Texas Instruments

Design of Active Noise Control Systems With the TMS320 Family

Application Report

1996 Digital Signal Processing Solutions

Printed in U.S.A., June 1996

SPRA042

Application Report

Design of Active Noise Control Systems With the TMS320 Family

1996

If the spine is too narrow to print this text on, reduce ALL spine copy (including TI bug at the top of the spine and the year at the bottom) the same amount and reposition at the reference marks as shown for the blueline.

If the reduction required is such that the resulting copy is very small, we may opt to print the spine with no text.

Design of Active Noise Control Systems With the TMS320 Family

Sen M. Kuo, Ph.D.
Issa Panahi, Ph.D.
Kai M. Chung
Tom Horner
Mark Nadeski
Jason Chyan

Digital Signal Processing Products—Semiconductor Group

SPRA042
June 1996

PRINTED WITH SOY INK

TEXAS INSTRUMENTS

Printed on Recycled Paper

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain applications using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

Copyright © 1996, Texas Instruments Incorporated

Content

Title *Page*

ABSTRACT **1**

INTRODUCTION **3**
The General Concept of Acoustic Noise Control 3
General Applications of Active Noise Control 4
The Development of Active Techniques for Acoustic Noise Control 5

EVALUATING THE PERFORMANCE OF ANC SYSTEMS **7**

TYPES OF ANC SYSTEMS **9**
The Broadband Feedforward System 9
The Narrowband Feedforward System 10
The Feedback ANC System 11
The Multiple-Channel ANC System 12

ALGORITHMS FOR ANC SYSTEMs **13**
Algorithms for Broadband Feedforward ANC Systems 13
Secondary-Path Effects 14
Filtered-X Least-Mean-Square (FXLMS) Algorithm 15
Leaky FXLMS Algorithm 20
Acoustic Feedback Effects and Solutions (FBFXLMS Algorithm) 20
Filtered-U Recursive LMS (RLMS) Algorithm 24
Algorithms for Narrowband Feedforward ANC Systems 27
Waveform Synthesis Method of Synthesizing the Reference Signal (Essex Algorithm) 27
Adaptive Notch Filters 31
Algorithms for Feedback ANC Systems 35

DESIGN OF ANC SYSTEMS **39**
System Considerations 39
Sampling Rate and Filter Length 40
Coherence Function 41
Causality 42
Constraints and Solutions 43
Automatic Gain Controller 44
Antialiasing and Reconstruction Analog Filters 45
Analog Interface 46

ANC SYSTEM SOFTWARE **47**
Implementation Considerations 47
Quantization Effects in Digital Adaptive Filters 47
Real-Time Software Implementation Process 50
Implementation of Adaptive Filters With the TMS320C25 51
Using the TMS320C2x Simulator to Observe Noise Cancellation 55
Understanding How Individual Parameters Affect Algorithm Performance 56

PHYSICAL SETUP OF EXPERIMENTAL ANC SYSTEM IN AN ACOUSTIC DUCT 59

OPTIMIZATION OF THE EXPERIMENTAL SYSTEM 61
Determining the Value of μ 61
Determining the Value of LEAKY 63
Determining the Gain of the Preamplifier 64
Single-Tone Sinusoidal Noise Source Case 66
Multiple-Tone Sinusoidal Noise Source Case 69

CONCLUSION 75

REFERENCES 77

Appendixes

Title *Page*

APPENDIX A: PSEUDO RANDOM NUMBER GENERATOR 81

APPENDIX B: DIGITAL SINE-WAVE GENERATOR 83
Table Look-Up Method 83
Digital Oscillator 84

APPENDIX C: TMS320C25 ARIEL BOARD IMPLEMENTATION OF ANC ALGORITHMS 85
The Filtered-X LMS Algorithm 85
Filtered-U RLMS Algorithm 95
Filtered-X LMS Algorithm With Feedback Cancellation 107

APPENDIX D: GENERAL CONFIGURABLE SOFTWARE FOR ANC EVALUATION 121
Configuration File (config.asm) Description 122
ANC Algorithm Module Listing (anc.asm) 127
ANC Linker Command File (anc.cmd) 138
ANC System Configuration File (config.asm) 139
TMS320C2x EVM Initialization Command File (evminit.cmd) 141
Global Constants and Variables (globals.asm) 141
System Initialization File (init.asm) 144
Macro Library File (macros.asm) 147
ANC System Supervisor Program (main.asm) 148
Memory Definitions File (memory.asm) 149
Simulation Models and Waveform Generators File (models.asm) 152
Interrupt Vectors and Interrupt Service Routine Traps File (vectors.asm) 155

APPENDIX E: SCHEMATIC DIAGRAM OF 8-ORDER BUTTERWORTH LOW-PASS FILTER 157

APPENDIX F: ANC UNIT SYSTEM SETUP AND OPERATION PROCEDURE 159
Hardware 159
Software 159
Operation Procedure 160

APPENDIX G: TMS320C26 DSP STARTER KIT, AN ALTERNATIVE TO THE SPECTRUM ANALYZER 161

List of Illustrations

Figure *Title* *Page*

1 Physical Concept of Active Noise Cancellation 4
2 Single-Channel Broadband Feedforward ANC System in a Duct 10
3 Narrowband Feedforward ANC System 10
4 Feedback ANC System 11
5 Multiple-Channel ANC System for a 3-D Enclosure 12
6 System Identification Approach to Broadband Feedforward ANC 14
7 Block Diagram of ANC System Modified to Include H(z) 14
8 Block Diagram of the FXLMS Algorithm for ANC 16
9 Experimental Setup for the Off-Line Secondary-Path Modeling 18
10 Active Noise Control Using the FXLMS Algorithm 19
11 ANC System With Acoustic Feedback Cancellation 21
12 Off-Line Modeling of Secondary and Feedback Paths 22
13 ANC System With the Filtered-U RLMS Algorithm 25
14 Spectrum of Original Noise Signal 27
15 Pole-Zero Placement in z Plane 30
16 Effect of Pole on Notch Bandwidth 31
17 Single-Tone ANC System With Adaptive Notch Filter 32
18 Multiple 2-Weight Adaptive Filters in Parallel 35
19 Block Diagram of the Feedback ANC System 36
20 Probe Tube Used to Increase Coherence 41
21 Microphone Mounting Method to Reduce Flow Turbulence 42
22 ANC System in Duct-Like Machine Chamber 44
23 TMS320C25-Based ANC System Hardware 44
24 Block Diagram of an AGC 45
25 Fixed-Point Arithmetic Model of the LMS Algorithm 48
26 Adaptive Filter Implementation Process 51
27 Memory Layout of Weight Vector and Data Vector 53
28 TMS320C25 Central Arithmetic Logic Unit (CALU) 54
29 The Error Signal Imported From MATLAB 56
30 Error Signal Generated With $\mu = 2048$ 56
31 Experimental Setup of the One-Dimensional Acoustic ANC Duct System 60
32 Level of Attenuation of the Noise Source Versus μ 62
33 Overall Performance as a Function of Equation (95) 63
34 Noise Reduction of System as a Function of LEAKY 64
35 Noise Reduction of the System as a Function of Preamplifier Gain 65
36 Error Spectra for FXLMS Algorithm, Noise Source Is a 200-Hz Single-Tone Sinusoid 66

37 Frequency Response of Primary Path P(z) 68
38 Frequency Response of Secondary Path H(z) 68
39 Frequency Response of Feedback Path F(z) 69
40 Error Spectra for FXLMS Algorithm, Noise Source Is a 3-Tone Sinusoid, Order of W(z) = 64, Order of C(z) = 64 70
41 Error Spectra for FXLMS Algorithm, Noise Source Is a 3-Tone Sinusoid, Order of W(z) = 127, Order of C(z) = 128 71
42 Error Spectra for FBFXLMS Algorithm, Noise Source Is a 3-Tone Sinusoid, Order of W(z) = 64, Order of C(z) = 64, Order of D(z) = 64 72
43 Error Spectra for FURLMS Algorithm, Noise Source Is a 3-Tone Sinusoid, Order of A(z) = 63, Order of B(z) = 63, Order of C(z) = 64 73
44 Pseudo Random Number Generator, 16-Bit Case 81
45 How Constants Are Used in Modeling Acoustic-Channel Transfer Function s126

List of Tables

Table *Title* *Page*
1 Complexity of Broadband ANC and Narrowband ANC 29
2 Performance of the System as a Function of μ 61
3 Noise Attenuation for a Single-Tone Sinusoidal Noise Source 67
4 Filter Orders for 3-Tone Sinusoidal Noise Source 69
5 Section 1 of the Configuration File 122
6 Section 2 of the Configuration File 123
7 Number of Instruction Cycles, DSP Execution Time, and TMS320C25 DSP Overhead per Algorithm 125
8 How Output Signal Arrays Are Used With Various Algorithms 126

Program Listings

Title *Page*
The Filtered-X LMS Algorithm 85
Filtered-U RLMS Algorithm 95
Filtered-X LMS Algorithm With Feedback Cancellation 107
ANC Algorithm Module Listing (anc.asm) 127
ANC Linker Command File (anc.cmd) 138
ANC System Configuration File (config.asm) 139
TMS320C2x EVM Initialization Command File (evminit.cmd) 141
Global Constants and Variables (globals.asm) 141
System Initialization File (init.asm) 144
Macro Library File (macros.asm) 147
ANC System Supervisor Program (main.asm) 148
Memory Definitions File (memory.asm) 149
Simulation Models and Waveform Generators File (models.asm) 152
Interrupt Vectors and Interrupt Service Routine Traps File (vectors.asm) 155

ABSTRACT

An active noise control (ANC) system based on adaptive filter theory was developed in the 1980s; however, only with the recent introduction of powerful but inexpensive digital signal processor (DSP) hardware, such as the TMS320 family, has the technology become practical. The specialized DSPs were designed for real-time numerical processing of digitized signals. These devices have enabled the low-cost implementation of powerful adaptive ANC algorithms and encouraged the widespread development of ANC systems. ANC that uses adaptive signal processing implemented on a low-cost, high-performance DSP is an emerging new technology.

This application report presents general background information about ANC methods. Contrasts between passive and active noise control are described, and the circumstances under which ANC is preferable are shown. Different types of noise-control algorithms are discussed: feedforward broadband, feedforward narrowband, and feedback algorithms. The report details the design of a simple ANC system using a TMS320 DSP and the implementation of that design.

INTRODUCTION

The General Concept of Acoustic Noise Control

Acoustic noise problems in the environment become more noticeable for several reasons:

- Increased numbers of large industrial equipments being used:
  - Engines
  - Blowers
  - Fans
  - Transformers
  - Compressors
  - Motors
- The growth of high-density housing increases the population's exposure to noise because of the proximity to neighbors and traffic
- The use of lighter materials for building and transportation equipment, resulting from cost constraints in construction and fabrication

Two types of acoustic noise exist in the environment. One is caused by turbulence and is totally random. Turbulent noise distributes its energy evenly across the frequency bands. It is referred to as broadband noise, and examples are the low-frequency sounds of jet planes and the impulse noise of an explosion. Another type of noise, called narrowband noise, concentrates most of its energy at specific frequencies. This type of noise is related to rotating or repetitive machines, so it is periodic or nearly periodic. Examples of narrowband noise include the noise of internal combustion engines in transportation, compressors as auxiliary power sources and in refrigerators, and vacuum pumps used to transfer bulk materials in many industries.

There are two approaches to controlling acoustic noise: passive and active. The traditional approach to acoustic noise control uses passive techniques such as enclosures, barriers, and silencers to attenuate the undesired noise. Passive silencers use either the concept of impedance change caused by a combination of baffles and tubes to silence the undesired sound (reactive silencers) or the concept of energy loss caused by sound propagation in a duct lined with sound-absorbing material to provide the silencing (resistive silencers). Reactive silencers are commonly used as mufflers on internal combustion engines, while resistive silencers are used mostly for duct-borne fan noise. These passive silencers are valued for their high attenuation over a broad frequency range. However, they are relatively large, costly, and ineffective at low frequencies, making the passive approach to noise reduction often impractical. Furthermore, these silencers often create an undesired back pressure if there is airflow in the duct.

In an effort to overcome these problems, considerable interest has been shown in active noise control. The active noise control system contains an electroacoustic device that cancels the unwanted sound by generating an antisound (antinoise) of equal amplitude and opposite phase. The original, unwanted sound and the antinoise acoustically combine, resulting in the cancellation of both sounds. Figure 1 shows the waveforms of the unwanted noise (the primary noise), the canceling noise (the antinoise), and the residual noise that results when they superimpose. The effectiveness of cancellation of the primary noise depends on the accuracy of the amplitude and phase of the generated antinoise.

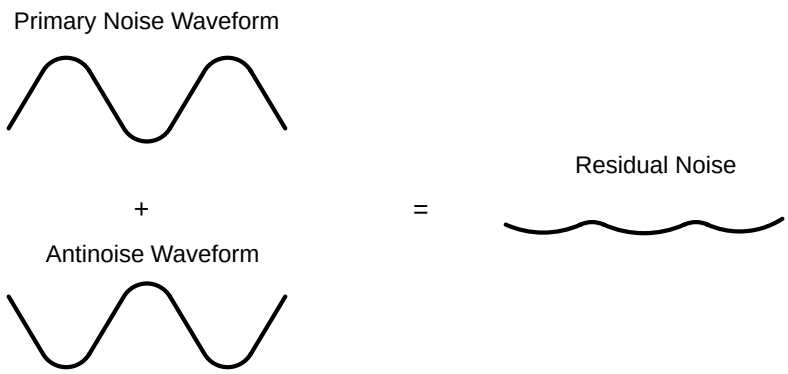


Figure 1. Physical Concept of Active Noise Cancellation

General Applications of Active Noise Control

The successful application of active control is determined on the basis of its effectiveness compared with passive attenuation techniques. Active attenuation is an attractive means to achieve large amounts of noise reduction in a small package, particularly at low frequencies (below 600 Hz). At low frequencies, where lower sampling rates are adequate and only plane wave propagation is allowed, active control offers real advantages.

From a geometric point of view, active noise control applications can be classified in the following four categories:

- Duct noise: one-dimensional ducts such as ventilation ducts, exhaust ducts, air-conditioning ducts, pipework, etc.
- Interior noise: noise within an enclosed space
- Personal hearing protection: a highly compacted case of interior noise
- Free space noise: noise radiated into open space

Specific applications for active noise control now under development include attenuation of unavoidable noise sources in the following end-equipment:

- Automotive (car, van, truck, earth-moving machine, military vehicle)
  - Single-channel (one-dimensional) systems: Electronic muffler for exhaust system, induction system, etc.
  - Multiple-channel (three-dimensional) systems: Noise attenuation inside passenger compartment and heavy-equipment operator cabin, active engine mount, hands-free cellular phone, etc.
- Appliance
  - Single-channel systems: Air conditioning duct, air conditioner, refrigerator, washing machine, furnace, dehumidifier, etc.
  - Multiple-channel systems: Lawn mower, vacuum cleaner, room isolation (local quiet zone), etc.
- Industrial: fan, air duct, chimney, transformer, blower, compressor, pump, chain saw, wind tunnel, noisy plant (at noise sources or many local quiet zones), public phone booth, office cubicle partition, ear protector, headphones, etc.
- Transportation: airplane, ship, boat, helicopter, snowmobile, motorcycle, diesel locomotive, etc.

The algorithms developed for active noise control can also be applied to active vibration control. Active vibration control can be used for isolating the vibrations from a variety of machines and to stabilizing various platforms in the presence of vibration disturbances. As the performance and reliability continue to improve and the initial cost continues to decline, active systems may become the preferred solution to a variety of vibration-control problems.

The Development of Active Techniques for Acoustic Noise Control

Active noise control is developing rapidly because it permits significant improvements in noise control, often with potential benefits in size, weight, volume, and cost of the system. The book *Active Control of Sound* [1] provides detailed information on active noise control with an emphasis on the acoustic point of view.

The design of an active noise canceler using a microphone and an electronically driven loudspeaker to generate a canceling sound was first proposed and patented by Lueg in 1936 [2]. While the patent outlined the basic idea of ANC, the concept did not have real-world applications at that time. Because the characteristics of an acoustic noise source and the environment are not constant, the frequency content, amplitude, phase, and velocity of the undesired noise are nonstationary (time varying). An active noise control system must be adaptive in order to cope with these changing characteristics.

In the field of digital signal processing, there is a class of adaptive systems in which the coefficients of a digital filter are adjusted to minimize an error signal (the desired signal minus the actual signal; the desired signal is typically defined to be zero). A duct-noise cancellation system based on adaptive filter theory was developed by Burgess in 1981 [3]. Later in the 1980s, research on active noise control was dramatically affected by the development of powerful DSPs and the development of adaptive signal processing algorithms [4]. The specialized DSPs were designed for real-time numerical processing of digitized signals. These devices enabled the low-cost implementation of powerful adaptive algorithms [5] and encouraged the widespread development and application of active noise control systems based on digital adaptive signal processing technology.

Many modern active noise cancelers rely heavily on adaptive signal processing—without adequate consideration of the acoustical elements. If the acoustical design of the system is not optimized, the digital controller may not be able to attenuate the undesired noise adequately. Therefore, it is necessary to understand the acoustics of the installation and to design the system to assist the adaptive active noise controller to carry out its work. For electrical engineers involved in the development of active control systems, Nelson's book [1] provides an excellent introduction to acoustics from the active noise control point of view.

EVALUATING THE PERFORMANCE OF ANC SYSTEMS

Analysis of the performance of a given DSP-based controller for different types of source noise and different ANC algorithms is an integral part of successful and optimal design methodology.

An approach to adaptive ANC performance analysis that involves a hierarchy of techniques, starting with an ideal simplified problem and progressively adding practical constraints and other complexities, was developed by Morgan [8]. Performance analysis provides answers to the following questions:

- What are the fundamental performance limitations?
- What are the practical constraints that limit performance?
- How is performance balanced against complexity?
- What is a practical design architecture?

To aid in answering these questions, four levels of performance analysis are defined:

- Level I derives fundamental performance limits, given continuous measurements over the entire performance surface.
- Level II adds the practical constraint of a fixed number of sensors at discrete locations.
- Level III incorporates knowledge of the transfer function structure between sensor(s) and activator(s).
- Level IV adds in all of the other practical effects and design constraints required for detailed performance calculations.

At each step, a degree of confidence is gained and a benchmark is established for comparison and cross-checking with the next level of complexity.

The principle of ANC is simple; however, when it is applied in the real world, the following questions must be answered [9]:

- Which algorithm should be adopted?
- Where should speakers and microphones be located?
- How is the flow noise (the noise of air passing over the surface of the microphone) going to be reduced?
- How is the power of the speakers going to be increased?
- How is the durability of the microphones and the speakers going to be increased?
- How is the cost of the hardware (controller, microphone, and speaker) going to be reduced?

To be suitable for industrial or commercial use, the ANC system must have certain properties [10]:

- Maximum efficiency over the desired frequency band
- Autonomy with regard to the installation (the system could be built and preset at the time of manufacture and then installed on site)
- Self-adaptability of the system to deal with any variations in the physical parameters (temperature, airflow, etc.)
- Robustness and reliability of the elements of the system and simplification of the control electronics

The continuous progress of active noise control involves the development of improved adaptive signal processing algorithms, transducers, and digital signal processing hardware. More sophisticated adaptive

filtering algorithms allow faster convergence (the equalization of the phase and magnitude of the undesired noise and the antinoise so that cancellation occurs), greater noise attenuation, and are more resistant to interference. The DSP hardware implementation allows these more sophisticated algorithms to be applied in real time to improve system performance.

TYPES OF ANC SYSTEMS

Broadband noise cancellation requires knowledge of the noise source (the primary noise) in order to generate the antinoise signal. The measurement of the primary noise is used as a reference input to the noise canceler. Primary noise that correlates with the reference input signal is canceled downstream of the noise generator (a loudspeaker) when phase and magnitude are correctly modeled in the digital controller.

For narrowband noise cancellation (reduction of periodic noise caused by rotational machinery), active techniques have been developed that are very effective and that do not rely on causality (having prior knowledge of the noise signal). Instead of using an input microphone, a tachometer signal provides information about the primary frequency of the noise generator. Because all of the repetitive noise occurs at harmonics of the machine's basic rotational frequency, the control system can model these known noise frequencies and generate the antinoise signal. This type of control system is desirable in a vehicle cabin, because it will not affect vehicle warning signals, radio performance, or speech, which are not normally synchronized with the engine rotation.

Active noise control systems are based on one of two methods. Feedforward control is where a coherent reference noise input is sensed before it propagates past the canceling speaker. Feedback control [6, 7] is where the active noise controller attempts to cancel the noise without the benefit of an upstream reference input.

Feedforward ANC systems are the main techniques used today. Systems for feedforward ANC are further classified into two categories:

- Adaptive broadband feedforward control with an acoustic input sensor
- Adaptive narrowband feedforward control with a nonacoustic input sensor

The Broadband Feedforward System

A considerable amount of broadband noise is produced in ducts such as exhaust pipes and ventilation systems. A relatively simple feedforward control system for a long, narrow duct is illustrated in Figure 2. A reference signal x(n) is sensed by an input microphone close to the noise source before it passes a loudspeaker. The noise canceler uses the reference input signal to generate a signal y(n) of equal amplitude but 180° out of phase. This antinoise signal is used to drive the loudspeaker to produce a canceling sound that attenuates the primary acoustic noise in the duct.

The basic principle of the broadband feedforward approach is that the propagation time delay between the upstream noise sensor (input microphone) and the active control source (speaker) offers the opportunity to electrically reintroduce the noise at a position in the field where it will cause cancellation. The spacing between the microphone and the loudspeaker must satisfy the principles of causality and high coherence, meaning that the reference must be measured early enough so that the antinoise signal can be generated by the time the noise signal reaches the speaker. Also, the noise signal at the speaker must be very similar to the measured noise at the input input microphone, meaning the acoustic channel cannot significantly change the noise. The noise canceler uses the input signal to generate a signal y(n) that is of equal amplitude and is 180° out of phase with x(n). This noise is output to a loudspeaker and used to cancel the unwanted noise.

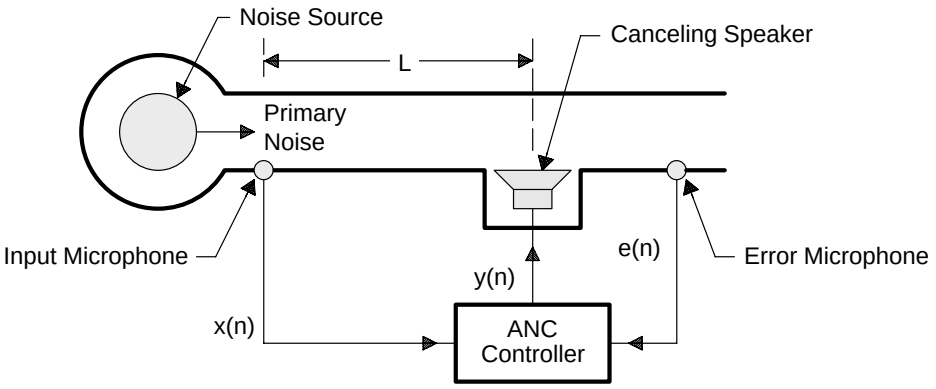


Figure 2. Single-Channel Broadband Feedforward ANC System in a Duct

The error microphone measures the error (or residual) signal e(n), which is used to adapt the filter coefficients to minimize this error. The use of a downstream error signal to adjust the adaptive filter coefficients does not constitute feedback, because the error signal is not compared to the reference input.

Actual implementations require additional considerations to handle acoustic effects in the duct. These considerations are discussed in the section *Algorithms for ANC Systems*, page 13.

The Narrowband Feedforward System

In applications where the primary noise is periodic (or nearly periodic) and is produced by rotating or reciprocating machines, the input microphone can be replaced by a nonacoustic sensor such as a tachometer, an accelerometer, or an optical sensor. This replacement eliminates the problem of acoustic feedback (described in the subsection *Acoustic Feedback Effects and Solutions*, page 20).

The block diagram of a narrowband feedforward active noise control system is shown in Figure 3. The nonacoustic sensor signal is synchronous with the noise source and is used to simulate an input signal that contains the fundamental frequency and all the harmonics of the primary noise. This type of system controls harmonic noises by adaptively filtering the synthesized reference signal to produce a canceling signal. In many cars, trucks, earth moving vehicles, etc., the revolutions per minute (RPM) signal is available and can be used as the reference signal. An error microphone is still required to measure the residual acoustic noise. This error signal is then used to adjust the coefficients of the adaptive filter.

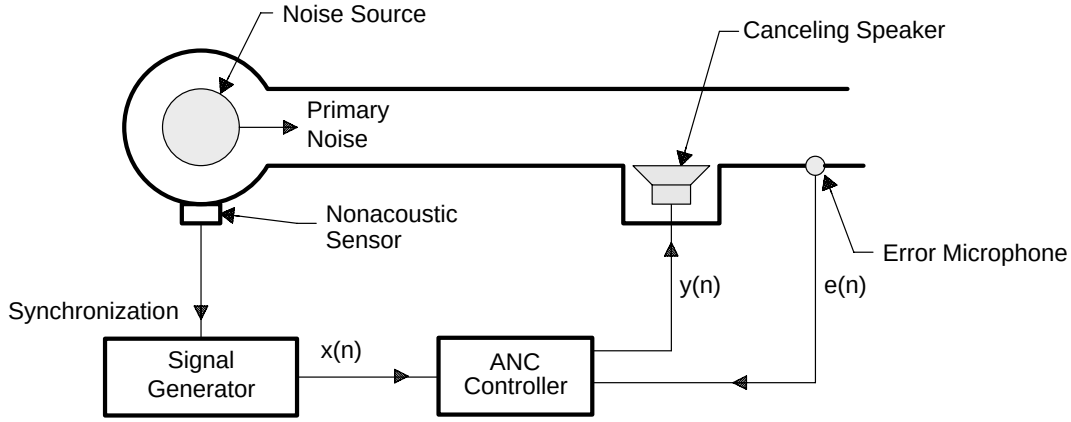


Figure 3. Narrowband Feedforward ANC System

Generally, the advantage of narrowband ANC systems is that the nonacoustic sensors are insensitive to the canceling sound, leading to very robust control systems. Specifically, this technique has the following advantages:

- Environmental and aging problems of the input microphone are automatically eliminated. This is especially important from the engineering viewpoint, because it is difficult to sense the reference noise in high temperatures and in turbulent gas ducts like an engine exhaust system.
- The periodicity of the noise enables the causality constraint to be removed. The noise waveform frequency content is constant. Only adjustments for phase and magnitude are required. This results in more flexible positioning of the canceling speaker and allows longer delays to be introduced by the controller.
- The use of a controller-generated reference signal has the advantage of selective cancellation; that is, it has the ability to control each harmonic independently.
- It is necessary to model only the part of the acoustic plant transfer function relating to the harmonic tones. A lower-order FIR filter can be used, making the active periodic noise control system more computationally efficient.
- The undesired acoustic feedback from the canceling speaker to the input microphone [16] is avoided.

The Feedback ANC System

Feedback active noise control was proposed by Olson and May in 1953 [6]. In this scheme, a microphone is used as an error sensor to detect the undesired noise. The error sensor signal is returned through an amplifier (electronic filter) with magnitude and phase response designed to produce cancellation at the sensor via a loudspeaker located near the microphone. This configuration provides only limited attenuation over a restricted frequency range for periodic or band-limited noise. It also suffers from instability, because of the possibility of positive feedback at high frequencies. However, due to the predictable nature of the narrowband signals, a more robust system that uses the error sensor's output to predict the reference input has been developed (see Figure 4). The regenerated reference input is combined with the narrowband feedforward active noise control system.

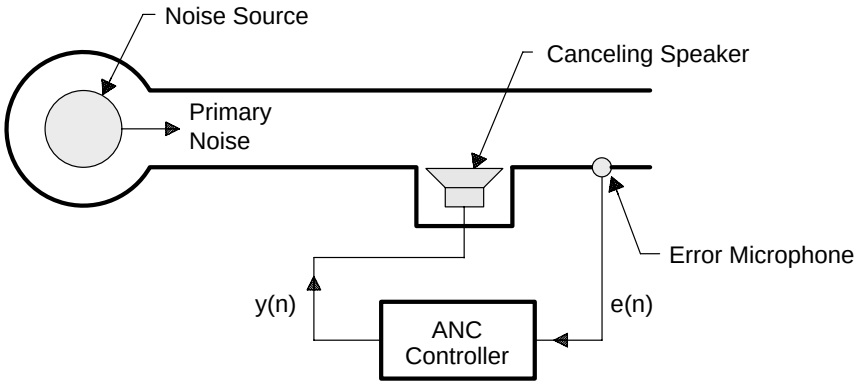


Figure 4. Feedback ANC System

One of the applications of feedback ANC recognized by Olson [7] is controlling the sound field in headphones and hearing protectors [27]. In this application, a system reduces the pressure fluctuations in the cavity close to a listener's ear. This application has been developed and made commercially available.

The Multiple-Channel ANC System

Many applications can display complex modal behavior. These applications include:

- Active noise control in large ducts or enclosures
- Active vibration control on rigid bodies or structures with multiple degrees of freedom
- Active noise control in passenger compartments of aircraft or automobiles

When the geometry of the sound field is complicated, it is no longer sufficient to adjust a single secondary source to cancel the primary noise using a single error microphone. The control of complicated acoustic fields requires both the exploration and development of optimum strategies and the construction of an adequate multiple-channel controller. These tasks require the use of a multiple-input multiple-output adaptive algorithm. The general multiple-channel ANC system involves an array of sensors and actuators. A block diagram of a multiple-channel ANC system for a three-dimensional application is shown in Figure 5.

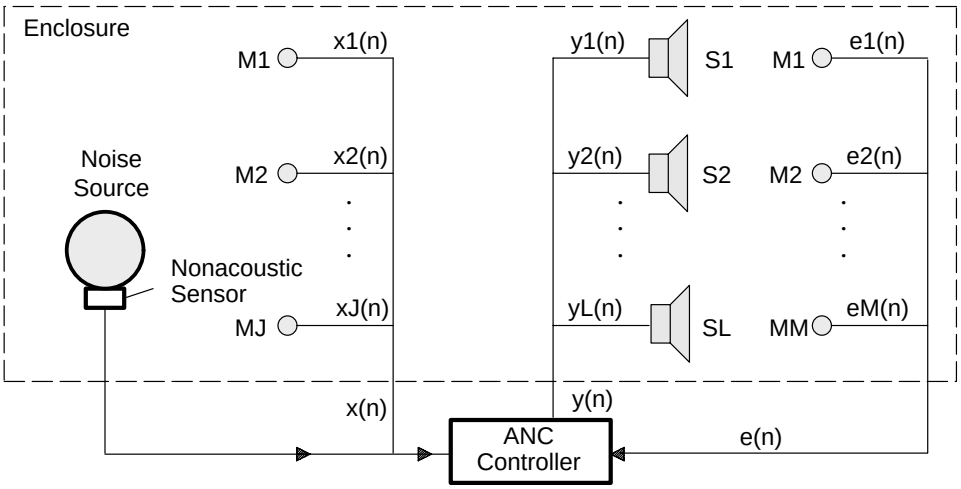


Figure 5. Multiple-Channel ANC System for a 3-D Enclosure

ALGORITHMS FOR ANC SYSTEMS

This section discusses the algorithms used in three kinds of ANC systems:

- Broadband feedforward ANC systems that use acoustic sensor (microphone) input
- Narrowband feedforward ANC systems that use nonacoustic sensor input
- Feedback ANC systems that use only an error sensor

Adaptive filters can be realized as:

- Transversal—finite impulse response (FIR)
- Recursive—infinite impulse response (IIR)
- Lattice filters
- Transform-domain filters

The most common algorithm applied to adaptive filters is the transversal filter using the least mean-squared (LMS) algorithm. The residual noise can be used as an error signal input to an adaptive algorithm that adjusts the filter coefficients to model (estimate) the acoustic-channel effects.

Algorithms for Broadband Feedforward ANC Systems

Broadband active noise control can be described in a system identification framework, as shown in Figure 6. Using a digital frequency-domain representation of the problem, the ideal active noise control system uses an adaptive filter W(z) to estimate the response of an unknown primary acoustic path P(z) between the reference input sensor and the error sensor. The z-transform of e(n) can be expressed as:

$$E(z) = D(z) + Y(z) = X(z)[P(z) + W(z)] \tag{1}$$

where E(z) is the error signal, X(z) is the input signal, and Y(z) is the adaptive filter output. After the adaptive filter W(z) has converged, $E(z) = 0$. Equation (1) becomes:

$$W(z) = -P(z) \tag{2}$$

which implies that:

$$y(n) = -d(n) \tag{3}$$

Therefore, the adaptive filter output y(n) has the same amplitude but is 180° out of phase with the primary noise d(n). When d(n) and y(n) are acoustically combined, the residual error becomes zero, resulting in cancellation of both sounds based on the principle of superposition.

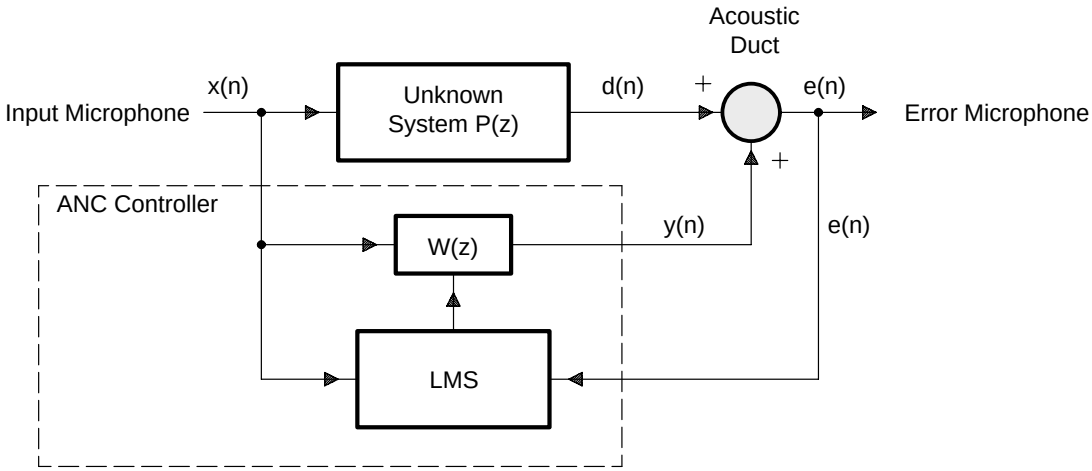


Figure 6. System Identification Approach to Broadband Feedforward ANC

Secondary-Path Effects

The error signal e(n) is measured at the error microphone downstream of the canceling speaker. The summing junction in Figure 6 represents the acoustical environment between the canceling speaker and the error microphone, where the primary noise d(n) is combined with the antinoise y(n) output from the adaptive filter. The antinoise signal can be modified by the secondary-path function H(z) in the acoustic channel from y(n) to e(n), just as the primary noise is modified by the primary path P(z) from the noise source to the error sensor. Therefore, it is necessary to compensate for H(z). A more detailed block diagram of an active noise control system that includes the secondary path H(z) is shown in Figure 7.

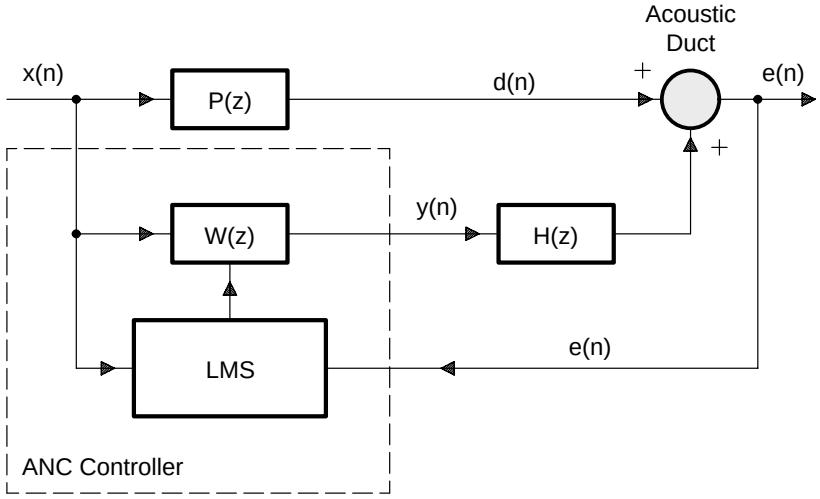


Figure 7. Block Diagram of ANC System Modified to Include H(z)

From Figure 7, the z-transform of error signal e(n) is:

$$E(z) = X(z)\ P(z) + X(z)\ W(z)\ H(z) \tag{4}$$

Assuming that W(z) has sufficient order, after the convergence of the adaptive filter, the residual error is zero (that is, $E(z) = 0$). This result requires W(z) to be:

$$W(z) = \frac{-P(z)}{H(z)} \tag{5}$$

to realize the optimal transfer function.

Thus, the adaptive filter W(z) has to model the primary path P(z) and inversely model the secondary path H(z). However, it is impossible to invert the inherent delay caused by H(z) if the primary path P(z) does not contain a delay of at least equal length. This is the overall limiting causality constraint in broadband feedforward control systems. Furthermore, from equation (5), the control system is unstable if there is a frequency ω such that $H(\omega) = 0$. Also, the control system is ineffective if there is a frequency ω where $P(\omega) = 0$, (that is, a zero in the primary path causes an unobservable control frequency). Therefore, the characteristics of the secondary path H(z) have significant effects on the performance of an ANC system.

Filtered-X Least-Mean-Square (FXLMS) Algorithm

To account for the effects of the secondary-path transfer function H(z), the conventional least-mean-square (LMS) algorithm [4] needs to be modified [3]. To ensure convergence of the algorithm, the input to the error correlator is filtered by a secondary-path estimate C(z). This results in the filtered-X LMS (FXLMS) algorithm developed by Morgan [11]. Burgess [3] has suggested using this FXLMS algorithm to compensate for the effects of the secondary path in ANC applications.

The FXLMS algorithm is illustrated in Figure 8, where the output y(n) is computed as:

$$y(n) = \underline{w}^{T}(n)\underline{x}(n) = \sum_{i=0}^{N-1} w_i(n)x(n-i) \tag{6}$$

where $\underline{w}^T(n) = [w_0(n)\ w_1(n)\ \ldots\ w_{N-1}(n)]^T$ is the coefficient vector of W(z) at time n and $\underline{x}(n) = [x(n)\ x(n-1)\ \ldots\ x(n-N+1)]^T$ is the reference signal vector at time n.

The filter is implemented on a DSP in the form:

$$y(n) = \sum_{i=0}^{N-1} w_i(n)x(n-i)$$

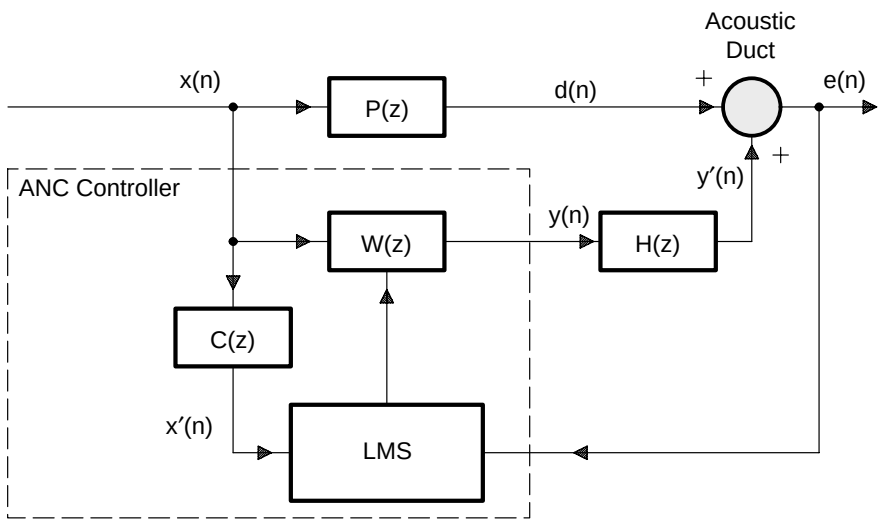


Figure 8. Block Diagram of the FXLMS Algorithm for ANC

The FXLMS algorithm can be expressed as:

$$\underline{w}(n+1) = \underline{w}(n) - \mu e(n)\underline{x}(n)h(n) \tag{7}$$

where μ is the step size of the algorithm that determines the stability and convergence of the algorithm and h(n) is the impulse response of H(z). Therefore, the input vector $\underline{x}(n)$ is filtered by H(z) before updating the weight vector. However, in practical applications, H(z) is unknown and must be estimated by the filter, C(z). Therefore:

$$w_i(n+1) = w_i(n) - \mu e(n)x'(n-i) \qquad i = 0, 1, \ldots, N-1 \tag{8}$$

and:

$$\underline{w}(n+1) = \underline{w}(n) - \mu e(n)\underline{x}'(n) \tag{9}$$

where:

$$x'(n) = \underline{c}^T\underline{x}(n) = \sum_{i=0}^{M-1} c_i x(n-i) \tag{10}$$

is the vector for the filtered version of reference input x′(n) that is computed as:

$$\underline{x}'(n) = [x'(n)\ x'(n-1)\ \ldots\ x'(n-N+1)]^T \tag{11}$$

and:

$$\underline{c} = [c_0 \ c_1 \ ... \ C_{M-1}]^T \tag{12}$$

is the coefficient vector of the secondary-path estimate, C(z).

When this algorithm is implemented, the convergence of the filter can be achieved much more quickly than theory suggests, and the algorithm appears to be very tolerant of errors made in the estimation of the secondary path H(z) by the filter C(z). As shown by Morgan [11], the algorithm still converges with nearly 90° of phase error between C(z) and H(z).

It is important that in equation (7), a minus sign is used for ANC applications instead of a plus sign as in a conventional LMS algorithm. This is because the error signal in an ANC system is $e(n) = d(n) + y'(n)$, due to the fact that the residual error e(n) is the result of acoustic superposition (addition) instead of electrical subtraction.

The transfer function H(z) is unknown and is time-varying due to effects such as aging of the loudspeaker, changes in temperature, and air flow in the secondary path. Thus, several on-line modeling techniques were developed by Eriksson [12]. Assuming the characteristics of H(z) are unknown but time-invariant, an off-line modeling technique can be used to estimate H(z) during a training stage. At the end of training, the estimated model C(z) is fixed and used for active noise control. The experimental setup for the direct off-line system modeling is shown in Figure 9, where an uncorrelated white noise is internally generated by the DSP. The training procedure is summarized following the figure. The algorithm of the white noise generator is given in Appendix A, *Pseudo Random Number Generator.*

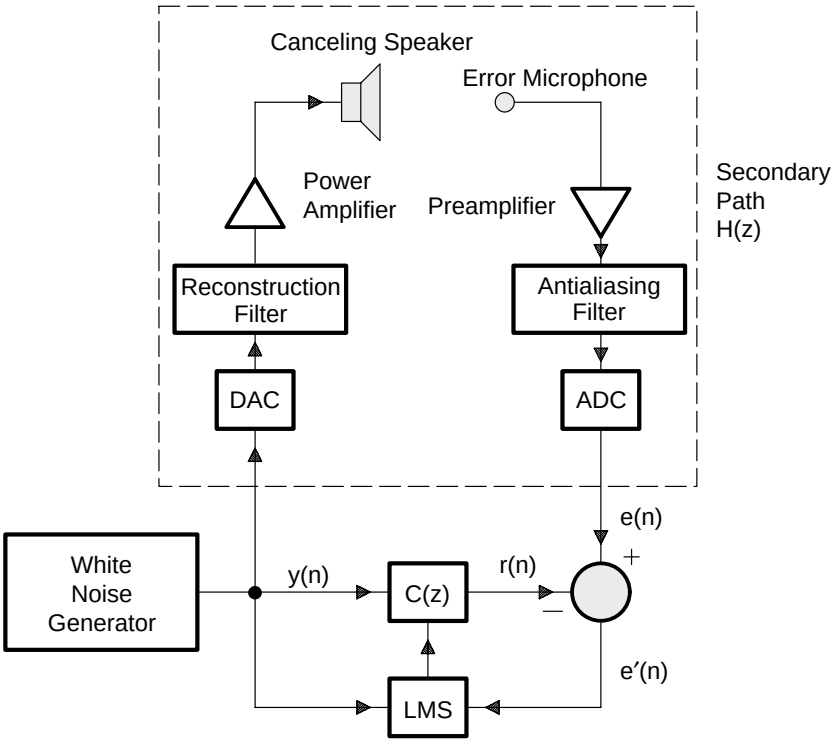


Figure 9. Experimental Setup for the Off-Line Secondary-Path Modeling

1. Generate a sample of white noise y(n) using the algorithm given in Appendix A. Output y(n) to drive the canceling loudspeaker. This internally generated white noise is used as the reference input for the adaptive filter C(z) and the LMS coefficient adaptation algorithm.
2. Input the secondary-path response e(n) from the error microphone.
3. Compute the response of the adaptive model r(n):

$$r(n) = \sum_{i=0}^{M-1} c_i(n)\, y(n - i) \tag{13}$$

where $c_i(n)$ is the ith coefficient of the adaptive filter C(z) at time n and M is the order of filter.

4. Compute the difference:

$$e'(n) = e(n) - r(n) \tag{14}$$

5. Update the coefficients of the adaptive filter C(z) using the LMS algorithm:

$$c_i(n+1) = c_i(n) + \mu e'(n) y(n-i), \quad i = 0, 1, \ldots, M-1 \tag{15}$$

where μ is the step size that must satisfy the following stability condition:

$$0 < \mu < \frac{1}{MP_y} \tag{16}$$

where P_y is the power of the generated white noise y(n).

6. Repeat the procedure for about 10 seconds. Save the coefficients of the adaptive filter C(z) and use them in the following noise cancellation mode.

After the off-line modeling is completed, the system is operated in the active noise cancellation mode. The algorithm is illustrated in Figure 10, and the procedure of on-line noise control is summarized following the figure.

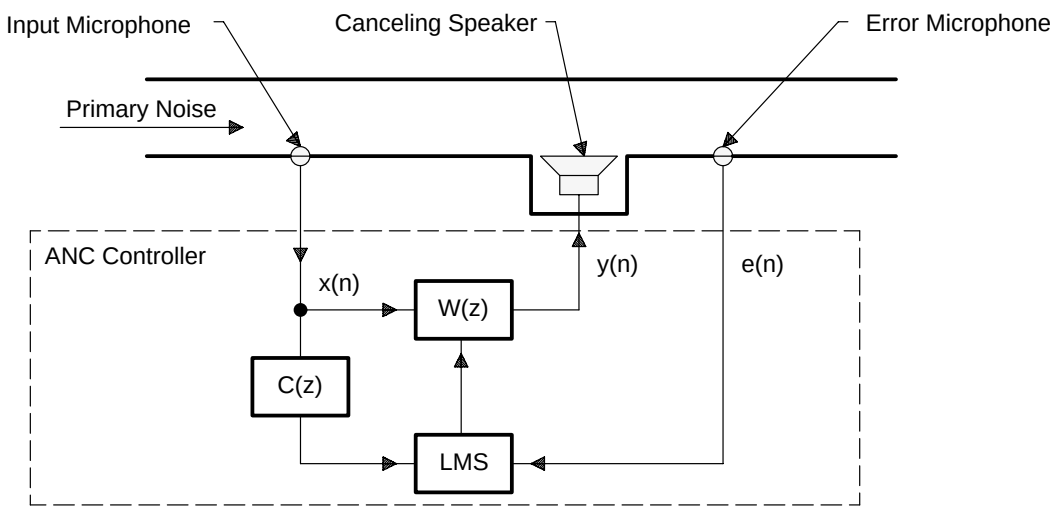


Figure 10. Active Noise Control Using the FXLMS Algorithm

1. Input the reference signal x(n) (from the input microphone) and the error signal e(n) (from the error microphone) from the input ports.

2. Compute the antinoise y(n):

$$y(n) = \sum_{i=0}^{N-1} w_i(n) x(n-i) \tag{17}$$

where $w_i(n)$ is the ith coefficient of the adaptive filter W(z) at time n and N is the order of filter w(z).

3. Output the antinoise y(n) to the output port to drive the canceling loudspeaker.

4. Compute the filtered-X version of $x'(n)$:

$$x'(n) = \sum_{i=0}^{M} c_i x(n - 1) \tag{18}$$

5. Update the coefficients of adaptive filter W(z) using the FXLMS algorithm:

$$w_i(n + 1) = w_i(n) - \mu e(n) x'(n - i), \quad i = 0, 1, \ldots, N - 1 \tag{19}$$

6. Repeat the procedure for the next iteration. Note that the total number of memory locations required for this algorithm is 2(N + M) plus some parameters.

Assembly language implementations of the FXLMS algorithm are given in Appendix C, *TMS320C25 Ariel Board Implementation of ANC Algorithms*, and Appendix D, *General Configurable Software for ANC Evaluation*.

Leaky FXLMS Algorithm

When an adaptive filter is implemented on a signal processor with fixed word lengths, roundoff noise is fed back to the filter weights and accumulates continuously. This can cause the coefficients to grow larger than the dynamic range of the processor (overflow), which results in inaccurate filter performance. One solution to the problem is based on adding a small forcing function, which tends to bias each filter weight toward zero. According to equation (9), this leaky FXLMS algorithm can be expressed as [5]:

$$\underline{w}(n + 1) = v\underline{w}(n) - \mu e(n)\underline{x}'(n) \tag{20}$$

where v (the leakage factor) is slightly less than 1 and $\underline{x}'(n)$ is defined in equation (11).

The leaky FXLMS algorithm can not only reduce numerical error in the finite precision implementation but also limit the output power of the loudspeaker to avoid nonlinear distortion, which is caused by overdriving the canceling speaker.

Acoustic Feedback Effects and Solutions (FBFXLMS Algorithm)

Referring again to the simple system shown in Figure 2 on page 10, the antinoise output to the loudspeaker not only cancels acoustic noise downstream, but unfortunately, it also radiates upstream to the input microphone, resulting in a contaminated reference input x(n). This acoustic feedback introduces a feedback loop or poles in the response of the model and results in potential instability in the control system.

This problem has been intensively studied in active noise and vibration control literature. Solutions such as the following have been proposed:

1. Using directional microphones and speakers [14]. (This has a limitation in that directional arrays are usually highly dependent on the spacing of the array elements and are directional over only a relatively narrow frequency range.)
2. Using fixed compensating signals (generated from the compensating filter whose coefficients are determined off-line by using a training signal) to cancel the effects of the acoustic feedback
3. Using a second off-line adaptive filter in parallel with the feedback path [15]
4. Using an adaptive IIR filter [16]

This report examines methods 2 and 4. An adaptive feedforward controller with feedback compensation is shown in Figure 11. The filter D(z) is an estimate of the feedback path F(z) from the adaptive filter output

y(n) to the output of the reference input microphone u(n). Filter D(z) removes the acoustic feedback from the reference sensor input; the filter C(z) compensates the secondary-path transfer function H(z) in the FXLMS algorithm. Removal of the acoustic feedback from the reference input adds a considerable margin of stability to the system if the model D(z) is accurate. The models C(z) and D(z) can be estimated simultaneously by an off-line modeling technique using an internally generated white noise.

The expressions for the antinoise y(n), filtered-X signal x′(n), and the adaptation equation for the FBFXLMS algorithm are the same as that for the FXLMS ANC system, except that x(n) in FBFXLMS algorithm is a feedback-free signal that can be expressed as:

$$x(n) = u(n) - \sum_{i=1}^{L} d_i y(n - i) \tag{21}$$

where u(n) is the signal from input microphone, d_i is the ith coefficient of D(z), and L is the order of D(z).

In the case of a perfect model of the feedback path (that is, D(z) = F(z)), the acoustic feedback is completely canceled by D(z). The adaptive filter converges to the transfer function given in equation (5), the ideal case without acoustic feedback. The function of D(z) is similar to the acoustic echo cancellation that is used in teleconferencing applications [16].

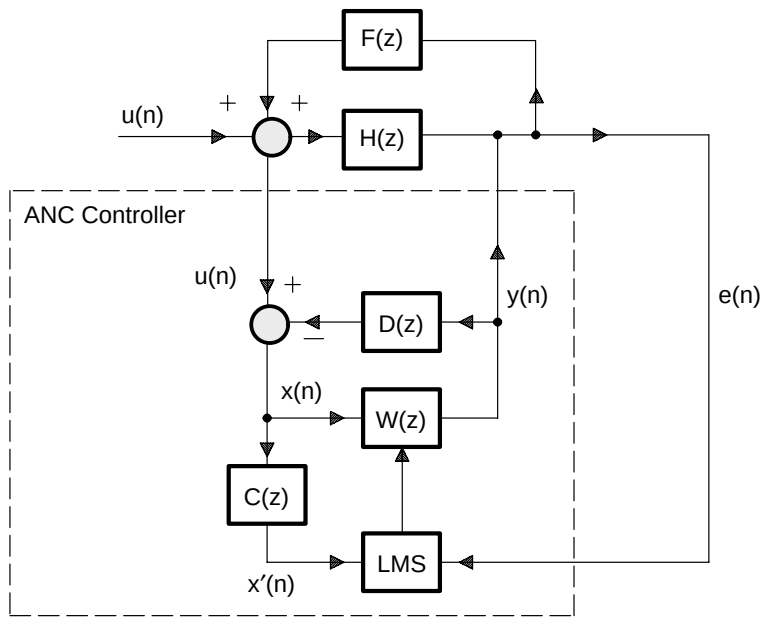


Figure 11. ANC System With Acoustic Feedback Cancellation

The system performs the off-line modeling first to estimate the secondary-path transfer function H(z) from the canceling speaker to the error microphone and the feedback path transfer function F(z) from the canceling speaker to the input microphone. The off-line modeling algorithm is illustrated in Figure 12 and the procedure is summarized following the figure.

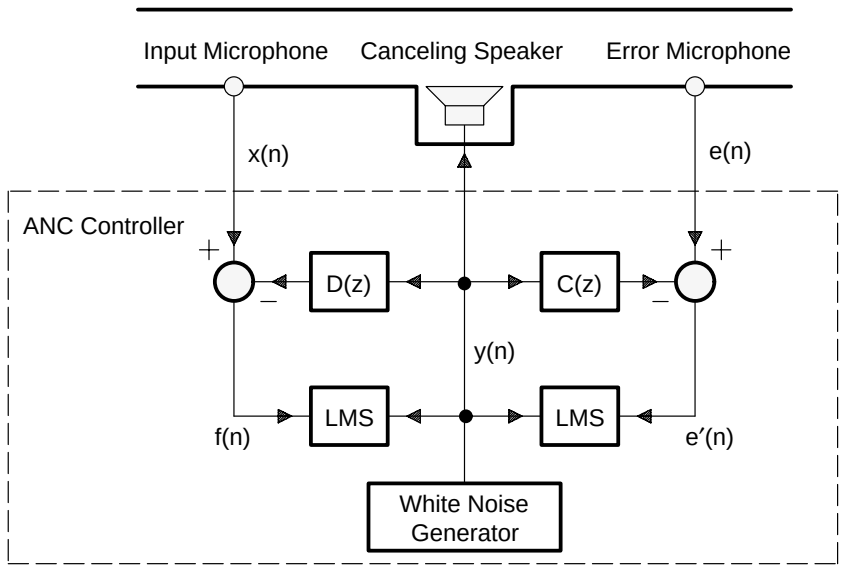


Figure 12. Off-Line Modeling of Secondary and Feedback Paths

1. a. Generate a white noise sample y(n).
   b. Output this excitation signal y(n) to drive the canceling loudspeaker.
   c. Send y(n) to the adaptive filters C(z) and D(z).
   d. Send y(n) to the LMS algorithm for updating C(z) and D(z).
2. Input x(n) from the input microphone and e(n) from the error microphone.
3. Compute e′(n) and f(n):

$$e'(n) = e(n) - \sum_{i=0}^{M-1} c_i(n)\,y(n-i) \tag{22}$$

and

$$f(n) = x(n) - \sum_{j=0}^{L-1} d_j(n)\,y(n-j) \tag{23}$$

4. Update the coefficients of the adaptive filters C(z) and D(z) using the LMS algorithm:

$$c_i(n + 1) = c_i(n) + \mu e'(n) y(n - i), \quad i = 0, 1, \ldots, M - 1 \tag{24}$$

and

$$d_j(n + 1) = d_j(n) + \mu f(n) y(n - j), \quad j = 0, 1, \ldots, L - 1 \tag{25}$$

5. Repeat the off-line modeling for about 10 seconds. Save the coefficients of adaptive filters C(z) and D(z) and use them in the following active noise cancellation mode.

After the off-line modeling, the ANC system is operated in active noise cancellation mode. The algorithm (illustrated in Figure 11) is summarized as follows:

1. Input u(n) and e(n) from the input ports.
2. Compute the feedback-free reference input x(n):

$$x(n) = u(n) - \sum_{j=0}^{L-1} d_i y(n - j) \tag{26}$$

3. Compute the antinoise y(n):

$$y(n) = \sum_{i=1}^{N-1} w_i(n) x(n - i) \tag{27}$$

where $w_i(n)$ is the ith coefficient of the adaptive filter W(z) at time n and N is the order of filter W(z).

4. Output the antinoise y(n) to the output port to drive the canceling loudspeaker.
5. Compute the filtered-X version of $x'(n)$:

$$x'(n) = \sum_{i=0}^{M} c_i x(n - i) \tag{28}$$

6. Update the coefficients of adaptive filter W(z) using the following FXLMS algorithm:

$$w_i(n + 1) = w_i(n) + \mu e(n) x'(n - i), \quad i = 0, 1, \ldots, N - 1 \tag{29}$$

7. Repeat the algorithm for the next iteration. Note that the total number of memory locations required in this algorithm is 2(N + M + L) plus some parameters.

Assembly language implementations of this algorithm are given in Appendix C, *TMS320C25 Ariel Board Implementation of ANC Algorithms*, and Appendix D, *General Configurable Software for ANC Evaluation*.

Filtered-U Recursive LMS (RLMS) Algorithm

The adaptive infinite impulse response (IIR) filter (method 4 on page 20) was proposed by Eriksson [17] for use in active noise control. This approach considers the acoustic feedback as a part of the whole acoustic plant, and the poles introduced by the acoustic feedback are removed by the poles of the adaptive IIR filter. This control system dynamically tracks changes in the secondary and feedback paths during cancellation operations. Also, as shown in equation (5), the IIR structure has the ability to model transfer functions directly with poles and zeros. Although there are various adaptive IIR algorithms that can be used, the recursive LMS (RLMS) algorithm developed by Feintuch [18] is selected here for reasons of computational simplicity.

The RLMS algorithm must also be modified to compensate for the transfer function of the secondary and feedback paths. A block diagram of an ANC system using an adaptive IIR filter is shown in Figure 13, where y(n) is the output signal of IIR filter computed by:

$$y(n) = \underline{a}^T(n)\underline{x}(n) + \underline{b}^T(n)\underline{y}(n-1) = \sum_{i=0}^{N-1} a_i(n)x(n-i) + \sum_{j=1}^{M} b_j(n)y(n-j) \tag{30}$$

where:

$\underline{a}(n) = [a_0(n)\ a_1(n) \ldots a_{N-1}(n)]^T$ is the weight vector of A(z) at time n
$\underline{b}(n) = [b_1(n)\ b_2(n) \ldots b_M(n)]^T$ is the weight vector of B(z) at time n
$\underline{y}(n-1) = [y(n-1)\ y(n-2) \ldots y(n-M)]^T$ is the signal vector containing output feedback with one delay
N = order of A(z)
M = order of B(z)

The filtered-U RLMS algorithm [12] can be expressed by two vector equations for adaptive filters A(z) and B(z) as follows:

$$\underline{a}(n+1) = \underline{a}(n) - \mu e(n)\ \underline{x}'(n) \tag{31}$$

and

$$\underline{b}(n+1) = \underline{b}(n) - \mu e(n)\ \underline{y}'(n-1) \tag{32}$$

where:

$$\underline{y}'(n-1) = [y'(n-1)\ y'(n-2)\ \ldots\ y'(n-M)]^T \tag{33}$$

and

$$y'(n) = \sum_{j=1}^{M} c_j y(n-j) \tag{34}$$

is the filtered y(n) from C(z), and $\underline{x}'(n)$ is defined in equation (11).

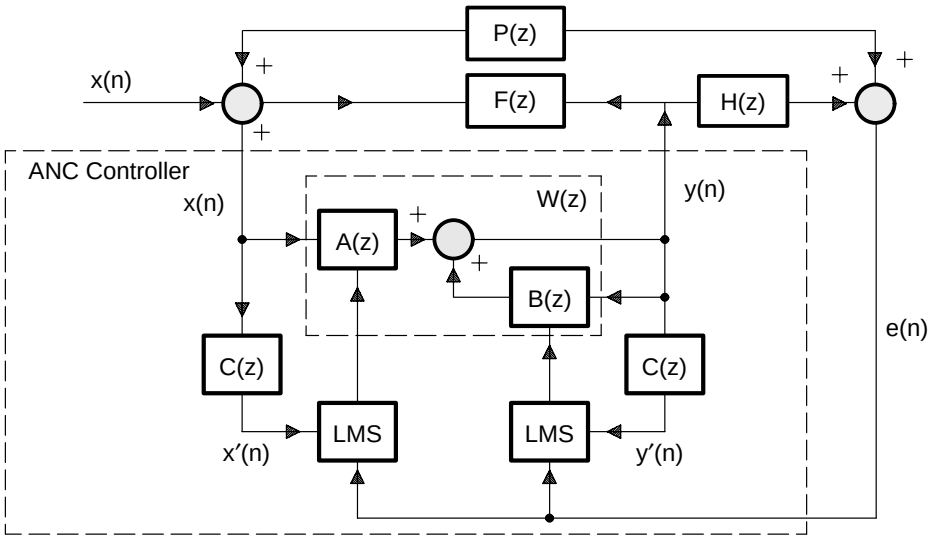


Figure 13. ANC System With the Filtered-U RLMS Algorithm

After both A(z) and B(z) converge, the measured residual error signal e(n) is equal to zero. Now:

$$W(z) = \frac{A(z)}{1 - B(z)} = \frac{-P(z)}{H(z) - P(z)\ F(z)} \tag{35}$$

Given the complexities and pole-zero structure of P(z), H(z), and F(z), the convergence of A(z) and B(z) cannot be generalized. The optimum solutions $A^*(z)$ and $B^*(z)$ are not unique; however, the algorithm will converge to a solution that minimizes the residual error signal e(n). Based on equation (35), one possible set of solutions is:

$$A^*(z) = \frac{-P(z)}{H(z)} \tag{36}$$

and

$$B^*(z) = \frac{P(z)\ F(z)}{H(z)} \tag{37}$$

Therefore, it is reasonable to use a higher order for B(z) than for A(z).

The system performs the off-line modeling to estimate the secondary-path transfer function using the algorithm summarized in the section on the FXLMS algorithm. After the off-line modeling, the ANC system is operated in noise cancellation mode. The detailed algorithm, shown in Figure 13, is summarized as follows:

1. Input the reference signal x(n) and the error signal e(n) from the input ports.
2. Compute the antinoise y(n):

$$y(n) = \sum_{i=0}^{N-1} a_i(n)x(n-i) + \sum_{j=1}^{J} b_j(n)y(n-j) \tag{38}$$

   where N is the order of the filter A(z) and J is the order of the filter B(z).

3. Output the antinoise y(n) to the output port to drive the canceling speaker.
4. Perform the filtered-U operation:

$$x'(n) = \sum_{i=0}^{M-1} c_i x(n-i) \tag{39}$$

   and

$$y'(n) = \sum_{i=0}^{M-1} c_i y(n-i-1) \tag{40}$$

   where M is the order of the filter C(z).

5. Update the coefficients of the adaptive filters A(z) and B(z) using the filtered-U RLMS algorithm:

$$a_i(n+1) = a_i(n) + \mu_a e(n)x'(n-i), \quad i = 0, 1, \ldots, N-1 \tag{41}$$

   and

$$b_j(n+1) = b_j(n) - \mu_b e(n)y'(n-j), \quad j = 1, 2, \ldots, J \tag{42}$$

6. Repeat the algorithm for the next iteration.

Assembly language implementations of the filtered-U RLMS algorithm are given in Appendix C, *TMS320C25 Ariel Board Implementation of ANC Algorithms*, and Appendix D, *General Configurable Software for ANC Evaluation*.

Algorithms for Narrowband Feedforward ANC Systems

In many practical applications, the acoustic measurement of the reference signal is not feasible, such as when the primary noise is produced by rotating machines and is periodic as illustrated in Figure 14. In these cases, an alternative method can be used. This method estimates the acoustic signal using an indirect measurement from a nonacoustic sensor in place of the reference microphone.

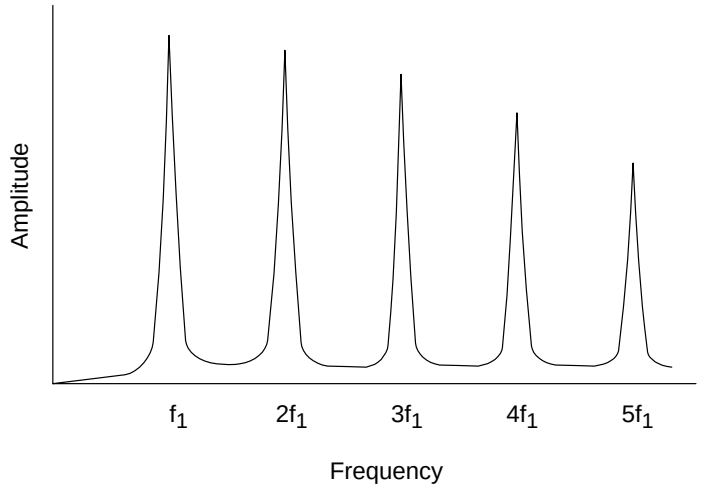


Figure 14. Spectrum of Original Noise Signal

The synthesis of a reference signal is triggered by the synchronized input pulse from the noise source, such as a tachometer signal synthesized from an automotive engine. In general, there are two types of reference signals that are commonly used in the narrowband ANC systems:

- Impulse train with a period equal to the inverse of the fundamental frequency of the periodic noise
- Sine waves that have the same frequencies as the corresponding harmonics to be canceled

The first technique is called the waveform synthesis method (also called the Essex algorithm), which was proposed by Chaplin [19]. This technique can be analyzed as the adaptive transversal filter excited by the impulse train and updated by the FXLMS algorithm [20]. The second technique is called the adaptive notch filter for interference cancellation. The single-frequency notch filter uses two adaptive weights and a 90° phase shifter [21] to cancel an undesired sinusoidal interference in the primary input. The application of this technique to the active periodic noise control was proposed by Ziegler [22].

Waveform Synthesis Method of Synthesizing the Reference Signal (Essex Algorithm)

A waveform synthesizer produces a canceling signal y(n) to drive the canceling speaker. The generated waveform is output sequentially to the canceling speaker and is synchronized with the pulse from the nonacoustic sensor. A microphone in the area of the quiet zone senses the residual sound and feeds this back to the adaptation unit that is used to modify the waveform synthesizer. Cancellation occurs only at the frequencies of the harmonics; the frequency bands between the harmonics remain unaffected. This enables, for example, normal speech to be heard clearly in an otherwise impossibly noisy room, or enables the radio to be heard through a headset while the wearer is riding a motorcycle. Another reason for removing only some parts of the noise spectrum is that in a car the driver needs some audible indication of engine speed to be able to control the vehicle safely.

The preferred synchronization signal is derived from a toothed wheel driven by the engine, generating an impulse train of perhaps a hundred equally spaced pulses in each cycle of the source. The waveform

synthesizer stores canceling waveform samples $\{w_j(n), j = 0, 1, \ldots, N - 1\}$, where N is the number of samples for one cycle of the waveform. The synchronization signal is used to derive a memory address pointer, which can be a software-incremented counter controlled by interrupts generated from the synchronization signal. These samples represent the required waveform to be generated and are presented sequentially to a digital-to-analog converter to produce the actual antinoise waveform for the canceling speaker. That is:

$$y(n) = w_j(n), \quad 0 \leq j \leq N - 1 \tag{43}$$

represents the jth element of $\{w_j(n)\}$, where j is a pointer. Some advanced digital signal processors such as TMS320C50, TMS320C30, and TMS320C40 have circular pointers for this type of addressing.

The residual noise picked up by the error microphone is sampled in synchronization with the reference and canceling signals. The sampled error signal e(n) is then used by the adaptation unit to adjust the values of the canceling waveform $\{w_j(n)\}$ by the following algorithm:

$$w_j(n + 1) = w_j(n) - \mu \ \mathrm{sign}[e(n)] \tag{44}$$

This algorithm is the sign-error LMS algorithm (since the reference input $x(n) = 1$), which is derived based on the criterion to minimize the absolute value of the instantaneous error signal. In order to provide faster convergence, the traditional LMS algorithm can be used:

$$w_j(n + 1) = w_j(n) - \mu e(n) \tag{45}$$

where μ is less than unity.

In practice, the current error signal e(n) does not correspond to the jth element of the canceling waveform $w_j(n)$. For a practical system, there is a delay of several milliseconds between the time the signal $[y(n) = w_j(n)]$ is fed to the speaker and the time it is received at the error microphone. This delay can be accommodated by subtracting a time offset from the circular pointer j that is pointing to the waveform:

$$w_{j-\Delta}(n + 1) = w_{j-\Delta}(n) - \mu e(n) \tag{46}$$

where Δ is the time delay of data samples between the output of the signal from the waveform synthesizer and its reception at the residual error microphone; that is:

$$\Delta = \frac{\delta t}{T} \tag{47}$$

where δt is the time delay (which is constant for a given speaker-microphone arrangement) and T is the sampling period. Because the sampling rate is synchronized with the noise source, this offset number is updated in correspondence with the changing sampling rate.

Greater degrees of cancellation can be achieved in the presence of unsynchronized background noise if the residual waveforms are averaged over a number of cycles. The performance improves by 3–5 dB per

frequency component. However, the necessary number of averages strongly depends on the characteristics of the noise. Thus, there is a tradeoff between the degree of cancellation and the adaptation time required for canceling stationary waveforms.

The complexity of the broadband ANC system discussed previously and the narrowband ANC system using the waveform synthesis method is summarized in Table 1, where N is the order of the filter and complexity is given in terms of the number of coefficients that must be updated per sample period.

Table 1. Complexity of Broadband ANC and Narrowband ANC

OPERATION	BROADBAND ANC	NARROWBAND ANC
Multiplication	2N + 1	1
Addition	2N – 1	1

The concept of the waveform synthesis method can be analyzed as if the adaptive FIR filter were excited by a periodic impulse train of period L [20]. To analyze the canceler output e(n) for a given input d(n), consider the transfer function G(z) between the initial input D(z) and the error output E(z). It is shown that [20]:

$$G(z) = \frac{E(z)}{D(z)} = \frac{1 - z^{-N}}{1 - (1 - \mu)z^{-N}} \tag{48}$$

The properties of the transfer function G(z), given in equation (48), are those of a comb filter with notches at each harmonic frequency of the interference. Therefore, the tonal components of the periodic noise at the fundamental and the harmonic frequencies can be attenuated by this multiple notch filter.

Equation (48) also shows the location of the poles and zeros of G(z). For a generic fundamental frequency $\omega_0 = 2\pi / L$, the poles and the zeros are aligned exactly at the same angles for any given value of step size μ. The zeros are at

$$z_k = e^{\pm jk\omega_0} \tag{49}$$

and the poles are at

$$P_k = (1 - \mu)e^{\pm jk\omega_0} \tag{50}$$

where $0 \le k \le N - 1$ is a frequency index. The pole-zero placement in the z plane is shown in Figure 15.

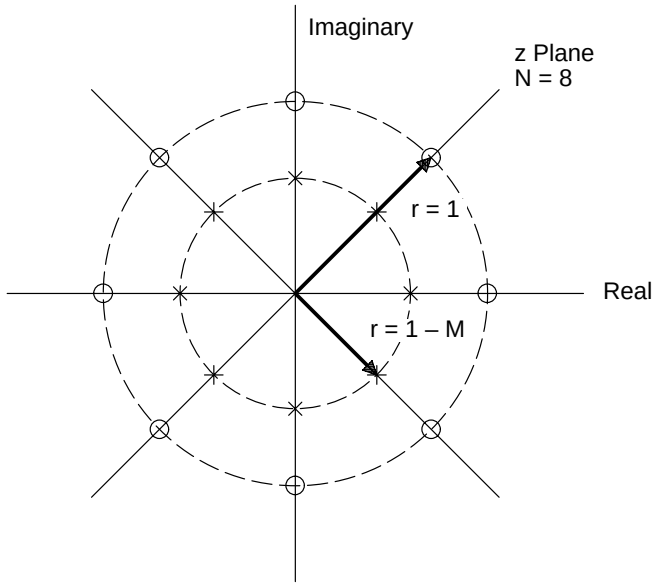


Figure 15. Pole-Zero Placement in z Plane

The zeros must have constant amplitude ($|z| = 1$) and be equally spaced ($2\pi / N$) on the unit circle of the z-plane to create nulls in the frequency response at frequencies $k\omega_0$. The poles have the same angle (frequency) as the zeros but are equally spaced on the circle at distance $(1 - \mu)$ from the origin. The effect of the poles is to introduce a resonance in the vicinity for the null, reducing the bandwidth of the notch. If $\mu << 1$ is used, the 3-dB bandwidth of each notch can be shown to be:

$$BW \approx \frac{\mu}{\pi T} \text{ (Hz)} \tag{51}$$

Therefore, the smaller the step size μ, the closer the poles are to the zeros and the narrower the bandwidths of the notches that can be achieved. This effect of a pole on notch bandwidth is shown in Figure 16.

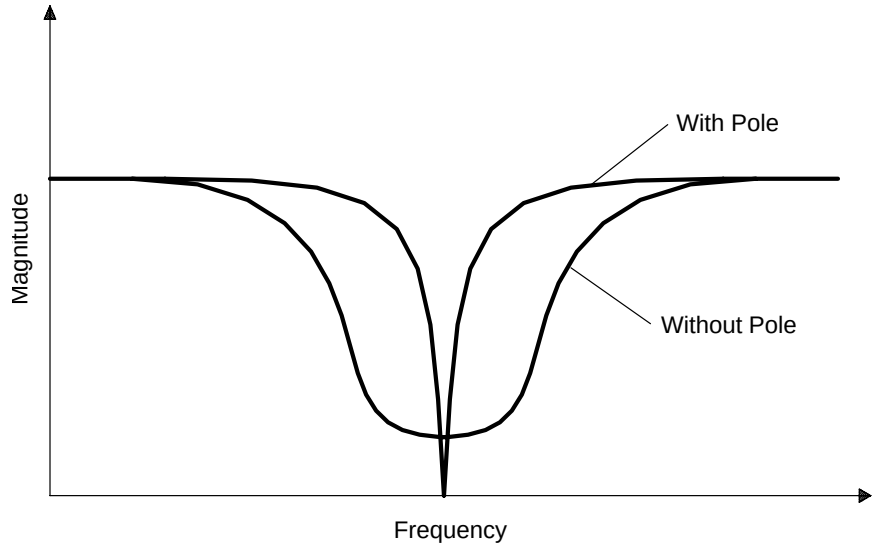


Figure 16. Effect of Pole on Notch Bandwidth

Adaptive Notch Filters

The second type of reference signal used in the narrowband ANC system is a sine wave with the same frequency as the narrowband noise to be canceled. When a sine wave is employed as the reference input, the LMS algorithm becomes an adaptive notch filter to remove the primary spectral components within a narrow band centered about the reference frequency. A very narrow notch is usually desired to filter out the interference without distorting the signal and can be realized by an adaptive noise canceler. The advantages of the adaptive notch filter are that it offers easy control of bandwidth, an infinite null, and the capability of adaptively tracking the exact frequency of the interference. This is especially true when the frequency of the interfering sinusoid changes slowly.

The application of the adaptive notch filter to active periodic noise control was developed by Ziegler [22]. A block diagram of this narrowband ANC system with two adaptive weights is shown in Figure 17. The timing signal sensor, such as an engine tachometer, is used to determine the fundamental frequency at which the repetitive noise is being generated. For example, an electric motor running at 1800 RPM completes 30 revolutions per second with a fundamental frequency of 30 Hz. A four-cylinder engine running at 1800 RPM also completes 30 revolutions per second but with only 15 complete firing cycles per second, and thus has a fundamental frequency 15 Hz.

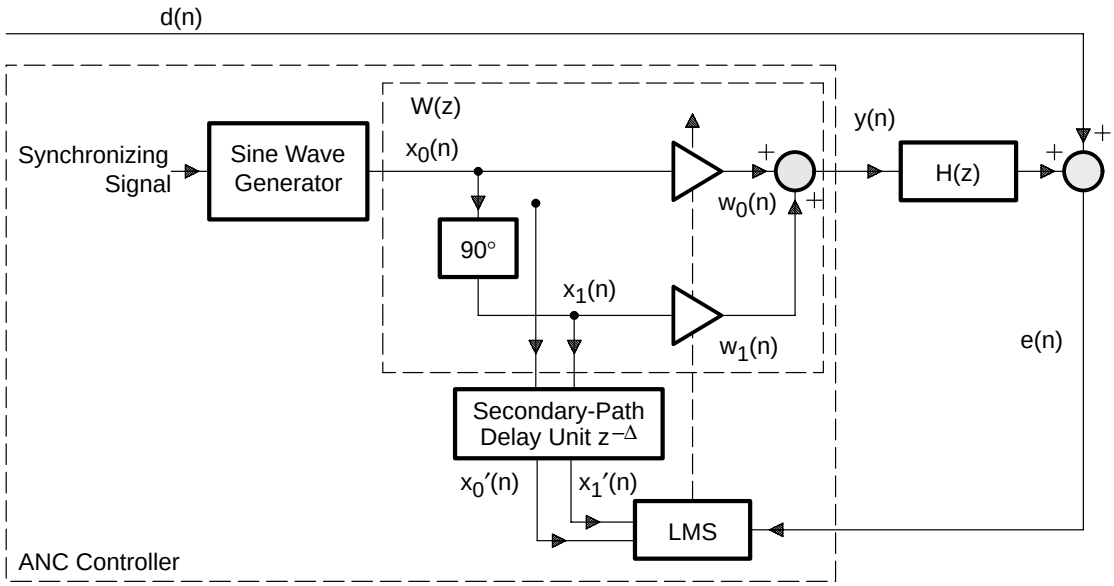


Figure 17. Single-Tone ANC System With Adaptive Notch Filter

The single-frequency active noise controller shown in Figure 17 can be configured in parallel or cascade structures [23] to cancel the narrowband noise at the fundamental frequency and its harmonics. A sine wave generator provides a sinusoidal reference signal at the desired frequency. Employing a Hilbert transform [24] as the 90° phase shifter, the sine wave is split into two orthogonal components, $x_0(n)$ and $x_1(n)$, which can be used as reference inputs for the adaptive filter. These two signals are separately weighted and then summed to produce the canceling signal y(n):

$$y(n) = w_0(n)\ x_0(n) + w_1(n)\ x_1(n) \tag{52}$$

where

$$x_0(n) = A\cos(k\omega_0 n) \tag{53}$$

and

$$x_1(n) = A\ \sin(k\omega_0 n) \tag{54}$$

where ω_0 is the fundamental frequency, k is the harmonic index, A is the amplitude of the reference signal, and n is the time index. The sine-wave generator can be implemented by a ROM table look-up technique or by a digital resonator [24]. Algorithms of a sine-wave generator using both the table look-up and the digital oscillator are given in Appendix B, *Digital Sine-Wave Generator.*

The magnitude and the phase of this reference signal are adjusted in the controller, which feeds one or more loudspeakers serving as the control source to cancel the corresponding noise components. The LMS algorithm updates the filter weights to minimize the residual error e(n):

$$w_0(n + 1) = w_0(n) - \mu e(n)\ x_0(n - \Delta_k) \tag{55}$$

and

$$w_1(n + 1) = w_1(n) - \mu e(n)\ x_1(n - \Delta_k) \tag{56}$$

where Δ_k is used to compensate for the effects of the secondary path at harmonic k. This delay represents the delay introduced between the adaptive filter output and the residual error input.

When the system time delay is fixed, the values can be estimated by an off-line secondary-path modeling technique (described previously; see page 18) and then built into the controller. In general, the values of the delay depend on the frequency. These delays can be determined by converting the impulse response of C(z) into the frequency domain by the discrete Fourier transform and then by calculating the delays from the phase values. That is:

$$t_f = \frac{-\Phi_f}{2\pi f} \tag{57}$$

where t_f is the time delay at frequency f in seconds, Φ_f is the phase at frequency f in radians, and f is the frequency in Hz. The values of Δ_k in equations (55) and (56) are then determined by:

$$\Delta_k = t_f\ f_s \tag{58}$$

where f_s is the sampling rate.

As mentioned previously, the secondary-path delay unit $z^{-\Delta}$ in Figure 17 can be replaced by the estimate of the secondary path. The adaptive notch filter algorithm using the FXLMS algorithm can be expressed as:

$$w_i\ (n + 1) = w_i\ (n) - \mu e(n) x_i^{'}(n) \tag{59}$$

for i = 0 or 1 and where $x_i^{'}$ (n) is the filtered version of x_i (n) by the secondary-path estimation C(z).

Structure for Multiple Frequency Cancellation

In practical applications, the periodic noise usually contains tones at the fundamental frequency and several harmonic frequencies. This type of noise can be attenuated by a filter with multiple notches. In general, realization of multiple notches requires a filter with higher order, which also can be realized by a parallel or cascade connection of multiple second-order sections. A method for eliminating multiple sinusoidals or other periodic interference was proposed by Glover [25]. The application of this technique to active periodic noise control is to generate the reference input as a sum of M sinusoids. That is:

$$x(n) = \sum_{m=1}^{M} A_m \cos(\omega_m n) \tag{60}$$

where A_m and ω_m are the amplitude and the frequency of the mth sinusoid, respectively.

When a sum of sinusoids is applied to an adaptive filter, the filter converges to a time-varying, tunable notch filter with a notch located at each of the reference frequencies. As long as a reference is available that includes every sinusoidal interference, the narrowband ANC system creates a notch over each sinusoid and follows it if it changes in frequency. This adaptive notch filter provides a simple method for the tracking and elimination of sinusoidal interferences. The application of Glover's method for actively attenuating engine-generated noise was patented by Pfaff [26]. The reference signal representing the selected multiple harmonic noise components is generated from a predetermined table of values.

A single-frequency sinusoid can be canceled by the simple 2-weight adaptive filter. For the case where the undesired primary noise contains M sinusoids, M 2-weight adaptive filters can be connected in parallel to attenuate these narrowband components. A set of closely spaced reference sinusoids is synthesized from the information provided by the synchronization signal. A specific sinusoid is used as the reference input for the corresponding channel of the 2-weight adaptive filter $W_m(z)$, which is connected in parallel with the other filters, as shown in Figure 18.

The structure of each individual channel is shown in Figure 17. The overall transfer function of this parallel configuration is:

$$W(z) = \sum_{m=1}^{M} W_m(z) \tag{61}$$

where $m = 1, 2, \ldots$ and M is the channel index. The canceling signal is a sum of M adaptive filter outputs. That is:

$$y(n) = \sum_{m=1}^{M} w_m(n) \tag{62}$$

Each reference input is filtered by the secondary-path estimate C(z) as:

$$x_m(n) = \sum_{i=0}^{L-1} c_i x_m(n-1), \quad m = 1, 2, \ldots, M \tag{63}$$

Because only one error microphone is used, there is only one error signal e(n) used to update M adaptive filters based on the FXLMS algorithm.

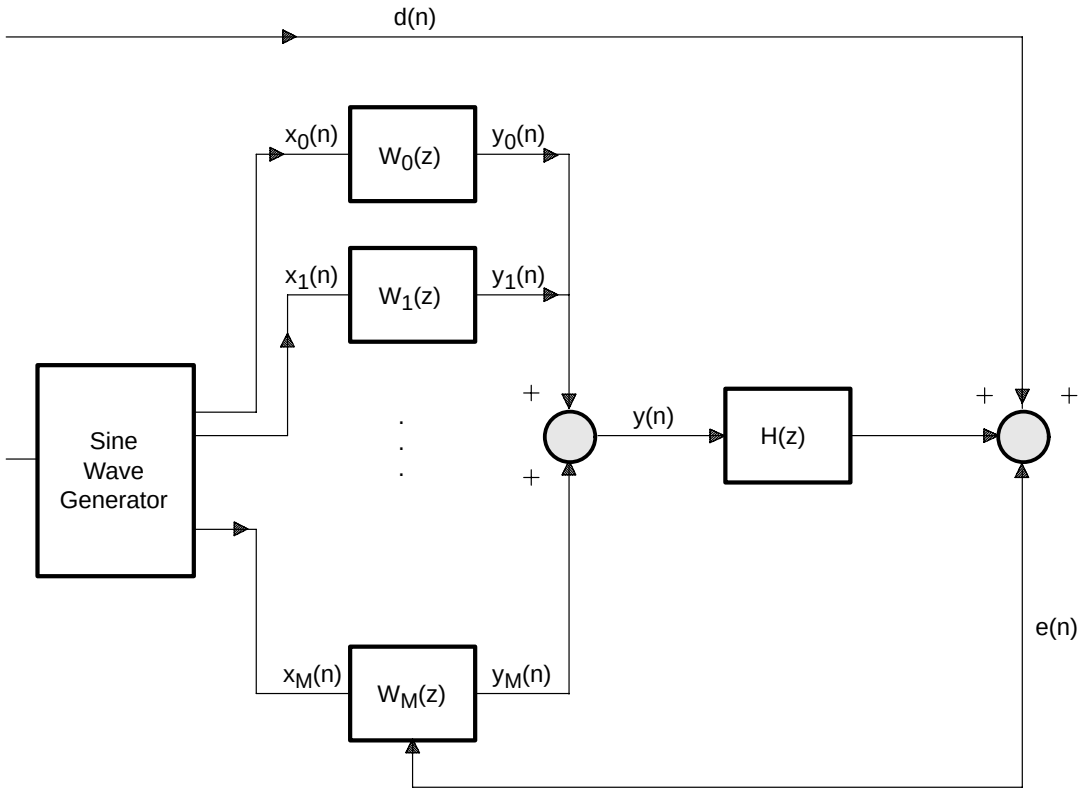


Figure 18. Multiple 2-Weight Adaptive Filters in Parallel

Algorithms for Feedback ANC Systems

The principle of feedback ANC for a single-channel case, which can be formulated as an adaptive predictor, is shown in Figure 19. Because this system requires only one error microphone, it avoids the acoustic feedback problem inherent in the 2-microphone feedforward systems that were discussed previously. Feedback ANC schemes depend on the signal having a periodic characteristic. Several nonadaptive feedback ANC systems have been described in the literature in recent years, as reviewed in Nelson's book [1].

Burgess [3] suggests the use of this configuration with the FXLMS algorithm to avoid the use of the input microphone. The basic idea of this algorithm is to estimate the primary noise d(n) and to use this as the reference input for the adaptive filter. As shown in Figure 19 and using the FXLMS algorithm, the primary noise is estimated as:

$$x(n) = e(n) - \sum_{i=0}^{M-1} c_i y(n-i) \tag{64}$$

where c_i ($i = 0, 1, \ldots M-1$) is the coefficient of the secondary-path estimation filter C(z) and M is the order of the filter C(z).

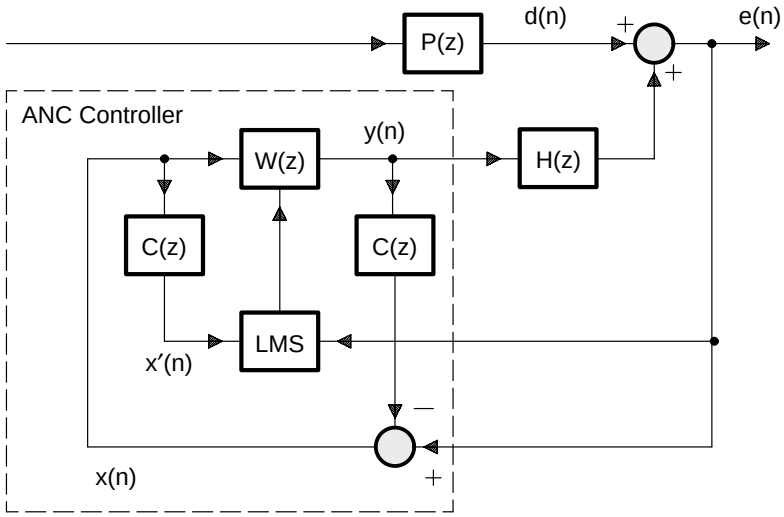


Figure 19. Block Diagram of the Feedback ANC System

From Figure 19:

$$D(z) = E(z) - H(z)Y(z) \tag{65}$$

where both E(z) and Y(z) are available. If the transfer function H(z) of the secondary path is modeled by C(z):

$$D(z) \approx X(z) = E(z) - C(z)Y(z) \tag{66}$$

The error signal can be shown as:

$$E(z) = D(z) - W(z)H(z)X(z) \tag{67}$$

The error signal for this feedback ANC system is 0 when:

$$W(z)\,H(z)X(z) = D(z) \tag{68}$$

which is possible if the primary noise D(z) is periodic and the transfer function W(z)H(z) is equal to a delay equivalent to a multiple of the signal period.

Off-line modeling is conducted first to estimate the secondary-path transfer function H(z) using the FXLMS algorithm. The noise canceling mode begins after the training. The feedback ANC algorithm illustrated in Figure 19 is summarized as follows:

1. Input the error signal e(n) from the error microphone.
2. Compute (estimate) the reference input signal x(n):

$$x(n) = e(n) - \sum_{i=0}^{M-1} c_i y(n - i) \tag{69}$$

3. Compute the antinoise y(n):

$$y(n) = \sum_{i=1}^{N-1} w_i(n) x(n - i) \tag{70}$$

   where $w_i(n)$ is the ith coefficient of the adaptive filter W(z) at time n and N is the order of filter W(z).

4. Output the antinoise y(n) to the output port to drive the canceling loudspeaker.
5. Compute the filtered-X version of $x'(n)$:

$$x'(n) = \sum_{i=0}^{M} c_i x(n - i) \tag{71}$$

6. Update the coefficients of adaptive filter W(z) using the FXLMS algorithm:

$$w_i(n + 1) = w_i(n) - \mu e(n) x'(n - i), \quad i = 0, 1, \ldots, N - 1 \tag{72}$$

7. Repeat the procedure for the next iteration.

Assembly language implementations of the feedback ANC algorithm are given in Appendix C, *TMS320C25 Ariel Board Implementation of ANC Algorithms*, and Appendix D, *General Configurable Software for ANC Evaluation*.

DESIGN OF ANC SYSTEMS

System Considerations

While numerous DSP devices with varying degrees of signal processing capability are becoming available, a particularly suitable choice for ANC is the TMS320C25 [42]. It combines the power of high speed, flexibility, low cost, and an architecture optimized for adaptive signal processing. The TMS320C25 can execute an instruction in as little as 80 ns, and the processor's architecture makes it possible to execute more than one operation per instruction cycle. For example, in one cycle the processor can generate an instruction address and fetch that instruction, decode the instruction, perform one or two data moves (if the second data is from program memory), update one address pointer, and perform one or two computations (multiplication and accumulation). A broad base of software support exists, and technical articles indicating the potential of the TMS320C25 have been published. The implementation of a variety of adaptive filter structures and adaptive algorithms can be found in the application report by Kuo and Chen [5].

Active noise control is a real-time application of adaptive filtering that requires extensive computations. The frequency bandwidth is 500 Hz–1000 Hz, which allows only 1–2 ms to perform all the calculations. The electronic hardware implementation in an ANC system requires tradeoffs that have a substantial impact on system performance. System hardware must allow software flexibility as well as fully automatic operation of the complete active noise control system [29]. Self-calibration and self-modeling are important system functions. The physical factors that limit the performance of ANC systems, such as spatial matching, coherence, filter length, stability, and causality [30], are discussed in this section as part of the implementation of active noise and vibration control systems using the TMS320C25 DSP.

In the broadband feedforward ANC system shown in Figure 2 on page 10, the input microphone should not be placed at the node (point of little or no sound magnitude) of any standing wave that may be present before or during cancellation [1]. The placement of the error microphone should also avoid nodal locations before cancellation. The microphones are selected to satisfy requirements of low cost, low impedance, large signal-to-noise ratio, nondirectivity, and high sensitivity. The loudspeaker is required to be able to generate a sound pressure level higher than the noise source pressure level, have good frequency response at low frequencies, have good humidity resistance, have a low cost, and be compact.

Sampling Rate and Filter Length

The task of the controller is to estimate precisely the delay and any amplitude changes that occur as the unwanted noise travels from the input microphone to the loudspeaker. This includes delays in the microphones, loudspeakers, and electronics. The controller must complete the entire signal processing task before the primary noise arrives at the loudspeaker. Real-time digital signal processing requires that the processing time t be less than the sampling period T. That is:

$$t < T = \frac{1}{f_s} \tag{73}$$

where f_s is the sampling rate, which must be held high enough to satisfy the Nyquist criterion. That is:

$$f_s \geq 2f_M \tag{74}$$

where f_M is the highest frequency of interest—approximately 500 Hz for most practical ANC applications. This yields a minimum sampling rate of 1 kHz and a maximum processing time of 1 ms.

The sampling rate can be expressed in terms of the physical distance and the ability of the system to resolve this distance at room temperature. The sampling resolution can be expressed as:

$$\Delta_s = \frac{c_0}{f_s} \tag{75}$$

where c_0 is the speed of sound in air, which is 343 meters per second at 75°F.

The modeling of the primary plant is done in the time domain using the FXLMS algorithm or the filtered-U RLMS algorithm. The number of direct weights (W(z) in the FXLMS algorithm and A(z) in the filtered-U RLMS algorithm) times the sampling resolution determines the model length in time or an equivalent distance. That is:

$$l = N\Delta_s = \frac{Nc_0}{f_s} \tag{76}$$

where l is the length of duct from the input microphone to the canceling loudspeaker that can be modeled by an adaptive filter. For example, $N = 64$ and a sampling rate of 2 kHz results in a 32-ms model, which corresponds to a duct length of 10.976 meters.

The length of the noise control filter depends upon the acoustics of the duct. The required length is reduced by the addition of passive damping material. The number of coefficients required also depends upon the sampling rate. This creates a conflict with the causality constraint, which is described later. To achieve broadband random noise cancellation, it is necessary that the filters be long enough to account for the physical distances within the plants. For periodic signals such as sine waves, this constraint no longer applies, because only adjustments to phase over one cycle of the sine wave are required. Another limitation imposed on the system is that the direct modeling filter must be sufficiently long to ensure adequate accuracy in the phase and amplitude response of the filter at the lowest frequency of interest.

Coherence Function

The structure shown in Figure 2 (see page 10) assumes that any noise that appears at the input microphone will appear at the loudspeaker after a delay. Unfortunately, both the input and the error microphones detect the primary noise plus the self-generated flow noise of the air passing over the surface of the microphone. Therefore, flow noise and turbulent pressure fluctuations at the microphones can limit cancellation effectiveness. This problem is rather significant for ducts of low sound pressure levels such as those in an air conditioner. A convenient measure of the amount of primary noise as compared with the flow noise is the coherence function [1] of the two microphone signals. The coherence, or similarity of phase relationship in the sound waves, between the sensors can be improved by reducing the flow velocity, using multiple distributed sensors, and by good fluid-mechanical design to minimize localized turbulent noise.

In heating, ventilating, and air conditioning (HVAC) systems, air flow velocities are around 13 meters per second. Therefore, flow microphones that reduce flow noise are required when active noise control is applied these systems [31]. Coherence can be improved dramatically by two methods:

- Using probe tubes that allow the propagating sound to reach the microphone while damping the turbulent pressure fluctuations (see Figure 20) [32]. Proper location of the probe tubes in the duct, away from the most turbulent part of the air stream, helps coherence. However, this has the disadvantage that the microphones and their supports generate turbulence and increase the flow noise of the microphone downstream.
- Nishimura shows [33] that placing the microphone in a small, outer turbulence tube connected with the duct through a small slit (see Figure 21) can significantly increase coherence. The placement of the microphone in the outer turbulence tube also has advantages in component protection and maintenance.

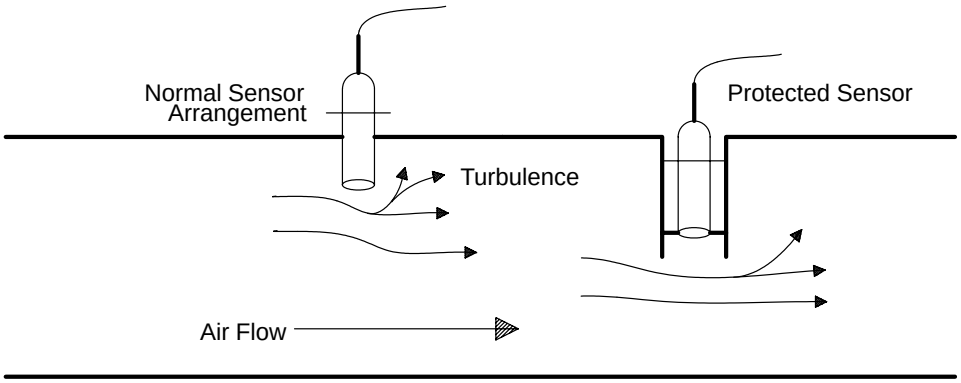


Figure 20. Probe Tube Used to Increase Coherence

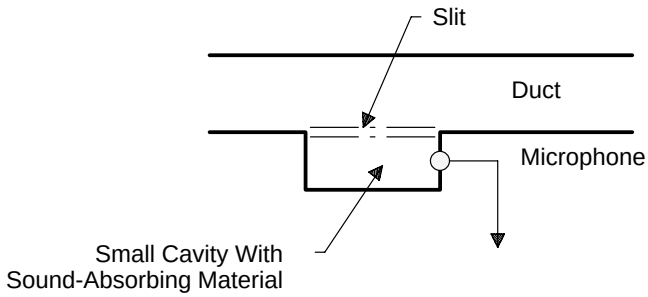


Figure 21. Microphone Mounting Method to Reduce Flow Turbulence

When the ANC system is applied to reduce the exhaust noise of an engine, the canceling speakers are located outside the exhaust duct but close to the outlet of a pipe to avoid exposure to high temperature gas. The resulting action is that the pipe outlet, a monopole, is converted to a dipole by the adjacent negative source and hence has a reduction in radiation efficiency at low frequencies [1]. Changing the location of the control sources from inside to outside the duct also significantly reduces the problem of acoustic feedback. In this system, the error microphone is placed outside and close to the end of the duct. This reduces the effect of flow noise, improving coherence. It also removes the possibility of the error microphone being at an acoustic node in the duct. Proper placement of the error sensor produces a global canceling effect at the end of the duct. The durability of speakers and microphones can be improved by planning for optimum arrangement and protection.

Causality

From Figure 2, the acoustic delay from the input microphone to the loudspeaker is given in seconds as:

$$\delta_A = \frac{L}{c_0} \tag{77}$$

where L is the distance from the input microphone to the canceling speaker. The electrical delay can be expressed as:

$$\delta_E = \delta_W + \delta_t \tag{78}$$

where δ_t, is the total delay in the antialiasing filter, the analog-to-digital converter (ADC), the digital-to-analog converter (DAC), the reconstruction filter, and the loudspeaker, plus the processing time (one sampling period, T). δ_W is the group delay of the digital filter W(z). The loudspeaker delay has a great influence on causality, especially at low frequency, and should be selected carefully.

To ensure that the adaptive filter has causal response, ensure that:

$$\delta_A > \delta_E \tag{79}$$

This condition is called causality and it sets the minimum length for a system that cancels random noise above a certain frequency. That is, the distance from the input microphone to the output speaker is:

$$L_{min} > c_0 \delta_E \tag{80}$$

The total delay is approximately [1]:

$$\delta_t \approx T\ (1 + \frac{3M}{8}) \tag{81}$$

where M is the number of poles in both the antialiasing and the reconstruction filters. For example, assuming that the sampling rate is 2 kHz and that each analog filter has 6 poles, M = 12 and δ_t = 2.75 ms.

The response of the controller is noncausal when the electrical delay is longer than the acoustic delay. To reduce the electrical delay, a higher sampling rate is required for a given length. However, this potentially reduces the model length if the maximum shown in equation (76) is required. To attenuate random noise in ducts, the standard ANC approach is to use a long duct. However, packaging constraints of commercial systems usually prevent this. Periodic noises (tones) are a special case where causality is not required and shorter systems are possible.

Typical HVAC noise spectra are broadband with prominent low frequencies (pink noise) and a few moderate-amplitude sinusoids [30]. This requires causality at low frequencies for good cancellation. The degree of antialiasing filtering required is determined by the high-frequency content of the noise, so it is desirable to include some passive damping material in the duct. This passive attenuation also helps to reduce the length of the filters [1].

Constraints and Solutions

The industrial applications of active noise control impose a different set of constraints from those of consumer applications. These different limitations include the power of the canceling loudspeaker due to high sound power levels inside the ducts and protection requirements for components due to the harsh environment. The noise is primarily narrowband or periodic and loud, produced by fans, vacuum pumps, compressors, or blowers. Gas flow rate is high, and there are sometimes suspended solids in the stream. Temperatures are often high and the gas stream is sometimes wet and corrosive.

These problems can be solved by using a range of high-power loudspeakers separated from the duct gas stream by a protective membrane that allows transmission of the sound energy into and out of the duct while maintaining a clean, dry environment that ensures long component life [34]. More than one loudspeaker (two or four connected to a single output from the controller) can be used, both to provide extra output power and for redundancy. Industrial systems are more often judged by performance on tones and are not limited by the length of systems. Protection of the loudspeakers and the microphones from a wet and/or dirty environment is essential for long term performance.

Active noise control is typically limited to low-frequency noise; therefore, when HVAC duct cross-sections are large, hybrid active-passive techniques (which use sound-absorptive lining inside the duct wall with the active components built into the absorptive duct section) are needed to attenuate noise over the full audible range. The electronics unit can be mounted either on the duct or on a nearby wall. The passive absorption also helps to reduce feedback from the canceling speaker to the input microphone.

To apply active noise control techniques to compressor noise in appliances such as refrigerators, the machine compartment structure in the appliance must be changed into a duct form [35, 36], as illustrated in Figure 22. With the noise source (compressor) is located in the duct, the low-frequency noise radiates like a plane wave. The machine chamber is sealed, excluding the opening for the heat radiation of the compressor. Sound radiation from the compressor can also be effectively reduced by controlling the shell vibration using piezoelectric actuators bonded to the surface of the compressor shell [37].

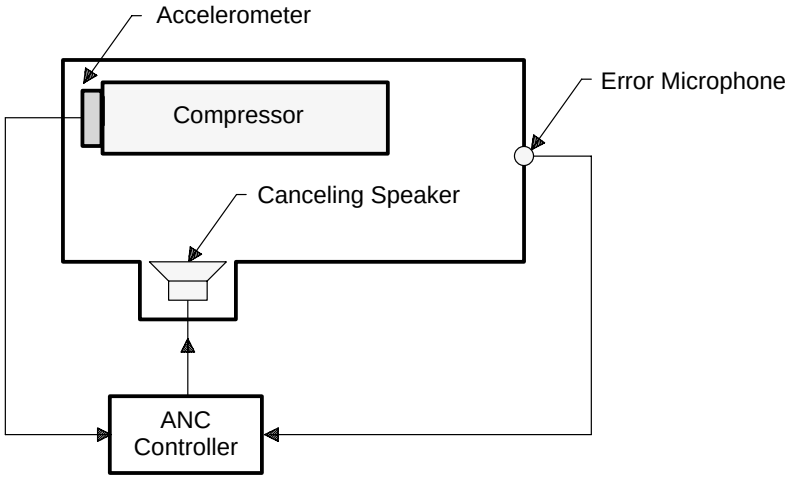


Figure 22. ANC System in Duct-Like Machine Chamber

Automatic Gain Controller

The block diagram of the TMS320C25-based hardware system is shown in Figure 23. The hardware is designed to accept two input signals, one from the input microphone and one from the error microphone. The output signal is converted to analog form to drive a canceling loudspeaker using a power amplifier. Since the DSP has a fixed-point data format, one analog automatic gain controller (AGC) must be used at each input to take advantage of the ADC's dynamic range and to avoid input saturation.

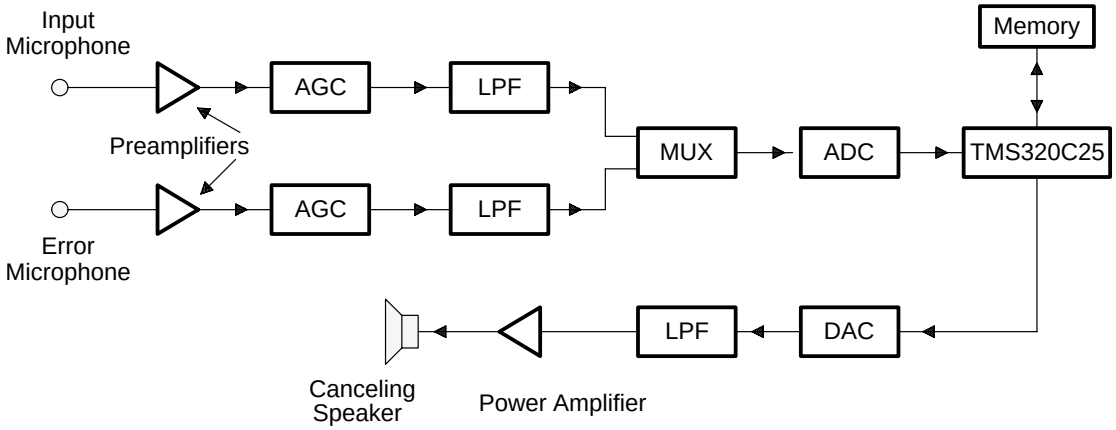


Figure 23. TMS320C25-Based ANC System Hardware

The input and error signals have slightly different requirements. When the noise cancellation mode starts, the error signal decreases substantially. In some systems, the input signal may increase slightly due to

acoustic feedback from the canceling speaker. Signal statistics also affect usable dynamic range; the maximum amplitude of a broadband signal that does not saturate a given system is less than the corresponding value for a sinusoid.

In general, an L-bit ADC typically has a dynamic range of 6L dB. If the input changes by more than that amount, a high-resolution ADC can be used, or an AGC can be used to keep the analog signal within the usable dynamic range of the existing ADC.

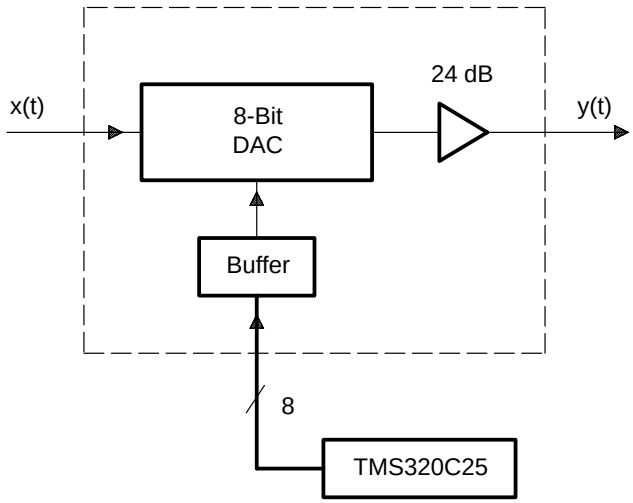


Figure 24. Block Diagram of an AGC

To account for the large dynamic range of sound pressure levels measured by microphones, the AGC can be implemented by using an 8-bit multiplying DAC. The analog output system has the same requirements as the analog input system, except that instead of programmable gain A, there is usually programmable attenuation. The block diagram of the AGC is shown in Figure 24, where an 8-bit DAC is used as an attenuator with a 48-dB dynamic range. The gain of the AGC is software-controlled by the TMS320C25, which writes an 8-bit value into the buffer of the 8-bit DAC. The algorithm of the AGC sets the gains on the input and the output signals. The implementation of the AGC maximizes the ADC signal-to-noise ratio and maintains the overall system dynamic range when used in different environments.

Antialiasing and Reconstruction Analog Filters

As shown in equation (74), to recover the original time-domain waveform from its sampled form, the original signal spectrum must be entirely constrained within a bandwidth of less than half the sampling rate. If the band limitation is not sufficient, the signal component over one half of the sampling frequency is folded into the signal band. This phenomenon is called aliasing, which cannot be isolated after sampling. Even if the input signal is naturally band limited, an antialiasing filter is still advisable to reduce out-of-band noise aliasing into the wanted frequency band.

Ideally, the antialiasing filter should have a flat amplitude and linear phase response over the bandwidth of the signal and infinite attenuation at half the sampling rate and beyond. In ANC systems, since the sampling rate is low (1 kHz to 2 kHz), a very-high-order antialiasing filter must be used. Unfortunately, these high-order filters have long group delays, as shown in equation (81). This can create a causality problem in the ANC system, particularly for broadband noise control in short ducts. The duct length from input microphone to canceling speaker must be increased to account for the extra delay.

If a low-order filter with a better phase response is used, such as a Butterworth filter, a lower group delay can be achieved. However, the filter transition to high attenuation occurs more slowly, thus a higher sampling rate is required. This method is known as oversampling and can be used if sufficient processing time is available. The decimation and interpolation methods can be used to reduce the internal processing rate; however, multirate signal processing increases complexity of the algorithm. Otherwise, the higher sampling rate puts more demand on the processor and also shortens the model length of the digital filter, degrading its ability to model the unknown system. High-order adaptive filters can be used to counteract this effect if there is sufficient processor power and memory available.

Furthermore, if some small negative dc component is present in the measured error signal e(n), an adaptive algorithm can gradually increase its output in an attempt to cancel the dc error component. However, the physical secondary path generally has no response to dc because of the frequency response of the loudspeaker and microphone. This dc output to the control source can reach a level high enough to saturate the controller and power amplifier. When the LMS algorithm is implemented using fixed-point arithmetic, a bandpass filter can be used to prevent dc and low-frequency elements from growing in the filter coefficients during the updating process. This dc offset can also be eliminated by an adaptive bias canceler [4], which is simply a first-order recursive high-pass filter.

A continuous signal can be recovered without distortion from its ideally sampled version by low-pass filtering. The ideal reconstruction filter has a flat gain response and linear phase characteristic in the passband, extending from dc to half of the sampling frequency, and infinite attenuation in the stopband and beyond. Any departure of the filter characteristic from the ideal introduces spectral distortion. Furthermore, due to the high levels of low-frequency noise, the very large, high-power amplifiers are required. High-pass filters may be required to prevent very low-frequency energy from overdriving the loudspeaker and causing premature failures.

Analog Interface

Interfacing a DSP to an analog environment involves a waveform conversion as illustrated in Figure 23. There are two main types of ADCs and DACs: parallel and serial. If the system requires more than one analog input, there are two types of architecture to consider. Figure 23 shows a multiplexed system in which the ADC is shared among the input channels. The input and error signals are sampled simultaneously and multiplexed using an analog multiplexer.

The digital ANC system assumes that the sampling period between samples is uniform. The spectrum distortion caused by the sampling jitters results in line broadening, or spectral smearing. In an adaptive ANC system, the performance loss can be significant and in severe cases may result in instability. One way to reduce sampling jitters is to initiate the analog-to-digital and digital-to-analog conversions with hardware instead of software. This hardware-initiated conversion is standard practice today.

ANC SYSTEM SOFTWARE

The test hardware is designed so that the system function is determined principally by the software, allowing the system to be modified and improved without hardware redesign. The software implementation of ANC comprises three stages: initialization, off-line secondary-path modeling or off-line secondary-path and feedback path modeling, and on-line active noise cancellation. A software-based start-up procedure is automatically performed by the system when it is turned on. This procedure includes processor initialization and gain adjustment of all input and output signals, and secondary-path modeling using additional white noise, which is discontinued after the on-line noise control begins.

The program is coded in assembly language and is optimized to minimize computation time and maximize the number of filter weights. An additional constraint is to yield a sampling rate high enough to give good resolution in the models and a filter length sufficient to model the real plants.

Implementation Considerations

In discrete-time signals and systems, the digital filter structures and algorithms are derived on the basis of infinite-precision arithmetic. However, when these filters are implemented in digital hardware, only finite precision is available. These effects can either cause deviations from the original design criteria or create an effective noise at the filter output. The difference in performance between a digital filter and its discrete model is said to be due to finite word length effects (or quantization effects). Previous sections dealt principally with the mathematical development of adaptive filtering and did not really address many practical details. This section discusses some practical considerations of the implementation of adaptive filters in digital hardware systems.

Quantization Effects in Digital Adaptive Filters

Finite word length effects are found in many forms. In general, the major finite word length effects that result in degradation of digital adaptive filter performance can be broadly categorized into the following classes:

- Quantization errors
  - Input quantization
  - Coefficient quantization
- Arithmetic errors
  - Roundoff (or truncation) noise
  - Overflow

An excellent treatment of these topics is available in digital signal processing books by Oppenheim [41] and Jackson [24].

In the digital implementation of adaptive algorithms, the filter coefficients and the computational results are quantized to a certain limited precision. This quantization error leads to degradation in the performance of the adaptive filter from the theoretically expected performance of an infinite-precision implementation. Therefore, digital adaptive filter implementation in limited precision requires special

attention. The goal is to minimize the potential accumulation of quantization errors in the filter coefficient adaptation algorithm computation so that they do not reach unacceptable levels. Effects of finite precision in adaptive filters have been reported in the literature [38, 39, 40].

Assuming that the input data samples are properly scaled, their values lie between –1 and 1. Each data sample and its filter coefficient are represented by B + 1 bits. The quantizer can be modeled as introducing an additive noise to the unquantized value x′. Thus, the following equation can be written:

$$x(n) = Q[x'(n)] = x'(n) + \gamma(n) \tag{82}$$

where x′(n) is an unquantized value and the associated quantization error γ(n) is a uniformly distributed random noise with zero mean (that is, E[γ(n)] = 0) and a variance of:

$$\sigma_\gamma^2 = \frac{2^{-2B}}{12} \tag{83}$$

Therefore, the longer the word length, the smaller the quantization noise. Each additional bit in the ADC results in a 6-dB gain of signal-to-quantization noise ratio or dynamic range.

Assuming that the input sequences and filter coefficients have been properly normalized, there is no error introduced in addition. However, the sum can become larger than 1; this is known as overflow. The technique used to inhibit the probability of overflow is scaling; that is, constraining the signal at each node within a digital filter to a magnitude less than unity. Since reducing the amplitude of the signal reduces the signal-to-noise ratio, which can cause early termination of the adaptive algorithm [39], the signals must be kept as large as possible.

For adaptive filters, the feedback path makes scaling far more complicated. The dynamic range of the filter output is determined by the time-varying filter coefficients, which are unknown at the design stage. For the LMS transversal filter, the scaling of the filter output and coefficients is set by scaling of the desired signal d(n) [40]. Figure 25 shows a block diagram of the traditional LMS algorithm using fixed-point arithmetic. This scaling technique uses the scale factor s, where 0 < s < 1, implemented by a shift to the right of the desired signal (instead of the input signal) to prevent the overflow of filter coefficients during the weight updating. Reducing the power of d(n) reduces the gain demand on the filter, therefore reducing the magnitude of the tap values. Usually, the required value of s is not expected to be very small. Since s scales the desired signal, it does not affect the rate of convergence. An alternative method to prevent the occurrence of overflow is to use the leaky LMS algorithm, as discussed previously.

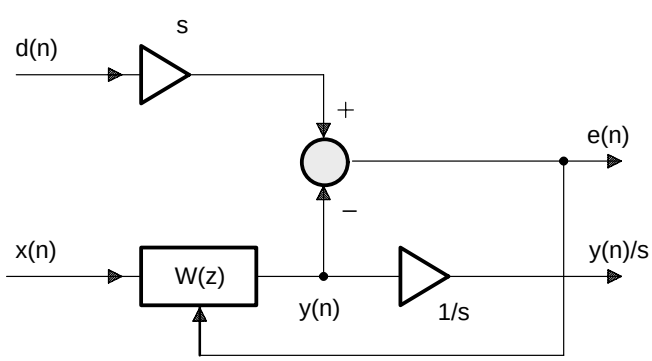


Figure 25. Fixed-Point Arithmetic Model of the LMS Algorithm

From Figure 10, the digitally implemented FXLMS algorithm is summarized as follows:

$$y(n) = Q[\underline{w}^T(n)\underline{x}(n)]$$

$$= Q\left[\sum_{i=0}^{N-1} w_i(n)x(n-i)\right] \tag{84}$$

$$x'(n) = Q[\underline{c}^T\underline{x}(n)]$$

$$= Q\left[\sum_{i=0}^{M-1} c_i x(n-i)\right] \tag{85}$$

$$\underline{w}(n+1) = \underline{w}(n) - Q\left[\mu e(n)\ \underline{x}'(n)\right] \tag{86}$$

or

$$w_i(n+1) = w_i(n) - Q\left\{Q\left[\mu e(n)\right]x'(n-i)\right\},\ i = 0,\ 1,\ldots,\ N-1 \tag{87}$$

where Q[x] denotes fixed-point quantization of the quantity x.

Assuming a scaling factor s_j is used to scale the reference input x(n) to prevent overflow during the computation of $x'(n)$ in equation (85), the output $x'(n)$ can be bounded as:

$$|x'(n)| = s_j\left|\sum_{i=0}^{M-1} c_i x(n-1)\right| = s_j x_{max}\sum_{i=0}^{M-1}|c_i| < 1 \tag{88}$$

where x(n – i) is replaced by its maximum value, x_{max}, and due to the fact that the magnitude of a sum is less than or equal to the sum of magnitudes.

From equations (86) and (87), a scaling factor s_j is chosen to satisfy:

$$s_j < \frac{1}{x_{max}\sum_{i=0}^{M-1}|c_i|} \tag{89}$$

where c_i can be determined at the end of the off-line modeling. Scaling the input in this way assures that overflow never occurs at any of the nodes in the filter.

Assuming that the input signal is a narrowband signal, overflow can be avoided for all sinusoidal signals if the input is scaled [42] by:

$$s_j \le \frac{1}{x_{max}\ \max|C(e^{j\omega})|},\ |\omega| \le \pi \tag{90}$$

As mentioned earlier, for the traditional LMS transversal filter, the scaling of the entire process (filter output and coefficients) is set by scaling of the desired signal d(n), which is inaccessible in an ANC system. The same effect can be achieved by scaling e(n). This scaling technique uses the scaling factor s, where $0 < s < 1$, to prevent overflow. To compensate for the power loss, the compensation factor 1/s is inserted in the filter output to drive the control source. This scheme is equivalent to scaling the desired signal by a factor s. Note that scaling on x(n), e(n), and y(n) by the scaling factors can be implemented in AGC blocks shown in Figure 23.

When the convolution sum in equations (84) and (85) is calculated using a multiplier with an internal double-precision accumulator, the internal quantization noise is avoided. When the product is transferred out of the accumulator, the result is quantized to single precision and a roundoff error is produced if the rounding operation is used. When updating weights according to equations (86) and (87), the product of $\mu e(n)$ produces a double-precision number, and this is quantized and multiplied by $x(n - i)$. This result is quantized again and then added to the original stored weight value, $w_i(n)$, to form the updated value, $w_i(n + 1)$.

Real-Time Software Implementation Process

The adaptive structures and algorithms described previously can be implemented on the ’C25. Figure 26 shows the flowchart of a process that can be used to minimize the amount of time spent on finite word-length effects analysis and real-time debugging.

In the first stage, algorithm design and study is performed on the general-purpose computer in a nonreal-time environment. Once the algorithm is understood, the filter is implemented using a high-level C program or MATLAB with double-precision coefficients and arithmetic. This filter is considered an ideal filter.

In the second stage, the C program is rewritten on the general-purpose computer in a way that emulates the same sequence of operations with the same parameters and state variables as will be implemented on the ’C25. It is carefully redesigned and restructured, tailoring it to the architecture, the I/O timing structure, and the speed and memory constraints of the ’C25. This program then serves as a detailed outline for the ’C25 assembly language program, or it can be compiled using the TMS320 fixed-point DSP C compiler.

In the third stage, the ’C25 assembly program is developed, assembled, and tested on the general-purpose computer, using the ’C2x software simulator with test data from a disk file. This test data is either a short version of the data used in the second stage that can be internally generated from the program or digitized data emulating a real application environment. Output from the simulator is saved as another disk file and is compared with the equivalent output of the C program from the second stage. Since the simulator requires data to be in some particular finite precision format, certain precision is lost during data conversion. Once an agreement is obtained between these two outputs within a tolerable range, the DSP assembly program is essentially correct.

The final stage is to download this assembled and linked program into the target hardware and bring it to real-time operation. Thus, the real-time debugging process is primarily constrained to debugging the I/O timing structure of the algorithm and testing the long-term stability of the algorithm. Once the algorithm is running, the parameters can be tuned again in a real-time environment.

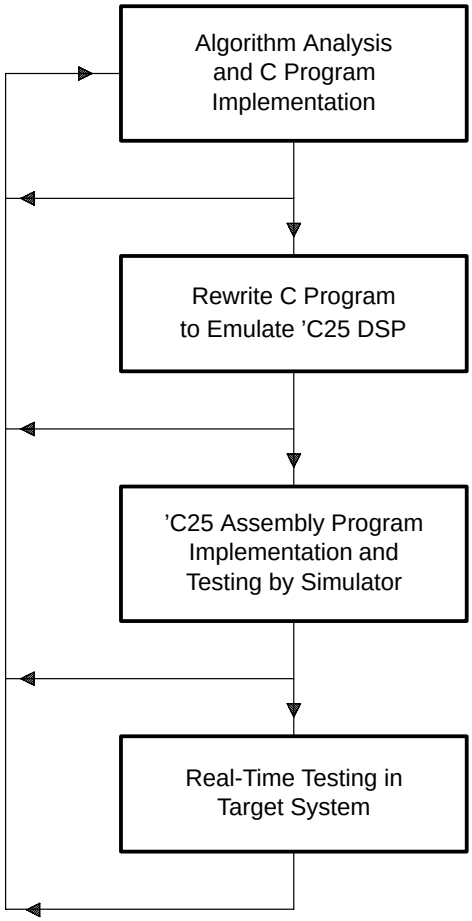


Figure 26. Adaptive Filter Implementation Process

Implementation of Adaptive Filters With the TMS320C25

The complexity of an adaptive filter is usually measured in terms of its multiplication rate and storage requirement. However, when these algorithms are implemented on commercially available DSP chips, data flow and handling considerations are also major factors in efficiently implementing adaptive filter systems. The parallel hardware multiplier, the pipeline architecture, and the amount of fast on-chip memory are important. High-speed parallel and serial ports enable fast data flow on and off chip. Implementation can be made more efficient by taking advantage of these attributes in the DSP’s architecture. Adaptive transversal filters with the two most widely used algorithms in active noise control (LMS and leaky LMS) are implemented here using the ’C25.

The ’C25 has 544 words of fast on-chip data RAM divided into three blocks: B0 (256 words), B1 (256 words), and B2 (32 words). Block B0 is configurable as either data memory or program memory. To produce the fastest possible adaptive filtering routine, all data buffer memories and filter coefficients are stored in data RAM. In general, B0 is used to store adaptive weights, B1 is used as data buffer memory, and B2 is used for constants and temporary storage.

The transversal filter generates its output y(n) by performing a convolution (or inner product) operation:

$$y(n) = \sum_{i=0}^{N-1} w_i(n)x(n-i) \tag{91}$$

The implementation of equation (91) is illustrated using C language as:

```
y[n] = 0;
for (i=0; i<N; i++)
{y[n] += wn[i]*xn[i]; }
```

where wn[i] represents $w_i(n)$ and xn[i] represents x(n – i).

The architecture of the ’C25 is optimized to implement a sum of products, such as an FIR filter. The MACD instruction enables complete multiply/accumulate, data move, and pointer update operations to be accomplished in a single instruction cycle (80 ns) if the filter coefficients are stored in on-chip RAM. An N-weight transversal filter can be implemented as:

```
LARP
LRLK   ARn,LASTAP       ;  point to the x(n–N+1)
RPTK   N–1              ;  repeat next instruction N times
MACD   COEFFP,*–        ;  multiply/accumulate
APAC
```

where ARn is an address register that points to x(n – N + 1) and the prefetch counter (PFC) points to the last weight, $w_{N-1}(n)$. When the MACD instruction is repeated, the coefficient address contained in the PFC is incremented by 1 during its operation. Therefore, the components of weight vector $\underline{w}(n)$ are stored in B0, as shown in Figure 27. The MACD in repeat mode also copies data pointed to by ARn, the next (higher) on-chip RAM location. The buffer memories of the transversal filter are stored as shown in Figure 27.

In general, roundoff noise occurs after each multiplication. However, the ’C25 has a 16-bit multiplier and a 32-bit accumulator, so there is no roundoff when summing the set of product terms. All multiplication products are presented in full precision, and rounding is performed after they are summed, so that we get y(n) from the accumulator with only one roundoff, which minimizes the roundoff noise in the output y(n).

The most widely used LMS algorithm is expressed as:

$$w_i(n+1) = w_i(n) + \mu e(n)x(n-i) \tag{92}$$

for i = 0, 1, ... , N – 1. Since $\mu e(n)$ is constant for N weight updates, the error signal e(n) is first multiplied by μ to get $\mu e(n)$. This constant can be stored in the T register and then multiplied by x(n – i) to update $w_i(n)$. An implementation method in C of the LMS algorithm in equation (92) is illustrated as

```
uen = u*e[n];
for (i=0; i<N; i++)
   {wn[i] += uen * xn[i]; }
```

where e[n] represents e(n).

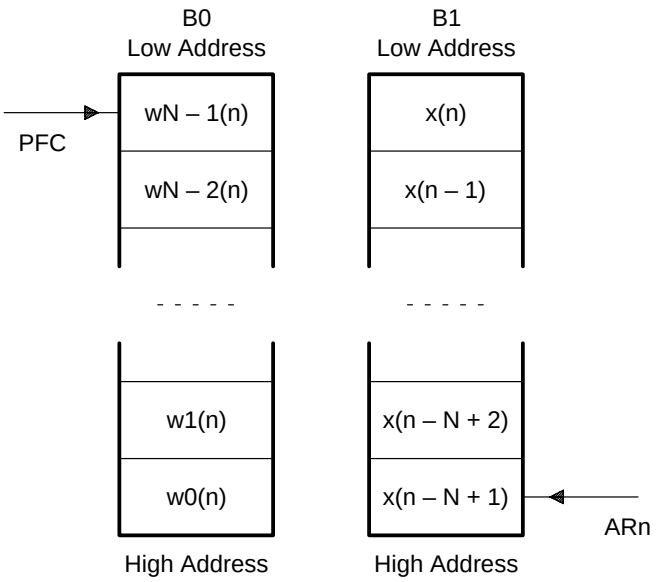


Figure 27. Memory Layout of Weight Vector and Data Vector

The ’C25 provides two powerful instructions to perform the updates in equation (92). The ZALR instruction loads a data memory value into the high-order half of the accumulator and rounds the value. This rounding occurs by setting bit 15 of the accumulator to 1 and clearing bits 0–14 of the accumulator to 0s. The MPYA instruction accumulates the previous product in the P register and multiplies the operand with the data in the T register. Assuming that ue(n) is stored in the T register and the current address pointer is AR3, the adaptation of each weight is shown in the following instruction sequence:

```
     LRLK  AR1,N–1       ;  load loop counter
     LRLK  AR2,COEFFD    ;  point to Equation
     LRLK  AR3,LASTAP+1  ;  point to x(n–N+1), since MACD already move
                         ;  elements of current x(n – i) to the next higher
                         ;  location
     LARP  AR3           ;
     MPY   *–,AR2        ;  P = ue(n) * x(n–N+1)
ADAP ZALR  *,AR3         ;  load Equation
     MPYA  *–,AR2        ;  ACC = P+Equation
     SACH  *+,0,AR1      ;  store Equation
     BANZ  ADAP,*–,AR2   ;  loop again if counter not expired
```

Figure 28 shows the architecture of the ’C25’s central arithmetic logic unit, including the multiplier, the accumulator, and the T and P registers.

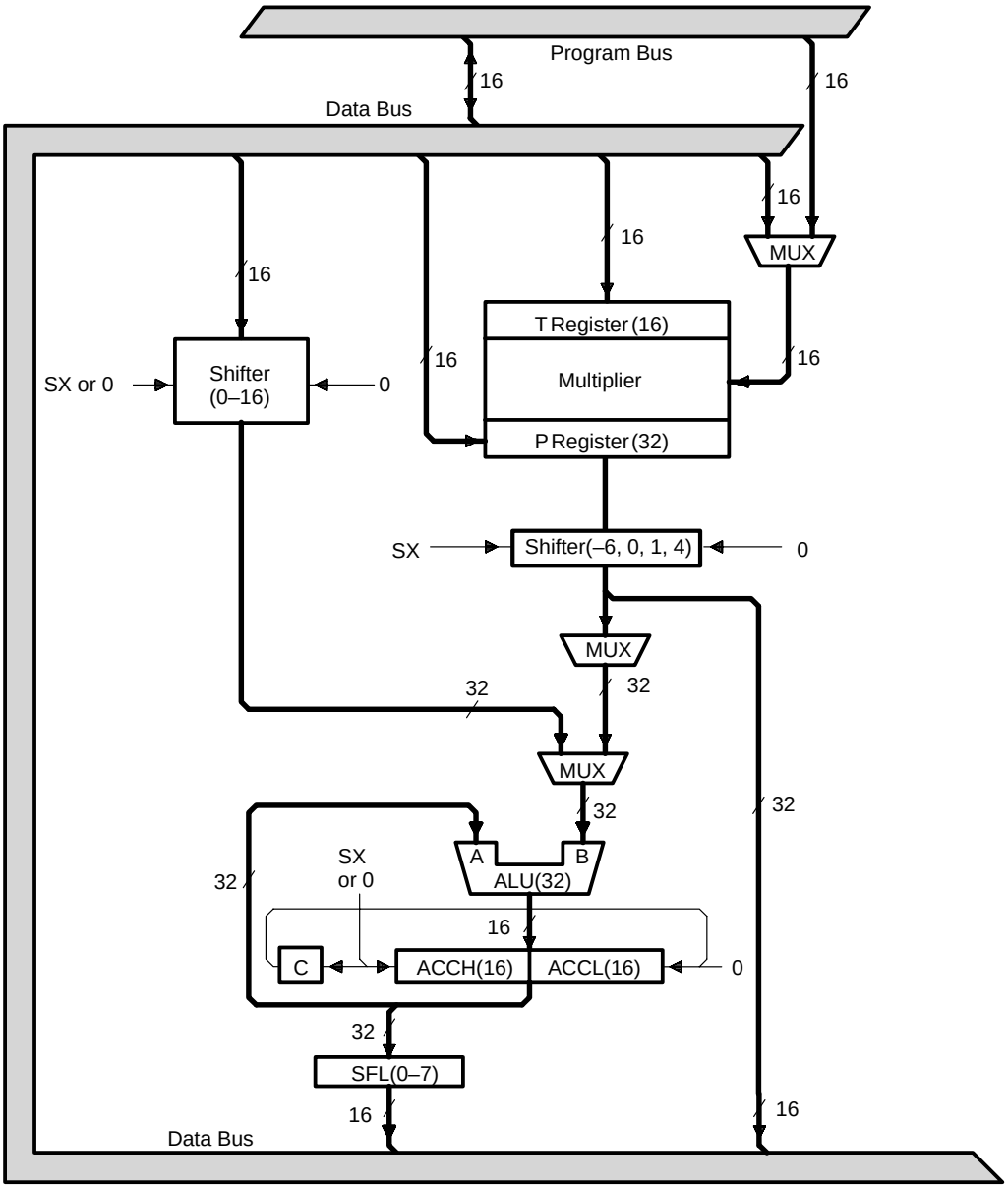


Figure 28. TMS320C25 Central Arithmetic Logic Unit (CALU)

The leaky LMS algorithm used in many fixed-point implementations has the form:

$$w_i(n + 1) = vw_i(n) + \mu e(n)x(n - i) \tag{93}$$

where v is slightly less than 1. Another way to realize this algorithm is to let $v = 1 - c$ and $c << 1$, which gives:

$$w_i(n + 1) = w_i(n) - cw_i(n) + \mu e(n)x(n - i) \tag{94}$$

The barrel shifter can be used to implement this modification of the leaky LMS algorithm efficiently.

To achieve the highest throughput using ZALR and MPYA, $cw_i(n)$ can be implemented by right-shifting $w_i(n)$ M bits, where 2^{-M} is close to c. Since the length of the accumulator is 32 bits and the high word (bits 16 to 31) is used for updating $w_i(n)$, shifting $w_i(n)$ right M bits can be implemented by loading $w_i(n)$ and shifting left 16 – M bits. The sequence of ’C25 instructions to implement equation (94) is:

```
       LRLK   AR1,N-1          ; load loop counter
       LRLK   AR2,COEFFD       ; point to Equation
       LRLK   AR3,LASTAP+1     ; point to x(n-N+1)
       LT     ERRF             ; T = ERRF = u*e(n)
       MPY    *-,AR2
ADAPT  ZALR   *,AR3
       MPYA   *-,AR2
       SUB    *,LEAKY          ; LEAKY=16-M
       SACH   *+,0,AR1
       BANZ   ADAPT,*-,AR2
```

Using the TMS320C2x Simulator to Observe Noise Cancellation

TI’s debugging tools can be an invaluable asset in understanding the operation of the included code. The ’C2x simulator provides the ability to single-step through a program and observe the contents of registers and memory locations and the states of status bits as they change from step to step. The simulator also provides features for watching variables, for viewing code simultaneously in both C and assembly language, and for setting breakpoints. Stepping through the provided code and observing the state of the DSP dictated by the program flow helps provide a quicker and deeper understanding of the programming involved in an ANC system.

The simulator has a feature that allows the programmer to send data to an output file. The ANC code uses this feature and creates an output file containing the error signal. This signal is the residual noise left after the original noise source has been summed with the canceling wave. If the code is working properly, this signal should get smaller and smaller as the noise is canceled.

The advantage of creating an output file stems from the ability to display the information graphically. The file created from the simulator is in the form of a stream of hexadecimal numbers. This data can be displayed in a variety of ways. For example, converting the file into binary (using TISIMDAT.EXE) allows the data to be displayed on the monitor using a program called SG (for Show Graphics; contact the DSP lab at Northern Illinois University at 815-753-9967 for a copy). Even more useful is to convert the file into ASCII and then load it into MATLAB. Once there, the data can not only be plotted graphically but can also be imported easily into documents and manipulated mathematically with MATLAB’s extensive capabilities. Figure 29 shows a plot of the error signal as obtained using MATLAB.

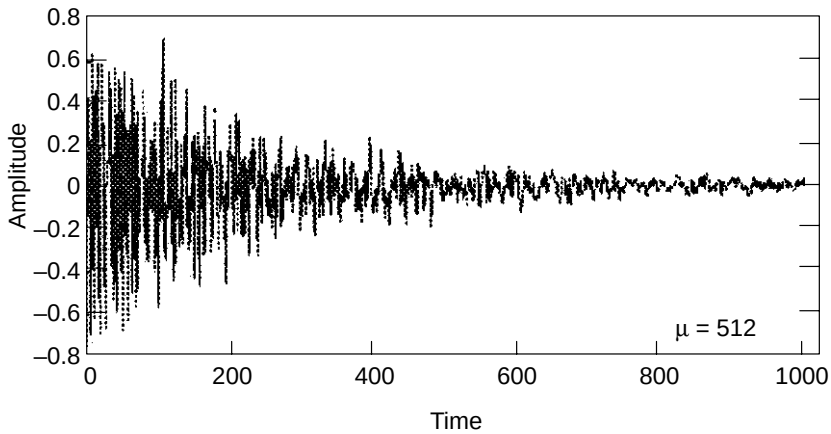


Figure 29. The Error Signal Imported From MATLAB

Understanding How Individual Parameters Affect Algorithm Performance

It is important to investigate how certain parameters affect the performance of the algorithm. For example, the value of μ alone can drastically affect the results of the system. μ is the step size of the adaptive filter used in active noise control. The size of μ determines the stability and convergence rate of the algorithm. As the step size of the filter increases, the algorithm converges more rapidly. However, if the step size is too large, the system may not converge at all. Even if the system avoids divergence, too large a μ value may produce an intolerable amount of residual error on the signal. The error signal shown in Figure 29 was generated using a μ value of 512. Notice the difference between it and the signal shown in Figure 30, which was generated using a μ value of 2048. Notice that the convergence is much quicker with the increased step size.

μ is just one of the many parameters that can be changed to vary performance. The role of each and how it affects the performance of the system must be investigated for any individual application.

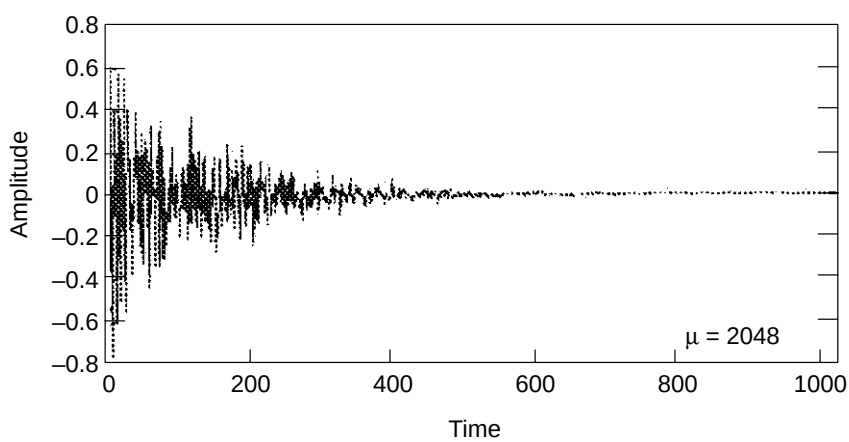


Figure 30. Error Signal Generated With μ = 2048

Understanding the function of the parameters and tracing through the code on the simulator allows the user to gain enough knowledge about the program to implement basic changes in the code itself. The code can be manipulated to implement active noise control in slightly different ways. Experiments can be done and performance can be optimized. In short, the code can now be customized to perform in various applications.

As an example of a simple modification, consider that the code created for the simulator allows for 16-bit precision on the data stream. What happens to the signal and the performance of the system if a lower precision ADC were used instead? How does the performance vary with precision? One way to find out is to modify the included code so that when the ADC is being used, a lower precision is fed in. This can be done by placing the following line of code in the appropriate places in the FIR filter and when updating the coefficients.

```
ANDK   0FFC0h,15        ; Takes the 10 most significant bits
```

The hexadecimal number in the ANDK instruction determines the number of bits of precision to be obtained. In the example line of code above, the precision is set for ten bits. By taking a logical AND with 0FFC0h (equivalent to binary 1111 1111 1100 0000), the ten most significant bits are preserved and the rest are set to zero.

This and other modifications can be tested and debugged on the simulator to aid in the evaluation and optimization of the alteration done.

PHYSICAL SETUP OF EXPERIMENTAL ANC SYSTEM IN AN ACOUSTIC DUCT

The acoustic duct system has been widely used for modeling the acoustic cavity in many applications such as exhaust systems, HVAC, and the motor/generator housings. The three feedforward ANC algorithms discussed previously are used for noise attenuation: the filtered-X LMS (FXLMS) algorithm [3], the filtered-X LMS algorithm with feedback cancellation (FBFXLMS) [15], and the filtered-U recursive LMS (FURLMS) algorithm [l2]. The coefficients of the adaptive filters in these algorithms are modified with the leakage factor [5] to reduce the effect of the overflow and quantization errors.

The simplest experimental system to set up is a one-dimensional duct system such as as that shown in Figure 2 on page 10. However, in addition to the components shown in the figure, several other pieces of equipment are necessary in order to make the system functional. The success of an ANC system strongly depends on the ability to manipulate the gain of the involved signals. Thus, it is vitally important to run the signals detected with the microphones through a preamplifier whose gain can be controlled. It is also necessary to have a power amplifier to control the strength of the antinoise sent through the canceling speaker. The gains of these amplifiers must be carefully adjusted if maximum performance of the system is to be achieved.

The entire setup of this simple ANC system is shown in Figure 31. The two microphone-detected signals are fed into the preamplifier before they are sent to the ANC hardware. The adaptive filtering occurs in the ANC hardware, resulting in the creation of the antinoise waveform. This signal is output through the power amplifier before being sent through the speaker to cancel the incoming noise signal. In this test system, the noise signal is created using a function generator and is passed through the power amplifier before emerging from the speaker.

This experimental ANC duct system is realized using the low-cost 16-bit fixed-point TMS320C25 DSP. Single- and multiple-tone sinusoids of different frequencies are applied to the system as the input noise source signals. The performance of the DSP-based system for each algorithm is analyzed. The experimental results are compared and practical factors in achieving high level of noise attenuation are discussed. It is shown that the best noise attenuation at a reasonable overhead to the DSP is obtained by using the FBFXLMS algorithm. The system using the FBFXLMS algorithm provides a feedback-free reference signal, which improves the system performance in the range where strong frequency response exists in the feedback path.

The 8-order Butterworth low-pass filters have a 500-Hz bandwidth. For the schematic diagram of this low-pass filter, refer to Appendix E. Because 500 Hz is the highest frequency of interest in ANC applications, the function of these low-pass filters is to eliminate the aliasing problem and the unwanted harmonics. The Ariel DSP board has two input and output ports, each using 16-bit ADCs and DACs, respectively.

For a complete description of the system setup and a list of the components used, see Appendix F, *ANC Unit System Setup and Operation Procedure*.

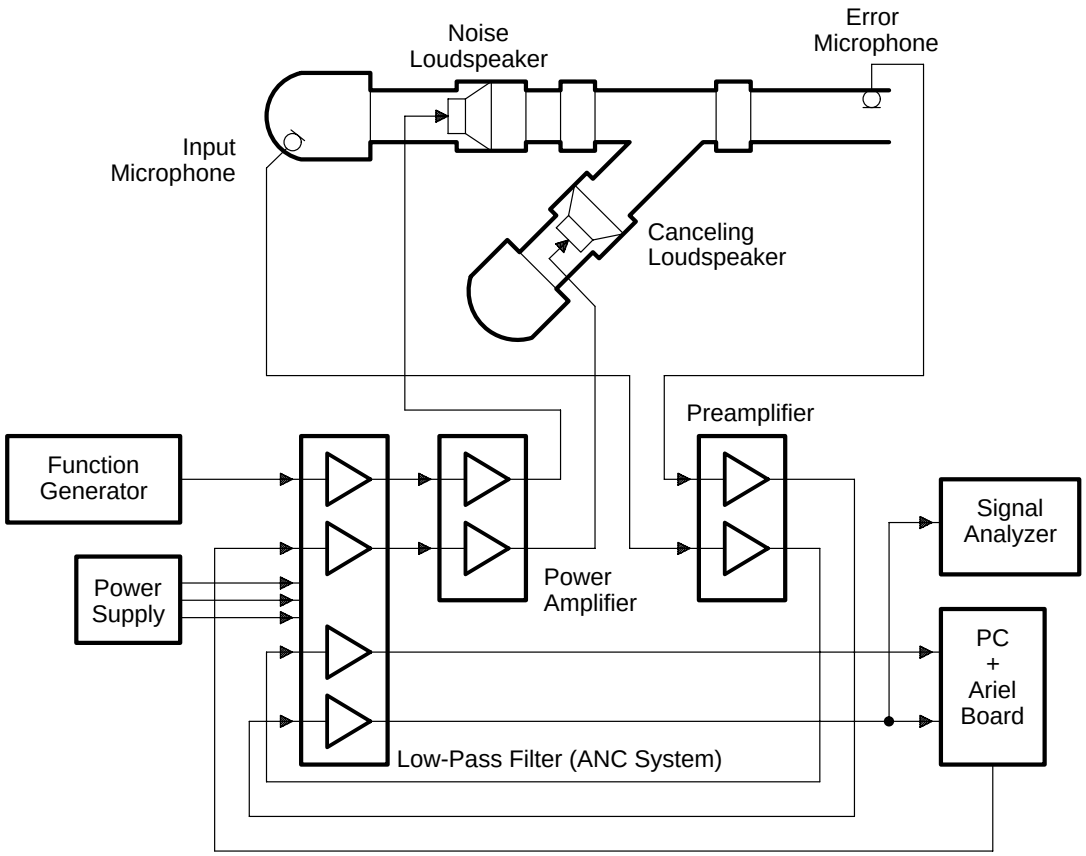


Figure 31. Experimental Setup of the One-Dimensional Acoustic ANC Duct System

OPTIMIZATION OF THE EXPERIMENTAL SYSTEM

The process of optimization begins when the designed hardware system is working as expected. The system parameters must be manipulated so that maximum performance is achieved. The key variables to be set for the system are the value of μ, the gain of the preamplifier, and the value of LEAKY. Data is gathered while varying one of these variables and holding all others at a constant value to best isolate its effect on the system. Unless otherwise noted, all data for this example were obtained with the following parameter values:

- Error-path filter C(z) order (NCz) = 127
- FIR filter W(z) order (NWz) = 127
- LEAKY = 2
- μ (off) = 128
- Preamplifier gain = 36 dB
- Input noise = 2 V sine wave (before power amplifier)

Determining the Value of μ

μ was the first parameter that was checked to see how it affected the performance of the system. The ability of the noise cancellation unit to reduce sound was measured at three different input levels: –30 dB, –26 dB, and –22 dB. The quietest setting was –30 dB, and –22 dB was the loudest. The frequency of the original noise source and the gain of the power amplifier for the canceling speaker were adjusted to get the maximum performance possible at each setting (see Table 2).

Table 2. Performance of the System as a Function of μ

μ	dB REDUCTION FOR –30 dB INPUT	dB REDUCTION FOR –26 dB INPUT	dB REDUCTION FOR –22 dB INPUT
64	21.19	29.57	34.94
96	22.91	30.05	34.85
128	24.68	30.32	30.70
160	30.42	37.45	41.01
192	32.21	37.98	40.35
224	32.52	37.27	39.18
256	34.38	38.42	40.14
288	34.38	38.01	37.86
320	37.95	40.68	40.66
352	38.35	39.25	37.94
384	36.96	38.27	37.27
416	37.26	38.62	37.23
448	39.08	38.64	37.58
480	33.85	33.07	--
512	34.93	34.15	33.15

The values of μ represent a filter step size for a 16-bit processor. The data in Table 2 is represented graphically in Figure 32.

As the value of μ increased from the low value of 64, the performance of the system increased at first, then leveled off somewhat at higher μ values. As μ increased further, the performance of the system tended to decline. One thing that can be seen from Table 2 is that the system did not have a linear dependence on the step size of the adaptive filter.

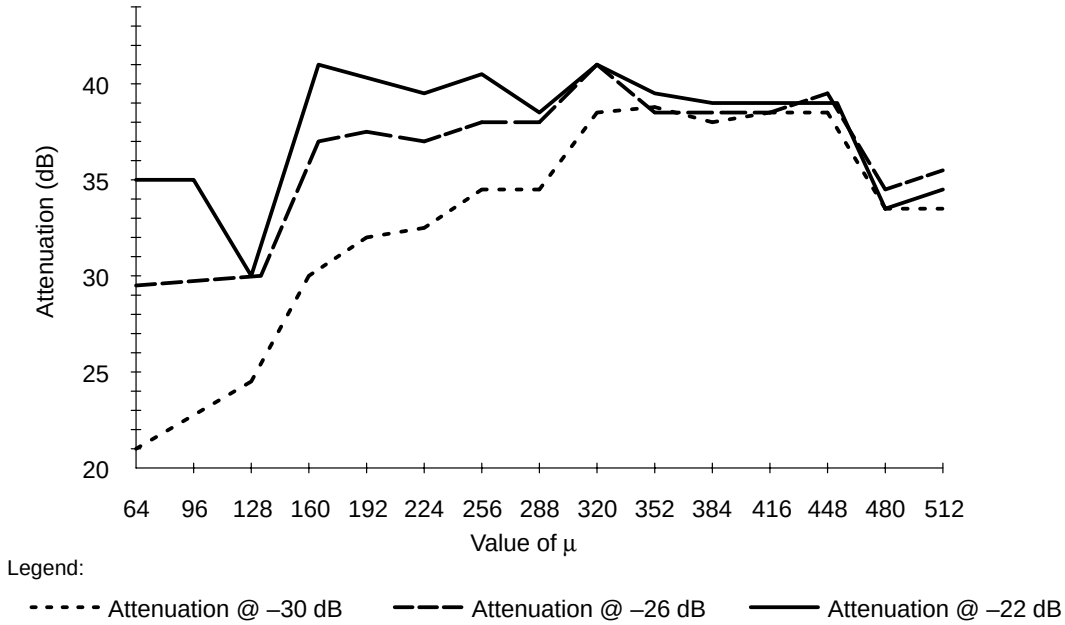


Figure 32. Level of Attenuation of the Noise Source Versus μ

The peak performance level at each sound level (settings –22 dB, –26 dB, and –30 dB on the preamplifier) is different. At the lowest tested sound level (–30 dB), a μ of 448 drew the best performance, and high levels of attenuation were obtained at μ values of 352 and 320, as well. When the sound was increased from –30 dB to –26 dB, it was the μ value of 320 that yielded the best system performance. At the highest sound level tested, the μ value of 160 reduced the original signal by the largest amount, with a μ value of 320 performing nearly as well.

Therefore, depending on which noise level is most relevant in the application, a different value of μ is chosen to optimize the system. If the system is to be optimized at all sound levels, the μ value that performs highly in all three sound levels is considered the best for the system. Because the μ value of 320 is the only value in the top three performances at each level, it seems to be the logical choice. But is there a way to better determine how the performances compare?

The most important column to look at in the data table is the reduction of the original signal that is given for a power amplifier setting of –22 dB. This column shows how well the system cancels loud noise, which is the noise generally chosen for canceling. Note that the performance of the system is greatest at this noise range for a μ value of 320 and less, but at higher μ values the system actually performs better at lower noise levels. This indicates that to attenuate loud noise, the system must have a μ value of 320 or less.

However, the system should not be characterized by how well it eliminates loud noises alone; the ability to eliminate loud noises should be weighted more heavily to emphasize that these are the noises most in need of attenuation, but other factors are also important. A crude way to evaluate the overall system performance is to apply the following formula to the data obtained.

$$\text{reduction @ } -30 \text{ dB} + 1.5(\text{reduction @ } -26 \text{ dB}) + 2(\text{reduction @ } -22 \text{ dB}) = \text{overall performance} \quad (95)$$

This formula takes into account performance over all ranges, but it places more emphasis on the ability to eliminate the louder noise sources. The results of this calculation are shown in graphical form in Figure 33, which shows that a μ value of 320 is verified as the best choice.

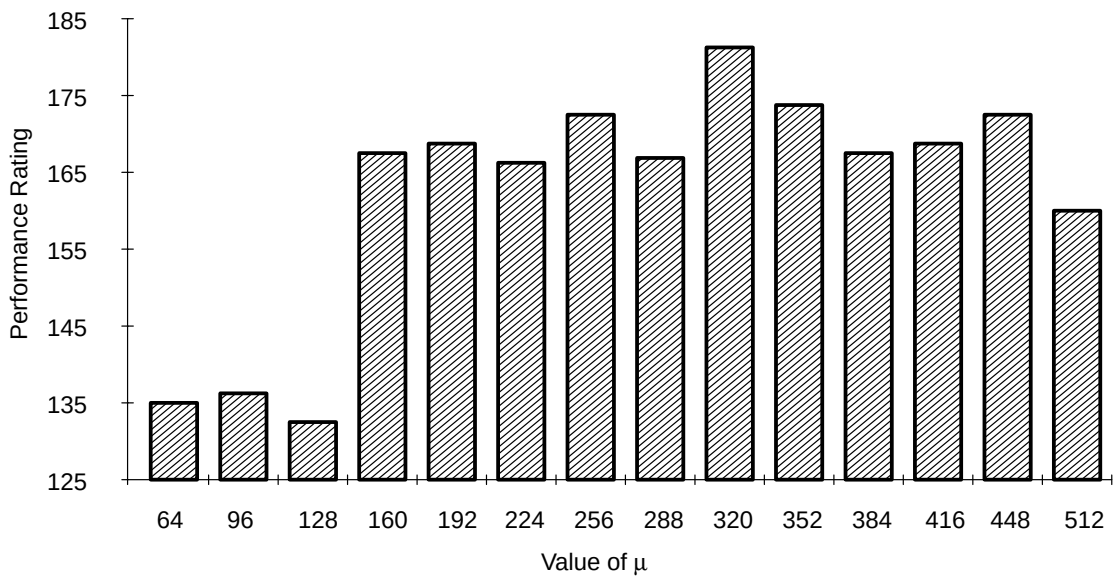


Figure 33. Overall Performance as a Function of Equation (95)

Determining the Value of LEAKY

The LEAKY parameter is used as part of a small forcing function that tends to bias each filter weight towards zero. This report previously described how this variable is used. The value of LEAKY can be chosen as 0, 1, or 2, which is the value of c for $w_i(n) = (1 - c)w_i(n - 1) + \mu\, e(n)x(n)$.

There is no reason to test the effect of the LEAKY parameter for all of the values of μ. It was determined that 320 was the best value of μ. However, the best value of LEAKY cannot be determined merely by selecting the value that yields the best results at a particular μ value. A μ value of 320 yielded the best results with a LEAKY value of 2, but that does not necessarily mean that it will also yield the best results with a LEAKY value of 1. Care must be taken to ensure that the best results of each LEAKY value are compared against each other.

For this reason, the following values of μ were chosen to be tested: 160, 256, 320, and 352. The top three values according to the performance rankings were all tested, as well as the value with the highest reduction at the loudest setting ($\mu = 160$). These four values of μ form a basic starting point that can be expanded if the data gathered indicates that the optimum system performance is achieved outside of these values.

As the value of LEAKY was varied for the four different μ values selected, it became apparent that a value of LEAKY = 2 yielded the superior performance. This can be seen from a graphic of a sample of the data gathered, shown in Figure 34. At every μ value tested, the system using LEAKY = 2 outperformed the same system using a lower LEAKY value.

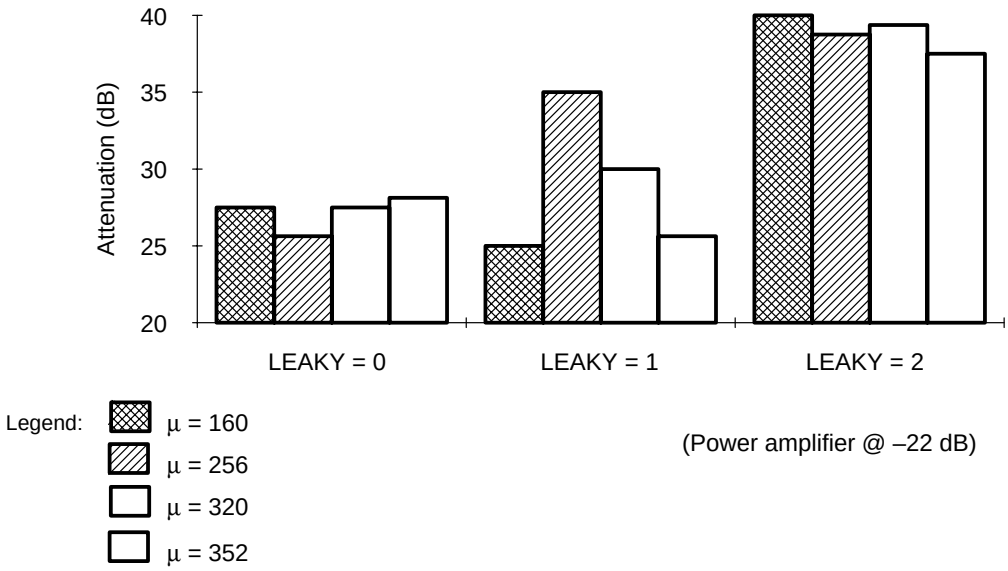


Figure 34. Noise Reduction of System as a Function of LEAKY

Determining the Gain of the Preamplifier

The last of the parameters that needs to be adjusted for this simple ANC system is the gain of the preamplifier. As in the previous section, the data was recorded for four different μ values: 160, 256, 320, and 352. The LEAKY value used was 2, because it was clearly the best choice for this system.

The best value for the microphone preamplifier was found to be a setting of 36 dB, as shown from the data graphed in Figure 35. This setting on the preamplifier proved to be the best setting on three of the four μ values tested. As μ decreased, the gain on the preamplifier was increased in order to achieve a high performance level. At higher μ values, however, too high a setting on the preamplifier saturated the system and caused the filter to diverge. Preamplifier settings of 48 and 52 dB could not be measured for μ greater than 320 because of this, although these settings produced good performance when using a smaller step size.

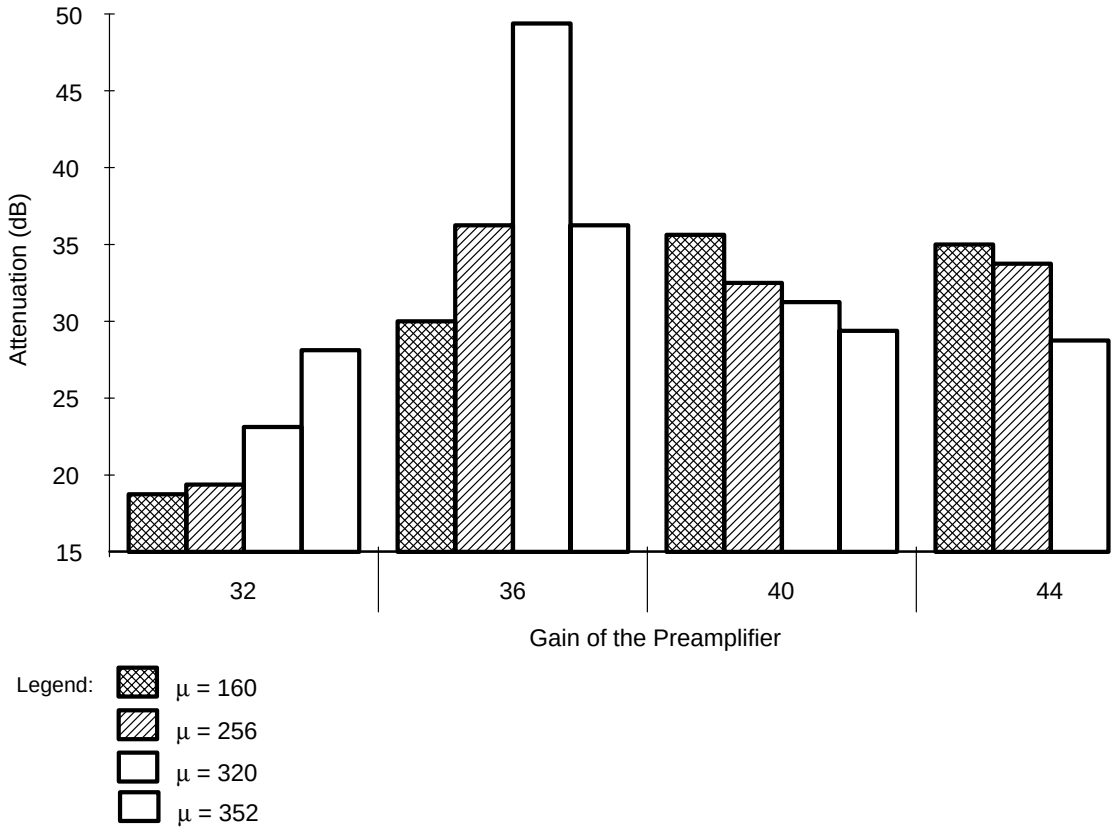


Figure 35. Noise Reduction of the System as a Function of Preamplifier Gain

Single-Tone Sinusoidal Noise Source Case

When the noise source is a 200-Hz single-tone sinusoid, the order of both W(z) and C(z) is 64, the adaptation step size of W(z) is 0.01 (μ/32768 for a 16-bit processor), and the LEAKY value is 2 [5]. Figure 36 shows the error spectra received at the error microphone while the ANC system is both turned on (dashed line) and turned off (solid line), using on the FXLMS algorithm.

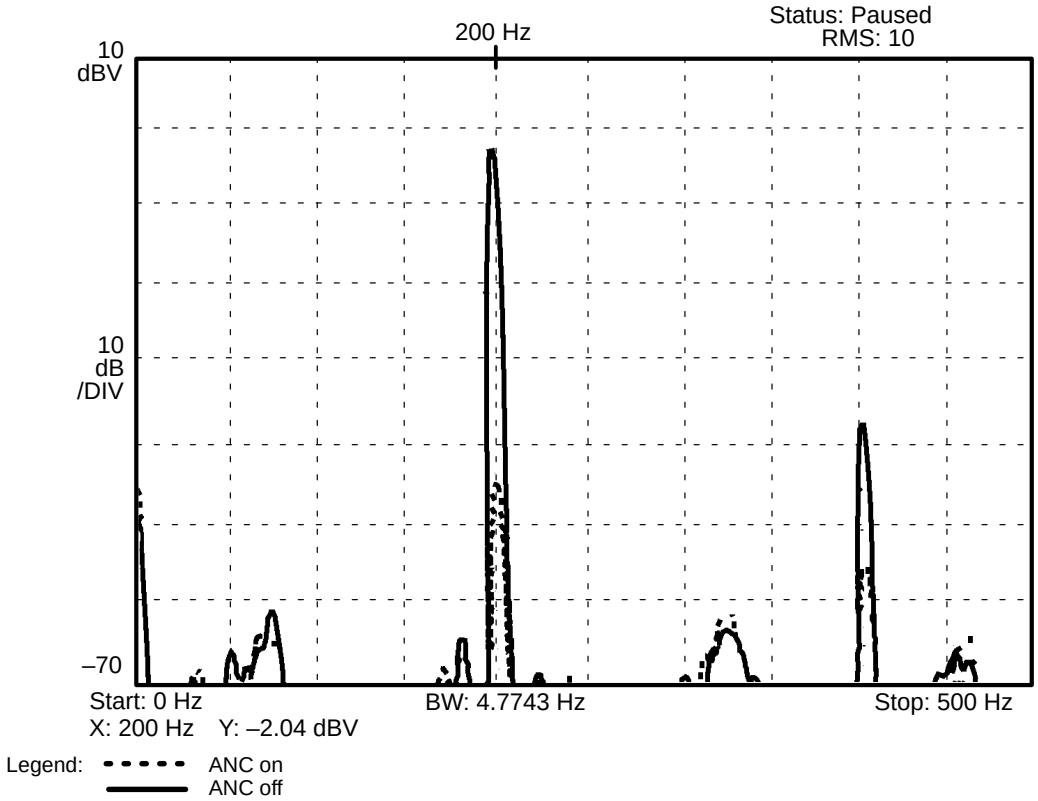


Figure 36. Error Spectra for FXLMS Algorithm, Noise Source Is a 200-Hz Single-Tone Sinusoid

Figure 36 shows that a 41.65-dB noise attenuation can be achieved when the ANC system is turned on. To evaluate the ability of the system setup, the noise frequency was varied from 100 Hz to 500 Hz in 50-Hz steps and the three different algorithms were tested. The noise attenuation data obtained are listed in Table 3.

Table 3. Noise Attenuation for a Single-Tone Sinusoidal Noise Source

NOISE SOURCE FREQUENCY (Hz)	ATTENUATION PER ALGORITHM (dB)		
	FXLMS	FBFXLMS	FURLMS
100	36.73	45.38	38.71
150	35.88	43.07	38.40
200	41.65	46.88	48.29
250	25.78	43.67	28.04
300	3.93	4.14	3.29
350	50.06	51.92	53.58
400	53.33	52.91	53.81
450	56.86	57.70	63.82
500	33.51	34.71	35.73
Order of Filters	W(z):64 C(z):64	W(z):64 C(z):64 D(z):64	A(z):63 B(z):63 C(z):63

As shown in Table 3, the attenuation achieved by the ANC system varied at different frequencies. The reason for this phenomenon is that every signal path has a different gain contribution at different frequencies of the signal; therefore, it is necessary to observe the transfer function of all the signal paths. White noise was used as the excitation signal to drive the target loudspeaker, and the signal from the target microphone was connected to the signal analyzer. A hard copy of the frequency response on the screen was obtained using the plotter. The frequency response of the transfer functions of primary path P(z), secondary path H(z), and feedback path F(z) are shown in Figure 37, Figure 38, and Figure 39, respectively. P(z) is the transfer function between the noise loudspeaker and the error microphone; H(z) is the transfer function between the canceling loudspeaker and the error microphone; and F(z) is the transfer function between the canceling loudspeaker and the input microphone.

As shown in the figures, the frequency response of H(z) was attenuated at approximately 300 Hz, but P(z) was amplified. Hence, if the primary noise to be canceled contains a frequency component in this range, the performance of the ANC system degrades, because the antinoise signal magnitude is constrained.

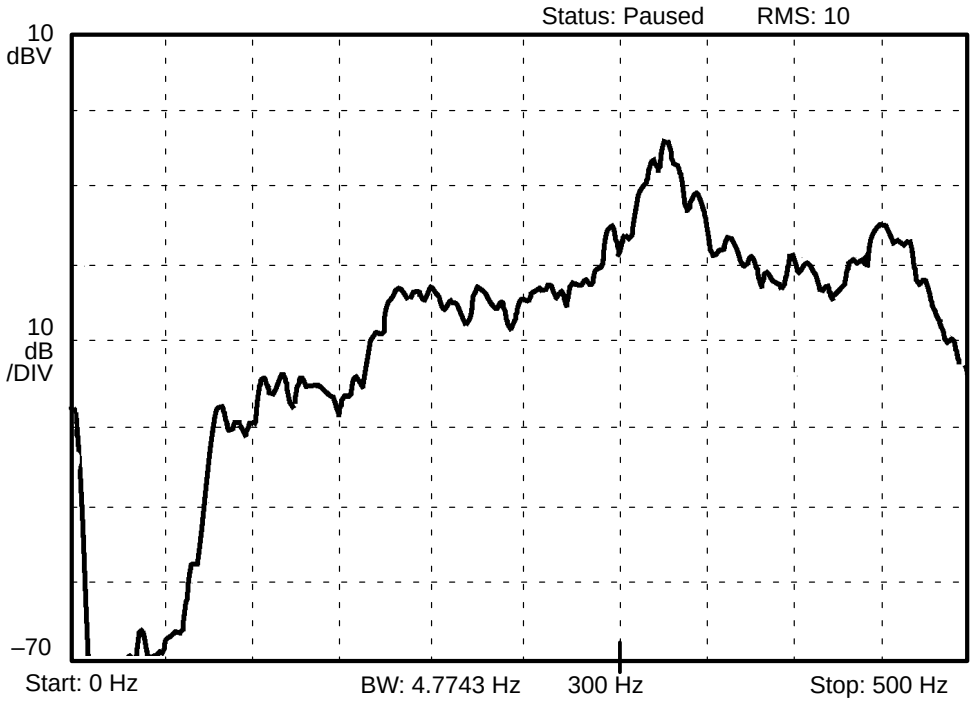


Figure 37. Frequency Response of Primary Path P(z)

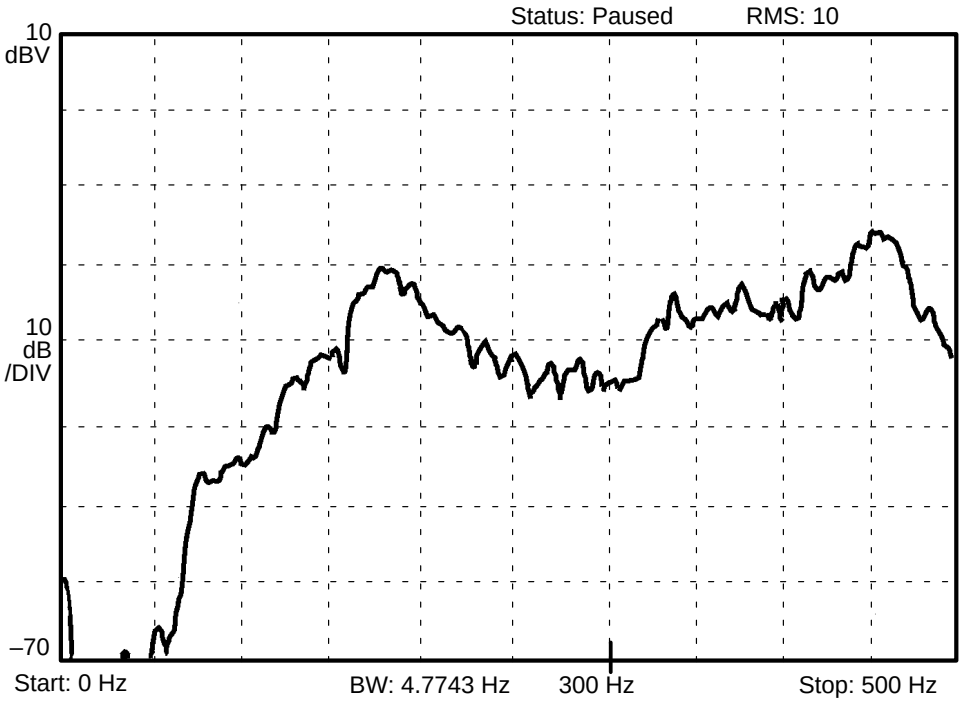


Figure 38. Frequency Response of Secondary Path H(z)

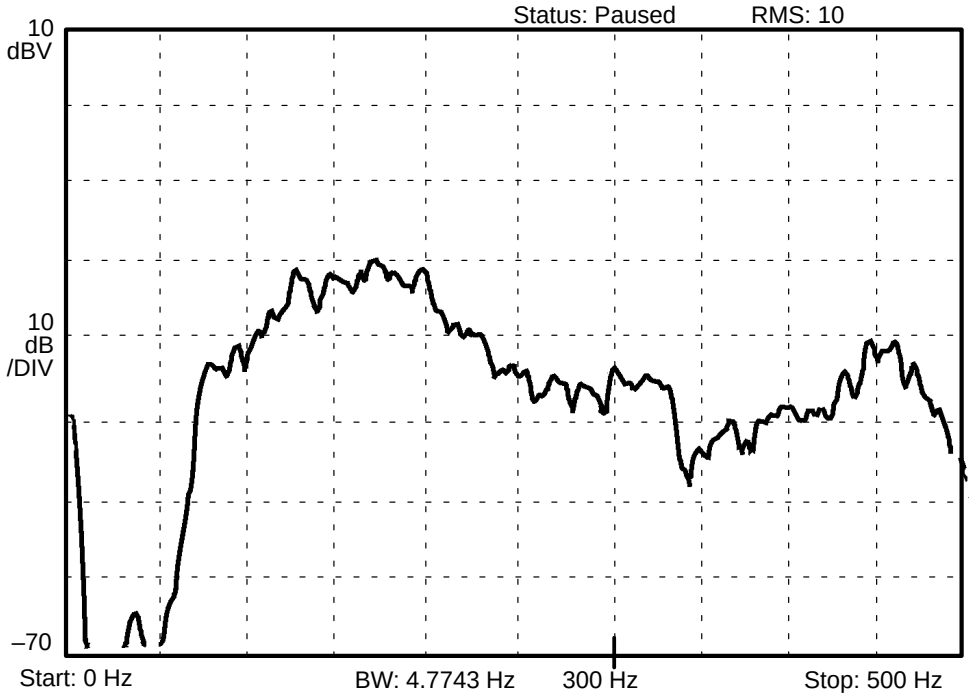


Figure 39. Frequency Response of Feedback Path F(z)

As shown in Figure 39, the frequency response of feedback path F(z) of the experimental setup is stronger in the range of 100 Hz–250 Hz than in the range of 250 Hz–500 Hz. This means that the contamination of the reference signal due to the feedback signal is stronger in the lower frequency range. As shown in Table 3, the FBFXLMS algorithm provides better noise attenuation (about 10 dB on average) than the FXLMS algorithm in the 100 Hz–250 Hz range. This shows the contribution of the feedback compensation filter D(z) in the ANC system when a strong feedback signal is present. Table 3 also shows that the use of the FURLMS algorithm resulted in only a few dB improvement over the FXLMS algorithm in our ANC system.

Multiple-Tone Sinusoidal Noise Source Case

For the case in which the primary noise is a 3-tone sinusoidal signal, Figure 40, Figure 41, Figure 42, and Figure 43 show the spectra of the error signal at the error microphone with the ANC system turned on (dashed line) and turned off (solid line). The frequencies of the primary noise are 200 Hz, 350 Hz, and 450 Hz. The step size and the value of LEAKY are the same as for the single-tone case, but the order of the filters varies as shown in Table 4.

Table 4. Filter Orders for 3-Tone Sinusoidal Noise Source

ALGORITHM	FIGURE 40	FIGURE 41	FIGURE 42	FIGURE 43
	FXLMS	FXLMS	FBFXLMS	FURLMS
Order of Filters	W(z):64 C(z):64	W(z):127 C(z):127	W(z):64 C(z):64 D(z):64	A(z):63 B(z):63 C(z):63

The results shown in these figures indicate that the FXLMS, the FBFXLMS, and the FURLMS algorithms are all effective for real-time ANC applications. Each of them shows that at least 40 dB of noise attenuation can be achieved for every frequency component. From Figure 40, Figure 42, and Figure 43, the same conclusions can be obtained as in the experiments using a single-tone noise source; that is, the ANC system shows better performance (200-Hz component) in the range that has a strong frequency response of the feedback path if the FBFXLMS algorithm is used instead of the FXLMS algorithm, and the FURLMS algorithm can help the ANC system to achieve higher noise attenuation than the FXLMS algorithm. Figure 40 and Figure 41 show that very slight improvement is achieved when the order of both filters W(z) and C(z) was increased from 64 to 127 in the FXLMS algorithm.

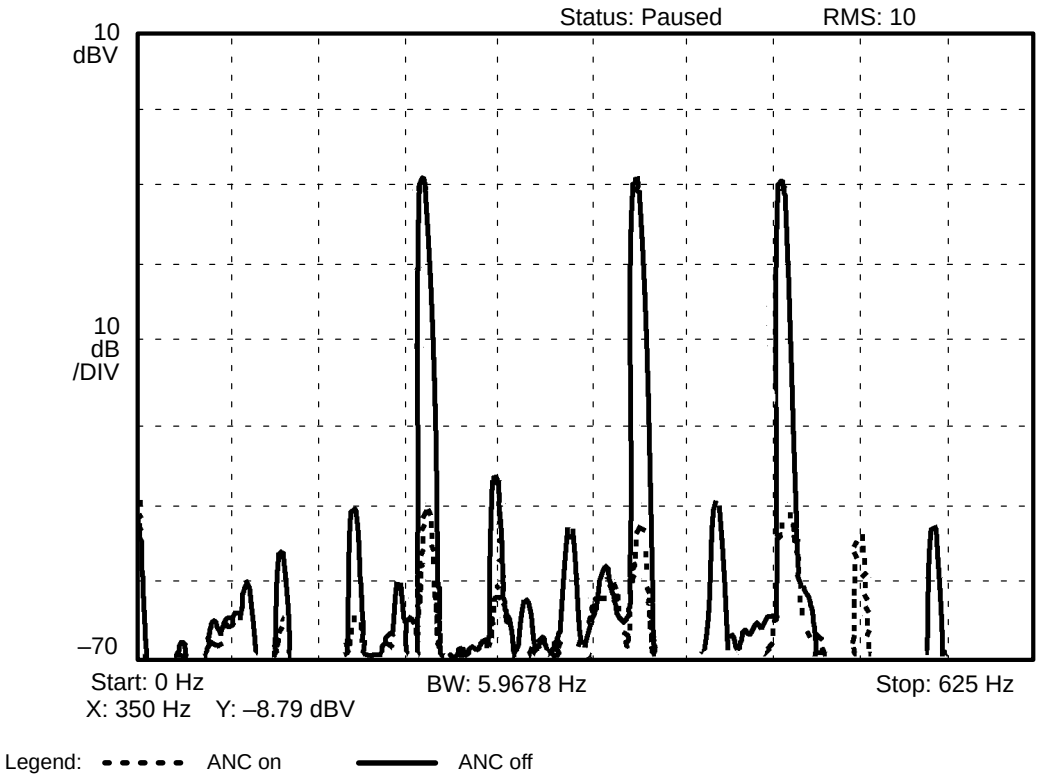


Figure 40. Error Spectra for FXLMS Algorithm, Noise Source Is a 3-Tone Sinusoid, Order of W(z) = 64, Order of C(z) = 64

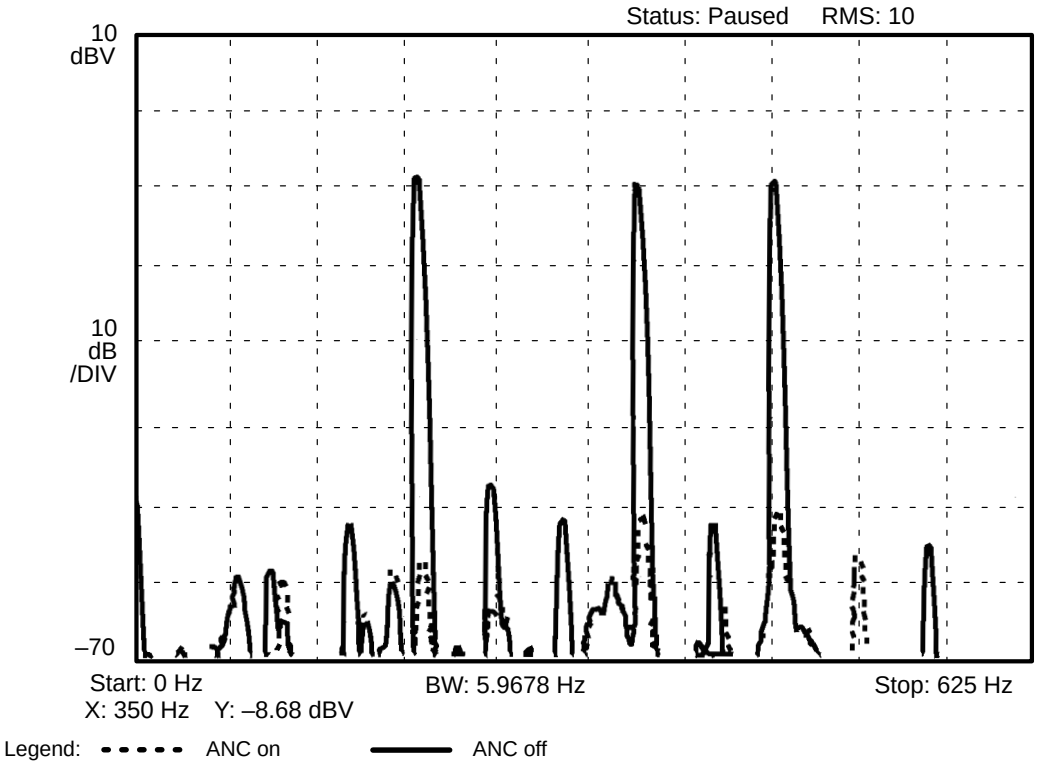


Figure 41. Error Spectra for FXLMS Algorithm, Noise Source Is a 3-Tone Sinusoid, Order of W(z) = 127, Order of C(z) = 127

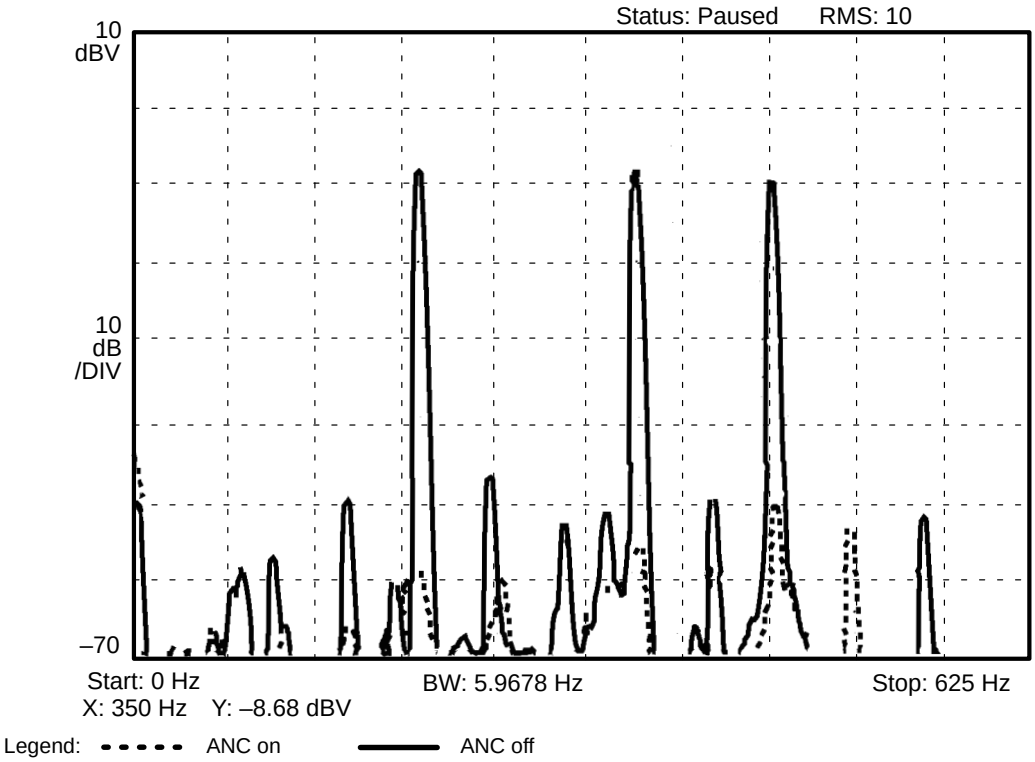


Figure 42. Error Spectra for FBFXLMS Algorithm, Noise Source Is a 3-Tone Sinusoid, Order of W(z) = 64, Order of C(z) = 64, Order of D(z) = 64

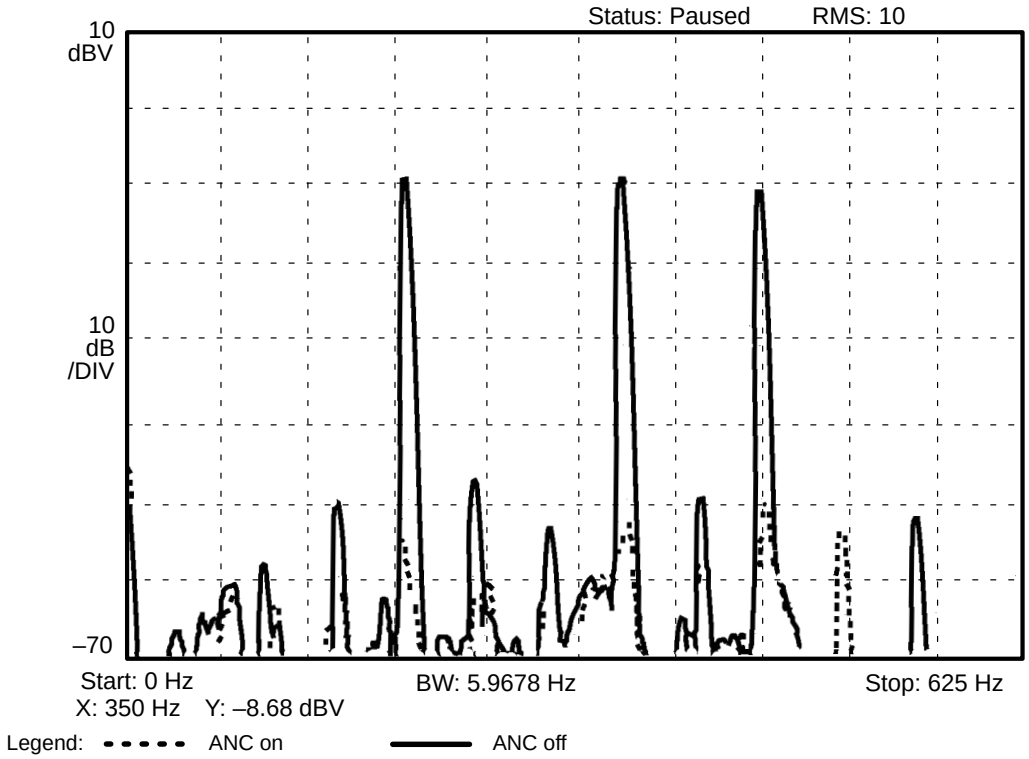


Figure 43. Error Spectra for FURLMS Algorithm, Noise Source Is a 3-Tone Sinusoid, Order of A(z) = 63, Order of B(z) = 63, Order of C(z) = 63

CONCLUSION

Each of the three algorithms can help an ANC system a high level of noise attenuation regardless of the type of sinusoidal noise source (single-tone or multiple-tone). The feedback compensation filter D(z) of the FBFXLMS algorithm can be used to get a feedback-free reference signal, which results in better performance than using the FXLMS algorithm in the frequency range that is affected by the feedback signal. The FURLMS algorithm can achieve an average of 3 dB more attenuation in the ANC system than the FXLMS algorithm.

The optimum parameter settings for this simple ANC system are $\mu = 320$, LEAKY $= 2$, and preamplifier gain $= 36$. The methods used in optimizing this system are similar to those needed for other ANC systems. Each parameter was looked at individually, and then its interrelation with the other parameters was considered to determine the best values. In this way, optimizing a complex and interrelated set of parameters can be greatly simplified into a well organized and structured procedure.

REFERENCES

1. Nelson, P. A., and S. J. Elliott, *Active Control of Sound*, Academic Press, San Diego, CA, 1992.
2. Lueg, P., "Process of Silencing Sound Oscillations," U.S. Patent No. 2,043,416, June, 1936.
3. Burgess, J. C., "Active Adaptive Sound Control in a Duct: A Computer Simulation," *J. Acoust. Soc. Am.*, Vol. 70, No. 3, Sept. 1981, pp. 715–726.
4. Widrow, B., and S. D. Stearns, *Adaptive Signal Processing*, Prentice-Hall, Englewood Cliffs, NJ, 1985.
5. Kuo, S. M., and C. Chen, "Implementation of Adaptive Filters with the TMS320C25 or the TMS320C30," *Digital Signal Processing Applications with the TMS320 Family*, Volume 3, edited by P. Papamichalis, Prentice-Hall, Englewood Cliffs, NJ, 1990, pp. 191–271.
6. Olson, H. F., and E. G. May, "Electronic Sound Absorber," *J. Acoust. Soc. Am.*, Vol. 25, No. 6, Nov. 1953, pp. 1130–1136.
7. Olson, H. F., "Electronic Control of Noise, Vibration, and Reverberation," *J. Acoust. Soc. Am.*, Vol. 28, No. 5, 1956. pp. 966–972.
8. Morgon, D. R., "A Hierarchy of Performance Analysis Techniques for Adaptive Active Control of Sound and Vibration," *J. Acoust. Soc. Am.*, Vol. 89, No. 5, May, 1991, pp. 2362–2369.
9. Nishimura, M., "Some Problems of Active Noise Control for Practical Use," *Proc. Int. Symp. Active Control of Sound and Vibration*, Tokyo, 1991, pp. 157–164.
10. Roure, A., "Self-Adaptive Broadband Active Sound Control System," *J. of Sound and Vibration*, Vol. 101, No. 3, 1985, pp. 429–441.
11. Morgan, D. R., "Analysis of Multiple Correlation Cancellation Loop With a Filter in the Auxiliary Path," *IEEE Trans. on ASSP*, Vol. ASSP-28, No. 4, August, 1980, pp. 454–467.
12. Eriksson, L. J., "Development of the Filtered-U Algorithm for Active Noise Control," *J. Acoust. Soc. Am.*, Vol. 89, No. 1, January, 1991, pp. 257–265.
13. Elliott, S. J., I. M. Stothers, and P. A. Nelson, "A Multiple Error LMS Algorithm and Its Application to the Active Control of Sound and Vibration", *IEEE Trans. on ASSP*, Vol. ASSP-35, No. 10, Oct., 1987, pp. 1423–1434.
14. Tichy, J., and G. E. Warnaka, "Effect of Evanescent Waves on the Active Attenuation of Sound in Ducts", Proc. of Inter-Noise, 1983. pp. 435–438.
15. Poole, L. A., G. E. Warnaka, and R. C. Cutter, "The Implementation of Digital Filter Using a Modified Widrow-Hoff Algorithm for the Adaptive Cancellation of Acoustic Noise," Proc. ICASSP, San Diego, CA, 1984. pp. 21.7.1–21.7.4.
16. Kuo, S. M., and J. Chen, "Multiple-Microphone Acoustic Echo Cancellation System with the Partial Adaptive Process," *Digital Signal Processing*, Vol. 3, No. 1, January 1993. pp. 1–10.
17. Eriksson, L. J., M. C. Allie, and R. A. Greiner, "The Selection and Application of an IIR Adaptive Filter for Use in Active Sound Attenuation," *IEEE Trans. on ASSP*, Vol. ASSP-35, No. 4. April 1987. pp. 433–437.
18. Feintuch, P. F., "An Adaptive Recursive LMS Filter," *Proc. of IEEE*, Vol. 64, No. 11, November 1976. pp. 1622–1624.
19. Chaplin, G. G. B., and R. A. Smith, "Waveform Synthesis - The Essex Solution to Repetitive Noise and Vibration," Proc. Inter-noise 83, pp. 399–402.
20. Elliott, S. J., and P. Darlington, "Adaptive Cancellation of Periodic, Synchronously Sampled Interference," *IEEE Trans. on ASSP*, Vol. ASSP-33, No. 3, June 1985. pp. 715–717.

21. Widrow, B., et al, “Adaptive Noise Canceling: Principles and Applications”, *Proc. of IEEE*, Vol. 63, No. 12, Dec. 1975. pp. 1692–1716.
22. Ziegler, E. W., “Selective Active Cancellation System for Repetitive Phenomena,” U.S. Patent, No. 4,878,188. Oct. 1989.
23. Morgan, D. R., and J. Thi, “A Multitone Pseudocascade Filtered-X LMS Adaptive Notch Filter,” *IEEE Trans. ASSP*, Vol. 41, No. 2, Feb. 1993. pp. 946–956.
24. Jackson, L. B., *Digital Filters and Signal Processing*, 2nd Edition, Charter 13, Kluwer Academic Publishers, Norwell, MA. 1989.
25. Glover, J. R., Jr., “Adaptive Noise Canceling Applied to Sinusoidal Interferences,” *IEEE Trans. on ASSP*, Vol. ASSP-25, No. 6, Dec. 1977. pp. 484–491.
26. Pfaff, D. D., N. S. Kapsokavathis, and N. A. Parks, “Methods for Actively Attenuating Engine Generated Noise,” US Patent 5,146,505. Sept. 1992.
27. Wheeler, P. D., and D. Smeatham, “On Spatial Variability in the Attenuation Performance of Active Hearing Protectors,” *Applied Acoustics*, Vol. 36, 1992. pp. 159–162.
28. Kuo, S. M., and D. Vijayan, “Feedback Active Noise Control Systems,” Proc. Int. Conf. Signal Processing Applications and Technology, 1993. pp. 132–141.
29. Allie, M. C., C. D. Bremigan, and L. J. Eriksson, “Hardware and Software Considerations for Active Noise Control,” Proc. ICASSP-88, New York, April 1988. pp. 2598–2601.
30. Goodman, S. D., “Electronic Design Considerations for Active Noise and Vibration Control Systems,” Proc. Recent Advances in Active Control of Sound and Vibration, Blacksburg, VA, 1993. pp. 519–526.
31. Goodman, S. D., and S. S. Wise, “A Discussion of Commercial Experience with Active Noise Control on Industrial Fans and Air Handlers Used for Heating, Ventilating and Air Conditioning,” Proc. Inter-noise, 1990. pp. 797–800.
32. Olson, D. A., A. D. Hallstrom, and S. S. Wise, “Active Noise Control Systems and Air Moving Devices,” Proc. Inter-noise 1989. pp. 475–478.
33. Nishimura, M., “Some Problems of Active Noise Control for Practical Use,” Proc. Int. Symp. Active Control of Sound and Vibration, Tokyo, 1991. pp. 157–164.
34. Burlage, K., et al., “An Update of Commercial Experience in Silencing Air Moving Devices with Active Noise Control,” Proc. Noise-Con., 1991. pp. 253–258.
35. Suzuki, S., et. al., “A Basic Study on an Active Noise Control System for Compressor Noise in a Refrigerator,” Proc. Int. Symp. Active Control of Sound and Vibration, Tokyo, 1991. pp. 255–260.
36. Elliott, S. J., I. M. Stothers, P. A. Nelson, A. M. McDonald, D. C. Quinn, and T. Saunders, “The Active Control of Engine Noise Inside Cars,” Proc. Inter-Noise, 1988. pp. 987–996.
37. Kuo, S. M., and B. M. Finn, “A General Multi-Channel Filtered LMS Algorithm for 3-D Active Noise Control Systems,” Second Int. Con. on Recent Developments in Air- And Structure-Borne Sound and Vibration, 1992. pp. 345–352.
38. Gitlin, R. D., H. Meadors, and S. B. Weinstein, “The Tap-Leakage Algorithm: An Algorithm for the Stable Operation of a Digital Implemented, Fractionally Adaptive Spaced Equalizer,” *Bell System Tech. J.*, Oct. 1982. pp.
39. Gitlin, R. D., J. E. Mazo, and M. G. Taylor, “On the Design of Gradient Algorithms for Digitally Implemented Adaptive Filters,” *IEEE Trans. Circuit Theory*, Vol. CT-20, March 1983. pp. 125–136.
40. Caraiscos, C., and B. Liu, “A Roundoff Error Analysis of the LMS Adaptive Algorithm,” *IEEE Trans. on ASSP*, Vol. ASSP-32, No. 1, Feb. 1984. pp. 34–41.

41. Oppenheim, A. V., and R. W. Schafer, *Discrete-Time Signal Processing*, Prentice-Hall, Englewood Cliffs, NJ. 1989.
42. Texas Instruments, *TMS320C2x User's Guide*, 1993.
43. Crochier, R., R. Cox, and J. Johnson, "Real-Time Speech Coding," *IEEE Trans. Communications*, April 1982. pp.
44. Elliott, S. J., P. A. Nelson, I. M. Stothers and C. C. Boucher, *In-Flight Experiments on the Active Control of Propeller-Induced Cabin Noise*, *Journal of Sound and Vibration*, Vol. 140, No. 2, 1990. pp. 219–238.
45. Melton, D. E., and R. A. Greiner, *Adaptive Feedforward Multiple-Input, Multiple-Output Active Noise Control*, Proc. ICASSP, 1992. pp. II–229–232.
46. Guicking, D., and M. Bronzel, *Multi-Channel Broadband Active Noise Control in Small Enclosures*, Proc. Inter-Noise, 1990. pp. 1255–1258.
47. Garcia, D., *Precision Digital Sine-Wave Generation with the TMS32010*, Chap. 8 in *Digital Signal Processing Applications with the TMS320 Family*, vol. 1, edited by K. S. Lin, Texas Instruments, 1989.
48. Texas Instruments, *TMS320C2x DSP Starter Kit User's Guide*, 1993.

APPENDIX A: PSEUDO RANDOM NUMBER GENERATOR

Two basic techniques can be used for pseudo random number (white noise) generation. The first technique is the table look-up method using a random set of stored samples, and the second technique is based on a shift register with feedback. Both techniques generate a pseudo random number sequence: a sequence that repeats itself after a finite period and is, therefore, not truly random for all time. The length of the sequence for the table look-up method is determined by the number of stored data samples, while the shift register technique's length is determined by the length of the register.

A shift register with feedback from specific elements can generate a continuous, repetitive random sequence. The algorithm of the 16-bit generator is shown in Figure 44, where XOR denotes the exclusive-OR logic operation. The maximum sequence length L before repetition is:

$$L = 2^M - 1$$

where M is the number of bits in the shift register. An output from the sequence generator is the entire M-bit word of the register.

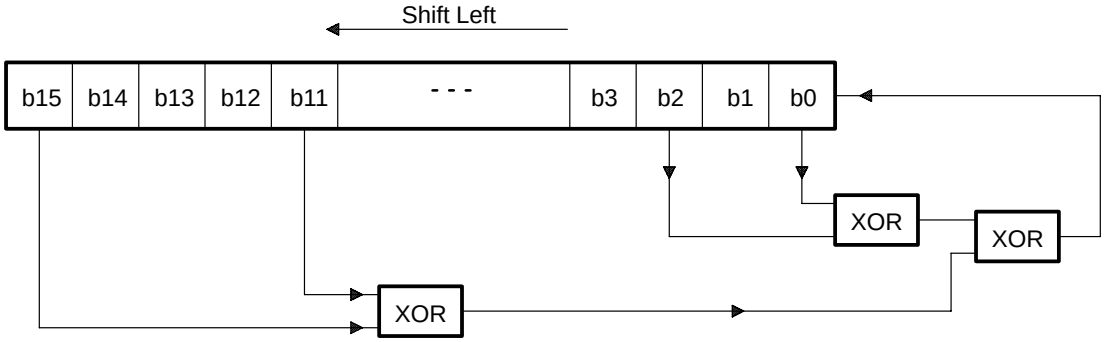


Figure 44. Pseudo Random Number Generator, 16-Bit Case

The assembly program implementation of this white noise generator on the TMS320C25 is given in Appendix C, *TMS320C25 Ariel Board Implementation of ANC Algorithms*.

APPENDIX B: DIGITAL SINE-WAVE GENERATOR

Similar to the pseudo random number generation, there are two commonly used techniques for sine-wave generation. The first technique is the table look-up method using a set of stored sine-wave samples, and the second technique is based on a digital filter.

Table Look-Up Method

The sine-wave generation using the table look-up technique is a more conceptually simple method of generating a given periodic waveform. The technique involves the reading of a series of stored data values representing discrete samples of the waveform to be generated. The data values can be obtained either by sampling the appropriate analog waveform or, more commonly, by computing the desired values. Provided that enough samples are stored to represent one complete period of the waveform accurately, continuous signals are generated by repeatedly cycling through the data memory locations.

The sine-wave table contains N sample values equally spaced in time over one period of the waveform. The values are easily computed by evaluating the function:

$$x(n) = \sin\left(n \times \frac{360^\circ}{N}\right), \quad n = 0, 1, \ldots, N - 1$$

A sine wave is generated by stepping through the table at a constant rate, wrapping around at the end of the table whenever 360° is exceeded. The frequency f of the sine wave depends on the sampling period T and the step size Δ. That is:

$$f \text{ (in Hz)} = \frac{\Delta}{T \times N}$$

There are two sources of error that cause harmonic distortion in the table look-up algorithm:

- Quantization error is introduced by representing the sine wave table values by M-bit binary numbers.
- Larger errors are introduced when points between table entries are sampled.

This harmonic distortion occurs when Δ is not an integer. The longer the table is, the less significant the second error source is. To decrease the harmonic distortion for a give table size N, an interpolation scheme can be used to compute the sine-wave values between table entries more accurately. Linear interpolation is the simplest method to implement. The implementation of a sine-wave generator using the table look-up method on the TMS320 was developed by Garcia [47].

Digital Oscillator

A very useful method of generating sine waves for a given frequency is to use a marginally stable two-pole resonator for which the complex-conjugate poles lie on the unit circle. This recursive oscillator is the most accurate and efficient method of generating sinusoidal waveforms, particularly if quadrature signals (sine and cosine waves) are required. In this appendix, only the sine-wave generator is considered.

Consider an impulse response of the form:

$$h_s(n) = A \sin(\omega_o n) \times u(n)$$

where u(n) is the unit step function, A is the amplitude of the generated sine wave, and ω_o is the angular frequency. The system transfer function (without the gain A) is:

$$H_s(z) = \frac{Y(z)}{X(z)} = \frac{\sin(\omega_0) z^{-1}}{1 - 2 \cos(\omega_0) z^{-1} + z^{-2}}$$

This equation can be expressed as:

$$Y(z)\left[1 - e \cos(\omega_0) z^{-1} + z^{-2}\right] = X(z)\left[\sin(\omega_0) z^{-1}\right]$$

Taking the inverse z-transform of both sides and rearranging both sides gives:

$$y(n) = 2 \cos(\omega_0) y(n - 1) - y(n - 2) + \sin(\omega_0) x(n - 1)$$

Applying the unit impulse as x(n) for values of n > 2, y(n) can be calculated as:

$$y(n) = 2 \cos(\omega_0) y(n - 1) - y(n - 2)$$

with initial conditions:

$$y(1) = \sin(\omega_0) \text{ and } y(0) = 0$$

The TMS320C25 implementation of this sine-wave generator is given in Appendix C, *TMS320C25 Ariel Board Implementation of ANC Algorithms*, and a more general implementation is given in Appendix D, *General Configurable Software for ANC Evaluation*.

APPENDIX C: TMS320C25 ARIEL BOARD IMPLEMENTATION OF ANC ALGORITHMS

The algorithms discussed in this application report can be implemented on the DSP-16 Plus Algorithm Development Board (from Ariel Corporation, 908–249–2900) with the code included in this appendix. The code is well commented to aid understanding. As mentioned in the report, the code is written in modular form, breaking the algorithms down into small, easily understood parts. Code written in this manner decreases the amount of time and effort necessary to understand DSP noise control solutions.

Reading through the code is an excellent method for becoming familiar with the software aspects of an ANC system.

The Filtered-X LMS Algorithm

```
;*
;*  FILE NAME : FXLMS.ASM
;*****************************************************************************
;*
;*        This program has been modified to run on the Ariel's DSP-16
;*        Plus TMS320C25 DSP Board.
;*
;*                                                        Kai-Ming Chung
;*                                                        Oct. 1994
;*
;*****************************************************************************
;*
;*  File: ANCFXLMS.ASM
;*
;*  One-dimensional Adaptive Active Noise Control System Using
;*  Filtered-X LMS (FXLMS) Algorithm on TMS320C25
;*
;*  Sen M. Kuo , Fall 1993
;*
;*****************************************************************************
;*
;*  SYSTEM CONFIGURATION:
;*
;*        x(n)
;*  M1    *|---> |———————————————————|
;*               |                   |
;*               |  Adaptive FXLMS ANC  |  y(n)
;*        e(n)   |         System    |------>  antinoise output
;*  M2    *|---> |                   |
;*               |———————————————————|
;*
;*  where:
;*        x(n) – signal from the input microphone
;*        e(n) – signal from the error microphone
;*        y(n) – antinoise to drive the canceling loudspeaker (noise
;*               cancellation mode), or
;*               the training signal (off-line modeling mode)
;*
```

```
;* Ports: x(n) from input port B
;*        e(n) from input port A
;*        y(n) to output  port A
;*
;* The flow of program is:
;*
;* C25 initialization --> off-line secondary-path modeling -->
;* on-line active noise control
;*
;* A. Off-line modeling:
;*
;*               |--->  Speaker . error MIC --->   |
;*        y(n)---|                                 |
;*               |   ----------z(n) -      +       |
;*               |--->|  C(z)  |----->(sum)<------|e(n)
;*                   ----------        |
;*                         :  < ........  e'(n)
;* where:
;*        y(n) is an internally generated white noise
;*        C(z) is an adaptive modeling filter, updated by LMS algorithm
;*
;*
;* B. On-line noise control:
;*
;*                  ----------  y(n)            e(n) from error MIC
;*   x(n) --------->|  W(z)  |----->             |
;*            |     ----------  to speaker        |
;*         -------       :                        |
;*         | C(z)|       :                        |
;*         -------   ----:----                    |
;*            |----->| FXLMS |<-------------------
;*           x'(n)   ---------
;*
;* where: C(z) is a fixed FIR filter from previous training mode
;*        W(z) is an adaptive noise control filter, updated by FXLMS
;*
;*****************************************************************************
;*
;*     define constants (can be modified for different applications)
;*
;*-----------------------------------------------------------------------------
;*
FS:       EQU    2000      ; assume sampling rate is 2 kHz.
TRNTIM:   EQU    30000     ; training time = 15 seconds
NCZ:      EQU    64        ; order of C(z), max = 127
NWZ:      EQU    64        ; order of W(z), max = 127
MU:       EQU    4096      ; coef. update stepsize, off-line modeling
MU1:      EQU    328       ; coef. update stepsize, on-line active noise
                           ; control
MASK:     EQU    8805h     ; to mask off bits 0,2,11,15
SEED:     EQU    12357     ; seed for white noise generator
LEAKY:    EQU    2         ; leaky factor
;*
;*****************************************************************************
;*
;* Memory map:
;* B2: page 0  - data I/O buffer
;*
;* B0: page 4  - coefs of AF C(z) : ci(n), i=0,1,..,(NCZ-1)
;*               (modeling filter)         NCZ <= 127
;*
;* B0: page 5  - coefs of AF W(z) : wi(n), i=0,1,..,(NWZ-1)
;*               (noise control filter)    NWZ <= 127
```

```
;*
;* B1: page 6  – data buffer for C(z) : y(n–i), i=0,1,..,(NCZ–1)
;*
;*               During off-line modeling, buffer contains training
;*                  signal y(n–i), i=0,1,..,(NCZ–1)
;*               During on-line canceling, buffer contains filtered
;*                  version of x(n) by C(z), x'(n), n=0,1,..(NWZ–1)
;*
;* B1: page 7   – data buffer for W(z) : x(n–i), i=0,1,..,(NWZ–1)
;*
;*****************************************************************************
;*
;*     PAGE 0 (memory-mapped regs and B2) DATA MEMORY ALLOCATION
;*
;*-----------------------------------------------------------------------------
;*
P0DM:     EQU    0      ; page 0 data RAM address
DAC:      EQU    13     ; DAC I/O port
ADC:      EQU    0      ; Serial receive address
IMR:      EQU    4      ; Interrupt mask register
BUF1:     EQU    96     ; Buffer for channel A input data
BUF2:     EQU    97     ; Buffer for channel B input data
BUF3:     EQU    98     ; Buffer for channel A output data
BUF4:     EQU    99     ; Buffer for channel B output data
WN0:      EQU    100    ; storage for white noise generator
WN1:      EQU    101    ;         ″
TRNCTR:   EQU    102    ; training mode (off-line modeling) counter
STSFLG:   EQU    103    ; program status flag, 1 = training mode
ONE:      EQU    104    ; ONE = 1
VMU:      EQU    105    ; value of mu
VMU1:     EQU    106    ; value of mu1
ADCE:     EQU    107    ; address of cN–1(n)
ADWE:     EQU    108    ; address of wN–1(n)
AY0:      EQU    109    ; address for y(n)
AYE:      EQU    110    ; address for y(n–N+1)
AXPE:     EQU    111    ; address for x'(n–N+1)
AX0:      EQU    112    ; address for x(n)
AXE:      EQU    113    ; end address for x(n–N+1)
AXCE:     EQU    114    ; end address for x(n–NCZ+1)
;* data buffer for signals

TEMP:     EQU    120    ; Temporary storage location
TST0:     EQU    121    ; Storage for ST0
;*
;*-----------------------------------------------------------------------------
;*
;*        PAGE 4 (B0) DATA MEMORY ALLOCATION
;* AF C(z): ci(n),  i=0,1,2,..,NCZ–1
;*
P4DM:     EQU    512       ; PAGE 4 DATA MEM ADRS
P4PM:     EQU    65280     ; PAGE 4 PROG MEM ADRS
CE:       EQU    0         ; ci(n), end of C(z) buffer
DA_CE:    EQU    P4DM+CE   ; address of ci(n), i=NCZ–1 at CNFD
PA_CE:    EQU    P4PM+CE   ; address of ci(n),    ″    at CNFP
;*
;*-----------------------------------------------------------------------------
;*
;* Memory map:  cN–1(n)   low address <–– PA_CE, AR2, ADCE
;*              cN–2(n)
;*              ...
;*              c1(n)
;*              c0(n)     high address
;*
;*-----------------------------------------------------------------------------
```

```
;*
;*     PAGE 5 (B0) DATA MEMORY ALLOCATION
;* AF W(z): wi(n),  i=0,1,2,..,NWZ-1
;*
P5DM:     EQU    640       ; PAGE 5 DATA MEM ADRS
P5PM:     EQU    65408     ; PAGE 5 PROG MEM ADRS
WE:       EQU    0         ; wi(n), end of W(z) buffer
DA_WE:    EQU    P5DM+WE   ; address of WE at CNFD
PA_WE:    EQU    P5PM+WE   ; address of WE at CNFP
;*
;*-----------------------------------------------------------------------
;*
;* Memory map:  wN-1(n)   low address, ADWE
;*              wN-2(n)
;*              ...
;*              w1(n)
;*              w0(n)     high address
;*
;*-----------------------------------------------------------------------
;*
;* PAGE 6 (B1) DATA MEMORY ALLOCATION for C(z)
;*              in off-line modeling, y(n-i), i=0,1,...,NCZ-1
;*              in on-line canceling, x'(n-i), i=0,1,...,NWZ-1
;*
P6DM:     EQU    768       ; PAGE 6 DATA MEM ADRS
Y0:       EQU    0         ; y(n)
YE:       EQU    NCZ-1     ; y(n-NCZ+1)
XPE:      EQU    NWZ-1     ; x'(n-NWZ+1)
A_Y0:     EQU    P6DM+Y0   ; address of y(n)
A_YE:     EQU    P6DM+YE   ; address of y(n-NCZ+1)
A_XPE:    EQU    P6DM+XPE  ; address of y(n-NWZ+1) for x'(n)
;*
;*-----------------------------------------------------------------------
;*
;* Memory map:  y(n)      low address  <-- AY0
;*              y(n-1)
;*              ...
;*              y(n-N+1)  high address <-- AYE
;*                                     <-- AR1
;*
;*-----------------------------------------------------------------------
;*
;* PAGE 7 (B1) DATA MEMORY ALLOCATION for W(z)
;*           x(n-i), i=0,1,...,NWZ-1
;*
P7DM:     EQU    896       ; PAGE 7 DATA MEMORY ADDRESS
X0:       EQU    0         ; x(n)
XE:       EQU    NWZ-1     ; x(n-NWZ+1)
XCE:      EQU    NCZ-1     ; X(n-NCZ+1)
A_X0:     EQU    P7DM+X0   ; address of x(n)
A_XE:     EQU    P7DM+XE   ; address of x(n-NWZ+1)
A_XCE:    EQU    P7DM+XCE  ; address of x(n-NCZ+1)
;*
;*-----------------------------------------------------------------------
;*
;* Memory map:      x(n)      ; <-- AX0
;*                  x(n-1)
;*                  ...
;*                  x(n-N+1)  ; <-- AXE
;*
;***********************************************************************
```

```
;*
;*        INTERRUPT BRANCHES
;*
;*****************************************************************************
;*
RESET:    B      32     ; On hardware reset go to INIT
          ORG    4
INT1:     B      1000   ; On INT1 go to interrupt 1 service routine
          ORG    6
INT2:     B      2000   ; On INT2 go to interrupt 2 service routine
          ORG    26
RCV:      B      3000   ; On RINT go to ADC service routine
;*
;*****************************************************************************
;*
;*       PROCESSOR INITIALIZATION ROUTINE
;*
;*****************************************************************************
;*
INIT:  ORG    32
       LDPK   0      ; Load page 0
       LALK   2E00H  ; 0010 1110 0000 0000 in binary
       SACL   TEMP   ; Initialize ST0
       LST    TEMP   ; 0 –> DP
                      ; 1 –> INTM, interrupts disabled
                      ; 1 –> OVM
                      ; 0 –> OV
                      ; 1 –> ARP

       SPM    1      ; P reg. output shift left 1 bit

;*
;*     INITIALIZE PAGE 0
;*
       LARP   AR1       ; AR1 as address pointer
       LARK   AR1,96    ; LOWEST PAGE 0 LOCATION –> AR1
       ZAC              ; 0 –> ACC
       RPTK   31        ; REPEAT NEXT INSTRUCTION 32 TIMES
       SACL   *+        ; ZERO PAGE 0

       LACK   1         ; ACC <– 1
       SACL   ONE       ; ONE <– 1
       SACL   STSFLG    ; STSFLG=1, training mode first

       LALK   TRNTIM    ; ACC <– # of training samples
       SACL   TRNCTR    ; training time = 3 seconds = 4500 samples

       LALK   MU        ; ACC <– mu
       SACL   VMU       ; value of mu

       LALK   MU1       ; ACC <– mu1
       SACL   VMU1      ; value of mu1

       LALK   22        ; ACC <– 22
       SACL   IMR       ; Enable INT1 , INT2 & RINT

       LALK   SEED      ; ACC <– SEED
       SACL   WN0       ; initial white noise = seed

;*     INITIALIZE ADDRESS POINTERS

       LALK   DA_CE
       SACL   ADCE      ; address of CN–1(n) in ADCE

       LALK   DA_WE
       SACL   ADWE      ; address of wN–1(n) in ADWE
       LALK   A_Y0
       SACL   AY0       ; address of y(n)
```

```
        LALK   A_YE
        SACL   AYE       ; address of y(n–N+1)
        LALK   A_XPE
        SACL   AXPE      ; address of x'(n–N+1)
        LALK   A_X0
        SACL   AX0       ; address of x(n)
        LALK   A_XE
        SACL   AXE       ; address of x(n–N+1)
        LALK   A_XCE
        SACL   AXCE      ; address of x(n–NCZ+1)
;*      CLEAR PAGES 4, 5, 6, and 7
        LARP   AR1       ; 1 –> ARP
        LRLK   AR1,512   ; LOWEST PAGE 4 ADDRESS –> AR1
        ZAC              ; 0 –> ACC
        RPTK   255
        SACL   *+        ; zero pages 4 & 5
        RPTK   255
        SACL   *+        ; zero pages 6 & 7

        EINT             ; ENABLE INTERRUPTS
    B   LOOP             ; Branch to LOOP,wait for interrupts
;*
;*****************************************************************************
;*
;*      CYCLE START ROUTINE
;*
;*****************************************************************************
;*
START   LARP   AR3       ; 3 –> ARP
        LAC    BUF1      ; ACC <– from input port A
        SACL   EN        ; value of error signal from error microphone
        LAC    BUF2      ; ACC <– from input port B
        SACL   XN        ; value of reference signal from input microphone
        LAC    YN        ; ACC <– YN
        SACL   BUF3      ; value of antinoise signal to output port A
        LAC    TN        ; ACC <– TN
        SACL   BUF4      ; value of concerned signal to output port B
        LAC    STSFLG    ; If STSFLG = 0, branch to CANCEL
        BZ     CANCEL    ; for next sample, loop forever
;*
;*****************************************************************************
;*
;*      Training mode (off-line modeling of secondary path)
;*
;*-----------------------------------------------------------------------------
;*
;*      White noise generator:
;*      Algorithm:
;*
;*                rotate left 1-bit
;*    b15 . b11         <----       b2 . b0 <------------------|
;*     |     |                       |    |->                 |
;*     |     |                       |------> XOR ->|          |
;*     |     |                                    |          |
;*     |     |----------->                        |->        |
;*     |-----------------> XOR ----------------------> XOR -->|
;*
;*      where the initial value of WN0 (b15..b0) = seed
;*
;*-----------------------------------------------------------------------------
;*
```

```
TRAIN  LAC    WN0       ; Load noise sequence
       ANDK   MASK      ; Mask off feedback bits
       SACL   WN1       ; Save temporary
       ADD    WN1,4     ; combine bits 11 and 15
       ADD    WN1,13    ; combine bit 2 with result
       ADD    WN1,15    ; combine bit 0 with result
       ANDK   MASK      ; reuse mask to mask off MSB
       ADDH   WN0       ; combine MSB with sequence
       SACH   WN0,1     ; save result (and shift out MSB)
       LAC    WN0,11    ; scale WN0
       SACH   YN        ; output white noise to excite secondary path
;*
;*****************************************************************************
;*
;*  Adaptive Off-line Secondary-Path Modeling
;*
;*                |---> Speaker  ...  error MIC --->|
;*      y(n) --->|                                  |
;*                |    ----------z(n)-     +        |
;*                |--->|  C(z)  |---->(sum)<--------|e(n)
;*                     ----------       |
;*                          :           |
;*                          :..<......... e'(n)
;*
;*     where
;*        y(n) is internally generate white noise
;*
;*****************************************************************************
;*
;*                  NCZ-1
;*  A. Computes z(n) =  sum  ci(n) * y(n-i)
;*                    i=0
;*
;*-----------------------------------------------------------------------------
;*
       LARP   AR1           ; AR1 as current address reg
FIR_C  MPYK   0             ; P=0
       LAR    AR1,AY0       ; AR1 pointing to y(n)
       SACH   *,AR1         ; inject white noise to buffer of C(z)
       LAR    AR1,AYE       ; AR1 pointing to y(n-NCZ+1)
       LAC    ONE,15        ; round-off offset to ACC
       CNFP
       RPTK   NCZ-1         ; for i = NCZ-1,NCZ-2,...,0
       MACD   PA_CE,*-      ; ci(n) * y(n-i) + ACC -> ACC
       CNFD                 ; also move data y(n-i)
       APAC                 ; P + ACC -> ACC = z(n)
;*
;*-----------------------------------------------------------------------------
;*
;*  B. computes error signals e'(n):
;*
;*     e'(n) = e(n) - z(n)
;*
;*-----------------------------------------------------------------------------
;*
       NEG              ; ACC = - z(n)
       ADDH   EN        ; ACC (e'(n) = e(n) - z(n)
       SACH   ZN        ; z(n) = e'(n)
;*
;*-----------------------------------------------------------------------------
;*
;*  C. update coefficients of C(z) using LMS Algorithm:
;*
```

```
;*        ci(n+1) = ci(n) + u*e'(n)*y(n-i)
;*
;*        AR1 - point to data buffer, y(n-i)
;*        AR2 - point to AF coefs,   ci(n)
;*
;*-----------------------------------------------------------------------------
;*
       LT     ZN            ; T = e'(n)
       MPY    VMU           ; P = mu*e'(n)
       PAC                  ; ACC <- P
       ADD    ONE,15        ; rounding
       SACH   ZN            ; ZN = mu*e'(n)
;*
       LARK   AR3,NCZ-1 ; initialize AR3 as loop counter
       LAR    AR1,AYE
       MAR    *+            ; AR1 pointing to y(n-N+1) due to
                            ; MACD data move effect
       LAR    AR2,ADCE      ; AR2 pointing to cN-1(n)
       LT     ZN            ; T = mu*e'(n)
       MPY    *-,AR2        ; P = mu*e'(n)*y(n-i)
;*
ADAP_C ZALR   *,AR1         ; load ACC with ci(n) and round
       MPYA   *-,AR2        ; ci(n+1) = ci(n) + P
                            ; P = mu*e'(n)*y(n-i) for next i
       SACH   *+,0,AR3      ; store ci(n+1)
       BANZ   ADAP_C,*-,AR2 ; go back to loop if counter (AR3) > 0
;*
;*-----------------------------------------------------------------------------
;*
;* check if end of training mode
;*
;*-----------------------------------------------------------------------------
;*
       LAC    TRNCTR        ; ACC <- training counter
       SUBK    1             ; decrement training counter
       SACL    TRNCTR       ; save counter
       BGZ    LOOP
;*
       ZAC                  ; end of training mode, ACC=0
       SACL   STSFLG        ; STSFLG <- 0, now in noise control mode
       LARP   AR1           ; make sure AR1 is address pointer
       LAR    AR1,AY0       ; AR1 pointing to y(n)
       RPTK   NCZ           ; repeat NCZ+1 time
       SACL   *+            ; clear C(z) buffer for x'(n) in noise
                            ; control mode
       B      START         ; end training, go to noise control mode
;*
;*****************************************************************************
;*
;*  Noise control mode:
;*
;*                              to speaker      from error MIC
;* from input MIC   ----------  y(n)            e(n)
;*   x(n) --------->|  W(z)  |----->             |
;*            |     ----------                   |
;*         -------       :                       |
;*         | C(z)|       :                       |
;*         -------   ----:----                   |
;*            |----->| FXLMS |<-------------------
;*           x'(n)   ---------
;*
;*****************************************************************************
;*
```

```
CANCEL LAC    XN                  ; inject reference input x(n) from port B
                                  ; (input MIC) into x(n)
       LARP   AR1                  ; ARP <– 1
       LAR    AR1,AX0              ; AR1 point to x(n)
       SACL   *
       LARP   AR1
;*
;*-----------------------------------------------------------------------------
;*
;* 1. Filtered-X, filtering x(n) by C(z) to get x'(n)
;*
;*              N–1
;*     x'(n) = sum ci * x(n–i)
;*              i=0
;*
;*    where ci, i=0,1,..,N–1 are from training mode filter C(z)
;*
;*-----------------------------------------------------------------------------
;*
FX C      MPYK   0                ; P <– 0
   LAR    AR1,AXCE                ; AR1 point to x(n–NCZ+1)
   LAC    ONE,15                  ; rounding
   CNFP
   RPTK   NCZ–1                   ; for i=N–1, .., 1, 0
   MAC    PA_CE,*–                ; ACC <– ACC+ci(n)*x(n–i)
   CNFD
   APAC                            ; ACC <– ACC+P = x'(n)
   LAR    AR1,AY0                 ; AR1 point to x'(n)
   SACH   *                       ; inject x'(n) into buffer
;*
;*-----------------------------------------------------------------------------
;*
;*  2. Filter x(n) by W(z) to get y(n), the antinoise
;*
;*             N–1
;*     y(n) = sum wi(n) * x(n–i)
;*             i=0
;*
;*-----------------------------------------------------------------------------
;*
FIR_W  MPYK   0                   ; P <– 0
       LAR    AR1,AXE              ; AR1 point to x(n–N+1)
       LAC    ONE,15               ; rounding
       CNFP
       RPTK   NWZ–1                ; for i=N–1, .., 1, 0
       MACD   PA_WE,*–             ; ACC = ACC + wi(n)*x(n–i)
       CNFD
       APAC                        ; ACC = y(n)
;*
   SACH   YN                       ; YN <– y(n), antinoise to output port A
;*
;*-----------------------------------------------------------------------------
;*
;* 3. FXLMS Algorithm to update coefficients of W(z)
;*
;*         wi(n+1) = wi(n) – u e(n) x'(n–i), for i=0,1,..,N–1
;*
;*  Note: using "–" when updating W(z) in noise control mode
;*     instead of "+" when updating C(z) in training mode
;*
;*     AR1 pointing to x'(n–i) data buffer
;*     AR2 pointing to wi(n) coefs buffer
;*
```

```
;*-----------------------------------------------------------------------------
;*
       LT     EN                ; T = e(n)
       MPY    VMU1              ; P = mu * e(n)
       PAC                      ; ACC <- P
       ADD    ONE,15            ; rounding
       SACH    EN               ; EN = mu * e(n)
;*
       LARK   AR3,NWZ-1         ; initialize AR3 as loop counter
       LAR    AR1,AXPE          ; AR1 point to x'(n-N+1)
       LAR    AR2,ADWE          ; AR2 point to wN-1(n)
;*
       LT     EN                ; T = mu * e(n)
       LARP   AR1
       MPY    *-,AR2            ; P = mu * e(n) * x'(n-N+1)
ADAP_W ZALR   *,AR1             ; ACC <- wi(n) and rounding
       MPYS   *,AR1             ; wi(n+1) = wi(n) - P
                                ; P = mu * e(n) * x'(n-i) for next i
       DMOV   *-,AR2            ; update x'(n) buffer
       SUB    *,LEAKY           ; wi(n+1)=wi(n+1)-(2exp(LEAKY-16))*wi(n+1)
       SACH   *+,0,AR3          ; wi(n+1) <- ACC
       BANZ   ADAP_W,*-,AR2     ; go back to loop if (AR3) > 0

;*
       B      LOOP

LOOP:  IDLE
       IDLE
       B      LOOP
;*
;*****************************************************************************
;*
;*     INT1 Interrupt Service Routine
;*
;*****************************************************************************
;*
       ORG    1000              ; Channel A interrupt
       SST    TST0              ; Save ST0
       LDPK   0
       OUT    BUF3,DAC          ; Output buffer to DAC
       LST    TST0              ; Restore ST0
       NOP
       SXF                      ; Set external flag bit
       EINT
       RET
;*
;*****************************************************************************
;*
;*        INT2 Interrupt Service Routine
;*
;*****************************************************************************
;*
       ORG    2000              ; Channel B interrupt
       SST    TST0              ; Save ST 0
       LDPK    0
       OUT    BUF4,DAC          ; Output buffer to DAC
       LST    TST0              ; Restore ST0
       NOP
       RXF                      ; Reset external flag bit
       EINT
       B      START
;*
;*****************************************************************************
;*
```

```
;*      RINT Interrupt Service Routine
;*
;*****************************************************************************
;*
       ORG    3000          ; ADC interrupt
       LDPK   0
       SST    TST0          ; Save ST0
       LAC    ADC           ; Read from serial port
       BIOZ   IN2           ; Skip if channel B
       SACL   BUF1          ; Move data to channel A buffer
       LST    TST0          ; Restore ST0
       EINT
       RET

IN2:   SACL   BUF2          ; For channel B, save data in channel B buffer
       LST    TST0          ; Restore ST0
       EINT
       RET
```

Filtered-U RLMS Algorithm

```
;*
;* FILE NAME : FURLMS.ASM
;*****************************************************************************
;*
;*            This program has been modified to run on the Ariel's DSP-16
;*            Plus TMS320C25 DSP Board
;*                                                  Kai-Ming Chung
;*                                                  Oct. 1994
;*
;*****************************************************************************
;*
;*   File: FURLMS.ASM
;*
;*   One-dimensional Adaptive Active Noise Control System Using
;*   Filtered-U RLMS (FURLMS) Algorithm on TMS320C25
;*
;*   Sen M. Kuo , Fall 1993
;*
;*****************************************************************************
;*
;*   SYSTEM CONFIGURATION:
;*
;*         x(n)
;*   M1  *|---->|-----------------------|
;*              |                       |
;*              |  Adaptive FURLMS ANC  | y(n)
;*         e(n) |        System         |------> antinoise output
;*   M2  *|---->|                       |
;*              |-----------------------|
;*
;*   where
;*         x(n) - signal from the input microphone
;*         e(n) - signal from the error microphone
;*         y(n) - antinoise to drive the canceling loudspeaker, or
;*                the training signal in the off-line modeling mode
;*
;*   Ports:
;*           x(n) from input port B
;*           e(n) from input port A
;*           y(n) to output  port A
;*
;*   The flow of program is:
```

```
;*
;*   C25 initialization --> off-line Secondary-Path modeling -->
;*   on-line active noise control
;*
;*   A. Off-line modeling:
;*
;*                |---> Speaker  ....  error MIC --->|
;*      y(n) ----|                                   |
;*                |    ----------z(n) -     +        |
;*                |--->|  C(z)  |----->(sum)<--------|e(n)
;*                     ----------        |
;*                          :..<.......... e'(n)
;*   where
;*         y(n) is an internally generated white noise
;*         C(z) is an adaptive modeling filter, updated by LMS
;*
;*
;*   B. On-line noise control:
;*
;*                                              to speaker   from error MIC
;* from input MIC   --------                    y(n) |             |e(n)
;*   x(n) --------->| A(z) |---->(+)--------------|             |
;*            |     --------      |               |             |
;*         -------      :         |   --------    |             |
;*         | C(z)|      :         |<--| B(z) |<---|             |
;*         -------      :             -------- --------          |
;*            |         :               :     | C(z) |          |
;*            |      -------            :     --------          |
;*            |----->| LMS |          -------     |             |
;*           x'(n)   -------          | LMS |<----|             |
;*                      |             -------                    |
;*                      |                |                       |
;*                       ----------------------------------------
;*
;*    where C(z) is a fixed FIR filter from previous modeling mode
;*          A(z) and B(z) are adaptive noise control filters,
;*          updated by FURLMS
;*
;*****************************************************************************
;*
;*    define constants (can be modified for different applications)
;*
;*-----------------------------------------------------------------------------
;*
FS:       EQU    2000          ; assume sampling rate is 2 kHz.
TRNTIM:   EQU    30000         ; training time = 15 seconds
NCZ:      EQU    63            ; order of C(z), max = 63
NAZ:      EQU    63            ; order of A(z), max = 63
NBZ:      EQU    63            ; order of B(z), max = 63
MU:       EQU    4096          ; coef. update stepsize for off-line modeling
MU1:      EQU    328           ; coef. update stepsize for W(z)
MASK:     EQU    8805h         ; to mask off bits 0,2,11,15
SEED:     EQU    12357         ; seed for white noise generator
LEAKY:    EQU    2             ; leaky factor
;*
;*****************************************************************************
;*
;*  Memory map:
;*  B2: page 0   - data I/O buffer
;*
;*  B0: page 4   - coefs of AF C(z) : ci(n),   i=0,1,..,(NCZ-1)
;*                 (modeling filter)            NCZ <= 63
;*
```

```
;*              – coefs of AF A(z) : ai(n),   i=0,1,..,(NAZ–1)
;*               (direct filter)              NAZ <= 63
;*
;*   B0: page 5  – coefs of AF B(z) : bi(n),   i=0,1,..,(NBZ–1)
;*               (feedback filter)            NBZ <= 63
;*
;*   B1: page 6  – data buffer for C(z)
;*
;*        During off-line modeling, buffer contains training
;*        signal y(n–i), i=0,1,..,(NCZ–1)
;*
;*        During on-line canceling, buffer contains:
;*           1. filtered version of x(n) by C(z), i.e., x'(n)
;*           2. filtered version of y(n) by C(z), i.e., y'(n)
;*
;*   B1: page 7 – data buffer for A(z) : x(n–i), i=0,1,..,(NAZ–1)
;*               data buffer for B(z) : y(n–i), i=1,2,..,NBZ
;*
;*****************************************************************************
;*
;*         PAGE 0 (memory-mapped regs and B2) DATA MEMORY ALLOCATION
;*
;*-----------------------------------------------------------------------------
;*
P0DM:     EQU    0         ; page 0 data RAM address
DAC:      EQU    13        ; DAC I/O port
ADC:      EQU    0         ; Serial receive address
IMR:      EQU    4         ; Interrupt mask register
BUF1:     EQU    96        ; Buffer for channel A input data
BUF2:     EQU    97        ; Buffer for channel B input data
BUF3:     EQU    98        ; Buffer for channel A output data
BUF4:     EQU    99        ; Buffer for channel B output data
WN0:      EQU    100        ; storage for white noise generator
WN1:      EQU    101       ;            ″
TRNCTR:   EQU    102    ; training mode (0ff-line modeling) counter
STSFLG:   EQU    103       ; program status flag, 1 = training mode
ONE:      EQU    104       ; HOLDS 1
VMU:      EQU    105       ; value of mu
VMU1:     EQU    106       ; value of mu1
ADCE:     EQU    107       ; address of cN–1(n)
ADAE:     EQU    108       ; address of aN–1(n)
ADBE:     EQU    109       ; address of bN–1(n)
AXP0:     EQU    110       ; address of x'(n)
AXPE:     EQU    111       ; address of x'(n–N+1)
AYP0:     EQU    112       ; address for y'(n)
AYPE:     EQU    113       ; address for y'(n–N+1)
AX0:      EQU    114       ; address for x(n)
AXE:      EQU    115       ; address for x(n–N+1)
AXCE:     EQU    116       ; address for x(n–NCZ+1)
AY0:      EQU    117       ; address for y(n)
AYE:      EQU    118       ; address for y(n–N+1)
AYCE:     EQU    119       ; address for y(n–NCZ+1)

;* data buffer for signals
XN:       EQU    120       ; x(n), from input MIC
EN:       EQU    121       ; e(n), from error MIC
ZN:       EQU    122       ; z(n) in training mode
YN:       EQU    123       ; buffer for y(n)
TN:       EQU    124       ; to output port B
TEMP:     EQU    125       ; Temporary storage for channel A
TST0:     EQU    126       ; Storage for ST0
;*
;*-----------------------------------------------------------------------------
```

```
;*
;*  PAGE 4 (B0) DATA MEMORY ALLOCATION
;*        AF C(z): ci(n),  i=0,1,2,..,NCZ–1
;*
P4DM:     EQU    512           ; PAGE 4 DATA MEM ADRS
P4PM:     EQU    65280         ; PAGE 4 PROG MEM ADRS
CE:       EQU    0             ; ci(n), end of C(z) buffer
DA_CE:    EQU    P4DM+CE       ; address of ci(n), i=NCZ–1 at CNFD
PA_CE:    EQU    P4PM+CE       ; address of ci(n),     ″  at CNFP
;*
;*-------------------------------------------------------------------------
;*
;*  Memory map:   cN–1(n)       low address <–– PA_CE, AR2, ADCE
;*                cN–2(n)
;*                ...
;*                c1(n)
;*                c0(n)          high address
;*
;*-------------------------------------------------------------------------
;*
;*   AF A(z): ai(n), i=0,1,2,..,NAZ–1
;*
AE:       EQU    64            ; ai(n), end of A(z) buffer
DA_AE:    EQU    P4DM+AE       ; address of ai(n), i=NAZ–1 at CNFD
PA_AE:    EQU    P4PM+AE       ; address of ai(n),     ″  at CNFP
;*
;*-------------------------------------------------------------------------
;*
;*  Memory map:   aN–1(n)       low address <–– PA_AE, ADAE
;*                aN–2(n)
;*                ...
;*                a1(n)
;*                a0(n)         high address
;*
;*-------------------------------------------------------------------------
;*
;*  PAGE 5 (B0) DATA MEMORY ALLOCATION
;*        AF B(z): bi(n),  i=0,1,2,..,NBZ–1
;*
P5DM:     EQU    640           ; PAGE 5 DATA MEM ADRS
P5PM:     EQU    65408         ; PAGE 5 PROG MEM ADRS
BE:       EQU    0             ; bi(n), end of B(z) buffer
DA_BE:    EQU    P5DM+BE       ; address of BE at CNFD
PA_BE:    EQU    P5PM+BE       ; address of BE at CNFP
;*
;*-------------------------------------------------------------------------
;*
;*  Memory map:      bN–1(n)       low address, ADBE
;*                   bN–2(n)
;*                   ...
;*                   b1(n)
;*                   b0(n)         high address
;*
;*-------------------------------------------------------------------------
;*
;*     PAGE 6 (B1) DATA MEMORY ALLOCATION for C(z)
;*            x'(n–i), i=0,1,...,NCZ–1
;*            y'(n–i), i=0,1,...,NCZ–1
;*
P6DM:     EQU    768           ; PAGE 6 DATA MEM ADRS
XP0:      EQU    0             ; x'(n)
XPE:      EQU    NCZ–1         ; y(n–NCZ+1)
A_XP0:    EQU    P6DM+XP0      ; address of x'(n)
```

```
A_XPE:    EQU    P6DM+XPE     ; address of x'(n-NCZ+1)
;*
;*-----------------------------------------------------------------------
;*
;* Memory map:      x'(n)            low address <-- AXP0
;*                  x'(n-1)
;*                  ...
;*                  x'(n-N+1)        high address <-- AXPE
;*
;*-----------------------------------------------------------------------
;*
YP0:      EQU    64        ; y'(n)
YPE:      EQU    YP0+NCZ-1 ; y'(n-NCZ+1)
A_YP0:    EQU    P6DM+YP0  ; address of y'(n)
A_YPE:    EQU    P6DM+YPE  ; address of y'(n-NCZ+1)
;*
;*-----------------------------------------------------------------------
;*
;* Memory map:      y'(n)            low address <-- AYP0
;*                  y'(n-1)
;*                  ...
;*                  y'(n-N+1)        high address <-- AYPE
;*
;*-----------------------------------------------------------------------
;*
;* PAGE 7 (B1) DATA MEMORY ALLOCATION for A(z) and B(z)
;*                  x(n-i), i=0,1,...,NAZ-1
;*                  y(n-i), i=0,1,...,NBZ-1
;*
P7DM:     EQU    896          ; PAGE 7 DATA MEMORY ADDRESS
X0:       EQU    0            ; address of x(n)
XE:       EQU    X0+NAZ-1     ; x(n-NAZ+1)
XCE:      EQU    X0+NCZ-1     ; x(n-NCZ+1)
A_X0:     EQU    P7DM+X0      ; address of x(n)
A_XE:     EQU    P7DM+XE      ; address of x(n-NAZ+1)
A_XCE:    EQU    P7DM+XCE     ; address of x(n-NCZ+1)
;*
;*-----------------------------------------------------------------------
;*
;* Memory map:      x(n)      ; <-- AX0
;*                  x(n-1)
;*                  ...
;*                  x(n-N+1)  ; <-- AXE
;*
;*-----------------------------------------------------------------------
;*
Y0:       EQU    64           ; address of y(n)
YE:       EQU    Y0+NBZ-1     ; y(n-NBZ+1)
YCE:      EQU    Y0+NCZ-1     ; y(n-NCZ+1)
A_Y0:     EQU    P7DM+Y0      ; address of y(n)
A_YE:     EQU    P7DM+YE      ; address of y(n-NBZ+1)
A_YCE:    EQU    P7DM+YCE     ; address of y(n-NCZ+1)
;*
;*-----------------------------------------------------------------------
;*
;* Memory map:      y(n)         ; <-- AY0
;*                  y(n-1)
;*                  ...
;*                  y(n-N+1)     ; <-- AYE
;*
;***********************************************************************
;*
;*        INTERRUPT BRANCHES
```

```
;*
;*****************************************************************************
;*
RESET: B      32               ; On hardware reset go to INIT
       ORG    4
INT1:  B      1000             ; On INT1 go to interrupt 1 service routine
       ORG    6
INT2:   B     2000          ; On INT2 go to interrupt 2 service routine
       ORG    26
RCV:   B      3000             ; On RINT go to ADC service routine
;*
;*****************************************************************************
;*
;*        PROCESSOR INITIALIZATION ROUTINE
;*
;*****************************************************************************
;*
INIT:   ORG      32
       LDPK      0             ; Load page 0
       LALK      2E00H         ; 0010 1110 0000 0000 in binary
       SACL      TEMP          ; Initialize ST0
       LST       TEMP          ; 0 –> DP
                               ; 1 –> INTM, interrupts disable
                               ; 1 –> OVM
                               ; 0 –> OV
                               ; 1 –> ARP
       SPM       1             ; P reg. output shift left 1 bit
;*
;*        INITIALIZE PAGE 0
;*
       LARP      AR1         ; AR1 as address pointer
       LARK      AR1,96    ; LOWEST PAGE 0 LOCATION –> AR1
       ZAC                 ; 0 –> ACC
       RPTK      31        ; REPEAT NEXT INSTRUCTION 32 TIMES
       SACL      *+        ; ZERO PAGE 0
       LACK      1          ; ACC <– 1
       SACL      ONE        ; ONE <–– 1
       SACL      STSFLG    ; STSFLG=1, training mode first
       LALK      TRNTIM    ; ACC <– # of training samples
       SACL      TRNCTR    ; training time = 3 sec = 4500 samples
       LALK      MU        ; ACC <– mu
       SACL      VMU       ; value of mu
       LALK      MU1       ; ACC <– mu1
       SACL      VMU1      ; value of mu1
       LALK      SEED      ; ACC <– SEED
       SACL      WN0       ; initial white noise = seed
       LALK      22        ; ACC <– 22
       SACL      IMR       ; Enable INT1, INT2, & RINT
;* initialize address pointers
       LALK      DA_CE
       SACL      ADCE      ; address of cN–1(n) in ADCE
       LALK      DA_AE
       SACL      ADAE      ; address of aN–1(n) in ADAE
       LALK      DA_BE
       SACL      ADBE      ; address of bN–1(n) in ADBE
       LALK      A_XP0
       SACL      AXP0      ; address of x’(n)
       LALK      A_XPE
       SACL      AXPE      ; address of x’(n–N+1)
       LALK      A_YP0
       SACL      AYP0      ; address of y’(n)
```

```
        LALK      A_YPE
        SACL      AYPE      ; address of y'(n-N+1)  LALK    A_Y0
        SACL      AY0       ; address of y(n)
        LALK      A_YE
        SACL      AYE       ; address of y(n-N+1)
        LALK      A_YCE
        SACL      AYCE      ; address of y(n-NCZ+1)
        LALK      A_X0
        SACL      AX0       ; address of x(n)
        LALK      A_XE
        SACL      AXE       ; address of x(n-N+1)
        LALK      A_XCE
        SACL      AXCE      ; address of x(n-NCZ+1)

;*      CLEAR PAGES 4, 5, 6, and 7

        LARP      AR1
        LRLK      AR1,512   ; LOWEST PAGE 4 ADDRESS -> AR1
        ZAC                 ; 0 -> ACC
        RPTK      255
        SACL      *+        ; zero page 4 & 5
        RPTK      255
        SACL      *+        ; zero page 6 & 7

        EINT                ; ENABLE INTERRUPTS
        B         LOOP
;*
;*******************************************************************************
;*
;*      CYCLE START ROUTINE
;*
;*******************************************************************************
;*
START     LARP      AR3    ; 3 -> ARP
          LAC       BUF1   ; ACC <- from input port A
          SACL      EN     ; value of error signal from error microphone
          LAC       BUF2   ; ACC <- from inpit port B
          SACL      XN     ; value of reference signal from input microphone
          LAC       YN     ; ACC <-YN
          SACL      BUF3   ; value of anti-noise signal to output port A
          LAC       TN     ; ACC <- TN
          SACL      BUF4   ; value of concerned signal to output port B

          LAC       STSFLG ; If STSFLG = 0, branch CANCEL
          BZ        CANCEL ; for next sample, loop forever
;*
;*******************************************************************************
;*
;*  Training mode (off-line modeling of error path)
;*
;*-----------------------------------------------------------------------------
;*
;*  White noise generator:
;*  Algorithm:
;*
;*               rotate left 1-bit
;*   b15 . b11         <----       b2 . b0 <-------------------|
;*    |     |                       |    |->                  |
;*    |     |                       |------> XOR ->|           |
;*    |     |                                    |            |
;*    |     |----------->                        |->          |
;*    |-----------------> XOR ------------------------> XOR -->|
;*
;*   where the initial value of WN0 (b15..b0) = seed
;*
```

```
;*-----------------------------------------------------------------------
;*
TRAIN  LAC       WN0       ; Load noise sequence
       ANDK      MASK      ; Mask off feedback bits
       SACL      WN1       ; Save temporary
       ADD       WN1,4     ; combine bits 11 and 15
       ADD       WN1,13    ; combine bit 2 with result
       ADD       WN1,15    ; combine bit 0 with result
       ANDK      MASK      ; re-use mask to mask off MSB
       ADDH      WN0       ; combine MSB with sequence
       SACH      WN0,1     ; save result (and shift out MSB)
       LAC       WN0,11    ; scale WN0
       SACH      YN        ; output white noise to excite secondary path
;*
;***********************************************************************
;*
;*   Adaptive Off-line Secondary-Path Modeling
;*
;*                |---> Speaker  ...  error MIC --->|
;*      y(n) --->|                                  |
;*                |    ----------z(n)-     +        |
;*                |--->|  C(z)  |---->(sum)<--------|e(n)
;*                     ----------       |
;*                          :           |
;*                          :..<......... e'(n)
;*
;*   where
;*         y(n) is internally generate white noise
;*
;***********************************************************************
;*
;*                      NCZ-1
;*  A. Computes z(n) =  sum  ci(n) * y(n-i)
;*                       i=0
;*
;*-----------------------------------------------------------------------
;*
       LARP      AR1           ; AR1 as current address reg
FIR_C  MPYK      0             ; P=0
       LAR       AR1,AY0       ; AR1 pointing to y(n)
       SACH      *,AR1         ; inject white noise to buffer of C(z)
       LAR       AR1,AYCE      ; AR1 pointing to y(n-NCZ+1)
       LAC       ONE,15        ; round-off offset to ACC
       CNFP
       RPTK      NCZ-1         ; for i = NCZ-1,NCZ-2,...,0
       MACD      PA_CE,*-      ; ci(n) * y(n-i) + ACC -> ACC
       CNFD                    ; also move data y(n-i)
       APAC                    ; P + ACC -> ACC = z(n)
;*
;*-----------------------------------------------------------------------
;*
;*  B. computes error signals e'(n):
;*
;*       e'(n) = e(n) - z(n)
;*
;*-----------------------------------------------------------------------
;*
   NEG                         ; ACC = - z(n)
   ADDH      EN                ; ACC (e'(n)) = e(n)-z(n)
   SACH      ZN                ; z(n) = e'(n)
;*
;*-----------------------------------------------------------------------
;*
```

```
;* C. update coefficients of C(z) using LMS Algorithm:
;*         ci(n+1) = ci(n) + u*e'(n)*y(n–i)
;*         AR1 – point to data buffer, y(n–i)
;*         AR2 – point to AF coefs,     ci(n)
;*
;*-----------------------------------------------------------------------------
;*
       LT        ZN            ; T = e'(n)
       MPY       VMU           ; P = mu*e'(n)
       PAC                      ; ACC <– P
       ADD       ONE,15        ; rounding
       SACH      ZN            ; ZN = mu*e'(n)
;*
       LARK      AR3,NCZ–1     ; initialize AR3 as loop counter
       LAR       AR1,AYCE
       MAR       *+            ; AR1 pointing to y(n–N+1) due to
                               ; MACD data move effect
       LAR       AR2,ADCE      ; AR2 pointing to cN–1(n)
       LT        ZN            ; T = mu*e'(n)
       MPY       *–,AR2        ; P = mu*e'(n)*y(n–i)
;*
ADAP_C ZALR      *,AR1         ; load ACC with ci(n) and round
       MPYA      *–,AR2        ; ci(n+1) = ci(n) + P
                               ; P = mu*e'(n)*y(n–i) for next i
       SACH      *+,0,AR3      ; store ci(n+1)
       BANZ      ADAP_C,*–,AR2 ; go back to loop if counter (AR3) > 0
;*
;*-----------------------------------------------------------------------------
;*
;* check if end of training mode
;*
;*-----------------------------------------------------------------------------
;*
       LAC    TRNCTR           ; ACC <– training counter
       SUBK   1                ; decrement training counter
       SACL   TRNCTR           ; save counter
       BGZ    LOOP
;*
       ZAC                     ; end of training mode, ACC=0
       SACL   STSFLG           ; STSFLG <– 0, now in noise control mode
       LARP   AR1              ; make sure AR1 is address pointer
       LAR    AR1,AY0          ; AR1 pointing to y(n)
       RPTK   NCZ              ; repeat NCZ+1 time
       SACL   *+                ; clear C(z) buffer for x'(n) in noise
                               ; control mode
       B      START            ; end training, go to noise control mode
```

```
;*****************************************************************************
;*
;*   Noise control mode:
;*
;*                                                  to speaker   from error MIC
;* from input MIC   -------- z(n)                   y(n) |             |e(n)
;*   x(n) --------->| A(z) |---->(+)--------------|             |
;*            |     --------      |               |             |
;*         -------      :    z'(n)|   --------    |             |
;*         | C(z)|      :         |<--| B(z) |<---|             |
;*         -------      :             -------- --------         |
;*            |         :                :     | C(z) |         |
;*            |      -------             :     --------         |
;*            |----->| LMS |          -------     |             |
;*           x'(n)   -------          | LMS |<----|             |
;*                      |              -------                   |
;*                      |                |                      |
;*                      ----------------------------------------
;*
;*****************************************************************************
;*
CANCEL LAC    XN
       LARP   AR1                 ; ARP <- 1
       LAR    AR1,AX0             ; AR1 point to x(n)
       SACL   *
;*
;*-----------------------------------------------------------------------------
;*
;* 1. Filtered-U:
;*
;*    filtering x(n) by C(z) to get x'(n)
;*
;*             N-1
;*     x'(n) = sum ci * x(n-i)
;*             i=0
;*
;*     where ci, i=0,1,...,N-1 are from training mode filter C(z)
;*
;*-----------------------------------------------------------------------------
;*
FX_C   MPYK   0            ; P <- 0
       LAR    AR1,AXCE     ; AR1 point to x(n-N+1)
       LAC    ONE,15       ; rounding
       CNFP
       RPTK   NCZ-1        ; for i=N-1, .., 1, 0
       MAC    PA_CE,*-     ; ACC <- ACC+ci(n)*x(n-i)
       CNFD
       APAC                ; ACC <- ACC+P = x'(n)
       LAR    AR1,AXP0     ; AR1 point to x'(n)
       SACH   *            ; inject x'(n) into buffer
;*
;*-----------------------------------------------------------------------------
;*
;*  2. Filter x(n) by A(z) to get z(n)
;*
;*            N-1
;*     z(n) = sum ai(n) * x(n-i)
;*            i=0
;*
;*-----------------------------------------------------------------------------
;*
FIR_A  MPYK   0            ; P <- 0
       LAR    AR1,AXE      ; AR1 point to x(n-N+1)
```

```
      LAC    ONE,15        ; rounding
      CNFP
      RPTK   NAZ–1         ; for i=N–1, .., 1, 0
      MACD   PA_AE,*–      ; ACC = ACC + ai(n)*x(n–i)
      CNFD
      APAC
      SACH   ZN
;*
;*-----------------------------------------------------------------------------
;*
;* 3. LMS Algorithm to update coefficients of A(z)
;*
;*         ai(n+1) = ai(n) – u e(n) x'(n–i), for i=0,1,..,N–1
;*
;*  Note: using "–" when updating A(z) and B(z) in noise control mode
;*            instead of "+" when updating C(z) in training mode
;*
;*-----------------------------------------------------------------------------
;
      LT     EN            ; T = e(n)
      MPY    VMU1          ; P = mu * e(n)
      PAC                  ; ACC <– P
      ADD    ONE,15        ; rounding
      SACH   EN            ; EN = mu * e(n)
;*
      LARK    AR3,NAZ–1    ; initialize AR3 as loop counter
      LAR    AR1,AXPE      ; AR1 point to x'(n–N+1)
      LAR    AR2,ADAE      ; AR2 point to aN–1(n)
;*
      LT     EN            ; T = mu * e(n)
      MPY    *–,AR2        ; P = mu * e(n) * x'(n–N+1)
ADAP_A ZALR   *,AR1        ; ACC <– ai(n) and rounding
      MPYS   *,AR1         ; ai(n+1) = ai(n) – P
                            ; P = mu * e(n) * x'(n–i) for next i
      DMOV   *–,AR2        ; update x'(n) buffer
      SUB    *,LEAKY       ; ai(n+1) = ai(n+1) – (2exp(LEAKY–16)) * ai(n+1)
      SACH   *+,0,AR3      ; ai(n+1) <– ACC
      BANZ   ADAP_A,*–,AR2 ; go back to loop if (AR3) > 0
;*
;*-----------------------------------------------------------------------------
;*
;* 4. Filtered–U:
;*
;*    filtering y(n–1) by C(z) to get y'(n–1)
;*
;*                N–1
;*     y'(n–1) = sum ci * y(n–j–1)
;*                j=0
;*
;*     where ci, i=0,1,...,N–1 are from training mode filter C(z)
;*
;*-----------------------------------------------------------------------------
;*
FY_C   MPYK   0             ; P <– 0
      LAR    AR1,AYCE      ; AR1 point to y(n–N+1)
      LAC    ONE,15        ; rounding
      CNFP
      RPTK   NCZ–1         ; for i=N–1, .., 1, 0
      MACD   PA_CE,*–      ; ACC <– ACC+ci(n)*y(n–i)
      CNFD
      APAC                 ; ACC <– ACC+P = y'(n)
      LAR    AR1,AYP0      ; AR1 point to y'(n)
      SACH   *             ; inject y'(n) into buffer
```

```
;*
;*-----------------------------------------------------------------------------
;*
;*  5. Filter y(n) by B(z) to get z'(n)
;*
;*                M
;*     z'(n) = sum bj(n) * y(n-j-1)
;*               j=1
;*
;*-----------------------------------------------------------------------------
;*
FIR_B  LAR    AR1,AYE          ; AR1 point to y(n-N+1)
       LAC    ONE,15           ; rounding
       CNFP
       RPTK   NBZ-1            ; for i=N-1, .., 1, 0
       MAC    PA_BE,*-         ; ACC = ACC + bi(n)*y(n-i)
       CNFD
       APAC                    ; ACC = y(n)
       ADDH   ZN               ; z(n) + z'(n)
       SACH   YN               ; y(n) = z(n) + z'(n)
;*
;*-----------------------------------------------------------------------------
;*
;*  6. LMS Algorithm to update coefficients of B(z)
;*
;*         bj(n+1)=bj(n)-ue(n)y'(n-j), for j=1,2,...,M
;*
;*-----------------------------------------------------------------------------
;*
       LARP   AR1           ; AR1 as address pointer
       LARK   AR3,NBZ-1     ; initialize AR3 as loop counter
       LAR    AR1,AYPE      ; AR1 point to y'(n-N+1)
       LAR    AR2,ADBE      ; AR2 point to bN-1(n)
;*
       LT     EN            ; T = mu * e(n)
       MPY    *-,AR2        ; P = mu * e(n) * y'(n-N+1)
ADAP_B ZALR   *,AR1         ; ACC <- bi(n) and rounding
       MPYS   *,AR1         ; bi(n+1) = ai(n) - P
                            ; P = mu * e(n) * y'(n-i) for next i
       DOV    *-,AR2        ; update y'(n) buffer
       SUB    *,LEAKY       ; bi(n+1) = bi(n+1) - (2exp(leaky-16)) * bi(n+1)
       SACH   *+,0,AR3      ; bi(n+1) <- ACC
       BANZ   ADAP_B,*-,AR2 ; go back to loop if (AR3) > 0
;*
       LAC    YN            ; ACC <- y(n)
       LAR    AR2,AY0       ; AR2 point to y(n)
       SACL   *,AR1         ; inject y(n) into y(n) buffer, it
                            ; is delayed by 1 now
       B      LOOP

LOOP:  IDLE
       IDLE
       B      LOOP
;*****************************************************************************
;*
;*             INT1 Interrupt Service Routine
;*
;*****************************************************************************
       ORG    1000          ; Channel A interrupt
       SST    TST0          ; Save ST0
       LDPK   0
       OUT    BUF3,DAC      ; Output buffer to DAC
       LST    TST0          ; Restore ST0
```

```
        NOP
        SXF                 ; Reset external flag bit
        EINT
        RET
;*****************************************************************************
;*
;*         INT2 Interrupt Service Routine
;*
;*****************************************************************************
        ORG    2000             ; Channel B interrupt
        SST    TST0             ; Save ST0
        LDPK   0
        OUT    BUF4,DAC         ; Output buffer to DAC
        LST    TST0             ; Restore ST0
        NOP
        RXF                     ; Reset external flag bit
        EINT
        B      START
;*****************************************************************************
;*
;*            RINT Interrupt Service Routine
;*
;*****************************************************************************
        ORG    3000             ; ADC interrupt
        LDPK   0
        SST    TST0             ; Save ST0
        LAC    ADC              ; Read from serial port
        BIOZ   IN2              ; Skip if channel B
        SACL   BUF1             ; Move data to channel A buffer
        LST    TST0             ; Restore ST0
        EINT
        RET
IN2:    SACL   BUF2             ; For channel B, save data in channel B buffer
        LST    TST0             ; Restore ST0
        EINT
        RET
```

Filtered-X LMS Algorithm With Feedback Cancellation

```
;*
;* FILE NAME : FBFXLMS.ASM
;*****************************************************************************
;*
;*       This program has been modified to run on the Ariel's DSP-16
;*       plus TMS320C25 DSP Board.
;*                                                          Kai-Ming Chung
;*                                                          Oct. 1994
;*****************************************************************************
;*
;*   File: FXLMSFC.ASM
;*
;*   One-dimensional Adaptive Active Noise Control System Using
;*   FXLMS Algorithm with Feedback Cancellation on TMS320C25
;*
;*   Sen M. Kuo , Fall 1993
;*
;*************************************************************************
;*
;*   SYSTEM CONFIGURATION:
```

```
;*
;*          x(n)
;*   M1  *|---->|-----------------------|
;*             |                       |
;*             |  Adaptive FXLMS ANC   | y(n)
;*          e(n) |        System        |------> antinoise output
;*   M2  *|---->|                       |
;*             |-----------------------|
;*
;*   where
;*         x(n) – signal from the input microphone
;*         e(n) – signal from the error microphone
;*         y(n) – antinoise to drive the canceling loudspeaker
;*                (noise cancellation mode), or
;*                the training signal (off-line modeling mode)
;*
;*   Ports: x(n)  from input port B
;*          e(n)  from input port A
;*          y(n)  to output port B
;*
;*   The flow of program is:
;*
;*   C25 initialization –-> off-line secondary path and feedback
;*                          path modeling –-> on-line active noise control
;*
;*   A. Off-line modeling:
;*
;* input MIC              to Speaker                  error MIC
;*   |x(n)                     |                          |
;*   |                         |                          |
;*   |  +   – f(n) --------    |    --------z(n) –   +    |
;*   |-->(S)<------| D(z) |<---|--->| C(z) |----->(s)<-----|e(n)
;*         |        --------    |    --------       |
;*   e1(n).........>...:      y(n)      :..<........ e2(n)
;*
;*   where
;*         y(n) is an internally generate white noise
;*         C(z) and D(z) are an adaptive modeling filter,
;*                       both are updated by LMS algorithm
;*
;*
;*   B. On-line noise control:
;*
;*      from input MIC
;*            |
;*           +| – f(n)  ---------
;*           (S)<-------|  D(z) |<-----|
;*            |         ---------      |
;*            |     ----------         |y(n)     e(n) from error MIC
;*            |---->|  W(z)  |------------>          |
;*            |     ----------  to speaker           |
;*         -------       :                            |
;*         | C(z)|       :                            |
;*         -------   ----:----                        |
;*            |----->| FXLMS |<-----------------------
;*           x'(n)   ---------
;*
;*   where C(z) is a fixed error path modeling filter from training mode
;*         D(z) is a fixed feedback canceling filter from training mode
;*         W(z) is an adaptive noise control filter, updated by FXLMS
;*
;*****************************************************************************
;*
```

```
;*    define constants (be modified for different applications)
;*
;*-----------------------------------------------------------------------
;*
FS:       EQU    2000      ; assume sampling rate is 2 kHz.
TRNTIM:   EQU    30000     ; training time = 15 seconds
NCZ:      EQU    64        ; order of C(z), max = 64
NDZ:      EQU    64        ; order of D(z), max = 64
NWZ:      EQU    64        ; order of W(z), max = 94
MU:       EQU    4096      ; coef. update stepsize for off-line modeling
MU1:      EQU    328       ; coef. update stepsize for W(z)
MASK:     EQU    8805h     ; to mask off bits 0,2,11,15
SEED:     EQU    12357     ; seed for white noise generator
LEAKY:    EQU    2         ; leaky factor
;*
;***********************************************************************
;*
;*   Memory map:
;*   B2: page 0  – data I/O buffer
;*
;*   B0: page 4  – coefs of AF C(z) : ci(n), i=0,1,..,(NCZ–1)
;*                 (error path modeling filter), NCZ <= 64
;*               – coefs of AF D(z) : di(n), i=0,1,..,(NDZ–1)
;*                 (feedback path modeling filter), NDZ <=64
;*
;*   B0: page 5  – coefs of AF W(z) : wi(n), i=0,1,..,(NWZ–1)
;*                 (noise control filter)    NWZ <= 94
;*
;*   B1: page 6  – data buffer for C(z) and D(z):
;*
;*                 During off-line modeling, buffer contains training
;*                        signal y(n–i), i=0,1,..,(NCZ–1)
;*
;*                 During on-line canceling, buffer is for D(z),
;*                        y(n–i), i=0,1,...(NDZ–1)
;*
;*                 833–895 data buffer for x'(n)
;*
;*   B1: page 7  – 896–927 data buffer for x'(n)
;*                 928–1023 data buffer for W(z) : x(n–i), i=0,1,...,(NWZ–1)
;*
;***********************************************************************
;*
;*         PAGE 0 (memory-mapped regs and B2) DATA MEMORY ALLOCATION
;*
;*-----------------------------------------------------------------------
;*
P0DM:     EQU    0         ; page 0 data RAM address
DAC:      EQU    13        ; DAC I/O port
ADC:      EQU    0         ; Serial receive address
IMR:      EQU    4         ; Interrupt mask register
BUF1:     EQU    96        ; Buffer for channel A input data
BUF2:     EQU    97        ; Buffer for channel B input data
BUF3:     EQU    98        ; Buffer for channel A output data
BUF4:     EQU    99        ; Buffer for channel B output data
WN0:      EQU    100       ; storage for white noise generator
WN1:      EQU    101       ;          ″
TRNCTR:   EQU    102       ; training mode (0ff-line modeling) counter
STSFLG:   EQU    103       ; program status flag, 1 = training mode
ONE:      EQU    104       ; ONE = 1
VMU:      EQU    105       ; value of mu
VMU1:     EQU    106       ; value of mu1
```

```
ADCE:     EQU    107       ; address of cN–1(n)
ADDE:     EQU    108       ; address of dN–1(n)
ADWE:     EQU    109       ; address of wN–1(n)
AY0:      EQU    110       ; address for y(n)
AYE:      EQU    111       ; end address for y(n–N+1)
AX0:      EQU    112       ; address for x(n)
AXE:      EQU    113       ; end address for x(n–N+1)
AXCE:     EQU    114       ; end address for x(n–NCZ+1)
AXP0:     EQU    115       ; address for x'(n)
AXPE:     EQU    116       ; end address for x'(n–N+1)

;* data buffer for signals

YN:       EQU    117       ; y(n), to loudspeaker
XN:       EQU    118       ; x(n), from input MIC
EN:       EQU    119       ; e(n), from error MIC
ZN:       EQU    120       ; z(n), output from W(z)
FN:       EQU    121       ; f(n), output from D(z)
TN:       EQU    122       ; to output port B

TEMP:     EQU    123       ; Temporary storage location
TST0:     EQU    124       ; Storage for ST0
;*
;*-----------------------------------------------------------------------------
;*
;*   PAGE 4 (B0) DATA MEMORY ALLOCATION
;*   AF C(z): ci(n), i=0,1,2,..,NCZ–1
;*
P4DM:     EQU    512           ; PAGE 4 DATA MEM ADRS
P4PM:     EQU    65280         ; PAGE 4 PROG MEM ADRS
CE:       EQU    0             ; ci(n), end of C(z) buffer
DA_CE:    EQU    P4DM+CE       ; address of ci(n), i=NCZ–1 at CNFD
PA_CE:    EQU    P4PM+CE       ; address of ci(n),    "      at CNFP
;*
;*-----------------------------------------------------------------------------
;*
;*   Memory map:   cN–1(n)     low address <–– PA_CE, ADCE
;*
;*                 cN–2(n)
;*                   ...
;*                 c1(n)
;*                 c0(n)       high address
;*
;*-----------------------------------------------------------------------------
;*
;*   AF D(z): di(n),  i=0,1,2,..,NDZ–1
;*
DE:       EQU    64            ; di(n), end of D(z) buffer
DA_DE:    EQU    P4DM+DE       ; address of di(n), i=NDZ–1 at CNFD
PA_DE:    EQU    P4PM+DE       ; address of di(n),    "  at CNFP
;*
;*-----------------------------------------------------------------------------
–
;*
;*   Memory map:   dN–1(n)     low address <–– PA_DE, ADCE
;*                 dN–2(n)
;*                   ...
;*                 d1(n)
;*                 d0(n)       high address
;*
;*-----------------------------------------------------------------------------
;*
;*       PAGE 5 (B0) DATA MEMORY ALLOCATION
;*   AF W(z): wi(n),  i=0,1,2,..,NWZ–1
;*
```

```
P5DM:     EQU    640          ; PAGE 5 DATA MEM ADRS
P5PM:     EQU    65408        ; PAGE 5 PROG MEM ADRS
WE:       EQU    0            ; wi(n), end of W(z) buffer
DA_WE:    EQU    P5DM+WE      ; address of WE at CNFD
PA_WE:    EQU    P5PM+WE      ; address of WE at CNFP
;*
;*-----------------------------------------------------------------------------
;*
;*  Memory map:      wN-1(n)       low address, ADWE
;*                   wN-2(n)
;*                   ...
;*                   w1(n)
;*                   w0(n)         high address
;*
;*-----------------------------------------------------------------------------
;*
;*     PAGE 6 (B1) DATA MEMORY ALLOCATION for C(z)
;*                   x'(n-i), i=0,1,...,NCZ-1
;*
P6DM:     EQU    768          ; PAGE 6 DATA MEM ADRS
Y0:       EQU    0
YE:       EQU    Y0+NDZ-1     ; y(n-NDZ+1)
A_Y0:     EQU    P6DM+Y0      ; address of y(n)
A_YE:     EQU    P6DM+YE      ; address of y(n-NDZ+1)
;*
;*-----------------------------------------------------------------------------
;*
;*  Memory map:      y(n)          low address <-- AY0
;*                   y(n-1)
;*                   ...
;*                   y(n-N+1)      high address <-- AYE
;*
;*-----------------------------------------------------------------------------
;*
XP0:          EQU    65        ; x'(n), (64+1), leave one space for y(n)
XPE:          EQU    XP0+NWZ-1  ; x'(n-NWZ+1)
A_XP0:    EQU    P6DM+XP0     ; address of x'(n)
A_XPE:    EQU    P6DM+XPE     ; address of x'(n-NWZ+1)
;*
;*-----------------------------------------------------------------------------
;*
;*  Memory map:      x'(n)         low address <-- AXP0
;*                    x'(n-1)
;*                   ...
;*                   x'(n-N+1)     high address <-- AXPE
;*
;*-----------------------------------------------------------------------------
;*
;*   PAGE 7 (B1) DATA MEMORY ALLOCATION for W(z)
;*                 x(n-i), i=0,1,...,NWZ-1,
;*                 part of this page is for x'(n) (896-927).
;*
P7DM:     EQU    896          ; PAGE 7 DATA MEMORY ADDRESS
X0:       EQU    32           ; x(n)
XE:       EQU    X0+NWZ-1     ; x(n-NWZ+1)
XCE:      EQU    X0+NCZ-1     ; x(n-NCZ+1)
A_X0:     EQU    P7DM+X0      ; address of x(n)
A_XE:     EQU    P7DM+XE      ; address of x(n-NWZ+1)
A_XCE:    EQU    P7DM+XCE     ; address of x(n-NCZ+1)
;*
;*-----------------------------------------------------------------------------
;*
;*  Memory map:      x(n)          ; <-- AX0
```

```
;*                  x(n-1)
;*                  ...
;*                  x(n-N+1)      ; <-- AXE
;*
;*****************************************************************************
;*
;*     INTERRUPT BRANCHES
;*
;*****************************************************************************
;*
RESET: B      32               ; On hardware reset go to INIT
       ORG    4
INT1:  B      1000             ; On INT1 go to interrupt 1 service routine
       ORG    6
INT2:  B      2000             ; On INT2 go to interrupt 2 service routine
       ORG    26
RCV:   B      3000             ; On RINT go to ADC service routine
;*
;*****************************************************************************
;*
;*         PROCESSOR INITIALIZATION ROUTINE
;*
;*****************************************************************************
;*
INIT:  ORG    32
       LDPK   0                ; Load page 0
       LALK   2E00H            ; 0010 1110 0000 0000 in binary
       SACL   TEMP             ; Initialize ST0
       LST    TEMP             ; 0 -> DP
                               ; 1 -> INTM, interrupts disabled
                               ; 1 -> OVM
                               ; 0 -> OV
                               ; 1 -> ARP
       SPM    1                ; P reg. output shift left 1 bit
;*
;*        INITIALIZE PAGE 0
;*
       LARP      AR1         ; AR1 as address pointer
       LARK      AR1,96    ; LOWEST PAGE 0 LOCATION -> AR1
       ZAC                  ; 0 -> ACC
       RPTK      31        ; REPEAT NEXT INSTRUCTION 32 TIMES
       SACL      *+        ; ZERO PAGE 0
       LACK      1         ; ACC <- 1
       SACL      ONE       ; ONE <-- 1
       SACL      STSFLG    ; STSFLG=1, training mode first
       LALK      TRNTIM    ; ACC <- # of training samples
       SACL      TRNCTR    ; training time = 3 seconds = 4500 samples
       LALK      MU        ; ACC <- mu
       SACL      VMU       ; value of mu
       LALK      MU1       ; ACC <- mu1
       SACL      VMU1      ; value of mu1
       LACK      22        ; ACC <- 22
       SACL      IMR        ; Enable INT1, INT2 & RINT
       LALK      SEED      ; ACC <- SEED
       SACL      WN0       ; initial white noise = seed

;* initialize address pointers
       LALK      DA_CE
       SACL      ADCE      ; address of cN-1(n) in ADCE
       LALK      DA_DE
       SACL      ADDE      ; address of dN-1(n) in ADDE
```

```
        LALK      DA_WE
        SACL      ADWE      ; address of wN–1(n) in ADWE
        LALK      A_Y0
        SACL      AY0       ; address of y(n)
        LALK      A_YE
        SACL      AYE       ; address of y(n–N+1)
        LALK      A_X0
        SACL      AX0       ; address of x(n)
        LALK      A_XE
        SACL      AXE       ; address of x(n–N+1)
        LALK      A_XP0
        SACL      AXP0      ; address of x'(n)
        LALK      A_XPE
        SACL      AXPE      ; address of x'(n–N+1)
        LALK      A_XCE
        SACL      AXCE      ; address of x(n–NCZ+1)
;*      CLEAR PAGES 4, 5, 6, and 7
        LARP      AR1
        LRLK      AR1,512   ; LOWEST PAGE 4 ADDRESS –> AR1
        ZAC                 ; 0 –> ACC
        RPTK      255
        SACL      *+        ; zero page 4 & 5
        RPTK      255
        SACL      *+        ; zero page 6 & 7
        EINT                ; ENABLE INTERRUPTS
        B         LOOP
;*
;*****************************************************************************
;*
;*  CYCLE START ROUTINE
;*
;*****************************************************************************
;*
START   LARP   AR3          ; 3 –> ARP
        LAC    BUF1         ; ACC <– from input port A
        SACL   EN           ; value of error signal from error microphone
        LAC    BUF2         ; ACC <– from input port B
        SACL   XN           ; value of reference signal from input microphone
        LAC    YN           ; ACC <– YN
        SACL   BUF3         ; value of anti-noise signal to output port A
        LAC    TN           ; ACC <– TN
        SACL   BUF4         ; value of concerned signal to output port B
        LAC    STSFLG       ; if STSFLG = 1, goto CANCEL
        BZ     CANCEL       ; for next sample, loop forever
;*
;*****************************************************************************
;*
;*  Training mode (off-line modeling of secondary path and feedback path)
;*
;*-----------------------------------------------------------------------------
;*
;*   White noise generator:
;*   Algorithm:
;*
;*                rotate left 1-bit
;*    b15 . b11         <----       b2 . b0 <------------------|
;*     |     |                       |    |–>                 |
;*     |     |                       |------> XOR –>|          |
;*     |     |                                    |          |
;*     |     |----------->                        |–>        |
;*     |-----------------> XOR ------------------------> XOR –->|
;*
```

```
;*  where the initial value of WN0 (b15..b0) = seed
;*
;*-----------------------------------------------------------------------
;*
TRAIN  LAC    WN0       ; Load noise sequence
       ANDK   MASK      ; Mask off feedback bits
       SACL   WN1       ; Save temporary
       ADD    WN1,4     ; combine bits 11 and 15
       ADD    WN1,13    ; combine bit 2 with result
       ADD    WN1,15    ; combine bit 0 with result
       ANDK   MASK      ; re-use mask to mask off MSB
       ADDH   WN0       ; combine MSB with sequence
       SACH   WN0,1     ; save result (and shift out MSB)

       LAC    WN0,11    ; scale WN0
       SACH   YN        ; output white noise
;*
;***********************************************************************
;*
;*   Adaptive Off-line Secondary-Path and Feedback Path Modeling
;*
;* input MIC              to Speaker                  error MIC
;*   |x(n)                     |                          |
;*   |                         |                          |
;*   |  +   – f(n) --------    |    --------z(n) –   +    |
;*   |-->(S)<------| D(z) |<---|--->| C(z) |----->(s)<-----|e(n)
;*        |        --------    |    --------       |
;*   e1(n).........>...:      y(n)      :..<........ e2(n)
;*
;*   where
;*         y(n) is an internally generate white noise
;*         C(z) and D(z) are an adaptive modeling filter,
;*                        both are updated by LMS algorithm
;*
;***********************************************************************
;*
;*                      NCZ–1
;*  A. Computes z(n) =  sum  ci(n) * y(n–i)
;*                       i=0
;*-----------------------------------------------------------------------
;*
          LARP   AR1              ; AR1 as current address reg
FIR_C     MPYK   0                ; P=0
          LAR    AR1,AY0          ; AR1 pointing to y(n)
          SACH   *,AR1            ; inject white noise to buffer y(n)
          LAR    AR1,AYE          ; AR1 pointing to y(n–NCZ+1)
          LAC    ONE,15           ; round-off offset to ACC
          CNFP
          RPTK   NCZ–1            ; for i = NCZ–1,NCZ–2,...,0
          MAC    PA_CE,*–         ; ci(n) * y(n–i) + ACC –> ACC
          CNFD                    ;
          APAC                    ; P + ACC –> ACC = z(n)
;*
;*-----------------------------------------------------------------------
;*
;*  B. computes error signals e1(n):
;*
;*        e1(n) = e(n) – z(n)
;*
;*-----------------------------------------------------------------------
;*
          NEG                     ; ACC = – z(n)
          ADDH      EN            ; e1(n) = e(n) – z(n)
          SACH      ZN            ; z(n) = e1(n)
```

```
;*
;*-----------------------------------------------------------------------------
;*
;*  C. update coefficients of C(z) using LMS Algorithm:
;*
;*          ci(n+1) = ci(n) + u*e1(n)*y(n-i)
;*
;*          AR1 - point to data buffer, y(n-i)
;*          AR2 - point to AF coefs, ci(n)
;*
;*-----------------------------------------------------------------------------
;*
          LT     ZN                ; T = e1(n)
          MPY    VMU                ; P = mu*e1(n)
          PAC                       ; ACC <- P
          ADD    ONE,15            ; rounding
          SACH   ZN                ; ZN = mu*e1(n)
;*
          LARK   AR3,NCZ-1         ; initialize AR3 as loop counter
          LAR    AR1,AYE            ; AR1 pointing to y(n-N+1)
          LAR    AR2,ADCE          ; AR2 pointing to cN-1(n)
          LT     ZN                ; T = mu*e1(n)
          LARP   AR1
          MPY    *-,AR2            ; P = mu*e1(n)*y(n-i)
;*
ADAP_C    ZALR   *,AR1             ; load ACC with ci(n) and round
          MPYA   *-,AR2            ; ci(n+1) = ci(n) + P
                                   ; P = mu*e1(n)*y(n-i) for next i
          SACH   *+,0,AR3          ; store ci(n+1)
          BANZ   ADAP_C,*-,AR2     ; go back to loop if counter (AR3) > 0
;*
;*-----------------------------------------------------------------------------
;*                      NDZ-1
;*  D. Computes f(n) =  sum  di(n) * y(n-i)
;*                       i=0
;*-----------------------------------------------------------------------------
;*
          LARP   AR1               ; AR1 as current address reg
FIR_D     MPYK   0                 ; P=0
          LAR    AR1,AYE           ; AR1 pointing to y(n-NDZ+1)
          LAC    ONE,15            ; round-off offset to ACC
          CNFP
          RPTK   NDZ-1              ; for i = NDZ-1,NDZ-2,...,0
          MACD   PA_DE,*-          ; di(n) * y(n-i) + ACC -> ACC
          CNFD                     ; also move data y(n-i)
          APAC                     ; P + ACC -> ACC = f(n)
;*
;*-----------------------------------------------------------------------------
;*
;*  E. computes error signals e2(n):
;*
;*       e2(n) = x(n) - f(n)
;*
;*-----------------------------------------------------------------------------
;*
          NEG                      ; ACC = - f(n)
          ADDH      XN             ; e2(n) = x(n) - f(n)
          SACH      ZN             ; ZN = e2(n)
;*
;*-----------------------------------------------------------------------------
;*
;*  F. update coefficients of D(z) using LMS Algorithm:
;*
```

```
;*       di(n+1) = di(n) + u*e2(n)*y(n–i)
;*
;*       AR1 – point to data buffer, y(n–i)
;*       AR2 – point to AF coefs,    di(n)
;*
;*-----------------------------------------------------------------------------
;*
       LT        ZN                ; T = e2(n)
          MPY    VMU               ; P = mu*e2(n)
          PAC                      ; ACC <– P
          ADD    ONE,15            ; rounding
          SACH   ZN                ; ZN = mu*e2(n)
;*
          LARK   AR3,NDZ–1         ; initialize AR3 as loop counter
          LAR    AR1,AYE
          MAR    *+                ; AR1 pointing to y(n–N+1) due to
                                   ; MACD data move effect
          LAR    AR2,ADDE          ; AR2 pointing to dN–1(n)
          LT     ZN                ; T = mu*e2(n)
          MPY    *–,AR2            ; P = mu*e2(n)*y(n–i)
;*
ADAP_D    ZALR   *,AR1             ; load ACC with di(n) and round
          MPYA   *–,AR2            ; di(n+1) = di(n) + P
                                   ; P = mu*e2(n)*y(n–i) for next i
          SACH   *+,0,AR3          ; store di(n+1)
          BANZ   ADAP_D,*–,AR2     ; go back to loop if counter (AR3) > 0
;*
;*-----------------------------------------------------------------------------
;*
;* check if end of training mode
;*
;*-----------------------------------------------------------------------------
;*
       LAC    TRNCTR               ; ACC <– training counter
       SUBK   1                     ; decrement training counter
       SACL   TRNCTR               ; save counter
       BGZ    LOOP
;*
       ZAC                         ; end of training mode, ACC=0
       SACL   STSFLG               ; STSFLG <– 0, now in noise control mode
       LARP   AR1                  ; make sure AR1 is address pointer
       LAR    AR1,AY0              ; AR1 pointing to y(n)
       RPTK   NCZ                  ; repeat NCZ+1 time
       SACL   *+                   ; clear C(z) buffer for x'(n) in noise
                                   ; control mode
       B      START                ; end training, go to noise control mode
;*
;*****************************************************************************
;*
;*   Noise control mode:
;*
;*      from input MIC
;*            |
;*           +| – f(n)  ---------
;*           (S)<-------|  D(z) |<-----|
;*            |         ---------      |
;*            |     ----------         |y(n)     e(n) from error MIC
;*            |----->|  W(z)  |------------>          |
;*            |     ----------  to speaker            |
;*         -------       :                            |
;*         | C(z)|       :                            |
;*         -------   ----:----                        |
;*            |----->| FXLMS |<-----------------------
```

```
;*           x'(n)   ---------
;*
;* where C(z) is a fixed error path modeling filter from training mode
;*       D(z) is a fixed feedback canceling filter from training mode
;*       W(z) is an adaptive noise control filter, updated by FXLMS
;*
;*****************************************************************************
;*
;*  In real-time x(n) is from input microphone
;*-----------------------------------------------------------------------------
;*
CANCEL LAC    XN
       LARP   AR1              ; ARP = 1
       LAR    AR1,AX0
       SACL   *
;*
;*-----------------------------------------------------------------------------
;*
;*  1. compute feedback free input for adaptive filter W(z)
;*
;*     x(n) = x(n) - f(n)
;*
;*-----------------------------------------------------------------------------
;*
       LAC    XN               ; x(n) -> ACC
       SUB    FN               ; x(n) - f(n)
       LAR    AR1,AX0          ; x(n) = x(n) - f(n)
       SACL   *
;*
;*-----------------------------------------------------------------------------
;*
;* 2. Filtered-X, filtering x(n) by C(z) to get x'(n)
;*
;*             N-1
;*     x'(n) = sum ci * x(n-i)
;*             i=0
;*
;*    where ci, i=0,1,..,N-1 are from training mode filter C(z)
;*
;*-----------------------------------------------------------------------------
;*
FX_C   MPYK   0                   ; P <- 0
       LAR    AR1,AXCE            ; AR1 point to x(n-NCZ+1)
       LAC    ONE,15              ; rounding
       CNFP
       RPTK   NCZ-1               ; for i=N-1, .., 1, 0
       MAC    PA_CE,*-            ; ACC <- ACC+ci(n)*x(n-i)
       CNFD
       APAC                       ; ACC <- ACC+P = x'(n)
       LAR    AR1,AXP0            ; AR1 point to x'(n)
       SACH   *                   ; inject x'(n) into buffer
;*
;*-----------------------------------------------------------------------------
;*
;*  3. Filter x(n) by W(z) to get y(n), the anti-noise
;*
;*            N-1
;*     y(n) = sum wi(n) * x(n-i)
;*            i=0
;*
;*-----------------------------------------------------------------------------
;*
FIR_W  MPYK   0                   ; P <- 0
```

```
      LAR    AR1,AXE             ; AR1 point to x(n–N+1)
      LAC    ONE,15              ; rounding
      CNFP
      RPTK   NWZ–1               ; for i=N–1, .., 1, 0
      MACD   PA_WE,*–            ; ACC = ACC + wi(n)*x(n–i)
      CNFD
      APAC                       ; ACC = y(n)
      LAR    AR1,AY0             ;
      SACH   *                   ; inject y(n) into y(n–i) buffer
      SACH   YN
;*
;*-----------------------------------------------------------------------------
;*
;* 4. FXLMS Algorithm to update coefficients of W(z)
;*
;*       wi(n+1) = wi(n) – u e(n) x'(n–i), for i=0,1,..,N–1
;*
;*  Note: using "–" when updating W(z) in noise control mode
;*        instead of "+" when updating C(z) in training mode
;*
;*       AR1 pointing to x'(n–i) data buffer
;*       AR2 pointing to wi(n) coefs buffer
;*
;*-----------------------------------------------------------------------------
;*
      LT     EN            ; T = e(n)
      MPY    VMU1          ; P = mu * e(n)
      PAC                  ; ACC <– P
      ADD    ONE,15        ; rounding
      SACH   EN            ; EN = mu * e(n)
;*
      LARK   AR3,NWZ–1     ; initialize AR3 as loop counter
      LAR    AR1,AXPE      ; AR1 point to x'(n–N+1)
      LAR    AR2,ADWE      ; AR2 point to wN–1(n)
;*
      LT     EN            ; T = mu * e(n)
      LARP   AR1
      MPY    *–,AR2        ; P = mu * e(n) * x'(n–N+1)
ADAP_W ZALR  *,AR1         ; ACC <– wi(n) and rounding
      MPYS   *,AR1         ; wi(n+1) = wi(n) – P
                            ; P = mu * e(n) * x'(n–i) for next i
      DMOV   *–,AR2        ; update x'(n) buffer
      SUB    *,LEAKY       ; wi(n+1) = wi(n+1) – (2exp(LEAKY–16)) * wi(n+1)
      SACH   *+,0,AR3      ; wi(n+1) <– ACC
      BANZ   ADAP_W,*–,AR2 ; go back to loop if (AR3) > 0
;*
;*****************************************************************************
;*
;*  5. Filter y(n) by D(z) to get f(n), the feedback from anti-noise
;*     speaker to input microphone.
;*
;*                     NDZ–1
;*               f(n) = sum di * y(n–i)
;*                       i=0
;*
;*-----------------------------------------------------------------------------
;*
FD    MPYK   0                ; P <– 0
      LAR    AR1,AYE          ; AR1 point to y(n–N+1)
      LAC    ONE,15           ; rounding
      CNFP
      RPTK   NDZ–1            ; for i=N–1, .., 1, 0
      MACD   PA_DE,*–         ; ACC = ACC + di(n)*y(n–i)
```

```
        CNFD
        APAC                    ; ACC = f(n)
        SACH   FN               ; f(n) –> FN

        B      LOOP

LOOP    IDLE
        IDLE
        B      LOOP
;*****************************************************************************
;*
;*         INT1 Interrupt Service Routine
;*
;*****************************************************************************

        ORG    1000          ; Channel A interrupt
        SST    TST0          ; Save ST0
        LDPK   0
        OUT    BUF3,DAC      ; Output buffer to DAC
        LST    TST0          ; Restore ST0
        NOP
        SXF                  ; Set external flag bit
        EINT
        RET
;*****************************************************************************
;*
;*         INT2 Interrupt Service Routine
;*
;*****************************************************************************

        ORG    2000          ; Channel B interrupt
        SST    TST0          ; Save ST0
        LDPK   0
        OUT    BUF4,DAC      ; Output buffer to DAC
        LST    TST0          ; Restore ST0
        NOP
        RXF                  ; Set external flag bit
        EINT
        B      START
;*****************************************************************************
;*
;*         RINT Interrupt Service Routine
;*
;*****************************************************************************

        ORG    3000             ; ADC interrupt
        LDPK   0
        SST    TST0             ; Save ST0
        LAC    ADC              ; Read from serial port
        BIOZ   IN2              ; Skip if channel B
        SACL   BUF1             ; Move data to channel A buffer
        LST    TST0             ; Restore ST0
        EINT
        RET
IN2:    SACL   BUF2             ; For channel B, save data in channel B buffer
        LST    TST0             ; Restore ST0
        EINT
        RET
```

APPENDIX D: GENERAL CONFIGURABLE SOFTWARE FOR ANC EVALUATION

These software modules together provide an easy way to evaluate a number of one-dimensional ANC algorithms using any of several tools. A configuration module is included that allows the user to specify different algorithms, adaptive filter characteristics, and simulation model characteristics. The conditional assembly capability of TI's fixed-point macro assembler and linker is used to construct a custom executable file that can be run on a simulator, an evaluation module (EVM), or a target system. The software modules described in this section run on any TMS320C2x DSP.

Each major function performed by the software is coded in a separate module. A list of the modules is given here with a brief functional description of each. The configuration file (config.asm) is described in more detail, and assembly-language program code listings for all of the modules follow.

ANC.ASM	Contains training and cancellation routines. This is where the body of the ANC algorithm resides.
CONFIG.ASM	Contains software configuration settings. This is the only module that must be modified to change the characteristics of the executable file.
GLOBALS.ASM	Contains global constant and variable definitions.
INIT.ASM	Contains processor and algorithm initialization routines.
MACROS.ASM	Contains special macro routines.
MAIN.ASM	Contains code to control program flow through the different software modules.
MEMORY.ASM	Contains memory configuration directives that allocate data and program memory.
MODEL.ASM	Contains simulation transfer-function models and waveform generator code.
VECTOR.ASM	Contains interrupt vectors and unused interrupt traps.
ANC26.CMD	Linker command file for 'C26-based simulator or EVM. Modifications to the memory map must be made if a different target system is used to run the algorithms or if a change to the default data and program space definitions is desired. The default memory map assigns the memory locations 8000h through 0f9ffh to external data, which is used for the trace buffer function if it is enabled.
MAKE.BAT	Batch file that assembles and links software modules into an executable file after changes have been made in the configuration file CONFIG.ASM.
EVMINIT.CMD	Control file that contains the memory configuration to be used by the HLL debugger when running the ANC code on the 'C2x EVM. The memory configuration must match the memory map defined in the linker command file. A similar configuration file called SIMINIT.CMD is required if the software simulator is used to run the executable code.

Configuration File (config.asm) Description

Section 1 of the configuration file is used to configure the software for different I/O, adaptive filter forms, and adaptation methods. It also allows acoustic channel simulation and trace buffers to be enabled. Some of the switch settings in Section 1 determine which constants are active in Section 2. Explanations of the settings are summarized in Table 5 and more detail follows the table.

Table 5. Section 1 of the Configuration File

SWITCH NAME	SETTINGS	MEANING OF EACH SETTING
PROCESSOR	C26	Target processor is TMS320C26
	C25	Target processor is TMS320C25
TIMEBASE	timer	Onboard timer controls sample rate
	external	External interrupt #0 controls sample rate
	freerun	Run as fast as possible (simulator)
SIMULATION	yes	Simulation mode enabled
	no	Simulation mode disabled
TRACE	in_out	Trace algorithm input and output
	out	Trace algorithm output only
	none	No trace
GENERATOR	white_noise	White noise generator enabled
	sinewave	Sine-wave generator enabled
ALGORITHM	fxlms	Filtered-X LMS enabled (FXLMS)
	fbfxlms	FXLMS with acoustic feedback enabled (FBFXLMS)
	furlms	Filtered-U recursive LMS enabled (FURLMS)
	fanc	Feedback ANC enabled
ADAPTATION	lms	Standard LMS coefficient adaptation enabled
	leaky_lms	Leaky LMS coefficient adaptation enabled

The PROCESSOR switch selects the target processor on which the software is to run.

The TIMEBASE switch selects the method of controlling the effective sample rate of the algorithm. The freerun selection speeds up execution in the simulator.

The SIMULATION switch enables the modeling of the acoustic channel response. This allows the algorithm to be tested in an ideal acoustic environment prior to testing in the real world. Combined with the TRACE switch, SIMULATION can provide performance information about the algorithm.

The TRACE switch enables a trace buffer for the error signal for both training and cancellation modes. If in_out is selected, the input signal for the cancellation mode is also traced. It is intended that the trace buffer be implemented in external data RAM like that used on the ’C2x EVM. A total of approximately 30K words of memory are available with the default memory map defined in the module anc26.cmd. This function is intended to be used with the SIMULATION option.

The GENERATOR switch selects either the white noise or the sine-wave generator. The training mode and most verification tests use the white noise generator to produce a broadband input signal to the system. The sine-wave generator produces a signal that is a summation of a 150-Hz and a 250-Hz signal (assuming a 1500-Hz sample rate).

The ALGORITHM switch selects one of the one-dimensional ANC algorithms described in this report.

The ADAPTATION switch selects either the standard LMS algorithm or the leaky LMS algorithm for the coefficient adaptation routine. The leaky LMS algorithm is typically used with fixed-point processors to prevent coefficient overflow.

Section 2 of the configuration file contains constants that define the characteristics of the algorithm and simulation (if it is enabled). The constants and their values are summarized in Table 6 and more detail follows the table.

Table 6. Section 2 of the Configuration File

CONSTANT	VALUE	DESCRIPTION
CLKOUT	10000000	DSP instruction-cycle rate (in Hz)
FS	1500	Sample rate (samples/second)
TIME	3	Number of seconds to run training mode
NSAMPLES	TIME*FS	Number of samples processed in training mode
NTICKS	(CLKOUT/FS)-1	TIMER period
μ	1024	Coefficient update step size
LEAKAGE	1	Shift value to control leak off
NAz	64	IIR feedforward filter order A(z)
NBz	64	IIR feedback filter order B(z)
NCz	64	Error path model order C(z)
NDz	64	Acoustic feedback model order D(z)
NWz	64	FIR filter order W(z)
SIMLENGTH	4500	Cancellation routine simulation run time
Gs	2458	Noise source path gain: $G_S = 0.6$
Ts	10	Noise source path sample delay period
Ge	3277	Error path gain: $G_e = 0.8$
Te	2	Error path sample delay period
Gf	2867	Acoustic feedback path gain: $G_f = 0.7$
Tf	Ts–Te	Acoustic feedback path sample delay period

CLKOUT defines the instruction cycle rate of the DSP. It and the sample rate are used together to define the onboard timer period if the TIMEBASE switch in Section 1 is set to timer. The value of 10 MHz shown above is the instruction cycle rate for a 40-MHz input clock; faster and slower system clocks exist and depend on the particular DSP. The value for CLKOUT is obtained by dividing the system clock by 4.

FS defines the sample rate of the ANC system. It and the constant TIME are used to determine how long the training mode runs. Also, if the TIMEBASE switch is set to timer, FS is used with CLKOUT to determine the value to enter into the onboard timer's period register. If the value of FS is changed and the GENERATOR switch is set to sinewave, the two sine-wave frequencies generated are not 150 and 250 Hz. Equations showing the relationship between FS and the sine-wave frequencies are given at the end of this section.

TIME defines sets the number of seconds the training mode runs. The default value of 3 seconds allows adequate time for the adaptation routine to converge on the filter coefficient values that model the acoustic channel error path (and the feedback path for the feedback FXLMS algorithm). If the error term has not approached zero by the end of the period TIME, a problem is likely to exist.

NSAMPLES is the product of training-mode run time (TIME) and the sample rate (FS). NSAMPLES is used as a loop counter to control how many samples the training routine processes.

NTICKS is the value placed in the onboard timer's period register. It is computed using the DSP instruction cycle rate CLKOUT) and the desired sample rate (FS). The value of NTICKS is the number of instruction cycles (clock ticks) to elapse between timer interrupts, which is used to control the sample rate. This constant is valid only when the TIMEBASE switch is set to timer.

The constant μ defines the step size of the adaptive filter coefficient adaptation. It controls how fast the error term causes the coefficients to change and the minimum magnitude of the error term. The larger the value of μ, the faster the coefficients adapt and the larger the minimum error signal. If too large a value of μ is used, the adaptation routing can become unstable. A value for μ of less than 1/N, where N is the adaptive filter order, should be used.

LEAKAGE is used when the ADAPTATION switch is set to leaky_lms. It controls how much of the previous value of the adaptive filter coefficient is used in the adaptation routine. LEAKAGE is used as a shift value as shown in the equation:

$$w_i(n + 1) = \left(1 - \frac{1}{2^{\mathrm{LEAKAGE}}}\right) w_i(n) - \mu e(n)x(n) \tag{96}$$

NAz through NWz define the order of each filter used in the ANC algorithm, where the filters are A(z), B(z), C(z), D(z), and W(z). The setting of the ALGORITHM switch determines which of these constants are valid. These constants are also used in the module memory.asm to define array sizes for the coefficients placed in block B0 of the 'C2x DSP's on-chip RAM. Note that all of the coefficient arrays must fit within block B0, which is 256 words long in the 'C25, while in the 'C26 it is 512 words long. In the memory map, all of the filter coefficients, input signals, and output signals were allocated in the on-chip memory blocks B0, Bl, and B2 of the 'C25. Table 7 shows the coefficient arrays used with each algorithm and how to compute the total amount of memory used. Typically, the order of the adaptive filter defined by NAz, NBz, and NWz is greater than or equal to the fixed correction filters defined by NCz and NDz.

The 'C2x chip used was a 100-ns (single-cycle instruction time) 'C25 device. Therefore, the required DSP time for the 'C25 to perform an algorithm in real time is equal to the number of instruction cycles needed times 100 ns. The sampling period between two sampling points is determined by the sampling frequency. The higher the sampling frequency, the shorter the sampling period. The DSP overhead is the percentage of the sampling period in which the algorithm is executed by the DSP. Hence, the calculation of DSP overhead for each algorithm can be expressed as:

$$\text{DSP overhead} = \frac{\text{DSP execution time}}{\text{Sampling period}} \times 100\% \tag{97}$$

Based on equation (97) and NWz = NAz = NBz = NCz = NDz = 64 from Table 6, the number of instruction cycles, DSP execution time, and DSP overhead for the ANC setup using FXLMS, FBFXLMS, and FURLMS algorithms at a 2000-Hz sampling frequency are calculated as shown in Table 7.

Table 7. Number of Instruction Cycles, DSP Execution Time, and TMS320C25 DSP Overhead per Algorithm

ALGORITHM	CALCULATION FOR NUMBER OF INSTRUCTION CYCLES	NUMBER OF INSTRUCTION CYCLES	DSP EXECUTION TIME (μs)	TMS320C25 DSP OVERHEAD (%)
FXLMS	77 + NWz* 9 + NCz * 2	781	78.1	15.62
FBFXLMS	102 + NWz * 9 + NCz * 2 + NDz * 2	934	93.4	18.68
FURLMS	107 + NAz * 9 + NBz * 9 + NCz * 4	1515	151.5	30.3

For the detailed calculations of the instruction cycles needed, refer to the application book *Digital Signal Processing Applications with the TMS320 Family,* Volume 3 [5], published by TI.

SIMLENGTH defines the cancellation routine run time if the SIMULATION switch is set to yes. SIMLENGTH is used as a loop counter to control the number of samples processed by the cancellation routine. The default value of 4500 allows adequate time for the error signal to reach a minimum. If the error signal has not converged to almost zero by the end of this time period, the other switch and constant settings must be checked.

Gs through Tf define the characteristics of the acoustic channel model used to compute the error-microphone input signal. These constants are valid only if the SIMULATION switch is set to yes. The basic form of the model is:

$$e(n) = Gx \times s(n - Tx) \tag{98}$$

where x = s, e, or f, and the specific constant is from Table 6.

As shown in Figure 45, the constants Gs and Ts are used to model the transfer function between the noise source and the error microphone. The constants Ge and Te are used to model the transfer function between the output speaker and the error microphone. The constants Gf and Tf are used to model the transfer function between the output speaker and the input microphone. Gf and Tf are valid only when the ALGORITHM switch is set to fbfxlms. It is assumed that the feedback and error path delays must equal the noise source delay (Ts – Te = Tf) and that Tf > Te. Also, the value of unity gain for Gs, Ge, and Gf is 4095 and not 32767, as might be expected. This is because the calculation of the acoustic channel models makes use of a feature on the ’C2x DSP that allows one of the inputs to the multiplier to be a 13-bit signed constant that is embedded in the instruction word. Using a 13-bit signed number makes the maximum positive number equal to $2^{12} - 1$, or 4095.

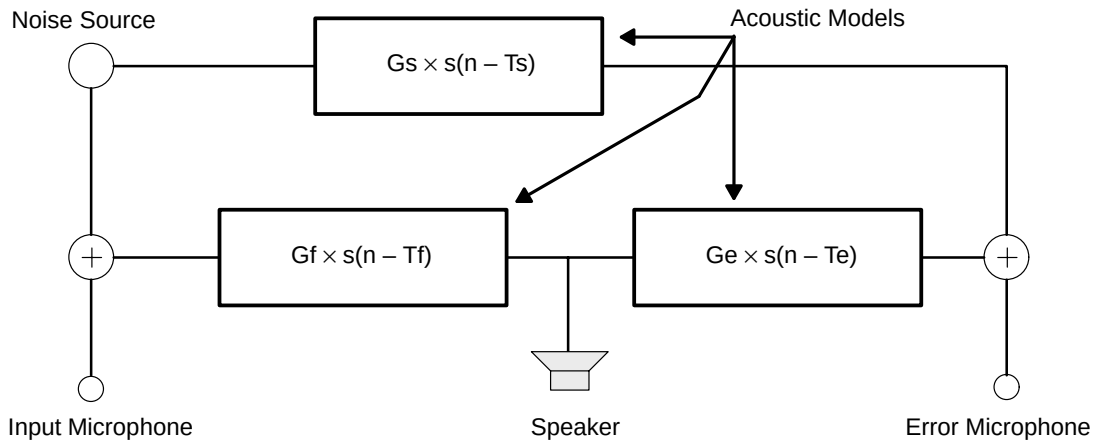


Figure 45. How Constants Are Used in Modeling Acoustic-Channel Transfer Functions

The adaptive filter order defined by the constants NAz through NWz and sample delay periods Ts through Tf (if simulation of an acoustic channel is enabled) determines the size of the input, intermediate, and output signal arrays. These arrays are defined in the file MEMORY.ASM to allocate memory space in block B1 of the 'C2x DSP's on-chip RAM. All of the arrays must fit within block B1, which is the same size as block B0. Table 8 shows which arrays are used with the different algorithms and how to compute the total amount of memory used. This table assumes that acoustic channel simulation is enabled.

Table 8. How Output Signal Arrays Are Used With Various Algorithms

ALGORITHM	SIGNALS
FXLMS	(NCz + 1) + (NWz + 1) + (Ts + 1) +(Te + 1)
FBFXLMS	(NCz + 1) + 2(NDz + 1) + (NWz + 1)
FURLMS	2(NAz + 1) + 2(NBz + 1) + 1 + (Ts + 1)
FANC	(NCz + 1) + 2(NWz + 1) + (Ts + 1)

ANC Algorithm Module Listing (anc.asm)

```
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
; This code performs both the filter calculations and the coefficient
;  adaptation.
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;  INCLUME FILES
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
   .include  config.asm
   .include  macros.asm

   .if ALGORITHM == fxlms
   .title "FXLMS Active Noise Control Algorithm"
   .elseif   ALGORITHM == fbfxlms
   .title "FBFXLMS Active Noise Control Algorithm"
   .elseif   ALGORITHM == furlms
   .title "FURLMS Active Noise Control Algorithm"
   .elseif   ALGORITHM == fanc
   .title "FANC Active Noise Control Algorithm"
   .endif

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;  PROGRAM
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

   .text

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
; Off-Line Training Routine
   .if  TIMEBASE != freerun
Training
   idle
   .else
Training
   .endif

;-----------------------------------------------------------------------
;  A1. Generate the white noise signal and output to speaker
;-----------------------------------------------------------------------
   call      Noisegen

   .if  SIMULATION == no
; Output white noise y(n) to speaker thru I/O port.
   out    wng,OUTPORT
      .if ALGORITHM == fbfxlms
   ldpk      wnn       ;Set DP to speaker output array
   sach      wnn, 1    ;Store wng to wn(n) array
   .else
   ldpk      xn        ;Set DP to speaker output array
   sach      xn, 1     ;Store wng to x(n) array
      .endif
; Read sample from error microphone: e(n) thru I/O port.
   ldpk      en        ;Set DP to speaker output array
   in     en,INPORT    ;Save latest value to memory

   .elseif  SIMULATION == yes
; Use this for simulation of error path mike input
   ldpk      yn        ;Set DP to speaker output array
   sach      yn, 1     ;Save WNG output to y(n)
      .if ALGORITHM == fbfxlms
   ldpk      wnn       ;Set DP to speaker output array
   sach      wnn, 1    ;Store wng to wn(n) array
      .else
   ldpk      xn        ;Set DP to speaker output array
```

```
   sach      xn, 1     ;Store wng to x(n) array
      .endif
;-----------------------------------------------------------------
;  S1. Call error path response simulation for error microphone
;
;       e(n) = Ge * y(n–Te)
;
;-----------------------------------------------------------------
   call      Error
   .endif
;-----------------------------------------------------------------
;
;  A2. Model the error path response
;
;              NCZ–1
;       z(n) =  sum  ci(n) * x(n–i)
;               i=0
;
;      NOTE: Use wn(n–i) in place of x(n–i) for FXLMS algorithm using
;             Acoustic Feedback (ALGORITHM == fbfxlms)
;
;-----------------------------------------------------------------
   .if ALGORITHM == fbfxlms
   lrlk      AR2,wnn+NCz–1;Initialize AR2 to end of wn(n) array
   .else
   lrlk      AR2,xn+NCz–1 ;Initialize AR2 to end of x(n) array
   .endif
   larp      AR2
   zac                    ;Zero math channel
   mpyk      0

   rptk      NCz–1        ;Compute filter output
   macd      PCi,*–       ;DMOV here.  Use shifted array in Adapt
   apac
   ldpk      zn
   sach      zn
;-----------------------------------------------------------------
;
;  A3. Compute the error path model error
;
;       e1(n) = e(n) – z(n)
;
;-----------------------------------------------------------------
   lac    en        ;Compute difference
   sub    zn        ;ACC = e(n)–z(n)
   sacl      e1n    ;Save difference

   .if  ((SIMULATION == yes) & (TRACE != no))
   larp      AR7    ;Store error in ttrace[] array in XDATA
   sacl      *+
   .endif
;-----------------------------------------------------------------
;
;  A4. Update coefficients of C(z) with LMS (or Leaky LMS) Algorithm
;
;       ci(n+1) = v*ci(n) + u*e1(n)*x(n–i), for i=0,1,..,N–1
;
;      Note: (1) use wn(n–i) in place of x(n–i) for FXLMS algorithm
;                 using Acoustic Feedback (ALGORITHM == fbfxlms)
;            (2) use v*ci(n) for Leaky LMS, or use ci(n) for LMS
;            (3) use ”+” when updating C(z) in training mode
;
;-----------------------------------------------------------------
```

```
   lt     e1n              ;Compute mu*e1(n) and store
   mpyk   MU               ; w/ Q31 scaling
   sph    adaptemp

   .if ALGORITHM == fbfxlms
   lrlk      AR2, wnn+1    ;AR2 = top of DMOV'd white noise array
   .else
   lrlk      AR2, xn+1     ;AR2 = top of DMOV'd white noise array
   .endif
   lrlk      AR3, Ci+NCz-1 ;AR3 = bottom (rev order) of C(i) array
   lark      AR4, NCz-1    ;AR4 = loop counter
   larp      AR2
   .if  PROCESSOR == C26
   conf      0             ;Put B0 into DATA space
   .elseif  PROCESSOR == C25
   cnfd                    ;Put B0 into DATA space
   .endif

   lt     adaptemp         ;T=mu*e1(n)
Trn1
   mpy    *+, AR3          ;P=mu*e1(n)*y(n-i)
   zalr      *             ;ACC=Ci(n) w/ rounding
   .if  ADAPTATION == leaky_lms
   sub    *, LEAKAGE       ;ACC=(1-1/2^15)*Ci(n)
   .endif
   apac                    ;ACC=Ci(n)+mu*e1(n)*y(n-i)
   sach      *-, AR4       ;Update Ci with new value
   banz      Trn1, AR2     ;Check if coefficient update done
   .if  PROCESSOR == C26
   conf      1             ;Put B0 into PROGRAM space
   .elseif  PROCESSOR == C25
   cnfp                    ;Put B0 into PROGRAM space
   .endif

   .if ALGORITHM == fbfxlms
;-----------------------------------------------------------------
;
;  S2. Call acoustic feedback path response simulation for input
;       microphone
;
;       x(n) = Gf * y(n-Tf)
;
;-----------------------------------------------------------------
   call      Feedback

;-----------------------------------------------------------------
;
;  A5. Model the acoustic feedback path response
;
;              NDZ-1
;       f(n) =  sum  di(n) * wn(n-i)
;               i=0
;
;      NOTE: wn(n) array DMOV'd during z(n) calculation
;
;-----------------------------------------------------------------
   lrlk      AR2,wnn+NDz   ;Initialize AR2 to end of DMOV'd wn(n) array
   larp      AR2
   zac                     ;Zero math channel
   mpyk      0

   rptk      NDz-1         ;Compute filter output
   mac    PDi,*-           ;No DMOV here.  Done in z(n) calculation.
   apac
   ldpk      fn
   sach      fn
```

```
;-----------------------------------------------------------------------
;
;  A6. Compute the acoustic feedback path model error
;
;       e2(n) = x(n) – f(n)
;
;-----------------------------------------------------------------------
   lac    xn            ;Compute difference
   sub    fn            ;ACC = x(n)–f(n)
   sacl      e2n        ;Save difference
      .if  ((SIMULATION == yes) & (TRACE != no))
   larp      AR6        ;Store error in fbtrace[] array in XDATA
   sacl      *+
      .endif
;-----------------------------------------------------------------------
;
;  A7. Update coefficients of D(z) with LMS (or Leaky LMS) Algorithm
;
;       di(n+1) = v*di(n) + u*e2(n)*wn(n–i),      for i=0,1,..,N–1
;
;      Note: (1) use v*ci(n) for Leaky LMS, or use ci(n) for LMS
;            (2) use ”+” when updating C(z) in training mode
;
;-----------------------------------------------------------------------
   lt     e2n              ;Compute mu*e2(n) and store
   mpyk      MU            ; w/ Q31 scaling
   sph    adaptemp
   lrlk      AR2, wnn+1    ;AR2 = top of DMOV’d white noise array
   lrlk      AR3, Di+NDz–1 ;AR3 = bottom (rev order) of D(i) array
   lark      AR4, NDz–1    ;AR4 = loop counter
   larp      AR2
      .if  PROCESSOR == C26
   conf      0             ;Put B0 into DATA space
      .elseif  PROCESSOR == C25
   cnfd                    ;Put B0 into DATA space
      .endif
   lt     adaptemp         ;T=mu*e2(n)
Trn2
   mpy    *+, AR3          ;P=mu*e2(n)*wn(n–i)
   zalr      *             ;ACC=Di(n) w/ rounding
      .if  ADAPTATION == leaky_lms
   sub    *, LEAKAGE       ;ACC=(1–1/2^15)*Di(n)
      .endif
   apac                    ;ACC=Di(n)+mu*e2(n)*wn(n–i)
   sach      *–, AR4       ;Update Di with new value
   banz      Trn2, AR2     ;Check if coefficient update done
      .if  PROCESSOR == C26
   conf      1             ;Put B0 into PROGRAM space
      .elseif  PROCESSOR == C25
   cnfp                    ;Put B0 into PROGRAM space
      .endif
   .endif
; Check if TRAINING complete
   larp      AR1           ;Activate counter ARP
   banz      Training
   .if  TIMEBASE == timer
   dint                    ;Disable GLOBAL interrupt
   .endif
   ret                     ;Return to main when complete
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
; On–Line Noise Control Routine
```

```
Control
   .if  SIMULATION == no
      .if ALGORITHM != fanc
;Read sample from noise source microphone x(n) thru I/O port
   ldpk      xn
   in     xn,INPORT1   ;Save latest value to beginning of array.
      .endif
;Read sample from error microphone e(n) thru I/O port
   ldpk      en
   in     en,INPORT2   ;Save latest value to memory
   .elseif  SIMULATION == yes
;-----------------------------------------------------------------
;
;  S1. Simulate noise source signal
;
;-----------------------------------------------------------------
      .if  GENERATOR == sinewave
   call      Sinewave     ;Sinewave generator
   ldpk      wnn          ;Save generator output to wn(n) array
   sach      wnn
         .if  ((ALGORITHM == fxlms) | (ALGORITHM == furlms))
   ldpk      xn           ;Save generator output to x(n) array
   sach      xn
         .endif
      .elseif  GENERATOR == white_noise
   call      Noisegen     ;Random number generator
   ldpk      wnn          ;Save generator output to wn(n) array
   sach      wnn,1
         .if  ((ALGORITHM == fxlms) | (ALGORITHM == furlms))
   ldpk      xn           ;Save generator output to x(n) array
   sach      xn, 1
         .endif
      .endif
      .if TRACE == in_out
   larp      AR7          ;Store input in gtrace[] array in XDATA
   sach      *+, 1
      .endif
   .endif
   .if  ALGORITHM == fbfxlms
;-----------------------------------------------------------------
;
;  S2. Simulate input microphone signal including acoustic feedback
;
;       x(n) = wn(n) + Gf * y(n–Tf)
;
;-----------------------------------------------------------------
   call      Feedback1
;-----------------------------------------------------------------
;
;  A1. Correct input signal for acoustic feedback
;
;                      NDZ–1
;       x1(n) = x(n) – sum  di * y(n–i)
;                       i=0
;
;-----------------------------------------------------------------
   lrlk      AR2,yn+NDz–1     ;Initialize AR2 to end of yn array
   larp      AR2
   zac                        ;Zero math channel
   mpyk      0
   rptk      NDz–1            ;Model acoustic feedback effect
   macd      PDi,*–
```

```
   apac                ;ACC = f(n)
   neg                 ;ACC = –f(n)
   addh      xn        ;ACC = x(n)–f(n)
   ldpk      x1n       ;Set DP to x1(n) array
   sach      x1n       ; and store result
   .endif

   .if  ALGORITHM == fanc
;-----------------------------------------------------------------
;
;  S2. Simulate the acoustic channel response at the error microphone
;
;       e(n) = Gs * wn(n–Ts) + Ge * y(n–Te)
;
;-----------------------------------------------------------------
   call      Source       ;Call source+error path simulation
      .if TRACE != none
   larp      AR6          ;Store output in ctrace[] array in XDATA
   sach      *+,3         ; w/ Q28 shift
      .endif

;-----------------------------------------------------------------
;
;  A1. Regenerate (extract) primary noise signal
;
;                     NCZ–1
;       x(n) = e(n) – sum  ci * y(n–i)
;                      i=0
;
;-----------------------------------------------------------------
   lrlk      AR2,yn+NCz–1 ;Initialize AR2 to end of yn array
   larp      AR2
   mpyk      0            ;Zero P register
   lalk      1,15         ;Set up rounding on x(n) calculation

   rptk      NCz–1        ;Model antinoise output at error mike
   macd      PCi,*–       ;DMOV yn here
   apac
   neg                    ;Extract xn from en=xn+yn
   addh      en           ; xn=en–yn

   ldpk      xn           ;Set DP to extracted noise source
   sach      xn           ; and store
   .endif

   .if  ALGORITHM == furlms
;-----------------------------------------------------------------
;
;  A1. Generate the anti–noise signal and output to speaker
;
;              NAZ–1                 NBZ–1
;       y(n) = sum ai(n) * x(n–i) + sum bj(n) * y(n–j)
;               i=0                   j=1
;
;      Note: there is delay=1 in B(z)
;
;-----------------------------------------------------------------
   lrlk      AR2,xn+NAz–1 ;Initialize AR2 to end of xn array
   larp      AR2
   zac                    ;Zero math channel
   mpyk      0
   rptk      NAz–1        ;Compute antinoise output
   mac    PAi,*–          ;Don’t DMOV xn here

   lrlk      AR2,yn+NBz–1 ;Initialize AR2 to end of y(n–1) array
   rptk      NBz–1        ;Compute antinoise output
```

```
   macd      PBi,*–        ;DMOV y(n–1) array here
   apac
   .elseif  ALGORITHM == fbfxlms
;-----------------------------------------------------------------
;
;  A2. Generate the antinoise signal and output to speaker
;
;             NWZ–1
;       y(n) = sum wi(n) * x1(n–i)
;              i=0
;
;-----------------------------------------------------------------
   lrlk      AR2,x1n+NWz–1    ;Initialize AR2 to end of x1n array
   .elseif  ALGORITHM == fxlms
;-----------------------------------------------------------------
;
;  A1. Generate the antinoise signal and output to speaker
;
;             NWZ–1
;       y(n) = sum wi(n) * x(n–i)
;              i=0
;
;-----------------------------------------------------------------
   lrlk      AR2,xn+NWz–1     ;Initialize AR2 to end of xn array
   .elseif  ALGORITHM == fanc
;-----------------------------------------------------------------
;
;  A2. Generate the antinoise signal and output to speaker
;
;             NWZ–1
;       y(n) = sum wi(n) * x(n–i)
;              i=0
;
;-----------------------------------------------------------------
   lrlk      AR2,xn+NWz–1     ;Initialize AR2 to end of xn array
   .endif

   .if ALGORITHM != furlms
   larp      AR2
   mpyk      0                ;Zero P register
      .if ALGORITHM == fanc
   lalk      1,15             ;Used for rounding
      .endif
   rptk      NWz–1            ;Compute antinoise output
   mac    PWi,*–              ;Don't DMOV xn here
   apac
   .endif

   ldpk      yn               ;Set DP to adaptive filter output
   sach      yn               ; and store

   .if  SIMULATION == no
;Output yn source to speaker
   out    yn,OUTPORT

   .elseif  ((SIMULATION == yes) & (ALGORITHM != fanc))
      .if ALGORITHM == fbfxlms
;-----------------------------------------------------------------
;
;  S3. Simulate the acoustic channel response at the error microphone
;
;       e(n) = Gs * wn(n–Ts) + Ge * y(n–Te)
;
;-----------------------------------------------------------------
      .else
```

```
;-----------------------------------------------------------------------
;
;  S2. Simulate the acoustic channel response at the error microphone
;
;       e(n) = Gs * wn(n–Ts) + Ge * y(n–Te)
;
;-----------------------------------------------------------------------
      .endif
   call      Source        ;Call source+error path simulation
      .if TRACE != none
   larp      AR6           ;Store output in ctrace[] array in XDATA
   sach      *+,3          ; w/ Q28 shift
      .endif
   .endif

   .if ALGORITHM == fbfxlms
;-----------------------------------------------------------------------
;
;  A3. Correct the input signal for error path delay
;
;              NCZ–1
;       x2(n) = sum ci * x1(n–i)
;               i=0
;
;-----------------------------------------------------------------------
   lrlk      AR2,x1n+NCz–1    ;Initialize AR2 to end of x1n array
   .elseif  ALGORITHM == furlms
;-----------------------------------------------------------------------
;
;  A2a. Correct the input signal for error path delay
;
;              NCZ–1
;       x1(n) = sum ci * x(n–i)
;               i=0
;
;-----------------------------------------------------------------------
   lrlk      AR2,xn+NCz–1     ;Initialize AR2 to end of xn array
   .elseif  ALGORITHM == fxlms
;-----------------------------------------------------------------------
;
;  A2. Correct the input signal for error path delay
;
;              NCZ–1
;       x1(n) = sum ci * x(n–i)
;               i=0
;
;-----------------------------------------------------------------------
   lrlk      AR2,xn+NCz–1     ;Initialize AR2 to end of xn array
   .elseif  ALGORITHM == fanc
;-----------------------------------------------------------------------
;
;  A3. Correct the input signal for error path delay
;
;              NCZ–1
;       x1(n) = sum ci * x(n–i)
;               i=0
;
;-----------------------------------------------------------------------
   lrlk      AR2,xn+NCz–1     ;Initialize AR2 to end of xn array
   .endif

   larp      AR2
   mpyk      0                ;Zero P register
   lalk      1,15             ;Set up rounding for x1(n) calculation
```

```
   rptk      NCz–1            ;Compute filter output
   macd      PCi,*–           ;DMOV input array here
   apac
   .if ALGORITHM == fbfxlms
   ldpk      x2n
   sach      x2n              ;Store x2(n) to top of array
   .else
   ldpk      x1n
   sach      x1n              ;Store x1(n) to top of array
   .endif

   .if ALGORITHM == furlms
;-----------------------------------------------------------------
;
;  A2b. Correct the delayed output signal for error path delay
;
;                 NCZ–1
;        y1(n–1) = sum cj * y(n–j)
;                   j=1
;
;-----------------------------------------------------------------
   lrlk      AR2,yn+NCz    ;Initialize AR2 to end of DMOV'd yn array
   larp      AR2
   zac                     ;Zero math channel
   mpyk      0

   rptk      NCz–1         ;Compute filter output
   mac    PCi,*–           ;Don't DMOV yn array here
   apac
   ldpk      y1n
   sach      y1n           ;Store y1(n) to top of array
   .endif

   .if ALGORITHM == furlms
;-----------------------------------------------------------------
;
;  A3a. Update coefficients of A(z) with LMS (or Leaky LMS) Algorithm
;
;       ai(n+1) = v*ai(n) – u*e(n)*x1(n–i), for i=0,1,..,N–1
;
;      Note: (1) use v*ai(n) for Leaky LMS, or use ai(n) for LMS
;            (2) use "–" when updating A(z) in noise control mode
;              instead of "+" used to update C(z) in training mode
;-----------------------------------------------------------------
   ldpk      en
   lt     en               ;Compute mu*e(n) and store
   mpyk      MU            ; w/Q31 scaling
   sph    adaptemp
   larp      AR2
   lrlk      AR2,x1n+NAz–1 ;AR2 = bottom of x1(n) array
   lrlk      AR3,Ai        ;AR3 = top/end (rev order) of Ai array
   lark      AR4,NAz–1     ;AR4 = loop counter
      .if  PROCESSOR == C26
   conf      0             ;Put B0 into DATA space
      .elseif  PROCESSOR == C25
   cnfd                    ;Put B0 into DATA space
      .endif

   lt     adaptemp         ;T=mu*e(n)
Control1
   mpy    *                ;P=mu*e(n)*x1(n–i)
   dmov      *–, AR3       ;DMOV x1n array
   zalr      *             ;ACC=Ai(n) w/ rounding
      .if  ADAPTATION == leaky_lms
   sub    *, LEAKAGE       ;ACC=(1–1/2^15)*Ai(n)
```

```
      .endif
   spac                        ;ACC=Ai(n)–mu*e(n)*x1(n–i)
   sach      *+, AR4           ;Update Ai with new value
   banz      Control1, AR2     ;Check if coefficient update done
      .if  PROCESSOR == C26
   conf      1                 ;Put B0 into PROGRAM space
      .elseif  PROCESSOR == C25
   cnfp                        ;Put B0 into PROGRAM space
      .endif
;-----------------------------------------------------------------------
;
;  A3b. Update coefficients of B(z) with LMS (or Leaky LMS) Algorithm
;
;       bi(n+1) = v*bi(n) – u*e(n)*y1(n–i), for i=1,2,..,N–1
;
;      Note: (1) use v*bi(n) for Leaky LMS, or use bi(n) for LMS
;            (2) use ”–” when updating B(z) in noise control mode
;              instead of ”+” used to update C(z) in training mode
;-----------------------------------------------------------------------
   larp      AR2
   lrlk      AR2,y1n+NBz–1     ;AR2 = bottom of y1n array
   lrlk      AR3,Bi            ;AR3 = top/end (rev order) of Bi array
   lark      AR4,NBz–1         ;AR4 = loop counter
      .if  PROCESSOR == C26
   conf      0                 ;Put B0 into DATA space
      .elseif  PROCESSOR == C25
   cnfd                        ;Put B0 into DATA space
      .endif

   lt     adaptemp             ;T=mu*e(n)
Control2
   mpy    *                    ;P=mu*e(n)*y1(n–i)
   dmov      *–, AR3           ;DMOV xn array
   zalr      *             ;ACC=Bi(n) w/ rounding
      .if  ADAPTATION == leaky_lms
   sub    *, LEAKAGE           ;ACC=(1–1/2^15)*Bi(n)
      .endif
   spac                        ;ACC=Bi(n)–mu*e(n)*y1(n–i)
   sach      *+, AR4           ;Update Bi with new value
   banz      Control2, AR2     ;Check if coefficient update done
      .if  PROCESSOR == C26
   conf      1                 ;Put B0 into PROGRAM space
      .elseif  PROCESSOR == C25
   cnfp                        ;Put B0 into PROGRAM space
      .endif

   .elseif  ALGORITHM != furlms
      .if ALGORITHM == fbfxlms
;-----------------------------------------------------------------------
;
;  A4. Update coefficients of W(z) with LMS (or Leaky LMS) Algorithm
;
;       wi(n+1) = v*wi(n) – u*e(n)*x2(n–i), for i=0,1,..,N–1
;
;      Note: (1) use v*wi(n) for Leaky LMS, or use wi(n) for LMS
;            (2) use ”–” when updating W(z) in noise control mode
;              instead of ”+” used to update C(z) in training mode
;-----------------------------------------------------------------------
      .elseif  ALGORITHM == fxlms
;-----------------------------------------------------------------------
;
;  A3. Update coefficients of W(z) with LMS (or Leaky LMS) Algorithm
;
;       wi(n+1) = v*wi(n) – u*e(n)*x1(n–i), for i=0,1,..,N–1
```

```
;
;      Note: (1) use v*wi(n) for Leaky LMS, or use wi(n) for LMS
;            (2) use "–" when updating W(z) in noise control mode
;              instead of "+" used to update C(z) in training mode
;-----------------------------------------------------------------
      .elseif  ALGORITHM == fanc
;-----------------------------------------------------------------
;
;  A4. Update coefficients of W(z) with LMS (or Leaky LMS) Algorithm
;
;       wi(n+1) = v*wi(n) – u*e(n)*x1(n–i), for i=0,1,..,N–1
;
;      Note: (1) use v*wi(n) for Leaky LMS, or use wi(n) for LMS
;            (2) use "–" when updating W(z) in noise control mode
;              instead of "+" used to update C(z) in training mode
;-----------------------------------------------------------------
      .endif
   ldpk      en
   lt     en                  ;Compute mu*e(n) and store
   mpyk      MU                ; w/Q31 scaling
   sph    adaptemp
   larp      AR2
      .if ALGORITHM == fbfxlms
   lrlk      AR2, x2n+NWz–1   ;AR2 = bottom of x2n array
      .else
   lrlk      AR2, x1n+NWz–1   ;AR2 = bottom of x1n array
      .endif
   lrlk      AR3,   Wi        ;AR3 = top/end (rev order) of Wi array
   lark      AR4, NWz–1       ;AR4 = loop counter
      .if  PROCESSOR == C26
   conf      0                ;Put B0 into DATA space
      .elseif  PROCESSOR == C25
   cnfd                       ;Put B0 into DATA space
      .endif


   lt     adaptemp            ;T=mu*e(n)
Control1
      .if ALGORITHM == fbfxlms
   mpy    *                   ;P=mu*e(n)*x2(n–i)
      .else
   mpy    *                   ;P=mu*e(n)*x1(n–i)
      .endif
   dmov      *–, AR3          ;DMOV x1n array
   zalr      *                ;ACC=Wi(n) w/ rounding
      .if  ADAPTATION == leaky_lms
   sub    *, LEAKAGE          ;ACC=(1–1/2^15)*Wi(n)
      .endif
      .if ALGORITHM == fbfxlms
   spac                       ;ACC=Wi(n)–mu*e(n)*x2(n–i)
      .else
   spac                       ;ACC=Wi(n)–mu*e(n)*x1(n–i)
      .endif
   sach      *+, AR4          ;Update Wi with new value
   banz      Control1, AR2    ;Check if coefficient update done
      .if  PROCESSOR == C26
   conf      1                ;Put B0 into PROGRAM space
      .elseif  PROCESSOR == C25
   cnfp                       ;Put B0 into PROGRAM space
      .endif
   .endif


   ret                        ;Return to main when complete
```

ANC Linker Command File (anc.cmd)

```
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
/*        ANC Linker Command File                                        */
/*        -----------------------                                        */
/*                                                                       */
/* File:     ANC.CMD              Rev: 1.0                               */
/* Last Change: 8/26/93         Start Date:  8/3/93                      */
/*                                                                       */
/* Processor:   TMS320C25                                                */
/* Assembler Rev:   6.40                                                 */
/*                                                                       */
/* Programmer:  Thomas G. Horner                                         */
/*           TI – Dallas RTC                                             */
/*           (214) 917–5051                                              */
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
/* Linker command file for Active Noise Cancellation for TMS320C25 */
/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/

/* INPUT/OUTPUT  */
vectors.obj
main.obj
init.obj
anc.obj
memory.obj
model.obj    /* This module for simulation only. */

–m anc.map
–o anc.out

/* PHYSICAL MEMORY DEFINITION  */
MEMORY
{
    PAGE 0 : XVECS: origin = 00000h,  length = 00008h
             PVECS: origin = 00018h,  length = 00008h
           PROG:      origin = 00020h,  length = 07fd0h
           PRAMB0:  origin = 0fa00h,  length = 00200h
             PRAMB1: origin = 0fc00h,  length = 00200h
             PRAMB3: origin = 0fe00h,  length = 00200h

    PAGE 1 : REGS:       origin = 00000h,  length = 00006h
             RAMB0:  origin = 00200h,  length = 00200h
             RAMB1:  origin = 00400h,  length = 00200h
             RAMB2:  origin = 00060h,  length = 00020h
             RAMB3:  origin = 00600h,  length = 00200h
             XDATA:  origin = 08000h,  length = 07400h
}

/* S/W MODULE ALLOCATION TO MEMORY  */
SECTIONS
{
   x_vecs :        { }   > XVECS   PAGE 0  /* External interrupt vecs    */
   p_vecs :        { }   > PVECS   PAGE 0  /* Internal interrupt vecs    */
   .text  :        { }   > PROG    PAGE 0  /* Code                       */
   .data  :        { }   > PROG    PAGE 0  /* Data table                 */
   traps  :        { }   > PROG    PAGE 0  /* Unused interrupt traps     */
   anc_pma:        { }   > PRAMB0  PAGE 0  /* Coeffcient arrays (PROG)   */
   anc_coef:       { }   > RAMB0   PAGE 1  /* Coeffcient arrays (DATA)   */
   anc_vars:       { }   > RAMB1   PAGE 1  /* Input/Output arrays        */
   .bss   :        { }   > RAMB2   PAGE 1  /* General purpose variables  */
   verify :        { }   > XDATA   PAGE 1  /* Simulation output          */
}
```

ANC System Configuration File (config.asm)

```
   .title "ANC System Configuration"
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;            ANC System Supervisor
;            ---------------------
;
;  File:     CONFIG.ASM            Rev: 1.0
;  Last Change:  10/13/93        Start Date:  10/13/93
;
;  Processor:   TMS320C25
;  Language:    Assembly
;  Assembler Rev:   6.40
;
;  Programmer:  Thomas G. Horner
;            TI - Dallas RTC
;            (214) 917-5051
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
; Active Noise Cancellation system configuration module.  The settings
;  defined in this module determine how the code is generated.
;  Conditional assembly allows easy configuration for different
;  processors, I/O, adaptive filter forms, and adaptation methods.
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;  INCLUDE FILES
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
   .include  globals.asm

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;  CONFIGURATION SETTINGS
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
; This section is used to configure the S/W for different I/O, adaptive
;  filter forms, and adaptation methods.  Set the following constants to
;  the desired value to control the configuration.  Explanations of the
;  constants and settings are given at the end of this module.
;
PROCESSOR .set      C26
TIMEBASE      .set      freerun
SIMULATION   .set      yes
TRACE     .set      in_out
GENERATOR .set      white_noise
ALGORITHM .set      fxlms
ADAPTATION   .set      leaky_lms

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
; This section contains constants which define the characteristics of the
;  algorithm and simulation (if enabled).
;
CLKOUT    .set 10000000    ;DSP instruction clock rate
FS        .set   1500      ;Sample rate (samples/sec)
TIME      .set 2           ;Number of seconds for error path coef training
NSAMPLES  .set   TIME*FS   ;Number of samples for error path coef training
NTICKS    .set (CLKOUT/FS)-1  ;TIMER period
MU        .set 1024        ;Coefficient update stepsize (Q31): MU<1/NWz
LEAKAGE      .set   1      ;Shift value to leak off 2^-15 of coefficient

NAz       .set   64        ;IIR feedforward filter order A(z)
NBz       .set   64        ;IIR feedback filter order B(z)
NCz       .set   64        ;Error path model order C(z)
NDz       .set   64        ;Acoustic feedback model order D(z)
NWz       .set   64        ;FIR filter order W(z)

SIMLENGTH .set 3000        ;Cancellation routine simulation run time
                             ; (number of loops through the routine)
```

```
;!!!!!!!! ASSUMPTION: Te<Tf and Ts=Te+Tf !!!!!!!!!
Gs        .set 2458    ;Noise source path gain (Q27): 0.6*2^12
Ge        .set 3277    ;Error path gain (Q27): 0.8*2^12
Gf        .set 2867    ;Acoustic feedback path gain (Q27): 0.7*2^12
Ts        .set 10      ;Noise source path sample delay period
Te        .set  2      ;Error path sample delay period
Tf        .set Ts–Te   ;Acoustic feedback path sample delay period

; I/O Port Definitions
INPORT1      .set  PA0    ;Input port for input mike defined
INPORT2      .set  PA1    ;Input port for error mike defined
OUTPORT      .set  PA1    ;Output port for speaker defined

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;    SWITCH SETTING EXPLANATIONS
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;
; CONSTANT         SETTINGS           MEANING
;------------     ---------------     --------------------------------
; PROCESSOR        C26                 Target processor is TMS320C26
;                  C25                 Target processor is TMS320C25
;
; TIMEBASE         timer               Onboard timer controls sample rate
;                  external            Ext interrupt controls sample rate
;                  freerun             Run as fast as possible
;
; SIMULATION       yes                 Simulation mode enabled
;                  no                  Simulation mode disabled
;
;
; TRACE            in_out              Trace algorithm input and output
;                  out                 Trace algorithm output only
;                  none                No trace
;
; GENERATOR        white_noise         White noise generator enabled
;                  sinewave            Sinewave generator enabled
;
; ALGORITHM        fxlms               Filtered–X LMS enabled
;                  fbfxlms           FXLMS w/ acoustic feedback enabled
;                  furlms              Filtered–U recursive LMS enabled
;                  fanc                Feedback ANC enabled
;
; ADAPTATION       lms                 Std LMS coef adaptation enabled
;                  leaky_lms           Leaky LMS coef adaptation enabled
;
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

TMS320C2x EVM Initialization Command File (evminit.cmd)

```
;Map File for EVM2X: PROG space 0–8K, DATA space 8K–16K
ma 0x0000,0,0x8000,ram  ;external program memory
ma 0xfa00,0,0x0600,ram  ;program memory block B0, B1, and B3
ma 0x0000,1,0x0006,ram  ;memory mapped registers
ma 0x0060,1,0x20,ram    ;data memory block B2
ma 0x0200,1,0x0600,ram  ;data memory block B0, B1, and B3
ma 0x8000,1,0x7400,ram  ;external data memory
ma 0xfa00,1,0x600,ram   ;program memory block B0, B1, and B3
ma 0x0000,2,16,ioport   ;io ports
map on

DASM PC
load anc
sconfig ancsim.cfg
mem1 0x200
mem2 0x400
mem3 0x600
;ba Wait
;run
;take savemem.bat
```

Global Constants and Variables (globals.asm)

```
   .title "Global Constants and Variables"
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;            Global Constants and Variables
;              ------------------------------
;
;   File:     GLOBALS.ASM           Rev: 1.0
;   Last Change:  11/9/93          Start Date:   8/3/93
;
;   Processor:    TMS320C25
;   Language:     Assembly
;   Assembler Rev:   6.40
;
;   Programmer:   Thomas G. Horner
;            TI – Dallas RTC
;            (214) 917–5051
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
; Global constant and variable declarations.
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
   .mmregs
   .fcnolist
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;  CONSTANT DEFINITIONS
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
; Onchip memory block addresses for TMS320C25/C26
C2X_B0    .set 0200h       ;Starting address of block B0 for all
C25_B1    .set 0300h       ;Starting address of block B1 for C25
C26_B1    .set 0400h       ;Starting address of block B1 for C26
C2X_B2    .set    060h     ;Starting address of block B2 for all
C26_B3    .set 0600h    ;  Starting address of block B3 for C26
; Onchip memory block lengths
C2X_SHORT .set  020h       ;B2 length (32) – all
C25_LONG     .set    100h  ;B0 and B1 length (256) – C25
C26_LONG     .set  200h    ;B0, B1 and B3 length (512) – C26
; External DATA memory block description
XDATA_START  .set  08000h ;Ext DATA RAM origin
XDATA_SIZE   .set  07000h ;Ext DATA RAM length
```

```
; Interrupt Mask Register (IMR) enable/disable masks
ENABLE_int0  .set 01h      ;OR value w/ IMR to enable ext interrupt #0
DISABLE_int0 .set 0fffeh   ;AND value w/ IMR to disable ext interrupt #0
ENABLE_int1  .set 02h      ;OR value w/ IMR to enable ext interrupt #1
DISABLE_int1 .set 0fffdh   ;AND value w/ IMR to disable ext interrupt #1
ENABLE_int2  .set 04h      ;OR value w/ IMR to enable ext interrupt #2
DISABLE_int2 .set 0fffbh   ;AND value w/ IMR to disable ext interrupt #2
ENABLE_tint  .set 08h      ;OR value w/ IMR to enable timer
DISABLE_tint .set 0fff7h   ;AND value w/ IMR to disable timer
ENABLE_rec   .set 010h     ;OR value w/ IMR to enable serial port rec
DISABLE_rec  .set 0ffefh   ;AND value w/ IMR to disable serial port rec
ENABLE_xmit  .set 020h     ;OR value w/ IMR to enable serial port xmit
DISABLE_xmit .set 0ffdfh   ;AND value w/ IMR to disable serial port xmit

; System configuration constants
C25       .set      25     ;Processor switch settings
C26       .set      26

timer     .set      2      ;Sample rate / time base control settings
external      .set      1
freerun       .set      0

yes       .set      1      ;Simulation enable switch settings
no        .set      0

in_out    .set      2      ;Trace buffer switch settings
out       .set      1
none      .set      0

white_noise  .set      1   ;Waveform generator switch settings
sinewave     .set      0

fxlms     .set      3      ;Algorithm switch settings
fbfxlms       .set      2
furlms    .set      1
fanc      .set      0

lms       .set      1      ;Adaptation method switch settings
leaky_lms .set      0

; Random Number Generator
SEED      .set 12357       ;Random number generator seed value (Train)
SEED2     .set 53210       ;Random number generator seed value (Simulate)
MASK      .set 08805h      ;Mask to extract bits 0, 2, 11, 15

; Sinewave generator coefficients(for fs=1.5 kHz and fd=0.15 kHz)
A1d2      .set 0678dh      ;A/2=cos(2*pi*fd/fs)=0.80902
S10       .set 0    ;s(n–2) IC = 0
S11       .set 04b3ch      ;s(n–1) IC =sin(2*pi*fd/fs)=0.58779

; Sinewave generator coefficients(for fs=1.5 kHz and fd=0.25 kHz)
A2d2      .set 04000h   ;A/2=cos(2*pi*fd/fs)=0.5
S20       .set 0    ;s(n–2) IC = 0
S21       .set 06ed9h      ;s(n–1) IC =sin(2*pi*fd/fs)=0.86603

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;  VARIABLE DECLARATIONS
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

; Global constants for use in HLL debugger control
   .global   SIMLENGTH
   .global   NAz
   .global   NBz
   .global   NCz
   .global   NDz
   .global   NWz
   .global   NSAMPLES

; Global variables/constants for CONFIG.ASM
   .global   C26
```

```
    .global   C25
    .global   yes
    .global   no
    .global   in_out
    .global   out
    .global   none
    .global   white_noise
    .global   sinewave
    .global   fir
    .global   iir
    .global   fxlms
    .global   fbfxlms
    .global   furlms
    .global   fanc
    .global   lms
    .global   leaky_lms
    .global   PROCESSOR
    .global   SIMULATION
    .global   TRACE
    .global   GENERATOR
    .global   ALGORITHM
    .global   FILTER
    .global   ADAPTATION
    .global   INPORT
    .global   OUTPORT
    .global   Gs
    .global   Ge
    .global   Gf
    .global   Ts
    .global   Te
    .global   Tf

; Global variables from VECTORS.ASM
    .global   Reset     ;Reset vector
    .global   Int0         ;External interrupt #0
    .global   Int1         ;External interrupt #1
    .global   Int2         ;External interrupt #2
    .global   Tint         ;Serial port – transmit
    .global   Rint         ;Serial port – receive
    .global   Xint         ;Timer
    .global   Trap         ;Trap

; Global variables from MEM_DEF.ASM
    .global   Ai
    .global   Bi
    .global   Ci
    .global   Di
    .global   Wi
    .global   PAi
    .global   PBi
    .global   PCi
    .global   PDi
    .global   PWi
    .global   wnn
    .global   xn
    .global   x1n
    .global   x2n
    .global   yn
    .global   y1n
    .global   en
    .global   e1n
    .global   e2n
    .global   zn
    .global   fn
```

```
   .global   adaptemp
   .global   wng
   .global   ctrace
   .global   gtrace
   .global   ttrace
   .global   fbtrace

; Global variables from ANC.ASM
   .global   Init
   .global   Training
   .global   Reinit
   .global   Control
   .global   Wait

; Global variables from MODEL.ASM
   .global   Error
   .global   Source
   .global   Feedback
   .global   Feedback1
   .global   Noisegen
   .global   Sineinit
   .global   Sinewave
   .global   a1
   .global   s1n_1
   .global   s1n_2
   .global   a2
   .global   s2n_1
   .global   s2n_2
```

System Initialization File (init.asm)

```
   .title "Intialization"

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;            Processor Initialization
;              ------------------------
;
;  File:     INIT.ASM        Rev:         1.0
;  Last Change:  12/9/93         Start Date:  8/3/93
;
;  Processor:   TMS320C25
;  Language:    Assembly
;  Assembler Rev:   6.40
;
;  Programmer:  Thomas G. Horner
;            TI – Dallas RTC
;            (214) 917–5051
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
; Processor, system and algorithm initialization routines
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;  INCLUDE FILES
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
   .include  config.asm
   .include  macros.asm

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;  PROGRAM
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

   .text

;-------------------------------------------------------------------------
;  PROCESSOR INITIALIZATION
```

```
;-----------------------------------------------------------------------
Init
; Disable all interrupts
   ldpk      0            ;DP = mmregs data page
   lack      0            ;Reset IMR bits to disable interrupts
   sacl      IMR

; Disable OVERFLOW mode (ACC won't saturate)
   rovm

; Setup P register shift mode
   spm    1               ;Shift left 1 on PREG ==> ACC

; Enable sign extention mode
   ssxm

; Initialize Serial Port
   fort      0            ;Set for 16-bit word operation
   sfsm                   ;Set for frame sync control
   rtxm                   ;Set for external Xmit frame sync

; Clear onchip memory to initialize
   .if  PROCESSOR == C26
   memclear  C26          ;Macro call to clear C26 RAM blocks
   conf      1            ;RAM B0 ==> PROGRAM space
   .elseif  PROCESSOR == C25
   memclear  C25          ;Macro call to clear C26 RAM blocks
   cnfp                   ;RAM B0 ==> PROGRAM space
   .endif

;-----------------------------------------------------------------------
;  SYSTEM INITIALIZATION
;-----------------------------------------------------------------------
   .if TRACE != none
; Clear external DATA RAM on C2x EVM
   larp      AR0
   lrlk      AR0, XDATA_START    ;AR0 = Ext. DATA RAM pointer
   lrlk      AR1, XDATA_SIZE-1   ;AR1 = Memory block size counter
   zac
Again
   sacl      *+,AR1
   banz      Again,AR0
   .endif

;-----------------------------------------------------------------------
;  ALGORITHM INITIALIZATION
;-----------------------------------------------------------------------
; Initialize starting value for white noise generator.
   ldpk      wng          ;Set DP for random number
   lalk      SEED         ;Initialize WNG w/ seed value
   sacl      wng

; Initialize sample counter to set training period
   lrlk      AR1,NSAMPLES ;Initialize AR1 for sample count

   .if  ((SIMULATION == yes) & (TRACE != none))
; Initialize pointer to trace buffer for simulation
   lrlk      AR7, ttrace  ;AR7 = training error trace buffer pntr
      .if  ALGORITHM == fbfxlms
   lrlk      AR6, fbtrace ;AR6 = training feedback trace buffer pntr
      .endif
   .endif

   .if  TIMEBASE == timer
; Initialize timer period to set sample rate
   ldpk      0            ;Set DP for memory mapped regs
   lalk      NTICKS       ;Initialize PERIOD register for Fs setting
   sacl      PRD
```

```
; Enable timer interrupt
   lac    IMR              ;Load current Interrupt Mask Register setting
   ork    ENABLE_tint      ;Set TIMER control bit
   sacl      IMR           ;Store back to IMR
   .endif

   .if  TIMEBASE != freerun
; Enable global interrupt
   eint
   .endif

   ret                     ;Return to main when complete

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;  SIGNAL ARRAY REINITIALIZATION
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Reinit
;Re-initialize input array to zero for cancellation mode
   .if ALGORITHM == fbfxlms
   lrlk      AR1, wnn         ;AR1 = top of white noise array
   .else
   lrlk      AR1, xn          ;AR1 = top of white noise array
   .endif
   larp      AR1
   zac              ;ACC = 0
   rptk      NCz              ;Repeat (NCz+1)-1 times where NCz<=255
   sacl      *+               ;Clear input array

;Re-initialize output array to zero for cancellation mode
   lrlk      AR1, yn          ;AR1 = top of speaker output array
   larp      AR1
   zac                        ;ACC = 0
   rptk      Tf               ;Repeat Tf+1 times where Tf<=255
   sacl      *+               ;Clear output array

   .if SIMULATION == yes
; Initialize system input generator (White Noise or Sine)
      .if GENERATOR == white_noise
   ldpk      wng              ;Set DP for white noise
   lalk      SEED2            ;Init white noise generator w/ seed value
   sacl      wng
      .elseif GENERATOR == sinewave
   call      Sineinit         ;Initialize sinewave generator
      .endif

; Initialize counters and pointers for simulation period and trace buffers
   lrlk      AR5, SIMLENGTH-1 ;AR5 = simulation run counter
      .if  TRACE == in_out
   lrlk      AR6, ctrace      ;AR6 = error mike trace buffer pntr
   lrlk      AR7, gtrace      ;AR7 = source mike trace buffer pntr
      .elseif  TRACE == out
   lrlk      AR6, ctrace      ;AR6 = error mike trace buffer pntr
      .endif
   .endif

   .if TIMEBASE != freerun
   eint                       ;Enable GLOBAL interrupt
   .endif
   ret                        ;Return to Main when complete
```

Macro Library File (macros.asm)

```
   .title "Macro Library"
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;            Macro Definitions
;              -----------------
;
;  File:     MACROS.ASM            Rev: 1.0
;  Last Change:  10/11/93         Start Date:   10/11/93
;
;  Processor:   TMS320C25
;  Language:    Assembly
;  Assembler Rev:   6.40
;
;  Programmer:  Thomas G. Horner
;            TI – Dallas RTC
;            (214) 917–5051
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
; Macro definitions
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;  MACRO: CONF
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
; Required for C26 using R6.40 of ASSEMBLER.  Can't use the –v26 switch,
; so CONF instruction is illegal.  Causes problems w/ LARK, LARP, LRLK.
;
; This command modifies ST1 bits 7 & 12 to configure RAM blocks as DATA
; or PROGRAM.  Same result as using CONF instruction w/ –v26 switch.
conf   .macro x
   .if    (x == 0)
      .word  0ce3ch
   .elseif   (x == 1)
      .word  0ce3dh
   .elseif   (x == 2)
      .word  0ce3eh
   .elseif   (x == 3)
      .word  0ce3fh
   .endif
      .endm

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;  MACRO: MEMCLEAR
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
; This macro is designed to generate code to clear the onchip RAM blocks
;  for either a TMS320C25 or TMS320C26 during initialization.

memclear  .macro n
   .if    (n == 25)        ;Zero onchip RAM memory for C25
   cnfd                    ;Make all onchip RAM blocks DATA
   zac                     ;Zero accumulator
   larp      AR0           ;Point to AR
   lark      AR0, C2X_B2   ;Point to B2
   rptk      C2X_SHORT–1   ;Memory block length–1
   sacl      *+            ;Fill B2 with 0's
   lrlk      AR0, C2X_B0   ;Point to B0
   rptk      C25_LONG–1    ;Memory block length–1
   sacl      *+            ;Fill B0 with 0's
   rptk      C25_LONG–1    ;Memory block length–1
   sacl      *+            ;Fill B1 with 0's
   .elseif   (n == 26)     ;Zero onchip RAM memory for C26
   conf      0             ;Make all onchip RAM blocks DATA
   zac                     ;Zero accumulator
   larp      AR0       ;Activate AR0
```

```
   lark      AR0, C2X_B2      ;Point to B2 RAM block
   rptk      C2X_SHORT–1      ;Memory block length–1
   sacl      *+               ;Fill B2 with 0's
   lrlk      AR0, C26_LONG–1  ;AR0 = Memory block length–1
   lrlk      AR1, C2X_B0      ;AR1 = B0 pointer
   lrlk      AR2, C26_B1      ;AR2 = B1 pointer
   lrlk      AR3, C26_B3      ;AR3 = B3 pointer
   larp      AR1
Zero?
   sacl      *+, AR2          ;Fill B0,B1,B3 RAM with 0's
   sacl      *+, AR3
   sacl      *+, AR0
   banz      Zero?, AR1       ;Done??
   .else                      ;Generate error if not C25 or C26
   .emsg  "ERROR – "Incorrect device.  Use 25 or 26."
   .endif
      .endm
```

ANC System Supervisor Program (main.asm)

```
   .title "ANC System Supervisor"
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;            ANC System Supervisor
;            ---------------------
;
;  File:     MAIN.ASM              Rev: 1.0
;  Last Change:  10/12/93        Start Date:  8/3/93
;
;  Processor:   TMS320C25
;  Language:    Assembly
;  Assembler Rev:   6.40
;
;  Programmer:  Thomas G. Horner
;            TI – Dallas RTC
;            (214) 917–5051
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
; Active Noise Cancellation system supervisor. This code controls the
;  overall operation of the ANC system.
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;  INCLUDE FILES
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
   .include  config.asm
   .include  macros.asm

      .text
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;  PROGRAM
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Reset
   call      Init          ;Call system initialization routine
   call      Training      ;Call error path coef training routine
   call      Reinit        ;Call cancellation initialization routine

; Main routine
Main
   .if  TIMEBASE == timer
   idle                    ;Wait for Interrupt
    .endif
   call      Control       ;Call noise cancellation routine

   .if  SIMULATION == no
   b      Main             ;Endless loop
```

```
   .elseif  SIMULATION == yes
   larp      AR5          ;Activate simulation run counter
   banz      Main         ;Check if simulation is complete
   .if  TIMEBASE == timer
   dint                   ;Disable Global interrupt
   .endif
Wait
   b     Wait             ;Endless loop
   .endif
   .if  TIMEBASE == external
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;  INTERRUPT SERVICE ROUTINES
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;
; The External Interrupt #0 ISR is used to pace the Training and
; Control routines.  None of the algorithm is executed in the ISR.
Int0
   eint                   ;Re–enable GLOBAL interrupt
   ret                    ;Return to MAIN
   .endif
   .if  TIMEBASE == timer
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;  INTERRUPT SERVICE ROUTINES
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;
; The Timer ISR is used to pace the Training and Control routines.
;  None of the algorithm is executed in the ISR.
Tint
   eint                   ;Re–enable GLOBAL interrupt
   ret                    ;Return to MAIN
   .endif
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
   .end
```

Memory Definitions File (memory.asm)

```
   .title "Memory Definitions"
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;            Memory Definitions
;            ------------------
;
;  File:     MEMORY.ASM            Rev: 1.0
;  Last Change:  10/9/93         Start Date:  8/3/93
;
;  Processor:   TMS320C25
;  Language:    Assembly
;  Assembler Rev:   6.40
;
;  Programmer:  Thomas G. Horner
;            TI – Dallas RTC
;            (214) 917–5051
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
; Initialized and uninitialized memory definitions
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;  INCLUDE FILES
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
   .include  config.asm
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;  INITIALIZED MEMORY (PROG)
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
; BLOCK B0
; --------
; Labels to B0 coefficient arrays in PROGRAM space.
PCi    .sect "anc_pma"     ;Ci(n) in PROGRAM space
       .space 16*NCz
   .if  ALGORITHM == fbfxlms
PDi    .sect "anc_pma"     ;Di(n) in PROGRAMspace
       .space 16*NDz
   .endif
   .if  ALGORITHM == furlms
PAi    .sect "anc_pma"     ;Ai(n) in PROGRAM space
       .space 16*NAz
PBi    .sect "anc_pma"     ;Bi(n) in PROGRAM space
       .space 16*NBz
   .else
PWi    .sect "anc_pma"     ;Wi(n) in PROGRAM space
       .space 16*NWz
   .endif
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;  UNINITIALIZED MEMORY (DATA)
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
; BLOCK B0
; --------
;
; These coefficients are used by the ANC algorithm.  When the filters are
;  computed, the coefficients should be in PROGRAM space to use with MACD.
;  The total number of coefficients must all fit in RAM block B0.  This
;  number varies by device (C25 vs. C26).  Due to the DMOV embedded in the
;  MACD, the coefficient arrays must be stored in reverse order.
;
;           +----------------+
;           |     c   (n)    |
;           |      N-1       |
;           +----------------+
;           |     c   (n)    |
;           |      N-2       |
;           +----------------+
;           |                |
;           |        .       |
;           |        .       |
;           |        .       |
;           |                |
;           +----------------+
;           |     c (n)      |
;           |      0         |
;           +----------------+
;
;  The arrays are defined as follows:
;    Ci = error path coefficients
;    Di = acoustic feedback path coefficients
;    Wi = adaptive FIR filter coefficients
;    Ai = adaptive IIR filter feedforward coefficients
;    Bi = adaptive IIR filter feedback coefficients
;
;
Ci     .usect "anc_coef", NCz  ;Ci(n) in DATA space
   .if  ALGORITHM == fbfxlms
Di     .usect "anc_coef", NDz  ;Di(n) in DATA space
   .endif
   .if  ALGORITHM == furlms
Ai     .usect "anc_coef", NAz  ;Ai(n) in DATA space
Bi     .usect "anc_coef", NBz  ;Bi(n) in DATA space
   .else
```

```
Wi     .usect ”anc_coef”, NWz  ;Wi(n) in DATA space
    .endif
;-----------------------------------------------------------------
; BLOCK B1
; --------
;
; These arrays hold the inputs and outputs used by the adaptation and filter
;  routines.  Keep these arrays in DATA space.
    .if  ALGORITHM == fxlms
wnn    .usect ”anc_vars”, Ts+1    ;wn(n) in DATA space
xn     .usect ”anc_vars”, NWz+1   ;x(n) in DATA space (+1 for DMOV)
yn     .usect ”anc_vars”, Te+1    ;y(n) in DATA space
x1n    .usect ”anc_vars”, NCz+1   ;x1(n) in DATA space (+1 for DMOV)
    .elseif  ALGORITHM == fbfxlms
wnn    .usect ”anc_vars”, NDz+1   ;wn(n) in DATA space (+1 for DMOV)
yn     .usect ”anc_vars”, NDz+1   ;y(n) in DATA space (+1 for DMOV)
x1n    .usect ”anc_vars”, NWz+1   ;x1(n) in DATA space (+1 for DMOV)
x2n    .usect ”anc_vars”, NCz+1   ;x2(n) in DATA space (+1 for DMOV)
    .elseif  ALGORITHM == furlms
wnn    .usect ”anc_vars”, Ts+1;wn(n) in DATA space
xn     .usect ”anc_vars”, NAz+1   ;x(n) in DATA space (+1 for DMOV)
x1n    .usect ”anc_vars”, NAz+1   ;x1(n) in DATA space (+1 for DMOV)
yn     .usect ”anc_vars”, NBz+2   ;y(n) in DATA space (+2 for y(n–1) DMOV)
y1n    .usect ”anc_vars”, NBz+1   ;y1(n) in DATA space (+1 for DMOV)
    .elseif  ALGORITHM == fanc
wnn    .usect ”anc_vars”, Ts+1;wn(n) in DATA space
xn     .usect ”anc_vars”, NWz+1   ;x(n) in DATA space (+1 for DMOV)
x1n    .usect ”anc_vars”, NWz+1   ;x1(n) in DATA space (+1 for DMOV)
yn     .usect ”anc_vars”, NCz+1   ;y(n) in DATA space (+1 for DMOV)
    .endif
;-----------------------------------------------------------------
; BLOCK B2
; --------
;
; These are general purpose variables used in the program
    .bss      adaptemp,1    ;Adaptation intermediate value
    .bss      en, 1         ;Error mike input: e(n)
    .bss      zn, 1         ;Error path model output: z(n)
    .bss      e1n, 1        ;Error path model error: e1(n)
    .if  ALGORITHM == fbfxlms
    .bss      xn, 1         ;Input mike input: x(n)
    .bss      fn, 1         ;Acoustic feedback path model output: f(n)
    .bss      e2n, 1        ;Acoustic feedback path model error: e2(n)
    .endif
    .if  GENERATOR == white_noise
    .bss      wng, 2        ;White noise generator storage
                            ; current output + intermediate value
    .elseif GENERATOR == sinewave
    .bss      a1, 1         ;Coefficient A1/2
    .bss      s1n_1,1       ;s1(n–1)
    .bss      s1n_2,1       ;s1(n–2)
    .bss      a2, 1         ;Coefficient A2/2
    .bss      s2n_1,1       ;s2(n–1)
    .bss      s2n_2,1       ;s2(n–2)
    .endif
;-----------------------------------------------------------------
; BLOCK B3
; --------
;
;-----------------------------------------------------------------
; EXTERNAL RAM
```

```
; ------------
;
; These arrays hold simulation trace data.  The error mike reading
;  should decay to a very small value if the algorithm works.  Use
;  broadband input to verify algorithm.
;
; NOTE: TOTAL TRACE BUFFER LENGTH USING EVM IS 30K SAMPLES USING
;       SUPPLIED SYSTEM CONFIGURATION.

   .if  TRACE != none
ttrace .usect "verify", NSAMPLES     ;Simulated error mike reading (Train)
ctrace .usect "verify", SIMLENGTH    ;Simulated error mike reading (Cancel)
       .if  ALGORITHM == fbfxlms
fbtrace   .usect "verify", NSAMPLES  ;Simulated feedback reading at input
       .endif
       .if  TRACE == in_out
gtrace .usect "verify", SIMLENGTH    ;Simulated input mike reading (Cancel)
       .endif
   .endif
```

Simulation Models and Waveform Generators File (models.asm)

```
   .title "Simulation Models and Waveform Generators"

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;            Simulation Models and Waveform Generators
;              ----------------------------------------
;
;  File:     MODEL.ASM            Rev: 1.0
;  Last Change:  11/11/93        Start Date:  8/23/93
;
;  Processor:   TMS320C25
;  Language:    Assembly
;  Assembler Rev:   6.40
;
;  Programmer:  Thomas G. Horner
;            TI – Dallas RTC
;            (214) 917–5051
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
; Contains models of acoustic channel and waveform generators required
;  to simulate complete ANC system for algorithm verification.  The
;  random number generator is also used for error path coefficient
;  modelling in the Training module.
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;  INCLUDE FILES
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
   .include  config.asm

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;  PROGRAM
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

   .text

   .if SIMULATION == yes
; Error path model  (speaker to error mike)
;  e(n)=Ge*y(n–Te)
Error
   spm    2            ;Setup for Q27 result in Preg
   ldpk      yn
   lt     yn+Te        ;T=y(n–Te)
   mpyk      Ge        ;P=Ge*y(n–Te) NOTE: Coefficient is 13 bits
   ldpk      en        ;Set DP to error mike input
```

```
    sph    en               ;Save to error mike location
    spm    1                ;Return to standard Q30 result
       .if  ALGORITHM != fbfxlms
    larp      AR4           ;Time shift adaptive filter outputs
    lrlk      AR4, yn+Te–1  ; except the oldest entry, yn(n–Te)
    lark      AR3, Te–1
Err1
    dmov      *–, AR3
    banz      Err1, AR4
       .endif
    ret

       .if  ALGORITHM == fbfxlms
; Acoustic feedback path model for Training mode (speaker to input mike)
;  x(n)=Gf*y(n–Tf)
Feedback
    spm    2                ;Setup for Q27 result in Preg
    ldpk      yn
    lt     yn+Tf            ;T=y(n–Tf)
    mpyk      Gf            ;P=Gf*y(n–Tf) NOTE: Coefficient is 13 bits
    ldpk      xn            ;Set DP to input mike input
    sph    xn               ; and save result
    spm    1                ;Return to standard Q30 result
    larp      AR4           ;Time shift y(n) samples
    lrlk      AR4, yn+Tf–1  ; except the oldest entry, yn(n–Tf)
    lark      AR3, Tf–1
Fb1
    dmov      *–, AR3
    banz      Fb1, AR4
    ret

; Acoustic feedback path model for Cancel mode (speaker to input mike)
;  x(n)=wn(n)+Gf*y(n–Tf)
Feedback1
    spm    2             ;Setup for Q27 result in Preg
    ldpk      yn
    lt     yn+Tf         ;T=y(n–Tf)
    mpyk      Gf         ;P=Gf*y(n–Tf) NOTE: Coefficient is 13 bits
    pac                  ;ACC=Gf*y(n–Tf)
    ldpk      wnn        ;Set DP to white noise array
    addh      wnn        ; and save result
    ldpk      xn         ;Set DP to input mike input
    sach      xn         ; and save result
    spm    1             ;Return to standard Q30 result
    ret
       .endif

; Acoustic channel model (noise source + anti–noise to error mike)
;  e(n)=Gs*wn(n–Ts) + Ge*y(n–Te)
Source
    ldpk      wnn        ;Set DP to input array
       .if  ALGORITHM == fanc
    lt     wnn           ;T=wn(n–Ts), where Ts=0 for fanc algorithm
       .else
    lt     wnn+Ts        ;T=wn(n–Ts)
       .endif
    mpyk      Gs         ;P=Gs*wn(n–Ts) NOTE: Coefficient is 13 bits
    pac                  ;ACC=Gs*wn(n)
    ldpk      yn         ;Set DP to adaptive filter output
    lt     yn+Te         ;T=y(n–Te)
    mpyk      Ge         ;P=Te*y(n–Te) NOTE: Coefficient is 13 bits
    apac                 ;ACC=Gs*wn(n–Ts)+Ge*y(n–Te)
    ldpk      en
    sach      en,3       ;Save to error mike location w/ Q28 shift
       .if  ALGORITHM != fanc
```

```
    larp      AR4           ;Time shift wn(n) array
    lrlk      AR4, wnn+Ts-1 ; except the oldest entry, wn(n-Ts)
    lark      AR3, Ts-1
Src1
    dmov      *-, AR3
    banz      Src1, AR4
       .endif
       .if ALGORITHM == fxlms
    larp      AR4           ;Time shift y(n) array
    lrlk      AR4, yn+Te-1  ; except the oldest entry, y(n-Te)
    lark      AR3, Te-1
Src2
    dmov      *-, AR3
    banz      Src2, AR4
       .endif
    ret
    .endif

    .if  GENERATOR == white_noise
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;  White Noise Generator Routine
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
; This code computes a stream of random numbers.  The algorithm was
;  is taken from the book Digital Signal Processing Design by Bateman
;  and Yates.
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Noisegen
    ldpk      wng
    lac    wng              ;Load current random number
    andk      MASK          ;Extract bits 0, 2, 11, 15
    sacl      wng+1         ;Save for future use
    add    wng+1, 4         ;Combine bit 11 with 15
    add    wng+1, 13        ;Combine bit 2 with result
    add    wng+1, 15        ;Combine bit 0 with result
    andk      MASK          ;Extract bit 15 (others extraneous)
    addh      wng           ;Combine bit 15 w/ previous number
    sach      wng, 1        ;Store new random number
    ret

    .elseif  GENERATOR == sinewave
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;  Sinewave Generator Routine
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
; Sinewave generator to be used for checkout
;  s(n)=(2*cos(2*pi*fd/fs))*s(n-1)-s(n-2)

; Initialize coefficient and ICs for sinewave generator
Sineinit
    ldpk      a1
    lalk      A1d2      ;A1/2 = cos(2*pi*fd/fs)
    sacl      a1
    lalk      S10       ;s1[n-2] IC
    sacl      s1n_2
    lalk      S11       ;s1[n-1] IC
    sacl      s1n_1

    lalk      A2d2          ;A2/2 = cos(2*pi*fd/fs)
    sacl      a2
    lalk      S20       ;s2[n-2] IC
    sacl      s2n_2
    lalk      S21       ;s2[n-1] IC
    sacl      s2n_1
    ret
```

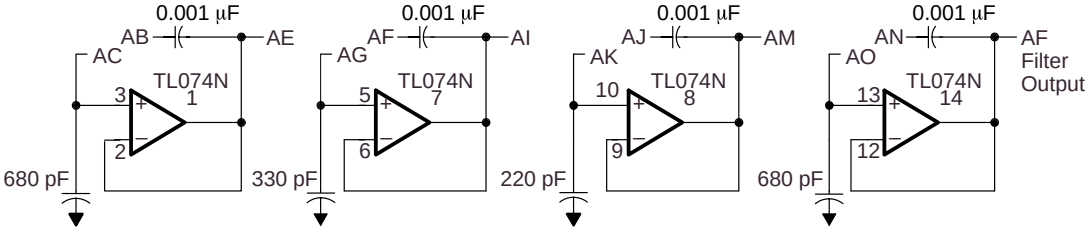
```
;Sinewave generator algorithm.
Sinewave
   ldpk      a1        ;Set DP to coefficient a

   zac                 ;Zero math channel
   mpyk      0         ;
   subh      s1n_2     ; y1[n] = 2*(A1/2*y1[n–1]) – y1[n–2]
   ltd    s1n_1
   mpy    a1
   apac
   apac
   sach      s1n_1     ;Store result back to memory

   zac                 ;Zero math channel
   mpyk      0         ;
   subh      s2n_2     ; y2[n] = 2*(A2/2*y2[n–1]) – y2[n–2]
   ltd    s2n_1
   mpy    a2
   apac
   apac
   sach      s2n_1     ;Store result back to memory

   lac    s1n_1,15     ;ACC = y1[n] + y2[n] w/ gain adjust
   add    s2n_1,15

   ret
   .endif
```

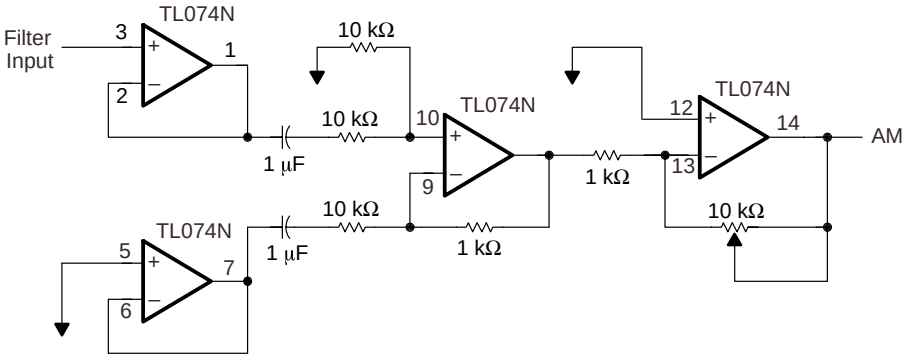
Interrupt Vectors and Interrupt Service Routine Traps File (vectors.asm)

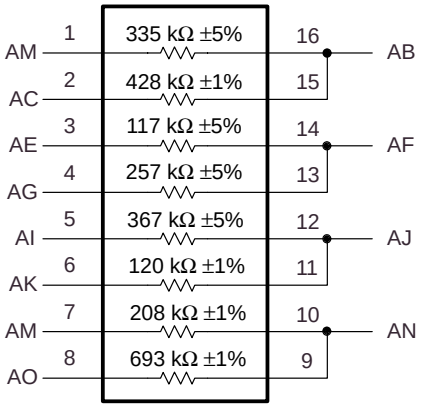
```
   .title "Interrupt Vectors and ISR Traps"

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;         Interrupt Vector Definitions
;         ----------------------------
;
;   File:     VECTORS.ASM           Rev: 1.0
;   Last Change:  10/11/92         Start Date:  10/5/92
;
;   Processor:    TMS320C25
;   Language:     Assembly
;   Assembler Rev:   6.40
;
;   Programmer:   Thomas G. Horner
;             TI – Dallas RTC
;             (214) 917–5051
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
; Interrupt Vectors and Unused Interrupt Traps
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;   INCLUDE FILES
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
   .include  config.asm

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;   VECTORS
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
; Define interrupt vectors with addresses of ISRs.  Any ISR which is not
; active is trapped to an idle state for debug.
;
   .sect  "x_vecs"

   b  Reset     ;Power Up Reset
   b  Int0         ;External Interrupt #0
```

```
   b   Int1          ;External Interrupt #1
   b   Int2          ;External Interrupt #2

   .sect  "p_vecs"
   b   Tint          ;Timer Interrupt
   b   Rint          ;Serial Port Receive Interrupt
   b   Xint          ;Serial Port Transmit Interrupt
   b   Trap          ;S/W Trap

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;  UNUSED INTERRUPT TRAPS
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
; These traps can be used for debug purposes or for an extra measure of
; security in production S/W.  Alternatively, this code can be altered
; to perform a soft reset by replacing the "idle" nmemonic with "b  reset".
;
   .sect  "traps"
   .if  TIMEBASE == external
;Int0  idle          ;External Interrupt #0
   .else
Int0   idle          ;External Interrupt #0
   .endif
Int1   idle          ;External Interrupt #1
Int2   idle          ;External Interrupt #2
   .if  TIMEBASE == timer
;Tint  idle          ;Timer Interrupt
   .else
Tint   idle          ;Timer Interrupt
   .endif
Rint   idle          ;Serial Port Receive Interrupt
Xint   idle          ;Serial Port Transmit Interrupt
Trap   idle          ;S/W Trap

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
   .end
```

APPENDIX E: SCHEMATIC DIAGRAM OF 8-ORDER BUTTERWORTH LOW-PASS FILTER







APPENDIX F: ANC UNIT SYSTEM SETUP AND OPERATION PROCEDURE

This is a detailed description of the system setup and the operation procedure for the ANC unit shown in Figure 31, page 60.

Hardware

The test system was built using PVC pipes of inner diameter 6 inches. The overall length of the duct was 110.7 cm. The distance from noise loudspeaker to error microphone was 69.0 cm. In addition to the PVC pipes, the system included the components and auxiliary equipment in the following list:

- 2 Realistic 33-1063 tie-pin microphones
- 2 Kicker 6.5” free-air subwoofers
- 1 Symetrix SX202 dual microphone preamplifier
- 1 Carvin FET 450 power amplifier
- 4 8-order Butterworth low-pass filters
- 1 LAMBDA LPD-422A-FM dual regulated power supply
- 1 Ariel DSP-16 Plus TMS320C25 DSP board with a ’C25 device
- 1 personal computer
- 1 HP 3561A dynamic signal analyzer or TMS320C26 DSP starter kit with another personal computer
- Function generator(s)

With two input and output ports, the Ariel DSP-16 Plus TMS320C25 DSP board can fulfill the experimental requirements. One input port receives the correction signal from the error microphone, and the other one receives the reference signal from the input microphone. Meanwhile, one of the output ports is used to send out the antinoise signal, and the other is used to send out the signal to observe.

Software

The Ariel DSP board has a set of supported software; however, only part of it is necessary in these experiments. A listing of required files with brief descriptions of their functions follows:

DSPBUG.EXE	The basic DSPBUG executable file. When invoked, this program provides a resident monitor that can be used to load and run the source hex file.
DSPBUG.CFG	The configuration file of the DSPBUG resident monitor
DSPBUG.HLP	Contains the DSPBUG resident monitor help system
DSP320.CFG	The configuration file of the DSP-16 Plus TMS320C25 DSP board. Contains ASCII information on installed options, board revision, I/O address setting, memory address, etc.
RESMON.HEX	Hex file of the DSPBUG resident monitor. Contains all the code that DSPBUG needs to control the operation of the source program when debugging.
ASM320.EXE	Two-pass assembler that reads a source file from disk and writes a standard INTEL format hex object file to disk.

The algorithms involved in the ANC experiments are FXLMS, FBFXLMS, and FURLMS. The filenames of the related assembly code and object files in the software package are as follows:

ALGORITHM	ASSEMBLY CODE FILE	OBJECT FILE FILE
FXLMS	FXLMS.ASM	FXLMS.HEX
FBFXLMS	FBFXLMS.ASM	FBFXLMS.HEX
FURLMS	FURLMS.ASM	FURLMS.HEX

Operation Procedure

The assembly programs and the files required for using the Ariel DSP-16 Plus TMS320C25 DSP board are in the same directory, and the one-dimensional ANC duct system is set up. The operational procedure of this ANC system is as follows:

1. Edit the source program: Type *edit <filename>.asm* at the DOS prompt, where *<filename>* is the name of the source file. For each source file, there must be matching *.asm* file.
2. Assemble the source program: Type *asm320/c25 <filename>* at the DOS prompt. Upon successful assembly, one additional hex file (*<filename>.hex*) is generated.
3. Enter the DSPBUG resident monitor: Type *dspbug* and press enter at the DOS prompt.
4. Set the sampling rate: Type *splrate* and press enter on the command line of the DSPBUG resident monitor, then set the sampling rate of both the input and output ports to 2000 Hz.
5. Run the program:
   a. Turn on the error microphone.
   b. Set the volume of the canceling loudspeaker.
   c. Type *dwnld <filename>* and press enter on the command line of DSPBUG resident monitor.
   d. After the off-line modeling stage, set the volume of the noise loudspeaker.
   e. Turn on the input microphone.
6. Turn off both of the microphones and adjust the volume of the two loudspeakers to zero when the demonstration is complete.
7. Return to DOS: Type *quit* and press enter on the command line of DSPBUG.

Note that the procedure above is valid only for the FXLMS and FURLMS algorithms. For the FBFXLMS algorithm, part d of step 5 should be performed at the same time as part a.

There is a memory conflict between this program and Ariel's DSPBUG resident monitor program (resmon.hex). To avoid the conflict, rearrange the memory locations in this program from on-chip memory to external memory. Extra time may be required to access data from the external memory instead of from on-chip memory. When many calculations are required during continued testing of the system performance in the higher sampling rates, the use of on-chip memory is suggested. In using on-chip memory, the memory conflict occurs when this program is loaded. Error messages are shown on the screen, indicating the crash of the DSPBUG resident monitor. The error messages do not adversely affect the program run, but if the error messages need to be removed from the screen, just press the space bar or the enter key several times.

To make sure that the compensation filters (C(z) in the FXLMS and FURLMS algorithms, C(z) and D(z) in the FBFXLMS algorithm) are well estimated, it is necessary to monitor the modeling error during the off-line modeling stage. Connect the modeling error signal out from output channel B on the Ariel DSP board. Monitor that signal in real time by using a scope. If the observed signal converges, this modeling process is successful. Otherwise, changing the adaptation step size for the estimated filter is required.

APPENDIX G: TMS320C26 DSP STARTER KIT, AN ALTERNATIVE TO THE SPECTRUM ANALYZER

Instead of using a spectrum analyzer to observe the signal, the TMS320C26 DSP starter kit (DSK) provides an alternative approach. The following checklist details items that are necessary to implement this kit [48].

1. Hardware checklist
   a. Host: IBM PC/AT or 100%-compatible PC with a hard disk system and a 1.44-megabyte floppy-disk drive
   b. Memory: minimum of 640K bytes
   c. Display: EGA/VGA
   d. Power supply: 9 V ac at 250 mA with a 2.1-mm power jack connector
   e. Board: DSK circuit board
   f. Port: asynchronous RS-232 serial communications link
   g. Cable: RS-232 with a DB9 interface

2. Software checklist
   a. Operating System : MS™-DOS or PC™-DOS (version 4.01 or later)
   b. Files: DSK_SA26.EXE
      DSK_COM2.DSK
      AV26.ASM
      AV26.DSK
      EGAVGA.BGI
      DSKA.EXE

The DSK_SA26.EXE is an executable real-time FFT file. This file outputs a log magnitude of the FFT result every 256 sampling points. The vertical axis of the spectrum analysis shown on the screen of the PC is dB-based, and the horizontal axis is based on the normalized frequency. The minimum sampling frequency that can be selected on the DSK is 2560 Hz.

After connecting the DSK to the PC and power, plug in the observed signal source. At the DOS prompt, type *dsk_sa26* and press enter. A working spectrum analysis is shown on the screen of the PC.

MS is a registered trademark of Microsoft Corporation.
PC is a trademark of International Business Machines Corp.