INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor MI 48106-1346 USA
313/761-4700 800/521-0600

**An Investigation of Active Control of Turbo-Prop Cabin Noise and
Vibration Using Piezoelectric Actuators**

by

Weiping Xu, B.Eng., M. Eng.

Northwestern Polytechnic University

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

Department of Mechanical and Aerospace Engineering

Ottawa-Carleton Institute

for Mechanical and Aerospace Engineering

Carleton University

Ottawa, Ontario

02, September 1998

©Copyright

1998, Weiping Xu

National Library
of Canada

Acquisitions and
Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale
du Canada

Acquisitions et
services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-36902-1

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

**An Investigation of Active Control of Turbo-Prop Cabin Noise and
Vibration Using Piezoelectric Actuators**

submitted by

Weiping Xu, B.Eng., M. Eng.
Northwestern Polytechnic University

in partial fulfillment of
the requirements for the degree of
Master of Engineering

Thesis Supervisor

Chairman,

Department of Mechanical and Aerospace Engineering

Carleton University

21, September 1998

Abstract

This study was concerned with an experimental investigation of active control of cabin noise and fuselage vibration in a commercial turbo-prop aircraft using piezoelectric actuators. A full-size deHavilland Dash-8 series 100 fuselage was used.

The flight data representing the actual propeller acoustic field and pressure footprints on the fuselage was simulated in the laboratory using a sound source consisting of four loud-speakers and a PC-based signal generator.

Using the simulated sound field, the operating deflection shapes (ODSs) of the fuselage for various sound fields of interest were measured. Based on the analysis of these ODS results, appropriate actuation patterns were suggested that were also verified experimentally as to their effectiveness.

Finally, a multi-input-multi-output (MIMO) feedforward control strategy was developed and employed on the fuselage. The corresponding computer code, control configuration and implementation were initially tested and optimized using a simple test-beam.

Significant vibration and noise reductions were observed. Moreover, different control schemes were tested to investigate the effects of various factors and to determine an effective approach to achieving satisfactory control results.

Acknowledgements

I would like to express my appreciation to my thesis supervisor Prof. F. F. Afagh for his support, encouragement and guidance during the course of this work. I would also like to thank Dr. A. Grewal from the Aeroacoustics and Structural Dynamic Group at Structures, Materials and Propulsion Laboratory (SMPL), Institute for Aerospace Research (IAR) of National Research Council (NRC) for his invaluable cooperation and help during the course of this investigation. Moreover, the generous cooperation and assistance of Dr. L. Pavel and the technical staff from the Aeroacoustics and Structural Dynamic Group in conducting the experimental work at the SMPL/IAR/NRC laboratory is gratefully appreciated.

Finally, I would like to acknowledge the financial support provided by SMPL/IAR/NRC during the course of this investigation.

Table of Contents

Abstract	i
Acknowledgements	ii
Table of Contents	iii
List of Figures	vi
List of Tables	viii
List of Abbreviations	ix
List of Principle Symbols	x
Units	xi
Chapter One: Introduction	1
1.1 Background	1
1.2 Fuselage Vibration and Noise Transmission	2
1.3 Passive vs. Active Noise Control	3
1.4 Active Structural Acoustic Control (ASAC)	5
1.5 Feedforward Algorithm and Adaptive Processing	6
Chapter Two: Sound Field Characteristics and Simulation	8
2.1 Introduction	8
2.2 Installation and Settings of DAC AT-AO-10 Analog Output Board	11
2.3 Program Development	12
2.4 Synthesis and Measurement of Sound Pressure	14
2.5 Simulation of the Desired Sound Field	16
Chapter Three: ODS Measurements and Actuators Array Design	21
3.1 Background	21
3.2 Instrumentation and Data Acquisition	22

3.3	ODS Measurements and Analysis	25
3.4	Piezo Actuator Properties	34
3.5	Optimization of Actuation Pattern	36
3.6	Verification of Actuators Array	38
Chapter Four:	Control Strategy	47
4.1	Adaptive Filters and LMS Algorithm	47
4.2	Recursive LMS Algorithm	50
4.3	SISO Filtered-x LMS Algorithm	52
4.4	MIMO LMS Algorithm	56
4.5	MIMO Filtered-x LMS Algorithm	58
4.6	Off-line and On-line Identifications	60
Chapter Five:	Control Programming and Preliminary Implementation	62
5.1	The DSP System	62
5.2	Mixed C/Assembly Real-time Programming	63
5.3	Program Debugging and Efficiency Analysis	67
5.4	C/Assembly Source Files	67
5.5	Preliminary Implementation of SISO Control	70
5.6	Preliminary Implementation of MIMO Control	71
Chapter Six:	Implementation of Real-Time Vibration and Noise Control on the Fuselage	73
6.1	Control Configuration	73
6.2	System Identification of Piezo-actuated Fuselage	75
6.3	Vibration and Noise Reduction	79
Chapter Seven:	Conclusions and Recommendations	83
7.1	Conclusions	83
7.2	Recommendations	84
References		86
Appendix A	Accelerometer Positions for ODS Measurements	89

Appendix B	Measured FRFs for ODSs	90
Appendix C	Accelerometer Positions to Measure Deflection Patterns of the Piezo-actuated Fuselage	92
Appendix D	Results of Fuselage Vibration and Noise Control	93
Appendix E	User Manual and Program Source Code for Sound Field Simulation Program	135
Appendix F	Source Code and Memory Configuration for SISO Feedforward Control	157
Appendix G	Source Code and Memory Configuration for MIMO Feedforward Control	167

List of Figures

Fig. 1.1	Components of a typical active noise control system for an enclosure	6
Fig. 2.1	Port side propeller pressure distribution in dB	8
Fig. 2.2	Exterior SPL contours, BPF, 910 rpm	9
Fig. 2.3	Flow chart for program WDAQ.CPP	13
Fig. 2.4	Schematic of synthesis and measurement of the simulated sound field	16
Fig. 2.5	Achieved vs. desired SPL and phase of the simulated propeller sound field at 61Hz	18
Fig. 2.6	Achieved vs. desired SPL and phase of the simulated propeller sound field at 70Hz	19
Fig. 2.7	Achieved vs. desired SPL and phase of the simulated propeller sound field at 121Hz	19
Fig. 2.8	Achieved vs. desired SPL and phase of the simulated propeller sound field at 140Hz	20
Fig. 3.1	Schematic of accelerometer locations at Stations C and E for ODS measurements	23
Fig. 3.2	Schematic of the experimental set-up for ODS measurements	24
Fig. 3.3	Acceleration spectrum (in mm/s^2) at station E05 under 61 Hz sound field	27
Fig. 3.4	Acceleration spectrum (in mm/s^2) at station E05 under random sound field	27
Fig. 3.5	ODS at Station C under 61Hz sound field	30
Fig. 3.6	ODS at Station C under 121Hz sound field	31
Fig. 3.7	ODS at Station C under 70Hz sound field	31
Fig. 3.8	ODS at Station C under 140Hz sound field	32
Fig. 3.9	ODS at Station E under 61Hz sound field	32
Fig. 3.10	ODS at Station E under 121Hz sound field	33
Fig. 3.11	ODS at Station E under 70Hz sound field	33
Fig. 3.12	ODS at Station E under 140Hz sound field	34
Fig. 3.13	Actuation induced by piezo-elements	37
Fig. 3.14	Schematic of accelerometer locations for actuation pattern measurements	41
Fig. 3.15	Response to piezo actuation pattern #5 at 61Hz	43
Fig. 3.16	Response to piezo actuation pattern #6 at 61Hz	44
Fig. 3.17	Response to piezo actuation pattern #4 at 121Hz	44
Fig. 3.18	Response to piezo actuation pattern #8 at 121Hz	45

Fig. 3.19 Response to piezo actuation pattern #9 at 121Hz	45
Fig. 3.20 Response to piezo actuation pattern #4 at 140Hz	46
Fig. 3.21 Response to piezo actuation pattern #8 at 140Hz	46
Fig. 4.1 Block diagram of a simple adaptive controller	47
Fig. 4.2 Block diagram of a recursive controller	50
Fig. 4.3 Block diagram of a controller with secondary path	53
Fig. 4.4 Block diagram of a filtered-x LMS controller	55
Fig. 5.1 TMS320C3x/C4x software development flow	64
Fig. 5.2 Program flow chart for system identification	69
Fig. 5.3 Program flow chart for control operation	70
Fig. 5.4 The test beam, actuator and accelerometer for SISO vibration control	71
Fig. 5.5 The test beam, actuators and accelerometers for MIMO vibration control	72
Fig. 6.1 Schematic of system identification and control setup	74
Fig. 6.2 Measured and identified transfer functions between input #4 and output #4	78
Fig. 6.3 Open-loop and closed-loop acceleration spectra of SPL at aisle seats of second row	81

List of Tables

Table 2.1 Desired SPLs and phases at eight grid centers of the speakers	10
Table 3.1 Material characteristics of BM400 piezo-ceramics	35
Table 3.2 Verified actuation patterns	39
Table 6.1 Identification cases	77

List of Abbreviations

ADC	Analog to Digital Converter
ALU	Algorithm Logic Unit
ANC	Active Noise Control
ANSI	American National Standard Institute
ASAC	Active Structural Acoustic Control
BPF	Blade Passage Frequency
DAC	Digital to Analog Converter
DMA	Direct Memory Access
DSP	Digital Signal Processing
FIFO	First-In First-Out
FIR	Finite Impulse Response
FRF	Frequency Response Function
GUI	Graphic User Interface
IAR	Institute for Aerospace Research
IIR	Infinite Impulse Response
LMS	Least Mean Square
MIMO	Multi-Input and Multi-Output
NRC	National Research Council
ODS	Operating Deflection Shape
RLMS	Recursive Least Mean Square
SISO	Single-Input and Single-Output
SMPL	Structures, Materials and Propulsion Laboratory
SPL	Sound Pressure Level

List of Principle Symbols

Symbol	Description
P	Vector of sound pressures
V	Vector of voltage output
F	Transfer matrix for sound field simulation
$r_{xy}(f)$	Coherence function
$G_{xy}(f)$	Cross power spectrum density function
$G_x(f)$	Input auto power spectrum density function
$G_y(f)$	Output auto power spectrum density function
$H(f)$	Frequency response function
d_{31}	Piezo charge constant
S_{11}^E	Piezo elastic compliance
ρ	Piezo mass density
C	Direct filter for plant
D	Recursive filter for plant
$\hat{C}$	Direct filter for plant model
$\hat{D}$	Recursive filter for plant model
A	Direct filter for controller
B	Recursive filter for controller
G_p	Primary path
G_s	Secondary path
a	Direct coefficient of control filter
b	Recursive coefficient of control filter
c	Direct coefficient of identification filter
d	Recursive coefficient of identification filter
μ	Convergence coefficient
$x(n)$	Reference sequence
$u(n)$	Output sequence of controller
$y(n)$	Output sequence of plant
$e(n)$	Error sequence
$J(n)$	Objective function sequence

$d(n)$	Disturbance sequence
$v(n)$	Training signal sequence
$r(n)$	Direct filtered sequence
$s(n)$	Recursive filtered sequence

Units

In keeping with the practices of the North American aerospace industry and the Institute for Aerospace Research (NRC), the Imperial system of units is employed in this thesis. SI system of units are used within the text, and in the figures where practical. The following conversion factors are useful:

$$1 \text{ inch} = 25.4 \text{ mm}$$

$$1 \text{ foot} = 0.3048 \text{ m}$$

$$1 \text{ ksi} = 6.8948 \times 10^{15} \mu\text{Pa}$$

Chapter One

Introduction

1.1 Background

The cabin noise within commercial turbo-prop aircraft is a major source of discomfort to the passengers. During the recent years, the Aeroacoustics and Structural Dynamics Group of the Structures, Materials and Propulsion Laboratory (SMPL) at Institute for Aerospace Research (IAR) of the National Research Council (NRC) of Canada has been investigating the possibility of controlling noise and vibration in the deHavilland Dash-8 Series 100 turbo-prop aircraft using different control techniques. This research work at SMPL/IAR/NRC has been carried out mainly under contract from deHavilland of Canada.

Launched in 1980, the deHavilland Dash-8 is the most successful regional airliner in the world, with over 400 in service. The first of the four member family of the Series 100, 200, 300 and 400, the Series 100 combines advanced technology to accommodate for all the superior design features in order to provide the ruggedness, reliability and performance that is required from a short-haul, high-frequency regional airlines. Their spacious interior, outstanding performance and reliability make them an outstanding passenger aircraft as well as an ideal multi-mission platform.

The main focus of the research work at SMPL/IAR/NRC has been to reduce the internal cabin noise by controlling the fuselage vibration and consequently attenuating noise transmission from the outside sound field to the cabin interior. This so-called active

structural acoustic control (ASAC) has shown a good potential to be applied to commercial aircraft. In fact, successful vibration and noise control has been reported by employing a feedback control strategy during the initial phases of this research at SMPL/IAR/NRC [11,12,13]. Both computer simulation and experimental implementation were used in these earlier studies. The simulation was carried out by modelling the vibration-acoustic coupling models while experimental work was implemented on a full-scale fuselage in the laboratory.

However, the feedback control strategy appears to be suitable in controlling a narrow-band disturbance with a fixed frequency [23]. Whereas, during real flight conditions, the engine speed can vary throughout a considerable range of magnitudes from take-off to cruising state, thus resulting in a corresponding variation in the frequency of disturbance. Therefore, from a practical and operational point of view, due to its capability of adapting to a varying narrow-band or even broad-band disturbance, a feedforward control strategy should be more suitable for this application than the conventional feedback control approach [6,7]. Hence, the main objective of the work at hand was to expand further the previous work done at SMPL/IAR/NRC by using a feedforward control strategy in an ASAC approach to control the fuselage noise and vibration. The feasibility of this approach was tested experimentally on a full-size deHavilland Dash-8 Series 100 fuselage.

1.2 Fuselage Vibration and Noise Transmission

Cabin noise in turboprop aircraft is generated by various sources, including

boundary layer flow noise, acoustic excitation of the fuselage from the propeller, and structure-borne noise due to engine vibration. For typical turbo-prop aircraft, the proximity of the propeller disc to the fuselage results in the narrow-band acoustic excitation of the fuselage being the dominant source of noise [14].

The propeller noise transmission from the cabin exterior to its interior occurs due to the coupling of the propeller-induced fuselage vibration and the interior acoustical field. A general, yet detailed theoretical treatment of acoustic coupling between a structure and its enclosed volume of fluid can be found in reference [16]. However, Grewal, et al. [13] have presented a vibration-acoustic modelling and simulation approach specifically for aircraft cabin noise transmission in which the uncoupled equations governing the structural and acoustic subsystems are presented. The coupling terms appear due to integrating the product of the structural and acoustic mode shape functions at the structural-acoustic boundary over the entire area. Various studies have shown that not all the structural and acoustic modes are necessarily strongly coupled, and therefore, interior noise reduction can be attained by controlling only those structural vibration modes which are well coupled with the interior acoustic modes [22].

Thus, using a distributed piezo array, as was done in this investigation, allows to control appropriate structural modes by inducing properly distributed bending moments, and consequently controlling the corresponding acoustic modes more effectively.

1.3 Passive vs. Active Noise Control

A significant amount of work has been carried out by various researchers to reduce

the noise level in the passenger cabin as a source of discomfort to passengers.

Passive control of noise and vibration is a technique that has been adopted extensively in engineering noise and vibration control. The common practice in this approach is to either modify the stiffness of the structure or augment its damping to achieve passive control by avoiding structural resonances or absorbing vibration and/or acoustic energy. However, for the type of problem in this investigation, such passive control techniques are generally not very effective. In fact, increasing the damping does not reduce the noise and vibration levels appreciably since the excitations are usually neither broadband nor resonant [11]. Moreover, using damping materials is generally ineffective at the low frequencies (50 to 300 Hz) where propeller noise is significant. Also, narrow-band damping, using tuned vibration absorbers can be effective in reducing noise, but the improvement is limited due to the difficulty in keeping the devices tuned in a varying environment.

The successful use of active noise control (ANC), where a large number of microphones and speakers are employed, has been reported in various applications with significant noise reductions [6,16,25,26]. This approach, generally referred to as noise cancellation, is realized by generating sound waves using secondary sources that interfere with the noise, thereby using the destructive interference of the component waves to reduce the level of the noise. However, the size of the required devices and the complexity of the configuration make the implementation of this approach inside the cabin of an aircraft unattractive.

1.4 Active Structural Acoustic Control (ASAC)

Rather than using a large number of microphones and loudspeakers to generate a cancelling sound field, active structural acoustic control involves the use of structural actuators in conjunction with a combination of acoustic and structural sensors to address the problem before the noise is transmitted inside the aircraft cabin. In other words, ASAC is achieved by modifying the acoustic transmission properties of the fuselage and by weakening the coupling between the exterior and interior acoustic. A simulation for active control of the transmission of sound through a thin cylindrical shell has been reported by D.R. Thomas and et. al. [22]. In this simulation the fuselage was modelled as a simple cylinder and the control effectiveness for various actuation modes was presented. An experimental implementation of this approach was adapted for the present investigation where a number of related issues are raised and addressed. For example, the operating deflection shapes and piezo actuating patterns are determined to explore the most efficient control configuration.

Comparing to passive control, a typical active noise control system consists of: (1) error sensors such as microphones, strain gauges, and/or accelerometers; (2) control actuators; and (3) a controller; Fig. 1.1.

Using an ASAC approach, the initial research at IAR of NRC referred to earlier, indicated that the approach involving the feedback of local structural vibration is promising, with a significant level of vibration and noise reduction achieved in a limited implementation of this concept [11,12]. In that investigation, collocated actuator/sensor pairs were bonded to the fuselage in the propeller footprint area. A Single-Input and

Single-Output (SISO) control loop was used for each actuator/sensor pair and a classical second order feedback algorithm was implemented in real time. Using a full-scale fuselage, open-loop transfer functions from actuator to strain, actuator to acceleration, and actuator to interior sound level were obtained. Furthermore, the performance of the control system in the presence of an acoustic disturbance was evaluated by comparing a series of open loop transfer functions for (i) disturbance to average strain, (ii) disturbance to average acceleration, and (iii) disturbance to interior sound pressure to their corresponding closed loop transfer functions.

Fig. 1.1 Components of a typical active noise control system for an enclosure (ref. [4])

1.5 Feedforward Algorithm and Adaptive Processing

A feedforward control strategy was used in this investigation. To control a narrow-

band harmonic disturbance, both feedback and feedforward strategies have been shown to be equally effective [17]. However, once a feedback controller has been designed it can only control a disturbance with fixed frequency bands. In contrast, a feedforward controller can keep adapting the frequency change of disturbance through the control process, and therefore it is suitable to control a frequency-varying disturbance.

In feedforward control strategy, active control of a periodic signal is achieved by assuming the availability of a reference signal that is well-correlated with the periodic signal that is to be cancelled. In contrast to the feedback algorithm, the generation of this reference signal is completely independent of any action of the secondary source. Digital adaptive filters are employed as the basic components in system identification and control. Multi-Input and Multi-Output (MIMO) capability of the control system is deemed to be of paramount importance if global vibration and noise attenuation is to be achieved.

The implementation and application of this technique has been greatly influenced by the development of powerful digital signal processors (DSPs) and adaptive signal processing algorithms since late 1980s. Due to the high speed DSPs real-time numerical processing of digitized signals have become possible. Moreover, adaptive algorithms allow for a varying environment and can process several frequencies simultaneously or even handle broadband disturbances.

Chapter Two

Sound Field Characteristics and Simulation

2.1 Introduction

As it was pointed out earlier, for a typical turbo-prop aircraft the narrow-band acoustic excitation of the fuselage is the dominant source of cabin noise. For the deHavilland Dash-8 aircraft, this source of noise occurs at a fundamental Blade Passage Frequency (BPF) varying between 61 Hz and 70 Hz and its higher harmonics. It should be noted that a BPF depends on the engine speed which can vary from 910 rpm at cruising condition to 1050 rpm at take-off. For a four bladed propeller, the BPF's at 910 rpm and 1050 rpm correspond to 61 Hz and 70 Hz respectively. The sound pressure created by the propeller is essentially concentrated in the vicinity of the propeller. Figs. 2.1 and 2.2 show examples of the sound pressure level (SPL) contour on a typical Dash-8 fuselage.

Fig. 2.1. Port side propeller pressure distribution in dB, re: maximum value (ref [11])

Fig. 2.2 Exterior SPL contours, BPF, 910 rpm

To demonstrate the vibration and noise control on a full-scale fuselage, it was necessary to simulate, in the laboratory, the sound field created by the propeller. This simulation was based on the external acoustic pressure field data for various flight conditions that was supplied by the propeller manufacturer, Hamilton Standard. This data, that reflects the characteristics of the power-plant, propeller and fuselage at flight conditions, refers to the SPLs and phases at eight given locations indicated in Fig. 2.2 and is presented in Table 2.1. To provide the desired sound field in the propeller footprint area, important characteristics, such as magnitude and phase of this flight data were simulated using the four-speaker sound source developed at Structures, Materials and Propulsion Laboratory (SMPL) of IAR/NRC. This system was driven by audio amplifiers

that were controlled by a PC-plug-in digital to analog convertor (DAC) output board of type AT-AO-10.

Location	BPF of 61 Hz		BPF of 121 Hz		BPF of 70Hz		BPF of 140 Hz	
	SPL dB	Phase deg	SPL dB	Phase deg	SPL dB	Phase deg	SPL dB	Phase deg
D2	-5.64	49.8	-9.40	173.8	-5.33	66.9	-9.77	-146.1
D1	-3.29	25.7	-5.33	115.1	-3.31	38.7	-6.73	138.0
C2	-1.07	-0.2	-2.13	81.8	-1.02	9.7	-2.45	101.8
C1	-0.83	-23.3	-4.13	30.5	-0.83	-13.8	-4.71	51.5
B2	0	-48.3	-0.10	-20.9	0	-39.0	-0.57	-1.8
B1	-0.28	-58.9	0	-36.5	-0.14	-48.5	0	-15.6
A2	-2.04	-69.8	-2.55	-63.3	-1.69	-55.7	-1.97	-33.3
A1	-3.26	-72.0	-3.95	-73.2	-2.94	-53.8	-3.97	-34.8

Table 2.1 Desired SPLs and phases at eight grid centers of the speakers

A Windows-based program with a graphic-user-interface (GUI) was developed in C++ language to implement the following features: (1) to set desired characteristics, such as frequencies, output voltage levels and phases; (2) to pre-display the output voltage waveforms; (3) and to synthesize and output analog voltages out of the analog output board. The documentation of this code is presented in Appendix E.

An effective simulation procedure was developed by first forming the transfer function matrix between speaker input voltages and the sound pressure distribution and then calculating the required output voltages for a desired pressure distribution. Four transfer function matrices associated with four desired frequencies were obtained. Each of these was developed by measuring the sound pressure distribution due to a single

channel output of unit magnitude. Using these transfer function matrices the analog voltage outputs corresponding to the required sound pressure distribution were calculated by Matlab.

2.2 Installation and Settings of DAC AT-AO-10 Analog Output Board

The AT-AO-10 is a high-performance analog output and digital I/O board for the PC, and is designed for various applications such as: process control, instrumentation, and electronic test signal generation. The major features of this board are as follows:

- (1) 10 double-buffered 12-bit DACs
- (2) Unipolar/bipolar voltage output
- (3) 4-20mA current output
- (4) Onboard 1,024-word FIFO buffer
- (5) Transfer rates up to 200 K Samples/Sec. per channel
- (6) 8 Digital I/O lines
- (7) Full PC I/O Channel DMA

This board was installed into a Dell 486 PC following the steps in the AT-AO-10 User Manual [15]. For applications in this project, only the first 4 DACs were used in FIFO buffer mode. The hardware settings were as follows:

- (1) Base I/O address: Hex1C0 (to 1DF)
- (2) DMA Channel: channel 5
- (3) Interrupt level: Group 1 -- level 11, Group 2 -- level 10

(4) Bipolar voltage output

(5) External reference voltage: on-board 2.5-volt reference voltage output

2.3 Program Development

A C++ code was developed under Borland C++ 4.5 environment (Appendix E). Borland C++ 4.5 is a powerful C/C++ development system with a number of useful features. Both 16-bit Windows application (Windows 3.1) and 32-bit Windows application (Windows 95/NT) could be created with this source code.

The developed Windows-based executive file provides a graphic user interface that can conveniently set the required output analog signal for each of the four channels by pre-displaying the output waveform on the screen, saving or loading the necessary setting parameters to or from the disk files, and programming and controlling the DAC AT-AO-10 board.

The corresponding flow chart is shown in Fig. 2.3.

The code for GUI parameter settings was developed based on two Windows OWL classes: TframeWindows and Tapplication. Two new dialog classes were derived from OWL class TDialog .

The configuration of DAC board was complemented by programming a series of registers. The following steps were used in this process:

- (1) Select external reference voltage;
- (2) Set two's complement bipolar output mode;
- (3) Forbid any interrupts;

Fig. 2.3 Flow chart for the program WDAQ.CPP

- (4) Select the source of counter 1 of MSM82C53 Programmable Timer to be 1MHz;
- (5) Enable FIFO;
- (6) Set DA0, DA1, DA2 and DA3 as group 1;
- (7) Select double-buffer DAC output mode. The output of group 1 is updated by

counter 1 and is scanned from FIFO sequentially;

(8) Set counter 1 of MSM82C53 as mode 2 (Rate Generator) with count value of 100 (therefore the output sample rates will be 10KHz since the clock rate is 1MHz as set in step (4)).

The limitations of the program are as follows:

(1) In order to achieve a continuous waveform by using a buffer of finite length, the output frequency is designed at 1.0 Hz resolution.

(2) To keep the analog waveform sufficiently smooth, the frequency is restricted to a maximum of 999 Hz, thus, allowing at least ten samples per cycle.

(3) Up to ten components of frequency are allowed for each waveform

(4) Maximum output is restricted to connected external reference voltage

(5) This program can only be executed with DAC AT-AO-10 analog output board and under the exact hardware settings described in Section 2.2.

2.4 Synthesis and Measurement of Sound Pressure

The experimental setup for synthesizing and measuring the simulated sound field is shown in Fig. 2.4.

Voltage signals supplied by AT-AO-10 board were sent to speakers after being amplified by ASHLY MFA-8000 audio amplifiers.

Four speakers, each with an opening measuring 30"x24" in surface area, covered a total of approximately 100 degrees around the fuselage at the footprint area. The center

of speaker B was located at the point of maximum sound field pressure, which was located at 305 degrees in the circumferential direction from the top point (see Figs. 2.1 and 2.2). There was a 2" distance between the speakers and the fuselage surface to allow the installation of a microphone and for the blending of sound of each speaker.

A Brüel-Kjær (B&K) 4136 microphone was used to detect sound pressure at desired points. These were located at the center of each of the four grids on a speaker (Fig. 2.4). The B&K Type 4136 is a pressure microphone with a quarter inch diameter [5]. No specific orientation is required for this type of microphone. The maximum allowable sound pressure level for this microphone is 186 dB re 20 μ Pa, which is adequate for this investigation since the maximum simulated sound pressure would not exceed 130 dB. The frequency response is sufficiently flat (less than 0.2 dB correction is required when the frequency is below 4 KHz). Based on the requirement of the microphone, a polarization voltage of 200 V was used for the B&K Type 2610 measuring amplifier that was employed in this investigation.

Signals from the microphone were analyzed using a ZONIC 7000/ZETA data acquisition system. The analysis included checking the waveform, auto-spectra analysis to determine the sound pressure magnitude in the frequency domain, and frequency response analysis with reference to the DA0 output of the AT-AO-10 board.

Fig. 2.4 Schematic of synthesis and measurement of the simulated sound field

2.5 Simulation of the Desired Sound Field

Using the propeller pressure flight data presented in Table 2.1, the sound pressure levels and phases at sixteen locations around the fuselage surface were selected. These locations correspond to the centers of the four grids on each of the four speakers. However, due to symmetry of the simulated sound field with respect to the circumferential center line of the speakers' grid, only half of these locations are considered in the following development. The output voltages and phases of the 4-channel

DAC board were set for each of the following four frequencies: 61Hz (910rpm, BPF), 121Hz (910rpm, 2xBPF), 70Hz(1050rpm, BPF) and 140Hz (1050rpm, 2xBPF).

Since the simulated sound pressure at every spatial point would be the resultant of the magnitude and phase from each individual speaker source, it is necessary to obtain an appropriate transfer function matrix. This matrix represents the relationship between DAC outputs and the resulting sound pressures, i.e.,

$$P_{8 \times 1} = F_{8 \times 4} V_{4 \times 1} \quad (2.1)$$

In equation (2.1) , V is a vector of voltage outputs, P is the vector of sound pressures at 8 locations, and F is the transfer matrix. Each column of F , representing the response from a specific speaker, is obtained by measuring the response when only that speaker is driven by a one Volt output from the DAC board and all other speakers are turned off. Since the voltage outputs from the DAC and the resulting response have a magnitude as well as a phase, the components of F , P and V are all complex numbers.

Once the transfer matrix F is obtained, the required DAC voltage output, V , corresponding to any desired response P could be obtained by the method of Least Squares, i.e.,

$$V_{4 \times 1} = (F^T F)^{-1} F^T P_{8 \times 1} \quad (2.2)$$

Using the required voltage outputs obtained from equation (2.2) in the PC program WDAQ.EXE and driving the speakers through the amplifiers the desired sound fields were created. In driving the amplifiers the gains were set at exactly the same levels

which were used in obtaining the transfer function matrix.

The results of simulations at four different frequencies of 61Hz, 121Hz, 70Hz and 140Hz are presented in Figs. 2.5-2.8. The resulting congruence between the required and the simulated values indicates that the approach taken for sound field simulation is highly feasible and effective.

Fig. 2.5 Achieved (dashed) vs. desired (solid) SPL and phase of the simulated propeller sound field at the BPF, 910 rpm (61 Hz); measured over 8 microphone locations.

Fig. 2.6 Achieved (dashed) vs. desired (solid) SPL and phase of the simulated propeller sound field at the BPF, 1050 rpm (70 Hz); measured over 8 microphone locations.

Fig. 2.7 Achieved (dashed) vs. desired (solid) SPL and phase of the simulated propeller sound field at the BPF, 2x910 rpm (121 Hz); measured over 8 microphone locations.

Fig. 2.8 Achieved (dashed) vs. desired (solid) SPL and phase of the simulated propeller sound field at the BPF, 2×1050 rpm (140 Hz); measured over 8 microphone locations.

Chapter Three

ODS Measurements and Actuators Array Design

3.1 Background

In active structural acoustic control (ASAC), the objective is to control the noise within an enclosure by controlling the associated structural vibration. Therefore, it becomes necessary to determine the operating deflection shape (ODS) of the bounding structure in advance. The ODS refers to the dynamic deflection pattern of a structure under an operating load with a particular magnitude and phase distribution. Specifically, in this investigation, it refers to the dynamic deflection pattern of the fuselage under the exterior propeller pressure field. The experimental simulation of this pressure field in the laboratory setting is discussed in Chapter Two.

The measured ODSs were used to determine the most effective arrangement of actuators. In contrast to vibration mode shapes, ODSs are obtained from the forced vibration of the structure caused by the simulated sound field. In fact, an ODS is composed of a number of mode shapes in a weighted sense. A significant difference between a mode shape and an ODS is that the former is phase independent whereas the latter is not. ODS's are discussed more thoroughly in Section 3.3.

Since two particular BPFs and their first harmonics were the main focus of our control, only the ODSs corresponding to four such particular sound fields were measured. These four sound fields were at 61Hz, 121Hz, 70Hz and 140 Hz frequencies. For details of magnitude distribution, phase distribution and the simulation procedure one should

refer to Chapter Two.

Two stations along the fuselage were selected for ODS measurements. These were called Stations C and E, which are designated, respectively, as 291.4 x and 312.3 x along the fuselage. To investigate the effect of actuators on the stiffness of the structure, the ODS's at Station C were determined without any actuators mounted, whereas at Station E the ODS's were determined with actuators mounted.

3.2 Instrumentation and Data Acquisition

At each of the two Stations C and E, the fuselage above the floor level was instrumented at thirty-three locations. These positions are shown in Fig. 3.1 while exact circumferential locations are given in Appendix A. Fifteen B&K Type 4393 accelerometers, each with a mass of 2.4 grams, were available for measuring radial accelerations at every one of these locations. Compared to the size and the mass of the fuselage, the additional mass of the accelerometers was deemed to be too small to be of any significance. Moreover, the accelerometers had a frequency upper limit of 10 KHz that was more than sufficient for this application. All accelerometers were mounted in the radial direction. Due to the limited number of available accelerometers, charge amplifiers and channels on the data acquisition system, the required data had to be collected in a sequential manner using several set-ups. The signal from the accelerometer at the highest point on the fuselage (location C40 or E40) was used as the reference signal in determining the FRFs and coherence of the signals from all other accelerometers.

Fig.3.1 Schematic of accelerometer locations on Stations C and E for ODS measurements

Fifteen B&K type 2635 charge amplifiers were also employed to amplify the charge signals. Signal components with frequencies higher than 1 KHz were filtered out by the built-in low pass analog filter. The output gain for each amplifier was carefully set to achieve the highest possible signal to noise ratio.

A ZONIC 7000 Data Acquisition System with 16 input channels and 2 output channels was employed to collect the signals from the charge amplifiers as well as from

the audio amplifier. This system was also used to output a random signal to the audio amplifier to drive the speakers. The ZONIC 7000 was controlled by a host HP9000 UNIX workstation. Communication between these two machines was performed through an EtherNet connection. The software package ZETA developed by ZONIC Corporation particularly for this system has many general digital processing features, such as spectrum analysis, coherence analysis and frequency response analysis.

Fig. 3.2 Schematic of the experimental set-up for ODS measurements (ref. [11])

The calibration of all accelerometers was carried out after completing the whole instrumentation set up. In this way, the calibration included the possible effects from all components such as the accelerometers, charge amplifiers, wires and connectors. A B&K

Type 4292 Calibration Exciter was used which provided a standard sinusoidal excitation with 10 ms^{-2} RMS magnitude and 159.2 Hz frequency. A schematic of all the corresponding instrumentation set-up is shown in Fig. 3.2.

Theoretically, an acceleration level can be read directly in the time domain. However, the signal provided by the exciter was not a perfect sinusoidal wave with a constant amplitude. In such a case, the standard practice is to use the statistical value from either the auto-spectrum analysis or the 1/3 octave analysis. The 1/3 octave analysis with no weight is reported to be more accurate due to the absence of the picket-fence effect [3]. Therefore this method was used in this investigation.

The data was acquired, reduced and saved on disk files with the ZONIC 7000/ZETA and the host HP 9000 UNIX workstation. In each measurement, depending on the dominant frequency a different frequency span was used for the analysis. For example, for the 61 Hz signal the frequency span used was 0-156.25 Hz. For a random signal, the frequency span used was 0-625 Hz. In order to obtain a sufficiently high frequency resolution and a low random error a 4096 sample block size with 15 to 20 averages was selected. Moreover, a Hanning window was applied during frequency domain analysis to minimize the leakage effect [3].

3.3 ODS Measurements and Analysis

Considering the objectives of the investigation, the ODSs for the following ten cases were investigated:

- I) At Station C without any mounted actuators and under

- (1) a random sound field
- (2) the 61 Hz sound field
- (3) the 121 Hz sound field
- (4) the 70 Hz sound field
- (5) the 140 Hz sound field

II) At Station E with mounted actuators and under

- (6) a random sound field
- (7) the 61 Hz sound field
- (8) the 121 Hz sound field
- (9) the 70 Hz sound field
- (10) the 140 Hz sound field.

For the random sound cases, a single signal source provided each of the four speakers with the same random signal. Therefore, the exterior surface area of the fuselage that was covered by the four speakers was subjected to a sound field with an almost uniform magnitude and phase.

For each load case, the first task of the analysis was to determine the auto-spectrum. For a sinusoidal load, this shows the dominant frequency and its harmonics (see for example Fig. 3.3). For a random case, the auto-spectrum shows the response over the whole frequency span of interest. (see for example Fig. 3.4).

Fig. 3.3 Acceleration spectrum (in mm/s^2) at station E05 under 61 Hz sound field

Fig. 3.4 Acceleration spectrum (in mm/s^2) at station E05 under random sound field

Next, the coherence for each case was determined. In general, the coherence $r_{xy}(f)$ is a function reflecting the relationship between two signals over a frequency span. It is defined as:

$$r_{xy}(f) = \frac{|G_{xy}(f)|^2}{G_x(f)G_y(f)}, \quad (3.1)$$

where, $G_{xy}(f)$ = cross power spectrum density function,

$G_x(f)$ = input auto power spectrum density function,

$G_y(f)$ = output auto power spectrum density function.

In an ideal situation, where a frequency response function is highly valid and acceptable, the coherence should be close to unity. Otherwise, a poor coherence could be indicative of one or a combination of the following three possibilities: (1) the output is not linear with respect to the input; (2) the input is not the sole cause of the output; (3) the output includes certain noise. In general, the coherence may be improved by raising the signal to noise ratio by various means such as setting a proper amplifier gain, or shutting down other possible sources of noise in the environment such as the air handler in the laboratory. All these measures were taken in this work to improve the coherence throughout the whole frequency span for the random case.

As long as the coherence is close to unity, the frequency response function (FRF) would be acceptable. Fortunately, for the measured sinusoidal cases, the coherence values at the dominant frequencies were all higher than 0.97.

The frequency response $H(f)$, is defined as:

$$H(f) = \frac{G_{xy}(f)}{G_x(f)}, \quad (3.2)$$

where, $G_{xy}(f)$ and $G_x(f)$ are defined in equation (3.1).

The FRF represents the rates and phase differences between a measured signal and the reference signal over the whole frequency span. For a sinusoidal case, only the FRF at the dominant frequency is acceptable, since due to poor coherence over the rest of the frequency range the values of FRF at frequencies other than the dominant frequency are useless. The FRFs corresponding to all eight sinusoidal cases are given in Appendix B. In each case, the corresponding ODS was obtained by using the respective FRF. These ODS's are used to determine the deflections at eight different instants during a single period of the sound wave. In Appendix B, no ODS measurements are reported for the random sound fields. This is because a random load does not represent an operating load with a particular magnitude and phase distribution, and therefore the concept of ODS is not applicable to such a load. In fact, the measurements for random sound fields were carried out only to gain a better insight and a general appreciation of the dynamic response of the fuselage throughout a wide frequency span.

For example, for a certain sinusoidal case, let the magnitude and the phase of the FRF at the fuselage circumferential station i ($i=1,2, \dots, N$) be A_i and ϕ_i respectively. The deflection can then be represented as

$$d_i(t) = A_i \sin(2\pi f t + \phi_i) \quad i=1,2, \dots, N$$

where, f is the frequency. Now, considering eight equidistant time intervals during a single period of sound wave, one has the eight time instants: $t_1=0$, $t_2=\pi/(4f)$, $t_3=\pi/(2f)$, ..., $t_8=7\pi/(4f)$. Therefore, using above expression one can obtain eight groups of deflection

values $[d_1(t_1), d_2(t_1), \dots, d_N(t_1)]$, $[d_1(t_2), d_2(t_2), \dots, d_N(t_2)]$, $\dots$, $[d_1(t_8), d_2(t_8), \dots, d_N(t_8)]$ for all $i=1, 2, \dots, N$ corresponding to the eight equidistant instants. Using this data, the ODSs of the fuselage at the eight equidistant time intervals are obtained as shown in Figs. 3.5-3.12. Since the signal from the accelerometer at C40 (or E40) was used as the reference signal in determining the FRFs, similar to modes, ODSs are dimensionless and an arbitrary scale can be used to present them.

It should be noted that the ODSs at Stations C and E are noticeably different. This variance can be attributed to two factors: (1) due to their different axial locations along the fuselage the effective stiffnesses of the fuselage at the two stations are different; (2) Station C was measured without any piezo actuators mounted on it.

Fig. 3.5 ODS at Station C under 61Hz Sound Field

Fig. 3.6 ODS at Station C under 121Hz Sound Field

Fig. 3.7 ODS at Station C under 70Hz Sound Field

Fig. 3.8 ODS of Frame C under 140Hz Sound Field

Fig. 3.9 ODS of Frame E under 61Hz Sound Field

Fig. 3.10 ODS of Frame E under 121Hz Sound Field

Fig. 3.11 ODS of Frame E under 70Hz Sound Field

Fig. 3.12 ODS at Station E under 140Hz Sound Field

3.4 Piezo Actuator Properties

Lead zirconate titanate (PZT)-based ceramics were used as piezoelectric actuators in this investigation. Under stress-free conditions when an electric field is applied to a piezo-ceramic its ceramics dimensions change in all three directions.

The degree to which these dimensions change relative to an applied field depends on the Strain Constant d_{31} of the piezo material, where,

$$d_{31} = \frac{\text{strain developed along axis 1}}{\text{field applied along axis 3}} = \frac{m/m}{V/m} = \frac{m}{V} \quad (3.3)$$

or,

$$d_{31} = \frac{\text{charge per electrode area}}{\text{stress applied along axis 1}} = \frac{C/m^2}{N/m^2} = \frac{C}{N} \quad (3.4)$$

The piezo actuators employed in this investigation, each 1"x1"x0.25" in size, were Type I d_{31} -mode piezo-ceramic BM400 that are manufactured by Sensor Technology Limited. The material properties of this ceramic are listed in Table 3.1.

Property	Value
Charge constant, d_{31}	$-115 \times 10^{-12} \text{ C/N}$
Elastic compliance, S_{11}^E	$12.5 \times 10^{-12} \text{ m}^2/\text{N}$
Mass density, ρ	7.6 g/cm^3

Table 3.1 Material characteristics of BM400 piezo-ceramics

These segmented actuators were distributed on the fuselage in the circumferential direction at three different stations along the fuselage by bonding them onto the interior surface of the structure. Thus, due to the piezo-electric strain induced by the applied electrical field at each bonded actuator, a distributed bending moment is applied around the fuselage at each station. This kind of one-sided actuation will induce flexural as well as in-plane vibrations. To control noise transmission into an aircraft cabin, flexural vibrations are of greater significance due to their ability to couple with acoustic waves [12].

Numerous studies of interaction between piezo actuators and the structures to which they were bonded have been reported. Studies on piezo actuators bonded onto beams indicate that under ideal bonding conditions the load transfer between the actuator and substructure occurs over a very small region at each of the two ends of the actuator element [19]. Such ideal bonding condition also requires a very thin and stiff layer of piezo actuator. Therefore, if a series of piezo elements are placed at sufficiently close

intervals and are subjected to the same electric field they would behave as a single long actuator with only the outer ends of the first and last elements inducing strain on the substructure. In order to achieve this effect, a small distance of 1/8" was maintained between neighboring piezos [11].

3.5 Optimization of Actuation Pattern

Based on the measured ODSs, various actuation patterns were considered which would be important in achieving an effective control.

For purposes of control, a total of 197 piezo actuators were bonded onto the inside surface of the fuselage at three different stations. At Station A, located at 270.45 x in the axial direction, sixty-eight piezos were distributed between $\theta \approx -108$ degrees and $\theta \approx -12$ degrees (cf. Fig. 3.1). Moreover, sixty piezos were distributed at Station C between $\theta \approx -98$ degrees and $\theta \approx -14$ degrees (cf. Fig. 3.1), and sixty-nine piezos were distributed at Station E between $\theta \approx -108$ degrees and $\theta \approx -12$ degrees (cf. Fig. 3.1). In order to effectively reduce the vibration, almost the whole arc of the fuselage on the port side that was covered by the four speakers was spanned by distributed piezo actuators. The wiring of piezos was configured to allow for the power to individual piezos to be switched on and off or have a reversed phase as desired. Thus, the piezos could be actuated in any desired combination to induce the required bending moments on the fuselage.

The basic control concept employed was to simply cancel or reduce the vibration patterns obtained in the measured ODSs. To this end, appropriate combinations of piezos can be actuated either in or out of phase providing the bending moments required to

effectively reduce the flexure, Fig. 3.13.

Fig. 3.13 Actuation induced by piezo-elements (ref. [11])

Based on the analysis of obtained ODSs, the following actuation patterns were considered.

For ODS at Station E at 61 Hz

- (a) E14/15 to E27
- (b) E14/15 to E27 --positive phase
- E7 to E14/15 --Negative phase
- E27 to E36 --Negative phase
- (c) E16 to E36
- (d) E16 to E36 --positive phase
- E7 to E16 --Negative phase

For ODS at Station E at 121 Hz

- (e) E20/23 to E31/34

(f) E20/23 to E31/34 -- positive phase

E31/34 to E36 --Negative phase

E8/11 to E20/23 --Negative phase

(g) E15/16 to E36

(h) E15/16 to E26 --positive phase

E7 to E26/28 --Negative phase

For ODS at Station E at 140 Hz

(i) E16/17 to E31/34

(j) E16/17 to E31/34 -- positive phase

E8/11 to E16/17 --Negative phase

E31/34 to E36 --Negative phase

(k) E23/26 to E36

(m) E14/15 to E23/26 --positive phase

E23/26 to E36 --Negative phase

(n) E14/15 to E36

Due to considerable similarity of ODSs at 61 Hz and 70 Hz, the same actuation pattern was considered for both of these frequencies.

3.6 Verification of Actuation Patterns

An experimental verification of the actuation patterns that were considered in Section 3.4 was deemed to be necessary before implementing any control schemes on the actual structure. This verification was performed at Station E.

In contrast to ODS measurements, verification had to be accomplished by measuring and analyzing fuselage deflection patterns due to actuation by piezos rather than excitation by the sound fields.

Using the actuation patterns suggested in Section 3.5, appropriate combinations of piezos were to be activated to induce the required bending moments. However, due to electrical grounding problems encountered during the experimentation it was not possible to verify all of the previously suggested actuation patterns. The adhesive that was used to bond the piezos to the substructure proved to have insufficient electrical insulation property. As the result, the actuation patterns that included phase reversing amongst groups of actuators had to be abandoned, and the experimental verification was carried out for a slightly revised set of ten actuation patterns in which phase reversals were absent. These patterns and locations of corresponding piezos are given in Table 3.2.

Pattern	Locations
1	E14/15 to E27
2	E16 to E36
3	E7 to E36
4	E20/23 to E31/34
5	E15/16 to E36
6	E15/16 to E26
7	E7 to E26/28
8	E16/17 to E31/34
9	E23/26 to E36
10	E14/15 to E36

Table 3.2 Verified actuation patterns

For each of the patterns in Table 3.2, the piezos were actuated by one random signal and eight individual sinusoidal signals with frequencies of 61Hz, 121Hz, 182Hz, 243 Hz, 70Hz, 140Hz, 210Hz and 280Hz. These signals were synthesized by the ZONIC 7000/ZETA data acquisition system and output to Trek Model 50/750 power amplifier to drive the piezo actuators.

The instrumentation, data acquisition system, and the measurement technique employed for the verification were all similar to those used for ODS measurements.

At Station E, the fuselage above the floor level was instrumented at thirty locations as shown in Fig. 3.14. Twelve ENDEVCO 2222C and three B&K Type 4393 accelerometers were available for measuring radial accelerations at these positions. The accelerometers were mounted in the radial directions. Due to the limited number of available accelerometers, charge amplifiers and data acquisition system channels, the measurements at the thirty stations were obtained using the available fifteen accelerometers in two separate setups .

The same number of B&K Type 2635 charge amplifiers were employed to amplify the charge signals. Once again, signal components with frequencies higher than 1 KHz were filtered out by the built-in low pass analog filter, and the output gain for each amplifier was set to achieve the highest possible signal to noise ratio.

The same ZONIC 7000/ZETA data acquisition system that was used to output signals to the Trek Model 50/750 amplifier in order to drive the actuators was also employed to collect the signals from the charge amplifiers. The Trek Model 50/750 amplifier is designed to amplify and drive high voltage but low current loads such as

piezo actuators. In this application, a gain of 150 was set which means that a 5-Volt input would create a 750-Volt output.

Fig. 3.14 Schematic of accelerometer locations for actuation pattern measurements

Both coherence and frequency response function (FRF) were determined for each acceleration measurement. In these measurements, the reference signal was the ZONIC output which was connected to the Trek Model 50/750 amplifier.

For each pattern described in Table 3.2, the following measurements were obtained:

(1) The piezo actuators were driven by 600-volt random signal with a frequency span of zero to 315.25 Hz (ZONIC output of 4 volts), and the auto-spectra, coherence and FRFs of all thirty acceleration signals were measured. The measurements from the random signal were used to assess the frequency response of the actuator over the whole frequency span of interest.

(2) Then the piezo actuators were driven by 150-volt sinusoidal signals with frequencies of 61Hz, 121Hz, 182Hz, 243 Hz, 70Hz, 140Hz, 210Hz and 280Hz, respectively, and the corresponding auto-spectra, coherence and FRFs of all thirty acceleration signals were measured. In general, for each frequency, the coherence value at the driving signal frequency was made as close to unity as possible. The same measures that were employed to improve a poor coherence in ODS measurements were also employed during verification. The magnitude and the phase of the FRF at a driving frequency were noted. Using these values, the response of the fuselage to each actuation pattern was determined for eight equidistant instants of time during a single period. The same approach that was used to determine and plot the ODSs was followed here to obtain the deflections resulting from piezo actuation. Typical responses are shown in Figs. 3.15-3.21.

Clearly, for any given ODS, the actuation pattern whose response resembles this ODS the closest, would reduce the fuselage vibration most effectively. Based on this observation, the following patterns were selected to be used in controlling the vibrations due to each of the four sound fields:

(1) Actuation pattern #5 or #6 to control the 61Hz and 70Hz ODSs.

- (2) Actuation pattern #4, #8 or #9 to control the 121Hz ODS.
- (3) Actuation pattern #4 or #8 to control the 140Hz ODS.

Fig. 3.15 Response to piezo actuation pattern #5 at 61Hz

Fig. 3.16 Response to piezo actuation pattern #6 at 61Hz

Fig. 3.17 Response to piezo actuation pattern #4 at 121Hz

Fig. 3.18 Response to piezo actuation pattern #8 at 121Hz

Fig. 3.19 Response to piezo actuation pattern #9 at 121Hz

Fig. 3.20 Response to piezo actuation pattern #4 at 140Hz

Fig. 3.21 Response to piezo actuation pattern #8 at 140Hz

Chapter Four

Control Strategy

Before presenting the specific control strategy that was developed and used in this investigation, main features of the important underlying principles that are incorporated into the control strategy are briefly outlined in the following. Specifically, the underlying principles of conventional adaptive filters, LMS algorithm, and filtered-x LMS algorithm for SISO and MIMO feedforward controls are presented in this chapter. Off-line and on-line system identification techniques are also discussed.

4.1 Adaptive Filters and LMS Algorithm

Adaptive control systems are designed to adapt their behavior to the changing properties of controlled processes and their signals.

Fig. 4.1 Block diagram of a simple adaptive controller

Fig. 4.1. illustrates a simple adaptive control system consisting of a single adaptive linear combiner, that is used to make the output $y(n)$ from a reference sequence $x(n)$ as

close as possible to the desired sequence $d(n)$ by reducing the error sequence $e(n)$. This simple adaptive linear combiner is actually a finite impulse response (FIR) digital filter. The output $y(n)$ is a linear combination of the input samples, i.e.,

$$y(n) = \sum_{i=0}^{N_a} a_i x(n-i) \quad (4.1)$$

so that the error is

$$\begin{aligned} e(n) &= d(n) - y(n) \\ &= d(n) - \sum_{i=0}^{N_a} a_i x(n-i) . \end{aligned} \quad (4.2)$$

The filter design problem is now to adjust each a_i to best reduce the error sequence, $e(n)$, according to some criterion. A very common and useful criterion is that each a_i is adjusted so that the mean of squared error sequences, $E[e^2(n)]$, is minimized [27]. Here E is the conventional *mathematical expectation*, $E[f(n)] = (f(1) + f(2) + \dots + f(n))/n$, of a random discrete distribution, and it denotes an average over ensembles of the random performance surface error sequences $e(n)$. Using equation (4.2) this mean-squared error can be written as

$$E[e^2(n)] = E\left[\left(d(n) - \sum_{i=0}^{N_a} a_i x(n-i)\right)^2\right] . \quad (4.3)$$

Also referred to as a "cost function", the quadratic function given in Equation 4.3 is most attractive, because it guarantees a unique minimum. A number of iterative or adaptive methods are available to modify the filter coefficients gradually in such a way

that the mean-squared error is reduced. Out of these, the least-mean-square (LMS) algorithm is generally the best choice for many different applications of adaptive signal processing [16], mainly because of its simplicity and ease of computation, and because it does not require off-line gradient estimations or repetitions of data.

To develop the LMS algorithm, $e^2(n)$ itself is taken as an estimate of $E[e^2(n)]$ at each iteration in the adaptive process. Then a steepest-descent type of adaptive algorithm can be specified as:

$$\begin{aligned}
 a_i^{(n+1)} &= a_i^{(n)} - \alpha \frac{\partial e^2(n)}{\partial a_i} \\
 &= a_i^{(n)} + 2\alpha e(n) \frac{\partial y(n)}{\partial a_i} \\
 &= a_i^{(n)} + \mu e(n) x(n-i) ,
 \end{aligned} \tag{4.4}$$

where $\mu=2\alpha$ is a gain constant that regulates the speed and stability of adaption. Since the weight changes at each iteration are based on imperfect gradient estimates, the adaptive process could be expected to be noisy; that is, it would not follow the true line of steepest descent on the performance surface.

From its form in equation (4.4), the elegance, simplicity and efficiency of LMS can be appreciated and it can be seen that this LMS algorithm can be implemented in a practical system without squaring, averaging or differentiation. As noted above, each component of the gradient vector is obtained from a single data sample without perturbing the weight vector. Without averaging, the gradient components do contain a large component of noise, but the noise is attenuated with time by the adaptive process, which acts as a low-pass filter in this respect.

The convergence properties of the LMS algorithm have been widely discussed in the literature [25,27]. A simplification which provides a useful "rule of thumb" for the convergence of the LMS algorithm is that the algorithm will usually converge towards the minimum solution of $E[e^2(n)]$ if the convergence coefficient μ satisfies

$$0 < \mu \leq \frac{1}{IE[X^2(n)]}, \quad (4.5)$$

where $E[X^2(n)]$ is the mean square value of the reference sequence, and I is the number of adaptive filter coefficients [25].

4.2 Recursive LMS Algorithm

Rather than employing an FIR filter, in most approaches an infinite impulse response (IIR) filter is used as the controller. Also referred to as recursive controller, the basic configuration of an IIR filter is shown in Fig.4.2, where $A(z)$ and $B(z)$ are individual FIR filters. These are combined to generate the IIR controller response given by

$$G(z) = \frac{A(z)}{1-B(z)}. \quad (4.6)$$

Fig. 4.2 Block diagram of a recursive controller

Compared with an FIR filter, an IIR filter, with both poles and zeros, offers attractive advantages such as resonance, sharp cutoff, etc., and therefore, it is capable of achieving a response similar to an FIR filter by a much shorter filter length. In other words, using an IIR filter can save a great deal of computation. However, IIR filters have two major disadvantages:

1. They become unstable if the poles move outside the unit circle during the adaptive process.
2. Their performance surfaces are generally non-quadratic and may even have local minima.

The instability problem can be prevented by adjusting the convergence coefficients and choosing proper initial values for filter coefficients. Regarding the second disadvantage, there is no well-recognized theorem to avoid local minima. One can obtain an acceptable result by trial and error on the filter length, convergence coefficient, etc.. The acceptance is then determined by observing the error reduction during the convergence process.

To develop LMS algorithm for the recursive adaptive filter as illustrated in Figure 4.2, assume the output of the recursive filter to be

$$y(n) = \sum_{i=0}^{N_a} a_i x(n-i) + \sum_{j=1}^{N_b} b_j y(n-j) , \quad (4.7)$$

where $a_i(n)$ and $b_j(n)$ are the i th direct and j th recursive coefficients in the controller at the n th sample time.

Similar to the non-recursive case, the error sequence can be written as:

$$\begin{aligned} e(n) &= d(n) - y(n) \\ &= d(n) - \sum_{i=0}^{N_a} a_i x(n-i) - \sum_{j=1}^{N_b} b_j y(n-j) . \end{aligned} \quad (4.8)$$

Considering the LMS algorithm, again the gradient approximation is used by taking $e^2(n)$ as an estimate of $E[e^2(n)]$ at each iteration in the adaptive process, and specifying a steepest-descent type of adaptive algorithm as:

$$\begin{aligned} a_i^{(n+1)} &= a_i^{(n)} - \alpha \frac{\partial e^2(n)}{\partial a_i} \\ &= a_i^{(n)} + 2\alpha e(n) \frac{\partial y(n)}{\partial a_i} \\ &= a_i^{(n)} + \mu e(n) x(n-i) \end{aligned} \quad (4.9)$$

and

$$\begin{aligned} b_j^{(n+1)} &= b_j^{(n)} - \alpha \frac{\partial e^2(n)}{\partial b_j} \\ &= b_j^{(n)} + 2\alpha e(n) \frac{\partial y(n)}{\partial b_j} \\ &= b_j^{(n)} + \mu e(n) y(n-j) \end{aligned} \quad (4.10)$$

where $\mu=2\alpha$ is a convergence coefficient that regulates the speed and stability of adaption.

4.3 SISO Filtered-x LMS Algorithm

The well-known LMS algorithm is widely used for electrical noise cancellation. If this algorithm is used without modification in an active control application as shown

in Fig. 4.3, however, the result is likely to be unstable. This is because the signal $u(n)$ from the adaptive filter, suffers a phase shift in passing through the secondary path, that is represented by filters C and D in Fig. 4.3. The instantaneous measurement of the gradient of the mean-square-error with respect to the coefficient vector, $x(n) e(n)$, is thus no longer an unbiased estimate of the true gradient. The solution to this problem is to introduce a similar phase shift into the reference signal path, before the gradient estimate is formed. This is achieved by using an electrical IIR filter that consists of two FIR filters $\hat{C}$ and $\hat{D}$, Fig. 4.4. This models the response of the secondary path and its role in generating a filtered reference signal. The reference signal is then multiplied by the error to form the gradient estimate. The resulting updated equation constitutes the so-called “filtered-x LMS” algorithm.

In this investigation, the path from the controller output to the charge amplifier output that includes the piezo amplifier, anti-image filter, piezo actuator, etc., all constitute the secondary path.

Fig. 4.3 Block diagram of a controller with secondary path

To develop the filtered-x LMS algorithm for a system having secondary path, one

may follow the general outline presented in [17] by taking $e^2(n)$ as an estimate of $E[e^2(n)]$ similar to the procedure outlined in Sections 4.1 and 4.2. Then, due to the presence of the secondary path, one has

$$\begin{aligned} e(n) &= d(n) - y(n) \\ &= d(n) - \sum_{k=0}^{N_c} c_k u(n-k) - \sum_{p=1}^{N_d} d_p y(n-p) \end{aligned} \quad (4.11)$$

and

$$u(n) = \sum_{i=0}^{N_a} a_i x(n-i) + \sum_{j=1}^{N_b} b_j u(n-j) . \quad (4.12)$$

So, for a steepest-descent type of adaptive algorithm one has:

$$\begin{aligned} a_i^{(n+1)} &= a_i^{(n)} - \alpha \frac{\partial e^2(n)}{\partial a_i} \\ &= a_i^{(n)} + 2\alpha e(n) \frac{\partial y(n)}{\partial a_i} \\ &= a_i^{(n)} + \mu e(n) r(n-i) \end{aligned} \quad (4.13)$$

and

$$\begin{aligned} b_j^{(n+1)} &= b_j^{(n)} - \alpha \frac{\partial e^2(n)}{\partial b_j} \\ &= b_j^{(n)} + 2\alpha e(n) \frac{\partial y(n)}{\partial b_j} \\ &= b_j^{(n)} + \mu e(n) s(n-j) \end{aligned} \quad (4.14)$$

where, again $\mu=2\alpha$ is a convergence coefficient that regulates the speed and stability of

adaption.

In (4.13) and (4.14), the signals $r(n)$ and $s(n)$ are filtered reference signals such that

$$r(n) = \sum_{k=0}^{N_c} c_k x(n-k) + \sum_{p=1}^{N_d} d_p r(n-p) \quad (4.15)$$

and

$$s(n) = \sum_{k=0}^{N_c} c_k u(n-k) + \sum_{p=1}^{N_d} d_p s(n-p) \quad (4.16)$$

Fig. 4.4 Block diagram of a filtered-x LMS controller (ref. [17])

Since only a modeled secondary path can be achieved using a regular LMS algorithm, the C and D in equations (4.15) and (4.16) have to be replaced by the identified secondary path that consists of $\hat{C}$ and $\hat{D}$. A block diagram of a filtered-x LMS controller is represented in Fig. 4.4.

4.4 MIMO LMS Algorithm

An MIMO system is similar to an SISO system shown in Fig. 4.2. The only difference is that for an MIMO system $y(n)$ and $x(n)$ represent vectors $[y_1(n), y_2(n), \dots, y_L(n)]^T$ and $[x_1(n), x_2(n), \dots, x_M(n)]^T$ respectively. To develop an LMS algorithm for an MIMO system, the output is given as

$$y_l(n) = \sum_{m=1}^M \sum_{i=0}^{N_a} a_{lmi}^{(n)} x_m(n-i) + \sum_{q=1}^L \sum_{j=1}^{N_b} b_{lqj}^{(n)} y_q(n-j), \quad (4.17)$$

for $l=1,2,\dots,L(L \leq M)$, where a_{lmi} and b_{lqj} are, respectively, the direct and recursive coefficients at the n th sample time; M is the number of input; L is the number of output; and N_a and N_b are the total number of coefficients a_{lmi} and b_{lqj} , respectively.

Similar to the SISO case, the sum of error squares is taken as the cost function

$$J(n) = \sum_{h=1}^L e_h^2(n) = \|e(n)\|^2 \quad (4.18)$$

with $e_h(n) = d_h(n) - y_h(n)$.

For a steepest-descent type of adaptive algorithm, the resulting update algorithm

for the control filter is

$$\begin{aligned}
 a_{lmi}^{(n+1)} &= a_{lmi}^{(n)} - \alpha \frac{\partial J(n)}{\partial a_{lmi}} \\
 &= a_{lmi}^{(n)} + 2\alpha \sum_{h=1}^L e_h(n) \frac{\partial y_h(n)}{\partial a_{lmi}} \\
 &= a_{lmi}^{(n)} + \mu \sum_{h=1}^L e_h(n) \frac{\partial y_h(n)}{\partial a_{lmi}},
 \end{aligned} \tag{4.19}$$

and

$$\begin{aligned}
 b_{lqj}^{(n+1)} &= b_{lqj}^{(n)} - \alpha \frac{\partial J(n)}{\partial b_{lqj}} \\
 &= b_{lqj}^{(n)} + 2\alpha \sum_{h=1}^L e_h(n) \frac{\partial y_h(n)}{\partial b_{lqj}} \\
 &= b_{lqj}^{(n)} + \mu \sum_{h=1}^L e_h(n) \frac{\partial y_h(n)}{\partial b_{lqj}},
 \end{aligned} \tag{4.20}$$

for $l=1,2,\dots,L$; $q=1,2,\dots,L$; $i=0,1,\dots,N_a$; and $j=1,2,\dots,N_b$, and where, as before, $\mu=2\alpha$ is a convergence coefficient that regulates the speed and stability of adaption.

From (4.17) one gets:

$$\frac{\partial y_h(n)}{\partial a_{lmi}} = \begin{cases} x_m(n-i) & \text{for } h=l \\ 0 & \text{for } h \neq l \end{cases} \tag{4.21}$$

$$\frac{\partial y_h(n)}{\partial b_{lqj}} = \begin{cases} y_q(n-j) & \text{for } h=l \\ 0 & \text{for } h \neq l \end{cases} \tag{4.22}$$

Substituting (4.21) into (4.19) and (4.22) into (4.20), the final expression for this adaptive algorithm is obtained as:

$$a_{lmi}^{(n+1)} = a_{lmi}^{(n)} + \mu e_l(n) x_m(n-i) , \quad (4.23)$$

$$b_{lqj}^{(n+1)} = b_{lqj}^{(n)} + \mu e_l(n) y_q(n-j) , \quad (4.24)$$

4.5 MIMO Filtered-x LMS Algorithm

An MIMO system with a second path(plant) is similar to an SISO system shown in Fig. 4.3. The difference is that, as outlined earlier, $x(n)$ and $y(n)$ as well as $u(n)=[u_1(n), u_2(n), \dots, u_M(n)]^T$ represent vector quantities. Then the controller output is given as

$$u_m(n) = \sum_{k=1}^K \sum_{i=0}^{N_a} a_{mki}^{(n)} x_k(n-i) + \sum_{p=1}^M \sum_{j=1}^{N_b} b_{mpj}^{(n)} u_p(n-j) , \quad (4.25)$$

for $m=1,2,\dots,M$, where a_{lmi} and b_{lqj} are the direct and recursive coefficients at the n th sample time.

Letting the plant output to be

$$y_l(n) = \sum_{h=1}^M \sum_{i=0}^{N_c} c_{lhi}^{(n)} u_h(n-i) + \sum_{q=1}^L \sum_{j=1}^{N_d} d_{lqj}^{(n)} y_q(n-j) , \quad (4.26)$$

for $l=1,2,\dots,L$ ($L \leq M$), where c_{lmi} and d_{lqj} are, respectively, the direct and recursive coefficients at the n th sample time, as before the sum of error squares is taken as the cost function

$$J(n) = \sum_{l=1}^L e_l^2(n) = \|e(n)\|^2 \quad (4.27)$$

with $e_h(n) = d_h(n) - y_h(n)$.

For a steepest-descent type of adaptive algorithm, the resulting update algorithm for the control filter is

$$\begin{aligned}
 a_{mki}^{(n+1)} &= a_{mki}^{(n)} - \alpha \frac{\partial J(n)}{\partial a_{mki}} \\
 &= a_{mki}^{(n)} + 2\alpha \sum_{l=1}^L e_l(n) \frac{\partial y_l(n)}{\partial a_{mki}} \\
 &= a_{mki}^{(n)} + \mu \sum_{l=1}^L e_l(n) r_{lmki},
 \end{aligned} \tag{4.28}$$

and

$$\begin{aligned}
 b_{mpj}^{(n+1)} &= b_{mpj}^{(n)} - \alpha \frac{\partial J(n)}{\partial b_{mpj}} \\
 &= b_{mpj}^{(n)} + 2\alpha \sum_{l=1}^L e_l(n) \frac{\partial y_l(n)}{\partial b_{mpj}} \\
 &= b_{mpj}^{(n)} + \mu \sum_{l=1}^L e_l(n) s_{lm pj},
 \end{aligned} \tag{4.29}$$

for $m=1,2,\dots,M$; $k=1,2,\dots,K$; $p=1,2,\dots,M$; $i=0,1,\dots,N_a$; $j=1,2,\dots,N_b$, where

$$\begin{aligned}
 r_{lmki}^{(n)} &= \frac{\partial y_l(n)}{\partial a_{mki}} \\
 &= \sum_{t=0}^{N_c} c_{lmt}^{(n)} x_k(n-i-t) + \sum_{q=1}^L \sum_{t=1}^{N_d} d_{lqt}^{(n)} r_{qmki}(n-t)
 \end{aligned} \tag{4.30}$$

and

$$\begin{aligned}
 s_{lm pj}^{(n)} &= \frac{\partial y_l(n)}{\partial b_{mpj}} \\
 &= \sum_{t=0}^{N_c} c_{lmt}^{(n)} u_p(n-j-t) + \sum_{q=1}^L \sum_{t=1}^{N_d} d_{lqt}^{(n)} s_{qm pj}(n-t)
 \end{aligned} \tag{4.31}$$

for $l=1,2,\dots,L$; $p=1,2,\dots,M$; $m=1,2,\dots,M$; $k=1,2,\dots,K$; $i=0,1,\dots,N_a$; $j=1,2,\dots,N_b$.

4.6 Off-line and On-line Identification

The stability of the filtered-x LMS algorithm is greatly affected by the accuracy of the filter matrices $\hat{C}$ and $\hat{D}$ which model the secondary path. There are two ways to identify the secondary path: off-line and on-line.

When the filter that is used to model the secondary path is created prior to the control, one has off-line identification. In this case, a band-limited pseudo-random training signal is fed into the secondary path. An LMS or RLMS algorithm independent of the control path could be employed. The length of the FIR or IIR filter should be selected based on the complexity of the actual case. Too few coefficients can result in absence of convergence during the adaptive process. Usually there is no certain rule to predict the sufficient number of required filter coefficients. A trial and error method has to be used.

In the off-line technique, once the identification LMS or RLMS process converges by the error approaching zero, the identification of the secondary path is completed, and the filter that models the secondary path can be used to pre-filter the reference signal in the control path.

Irrespective of how accurately an acoustical, electrical or mechanical system is modeled by using off-line techniques, in time, that system will in some way deviate from the model. Every system could be time-varying due to a number of reasons, such as the ambient temperature change, sensor or actuator degradation, etc. These inevitable system uncertainties cause inaccuracies in off-line models. Therefore, when the secondary path changes significantly over time, controllers must be capable of adapting on-line in order to be continually effective in reducing noise or vibration in practical applications.

With on-line identification of the secondary path, the identification is accomplished in parallel with control operation. Every certain number of samples, the control model is updated by copying the identified model coefficients into the control model as the pre-filter coefficients. Due to the interaction between control and the identification processes, obviously, the control model is not able to find its optimal parameters and to achieve convergence until the identification provides the appropriate secondary path models. According to [17], a sufficiently large interval of updating, such as 5,000 samples, is necessary to help the control model to converge.

The on-line identification is carried on by feeding a band-limited pseudo-random signal into the secondary path. This random signal has to meet two conditions: (1) it should not be correlated to the reference signal so that the identification LMS or RLMS will not be affected by the control signal; (2) compared to the control output signal, this random signal has to be relatively weaker so that it will not affect the final control result significantly; usually 10 percent of the control output magnitude is suitable [27].

The on-line identification technique can closely track the changes in the system to affect continual cancellation of noise or vibration and therefore is the only choice for significantly-time-varying systems. For a relatively static system, control with off-line identification might be a good choice, since due to the complexity of interaction between control and on-line identification the convergence is comparatively difficult to achieve.

Chapter Five

Control Programming and Preliminary Implementation

The control strategy was implemented on a PC based DSP system. The DSP system consists of a SPECTRUM QPC/C40B processor board with a Texas Instruments (TI) TMS320C40 processor. Input and output are provided by a SPECTRUM PC/16IO8 board.

A mixed C/Assembly code was developed using TMS320C3x/C4x C compiler V4.6, and it was debugged using SPECTRUM DB40 debugging environment. Initially, the code was tested by controlling the vibration of a "test beam" using both off-line and on-line identification techniques. A successful convergence was achieved during this testing.

5.1 The DSP System

The QPC/C40B board can support up to four TIM-40 modules each of which is equipped with a TI TMS320C40 processor and necessary memory blocks [21]. This PC-based board is used to accomplish high speed DSP and I/O control by operating either a single or multiple CPU. For this application, only a single CPU was installed on the board.

The TMS320C40 processor is the latest generation of TMS320 floating point processor family, and is one of the fastest floating point DSP processors commercially available at this time. It has been used extensively in broad applications such as general

purpose DSP, graphic or image processing, industry control, telecommunication, etc. For its detailed features, one can refer to the user manual [21].

A PC/16IO8 multi-channel I/O board was used to collect the input data and output the control signals. This board was connected directly to QPC/C40B via a DSPLINK2 interface. Sampled data was transmitted to and received from the QPC/C40B over this interface. User configurable hardware links were set in such a manner that the on-board input and output filters provided a cut-frequency of 7.2 Hz.

5.2 Mixed C/Assembly Real-time Programming

A general TMS320C3x/C4x software development flow is illustrated in Fig. 5.1. The shaded portion of the figure highlights the typical software development path; the remaining portions are optional.

Generally referred to as low-level language statements, the assembly language statements, resemble the processor's native set of instructions. It is well known that a program developed directly in assembly language can have a high efficiency and a small executable file because it can take full advantage of all the particular CPU features. The disadvantage is that a program written in a low-level language is difficult to read and write. Besides, low-level languages are specific to a processor chip. For example, a program written in a low-level language for a Motorola 68000 would have to be completely rewritten in a TMS320C40. Therefore it offers no portability at all.

C is a programming language that falls somewhere between a low-level and a high-level language. A source code in C can be translated into a processor-specified

Fig. 5.1 TMS320C3x/C4x software development flow (ref. [24])

assembly source code by a processor-specified C compiler. Also, a well-written C program can be very readable. An American National Standard Institute (ANSI) standard for the C language now exists. All these features make C a very popular language for a

large number of applications. Usually, C programs are more efficient than programs written in other high-level languages. However, they are not as proficient as those programs written in a purely low-level language since certain overheads are added into the code during translation from C to assembly, and not all the processor-specified advantages could be employed. Considering the easy development and maintenance, a C programming is usually the best choice for most industrial control applications as long as the code execution efficiency or code size is not of primary concern.

For many control applications, real-time programming is necessary. Real-time operation refers to an operation in which all the digital signal processing for one set of samples is finished before the next set becomes available for collection. Such digital processing includes collecting the data, processing the control algorithm and generating the output. The time interval between any two sets of samples is called sampling period, the inverse of which is known as sampling rate. This sampling rate needs to be sufficiently low so that the corresponding sampling period would be longer than the time required to finish all the processing for one set of samples. Therefore, a code's capacity for high sampling rate reflects a high efficiency in that code. With a higher sampling frequency, better control performance can be achieved since the discrete signal used in the digital signal processing would be closer to the actual analog signal. Moreover, this makes it possible to control systems with wider ranges of frequency.

A feedforward control strategy is proposed in this investigation. The corresponding algorithm is relatively more complicated than a conventional feedback algorithm and requires a larger amount of computation. Therefore, computational efficiency was of

critical concern, particularly for MIMO cases. Considering the advantages and disadvantages of both C and assembly languages, a compromise solution adopted for this investigation was to use mixed C and assembly programming. This translated to using C code for developing the main program while the time-critical operations such as interrupt service routines were developed using assembly code. With this approach, a certain degree of optimization was achieved.

Such an optimization was realized by employing the following programming techniques which can only be implemented in the assembly code:

(1) Circular addressing mode is used to greatly reduce the filter implementation instructions that comprise the majority of the feedforward control algorithm operations. This addressing mode is particularly designed in TMS320C3x/C4x for applications requiring circular buffer, such as filters and correlations. Much of the overhead required for the filter operation in C programming, such as array or matrix shifting, or pointer out-bound checking were omitted using circular addressing mode.

(2) Parallel instructions are used in filter and coefficient update operations. This makes it possible to implement one more instruction in one single CPU cycle.

(3) Based on the pipeline architecture of TMS320C40, certain delay branch instructions are employed to decrease the branch conflicts.

(4) With the help of DB40 debugging tools and the SIMC3x/C4x simulator, the instructions are carefully organized to reduce the register and memory conflicts.

(5) The assembly code translated from C code is modified to eliminate unnecessary register saving operations.

In addition to code optimization, the memory locations for text, data, and stack were assigned in such a way that the more frequently accessed blocks were placed in the internal memory to achieve a faster speed. The CPU cache was enabled to store more frequently used instructions and thus reduce instruction fetching time.

5.3 Program Debugging and Efficiency Analysis

In a debugging process one seeks to: (1) check the code and algorithm correctness by tracking the variables, memory or registers during execution; (2) check the efficiency of a particular routine of code by counting the cycles.

The DB40 debugger provided with the C4x Network API software was employed to debug the control program. After making sure the program was executing correctly, a code efficiency check was conducted. The time-critical operation which refers to the interrupt service routine was divided into several segments. For each of these segments, the CPU cycles were counted and compared to the corresponding number of instructions. Since, in the absence of any pipeline conflicts, one CPU cycle is needed for every instruction, when a pipeline conflict occurred, the instructions were reorganized or modified to minimize the total CPU cycles.

5.4 C/Assembly Source Files

The development of an SISO code was accomplished by creating two source files, namely, DSP.C and FIR.ASM. The main file, DSP.C, executes both identification and control using regular LMS feedforward strategy. FIR.ASM contains the time-critical

functions which are the FIR filter operation and coefficients update. DSP.C needs to be linked to FIR.ASM to create an executable file for the DSP board. The real-time sampling rate of DSP.C is allowed to be as high as 20 KHz.

The development of an MIMO code was accomplished by creating three source files, namely, MXLMS33.C, MXLMS412.C and MASM.ASM. The main file that executes both identification and control using regular LMS feedforward strategy is MXLMS33.C. Another main file, MXLMS412.C, executes both identification and control using a simplified LMS feedforward strategy in which FIR filters are used for the controller and cross terms in the recursive filters are neglected [18]. The time-critical operations such as filter operation and coefficient update are carried out by MASM.ASM. Either MXLMS33.C or MXLMS412.C needs to be linked to MASM.ASM to create an executable file for the DSP board.

At a sampling rate of 1800 Hz, MXLMS33.C can accommodate up to three outputs and three inputs, or two outputs and four inputs. On the other hand, MXLMS412.C can accommodate up to four outputs and twelve inputs at a sampling rate of 1800 Hz. Each of them can have up to fifteen direct and fourteen recursive coefficients for either identification or control filter.

The program flow diagrams for identification and control are shown in Figs. 5.2 and 5.3, respectively.

The source code and corresponding memory configurations for SISO and MIMO controls are presented in Appendices F and G, respectively.

Fig 5.2 Program Flow for the System Identification

Fig 5.3 Program flow chart for control operation

5.5 Preliminary Implementation of SISO Control

As mentioned earlier, the optimized code was implemented to run an SISO vibration control of a test beam using both off-line and on-line identification techniques. The goal of this implementation was to check the accuracy and efficiency of the code.

An aluminum beam was simply suspended by rubber bands at both ends. A piezo actuator was bonded to the beam near midspan, and a B&K Type 4393 accelerometer was

fixed adjacent to the actuator. The dimensions of the beam and the locations of the actuator and accelerometer are shown in Fig. 5.4. The disturbance was provided by an electrodynamic shaker and it was detected by a B&K Type 8200 force transducer. The TMS320C40 based controller was used to minimize the output of the accelerometer. The details of instrumentation, configuration and controller settings are the same as reported in [17].

At a sampling rate of 5000 Hz, sixteen direct and fifteen recursive coefficients were used for the control part. For the identification the corresponding numbers were eighteen direct and seventeen recursive coefficients. Acceptable convergence for both off-line and on-line techniques was observed and the control scheme was able to effectively control the vibration of the beam in real time. This indicated the success of the code and its optimization.

Fig. 5.4 The test beam, actuator and accelerometer for SISO vibration control

5.6 Preliminary Implementation of MIMO Control

The code compiled from MXLSM33.C and MASM.ASM files was implemented

to run an MIMO algorithm with two outputs and four inputs. This implementation was carried on the same test beam that was used for SISO control. Off-line identification technique was used with the intention of checking the accuracy and the efficiency of the code.

Again, the aluminum beam was simply suspended by rubber bands at both ends, Fig. 5.5. Two piezo actuators were bonded to the beam at near midspan, and two accelerometers were fixed adjacent to each actuator. An electrodynamic shaker provided a 200Hz sinusoidal disturbance that was detected by a B&K Type 8200 force transducer. The TMS320C40 based controller was used to minimize the mean square of the accelerometer outputs.

At a sampling rate of 2000 Hz, sixteen direct and fifteen recursive coefficients were employed for the control, while thirteen direct and twelve recursive coefficients were used for the identification. Satisfactory convergence was observed and the control scheme was able to effectively control the vibration of the beam by reducing the induced accelerations up to 10-20dB. This indicated a satisfactory real-time control with two outputs and four inputs at a sampling rate of 2000 Hz.

Fig. 5.5 The test beam, actuators and accelerometers for MIMO vibration control

Chapter Six

Implementation of Real-Time Vibration and Noise Control on the Fuselage

6.1 Control Configuration

The main objective of this research was to investigate the feasibility of cabin noise and vibration control using segmented piezoelectric actuators. Therefore, once the control and identification codes were optimized and tested, the real-time vibration and noise control of the fuselage was attempted. The setup used in this system identification and control is shown in Fig. 6.1. The setup may be divided into five sub-systems consisting of (1) sound field simulation system, (2) accelerometers and microphones, (3) piezo-actuators, (4) DSP controller, (5) and data acquisition and monitoring system.

The sound field simulation system includes a PC with AT-AO-10 output board, ASHLY MFA-8000 audio amplifiers and four speakers. This system was used to simulate four different sound fields which represent the real flight conditions. The investigation to control the fuselage vibration and noise was carried out for these four sound fields.

Two different sensing schemes were used in this part of the study. In the first approach, four B&K Type 4393 or ENDEVCO 2222C accelerometers were installed on the fuselage at appropriate circumferential locations at each of the longitudinal stations A, C and E to directly sense and control the fuselage accelerations. In the second approach microphones were used to sense the sound pressures inside the cabin. Nine B&K Type 4134 microphones were installed at window and aisle seats and at the standing

aisles in the first, second and third rows from the port. Using the sound pressure measurements from these microphones a fairly global sound field sensing and monitoring was achieved. Moreover, twelve B&K 2635 charge amplifiers, one B&K Type 2811 8-channel multiplexer, two B&K Type 2610 measuring amplifiers and three B&K Type 2609 amplifiers were used respectively to amplify the acceleration and sound pressure signals.

A total of 197 piezo actuators were installed at Stations A, B and C at the locations presented in Section 3.5. These piezos could be activated in any necessary combination to provide a desired bending moment on the fuselage in the circumferential direction. The magnitudes and phases of these moments were controlled by the voltage signals from the DSP controller. Two Trek Model 50/750 amplifiers were used to amplify the control signals to high voltages that were used to drive piezos.

Fig. 6.1 Schematic of system identification and control setup (ref. [11])

The DSP controller consisted of a PC, a plug-in QPC/C40B board and a PC/16IO8 board. This system collected the acceleration or sound pressure signals via ADC ports, processed the feedforward control operation, and output the control signals out of the DAC ports. To minimize the high frequency noise, a Krohn-Hite Model 3384 filter was used to smooth the output signals from the DAC ports.

The data acquisition and monitoring system consisted of the ZONIC 7000/ZETA data acquisition system and its host HP workstation. The measured accelerations and sound pressures were monitored, recorded and saved onto a disk.

6.2 System Identification of Piezo-actuated Fuselage

In this investigation the system refers to the secondary path (plant), which represents the path of the signal from the DSP board as a voltage output to the fuselage as an acceleration input. It consists of an anti-image filter, a power amplifier, piezo actuator arrays and the fuselage as a dynamic system. The objective of the system identification was to predetermine the properties of response to each of the output voltage signals of DSP controller. These properties were then used in the DSP controller to determine how to properly adjust the output signals in order to achieve the best vibration or noise cancellation. Obviously, the accuracy of the system identification greatly affects the stability of the control performance.

Since the system was modelled by an IIR filter represented by matrices $\hat{C}$ and $\hat{D}$ (cf Fig. 4.4), the system identification refers to the adaptation of these two matrices. This adaptation was carried out by the regular MIMO LMS algorithm presented in Section 4.4.

Many factors, ranging from filter construction to training signals, could affect the accuracy of identification. For any given plant, there exists some proper IIR filter that could model the plant with sufficient accuracy without requiring over-complicated calculations. Such a proper filter for the plant was developed in this study after investigating a number of different identification systems. Various types of filters with different filter lengths were considered. In each case a different type of random training signal was fed to the secondary path.

A detailed study of system identification that is described next was carried out for the piezo actuation pattern. This comprised nine identification schemes that are given in Table 6.1. In each scheme the identification process involved the following steps:

(1) The plant transfer functions between the supplied random signal and the resulting accelerations of the actuated fuselage were measured;

(2) A set of either uniformly distributed or bandpass random signals were fed into the plant, and the identification LMS code was executed to obtain the adapted model presented by $\hat{C}$ and $\hat{D}$;

(3) The transfer functions of this model presented by $\hat{C}$ and $\hat{D}$ were computed and the results were compared to the measured quantities from step (1);

(4) In each case, a sampling rate of 2000 Hz with a DSP board gain of 1.0 were used and the identification process was executed for 3 minutes. The DSP outputs and inputs were as follows:

Output #1: Voltage to actuate piezo groups A14/15-A28, C14/15-C28 and E14/15-E28;

Scheme #	Filter Type	Filter Coefficient #		Convergence Coefficient		Training Signal	
		Direct	Recursive	Direct	Recursive	Voltage	Frequency
1	IIR	15	14	0.001	0.001	2V	0-312Hz
2	IIR	15	14	0.001	0.001	2V	50-150Hz
3	IIR	15	14	0.001	0.001	3V	60-75Hz
4	IIR	6	5	0.002	0.002	2V	0-312Hz
5	FIR	15	0	0.002	none	2V	0-312Hz
6	FIR	31	0	0.001	none	2V	0-312Hz
7	IIR	15	14	0.001	0.001	3V	55-75Hz & 115-145Hz
8	IIR	15	14	0.001	0.005	2V	55-75Hz & 115-145Hz
9	IIR	31	30	0.001	0.01	3V	55-75Hz & 115-145Hz

Table 6.1 Identification cases

Output #2: Voltage to actuate piezo groups A28-A36, C28-C36 and E28-E36;

Input #1: Sum of accelerations at A16, C16 and E16;

Input #2: Sum of accelerations at A23, C23 and E23;

Input #3: Sum of accelerations at A34, C34 and E34;

Input #4: Sum of accelerations at A40, C40 and E40.

Fig. 6.2 shows a typical comparison between the identified model and the measured transfer function in scheme #8. This scheme was determined to be the best identification system.

Fig 6.2 Measured (solid) and identified (dashed) transfer functions between input #4 and output #4.

Based on the results of these nine cases, the following observations were made:

(1) Compared to an FIR filter, because of its poles, an IIR filter can achieve more accurate identification while using fewer coefficients. Therefore, considering the complexities of the plant in this investigation, IIR filter is strongly recommended for

identification.

(2) The narrowband training signals fed into the plant can increase the ratio of signal to noise considerably in the frequency bands of interest. Therefore, a more accurate identification can be achieved on such frequency bands.

(3) In Table 6.1, Case #9 where an IIR filter with thirty-one direct and thirty recursive coefficients was used, shows the best result. However, the result of Case #8, which also included an IIR filter but with half as many coefficients was only slightly different than that in Case #9. This indicates that there appears to exist an efficient filter length for every system. Exceeding this length seems to result in a larger amount of computation without much significant improvement in the accuracy of identification.

6.3 Vibration and Noise Reduction

Real-time vibration and noise control can be implemented with either off-line or on-line identification as described in Section 4.6. In this implementation, the off-line technique was used and the identification was performed before running a control case. The reasons for choosing the off-line over the on-line technique are: (1) the off-line technique requires less complicated computations and, therefore, it has the capacity of accommodating more input and output channels and achieving improved global vibration and noise reduction; (2) the electrical and mechanical properties of the plant were found to be considerably invariant during the control test that lasted about 30 minutes, therefore it was not necessary to update the plant during the control process.

An MIMO feedforward control strategy was used in order to achieve a global

vibration and noise reduction. An instrumentation configuration similar to the one employed by Grewal, A., et. al was used [11]. Fig. 6.1 shows a schematic of such a configuration without a reference microphone, because such a microphone was not used in this investigation.

As mentioned earlier, two different control approaches were investigated. In the first approach the fuselage accelerations were directly sensed and controlled, whereas in the second approach the internal sound pressure was the variable which was used for sensing. Twenty-nine different cases were investigated in the first approach (Appendix D, cases 1-29), and twelve cases in the second approach (Appendix D, cases A-L). Moreover, in the first approach, besides the accelerations the sound pressure levels were also monitored, whereas in the second approach only sound pressure levels were monitored. The open-loop and closed-loop spectra for these monitored signals were measured. As an example, the spectral results from case #26 are presented in Fig. 6.3. For each case, magnitudes at the dominant frequency were recorded for both open and closed loops. These magnitudes and their corresponding attenuations are presented in tables in Appendix D. In each of these tables, NA, NB, NC and ND represent the number of coefficients of filters A, B, $\hat{C}$, and $\hat{D}$, respectively. Also in designating an actuation pattern, a piezo group, for example, A, C, E 16-28 means that at each of the Stations A, C and E the piezo actuators between locations 16 and 28 were driven by the same output voltage in parallel. Finally, a typical designation of "Random Signal for Identification: 3V random 60-62Hz" represents a 3V random voltage with a bandwidth from 60Hz to 62Hz.

Fig 6.3 Open-loop (solid) and closed-loop (dashed) acceleration spectra of SPL at aisle seats in second row (in dB with respect to maximum open-loop value)

Main objectives of this series of investigations were to study:

- (1) the convergence of the control algorithm
- (2) the relationship between structural vibration attenuation and internal noise attenuation
- (3) the effectiveness of actuator settings and sensor positions.

Based on the analysis of the results, the following conclusions are arrived at:

- (1) Significant reductions in fuselage accelerations were observed when the control was based on direct sensing of these accelerations. Although the global internal noise reduction in this approach was not as high when compared to the results obtained from sensing the internal sound pressure (Cases A to L in Appendix D), yet considerable noise reduction was observed for various sound field pressures at 61 Hz (Case #26), 70 Hz

(Case #6), 121 Hz (Case #9) and 140 Hz (Case #21).

(2) As long as the identified model emulates the plant properly, the feedforward control results in convergence with significant attenuation of errors. These errors that were used as the input to the control system could be either the internal sound pressure or fuselage acceleration. The level of control that was achieved depended on the accuracy of the identification.

(3) Sensing the fuselage acceleration is an indirect approach to reduce the internal noise by controlling the fuselage vibration. The aim of this approach is to reach a global noise reduction by using a simple sensor setup. However, due to the complexity of coupling between the fuselage structural vibration modes and the internal acoustic modes, a reduction in the vibration energy of the fuselage does not necessarily guarantee a corresponding reduction in the internal sound level. This phenomenon can be observed in a few cases where achieving reductions as high as 20 dB in accelerations does not result in an appreciable reduction in noise. In fact, in some cases the reduction of accelerations had an adverse effect on the associated sound level.

(4) In order to achieve a good reduction in sound level, a proper combination of actuator setting and accelerometer positioning is critical in controlling the structural vibration modes that couple with the internal acoustic modes most severely.

(5) Since the signal to control is a narrowband one dominated by the blade passage frequency and one or two of its harmonics, the controller does not need to be too complicated. An FIR rather than an IIR shall be sufficient. Moreover, the filter length can also be relatively short, resulting in considerable savings in the operations.

Chapter Seven

Conclusion and Recommendation

7.1 Conclusion

An experimental investigation of the active control of turbo-prop cabin noise using piezoelectric actuators and feedforward control strategy is presented in this thesis. It presents a detailed description of the experiments conducted and methods of analysis used, the development steps taken, the results obtained, and the relevant conclusions arrived at based on this investigation. The major contributions are as follows:

(1) A laboratory simulation of propeller pressure footprints was developed by using the 4-speaker sound source system available at SMPL/IAR/NRC. An effective procedure was developed and implemented in which the important characteristics of the field data representing the propeller pressure footprints, such as magnitude and phase distribution, were simulated.

(2) The operating deflection shapes (ODSs) for various sound fields of interest were measured. Based on the analysis of ODS results, appropriate actuation patterns were suggested that were also verified experimentally as to their effectiveness.

(3) A feedforward control strategy was adopted in this investigation. Effective real-time control programs were developed for Single-Input and Single-Output (SISO) as well as Multi-Input and Multi-Output (MIMO) feedforward controllers.

(4) The experimental implementation of the adaptive feedforward control strategy was carried out first on a test beam then on a full-scale fuselage. The off-line system identification for the fuselage was studied experimentally by testing various cases in

which the effects of different filter types, filter lengths and random signals were considered. A total of forty-eight different schemes were investigated for the vibration and noise control of the fuselage. Depending on the type of signal to be sensed, significant reductions were achieved in either the fuselage acceleration or the internal noise of the cabin. Finally, issues related to control convergence and suggestions for reduction of the fuselage vibration and/or internal cabin noise are addressed.

This preliminary study has demonstrated the feasibility of active control of turbo-prop cabin noise using piezoelectric actuators and feedforward control strategy. It demonstrates the potential of implementing this technique by the turbo-prop aircraft industry.

7.2 Recommendation

In order to improve the control, and with a view of actual industrial implementation of this technique, the following further investigations are recommended:

(1) For a practical application, the control of a disturbance with continuously varying frequency has to be investigated. An accurate system identification will be necessary over the whole frequency span of interest. Special attention should be paid to those frequency bands where the transfer function for a secondary path is not smooth.

(2) In this study, the piezo actuator set-up was capable of accommodating a sound disturbance at 20dB or 30 dB below the SPL at real flight conditions. To increase the capacity of actuation and to provide sufficient bending moment on the fuselage, the feasibility of adapting more powerful piezo actuators or multi-layered piezo actuators

needs to be studied.

(3) The achieved results indicate that a given single actuator array can be suitable for only one or two frequencies of disturbance. In order to accommodate a wide range of frequencies one should consider a collection of individual shorter piezo arrays with a number of control channels.

References

- [1] *ASHLY MFA-8000 Operating Manual*, ASHLY AUDIO INC, 1994.
- [2] Bao, C., Sas, P., and Brussel, H. Van, "Comparison of Two On-line Identification Algorithms for Active Control", *Proc. Recent Advances in Active Control of Sound and Vibration*, 1993, pp. 38-50
- [3] Bergland, G. D., "A Guided Tour of the Fast Fourier Transform", *IEEE Spectrum*, Vol. 6, July 1969
- [4] Bies, D.A. and Hansen, C.H., *Engineering Noise Control, Theory and Practice*, E & FN SPON, London, 1996
- [5] Brüel & Kjær, *Condenser Microphone and Microphone Preamplifiers for Acoustic Measurements*, Data Handbook, September 1982.
- [6] Elliott, S.J. and Nelson, P.A., "Active Noise Control", *IEEE Signal Processing Magazine*, October 1993, pp. 12-35.
- [7] Elliott, S.J., "Active Control of Structure-Borne Noise", *Journal of Sound and Vibration*, Vol. 177, No. 5, 1994, pp. 651-673
- [8] Elliott, S.J., Boucher C. C. and Nelson, P.A., "The Behavior of a Multiple Channel Active Control System", *IEEE Transactions Signal Processing*, Vol. 40, No. 5, May 1992, pp. 1041-1052
- [9] Elliott, S.J., Stothers and Nelson, P.A., "A Multiple Error LMS Algorithm and Its Application to the Active Control of Sound and Vibration", *IEEE Transactions on Acoustics, Speech and Signal Processing*, Vol. ASSP 35, No. 10, October 1987, pp.1423-1434
- [10] Fuller, C.R., Elliott, S.J. and Nelson, P.A., *Active Control of Vibration*, Academic Press Ltd., London, 1996
- [11] Grewal, A., Nitzsche, F., Zimcik D. G. and Leigh B., "Active Control of Aircraft Cabin Noise Using Smart Structures", *AIAA/ASME/AHS Adaptive Structure Form*,

AIAA Paper 96-1274, Salt Lake City, Utah, April 1996

- [12] Grewal, A., Zimcik D. G., Hurtubise, L.C. and Leigh B., "Active Noise and Vibration Control of Turboprop Aircraft Cabin Using Multiple Piezoelectric Actuator", *3rd AIAA/CEAS Aeroacoustics Conference*, Atlanta, GA, May 1997
- [13] Grewal, A., Zimcik D. G. and Lapointe, R., "Vibro-Acoustic Modelling in Aircraft Cabin Noise Transmission and Control", *38th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference and Exhibit, AIAA/ASME/AHS Adaptive Structures Forum*, Kissimmee, FL, April, 1997
- [14] MacMartin, D. G., "Collocated Structural Control For Reduction of Aircraft Cabin Noise", *Journal of Sound and Vibration*, Vol. 190, No. 1, 1996, pp 1-5-119
- [15] National Instruments, *DAQ AT-AO-6/10 User Manual*, September 1994.
- [16] Nelson, P.A. and Elliot, S.J., *Active Control of Sound* , Academy Press, 1990.
- [17] Pavel, L., Grewal, A. and Hurtubise, L. C. "Beam Vibration Control Experiments using Adaptive Feedforward Algorithms", *NRC Laboratory Technical Report*, June, 1997.
- [18] Pavel, L., Personal Communication, 1997
- [19] Pennwalt Corporation, *Kynar Piezo Film*, Technical Manual, 1987
- [20] Spectrum Signal Processing Inc, *PC/16IO8 Multi Channel I/O Board*, 1994
- [21] Spectrum Signal Processing Inc, *QPC/C40B Board User Manual*, 1994
- [22] Thomas, D. R., Nelson, P.A. and Elliott, S.J., "Active Control of the Transmission of Sound Through a Thin Cylindrical Shell, Part I: The Minimization of Vibration Energy", *Journal of Sound and Vibration*, Vol. 167, No. 1, 1993, pp. 91-111
- [23] TI Digital Signal Processing Solutions, *Design of Active Noise Control System With the TMS320 Family*, 1996

- [24] TI Digital Signal Processing Products, *TMS320C3x/C4x C Compiler User's Guide*, 1997
- [25] Tokhi, M.O. and Leitch, R.R., *Active Noise Control*, Clarendon Press, Oxford, 1992.
- [26] Warner, J.V. and Bernhard, R.J., "Digital Control of Local Sound Fields in an Aircraft Passenger Compartment", *AIAA Journal*, Vol.28 No 2, 1990, pp. 284-289.
- [27] Widrow, B. and Stearns, S.D., *Adaptive Signal Processing*, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1985.
- [28] Xu W., and Afagh, F. F., "Active Control of Turbo-prop Cabin Noise Using Piezoelectric Actuators - Task Report No. 1", PWGSC Contract No. 31184-6-0728/001/ST, Carleton University, Ottawa, Canada, March 1997
- [29] Xu W., and Afagh, F. F., "Active Control of Turbo-prop Cabin Noise Using Piezoelectric Actuators - Task Report No. 2", PWGSC Contract No. 31184-6-0728/001/ST, Carleton University, Ottawa, Canada, May 1997
- [30] Xu, W., and Afagh, F. F., "Active Control of Turbo-prop Cabin Noise Using Piezoelectric Actuators - Task Report No. 3", PWGSC Contract No. 31184-6-0728/001/ST, Carleton University, Ottawa, Canada, September 1997
- [31] Xu, W., and Afagh, F. F., "Active Control of Turbo-prop Cabin Noise Using Piezoelectric Actuators - Task Report No. 4", PWGSC Contract No. 31184-6-0728/001/ST, Carleton University, Ottawa, Canada, December 1997

Appendix A Accelerometer Positions for ODS Measurements

Accel. No.	Circumferential Position		Axial Position	
	Stringer	Angle (deg)	C (frame) X291.4	E (frame) X312.35
1	(22P, 23P)	-125	C01	E01
2	(22P)	-120.9	C02	E02
3	(20P, 21P)	-113.0	C05	E05
4	(19P)	-105.1	C08	E08
5	(17P, 18P)	-97.9	C11	E11
6	(16P)	-90.3	C14	E14
7	(14P, 15P)	-80.7	C15	E15
8	(13P)	-71.0	C16	E16
9	(12P, 13P)	-68.1	C17	E17
10	(11P)	-59.7	C20	E20
11	(9P, 10P)	-49.8	C23	E23
12	(8P)	-41.0	C26	E26
13	(7P)	-35.1	C28	E28
14	(5P, 6P)	-25.9	C31	E31
15	(4P)	-16.6	C34	E34
16	(2P, 3P)	-7.8	C37	E37
17	(1P/S)	0	C40	E40
18	(2S, 3S)	7.8	C43	E43
19	(4S)	16.6	C46	E46
20	(5S, 6S)	25.9	C49	E49
21	(7S)	35.1	C52	E52
22	(8S)	41.0	C54	E54
23	(9S, 10S)	49.8	C57	E57
24	(11S)	59.7	C60	E60
25	(12S, 13S)	68.1	C63	E63
26	(13S)	71.0	C64	E64
27	(14S, 15S)	80.7	C65	E65
28	(16S)	90.3	C66	E66
29	(17S, 18S)	97.9	C69	E69
30	(19S)	105.1	C72	E72
31	(20S, 21S)	113.0	C75	E75
32	(22S)	121.0	C78	E78
33	(22S, 23S)	125	C79	E79

Appendix B Measured FRFs for ODSs

Station C:

Location	Magnitude:				Phase:			
	61Hz	121Hz	70Hz	140Hz	61Hz	121Hz	70Hz	140Hz
1	0.28	0.63	0.25	0.43;	2	91	11	6 ;
2	0.42	0.79	0.33	0.71;	352	100	342	10 ;
5	0.42	1.65	0.45	1.41;	344	88	343	13 ;
8	0.36	2.46	0.62	1.28;	332	76	334	7 ;
11	0.32	3.08	0.73	0.71;	298	63	324	339;
14	0.34	3.20	0.82	0.91;	228	45	315	263;
15	0.72	1.22	0.62	0.91;	198	320	278	215;
16	0.95	3.33	0.62	0.62;	186	255	226	134;
17	0.90	3.61	0.76	0.54;	187	248	219	98 ;
20	0.80	3.81	0.97	0.72;	177	232	118	59 ;
23	0.54	2.88	1.05	0.40;	157	233	168	134;
26	0.33	1.28	1.02	0.95;	114	210	146	190;
28	0.40	0.31	0.80	1.08;	40	120	142	196;
31	0.82	1.58	0.56	0.79;	17	43	100	190;
34	1.15	2.59	0.73	0.18;	7	33	45	138;
37	1.44	2.29	1.26	0.38;	351	3	29	341;
40	1	1	1	1;	0	0	0	0;
43	0.65	0.35	0.97	0.95;	357	268	344	350;
46	0.11	0.99	0.73	0.51;	339	204	332	328;
49	0.49	1.44	0.26	0.22;	191	187	276	191;
52	0.97	1.02	0.44	0.59;	181	177	181	152;
54	1.17	0.76	0.69	0.53;	181	180	164	145;
57	1.19	0.18	1.06	0.49;	179	0	160	82 ;
60	0.93	0.71	1.10	0.63;	178	340	157	15 ;
63	0.47	0.72	0.85	0.86;	175	4	143	345;
64	0.24	0.50	0.72	0.92;	168	8	137	338;
65	0.32	0.89	0.39	0.03;	7	126	73	66 ;
66	0.75	1.35	0.57	0.89;	0	121	357	148;
69	0.84	0.59	0.77	0.54;	-1	116	341	119;
72	0.66	0.02	0.71	0.22;	359	60	338	28 ;
75	0.46	0.44	0.62	0.37;	353	299	321	0 ;
78	0.16	0.46	0.38	0.37;	344	322	305	332;
79	0.03	0.44	0.32	0.38;	297	307	281	322;

Station E:

Location	Magnitude:				Phase (deg.):			
	61Hz	121Hz	70Hz	140Hz	61Hz	121Hz	70Hz	140Hz
1	0.22	0.14	0.17	0.28;	151	113	11	9 ;
2	0.27	0.32	0.05	0.54	354	119	63	27 ;
5	0.60	0.89	0.45	1.08;	357	138	14	49;
8	0.38	0.57	0.35	0.66;	312	181	79	77 ;
11	0.14	0.73	0.41	0.56	291	43	332	302
14	0.20	0.69	0.25	0.43;	308	46	338	326;
15	1.05	0.82	0.63	0.83;	198	9	229	268;
16	1.10	1.15	0.56	0.46;	187	332	204	224;
17	1.08	1.34	1.11	0.82;	194	310	197	210;
20	0.72	0.74	0.97	1.11;	180	289	191	184;
23	0.56	0.82	1.08	1.25;	164	215	168	163;
26	0.32	0.99	0.69	0.94;	130	190	115	156;
28	0.30	0.96	0.64	0.69;	51	182	129	150;
31	0.83	0.56	0.65	0.18;	22	155	64	146;
34	1.46	0.58	1.05	0.31;	2	40	29	285;
37	0.96	0.75	1.07	0.58;	2	19	23	349;
40	1	1	1	1;	0	0	0	0;
43	0.80	0.86	0.80	0.88;	336	353	338	352;
46	0.24	0.44	0.56	0.45;	286	322	326	342;
49	0.64	0.34	0.08	0.18;	188	177	186	240;
52	1.33	0.47	0.33	0.39;	178	159	154	178;
54	1.48	0.45	0.38	0.42;	177	120	142	151;
57	1.33	0.45	0.81	0.40;	172	29	121	101;
60	0.99	0.73	0.66	0.45;	166	350	129	60;
63	0.67	0.54	0.57	0.15;	165	337	129	36;
64	0.23	0.96	0.55	0.39;	137	322	104	337;
65	0.31	0.39	0.39	0.21;	46	227	62	256;
66	0.83	1.04	0.47	0.20;	7	169	345	211;
69	0.95	1.12	0.54	0.11;	2	153	333	96;
72	0.12	0.79	0.38	0.30;	13	139	338	54;
75	0.24	0.47	0.35	0.29;	323	126	342	53;
78	0.18	0.10	0.36	0.10;	253	15	243	337;
79	0.20	0.23	0.37	0.06;	309	344	249	6;

Appendix C Accelerometer Positions to Measure Deflection Patterns of the Piezo-Actuated Fuselage

Accel. No.	Circumferential Position		Axial Position
	Stringer	Angle (deg)	E (frame) X312.35
1	(22P, 23P)	-125	E01
2	(22P)	-120.9	E02
3	(20P, 21P)	-113.0	E05
4	(19P)	-105.1	E08
5	(17P, 18P)	-97.9	E11
6	(16P)	-90.3	E14
7	(14P, 15P)	-80.7	E15
8	(13P)	-71.0	E16
9	(11P)	-59.7	E20
10	(9P, 10P)	-49.8	E23
11	(8P)	-41.0	E26
12	(7P)	-35.1	E28
13	(5P, 6P)	-25.9	E31
14	(4P)	-16.6	E34
15	(2P, 3P)	-7.8	E37
16	(1P/S)	0	E40
17	(2S, 3S)	7.8	E43
18	(4S)	16.6	E46
19	(5S, 6S)	25.9	E49
20	(7S)	35.1	E52
21	(8S)	41.0	E54
22	(9S, 10S)	49.8	E57
23	(11S)	59.7	E60
24	(13S)	71.0	E64
25	(14S, 15S)	80.7	E65
26	(16S)	90.3	E66
27	(17S, 18S)	97.9	E69
28	(19S)	105.1	E72
29	(20S, 21S)	113.0	E75
30	(22S, 23S)	125	E79

Appendix D

Results of Fuselage Vibration and Noise Control

Control Case A

Test Date: 10-23-97

Error Sensor: Microphone

Simulated sound field: 61Hz; 30dB below the SPL at flight condition
Regular MIMO filtered-x LMS

Number of input channels: 3

(1) Mic. #2

(2) Mic. #5

(3) Mic. #8

Number of output channels: 2

(1) Piezo group A,C,E 16-28

(2) Piezo group A,C,E 30-36

Control Settings:

NA=15

NB=14

NC=15

ND=14

Random Signal for Identification: 3V random 60-62 Hz

Convergence coeff. for control=0.001

Convergence coeff. for identif.=0.001

Sampling Rate=2000 Hz

DSP Gain=10

Noise Attenuations:

Mic Voltage (mV)*

Mic#	Location**	Open Loop	Closed Loop	Attenu. (dB)
1	Row1, W.	26	20	2.3
2	Row2, W.	37	15	7.8
3	Row3, W.	48	35	2.7
4	Row1, A.	43	14	9.7
5	Row2, A.	50	9	14.9
6	Row3, A.	66	18	11.3
7	Row1, S.	79	20	11.9
8	Row2, S.	96	5	25.7
9	Row3, S.	102	23	12.9

* 1 mV represents 83 uPa

** W. and A. refer to window and aisle seats. S. refers to standing aisle

Control Case B

Test Date: 10-23-97

Error Sensor: Microphone

Simulated sound field: 70Hz; 30dB below the SPL at flight condition
 Regular MIMO filtered-x LMS

Number of input channels: 3

(1) Mic. #2

(2) Mic. #5

(3) Mic. #8

Number of output channels: 2

(1) Piezo group A,C,E 16-28

(2) Piezo group A,C,E 30-36

Control Settings:

NA=15

NB=14

NC=15

ND=14

Random Signal for Identification: 3V random 69-71 Hz

Convergence coeff. for control=0.001

Convergence coeff. for identif.=0.001

Sampling Rate=2000 Hz

DSP Gain=10

Noise Attenuations:

Mic Voltage (mV)*

Mic#	Location**	Open Loop	Closed Loop	Attenu. (dB)
1	Row1, W.	126	91	2.8
2	Row2, W.	147	86	4.7
3	Row3, W.	164	74	6.9
4	Row1, A.	99	77	2.2
5	Row2, A.	69	44	3.9
6	Row3, A.	80	30	8.5
7	Row1, S.	41	60	-3.3
8	Row2, S.	11	22	-6.0
9	Row3, S.	51	48	0.5

* 1 mV represents 83 uPa

** W. and A. refer to window and aisle seats. S. refers to standing aisle

Control Case C

Test Date: 10-23-97

Error Sensor: Microphone

Simulated sound field: 121Hz; 20dB below the SPL at flight condition
Regular MIMO filtered-x LMS

Number of input channels: 3

(1) Mic. #2

(2) Mic. #5

(3) Mic. #8

Number of output channels: 2

(1) Piezo group A,C,E 16-28

(2) Piezo group A,C,E 30-36

Control Settings:

NA=15

NB=14

NC=15

ND=14

Random Signal for Identification: 3V random 120-122 Hz

Convergence coeff. for control=0.001

Convergence coeff. for identif.=0.001

Sampling Rate=2000 Hz

DSP Gain=10

Noise Attenuations:

Mic Voltage (mV)*

Mic#	Location**	Open Loop	Closed Loop	Attenu. (dB)
1	Row1, W.	155	99	3.9
2	Row2, W.	122	17	17.1
3	Row3, W.	169	48	10.9
4	Row1, A.	80	40	6.0
5	Row2, A.	58	20	9.2
6	Row3, A.	75	45	4.4
7	Row1, S.	146	45	10.2
8	Row2, S.	87	66	2.4
9	Row3, S.	108	53	6.2

* 1 mV represents 83 uPa

** W. and A. refer to window and aisle seats. S. refers to standing aisle

Control Case D

Test Date: 10-23-97

Error Sensor: Microphone

Simulated sound field: 140Hz; 30dB below the SPL at flight condition
 Regular MIMO filtered-x LMS

Number of input channels: 3

(1) Mic. #2

(2) Mic. #5

(3) Mic. #8

Number of output channels: 2

(1) Piezo group A,C,E 16-28

(2) Piezo group A,C,E 30-36

Control Settings:

NA=15

NB=14

NC=15

ND=14

Random Signal for Identification: 2V random 139-141 Hz

Convergence coeff. for control=0.001

Convergence coeff. for identif.=0.001

Sampling Rate=2000 Hz

DSP Gain=10

Noise Attenuations:

Mic Voltage (mV)*

Mic#	Location**	Open Loop	Closed Loop	Attenu. (dB)
1	Row1, W.	96	37	8.3
2	Row2, W.	110	6	25.3
3	Row3, W.	99	80	1.9
4	Row1, A.	43	16	8.6
5	Row2, A.	35	22	4.0
6	Row3, A.	42	51	-1.7
7	Row1, S.	44	26	4.6
8	Row2, S.	53	11	13.7
9	Row3, S.	30	84	-8.9

* 1 mV represents 83 uPa

** W. and A. refer to window and aisle seats. S. refers to standing aisle

Control Case E

Test Date: 10-23-97

Error Sensor: Microphone

Simulated sound field: 61Hz; 30dB below the SPL at flight condition
 Regular MIMO filtered-x LMS

Number of input channels: 3

(1) Mic. #2

(2) Mic. #5

(3) Mic. #8

Number of output channels: 3

(1) Piezo group A,C,E 12-18

(2) Piezo group A,C,E 18-28

(3) Piezo group A,C,E 28-36

Control Settings:

NA=15

NB=14

NC=15

ND=14

Random Signal for Identification: 3V random 60-62 Hz

Convergence coeff. for control=0.001

Convergence coeff. for identif.=0.001

Sampling Rate=1800 Hz

DSP Gain=10

Noise Attenuations:

Mic Voltage (mV)*

Mic#	Location**	Open Loop	Closed Loop	Attenu. (dB)
1	Row1, W.	26	25	0.3
2	Row2, W.	35	9	11.8
3	Row3, W.	46	15	9.7
4	Row1, A.	45	20	7.0
5	Row2, A.	50	8	15.9
6	Row3, A.	65	10	16.3
7	Row1, S.	79	22	11.1
8	Row2, S.	94	6	23.9
9	Row3, S.	99	19	14.3

* 1 mV represents 83 uPa

** W. and A. refer to window and aisle seats. S. refers to standing aisle

Control Case F

Test Date: 10-23-97

Error Sensor: Microphone

Simulated sound field: 70Hz; 30dB below the SPL at flight condition
 Regular MIMO filtered-x LMS

Number of input channels: 3

- (1) Mic. #2
- (2) Mic. #5
- (3) Mic. #8

Number of output channels: 3

- (1) Piezo group A,C,E 12-18
- (2) Piezo group A,C,E 18-28
- (3) Piezo group A,C,E 28-36

Control Settings:

NA=15

NB=14

NC=15

ND=14

Random Signal for Identification: 3V random 69-71 Hz

Convergence coeff. for control=0.001

Convergence coeff. for identif.=0.001

Sampling Rate=1800 Hz

DSP Gain=10

Noise Attenuations:

Mic Voltage (mV)*

Mic#	Location**	Open Loop	Closed Loop	Attenu. (dB)
1	Row1, W.	137	39	10.9
2	Row2, W.	153	29	14.4
3	Row3, W.	178	67	8.5
4	Row1, A.	101	29	10.8
5	Row2, A.	66	21	9.9
6	Row3, A.	90	30	9.5
7	Row1, S.	33	44	-2.5
8	Row2, S.	13	27	-6.3
9	Row3, S.	58	30	5.7

* 1 mV represents 83 uPa

** W. and A. refer to window and aisle seats. S. refers to standing aisle

Control Case G

Test Date: 10-23-97

Error Sensor: Microphone

Simulated sound field: 121Hz; 20dB below the SPL at flight condition
 Regular MIMO filtered-x LMS

Number of input channels: 3

(1) Mic. #2

(2) Mic. #5

(3) Mic. #8

Number of output channels: 3

(1) Piezo group A,C,E 12-18

(2) Piezo group A,C,E 18-28

(3) Piezo group A,C,E 28-36

Control Settings:

NA=15

NB=14

NC=15

ND=14

Random Signal for Identification: 2V random 120-122 Hz

Convergence coeff. for control=0.001

Convergence coeff. for identif.=0.001

Sampling Rate=1800 Hz

DSP Gain=10

Noise Attenuations:

Mic Voltage (mV)*

Mic#	Location**	Open Loop	Closed Loop	Attenu. (dB)
1	Row1, W.	155	193	-1.9
2	Row2, W.	114	16	17.1
3	Row3, W.	151	156	-0.3
4	Row1, A.	82	86	-0.4
5	Row2, A.	62	28	6.9
6	Row3, A.	67	73	-0.7
7	Row1, S.	135	105	2.2
8	Row2, S.	90	40	7.0
9	Row3, S.	114	103	0.9

* 1 mV represents 83 uPa

** W. and A. refer to window and aisle seats. S. refers to standing aisle

Control Case H

Test Date: 10-23-97

Error Sensor: Microphone

Simulated sound field: 140Hz; 30dB below the SPL at flight condition
 Regular MIMO filtered-x LMS

Number of input channels: 3

(1) Mic. #2

(2) Mic. #5

(3) Mic. #8

Number of output channels: 3

(1) Piezo group A,C,E 12-18

(2) Piezo group A,C,E 18-28

(3) Piezo group A,C,E 28-36

Control Settings:

NA=15

NB=14

NC=15

ND=14

Random Signal for Identification: 2V random 139-141 Hz

Convergence coeff. for control=0.001

Convergence coeff. for identif.=0.001

Sampling Rate=1800 Hz

DSP Gain=10

Noise Attenuations:

Mic Voltage (mV)*

Mic#	Location**	Open Loop	Closed Loop	Attenu. (dB)
1	Row1, W.	89	2	33.0
2	Row2, W.	107	16	16.5
3	Row3, W.	98	85	1.2
4	Row1, A.	40	21	5.6
5	Row2, A.	33	16	6.3
6	Row3, A.	40	50	-1.9
7	Row1, S.	46	37	1.9
8	Row2, S.	54	43	2.0
9	Row3, S.	30	55	-5.3

* 1 mV represents 83 uPa

** W. and A. refer to window and aisle seats. S. refers to standing aisle

Control Case I

Test Date: 12-09-97

Error Sensor: Microphone

Simulated sound field: 61Hz; 30dB below the SPL at flight condition
Regular MIMO filtered-x LMS

Number of input channels: 3

(1) Mic. #2

(2) Mic. #5

(3) Mic. #8

Number of output channels: 3

(1) Piezo group A 12-28

(2) Piezo group C 12-28

(3) Piezo group E 12-28

Control Settings:

NA=15

NB=0

NC=15

ND=14

Random Signal for Identification: 3V random 60-62 Hz

Convergence coeff. for control=0.001

Convergence coeff. for identif.=0.001

Sampling Rate=1800 Hz

DSP Gain=10

Noise Attenuations:

Mic Voltage (mV)*

Mic#	Location**	Open Loop	Closed Loop	Attenu. (dB)
1	Row1, W.	14	12	1.3
2	Row2, W.	20	6	10.5
3	Row3, W.	41	19	6.7
4	Row1, A.	31	11	9.0
5	Row2, A.	34	4	18.6
6	Row3, A.	41	9	13.2
7	Row1, S.	57	7	18.2
8	Row2, S.	71	7	20.1
9	Row3, S.	74	16	13.3

* 1 mV represents 83 uPa

** W. and A. refer to window and aisle seats. S. refers to standing aisle

Control Case J

Test Date: 12-09-97

Error Sensor: Microphone

Simulated sound field: 70Hz; 30dB below the SPL at flight condition
 Regular MIMO filtered-x LMS

Number of input channels: 3

- (1) Mic. #2
- (2) Mic. #5
- (3) Mic. #8

Number of output channels: 3

- (1) Piezo group A 12-28
- (2) Piezo group C 12-28
- (3) Piezo group E 12-28

Control Settings:

NA=15

NB=0

NC=15

ND=14

Random Signal for Identification: 3V random 69-71 Hz

Convergence coeff. for control=0.001

Convergence coeff. for identif.=0.001

Sampling Rate=1800 Hz

DSP Gain=10

Noise Attenuations:

Mic Voltage (mV)*

Mic#	Location**	Open Loop	Closed Loop	Attenu. (dB)
1	Row1, W.	138	111	1.9
2	Row2, W.	154	115	2.5
3	Row3, W.	183	135	2.6
4	Row1, A.	110	92	1.6
5	Row2, A.	70	60	1.3
6	Row3, A.	91	65	2.9
7	Row1, S.	39	46	-1.4
8	Row2, S.	37	33	1.0
9	Row3, S.	71	63	1.0

* 1 mV represents 83 μ Pa

** W. and A. refer to window and aisle seats. S. refers to standing aisle

Control Case K

Test Date: 12-09-97

Error Sensor: Microphone

Simulated sound field: 121Hz; 20dB below the SPL at flight condition
Regular MIMO filtered-x LMS

Number of input channels: 3

- (1) Mic. #2
- (2) Mic. #5
- (3) Mic. #8

Number of output channels: 3

- (1) Piezo group A 12-28
- (2) Piezo group C 12-28
- (3) Piezo group E 12-28

Control Settings:

NA=15

NB=0

NC=15

ND=14

Random Signal for Identification: 2V random 120-122 Hz

Convergence coeff. for control=0.001

Convergence coeff. for identif.=0.001

Sampling Rate=1800 Hz

DSP Gain=10

Noise Attenuations:

Mic Voltage (mV)*

Mic#	Location**	Open Loop	Closed Loop	Attenu. (dB)
1	Row1, W.	115	141	-1.8
2	Row2, W.	78	49	4.0
3	Row3, W.	148	134	0.9
4	Row1, A.	75	121	-4.2
5	Row2, A.	49	58	-1.5
6	Row3, A.	59	69	-1.4
7	Row1, S.	148	169	-1.2
8	Row2, S.	86	69	1.9
9	Row3, S.	108	122	-1.1

* 1 mV represents 83 uPa

** W. and A. refer to window and aisle seats. S. refers to standing aisle

Control Case L

Test Date: 12-09-97

Error Sensor: Microphone

Simulated sound field: 140Hz; 30dB below the SPL at flight condition
 Regular MIMO filtered-x LMS

Number of input channels: 3

- (1) Mic. #2
- (2) Mic. #5
- (3) Mic. #8

Number of output channels: 3

- (1) Piezo group A 12-28
- (2) Piezo group C 12-28
- (3) Piezo group E 12-28

Control Settings:

NA=15

NB=0

NC=15

ND=14

Random Signal for Identification: 2V random 139-141 Hz

Convergence coeff. for control=0.001

Convergence coeff. for identif.=0.001

Sampling Rate=1800 Hz

DSP Gain=10

Noise Attenuations:

Mic Voltage (mV)*

Mic#	Location**	Open Loop	Closed Loop	Attenu. (dB)
1	Row1, W.	93	139	-3.5
2	Row2, W.	101	76	2.5
3	Row3, W.	102	35	9.3
4	Row1, A.	48	39	1.8
5	Row2, A.	33	33	0.0
6	Row3, A.	40	35	1.2
7	Row1, S.	48	79	-4.3
8	Row2, S.	48	41	1.4
9	Row3, S.	29	93	-10.1

* 1 mV represents 83 uPa

** W. and A. refer to window and aisle seats. S. refers to standing aisle

Control Case #1

Test Date: 9-30-97

Error Sensor: Accelerometer

Simulated sound field: 61Hz; 30dB below the SPL at flight condition
Regular MIMO filtered-x LMS

Number of input channels: 4

(1) Accel. A16+C16+E16

(2) Accel. A23+C23+E23

(3) Accel. A34+C34+E34

(4) Accel. A40+C40+E40

Number of output channels: 2

(1) Piezo group A,C,E 14/15-28

(2) Piezo group A,C,E 28-36

Control Settings:

NA=15

NB=14

NC=15

ND=14

Random Signal for Identification: 2V random 50-80 Hz

Convergence coeff. for control=0.001

Convergence coeff. for identif.=0.001

Sampling Rate=2000 Hz

DSP Gain=10

Acceleration and Noise Attenuations:

Accel. (mm/s/s)					Mic Voltage (mV)*				
Ch#	Location	Open Loop	Closed Loop	Attenu. (dB)	Mic#	Location**	Open Loop	Closed Loop	Attenu. (dB)
1	E16	155	83	5.4	1	Row1, W.	15	37	-7.8
2	E23	128	39	10.3	2	Row2, W.	48	48	0.0
3	E34	169	50	10.6	3	Row3, W.	62	73	-1.4
4	E40	157	47	10.5	4	Row1, A.	39	13	9.5
5	C16	149	60	7.9	5	Row2, A.	52	36	3.2
6	C23	139	62	7.0	6	Row3, A.	48	32	3.5
7	C34	201	77	8.3	7	Row1, S.	55	39	3.0
8	C40	167	43	11.8	8	Row2, S.	82	33	7.9
9	A16	103	55	5.4	9	Row3, S.	96	38	8.0
10	A23	127	35	11.2					
11	A34	167	73	7.2					
12	A40	144	26	14.9					

* 1 mV represents 83 uPa

** W. and A. refer to window and aisle seats. S. refers to standing aisle

Control Case #2

Test Date: 9-30-97

Error Sensor: Accelerometer

Simulated sound field: 70Hz; 30dB below the SPL at flight condition
 Regular MIMO filtered-x LMS

Number of input channels: 4

(1) Accel. A16+C16+E16

(2) Accel. A23+C23+E23

(3) Accel. A34+C34+E34

(4) Accel. A40+C40+E40

Number of output channels: 2

(1) Piezo group A,C,E 14/15-28

(2) Piezo group A,C,E 28-36

Control Settings:

NA=15

NB=14

NC=15

ND=14

Random Signal for Identification: 2V random 50-80 Hz

Convergence coeff. for control=0.001

Convergence coeff. for identif.=0.001

Sampling Rate=2000 Hz

DSP Gain=10

Acceleration and Noise Attenuations:

Accel. (mm/s/s)					Mic Voltage (mV)*				
Ch#	Location	Open Loop	Closed Loop	Attenu. (dB)	Mic#	Location**	Open Loop	Closed Loop	Attenu. (dB)
1	E16	106	93	1.1	1	Row1, W.	158	151	0.4
2	E23	160	63	8.1	2	Row2, W.	117	109	0.6
3	E34	115	98	1.4	3	Row3, W.	115	77	3.5
4	E40	93	24	11.8	4	Row1, A.	99	109	-0.8
5	C16	82	53	3.8	5	Row2, A.	69	72	-0.4
6	C23	165	29	15.1	6	Row3, A.	49	30	4.3
7	C34	115	109	0.5	7	Row1, S.	64	129	-6.1
8	C40	150	35	12.6	8	Row2, S.	16	75	-13.4
9	A16	63	47	2.5	9	Row3, S.	40	67	-4.5
10	A23	251	108	7.3					
11	A34	112	136	-1.7					
12	A40	170	8	26.5					

* 1 mV represents 83 uPa

** W. and A. refer to window and aisle seats. S. refers to standing aisle

Control Case #3

Test Date: 10-22-97

Error Sensor: Accelerometer

Simulated sound field: 61Hz; 30dB below the SPL at flight condition
Regular MIMO filtered-x LMS

Number of input channels: 4

Number of output channels: 2

(1) Accel. A24+C24+E24

(1) Piezo group A,C,E 16-28

(2) Accel. A23+C23+E23

(2) Piezo group A,C,E 30-36

(3) Accel. A34+C34+E34

(4) Accel. A40+C40+E40

Control Settings:

NA=15

Convergence coeff. for control=0.001

NB=14

Convergence coeff. for identif.=0.001

NC=15

Sampling Rate=2000 Hz

ND=14

DSP Gain=10

Random Signal for Identification: 2V random 50-80 Hz

Acceleration and Noise Attenuations:

Accel. (mm/s/s)					Mic Voltage (mV)*				
Ch#	Location	Open Loop	Closed Loop	Attenu. (dB)	Mic#	Location**	Open Loop	Closed Loop	Attenu. (dB)
1	E24	44	25	4.9	1	Row1, W.	28	50	-5.0
2	E23	124	26	13.6	2	Row2, W.	39	76	-5.8
3	E34	152	47	10.2	3	Row3, W.	50	101	-6.1
4	E40	134	18	17.4	4	Row1, A.	44	46	-0.4
5	C24	65	33	5.9	5	Row2, A.	54	66	-1.7
6	C23	136	16	18.6	6	Row3, A.	68	91	-2.5
7	C34	179	49	11.3	7	Row1, S.	80	77	0.3
8	C40	150	21	17.1	8	Row2, S.	101	95	0.5
9	A24	54	47	1.2	9	Row3, S.	105	117	-0.9
10	A23	104	22	13.5					
11	A34	149	54	8.8					
12	A40	132	27	13.8					

* 1 mV represents 83 uPa

** W. and A. refer to window and aisle seats. S. refers to standing aisle

Control Case #4

Test Date: 10-22-97

Error Sensor: Accelerometer

Simulated sound field: 70Hz; 30dB below the SPL at flight condition
Regular MIMO filtered-x LMS

Number of input channels: 4

(1) Accel. A24+C24+E24

(2) Accel. A23+C23+E23

(3) Accel. A34+C34+E34

(4) Accel. A40+C40+E40

Number of output channels: 2

(1) Piezo group A,C,E 16-28

(2) Piezo group A,C,E 30-36

Control Settings:

NA=15

NB=14

NC=15

ND=14

Random Signal for Identification: 2V random 50-80 Hz

Convergence coeff. for control=0.001

Convergence coeff. for identif.=0.001

Sampling Rate=2000 Hz

DSP Gain=10

Acceleration and Noise Attenuations:

Accel. (mm/s/s)					Mic Voltage (mV)*				
Ch#	Location	Open Loop	Closed Loop	Attenu. (dB)	Mic#	Location**	Open Loop	Closed Loop	Attenu. (dB)
1	E24	200	29	16.8	1	Row1, W.	113	89	2.1
2	E23	160	43	11.4	2	Row2, W.	134	111	1.6
3	E34	81	47	4.7	3	Row3, W.	155	115	2.6
4	E40	56	26	6.7	4	Row1, A.	83	86	-0.3
5	C24	218	36	15.6	5	Row2, A.	59	63	-0.6
6	C23	157	25	16.0	6	Row3, A.	80	39	6.2
7	C34	75	49	3.7	7	Row1, S.	22	107	-13.7
8	C40	131	40	10.3	8	Row2, S.	14	82	-15.4
9	A24	256	61	12.5	9	Row3, S.	49	98	-6.0
10	A23	186	46	12.1					
11	A34	86	80	0.6					
12	A40	148	45	10.3					

* 1 mV represents 83 uPa

** W. and A. refer to window and aisle seats. S. refers to standing aisle

Control Case #5

Test Date: 10-24-97

Error Sensor: Accelerometer

Simulated sound field: 61Hz; 30dB below the SPL at flight condition
Simplified MIMO filtered-x LMS

Number of input channels: 12

(1) Accel. E24

(2) Accel. E23

...

...

(12) Accel. A40

Number of output channels: 3

(1) Piezo group A,C,E 12-18

(2) Piezo group A,C,E 18-28

(3) Piezo group A,C,E 28-36

Control Settings:

NA=15

NB=14

NC=15

ND=14

Random Signal for Identification: 2V random 55-75 Hz

Convergence coeff. for control=0.001

Convergence coeff. for identif.=0.001

Sampling Rate=1500 Hz

DSP Gain=10

Acceleration and Noise Attenuations:

Accel. (mm/s/s)					Mic Voltage (mV)*				
Ch#	Location	Open Loop	Closed Loop	Attenu. (dB)	Mic#	Location**	Open Loop	Closed Loop	Attenu. (dB)
1	E24	54	13	12.4	1	Row1, W.	21	47	-7.0
2	E23	132	24	14.8	2	Row2, W.	31	72	-7.3
3	E34	160	17	19.5	3	Row3, W.	36	88	-7.8
4	E40	152	24	16.0	4	Row1, A.	40	41	-0.2
5	C24	75	37	6.1	5	Row2, A.	48	51	-0.5
6	C23	143	62	7.3	6	Row3, A.	57	63	-0.9
7	C34	188	46	12.2	7	Row1, S.	72	70	0.2
8	C40	160	45	11.0	8	Row2, S.	91	80	1.1
9	A24	65	23	9.0	9	Row3, S.	89	75	1.5
10	A23	107	24	13.0					
11	A34	156	43	11.2					
12	A40	140	34	12.3					

* 1 mV represents 83 uPa

** W. and A. refer to window and aisle seats. S. refers to standing aisle

Control Case #6

Test Date: 10-24-97

Error Sensor: Accelerometer

Simulated sound field: 70Hz; 30dB below the SPL at flight condition
Simplified MIMO filtered-x LMS

Number of input channels: 12

(1) Accel. E24

(2) Accel. E23

(12) Aceel. A40

Number of output channels: 3

(1) Piezo group A,C,E 12-18

(2) Piezo group A,C,E 18-28

(3) Piezo group A,C,E 28-36

Control Settings:

NA=15

NB=14

NC=15

ND=14

Random Signal for Identification: 2V random 55-75 Hz

Convergence coeff. for control=0.001

Convergence coeff. for identif.=0.001

Sampling Rate=1500 Hz

DSP Gain=10

Acceleration and Noise Attenuations:

Accel. (mm/s/s)					Mic Voltage (mV)*				
Ch#	Location	Open Loop	Closed Loop	Attenu. (dB)	Mic#	Location**	Open Loop	Closed Loop	Attenu. (dB)
1	E24	242	34	17.0	1	Row1, W.	144	72	6.0
2	E23	198	46	12.7	2	Row2, W.	162	57	9.1
3	E34	86	33	8.3	3	Row3, W.	182	65	8.9
4	E40	137	117	1.4	4	Row1, A.	114	29	11.9
5	C24	256	46	14.9	5	Row2, A.	69	27	8.1
6	C23	192	16	21.6	6	Row3, A.	95	4	27.5
7	C34	78	76	0.2	7	Row1, S.	53	45	1.4
8	C40	135	113	1.5	8	Row2, S.	18	24	-2.5
9	A24	299	73	12.2	9	Row3, S.	66	13	14.1
10	A23	192	40	13.6					
11	A34	99	94	0.5					
12	A40	159	105	3.6					

* 1 mV represents 83 uPa

** W. and A. refer to window and aisle seats. S. refers to standing aisle

Control Case #7

Test Date: 11-04-97

Error Sensor: Accelerometer

Simulated sound field: 61Hz; 30dB below the SPL at flight condition
Regular MIMO filtered-x LMS

Input CH#: 4

- (1) Accel. A16+C16+E16
- (2) Accel. A23+C23+E23
- (3) Accel. A34+C34+E34
- (4) Accel. A40+C40+E40

Output CH#: 2

- (1) Piezo group A,C,E 14/15-28
- (2) Piezo group A,C,E 28-36

Note: [C0] & [D0] from curve fit of transfer function

Control Settings:

NA=15

NB=0

NC=5

ND=4

Random Signal for Identification: none

Convergence coeff. for control=0.001

Convergence coeff. for identif.=0.0

Sampling Rate=2000 Hz

DSP Gain=10

Acceleration and Noise Attenuations:

Accel. (mm/s/s)					Mic Voltage (mV)*				
Ch#	Location	Open Loop	Closed Loop	Attenu. (dB)	Mic#	Location*	Open Loop	Closed Loop	Attenu. (dB)
1	E16	155	86	5.1	1	Row1, W	31	43	2.8
2	E23	128	35	11.3	2	Row2, W	45	55	1.7
3	E34	152	35	12.8	3	Row3, W	54	69	2.1
4	E40	131	11	21.5	4	Row1, A.	46	20	-7.2
5	C16	154	59	8.3	5	Row2, A.	54	34	-4.0
6	C23	140	42	10.5	6	Row3, A.	63	51	-1.8
7	C34	179	51	10.9	7	Row1, S.	86	39	-6.9
8	C40	147	19	17.8	8	Row2, S.	102	39	-8.4
9	A16	101	50	6.1	9	Row3, S.	100	51	-5.8
10	A23	113	26	12.8					
11	A34	148	56	8.4					
12	A40	130	3	32.7					

* 1 mV represents 83 uPa

** W. and A. refer to window and aisle seats. S. refers to standing aisle

Control Case #8

Test Date: 11-04-97

Error Sensor: Accelerometer

Simulated sound field: 70Hz; 30dB below the SPL at flight condition
Regular MIMO filtered-x LMS

Input CH#: 4

- (1) Accel. A16+C16+E16
- (2) Accel. A23+C23+E23
- (3) Accel. A34+C34+E34
- (4) Accel. A40+C40+E40

Output CH#: 2

- (1) Piezo group A,C,E 14/15-28
- (2) Piezo group A,C,E 28-36

Note: [C0] & [D0] from curve fit of transfer function

Control Settings:

NA=15

NB=0

NC=5

ND=4

Random Signal for Identification: none

Convergence coeff. for control=0.001

Convergence coeff. for identif.=0.0

Sampling Rate=2000 Hz

DSP Gain=10

Acceleration and Noise Attenuations:

Accel. (mm/s/s)					Mic Voltage (mV)*				
Ch#	Location	Open Loop	Closed Loop	Attenu. (dB)	Mic#	Location*	Open Loop	Closed Loop	Attenu. (dB)
1	E16	106	109	-0.2	1	Row1, W	130	129	-0.1
2	E23	178	60	9.4	2	Row2, W	144	137	-0.4
3	E34	82	70	1.4	3	Row3, W	174	135	-2.2
4	E40	105	22	13.6	4	Row1, A.	86	112	2.3
5	C16	88	50	4.9	5	Row2, A.	59	67	1.1
6	C23	172	21	18.3	6	Row3, A.	77	63	-1.7
7	C34	74	82	-0.9	7	Row1, S.	25	100	12.0
8	C40	134	22	15.7	8	Row2, S.	16	52	10.2
9	A16	58	36	4.1	9	Row3, S.	45	68	3.6
10	A23	186	26	17.1					
11	A34	81	124	-3.7					
12	A40	154	5	29.8					

* 1 mV represents 83 uPa

** W. and A. refer to window and aisle seats. S. refers to standing aisle

Control Case #9

Test Date: 11-04-97

Error Sensor: Accelerometer

Simulated sound field: 121Hz; 20dB below the SPL at flight condition
Simplified MIMO filtered-x LMS

Input CH#: 4

- (1) Accel. A16+C16+E16
- (2) Accel. A23+C23+E23
- (3) Accel. A34+C34+E34
- (4) Accel. A40+C40+E40

Output CH#: 2

- (1) Piezo group A,C,E 14/15-28
- (2) Piezo group A,C,E 28-36

Control Settings:

NA=15

NB=0

NC=15

ND=14

Random Signal for Identification: 2V random 120-122 Hz

Convergence coeff. for control=0.0001

Convergence coeff. for identif.=0.001

Sampling Rate=2000 Hz

DSP Gain=10

Acceleration and Noise Attenuations:

Accel. (mm/s/s)					Mic Voltage (mV)*				
Ch#	Location	Open Loop	Closed Loop	Attenu. (dB)	Mic#	Location*	Open Loop	Closed Loop	Attenu. (dB)
1	E16	358	214	4.5	1	Row1, W	148	83	-5.0
2	E23	260	19	22.7	2	Row2, W	99	22	-13.1
3	E34	115	137	-1.5	3	Row3, W	136	60	-7.1
4	E40	79	25	10.0	4	Row1, A.	82	51	-4.1
5	C16	250	50	14.0	5	Row2, A.	68	72	0.5
6	C23	415	92	13.1	6	Row3, A.	54	41	-2.4
7	C34	200	38	14.4	7	Row1, S.	165	212	2.2
8	C40	237	88	8.6	8	Row2, S.	117	137	1.4
9	A16	216	47	13.2	9	Row3, S.	127	123	-0.3
10	A23	405	133	9.7					
11	A34	208	125	4.4					
12	A40	205	125	4.3					

* 1 mV represents 83 uPa

** W. and A. refer to window and aisle seats. S. refers to standing aisle

Control Case #10

Test Date: 11-04-97

Error Sensor: Accelerometer

Simulated sound field: 140Hz; 30dB below the SPL at flight condition
Simplified MIMO filtered-x LMS

Input CH#: 4

- (1) Accel. A16+C16+E16
- (2) Accel. A23+C23+E23
- (3) Accel. A34+C34+E34
- (4) Accel. A40+C40+E40

Output CH#: 2

- (1) Piezo group A,C,E 14/15-28
- (2) Piezo group A,C,E 28-36

Control Settings:

NA=15

NB=0

NC=15

ND=14

Random Signal for Identification: 2V random 139-141 Hz

Convergence coeff. for control=0.0001

Convergence coeff. for identif.=0.001

Sampling Rate=2000 Hz

DSP Gain=10

Acceleration and Noise Attenuations:

Accel. (mm/s/s)					Mic Voltage (mV)*				
Ch#	Location	Open Loop	Closed Loop	Attenu. (dB)	Mic#	Location*	Open Loop	Closed Loop	Attenu. (dB)
1	E16	63	72	-1.2	1	Row1, W	121	109	-0.9
2	E23	146	107	2.7	2	Row2, W	153	130	-1.4
3	E34	54	80	-3.4	3	Row3, W	128	138	0.7
4	E40	129	150	-1.3	4	Row1, A.	48	40	-1.6
5	C16	78	55	3.0	5	Row2, A.	41	39	-0.4
6	C23	127	71	5.1	6	Row3, A.	45	50	0.9
7	C34	48	36	2.5	7	Row1, S.	62	57	-0.7
8	C40	152	131	1.3	8	Row2, S.	68	52	-2.3
9	A16	44	100	-7.1	9	Row3, S.	31	39	2.0
10	A23	67	94	-2.9					
11	A34	61	14	12.8					
12	A40	84	48	4.9					

* 1 mV represents 83 uPa

** W. and A. refer to window and aisle seats. S. refers to standing aisle

Control Case #11

Test Date: 11-04-97

Error Sensor: Accelerometer

Simulated sound field: 61Hz; 30dB below the SPL at flight condition
Simplified MIMO filtered-x LMS

Input CH#: 12

(1) Accel. E16

(2) Accel. E23

...

...

(12) Accel. A40

Output CH#: 2

(1) Piezo group A,C,E 14/15-28

(2) Piezo group A,C,E 28-36

Control Settings:

NA=15

NB=0

NC=15

ND=14

Random Signal for Identification: 2V random 55-75 Hz

Convergence coeff. for control=0.001

Convergence coeff. for identif.=0.001

Sampling Rate=2000 Hz

DSP Gain=10

Acceleration and Noise Attenuations:

Accel. (mm/s/s)					Mic Voltage (mV)*				
Ch#	Location	Open Loop	Closed Loop	Attenu. (dB)	Mic#	Location*	Open Loop	Closed Loop	Attenu. (dB)
1	E16	149	89	4.5	1	Row1, W	33	46	2.9
2	E23	126	38	10.4	2	Row2, W	41	59	3.2
3	E34	151	39	11.8	3	Row3, W	48	73	3.6
4	E40	131	18	17.2	4	Row1, A.	46	19	-7.7
5	C16	149	63	7.5	5	Row2, A.	51	32	-4.0
6	C23	139	41	10.6	6	Row3, A.	61	51	-1.6
7	C34	178	56	10.0	7	Row1, S.	82	34	-7.6
8	C40	149	23	16.2	8	Row2, S.	95	32	-9.5
9	A16	98	52	5.5	9	Row3, S.	90	45	-6.0
10	A23	115	26	12.9					
11	A34	147	61	7.6					
12	A40	132	9	23.3					

* 1 mV represents 83 uPa

** W. and A. refer to window and aisle seats. S. refers to standing aisle

Control Case #12

Test Date: 11-04-97

Error Sensor: Accelerometer

Simulated sound field: 61Hz; 30dB below the SPL at flight condition
Simplified MIMO filtered-x LMS

Input CH#: 12

(1) Accel. E16

(2) Accel. E23

...

...

(12) Aceel. A40

Output CH#: 2

(1) Piezo group A,C,E 14/15-28

(2) Piezo group A,C,E 28-36

Control Settings:

NA=15

NB=0

NC=15

ND=14

Random Signal for Identification: 3V random 60-62 Hz

Convergence coeff. for control=0.001

Convergence coeff. for identif.=0.001

Sampling Rate=2000 Hz

DSP Gain=10

Acceleration and Noise Attenuations:

Accel. (mm/s/s)					Mic Voltage (mV)*				
Ch#	Location	Open Loop	Closed Loop	Attenu. (dB)	Mic#	Location*	Open Loop	Closed Loop	Attenu. (dB)
1	E16	149	82	5.2	1	Row1, W	33	45	2.7
2	E23	126	34	11.4	2	Row2, W	41	57	2.9
3	E34	151	38	12.0	3	Row3, W	48	70	3.3
4	E40	131	19	16.8	4	Row1, A.	46	20	-7.2
5	C16	149	57	8.3	5	Row2, A.	51	33	-3.8
6	C23	139	41	10.6	6	Row3, A.	51	51	-1.6
7	C34	178	54	10.4	7	Row1, S.	82	34	-7.6
8	C40	149	25	15.5	8	Row2, S.	95	33	-9.2
9	A16	98	47	6.4	9	Row3, S.	90	45	-6.0
10	A23	115	24	13.6					
11	A34	147	58	8.1					
12	A40	132	10	22.4					

* 1 mV represents 83 uPa

** W. and A. refer to window and aisle seats. S. refers to standing aisle

Control Case #13

Test Date: 11-04-97

Error Sensor: Accelerometer

Simulated sound field: 70Hz; 30dB below the SPL at flight condition
Simplified MIMO filtered-x LMS

Input CH#: 12

(1) Accel. E16

(2) Accel. E23

...

...

(12) Accel. A40

Output CH#: 2

(1) Piezo group A,C,E 14/15-28

(2) Piezo group A,C,E 28-36

Control Settings:

NA=15

NB=0

NC=15

ND=14

Random Signal for Identification: 3V random 69-71 Hz

Convergence coeff. for control=0.001

Convergence coeff. for identif.=0.001

Sampling Rate=2000 Hz

DSP Gain=10

Acceleration and Noise Attenuations:

Accel. (mm/s/s)					Mic Voltage (mV)*				
Ch#	Location	Open Loop	Closed Loop	Attenu. (dB)	Mic#	Location*	Open Loop	Closed Loop	Attenu. (dB)
1	E16	108	106	0.2	1	Row1, W	118	111	-0.5
2	E23	196	63	9.9	2	Row2, W	128	120	-0.6
3	E34	82	80	0.2	3	Row3, W	148	115	-2.2
4	E40	85	20	12.6	4	Row1, A.	85	109	2.2
5	C16	96	52	5.3	5	Row2, A.	58	65	1.0
6	C23	194	31	15.9	6	Row3, A.	66	53	-1.9
7	C34	72	93	-2.2	7	Row1, S.	28	108	11.7
8	C40	128	21	15.7	8	Row2, S.	22	70	10.1
9	A16	64	39	4.3	9	Row3, S.	61	91	3.5
10	A23	221	39	15.1					
11	A34	96	139	-3.2					
12	A40	155	1	43.8					

* 1 mV represents 83 μ Pa

** W. and A. refer to window and aisle seats. S. refers to standing aisle

Control Case #14

Test Date: 11-06-97

Error Sensor: Accelerometer

Simulated sound field: 61Hz, 30dB below the SPL at flight condition
Simplified MIMO filtered-x LMS

Number of input channels: 12

(1) Accel. E15

(2) Accel. E16

(12) Accel. A21

Number of output channels: 3

(1) Piezo group A 14/15-28

(2) Piezo group C 14/15-28

(3) Piezo group E 14/15-28

Control Settings:

NA=15

NB=0

NC=15

ND=14

Random Signal for Identification: 3V random 55-75 Hz

Convergence coeff. for control=0.001

Convergence coeff. for identif.=0.001

Sampling Rate=2000 Hz

DSP Gain=10

Acceleration and Noise Attenuations:

Accel. (mm/s/s)					Mic Voltage (mV)*				
Ch#	Location	Open Loop	Closed Loop	Attenu. (dB)	Mic#	Location**	Open Loop	Closed Loop	Attenu. (dB)
1	E15	142	47	9.6	1	Row1, W.	21	37	-4.9
2	E16	156	26	15.6	2	Row2, W.	32	47	-3.3
3	E19	180	26	16.8	3	Row3, W.	39	54	-2.8
4	E21	118	26	13.1	4	Row1, A.	43	28	3.7
5	C15	91	13	16.9	5	Row2, A.	49	40	1.8
6	C16	159	12	22.4	6	Row3, A.	60	51	1.4
7	C19	158	29	14.7	7	Row1, S.	80	47	4.6
8	C21	177	16	20.9	8	Row2, S.	97	60	4.2
9	A15	66	19	10.8	9	Row3, S.	93	66	3.0
10	A16	105	9	21.3					
11	A19	93	9	20.3					
12	A21	120	16	17.5					

* 1 mV represents 83 μ Pa

** W. and A. refer to window and aisle seats. S. refers to standing aisle

Control Case #15

Test Date: 11-06-97

Error Sensor: Accelerometer

Simulated sound field: 61Hz, 30dB below the SPL at flight condition
Simplified MIMO filtered-x LMS

Number of input channels: 3

(1) Accel. E15+E16+E19+E21

(2) Accel. C15+C16+C19+C21

(1) Accel. A15+A16+A19+A21

Number of output channels: 3

(1) Piezo group A 14/15-28

(2) Piezo group C 14/15-28

(3) Piezo group E 14/15-28

Control Settings:

NA=15

NB=0

NC=15

ND=14

Random Signal for Identification: 3V random 55-75 Hz

Convergence coeff. for control=0.001

Convergence coeff. for identif.=0.001

Sampling Rate=2000 Hz

DSP Gain=10

Acceleration and Noise Attenuations:

Accel. (mm/s/s)					Mic Voltage (mV)*				
Ch#	Location	Open Loop	Closed Loop	Attenu. (dB)	Mic#	Location**	Open Loop	Closed Loop	Attenu. (dB)
1	E15	138	57	7.7	1	Row1. W.	20	34	-4.6
2	E16	149	26	15.2	2	Row2. W.	30	43	-3.1
3	E19	169	29	15.3	3	Row3. W.	36	47	-2.3
4	E21	111	9	21.8	4	Row1. A.	42	29	3.2
5	C15	87	8	20.7	5	Row2. A.	47	40	1.4
6	C16	151	13	21.3	6	Row3. A.	56	53	0.5
7	C19	150	13	21.2	7	Row1. S.	77	56	2.8
8	C21	165	17	19.7	8	Row2. S.	94	64	3.3
9	A15	63	26	7.7	9	Row3. S.	90	71	2.1
10	A16	100	8	21.9					
11	A19	86	7	21.8					
12	A21	110	15	17.3					

* 1 mV represents 83 uPa

** W. and A. refer to window and aisle seats. S. refers to standing aisle

Control Case #16

Test Date: 11-07-97

Error Sensor: Accelerometer

Simulated sound field: 61Hz, 30dB below the SPL at flight condition

Simplified MIMO filtered-x LMS

Number of input channels: 12

(1) Accel. E15

(2) Accel. E16

...

...

(12) Accel. A21

Number of output channels: 3

(1) Piezo group A 12-28

(2) Piezo group C 12-28

(3) Piezo group E 12-28

Control Settings:

NA=15

NB=0

NC=15

ND=14

Random Signal for Identification: 3V random 55-75 Hz

Convergence coeff. for control=0.001

Convergence coeff. for identif.=0.001

Sampling Rate=2000 Hz

DSP Gain=10

Acceleration and Noise Attenuations:

Accel. (mm/s/s)					Mic Voltage (mV)*				
Ch#	Location	Open Loop	Closed Loop	Attenu. (dB)	Mic#	Location**	Open Loop	Closed Loop	Attenu. (dB)
1	E15	142	17	18.4	1	Row1, W.	18	31	-4.7
2	E16	151	27	15.0	2	Row2, W.	32	41	-2.2
3	E19	169	25	16.6	3	Row3, W.	42	48	-1.2
4	E21	93	38	7.8	4	Row1, A.	42	17	7.9
5	C15	90	20	13.1	5	Row2, A.	48	28	4.7
6	C16	154	28	14.8	6	Row3, A.	59	41	3.2
7	C19	151	14	20.7	7	Row1, S.	81	35	7.3
8	C21	161	13	21.9	8	Row2, S.	98	39	8.0
9	A15	66	20	10.4	9	Row3, S.	96	48	6.0
10	A16	99	2	33.9					
11	A19	85	9	19.5					
12	A21	107	22	13.7					

* 1 mV represents 83 uPa

** W. and A. refer to window and aisle seats. S. refers to standing aisle

Control Case #17

Test Date: 11-07-97

Error Sensor: Accelerometer

Simulated sound field: 61Hz, 30dB below the SPL at flight condition
Simplified MIMO filtered-x LMS

Number of input channels: 3
(1) Accel. E15+E16+E19+E21
(2) Accel. C15+C16+C19+C21
(1) Accel. A15+A16+A19+A21

Number of output channels: 3
(1) Piezo group A 12-28
(2) Piezo group C 12-28
(3) Piezo group E 12-28

Control Settings:

NA=15

NB=0

NC=15

ND=14

Random Signal for Identification: 3V random 55-75 Hz

Convergence coeff. for control=0.001

Convergence coeff. for identif.=0.001

Sampling Rate=2000 Hz

DSP Gain=10

Acceleration and Noise Attenuations:

Accel. (mm/s/s)					Mic Voltage (mV)*				
Ch#	Location	Open Loop	Closed Loop	Attenu. (dB)	Mic#	Location**	Open Loop	Closed Loop	Attenu. (dB)
1	E15	139	34	12.2	1	Row1, W.	26	27	-0.3
2	E16	149	33	13.1	2	Row2, W.	30	31	-0.3
3	E19	168	18	19.4	3	Row3, W.	35	37	-0.5
4	E21	109	46	7.5	4	Row1, A.	48	7	16.7
5	C15	87	58	3.5	5	Row2, A.	53	15	11.0
6	C16	150	27	14.9	6	Row3, A.	64	32	6.0
7	C19	148	40	11.4	7	Row1, S.	87	34	8.2
8	C21	163	52	9.9	8	Row2, S.	105	31	10.6
9	A15	64	25	8.2	9	Row3, S.	99	35	9.0
10	A16	100	6	24.4					
11	A19	85	7	21.7					
12	A21	108	22	13.8					

* 1 mV represents 83 μ Pa

** W. and A. refer to window and aisle seats. S. refers to standing aisle

Control Case #18

Test Date: 11-13-97

Error Sensor: Accelerometer

Simulated sound field: 70Hz, 30dB below the SPL at flight condition
Regular MIMO filtered-x LMS

Number of input channels: 3
(1) Accel. E15+E16+E19+E21
(2) Accel. C15+C16+C19+C21
(1) Accel. A15+A16+A19+A21

Number of output channels: 3
(1) Piezo group A 12-28
(2) Piezo group C 12-28
(3) Piezo group E 12-28

Control Settings:

NA=15

NB=0

NC=15

ND=14

Random Signal for Identification: 3V random 55-75 Hz

Convergence coeff. for control=0.001

Convergence coeff. for identif.=0.001/0.005

Sampling Rate=1800 Hz

DSP Gain=10

Acceleration and Noise Attenuations:

Accel. (mm/s/s)					Mic Voltage (mV)*				
Ch#	Location	Open Loop	Closed Loop	Attenu. (dB)	Mic#	Location**	Open Loop	Closed Loop	Attenu. (dB)
1	E15	116	106	0.8	1	Row1, W.	122	119	0.2
2	E16	97	28	10.8	2	Row2, W.	138	133	0.3
3	E19	179	125	3.1	3	Row3, W.	172	149	1.2
4	E21	233	202	1.2	4	Row1, A.	53	97	-5.2
5	C15	78	62	2.0	5	Row2, A.	39	59	-3.6
6	C16	75	19	11.9	6	Row3, A.	46	75	-4.2
7	C19	128	56	7.2	7	Row1, S.	9	47	-14.4
8	C21	256	200	2.1	8	Row2, S.	18	40	-6.9
9	A15	73	80	-0.8	9	Row3, S.	41	87	-6.5
10	A16	62	5	21.9					
11	A19	125	60	6.4					
12	A21	263	149	4.9					

* 1 mV represents 83 uPa

** W. and A. refer to window and aisle seats. S. refers to standing aisle

Control Case #19

Test Date: 11-13-97

Error Sensor: Accelerometer

Simulated sound field: 70Hz, 30dB below the SPL at flight condition
Simplified MIMO filtered-x LMS

Number of input channels: 12

(1) Accel. E15

(2) Accel. E16

...

...

(12) Accel. A21

Number of output channels: 3

(1) Piezo group A 12-28

(2) Piezo group C 12-28

(3) Piezo group E 12-28

Control Settings:

NA=15

NB=0

NC=15

ND=14

Random Signal for Identification: 3V random 55-75 Hz

Convergence coeff. for control=0.001

Convergence coeff. for identif.=0.001

Sampling Rate=1800 Hz

DSP Gain=10

Acceleration and Noise Attenuations:

Accel. (mm/s/s)					Mic Voltage (mV)*				
Ch#	Location	Open Loop	Closed Loop	Attenu. (dB)	Mic#	Location**	Open Loop	Closed Loop	Attenu. (dB)
1	E15	116	45	8.2	1	Row1, W.	122	93	2.4
2	E16	97	75	2.2	2	Row2, W.	138	94	3.3
3	E19	179	127	3.0	3	Row3, W.	172	98	4.9
4	E21	233	143	4.2	4	Row1, A.	53	79	-3.5
5	C15	78	13	15.6	5	Row2, A.	39	47	-1.6
6	C16	75	73	0.2	6	Row3, A.	46	48	-0.4
7	C19	128	65	5.9	7	Row1, S.	9	49	-14.7
8	C21	256	109	7.4	8	Row2, S.	18	34	-5.5
9	A15	73	47	3.8	9	Row3, S.	41	66	-4.1
10	A16	62	39	4.0					
11	A19	125	52	7.6					
12	A21	263	106	7.9					

* 1 mV represents 83 uPa

** W. and A. refer to window and aisle seats. S. refers to standing aisle

Control Case #20

Test Date: 11-13-97

Error Sensor: Accelerometer

Simulated sound field: 121Hz, 20dB below the SPL at flight condition
Simplified MIMO filtered-x LMS

Number of input channels: 12

(1) Accel. E15

(2) Accel. E16

...

...

(12) Accel. A21

Number of output channels: 3

(1) Piezo group A 12-28

(2) Piezo group C 12-28

(3) Piezo group E 12-28

Control Settings:

NA=15

NB=0

NC=15

ND=14

Random Signal for Identification: 3V random 120-122 Hz

Convergence coeff. for control=0.001

Convergence coeff. for identif.=0.001

Sampling Rate=1800 Hz

DSP Gain=10

Acceleration and Noise Attenuations:

Accel. (mm/s/s)					Mic Voltage (mV)*				
Ch#	Location	Open Loop	Closed Loop	Attenu. (dB)	Mic#	Location**	Open Loop	Closed Loop	Attenu. (dB)
1	E15	303	262	1.3	1	Row1, W.	148	192	-2.3
2	E16	356	90	11.9	2	Row2, W.	110	115	-0.4
3	E19	367	63	15.3	3	Row3, W.	152	206	-2.6
4	E21	249	70	11.0	4	Row1, A.	81	105	-2.3
5	C15	114	119	-0.4	5	Row2, A.	65	75	-1.2
6	C16	273	127	6.6	6	Row3, A.	66	98	-3.4
7	C19	385	142	8.7	7	Row1, S.	148	199	-2.6
8	C21	568	217	8.4	8	Row2, S.	101	89	1.1
9	A15	91	97	-0.6	9	Row3, S.	121	148	-1.7
10	A16	249	122	6.2					
11	A19	286	44	16.3					
12	A21	469	80	15.4					

* 1 mV represents 83 uPa

** W. and A. refer to window and aisle seats. S. refers to standing aisle

Control Case #21

Test Date: 11-13-97

Error Sensor: Accelerometer

Simulated sound field: 140Hz, 30dB below the SPL at flight condition
Simplified MIMO filtered-x LMS

Number of input channels: 12

(1) Accel. E15

(2) Accel. E16

(12) Accel. A21

Number of output channels: 3

(1) Piezo group A 12-28

(2) Piezo group C 12-28

(3) Piezo group E 12-28

Control Settings:

NA=15

NB=0

NC=15

ND=14

Random Signal for Identification: 3V random 139-141 Hz

Convergence coeff. for control=0.001

Convergence coeff. for identif.=0.001/0.005

Sampling Rate=1800 Hz

DSP Gain=10

Acceleration and Noise Attenuations:

Accel. (mm/s/s)					Mic Voltage (mV)*				
Ch#	Location	Open Loop	Closed Loop	Attenu. (dB)	Mic#	Location**	Open Loop	Closed Loop	Attenu. (dB)
1	E15	68	94	-2.8	1	Row1, W.	85	83	0.2
2	E16	57	62	-0.7	2	Row2, W.	114	95	1.6
3	E19	146	120	1.7	3	Row3, W.	99	77	2.2
4	E21	157	123	2.1	4	Row1, A.	40	40	0.0
5	C15	83	35	7.5	5	Row2, A.	37	32	1.3
6	C16	56	36	3.8	6	Row3, A.	42	40	0.4
7	C19	84	81	0.3	7	Row1, S.	46	50	-0.7
8	C21	117	73	4.1	8	Row2, S.	55	56	-0.2
9	A15	76	64	1.5	9	Row3, S.	28	31	-0.9
10	A16	71	35	6.1					
11	A19	56	10	15.0					
12	A21	70	27	8.3					

* 1 mV represents 83 uPa

** W. and A. refer to window and aisle seats. S. refers to standing aisle

Control Case #22

Test Date: 11-13-97

Error Sensor: Accelerometer

Simulated sound field: 61Hz; 30dB below the SPL at flight condition
Simplified MIMO filtered-x LMS

Number of input channels: 12

(1) Accel. E23

(2) Accel. E16

...

...

(12) Accel. A21

Number of output channels: 3

(1) Piezo group A 12-32

(2) Piezo group C 12-32

(3) Piezo group E 12-32

Control Settings:

NA=15

NB=0

NC=15

ND=14

Random Signal for Identification: 3V random 55-75 Hz

Convergence coeff. for control=0.001

Convergence coeff. for identif.=0.001/0.005

Sampling Rate=1800 Hz

DSP Gain=10

Acceleration and Noise Attenuations:

Accel. (mm/s/s)					Mic Voltage (mV)*				
Ch#	Location	Open Loop	Closed Loop	Attenu. (dB)	Mic#	Location**	Open Loop	Closed Loop	Attenu. (dB)
1	E23	45	54	-1.6	1	Row1, W.	20	32	-4.1
2	E16	148	58	8.1	2	Row2, W.	34	51	-3.5
3	E19	166	73	7.1	3	Row3, W.	42	70	-4.4
4	E21	78	79	-0.1	4	Row1, A.	43	17	8.1
5	C23	64	9	17.0	5	Row2, A.	50	32	3.9
6	C16	151	36	12.5	6	Row3, A.	61	53	1.2
7	C19	149	49	9.7	7	Row1, S.	84	19	12.9
8	C21	158	54	9.3	8	Row2, S.	100	26	11.7
9	A23	54	26	6.3	9	Row3, S.	98	45	6.8
10	A16	100	36	8.9					
11	A19	86	35	7.8					
12	A21	108	48	7.0					

* 1 mV represents 83 uPa

** W. and A. refer to window and aisle seats. S. refers to standing aisle

Control Case #23

Test Date: 11-13-97

Error Sensor: Accelerometer

Simulated sound field: 70Hz; 30dB below the SPL at flight condition
Simplified MIMO filtered-x LMS

Number of input channels: 12

(1) Accel. E23

(2) Accel. E16

...

...

(12) Accel. A21

Number of output channels: 3

(1) Piezo group A 12-32

(2) Piezo group C 12-32

(3) Piezo group E 12-32

Control Settings:

NA=15

NB=0

NC=15

ND=14

Random Signal for Identification: 3V random 55-75 Hz

Convergence coeff. for control=0.001

Convergence coeff. for identif.=0.001/0.005

Sampling Rate=1800 Hz

DSP Gain=10

Acceleration and Noise Attenuations:

Accel. (mm/s/s)

Mic Voltage (mV)*

Ch#	Location	Open Loop	Closed Loop	Attenu. (dB)	Mic#	Location**	Open Loop	Closed Loop	Attenu. (dB)
1	E23	220	26	18.5	1	Row1, W.	127	106	1.6
2	E16	96	53	5.2	2	Row2, W.	142	95	3.5
3	E19	179	45	12.0	3	Row3, W.	176	99	5.0
4	E21	303	51	15.5	4	Row1, A.	82	89	-0.7
5	C23	237	87	8.7	5	Row2, A.	63	55	1.2
6	C16	77	18	12.6	6	Row3, A.	69	40	4.7
7	C19	127	54	7.4	7	Row1, S.	13	69	-14.5
8	C21	258	31	18.4	8	Row2, S.	28	35	-1.9
9	A23	276	144	5.7	9	Row3, S.	64	67	-0.4
10	A16	82	13	16.0					
11	A19	125	1	41.9					
12	A21	265	21	22.0					

* 1 mV represents 83 uPa

** W. and A. refer to window and aisle seats. S. refers to standing aisle

Control Case #24

Test Date: 11-13-97

Error Sensor: Accelerometer

Simulated sound field: 121Hz; 20dB below the SPL at flight condition
Simplified MIMO filtered-x LMS

Number of input channels: 12

(1) Accel. E23

(2) Accel. E16

(12) Accel. A21

Number of output channels: 3

(1) Piezo group A 12-32

(2) Piezo group C 12-32

(3) Piezo group E 12-32

Control Settings:

NA=15

NB=0

NC=15

ND=14

Random Signal for Identification: 2V random 120-122 Hz

Convergence coeff. for control=0.001

Convergence coeff. for identif.=0.001/0.005

Sampling Rate=1800 Hz

DSP Gain=10

Acceleration and Noise Attenuations:

Accel. (mm/s/s)					Mic Voltage (mV)*				
Ch#	Location	Open Loop	Closed Loop	Attenu. (dB)	Mic#	Location**	Open Loop	Closed Loop	Attenu. (dB)
1	E23	218	149	3.3	1	Row1, W.	155	209	-2.6
2	E16	369	35	20.5	2	Row2, W.	113	120	-0.5
3	E19	376	30	22.0	3	Row3, W.	153	171	-1.0
4	E21	247	14	24.9	4	Row1, A.	90	114	-2.1
5	C23	309	100	9.8	5	Row2, A.	67	64	0.4
6	C16	283	146	5.7	6	Row3, A.	68	92	-2.6
7	C19	404	24	24.5	7	Row1, S.	157	100	3.9
8	C21	583	81	17.1	8	Row2, S.	103	55	5.4
9	A23	372	165	7.1	9	Row3, S.	121	103	1.4
10	A16	252	133	5.6					
11	A19	301	33	19.2					
12	A21	490	39	22.0					

* 1 mV represents 83 uPa

** W. and A. refer to window and aisle seats. S. refers to standing aisle

Control Case #25

Test Date: 11-13-97

Error Sensor: Accelerometer

Simulated sound field: 140Hz; 30dB below the SPL at flight condition
Simplified MIMO filtered-x LMS

Number of input channels: 12

(1) Accel. E23

(2) Accel. E16

...

...

(12) Accel. A21

Number of output channels: 3

(1) Piezo group A 12-32

(2) Piezo group C 12-32

(3) Piezo group E 12-32

Control Settings:

NA=15

NB=0

NC=15

ND=14

Random Signal for Identification: 2V random 139-141 Hz

Convergence coeff. for control=0.001

Convergence coeff. for identif.=0.001/0.005

Sampling Rate=1800 Hz

DSP Gain=10

Acceleration and Noise Attenuations:

Accel. (mm/s/s)					Mic Voltage (mV)*				
Ch#	Location	Open Loop	Closed Loop	Attenu. (dB)	Mic#	Location**	Open Loop	Closed Loop	Attenu. (dB)
1	E23	124	57	6.8	1	Row1, W.	89	67	2.5
2	E16	55	72	-2.3	2	Row2, W.	117	87	2.6
3	E19	142	70	6.1	3	Row3, W.	102	65	3.9
4	E21	157	26	15.6	4	Row1, A.	39	37	0.5
5	C23	61	55	0.9	5	Row2, A.	35	32	0.8
6	C16	55	24	7.2	6	Row3, A.	42	39	0.6
7	C19	86	27	10.1	7	Row1, S.	47	41	1.2
8	C21	123	40	9.8	8	Row2, S.	55	54	0.2
9	A23	44	65	-3.4	9	Row3, S.	31	34	-0.8
10	A16	77	63	1.7					
11	A19	59	4	23.4					
12	A21	77	48	4.1					

* 1 mV represents 83 uPa

** W. and A. refer to window and aisle seats. S. refers to standing aisle

Control Case #26

Test Date: 12-08-97

Error Sensor: Accelerometer

Simulated sound field: 61Hz; 30dB below the SPL at flight condition
 Regular MIMO filtered-x LMS

Number of input channels: 3

(1) Accel. E15+E16+E19+E21

(2) Accel. C15+C16+C19+C21

(1) Accel. A15+A16+A19+A21

Number of output channels: 3

(1) Piezo group A 12-28

(2) Piezo group C 12-28

(3) Piezo group E 12-28

Control Settings:

NA=15

NB=14

NC=15

ND=14

Random Signal for Identification: 3V random 55-75 Hz

 μ_0 (control)=0.001

Convergence coeff. for identif.=0.001

Sampling Rate=1800 Hz

DSP Gain=10

Acceleration and Noise Attenuations:

Accel. (mm/s/s)					Mic Voltage (mV)*				
Ch#	Location	Open Loop	Closed Loop	Attenu. (dB)	Mic#	Location**	Open Loop	Closed Loop	Attenu. (dB)
1	E15	105	41	8.2	1	Row1, W.	23	18	2.1
2	E16	138	30	13.3	2	Row2, W.	27	9	9.5
3	E19	132	11	21.6	3	Row3, W.	45	9	14.0
4	E21	73	44	4.4	4	Row1, A.	39	2	25.8
Avg. 1 to 4		111	2.41	33.3	5	Row2, A.	38	13	9.3
5	C15	101	56	5.1	6	Row3, A.	44	14	9.9
6	C16	140	22	16.1	7	Row1, S.	66	32	6.3
7	C19	140	29	13.7	8	Row2, S.	78	36	6.7
8	C21	116	51	7.1	9	Row3, S.	75	38	5.9
Avg. 5 to 8		122	0.98	41.9					
9	A15	66	38	4.8					
10	A16	96	13	17.4					
11	A19	101	12	18.5					
12	A21	88	24	11.3					
Avg. 9 to 12		88	0.96	39.2					

* 1 mV represents 83 μ Pa

** W., A. and S. refer to window, aisle and standing seats

Control Case #27

Test Date: 12-08-97

Error Sensor: Accelerometer

Simulated sound field: 70Hz; 30dB below the SPL at flight condition
 Regular MIMO filtered-x LMS

Number of input channels: 3	Number of output channels: 3
(1) Accel. E15+E16+E19+E21	(1) Piezo group A 12-28
(2) Accel. C15+C16+C19+C21	(2) Piezo group C 12-28
(1) Accel. A15+A16+A19+A21	(3) Piezo group E 12-28

Control Settings:

NA=15

NB=14

NC=15

ND=14

Random Signal for Identification: 3V random 55-75 Hz

 μ_0 (control)=0.001

Convergence coeff. for identif.=0.001

Sampling Rate=1800 Hz

DSP Gain=10

Acceleration and Noise Attenuations:

Accel. (mm/s/s)					Mic Voltage (mV)*				
Ch#	Location	Open Loop	Closed Loop	Attenu. (dB)	Mic#	Location**	Open Loop	Closed Loop	Attenu. (dB)
1	E15	96	101	-0.4	1	Row1, W.	141	135	0.4
2	E16	106	17	15.9	2	Row2, W.	163	172	-0.5
3	E19	159	114	2.9	3	Row3, W.	184	187	-0.1
4	E21	206	196	0.4	4	Row1, A.	111	129	-1.3
Avg. 1 to 4		117	69	4.6	5	Row2, A.	70	77	-0.8
5	C15	96	118	-1.8	6	Row3, A.	89	95	-0.6
6	C16	82	12	16.7	7	Row1, S.	38	62	-4.3
7	C19	146	88	4.4	8	Row2, S.	46	72	-3.9
8	C21	192	155	1.9	9	Row3, S.	79	107	-2.6
Avg. 5 to 8		91	40	7.1					
9	A15	76	103	-2.6					
10	A16	45	41	0.8					
11	A19	168	132	2.1					
12	A21	226	199	1.1					
Avg. 9 to 12		85	51	4.4					

* 1 mV represents 83 μ Pa

** W., A. and S. refer to window, aisle and standing seats

Control Case #28

Test Date: 12-08-97

Error Sensor: Accelerometer

Simulated sound field: 121Hz; 20dB below the SPL at flight condition
 Regular MIMO filtered-x LMS

Number of input channels: 3

(1) Accel. E15+E16+E19+E21

(2) Accel. C15+C16+C19+C21

(1) Accel. A15+A16+A19+A21

Number of output channels: 3

(1) Piezo group A 12-28

(2) Piezo group C 12-28

(3) Piezo group E 12-28

Control Settings:

NA=15

NB=14

NC=15

ND=14

Random Signal for Identification: 2V random 120-122 Hz

 μ_0 (control)=0.001

Convergence coeff. for identif.=0.001

Sampling Rate=1800 Hz

DSP Gain=10

Acceleration and Noise Attenuations:

Accel. (mm/s/s)					Mic Voltage (mV)*				
Ch#	Location	Open Loop	Closed Loop	Attenu. (dB)	Mic#	Location**	Open Loop	Closed Loop	Attenu. (dB)
1	E15	235	269	-1.2	1	Row1, W.	123	146	-1.5
2	E16	331	70	13.5	2	Row2, W.	77	46	4.5
3	E19	283	123	7.2	3	Row3, W.	109	124	-1.1
4	E21	233	217	0.6	4	Row1, A.	83	104	-2.0
Avg. 1 to 4		230	21	20.8	5	Row2, A.	55	61	-0.9
5	C15	115	185	-4.1	6	Row3, A.	46	66	-3.1
6	C16	239	41	15.3	7	Row1, S.	137	191	-2.9
7	C19	370	76	13.7	8	Row2, S.	103	114	-0.9
8	C21	379	121	9.9	9	Row3, S.	89	114	-2.2
Avg. 5 to 8		238	24	19.9					
9	A15	68	154	-7.1					
10	A16	202	75	8.6					
11	A19	334	79	12.5					
12	A21	367	146	8.0					
Avg. 9 to 12		218	21	20.3					

* 1 mV represents 83 uPa

** W., A. and S. refer to window, aisle and standing seats

Control Case #29

Test Date: 12-08-97

Error Sensor: Accelerometer

Simulated sound field: 140Hz; 30dB below the SPL at flight condition
Regular MIMO filtered-x LMS

Number of input channels: 3
(1) Accel. E15+E16+E19+E21
(2) Accel. C15+C16+C19+C21
(1) Accel. A15+A16+A19+A21

Number of output channels: 3
(1) Piezo group A 12-28
(2) Piezo group C 12-28
(3) Piezo group E 12-28

Control Settings:

NA=15

NB=14

NC=15

ND=14

Random Signal for Identification: 2V random 139-141 Hz

 μ_0 (control)=0.001

Convergence coeff. for identif.=0.001

Sampling Rate=1800 Hz

DSP Gain=10

Acceleration and Noise Attenuations:

Accel. (mm/s/s)					Mic Voltage (mV)*				
Ch#	Location	Open Loop	Closed Loop	Attenu. (dB)	Mic#	Location**	Open Loop	Closed Loop	Attenu. (dB)
1	E15	62	191	-9.8	1	Row1, W.	89	97	-0.7
2	E16	55	92	-4.5	2	Row2, W.	106	98	0.7
3	E19	123	98	2.0	3	Row3, W.	98	90	0.7
4	E21	141	65	6.7	4	Row1, A.	43	49	-1.1
Avg. 1 to 4		71	3.7	25.7	5	Row2, A.	29	27	0.6
5	C15	123	80	3.7	6	Row3, A.	39	39	0.0
6	C16	61	23	8.5	7	Row1, S.	49	54	-0.8
7	C19	113	39	9.2	8	Row2, S.	52	59	-1.1
8	C21	102	24	12.6	9	Row3, S.	27	17	4.0
Avg. 5 to 8		52	8.92	15.3					
9	A15	92	124	-2.6					
10	A16	89	22	12.1					
11	A19	89	70	2.1					
12	A21	76	69	0.8					
Avg. 9 to 12		71	2.37	29.5					

* 1 mV represents 83 uPa

** W., A. and S. refer to window, aisle and standing seats

Appendix E User Manual and Source Code for Sound Field Simulation Program

User manual:

The executable program is named WDAQ.EXE. The installation and execution procedures are as follows:

- (1) Create a subdirectory, and copy WDAQ.EXE file into this subdirectory;
- (2) Run this program in Windows 3.1 environment;
- (3) Pull down the SETTING menu and choose one of the four channels. A dialogue box will appear;
- (4) In this dialogue box, fill the characteristics for each of up to 10 waveform components, including frequency, voltage level and phase (frequency could be 1,2,....,999Hz, voltage level could be 0 to 5 volt, phase could be -999 to 999 degree);
- (5) For convenience, the setting for one channel could be copied from the setting for another channel;
- (6) Once the setting is complete, the output waveforms from the 4 channels will be displayed on the PC screen. This is to ensure the correctness of the data;
- (7) All the setting could be saved to a disk file or loaded back by running the SAVE, SAVE AS and LOAD command in FILE menu;
- (8) To start a new setting, run the NEW command in FILE menu;
- (9) Run the command START in RUN menu, the DAC board will output exactly the analog voltages with desired frequencies, levels and phases;
- (10) Click any key to terminate the voltage outputs; .
- (11) This program was carefully designed to avoid supplying a harmful impact on amplifier. That is to provide a gradual and smooth rise and decline of the output waveform at the start and end of the period.

Source Code files:

(1) WDAQ.CPP	Main file for synthesizing waveforms and outputting to DACs
(2) WDAQ.H	Head file of WDAQ.CPP
(3) WDAQ.DEF	Description file of WDAQ.CPP
(4) WDAQ.RC	Resource file of WDAQ.CPP

WDAQ.CPP

```

/////////////////////////////////////////////////////////////////
//
// WDAQ.CPP: Synthesis waveforms and output to DACs
//
// (Hardware setting refers to WDAQ.H)
//
// Written by: Weiping Xu
//             Carleton University
//             March 1997
//
/////////////////////////////////////////////////////////////////

// Specifications:
// (1) Ch0 to Ch3 use external reference 2.05 V at pin 38
// (2) Output is restricted to 2.0 V

#include <owl\applicat.h>
#include <owl\framewin.h>
#include <owl\dialog.h>
#include <owl\edit.h>
#include <owl\validate.h>
#include <owl\dc.h>
#include <owl\editfile.rh>
#include <owl\opensave.h>
#include <string.h>
#include "wdaq.rc"
#include "wdaq.h"

#include <fstream.h>
#include <math.h>
// #include <stdio.h>
#include <stdlib.h>
#include <conio.h>

ifstream fpin;
ofstream fpout;

#define RAMP_LENGTH 5 // (in second), total start ramp points = SAMP_No
*
RAMP_LENGTH
#define MAX_HZ 10 // maximum number of frequency components
#define CH_NO 4 // total output channel number
#define SAMP_NO 5000 // output buffer size
#define V_REF 2.04 // external voltage at pin 38
#define DRAW_SCALE 2 // (in volt), full scale for ex-display
#define PI 3.14159

char filename[40]="\0";
char tmpfilename[40]="\0";
UINT flag=TRUE; // TRUE indicates: wave[][] needs to be recalculated
int ch_no;
int wave[CH_NO][SAMP_NO];
void initilize(void);

// Transfer buffer for the dialog1 box.
struct TTransBuf
{
    char Edit_ChNo[2];
    char Edit_Hz[MAX_HZ][4];
    char Edit_Volt[MAX_HZ][5];

```


WDAQ.CPP

```

    char Edit_Deg[MAX_HZ][5];
} transBuf[CH_NO];

// Transfer buffer for the dialog2 box.
char copych[2];

////////////////////////////////////
// initilize the DAQ board
////////////////////////////////////
void initialize(void)
{
    CFG1_COMMAND cfg1_command;
    CFG2_COMMAND cfg2_command;
    CFG3_COMMAND cfg3_command;
    CNTRCMD_COMMAND cntrcmd_command;

//-----
//---config CFG1-----
    *((int *) &cfg1_command) = 0;
    cfg1_command.CH3_1 = 3;      ///!! Channel select (0-9) in Group 1
    cfg1_command.FIFOEN = ENABLE; ///!! FIFO memory enable
    outpw(CFG1, *((int *) &cfg1_command));

//-----
//---config CFG2-----
    *((int *) &cfg2_command) = 0;
    cfg2_command.LDAC0 = SET;      //Double-buffered DAC output
    cfg2_command.LDAC2 = SET;      //Double-buffered DAC output
    outpw(CFG2, *((int *) &cfg2_command));

//-----
//---config CFG3-----
    *((int *) &cfg3_command) = 0;
    cfg3_command.SCANEN = ENABLE;  //DAC channel scan mode enable
    cfg3_command.EN25V = SET;      //2.5V reference is available on
pin 38
    outpw(CFG3, *((int *) &cfg3_command));

//-----
//---config CNTRCMD-----
    cntrcmd_command.BCDSEL = CLEAR; //Binary coded decimal select
    cntrcmd_command.MODESEL = 3;    //Counter mode select (000--Mode 0)
    cntrcmd_command.RWSEL = 1;      //Read/Write select (01--R/W LSB
only)
    cntrcmd_command.CNTRSEL = 0;    //Counter select (00--select
Counter 1)
    outp(CNTRCMD, *((char *) &cntrcmd_command));

//-----
//---set counter1-----
    outp(CNTR1, CNRATE);

//-----
    inpw(FIFO);
}

////////////////////////////////////
// The Application's Class.      Class Definition
////////////////////////////////////
class TApp : public TApplication
{
public:
    TApp() : TApplication() {}
    void InitMainWindow();
};

```

WDAQ.CPP

```

/////////////////////////////////////////////////////////////////
// The Main Window's Class.      Class Definition
/////////////////////////////////////////////////////////////////
class TWndw : public TFrameWindow
{
public:
    TWndw(TWindow *parent, const char far *title);

protected:
    void Paint(TDC&, BOOL, TRect&);
    void ReDraw();
    void Display(TDC&);
    void CmFileNew();
    void CmFileOpen();
    void CmFileSave();
    void CmFileSaveAs();
    void CmSetCh0();
    void CmSetCh1();
    void CmSetCh2();
    void CmSetCh3();
    void CmRunStart();
    void Smooth_s(int);
    void Smooth_e(int);
    DECLARE_RESPONSE_TABLE(TWndw);
};

DEFINE_RESPONSE_TABLE1(TWndw, TFrameWindow)
    EV_COMMAND(CM_FILENEW, CmFileNew),
    EV_COMMAND(CM_FILEOPEN, CmFileOpen),
    EV_COMMAND(CM_FILESAVE, CmFileSave),
    EV_COMMAND(CM_FILESAVEAS, CmFileSaveAs),
    EV_COMMAND(CM_SET_CH0, CmSetCh0),
    EV_COMMAND(CM_SET_CH1, CmSetCh1),
    EV_COMMAND(CM_SET_CH2, CmSetCh2),
    EV_COMMAND(CM_SET_CH3, CmSetCh3),
    EV_COMMAND(CM_RUN_START, CmRunStart),
END_RESPONSE_TABLE;

/////////////////////////////////////////////////////////////////
// DIALOG_1      Class Definition
/////////////////////////////////////////////////////////////////
class TDlg1 : public TDialog
{
protected:
    TTransBuf oldTransBuf;

public:
    TDlg1(TWindow *parent, TResId resId);

protected:
    void SetupWindow();
    void CmCancel();
    void CmCopy();

    DECLARE_RESPONSE_TABLE(TDlg1);
};

DEFINE_RESPONSE_TABLE1(TDlg1, TDialog)
    EV_COMMAND(IDCANCEL, CmCancel),
    EV_COMMAND(IDC_COPY, CmCopy),

```

WDAQ.CPP

END_RESPONSE_TABLE;

```

////////////////////////////////////
// DIALOG_2      Class Definition
////////////////////////////////////
class TDlg2 : public TDialog
{
public:
    TDlg2(TWindow *parent, TResId resId);

protected:
    void CmOk();

    DECLARE_RESPONSE_TABLE(TDlg2);
};

DEFINE_RESPONSE_TABLE1(TDlg2, TDialog)
// no fuction !!!
END_RESPONSE_TABLE;

```

```

////////////////////////////////////
// The TWndw Class's implementation.
// TWndw      TWndw      TWndw      TWndw
////////////////////////////////////
// TWndw::TWndw()
//
// This is the main window's constructor.
////////////////////////////////////
TWndw::TWndw(TWindow *parent, const char far *title) :
    TFrameWindow(parent, title)
{
    // Assign the window's menu.
    AssignMenu(MENU_1);

    // Size and position the window.
    Attr.X = 0;
    Attr.Y = 0;
    Attr.W = GetSystemMetrics(SM_CXSCREEN) / 1.0;
    Attr.H = GetSystemMetrics(SM_CYSCREEN) / 1.0;

    // Initialize the transfer buffer.
    for(int i=0; i<CH_NO; i++) {
        memset(&transBuf[i], 0, sizeof(transBuf[i]));
        transBuf[i].Edit_ChNo[0] = 0x30 + i;
    }
}

```

```

////////////////////////////////////
// TWndw::Paint()
//
// This function, which overrides TWindow's Paint() member
// function, responds to WM_PAINT messages, which Windows
// sends to a window when it must be repainted. However,
// by calling this function with a TPrinter object's DC for
// the first parameter, it can be used to print the
// contents of the window on the printer.
////////////////////////////////////
void TWndw::Paint(TDC &dc, BOOL, TRect&)

```

WDAQ.CPP

```

{
    Display(dc);
}

////////////////////////////////////
void TWndw::ReDraw()
{
    TClientDC dc(HWindow);
    Display(dc);
}

////////////////////////////////////
void TWndw::Display(TDC& dc)
{
    float hz[CH_NO][MAX_HZ], volt[CH_NO][MAX_HZ], deg[CH_NO][MAX_HZ];
    float wave_tmp;
    int n, k, xx, yy;

    //if flag=true, recalculate wave[CH_NO][SAMP_NO]
    if(flag){
        for(n=0; n<CH_NO; n++){
            for(k=0; k<MAX_HZ; k++){
                hz[n][k]=atof(transBuf[n].Edit_Hz[k]);
                volt[n][k]=atof(transBuf[n].Edit_Volt[k]);
                deg[n][k]=atof(transBuf[n].Edit_Deg[k]);
            }

            for(n=0; n<CH_NO; n++){
                for(int i=0; i<SAMP_NO; i++) {
                    wave_tmp = 0.0;
                    for(k=0; k<MAX_HZ; k++){
                        if(hz[n][k]>=1.0) wave_tmp = wave_tmp +
2048/V_REF*volt[n][k]*sin(2*PI*(hz[n][k]*i/SAMP_NO+deg[n][k]/360));
                    wave[n][i] = (int) wave_tmp;
                    if(wave[n][i] > 2047) wave[n][i] = 2047;
                    if(wave[n][i] < -2048) wave[n][i] = -2048;
                }
            }
            flag = FALSE;

            // Initialize pen width and color.
            int pw = 2;
            TPen *pen = new TPen(TColor::Gray, pw, PS_SOLID);
            TBrush *brush = new TBrush(TColor::White);
            // Select the new pen and brush into the DC.
            dc.SelectObject(*pen);
            dc.SelectObject(*brush);

            // Draw a rectangle.
            dc.Rectangle(100, 50, 570, 400);
            dc.TextOut(50, 94, "CH 0");
            dc.TextOut(50, 174, "CH 1");
            dc.TextOut(50, 254, "CH 2");
            dc.TextOut(50, 334, "CH 3");

            for(n=0; n<CH_NO; n++) {
                dc.MoveTo(100, 100 + 80*n);
                for(k=100; k<570; k++) {
                    xx=k;
                    yy=100 + 80*n +

```

WDAQ.CPP

```

wave[n] [(k-100)*7]/2048.0*80*V_REF/DRAW_SCALE;
        dc.LineTo(xx,yy);
    }
}

////////////////////////////////////
// TWndw::CmFileNew()
//
// This function responds to the File/New File menu
// command by ...
//
////////////////////////////////////
void TWndw::CmFileNew()
{
    filename[0]='\0';
    // Initialize the transfer buffer.
    for(int i=0; i<CH_NO; i++) {
        memset(&transBuf[i], 0, sizeof(transBuf[i]));
        transBuf[i].Edit_ChNo[0] = 0x30 + i;
    }
    flag = TRUE;
    ReDraw();
}

////////////////////////////////////
// TWndw::CmFileOpen()
//
// This function responds to the File/Open File menu
// command by displaying Windows' Open common dialog
// box.
//
////////////////////////////////////*/
void TWndw::CmFileOpen()
{
    char errorMsg[81];

    // Create the dialog box's TData object.
    TOpenSaveDialog::TData fileData(OFN_FILEMUSTEXIST|
        OFN_HIDEREADONLY|OFN_PATHMUSTEXIST,
        "Data Files (*.dat)|*.dat|All Files (*.*)|*.*|",
        0, 0,
        "");

    // Create the Open dialog box.
    TFileOpenDialog *dialog =
        new TFileOpenDialog(this, fileData);

    // Execute the Open dialog box.
    int result = dialog->Execute();

    // Respond to the dialog box's OK button.
    if (result == IDOK) {
        // copy the new name into memory
        strcpy(filename, fileData.FileName);
        fpin.open(fileData.FileName, ios::in);
        for(int i=0; i<CH_NO; i++)
            fpin.read((char*) &transBuf[i], sizeof(struct
TTransBuf));
        fpin.close();
        flag = TRUE;
        ReDraw();
    }
}

```

WDAQ.CPP

```

        else if (fileData.Error != 0) {
            wsprintf(errorMsg,
                "Error #%ld occurred.", fileData.Error);
            MessageBox(errorMsg, "ERROR",
                MB_OK | MB_ICONEXCLAMATION);
        }
    }

    //////////////////////////////////////
    // TWndw::CmFileSave()
    //
    // This function responds to the Save menu command by
    // displaying Windows' Save common dialog box.
    ////////////////////////////////////// */
    void TWndw::CmFileSave()
    {
        char errorMsg[81];

        // check if this is a new file, if yes pick the name from memory
        // if no, ask a new filename
        if(!filename[0]=='\0') {
            fpout.open(filename, ios::out);
            for(int i=0; i<CH_NO; i++)
                fpout.write((char*) &transBuf[i], sizeof(struct
TTransBuf));
            fpout.close();
            return ;
        }

        // Create the dialog box's TData object.
        TOpenSaveDialog::TData fileData(
            OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT,
            "Data Files (*.dat)|*.dat|All Files (*.*)|*.*|",
            0, "\\cpp", "TXT");

        // Create the Save dialog box.
        TFileSaveDialog *dialog =
            new TFileSaveDialog(this, fileData);

        // Execute the Save dialog box.
        int result = dialog->Execute();

        // Respond to the dialog box's OK button.
        if (result == IDOK){
            // copy the new name into memory
            strcpy(filename, fileData.FileName);
            fpout.open(filename, ios::out);
            for(int i=0; i<CH_NO; i++)
                fpout.write((char*) &transBuf[i], sizeof(struct TTransBuf));
            fpout.close();
        }

        else if (fileData.Error != 0) {
            wsprintf(errorMsg,
                "Error #%ld occurred.", fileData.Error);
            MessageBox(errorMsg, "ERROR",
                MB_OK | MB_ICONEXCLAMATION);
        }
    }
}

```

WDAQ.CPP

```

/////////////////////////////////////////////////////////////////
// TWndw::CmFileSaveAs()
//
// This function responds to the Save menu command by
// displaying Windows' Save common dialog box.
/////////////////////////////////////////////////////////////////
void TWndw::CmFileSaveAs()
{
    char errorMsg[81];

    // Create the dialog box's TData object.
    TOpenSaveDialog::TData fileData(
        OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT,
        "Data Files (*.dat)|*.dat|All Files (*.*)|*.*|",
        0, "\\cpp", "TXT");

    // Create the Save dialog box.
    TFileSaveDialog *dialog =
        new TFileSaveDialog(this, fileData);

    // Execute the Save dialog box.
    int result = dialog->Execute();

    // Respond to the dialog box's OK button.
    if (result == IDOK){
        // copy the new name into memory
        strcpy(filename, fileData.FileName);
        fpout.open(fileData.FileName, ios::out);
        for(int i=0; i<CH_NO; i++)
            fpout.write((char*) &transBuf[i], sizeof(struct
TTransBuf));
        fpout.close();
    }
    //
    // MessageBox(fileData.FileName,
    // "Dialog Result", MB_OK);
    //
    else if (fileData.Error != 0) {
        wsprintf(errorMsg,
            "Error #%ld occurred.", fileData.Error);
        MessageBox(errorMsg, "ERROR",
            MB_OK | MB_ICONEXCLAMATION);
    }
}

/////////////////////////////////////////////////////////////////
// TWndw::CmSetCh0()
//
// This function responds to the main menu's CM_SET_CH0
// command, displaying and handling the dialog box.
/////////////////////////////////////////////////////////////////
void TWndw::CmSetCh0()
{
    ch_no = 0;
    // Construct the dialog box.
    TDialog *dialog = new TDlg1(this, DIALOG_1);

    // Display the dialog box.
    int result = dialog->Execute();

    // Check whether the user exited the dialog
    // box via the OK button and display the
    // new dialog box data if he did.

```

WDAQ.CPP

```

        if (result == IDOK) {
            Invalidate();
            flag = TRUE;
        }
    }

    //////////////////////////////////////
    // TWndw::CmSetCh1()
    //
    void TWndw::CmSetCh1()
    {
        ch_no = 1;
        // Construct the dialog box.
        TDialog *dialog = new TDlg1(this, DIALOG_1);

        // Display the dialog box.
        int result = dialog->Execute();

        // Check whether the user exited the dialog
        // box via the OK button and display the
        // new dialog box data if he did.
        if (result == IDOK) {
            Invalidate();
            flag = TRUE;
        }
    }

    //////////////////////////////////////
    // TWndw::CmSetCh2()
    //
    void TWndw::CmSetCh2()
    {
        ch_no = 2;
        // Construct the dialog box.
        TDialog *dialog = new TDlg1(this, DIALOG_1);

        // Display the dialog box.
        int result = dialog->Execute();

        // Check whether the user exited the dialog
        // box via the OK button and display the
        // new dialog box data if he did.
        if (result == IDOK) {
            Invalidate();
            flag = TRUE;
        }
    }

    //////////////////////////////////////
    // TWndw::CmSetCh3()
    //
    void TWndw::CmSetCh3()
    {
        ch_no = 3;
        // Construct the dialog box.
        TDialog *dialog = new TDlg1(this, DIALOG_1);

        // Display the dialog box.
        int result = dialog->Execute();

        // Check whether the user exited the dialog

```


WDAQ.CPP

```

        // box via the OK button and display the
        // new dialog box data if he did.
        if (result == IDOK) {
            Invalidate();
            flag = TRUE;
        }
    }

////////////////////////////////////
// TWndw::CmRunStart()
//
// This function responds to the main menu's CM_START_RUN
// command, displaying and handling the dialog box.
////////////////////////////////////
void TWndw::CmRunStart()
{
    // Initialize pen width and color.
    TClientDC dc(HWindow);
    int pw = 6;
    TPen *pen = new TPen(TColor::LtRed, pw, PS_SOLID);
    TBrush *brush = new TBrush(TColor::White);
    // Select the new pen and brush into the DC.
    dc.SelectObject(*pen);
    dc.SelectObject(*brush);
    dc.Rectangle(150,150,350,200);
    dc.TextOut(165, 165 , "... Click Any Key to Stop!");

    int w0, w1, w2, w3, count = 0;
    STATUS_VALUE status_value;

    // Clear FIFO.
    inpw(FIFO);

    // smooth the start part of wave from zero
    for(int ir=0; ir<RAMP_LENGTH; ir++) Smooth_s(ir);

    //start output data to four channels
    while(!GetInputState()) {
        do { *((int*) &status_value) = inpw(STATUS);
        }while(status_value.FH ==0);
        for(int i=0; i<64; i++) {
            w0 = wave[0][count];
            w1 = wave[1][count];
            w2 = wave[2][count];
            w3 = wave[3][count++];
            outpw(FIFO, w0);
            outpw(FIFO, w1);
            outpw(FIFO, w2);
            outpw(FIFO, w3);
            if(count >= SAMP_NO) count = 0;
            outpw(FIFO, (w0 + wave[0][count])/2);
            outpw(FIFO, (w1 + wave[1][count])/2);
            outpw(FIFO, (w2 + wave[2][count])/2);
            outpw(FIFO, (w3 + wave[3][count])/2);
        }
    }
    // smooth the end part of wave to zero
    Smooth_e(count);

    //refresh the windows

```

WDAQ.CPP

```

    ReDraw();

    //clear the last message
    MSG lpmsg;
    GetMessage(&lpmsg, NULL, 0, 0);
    GetMessage(&lpmsg, NULL, 0, 0);
}

// call this function before starting regular wave
///////////////////////////////////////////////////////////////////
void TWndw::Smooth_s(int ir)
{
    float factor;
    int w0, w1, w2, w3, count = 0;
    STATUS_VALUE status_value;
    //start output data to four channels
    for(;;) {
        do { *((int*) &status_value) = inpw(STATUS);
        }while(status_value.FH ==0);
        for(int i=0; i<64; i++) {
            factor = (float(SAMP_NO)*ir +
count)/(float(SAMP_NO)*RAMP_LENGTH);
            factor *= factor;
            w0 = wave[0][count]*factor;
            w1 = wave[1][count]*factor;
            w2 = wave[2][count]*factor;
            w3 = wave[3][count++]*factor;
            outpw(FIFO, w0);
            outpw(FIFO, w1);
            outpw(FIFO, w2);
            outpw(FIFO, w3);

            outpw(FIFO, w0);
            outpw(FIFO, w1);
            outpw(FIFO, w2);
            outpw(FIFO, w3);

            if(count >= SAMP_NO) count = 0;
//            outpw(FIFO, (w0 + wave[0][count])/2*factor);
//            outpw(FIFO, (w1 + wave[1][count])/2*factor);
//            outpw(FIFO, (w2 + wave[2][count])/2*factor);
//            outpw(FIFO, (w3 + wave[3][count])/2*factor);
            if(count == 0) break;
        }
        if(count == 0) break;
    }
}

// call this function after ending regular wave
///////////////////////////////////////////////////////////////////
void TWndw::Smooth_e(int count)
{
    int factor, ik = SAMP_NO;
    int w0, w1, w2, w3;
    STATUS_VALUE status_value;
    //start output data to four channels
    for(;;) {
        do { *((int*) &status_value) = inpw(STATUS);
        }while(status_value.FH ==0);
        for(int i=0; i<64; i++) {
            factor = SAMP_NO / ik;

```

```

        w0 = wave[0][count]/factor;
        w1 = wave[1][count]/factor;
        w2 = wave[2][count]/factor;
        w3 = wave[3][count++]/factor;
        outpw(FIFO, w0);
        outpw(FIFO, w1);
        outpw(FIFO, w2);
        outpw(FIFO, w3);
        if(count >= SAMP_NO) count = 0;
        outpw(FIFO, (w0 + wave[0][count])/2/factor);
        outpw(FIFO, (w1 + wave[1][count])/2/factor);
        outpw(FIFO, (w2 + wave[2][count])/2/factor);
        outpw(FIFO, (w3 + wave[3][count])/2/factor);
        ik-- ;
        if(ik < 1) break;
    }
    if(ik < 1) break;
}

// The TDlg1 Class's implementation.
// TDlg1      TDlg1 TDlg1      TDlg1
// TDlg1::TDlg1()
//
// This is the dialog box's constructor.
// TDlg1::TDlg1(TWindow *parent, TResId resId) :
//     TDialog(parent, resId)
// {
//     TEdit *edit;
//     TValidator *valid;

// Create an OWL control object for each control
// element in the dialog box that will participate
// in the transfer. Also, attach validator objects
// to edit controls that need to be verified.
new TEdit(this, IDC_EDIT_00, sizeof(transBuf[0].Edit_ChNo));
int idc=IDC_EDIT_1;
for( ; idc<=IDC_EDIT_10; idc++) {
    edit=new TEdit(this, idc, sizeof(transBuf[0].Edit_Hz[0]));
    valid = new TFilterValidator("0-9");
    edit->SetValidator(valid);
}
for( ; idc<=IDC_EDIT_20; idc++) {
    edit=new TEdit(this, idc, sizeof(transBuf[0].Edit_Volt[0]));
    valid = new TFilterValidator("0-9.");
    edit->SetValidator(valid);
}
for( ; idc<=IDC_EDIT_30; idc++) {
    edit=new TEdit(this, idc, sizeof(transBuf[0].Edit_Deg[0]));
    valid = new TFilterValidator("0-9-");
    edit->SetValidator(valid);
}

// Assign the dialog box's transfer buffer.
TransferBuffer = &transBuf[ch_no];
}

```

WDAQ.CPP

```

////////////////////////////////////
// TDlg1::SetupWindow()
//
// This function overrides TDialog's SetupWindow() and is
// called right before the dialog is displayed. This is a
// good place to perform last minute initialization.
////////////////////////////////////
void TDlg1::SetupWindow()
{
    // IMPORTANT! Allow regular setup to be performed.
    TDialog::SetupWindow();

    // Save the current dialog data in case the user
    // changes his mind by selecting the Cancel button.
    memcpy(&oldTransBuf, &transBuf[ch_no], sizeof(transBuf[ch_no]));
}

////////////////////////////////////
// TDlg1::CmCancel()
//
// This function, which overrides a TDialog member
// function, responds to the dialog box's Cancel button,
// restoring the transfer buffer to its original state.
////////////////////////////////////
void TDlg1::CmCancel()
{
    // Restore the transfer buffer.
    memcpy(&transBuf[ch_no], &oldTransBuf, sizeof(oldTransBuf));

    // Allow the overridden CmCancel() function to execute.
    TDialog::CmCancel();
}

////////////////////////////////////
// TDlg1::CmCopy()
//
void TDlg1::CmCopy()
{
    // Construct the dialog box.
    TDialog *dialog = new TDlg2(this, DIALOG_2);

    // Display the dialog box.
    int result = dialog->Execute();

    // Check whether the user exited the dialog
    // box via the OK button and display the
    // new dialog box data if he did.
    if (result == IDOK) {
        int icopych = atoi(copych);
        memcpy(&transBuf[ch_no], &transBuf[icopych],
sizeof(TTransBuf));
        transBuf[ch_no].Edit_ChNo[0] = 0x30 + ch_no;
        TransferData(tdSetData);
        Invalidate();
    }
}

////////////////////////////////////
// The TDlg2 Class's implementation.

```

WDAQ.CPP

```

// TDlg2      TDlg2 TDlg2      TDlg2
//////////////////////////////////////////////////
// TDlg2::TDlg2()
//
// This is the dialog box's constructor.
//////////////////////////////////////////////////
TDlg2::TDlg2(TWindow *parent, TResId resId) :
    TDialog(parent, resId)
{
    TEdit *edit;
    TValidator *valid;

    // Create an OWL control object for each control
    // element in the dialog box that will participate
    // in the transfer. Also, attach validator objects
    // to edit controls that need to be verified.
    edit=new TEdit(this, IDC_EDIT_COPYCH, sizeof(copych));
    valid = new TFilterValidator("0-3");
    edit->SetValidator(valid);

    // Initialize the buffer
    itoa(ch_no, copych, 10);
    copych[1] = 0;

    // Assign the dialog box's transfer buffer.
    TransferBuffer = &copych;
}

//////////////////////////////////////////////////
// TDlg2::CmOK()
//
// This function, which overrides a TDialog member
// function, responds to the dialog box's OK button,
// restoring the transfer buffer to its original state.
//////////////////////////////////////////////////
void TDlg2::CmOk()
{
    // Restore the transfer buffer.
    memcpy(&transBuf[ch_no], &transBuf[0], sizeof(TTransBuf));
    // Allow the overridden CmCancel() function to execute.
    TDialog::CmOk();
}

//////////////////////////////////////////////////
// The TApp class's implementation.
// TApp      TApp TApp      TApp
//////////////////////////////////////////////////

//////////////////////////////////////////////////
// TApp::InitMainWindow()
//
// This function, which overrides TApplication's
// InitMainWindow(), creates the application's main window.
//////////////////////////////////////////////////
void TApp::InitMainWindow()
{
    TFrameWindow *wndw = new TWndw(0, "Waveform Synthesis and Output");
    SetMainWindow(wndw);
}

```

WDAQ.CPP

```
////////////////////////////////////  
// OwlMain()  
////////////////////////////////////  
int OwlMain(int, char*[])  
{  
    initialize();  
    return TApp().Run();  
}
```

WDAQ.H

```

////////////////////////////////////
// WDAQ.H
//
// Head file for WDAQ.CPP
//
// Written by: Weiping Xu
//             Carleton University
//             March 1997
//
////////////////////////////////////

// AT-AO-10 hardware configuration
// -----
// Base I/O address: Hex 1C0   | U72 (A9 A8 A7 A6 A5)=01110
// -----
// DMA Channel: 5              | W22: DRQ5, DACK5
//                          | W24: no jumpers
// -----
// Interrupt Level:           |
//   Group 1-- level 11       | W23: Row 11
//   Group 2-- level 10       | W24: Row 10
// -----
// Reference: CH0--external   | W1: A-B
//             CH1--external   | W2: A-B
//             CH2--external   | W5: A-B
//             CH3--external   | W6: A-B
// -----
// Output polar:CH0--bipolar  | W3: B-C
//             CH1--bipolar    | W4: A-B
//             CH2--bipolar    | W7: B-C
//             CH3--bipolar    | W8: A-B

# define CNRATE 100 // sample rate = 1 MHz / CNRATE = 10 KHz

//-----
#define SET      0x01
#define CLEAR   0x00
#define ENABLE  0x01
#define DISABLE 0x00
//-----
// PC I/O Space Base Address for DAQ Control Registers
#define iobase 0x01c0
#define CFG1    iobase + 0x0a
#define CFG2    iobase + 0x02
#define CFG3    iobase + 0x04
#define STATUS  iobase + 0x0a
#define CNTR1   iobase + 0x06
#define CNTRCMD iobase + 0x09
#define FIFO    iobase + 0x0c

typedef struct
{
    unsigned int CH3_1      : 4; //Channel select (0-9)
    unsigned int DMAEN      : 1; //DMA operation enable
    unsigned int DMARQ      : 1; //DMA request
    unsigned int EXTUPDEN   : 1; //DAC group 1 update soucre
    selectionexternal interrupt 2 enable
                                //(CLEAR--by counter 1; SET--by EXT)
    unsigned int GRP2WR     : 1; //Registers selection
    unsigned int FIFOEN     : 1; //FIFO memory enable

```

WDAQ.H

```

        unsigned int CNT1SRC      : 1; //Counter 1 soucre select
(CLEAR--1MHz)
        unsigned int CNT2SRC      : 1; //Counter 2 soucre select
(SER--counter3)
        unsigned int TCINTEN      : 1; //DMA TC interrupt enable
        unsigned int CNTINT1EN    : 1; //Counter interrupt 1 enable
        unsigned int CNTINT2EN    : 1; //Counter interrupt 2 enable
        unsigned int EXTINT1EN    : 1; //External interrupt 1 enable
        unsigned int EXTINT2EN    : 1; //External interrupt 2 enable
} CFG1_COMMAND;

typedef struct
{
        unsigned int SDATA        : 1; //For EEPROM (=CLEAR)
        unsigned int SCLK         : 1; //For EEPROM (=CLEAR)
        unsigned int PROMEN       : 1; //EEPROM enable
        unsigned int LDAC0        : 1; //Double-buffered DAC output
        unsigned int LDAC2        : 1; //((CLEAR--immediately)
        unsigned int LDAC4        : 1; //((SET--by update source)
        unsigned int LDAC6        : 1; //
        unsigned int LDAC8        : 1; //
        unsigned int DAC2S0       : 1; //DAC inout data format select
        unsigned int DAC2S2       : 1; //((CLEAR--two's complement format)
        unsigned int DAC2S4       : 1; //((SET--straight binary)
        unsigned int DAC2S6       : 1; //
        unsigned int DAC2S8       : 1; //
        unsigned int FFR TEN      : 1; //FIFO restransmit enable
        unsigned int CALLD        : 2; //Calibration enable (00-no operation)
} CFG2_COMMAND;

typedef struct
{
        unsigned int SCANEN       : 1; //DAC channel scan mode enable
        unsigned int EN25V        : 1; //2.5V output enable (CH 9)
        unsigned int DOUTEN1      : 1; //Digital output enable(I/O 0-3)
        unsigned int DOUTEN2      : 1; //Digital output enable(I/O 4-7)
        unsigned int RCKLEN       : 1; //RTI Bus clock enable
        unsigned int CLKOUT       : 1; //Onboard clock output enable
        unsigned int DMAMODE      : 1; //DMA request mode select (no matter)
        unsigned int XXX          : 10; //reserved, always be cleared.
} CFG3_COMMAND;

typedef struct
{
        unsigned int XXX1        : 5; //Don't care bits
        unsigned int FE          : 1; //FIFO empty
        unsigned int FH          : 1; //FIFO half-full
        unsigned int XXX2        : 9; //Don't care bits
} STATUS_VALUE;

typedef struct
{
        unsigned char BCDSEL      : 1; //Binary coded decimal select
        unsigned char MODESEL     : 3; //Counter mode select (000--Mode 0)
        unsigned char RWSEL       : 2; //Read/Write select (01--R/W LSB only)
        unsigned char CNTRSEL     : 2; //Counter select (00--select Counter
1)
} CNTRCMD_COMMAND;

```


WDAQ.DEF

```
////////////////////////////////////
//
// WDAQ.DEF
//
// Written by: Weiping Xu
//             Carleton University
//             March 1997
//
////////////////////////////////////
NAME WDAQ
DESCRIPTION 'Waveform Synthesis'
EXETYPE WINDOWS
STUB 'WINSTUB.EXE'
CODE PRELOAD MOVEABLE DISCARDABLE
DATA PRELOAD MOVEABLE MULTIPLE
HEAPSIZE 1024
STACKSIZE 8192
```

WDAQ.RC

/*****

WDAQ.RC

produced by Borland Resource Workshop

Written by: Weiping Xu
 Carleton University
 March 1997

*****/

```
#ifndef WORKSHOP_INVOKED
#include "windows.h"
#endif
```

#include <owl\printer.rh>

```
#define MENU_1 100
#define CM_SET_CH3 110
#define CM_SET_CH2 109
#define CM_SET_CH1 108
#define CM_SET_CH0 101
#define CM_FILEEXIT 24310
#define CM_FILESAVEAS 24334
#define CM_FILESAVE 24333
#define CM_FILEOPEN 24332
#define CM_FILENEW 24331
#define CM_RUN_START 104
#define CM_HELP_DATA 105
#define CM_HELP_VIEW 106
#define CM_HELP_RUN 107
```

```
#define DIALOG_1 1
#define IDC_EDIT_00 100
#define IDC_EDIT_1 101
#define IDC_EDIT_2 102
#define IDC_EDIT_3 103
#define IDC_EDIT_4 104
#define IDC_EDIT_5 105
#define IDC_EDIT_6 106
#define IDC_EDIT_7 107
#define IDC_EDIT_8 108
#define IDC_EDIT_9 109
#define IDC_EDIT_10 110
#define IDC_EDIT_11 111
#define IDC_EDIT_12 112
#define IDC_EDIT_13 113
#define IDC_EDIT_14 114
#define IDC_EDIT_15 115
#define IDC_EDIT_16 116
#define IDC_EDIT_17 117
#define IDC_EDIT_18 118
#define IDC_EDIT_19 119
#define IDC_EDIT_20 120
#define IDC_EDIT_21 121
#define IDC_EDIT_22 122
#define IDC_EDIT_23 123
#define IDC_EDIT_24 124
#define IDC_EDIT_25 125
#define IDC_EDIT_26 126
```

WDAQ.RC

```
#define IDC_EDIT_27 127
#define IDC_EDIT_28 128
#define IDC_EDIT_29 129
#define IDC_EDIT_30 130
#define IDC_COPY 150
#define DIALOG_2 2
#define IDC_EDIT_COPYCH 101
```

```
#ifdef RC_INVOKED
#include <owl\printer.rc>
#include <owl\except.rc>
```

MENU_1 MENU

```
{
  POPUP "&File"
  {
    MENUITEM "&New", CM_FILENEW
    MENUITEM "&Open...", CM_FILEOPEN
    MENUITEM "&Save", CM_FILESAVE
    MENUITEM "Save &as...", CM_FILESAVEAS
    MENUITEM SEPARATOR
    MENUITEM "E&xit", CM_FILEEXIT
  }

  POPUP "&Settings"
  {
    MENUITEM "Set_Ch&0", CM_SET_CH0
    MENUITEM "Set_Ch&1", CM_SET_CH1
    MENUITEM "Set_Ch&2", CM_SET_CH2
    MENUITEM "Set_Ch&3", CM_SET_CH3
  }

  POPUP "&Run"
  {
    MENUITEM "St&art", CM_RUN_START
  }

  POPUP "&HELP"
  {
    MENUITEM "Data Input", CM_HELP_DATA
    MENUITEM "View", CM_HELP_VIEW
    MENUITEM "Run", CM_HELP_RUN
  }
}
```

DIALOG_1 DIALOG 29, 37, 267, 210

STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU

CAPTION "Waveform Setting"

FONT 8, "MS Sans Serif"

```
{
  DEFPUSHBUTTON "OK", IDOK, 191, 119, 42, 14, BS_DEFPUSHBUTTON |
WS_TABSTOP
  PUSHBUTTON "Cancel", IDCANCEL, 191, 141, 42, 14, WS_TABSTOP
  PUSHBUTTON "Help", IDHELP, 191, 163, 42, 14, WS_TABSTOP
  PUSHBUTTON "Copy", IDC_COPY, 191, 68, 42, 14
  LTEXT "Hz", -1, 36, 34, 16, 8
  LTEXT "Volt", -1, 77, 34, 16, 8
  LTEXT "Deg.", -1, 120, 34, 16, 8
```

WDAQ.RC

```

LTEXT "CHANNEL:", -1, 88, 8, 37, 8
EDITTEXT IDC_EDIT_00, 132, 8, 10, 10, ES_READONLY | NOT WS_BORDER |
WS_DISABLED | WS_TABSTOP
LTEXT "1", -1, 13, 50, 11, 12
LTEXT "2", -1, 13, 65, 11, 12
LTEXT "3", -1, 13, 80, 11, 12
LTEXT "4", -1, 13, 95, 11, 12
LTEXT "5", -1, 13, 110, 11, 12
LTEXT "6", -1, 13, 125, 11, 12
LTEXT "7", -1, 13, 140, 11, 12
LTEXT "8", -1, 13, 155, 11, 12
LTEXT "9", -1, 13, 170, 11, 12
LTEXT "10", -1, 13, 185, 11, 12
EDITTEXT IDC_EDIT_1, 32, 50, 28, 12
EDITTEXT IDC_EDIT_2, 32, 65, 28, 12
EDITTEXT IDC_EDIT_3, 32, 80, 28, 12
EDITTEXT IDC_EDIT_4, 32, 95, 28, 12
EDITTEXT IDC_EDIT_5, 32, 110, 28, 12
EDITTEXT IDC_EDIT_6, 32, 125, 28, 12
EDITTEXT IDC_EDIT_7, 32, 140, 28, 12
EDITTEXT IDC_EDIT_8, 32, 155, 28, 12
EDITTEXT IDC_EDIT_9, 32, 170, 28, 12
EDITTEXT IDC_EDIT_10, 32, 185, 28, 12
EDITTEXT IDC_EDIT_11, 72, 50, 28, 12
EDITTEXT IDC_EDIT_12, 72, 65, 28, 12
EDITTEXT IDC_EDIT_13, 72, 80, 28, 12
EDITTEXT IDC_EDIT_14, 72, 95, 28, 12
EDITTEXT IDC_EDIT_15, 72, 110, 28, 12
EDITTEXT IDC_EDIT_16, 72, 125, 28, 12
EDITTEXT IDC_EDIT_17, 72, 140, 28, 12
EDITTEXT IDC_EDIT_18, 72, 155, 28, 12
EDITTEXT IDC_EDIT_19, 72, 170, 28, 12
EDITTEXT IDC_EDIT_20, 72, 185, 28, 12
EDITTEXT IDC_EDIT_21, 115, 50, 28, 12
EDITTEXT IDC_EDIT_22, 115, 65, 28, 12
EDITTEXT IDC_EDIT_23, 115, 80, 28, 12
EDITTEXT IDC_EDIT_24, 115, 95, 28, 12
EDITTEXT IDC_EDIT_25, 115, 110, 28, 12
EDITTEXT IDC_EDIT_26, 115, 125, 28, 12
EDITTEXT IDC_EDIT_27, 115, 140, 28, 12
EDITTEXT IDC_EDIT_28, 115, 155, 28, 12
EDITTEXT IDC_EDIT_29, 115, 170, 28, 12
EDITTEXT IDC_EDIT_30, 115, 185, 28, 12
}

DIALOG_2 DIALOG 64, 47, 151, 67
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
// CLASS "bordlg"
FONT 8, "MS Sans Serif"
{
    DEFPUSHBUTTON "OK", IDOK, 31, 40, 36, 14
    PUSHBUTTON "Cancel", IDCANCEL, 84, 40, 36, 14
    LTEXT "Copy Setting From Channel:", -1, 17, 16, 93, 8
    EDITTEXT IDC_EDIT_COPYCH, 115, 15, 12, 11
}

// CAPTION "Car Libaray Maintenance"

#endif

```

Appendix F

Source Code and Memory Configuration for SISO Feedforward Control

(1) DSP.C	Main file for implementing SISO filtered-x LMS algorithm on QPC/40B board and PC/16IO8 board with online identification for the secondary path
(2) FIR.ASM	Assembly code for FIR filter, adaptive coefficient update and circular addressing vector renew

Memory configuration for DSP.C:

- circular addressing mode buffer: internal memory
- .text section: internal memory
- .bss section: internal memory
- .stack section: global external memory

DSP.C

```

/*****
/* SISO filtered-x LMS
/*
/* This c code is used to implement filtered-x LMS algorithm
/* on QPC/40B board and PC/16IO8 board with online identification
/* for the secondary path.
/* FIR.asm has to be linked.
/*
/* Written by: Weiping Xu
/* Carleton University
/* September 1997
/*
*****/
#include "\dsptools\stdlib.h"
#include "math.h"
#include "intpt40.h" /* Interrupt support (Parallel Support
Lib) */

/*****
/* DSPLINK Addresses (1 ws access) */
#define DSPLINK 0xB0000100 /* 1 ws */

/* DSPLINK Addresses/Registers for PC/16IO8 now follow */
#define ADC0 (long*)(DSPLINK+0)
#define ADC1 (long*)(DSPLINK+1)
#define ADC2 (long*)(DSPLINK+2)
#define ADC3 (long*)(DSPLINK+3)

#define DAC0 (long*)(DSPLINK+0x10)
#define DAC1 (long*)(DSPLINK+0x11)
#define DAC2 (long*)(DSPLINK+0x12)
#define DAC3 (long*)(DSPLINK+0x13)
#define DAC4 (long*)(DSPLINK+0x14)
#define DAC5 (long*)(DSPLINK+0x15)
#define DAC6 (long*)(DSPLINK+0x16)
#define DAC7 (long*)(DSPLINK+0x17)

#define CR ((unsigned long*)(DSPLINK+0x18))
#define SR ((unsigned long*)(DSPLINK+0x18))
#define TIMER16 ((unsigned long*)(DSPLINK+0x19))
#define TIMER2 ((unsigned long*)(DSPLINK+0x1A))
#define PGR ((unsigned long*)(DSPLINK+0x1B))

#define FSAMPLE 5000

/*****
/* control constants and global variables */
#define NA 22
#define NB 21
#define NC 28
#define ND 27
#define MU0 0.01 /* converge coefficient of control path */
#define MU1 0.02 /* converge coefficient of identif. path */
#define BOARD_GAIN 10
#define SCALE_FACTOR 2048 /* 12-bit scale factor */

float *ptrv= (float*) 0x002ffd00; /* filter length must <= 64 */
float *ptrz= (float*) 0x002ffd40;
float *ptrx= (float*) 0x002ffd80;
float *ptrr= (float*) 0x002ffdc0;
float *ptrs= (float*) 0x002ffe00;

```

DSP.C

```

float *ptru= (float*) 0x002ffe40;
float      *a= (float*) 0x002ffe80;
float      *b= (float*) 0x002ffec0;
float      *c= (float*) 0x002fff00;
float      *d= (float*) 0x002fff40;
float      *cl= (float*) 0x002fff80;
float      *dl= (float*) 0x002fffc0;
volatile float x, e, v;
int na, nb, nc, nd, nac, nbc, nad, nbd;
float mu0, mu1;
/*****/

/*****/
/* function prototypes *****/
void main(void);
void init_board0(void);
void c_int04(void);

/*****/
/*                               */
/* main()                        */
/*                               */
/*****/
volatile int count=0;
volatile int interval=0;
volatile int dummy;

void main(void)
{
    int i;
    float *ptr0=ptrv;

    na=NA;
    nb=NB;
    nc=NC;
    nd=ND;
    if(NA >= NC) nac=NA; else nac=NC;
    if(NB >= NC) nbc=NB; else nbc=NC;
    if(NA >= ND) nad=NA; else nad=ND;
    if(NB >= ND) nbd=NB; else nbd=ND;
    mu0=MU0;
    mu1=MU1;
    /* initialize the global variable */
    for(i=0; i< 64*12; i++) *(ptr0++)=0;

    /*****/
    /* Set up the C40 interrupts */
    /*****/

    /* Need to perform an IACK instruction to allow external */
    /* interrupts through to the C40 (for DB40) */
    /*
    asm("    PUSH    AR0");
    asm("    PUSH    DP");
    asm("    LDI     030H, AR0");
    asm("    LSH     16, AR0");
    asm("    IACK    *AR0");
    asm("    POP     DP");
    asm("    POP     AR0");
    */

```

DSP.C

```

/* Set up the C40 processor */
/* Global disable of interrupts */
    INT_DISABLE();

/* Explicitly set IVTP on 512 word boundary */
    set_ivtp((void *) 0x002ffc00);

/* Set Interrupt Vector for II0F1 */
    install_int_vector((void *) c_int04, 0x04);

/* Enable IIF01 to be level triggered int. */
    load_iif(0x00B0);

/* Initialise the multi-channel board */
    init_board0();

/* Clear pending ints and enable C40 Interrupts for DSPLINK */
    dummy = *ADC0;
    INT_ENABLE();

/* Turn ON the Cache */
    CACHE_ON();

/* Loop for interrupts */
    while(count < 50*FSAMPLE);

    INT_DISABLE();

    asm("stop    NOP");
    asm("        .globl stop");
}    /* end of main() */

/*****
/*
/* Function: init_board0()
/* Purpose:  Initialise the Multi-Channel Analog I/O Board.
/*
*****/
void init_board0(void)
{
/* Control Register - reset the board */
    *CR    = 0L;

/* Clear the DAC input latches */
    *DAC0 = 0x00000000;
    *DAC1 = 0x00000000;
    *DAC2 = 0x00000000;
    *DAC3 = 0x00000000;
    *DAC4 = 0x00000000;
    *DAC5 = 0x00000000;
    *DAC6 = 0x00000000;
    *DAC7 = 0x00000000;

/* Calculate the TIMER Values */
#define TVAL16 0x00010000 * (long) (10000000.0 / (float) FSAMPLE);
#define TVAL2  (long) (10000000.0 / (float) FSAMPLE);
    *TIMER16 = TVAL16;
    *TIMER2  = TVAL2;

```


DSP.C

```

/* Set up Programmable Gain Amps */
if (BOARD_GAIN == 1) *PGR = 0x00000000L; /* Gain = 1 */
if (BOARD_GAIN == 10) *PGR = 0x00550000L; /* Gain = 10 */
if (BOARD_GAIN == 100) *PGR = 0x00AA0000L; /* Gain = 100 */

/* Set up the Control Register for Interrupts */
*CR = 0x30010000; /* Enable interrupts - Level Trig */

} /* end of init_board0() */

/*****
/*
/* Function: c_int04() --- xLMS ----- */
/*
*****/
void c_int04(void)
{
    long output, i;
    count++;
    interval++;

    if(interval > 5000) {
        for(i=0; i<NC; i++) c1[i]=c[i];
        for(i=0; i<ND; i++) d1[i]=d[i];
        interval=0;
    }

    asm(" .globl FIR");
    asm(" .globl UPDATE");

/* (1) get the new x, e, v */
    x = ((float)(*ADC0 >> 16))/SCALE_FACTOR; /* reference signal */
    e = ((float)(*ADC1 >> 16))/SCALE_FACTOR; /* error sensor (accel.) */
/*
    v = ((float)(*ADC2 >> 16))/SCALE_FACTOR; /* ext. injected noise */
/*

/* (2) renew X[] */
    asm(" LDI @_ptrx, AR1");
    asm(" LDI @_nac, BK");
    asm(" LDF @_x, R0");
    asm(" LDF *AR1--(1), R1"); /* dummy read to decrease AR1 */
    asm(" STF R0, *AR1");
    asm(" STI AR1, @_ptrx");

/* (3) C=C-u*(z-e)*V */
    asm(" LDI @_ptrz, AR0");
    asm(" LDF *AR0, R1");
    asm(" SUBF @_e, R1");
    asm(" MPYF @_mul, R1"); /* R1 = mul*(z-e) */

    asm(" LDI @_c, AR0");
    asm(" LAJU UPDATE");
    asm(" LDI @_ptrv, AR1");
    asm(" LDI @_nc, BK");
    asm(" LDI BK, RC");

/* D=D-u*(z-e)*Z R1 = mul*(z-e) */
    asm(" LDI @_d, AR0");

```

DSP.C

```

asm("      LAJU    UPDATE");
asm("      LDI     @_ptrz, AR1");
asm("      LDI     @_nd, BK");
asm("      LDI     BK, RC");

/* (4) renew V[] */
asm("      LDI     @_ptrv, AR1");
asm("      LDI     @_nc, BK");
asm("      LDF     @_v, R0");
asm("      LDF     *AR1--(1)&, R1"); /* dummy read to decrease AR1
*/
asm("      STF     R0, *AR1");
asm("      STI     AR1, @_ptrv");

/* (5) z=C'V+D'Z      */
asm("      LDI     @_c, AR0");
asm("      LAJU    FIR");
asm("      LDI     @_ptrv, AR1");
asm("      LDI     @_nc, BK");
asm("      LDI     BK, RC");

asm("      PUSHF   R0");

asm("      LDI     @_d, AR0");
asm("      LAJU    FIR");
asm("      LDI     @_ptrz, AR1");
asm("      LDI     @_nd, BK");
asm("      LDI     BK, RC");

asm("      POPF    R1");
asm("      ADDF    R1, R0");
asm("      * same AR1 & BK ");
asm("      LDF     *AR1--(1)&, R1");
asm("      STF     R0, *AR1");
asm("      STI     AR1, @_ptrz");

/* (6) r=C'X+D'R      */
asm("      LDI     @_c1, AR0");
asm("      LAJU    FIR");
asm("      LDI     @_ptrx, AR1");
asm("      LDI     @_nac, BK");
asm("      LDI     @_nc, RC");

asm("      PUSHF   R0");

asm("      LDI     @_d1, AR0");
asm("      LAJU    FIR");
asm("      LDI     @_ptrr, AR1");
asm("      LDI     @_nad, BK");
asm("      LDI     @_nd, RC");

asm("      LDI     @_ptrr, AR1");
asm("      POPF    R1");
asm("      ADDF    R1, R0");
asm("      * same BK ");
asm("      LDF     *AR1--(1)&, R1");
asm("      STF     R0, *AR1");
asm("      STI     AR1, @_ptrr");

/*      s=C'U+D'S      */

```

DSP.C

```

asm(" LDI      @_c1, AR0");
asm(" LAJU     FIR");
asm(" LDI      @_ptru, AR1");
asm(" LDI      @_nbc, BK");
asm(" LDI      @_nc, RC");

asm(" PUSHF    R0");

asm(" LDI      @_d1, AR0");
asm(" LAJU     FIR");
asm(" LDI      @_ptrs, AR1");
asm(" LDI      @_nbd, BK");
asm(" LDI      @_nd, RC");

asm(" LDI      @_ptrs, AR1");
asm(" POPF     R1");
asm(" ADDF     R1, R0");
asm(" * same BK ");
asm(" LDF      *AR1--(1)%, R1");
asm(" STF      R0, *AR1");
asm(" STI      AR1, @_ptrs");

/* (7) A=A-u*e*R */
asm(" LDF      @_mu0, R1");
asm(" MPYF     @_e, R1"); /* R1 = mu0*e */

asm(" LDI      @_a, AR0");
asm(" LAJU     UPDATE");
asm(" LDI      @_ptrr, AR1");
asm(" LDI      @_nad, BK");
asm(" LDI      @_na, RC");

/* B=B-u*e*S R1 = mu0*e */
asm(" LDI      @_b, AR0");
asm(" LAJU     UPDATE");
asm(" LDI      @_ptrs, AR1");
asm(" LDI      @_nbd, BK");
asm(" LDI      @_nb, RC");

/* (8) u=A'X+B'U */
asm(" LDI      @_a, AR0");
asm(" LAJU     FIR");
asm(" LDI      @_ptrx, AR1");
asm(" LDI      @_nac, BK");
asm(" LDI      @_na, RC");

asm(" PUSHF    R0");

asm(" LDI      @_b, AR0");
asm(" LAJU     FIR");
asm(" LDI      @_ptru, AR1");
asm(" LDI      @_nbc, BK");
asm(" LDI      @_nb, RC");

asm(" LDI      @_ptru, AR1");
asm(" POPF     R1");
asm(" ADDF     R1, R0");
asm(" * same BK ");
asm(" LDF      *AR1--(1)%, R1");
asm(" STF      R0, *AR1");

```

DSP.C

```
        asm("    STI        AR1, @_ptru");

/* send u+v and z to the DAC port */
    output=(long)((*ptru + v)*SCALE_FACTOR);
    if(output > 700) output=700;
    if(output < -700) output=-700;
    *DAC0=output << 16;
    *DAC1=(long)((*ptrz - e) * SCALE_FACTOR) << 16;

} /* Return (end of c_int04()) */
```

FIR.ASM

```

*****
* FIR.ASM
*
* This is the assembly code for FIR filter and
* adaptive coefficient update
*
* It's used to link to DSP.C
*
* Written by: Weiping Xu
*             Carleton University
*             September 1997
*
*****
*   TITLE FIR FILTER
*****
*   SUBROUTINE FIR
*
*   EQUATION:  $y(n) = h(0) * x(n) + h(1) * x(n-1) + h(N-1) * x(n-(N-1))$ 
*
*****
* LOAD AR0
* LAJU FIR
* LOAD AR1
* LOAD BK
* LOAD RC
*
*****
* ARGUMENT ASSIGNMENTS:
*
*   ARGUMENT | FUNCTION
*   -----|-----
*   AR0      | ADDRESS OF h(0)
*   AR1      | ADDRESS OF x(n)
*   BK       | SIZE OF X
*   RC       | LENGTH OF FILTER
*
* RESIGSTERS USED AS INPUT: AR0, AR1, BK, RC
* RESIGSTERS MODIFIED: R0, R2, AR0, AR1, RC.
* RESIGSTER CONTAINING RESULT: R0
* (AR1 returns to original when BK=RC)
*****

        .version 40
FIR     .globl  FIR

        SUBI 2,RC
        RPTBD      CONV
* Initialize R0
        MPYF3      *AR0++(1), *AR1++(1)%, R0
        LDF 0.0, R2
        NOP
*
*   FILTER (1 <= i < N)
*
CONV MPYF3      *AR0++(1), *AR1++(1)%, R0
||   ADDF3      R0,R2,R2
*
        BUD R11
        ADDF R0,R2,R0
        NOP
        NOP

```

FIR.ASM

```

*
*
* end FIR
*
*
*****
*   TITLE UPDATE quation
*****
*   SUBROUTINE UPDATE
*
*
* A=A-k*R   *
*****
* calculate R1
* LOAD AR0
* LAJU UPDATE
* LOAD AR1
* LOAD BK
* LOAD RC
*
*****
* ARGUMENT ASSIGNMENTS:
*
*   ARGUMENT | FUNCTION
*   -----|-----
*   R1      | k
*   AR0      | ADDRESS OF A(0)
*   AR1      | ADDRESS OF R(0)
*   BK      | SIZE OF R
*   RC      | LENGTH OF A
*
* RESIGSTERS USED AS INPUT: R1, AR0, AR1, BK
* RESIGSTERS MODIFIED: R0, R2, AR0, AR1, RC.
* (AR1 returns to original if BK=RC)
*****
UPDATE    .globl    UPDATE

        SUBI 2,RC
        RPTBD      L10
        MPYF3      R1, *AR1++(1)%, R0
        NOP
        NOP

        MPYF3      R1, *AR1++(1)%, R0
        || SUBF3    R0, *AR0, R2
L10     STF  R2, *AR0++

        BUD  R11
        SUBF3    R0, *AR0, R2
        STF  R2, *AR0++
        NOP
*
* end UPDATE
*
        .end

```

Appendix G

Source Code and Memory Configuration for MIMO Feedforward Control

- (1) MXLMS33.C Main file for implementing 3-input and 3-output filtered-x LMS algorithm on QPC/40B board and PC/16IO8 board with off-line identification for the secondary path
- (2) MXLMS412.C Main file for implementing 12-input and 4-output filtered-x LMS algorithm on QPC/40B board and PC/16IO8 board with off-line
- (3) MASM.ASM Assembly code for FIR/IIR filter, adaptive coefficient update and circular addressing vector renew

Memory configuration for MXLMS33.C:

- circular addressing mode buffer: internal memory
- .text section: local external memory
- .bss section: internal memory
- .stack section: global external memory

Memory configuration for MXLMS412.C:

- circular addressing mode buffer: internal memory
- .text section: local external memory
- .bss section: global external memory
- .stack section: global external memory

MXLMS33.C

```

/*****
/* MIMO filtered-x LMS */
/*
/* This c code is used to implement filtered-x LMS algorithm */
/* on QPC/40B board and PC/16IO8 board with off-line identification*/
/* for the secondary path. */
/* MASM.ASM has to be linked. */
/*
/* Written by: Weiping Xu */
/* Carleton University */
/* December 1997 */
/*
/*****
/*
/* 3-input/3-output */
/* IIR(ID.) /IIR(control) */
/* Off-line identification */
/* regular LMS */
/*
/*****
/*
/* (CH0+CH1+CH2+CH3) e[0] */
/* (CH4+CH5+CH6+CH7) to e[1] */
/* (CH8+CH9+CH10+CH11) to e[2]*/
/*
/*****

/*****
#include "dsptools\stdlib.h"
#include "math.h"
#include "intpt40.h" /* Interrupt support (Parallel Support Lib)
*/

/*****
/* control constants and global variables */
#define M 3 /* channel number of output */
#define L 3 /* ... error */
#define NA 15 /* length of filter A */
#define NB 14 /* ... B */
#define NC 15 /* ... C */
#define ND 14 /* ... D */
#define MU0a 0.001 /* converge coeff. of control path */
#define MU0b 0.001 /* converge coeff. of control path */
#define MU1a 0.001 /* ... identif. path */
#define MU1b 0.001 /* ... identif. path */
#define BOARD_GAIN 10
#define SCALE_FACTOR 2048 /* 12-bit scale factor */
#define FSAMPLE 2000 /* sampling rate in Hz */
#define ID_STEP (60*FSAMPLE) /* off-line identification step number */
#define CN_STEP (20*FSAMPLE) /* control step number */

#define BUFFER_ADDR 0x002ffc00 /* first circular buffer location
*/
#define BUFFER_SIZE 16 /* circular buffer size */
#define PI 3.14159
#define MAX_OUT 1000 /* maximum output voltage =
MAX_OUT*10/2048 */

/*****
/* DSPLINK Addresses (1 ws access) */

```



```

#define DSPLINK 0xB0000100          /* 1 ws */

/* DSPLINK Addresses/Registers for PC/16IO8 now follow */
#define ADC0      (long*)(DSPLINK+0)
#define ADC1      (long*)(DSPLINK+1)
#define ADC2      (long*)(DSPLINK+2)
#define ADC3      (long*)(DSPLINK+3)
#define ADC4      (long*)(DSPLINK+4)
#define ADC5      (long*)(DSPLINK+5)
#define ADC6      (long*)(DSPLINK+6)
#define ADC7      (long*)(DSPLINK+7)
#define ADC8      (long*)(DSPLINK+8)
#define ADC9      (long*)(DSPLINK+9)
#define ADC10     (long*)(DSPLINK+10)
#define ADC11     (long*)(DSPLINK+11)
#define ADC12     (long*)(DSPLINK+12)
#define ADC13     (long*)(DSPLINK+13)
#define ADC14     (long*)(DSPLINK+14)
#define ADC15     (long*)(DSPLINK+15)

#define DAC0      (long*)(DSPLINK+0x10)
#define DAC1      (long*)(DSPLINK+0x11)
#define DAC2      (long*)(DSPLINK+0x12)
#define DAC3      (long*)(DSPLINK+0x13)
#define DAC4      (long*)(DSPLINK+0x14)
#define DAC5      (long*)(DSPLINK+0x15)
#define DAC6      (long*)(DSPLINK+0x16)
#define DAC7      (long*)(DSPLINK+0x17)

#define CR ((unsigned long*)(DSPLINK+0x18))
#define SR ((unsigned long*)(DSPLINK+0x18))
#define TIMER16 ((unsigned long*)(DSPLINK+0x19))
#define TIMER2  ((unsigned long*)(DSPLINK+0x1A))
#define PGR ((unsigned long*)(DSPLINK+0x1B))

/*****/
/* function prototypes *****/
void main(void);
void init_var(void);
void init_board0(void);
void c_int04(void);
float FIR(float*, float*, int);
float IIR(float*, float*, int, float*, float*, int);
void UPDATE(float*, float*, int, float);
float* renew_vect(float*, float);
void identif(float*, float*, float*);
void control(float, float*, float*);
void bandrand(float*);
float bandp(float, float*, float*, float*,
            float*, float*, float*, float*, int);

/*****/
/* global variables */
float *ptrx;          /* pointer to vector x */
float *ptru[M];       /* pointers to vectors u[M] */
float *ptrv[M];       /* pointers to vectors v[M] */
float *ptrz[L];       /* pointers to vectors z[L] */
float *ptrrr[L][M];   /* pointers to vectors r[L][M] */
float *ptrrs[L][M][M]; /* pointers to vectors s[L][L][M] */
float A[M][NA], B[M][M][NB]; /* adaptive filter A */

```

MXLMS33.C

```

float C[L][M][NC], D[L][L][ND]; /* Model C & D */
# define C_New C /* C_New is the copy of model C */
# define D_New D /* D_New is the copy of model D */

volatile int count=0; /* step counter */
volatile int converge=1; /* 0 -- diverge */
volatile int dummy;
int buffer_size=BUFFER_SIZE-1; /* buffer size */

/*****
/*
/* main()
/*
/*****/
void main(void)
{
register int l,m,k;

/* initialize the model */
for(k=0; k< L*M*NC; k++) {
    C[0][0][k]=0;
}

for(k=0; k< L*L*ND; k++) {
    D[0][0][k]=0;
}

init_var();

/*****
/* Set up the C40 interrupts */
/*****/
/* Need to perform an IACK instruction to allow external */
/* interrupts through to the C40 (for DB40) */
asm(" PUSH AR0");
asm(" PUSH DP");
asm(" LDI 030H, AR0");
asm(" LSH 16, AR0");
asm(" IACK *AR0");
asm(" POP DP");
asm(" POP AR0");

/* Set up the C40 processor */
/* Global disable of interrupts */
INT_DISABLE();

/* Explicitly set IVTP on 512 word boundary */
set_ivtp((void *) 0x00310000);

/* Set Interrupt Vector for II0F1 */
install_int_vector((void *) c_int04, 0x04);

/* Enable IIF01 to be level triggered int. */
load_iif(0x00B0);

/* Initialise the multi-channel board */
init_board0();

/* Clear pending ints and enable C40 Interrupts for DSPLINK */
dummy = *ADC0;

```

```

        INT_ENABLE();

/* Turn ON the Cache */
        CACHE_ON();

/*****
/* Process off-line identification */
*****/
/* Loop for interrupts -- */
        while(count < ID_STEP);
        INT_DISABLE();

        asm("stage1    NOP");
        asm("          .globl stage1");

/*****
/* Process control */
*****/
control:

/* Initialise the multi-channel board */
        init_board0();

/* Clear pending ints and enable C40 Interrupts for DSPLINK */
        dummy = *ADC0;
        INT_ENABLE();

/* Loop for interrupts -- control */
        while(count < (CN_STEP+ID_STEP));
        INT_DISABLE();

        asm("stage2    NOP");
        asm("          .globl stage2");

        count = ID_STEP;
        init_var();
        goto control;

}      /*    end of main()    */

/*****
/*
*****/
/* Function: init_var() */
/* Purpose:  Initialise the global variables (exclude model) */
/*
*****/
void init_var(void)
{
        register int l,m,q;
        register float *ptr0;

/* initialize the global variables */
        for(m=0; m< M*NA; m++)    A[0][m]=0;
        for(m=0; m< M*M*Nb; m++)  B[0][0][m]=0;

/* clear each circular addressing buffers */
        ptr0 =(float*) BUFFER_ADDR;
        for(m=0; m< (1+M+M+L+L*M+L*M*M)*BUFFER_SIZE; m++) *(ptr0++)=0;

/* assign address for each circular addressing buffer block */

```

```

    ptr0 =(float*) BUFFER_ADDR;
    ptrx = ptr0; ptr0 += BUFFER_SIZE;
    for(m=0; m<M; m++) { ptru[m] = ptr0; ptr0 += BUFFER_SIZE;}
    for(m=0; m<M; m++) { ptrv[m] = ptr0; ptr0 += BUFFER_SIZE;}
    for(l=0; l<L; l++) { ptrz[l] = ptr0; ptr0 += BUFFER_SIZE;}
    for(l=0; l<L; l++)
        for(m=0; m<M; m++)
            { ptrr[l][m]=ptr0; ptr0 += BUFFER_SIZE;}
    for(l=0; l<L; l++)
        for(m=0; m<M; m++)
            for(q=0; q<Q; q++)
                { ptrs[l][m][q]=ptr0; ptr0 += BUFFER_SIZE;}

} /* end of init_var() */

/*****
/*
/* Function: init_board0()
/* Purpose: Initialise the Multi-Channel Analog I/O Board.
/*
/*****
void init_board0(void)
{
/* Control Register - reset the board */
*CR = 0L;

/* Clear the DAC input latches */
*DAC0 = 0x00000000;
*DAC1 = 0x00000000;
*DAC2 = 0x00000000;
*DAC3 = 0x00000000;
*DAC4 = 0x00000000;
*DAC5 = 0x00000000;
*DAC6 = 0x00000000;
*DAC7 = 0x00000000;

/* Calculate the TIMER Values */
#define TVAL16 0x00010000 * (long) (10000000.0 / (float) FSAMPLE);
#define TVAL2 (long) (10000000.0 / (float) FSAMPLE);
*TIMER16 = TVAL16;
*TIMER2 = TVAL2;

/* Set up Programmable Gain Amps */
if (BOARD_GAIN == 1) *PGR = 0x00000000L; /* Gain = 1 */
if (BOARD_GAIN == 10) *PGR = 0x00550000L; /* Gain = 10 */
if (BOARD_GAIN == 100) *PGR = 0x00AA0000L; /* Gain = 100 */

/* Set up the Control Register for Interrupts */
*CR = 0x30010000; /* Enable interrupts - Level Trig */

} /* end of init_board0() */

/*****
/*
/* Function: c_int04() --- xLMS -----
/*
/* Excute either off-line identification or control
/*
/*****
void c_int04(void)

```

```

{
    register long m, l, output;
    register float x;
    register long *ADC, *DAC;
    float e[L], v[M], u[M], z[L], AD[16];

    ADC = (long*) DSPLINK + 0;
    DAC = (long*) DSPLINK + 16;

    /* get the new e & x */
    for(l=0; l<16; l++) {
        AD[l] = ((float)(*ADC++ >> 16))/SCALE_FACTOR;    /* error
sensor (accel.) */
    }

    e[0] = AD[0]+AD[1]+AD[2]+AD[3];
    e[1] = AD[4]+AD[5]+AD[6]+AD[7];
    e[2] = AD[8]+AD[9]+AD[10]+AD[11];
    x = AD[15];

    for(m=0; m<M; m++) {
        v[m]=0;
        u[m]=0;
    }

    if(count < ID_STEP) {          /*** process identification ***/
        bandrand(v); /* create new identif. signal v[0], v[1] */
        v[count*M/ID_STEP] = sin(2*PI*61*count/FSAMPLE)/3;
        identif(e, v, z);
    }
    else {                          /*** process control ***/
        control(x, e, u);
    }

    /* send u+v and z to the DAC port */
    for(m=0; m<M; m++) {
        output=(long)((u[m] + v[m])*SCALE_FACTOR);
        if(output > MAX_OUT) {output=MAX_OUT; converge=0; }
        if(output < -MAX_OUT) {output=-MAX_OUT; converge=0; }
        *DAC++=output << 16;
    }

    /* output for monitoring tye error */
    *DAC++=(long)(e[0]*SCALE_FACTOR)<<16;
    *DAC++=(long)(e[1]*SCALE_FACTOR)<<16;
    *DAC++=(long)(e[2]*SCALE_FACTOR)<<16;

    if(! converge) {
        for(m=0; m< M*NA; m++) {
            A[0][m] *= 0.98;
        }

        for(m=0; m< M*M*NB; m++) {
            B[0][0][m] *= 0.98;
        }

        converge=1; /* reset converge flag */
    }

    count++;
}

```

```
} /*** end of c_int04() ***/
```

```

/*****
/*
/* Function: identif()      -- identification, called in c_int04 --
/*
/* pass in: e[], v[];
/* pass out: z[]
/* modify: ptrv, *ptrv, C[], D[], ptrz, *ptrz
/*****
void identif(float e[], float v[], float z[])
{

```

```
    register int l, m, q;
```

```

/* (i_1) C=C-u*(z-e)*V
    for(l=0; l<L; l++)
        for(m=0; m<M; m++)
            UPDATE(C[l][m], ptrv[m], NC, MU1a*(*ptrz[l]-e[l]));

```

```

/*      D=D-u*(z-e)*Z
    for(l=0; l<L; l++)
        for(q=0; q<L; q++)
            UPDATE(D[l][q], ptrz[q], ND, MU1b*(*ptrz[l]-e[l]));

```

```

/* (i_2) renew V[]
    for(m=0; m<M; m++)
        ptrv[m]=renew_vect(ptrv[m], v[m]);

```

```

/* (i_3) z=C'V+D'Z
    for(l=0; l<L; l++) {
        z[l]=0;
        for(m=0; m<M; m++)
            z[l] += FIR(C[l][m], ptrv[m], NC);
        for(q=0; q<L; q++)
            z[l] += FIR(D[l][q], ptrz[q], ND);
    }

```

```

    for(l=0; l<L; l++)
        ptrz[l]=renew_vect(ptrz[l], z[l]);

```

```
}/*** end of identif() ***/
```

```

/*****
/*
/* Function: control()      --- control(), called in c_int04 --
/*
/* pass in: x, e[]; return: u[]
/* pass out: u[]
/* modify: ptrx, *ptrx, A[], B[], ptrr, *ptrr, ptrs, *ptrs, ptru, *ptru
/*****
void control(float x, float e[], float u[])
{

```

```

    register int l, m, p, q, new_offset;
    float r[L][M], s[L][M][M];

```

```

/* (c_1) renew X[]
    new_offset = (int) ptrx;
    ptrx = renew_vect(ptrx, x);

```

```

        new_offset = (int) ptrx - new_offset;

/* (c_2) r=C1'X+D1'R          */
    for(l=0; l<L; l++)
        for(m=0; m<M; m++) {
            r[l][m] = FIR(C_New[l][m], ptrx, NC);
            for(q=0; q<L; q++)
                r[l][m] += FIR(D_New[l][q], ptrr[q][m], ND);
        }

    for(m=0; m<L*M; m++) {
        ptrr[0][m] += new_offset;
        *ptrr[0][m] = r[0][m];
    }

/*      s=C1'U+D1'S          */
    for(l=0; l<L; l++)
        for(m=0; m<M; m++)
            for(p=0; p<M; p++) {
                s[l][m][p] = IIR(C_New[l][m], ptru[p], NC,
                                D_New[l][0], ptrs[0][m][p], ND);
                for(q=1; q<L; q++)
                    s[l][m][p] += FIR(D_New[l][q], ptrs[q][m][p], ND);
            }

    for(p=0; p<L*M*M; p++) {
        ptrs[0][0][p] += new_offset;
        *ptrs[0][0][p] = s[0][0][p];
    }

/* (c_3) A=A-u*e*R          */
    for(m=0; m<M; m++)
        for(l=0; l<L; l++)
            UPDATE(A[m], ptrr[l][m], NA, MU0a*e[l]);

/*      B=B-u*e*S          */
    for(m=0; m<M; m++)
        for(p=0; p<M; p++)
            for(l=0; l<L; l++)
                UPDATE(B[m][p], ptrs[l][m][p], NB, MU0b*e[l]);

/* (c_4) u=A'X+B'U          */
    for(m=0; m<M; m++) {
        u[m]=FIR(A[m], ptrx, NA);
        for(p=0; p<M; p++)
            u[m] += FIR(B[m][p], ptru[p], NB);
    }

    for(m=0; m<M; m++) {
        ptru[m] += new_offset;
        *ptru[m] = u[m];
    }

}**** end of control() ****/

/*****
/*
/* Function: void bandrand(float v[M])
/* creat bandpass random series
/* 5-order chebyl filter
*****/

```

MXLMS33.C

```

#define N_b 5 /* bandpass filter length */
void bandrand(float v[])
{
    register int i;
    register float v0;
    static float x[3*M][N_b+1];

    static float Fa1[] = { /* low-pass(cheby1)--75 Hz at S. Rate=2K
    */
        1.584668002951872e-005,
        7.923340014759361e-005,
        1.584668002951872e-004,
        1.584668002951872e-004,
        7.923340014848179e-005,
        1.584668002946321e-005};

    static float Fb1[] = {
        1.000000000000000e+000,
        -4.238025415356931e+000,
        7.234387754532702e+000,
        -6.212493503386992e+000,
        2.682125465757414e+000,
        -4.654872077852484e-001};

    static float Fa2[] = { /* high-pass(cheby1) 55-- Hz at S.
    Rate=2K */
        7.557783380035952e-001,
        -3.778891690017976e+000,
        7.557783380035954e+000,
        -7.557783380035952e+000,
        3.778891690017976e+000,
        -7.557783380035952e-001};

    static float Fb2[] = {
        1.000000000000000e+000,
        -4.441048699933009e+000,
        7.917234373734317e+000,
        -7.080144881564633e+000,
        3.175277964701176e+000,
        -5.712008961819109e-001};

    /*****
    if(!count) {
        for(i=0; i<3*M*(N_b+1); i++) {
            x[0][i]=0;
        }
    }

    for(i=0; i<M; i++) {
        v0 = ((float)rand()/RAND_MAX - 0.5);
        v[i] = bandp(v0, x[3*i], x[3*i+1], x[3*i+2], Fa1, Fb1, Fa2,
        Fb2, N_b);
    }

} /* end of bandrand() */

```


MXLMS412.C

```

/*****
/* MIMO filtered-x LMS
/*
/* This c code is used to implement filtered-x LMS algorithm
/* on QPC/40B board and PC/16IO8 board with off-line identification*/
/* for the secondary path.
/* masm.asm has to be linked.
/*
/* Written by: Weiping Xu
/* Carleton University
/* December 1997
/*
/*****
/* 12-input/4-output
/* IIR(ID.) /FIR(control)
/* Off-line identification
/* Simlified LMS
/*
/*****
#include "\dsptools\stdlib.h"
#include "math.h"
#include "intpt40.h" /* Interrupt support (Parallel Support Lib)
*/

/*****
/* control constants and global variables */
#define M 4 /* channel number of output */
#define L 12 /* ... error */
#define NA 15 /* length of filter A */
#define NC 15 /* ... C */
#define ND 14 /* ... D */
#define MU0a 0.001 /* converge coeff. of control path */
#define MU1a 0.001 /* ... identif. path */
#define MU1b 0.001 /* ... identif. path */
#define BOARD_GAIN 10
#define SCALE_FACTOR 2048 /* 12-bit scale factor */
#define FSAMPLE 1800 /* sampling rate in Hz */
#define ID_STEP (60*FSAMPLE) /* off-line identification step number */
#define CN_STEP (60*FSAMPLE) /* control step number */

#define BUFFER_ADDR 0x002ff900 /* first circular buffer location
*/
#define BUFFER_SIZE 16 /* circular buffer size */
#define PI 3.14159
#define MAX_OUT 1000 /* maximum output voltage =
MAX_OUT*10/2048 */

/*****
/* DSPLINK Addresses (1 ws access) */
#define DSPLINK 0xB0000100 /* 1 ws */

/* DSPLINK Addresses/Registers for PC/16IO8 now follow */
#define ADC0 (long*) (DSPLINK+0)
#define ADC1 (long*) (DSPLINK+1)
#define ADC2 (long*) (DSPLINK+2)
#define ADC3 (long*) (DSPLINK+3)
#define ADC4 (long*) (DSPLINK+4)
#define ADC5 (long*) (DSPLINK+5)
#define ADC6 (long*) (DSPLINK+6)
#define ADC7 (long*) (DSPLINK+7)
#define ADC8 (long*) (DSPLINK+8)

```

MXLMS412.C

```

#define ADC9      (long*) (DSPLINK+9)
#define ADC10     (long*) (DSPLINK+10)
#define ADC11     (long*) (DSPLINK+11)
#define ADC12     (long*) (DSPLINK+12)
#define ADC13     (long*) (DSPLINK+13)
#define ADC14     (long*) (DSPLINK+14)
#define ADC15     (long*) (DSPLINK+15)

#define DAC0      (long*) (DSPLINK+0x10)
#define DAC1      (long*) (DSPLINK+0x11)
#define DAC2      (long*) (DSPLINK+0x12)
#define DAC3      (long*) (DSPLINK+0x13)
#define DAC4      (long*) (DSPLINK+0x14)
#define DAC5      (long*) (DSPLINK+0x15)
#define DAC6      (long*) (DSPLINK+0x16)
#define DAC7      (long*) (DSPLINK+0x17)

#define CR ((unsigned long*) (DSPLINK+0x18))
#define SR ((unsigned long*) (DSPLINK+0x18))
#define TIMER16 ((unsigned long*) (DSPLINK+0x19))
#define TIMER2 ((unsigned long*) (DSPLINK+0x1A))
#define PGR ((unsigned long*) (DSPLINK+0x1B))

/*****
/* function prototypes *****/
void main(void);
void init_var(void);
void init_board0(void);
void c_int04(void);
float FIR(float*, float*, int);
float IIR(float*, float*, int, float*, float*, int);
void UPDATE(float*, float*, int, float);
float* renew_vect(float*, float);
void identif(float*, float*, float*);
void control(float, float*, float*);
void bandrand(float*);
float bandp(float, float*, float*, float*,
            float*, float*, float*, float*, int);

/*****
/* global variables */
float *ptrx; /* pointer to vector x */
float *ptru[M]; /* pointers to vectors u[M] */
float *ptrv[M]; /* pointers to vectors v[M] */
float *ptrz[L]; /* pointers to vectors z[L] */
float *ptrr[L][M]; /* pointers to vectors r[L][M] */
float A[M][NA]; /* adaptive filter A */
float C[L][M][NC], D[L][ND]; /* Model C & D */
#define C_New C /* C_New is the copy of model C */
#define D_New D /* D_New is the copy of model D */

volatile int count=0; /* step counter */
volatile int converge=1; /* converger flag (0 -- diverge) */
volatile int dummy;
int buffer_size=BUFFER_SIZE-1; /* buffer size */

/*****
/*
/* main()
/*

```

```

/*****/
void main(void)
{
    register int l,m,k;

    /* initialize the model */
    for(k=0; k< L*M*NC; k++) {
        C[0][0][k]=0;
    }

    for(k=0; k< L*ND; k++) {
        D[0][k]=0;
    }

    /***** copy model from the initial one if it is given *****/
    /*
        for(l=0; l< L; l++) {
            for(m=0; m< M; m++) {
                for(k=0; k< NC; k++) {
                    C[l][m][k]=C0[m][l][k];
                }
            }
        }

        for(l=0; l< L; l++) {
            for(k=0; k< ND; k++) {
                D[l][k]=-D0[l][k];
            }
        }
    */

    init_var();

    /*****/
    /* Set up the C40 interrupts */
    /*****/
    /* Need to perform an IACK instruction to allow external */
    /* interrupts through to the C40 (for DB40) */
    asm("    PUSH    AR0");
    asm("    PUSH    DP");
    asm("    LDI     030H, AR0");
    asm("    LSH     16, AR0");
    asm("    IACK    *AR0");
    asm("    POP     DP");
    asm("    POP     AR0");

    /* Set up the C40 processor */
    /* Global disable of interrupts */
    INT_DISABLE();

    /* Explicitly set IVTP on 512 word boundary */
    set_ivtp((void *) 0x00310000);

    /* Set Interrupt Vector for IIOF1 */
    install_int_vector((void *) c_int04, 0x04);

    /* Enable IIF01 to be level triggered int. */
    load_iif(0x00B0);

    /* Initialise the multi-channel board */
    init_board0();

```

MXLMS412.C

```

/* Clear pending ints and enable C40 Interrupts for DSPLINK */
dummy = *ADC0;
INT_ENABLE();

/* Turn ON the Cache */
CACHE_ON();

/*****
/* Process off-line identification */
*****/
/* Loop for interrupts -- */
while(count < ID_STEP);
INT_DISABLE();

asm("stage1 NOP");
asm(" .globl stage1");

/*****
/* Process control */
*****/
control:
/* Initialise the multi-channel board */
init_board0();

/* Clear pending ints and enable C40 Interrupts for DSPLINK */
dummy = *ADC0;
INT_ENABLE();

/* Loop for interrupts -- control */
while(count < (CN_STEP+ID_STEP));
INT_DISABLE();

asm("stage2 NOP");
asm(" .globl stage2");

count = ID_STEP;
init_var();
goto control;

} /* end of main() */

/*****
/*
Function: init_var()
Purpose: Initialise the global variables (exclude model)
*/
*****/
void init_var(void)
{
    register int l,m,k;
    register float *ptr0;

/* initialize the global variables */
for(k=0; k< M*NA; k++) {
    A[0][k]=0;
}

/* clear each circular addressing buffers */
ptr0 =(float*) BUFFER_ADDR;

```

MXLMS412.C

```

        for(k=0; k< (1+M+M+L+L*M)*BUFFER_SIZE; k++) {
            *(ptr0++)=0;
        }

/* assign address for each circular addressing buffer block */
    ptr0 =(float*) BUFFER_ADDR;
    ptrx = ptr0;

    ptr0 += BUFFER_SIZE;
    for(m=0; m<M; m++) {
        ptru[m] = ptr0;
        ptr0 += BUFFER_SIZE;
    }

    for(m=0; m<M; m++) {
        ptrv[m] = ptr0;
        ptr0 += BUFFER_SIZE;
    }

    for(l=0; l<L; l++) {
        ptrz[l] = ptr0;
        ptr0 += BUFFER_SIZE;
    }

    for(l=0; l<L; l++) {
        for(m=0; m<M; m++) {
            ptrr[l][m]=ptr0;
            ptr0 += BUFFER_SIZE;
        }
    }

} /* end of init_var() */

/*****
/*
/* Function: init_board0()
/* Purpose:  Initialise the Multi-Channel Analog I/O Board.
/*
/*****
void init_board0(void)
{
/* Control Register - reset the board */
    *CR    = 0L;

/* Clear the DAC input latches */
    *DAC0 = 0x00000000;
    *DAC1 = 0x00000000;
    *DAC2 = 0x00000000;
    *DAC3 = 0x00000000;
    *DAC4 = 0x00000000;
    *DAC5 = 0x00000000;
    *DAC6 = 0x00000000;
    *DAC7 = 0x00000000;

/* Calculate the TIMER Values */
#define TVAL16 0x00010000 * (long) (10000000.0 / (float) FSAMPLE);
#define TVAL2  (long) (10000000.0 / (float) FSAMPLE);
    *TIMER16 = TVAL16;
    *TIMER2  = TVAL2;

/* Set up Programmable Gain Amps */

```

MXLMS412.C

```

if (BOARD_GAIN == 1)    *PGR = 0x00000000L;        /* Gain = 1    */
if (BOARD_GAIN == 10)   *PGR = 0x00550000L;        /* Gain = 10   */
if (BOARD_GAIN == 100)  *PGR = 0x00AA0000L;        /* Gain = 100  */

/* Set up the Control Register for Interrupts */
*CR = 0x30010000;      /* Enable interrupts - Level Trig */
} /* end of init_board0() */

/*****
/*
/* Function: c_int04()      --- xLMS -----
/*
/* Excute either off-line identification or control
/*
*****/
void c_int04(void)
{
    register long m, l, output;
    register float x;
    register long *ADC, *DAC;
    float e[L], v[M], u[M], z[L], AD[16];

    ADC = (long*) DSPLINK + 0;
    DAC = (long*) DSPLINK + 16;

    /* get the new error & x */
    for(l=0; l<16; l++) {
        AD[l] = ((float) (*ADC++ >> 16))/SCALE_FACTOR;    /* error
sensor (accel.) */
    }

    for(l=0; l<L; l++) {
        e[l]=AD[l];
    }
    x=AD[15];

    for(m=0; m<M; m++) {
        v[m]=0;
        u[m]=0;
    }

    if(count < ID_STEP ) {          /*** process identification ***/
        bandrand(v);                /* create random ID signal v[M] */
        identif(e, v, z);
    }
    else {                          /*** process control ***/
        control(x, e, u);
    }

    /* send u+v to the DAC port */
    for(m=0; m<M; m++) {
        output=(long)((u[m] + v[m])*SCALE_FACTOR);
        if(output > MAX_OUT) {output=MAX_OUT; converge=0; }
        if(output < -MAX_OUT) {output=-MAX_OUT; converge=0; }
        *DAC++=output << 16;
    }

    if(! converge) {

```

```

        for(m=0; m< M*NA; m++) {
            A[0][m] *= 0.98;
        }
        converge=1; /* reset converge flag */
    }

    count++;
} /**** end of c_int04() ****/

/*****/
/*
/* Function: identif()      -- identification, called in c_int04 --      */
/*
/* pass in: e[], v[]
/* pass out: z[]
/* modify: ptrv, *ptrv, C[], D[], ptrz, *ptrz
/*****/
void identif(float e[], float v[], float z[])
{
    register int l, m;

    /* (i_1) C=C-u*(z-e)*V      */
    for(l=0; l<L; l++) {
        for(m=0; m<M; m++) {
            UPDATE(C[l][m], ptrv[m], NC, MU1a*(*ptrz[l]-e[l]));
        }
    }

    /*      D=D-u*(z-e)*Z      */
    for(l=0; l<L; l++) {
        UPDATE(D[l], ptrz[l], ND, MU1b*(*ptrz[l]-e[l]));
    }

    /* (i_2) renew V[]      */
    for(m=0; m<M; m++) {
        ptrv[m]=renew_vect(ptrv[m], v[m]);
    }

    /* (i_3) z=C'V+D'Z      */
    for(l=0; l<L; l++) {
        z[l] = FIR(D[l], ptrz[l], ND);
        for(m=0; m<M; m++) {
            z[l] += FIR(C[l][m], ptrv[m], NC);
        }
        ptrz[l]=renew_vect(ptrz[l], z[l]);
    }

} /**** end of identif() ****/

/*****/
/*
/* Function: control()      --- control(), called in c_int04 --      */
/*
/* pass in: x, e[]
/* pass out: u[]
/* modify: ptrx, *ptrx, A[], ptrr, *ptrr, ptru, *ptru
/*****/
void control(float x, float e[], float u[])

```

```

{
    register int l, m, new_offset;
    register float rlm;

/* (c_1) renew X[] */
    new_offset = (int) ptrx;
    ptrx = renew_vect(ptrx, x);
    new_offset = (int) ptrx - new_offset;

/* (c_2) r=C1'X+D1'R */
    for(l=0; l<L; l++) {
        for(m=0; m<M; m++) {
            rlm = IIR(C_New[l][m], ptrx, NC, D_New[l], ptrr[l][m], ND);
            ptrr[l][m] += new_offset;
            *ptrr[l][m] = rlm;
        }
    }

/* (c_3) A=A-u*e*R */
    for(m=0; m<M; m++) {
        for(l=0; l<L; l++) {
            UPDATE(A[m], ptrr[l][m], NA, MU0a*e[l]);
        }
    }

/* (c_4) u=A'X */
    for(m=0; m<M; m++) {
        u[m]=FIR(A[m], ptrx, NA);
        ptru[m] += new_offset;
        *ptru[m] = u[m];
    }

}/*** end of control() ***/

/*****
/*
/* Function: void bandrand(float v[M])
/* creat bandpass random series
/* 5-order cheby1 filter
*****/
#define N_b 5 /* bandpass filter length */
void bandrand(float v[])
{
    register int i;
    register float v0;
    static float x[3*M][N_b+1];

    static float Fa1[] = { /* low-pass(cheby1)--75 Hz at S. Rate=1.8
K */
        5.940759347566882e-006,
        2.970379673783441e-005,
        5.940759347389246e-005,
        5.940759347744518e-005,
        2.970379673605805e-005,
        5.940759348010971e-006};

    static float Fb1[] = {
        1.000000000000000e+000,
        -4.611431264562694e+000,
        8.593382924706830e+000,
        -8.084943286325208e+000,

```


MXLMS412.C

```

        3.838974682225546e+000,
        -7.357929517453515e-001};

static float Fa2[] = {          /* high-pass(cheby1) 55-- Hz at S.
Rate=1.8 K */
    6.742587469515223e-001,
    -3.371293734757613e+000,
    6.742587469515229e+000,
    -6.742587469515229e+000,
    3.371293734757615e+000,
    -6.742587469515232e-001};

static float Fb2[] = {
    1.000000000000000e+000,
    -4.228406763519255e+000,
    7.185226208367957e+000,
    -6.118445961615839e+000,
    2.603412815520780e+000,
    -4.407881534248717e-001};

/*****/
    if(!count) {
        for(i=0; i<3*M*(N_b+1); i++) {
            x[0][i]=0;
        }
    }

    for(i=0; i<M; i++) {
        v0 = ((float)rand()/RAND_MAX - 0.5);
        v[i] = bandp(v0, x[3*i], x[3*i+1], x[3*i+2], Fa1, Fb1, Fa2,
Fb2, N_b);
    }

} /* end of bandrand() */

```

MASM.ASM

```

*****
* MASM.ASM
*
* This is the assembly code for FIR/IIR filter,
* adaptive coefficient update and vector-renew.
* It's used to link to the MXLMS33.C or MXLMS412.C
*
* Written by: Weiping Xu
*             Carleton University
*             December 1997
*
*****
        .version      40
FP       .set          AR3
        .globl        _buffer_size

*****
*       TITLE FIR FILTER
*****
*       SUBROUTINE FIR
*
*       EQUATION:  $y(n) = A(0) * X(n) + A(1) * X(n-1) + \dots$ 
*                   $+ A(N-1) * X(n - (N-1))$ 
*
*       Buffer size of X = _buffer_size
*****
* float FIR(float *A, float *X, int NA)
*
*       ARGUMENT | FUNCTION
*       -----|-----
*       *A       | ADDRESS OF A(0)
*       *X       | ADDRESS OF X(n) (circular addressing)
*       NA       | LENGTH OF FILTER A (NA <= _buffer_size)
*       y(n)     | Return Value in R0
*****
        .globl        _FIR
_FIR:
        PUSH    FP
        LDI     SP, FP
        LDA     *-FP(2), AR0
        LDA     *-FP(3), AR1
        LDI     *-FP(4), RC
        LDI     @_buffer_size, BK

        SUBI    2, RC
        RPTBD   FIR01
* Initialize R0
        MPYF3   *AR0++(1), *AR1++(1)%, R0
        LDF     0.0, R2
        NOP

FIR01 MPYF3 *AR0++(1), *AR1++(1)%, R0
||      ADDF3   R0, R2, R2
        ADDF    R0, R2, R0      ; Return Value
FIR_return:
        LDI     *-FP(1), R1
        BD      R1
        LDI     *FP, FP
        NOP
        SUBI    2, SP
***      B      R1      ;BANCH OCCURS

```

MASM.ASM

```

*
*
* end FIR
*
*****
*      TITLE IIR FILTER
*****
*      SUBROUTINE IIR
*
*      EQUATION:  $y(n) = A(0) * X(n) + A(1) * X(n-1) + \dots$ 
*                   $+ A(N-1) * X(n-(N-1))$ 
*                   $+ B(1) * Y(n-1) + \dots$ 
*                   $+ B(N-1) * Y(n-(N-1))$ 
*
*      Buffer size of X&Y = _buffer_size
*****
* float IIR(float *A, float *X, int NA, float *B, float *Y, int NB)
*
*      ARGUMENT | FUNCTION
*      -----|-----
*      *A      | ADDRESS OF A(0)
*      *X      | ADDRESS OF X(n) (circular addressing)
*      NA      | LENGTH OF FILTER A (NA <= _buffer_size)
*      *B      | ADDRESS OF B(1)
*      *Y      | ADDRESS OF Y(n-1) (circular addressing)
*      NB      | LENGTH OF FILTER B (NB <= _buffer_size)
*      y(n)    | Return Value in R0
*****
      .globl      _IIR
_IIR:
      PUSH      FP
      LDI       SP,FP
      LDA       *-FP(2), AR0
      LDA       *-FP(3), AR1
      LDI       *-FP(4), RC
      LDI       @_buffer_size, BK

      SUBI      2,RC
      RPTBD     IIR01
* Initialize R0
      MPYF3     *AR0++(1), *AR1++(1)%, R0
      LDF       0.0, R2
      NOP

IIR01 MPYF3     *AR0++(1), *AR1++(1)%, R0
      ||
      ADDF3     R0,R2,R2
      ADDF      R0,R2

      LDA       *-FP(5), AR0
      LDA       *-FP(6), AR1
      LDI       *-FP(7), RC

      SUBI      2,RC
      RPTBD     IIR02
* Initialize R0
      MPYF3     *AR0++(1), *AR1++(1)%, R0
      NOP
      NOP

IIR02 MPYF3     *AR0++(1), *AR1++(1)%, R0

```

MASM.ASM

```

||      ADDF3 R0,R2,R2
        ADDF  R0,R2,R0      ; Return Value

IIR_return:
        LDI    *-FP(1),R1
        BD     R1
        LDI    *FP,FP
        NOP
        SUBI   2,SP
***      B      R1      ;BANCH OCCURS
*
*
* end IIR
*
*****
*      TITLE UPDATE quation
*****
*      SUBROUTINE UPDATE
*
*      A=A-k*X      *
*
*      Buffer size of X = _buffer_size
*****
* void UPDATE(float *A, float *X, int RC, float k)
*
*      ARGUMENT | FUNCTION
*      -----|-----
*      *A      | ADDRESS OF A(0)
*      *X      | ADDRESS OF X(0) (circular addressing)
*      RC      | LENGTH OF A (RC <= _buffer_size)
*      k       | k
*
*****
        .globl      _UPDATE
_UPDATE:
        PUSH  FP
        LDI   SP,FP
        LDA   *-FP(2), AR0
        LDA   *-FP(3), AR1
        LDI   *-FP(4), RC
        LDF   *-FP(5), R1
        LDI   @_buffer_size, BK

        SUBI  2,RC
        RPTBD UPD01
        MPYF3 R1, *AR1++(1)%, R0
        NOP
        NOP

        MPYF3 R1, *AR1++(1)%, R0
||      SUBF3 R0, *AR0, R2
UPD01   STF   R2, *AR0++
        SUBF3 R0, *AR0, R2
        STF   R2, *AR0++
UPDAT_return:
        LDI    *-FP(1),R1
        BD     R1
        LDI    *FP,FP
        NOP
        SUBI   2,SP

```

MASM.ASM

```

***      B      R1      ;BANCH OCCURS
*
* end UPDATE
*

```

```

*****
* FUNCTION DEF : _renew_vect
*      X=[x, X(1:n-1)]
*
*      Buffer size of X = _buffer_size
*****
* float* renew_vect(float *X, float x);
*
*      ARGUMENT | FUNCTION
*      -----|-----
*      *X      | ADDRESS OF X(0) (circular addressing)
*      x       | New x
*      *X      | Return Value in R0 (new address of X(0))
*****

```

```

      .globl      _renew_vect
_renew_vect:
      PUSH      FP
      LDI      SP,FP
      LDA      *-FP(2), AR0
      LDF      *-FP(3), R0
      LDI      @_buffer_size, BK
      LDF      *AR0--(1)%, R1      ; dummy read to decrease the AR0
      STF      R0, *AR0
      LDI      AR0, R0      ; Return Value
r_v_return:
      LDI      *-FP(1),R1
      BD      R1
      LDI      *FP,FP
      NOP
      SUBI     2,SP
***      B      R1      ;BANCH OCCURS
*
* end renew_vect
*

```

```

*****
* FUNCTION DEF : _bandp
*
* Function: bandpass filter
*      y0=ha[0]*x[n]+a[1]*x[n-1]+ ... + a[n]*x[0]
*           -b[1]*y[n] + ... + b[n]*y[1]
*
*      z0=ha[0]*y[n]+c[1]*y[n-1]+ ... + c[n]*y[0]
*           -d[1]*z[n] + ... + d[n]*z[1]
*
* pass in: x0;
* pass out: z0;
* modify: x[], y[], z[];
*****
* float bandp(float x0, float x[], float y[], float z[],
*           float a[], float b[], float c[], float d[], int n)
*
*      ARGUMENT | FUNCTION
*      -----|-----
*      x0      | x[n]

```

MASM.ASM

```

*          y0      | Return Value in R0
*****
                .globl      _bandp
_bandp:
        PUSH    FP
        LDI     SP,FP
        PUSH    AR4
        PUSH    AR5
        PUSH    AR6
        PUSH    AR7

        LDF     *-FP(2),R10
        LDI     *-FP(3),R3
        LDI     *-FP(4),R2
        LDA     *-FP(5),AR7
        LDA     *-FP(6),AR6
        LDA     *-FP(10),IR1
        CMPI    0,IR1
        BLE     L4
        LDA     R3,AR2
        ADDI    IR1,-1,RC
        RPTB    L25
        LDF     *+AR2(1),R9
L25:      STF     R9,*AR2++
L4:
        ADDI    IR1,R3,AR2
        STF     R10,*AR2
        MPYF    R10,*AR6,R9
        CMPI    0,IR1
        BLE     L12
        ADDI    IR1,R2,AR4
        LDA     *-FP(7),AR5
        ADDI    AR5,1,AR5
        SUBI    1,AR2,AR2
        ADDI    1,AR6,AR6
        ADDI    IR1,-1,RC
        RPTB    L24
        MPYF    *AR2--, *AR6++,R0
        ADDF    R0,R9
        MPYF    *AR4--, *AR5++,R0
L24:      SUBF    R0,R9
        CMPI    0,IR1
        BLE     L12
        LDA     R2,AR2
        ADDI    IR1,-1,RC
        RPTB    L23
        LDF     *+AR2(1),R10
L23:      STF     R10,*AR2++
L12:
        ADDI    IR1,R2,AR5
        STF     R9,*AR5
        LDA     *-FP(8),AR0
        MPYF    *AR0,R9
        CMPI    0,IR1
        BLE     L20
        ADDI    IR1,AR7,AR2
        LDA     *-FP(9),AR4
        ADDI    AR4,1,AR4
        LDA     *-FP(8),AR6
        ADDI    AR6,1,AR6
        SUBI    1,AR5,AR5

```

MASM.ASM

```

        ADDI    IR1,-1,RC
        RPTB    L22
        MPYF    *AR5--, *AR6++,R0
        ADDF    R0,R9
        MPYF    *AR2--, *AR4++,R0
L22:    SUBF    R0,R9
        CMPI    0,IR1
        BLE     L20
        LDA     AR7,AR2
        ADDI    IR1,-1,RC
        RPTB    L21
        LDF     *+AR2(1),R10
L21:    STF     R10,*AR2++
L20:
        STF     R9,*+AR7(IR1)
        LDF     R9,R0
bandp_return:
        LDI     *-FP(1),R1
        LDI     *FP,FP
        POP     AR7
        POP     AR6
        BD      R1
        POP     AR5
        POP     AR4
        SUBI    2,SP
***     B       R1      ;BRANCH OCCURS
*
* end bandp
*

        .end

```

IMAGE EVALUATION TEST TARGET (QA-3)

APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989

© 1993, Applied Image, Inc., All Rights Reserved