1) = W(k) + u\ Y\ U(k)$$

where W(k) is the weight vector, Y is the output of the adaptive filter, U(k) is the reference input array and u is the adaptive coefficient.

The reference signal was filtered and then subtracted from the primary signal. The result is treated as an error signal for the filter. For a positive output, the adaptive algorithm will tend to increase the magnitude of the weight vector. The output of the filter will increase causing the output of the adaptive filter to decrease or tend negatively. When the output goes negative, the algorithm will again tend to alter the weights to decrease the output. This tendency to minimize the output is where the LMS algorithm gets its name.

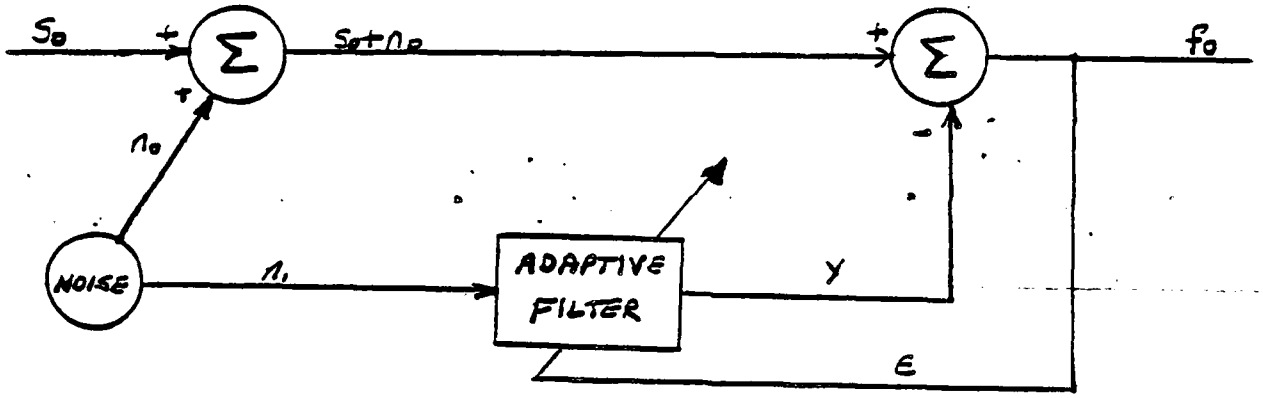
It would seem that if this filter were working in the ideal case, the weights would converge to completely cancel the output. The adaptive coefficient is what prevents this from happening. The magnitude of the adaptive coefficient must be small. Since the adaptive coefficient has a direct affect on the rate of convergence of the weights, a small value for u would reduce the tendency to completely cancel the output and yet suppress most of the interference.

The cancellation of the noise in the primary input is accomplished by reshaping the reference input, through filtering, to reflect the noise in the primary input. If the filter is to adjust its characteristic to accomplish this reshaping, it would be desirable for the characteristic to change slowly. This avoids oscillating around the desired response as the case would be if the weights were allowed to converge too quickly. A large adaptive coefficient may reduce the effectiveness of the overall filter to the point of uselessness (greater than $2.0 \cdot 10^{-9}$ in this application) or cause the filter to become unstable and blow-up (u greater than 2.0). A coefficient that is too small will impede the weights from converging or even completely preventing them from reducing any noise in the primary input. A compromise must be reached between rate of convergence and accuracy of the convergence in the proper selection of u. The adaptive coefficient must be small enough to allow the filter to accurately reshape the reference input and allow the weights to converge in a reasonable time.

Introduction

A common method of estimating a voice signal distorted by additive noise is to pass it through a filter that tends to reduce the noise level in the signal. This filter can either be fixed or adaptive. The fixed filter requires a prior knowledge of the signal and noise in order for it to be effective.

The adaptive filter does not require any or little knowledge of the signal or noise. It has the ability to alter its impulse response automatically to suppress the noise from the signal. The adaptive algorithm used in this application has two inputs: a primary input that contains the signal (s_1) and noise (n_1) and a reference input that contains noise (n_2) that is somehow correlated with the noise (n_1) in the primary signal The reference input should contain none or very little of the signal since the filter may converge to cancel to signal as well as the noise in the primary input.



BLOCK DIAGRAM OF ADAPTIVE FILTER

The adaptive filter can be used to eliminate the 60 Hz hum in EKG's. The 117vac outlet can be used as the reference input and the normal chest leads as the primary input. In a similar application, the mother's heartbeat can be filtered away from an infant's EKG. The reference leads are placed near the mother's heart and the primary leads are placed on the mother's abdomen. Interference entering the side lobes of a receiving antenna can be suppressed by using a circular array of primary antennas and a reference antenna chosen in the area of the incoming interference. In voice communications, where background noise is a problem, two microphones may be used. The operator speaks into the primary microphone while the reference microphone is place in the relative vicinity of the operator picking up the background noise.

Table of Contents

Abstract	2
Introduction	4
Implementation	5
Data and Results	7
Interface design of TRW's TDC1017J to the LSI-11	9
Flow chart of Fortran IV program	11
Appendix I	13

Abstract

An LMS adaptive filter was implemented on the LSI-11/2 computer. The computer could not process fast enough for a real time application so the filter was simulated on input data sets already contained in the computer's memory. The filter used two inputs: a primary input that contained noise and a desired signal and a reference input that contained noise that was somehow correlated with the noise in the primary input. The reference input was filtered and subtracted from the primary input suppressing the noise. The filter produced signal to noise improvements of around 20 dB in most cases and worked quite well with and adaptive gain of $5.0 \cdot 10^{-4}$.

Appendix A

LMS FILTERS IMPLEMENTATION AND PERFORMANCE

By Cadet 1/C Michael Sakahara

Instructor : LCDR Benjamin B. Peterson

Fall 1983

Department of Applied Sciences
United States Coast Guard Academy
New London, Ct. 06320

affect of background noise on the parameters internal to the algorithm and this is considered essential to basic understanding of the problem.

cancellation compared to noise canceling microphones but adaptive noise cancellation where the primary input is a noise canceling microphone vs a single noise canceling microphone. In related work for the Air Force researchers at Bolt, Beranek, and Newman, Inc. [22] have experimented with various multisensor configurations and have found a combination of a noise canceling microphone at the lips and an accelerometer attached to the throat to be best. The accelerometer is insensitive to environmental noise but is bandlimited to 2 kHz. The accelerometer output was added to the microphone output high pass filtered at 1200 Hz. Also, in work for the Navy, Ketron, Inc. [23] has developed a second order gradient noise canceling microphone (NC-104LF). Their tests have shown substantially improved performance over first order gradient microphones.

As noted in the introduction, background noise is a more serious problem in digital voice privacy systems employing the LPC-10 speech compression algorithm [12]. Therefore, future efforts should incorporate the LPC-10 algorithm and measure intelligibility after synthesis. Cadet 1/c P. A. MENCEL is presently attempting to implement LPC-10 in FORTRAN IV on an LSI-11/2. We have also requested the PDP-11 FORTRAN 77 program referred to in [12] from the National Security Agency and we hope to modify it for running on an LSI-11. These programs will not run in real time as required for DRT's. The best method of real time implementation would be a TMS 320 based coprocessor board for an LSI-11. An alternate method would be to acquire two digital voice privacy units but this would not allow access the

weight FIR and IIR filters, plot their impulse and frequency responses and automatically generate TMS 320 assembly language code for downloading and real time implementation. The program for FIR design is that described in [21] and installed on DTSS when PROF WOLCIN joined the Academy faculty in 1984. Cadet 1/c B. M. LAM is presently modifying the program to produce TMS 320 code for real time implementation.

V. Conclusions and Recommendations for Future Research

We have demonstrated that under some conditions digital filtering can improve the signal to noise ratio of speech corrupted by background engine noise. There are several questions suggested by our research that remain unanswered.

Does improved intelligibility result from this improved SNR? Diagnostic rhyme tests [11] are necessary to resolve this. DRT's are quite simple and straightforward but will require many man-hours of effort. Cadet projects may be the best method of performing DRT's. How intelligibility is measured is worth learning and the data analysis is a good exercise in undergraduate statistics.

The type of analysis done in the context of this project has not lent itself to a good comparison of adaptive noise cancellation vs noise canceling microphones. Again, this sort of question can only be answered using DRT's. The general consensus among experts in the field is that noise canceling microphones are essential in high ambient noise environments independent what else is done. Therefore the basic question is not adaptive noise

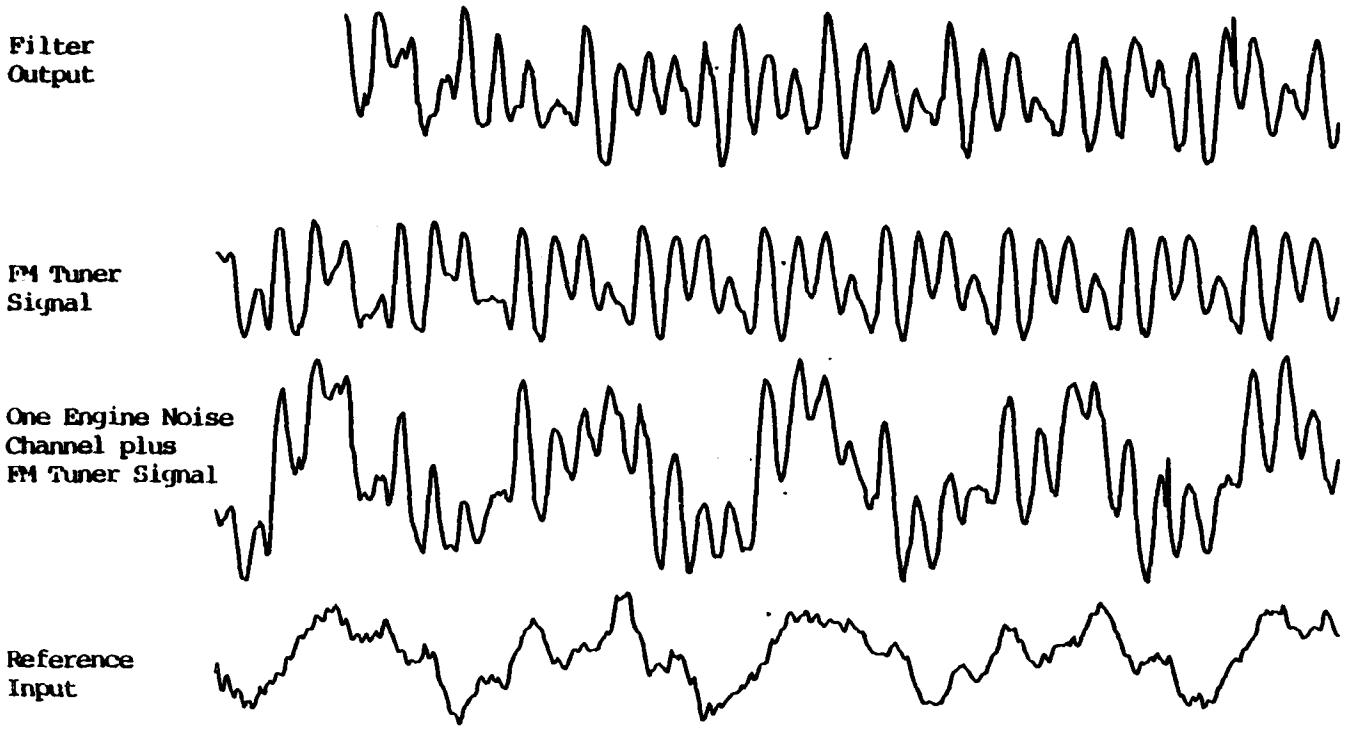
Conversations with other researchers in the field indicate that improved signal to noise ratio may not necessarily imply improved intelligibility and that DRT's are necessary to evaluate a proposed system. The human auditory system is very sophisticated and can extract information from highly corrupted signals but non-linear processing may destroy cues the human system is using not improve intelligibility.

IV. Publications and Presentations

Presentations of papers based the research have been made at two IEEE sponsored conferences and one local IEEE meeting to date and a fourth abstract has been submitted for presentation at a future conference. As noted in [16] M. D. SAKAHARA presented his paper "Adaptive Noise Cancellation" at Electro '84' at Boston in May 1984. He also presented "Real Time Adaptive Noise Cancellation" at the April 1984 IEEE New London Subsection meeting. This paper was included in [16].

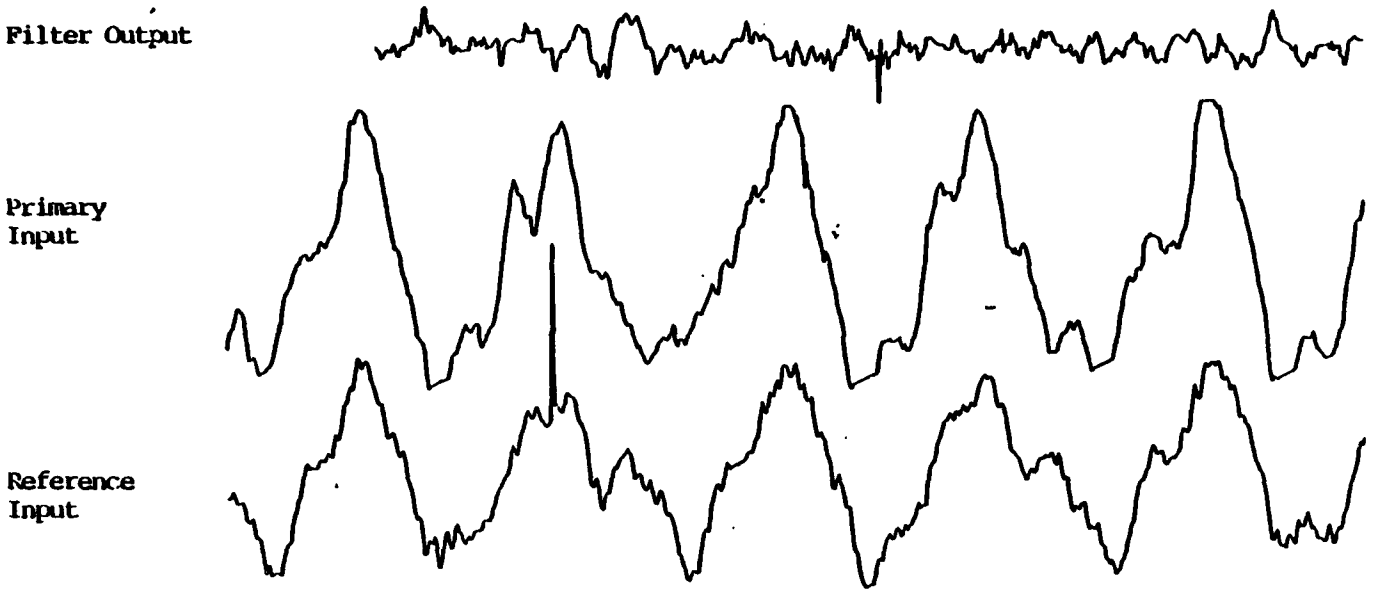
B. B. PETERSON presented "Audio Frequency Adaptive Filtering Using the TMS 320 Microprocessor" at the IEEE Digital Signal Processing Workshop at Chatham, MA in October 1984. In addition K. U. DYKSTRA and M. D. SAKAHARA were coauthors of the published summary and LCDR DYKSTRA attended.

An abstract titled "An Integrated Approach to the Design and Implementation of Digital Filters" has been submitted to the IEEE/ASEE Frontiers in Education Conference at Golden, CO in October 1985. In addition to the BASIC program in Appendix C, the paper will describe interactive programs to design fixed



Results of Adaptively Filtering Automobile Engine
Noise when Audio Signal is added in Software

Figure 2



Results of Adaptively Filtering Engine Noise
No audio signal added

Figure 1

filtering the reference input with an 80 weight filter and subtracting. The sampling frequency was 8 kHz.

The program was then modified to sample a third channel which was the audio output of an FM tuner. In software this signal was then amplified and added to the primary input and resultant signals run through the adaptive filter algorithm. Appendix F is this FORTRAN program. A listing of the graphics package written to drive the DEC Gigi Graphics Terminal and the HP 7470A Plotter is not included because of it's length and because the subroutine calls are obvious. This package was the most extensive programming effort undertaken in the context of this research and the excellent work of LTJG W. M. DUPRIEST is acknowledged.

Unlike previous efforts, it is now possible to know exactly what is signal and what is noise. Figure 2 is a sample plot of the results. The signal to noise ratio has been improved from -3.3 db to +6.0 db. Improvements in signal to noise ratio were consistently near 10 db. Reductions in noise power of 20 db as noted when no signal was added (Figure 1) are not expected because what is signal to the overall filter algorithm is actually noise to the weight update algorithm. The fluctuation in weights caused by the larger output degrades the filter performance. This effect is minimized by reducing the adaptive gain at the expense slower convergence and poorer tracking in time varying conditions.

What these recent results do indicate is that adaptive filtering can work very well when the noise is highly correlated.

e. Frequency Domain Adaptive Filtering

Just as their fixed weight counterparts, frequency domain adaptive filters require only a fraction of the arithmetic operations required in a time domain filter of equivalent complexity. The frequency domain adaptive filter algorithm described in [5] was implemented in FORTRAN for filter lengths of up to 512 samples. Due to filter length and good convergence properties the filter performance was better than that of the adaptive filters in a. and c. above but not as good as the notch filter in d. Detailed analysis of the filter is contained in Appendix E.

DRT's using a real time version of the algorithm would be necessary to make a more complete evaluation. The TMS 320 is fast enough but due to limited memory, implementation with our existing hardware is not possible.

f. Adaptive Filtering of Automobile Engine Noise

Our most recent effort was to make a two microphone recording of engine noise from a Datsun 710 with no muffler running at idle (approximately 1000 rpm) in the Power Engineering Laboratory. Due to the faulty exhaust system, it was necessary to operate the ventilation system at full power adding another significant component of noise.

When the data was plotted in the laboratory, the two signals were seen to be much more correlated than had been observed in previous recordings. Figure 1 shows two signals and the results of adaptive filtering with no voice signal added. 98.9% of the power in the primary input has been canceled by adaptively

Results and Conclusions

Pure signals (sine, square and triangle) were used to simulate noise and desired signal. A square wave was used to simulate noise since it contained the most harmonics and could be shaped to any of the noise signals placed in the primary input.

Three Wavetek signal generators were used to supply the noise, signal, and clock pulses for the sampler. Hewlett-Packard frequency counters and oscilloscopes were used to monitor the inputs and outputs of the computer simulation. In the simulations, the noise in the reference and primary inputs were of the same frequency and phase. This simplifies predicting the impulse response the filter should converge to allowing for an evaluation of the filter's performance.

The primary input contained a square wave as the desired signal. This enables the calculation of signal to noise ratios (SNR) based on the amplitudes of the noise and the desired signal. The SNR improvement of the filter could be estimated and the performance of the filter evaluated based on the SNR improvement.

A triangle wave represented the noise in the primary input. This simplified predicting the impulse response the filter should converge to. The Fourier series of a square of unity amplitude is

$$4/\pi \ (\sin wt + 1/3 \sin 3wt + 1/5 \sin 5wt + \ldots)$$

and a triangle wave of same amplitude

$$8/\pi \ (\sin wt - 1/9 \sin 3wt + 1/25 \sin 5wt - \ldots).$$

In order to change the square wave in the reference input into the triangle wave in the primary input, the square wave must be multiplied by another square wave of the same frequency.

When the weights were plotted, this indeed seemed to be the result. There were small deviations in the impulse response of the filter and was evident in the incomplete cancellation of the noise in the primary input. Although noise cancellation was not total, SNR improvements of up to 32 dB were observed.

The adaptive coefficient was varied and had a proportional affect on the convergence time of the filter. When u was $1.25 \cdot 10^{-3}$, the filter converged in 12 msec. When u was $4.0 \cdot 10^{-4}$, the convergence time was 45 msec or it took about four times as long. In both cases, SNR improvements of 27 dB was observed.

When u was increased beyond $2.0 \cdot 10^{-8}$, the filter seemed to have no effect on the primary input. Since the adaptive coefficient was so large, the weights constantly over shot the desired solution and did not reduce the noise levels in the primary input. The adaptive gain was not sufficiently high enough to cause instability, just large enough to cause excessive oscillations. When the adaptive gain was increased beyond 2.0, the filter quickly became unstable and blew-up. This was the threshold where the weights were allowed swing so far that they overcame the reducing tendencies of u.

The advantages of an adaptive filter were especially evident in the case when the noise and signal were similar in frequency. A run with the noise at 500 Hz and the desired signal at 750 Hz produced a SNR improvement of 32 dB!

The LMS filter seemed to suffer when the noise was at a significantly lower frequency than the signal. The filter would " glitch " every time the triangle wave would change slope. The filter would quickly converge to the desired signal while the slope was constant, however would loose track when the slope changed. this is due to the fact that the LMS filter follows the gradient of the noise. A low frequency triangle wave has a low gradient while scribing constant slope, however the gradient greatly increases when the triangle wave changes slope.

Over all, the LMS algorithm performed quite well in this application. It produced SNR improvements of around 20dB and exceeded 30 dB in some cases. The adaptive coefficient was critical in the performance of the system. The lower frequencies tended to require higher adaptive gains to optimize the filter, while the higher frequencies required lower adaptive gains. The adaptive gains depended mostly on the contents of the reference input and the noise frequencies. The adaptive gains seemed to be independent of the desired signal. An adaptive gain of $5.0 \cdot 10^{-9}$ worked the best for all cases of noise and signal combinations. The LMS filter worked well for most combinations of noise and desired signals.

Design of interface between LSI-11/2 and TRW's TDC-1010J

A cookbook approach was used in designing the interface between TRW's TDC-1010J Multiplier Accumulator chip and the LSI-11/2 computer. MDB Systems general purpose interface module (MSI-1710) was used to mount the TDC-1010J and supporting hardware.

The interface module provided input,output and address ports that were common on the Q-bus of the LSI-11. This simplified the decoding and interface logic greatly since bus protocol did not have to be considered. It was determined that the TDC-1010J could multiply and accumulate with in the time of a DATO cycle of the LSI-11. Thus the only timing consdierations made were in delaying signals going into the TDC-1010.

The Y multiplicand and the least significant 16 bits of the output share pins on the TDC-1010. Logic had to be provided to isolate the input and output ports on the interface module. Noninverting tri-state bus transceivers (Ti's 74LS243) were used to isolate the ports. The B side of the transceivers were connected to the TDC-1010J and the A side to the respective I/O location. The control inputs for the transceivers are Gab (L) and Gba (H). When Gab is asserted, the data is allowed to flow from the A side to the B side. When Gba is asserted, the data is alowed to flow from the B side, to the A side. Asserting both control inputs could result in destructive oscillations because the transceiver is trying to pass data in both directions at the same time. Not asserting both control inputs isolates side A from side B.

In the case of a Y input, the Gba input is held low not asserting it and Gab was asserted when the input data was valid. The reason Gba was held low was to prevent the possibility of sending the transceiver into destructive oscillations. By keeping Gba low, the transceiver can be alternated from isolation to passing data from A to B (the desired direction). The logic for the Gab input was (BDOUT • ADD3) (L). ADD3 is the address location indicating a Y-input on the bus and BDOUT is asserted when the data in the input port is valid. The Gab signal was used to supply the CLK Y signal. The three inverters delay the CLK Y signal to account for propagation delay of the transceivers.

Since the X-input shares the input port with the Y-input and control byte, it was isolated from the input port when not in use. This was accomplished by using tri-state noninverting bus drivers (Ti's 74367). This circuitry is not absolutely necessary because the X-input is loaded on to the TDC-1010J only when CLK X is asserted. But was added as a precautionary measure

only. The control logic for the bus drivers is (ADD2 • BDOUT) (L). ADD 2 is the address location indicating an X-input will appear on the input port and BDOUT is asserted when the data on the input port is valid. This control signal was used to supply the CLK X signal with inverters to delay to signal to account for propagation delays in the bus drivers.

The TDC-1010J has five data control bits:

Accumulate	(ACC)	bit 00
Subtract	(SUB)	bit 01
Preload	(PREL)	bit 02
Two's Complement	(TC)	bit 03
Round off	(RND)	bit 04

all asserted high. It was decided for versatility that these control bits be controlled by the LSI-11. The control bits are loaded onto the TDC-1010J at different times making it difficult or impossible to send them with the data going into the board. Type D flip flops (Ti's 7474) are used to store the control byte. The clock signal for the flip flops is (ADD1 • BDOUT) (H). ADD1 is the address location indicating that the control byte is to appear on the input port and BDOUT is asserted when the data on the input port is valid.

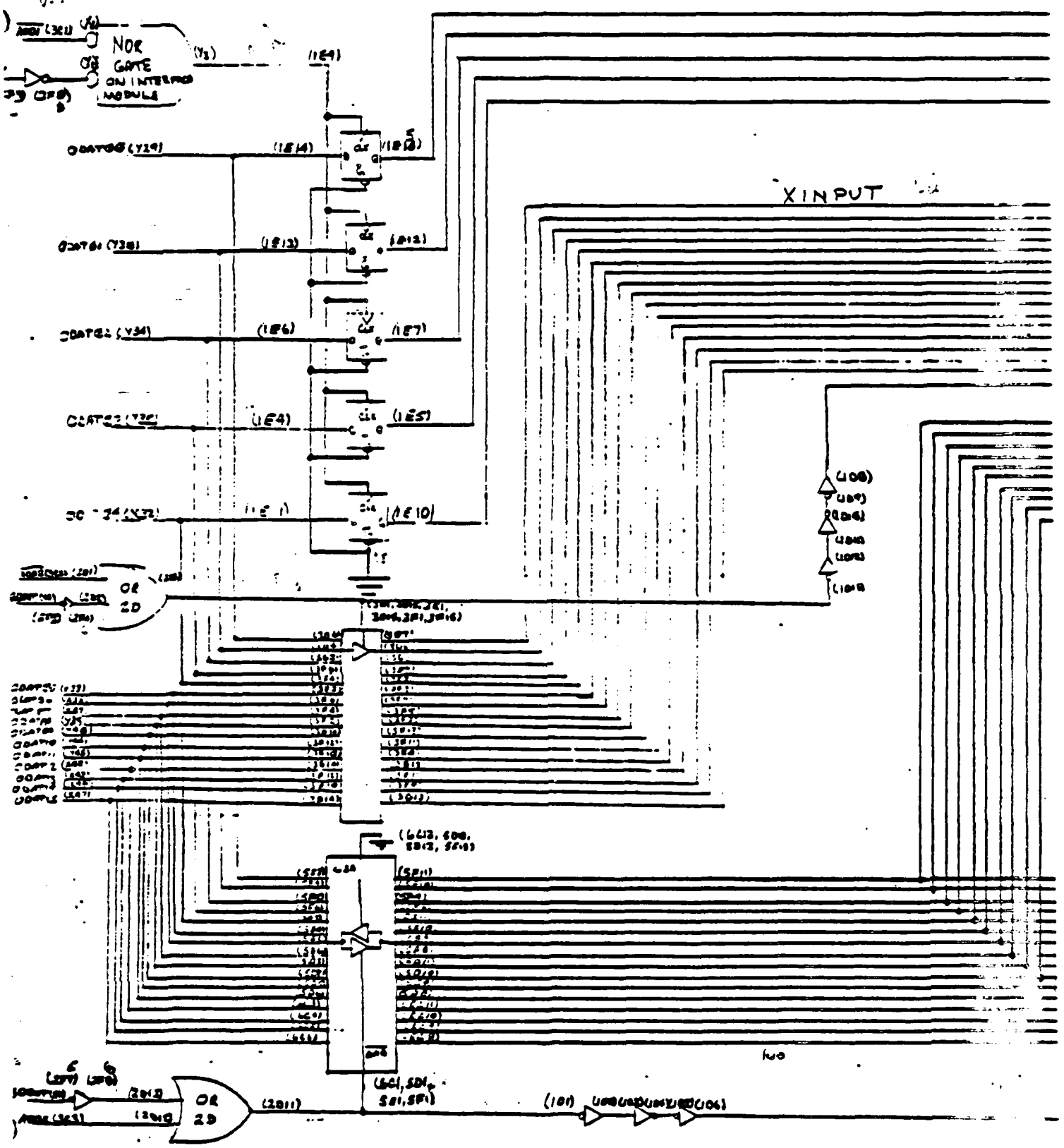
The output of the TDC-1010J is divided into three 16 bit words: the least significant product (LSP), most significant product (MSP) and extended control product (XTP). The LSP is time shared wit the Y-input and similar circuitry is used to isolate the input and output ports as in the Y-input. The tri-state output of the TDC-1010J is controlled by three inputs TSX, TSM and TSL. PREL must be held low if the TDC-1010J is to generate output. The control inputs are asserted low and only one of them may be asserted at a time for this design. A separate address has been allocated for each output word with TSM clocking the output to the output registers. The output of the address decoder on the interface module is asserted low and is connected directly to the appropriate output control pins. The interface module requires an input enable signal to indicate that the data on the output port is valid. This is accomplished by ORing the output control signals and using an inverter to account for propagation delays.

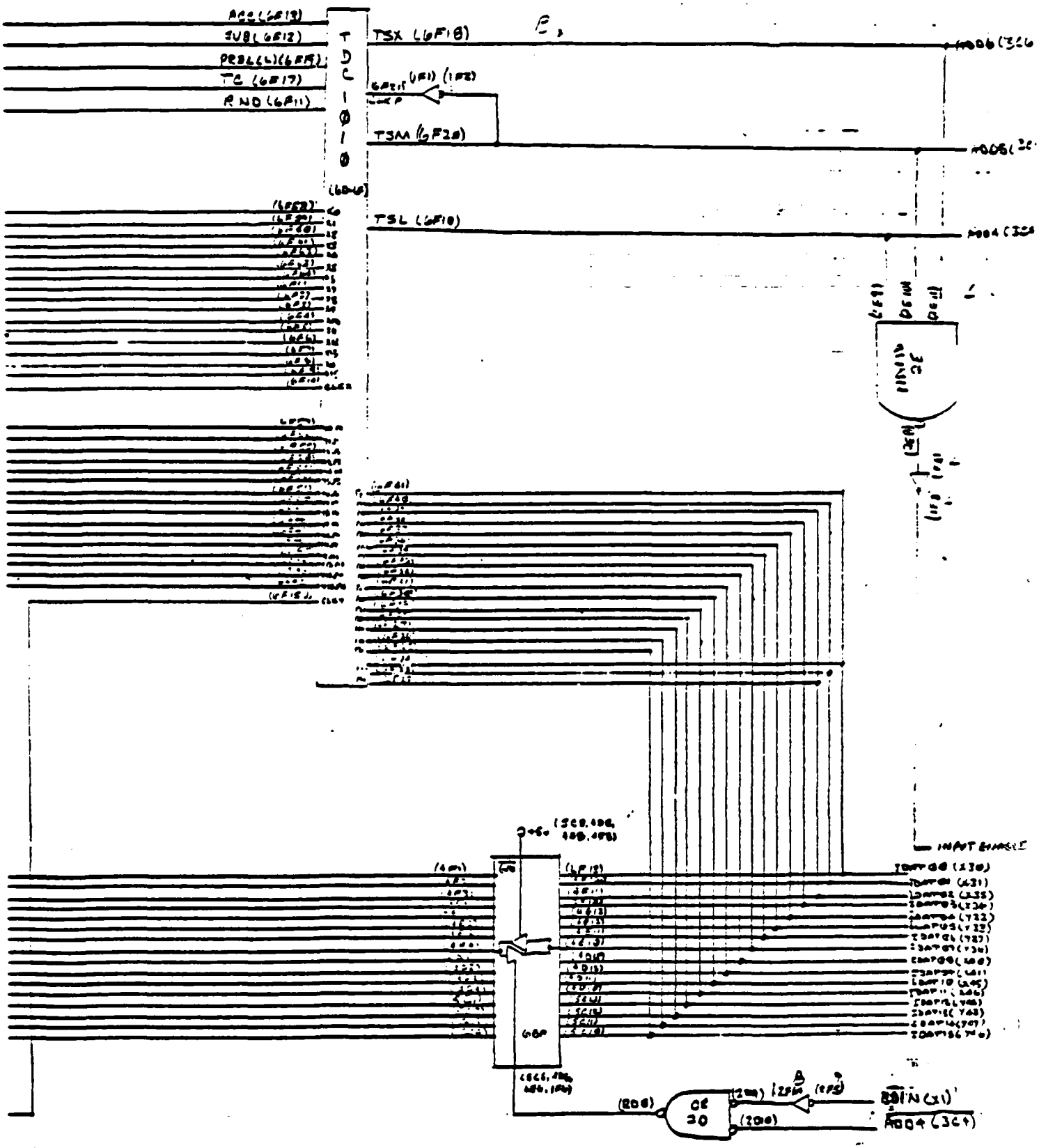
Six address locations were used to implement the TDC-1010J :

Address	Function number	Location
0	CTRL Byte	170000
1	X- Input	170010
2	Y- Input	170002
3	LSP Prod.	170012
4	MSP Prod.	170004
5	XTP Prod.	170014

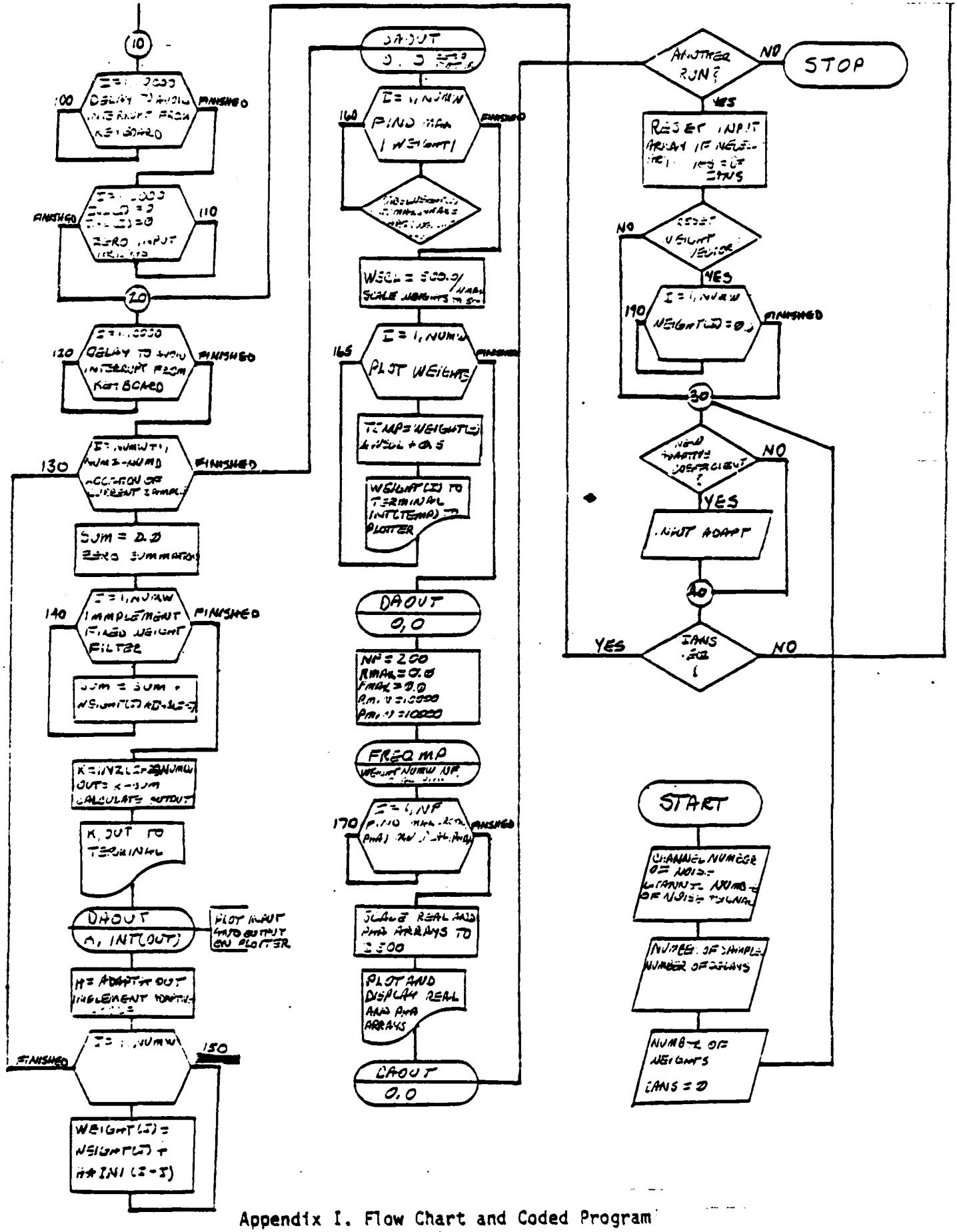
The unusual order of address locations is due to the address decoder on the interface module. The decoder an AND gate for 3 significant bits and a BCD to Decimal (Ti's 7442) for the lower three bits. Bits 01 and 02 are hard wired to the B and C inputs and the output of the AND gate is wired to the D input of the decoder. Bit 03 was wired to the A input and resulted in the unusual address order. Bit 03 was brought to the decoder to enable it to decode eight address instead of four.

To use the TDC-1010J , the output registers must be zeroed. This can be accomplished by sending a 0 to the X-input and multiplying without accumulation. The MSP must be outputted to a dummy location to clock the output registers. The other functions of the TDC-1010J can then be executed by altering the control byte as necessary. When ever output, accumulation or subtraction is desired, the MSP must be moved to a dummy location since asserting the MSP clocks the output clock and activates the accumulator and clocks the output registers.


NOR
GATE
ON INTERFACE
MODULE
XINPUT
OR
2D
OR
2B



NOTE: NUMBERS IN PARENTHESIS IDENTIFY LOCATION ON THE MLS1-1710 INTERFACE MODULE. THE FORM (1A7) INDICATES COLUMN 1 ROW A PIN #7 WITH #1 BEING IN THE UPPER LEFT PIN ON THE CHIP.



Appendix I. Flow Chart and Coded Program for LMS Adaptive Filter.

```
      C  PROGRAMMER: Michael D. Sakahara
      C
      C  DATE: November 13, 1983
      C
0001        PROGRAM ADAPT
      C  THIS PROGRAM IMPLEMENTS AN ADAPTIVE FILTER ALGORYTHM. THE FILTER
      C  IS IMPLEMENTED IN THE BEGINING OF THE PROGRAM AND THE USER INPUT
      C  PORTION OF THE PROGRAM APPEARS AT THE END OF THIS PROGRAM.
      C
      C
      C  INSTRUCTIONS:  WHEN ANSWERING YES TO ANY OF THE QUESTIONS, ENTER
      C     ANY VALID INTEGER, HOWEVER WHEN ENTERING NO ENTER A '1' ONLY.
      C
      C
      C
0002        DIMENSION WEIGHT(100), IN1(5000), IN2(5000), REAL(201), PHA(201)
0003        GOTO 50
      C
      C  ADD DELAY TO AVOID INTERRUPT FROM KEYBOARD
      C
0004     10 DO 100 I=1, 10000
0005    100 CONTINUE
0006        DO 110 I=1,5000
0007           IN1(I)=0
0008           IN2(I)=0
0009    110 CONTINUE
      C
      C  GET NUMS SAMPLES FROM A/D CONVERTER
      C
0010        CALL XTSMP2 (NUMS,NC1,NC2,IN1,IN2)
      C
      C  ADD DELAY TO AVOID INTERRUPT FROM KEYBOARD
      C
0011     20 DO 120 I=1, 10000
0012    120 CONTINUE
0013        DO 130 I= NUMW+1,NUMS-NUMD
0014           SUM=0.0
      C
      C  IMPLEMENT FIXED WEIGHT FILTER
      C
0015           DO 140 J=1,NUMW
0016                 SUM=SUM+WEIGHT(J)*IN1(I-J)
0017    140    CONTINUE
0018           K=IN2(I+NUMD)
0019           OUT=K-SUM
0020           WRITE (7,800) K,OUT
0021    800    FORMAT (1X,I8,G12.5)
      C
      C  OUTPUT OF FIXED WEIGHT FILTER TO D/A (INPUT, OUTPUT)
      C
0022           CALL DAOUT(K,INT(OUT))
      C
      C  IMPLEMENT ADAPTIVE FILTER
      C
```

```
0023              H=ADAPT*OUT
0024              DO 150 J=1, NUMW
0025                    WEIGHT (J)= WEIGHT (J) +(H*IN1(I-J))
0026    150       CONTINUE
0027    130   CONTINUE
0028          CALL DAOUT(0,0)
0029          READ (5,905) IANS
0030          IF ( IANS .EQ. 0 ) GOTO 25
      C
      C  PLOT WEIGHTS ON STRIP CHART
      C
0032          DO 160 I=1,NUMW
0033              IF (ABS (WEIGHT (I)) .GT. WMAX ) WMAX= ABS (WEIGHT (I))
0035    160   CONTINUE
0036          WRITE (7,810) WMAX
0037          WSCL=500.0/WMAX
0038          DO 165 I=1, NUMW
0039              WRITE (7,800) I, WEIGHT (I)
0040              TEMP=WEIGHT (I) * WSCL + 0.5
0041              CALL DAOUT (INT (TEMP), 0)
0042    165   CONTINUE
0043          CALL DAOUT (0,0)
      C
      C  PLOT FREQUENCY RESPONSE OF FILTER (MAGNITUDE AND PHASE)
      C
0044          NF=200
0045          CALL FREQMP (WEIGHT, NUMW, NF, REAL, PHA)
0046          RMAX=0.0
0047          PMAX=0.0
0048          RMIN=10000.0
0049          PMIN=10000.0
0050          DO 170 I= 1, NF
0051              IF ( REAL(I) .GT. RMAX ) RMAX=REAL(I)
0053              IF ( REAL(I) .LT. RMIN ) RMIN=REAL(I)
0055              IF ( PHA(I) .GT. PMAX ) PMAX=PHA(I)
0057              IF ( PHA(I) .LT. PMIN ) PMIN=PHA(I)
0059    170   CONTINUE
0060          RSCL=RMIN*(-1.0)
0061          IF ( RMAX .GT. (-1.0)*RMIN ) RSCL=RMAX
0063          PSCL=PMIN*(-1.0)
0064          IF ( PMAX .GT. (-1.0)*PMIN ) PSCL=PMAX
0066          PSCL=500.0/PSCL
0067          RSCL=500.0/RSCL
0068          DO 180 I=1, NF
0069              REAL(I)=(REAL(I)*RSCL)+0.5
0070              N1=INT(REAL(I))
0071              PHA(I)=(PHA(I)*PSCL)+0.5
0072              N2=INT(PHA(I))
0073              WRITE (7,810) REAL(I),PHA(I)
0074    810       FORMAT (1X,G12.5,1X,G12.5)
0075              CALL DAOUT (N1,N2)
0076    180   CONTINUE
0077          CALL DAOUT (0, 0)
      C
```

```
      C  SET UP FOR RERUN IF DESIRED
      C
0078          WRITE (7,900)
0079    900   FORMAT (1X, 'ANOTHER GO: 1=NO ?',$)
0080          READ (5,905) IANS
0081    905   FORMAT (I1)
0082          IF (IANS .EQ. 1) GOTO 1000
0084     25   WRITE (7,910)
0085    910   FORMAT (1X, 'GET NEW INPUTS ?',$)
0086          READ (5,905) IANS
0087          WRITE (7, 915)
0088    915   FORMAT (1X, 'RESET WEIGHT VECTOR ?',$)
0089          READ (5,905) IANS1
0090          IF (IANS1 .EQ. 1) GOTO 30
0092          DO 190 I= 1, NUMW
0093              WEIGHT(I)=0
0094    190   CONTINUE
0095     30   WRITE (7,920)
0096    920   FORMAT (1X, 'NEW ADAPTIVE CO-EFFICEINT ?',$)
0097          READ (5, 905) IANS1
0098          IF (IANS1 .EQ. 1) GOTO 40
0100          WRITE (7, 925)
0101    925   FORMAT (1X, 'INPUT NEW ADAPTIVE WEIGHT COEFFICIENT :',$)
0102          READ (5,930) ADAPT
0103    930   FORMAT (G14.7)
0104     40   IF (IANS .EQ. 1) GOTO 20
0106          GOTO 10
      C
      C  ACTUAL BEGINNING OF PROGRAM ( FIRST RUN )
      C
0107     50   WRITE (7,935)
0108    935   FORMAT (1X, 'ENTER CHANNEL NUMBER OF NOISE SOURCE :',$)
0109          READ (5,940) NC1
0110          NC1=256*NC1+16
0111          WRITE (7,936)
0112    936   FORMAT (1X, 'ENTER CHANNEL NUMBER OF NOISE+SIGNAL :',$)
0113          READ (7,940) NC2
0114          NC2=256*NC2+16
0115          WRITE (7,937)
0116    937   FORMAT (1X, 'ENTER NUMBER OF SAMPLES :',$)
0117          READ (5,940) NUMS
0118    940   FORMAT (I5)
0119          WRITE (7,950)
0120    950   FORMAT (1X, 'ENTER NUMBER OF DELAYS FOR FILTER :',$)
0121          READ (5,940) NUMD
0122          NUMD=NUMD-1
0123          WRITE (7, 955)
0124    955   FORMAT (1X, 'ENTER NUMBER OF WEIGHTS UPTO 100 :',$)
0125          READ (5,940) NUMW
0126          IANS=0
0127          GOTO 30
0128   1000   CALL DACOUT(0,0)
0129          STOP
0130          END
```

```
ORTRAN IV      Storage Map for Program Unit ADAPT

ocal Variables, .PSECT $DATA, Size = 053154 (11084. words)

ame    Type  Offset      Name    Type  Offset      Name    Type  Offset
DAPT   R*4   053052      H       R*4   053046      I       I*2   053016
ANS    I*2   053056      IANS1   I*2   053132      J       I*2   053036
       I*2   053040      NC1     I*2   053022      NC2     I*2   053024
F      I*2   053074      NUMD    I*2   053030      NUMS    I*2   053020
UMW    I*2   053026      N1      I*2   053126      N2      I*2   053130
UT     R*4   053042      RMAX    R*4   053102      RMIN    R*4   053112
SCL    R*4   053122      RMAX    R*4   053076      RMIN    R*4   053106
SCL    R*4   053116      SUM     R*4   053032      TEMP    R*4   053070
MAX    R*4   053060      WSCL    R*4   053064

ocal and COMMON Arrays:

ame      Type    Section Offset  ------Size------ Dimensions
N1       I*2     $DATA   000620  023420 ( 5000.) (5000)
N2       I*2     $DATA   024240  023420 ( 5000.) (5000)
HA       R*4     $DATA   051324  001444 (  402.) (201)
EAL      R*4     $DATA   047660  001444 (  402.) (201)
EIGHT    R*4     $DATA   000000  000620 (  200.) (100)

ubroutines, Functions, Statement and Processor-Defined Functions:

ame    Type    Name    Type    Name    Type    Name    Type    Name    Type
BS      R*4    DAOUT    R*4    FREQMP  R*4     INT      I*2    XTSMP2  R*4
```

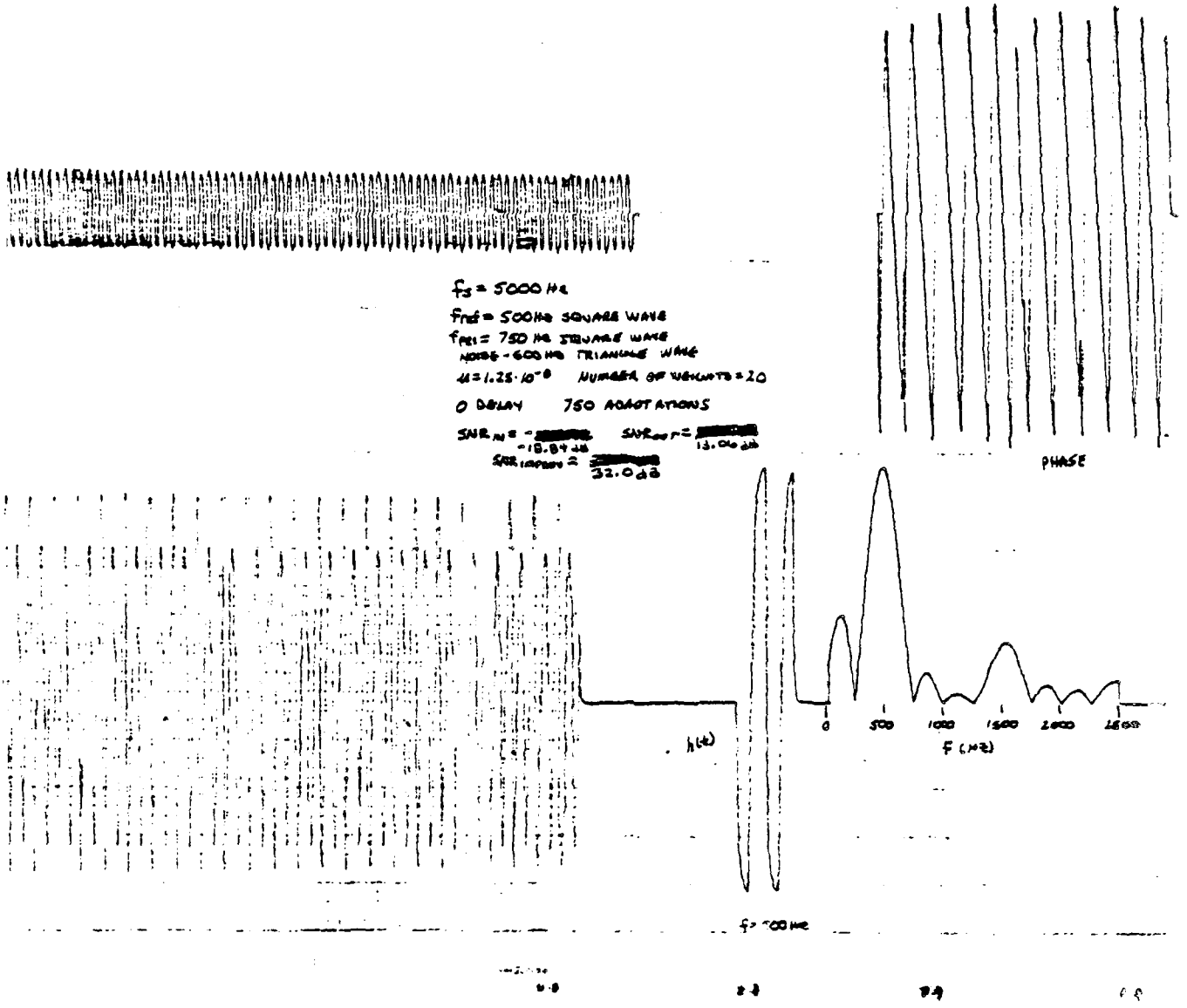
VARIABLE LIST

ADAPT- Adaptive coefficient
H - Product of the filter output and adaptive coefficient
I - Pointer for Do loops
IANS - Variable for responses from the keyboard
IANS1- Variable for responses from the keyboard
J - Pointer for Do loops
K - The present input value with delay accounted for
NC1 - Address pointer for the channel of the noise source
NC2 - Address pointer for the channel of the signal + noise source
NF - Number of frequencies between 0 and fs/2 the frequency spectrum of the weight vector is calculated
NUMD - Number of delays in the filter
NUMS - Number of samples taken
NUMW - Number of weights in the filter
N1 - Integer value of array REAL scaled between 0 and 500
N2 - Integer value of array PHA scaled between 0 and 500
OUT - Filter output
PMAX - Maximum value of the array PHA
PMIN - Minimum value of the array PHA
PSCL - Factor required to scale array PHA to range from 0 to 500
RMAX - Maximum value of the array REAL
RMIN - Minimum value of the array REAL
RSCL - Factor required to scale the array REAL to range from 0 to 500
SUM - Summation variable for the fixed filter
TEMP - Scaled value of the array WEIGHT
WMAX - Maximum value of the array WEIGHT
WSCL - Factor required to scale the array WEIGHT to range from 0 to 500

ARRAY LIST

IN1 - Input array containing noise samples (5000)
IN2 - Input array containing signal + noise samples (5000)
PHA - Phase values of the frequency spectrum of the weight vector (201)
REAL - Magnitude values of the frequency spectrum of the weight array(201)
WEIGHT- Weight vector (100)

APPENDIX I
SAMPLE RUNS



$$W_i(k+1) = W_i(k) + uYS(k-i)$$

where W_i are the N weights, u is the adaptive gain coefficient, Y is the last output of the system, and S are the reference inputs (N_0). The sequence of four instructions that performs this operation are:

```
ZALH *
MPY 68
APAC
SACH *-
ZALH *
MPY 67
APAC
SACH *-
ZALH *
MPY 66
APAC
SACH *-
etc.
```

The ZALH * instruction zeroes the accumulator and loads the contents of the data memory location pointed to by the current auxiliary register (set to point to the weights) into the upper 16 bits of the accumulator. The MPY multiplies the contents of the data memory location (location 68, 67 ... contains the past reference inputs) by the T register (which contains the adaptive gain coefficient and output product uxY). The APAC instruction adds the contents of the P register (result of last multiply) to the accumulator and stores result in the accumulator. The SACH *- instruction stores the upper 16 bits of the accumulator into the data memory location pointed to by the current auxiliary register (still points to the location of the weight) and the auxiliary register is decremented.

The tapped delay line portion of the program calculates:

$$F(k)=\sum_{i=1}^{N} W_i(k)S(k-i)$$

where F is the output of the adaptive filter, N is the number of weights, S are the past N reference inputs, and W are the N weights. This can be implemented with a sequence of two instructions.

```
LTD 67
MPY *-
LTD 66
MPY *-
LTD 65
MPY *-
etc.
```

The LTD instruction places the contents of the data memory location (locations 67, 66, ... contain the past reference inputs) in the T register to set up for the next multiply, add the result of the last multiply which is in the product register to the accumulator, and shifts the data memory to the next data memory location(contents of location 67 would be placed in 68). The MPY *- will multiply the contents of the T register by the contents of the data memory that the current auxiliary register points to (it is set up to point to the data memory containing the weights) and then decrement the auxiliary register to point to the next weight.

The adaptive filter portion of the program uses a series of four instructions. The new weights are calculated by:

Figure 2 shows a flow diagram of the a TMS-320 program that performs the adaptive noise cancellation. The program begins with an initialization portion to set up certain constants. In the main loop, there are several steps performed. First a reference input (N_0) is read. Next the tapped delay line filter is performed. The output of the system is next computed and transferred to the output port. Then the signal plus noise input ($S + N_1$) is read. Finally, the filter weights are adaptively adjusted. This loop is performed once every sampling period. Delay loops are also part of the loop to control the sampling rate. To show the power of the TMS-320 instruction set, the adaptive filter portions of the program are explained.

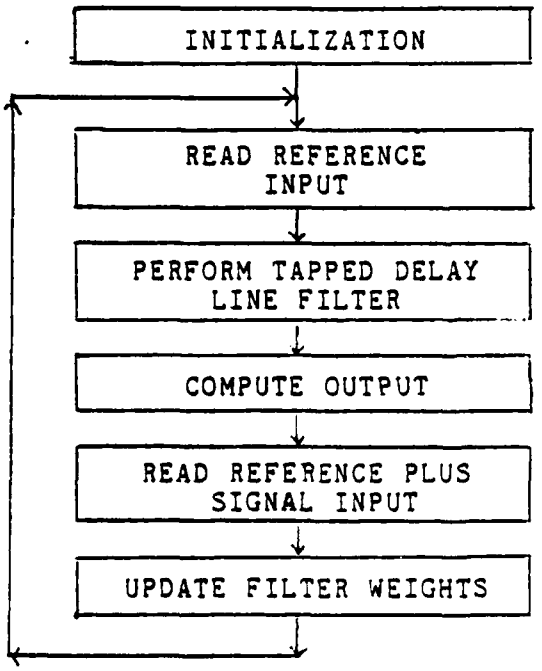


Figure 2. Flow diagram of adaptive noise cancellation program.

Appendix C.

Implementation of an Adaptive Noise Cancellation Algorithm on the TMS-320

An adaptive noise cancellation system has been developed on the Texas Instruments TMS-320. A block diagram of the system is shown in Figure C1. The TMS-320 can implement a 68 weight adaptive filter operating at a sampling frequency of 10.7 kHz.

The TMS-320 has an instruction set tailored to do digital signal processing. With its 200 nsec instruction cycle, it becomes a very powerful processor for applications such as this. The program to do the adaptive noise cancellation only requires six instruction per weight or 1.2 usec plus the overhead needed to do input, output, and minor data manipulations. The maximum number of filter weights that the TMS-320 can support without major hardware modification is 68.

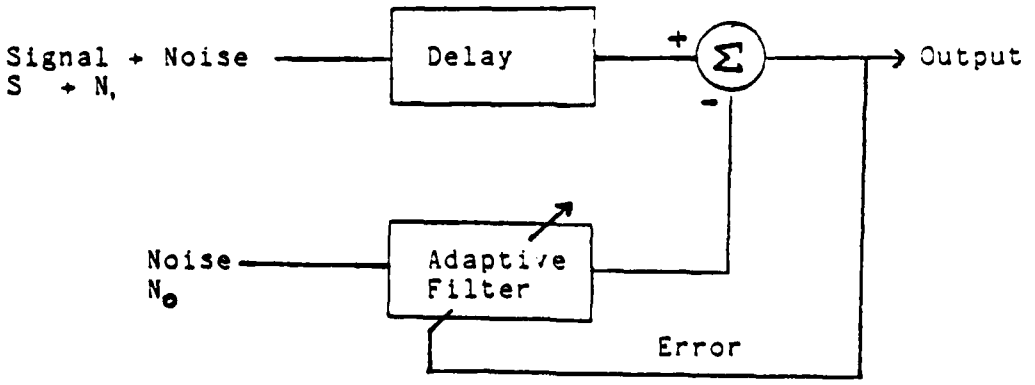


Figure C1. Adaptive Noise Cancellation System.

Enclosure 3 to Appendix D Assembly language sampling subroutine called from rain program. (Also used with programs in Appendixes E and F.)

```
        .TITLE XTSMP2
        .GLOBL XTSMP2
;
;       8. B. PETERSON 17 NOV 83
;       SUBROUTINE CALLED BY "CALL XTSMP2(N,N1,N2,IX1,IX2)". N= # OF
;       SAMPLES IN EACH OF CHANNELS NC1 AND NC2,(N1=256*NC1+16 ETC.)
;       AND IX1 AND IX2 ARE THE RETURNED DATA VECTORS.
;       MAXIMUM TRIGGER RATE IS 21,000 SAMPLES/SEC (10,500 PER CHANNEL)
;
        IOC=170400      ;ADDRESS OF A/D CONTROL REGISTER
        IOI=170402      ;ADDRESS OF A/D OUTPUT REGISTER
XTSMP2: TST (R5)+
        MOV @(R5)+,R0   ;R0 = # OF SAMPLES
        MOV @(R5)+,R1   ;R1 = N1
        MOV @(R5)+,R2   ;R2 = N2
        MOV (R5)+,R3    ;R3 = STARTING ADDRESS OF DATA VECTOR IX1
        MOV (R5)+,R4    ;R4 = STARTING ADDRESS OF IX2
        MOV R1,@#IOC    ;ENABLE EXTERNAL TRIGGER, SET ADD TO CH NC1
LOOP:   BIT #200,@#IOC  ;TEST DONE BIT
        BEQ LOOP
        MOV @#IOI,R5    ;RESET DONE BIT, THROW AWAY DATA
TEST1:  BIT #200,@#IOC  ;TEST DONE BIT
        BEQ TEST1       ;WAIT AND TEST AGAIN IF NOT SET
        MOV R2,@#IOC    ;SET ADDRESS TO CH NC2
        MOV @#IOI,(R3)+ ;READ A/D AND PUT IN MEMORY
LOOP2:  BIT #200,@#IOC  ;TEST DONE BIT
        BEQ LOOP2
        MOV R1,@#IOC    ;SET ADDRESS TO CH NC1
        MOV @#IOI,(R4)+ ;READ CH NC2
        DEC R0
        BNE TEST1       ;CHECK TO SEE IF N SAMPLES HAVE BEEN TAKEN
        RTS PC          ;IF SO, RETURN
        .END
```

Enclosure 2 to Appendix B

```
C       BY S. J. PETERSON 3 NOV 82
C
C       SUBROUTINE FOR SOLVING SIMULTANEOUS LINEAR ALGEBRAIC
C       EQUATIONS USING GAUSS ELIMINATION WITH ROW PIVOTING.
C
C       FROM "CIRCUIT THEORY, A COMPUTATIONAL APPROACH" BY S. W.
C       DIRECTOR, PAGE 211.
C
C       FORM OF CALL IS:
C               CALL GAUSS
C       THE NXN MATRIX A, THE N VECTOR B, THE N VECTOR X AND THE DIMENSION
C       N MUST BE COMMON VARIABLES.
C
        SUBROUTINE GAUSS
        COMMON A(44,44),B(44),N
        DO 5 I=1,N
        I1=I+1
        IF (ABS(A(I,I)).LE.1.E-10) GO TO 1
        GO TO 15
1       CONTINUE
        IF (I.EQ.N) GO TO 10
        DO 14 J=I1,N
        IF (ABS(A(J,I)).LE.1.E-10) GO TO 14
        IPIV=J
        GO TO 16
14      CONTINUE
        GO TO 10
16      DO 2 K=1,N
        PIV=A(IPIV,K)
        A(IPIV,K)=A(I,K)
2       A(I,K)=PIV
        PIV=B(IPIV)
        B(IPIV)=B(I)
        B(I)=PIV
15      IF (I.EQ.N) GO TO 2
        DO 8 J1=I1,N
8       A(I,J1)=A(I,J1)/A(I,I)
        B(I)=B(I)/A(I,I)
        DO 5 J=I1,N
        DO 4 K=I1,N
4       A(J,K)=A(J,K)-A(J,I)*A(I,K)
5       B(J)=B(J)-B(I)*A(J,I)
2       B(N)=B(N)/A(N,N)
        DO 6 K=2,N
        I=N+1-K
        SUM=0.0
        DO 7 J=I,N
7       SUM=SUM+A(I,J)*B(J)
6       B(I)=B(I)-SUM
        GO TO 11
10      WRITE(7,9)
9       FORMAT(' EQUATIONS ARE LINEARLY DEPENDENT')
        STOP
11      RETURN
        END
```

```
                DO 1925 ILOOP= 1 ,NLOOP
1925            CONTINUE
C
C               PLOT OUT DATA
C
                CALL DAOUT (IX1(I+N/2),INT(OUT))
1950            FORMAT (2I7,F10.2)
2000    CONTINUE
        WRITE (7,2100) AVG/(NS-NT+1),AIN/AVG
2100    FORMAT (' AVERAGE OUTPUT SQUARED',F14.2,'  SNR IMPROVEMENT',F8.2)
        WRITE (7,2200)
2200    FORMAT (' NEW DATA, SAME WEIGHTS? 1=YES, 0= NO')
        READ (5,2300) IFLAG
2300    FORMAT (I6)
        IF (IFLAG.EQ.0) GO TO 1050
        CALL XTSMP2(N,NC1,NC2,IX1,IX2)
        GO TO 1600
3000    STOP
        END
```

Enclosure 1 to Appendix B

```
1000    CONTINUE
1050    WRITE (7,1100)
1100    FORMAT (' NUMBER OF WEIGHTS,NUMBER OF DELAY LOOPS')
        READ (5,1200) N,NLOOP
1200    FORMAT(2I6)
        IF (N.GT.70) GO TO 3000
C
C       CALCULATE A & B FOR GAUSS SUBROUTINE
C
        DO 1300 J=1,N
                DO 1225 K=1,N
                        IF (IFLAG.EQ.0) GO TO 1240
                        ID=IABS(J-K)+1
                        A(J,K)=C(ID)
1225                    WRITE (7,1230) J,K,A(J,K)
1235            FORMAT (' A(',I2,',',I2,') = ',E12.5)
1240            B(J)=0.0
                DO 1250 I=1,NS+1-NT
1250                    B(J)=B(J)+FLOAT(IX2(I+J-1))*FLOAT(IX1(I+N/2))
                IF (IFLAG.EQ.0) GO TO 1300
                WRITE (7,1280) J,B(J)
1280            FORMAT (' B(',I2,') = ',E12.5)
1300    CONTINUE
C
C       USE GAUSS ELIMINATION SUBROUTINE TO CALCULATE OPTIMUM WEIGHTS
C
        CALL GAUSS
C
C       PRINT OUT WEIGHTS
C
        DO 1400 J=1,N
                WRITE (7,1500) J,B(J)
1400    CONTINUE
1500    FORMAT (I6,F13.7)
1600    AVG=0.0
C
C       CALCULATE FILTER OUTPUT USING OPTIMUM WEIGHTS
C
        DO 2000 I=1,NS+1-NT
                OUT=FLOAT(IX1(I+N/2))
                DO 1900 J=1,N
                        OUT=OUT-B(J)*IX2(I+J-1)
1900            CONTINUE
C
C               CALCULATE OUTPUT POWER
C
                AVG=AVG+OUT**2
C
C               PRINT OUT INPUT AND OUTPUT DATA
C
                WRITE (7,1950) I,IX1(I+N/2),OUT
C
C               DELAY SO THAT STRIP CHART RECORDER CAN KEEP UP
C
```

Enclosure 1 to Appendix B

```
      PROGRAM OPTFIL
C     B. B. PETERSON 10 NOV 83
C     PROGRAM READS IN TWO CHANNELS OF ANALOG DATA USING SUBROUTINE
C     XTSMP2. THE OPTIMUM WEIGHTS OF FIR FILTERS OF VARIOUS LENGTHS
C     ARE CALCULATED USING A ONE SHOT LINEAR LEAST SQUARES TECHNIQUE.
C     THE AVERAGE POWER IN THE OUTPUT IS THEN CALCULATED FOR EACH LENGTH.
C
C     ARRAYS:
C             IX1 & IX2       INPUT DATA FROM TWO MICROPHONES
C             A & B           MATRIX AND VECTOR IN SOLUTION OF
C                             SIMULTANEOUS EQUATIONS. (B IS BOTH THE
C                             PASSED VECTOR AND THE RETURNED ANSWER
C                             IN THE SOLUTION OF AX=B)
C             C               AUTOCORRELATION VECTOR OF THE REFERENCE
C                             INPUT (IX2) USED TO CALCULATE A.
C
      INTEGER IX1(3000),IX2(3000)
      COMMON A(50,50),B(50),N
      REAL C(50)
C
C     INPUT PARAMETERS
C
50    WRITE (7,100)
100   FORMAT (' ENTER # SAMPLES,S+N CH,NOISE CH,LARGEST FILTER LENGTH')
      READ (5,400) NS,NC1,NC2,NT
400   FORMAT (4I7)
      WRITE (7,500)
500   FORMAT (' TYPE DATA? (1=YES,0=NO)')
      READ (5,600) IFLAG
600.  FORMAT (I4)
      AIN=0.0
      DO 700 J=1,NT
650           C(J)=0.0
700   CONTINUE
C
C     DELAY BEFORE SAMPLING
C
      DO 850 I=1,10000
850   CONTINUE
      N1=256*NC1+16
      N2=256*NC2+16
      CALL XTSMP2(NS,N1,N2,IX1,IX2)
      IF (IFLAG.EQ.0) GO TO 940
900   DO 920 I=1,NS
920           WRITE (7,930) I,IX1(I),IX2(I)
930   FORMAT (3I7)
940   DO 1000 I=1,NS+1-NT
950           DO 990 J=1,NT
                    C(J)=C(J)+FLOAT(IX2(I+J-1))*FLOAT(IX2(I))
990           CONTINUE
C
C             CALCULATE INPUT POWER
C
              AIN=AIN+FLOAT(IX1(I))*FLOAT(IX1(I))
```

However,in the program (enclosure 1) because of the long time required for the large number of calculations, they were set equal which allowed the calculation of only one autocorrelation vector (C). The longer the data set the more valid this approximation is.

After the A matrix and the B vector are calculated, the optimum weight vector is calculated using a Gauss elimination subroutine (enclosure 2). The filter output was then calculated using this weight vector and the output energy compared to the primary input energy.

The results were not impressive. Data from both the Luder yawl engine and the gasoline research were analyzed with minimal reduction in noise power. In view of these results and others obtained since, the explanation of the poor performance of the adaptive early filters, was there limited length and not necessarily slow convergence.

Repeated versions of equation B1 can be written in matrix form:

$$Y = P - RH \tag{B2}$$

where $P(k) = x_1(k-N/2)$ and $R_{ki} = x_2(k-i)$. The total output energy is given by:

$$E = Y^TY = P^TP - 2\,H^TR^TP + H^TR^TRH \tag{B3}$$

This energy is minimized by setting it's gradient with respect to H equal to 0.

$$\nabla_H E = -2\,R^TP + 2\,R^TRH = 0 \tag{B4}$$

or $AH = B$

where $A = R^TR$ and $B = R^TP$.

The elements of A and B are given by:

$$A_{ij} = A_{ji} = \sum_{\text{Data Set}} x_2(k-i)\; x_2(k-j)$$

$$B_j = \sum_{\text{Data Set}} x_2(k+j)\; x_1(k+N/2) \tag{B5}$$

Essentially A is an N x N autocorrelation matrix of the reference input and B is a N vector of the cross-correllation of the reference input and the delayed primary input. Strictly speaking $A_{11} \neq A_{22}$ and $A_{12} \neq A_{23}$ etc. because A_{11} contains one early sample not in A_{22} and A_{22} contains a late sample not in A_{11}.

Appendix B

FORTRAN Program to Calculate Optimum Filter Weights

In the LMS adaptive noise cancellation scheme the weights of the adaptive filter are adjusted to minimize the output power. In previous FORTRAN implementations of the LMS algorithm, the filter was found to ineffective in canceling engine noise. It was felt this may be due to the weights not converging within the space of the data set. Using off-line parameter estimation techniques it is possible to calculate the optimum fixed weight filter for a given data set, and then analyze this filter using the data set. The filter is optimum in the sense that the remaining output power for the data set is the minimum of all possible weight vectors. The filter is illustrated in Figure B1.

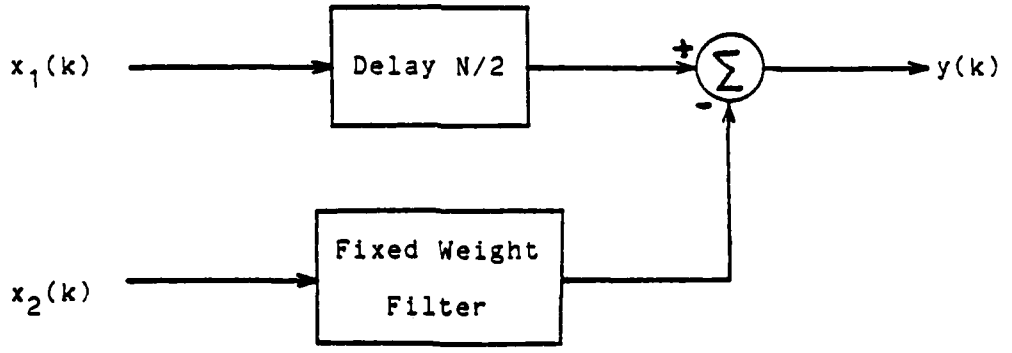
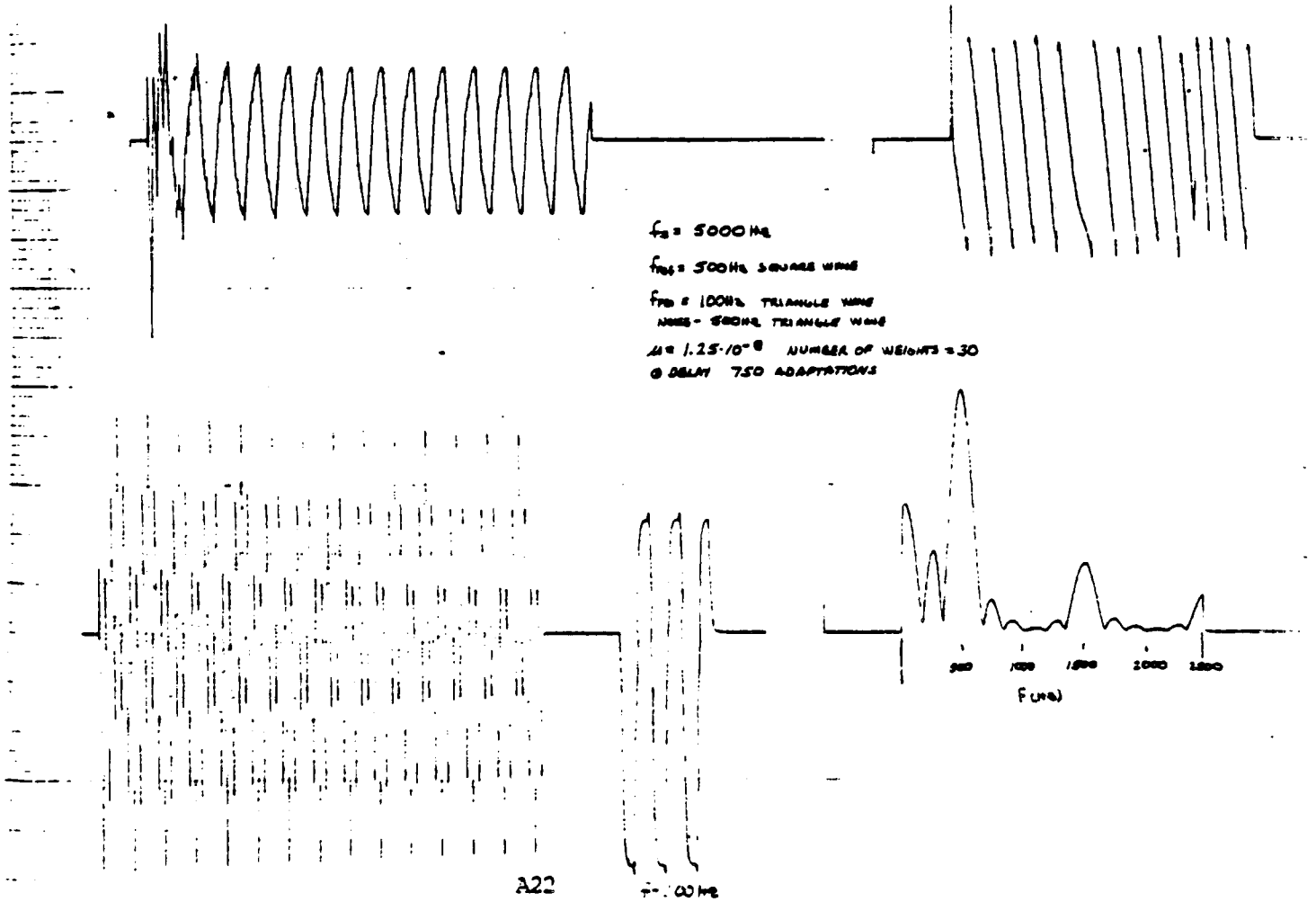
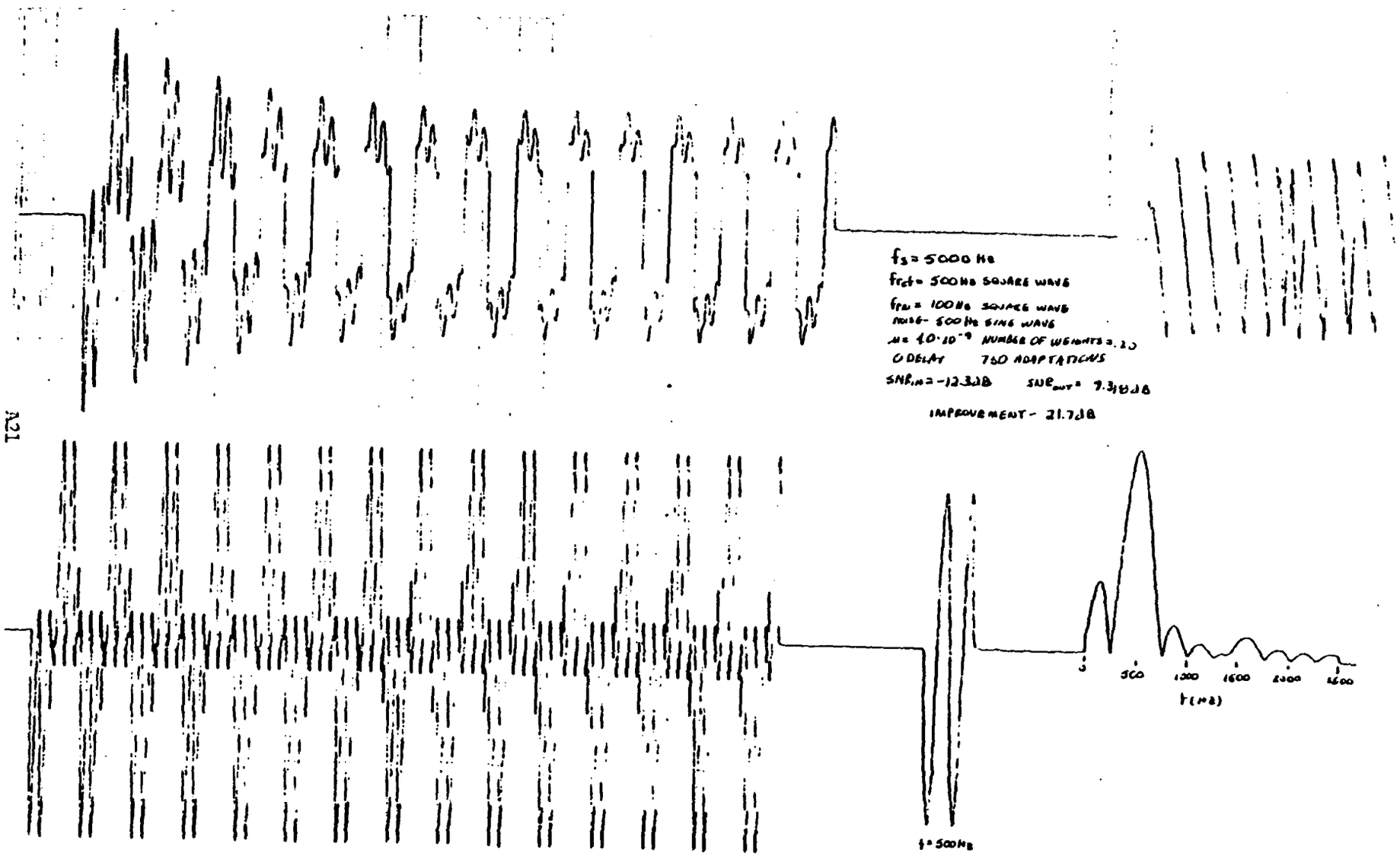


Figure B1

For a fixed weight vector H, the filter output, y(k) is given by:

$$y(k) = x_1(k-N/2) - \sum_{i=0}^{N-1} h_i \, x_2(k-i) \qquad (B1)$$


f_{s1} = 5000 Hz
f_{sig} = 500Hz SQUARE WAVE
f_{INT} = 100Hz TRIANGLE WAVE
NOISE - 500Hz TRIANGLE WAVE
μ = 1.25·10⁻⁸ NUMBER OF WEIGHTS = 30
@ DELAY 750 ADAPTATIONS
500
1000
1500
2000
2500
F (Hz)


f_s = 5000 Hz
f_{ref} = 500 Hz SQUARE WAVE
f_{sig} = 100 Hz SQUARE WAVE
NOISE - 500 Hz SINE WAVE
μ = $4.0 \cdot 10^{-9}$ NUMBER OF WEIGHTS = 32
0 DELAY 750 ADAPTATIONS
SNR_{in} = -12.3 dB SNR_{out} = 9.4 dB
IMPROVEMENT - 21.7 dB
121
f = 500 Hz
0
500
1000
1500
2000
2500
f (Hz)

The monitor on the TMS-320 development system allows assembly language programs to be down-loaded from another computer. It then assembles the program into its machine code for execution. To analyze the effects of various numbers of weights, sampling frequencies, and adaptive gain coefficients would require many modifications to the assembly language program. To make these modifications easy, a BASIC 7 program was developed on the Dartmouth Time Sharing system to generate TMS-320 assembly code. The program is interactive in that it asks for the number of weights, sampling frequency, and adaptive gain coefficient. It produces straight line code so the time spent on the tapped delay line filtering and weight adaptation is minimized. There are delay loops added to the code to provide a range of sampling frequencies. Enclosure C1 is a listing of the Basic program. Enclosure C2 is a listing of the assembly language code produced by this program for a 10 weight filter.

This system has been implemented to cancel engine background noise in voice communications. Table C1 summarizes the amount of noise reduction for various numbers of weights, sampling frequencies, and adaptive gain coefficients. The amount of noise reduction is calculated by comparing the RMS voltage of the signal plus noise input to the RMS voltage of the output. Anti-aliasing filters set at half the sampling frequency were used on both input signals.

From the data shown in Table C1, it is clear that the amount of noise reduction is improved by increasing the number of filter weights. At the higher sampling rates, there in very little noise reduction. At the lower sampling frequencies the noise reduction is even more apparent (8 dB at 2 kHz.). This indicates that to achieve the same amount of noise reduction at a sampling frequency of 10 kHz would require five times more weights or on the order of 300 weights. Also there is an apparent increase of signal reduction when the adaptive gain coefficient is increased. This is not only an increase in noise reduction; but, because the filter is adapting so quickly to the signal, it is also canceling some of the desired signal and not just the noise.

The noise cancellation system only performed marginally in the engine noise application at useful sampling frequencies. To improve the performance would require a processor capable of implementing more weights. The next generation of TMS-320 may have this capability. This work has shown that adaptive filtering with a maximum of 68 weights is realizable for speech processing and may have other applications.

Sampling Frequency KHz	Number of Weights	Adaptive Gain	Signal Reduction dB
10	68	14	3.5
10	68	12	2.6
10	68	10	2.4
10	68	8	1.4
10	68	6	*
10	50	14	3.2
10	50	12	2.3
10	50	10	2.3
10	50	8	1.5
10	50	6	*
10	30	14	2.5
10	30	12	1.9
10	30	10	1.9
10	30	8	1.5
10	30	6	*
5	68	14	2.4
5	68	12	2.4
5	68	10	2.1
5	68	8	-.8
5	68	6	*
5	50	14	2.1
5	50	12	2.2
5	50	10	2.2
5	50	8	0.0
5	50	6	*
5	30	14	1.9
5	30	12	1.9
5	30	10	1.9
5	30	8	0.9
5	30	6	*
2	68	14	8.4
2	68	12	7.9
2	68	10	6.0
2	68	8	0.6
2	68	6	*
2	50	14	6.4
2	50	12	5.6
2	50	10	4.9
2	50	8	0.4
2	50	6	-.4
2	30	14	2.4
2	30	12	2.4
2	30	10	2.1
2	30	8	-.8
2	30	6	*

(* - Did not track)

Table C1. Test results of Noise Cancellation System.

Enclosure 1 to Appendix C

```
100 !
110 !    PROGRAM NAME: ADAPT-F (ADAPTIVE FILTER)
120 !                  VERSION 4
130 !
140 !    PROGRAMMER: K. U. DYKSTRA
150 !
160 !    DATE: 20 SEPT 84
170 !
180 !    THIS PROGRAM PRODUCES THE ASSEMBLY LANGUAGE ADAPTIVE
190 !    FILTER PROGRAM FOR THE TMS-320 EVALUATION MODULE. IT
200 !    CAN BE USED IN CONJUNCTION WITH THIS MODULE TO DOWN
210 !    LOAD THE ASSEMBLY LANGUAGE PROGRAM. A TAPPED DELAY
220 !    LINE ALGORITM IS USED FOR THE ACTUAL FILTERING. THE
230 !    WEIGHTS ARE ADJUSTED USING A LMS ALGORITHM. INPUTS
240 !    ARE NUMBER OF WEIGHTS, SAMPLING FREQ, ADAPTIVE
         COEFFICIENT.
250 !    THE CODE THAT IS PRODUCED IS STRAIGHT LINE CODE WITH
260 !    NO LOOPING FOR MAXIMUM SPEED.
270 !
280 !    DATA MEMORY ASSIGNMENT
290 !
300 !    0-68  REFERENCE INPUTS
310 !    69    CHANNEL 1 CONTROL
320 !    70    CHANNEL 2 CONTROL
330 !    71    MASK FOR INPUTS
340 !    72    MASK FOR OUTPUTS
350 !    73    FILTER OUTPUT
360 !    74    SIGNAL + REFERENCE INPUT
370 !    75    OUTPUT
380 !    76-143 FILTER WEIGHTS
390 !
400 !    INPUT NUMBER OF WEIGHTS.
410 !
412 DO
414 LET ERROR = 0
420 PRINT "ENTER NUMBER OF WEIGHTS (1-68)."
430 INPUT NWTS
432 IF NWTS < 1 OR NWTS > 68 THEN LET ERROR = 1
434 LOOP UNTIL ERROR = 0
440 !
450 !    INPUT SAMPLING FREQUENCY
460 !
470 LET LOWF=1/(.0002*(37+(6*NWTS)+(20*256)))
472 LET HIGHF=1/(.0002*(37+(6*NWTS)+20))
474 IF HIGHF > 15.6 THEN LET HIGHF = 15.6
476 DO
478 LET ERROR = 0
480 PRINT "ENTER SAMPLING FREQUENCY (";LOWF; " TO ";HIGHF;"
    KHZ)."
490 INPUT FREQ
495 IF FREQ < LOWF OR FREQ > HIGHF THEN LET ERROR = 1
496 LOOP UNTIL ERROR = 0
```

```
500 LET DELAY=INT(((1/(FREQ*.0002))-33-(6*NWTS))/20)
510 LET AFREQ=1/(.0002*(37+(6*NWTS)+(20*DELAY)))
520 PRINT "SAMPLING FREQUENCY = ";AFREQ;" DELAY = ";DELAY
530 !
532 DO
534 LET ERROR = 0
540 PRINT "ADAPTIVE COEFFICIENT = 1/(2**(16-N)), ENTER N"
550 PRINT "IF YOU WANT TO DOWN LOAD END WITH <CTRL C> ELSE
    <RETURN>"
560 INPUT COEF
562 IF COEF < 0 OR COEF > 15 THEN LET ERROR = 1
564 LOOP UNTIL ERROR = 0
570 !
580 !     SET UP DATA CONSTANTS
590 !
600 PRINT ">"
610 PRINT "       AORG 0"
620 PRINT "       B BEGIN"
630 PRINT "       NOP"
640 PRINT "       NOP"
650 PRINT "       DATA >7FF0"
660 PRINT "       DATA >8000"
670 PRINT "BEGIN LACK 4"
680 PRINT "       TBLR 71"
690 PRINT "       LACK 5"
700 PRINT "       TBLR 72"
710 PRINT "       LACK 3"
720 PRINT "       SACL 69"
730 PRINT "       LACK >83"
740 PRINT "       SACL 70"
750 PRINT "       SOVM"
760 PRINT "       LARP 1"
770 !
780 !     INPUT REFERENCE INPUT
790 !
800 PRINT "LOOP   OUT 69,0"
810 PRINT "       NOP"
820 PRINT "       NOP"
830 PRINT "       IN 0,2"
840 PRINT "       ZALS 71"
850 PRINT "       XOR 0"
860 PRINT "       SACL 0"
870 !
880 !     DELAY FOR SAMPLING FREQUENCY
890 !
900 PRINT "       LARP 0"
910 PRINT "       LARK 0,";DELAY
920 PRINT "D1     NOP"
921 FOR I = 1 TO 7
922 PRINT "       NOP"
923 NEXT I
930 PRINT "       BANZ D1"
940 !
950 !     PERFORM TAPPED DELAY LINE FILTER
```

```
960 !
970 PRINT "       ZAC"
975 PRINT "       MPYK 0"
980 PRINT "       LARP 1"
990 PRINT "       LARK 1,143"
1000 FOR I = NWTS - 1 TO 0 STEP -1
1010     PRINT "       LTD ";I
1020     PRINT "       MPY *-"
1030 NEXT I
1040 PRINT "       APAC"
1050 PRINT "       SACH 73"
1060 !
1070 !    OUTPUT = SIGNAL&REF - FILTERED OUTPUT
1080 !
1090 PRINT "       LAC 74"
1100 PRINT "       SUB 73"
1110 PRINT "       SACL 75"
1120 !
1130 !    COEF*OUTPUT > T
1140 !
1150 PRINT "       LAC 75,";COEF
1160 PRINT "       SACH 73"
1170 PRINT "       LT 73"
1180 !
1190 !    OUTPUT
1200 !
1210 PRINT "       ZALS 72"
1220 PRINT "       XOR 75"
1230 PRINT "       SACL 75"
1240 PRINT "       OUT 75,2"
1250 !
1260 !    INPUT REFERENCE&SIGNAL
1270 !
1280 PRINT "       OUT 70,0"
1290 PRINT "       NOP"
1300 PRINT "       NOP"
1310 PRINT "       IN 74,2"
1320 PRINT "       ZALS 71"
1330 PRINT "       XOR 74"
1340 PRINT "       SACL 74"
1350 !
1360 !    DELAY FOR SAMPLING FREQUENCY
1370 !
1380 PRINT "       LARP 0"
1390 PRINT "       LARK 0,";DELAY
1400 PRINT "D2     NOP"
1401 FOR I = 1 TO 7
1402 PRINT "       NOP"
1403 NEXT I
1410 PRINT "       BANZ D2"
1420 PRINT "       LARP 1"
1430 PRINT "       LARK 1,143"
1440 !
1450 !    PERFORM ADAPTIVE PART
```

```
1460 !
1470 FOR I = NWTS-1 TO 0 STEP -1
1480     PRINT "       ZALH *"
1490     PRINT "       MPY ";I+1
1500     PRINT "       APAC"
1510     PRINT "       SACH *-"
1520 NEXT I
1530 PRINT "       B LOOP"
1540 PRINT "       END"
1550 PRINT "<"
1560 END
```

Enclosure 2 to Appendix C

```
ENTER NUMBER OF WEIGHTS (1-68).
?  10
ENTER SAMPLING FREQUENCY ( .958405  TO  15.6  KHZ).
?  10.0
SAMPLING FREQUENCY =  10.0604  DELAY =  20
ADAPTIVE COEFFICIENT = 1/(2**(16-N)), ENTER N
IF YOU WANT TO DOWN LOAD END WITH <CTRL C> ELSE <RETURN>
? 12
>
       AORG 0
       B BEGIN
       NOP
       NOP
       DATA >7FF0
       DATA >8000
BEGIN, LACK 4
       TBLR 71
       LACK 5
       TBLR 72
       LACK 3
       SACL 69
       LACK >83
       SACL 70
       SOVM
       LARP 1
LOOP   OUT 69,0
       NOP
       NOP
       IN 0,2
       ZALS 71
       XOR 0
       SACL 0
       LARP 0
       LARK 0, 20
D1     NOP
       NOP
       NOP
       NOP
       NOP
       NOP
       NOP
       NOP
       BANZ D1
       ZAC
       MPYK 0
       LARP 1
       LARK 1,143
       LTD  9
       MPY *-
       LTD  8
       MPY *-
       LTD  7
```

```
        MPY *-
        LTD  6
        MPY *-
        LTD  5
        MPY *-
        LTD  4
        MPY *-
        LTD  3
        MPY *-
        LTD  2
        MPY *-
        LTD  1
        MPY *-
        LTD  0
        MPY *-
        APAC
        SACH 73
        LAC 74
        SUB 73
        SACL 75
        LAC 75, 12
        SACH 73
        LT 73
        ZALS 72
        XOR 75
        SACL 75
        OUT 75,2
        OUT 70,0
        NOP
        NOP
        IN 74,2
        ZALS 71
        XOR 74
        SACL 74
        LARP 0
        LARK 0, 20
D2      NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        BANZ D2
        LARP 1
        LARK 1,143
        ZALH *
        MPY  10
        APAC
        SACH *-
        ZALH *
        MPY  9
        APAC
```

```
        SACH  *-
        ZALH  *
        MPY   8
        APAC
        SACH  *-
        ZALH  *
        MPY   7
        APAC
        SACH  *-
        ZALH  *
        MPY   6
        APAC
        SACH  *-
        ZALH  *
        MPY   5
        APAC
        SACH  *-
        ZALH  *
        MPY   4
        APAC
        SACH  *-
        ZALH  *
        MPY   3
        APAC
        SACH  *-
        ZALH  *
        MPY   2
        APAC
        SACH  *-
        ZALH  *
        MPY   1
        APAC
        SACH  *-
        B LOOP
        END
```

<

Appendix D.

Notch Filter Analysis and Results

In [1] and [2] it was pointed out that when the reference input ($x_2(k)$ in Figure D1) is periodic in the length N, of the adaptive filter, the result is a notch filter from $x_1(k)$ to $y(k)$.

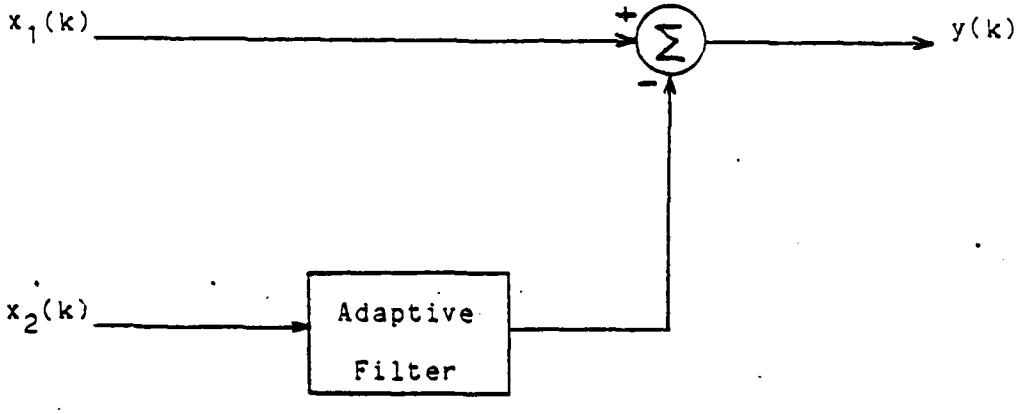


Figure D1

Because the background noise due to an engine is quite periodic, it follows that any reference that contains all the harmonics of the engine noise should suffice. A periodic reference input consisting of one "1" followed by N-1 "0"'s i.e.

$$x_2(k) = 1 \quad \text{for } k = mN \qquad (D1)$$
$$= 0 \quad \text{for } k = mN$$

where N is both the number of samples per period of the engine noise and the number of filter weights and m is an integer not

only contains all these harmonics but also makes possible implementing a filter of large N in real time.

The filter and adaptive algorithm equations now become:

$$y(mN+i) = x_1(mN+i) - \sum_{j=0}^{N-1} h_j(mN+i)\, x_2(mN+i-j) \qquad (D2)$$

$$= x_1(mN+i) - h_i(mN+i)$$

and

$$h_i((m+1)N+i) = h_i(mN+i) + a\, y(k) \qquad (D3)$$

Because the algorithm requires only one subtraction, one addition and one multiplication per sample period, it can be implemented very efficiently with the order, N, limited only by memory. Further, if $a = 2^{-n}$ the multiply can be accomplished with right shifts allowing implementation on a general purpose microprocessor.

Equations (D2) and (D3) can be written using z transforms:

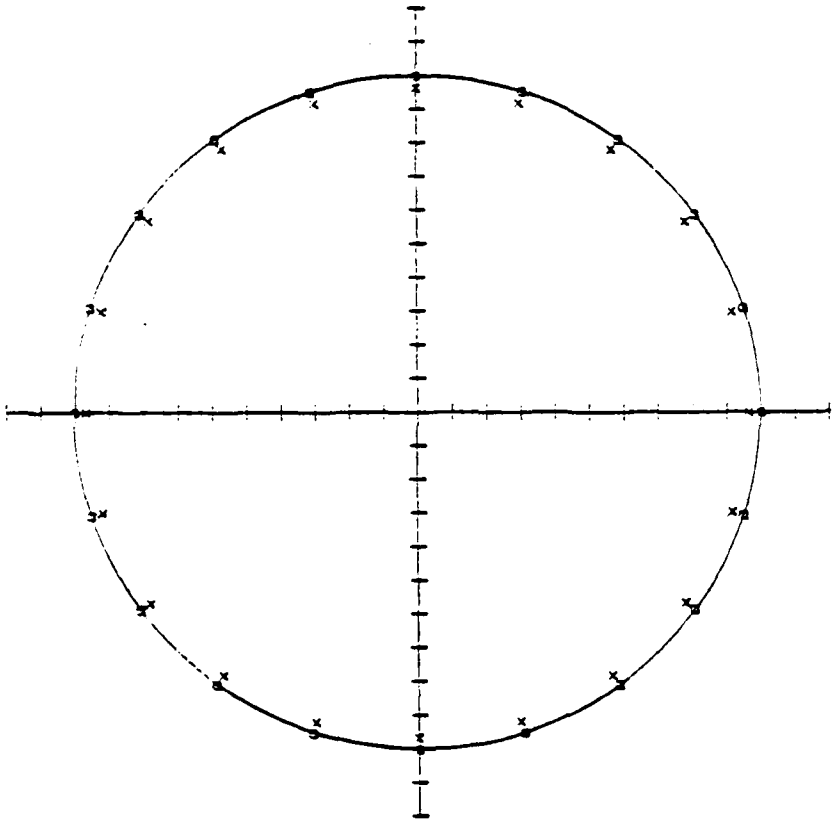
$$Y(z) = X_1(z) - H_i(z) \qquad (D4)$$

$$z^N H_i(z) = H_i(z) - a\, Y(z) \qquad (D5)$$

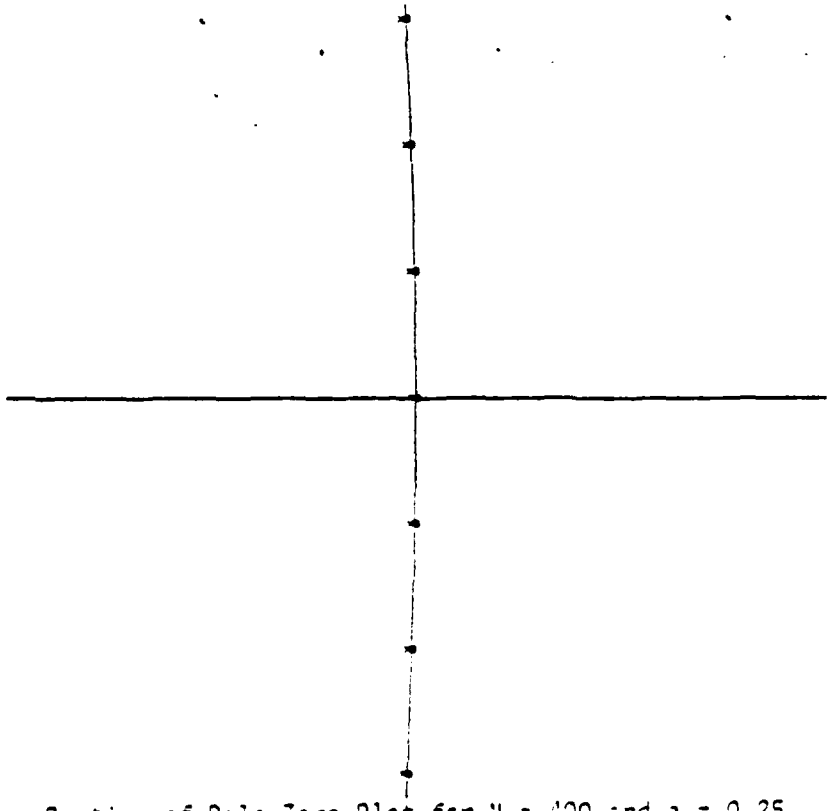
Combining (D4) and (D5) results in the transfer function from $x_1(k)$ to $y(k)$:

$$\frac{Y(z)}{X_1(z)} = \frac{z^N - 1}{z^N - (1-a)} \qquad (D6)$$

The N zeros of (D6) are on the unit circle at $e^{j2\pi k/N}$, k=0 to N-1 and each zero has a corresponding pole at $(1-a)1/N\ e^{j2\pi k/N}$. A pole-zero plot for a = 0.5 and N = 20 is shown in Figure D2. The actual filter implemented had a = 0.25 and N = 400. Each pole



Pole Zero Plot for N = 20 and a = 0.5
Figure D2



Portion of Pole Zero Plot for N = 400 and a = 0.25
Figure D3

was therefore at a radius of 0.999281. Figure D3 is a small section of the pole-zero plot for this case illustrating the seven poles and zeros nearest z=1. The net effect is a notch filter with a notch at each harmonic of $x_2(k)$. The width (distance between -3 db points) of each notch is approximately:

$$\frac{a\ f_s}{\pi N} = \frac{a\ f_o}{\pi}$$

where f_s and f_o are the sampling and fundamental frequencies respectively. For N = 400, a= 0.25 and f_s = 7 kHz, the notches are 1.4 Hz wide and adjacent notches are separated by 17.5 Hz. This frequency response is impossible to verify using our analog spectrum analyzers. Figure D4 shows the output spectrum for N = 160, a = 0.25 and f_s = 40 kHz. The input was a sinusoid swept from 20 Hz to 10 kHz. The notches are 19.9 Hz wide and spaced at 250 Hz intervals. The slight rolloff at high frequencies is due to the hold function of the D/A converter.

If the output of the filter is taken at the output of the adaptive filter before the summer, the resulting filter is a recursive comb filter passing only those components of $x_1(k)$ that are periodic in N samples. Similar non-recursive comb filters that exploited the periodicity of the speech waveform to eliminate white background noise were proposed in [3] and [4]. These filters were adaptive in the sense that the period of the filter was continuously adapted to match the period of the speech waveform. This type of filter is also the digital version of the analog waveform eductors of many years ago.

If $x_2(k)$ is an arbitrary periodic signal with a period of N

Enclosure 1 to Appendix E

```
      PROGRAM ADFDLS

      B. B. PETERSON
      PROGRAM TO IMPLEMENT ADAPTIVE FILTER IN FREQUENCY DOMAIN
      USING WEIGHTED LEAST SQUARES ADAPTIVE ALGORITHM
      REF: M. DENTINO, J. MCCOOL, & B. WIDROW, "ADAPTIVE FILTERING
      IN THE FREQUENCY DOMAIN", PROC IEEE, VOL.66,NO. 12. DEC 78.
      MAXIMUM SIZE OF EACH BLOCK OF DATA IS 512

      DIMENSION X1(257),Y1(257),X2(257),Y2(257),XO(257),YO(257)
      DIMENSION WR(512),WI(512),XH(256),YH(256),P2(256),CR(256)
      DIMENSION CI(256)
      INTEGER I1(5120),I2(5120)

      VARIABLES

      X1 & Y1 PRIMARY INPUT (BOTH TIME AND FREQUENCY DOMAIN DATA)
      X2 & Y2 REFERENCE INPUT
      XO & YO OUTPUT
      XH & YH WEIGHTS OF ADAPTIVE FILTER (FREQUENCY DOMAIN)
      WR & WI TABLES OF COSINES AND SINES
      I1 TIME DOMAIN PRIMARY INPUT
      I2 TIME DOMAIN REFERENCE INPUT

      CALCULATE VECTORS OF SINES AND COSINES

      NMAX=512
      TPON=6.28318/FLOAT(NMAX)
      DO 3 I=1,NMAX
              PH=FLOAT((I-1))*TPON
              WR(I)=COS(PH)
              WI(I)=-SIN(PH)
      CONTINUE

      INPUT PARAMETERS

      WRITE (7,10)
      FORMAT (' NEW DAT(1=Y),ZER WT,# SAMP,BLK SZ,CH #S,ADP GAIN')
      READ (5,20) IF1,IF2,N,MW,NC1,NC2,G
      FORMAT (6I6,F10.8)
      NW=2**MW
      OMG=1.0-G
      IF (IF1.NE.1) GO TO 100
      N1 = 256*NC1 + 16
      N2 = 256*NC2 + 16

      INSERT DELAY BEFORE SAMPLING

      DO 30 J=1,20000
      CONTINUE
      CALL XTSMP2 (N,N1,N2,I1,I2)
      NW2 = NW/2
      IF (IF2.EQ.0) GO TO 50
```

results are subtracted from $X_1(f)$, giving the FFT of the output, which is then inverse transformed to realize the time domain output. The data is processed in blocks of N (=2n) points.

Since each complex element of Y(f) is a function only of the corresponding coefficient of H(f) and not the entire vector the adaptive algorithm can be made to have much more efficient and predictable convergence properties. Therefore, unlike the time domain problem, when doing off-line processing on the LSI-11/2 with long filter lengths but limited data storage, in the frequency domain filter one can be assured the weights will converge within the data set. Also, because of the efficiency of the algorithm, the data can be processed in reasonable time.

In the FORTRAN implementation (enclosure 1) the complex LMS algorithm proposed in [6] has been changed to a weighted least squares algorithm for more efficient and predictable convergence properties. Each element of H(f) is the best least squares estimate with the input data exponentially weighted. The filter can therefore track time varying parameters with the time constant of the exponential weighting.

Appendix E.

Frequency Domain Adaptive Filtering

A fundamental problem in both real time implementation and in off-line analysis of adaptive noise cancellation is that the filter length is severely limited by memory, processor speed, or both. At the expense of much more complicated software, digital filters, including adaptive filters, can be implemented more efficiently in the frequency domain than the time domain. The basic structure of a frequency domain adaptive noise canceler, proposed in [5] is shown in Figure E1.

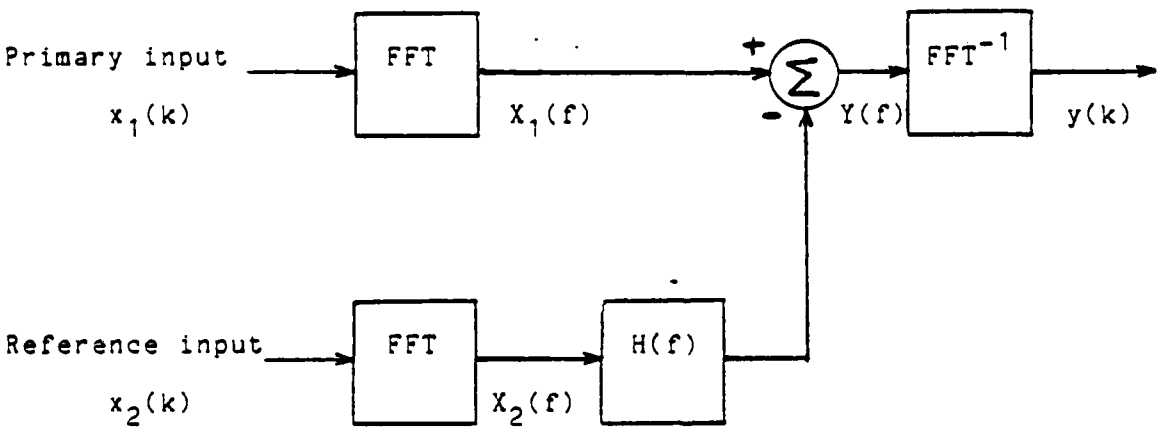


Figure E1

The Fast Fourier Transforms (FFT's) of the two signal inputs are calculated, the complex coefficients of the reference input FFT are multiplied by complex filter coefficients, H(f). The

Enclosure 3 to Appendix D

```
        .TITLE REALTI
        .GLOBL REALTI

        B. B. PETERSON, 9 MAY 84
        IMPLEMENTS NOTCH FILTER IN REAL TIME.
        CALLED FROM FORTRAN BY "CALL REALTI (NC,NW)"
        NC = 256* CH. NO. + 16
        NW = NO. WEIGHTS IN FILTER*2


        IOA = 170404            ;ADD OF CH. A D/A
        IOB = 170406            ;ADD OF CH. B D/A
        IOC = 170400            ;ADD OF CONTROL & STATUS REG.
        IOI = 170402            ;ADD OF A/D OUTPUT
EALTI:  TST (R5)+
        MOV @(R5)+,@#IOC        ;SETS CH NO AND EXT TRIG
        MOV @(R5)+,R0           ;R0 = NO OF WEIG-TS
        MOV R0,R2
OOP0:   MOV #0,DATA(R2)         ;SET WEIGHT VECTOR TO ZERO
        SOB R2,LOOP0
OOP1:   MOV R0,R2               ;INIT COUNTER
OOP2:   BIT #200,@#IOC          ;TEST DONE BIT
        BEQ LOOP2
        MOV @#IOI,R3            ;READ A/D
        MOV R2,@#IOA            ;OUTPUT RAMP TO CH. A D/A TO CHECK SYNCH
        SUB DATA(R2),R3         ;R3=R3-WEIGHT(R2)
        MOV R3,@#IOB            ;FILTER OUTPUT TO CH. B D/A
        ASR R3                  ;SHIFT R3 RIGHT 2 TIMES
        ASR R3
        ADD R3,DATA(R2)         ;ADJUST WEIGHT
        DEC R2
        SOB R2,LOOP2            ;DO THIS FOR NUMBER OF WEIG-TS
        BR LOOP1                ;GO BACK AND START AT BEGINNING OF VECTOR
OOP4:   RTS PC
ATA:    .BLKW 1000              ;RESERVE ROOM FOR WEIGHTS
        .END
```

Enclosure 2 to Appendix D

```
  STORE PRESENT POINTER IN DATA MEM 5
LP    SACL  5
  READ IN OLD WEIGHT
      TBLR  4
  READ A/D. PUT IN DATA MEM 6
      IN    6,2
  THEN INTO ACC
      LAC   6,0
  CONVERT FROM OFFSET BINARY TO 2'S COMPLEMENT
      XOR   7
  SUBTRACT WEIGHT
      SUB   4,0
  STORE RESULT IN DATA MEM 6
      SACL  6
  CONVERT BACK TO OFFSET BINARY
      XOR   7
      SACL  8
  OUTPUT TO D/A
      OUT   8,2
  LOAD HIGH ACC WITH .25 OF OUTPUT
      LAC   6,14
  NEW WEIGHT = OLD WEIGHT + .25 OUTPUT
      ADDH  4
      SACH  4
  LOAD ACC WITH LOCATION OF WEIGHT IN EXT MEM
      LAC   5,0
  STORE NEW WEIGHT IN EXT MEM
      TBLW  4
  INCREMENT ACC TO POINT TO NEXT WEIGHT
      ADDS  1
  PUT AUX REG 0 OUT TO PORT 3 TO CHECK SYNCHRONIZATION
      SAR   0,9
      OUT   9,3
  CHECK IF AT END OF WEIGHT VECTOR
      BANZ  SINT
  IF SO. START AT BEGINNING
      LAR   0,3
      LAC   2,0
INT   LARP  1
  INSERT DELAY TO PREVENT DOUBLE TRIGGER
      LARK  1,150
EL    NOP
      BANZ  DEL
      LARP  0
  ENABLE INTERRUPT
      EINT
  GO BACK AND WAIT FOR ANOTHER
       RET
      END
```

Enclosure 2 to Appendix D

```
*
*  NOTCH FILTER PROGRAM
*  B. B. PETERSON, 19 JUN 84
*  TRIGGER BY EXTERNAL INTERRUPT
*  # OF WEIGHTS-1 MUST BE PUT IN DATA MEM 3
       B    INIT
       NOP
*  INTERRUPT SERVICE
       B    ILP
*  INITIALIZE
INIT   LACK  0
*  PUT 0 IN DATA MEM 0 TO USE ZEROING TABLE BELOW
       SACL  0
*  SET UP A/D AND D/A CONTROL REGISTER
       LACK   7
       SACL    10
       OUT    10,0
*  PUT 1000000000000000 IN DATA MEM 7 FOR OFFSET BINARY TO 2'S COMPLEMENT CON
       LACK   1
       SACL   1
       LAC    1,15
       SACL   7
*  PUT 2 IN DATA MEM 1, INCREMENT IN TABLE
       LACK   2
       SACL   1
*  PUT LOCATION OF START OF TABLE IN DATA MEM 2
       LACK  251
       SACL  2
*  SET OVERFLOW MODE
       SOVM
       LARP  0
*  ZERO TABLE
*  PUT STARTING ADD OF TABLE IN ACCUMULATOR
       LAC   2,0
*  PUT # OF WEIGHTS-1 IN AUX REG 0
       LAR   0,3
*  PUT 0 IN THE EXT MEM POINTED TO BY THE ACC.
ZTB    TBLW  0
*  INCREMENT THE ACC BY 2
       ADDS  1
*  DEC AUX REG 0 AND LOOP AGAIN IF NOT ZERO
       BANZ  ZTB
*  MAIN LOOP TO START AT BEGINNING OF WEIGHT VECTOR
MLP    LAR   0,3
       LAC   2,0
*  ENABLE INTERRUPT
       EINT
*  WAIT FOR INTERRUPT
WAIT   NOP
       B     WAIT
```

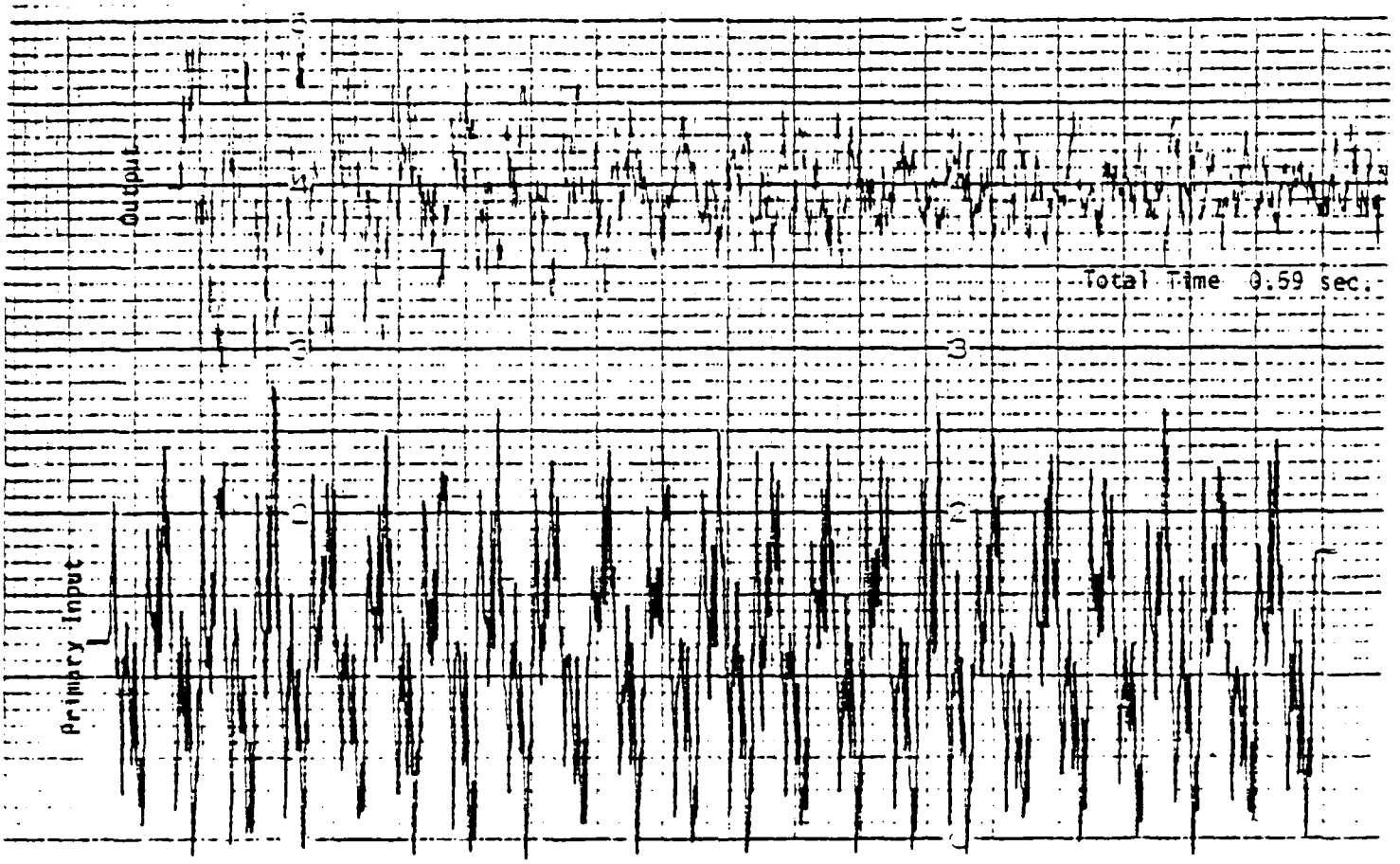
```
      DO 1085 J = 1,NW
              J1=J+NW
              J2=J+2*NW
              J3=J+3*NW
              J4=J+4*NW
              J5=J+5*NW
              J6=J+6*NW
              IF (IFLAG.EQ.1) GO TO 1080
      WRITE (6,1060) J,IX(J),IX(J1),IX(J2),IX(J3),IX(J4),IX(J5),IX(J6)
              GO TO 1085
1080  WRITE (7,1060) J,IX(J),IX(J1),IX(J2),IX(J3),IX(J4),IX(J5),IX(J6)
1085  CONTINUE
1060  FORMAT (8I7)
C
C     CALCULATE AND PRINT OUT SIGNAL TO NOISE RATIO
C
1090  SNR=SQIN/SOUT
      WRITE(7,1100) SQIN,SOUT,SNR
1100  FORMAT (3F10.2)
      WRITE (7,1200)
1200  FORMAT (' NEW DATA? 1=YES,0=NO')
      READ (5,140) IFLAG2
      GO TO 50
1300  STOP
      END
```

Assembly Language Sampling subroutine called from main program

```
      .GLOBL XTSAMP
;
;     B. B. PETERSON, 17 NOV 83
;     SUBROUTINE TO TAKE A GIVEN NUMBER OF SAMPLES FROM THE DTC785
;     ANALOG I/O BOARD USING EXTERNAL TRIGGERING AT A MAXIMUM RATE OF
;     25,000 SAMP/S. THE SUBROUTINE IS CALLED BY "CALL XTSAMP(N,NC,IX)"
;     WHERE N= # OF SAMPLES,
;     NC = 256*CHANNEL # +16, AND IX IS THE DATA VECTOR.
;
      IOC=170400              ;LOCATION OF CONTROL REGISTER
      IOI=170402              ;LOCATION OF A/D OUTPUT REGISTER
XTSAMP: TST (R5)+
      MOV @(R5)+,R0           ;R0=# OF SAMPLES
      MOV @(R5)+,@#IOC        ;SET CH# AND EXTERNAL TRIGGER ENABLE
      MOV (R5)+,R3            ;R3=STARTING ADDRESS OF DATA VECTOR
LOOP1:  BIT #200,@#IOC        ;TEST DONE BIT
      BEQ LOOP1               ;LOOP IF NOT SET
      MOV @#IOI,R1            ;READ TO CLR DONE BIT,THROW AWAY 1ST SAMPLE
CONV:   BIT #200,@#IOC        ;TEST DONE BIT
      BEQ CONV                ;LOOP IF NOT SET
      MOV @#IOI,(R3)+         ;READ A/D AND PUT DATA IN MEMORY
      DEC R0
      BNE CONV                ;CHECK TO SEE IF N SAMPLES HAVE BEEN TAKEN
      RTS PC
      .END
```

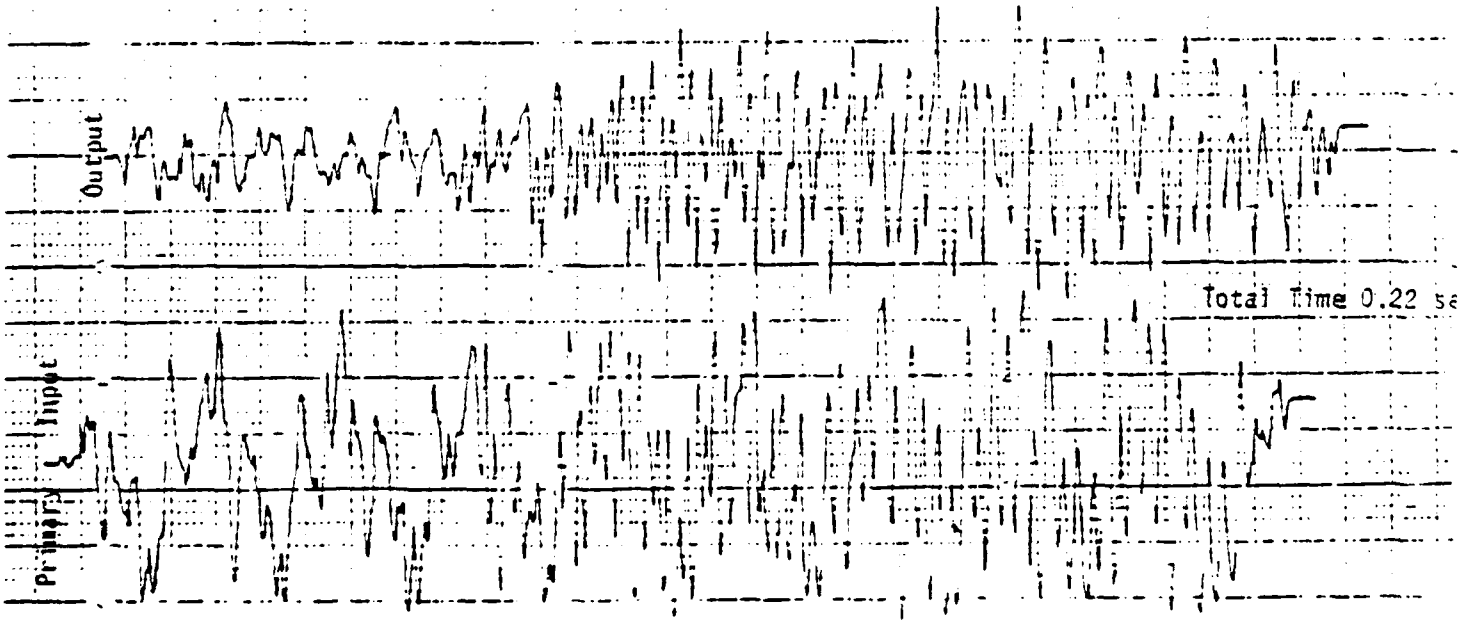
```
250             CONTINUE
C
C       GET N SAMPLES FROM SELECTED CHANNEL AND RETURN AS IX
C
300     CALL XTSAMP(N,NC,IX)
        WRITE (7,400)
400     FORMAT (' SAMPLING FINISHED')
C
C       INITIALIZE PLOTTER
C
        CALL GON
        CALL CLEAR
        CALL PLIMITS (.30,9.5,.3,6.5)
        CALL LIMITS(-FLOAT(N)/8.0,FLOAT(N),-1000.0,3000.0)
500     NP=(N/NW)
        CALL MOVE (-FLOAT(N)/9.0,50.0)
        CALL LABEL (' INPUT',2,0,0)
C
C       PLOT INPUT
C
        DO 525 I=1,N,ND
                CALL LINE (FLOAT(I),FLOAT(IX(I)))
525     CONTINUE
        CALL MOVE (-FLOAT(N)/9.0,2050.0)
        CALL LABEL (' OUTPUT',2,0,0)
C
C       CALCULATE AND PLOT OUTPUT
C
        DO 800 I=1,NP
                DO 700 J=1,NW
                        NS=(I-1)*NW+J
                        X=FLOAT(IX(NS))
                        Y=X-W(J)
                        W(J)=W(J)+Y*G
C
C       PLOT EVERY ND'TH. POINT
C
                        IF (ND1.LT.ND) GO TO 575
                        CALL LINE (FLOAT(NS),Y+2000.)
                        ND1=0
575                     ND1=ND1+1
C
C       CALCULATE INPUT AND OUTPUT POWER FOR LAST TWO PERIODS
C
                        IF (I.LT.NP-2) GO TO 600
                        SQIN=SQIN+.5*X*X/NW
                        SOUT=SOUT+.5*Y*Y/NW
700             CONTINUE
800     CONTINUE
        CALL GOFF
        IF (IFLAG.EQ.0) GO TO 1090
C
C       PRINT OUT 7 PERIODS OF INPUT DATA IN 7 COLUMNS
C
```

```
      PROGRAM PERDIC
C     B. B. PETERSON
C     8 MAY 1984
C     PROGRAM TAKES GIVEN NUMBER OF SAMPLES OF AUDIO WAVEFORM AND
C     CALCULATES HOW MUCH OF WAVEFORM CAN BE CANCELLED BY PERIODIC
C     WAVEFORM AT FUNDAMENTAL FREQUENCY OF ENGINE
C
C     ARRAYS:
C             IX      SIGNAL INPUT
C             C       COMMUNICATIONS VECTOR FOR PLOTTING
C             W       WEIGHT VECTOR
C
      INTEGER IX(10240)
      COMMON C(20)
      DIMENSION W(512)
C
C     SET FLAG TO GET DATA FIRST TIME
C
      IFLAG2 = 1
C
C     ENTER PARAMETERS
C
50    WRITE (7,100)
100   FORMAT (' ENTER # SAMP, # SAMP/PLT, # WGHTS, CH NO, ADAP GAIN')
      READ (5,200) N,ND,NW,NC,G
C
C     SELECT PRINTER AND PLOTTER OPTIONS
C
      WRITE (7,125)
125   FORMAT (' OPTIONS:0=NO PRINTOUT,1=CRT,2=PRINTER')
      READ (5,140) IFLAG
140   FORMAT (I3)
      WRITE (7,145)
145   FORMAT (' 0=PLOTS ON PLOTTER,1=CRT')
      READ (5,140) ID
      C(10)=FLOAT(ID)
C
C     ZERO WEIGHT VECTOR AND INPUT AND OUTPUT POWER
C
      DO 150 J=1,NW
150           W(J)=0.0
      SGIN=0.0
      SOUT=0.0
      IF (N.LT.0) GO TO 1300
200   FORMAT (4I6,F8.5)
C
C     SKIP SAMPLING TO USE OLD DATA
C
      IF (IFLAG2.EQ.0) GO TO 500
      NC=256*NC+16
C
C     DELAY BEFORE SAMPLING
C
      DO 250 J=1,30000
```



No Audic Signal in Primary Input

Figure D7



Audio Signal in Primary input

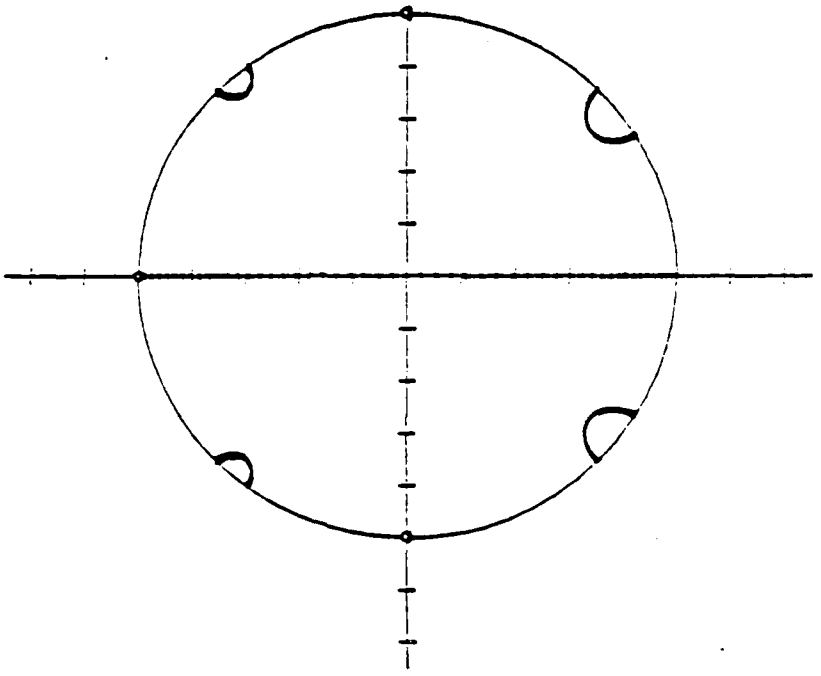
Figure D8

recorder. The plots in Figures D7 and D8 were produced using this program. Enclosure 1 is a listing of the program.

The filter was also implemented in real time on both the TMS 320 and the LSI-11/2. These assembly language program listings are enclosures 2 and 3 respectively. Because the TMS 320 has only 144 words of on-chip data memory, it was necessary to access external memory using the Table Read and Table Write instructions. These instructions require the address to be accessed to be placed in the low order accumulator. This results in relatively complicated code to implement a very simple algorithm. Despite this inefficient use of the TMS 320, the algorithm can still run at sampling rates of up to 150 kHz.

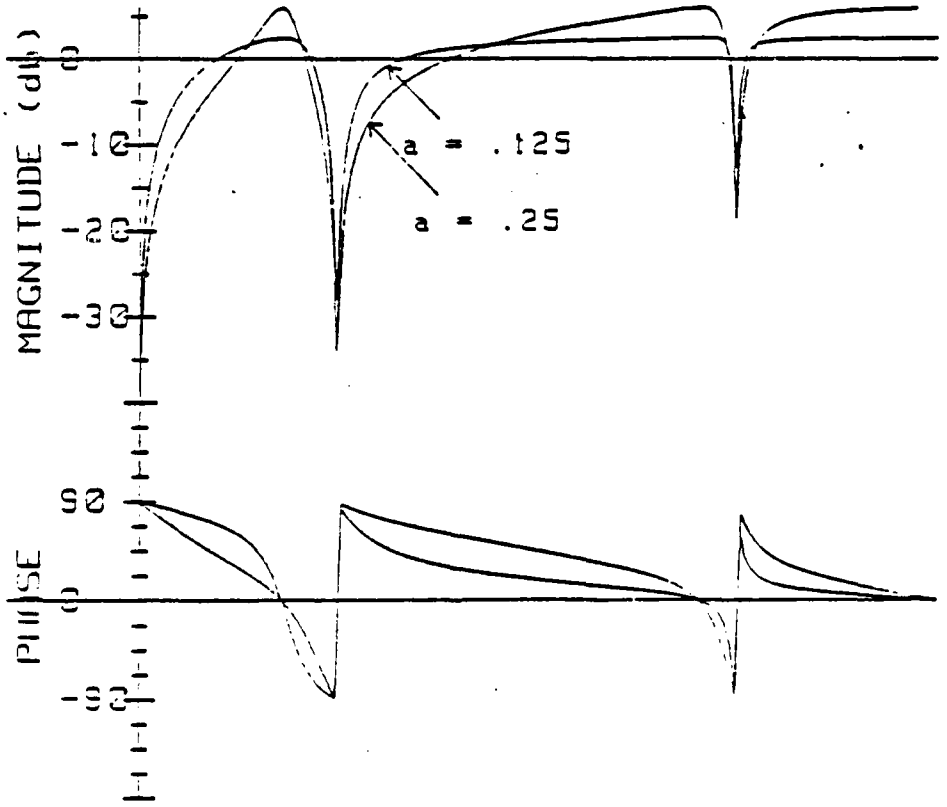
The MACRO-11 (PDP-11 assembly language) program in enclosure 3 illustrates the simplicity of the algorithm. On an LSI-11/2 this program will run at sampling rates of up to 11 kHz.

Cadet 1/c FAVERO is presently attempting to implement the filter on an AIM-65 microcomputer at sampling rates of at least 8 kHz.



ROOT LOCUS OF 8TH ORDER FILTER WHEN REFERENCE INPUT IS A SQUARE WAVE

Figure D5



TRANSFER FUNCTION OF 8TH ORDER FILTER WHEN REFERENCE INPUT IS A SQUARE WAVE

Figure D6

samples, the transfer function from x1(k) to y(k) is given by:

$$\frac{Y(z)}{X_1(z)} = \frac{z^N - 1}{z^N + \sum_{j=0}^{N-1} a\, d_j\, z^j \quad -1}$$

where

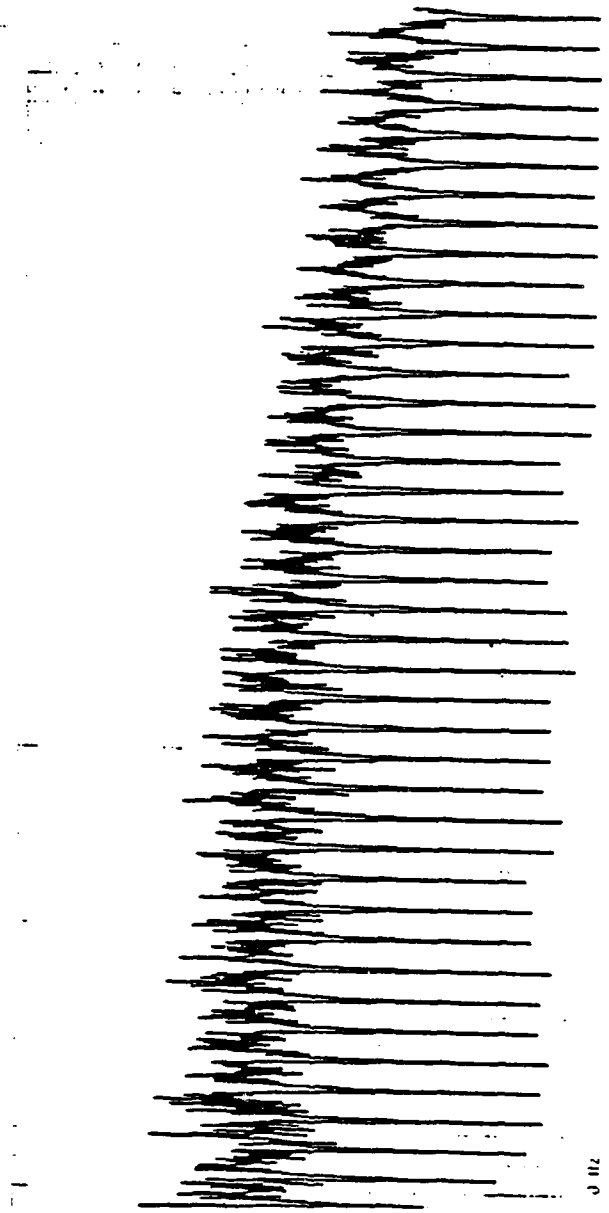
$$d_j = \sum_{i=0}^{N-1} x_2(i)\, x_2(i+j)$$

= N x the autocorrelation of x_2.

Figure D5 is a root locus plot as the adaptive gain, a, is varied for N = 8 and for $x_2(k)$ a square wave and Figure D6 is the frequency response of the same filter. Now the width of each notch is a function the amplitude of the corresponding harmonic in $x_2(k)$ in addition to the adaptive gain. This suggests the possibility of using apriori knowledge of which harmonics are strongest in the noise component of $x_1(k)$ to specify an $x_2(k)$ for more effective filtering.

Another possible interesting application of this type of filter is for very narrow notch and bandpass recursive digital filters. The major advantage is that even with low precision fixed point arithmetic (8 bit, for example) and for poles virtually on the unit circle, the filter is completely stable.

The filter has been implemented in three different ways. First, using off-line processing in FORTRAN, a data vector is obtained by externally triggering the LSI-11/2 A/D converter and then the filter implemented and the data plotted on a strip chart



Frequency Response of Notch Filter
for f_1 = 250, a = 0.25 and f_s = 40 kHz

Figure 14

```
C       INITIALIZE COEFFICIENTS TO CALCULATE WEIGHT VECTOR
C       AND ZERO WEIGHT VECTOR
C
        DO 40 I=1,NW2
                P2(I)=1.0E-10
                CR(I)=1.0E-10
                CI(I)=1.0E-10
                XH(I)=0.0
                YH(I)=0.0
40      CONTINUE
50      NBLOCS=N/NW
        DO 140 J=1,NBLOCS
                JNW=(J-1)*NW
C
C               FILL VECTORS TO CALCULATE FFT
C
                PI=0.0
                PO=0.0
                DO 120 I=1,NW2
                        IA1= 2*I+JNW
                        IA=IA1-1
                        X1(I) = FLOAT(I1(IA)/205)
                        Y1(I) = FLOAT(I1(IA1)/205)
                        X2(I) = FLOAT(I2(IA)/205)
                        Y2(I) = FLOAT(I2(IA1)/205)
120             CONTINUE
                CALL RFFT(NW,MW,NMAX,X1,Y1,WR,WI,1.0)
                CALL RFFT(NW,MW,NMAX,X2,Y2,WR,WI,1.0)
                DO 130 I=1,NW2
C
C                       CALCULATE FFT OF OUTPUT
C
                        XO(I)=X1(I)-XH(I)*X2(I)+YH(I)*Y2(I)
                        YO(I)=Y1(I)-YH(I)*X2(I)-XH(I)*Y2(I)
C
C                       CHECK FOR SMALL VALUES AND SET TO ZERO
C                       TO PREVENT UNDERFLOWS
C
                        IF (ABS(X1(I)).LT.1.0E-10) X1(I)=0.0
                        IF (ABS(Y1(I)).LT.1.0E-10) Y1(I)=0.0
                        IF (ABS(Y2(I)).LT.1.0E-10) Y2(I)=0.0
                        IF (ABS(X2(I)).LT.1.0E-10) X2(I)=0.0
C
C                       CALCULATE INPUT AND OUTPUT POWER
C
                        PI=PI+X1(I)*X1(I)+Y1(I)*Y1(I)
                        PO=PO+XO(I)*XO(I)+YO(I)*YO(I)
C
C                       UPDATE WEIGHTS
C
                        P2(I)=OMG*P2(I)+G*(X2(I)*X2(I)+Y2(I)*Y2(I))
                        CR(I)=OMG*CR(I)+G*(Y1(I)*X2(I)+Y1(I)*Y2(I))
                        CI(I)=OMG*CI(I)+G*(X2(I)*Y1(I)-X1(I)*Y2(I))
                        XH(I)=CR(I)/P2(I)
```

Enclosure 1 to Apperdix E

```
                    YH(I)=CI(I)/P2(I)
130             CONTINUE
C
C               CALCULATE OUTPUT
C
                CALL REFT(NN,MN,NMAX,XO,YO,WR,WI,-1.0)
C
                PO=PO/NN
                PI=PI/NN
                WRITE (7,137) J,PI,PO
137             FORMAT (I6,2F12.2)
140       CONTINUE
          GO TO 5
          END
```

Enclosure 1 to Appendix E FFT subroutines called from main program

```
C       FFT SUBROUTINES
C       D. B. PETERSON  5 FEB 85
C
        SUBROUTINE RFFT(N,M,NMAX,X,Y,WR,WI,DI)
C
C       SUBROUTINE TO CACLULATE N POINT REAL FFT USING
C       N/2 POINT COMPLEX FFT
C
C       REF: "TMS 320 DIGITAL SIGNAL PROCESSING SEMINAR NOTES,"
C       TEXAS INSTRUMENTS, INC., 1983.
C
C       N= # OF REAL DATA POINTS = 2**M
C       X IS REAL PART OF FFT (N/2 VECTOR) Y IS IMAG PART
C       WR AND WI ARE VECTORS OF COSINES AND SINES OF SIZE NMAX
C       DI= 1. FOR FORWARD FFT, -1. FOR  INVERSE.
C       FOR FORWARD FFT X AND Y SHOULD BE GENERATED BY
C       X(I) = DATA(2*I-1) AND Y(I) = DATA(2*I)
C       WHERE DATA() IS THE INPUT VECTOR OF REAL POINTS
C
        REAL X(1),Y(1),WR(1),WI(1)
        N2=N/2
        MM1 = M-1
        IF (DI.LT.0.0) GO TO 10
        CALL FFT2(N2,MM1,NMAX,X,Y,WR,WI,1.0)
10      NMON=NMAX/N
        N4=N2/2
        Y(N2+1)=Y(1)
        X(N2+1)=X(1)
        IP=1
        DO 20 I= 1, N4+1
                NMI=N2-I+2
                RA=X(I)+X(NMI)
                XA=Y(I)-Y(NMI)
                RB=(X(NMI)-X(I))
                XB=(-Y(I)-Y(NMI))
                CPH=WR(IP)
                SPH=-WI(IP)
                IF (DI.GT.0.0) GO TO 15
                RB=-RB
                XB=-XB
                SPH=-SPH
15              IP=IP+NMON
                RC=SPH*RB-CPH*XB
                XC=SPH*XB+CPH*RB
                X(I)=.5*(RA+RC)
                Y(I)=.5*(XA+XC)
                X(NMI)=.5*(RA-RC)
                Y(NMI)=.5*(XC-XA)
20      CONTINUE
        IF (DI.GT.0.0) GO TO 1000
        CALL FFT2(N2,MM1,NMAX,X,Y,WR,WI,-1.0)
1000    RETURN
        END
C
```

Enclosure 1 to Appendix E FFT subroutines called from main program

```
C       COMPLEX FFT WITH TABLE LOOKUP
C       REF:C. S. BURRIS AND T. W. PARKS,"DFT/FFT AND CONVOLUTION
C       ALGORITHMS," JOHN WILEY & SONS, NEW YORK, 1984
C
C       CALLING VARIABLES:
C               N       NUMBER OF COMPLEX POINTS
C               M       LOG2 OF N
C               NMAX    MAXIMUM VALUE OF N FOR INTERPRETING LOOKUP TABLE
C               X & Y   REAL AND IMAGINARY PARTS OF FFT
C               WR      TABLE OF COSINES
C               WI      TABLES OF -SINES
C               DI      +1.0 FOR FORWARD FFT, -1.0 FOR INVERSE
C
        SUBROUTINE FFT2(N,M,NMAX,X,Y,WR,WI,DI)
        REAL Y(1),X(1),WR(1),WI(1)
        NMON=NMAX/N
        N2=N
        DO 100 K=1,M
                N1=N2
                N2=N2/2
                IE=N/N1
                IA=1
                DO 50 J=1,N2
                        IA1=(IA-1)*NMON+1
                        C=WR(IA1)
                        S=-DI*WI(IA1)
                        IA=IA+IE
                        DO 40 I=J,N,N1
                                L=I+N2
                                XT=X(I)-X(L)
                                X(I)=X(I)+X(L)
                                YT=Y(I)-Y(L)
                                Y(I)=Y(I)+Y(L)
                                X(L)=C*XT+S*YT
                                Y(L)=C*YT-S*XT
40                      CONTINUE
50              CONTINUE
100     CONTINUE
        J=1
        N1=N-1
        DO 104 I=1,N1
                IF (I.GE.J)  GO TO 101
                XT=X(J)
                X(J)=X(I)
                X(I)=XT
                YT=Y(J)
                Y(J)=Y(I)
                Y(I)=YT
101             K=N/2
102             IF (K.GE.J)  GO TO 103
                J=J-K
                K=K/2
                GO TO 102
103             J=J+K
```

Enclosure 1 to Appendix E FFT subroutines called from main program

```
104     CONTINUE
        IF (DI.GT.0.0) GO TO 106
        DO 105 I=1,N
        X(I)=X(I)/N
        Y(I)=Y(I)/N
105     CONTINUE
106     RETURN
        END
C
        SUBROUTINE PACK (N,NMAX,IX,X,Y,IW,WIN)
C
C       PUTS INTEGER DATA IN REAL ARRAYS SUITABLE FOR FFT SUB ABOVE
C       MULTIPLIES EACH DATA POINT BY HAMMING WINDOW
C
        REAL X(1),Y(1),WIN(1)
        INTEGER IX(1)
        NMON=NMAX/N
        N2=N/2
        DO 10 I=1,N2
                I2=I*2
                X(I)=FLOAT (IX(I2-1))
                Y(I)=FLOAT (IX(I2))
                IF (IW.EQ.0) GO TO 10
                I3=(I2-2)*NMON+1
                X(I)=X(I)*WIN(I3)
                Y(I)=Y(I)*WIN(NMON+I3)
10      CONTINUE
        RETURN
        END
```

Appendix F LMS Adaptive Filter FORTRAN Program where Audio Signal is added in Software

```
      PROGRAM ADAPT
C     B. B. PETERSON
C     PROGRAM READS IN ANALOG DATA FROM AN FM TUNER
C     DECK USING SUBROUTINE XTSMP2. IT THEN SAMPLES TWO CHANNELS FROM
C     A TAPE DECK.  THE FM TUNER SIGNAL IS MULT. BY A GAIN AND ADDED
C     TO ONE NOISE CHANNEL TO GENERATE THE PRIMARY INPUT.THE OTHER NOISE
C     CHANNEL IS ADAPTIVELY FILTERED AND SUBTRACTED FROM THE DELAYED
C     PRIMARY INPUT.  THE WEIGHTS IN THE ADAPTIVE FILTER ARE ADJUSTED SO
C     AS TO MINIMIZE THE OUTPUT POWER.  THE REFERENCE INPUT, THE PRIMARY
C     INPUT, THE FM TUNER SIGNAL AND THE OUTPUT ARE PLOTTED ON THE CRT,
C     THE HP 7470 PLOTTER OR BOTH. THE SIGNAL TO NOISE RATIOS OF BOTH
C     INPUT AND OUTPUT ARE CALCULTED AFTER THE WEIGHTS HAVE HAD SOME
C     TIME TO CONVERGE.
C
C     ARRAYS:
C             IY      PRIMARY INPUT
C             IX      REFERENCE INPUT
C             ISIG    INTEGER SIGNAL VECTOR
C             H       WEIGHT VECTOR
C             C       COMMUNICATIONS VECTOR FOR PLTLIB
C
      INTEGER IX(4000),IY(4000),ISIG(4000)
      REAL H(100)
      COMMON C(20)
C
C     INPUT PARAMETERS
C
50    WRITE (7,100)
100   FORMAT (' NEW DATA?, RESET WEIGHT VECTOR? 1=YES,0=NO')
      READ (5,200) IF1,IF2
200   FORMAT (2I6)
      IF (IF1.EQ.0 .AND. IF2.EQ.0) GO TO 450
      WRITE (7,300)
300   FORMAT (' # SAMP,REF INP,PRI INP,SIG INP,# WTS')
      READ (5,325) N,NC1,NC2,NC3,NW
325   FORMAT (5I6)
      NT=NW/2
      N2=N/2
      WRITE (7,350)
350   FORMAT (' PLOT RAW DATA?,0=7470,1=GIGI,2=BOTH,-1=NO PLOT,STEP')
      READ (5,200) IC10,ISTP
      STP=FLOAT(ISTP)
      IF (IC10.LT.0) GO TO 370
      C(10)=FLOAT (IC10)
370   NC1=256*NC1+16
      NC2=256*NC2+16
      NC3=256*NC3+16
450   WRITE (7,500)
500   FORMAT (' ADAPTIVE GAIN (E FORMAT),SIGNAL GAIN (I FORMAT)')
      READ (5,600) G,IG
600   FORMAT (E14.4,I6)
      IF (IF2.EQ.0) GO TO 800
C
C     ZERO WEIGHT VECTOR
```

```
C
      DO 700 I=1,NW
700           H(I)=0.0
800   IF (IF1.EQ.0) GO TO 825
C
C     DELAY BEFORE SAMPLING
C
      DO 850 I=1,10000
850   CONTINUE
C
C     SAMPLE SIGNAL FROM TUNER
C
      CALL XTSMP2(N,NC3,NC2,ISIG,IY)
C
C     SAMPLE TWO CHANNELS FROM TAPE DECK
C
      CALL XTSMP2(N,NC1,NC2,IX,IY)
825   IF (IC10.LT.0) GO TO 900
      CALL PLIMITS (0.0,10.0,0.0,7.0)
      CALL LIMITS (0.0,FLOAT(N),-.8*STP,3.6*STP)
C     IF (IF1.EQ.0) GO TO 900
      DO 875 I=1,N
           CALL LINE (FLOAT(I),FLOAT(IX(I)))
875   CONTINUE
      CALL MOVE (1.0,STP)
      DO 890 I=1,N
           ISIG(I)=IG*ISIG(I)
           IY(I)=IY(I)+ISIG(I)
           CALL LINE (FLOAT(I),FLOAT(ISTP+IY(I)))
890   CONTINUE
      CALL MOVE (1.0,2*STP)
      OFFSET= 2.0*STP
      DO 895 I=1,N
           CALL LINE (FLOAT(I),FLOAT(ISIG(I))+OFFSET)
895   CONTINUE
      OFFSET=3.0*STP
      CALL MOVE (1.0,OFFSET)
C
C     ZERO SQUARE      PLT NOISE,OUTPUT NOISE, AND DESIRED OUTPUT.
C
900   SQIN=0.0
      SQOUT=0.0
      SQSIG=0.0
C
C     MAIN ADAPTIVE FILTER LOOP
C
      DO 1000 I=NW+1,N
           FOUT=0.0
C
C          CALCULATE FILTER OUTPUT
C
           DO 950 J=1,NW
                IS=(I-J)
950             FOUT=FOUT+H(J)*IX(IS)
```

```
                ID=(I-NT)
                OUT= IY(ID)-FOUT
C
C               SQUARE AND LOW PASS FILTER INPUT AND OUTPUT
C               FOR LAST HALF OF DATA
C
                IF (I.LT.N2) GO TO 980
                SQOUT=.98*SQOUT+.02*(OUT-FLOAT(ISIG(ID)))**2
                RIX=FLOAT(IY(ID)-ISIG(ID))
                SQIN=.98*SQIN+.02*RIX*RIX
                SQSIG=.98*SQSIG+.02*FLOAT(ISIG(I))*FLOAT(ISIG(I))
                SNRIN=SQSIG/SQIN
                SNROUT=SQSIG/SQOUT
C
C               PLOT OUTPUT
C
980             CALL LINE (FLOAT(I),OUT+OFFSET)
C
C               ADAPT WEIGHT VECTOR
C
960             DO 975 J=1,NW
                        IS=(I-J)
975                     H(J)=H(J)+G*OUT*IX(IS)
1000    CONTINUE
        CALL TURNOF
        WRITE (7,1050) SNRIN,SNROUT
1050    FORMAT (2F10.5)
1099    GO TO 50
1100    STOP
        END
```

References

[1] B. Widrow, et. al. "Adaptive Noise Cancelling: Principles and Applications," **Proceedings of the IEEE**, Vol. 63, No. 12, pp. 1692-1716, December 1975.

[2] J. R. Glover, Jr., "Adaptive Noise Cancelling Applied to Sinusoidal Interferences," **IEEE Trans. on Acoustics, Speech, and Signal Processing**, Vol. ASSP-25, No. 6, pp. 484-491, December 1977.

[3] R. H. Frazier, et. al., "Enhancement of Speech by Adaptive Filtering," **Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing**, pp. 251-253, April 1976.

[4] J. S. Lim, A. V. Oppenheim, and L. D. Braida, "Evaluation of an Adaptive Comb Filtering Method for Enhancing Speech Degraded by White Noise Addition," **IEEE Trans. on Acoustics, Speech, and Signal Processing**, Vol. ASSP-26, No. 5, pp. 354-358, August 1978.

[5] M. Dentino, J. McCool, and B. Widrow, "Adaptive Filtering in the Frequency Domain," **Proc. IEEE**, Vol. 66, No. 12, pp 1658-1659, December 1978.

[6] B. Widrow, J. McCool, and M. Ball, "The Complex LMS Algorithm," **Proc. IEEE**, Vol. 63, No. 4, pp. 719-720, April 1975.

[7] B. Gold and J. Tierney, "Vocoder Analysis Based on Properties of the Human Auditory System," Technical Report 670, Lincoln Laboratory, Massachusetts Institute of Technology, December 1983.

[8] B. Gold, "Robust Vocoding: Is Pitch the Problem?" **IEEE Digital Signal Processing Workshop**, p. 4.1, Chatham, MA, October 8-10, 1984.

[9] M. R. Sambur and N. S. Jayant, "LPC Analysis/Synthesis from Speech Inputs Containing Quantizing Noise or Additive White Noise," **IEEE Trans. on Acoustics, Speech and Signal Processing**, Vol. ASSP-24, No. 6, pp. 488-494, December 1976.

[10] H. Kobatake, J. Inari, and S. Kakuta, "Linear Predictive Coding of Speech Signals in a High Ambient Noise Environment," **IEEE International Conference on Acoustics, Speech, and Signal Processing**, pp. 472-475, June 1978.

[11] W. D. Voiers, A. D. Sharply, and C. J. Hemsoth, "Research on Diagnostic Evaluation of Speech Intelligibility," Final Report, Contract No. AF19628-70-C-0182, Air Force Cambridge Research Laboratories, 1973.

[12] T. E. Tremain, "The Government Standard Linear Predictive Coding Algorithm: LPC-10," **Speech Technology**, pp. 40-49, April 1982.

[13] B. B. Peterson, "A Proposal for Research on Application of Adaptive Noise Cancellation to Coast Guard Voice Communications," U. S. Coast Guard Academy, April 1983.

[14] **TMS 32020 Users Guide, Preliminary Draft,** Texas Instruments, Inc.,Houston, Texas, January 1985.

[15] S. W. Director, **Circuit Theory: A Computational Approach,** John Wiley & Sons, Inc, New York, 1975.

[16] B. B. Peterson, "Adaptive Noise Cancellation Applied to Coast Guard Voice Communications, Interim Report," U. S. Coast Guard Academy, July 1984.

[17] **Details on Signal Processing,** Issue 1, Texas Instruments, Inc. March1984.

[18] **Microcomputers and Memories,** Digital Equipment Corporation, 1982.

[19] **Model 562 Data Sheet,** Shure Brothers, Inc. 1980.

[20] Robert Olson, "Noise Cancelling Boom Microphone," **Electronics Systems Information Bulletin,** Commandant (G-TES), U. S. Coast Guard, November 1983.

[21] J. J. Wolcin, "A General Method and FORTRAN program For the Design of Recursive Digital Filters," NUSC Technical Report 4629, Naval Underwater Systems Center, New London Laboratory, September 1973.

[22] V. R. Viswanathan, et. al. "Multisensor Speech Input," RADC-TR-83-274, Rome Air Development Center, December 1983.

[23] C. F. Teacher and A. J. Brouns, "Microphone for Very High Noise Environments," **IEEE Digital Signal Processing Workshop**, pp. 4.2.1-2, Chatham, MA, October 1984.

END

FILMED

7-85

DTIC