NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

HI FI AUDIO TAPE TO SUN WORKSTATION TRANSFER SYSTEM FOR DIGITAL AUDIO DATA

by

Arie Gal Gartenlaub

March, 1994

Thesis Advisor: Charles W. Therrien

• Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE March 1994	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE HI FI DIGITAL AUDIO TAPE TO SUN WORKSTATION TRANSFER SYSTEM FOR DIGITAL AUDIO DATA	5. FUNDING NUMBERS
6. AUTHOR(S) Arie Gal Gartenlaub	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.	
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE A

13. ABSTRACT *(maximum 200 words)*

This thesis describes a subsystem developed to provide for the transfer of digital audio signals from a SUN SPARCstation 10 workstation to a digital audio tape (DAT) and vice versa. The new system expands the audio recording/reproduction options available in the laboratory by integrating an analog tape deck and a digital tape deck with the SUN workstation. The desired connection enables working with a larger audio bandwidth to achieve better audio performance and resolution in comparison to the present workstation audio capabilities. Performance measurements of the audio signal-to-noise ratio have shown an improvement of about 45 dB in the audio reproduction capability and about 35 dB in the audio recording capability. Total harmonic distortion for the new system is below the limit of the measuring instrumentation (less than 0.1%).

14. SUBJECT TERMS DAT, DIGITAL AUDIO TAPE, DIGITAL RECORDING, SUN WORKSTATION, AES/EBU, SPDIF, REAL-TIME UNIX APPLICATION.	15. NUMBER OF PAGES 133
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)

Prescribed by ANSI Std. 239-18

Approved for public release; distribution is unlimited.

HI FI DIGITAL AUDIO TAPE TO SUN WORKSTATION TRANSFER SYSTEM FOR DIGITAL AUDIO DATA

by

Arie Gal Gartenlaub
Lieutenant Commander, Israeli Navy
B.S., Technion Haifa, Israel, 1987

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
March 1994

Author: Arie Gal Gartenlaub
Arie Gal Gartenlaub

Approved by: Charles W. Therrien
Charles W. Therrien, Thesis advisor

Murali Tummala
Murali Tummala, Second reader

Michael A. Morgan
Michael A. Morgan, Chairman
Department of Electrical and Computer Engineering

ABSTRACT

This thesis describes a subsystem developed to provide for the transfer of digital audio signals from a SUN SPARCstation 10 workstation to a digital audio tape (DAT) and vice versa. The new system expands the audio recording/reproduction options available in the laboratory by integrating an analog tape deck and a digital tape deck with the SUN workstation. The desired connection enables working with a larger audio bandwidth to achieve better audio performance and resolution in comparison to the present workstation audio capabilities. Performance measurements of the audio signal-to-noise ratio have shown an improvement of about 45 dB in the audio reproduction capability and about 35 dB in the audio recording capability. Total harmonic distortion for the new system is below the limit of the measuring instrumentation (less than 0.1%).

TABLE OF CONTENTS

I. INTRODUCTION 1

A. SOUND PROCESSING/RECORDING AND THESIS GOAL 1

B. HUMAN PERCEPTION, HEARING AND SPEECH 2

1. Human hearing characteristics 3

2. Human speech 4

C. HI FI AUDIO SOURCES AND AUDIO USE 5

D. MAGNETIC RECORDING HISTORY REVIEW AND PRINCIPLES . 7

1. Magnetic recording historical review 7

2. Modern tape formats 8

E. DIGITAL AUDIO 9

1. Quantization and signal quality 9

2. Harmonic distortion due to quantization 13

II. BACKGROUND ON DAT AND STANDARDS OF DIGITAL AUDIO 16

A. S-DAT AND R-DAT 16

1. S-DAT 17

2. R-DAT 18

B. DIGITAL AUDIO TRANSMISSION FORMATS 21

C. AES/EBU AND SPDIF FORMATS 23

1. Subframe structure 24

2. The AES/EBU professional standard 25

3. The SPDIF consumer format 27

III. SYSTEM DESCRIPTION 29

A. GENERAL SYSTEM CONFIGURATION 29

1. System recording/reproduction options 31

B. THE INTERFACE HARDWARE 33

1. Interface board 33

2. AES/EBU Daughter Module 37

C. THE SYSTEM SOFTWARE 39

1. Main program "run_save" 43

2. Main program "run_play" 45

3. Data structure and double buffering method 47

4. Audio file structure 49

5. Subprogram "sp2ae_save" 50

6. Subprogram "sp2ae_play" and header inclusion 52

D. RETRIEVING AUDIO FILES INTO MATLAB 55

IV. PERFORMANCE TESTS 57

A. AUDIO REPRODUCTION PURITY PERFORMANCE 57

B. AUDIO RECORDING PERFORMANCE 63

C. SOFTWARE PERFORMANCE TESTS 72

V. CONCLUSION 74

APPENDIX A. OPERATING INSTRUCTIONS AND USER GUIDE 76

A. CAPTURE and DIGITIZATION OF AUDIO DATA 76

1. Hardware setting 76

a. Capturing from the DAT 76

b. Capturing sound from the analog cassette player 77

c. Speech capture using a microphone 77

2. Software operation 78

B. PLAYBACK OF AN AUDIO FILE AND RECORDING ON TAPE 79

1. Harware setting 79

a. Playback for monitoring setup 80

b. Playback and recording an audio segment on DAT 80

c. Recording on analog tape 81

d. DAT dubbing mode 81

2. Software operation 82

C. DUBBING FROM ONE TAPE TO THE OTHER AND RECORDING EXTERNAL ANALOG SOURCES 83

APPENDIX B. THE DAUGHTER MODULE REGISTERS 85

A. THE CONTROL AND CONFIGURATION REGISTERS 85

1. User control register 85

2. Configuration register 85

3. AMELIA control register 86

4. Interrupt register 87

B. STATUS REGISTERS 88

1. Status register I 88

2. Status register II 90

C. DATA REGISTERS 91

APPENDIX C. SOFTWARE LISTINGS 95

A. MAIN PROGRAM "run_save.c" 95

B. MAIN PROGRAM "run_play.c" 99

C. SUBPROGRAM "sp2ae_save.c" 104

D. SUBPROGRAM "sp2ae_play.c" 110

LIST OF REFERENCES 118

BIBLIOGRAPHY 119

INITIAL DISTRIBUTION LIST 120

LIST OF FIGURES

Figure 1 A simplified look at the human ear. [Ref. 1] 2

Figure 2 Quantization steps and PDF of the quantzation error. [Ref. 1] 11

Figure 3 S-DAT mechanism. [Ref. 1] 18

Figure 4 R-DAT; the cassette and the head. [Ref. 2] 19

Figure 5 Close look at the tracks and scan format of the R-DAT head. [Ref. 1] . 20

Figure 6 Specifications for various recording/playback modes of DAT. [Ref. 1] . 21

Figure 7 AES/EBU and SPDIF block format. [Ref. 12] 24

Figure 8 AES/EBU serial interface subframe format.[Ref. 12] 24

Figure 9 SPDIF C flag block format.[Ref. 1] 28

Figure 10 General block diagram of the system configuration. 30

Figure 11 Analog connections of the audio matrix. 32

Figure 12 The LSI interface board block diagram. [Ref. 10] 33

Figure 13 The Interface Memory Map (for 64K X 32 SRAM). [Ref. 10] 35

Figure 14 Main board data bus connection to the DM bus. 37

Figure 15 AES/EBU Daughter Module block diagram. [Ref. 12] 38

Figure 16 The software main block diagram. 41

Figure 17 Data flow stages between the DAT I/O port and the system disk. 42

Figure 18 Flow chart for the program "run_save." 44

Figure 19 Flow chart for the program "run_play." 46

Figure 20 Data base connected list and the unit structure. 47

Figure 21 Double buffering method block diagram. 49

Figure 22 Flow chart for program "sp2ae_save". 51

Figure 23 Flow chart for program "sp2ae_play." 52

Figure 24 The header signal as recorded on the start of an audio segment. 54

Figure 25 An audio segment captured using the 32-bit mode. 56

Figure 26 Output signal from the SUN speakerbox; the marker is on the signal. . 59

Figure 27 Output signal from the SUN speakerbox; the marker is set on a noise peak. 60

Figure 28 Spectrum of the 1 kHz sinusoidal output signal from the DAT. 60

Figure 29 Noise only output from the DAT with zero output signal. 61

Figure 30 Output signal from the SUN speakerbox; the marker is on the first order difference intermodulation product. 62

Figure 31 Spectrum of the output signal from the DAT. (The intermodulation products are under the measurement floor of the spectrum analyzer.) 63

Figure 32 FFT of the 1 kHz input signal through the SUN workstation microphone. 65

Figure 33 FFT of the 1 kHz signal recorded through the DAT input. 66

Figure 34 Photograph of the spectrum of the 1 kHz input signal to the DAT 67

Figure 35 FFT of the two-frequency signal recorded through the SUN microphone input. 68

Figure 36 FFT of the two-frequency signal recorded through the DAT input. . . . 69

Figure 37 Spectrum of the two-frequency input signal measured by the spectrum analyzer. 70

Figure 38 FFT of the two-frequency input signal recorded through the SUN microphone input. 71

Figure 39 Channel 0 and Channel 1 Input Data Registers. [Ref. 12] 92

Figure 40 Channel 0 and 1 Output Data Registers. [Ref. 12] 93

LIST OF TABLES

Table I TOTAL HARMONIC DISTORTION FOR VARIOUS VOLUME LEVELS AND QUANTIZATION WORD LENGTH. 15

Table II BYTE 0 OF THE *AES/EBU* STANDARD. [Ref. 1] 26

Table III *AMELIA* REGISTER MAP. 36

Table IV THE AUDIO FILE STRUCTURE. 50

Table V HEADER MESSEGE FORMAT. 53

Table VI AUDIO REPRODUCTION PERFORMANCE MEASUREMENTS . . . 58

Table VII AUDIO RECORDING PERFORMANCE MEASUREMENTS. 64

Table VIII *AMELIA* CONTROL REGISTER CONFIGURATION. 86

Table IX INTERRUPT REGISTER CONFIGURATION. 87

Table X STATUS REGISTER *I* CONFIGURATION. 89

Table XI STATUS REGISTER *II* CONFIGURATION. 90

Table XII 16 BIT MODE DATA REGISTER CONTENT 91

Table XIII CHANNEL REGISTER FLAGS 94

I. INTRODUCTION

This chapter presents information on human hearing and speech, high fidelity audio, a historical review of magnetic recording, and digital audio techniques. Motivation for this thesis is provided by discussing the benefits of enhanced audio accuracy.

A. SOUND PROCESSING/RECORDING AND THESIS GOAL

The digital signal processing laboratory (DSP lab) at the Naval Postgraduate School, Department of Electrical and Computer Engineering uses the SUN family of workstations based on the UNIX operating system. Many of the signals processed and analyzed are in the audio frequency range. The present system provides telephone line quality signal recording, storage and playback. That is, the audio signal processed using the above equipment is bandlimited to 4 kHz and quantized to 8 bits, thus limiting the processing accuracy. Some of the processing algorithms developed in the DSP lab require analysis by listening, and some of the audio signals processed have bandwidth wider then 4 kHz. The system developed in this thesis provides a way to record and process 16 bit digital audio with sampling rates up to 48 kHz so that the entire audible spectrum is covered. Using Digital Audio Tape (DAT) standards in conjunction with a professional quality DAT machine provides the highest possible audio quality. The sounds recorded on the DAT can be used also for demonstration and comparison when access to a workstation is not possible.

B. HUMAN PERCEPTION, HEARING AND SPEECH

When evaluating an audio system one should keep in mind that the system's main intent is to interface with the human ear and mind. Most of the desired parameters of any audio system will be derived from this fact.

Hearing is the second most important source of information (after vision) to the human brain. The bandwidth of audio perception is narrower then that of vision but is still wide enough to deliver large quantities of information. Hearing enables us to communicate, without even seeing one another. It enables recognition of persons we are talking to and grasping more information than is in the spoken words. We can often tell a person's intention by the intonation of his or her speech; the sound contains more information then if we had written down the words.

The hearing process takes advantage of mechanical media movements. Those movements, when they correspond to an intelligible message, convey audio information. The hearing process can be divided into several stages or levels. The first level is the

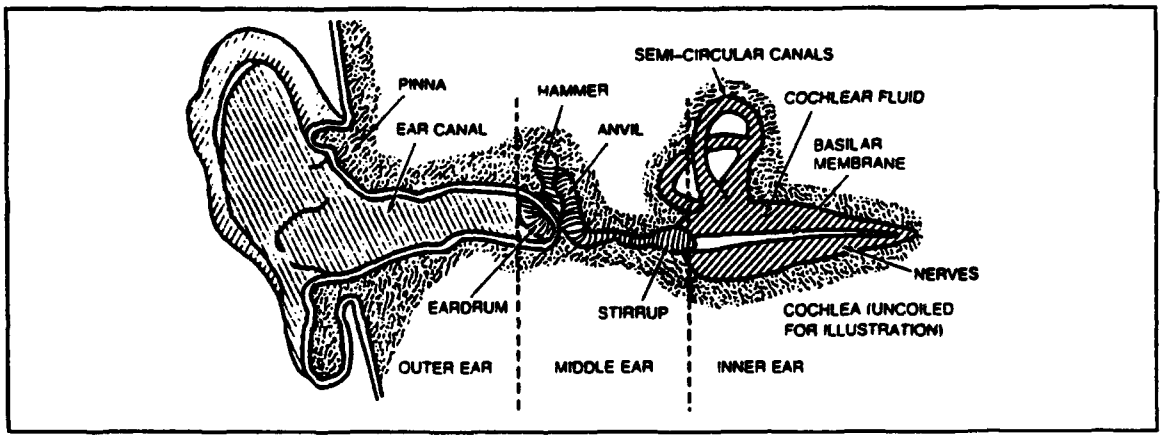


Figure 1 A simplified look at the human ear. [Ref. 1]

human ear which transduces the audio waves into electrical signals that are later sent to the brain via the nerve system. Figure 1 illustrates the construction of the human ear. The audio waves arriving at the ear are condensed through the ear canal to the eardrum. The waves are mechanically transferred to the middle ear and from there to the inner ear. The inner ear then translates the acoustic wave into electrical pulses that are sent to the brain. The cochlea located in the inner ear contains tiny hair-like receptors that resonate at different frequencies which overall comprise the entire audible spectrum.

1. Human hearing characteristics

The two most important parameters extracted from the audio information are the frequency and the amplitude of the sound, or the sound level. This is the information most needed to understand a vocal message. Although the human audible frequency spectrum is between 20 Hz and 20 kHz, sounds are not received uniformly in this range. The best human reception is in the 3 kHz to 4 kHz region.

Human amplitude reception capability spans a wider range than human frequency reception capability. In the frequency domain we are dealing with a ratio of 10^3 or 30 dB between the highest and lowest audible frequencies. In the amplitude domain the dynamic range (ratio between the hearing threshold and very high amplitudes that are still not harmful) is about 10^{12} or 120 dB. The least audible sound pressure level is about 0.0002 dyne/cm^2; this is considered to be the zero reference level or 0 dB Sound Pressure Level (SPL). With this reference a quiet home has 35 dB SPL, the Niagara falls have about 96 dB SPL, and the threshold of feeling pain is considered to be 140 dB SPL. The

pressure level is relative to the source power, and the power is relative to the square of voltages or currents. [Ref. 1]

Although frequency and amplitude as mentioned are the primary recognition parameters, if we want to add localization to the hearing process we need both of our ears. The human brain uses the time or phase difference between sounds to estimate the location of the sound source. Again, this recognition is not uniform across the entire audio spectrum, but is better at the higher frequencies.

2. Human speech

Whenever human hearing is discussed the production of human voices are mentioned as well. The voice production capabilities (dynamic range and frequency bandwidth) of human beings are generally reduced as compared with human hearing. The spectrum of regular speech does not usually extend above 4 kHz while singing might reach 10 kHz (including harmonics). Speech information is conveyed with very good comprehension through 4 kHz bandwidth channels, such as telephone lines.

Speech signals are comprised of voiced sounds (vowels) and unvoiced sounds (consonants). Voiced sounds are characterized by the pitch period ; voiced sounds are produced by motion of air through the vocal cords, which can be simulated by a pulse generator. The pulses are shaped and given their specific spectral content by the mouth-throat cavity. The unvoiced sounds are created by pressing air through the teeth, tongue and lips. Voice production levels (amplitudes) are also much more condensed in dynamic range than those of hearing.

C. HI FI AUDIO SOURCES AND AUDIO USE

Audio plays a great part in daily life. Audio and video systems are widespread these days and the two senses of hearing and vision are often connected. Audio use can be divided into some gross categories, namely

1. Individual communication,
2. Entertainment,
3. Professional civil use,
4. Military uses.

The first two categories are self explanatory; the other two categories include listening to medical equipment that uses audio, and military systems that have audio output, for example ECM, EW, and Sonar systems.

Many standards are associated with audio and audio transmission. As mentioned earlier telephone lines have a 4 kHz bandwidth. Radio broadcasts in the AM band have 5 kHz bandwidth, and FM radio broadcasts have 15 kHz bandwidth. The term Hi Fi itself is not a standard. Using the term "Hi Fi" or High Fidelity for describing a feature of an audio system indicates that the system reproduces the audible spectrum very accurately and with very low distortions. The multimedia encyclopedia [Ref. 4] defines high fidelity as follows:

> The term high fidelity (frequently shortened to hi-fi) has been in common usage since the 1950s and refers to the electronic reproduction of sound that corresponds closely to an original source or recording .... The ideal is to minimize unintentional inaccuracy or distortion by using a long series of recording and reproducing processes. The equipment used must have a wide frequency response: that is, the range of frequencies over which the signal is reproduced with minimal distortion must cover at least the range audible to the human ear, 50 to 15,000 hertz. The equipment is constantly improving: from the monophonic LPs of mid-century, to

STEREOPHONIC RECORDING, to the highly sophisticated digital processes available in HiFi VCRs, compact disc players, and DAT (digital audio tape) cassette decks.

In the above definition of High Fidelity the 50 Hz to 15 kHz bandwidth is stated, but good audio system specifications usually meet the complete 20 Hz to 20 Khz bandwidth. The larger the bandwidth of a signal (within the audio spectrum) the better it can be distinguished from a similar signal. For example, while communicating through a telephone at the 4 kHz bandwidth is sufficient for most purposes, when one wants to be certain about a certain word said the other person may be asked to spell the word. When one wants to classify sounds other than the human voice, for example musical instruments or sounds produced by underwater sonar monitoring equipment, it is even more important to have large audio bandwidth with low noise and distortions.

In consumer oriented audio systems the sound sources until recently have typically been analog devices such as (audio) cassette recorders, video cassette recorders (for video and audio signals), and LP records. For professional applications reel-to-reel magnetic tape has long been a standard because of its wide bandwidth, long play time, and multi channel capability. With the success of the Compact Disk (CD) and its sub-technologies (e.g., the Laser Video Disk) both consumer and professional segments are moving towards all-digital recording and reproduction methodology. This has led to the invention and subsequent success in the consumer market of the Digital Audio Tape (DAT).

D. MAGNETIC RECORDING HISTORY REVIEW AND PRINCIPLES

The magnetic recorder was the second tool developed to store and reproduce audio signals. The first was the gramophone which developed into the modern phonograph record player. This latter technology is slowly vanishing with the introduction and development of the CD. However magnetic recording is simply moving from analog to digital.

1. Magnetic recording historical review

Magnetic recording techniques became very popular in recent decades, mainly because of the ease of recording and reproduction of magnetic signals. This is true for audio signals as well as for digital data.

The principles of the magnetic field and the theory behind it were originated in Maxwell's laws and Faraday's reasearch. The first use of magnetic fields for signal recordings was described by Oberlin Smith in 1888. This was followed by the first practical patent issued to a Danish inventor named Valdemar Poulsen in 1898. Poulsen's device, called the Telegraphone, used a steel wire for recording. [Ref. 3][Ref. 4]

The sound quality of the first wire recorders was inferior to cylinder and disk records. A few experimental recorders started to use 1/2 inch wide steel tape instead of the wire during the 1920s and 1930s but not with a lot of success due to the fact that the tape was very heavy and costly. In 1928 a German patent was issued for a light weight paper tape coated with iron powder. This tape provided superior qualities over the all-steel tape. AEG Telefunken developed the Magnetophone (an early version of the audio tape) while another German company, BASF, worked on the tape. The paper tape was replaced

by a cellulose film and the iron powder was replaced by iron oxide. The Magnetophone sound quality was not very high; it was most adequate for speech but not for music.

The next improvment was the introduction of the "AC bias". This was first disovered in the US in 1927 but was not incorporated in the Magnetophone untill 1939. This AC bias was the key to high fidelity magnetic recording. The ultrasonic bias overcomes the nonlinearity in recording and reproduction of signals due to the hysteresis characteristics of the magnetic substance. In the US, Ampex Corporation was one of the leading companies in the magnetic recordering field. The Ampex machines started with a tape speed of 30 inches per second. This speed repeatedly halved with equalization circuit improvement to 15, 7-1/2, 3-3/4 and finally 1-7/8 inches per second. The last tape speed is a valid standard today for cassette tape recorders while professional studio units continue to use reel-to-reel systems at 7-1/2 or 15 inches per second. Additionally development of smaller magnetic heads enabled an increase in the number of tracks recorded on the tape. Cassette tapes contain two tracks in each direction, while wide reel-to-reel studio tapes can contain as many as 48 tracks. [Ref. 4]

2. Modern tape formats

In reel-to-reel tape technology the tape is spooled off a supply reel and rewound on a takeup reel after passing the record/playback heads. During the 50's and 60's many attempts were made to enclose the supply and takeup reel into a single enclosure. Philips introduced the compact cassette in 1964, which since became an international standard and a very successfull one. The cassette invention led to the development of car audio tape players, "walkman" and portable "boom-boxes" all through

the 1970's and 80's. By 1983 the compact cassette was the most successfull and popular medium for recording music. The compact cassette concept was then carried over to the micro-cassette, the VCR cassette, and finally to the DAT cassette.[Ref. 4]

E. DIGITAL AUDIO

When digital techniques were developed they were adopted for audio signals as well as for other fields. This led to a whole line of digital audio formats that differ in bandwidth (sample rates) and quantization. The telephone line standard remains at 4 kHz as it was developed before the digital audio era, and the quantization was set to 8 bit PCM. This standard is known to provide sufficiently good human voice quality and at the same time is economical. High Fidelity digital audio standards are much more elaborate than the telephone digital audio standard. The bandwidth is set at 20 Hz to 20 kHz, which requires at least a 40 kHz sample rate (according to the Nyquist theorem); it provides for two channels (stereo); and in order to get good signal to noise ratio and dynamic range it has 16-bit quantization. The CD sampling frequency is set to 44.1 kHz while the standard sampling frequencies for DAT are 48 kHz in the normal mode and 32 kHz in the long play mode. A rate of 44.1 kHz is also provided for direct digital compatability with CD.

1. Quantization and signal quality

To record and store an analog signal digitally an Analog to Digital Converter (ADC) is used. In the reverse process a Digital to Analog Converter (DAC) is used to

regenerate the analog signal. The ADC converts the analog signal into discrete levels that are represented by a binary word.

The number of bits in the digital word dictates the number of discrete levels the continuous signal is divided into. The number of levels L is given by

$$L = 2^n \tag{1}$$

where n is the number of bits in the sampled word. The value Q of a quantization step is given by

$$Q = \frac{V_{max} - V_{min}}{L} \tag{2}$$

where V_{max} and V_{min} here are assumed the largest and the smallest voltage level, respectively, to be converted. The least significant bit determines the accuracy of the quantization and its value is equal to the quantization step Q. The quantization error is the difference between the actual value of the signal and the value that is represented by the binary word. For linear quantization, used in most 16-bit audio devices the quantization error is uniform through all the dynamic range and is given by

$$E_q = \frac{1}{2}Q \, . \tag{3}$$

The best signal to quantization error performance of an n-bit system would then be

$$\frac{V_{PTP}}{E_q} = \frac{LQ}{\frac{1}{2}Q} = \frac{2^n}{\frac{1}{2}} = 2^{n+1} \tag{4}$$

where V_{PTP} is the peak-to-peak value denoted earlier by: $V_{max} - V_{min}$. For a 16-bit system this becomes 2^{17} or about 98 dB. The distribution of error assuming that the signal is totally uncorrelated with the sampling process can be thought to be uniform having the PDF shown in Figure 2.

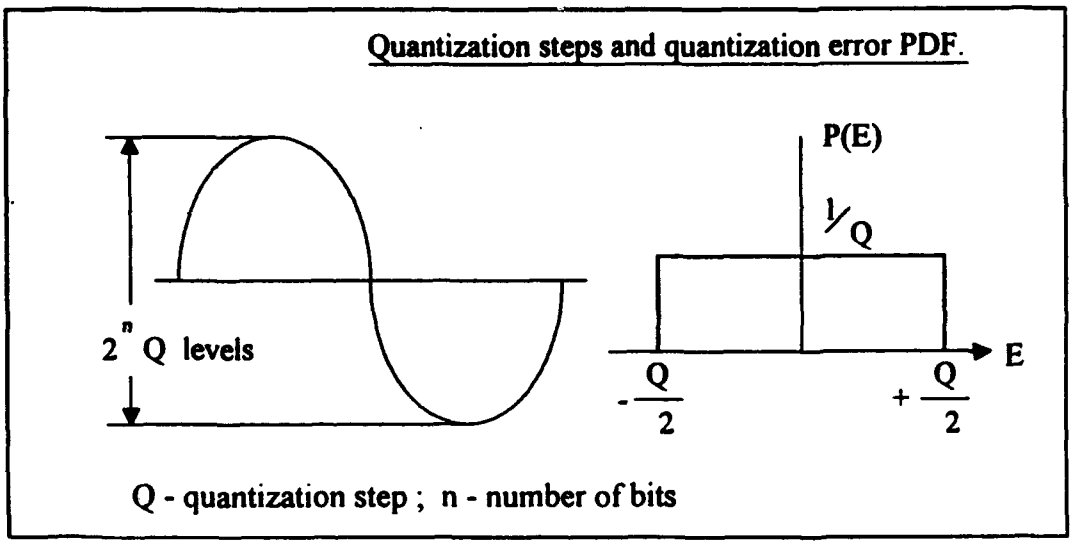


Figure 2 Quantization steps and PDF of the quantzation error. [Ref. 1]

The estimated error energy is therefore

$$\overline{E_q^2} = \int_{-\frac{Q}{2}}^{\frac{Q}{2}} E^2 P(E) dE = \frac{Q^2}{12} \quad . \tag{5}$$

The mean error voltage would then be

$$\overline{V_q} = \sqrt{\langle E_q^2 \rangle} = \frac{Q}{\sqrt{12}} \quad . \tag{6}$$

Using a sine wave signal as reference we get

$$V_{RMS} = \frac{LQ}{2\sqrt{2}} = \frac{2^n Q}{2\sqrt{2}} = \frac{2^{n-1} Q}{\sqrt{2}} \quad , \tag{7}$$

and finally the signal to quantization error ratio would be

$$\frac{V_{RMS}}{\overline{V_q}} = \frac{\frac{2^{n-1} Q}{\sqrt{2}}}{\frac{Q}{\sqrt{12}}} = 2^n \sqrt{1.5} = 6.02n + 1.76 \quad [dB] \quad . \tag{8}$$

For example, use of 16 bits yields about 98 dB SNR, 12 bits yields 74 dB and 8 bits yields only 49.9 dB. Another important fact that should be noticed is that the result is

computed for full scale signals. For a signal smaller in amplitude than V_{RMS} the SNR achieved would be worse.

When a small number of bits is used in the ADC, audio systems frequently use a nonlinear encoding scheme to improve the signal to quantization error performance. For example human speech is usually low in amplitude for the unvoiced sounds and much louder for the voiced ones. Common quantization schemes such as A-law or μ-law attempt to exploit these characteristics to improve the system signal-to-noise ratio performance. In these methods the quantization is finer near the zero amplitude level and larger for large amplitudes. A disadvantage that occurs is the fact that this is not a linear process. Thus if it is not perfectly compensated for in the DAC, it leads to undesired distortion that was not present in the original signal.

2. Harmonic distortion due to quantization

The Total Harmonic Distortion (THD) is a figure of merit that measures the energy of the harmonics created by nonlinearities in the system relative to the original signal energy. For a full scale signal the equations and results achieved in section E.1 above are valid, but for smaller signals the calculation and the results are different.

The full scale signal will be considered as the 0 dB reference level (having peak to peak unit amplitude). The peak to peak amplitude of the signal in dB is

$$V_{dB} = 20\log_{10}(V_{signal}) \qquad (9)$$

or

$$V_{signal} = 10^{\frac{V_{dB}}{20}} = IQ \tag{10}$$

Where Q is the quantization step and I is number of steps. The power of the signal assuming a sinusoid is

$$P_{signal} = \frac{V_{signal}^2}{8} = \frac{I^2Q^2}{8} \quad . \tag{11}$$

The noise power as before is

$$P_{noise} = \frac{Q^2}{12} \quad . \tag{12}$$

THD (as a percentage) is then given by

$$THD\% = \frac{P_{noise}}{P_{signal}} \times 100 = \frac{\frac{Q^2}{12}}{\frac{I^2Q^2}{8}} \times 100 = \frac{2}{3I^2} \times 100 \quad . \tag{13}$$

Finally THD as a function of the input peak-to-peak voltage in dB is

$$THD\% = \frac{2}{3(2^n 10^{\frac{V_{dB}}{20}})} \times 100 \quad . \tag{14}$$

Some numerical values for a linear quantization scheme are given in Table I below. The THD values for 8-bit A-law or μ-law are different due to the nonlinear quantization scheme, The THD is higher in the high amplitude levels and lower in the low amplitude levels (this is later supported by the measurements described in chapter IV).

Table I TOTAL HARMONIC DISTORTION FOR VARIOUS VOLUME LEVELS AND QUANTIZATION WORD LENGTH.

Volume level [dB]	THD %		
	8 - bits	12 - bits	16 - bits
0	0.2	0.016	0.001
-20	2.6	0.16	0.01
-40	26	1.6	0.1

II. BACKGROUND ON DAT AND STANDARDS OF DIGITAL AUDIO

The DAT or digital audio tape is the result of a mixture of several technologies. The evolution of digital audio, the development of better media for storage of digital data, development of better and more capable audio processing chips that can operate in real time at data rates desired for High Fidelity audio use, and finally the VCR technology were all very important to DAT development. The techniques of storing large quantities of digital audio that were first developed for the world wide consumer audio market were quickly adopted by the computer world because of their inherent inexpensive large scale storage capacity. For example a small DAT cassette is capable of storing about 1.38 Gigabytes and a CD can store about 0.5 Gigabyte of digital data. [Ref. 2] [Ref. 9]

A. S-DAT AND R-DAT

In order to store two channels of High Fidelity digital audio with 16 bit accuracy using a maximum sampling frequency of 48 kHz, the net rate of the recording is required to be

$$Bit_{rate} = 48\times10^{3}\times2\times16 = 1.536\times10^{6} \quad bits/s \tag{16}$$

The demand for large recording bandwidth is obvious when one considers the net bitstream. Even with the use of efficient modulation schemes having bit rates less then 1 Hz/Bit/sec, the required bandwidth is on the order of 1 MHz. The large bandwidth

required can be met by one of two techniques. The first, which is borrowed from the VCR technology is to increase the head-to-tape relative velocity in order to gain the desired bandwidth; the second is to divide the high rate data stream into streams of slower rates each of which feeds to an individual track of a multitrack recording head. The first technique is called R-DAT or Rotary scan head DAT; the second is called S-DAT or Stationary scan head DAT. Both techniques have been standardized and are available on the High Fidelity audio consumer market. There are advantages and disadvantages to each technology. These technologies are described briefly below.

1. S-DAT

The Stationary head DAT is based on principles similar to those of Frequency Division Multiplexing (FDM), where many narrowband channels are delivered through a wideband channel by modulating them on subcarrier frequencies. In the S-DAT the wideband data rate is divided into discrete channels each having a proportional bandwidth. For example if we have a channel with a data rate of 1.536×10^6 bits/s (as computed above) and we have a 20 track S-DAT, each individual channel will support $1.536 \times 10^6/20 = 76.8$ *Kbits/s*. This rate is manageable by a stationary head, and this rate is about the same as the bias frequency that is used in analog tape. Linearity is not kept because the data is binary and only saturation conditions are used. Figure 3 illustrates the S_DAT mechanism. An advantage of the S-DAT is the ability to record and play on a regular cassette tape; this was the motivation for developing the S-DAT technology. Another advantage has to do with the fact that since the head is not moving the mechanism and

control are slightly easier to manufacture. Among the disadvantages is the fact that S-DAT is more vulnerable to crosstalk and noise.

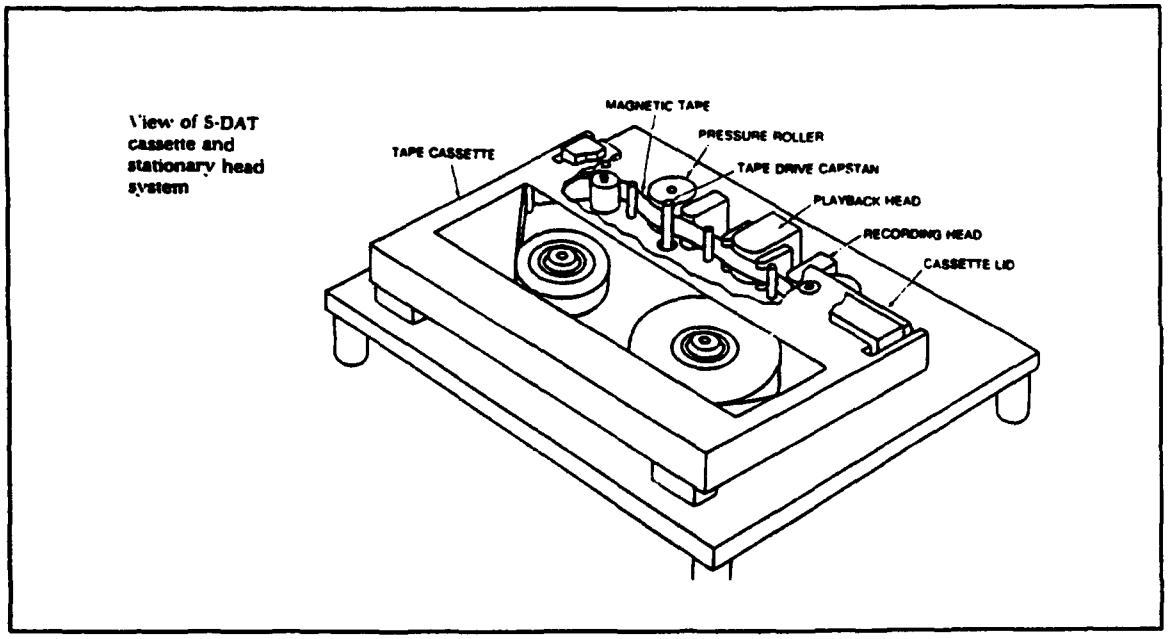


Figure 3 S-DAT mechanism. [Ref. 1]

2. R-DAT

The R-DAT uses VCR scanning head technology. The R-DAT is the more common type in the professional and high end consumer markets and is the technology used for this thesis. We will hereafter use the term DAT to refer to the R-DAT. The advantage of using the rotary head technology was not in its simplicity but rather in the fact that it was well understood and in wide use for VCR's. Thus there had been a lot of savings in mechanical development. The DAT head has a diameter of 30 mm, and revolves at 2000 RPM. There are 2 or 4 heads in a DAT recorder. The 2-head DAT was the first to be used; the 4 head was developed later and permits better reading and error proofing of the digital data.

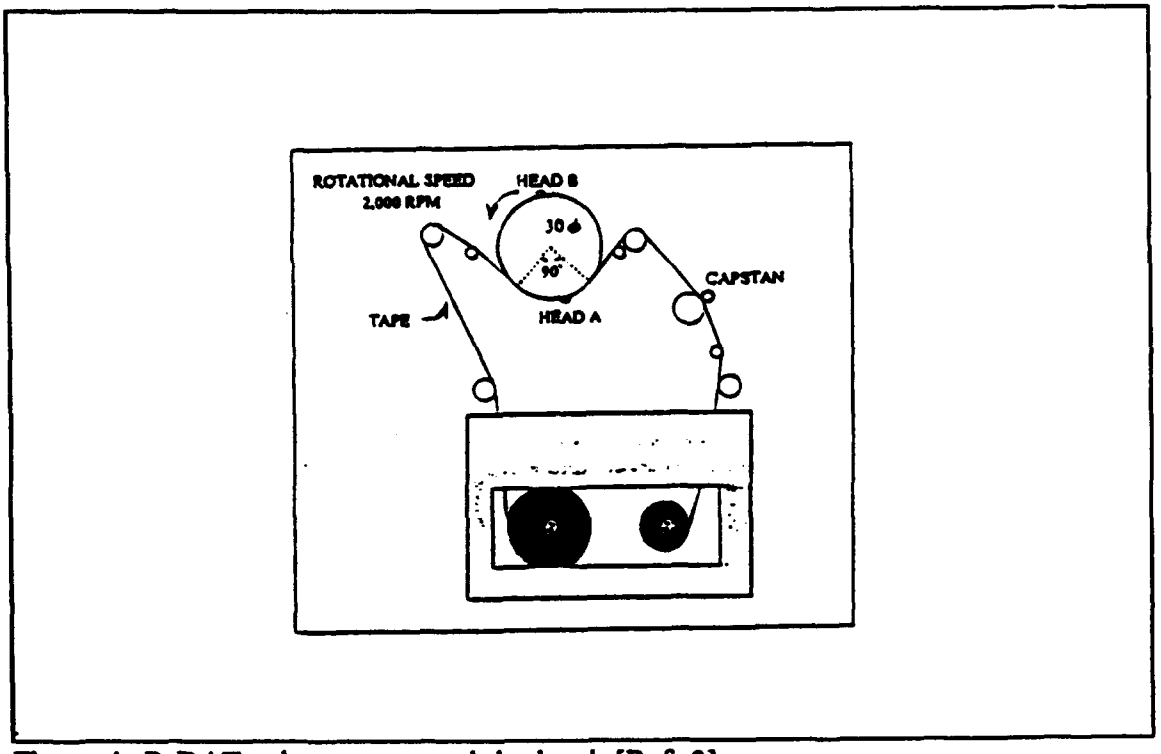


Figure 4 R-DAT; the cassette and the head. [Ref. 2]

Figure 4 shows the rotating head of the DAT, and the tape wrapped around it. The DAT's cassette is sealed and is 73 mm x 54 mm x 10.5 mm in size. Thus it is smaller then an 8 mm video camcorder cassette.

Figure 5 shows the track pattern of the DAT, head and the tilt angle of the head in relation to the track. On each track, which is 13.591 μm wide, there are some special areas. The subcode is on a part of the track which the DAT uses to record its own auxiliary data such as music ID, fast search codes, and various other information. The ATF or Automatic Track Finding area is used to synchronize the Phase Lock Loop (PLL)

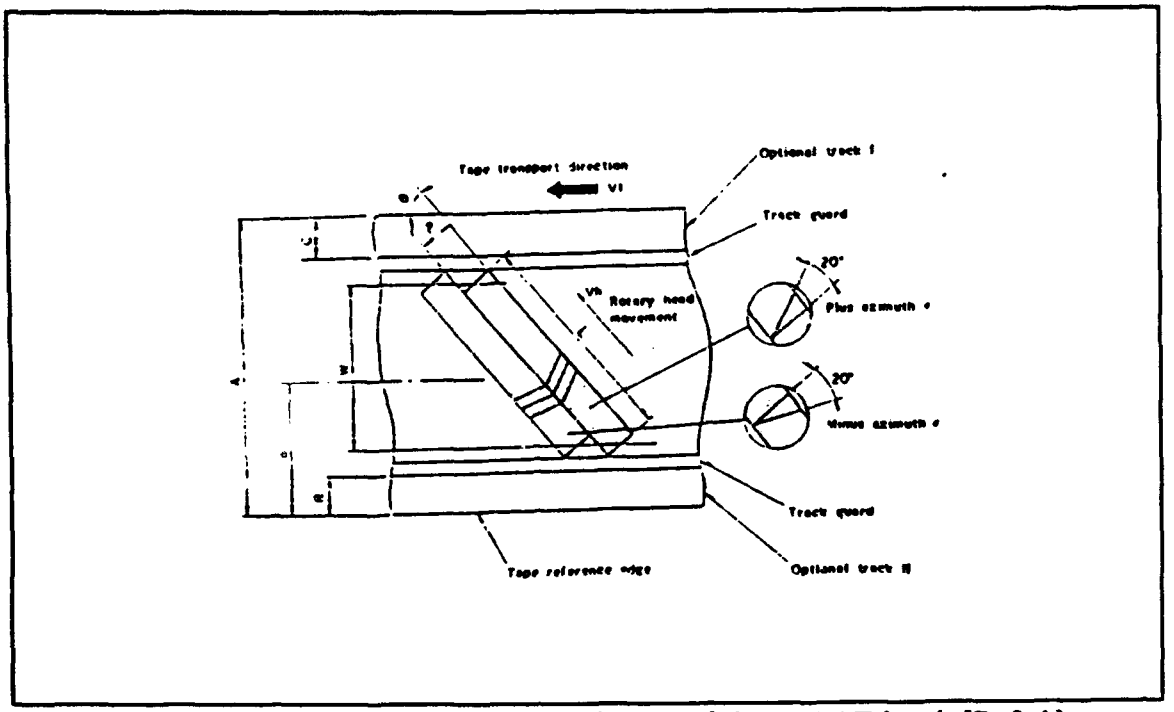


Figure 5 Close look at the tracks and scan format of the R-DAT head. [Ref. 1]

control of the head servo in order to lock very precisely on the track. The main area of the tape is used for sound recording. Due to the fact that recording is not continuous in time, storage and buffering for data compression and expansion is used.

Figure 6 gives some typical DAT specifications like sampling rates, data transfer rate, modulation scheme, error correction type, and record time. The modulation scheme used is called "8 to 10" since each byte is represented by 10 bits on the tape. This spreading reduces the DC recorded on the tape. For example if there is a long series of 1's in the audio sample this modulation scheme will break it into interlaced 1's and 0's and thus reduces the DC recorded on the tape. This technique does not have to be used in analog recording because the average of an analog signal recorded with the high frequency bias is zero.

ITEM \ MODE	DAT (REC PB MODE)				PRERECORDED TAPE (PB ONLY)	
	STANDARD	OPTION 1	OPTION 2	OPTION 3	NORMAL TRACK	WIDE TRACK
CHANNEL NUMBER (CH)	2	2	2	4	2	2
SAMPLING FREQUENCY (kHz)	48	32	32	32	44.1	
QUANTIZATION BIT NUMBER (BIT)	16 (LINEAR)	16 (LINEAR)	12 (NONLINEAR)	12 (NONLINEAR)	16 (LINEAR)	16 (LINEAR)
LINEAR RECORDING DENSITY (kBPI)	61.0	61.0			61.0	61.1
SURFACE RECORDING DENSITY (MBPI²)	114	114			114	76
TRANSMISSION RATE (MBPS)	2.46	2.46	1.23	2.46	2.46	
SUBCODE CAPACITY (KBPS)	273.1	273.1	136.5	273.1	273.1	
MODULATION SYSTEM	8-10 CONVERSION					
CORRECTION SYSTEM	DOUBLE REED-SOLOMON CODE					
TRACKING SYSTEM	AREA SHARING ATF					
CASSETTE SIZE (mm)	73 × 54 × 10.5					
RECORDING TIME (MIN)	120	120	240	120	120	80
TAPE WIDTH (mm)	3.81					
TAPE TYPE	METAL POWDER					OXIDE TAPE
TAPE THICKNESS (μm)	13 ± 1 μ					
TAPE SPEED (mm/s)	8.15	8.15	4.075	8.15	8.15	12.225
TRACK PITCH (μm)	13.591				13.591	20.41
TRACK ANGLE	6° 22'39.5"					6°23'29.4"
STANDARD DRUM SPECIFICATIONS	φ30 90° WRAP					
DRUM ROTATIONS (rpm)	2000		1000	2000	2000	
RELATIVE SPEED (m/s)	3.133		1.567	3.133	3.133	3.129
HEAD AZIMUTH ANGLE	± 20°					

Figure 6 Specifications for various recording/playback modes of DAT. [Ref. 1]

Correction codes are use to ensure high quality and reliability of the signal reproduction. As stated earlier the data is recorded very densely on the tape, and the crude Bit Error Rate (BER) is rather high. This is overcome by using error correction coding however.

B. DIGITAL AUDIO TRANSMISSION FORMATS

The development of the digital recorder introduced the need to develop data transfer protocols that would maintain the digital advantage of immunity to noise. These protocols for example enable one to record directly from the optical output port of a CD player with the appropriate interface. To accomplish this and to enable a variety of additional data to be transferred, the protocols developed contain auxiliary information, such as sample rate, whether preemphasis is used on the audio signal and the preemphasis characteristics,

copy prohibition codes, indexing of music for fast search, and even space for user defined data.

With the possibility of duplicating music with the highest possible quality the issue of copyright infringement arose. Actually, the issue arises also for analog recordings but there is little that can be done to prevent copying of music recorded in analog form. However the manufacturers realized that with the all-digital recording scheme, they could actually *inhibit* copying of music digitally. Thus a copy inhibition feature was written into the (consumer) standard.

Although several standards have been developed only two will be discussed in detail here. These are the SPDIF (Sony Philips Digital Interface Format) used for the domestic consumer machines and the AES/EBU format used in professional applications. AES stands for Audio Engineering Society, and EBU stands for European Broadcasting Union, the two organizations that created the professional DAT standard for audio studio and broadcasting station machines. Other audio protocols that will not be further discussed here include the SDIF-2 also developed by Sony, PD by Mitubishi, MADI (an extension of AES/EBU) for multiple channels of audio, DBS for Direct Satellite Broadcast and CADA, a standard for Cable Digital Audio. In all of these systems the digital audio is sent serially over a net or from one machine to another. It should be noticed that the structure of the data sent that way is different from the structure of the data actually recorded on the tape. Every machine or interface that conforms to one of the standards mentioned above can manipulate the data in any desired fashion internally but has to adjust to the correct configuration on transmission or reception. All the error codes and

control mentioned above (such as the Reed-Solomon codes, data interleaving, ATF and Sub codes) are internal and in most cases this information is not available outside the DAT. Internal standards were set to make the recorded cassettes transportable between different manufacturers.

C. AES/EBU AND SPDIF FORMATS

In both the AES/EBU and SPDIF formats the audio samples are organized into blocks. Each audio sample consists of 32 bits taken together, called a *subframe*. Two audio samples, corresponding to left and right channels in the stereo case, comprises a *frame*. Then 192 frames are gathered into a *block*. The beginning of each block is marked by a unique preamble, and each channel has a separate preamble. The channels are denoted by A and B (rather then left and right): the block start preamble is denoted as Z; the preamble for channel A is denoted by X; and the preamble for channel B is denoted by Y. The first channel A subframe of a block starts with Z; the rest of the preambles are Y and X consecutively, until the 192nd frame; and then it repeats. Figure 7 shows the structure of the block and preambles. The overall bit rate of transmission is between 2.048 Mbit/sec for the 32 kHz sample rate and 3.072 Mbit/sec for 48 kHz. The detailed structure of the preamble differs for the AES/EBU and SPDIF formats. The two standards also differ in their hardware physical interface and connector type.

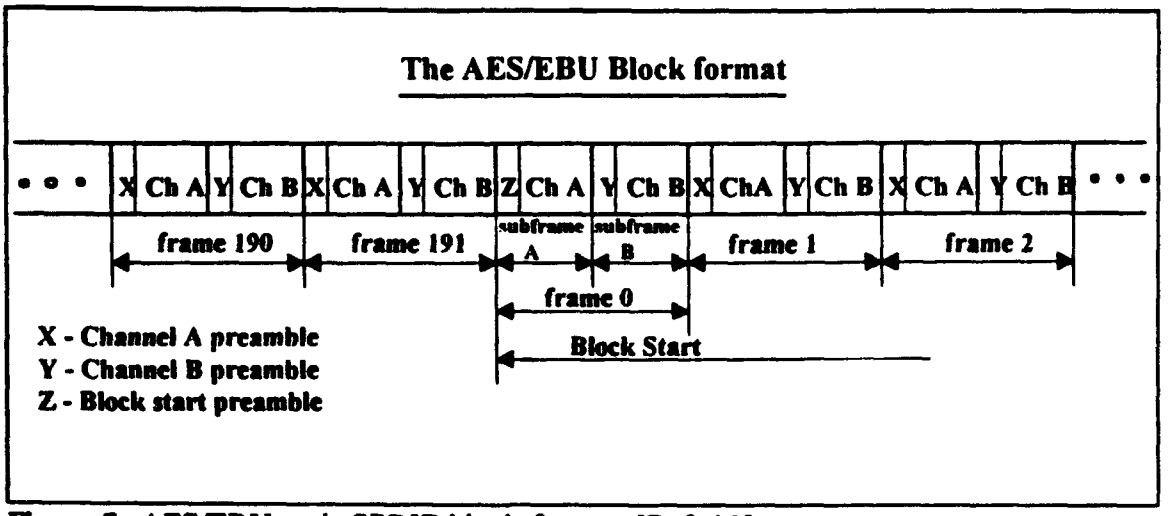


Figure 7 AES/EBU and SPDIF block format. [Ref. 12]

1. Subframe structure

Each audio sample has 32 bits. The subframe structure is shown in Figure 8. The first 4 bits, which comprise the preamble, are used for synchronization. (Since the data is sent on a single line the communication is asynchronous.) The next 4 bits carries auxiliary audio and other data; a specific interpretation is not provided by the standard.

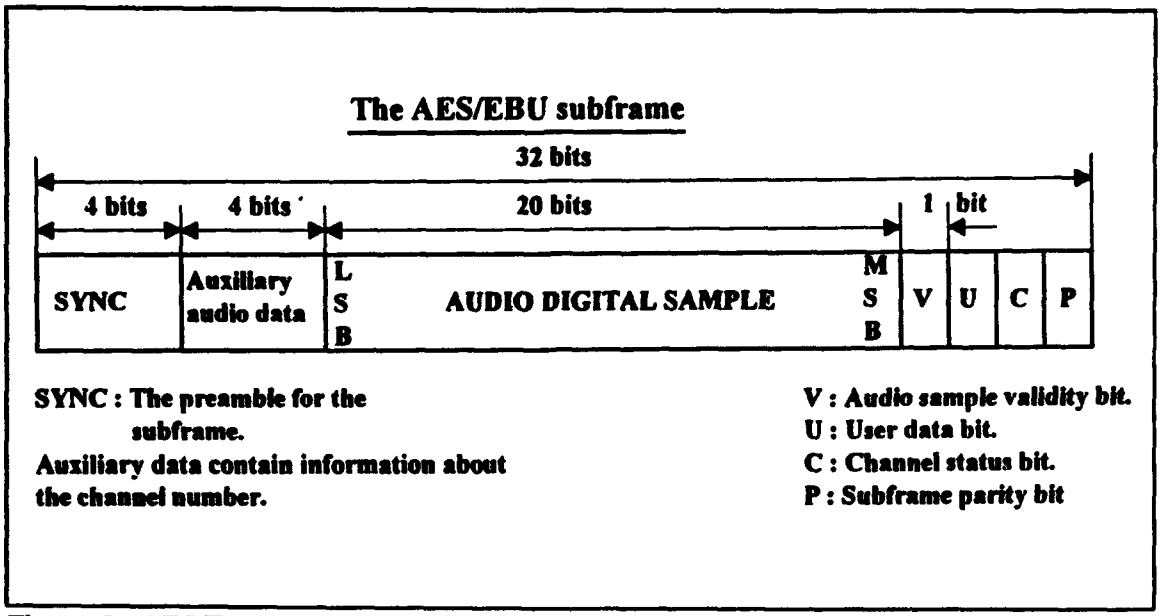


Figure 8 AES/EBU serial interface subframe format.[Ref. 12]

The audio sample itself can be a maximum of 20 bits long although there are applications that use 24 bits for the audio sample. In this case the auxiliary bits are used as part of the audio word. As mentioned earlier CD and DAT use 16-bit audio samples, so the standards support these formats easily.

The flags of the audio samples also convey important information. The most important is the C flag which delivers the channel information and is transmitted with every sample or subframe. The C flag is identical for the two subframes within the same frame. All the flags taken together in a block create the whole message which is 192 bits long and will be called the C-flag block. The 192 bits of the C-flag block conform to the standard in which they are delivered whether it is AES/EBU or SPDIF.

2. The AES/EBU professional standard

The AES/EBU codified as the ANSI S4.40-1985 is the professional standard for data exchange between professional audio devices. The AES/EBU provides for conveying two channels of periodically sampled and uniformly quantized audio signals on a single twisted wire pair. The format was designed to support data transmission up to 100 meters in distance.

The signal is a biphase self-clocking Manchester code, which saves the need for a separate line for clocking. The twisted-pair cable is designed to improve the noise immunity of the data. The hardware characteristics conform to the IEEE RS-422A standard. The AES dictates use of transformers for better isolation with a required impedance of 90 to 120 ohms and a voltage level of 3 to 10 volts peak-to-peak. The line

is actually a shielded twisted-pair cable with XLR type connector as used with other professional audio equipment; pin 1 is ground and pins 2 and 3 carry the signal.

The C flag block described earlier is organized into 24 8-bit bytes (192 bits total). The C flag block contains the information pertaining to the channel. The C flag block rate is about 250Hz. The most important bytes in the block are byte 0 and bytes 22 and 23. Byte 0 contains the information shown in Table II below.

Table II BYTE 0 OF THE *AES/EBU* STANDARD. [Ref. 1]

bit 0:	0 - the data is in the consumer (SPDIF) format 1 - professional format of block
bit 1:	0 - Normal audio mode 1 - Non audio mode (example: master synchronization device)
bits 2,3,4:	0 0 0 - No emphasis indicated receiver default is enabled 1 0 0 - No emphasis used receiver default disabled 1 1 0 - 50/15 μs emphasis (CD format) 1 1 1 - CCITT J17 emphasis (6.5 dB at 800 Hz)
bit 5:	0 - Source frequency un locked 1 - Default and source frequency locked.
bits 6,7:	0 0 - Sampling frequency not indicated 48 kHz is the default. 0 1 - 48 kHz sampling frequency. 1 0 - 44.1 kHz sampling frequency. 1 1 - 32 kHz sampling frequency.

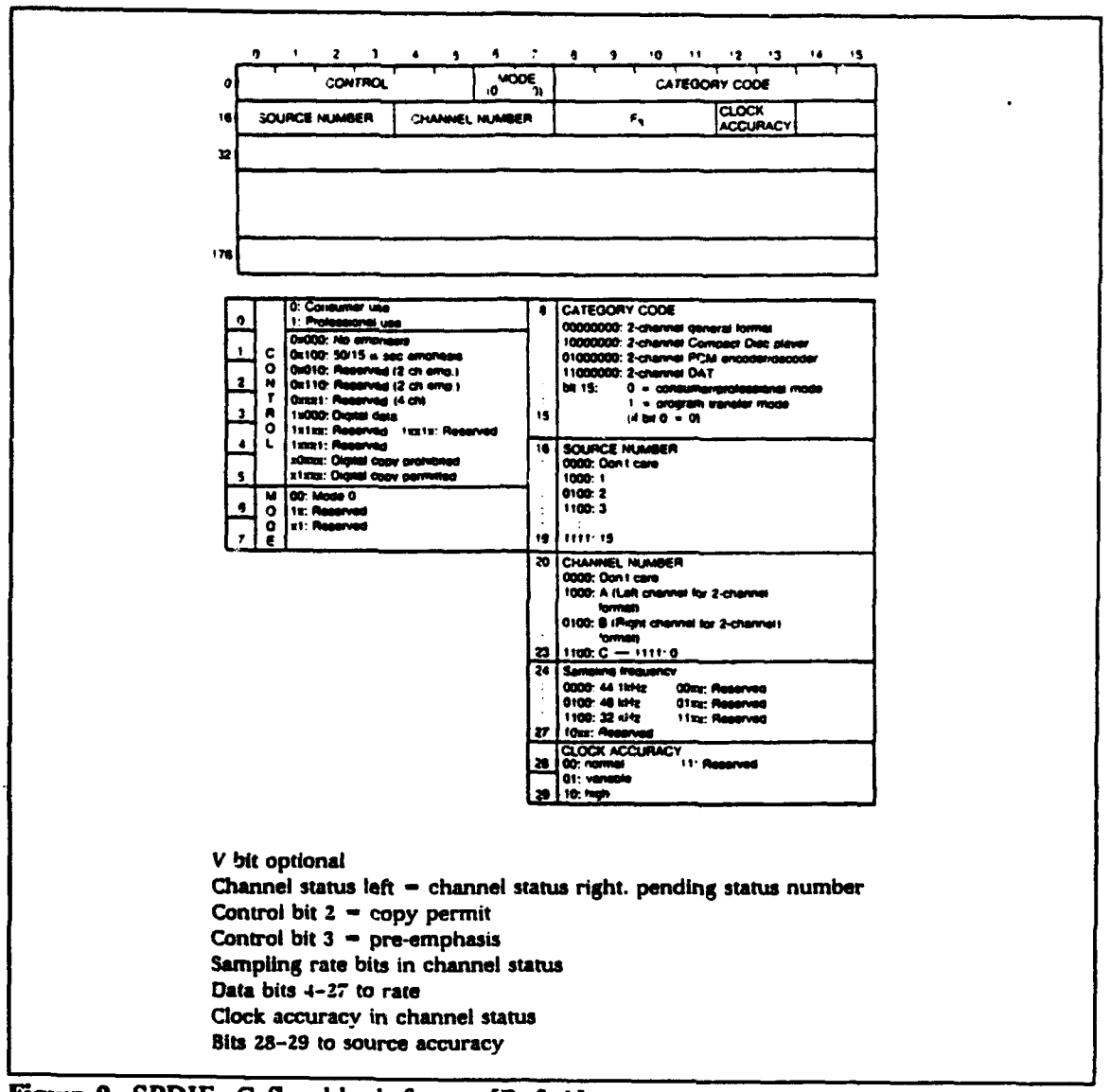
Byte 0 was very important for the development of the software in this thesis. Byte 22 contains information on unreliable samples in the audio block while byte 23 contains the channel status data cyclic redundancy check character (CRCC) of the bytes 0 through 22.

The associated generating polynomial for the CRCC is $G(X) = X^8 + X^4 + X^3 + X^2 + 1$, where X is a dummy variable in the Galua field; hence the binary divisor word is 100011101. [Ref 1]. The other bytes convey information such as channel mode and audio sample length (bytes 1,2), addressing information (bytes 6 to 13), sample index (bytes 6 to 13), and the time of the day (bytes 18 to 21). Bytes 3 to 5 are reserved for future use.

3. The SPDIF consumer format

The SPDIF consumer format is very similar to the AES/EBU professional format. The 192 frames per block and two subframes in every frame is maintained as is the overall subframe structure. The difference arises in the C block information and its structure. The C flag block is arranged in twelve 16-bit words (again a total of 192 bits) with interpretation as shown in Figure 9. The copy prohibition bit mentioned earlier is the second bit of the control field. Another field that is not specified for the AES/EBU professional standard is the clock accuracy field. The professional standard assumes the clock used is of high accuracy and its tolerance is small. Detailed discussion of the formats can be found in Reference 1.

The hardware connection is also different. This format, being consumer oriented uses a less expensive interface. The line is a coaxial cable with an RCA type connector and is designed for short length. A high impedance is allowed on the order of 50 kΩ.



V bit optional
Channel status left = channel status right. pending status number
Control bit 2 = copy permit
Control bit 3 = pre-emphasis
Sampling rate bits in channel status
Data bits 4-27 to rate
Clock accuracy in channel status
Bits 28-29 to source accuracy

Figure 9 SPDIF C flag block format.[Ref. 1]

III. SYSTEM DESCRIPTION

This chapter describes the 16-bit digital recording/reproduction system. The description highlights the general system configuration, the hardware of the interface and the operating software. The major difficulty in developing the High Fidelity signal recording and reproduction system was the fact that high data rates are involved with the process and the fact that the process runs in the UNIX operating system environment which is not real time oriented. The desired High Fidelity recording/reproduction capability had to be incorporated with no change in the kernel of the operating system or to the general network architecture. This requirement along with the high rates of the digital audio data stream (3 Mbits/sec) required use of a dedicated microprocessor board designated to handle the data transfer in real time. The special microprocessor board also acts as an interface between the different data protocols, namely, the one used by the DAT and the one used by the SUN workstation and UNIX environment.

A. GENERAL SYSTEM CONFIGURATION

The recording/reproduction system is comprised of a SUN SPARC 10 workstation, an interface board based on the Texas Instruments (TI) TMS320-C30 microprocessor, a Sony DAT model PCM-2700A which conforms to the AES/EBU professional standard, on analog cassette tape recorder, a patch-cord board, a mixer/preamplifier, a power amplifier and a set of studio monitor type speakers. IN the main mode of operation digital

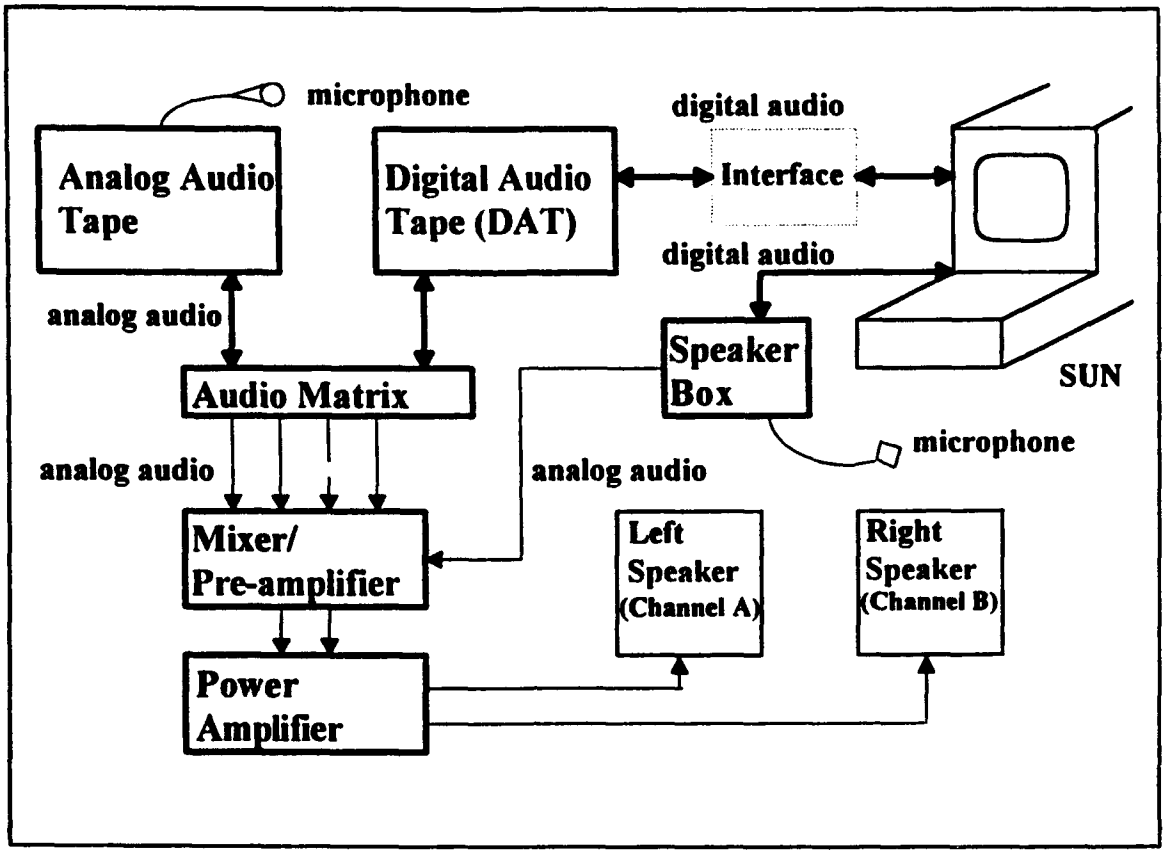


Figure 10 General block diagram of the system configuration.

audio data transfer takes place between the DAT and the SUN workstation. The other parts of the system provide a high quality listening environment for critical evaluation of the sound. The analog tape recorder provides a capability to both record and capture sound on standard audio cassette for compatibility with external systems that do not have DAT capability. Figure 10 shows the system configuration and connections between the individual blocks comprising the system. The audio output from the SUN workstation speakerbox is also connected. This provides for playing the normal 8-bit sound files using MATLAB or the SUN audio tools through the sound system. Reproduction of this audio output is enhanced via the preamplifier/mixer, power amplifier and speakers, however.

1. System recording/reproduction options

With this new system the options for receiving and storing sound signals have been considerably widened. In addition to the normal SUN workstation audio mode (8 kHz sample rate with 8-bit quantization) the new system provides 32 kHz, 44.1 kHz, and 48 kHz sample rates for two channels of audio, with 16-bit quantization at the 44.1 and 48 kHz sample rates and 12-bit (non-linear quantization) at the 32 kHz sample rate. The options for capturing data and storing in digital form are as follows:

- Digital audio data prerecorded on the DAT can be read into the SUN workstation and stored in a data file.
- Analog audio prerecorded on the analog tape can be digitized by the DAT and then transferred and stored as mentioned above.
- A microphone can be connected to the analog tape, amplified digitized by the DAT and stored at the SUN workstation.
- The SUN microphone using the normal audio mode of the station can also be used as before.

The options for High Fidelity reproduction and recording of digital data are:

- 16- or 32-bit digital audio files sent to the DAT can be converted to analog signals for monitoring and critical listening.
- Digital audio files from the workstation can be recorded directly on the DAT (no D/A conversion needed).
- Digital audio files can be converted to analog audio signals by the DAT and then recorded on the analog audio tape cassette.

The analog connections of the system are illustrated in Figure 11. There are two analog outputs from each of the tape recorders. Each of the output ports of each of the tape recorders feeds an input channel of the analog audio mixer/preamplifier and an analog input of the other tape recorder. This connection allows recording from one tape recorder to the other tape recorder without having the signal pass through the mixer. This direct recording capability saves additional distortion to the recorded signal that might be introduced in the analog mixer/preamplifier.

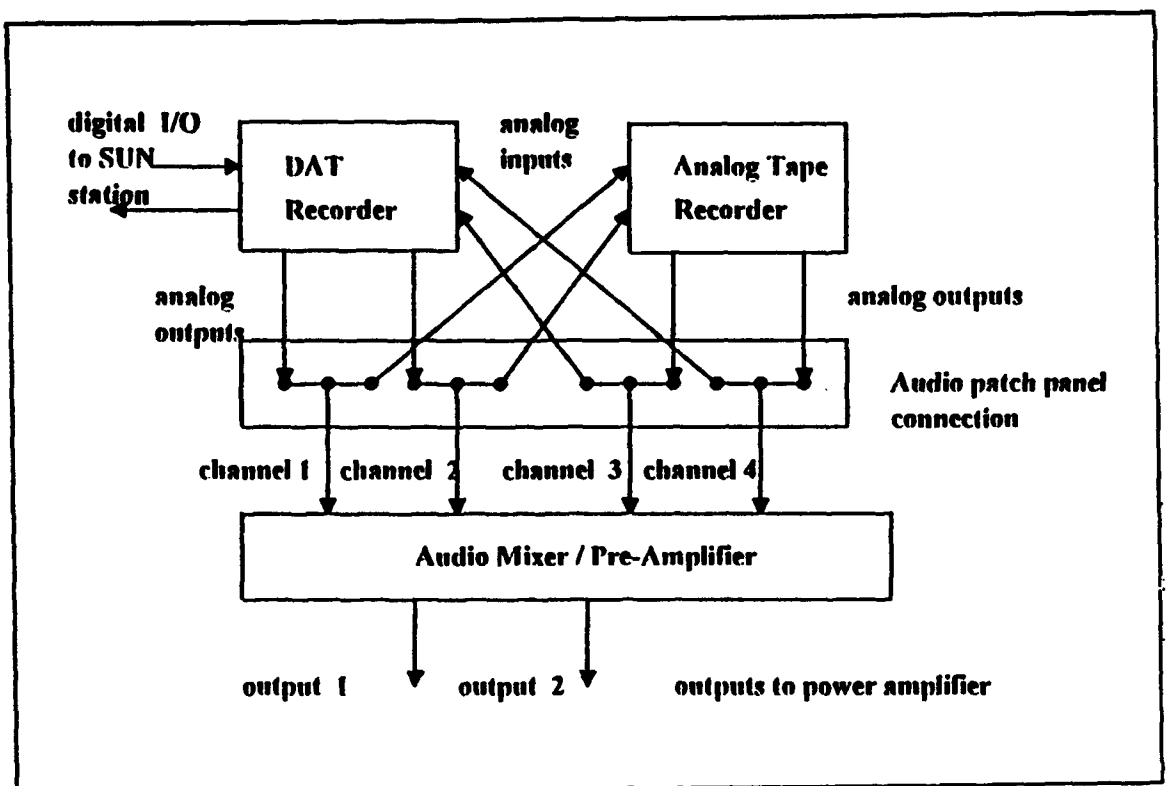


Figure 11 Analog connections of the audio matrix.

B. THE INTERFACE HARDWARE

The interface board is based on the TI TMS320-C30 micro-processor and a daughter module which serves as the AES/EBU and SPDIF transceiver. The TMS320-C30 board conforms to the S-Bus architecture of the SUN workstation. Figure 12 shows a block diagram of the interface board. The interface board and the daughter module are described separately below.

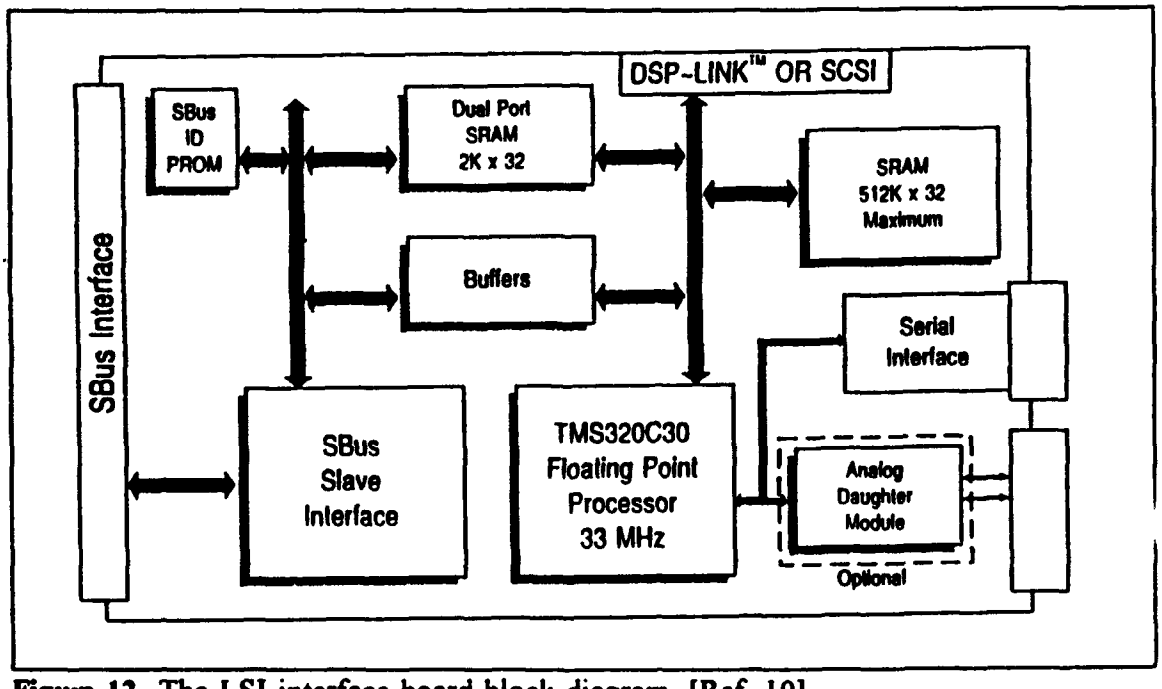


Figure 12 The LSI interface board block diagram. [Ref. 10]

1. Interface board

The main component on the board is the TI TMS320-C30 microprocessor. The board has 32-bit data bus internally. The daughter module (DM), which interfaces the board to the AES/EBU or SPDIF data stream, is shown enclosed in a dashed line in

Figure 12. The main data and address buses on the board are buffered from the host bus thus enabling different processes on the host and on the board to occur simultaneously. Transferring data in real time between the host and the board is done through the Dual Port Random Access Memory (DPRAM) . All the other devices, memory, and the TMS registers are accessible by the host, but the TMS320-C30 must be halted to avoid bus conflicts. The ID PROM on the board contains the access codes. These codes are read by the host at system boot up, and the board is assigned a device number and a virtual address. Access to a peripheral device is done using the Memory Mapping method, i.e., each device has an address in the memory space of the TMS320-C30. Selecting a device is done by placing its address on the address bus. This address is then decoded and an ENABLE signal is sent to the device along with a read or a write signal. The memory map of the board with 64k SRAM is shown in Figure 13. The DPRAM is mapped between 400000 Hexadecimal (H) and 4007FF H in the address space. The DPRAM address is calculated absolutely in the programs for the TMS and calculated relatively in the programs for the host process.

Memory Type	Size (words)	Wait States	Address (hex)
Bank 0 (U1)	64K	0	000000 to 00FFFF
Bank 1 (U2)	64K	0	010000 to 01FFFF
Bank 0 and Bank 1 Reflections	3968K	0	020000 to 3FFFFF
Dual-Port RAM	2K	1	400000 to 4007FF
Dual-Port RAM Reflections	62K	1	400800 to 40FFFF
Not Used	4032K	-	410000 to 7FFFFF
DSPLINK	8K	2	800000 to 801FFF
Reserved	8K	-	802000 to 803FFF
AMELIA	8K	2	804000 to 805FFF
Reserved	8K	-	806000 to 807FFF
On-Chip Peri-pherals	6K	Internal	808000 to 8097FF
RAM Block 0	1K	Internal	809800 to 809BFF
RAM Block 1	1K	Internal	809C00 to 809FFF
Bank 0 and Bank 1 Reflections	4056K	0	80A000 to BFFFFF
Dual-Port RAM Reflections	4096K	1	C00000 to FFFFFF

Figure 13 The Interface Memory Map (for 64K X 32 SRAM). [Ref. 10]

The Application ModulE Link Interface Adaptor (AMELIA) is the daughter module (DM) concept used by Loughborough Sound Images Ltd. (LSI). It is an inner bus that connects the main board to different plug-in modules and is mapped into the memory space of the main board. The DM used in the system is a module whose I/O conforms to the AES/EBU and SPDIF DAT standards. The address space of the AMELIA is between 804000 H and 805FFF H although the system in its present configuration uses only a small portion of this. The detailed address map of the AMELIA as reflected in the board Memory Map is given in Table III. The rest of the AMELIA address space is also not in use. Table III is specific for the AES/EBU DM and could be slightly different for

Table III *AMELIA* REGISTER MAP.

Location (hex)	Read access	Write access
804002	channel A input data	channel A output data
804005	*NU	Timer 1
804006	channel B input data	channel B output data
804008	NU	User Control
80400A	AMELIA status	AMELIA control
80400B	Interrupt status	Interrupt Mask
80400F	NU	Configuration

*NU denotes Not Used.

other types of daughter modules. The data bus hardware configuration of the board is such that all AMELIA registers are connected to the higher 16 bits (out of 32 bits) of the bus;

the lower 16 bits can be ignored. This fact needs to be considered in the programs written for the board, and the data in and out from the AMELIA should be shifted 16 bits left when writing and 16 bits right when reading. Figure 14 presents the data bus connection between the main interface board and the DM.

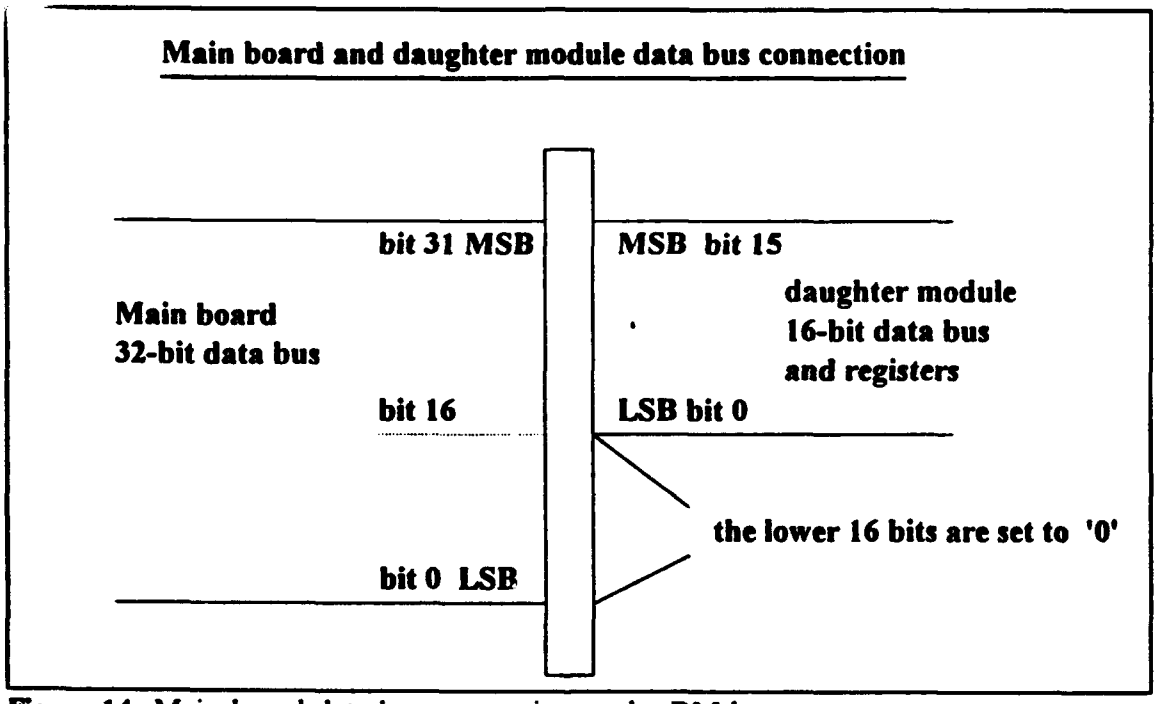


Figure 14 Main board data bus connection to the DM bus.

2. AES/EBU Daughter Module

The AES/EBU D24AES daughter module provides a digital audio interface to any equipment that conforms to the AES3 professional balanced data format and to the consumer CEI IEC 958 standard unbalanced data format, also known as SPDIF. A functional block diagram of the module is illustrated in Figure 15; both receive and transmit modes of operation are depicted. The main component of the DM is a digital audio transceiver. This transceiver conforms to the digital audio standard and is the device

audio transceiver. This transceiver conforms to the digital audio standard and is the device that is connected to the digital line in and digital line out of the DAT. The data that arrives from the DAT is locked on to and decoded by the transceiver. The output of the transceiver is a parallel 16 bit word originating from the incoming serial digital audio transmission. The reverse procedure takes place when a digital audio sample is transmitted; the sample is sent as a parallel word to the transceiver which is then translated and sent out serially.

The D24AES DM has two main modes of operation, the 16-bit mode and the 32-bit mode. These modes of operation are programmed via the Control Register. In the

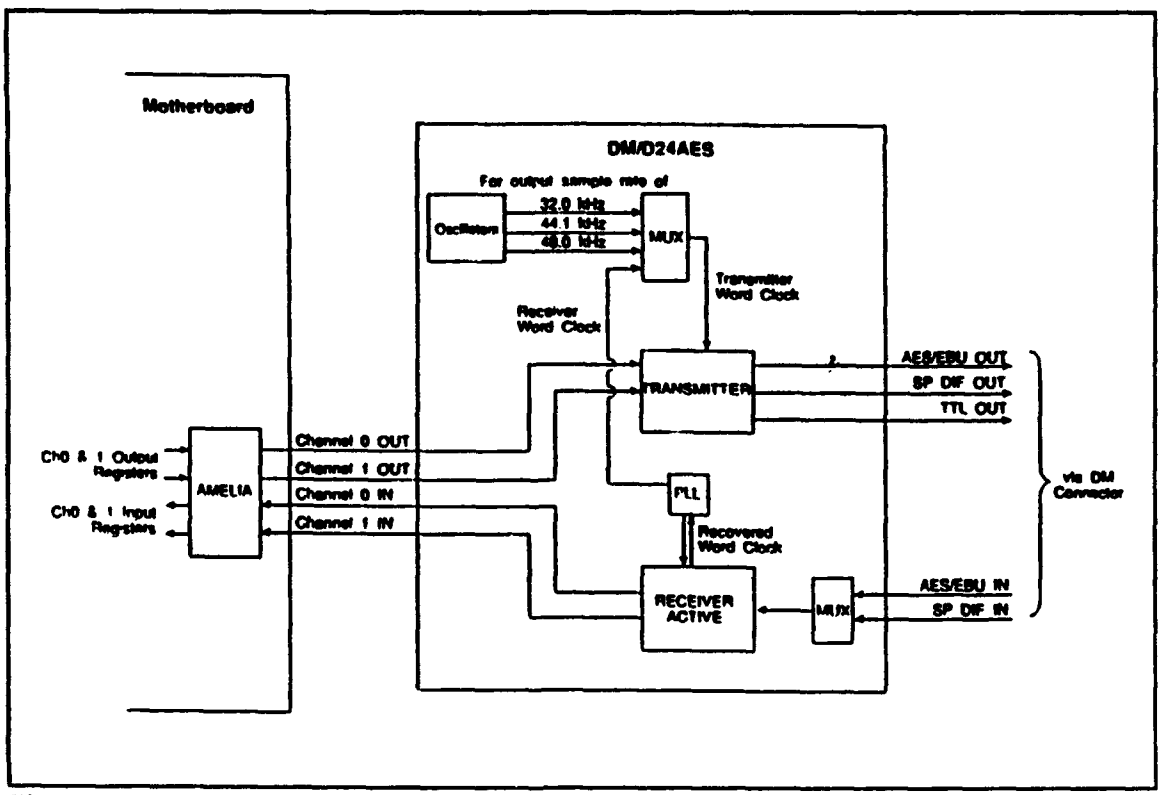


Figure 15 AES/EBU Daughter Module block diagram. [Ref. 12]

16-bit *receive* mode each audio sample is filtered by the transceiver, all framing is stripped off, and only the 16 most significant bits are transferred to the user. In 16-bit *transmit* mode, i.e., when the data flow is from the board to the DAT, only the 16-bit audio sample should be sent to the transceiver; all necessary framing is added by the transceiver in this mode of transmission. The 16-bit mode is the simpler of the two modes of operation. The advantage of the 16-bit mode is its simplicity and that it requires only 16 bits of storage per data sample. The disadvantage of this mode is that the transceiver adds default framing to the audio sample. This default framing allows data transfer to the DAT used in the system only at the 48 kHz sample rate, which may or may not be desired. The more general mode is the 32-bit mode. In this mode the program receives complete information for the audio sample including the flags and up to 24 bits of audio sample data. Due to the hardware configuration and the different data bus width of the board and the DM (Figure 15), two read cycles need to be executed in order to fetch the entire digital data word, and when transmitting data two write cycles are required to write the entire digital word into the transceiver. The program on-board the TMS320 has to properly set the C flag. (The C flag is the most important flag for proper system operation and is described in Chapter II). Appendix B provides additional information about the register structure and bit content of the AES/EBU daughter module.

C. THE SYSTEM SOFTWARE

The operating system (host) software and the software for the digital interface were written in the C language as part of this thesis. The host software is divided into two

main programs, one for storing a digital audio segment prerecorded on DAT to the system's disk, called "run_save," and another for playback/record (to DAT) an audio file stored on the system's disk called "run_play." These programs will be referred to as "main programs" or "main processes." Each of the main programs has a subprogram associated with it, which are designed to run on the interface board. The subprogram of "run_save" is called "sp2ae_save" while the sub-program of "run_play" is called "sp2ae_play." In the SUN UNIX file system the source codes for the above programs have the ".c" suffix, the main executable programs file have no suffix, and the subprogram object files have the ".out" suffix. Figure 16 illustrates the general software block diagram, relationship between the main program and the subprogram, and the data flow. Note that the main programs control the subprograms. The control of the subprograms is done by first downloading the subprograms to the interface board at a certain point in the execution of the main programs and then controlling the start and the end of the execution of the subprograms.

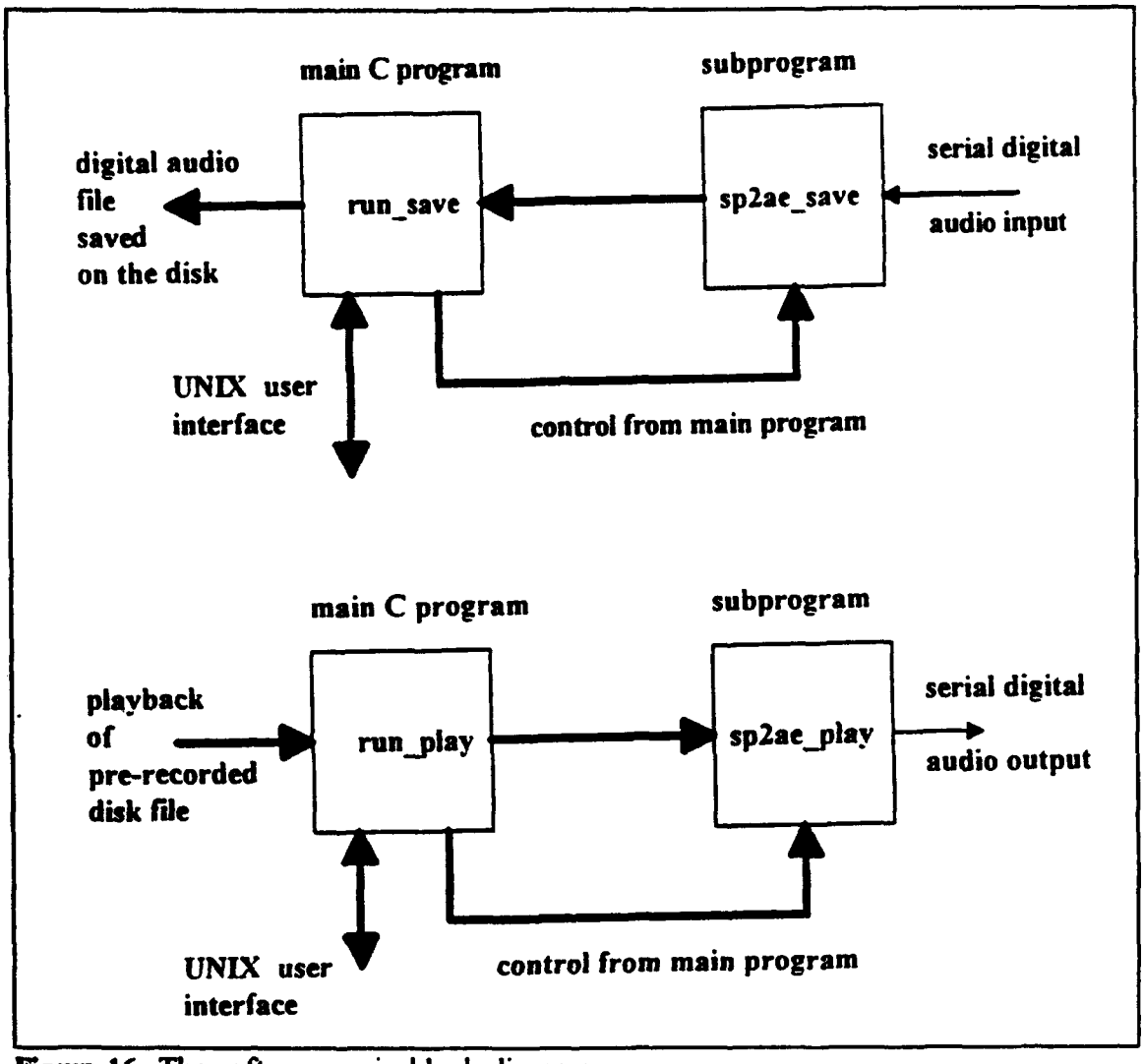


Figure 16 The software main block diagram.

The aim of the subprograms is to control the data transfer between the Input/Output (I/O) port of the DAT and the SUN workstation in *realtime*. The data transfer between the serial DAT I/O port and the SUN workstation is done in a few stages using the "Double Buffering" technique described in detail later in this chapter Figure 17 illustrates the data transfer process between the system disk and the digital audio I/O port. A gradual data transfer is needed to synchronize the real time fixed rate data stream of the DAT to the non-real time UNIX operating system. The main program builds an array for the entire audio segment in the RAM of the SUN workstation When storing an audio segment from the DAT, the entire array is filled and then sent to the disk. In outputing to the DAT the array is read from a file completely and then sent to the interface and

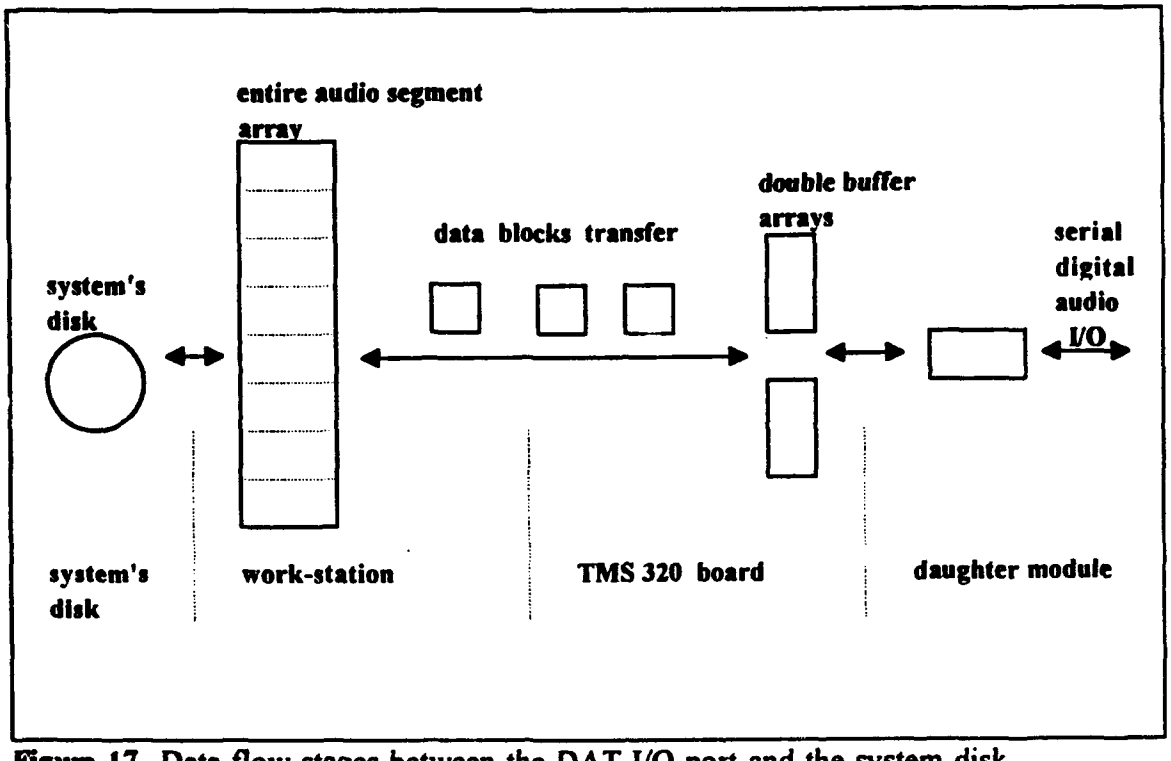


Figure 17 Data flow stages between the DAT I/O port and the system disk.

from there to the DAT. The data is transferred between the interface and the array in blocks of 960 32-bit words. The audio array although dynamically allocated is designed to contain an integer number of such blocks; thus the recording/playback time resolution is 20 msec at the 16-bit mode and 10 msec at the 32-bit mode assuming a 48 kHz sample rate.

The main C programs use some library functions from a library provided with the interface hardware. These C library and header files are called "sdsp30lib_st.c" and "sdsp30lib.h" respectively. The functions used in developing the main programs assist with data transfer and control of the board. The subprograms "sp2ae_save" and "sp2ae_play" were also written in C and debugged using the TMS320 assembler language. The TI TMS 320 software development kit was used to compile and link the written C source code of the subprograms. A debug software monitor "smon30" from LSI was used to examine the real time processes.

1. Main program "run_save"

The program "run_save" retrieves an audio segment from the DAT and stores it on the system's disk. Figure 18 illustrates the flow of the program. The first hardware check is meant to insure that no other process is using the interface, in which case an error might occur. The data structure for storing the entire audio segment is allocated dynamically. The data structure is explained later in the chapter in detail. Dynamic allocation is used to respond to the need for varying record time specified by the user at run time. The whole audio segment is recorded in the audio array and is then sent to the disk. Disk access is very slow compared to the other processes, thus it is done after

storing the entire audio segment in the data array. When the available memory is not sufficient to contain the desired data segment the user is informed about the maximum

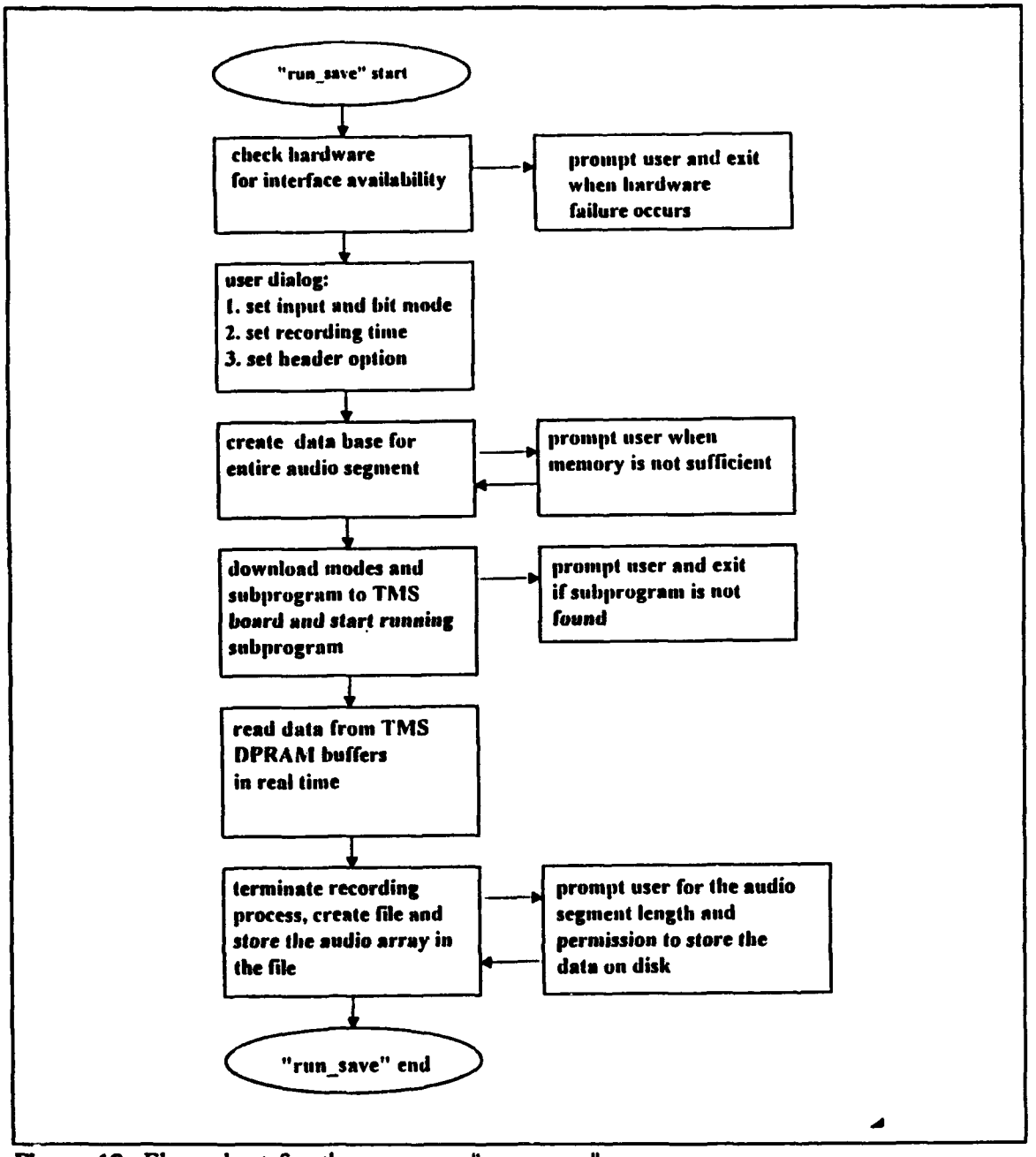


Figure 18 Flow chart for the program "run_save."

length segment that can be stored in the accessible memory. The user provides the name of the file in which the audio segment is stored, which should have suffix ".dau." If no suffix is specified ".dau" is appended automatically.

2. Main program "run_play"

The program "run_play" is the main C program for playing and/or recording (on DAT) an audio file which was previously stored on the disk. The functional flow of the program is illustrated in Figure 19. The structure of the program is very similar to "run_save" with some changes. The length of the audio array in this case is determined by the length of the audio file. The data from the file is read entirely into the data array; then the data from the array is sent in blocks to the real time process which is running on the interface board. The "run_play" program contains another related feature which is not shown in Figure 19. This feature is meant to check the hardware of the interface. A second DAT, using the SPDIF port, can be connected to the interface, and digital audio can be transferred from one DAT to the other through the interface. This is mainly a debug feature; when working properly there is no distortion of the sound recorded on the main DAT. This feature is run using the 32-bit mode.

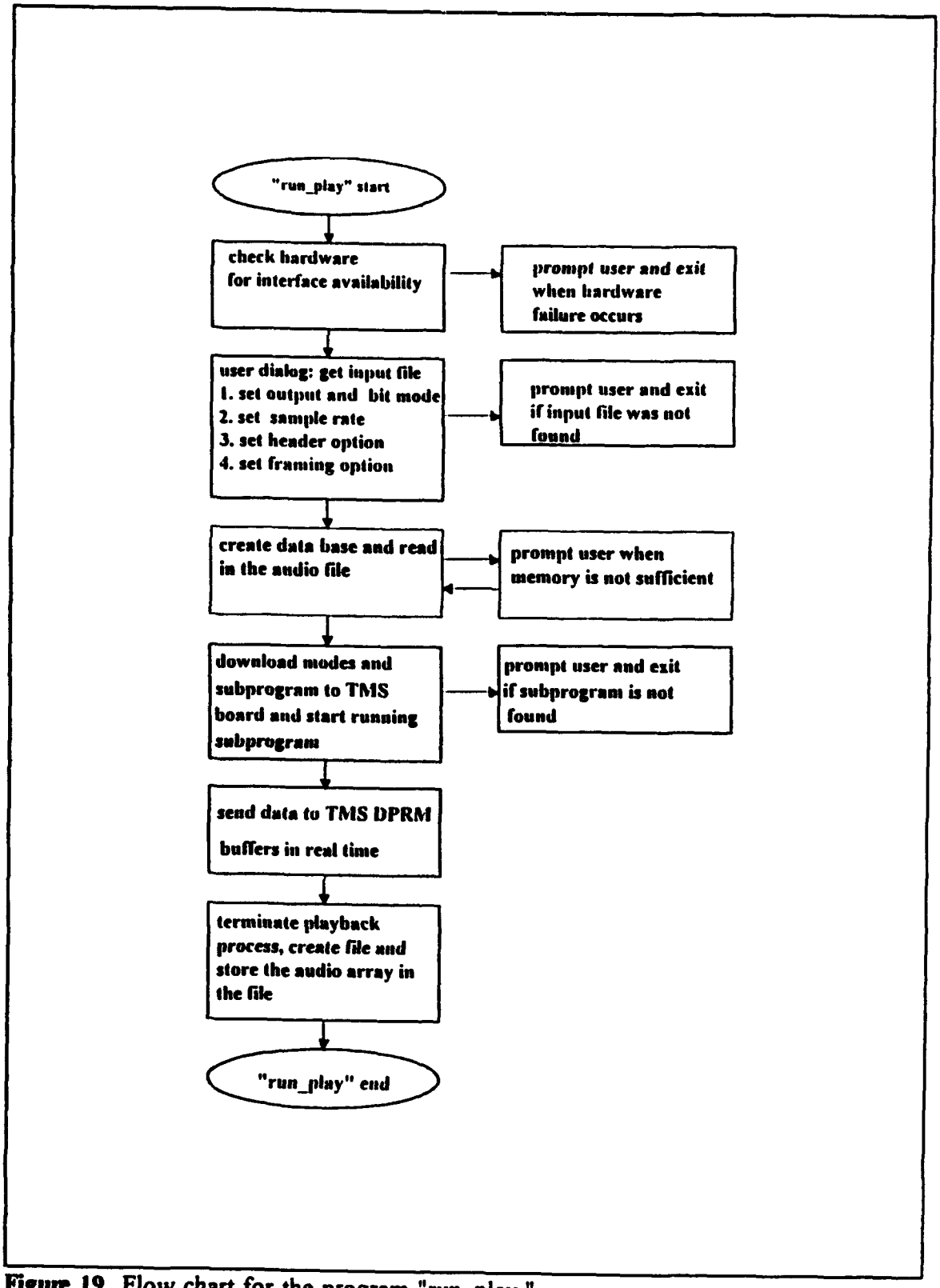


Figure 19 Flow chart for the program "run_play."

3. Data structure and double buffering method

The data structure dynamically created during run time is comprised of a connected list. Each element of the list is a C structure consisting of an array which contains 1920 32-bit words (10 AES/EBU blocks or two real time buffers sent or received from the sub-program), a flag word that denotes if the local array was filled, and a pointer to the next structure. Figure 20 illustrates the structure and the list connection. The pointer of the last structures points to 'NULL'. [Ref. 5]

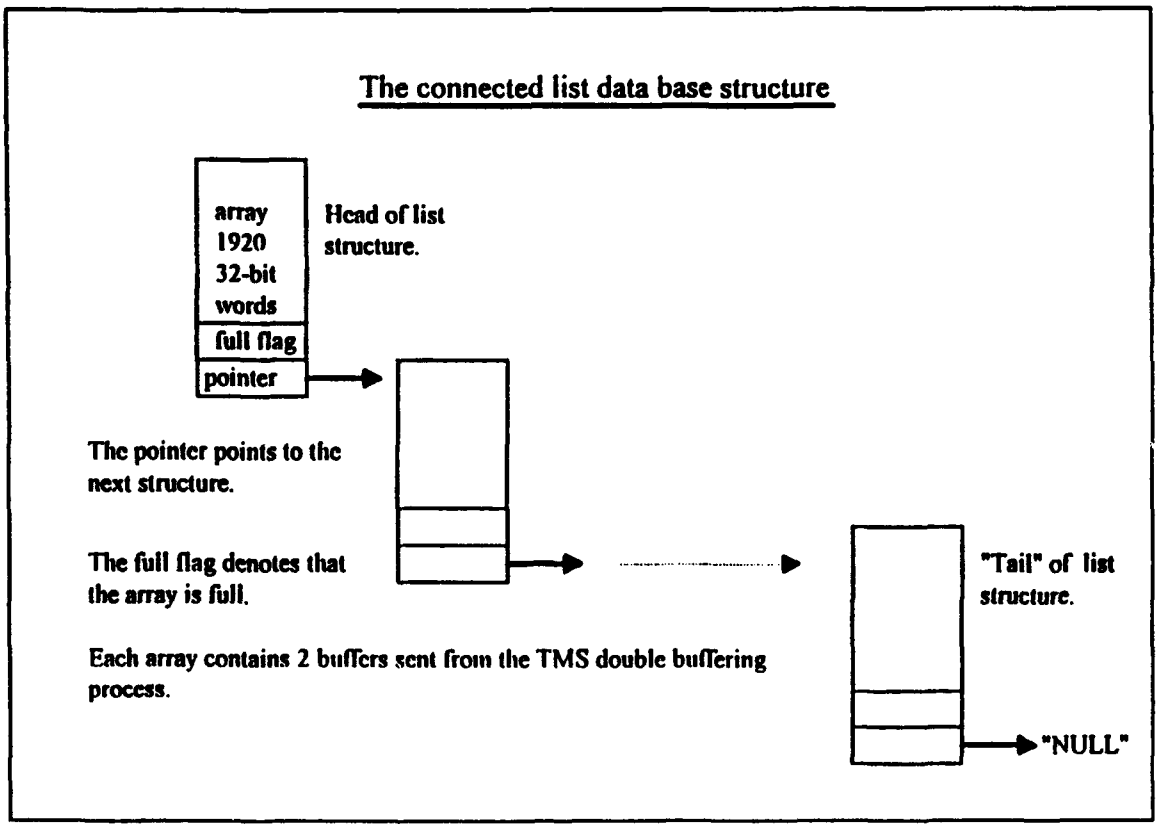


Figure 20 Data base connected list and the unit structure.

A double buffering technique is used to fill this structure because there is a need to coordinate the two processes with two different rates of data transfer. One is the rate between the interface and the DAT tape which is fixed and known; the other is the rate at which the data is finally stored on the system's disk, which is unknown and random due to the scheduling provided by the UNIX operating system and other processes that can occur simultaneously. The real time data transfer rate can be thought of as the customer, and the UNIX data transfer rate can be thought of as the server. The basic assumption that enables the whole process of data transfer to succeed is that the mean rate of servicing is faster than the rate of data entering the system. To fill gap times in service availability, buffers are used to store the data until the server is available. Then the server can process and fetch the whole buffer very rapidly. Double buffering is used when simultaneous access to a single buffer can lead to conflict errors. With the use of two buffers, when one buffer is filled by the fixed time process the other buffer can be read by the random time process. After the completion of servicing the buffers are switched, and the second buffer is filled by the fixed rate process, and the first is serviced by the server. This avoids any conflict. Figure 21 illustrates the double buffering data transfer and control used in the program. The flags and the buffer switching is done by the fixed time processes. The status is constantly read by the host, and when a buffer is ready its contents are transferred via the UNIX system to the storing program, or rewritten in the case of playback program.

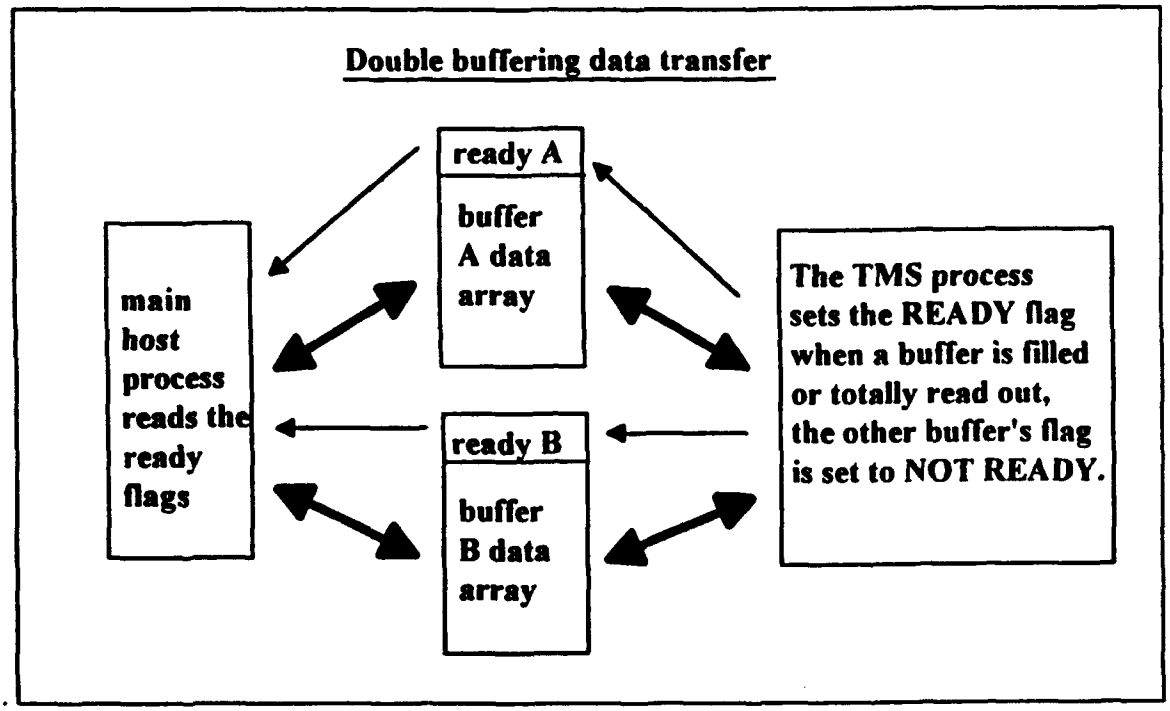


Figure 21 Double buffering method block diagram.

4. Audio file structure

The structure of the disk file in which the audio data is stored is related to the bit mode of operation (16-bit mode or 32-bit mode). In the 16-bit mode each audio sample is represented by 16 bits, and only the audio sample itself is stored as a 2's complement (signed) integer. The data samples are interleaved, i.e., the first sample is from channel A, and the second sample is from channel B. This repeats until the end of the file. In the 32-bit mode the audio and framing data are both stored. The first 16 bits is audio from channel A. This is followed by the audio from channel B, then 16 bits of framing for channel A, followed by 16 bits of framing for channel B. This repeats until

the end of the file. The data file format is summarized in the Table IV below. At the sampling rates used by the DAT, the audio files can be quite long. For example, a 10 second audio segment recorded at the 48 kHz sample rate would produce 480000 32-bit words in the 16-bit mode and 960000 32 bit words in the 32-bit mode. Even the 16-bit mode data file leads to a size on the order of a megabyte of data.

Table IV THE AUDIO FILE STRUCTURE.

The audio file structure in the 16-and 32-bit modes	
16 - Bit mode	32 - Bit mode
audio sample channel A 16 bits	audio sample channel A 16 bits
audio sample channel B 16 bits	audio sample channel B 16 bits
:	framing data channel A 16 bits
:	framing data channel B 16 bits
:	:
audio sample channel B 16 bits	framing data channel B 16 bits

5. Subprogram "sp2ae_save"

The real time sub-program "sp2ae_save" transfers data from the DAT to the host process "run_save" via the double buffering scheme. The program flow chart is shown in Figure 22. The modes of operation are set prior to program execution by the host process in specific designated addresses. The sub-program reads those preset modes and sets the hardware configuration accordingly. A delay loop of about 1/2 second was added to ensure stabilization of the Phase Lock Loop (PLL) in the receive mode. This

time was established after a few preliminary trials. The data transfer can start immediately after this loop. The process is signaled to start the recording by a flag sent from the host when the user hits the ENTER key. A header feature was added to synchronize the start of a desired data segment within 1 sample. The need for header and its implementation are described in the next subsection. The data to the host is sent through the DPRAM using the double buffering method described earlier.

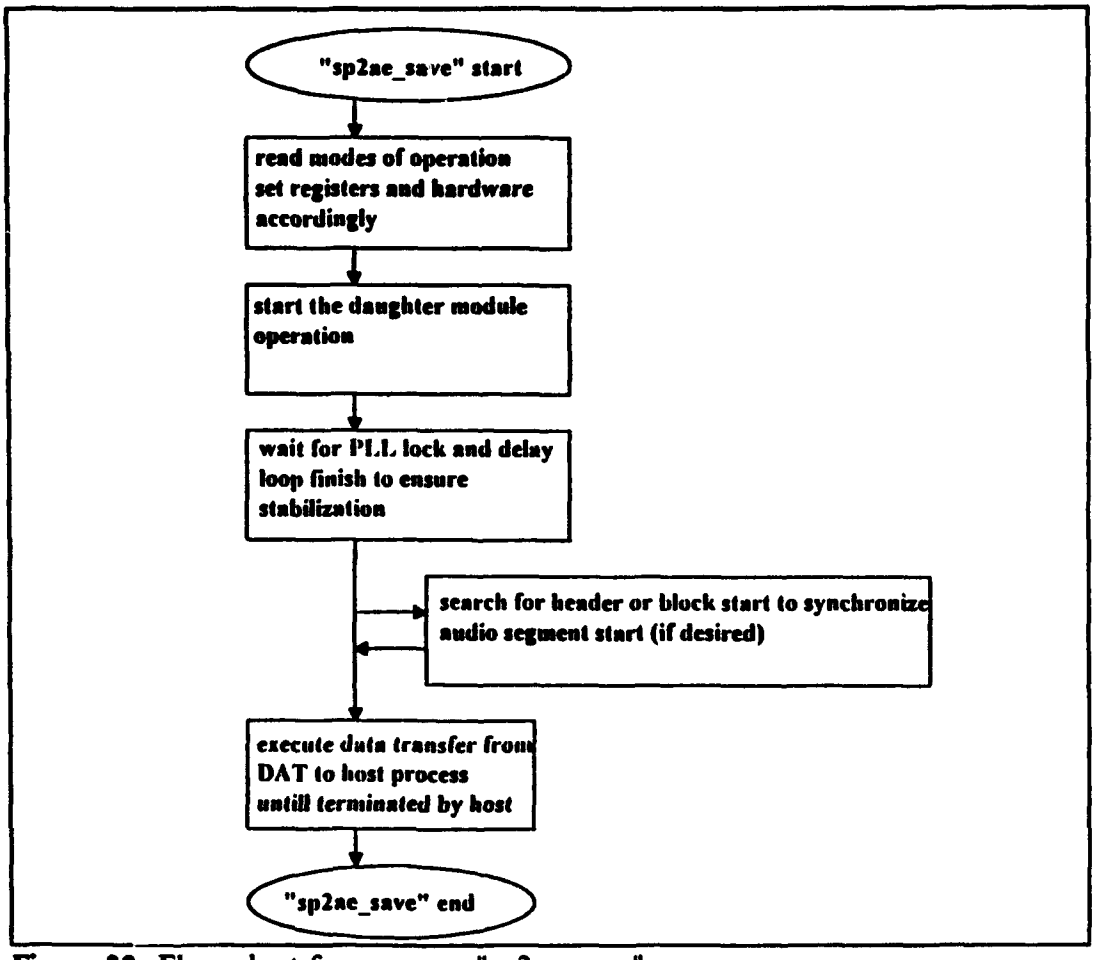


Figure 22 Flow chart for program "sp2ae_save".

6. Subprogram "sp2ae_play" and header inclusion

The "sp2ae_play" process is the subprogram for "run_play". This program transfers data from the host process to the DAT output port for playback or recording on a tape. The flow diagram is shown in Figure 23. This program creates the header that is meant to be the synchronization signal when retrieving the file again. The program also

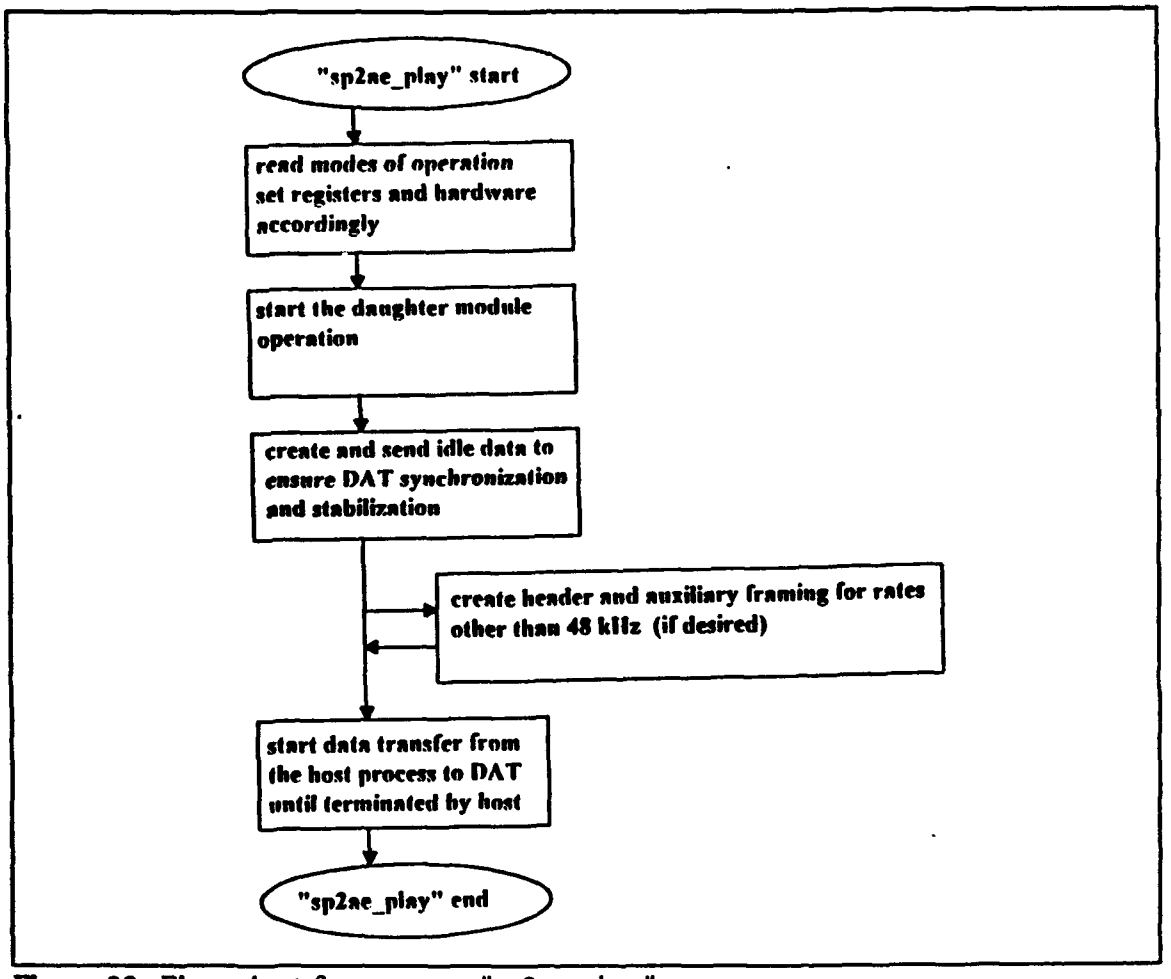


Figure 23 Flow chart for program "sp2ae_play."

creates default framing which is necessary to enable playback, using the 32-bit mode, of a file stored previously in 16-bit mode in sampling rates other than the 48 kHz sample

rate which is the default rate for the specific DAT used in the system. A serial message of zeros is sent to the DAT prior to the actual data transfer to let the DAT synchronize on the proper mode of operation and to allow the internal PLL circuity to become stable.

The header feature was added to help the user precisely locate the start of recording of an audio segment. It is very difficult to start the tape playback precisely using the controls on the DAT. The precision might be on the order of half a second which will introduce about 24000 additional samples. Thus without some kind of additional help it is practically impossible to locate the exact beginning of the desired audio segment. The header is a unique pattern recorded on each of the two channels; each channel has a different unique word, and both words need to appear simultaneously to declare the start of a header block. The header structure is illustrated in Table V and consists of a block of unique words (192 words) recorded on each channel followed by a block of zero words. The probability of a false header start declaration is 2^{-32}. (In trials

Table V HEADER MESSEGE FORMAT.

192 header unique word	192 zero word	Following audio segment

using the header with different types of audio segments a false header was never declared.) The unique header word for channel A is eeeeH and for channel B 1111H. The

absolute values are the same, but the sign is opposite. Figure 24 illustrates the header recorded on each of the channels when the audio data is later retrieved into MATLAB and plotted. Note that the header option operation is suitable for retrieving data from a DAT but not from the analog tape recorder.

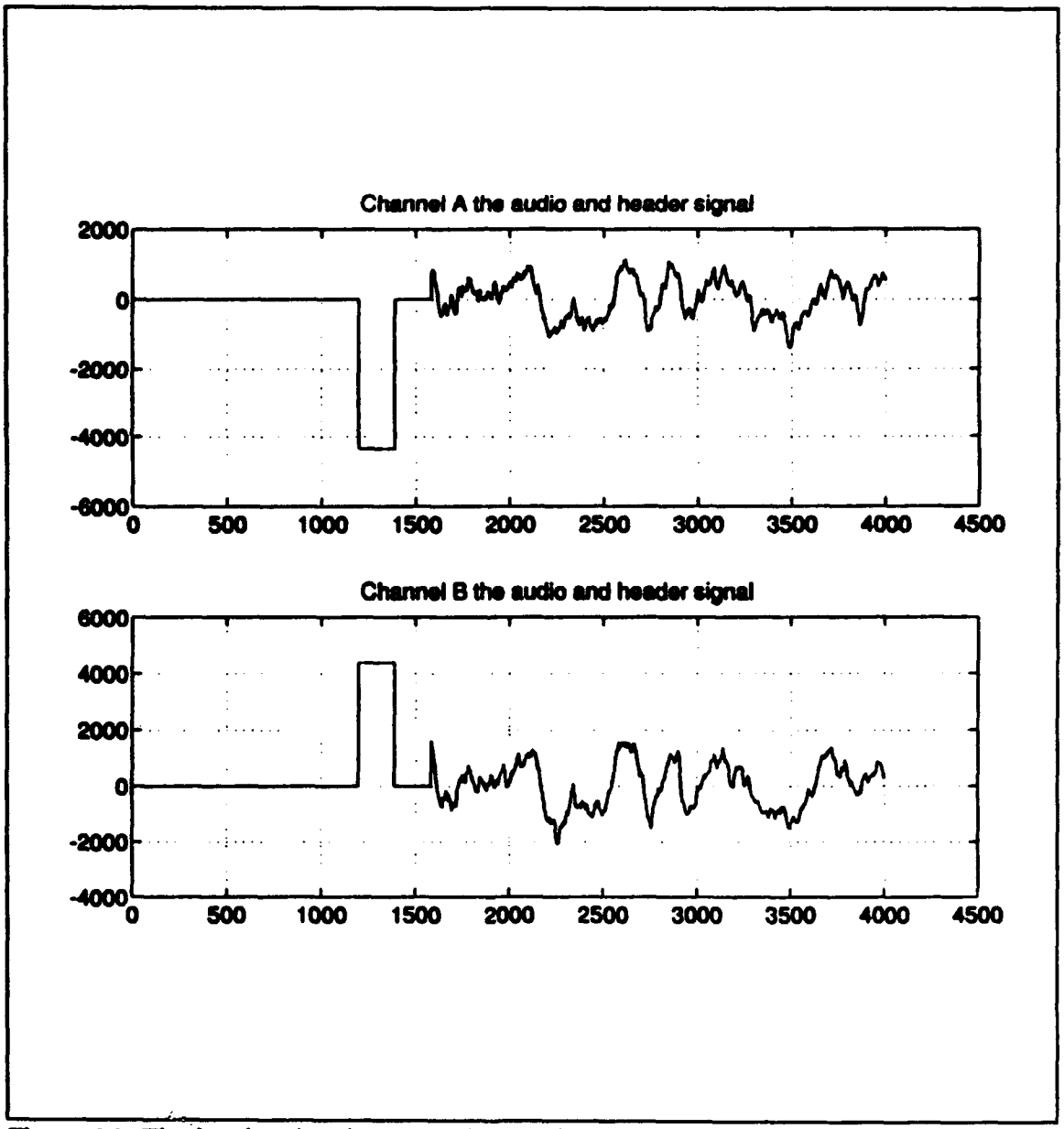


Figure 24 The header signal as recorded on the start of an audio segment.

D. RETRIEVING AUDIO FILES INTO MATLAB

The audio files used by the "run_play" and "run_save" programs can easily be read into MATLAB. Two MATLAB functions were written, "retrieve_dat.m" which reads a "*.dau" file in the MATLAB workspace, and "store_dat.m" which stores a vector of sound data into a "*.dau" file that can later be sent to the DAT for playback. When a file is brought into MATLAB it is in matrix form. If the file was recorded in the 16-bit mode then the matrix has two rows (one for each channel) and a number of columns equal to the time in seconds multiplied by the sample rate. If the file was recorded in the 32-bit mode the matrix has four rows. The first and second rows are the audio data for channels A and B respectively while the third and the fourth rows are the corresponding framing data. Figure 25 illustrates the matrix rows as plotted in MATLAB.

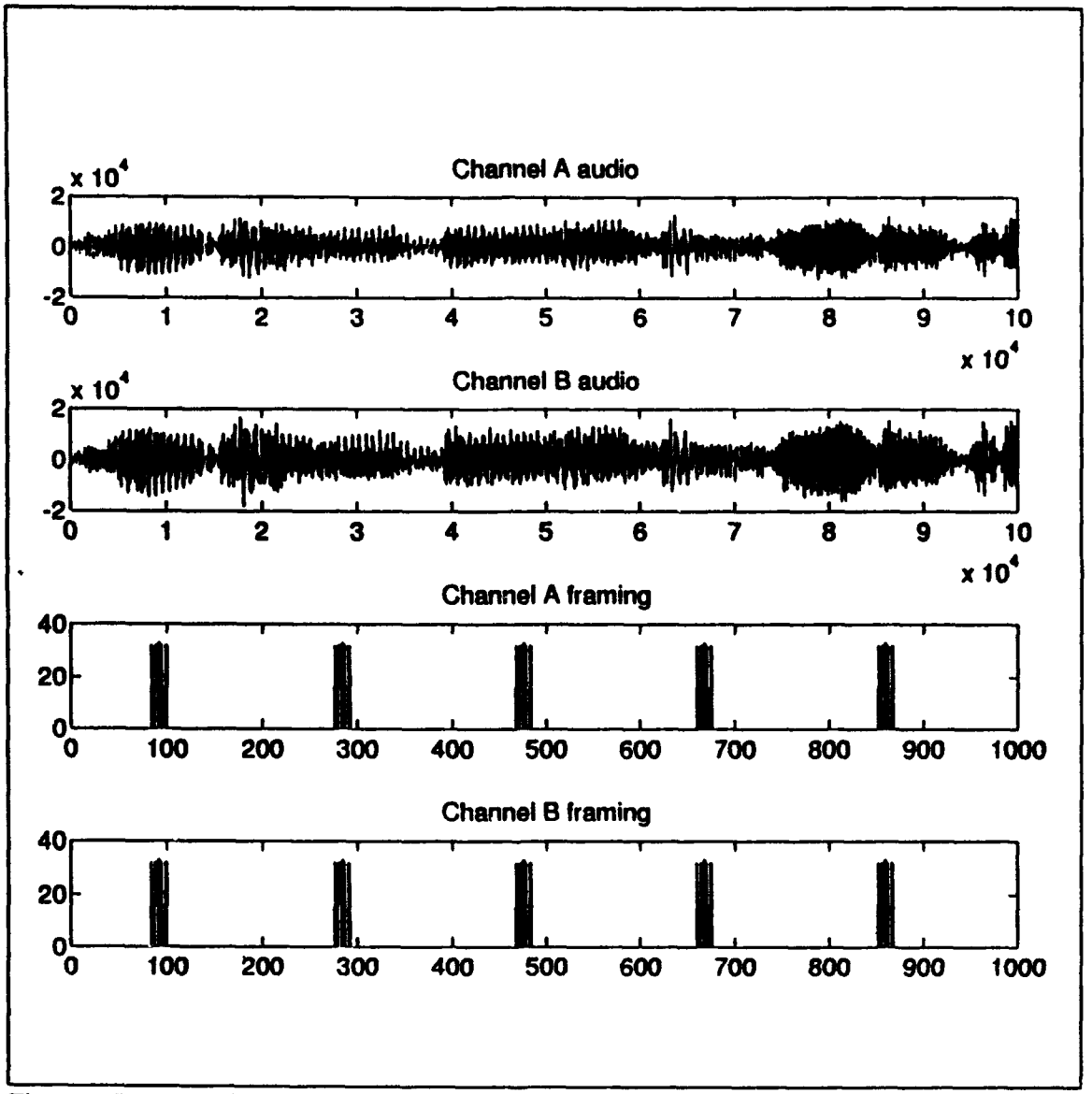


Figure 25 An audio segment captured using the 32-bit mode.

IV. PERFORMANCE TESTS

This chapter describes the measurements made to test the performance of the digital audio system. The measurements focused on two major subjects: purity of audio reproduction and purity of audio recording. The header synchronization capability of the system was also tested for proper operation. The measurements were made in the audio band between 20 Hz and 4 kHz. This bandwidth was chosen in order to compare the results to the present audio capability of the SUN workstation, which provides 8-bit digital audio only in this band, and which serves as the reference.

A. AUDIO REPRODUCTION PURITY PERFORMANCE

The audio reproduction performances were tested to measure the following parameters:

- Signal to noise ratio.
- Total Harmonic Distortion.
- Reproduction linearity (Intermodulation products).

The equipment used to measure these parameters included an HP 3582A audio frequency spectrum analyzer and an HP 334A audio frequency distortion analyzer. The measurement procedure was as follows. Discrete frequency sinusoidal signals were synthesized in MATLAB. The signals were then sent for audio reproduction to the preamplifier. The

input signal to the preamplifier was the signal measured in order to avoid any distortion or interference that night be added by the preamplifier. The signals to the preamplifier were sent through the interface output to the DAT, which provided A/D conversion, and from the analog balanced output of the DAT to the preamplifier (Chapter III, Figure 10). The reference measurement was made by sending the same signals from the SUN workstation through the speakerbox to the preamplifier input where the reproduced analog signals were again measured. The frequencies of the sinusoidal signals synthesized were 1 kHz and 1.5 kHz while the sampling rates were 8 kHz for the SUN workstation and speakerbox, and 48 kHz using the DAT. To provide some of the results shown here the monitor of the spectrum analyzer was photographed. Table VI summarizes the results.

Table VI AUDIO REPRODUCTION PERFORMANCE MEASUREMENTS

Measured parameter	Audio reproduction using the SUN speakerbox output. 8-bit μ-law, 8 kHz sample rate	Audio reproduction using the DAT output. 16-bit linear, 48 kHz sample rate
Signal to noise ratio	50.6 dB	94.9 dB
THD [%]	1.2 %	less then 0.1 % (the limit of the meter)
Signal to Intermodulation product ratio.	42 dB for the highest intermodulation product signal.	more then 66.8 dB. (Measurement was limited by the dynamic range of the spectrum analyzer.)

The following figures provide photographs of the spectrum analyzer monitor for the various measurements. Figure 26 shows the spectrum of the 1 kHz sinusoidal signal seen with the noise as measured at the output of the speakerbox. The marker is on the signal. Figure 27 is the same as Figure 26 with the marker located on a high peak of the noise. The difference between the marker values in Figure 26 and Figure 27 (-15.8 dB - (-66.4 dB)) is the signal-to-noise ratio i.e., signal peak power to noise peak power (50.6 dB). Note the center frequency of the signal reproduced by the speakerbox is 980 Hz and not 1 kHz because the sample frequency used in MATLAB to synthesize the signal was 8192 Hz and the speakerbox sample frequency is 8 kHz exactly. This is of no consequence in the measurements however.

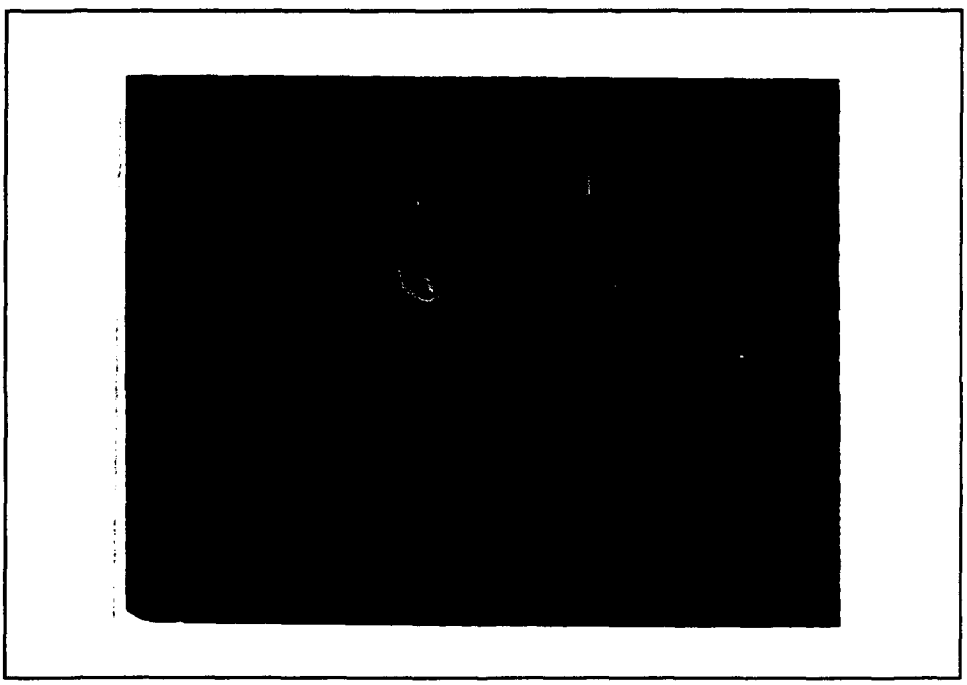


Figure 26 Output signal from the SUN speakerbox; the marker is on the signal.

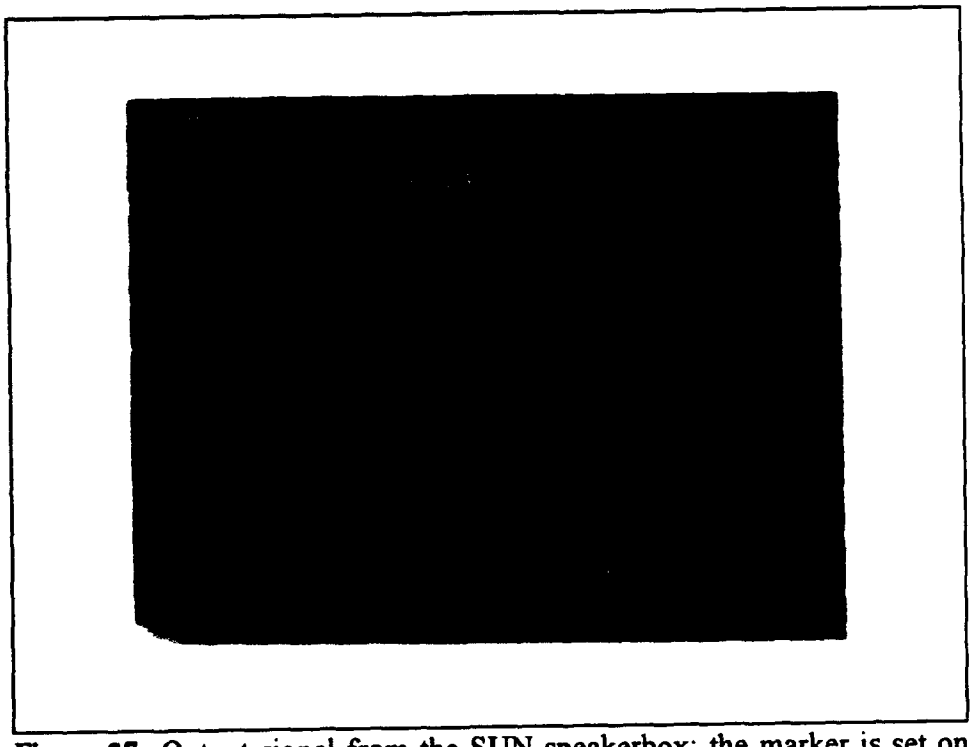


Figure 27 Output signal from the SUN speakerbox; the marker is set on a noise peak.

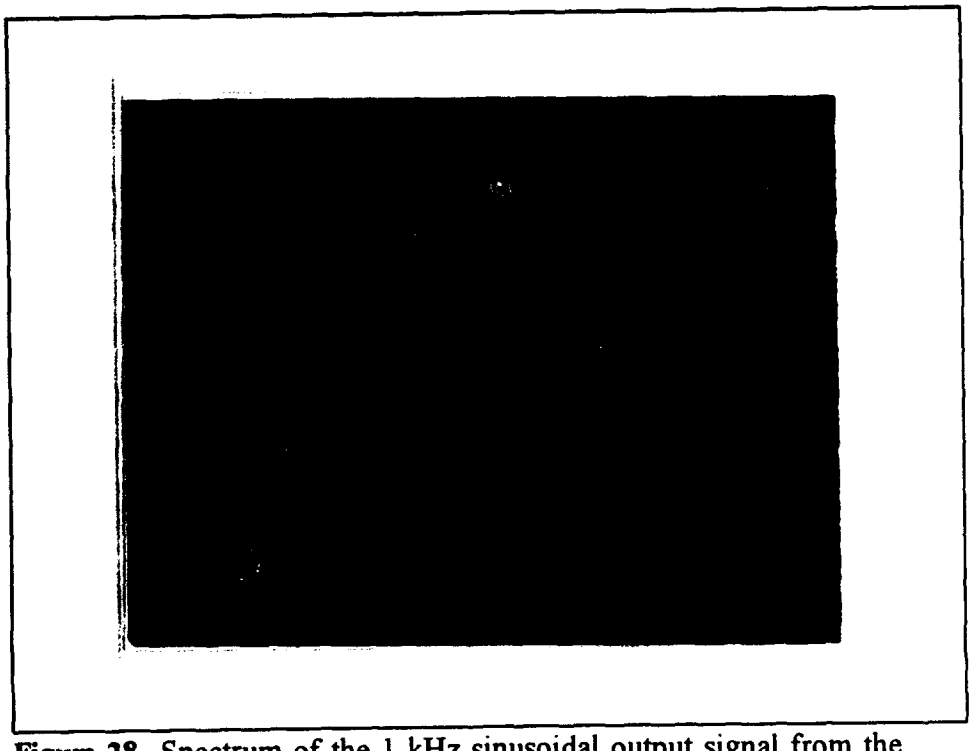


Figure 28 Spectrum of the 1 kHz sinusoidal output signal from the DAT.

Figure 28 shows the spectrum of the 1 kHz sinusoidal signal as reproduced by the DAT. The noise is not seen in Figure 28 due to the high signal-to-noise ratio and the dynamic range of the spectrum analyzer which is 80 dB. Figure 29 shows the noise only. In order to measure the noise a zero level signal was sent to the output of the DAT. This enabled us to reduce the measurement level of the spectrum analyzer, so the noise could then be measured.

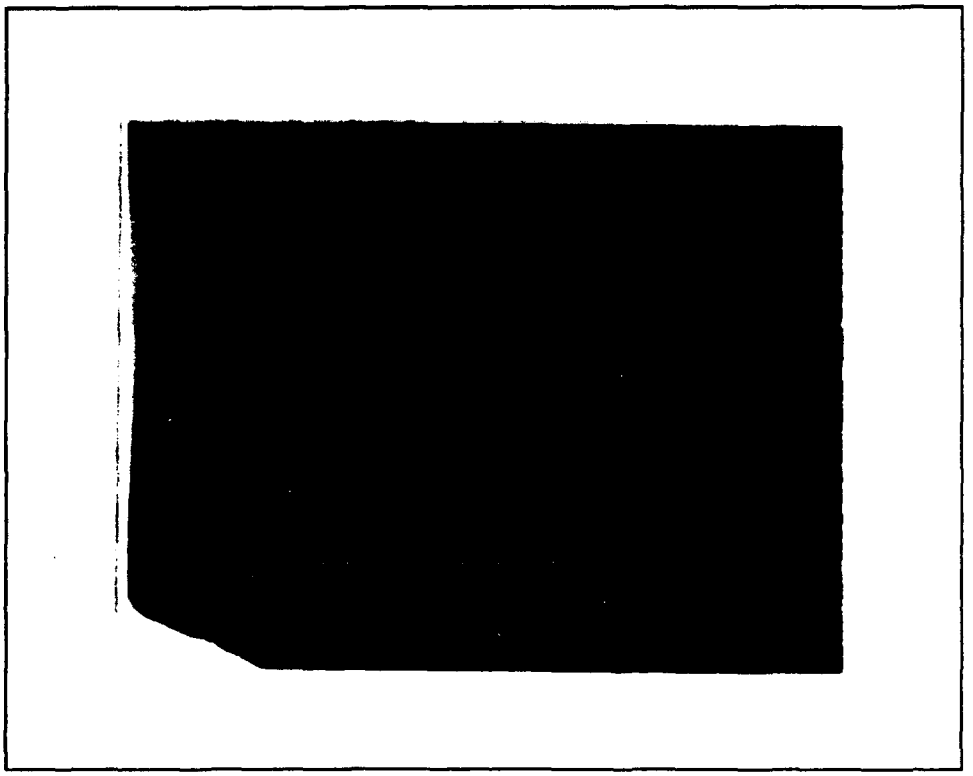


Figure 29 Noise only output from the DAT with zero output signal.

Figures 30 and 31 show the intermodulation products created when two sinusoidal signal were synthesized and sent to the audio ports. The frequencies used were approximately 1 kHz and 1.5 kHz. The first order difference signal is approximately 500 Hz. Several significant intermodulation product terms can be noticed in the measurement of the SUN workstation audio reproduction (Figure 30). For the DAT the level of any intermodulation product terms were beyond the minimum display limit of the spectrum analyzer. The measurement level could not be reduced due to the requirement that both of the signals should be present to make the measurement.

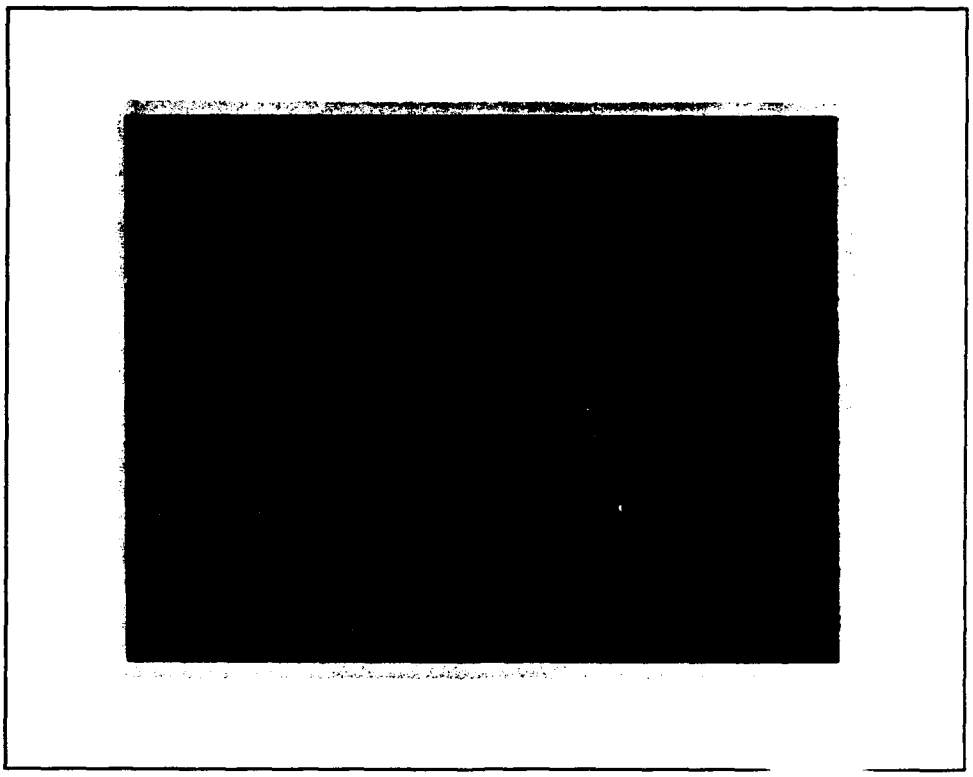


Figure 30 Output signal from the SUN speakerbox; the m[illegible]s on the first order difference intermodulation product.

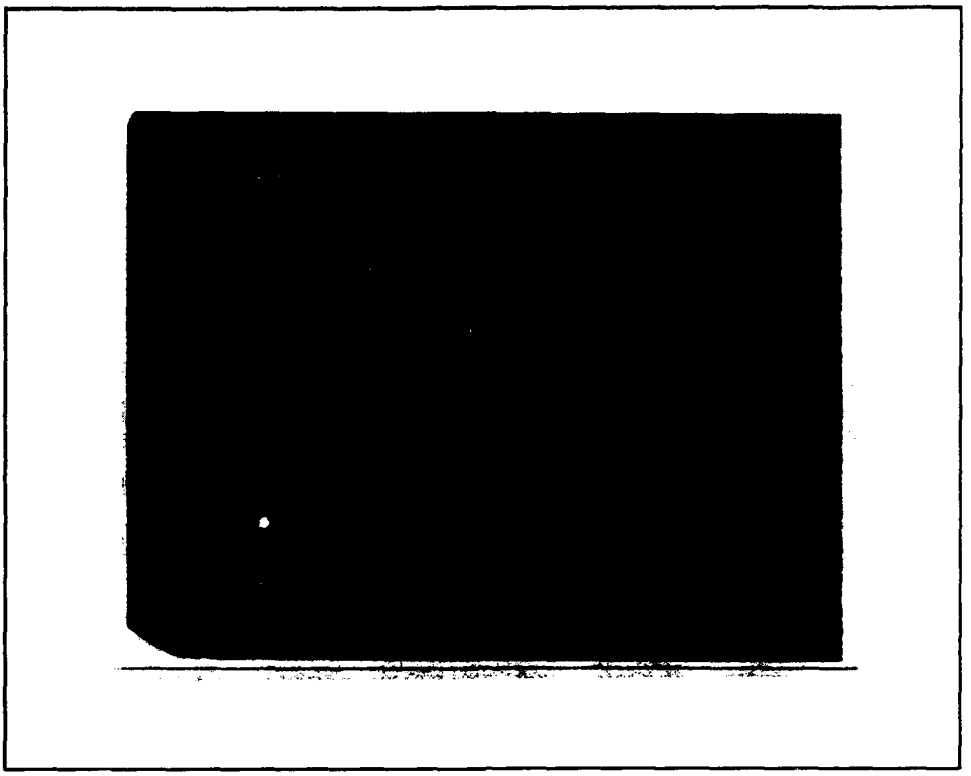


Figure 31 Spectrum of the output signal from the DAT. (The intermodulation products are under the measurement floor of the spectrum analyzer.)

B. AUDIO RECORDING PERFORMANCE

The second important quality required from the system is the accuracy of recording an external signal. The parameters measured are:

- Signal to noise ratio.
- Recording linearity.

The measurement procedure was as follows. A signal generator was connected to the input as a source providing a 1kHz sinusoidal signal. The signal was recorded into a file on the workstation disk. The spectrum of the signal was measured by the HP

spectrum analyzer to serve as a reference, and the monitor of the spectrum analyzer was photographed. The signal was then retrieved from the file into MATLAB. In MATLAB an FFT was performed, and the result of the FFT was compared to the input signal spectrum. The recording linearity of the system was measured by a similar procedure, but this time a combination of two signal generators having two discrete frequencies were used as the source.

The SUN workstation audio input is a microphone level input, so the source signals were injected to the preamplifier directly from the signal generators, the signal was amplified and send to the speakers, the microphone was put near one speaker, and the signal was recorded. The measurements are summarized in Table VII.

Table VII AUDIO RECORDING PERFORMANCE MEASUREMENTS.

Measured parameter	SUN audio tool recording 8-bit μ-law 8 kHz sample rate	DAT input recording 16-bit linear 48 kHz sample rate
Signal to noise ratio.	50 dB direct input 20 dB using microphone	greater than 85 dB using direct input recording.
Highest intermodulation product.	-30 dB	- 79 dB

Figure 32 shows the FFT of the input signal recorded through the SUN microphone. The audio signal driving the microphone was from the speakers when a 1 kHz signal was input to the preamplifier. There are a lot of undesired spectral lines probably originating from the equipment noise in the room (neon noise, computer cooling fan noise, and other ambient noise). We expect the signal would be considerably cleaner if it could be injected electrically. Unfortunately the "line in" of the speakerbox is not supported by the SUN software, and injecting the signal from the signal generator directly into the microphone input can produce many undesired harmonics due to the large impedance mismatch which drives the input of the speakerbox to nonlinear operation.

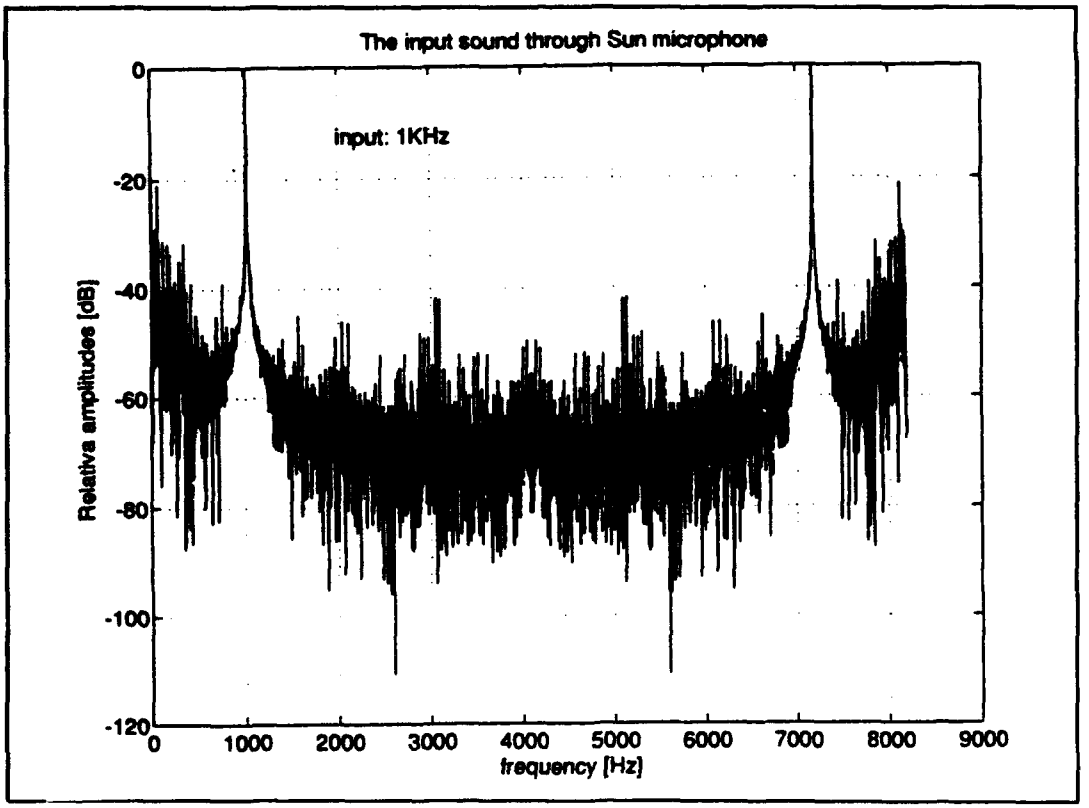


Figure 32 FFT of the 1 kHz input signal through the SUN workstation microphone.

Figure 33 illustrates the magnitude of the FFT of the 1 kHz signal recorded through the DAT input as computed in MATLAB. Although there are some harmonics that accompany the signal, these harmonics originated in the signal generator. Figure 34 shows a photograph of the signal generator input as measured by the audio spectrum analyzer. The FFT of the signal in MATLAB follows the input signal very accurately. The values of the FFT were normalized to the peak power value. Both measurements used a rectangular window.

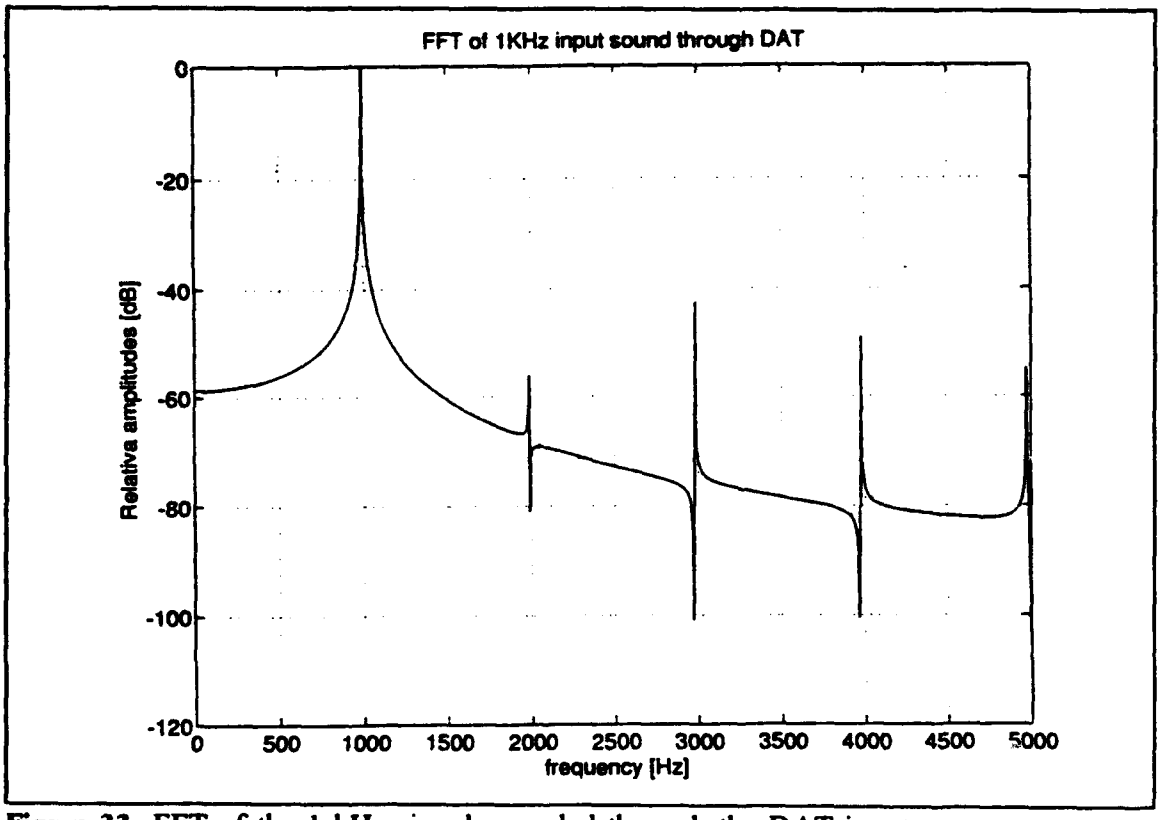


Figure 33 FFT of the 1 kHz signal recorded through the DAT input.

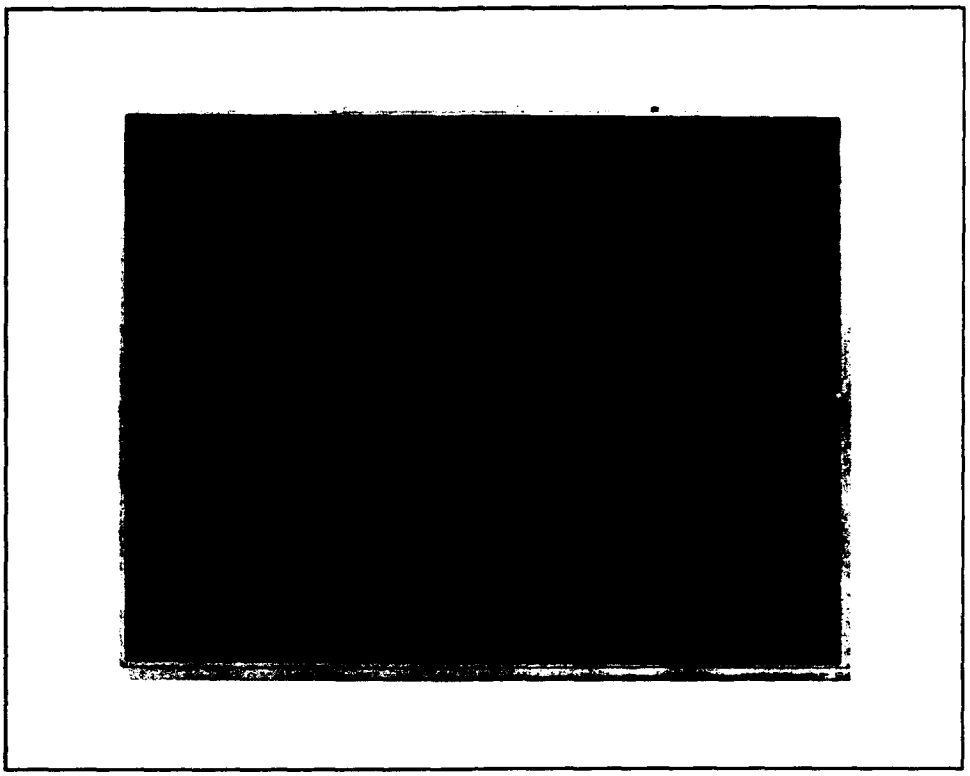


Figure 34 Photograph of the spectrum of the 1 kHz input signal to the DAT

Figure 35 illustrates the FFT of the signal comprised of two discrete frequencies that was recorded through the SUN microphone. The spectral line noise is again high. The first intermodulation difference product is marked with a point and an arrow. There are also spectral lines in the 60 Hz vicinity that are very high (20 dB below the signal) which are generated by the equipment in the recording room.

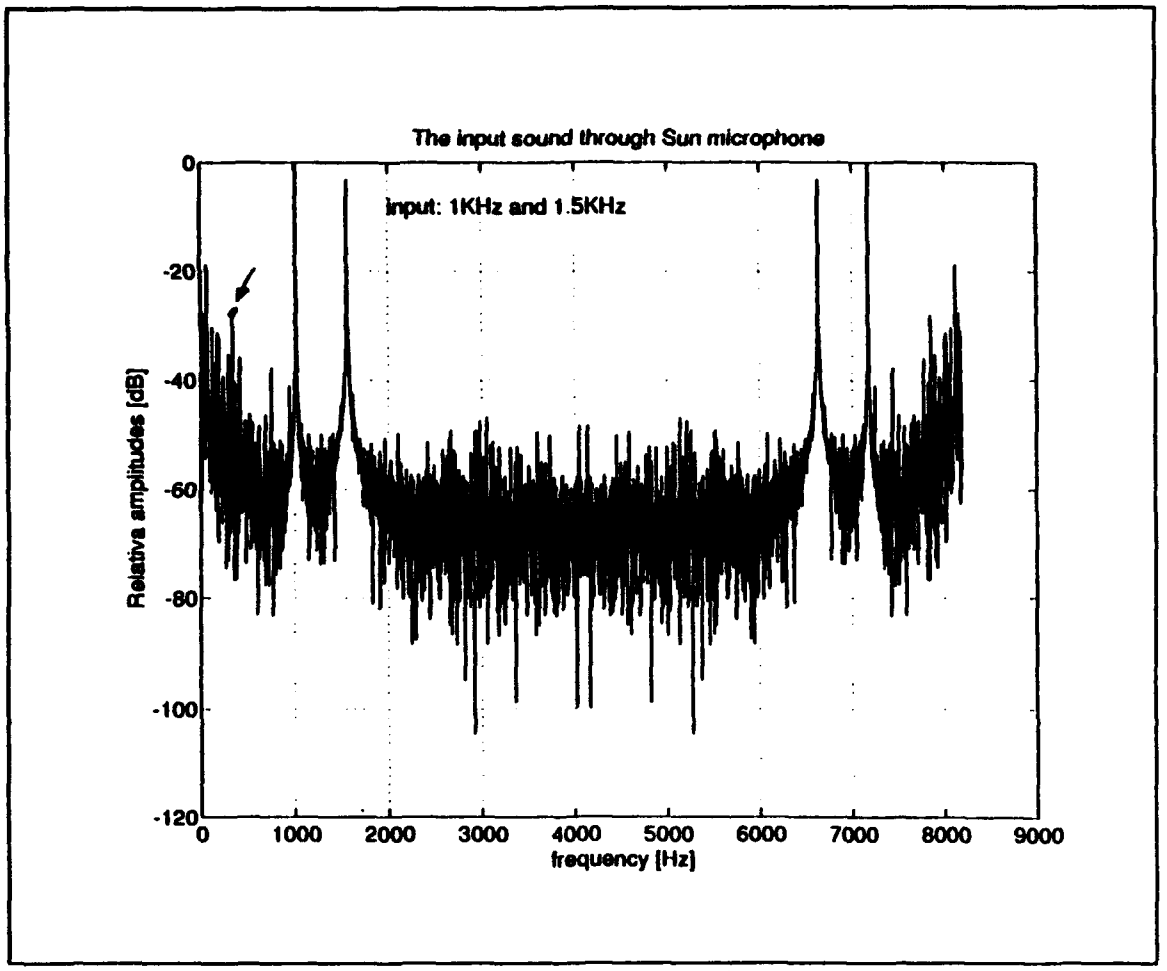


Figure 35 FFT of the two-frequency signal recorded through the SUN microphone input.

Figure 36 illustrates the FFT of the signal that was recorded through the DAT input. The signal is comprised of two frequencies, 1 kHz and 1.5 kHz. The signal was injected into the DAT analog input and then sent to the SUN workstation through the interface. The sample rate used is 48 kHz. The harmonics that appear in the FFT of the signal again originate in the signal generators. The signal intermodulation product that can be related to the DAT is the signal at the frequency 3.5 kHz, which is 78 dB blow the 0 dB reference level. Figure 37 shows the spectrum of the input signal as measured by the HP spectrum analyzer. The FFT of the signal in MATLAB and the spectrum analyzer both used a rectangular window. As mentioned earlier, direct connection of the signal

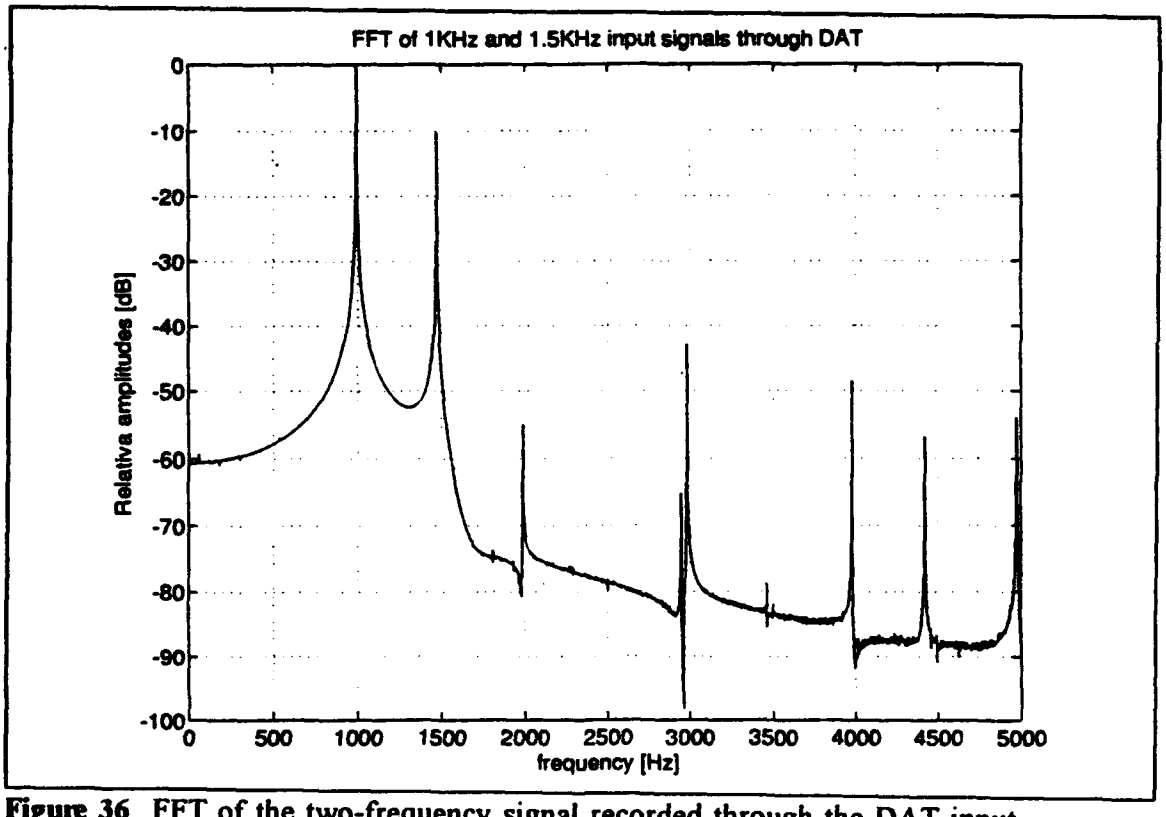


Figure 36 FFT of the two-frequency signal recorded through the DAT input.

generator output to the microphone input leads to production of multiple harmonics and is thus unsatisfactory.

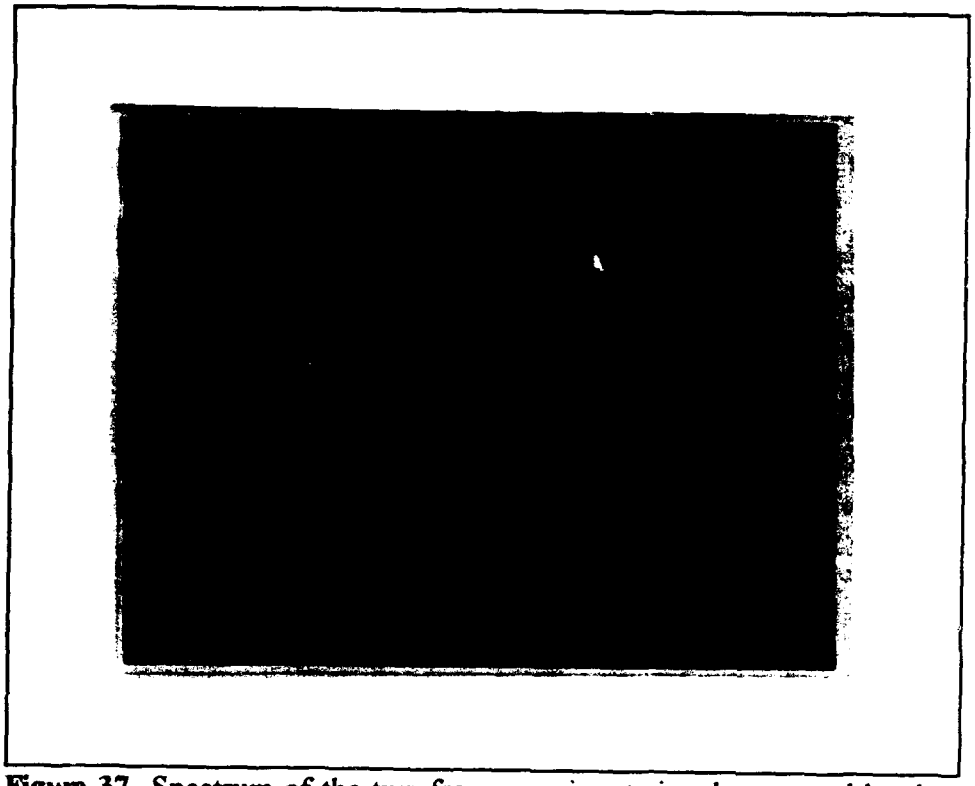


Figure 37 Spectrum of the two-frequency input signal measured by the spectrum analyzer.

Figure 38 illustrates the FFT of the two-frequency signal when it was injected directly into the microphone input. The original signal consists of 1 kHz and 1.5 kHz sinus waves. The many intermodulation products are due to the nonlinear operation when the signal generator is directly plugged into the SUN station microphone input.

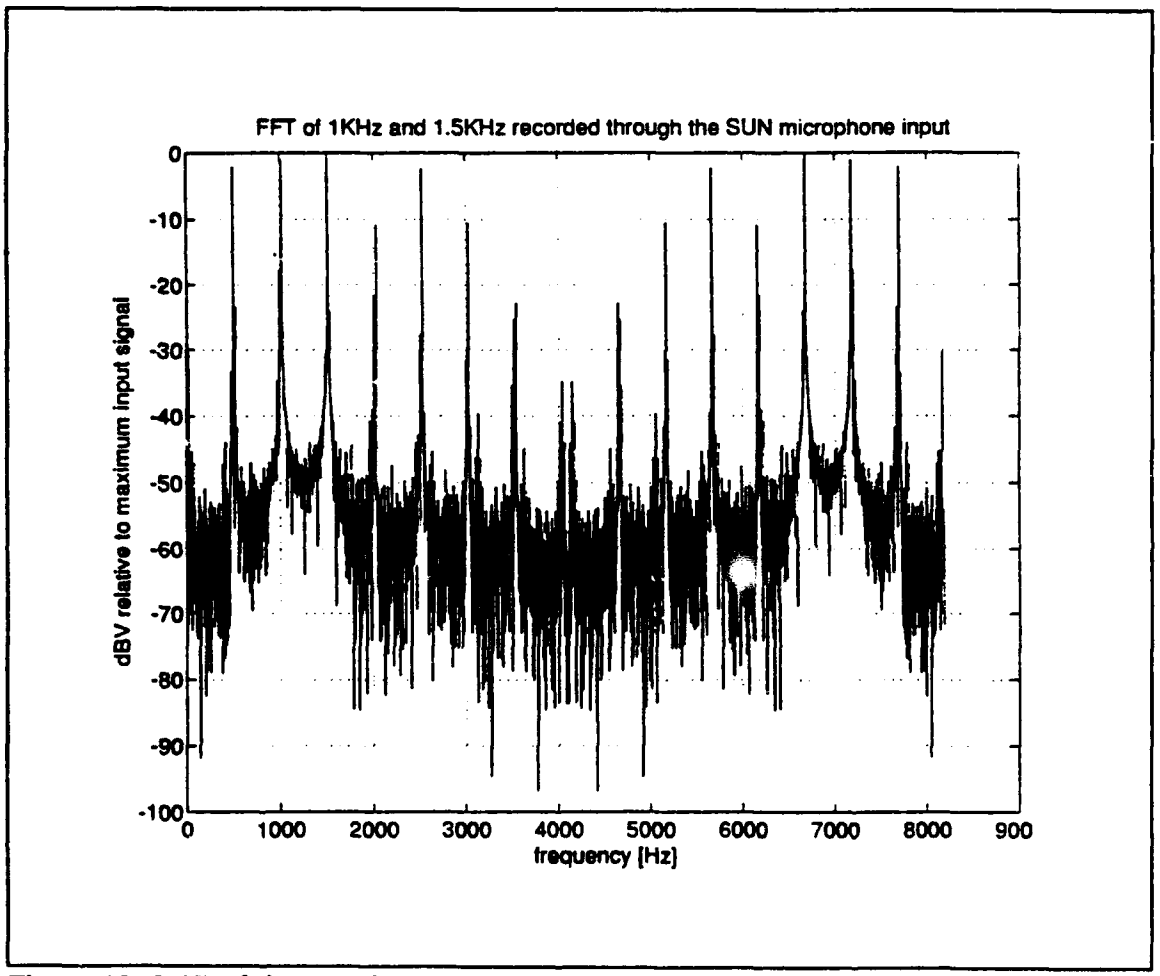


Figure 38 FFT of the two-frequency input signal recorded through the SUN microphone input.

C. SOFTWARE PERFORMANCE TESTS

The system operating software was tested for proper operation. The test procedure for the software is mainly operational. The system audio recording and reproduction performances were tested for proper undistorted operation.

Software counters were introduced in the main programs (during the development and debug stages) to indicate whether the application is fast enough in order not to lose any data sent in realtime. The counters indicated the number of times that the main C program tries to reach a buffer of data until the buffer is ready to be transferred by the realtime process that runs on the TMS board. The counter values varied with the change in the sample rate and the bit mode as expected. (The 32-bit mode is twice as fast as the 16-bit mode, and the data buffers are filled in half the time.)

The programs were allowed to record for various time durations, and the audio result of the recording was monitored for proper reproduction. The general results are that the system operates without any loss of data for about 30 seconds for the highest sample rate (48 kHz and 32-bit mode). The length of proper operation of data transfer is related to the application load on the system. When the workstation was loaded by many processes the proper operation time was reduced; when there were only a few tasks and none of these are time demanding the proper operation time was extended. The Windows background program was found to place a medium load on the system even if some of the applications were in the icon format. Better performance and longer uninterrupted recording time weree achieved when the program was run outside of the windows environment.

The header synchronization was tested for accurate synchronization of a segment. When the header was used the accuracy of the segment start was found to be precise to within one sample.

The last test was the extraction of the AES/EBU or SPDIF framing data from some audio samples recorded with the AES/EBU format or with the SPDIF format and comparing this data to the standard tables of the format described in Chapter II. The results of the framing data extraction showed perfect match to the standards.

V. CONCLUSION

The system developed and investigated in the course of this thesis considerably enhances the audio recording and reproduction capability of the SUN workstation used in the DSP lab for sound processing. The use of a smaller quantization step and extended sample word as well as the higher sample rate was proven to provide better signal-to-noise ratio and larger bandwidth for the analog audio signals. The audio reproduction performance was enhanced by more than 40 dB, and the recording performance was enhanced by more than 30 dB. In addition, using 16-bit linear quantzation improves the system linearity, which is of great importance to the audio evaluation of signal processing algorithms. The performance achieved assists in capturing audio input data much more precisely and cleanly for later processing as well as provides exceptionally clean output for monitoring and direct digital recording. The number of options for recording and reproducing audio data were increased, and the audio tools were integrated into a complete hardware/software system.

The real time operating regime, which the TMS interface board uses, demands programming considerations that are much different from a non-realtime C program. Two methods of handling data transfer service requirements, namely, the interrupt method and the polling method, were tested in the course of the software development. The polling

method proved to be more efficient for this application and provided higher undisturbed throughput to the non-realtime UNIX application running on the host workstation.

The UNIX environment in which the new audio application tools perform is not the ideal operating system. Although the throughput of the workstation S-bus is sufficient to support the high data rate required for High Fidelity audio transfer associated with the DAT format, the non-realtime UNIX operating system does not support this application in an optimal way. The host program running under UNIX and providing the data transfer has no control over the scheduling of time slices and is subject to interruption and subsequent loss of data. The maximum uninterrupted audio data transfer time is thus limited and depends on the applications load on the system. For direct digital recording of data segments lasting more then a half minute, therefore, it is advisable to ask remote users to log off of the workstation and to operate from the command line (i.e., not under the windows environment) to minimize interruptions.

To avoid some of these disadvantages, two approaches could be considered:

- Using some other (real time) operating system or modifying the kernel of the UNIX system in such a manner that will give higher priority to the host program controlling the realtime application.
- Enlarging the DPRAM buffer space on the interface board to the order of 100k words to compensate for the time loss introduced during application switching by the operating system.

The system as tested performed very accurately for short duration audio signals, such as transient signals or short speech segments however.

APPENDIX A. OPERATING INSTRUCTIONS AND USER GUIDE

Appendix A describes the operation of the system and can be used as a guide for the beginning user. The user is advised to read the DAT PCM 2700A operation manual and the analog tape user manual for proper operation of these devices.

A. CAPTURE and DIGITIZATION OF AUDIO DATA

This section describes the hardware setting and the software operation in order to save an audio segment onto the system disk for later use.

1. Hardware setting

The system can record the following audio sources:

- Audio segments prerecorded on a DAT cassette in digital format.
- Audio segments prerecorded on analog cassette.
- Speech using a microphone connected to the analog tape deck, which is amplified and then sent to the DAT for digitization.

In all of the operation modes the audio is sent digitally from the DAT to the workstation.

a. Capturing from the DAT

Capturing a digital audio segment prerecorded on a DAT cassette is the simplest mode of operation. The software procedure to save the data in digital form is

started (see subsection 2. below) and the DAT is operated in playback mode at the appropriate time.

b. Capturing sound from the analog cassette player

To capture an analog audio segment from the analog tape the hardware setting is as follows:

1) The DAT input selector is set to analog input. Then the INPUT MONITOR switch on the front panel is set to INPUT.

2) The analog cassette player is then operated in the playback mode. The audio levels should be adjusted not to reach the overload level of the analog tape and the DAT input.

The tape is then played back to the software that saves the data in digital form.

c. Speech capture using a microphone

The following setting should be made to capture speech or sound through the microphone:

1) A microphone is connected to the microphone input of the analog tape deck.

2) The analog tape deck is set to RECORD mode with the PAUSE button pressed.

3) The input level is adjusted not to exceed the overload input limit.

4) Steps 1,2 of subsection b above are then reapeted.

The sound is then recorded through the microphone to the software that saves the data in digital form. Note if the microphone output level is low a line amplifier is required to be connected in series between the microphone output and the tape deck input.

2. Software operation

The software is operated after the proper hardware audio settings have been made (see above). The recording program called "run_save" should be accessible from every directory for any user in the DSP SUN network. (If this is not the case notify to the system administrator.) The software is invoked by entering the program name (run_save) at the command prompt. The software is controlled from menus. (When the program is invoked it automatically checks to see if the interface board is responding and if no other application is using the interface; if the board is not ready the program is terminated with an error messege to the user). The first menu screen is:

Select mode of operation:
1 - AES/EBU 16 bit
2 - AES/EBU 32 bit
3 - SPDIF 16 bit
4 - SPDIF 32 bit

The default is the AES/EBU 16-bit mode, which is most used. AES/EBU 32-bit mode is used only if the user wishes to store the framing information along with the audio data. Using modes 3 or 4 (SPDIF) requires other DAT settings on the rear panel (see DAT instructions manual). The second menu question is the length of the sound segment to be captured in seconds. A number is expected here (not necessarily an integer). The time is rounded to the nearest 10 or 20 millisecond segment depending on the bit mode. The user *is not* asked about the sample rate, this parameter is adjusted automatically by the hardware of the interface. For prerecorded digital data this parameter is the same as the data recorded; for data recorded from the analog tape deck or microphone the sampling rate is set manually on the DAT deck. The third menu question asks whether a header is

prerecorded with the audio segment. The header choice should be selected *only* if a header already exists. If the user choses the header option when a header was not prerecorded the recording is not executed and the program hangs. (Use ^C to terminated the program.) On completion of recording the user is informed of the length of the captured audio segment in words and asked if the captured data should be saved. If the answer is NO the program is terminated. If the answer is YES (the default) the user is asked for a file name. The user should enter the name of the file without extension. The extension ".dau" is added automatically. The data is stored in the user's current directory and the user is prompted about the form in which the data is to be stored in the file.

B. PLAYBACK OF AN AUDIO FILE AND RECORDING ON TAPE

The playback procedure is similar to the sound capturing procedure but the steps are reversed in order. The playback modes are:

- Playback of an audio file for listening or monitoring.
- Playback of an audio file and recording it on the DAT.
- Playback of the audio file and recording it on the analog tape.
- Direct digital dubbing mode. This mode requires a second DAT and is for hardware check.

1. Harware setting

The hardware setup in the playback mode depends on the mode of operation. The digital audio data transfer is from the SUN workstation to the DAT via the digital inputs of the DAT. The PCM 2700A professional DAT deck has two digital audio inputs.

The input selection function is split between the rear panel switch (which determines whether an AES/EBU or an SPDIF format DAT will be the input) and a switch on the left side of the front panel (which determines whether an analog signal or a digital signal is the input, and the sample rate in the case of analog input signals). The selector positions should be set on AES/EBU for the rear switch and DIGITAL INPUT for the front switch.

a. Playback for monitoring setup

The setup for this mode is done by the following steps:

1) Set the INPUT selector on the rear panel to AES/EBU.
2) Set the INPUT MONITOR switch to INPUT.
3) Set The INPUT selector on the front panel to DIGITAL.
4) Adjust inputs 1 and 2 of the preamplifier/mixer and the master volume control to the desired sound level.
5) Start up the software to play the desired sound (see subsection B.2.).

b. Playback and recording an audio segment on DAT

To playback and record an audio segment on DAT, first follow steps 1 to 3 in subsection a. Then proceed as follows:

1) Press the RECORD button on the front panel. This will set the DAT to the PAUSE mode.
2) Start up the software (see subsection B.2. below) and proceed through the menu.
3) When prompted to start the DAT press the PLAY button.

4) At the end of recording press the STOP button.

c. Recording on analog tape

To record on analog tape first follow steps 1 to 3 of subsection a. Then proceed as follows:

1) Set the input of the analog tape deck to rear balanced input.

2) Press the RECORD and PAUSE buttons simultaneously. The tape deck should enter the record mode but the tape should not be moving

3) Start up the software (see section B.2.) and proceed through the menu.

4) Press the play button when prompted by the software.

5) At the end of the recording press the STOP button.

Notes:

- The header option is not usable when recording a signal on the analog tape.
- The input recording level should be set in advance by operating the software when the deck is in the record pause mode before recording in order not to exceed the overload level. The user is advised to make some trials to set the optimum level.

d. DAT dubbing mode

The last operating mode is DAT dubbing for hardware check. In this mode two DAT recorders are required, one as a source for digital audio data and the other as a receiver. Normally for this mode the source DAT is operated with SPDIF output (most commercial DAT recorders are adequate for this purpose) and the SPDIF input cable from the interface are connected to the digital output of this DAT. The receiver

DAT is the PCM 2700A which is normally operated in AES/EBU mode. It is possible that another professional deck could be used as the source that provides AES/EBU (balanced output). In this case the AES/EBU input cable from the interface would be disconnected from the PCM 2700A and run to the source DAT and the SPDIF or AES/EBU output cable would be left on the PCM 2700A which would then be operated either in SPDIF or in AES/EBU input modes. The source DAT is set to play and the receiver DAT is set to record. Digital data is then transmitted through the interface.

2. Software operation

The software for playback and record procedure is called "run_play" (see details in Chapter III section C). This program like the "run_save" program should be available from any user directory and is also menu driven. In the first menu screen the user is asked for the desired mode of operation.

This is the Main Menu. Please enter the desired option number:

1 - 16 Bit Mode file sent to the DAT.
-------- This will set the sample rate to 48 kHz ------
2 - 32 Bit Mode file send to the DAT.
------chose this option if the data sample rate is other then 48 kHz ---
3 - DUBBING from one DAT to another (for hardware check)
-----this sets the 32-bit mode automatically -----

The subsequent events are based on the option selected.

Option 1 - 16-bit mode

After chosing this option the user is asked for the file name. The file name should be entered without the extension which defaults to ".dau". The user is then asked whether to add the special header pattern to the recorded segment (yes/no answer is expected). The

user is then prompted to answer that the DAT is ready. Hitting the ENTER key causes the playback to begin.

Option 2 - 32-bit mode

When this option is chosen the user is asked for the file name and then asked to select a sample rate (multiple choice). Then the user is asked whether to include a header (see the 16-bit mode above) and finally if the data file contains framing information to be sent to the DAT. If the file does not contain framing information default framing is created by the program. This option enables the user to play back data recorded in the 16-bit mode at sample rates other than the 48 kHz standard. The user is then prompted to answer that the DAT is ready. Hitting the ENTER key causes the playback to begin.

Option 3 - DUBBING from one DAT to another

This is the hardware check option and is not normally selected. When it is chosen the user is then asked about the source DAT, whether the source DAT is connected to the AES/EBU port or the SPDIF port, and should respond accordingly. (See section 1.d. above.)

C. DUBBING FROM ONE TAPE TO THE OTHER AND RECORDING EXTERNAL ANALOG SOURCES

Since the output of each tape in the system is connected to the input of the other tape deck (see Figure 11), dubbing from one tape deck to the other is done by playback of the source tape and operating the other tape in the record mode. When dubbing to the

DAT, analog balanced input should be selected on the DAT. When the audio signal source is from an external device the input channels of the destination tape should be disconnected from the matrix and connected to the external audio signal source. (Refer to the tape deck operation manuals for regular recording procedures.)

APPENDIX B. THE DAUGHTER MODULE REGISTERS

This appendix summaries the configuration of the control registers of the daughter module and the information that can be obtained from the status registers. The control and status registers are mapped in the AMELIA address space and the programming of the registers is crucial for proper execution of the TMS programs. The addresses referred to later are relative addresses in the interface board address space. The data that is referred to in the various registers is in the *upper 16 bits.*

A. THE CONTROL AND CONFIGURATION REGISTERS

The Control and Configuration registers are the registers that the program writes to. For proper operation of the daughter module the sequence of programming is; programming the User Control Register then programming the AMELIA Control Register, and finally programming the Configuration Register.

1. User control register

The User Control Register address is 804008 Hexadecimal (H) and the content that should be written to the register is A000 H.

2. Configuration register

The Configuration Register address is 80400F H and the content that should be written to the register is 8FF8 H. This register should be programmed after the AMELIA Control Register.

3. AMELIA control register

This register sets all the operating modes for the data transfer. The register address is 80400A H; Table VIII summaries the configuration of the register.

Table VIII *AMELIA* CONTROL REGISTER CONFIGURATION.

bit 7 CM 7	CM 6	CM 5	CM 4	0	CM 2	CM 1	bit 0 CM 0

CM1	CM0	Transmit Frequency
0	0	PLL Clock
0	1	32 kHz
1	0	44.1 kHz
1	1	48 kHz

CM2	PLL Input
0	Receiver word clock
1	External word clock

CM4 & CM6	Mode of Operation
0	32 Bit Mode
1	16 Bit Mode

CM5	Receiver Input
0	SPDIF
1	AES/EBU

CM7	AMELIA Status Register to be read
0	Status Register I
1	Status Register II

Bits CM0 and CM1 determine the sample rate sent out to the DAT. In the dubbing and receive modes the rate is determined by the PLL. Bit CM2 sets the PLL input to lock on the DAT data or other external word clock. In the "sp2ae_save" program it is set to be locked on the received data. Bits CM4 and CM6 determine the mode of operation; the user indirectly sets these bits according the desired 16- or 32-bit mode selection. Bit CM5 is the input source to the interface corresponding to whether one uses a SPDIF or an AES/EBU DAT. Bit CM7 indicates what information is presented when reading the Status Register address. (It is detailed in the Status Register section below.)

4. Interrupt register

This register address is 80400B H for writing and reading. The configuration of this register is summarized in Table IX.

Table IX INTERRUPT REGISTER CONFIGURATION.

bit 2 UNLOCK	bit 1 TDE	bit 0 RDF

RDF **Input Data Registers full.**
Channels 0 and 1 are ready to be read
RDF bit is used when the interface is in receive mode.

TDE **Output Data Registers empty.**
Channel 0 and 1 are ready to be written again.
TDE bit is used when the interface is in transmit mode.

UNLOCK **The PLL loses lock on input signal.**
This bit is used for monitoring the PLL.

The Interrupt register is used both for writing and reading. Writing '1' to a particular bit enables an interrupt to be issued from this source; hence it is used as Mask register. Reading the Interrupt Register gives the status of the register and clear its content. The Interrupt Register can be used when programming in the interrupt methodology or the polling methodology.

B. STATUS REGISTERS

The Status Register address is 80400A H (same as for the AMELIA Control Register). There are two registers: Status Register I (SI_) and Status Register II (SII_). Access to the content of the registers I or II is determined by CM7 of the AMELIA Control Register (Section A.3).

1. Status register I

Status register I can be read when bit CM7 of the AMELIA Control Register is set to '0'. The content of the register is specified in Table X. SI0 and SI1 are important when working in the 32 bit mode to determine what part of the whole 32 bit word is being transferred. The word is manipulated according to this information.

Table X STATUS REGISTER *I* CONFIGURATION.

bit 3	bit 2	bit 1	bit 0
SI3	SI2	SI1	SI0

SI0 & SI1 are used only in the 32-bit mode.

SI0	**Data Registers Ch0 and Ch1 receive mode**
0	First 16 bits of serial data have been received in data input registers.
1	Second 16 bits of serial data have been received in the registers.
SI1	**Data Registers Ch0 and Ch1 transmit mode**
0	First 16 bits should be loaded in the Ch0 and Ch1 output registers.
1	Second 16 bits should be loaded in the Ch0 and Ch1 output registers.
SI2	**Digital Audio Input Present**
0	Valid signal is not present at the receiver.
1	Valid signal is present at the receiver.
SI3	**PLL locked**
0	PLL is not locked on the input signal
1	PLL is locked on the input signal.

2. Status register II

Status register II can be read when bit CM7 of the AMELIA Control Register is set to '1'. The content of the register is specified in Table XI. Bits SII0 and SII1 refer to the 32 bit mode and to the C flag. These bits are CRC check of the entire C flag block. SII2 and SII3 contain the information about the sample rate of the received signal.

Table XI STATUS REGISTER *II* CONFIGURATION.

bit 3 SII3	bit 2 SII2	bit 1 SII1	bit 0 SII0

SII0 & SII1 are used only in the 32-bit mode.

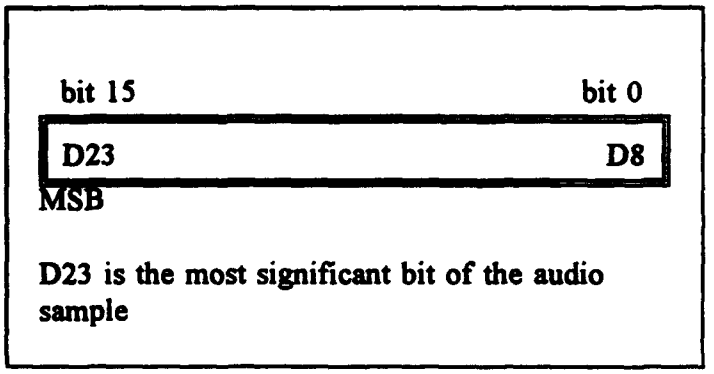
SII0	**CRCCfor Channel Status Block 0**
0	Invalid CRCC for last block of Channel 0 data to receiver
1	Valid CRCC for last block of Channel 0 data to receiver
SII1	**CRCC for Channel Status Block 1**
0	Invalid CRCC for last block of Channel 1 data to receiver
1	Valid CRCC for last block of Channel 1 data to receiver

SII2	**SII3**	**Input Signal Sampling Rate**
0	0	No signal detected by the receiver.
0	1	32 kHz
1	0	44.1 kHz
1	1	48 kHz

C. DATA REGISTERS

The data registers are located in addresses 804002 H and 804006 H for channels 0 and 1 respectively. The data registers contain the actual digital audio sample word. These registers are read in the receive mode when the data is sent from the DAT to the interface. In the transmit mode when the data is sent from the interface to the DAT the data registers are written to. The data registers are 16 bits long. In the 16-bit mode the registers contain only the 16 most significant bits of the digital audio sample. Table XII describes the register content in the 16-bit mode.

Table XII 16 BIT MODE DATA REGISTER CONTENT



In the 32-bit mode the content of the registers depends on whether it is the first 16 bits of the audio sample or the second 16 bits of the audio sample and whether the mode is transmit or receive. The content of the registers is illustrated in Figure 39 and Figure 40; the content of the flags is summarized in Table XIII.

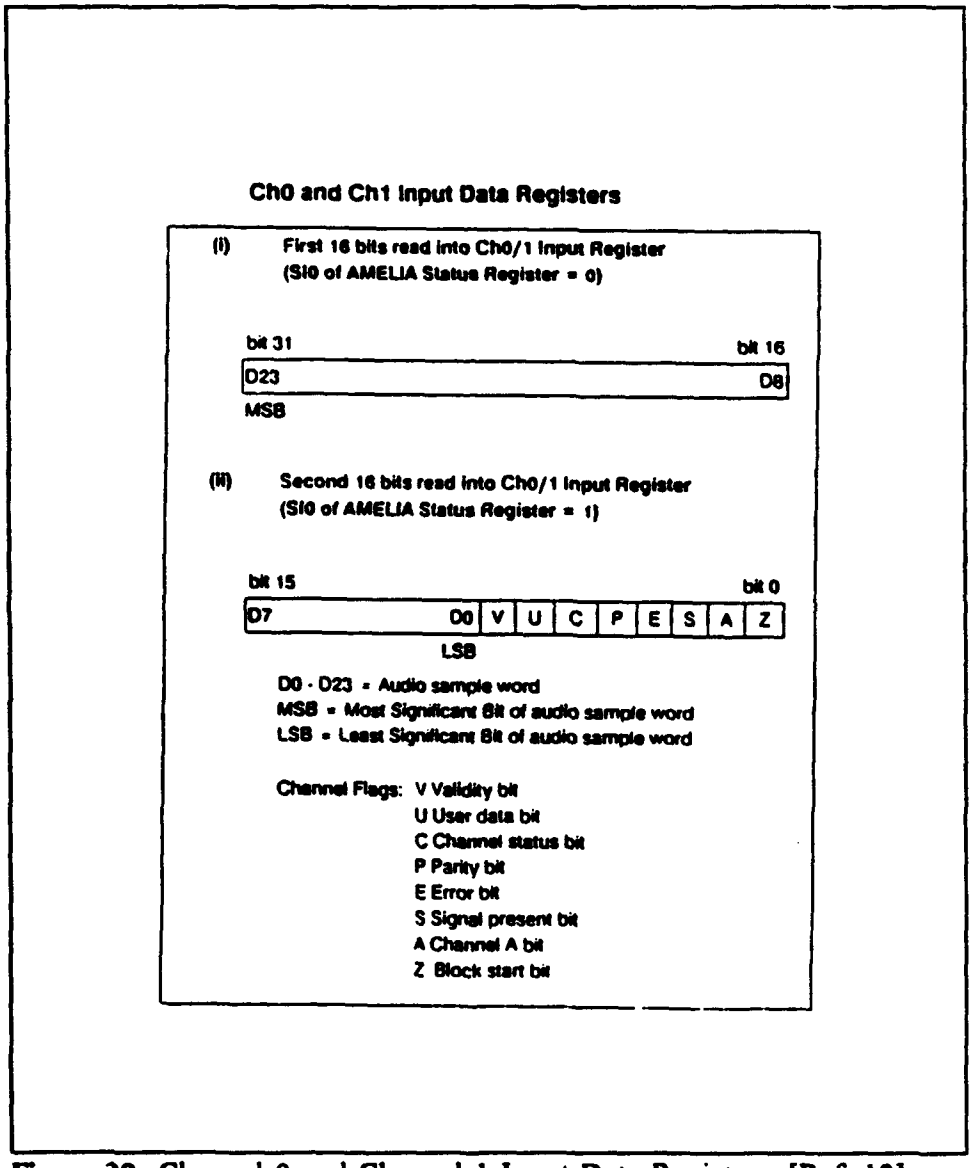


Figure 39 Channel 0 and Channel 1 Input Data Registers. [Ref. 12]

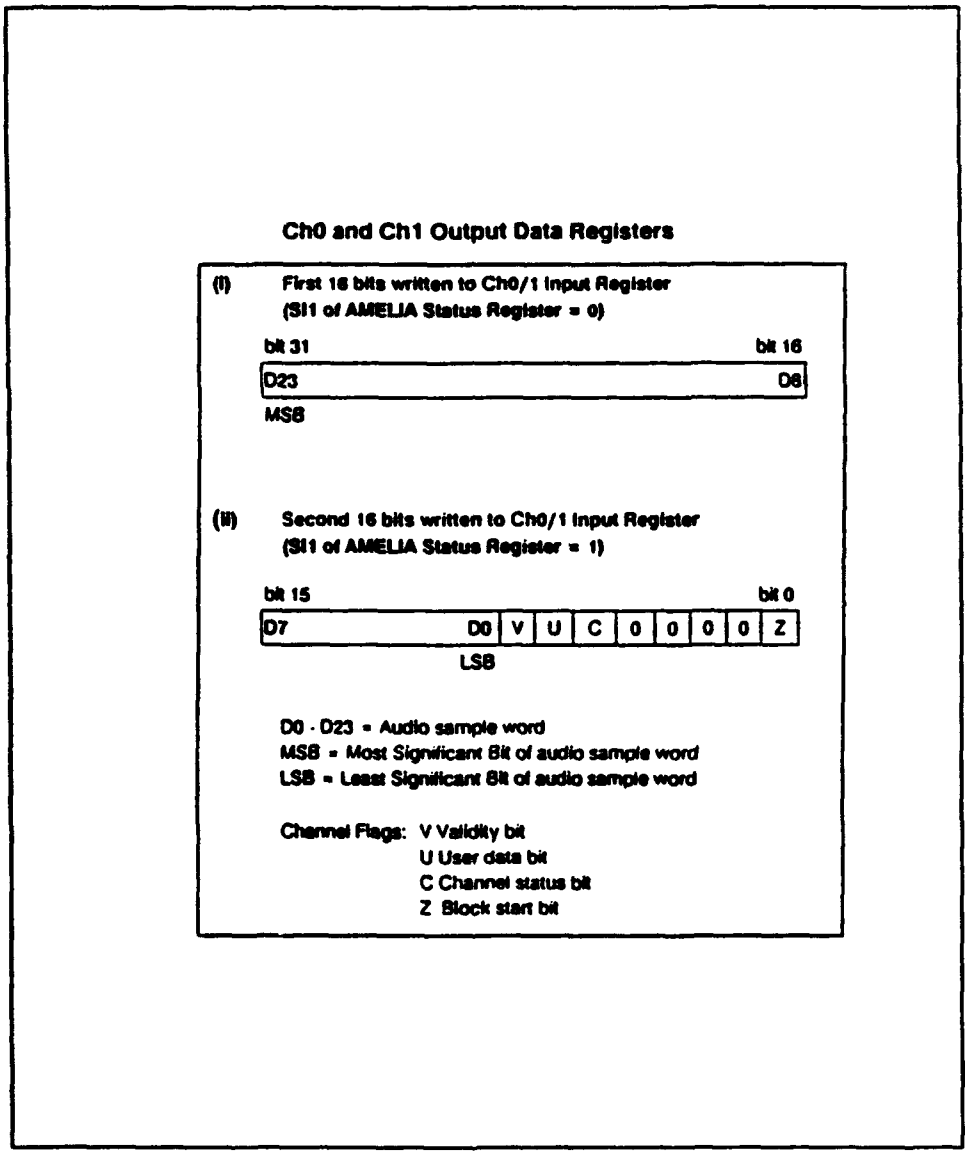


Figure 40 Channel 0 and 1 Output Data Registers. [Ref. 12]

Table XIII CHANNEL REGISTER FLAGS

Channel Flag	Channel 0,1 Input Registers	Channel 0,1 output Registers
V Validity	0 Valid digital audio sample 1 Invalid digital audio sample	
U User data	Used to carry information determined by the user.	
C Channel status	The configuration of this flag as dictated by the AES/EBU or SPDIF standards.	
P parity	1 Even parity 0 Odd parity	Not used
E Error	1 Error detected 0 No error	Not used
S Signal present	1 Signal present 0 Signal not present	Not used
A Channel A data	1 Channel A data received 0 Channel B data received	Not used
Z Block start	1 Sample is the first in the Block 0 Sample is not the first in the Block.	

APPENDIX C. SOFTWARE LISTINGS

This appendix includes the complete operating software all of which is written in C. The main programs are "run_save.c" for recording and "run_play.c" for playback. The TMS board programs are "sp2ae_save.c" and "sp2ae_play.c"

A. MAIN PROGRAM "run_save.c"

```
/* run_save.c */
/* program which sets up C30 and transfers */
/* double buffered data to the host via DPRAM */
/* Author: Arie Gal _Gartenlaub  */
/* last update: March 3rd 1994 */
/* Version: 1.00 */

#include "sdsp30lib.h" /* the interface library header file */

#define READY           0xeeeeeeee
#define NOT_READY       0x11111111
#define READY_A         0xeeee0000
#define READY_B         0x11110000
#define HIGH_MASK       0xffff0000
#define LOW_MASK        0x0000ffff

/* now define DSP DPRAM addresses */
#define BUFFER_A_STATUS_ADDRESS     0x0400000
#define BUFFER_B_STATUS_ADDRESS     0x0400001
#define BUFFER_A_BASE_ADDRESS       0x0400010
#define BUFFER_B_BASE_ADDRESS       0x0400408
#define FIND_HEADER_ADDRESS         0x040000c
#define MODE_OF_OPERATION           0x040000d
#define PLL_STATUS_ADDRESS          0x040000e
#define DP_RAM_DATA_BUFFER_LENGTH   0x3c0
#define times                       6000

#define SPDIF_16        0x00500000 /* AMELIA Control DATA for SPDIF input 16 bit */
#define AES_EBU_16      0x00700000 /* AMELIA Control DATA for AES/EBU input 16 bit */
#define SPDIF_32        0x00000000 /* AMELIA Control DATA for SPDIF input 32 bit */
#define AES_EBU_32      0x00200000 /* AMELIA Control DATA for AES/EBU input 32 bit */

#define MODE_16     1
#define MODE_32     2

FILE  *fp;
typedef struct node{unsigned long audio_array[2*DP_RAM_DATA_BUFFER_LENGTH];
                    unsigned long data_ready;
                    struct  node *next;
                    }block;

block *new_block()
{
  return((block *)malloc(sizeof(block)));
}
```

```
void main()
{
      char  error = 0 ;
      char  file_name[100];
      char  ch;
      block *head_ptr,*temp_ptr,*last_ptr;
      int buffA_count[times];
      int buffB_count[times];
      unsigned long temp,start_save,zero_detect,memory;
      unsigned long mode,flag,header,save;
      int choice,block_count,get_out,gen_counter=0;
      int min_wait,i,k,l,blocks,step,seconds,counter;
      float sec;

/* First check if the DSP board is responding */

      error = Select_Board("/dev/LSIsdsp300");

      if (!error) {

/* The board is selected  */

/* Hold the processor                                    */
error = Hold();

/* Now load the file into the TMS */
error = Load_Object_File("sp2ae_save.out");
if(error==SLIB_ERR_LOADF)
  { /* the file was not found so get out */
  (void)printf("\n Can't find sp2ae_save.out which is the TMS320 file");
  exit(8);
  }

/* Send mode of operation */
printf("\n Select input source and mode of operation : ");
printf("\n 1 - AES/EBU 16 bit");
printf("\n 2 - AES/EBU 32 bit");
printf("\n 3 - SPDIF 16 bit");
printf("\n 4 - SPDIF 32 bit \n==> ");
scanf("%d",&choice);
switch(choice)
{
case 1: printf("\n You chose AES/EBU 16 bit");
        Put_32_Bit(MODE_OF_OPERATION,AES_EBU_16);
        mode=MODE_16;
        break;
case 2: printf("\n You chose AES/EBU 32 bit");
        Put_32_Bit(MODE_OF_OPERATION,AES_EBU_32);
        mode=MODE_32;
        break;
case 3: printf("\n You chose SPDIF 16 bit");
        Put_32_Bit(MODE_OF_OPERATION,SPDIF_16);
        mode = MODE_16;
        break;
case 4: printf("\n You chose SPDIF 32 bit");
        Put_32_Bit(MODE_OF_OPERATION,SPDIF_32);
        mode = MODE_32;
        break;
default : printf("\n The Default is chosen and it is AES/EBU 16 bit.\n");
          Put_32_Bit(MODE_OF_OPERATION,AES_EBU_16);
          mode = MODE_16;
          break;
} /* end of choice switch */

printf("\n Enter the desired recording time in seconds.(sample rate assumed 48kh)\n ==>
");
scanf("%f",&sec);
blocks=(int)(24000*sec*mode/DP_RAM_DATA_BUFFER_LENGTH);
printf("\n Initializing Data Base to get the digital audio from the DAT %d
blocks",(blocks+1));
head_ptr=new_block();
last_ptr=new_block();
last_ptr->next=NULL;
head_ptr->next=last_ptr; /* linking the memory modules */
temp_ptr=head_ptr;       /* a temporary pointer used to scan the array */
temp_ptr->data_ready = NOT_READY;
```

```
memory=READY;
block_count=0;
while( block_count++ < blocks  &&  memory==READY )
      {
      last_ptr->next=new_block();
      if (last_ptr->next==NULL)
            {
            printf("\n Out of memory after %d seconds.\n",block_count/(50*mode));
            memory=NOT_READY;
            printf("\n Do you wish to continue recording ? 1 - yes 0 - no");
            scanf("%d",&get_out);
            if (get_out!=1) goto end_of_record;
            }
      last_ptr->data_ready=NOT_READY;
      last_ptr=last_ptr->next;
      last_ptr->next=NULL;
      }
printf("\n Data Base is ready to get the digital audio %d blocks
initialized.\n",block_count);
printf("\n Does the audio segment have a start header?   y/n \n ==> ");
(void)rewind(stdin);
ch=getchar();
if(ch=='y'| ch=='Y'){Put_32_Bit(FIND_HEADER_ADDRESS,NOT_READY);
                  header=READY;}
        else {Put_32_Bit(FIND_HEADER_ADDRESS,READY);
                  header = NOT_READY;}
printf("\n Method of Operation is :");
printf("\n 1. Start tape about 2 seconds or more before the desired location.");
printf("\n Note: In case you use a header the tape should be started before the header.");
printf("\n 2. To start recording hit the Enter key.  ");
(void)rewind(stdin);
while((ch=getchar())!='\n') putchar(ch);
printf("\n Starting to record.");
/* Pulse the reset line to start the program running */
error = Assert_Reset();
error = Release_Reset();
/* UnHold the processor and the program will run     */
error = Unhold();
/* now loop round pulling buffers from DPRAM - 10 times only here */
block_count=0;
min_wait=2000;
while ( block_count <= blocks && block_count<times )
    {   /* the main while waiting loop */
    l=0;    /* l is the accumulative pointer for the audio */
    flag = 0;
    counter = 0;
/* communicate with the c_tst30 DSP program ....*/
    while (flag != READY) /* wait for buffer A to be ready to read */
         {
         error = Get_32_Bit( BUFFER_A_STATUS_ADDRESS, &flag);
         counter++;
         }
/* we have a ready flag for buffer A, so read the data*/
    if ( min_wait > counter ) min_wait=counter;
    buffA_count[block_count]=counter;
            error =
Get_Block_32(BUFFER_A_BASE_ADDRESS,&temp_ptr->audio_array[l],DP_RAM_DATA_BUFFER_LENGTH);
            l+=DP_RAM_DATA_BUFFER_LENGTH;
/* now wait for buffer B */
    flag = 0;
    counter = 0;
    while (flag != READY) /* wait for buffer B to be ready to read */
        {
        error = Get_32_Bit( BUFFER_B_STATUS_ADDRESS, &flag);
        counter++;
        }
/* we have a ready flag for buffer B, so read the data. */
    buffB_count[block_count]=counter;
    if ( min_wait > counter ) min_wait=counter;
            error =
Get_Block_32(BUFFER_B_BASE_ADDRESS,&temp_ptr->audio_array[l],DP_RAM_DATA_BUFFER_LENGTH);
            l=0;
    temp_ptr->data_ready=READY;
    temp_ptr=temp_ptr->next;
    block_count=block_count + 1;
```

```
    }   /****************** end of main while loop  */

/* Deselect the board to tidy up                        */
          /* Deselect_Board(); */
error = Hold();
error = Assert_Reset();
error = Deselect_Board(); /* use correct fn name ! */

/* and finally echo the counter values */
/*for(i=0;i<block_count;i+=20)
    printf("LOOP %d: buffer A with %d reads of 'flag', buffer B with %d reads of
'flag'\n",i, buffA_count[i], buffB_count[i]); */
printf("\n the minimum number of wait read cycles is %d ",min_wait);

printf("\n\n the audio array size - %d wish to save? enter 'y' for yes
",sizeof(head_ptr->audio_array)*block_count/2);
(void)rewind(stdin);
ch=getchar();
if(ch=='y' | ch=='Y')
    {    /* start of if save begin with an open file */
      (void)rewind(stdin);
      (void)printf("\n Enter name of file without extension.");
      (void)printf(" \n The extension added is : '.dau' \n ==> ");
      (void)fgets(file_name,sizeof(file_name),stdin);
      file_name[strlen(file_name)-1]='.';
      (void)strcat(file_name,"dau");
      fp = fopen(file_name,"wr");
      if(fp==NULL)
            {
              (void)printf("\n can't open %s, file system may be full \n",file_name);
              exit(8);
            }
    printf("\n Preparing data for storage.");
    temp_ptr=head_ptr;
    while(temp_ptr->data_ready==READY)
      { /* start of while not end of chain */
      for (l=0;l<2*DP_RAM_DATA_BUFFER_LENGTH;l++)
        { /* start of for local block */
            temp=temp_ptr->audio_array[l];
            (void)fwrite(&temp,1,sizeof(temp),fp);
        } /* end of for local block */
      temp_ptr=temp_ptr->next;
      free(head_ptr);
      head_ptr=temp_ptr;
      } /* end of chain while */
    free(temp_ptr->next);
    free(temp_ptr);
    switch(mode)
            {
            case MODE_32 :
            printf("\n\n Data is stored in the file the format is:");
                  printf("\n 16 bit audio channel A\n 16 bit audio channel B");
                  printf("\n 16 bit information channel A\n 16 bit information channel
B");
            break;
            case MODE_16 :
            printf("\n\n Data is stored in the file the format is:");
                  printf("\n 16 bit audio channel A\n 16 bit audio channel B");
            break;
            }
    } /* end of save if */
(void)fclose(fp);
} /* end of the board was successfully selected */
else
{ /* The Select_Board routine has returned an error    */
printf("Can't select the board\n");
printf("Have you installed the device driver?\n");
}
end_of_record:error = Hold();
error = Assert_Reset();
error = Deselect_Board(); /* use correct fn name ! */
} /* end of main */
```

B. MAIN PROGRAM "run_play.c"

```
/* run_play.c */
/***************/
/* program which sets up C30 and transfers */
/* double bufferred data to the host via DPRAM */
/* Author: Arie Gal-Gartenlaub  */
/* last update  : March 3rd 1994 */
/* Version: 1.00 */

#include "sdsp30lib.h" /* the interface library header file */

#define READY           0xeeeeeeee
#define NOT_READY       0x11111111
#define READY_A         0xeeee0000
#define READY_B         0x11110000
#define HIGH_MASK       0xffff0000
#define LOW_MASK        0x0000ffff

/* now define DSP DPRAM addresses */
/* These are the addresses to interface the dsp board with data or flags */
#define BUFFER_A_STATUS_ADDRESS     0x0400000
#define BUFFER_B_STATUS_ADDRESS     0x0400001
#define BUFFER_A_BASE_ADDRESS       0x0400010
#define BUFFER_B_BASE_ADDRESS       0x0400408
#define START_RECORD_FLAG_ADDRESS   0x0400007
#define FRAME_MODE_ADDRESS          0x0400008
#define BIT_MODE_ADDRESS            0x0400009
#define DETECTED_RATE_ADDRESS       0x040000a
#define TRANSMIT_MODE_ADDRESS       0x040000b
#define FIND_HEADER_ADDRESS         0x040000c
#define MODE_OF_OPERATION           0x040000d
#define PLL_STATUS_ADDRESS          0x040000e
#define DP_RAM_DATA_BUFFER_LENGTH   0x3c0
#define times                       6000

#define SPDIF_16        0x00500000 /* AMELIA Control DATA for SPDIF input 16 bit */
#define AES_EBU_16      0x00700000 /* AMELIA Control DATA for AES/EBU input 16 bit */
#define SPDIF_32        0x00000000 /* AMELIA Control DATA for SPDIF input 32 bit */
#define AES_EBU_32      0x00200000 /* AMELIA Control DATA for AES/EBU input 32 bit */

#define     XMT_PLL     0x00000000 /* Transmit from PLL clock */
#define     XMT_32      0x00010000 /* Transmit in 32KHz */
#define     XMT_44      0x00020000 /* Transmit in 44.1KHz */
#define XMT_48          0x00030000 /* Transmit in 48KHz */

#define MODE_16     1
#define MODE_32     2

FILE  *fp;
typedef struct node{unsigned long music_array[2*DP_RAM_DATA_BUFFER_LENGTH];
                    unsigned long data_ready;
                    struct  node *next;
                    }block;

block *new_block()
{
  return((block *)malloc(sizeof(block)));
}

void main()
{
      char  error = 0 ;
      char  file_name[100];
      char  ch;
      block *head_ptr,*temp_ptr,*last_ptr;
      int buffA_count[times];
      int buffB_count[times];
      unsigned long header,start_save,zero_detect,memory,temp;
      unsigned long save,mode,flag,stop_flag,frame_flag,bit_mode;
      int   seconds,counter,choice,read_size,xmt_mode;
      int   i,k,l,blocks,step,block_count,get_out,min_wait;
```

```
/* firstly check to see if the DSP board is responding */

      error = Select_Board("/dev/LSIsdsp300");

      if (!error)
{
/* The board is selected  */
/* Hold the processor                                   */
  error = Hold();
/* Load a program                                       */
  error = Load_Object_File("sp2ae_play.out");
  if(error==SLIB_ERR_LOADF)
    { /* the file was not found so get out */
    (void)printf("\n Can't find sp2ae_save.out which is the TMS320 file");
    exit(8);
    }
/* Start dialog with user */
(void)printf("\n This is the Main Menu. Enter the Option No. accordingly.");
(void)printf("\n ------------------------------------------------------------");
(void)printf("\n 1 - 16 BIT MODE file send to DAT.\n");
(void)printf("\n --- This option sets 48kHz sample rate ----\n");
(void)printf("\n 2 - 32 BIT MODE file send to DAT.\n");
(void)printf("\n --- Chose this option if the data sample rate is other then 48 kHz \n");
(void)printf("\n 3 - DUBBING from one DAT to another (for hardware check)");
(void)printf("\n ------ this sets 32 bit mode automatically ----\n ==> ");
(void)scanf("%d",&choice);
if(choice==3)
  { /* this means a debug mode 32_bit mode ask now for input origin */
  /* Send mode of operation */
  (void)printf("\n Select input source :");
  (void)printf("\n 1 - SPDIF 32 bits");
  (void)printf("\n 2 - AES/EBU 32 bits\n ==>  ");
  (void)scanf("%d",&choice);
  switch(choice)
    {
    case 1: printf("\n You chose SPDIF_32");
        Put_32_Bit(MODE_OF_OPERATION,SPDIF_32);
        break;
   case 2: printf("\n You chose AES_EBU_32");
        Put_32_Bit(MODE_OF_OPERATION,AES_EBU_32);
        break;
    } /* end of choice switch */
  Put_32_Bit(TRANSMIT_MODE_ADDRESS,XMT_PLL);
  Put_32_Bit(BIT_MODE_ADDRESS,MODE_32);

/* Pulse the reset line to start the program running */
  error = Assert_Reset();
  error = Release_Reset();
/* UnHold the processor and the program will run     */
  error = Unhold();
  (void)printf("\n To stop the program enter 1.\n ==>  ");
  stop_flag = 0;
  while(stop_flag != 1)
    { /* poll input again till stop */
    (void)scanf("%d",&stop_flag);
    } /* end of polling */
  error = Hold();
  error = Assert_Reset();
  error = Deselect_Board();
  } /* end of DATA transfer and end of program */
else
  { /****************************** The Main Modes of Operation *****************/
  (void)rewind(stdin);
/*(void)fopen(stdin);*/
  (void)printf("\n Enter the file name without extension. ");
  (void)printf("\n The assumed extension is '.dau')\n ==> ");
  (void)fgets(file_name,sizeof(file_name),stdin);
  file_name[strlen(file_name)-1]='.';
  (void)strcat(file_name,"dau");
  fp = fopen(file_name,"r");
  if(fp==NULL)
      {
        (void)printf("\n Can't find/open %s please check path,name,etc... \n",file_name);
        exit(8);
      }
  switch(choice)
```

```
    { /* switch to determine whether mode is 32_bit or 16_bit */
    case 1: /* 16_bit mode */
      (void)printf("\n You chose 16_bit mode the rate is automatically set to 48 kHz");
      xmt_mode = XMT_48;
      Put_32_Bit(TRANSMIT_MODE_ADDRESS,XMT_48);
      Put_32_Bit(BIT_MODE_ADDRESS,MODE_16);
      mode = MODE_16;
      break;
    case 2:
      (void)printf("\n You have chosen 32_bit mode this option uses AES/EBU framing ");
        Put_32_Bit(MODE_OF_OPERATION,MODE_32);
      Put_32_Bit(BIT_MODE_ADDRESS,MODE_32);
        mode = MODE_32;
      (void)printf("\n Please Enter the Sampling Rate.");
      (void)printf("\n 1 - 32khz \n 2 - 44.1khz \n 3 - 48khz \n ==> ");
      (void)scanf("%d",&xmt_mode);
      switch(xmt_mode)
      {
      case 1:     Put_32_Bit(TRANSMIT_MODE_ADDRESS,XMT_32);
            (void)printf("\n XMT_32 was sent ");
            break;
      case 2:     Put_32_Bit(TRANSMIT_MODE_ADDRESS,XMT_44);
            (void)printf("\n XMT_44 was sent ");
            break;
      case 3:     Put_32_Bit(TRANSMIT_MODE_ADDRESS,XMT_48);
            (void)printf("\n XMT_48 was sent ");
            break;
      }
      (void)printf("\n Does your data file contain AES/EBU framing ? y/n");
      (void)printf("\n In the absence of framing chose NO. Default framing is created");
      (void)printf("\n If sure that your file contains framing chose YES \n ==> ");
      (void)rewind(stdin);
      ch = getchar();
/*    (void)scanf("%d",&choice); */
      if(ch=='y') Put_32_Bit(FRAME_MODE_ADDRESS,READY);
      if(ch=='n') Put_32_Bit(FRAME_MODE_ADDRESS,NOT_READY);
        break;
    } /************end of switch between 16 or 32 bit mode *****************/

  /* Send mode of operation */
  printf("\n Do you want to add a Header to the audio segment? y/n");
  printf("\n It is recommended for later retrieving the file");
  printf("\n ==> ");
  (void)rewind(stdin);
  ch = getchar();
/*  scanf("%d",&header); */
  switch(ch)
    {
    case 'y': printf("\n You chose to incorporate a header");
            Put_32_Bit(FIND_HEADER_ADDRESS,READY);
            break;
    case 'n': printf("\n You chose not to use a header");
            Put_32_Bit(FIND_HEADER_ADDRESS,NOT_READY);
            break;
    default : printf("\n The default not to incorporate a header was chosen.");
            Put_32_Bit(FIND_HEADER_ADDRESS,NOT_READY);
            break;
    } /* end of header switch */
  printf("\n Initializing Data Base to get the audio from '%s'",file_name);
  head_ptr=new_block();
  last_ptr=new_block();
  last_ptr->next=NULL;
  head_ptr->next=last_ptr; /* linking the memory modules */
  temp_ptr=head_ptr;       /* a temporary pointer used to scan the array */
  temp_ptr->data_ready = NOT_READY;
  memory=READY;
  block_count=0;
  l=0;
  while((read_size=fread(&temp,1,sizeof(temp),fp))==sizeof(temp))
    {
     temp_ptr->music_array[l++] = temp;
     if(l == 2*DP_RAM_DATA_BUFFER_LENGTH)
      {
      temp_ptr->data_ready = READY;
      last_ptr->next=new_block();
      if (last_ptr->next==NULL)
```

```
            {
            printf("\n Out of memory after %d seconds.\n",block_count/(50*mode));
            memory=NOT_READY;
            printf("\n Do you wish to continue recording ? 1 - yes 0 - no");
            scanf("%d",&get_out);
            if (get_out!=1) goto end_of_record;
            }
        last_ptr->data_ready=NOT_READY;
        temp_ptr=last_ptr;
        last_ptr=last_ptr->next;
        last_ptr->next=NULL;
        l=0;
        block_count+=1;
        } /* end of if and create a new block */
      } /* end of while and the data was read from the file to the array */
    (void)fclose(fp);
    (void)printf("\n Data Base is ready %d blocks were initialized.\n",block_count);
    (void)printf("\n Method of Operation is :");
    (void)printf("\n 1. Make sure the appropriate input is chosen to the DAT");
    (void)printf("\n 2. Start tape about 1 second or more before the desired location.");
    (void)printf("\n 3. Press REC button the DAT will go to REC and PAUSE mode.");
    (void)printf("\n 4. Press the PLAY button.");
    (void)printf("\n 5. When the tape starts rolling hit ENTER to start playback.\n ==>  ");

    temp_ptr = head_ptr;
    l=0;
    flag = NOT_READY;
    error = Put_32_Bit( BUFFER_A_STATUS_ADDRESS, &flag);
    error = Put_32_Bit( BUFFER_B_STATUS_ADDRESS, &flag);
    error =
Put_Block_32(BUFFER_A_BASE_ADDRESS,&temp_ptr->music_array[l],DP_RAM_DATA_BUFFER_LENGTH);
    l+=DP_RAM_DATA_BUFFER_LENGTH;
    error =
Put_Block_32(BUFFER_B_BASE_ADDRESS,&temp_ptr->music_array[l],DP_RAM_DATA_BUFFER_LENGTH);
    temp_ptr = temp_ptr->next;
    /* Pulse the reset line to start the program running */
    error = Assert_Reset();
    error = Release_Reset();
    /* UnHold the processor and the program will run     */
    error = Unhold();
    /* now loop round pulling buffers from DPRAM - 10 times only here */
    (void)rewind(stdin);
    while((ch=getchar()) != '\n') putchar(ch);
/* Operation starts */
    printf("\n Starting playback.");
    error = Put_32_Bit(START_RECORD_FLAG_ADDRESS,READY);
    block_count = 0;
    min_wait = 2000;
    while ( temp_ptr != NULL )
      {   /* the main while waiting loop */
      l=0;    /* l is the accumulative pointer for the music */
      flag = NOT_READY;
      counter = 0;
      /* communicate with the c_tst30 DSP program ....*/
      while (flag != READY) /* wait for buffer A to be ready to read */
           {
           error = Get_32_Bit( BUFFER_A_STATUS_ADDRESS, &flag);
           counter++;
           }
      /* we have a ready flag for buffer A, so read the data*/
      if (min_wait > counter) min_wait = counter;
      buffA_count[block_count]=counter;
      error =
Put_Block_32(BUFFER_A_BASE_ADDRESS,&temp_ptr->music_array[l],DP_RAM_DATA_BUFFER_LENGTH);
      l+=DP_RAM_DATA_BUFFER_LENGTH;
      /* now wait for buffer B */
      flag = NOT_READY;
      counter = 0;
      while (flag != READY) /* wait for buffer B to be ready to read */
          {
          error = Get_32_Bit( BUFFER_B_STATUS_ADDRESS, &flag);
          counter++;
          }
      /* we have a ready flag for buffer B, so read the data. */
      if (temp > counter) temp = counter;
      buffB_count[block_count]=counter;
```

```
    error =
Put_Block_32(BUFFER_B_BASE_ADDRESS,&temp_ptr->music_array[1],DP_RAM_DATA_BUFFER_LENGTH);
    temp_ptr=temp_ptr->next;
    block_count=block_count + 1;
    }   /********************* end of main while loop  */
  /* Deselect the board to tidy up */
  error = Hold();
  error = Assert_Reset();
  error = Deselect_Board(); /* use correct fn name ! */
  } /* end of the board was successfully selected */
}
else
{ /* The Select_Board routine has returned an error    */
printf("Can't select the board\n");
printf("Have you installed the device driver?\n");
}
/* for(i=0;i<block_count;i+=20)
    printf("LOOP %d: buffer A with %d reads of 'flag', buffer B with %d reads of
'flag'\n",i, buffA_count[i], buffB_count[i]); */
printf("\n The minimum 'wait for ready' read cycles was %d ",min_wait);
end_of_record:error = Hold();
error = Assert_Reset();
error = Deselect_Board(); /* use correct fn name ! */
} /* end of main */
```

C. SUBPROGRAM "sp2ae_save.c"

```
/* START OF PROGRAM sp2ae_save.c */
/*********************************/
/* sp2ae_save.c  Portion of TMS320c30 C code */
/* The program will be run by "run_save.c" */
/* This is a program that uses polling and not interrupt */
/* Author: Arie Gal Gartenlaub */
/* Date  : 8th March 1994  */
/* Version 1.00 */

/* Sbus global addresses definitions */
/*************************************/

#define DP_RAM_BUFFER_A_BASE_ADDRESS    0x0400010
#define DP_RAM_BUFFER_B_BASE_ADDRESS    0x0400408
#define BUFFER_A_STATUS_ADDRESS         0x0400000
#define BUFFER_B_STATUS_ADDRESS         0x0400001
#define START_RECORD_FLAG_ADDRESS       0x0400007
#define FRAME_MODE_ADDRESS              0x0400008
#define BIT_MODE_ADDRESS                0x0400009
#define DETECTED_RATE_ADDRESS           0x040000a
#define TRANSMIT_MODE_ADDRESS           0x040000b
#define FIND_HEADER_ADDRESS             0x040000c
#define MODE_OF_OPERATION               0x040000d
#define PLL_STATUS_ADDRESS              0x040000e
#define DP_RAM_BUFFER_LENGTH            0x3c0
#define AES_BLOCK_SIZE                  192

/* global control variables */

#define BUFFA     0
#define BUFFB     -1

#define READY           0xeeeeeeee
#define NOT_READY       0x11111111
#define READY_A         0xeeee0000
#define READY_B         0x11110000


#define MAX             0xffffffff
#define ZERO            0x00000000

/* Sbus hardware definitions   */

#define BUSADDR         0x00808064
#define BUSDATA         0x00000900 /* originally 9 */
#define BUSADDR1        0x00808060
#define BUSDATA1        0x00000068

/* AMELIA hardware addresses for signal flow control */
/*****************************************************/

#define CH0_ADDRESS     0x00804002 /* AMELIA Channel A address */
#define CH1_ADDRESS     0x00804006 /* AMELIA Channel B address */
#define UCTR_ADDRESS    0x00804008 /* AMELIA User control register Write address */
#define ACTR_ADDRESS    0x0080400a /* AMELIA Control register Write address */
#define ASTS_ADDRESS    0x0080400a /* AMELIA Status register Read address */
#define INTM_ADDRESS    0x0080400b /* AMELIA Interrupt Mask register Write address */
#define INTS_ADDRESS    0x0080400b /* AMELIA Interrupt status register Read address */
#define CNFG_ADDRESS    0x0080400f /* AMELIA Configuration register Write address */

/* AMELIA data words for the proper signal flow control */
/*******************************************************/
#define UCTR_DATA 0xa0000000 /* User Control register DATA to write */

#define MODE_16     1
#define MODE_32     2

#define FIRST_SAMPLE    0x10000000/* its the first sample to synchronize */
#define FIRST_16_BIT    0x00000000/* its the case where the first word is send*/
#define SECOND_16_BIT   0x00010000/* its the case where the second word is send*/

#define SPDIF_16        0x00500000 /* AMELIA Control DATA for SPDIF input 16 bit */
```

```
#define AES_EBU_16      0x00700000 /* AMELIA Control DATA for AES/EBU input 16 bit */
#define SPDIF_32        0x00000000 /* AMELIA Control DATA for SPDIF input 32 bit */
#define AES_EBU_32      0x00200000 /* AMELIA Control DATA for AES/EBU input 32 bit */

/* 5 - SPDIF  7 - AES/EBU 0 - for 32 bit accuracy*/

#define     XMT_PLL     0x00000000 /* Transmit from PLL clock */
#define     XMT_32      0x00010000 /* Transmit in 32KHz */
#define     XMT_44      0x00020000 /* Transmit in 44.1KHz */
#define XMT_48          0x00030000 /* Transmit in 48KHz */

#define CNFG_DATA       0x8ff80000 /* The AMELIA key for initializing the process */
/* Amelia  Interrupt register */
#define INTM_PLL        0x00040000 /* Mask to determine loss lock of PLL */
#define INTM_XMT        0x00020000 /* Mask for determining if the output buffer is empty
*/
#define INTM_DATA       0x00010000 /* Mask register for input buffers full */
/* Amelia status register */
#define PLL_MASK        0x00080000 /* Mask for testing PLL lock on the signal */
#define VALID_SIG_MASK  0x00040000 /* Mask to test signal validity at status reg.I*/
#define BLOCK_MASK_SEND 0x00020000 /* What block to be send in 32 bit at sts. reg.I*/
#define BLOCK_MASK_READ 0x00010000 /* What bloc is read in 32 bit mode at sts. reg.I*/

#define HIGH_MASK       0xffff0000 /* mask living only high bits */
#define LOW_MASK        0x0000ffff /* mask living only low bits */
#define TEST_MASK       0x00210021 /* mask the not necessary bits from the frame ward*/
#define BLOCK_START_MASK 0x00010001 /* the mask to test block start */


/* global variables and pointers */
/*********************************/

/* pointers to DP_RAM data */

long *buffA_ptr = (long *)DP_RAM_BUFFER_A_BASE_ADDRESS;
long *buffB_ptr = (long *)DP_RAM_BUFFER_B_BASE_ADDRESS;
long *buffA_status_ptr = (long *)BUFFER_A_STATUS_ADDRESS;
long *buffB_status_ptr = (long *)BUFFER_B_STATUS_ADDRESS;


long *bus_ptr = (long *)BUSADDR;
long *bus1_ptr = (long *)BUSADDR1;
long *ch_A_ptr = (long *)CH0_ADDRESS ;
long *ch_B_ptr = (long *)CH1_ADDRESS ;
long *usr_cntl_ptr = (long *)UCTR_ADDRESS;
long *amelia_cntl_ptr = (long *)ACTR_ADDRESS;
long *amelia_sts_ptr = (long *)ASTS_ADDRESS;
long *amelia_intm_ptr = (long *)INTM_ADDRESS;
long *amelia_ints_ptr = (long *)INTS_ADDRESS;
long *amelia_config_ptr = (long *)CNFG_ADDRESS;
long *operation_mode_ptr =(long *)MODE_OF_OPERATION;
long *pll_sts_ptr = (long *)PLL_STATUS_ADDRESS;
long *header_ptr = (long *)FIND_HEADER_ADDRESS;
long *transmit_mode_ptr = (long *)TRANSMIT_MODE_ADDRESS;
long *detected_rate_ptr = (long *)DETECTED_RATE_ADDRESS;
long *frame_flag_ptr = (long *)FRAME_MODE_ADDRESS;
long *start_record_ptr = (long *)START_RECORD_FLAG_ADDRESS;

/* initialize default value for fill ptr */

long *current_fill_ptr = (long *)DP_RAM_BUFFER_A_BASE_ADDRESS;
long *current_status_ptr = (long *)BUFFER_B_STATUS_ADDRESS;

/* variables used by in line assembler instructions */


int samp_count,counter,delay,l;
int current_buffer = BUFFA;

unsigned long     pll,int_read,count_max,mode,bit_mode,header=READY,xmt_mode,flag;
unsigned long     frame_flag,start_record = NOT_READY,pll_sts = NOT_READY,zero_detect;
unsigned long     temp,temp1,block_start=NOT_READY;

main()                  /* start of the main program */

{
```

```
*bus_ptr = BUSDATA;
*busl_ptr = BUSDATA1;
counter = 0;
samp_count = 0;
count_max = 0;
pll = 0;
flag = NOT_READY;

/* set up status flags for host program */

current_status_ptr = buffB_status_ptr;
*current_status_ptr = NOT_READY;
current_status_ptr = buffA_status_ptr;
*current_status_ptr = NOT_READY;
*usr_cntl_ptr = UCTR_DATA;

/* *amelia_cntl_ptr = ACTR_DATA; */

/* read the desired modes of operation set up by the host user */

mode = *operation_mode_ptr;  /* this is the right line */

/*  mode = SPDIF_32; */ /* the mode is set manually for test */

header = *header_ptr;

if((header!=READY)&&(header!=NOT_READY)) header=READY;

switch(mode){

    case SPDIF_16:  mode = MODE_16 ;
                    *amelia_cntl_ptr = SPDIF_16;
                    break;

    case SPDIF_32:  mode = MODE_32 ;
                    *amelia_cntl_ptr = SPDIF_32;
                    break;

    case AES_EBU_16: mode = MODE_16 ;
                    *amelia_cntl_ptr = AES_EBU_16;
                    break;

    case AES_EBU_32: mode = MODE_32 ;
                    *amelia_cntl_ptr = AES_EBU_32;
                    break;

    default :   mode = MODE_32 ;
                *amelia_cntl_ptr = AES_EBU_32;
                    break;
                } /* end of switch */

delay = mode * 48000 ;

switch(mode)

{  /* main switch mode between 16 and 32 bit mode */

case MODE_16:

    *amelia_intm_ptr = INTM_DATA;
    *amelia_config_ptr = CNFG_DATA;
    *pll_sts_ptr = NOT_READY;
    pll_sts=NOT_READY;
    while((pll_sts == NOT_READY) && (delay >= 0))
          {
      while((int_read = *amelia_intm_ptr & INTM_DATA) != INTM_DATA);
            delay-=1;
            if ((pll = *amelia_sts_ptr & PLL_MASK) != 0 )
                  { pll_sts = READY;
                 *pll_sts_ptr = READY;
                  } /* end of if */
       }/* end of while */

    int_read = *amelia_intm_ptr;
    *amelia_intm_ptr = 0;
    *amelia_intm_ptr = INTM_DATA;
```

```
/* start header detect block if wanted */

    if(header==NOT_READY)
      {
      while(header == NOT_READY)
        {
        while((int_read = *amelia_intm_ptr & INTM_DATA) != INTM_DATA);
            /* loop polling interrupts */
        if ((*ch_A_ptr==READY_A)&&(*ch_B_ptr==READY_B)){header=READY;}
        }
      zero_detect = NOT_READY;
      while(zero_detect == NOT_READY)
            {
            while((int_read = *amelia_intm_ptr & INTM_DATA) != INTM_DATA);
            if ((*ch_A_ptr==0)&&(*ch_B_ptr==0)){zero_detect=READY;}
            }
        for(l=0;l<AES_BLOCK_SIZE-1;l++)
            {
            while((int_read = *amelia_intm_ptr & INTM_DATA) != INTM_DATA);
            /* loop polling interrupts */
            }
      } /* end of header detect block */

    /* LOOP FOREVER POLLING INTERRUPTS */
        while (1 != 0)
        {
            while((int_read = *amelia_intm_ptr & INTM_DATA) != INTM_DATA);
            *current_fill_ptr = *ch_A_ptr | ((*ch_B_ptr >> 16)&LOW_MASK);/*store in A and
B*/
            *ch_A_ptr = *current_fill_ptr;/* copy channel A to out */
            *ch_B_ptr = (*current_fill_ptr++)<<16;/* copy channel B to out */
            samp_count += 1;
                /* constantly check to see if current buffer is full */
            if(samp_count == DP_RAM_BUFFER_LENGTH)
                {
                switch(current_buffer)
                    { /* toggle between buffers */
                    case BUFFA:
                    current_fill_ptr = (long *)DP_RAM_BUFFER_B_BASE_ADDRESS;
                    samp_count = 0;
                    *current_status_ptr = READY;
                    current_status_ptr = buffB_status_ptr;
                    *current_status_ptr = NOT_READY;
                    current_buffer = BUFFB;
                    break;
                    case BUFFB:
                    current_fill_ptr = (long *)DP_RAM_BUFFER_A_BASE_ADDRESS;
                    samp_count = 0;
                    *current_status_ptr = READY;
                    current_status_ptr = buffA_status_ptr;
                    *current_status_ptr = NOT_READY;
                    current_buffer = BUFFA;
                    break;
                    } /* end of switch */
                } /* end samp_count if */
        } /* end while */
    break;
/************************** the 32 bit mode *************/

case MODE_32:

    *amelia_intm_ptr = INTM_DATA;
    *amelia_config_ptr = CNFG_DATA;
    *pll_sts_ptr = NOT_READY;
    flag = NOT_READY;
    zero_detect = NOT_READY;
    pll_sts=NOT_READY;
    while(pll_sts == NOT_READY && delay>=0)
      {
      if((int_read = *amelia_intm_ptr & INTM_DATA) == INTM_DATA)
            {
            delay-=1;
              pll = *amelia_sts_ptr & PLL_MASK;
              if ( pll != 0 )
                  { pll_sts = READY;
                 *pll_sts_ptr = READY;
```

```
                    } /* end of if */
                } /* end of if*/
            }   /* end of while */

        int_read = *amelia_intm_ptr;
        *amelia_intm_ptr = 0;
        *amelia_intm_ptr = INTM_DATA;

    /* synchronization on start of desired sequence and on header */
    /*************************************************************/
    /* first find header if it was desired */

        if(header==NOT_READY)
          {
          while(header == NOT_READY)
            {
            while((int_read = *amelia_intm_ptr & INTM_DATA) != INTM_DATA);
                /* loop polling interrupts */
            if ((*ch_A_ptr==READY_A)&&(*ch_B_ptr==READY_B)){header=READY;}
            }
          zero_detect = NOT_READY;
          while(zero_detect == NOT_READY)
            {
            while((int_read = *amelia_intm_ptr & INTM_DATA) != INTM_DATA);
            if ((*ch_A_ptr==0)&&(*ch_B_ptr==0)){zero_detect=READY;}
            }
          for(l=0;l<2*AES_BLOCK_SIZE-1;l++)
            {
            while((int_read = *amelia_intm_ptr & INTM_DATA) != INTM_DATA);
            /* loop polling interrupts */
            }
          } /* end of header detect block */
        else
          { /* else  start and synchronize on the beginning of a block */

          while(block_start == NOT_READY)
            {
              while((int_read = *amelia_intm_ptr & INTM_DATA) != INTM_DATA);
            if ((int_read = *amelia_sts_ptr & BLOCK_MASK_READ )== SECOND_16_BIT)
              {
             temp=(*ch_A_ptr | ((*ch_B_ptr >> 16)&LOW_MASK));
             if((temp&BLOCK_START_MASK)==BLOCK_START_MASK)
                {
                  block_start = READY;
                  *current_fill_ptr++ = temp1;
                  *current_fill_ptr++ = temp;
                  samp_count+=2;
                }
              *ch_A_ptr = temp;   /* copy channel A to out */
              *ch_B_ptr = temp<<16;/* copy channel B to out */
            }
           else
             {
              temp1 = *ch_A_ptr | ((*ch_B_ptr >> 16)&LOW_MASK);
                /*store in A and B*/
                *ch_A_ptr = temp1;/* copy channel A to out */
                *ch_B_ptr = (temp1)<<16;/* copy channel B to out */
            }
          }  /* end of while start_block == NOT_READY */
        } /* end of else header != READY */

    /* now ready to start fill up the memory */
    /* LOOP FOREVER POLLING INTERRUPTS */

        while (1 != 0)
            {
              while((int_read = *amelia_intm_ptr & INTM_DATA) != INTM_DATA);
            if ((int_read = *amelia_sts_ptr & BLOCK_MASK_READ )== SECOND_16_BIT)
              {
              *current_fill_ptr = (*ch_A_ptr | ((*ch_B_ptr >> 16)&LOW_MASK))&TEST_MASK;
                *ch_A_ptr = *current_fill_ptr;/* copy channel A to out */
                *ch_B_ptr = (*current_fill_ptr++)<<16;/* copy channel B to out */
                samp_count += 1;
            }
          else
            {
```

```
        *current_fill_ptr = *ch_A_ptr | ((*ch_B_ptr >> 16)&LOW_MASK);
          /*store in A and B*/
          *ch_A_ptr = *current_fill_ptr;/* copy channel A to out */
          *ch_B_ptr = (*current_fill_ptr++)<<16;/* copy channel B to out */
          samp_count += 1;
       }

        /* constantly check to see if current buffer is full */

         if(samp_count == DP_RAM_BUFFER_LENGTH)
                {
                switch(current_buffer)
                    { /* toggle between buffers */
                    case BUFFA:
                    current_fill_ptr = (long *)DP_RAM_BUFFER_B_BASE_ADDRESS;
                    samp_count = 0;
                    *current_status_ptr = READY;
                    current_status_ptr = buffB_status_ptr;
                    *current_status_ptr = NOT_READY;
                    current_buffer = BUFFB;
                    break;
                    case BUFFB:
                    current_fill_ptr = (long *)DP_RAM_BUFFER_A_BASE_ADDRESS;
                    samp_count = 0;
                    *current_status_ptr = READY;
                    current_status_ptr = buffA_status_ptr;
                    *current_status_ptr = NOT_READY;
                    current_buffer = BUFFA;
                    break;
                    } /* end of switch */
                } /* end samp_count if */
        } /* end while */
       break;
    } /* end of mode switch whether 16 or 32 bits */
} /* ************ end of main ******* */
```

D. SUBPROGRAM "sp2ae_play.c"

```
/* START OF PROGRAM     sp2ae_play.c */
/*************************************/
/* sp2ae_play.c  Portion of TMS320c30 C code*/
/* The program is run by main program "run_save.c". */
/* This is a program that uses polling and not interrupt */
/* Author: Arie Gal Gartenlaub*/
/* Date  : 8th March 1994 */
/* Version 1.00 */

/* Sbus global addresses definitions for interfacing with the host */
/*******************************************************************/
#define DP_RAM_BUFFER_A_BASE_ADDRESS    0x0400010
#define DP_RAM_BUFFER_B_BASE_ADDRESS    0x0400408
#define BUFFER_A_STATUS_ADDRESS         0x0400000
#define BUFFER_B_STATUS_ADDRESS         0x0400001
#define START_RECORD_FLAG_ADDRESS       0x0400007
#define FRAME_MODE_ADDRESS              0x0400008
#define BIT_MODE_ADDRESS                0x0400009
#define DETECTED_RATE_ADDRESS           0x040000a
#define TRANSMIT_MODE_ADDRESS           0x040000b
#define FIND_HEADER_ADDRESS             0x040000c
#define MODE_OF_OPERATION               0x040000d
#define PLL_STATUS_ADDRESS              0x040000e
#define DP_RAM_BUFFER_LENGTH            0x3c0
#define AES_BLOCK_SIZE                  192

/* global control variables */

#define BUFFA     0
#define BUFFB     -1

#define READY           0xeeeeeeee
#define NOT_READY       0x11111111
#define READY_A         0xeeee0000
#define READY_B         0x11110000

#define MAX             0xffffffff
#define ZERO            0x00000000


/* Sbus hardware definitions   */

#define BUSADDR         0x00808064
#define BUSDATA         0x00000900 /* originally 9 */
#define BUSADDR1        0x00808060
#define BUSDATA1        0x00000068

/* AMELIA hardware addresses for signal flow control */
/*****************************************************/

#define CH0_ADDRESS     0x00804002 /* AMELIA Channel A address */
#define CH1_ADDRESS     0x00804006 /* AMELIA Channel B address */
#define UCTR_ADDRESS    0x00804008 /* AMELIA User control register Write address */
#define ACTR_ADDRESS    0x0080400a /* AMELIA Control register Write address */
#define ASTS_ADDRESS    0x0080400a /* AMELIA Status register Read address */
#define INTM_ADDRESS    0x0080400b /* AMELIA Interrupt Mask register Write address */
#define INTS_ADDRESS    0x0080400b /* AMELIA Interrupt status register Read address */
#define CNFG_ADDRESS    0x0080400f /* AMELIA Configuration register Write address */


/* AMELIA data words for the proper signal flow control */
/*******************************************************/
#define UCTR_DATA       0xa0000000 /* User Control register DATA to write */

#define MODE_16     1
#define MODE_32     2

#define FIRST_SAMPLE    0x10000000/* its the first sample to synchronize */
#define FIRST_16_BIT    0x00000000/* its the case where the first word is send*/
#define SECOND_16_BIT   0x00010000/* its the case where the second word is send*/

#define SPDIF_16        0x00500000 /* AMELIA Control DATA for SPDIF input 16 bit */
```

```
#define AES_EBU_16      0x00700000 /* AMELIA Control DATA for AES/EBU input 16 bit */
#define SPDIF_32        0x00000000 /* AMELIA Control DATA for SPDIF input 32 bit */
#define AES_EBU_32      0x00200000 /* AMELIA Control DATA for AES/EBU input 32 bit */

/* 5 - SPDIF  7 - AES/EBU 0 - for 32 bit accuracy*/

#define     XMT_PLL     0x00000000 /* Transmit from PLL clock */
#define     XMT_32      0x00010000 /* Transmit in 32KHz */
#define     XMT_44      0x00020000 /* Transmit in 44.1KHz */
#define XMT_48          0x00030000 /* Transmit in 48KHz */

#define CNFG_DATA       0x8ff80000 /* The AMELIA key for initilizing the process */
/* Amelia  Interrupt register */
#define INTM_PLL        0x00040000 /* Mask to determine loss lock of PLL */
#define INTM_XMT        0x00020000 /* Mask for determining if the output buffer is empty
*/
#define INTM_DATA       0x00010000 /* Mask register for input buffers full */
/* Amelia status register */
#define PLL_MASK        0x00080000 /* Mask for testing PLL lock on the signal */
#define VALID_SIG_MASK  0x00040000 /* Mask to test signal validity at status reg.I*/
#define BLOCK_MASK_SEND 0x00020000 /* What block to be send in 32 bit at sts. reg.I*/
#define BLOCK_MASK_READ 0x00010000 /* What bloc is read in 32 bit mode at sts. reg.I*/

#define HIGH_MASK       0xffff0000 /* mask living only high bits */
#define LOW_MASK        0x0000ffff /* mask living only low bits */
#define TEST_MASK       0x00210000 /* mask the not necessary bits from the block 1 */

/* AES_EBU Definitions */
#define Z_FLAG          0x00010000 /* the flag of block start */
#define C_FLAG          0x00200020 /* the flag for C bit */


/* global variables and pointers */
/*********************************/

/* pointers to DP_RAM data */
long *buffA_ptr = (long *)DP_RAM_BUFFER_A_BASE_ADDRESS;
long *buffB_ptr = (long *)DP_RAM_BUFFER_B_BASE_ADDRESS;
long *buffA_status_ptr = (long *)BUFFER_A_STATUS_ADDRESS;
long *buffB_status_ptr = (long *)BUFFER_B_STATUS_ADDRESS;


long *bus_ptr = (long *)BUSADDR;
long *bus1_ptr = (long *)BUSADDR1;
long *ch_A_ptr = (long *)CH0_ADDRESS ;
long *ch_B_ptr = (long *)CH1_ADDRESS ;
long *usr_cntl_ptr = (long *)UCTR_ADDRESS;
long *amelia_cntl_ptr = (long *)ACTR_ADDRESS;
long *amelia_sts_ptr = (long *)ASTS_ADDRESS;
long *amelia_intm_ptr = (long *)INTM_ADDRESS;
long *amelia_ints_ptr = (long *)INTS_ADDRESS;
long *amelia_config_ptr = (long *)CNFG_ADDRESS;
long *operation_mode_ptr =(long *)MODE_OF_OPERATION;
long *pll_sts_ptr = (long *)PLL_STATUS_ADDRESS;
long *bit_mode_ptr = (long *)BIT_MODE_ADDRESS;
long *header_ptr = (long *)FIND_HEADER_ADDRESS;
long *transmit_mode_ptr = (long *)TRANSMIT_MODE_ADDRESS;
long *detected_rate_ptr = (long *)DETECTED_RATE_ADDRESS;
long *frame_flag_ptr = (long *)FRAME_MODE_ADDRESS;
long *start_record_ptr = (long *)START_RECORD_FLAG_ADDRESS;


/* initialize default value for fill ptr */

long *current_read_ptr = (long *)DP_RAM_BUFFER_A_BASE_ADDRESS;
long *current_status_ptr = (long *)BUFFER_B_STATUS_ADDRESS;

/* variables used by in line assembler instructions */


int samp_count,counter,k,l;
int current_buffer = BUFFA;

unsigned long     sync_array[AES_BLOCK_SIZE];
unsigned long     pll,int_read,count_max,mode,bit_mode,header,xmt_mode,flag,delay;
unsigned long     frame_flag,start_record = NOT_READY,pll_sts = NOT_READY;
```

```
main()                  /* start of the main program */

{

*bus_ptr = BUSDATA;
*busl_ptr = BUSDATA1;
counter = 0;
samp_count = 0;
count_max = 0;
pll = 0;
flag = NOT_READY;

/* set up status flags for host program */

current_status_ptr = buffB_status_ptr;
*current_status_ptr = NOT_READY;
current_status_ptr = buffA_status_ptr;
*current_status_ptr = NOT_READY;

*usr_cntl_ptr = UCTR_DATA;


/* read the desired modes of operation set up by the host  user */

bit_mode = *bit_mode_ptr; /* for debug */  /* 16 or 32 bit mode */

/*bit_mode = MODE_32; for debug */

mode = *operation_mode_ptr;

/* read the desired rate of operation set up by the host user */

xmt_mode = *transmit_mode_ptr;

*pll_sts_ptr = xmt_mode;       /* for debug */

header = *header_ptr;

/* the default is not using a header */

if((header!=READY) && (header!=NOT_READY)) header=NOT_READY;

switch(bit_mode)
 { /* This is the main switch */
    case MODE_16 :

      /* Only 48khz is allowed in this mode of operation */

      *amelia_cntl_ptr = SPDIF_16 | XMT_48 ;
      *amelia_intm_ptr = INTM_XMT;
      *amelia_config_ptr = CNFG_DATA;

/* The delay block for synchronization */
/*****************************************/
      delay = 125*AES_BLOCK_SIZE ;
      int_read = *amelia_intm_ptr; /* Dummy read to clear int */
      *ch_A_ptr = 0;
      *ch_B_ptr = 0;

      *start_record_ptr = NOT_READY;
      start_record = NOT_READY;

      while (start_record != READY )
        {
        l=0;
        start_record = *start_record_ptr;
        while( l<delay)
          {
             while((int_read = *amelia_intm_ptr & INTM_XMT) != INTM_XMT);
            *ch_A_ptr = 0;
            *ch_B_ptr = 0;
            l+=1;
          }
       }

      if(header==READY)
```

```
      { /* this is the loop to put header on the beginning of the DAT */
            l=0;
            while(l<AES_BLOCK_SIZE)
              {
                    while((int_read = *amelia_intm_ptr & INTM_XMT) != INTM_XMT);
              *ch_A_ptr = READY_A;
              *ch_B_ptr = READY_B;
              l+=1;
              } /* end of BLOCK_SIZE while */
            l=0;
            while(l<AES_BLOCK_SIZE)
              {
                    while((int_read = *amelia_intm_ptr & INTM_XMT) != INTM_XMT);
              *ch_A_ptr = 0;
              *ch_B_ptr = 0;
              l+=1;
              } /* end of BLOCK_SIZE while */
      } /* End of header if */
/* Now after the sync start to put staff on the DAT */
      while(1!=0)
      {
        while((int_read = *amelia_intm_ptr & INTM_XMT) != INTM_XMT)
        {/* do nothing and wait for the "interrupt" */};
        *ch_A_ptr = *current_read_ptr; /* copy channel A to out */
        *ch_B_ptr = (*current_read_ptr++)<<16; /* copy channel B to out */
        samp_count += 1;
          if(samp_count == DP_RAM_BUFFER_LENGTH)
          {
            switch(current_buffer)
            { /* toggle between buffers */
              case BUFFA:
              current_read_ptr = (long *)DP_RAM_BUFFER_B_BASE_ADDRESS;
              samp_count = 0;
              *current_status_ptr = READY;
              current_status_ptr = buffB_status_ptr;
              *current_status_ptr = NOT_READY;
              current_buffer = BUFFB;
              break;
              case BUFFB:
              current_read_ptr = (long *)DP_RAM_BUFFER_A_BASE_ADDRESS;
              samp_count = 0;
              *current_status_ptr = READY;
              current_status_ptr = buffA_status_ptr;
              *current_status_ptr = NOT_READY;
              current_buffer = BUFFA;
              break;
            } /* end of switch */
          } /* end samp_count if */
      } /* end of while 1!=0 */
    break;

/*************************** the 32 bit mode *************/
    case MODE_32 :

      if(xmt_mode==XMT_PLL)
        { /*  if the mode is PLL this is data transfer  */
          *amelia_cntl_ptr = mode;
          *amelia_intm_ptr = INTM_DATA;
          *amelia_config_ptr = CNFG_DATA;
          while(pll_sts == NOT_READY)
              {
                pll = *amelia_sts_ptr & PLL_MASK;
                if ( pll != 0 )
                  { pll_sts = READY;
                    *pll_sts_ptr = READY;
                  }       /* end of if */
              }           /* end of while */

          int_read = *amelia_intm_ptr;
          *amelia_intm_ptr = 0;
          *amelia_intm_ptr = INTM_DATA;

          while(1!=0) /* loop forever transferring data  */
            {
                    while((int_read = *amelia_intm_ptr & INTM_DATA) != INTM_DATA);
             if ((int_read = *amelia_sts_ptr & BLOCK_MASK_READ )== SECOND_16_BIT)
```

```
                {
                        *ch_A_ptr = *ch_A_ptr & TEST_MASK;
                        *ch_B_ptr = *ch_B_ptr & TEST_MASK;
                }
              else
                {
                  *ch_A_ptr = *ch_A_ptr;
                        *ch_B_ptr = *ch_B_ptr;
                }
            } /* end of forever loop */
          } /* end of XMT_PLL if */

/****************************** end of data transfer in 32 bit mode */
       else
          { /************** start the data transfer to DAT routine */


/* what frame to use local or user prerecorded frames ? ************/
           frame_flag = *frame_flag_ptr;
/* check for default the default is use a local framing ************/
           if((frame_flag != READY)&&(frame_flag!= NOT_READY))frame_flag = READY;

          for(l=0;l<AES_BLOCK_SIZE;l++)
             { /* this for loop is to zero the synchronization matrices */
               sync_array[l]=0;
             } /* end of for */
          sync_array[0]= C_FLAG | Z_FLAG;
          switch(xmt_mode)
             { /* this switch is for setting the amelia register and sync arrays*/
             case XMT_32:
                  delay = 167*AES_BLOCK_SIZE;
                  /* now make sync matrix for 32khz sample rate */
                  sync_array[2]= C_FLAG;
                  sync_array[6]= C_FLAG;
                  sync_array[7]= C_FLAG;
                  sync_array[184]= C_FLAG;
                  sync_array[186]= C_FLAG;
                  sync_array[187]= C_FLAG;
                  sync_array[188]= C_FLAG;
                  sync_array[191]= C_FLAG;
                  *amelia_cntl_ptr = AES_EBU_32 | XMT_32 ;
                  break;

             case XMT_44:
                  delay = 229*AES_BLOCK_SIZE;
                  /* now make sync matrix for 44khz sample rate */
                  sync_array[2]= C_FLAG;
                  sync_array[6]= C_FLAG;
                  sync_array[186]= C_FLAG;
                  sync_array[188]= C_FLAG;
                  sync_array[189]= C_FLAG;
                  *amelia_cntl_ptr = AES_EBU_32 | XMT_44 ;
                  break;

             case XMT_48:
                  delay = 250*AES_BLOCK_SIZE ;
                  /* now make sync matrix for 48khz sample rate */
                  sync_array[2]= C_FLAG;
                  sync_array[7]= C_FLAG;
                  sync_array[184]= C_FLAG;
                  sync_array[188]= C_FLAG;
                  sync_array[189]= C_FLAG;
                  sync_array[190]= C_FLAG;
                        *amelia_cntl_ptr = AES_EBU_32 | XMT_48 ;
                  break;
             } /* end of switch */


/* The delay block for synchronization */
/***************************************/
           flag = NOT_READY;
           *amelia_intm_ptr = INTM_XMT;
           *amelia_config_ptr = CNFG_DATA;

           int_read = *amelia_intm_ptr; /* Dummy read to clear int */
           *ch_A_ptr = 0;
```

```
          *ch_B_ptr = 0;

/****************************** then synchronies on the second frame */
          while (flag != READY)
              {
               while((int_read = *amelia_intm_ptr & INTM_XMT) != INTM_XMT);
              if((int_read = *amelia_sts_ptr & BLOCK_MASK_SEND) == BLOCK_MASK_SEND)
                   { flag = READY; }
              *ch_A_ptr = 0;
              *ch_B_ptr = 0;
              }

       start_record = NOT_READY;
       *start_record_ptr = NOT_READY;

          while (start_record != READY )
          { /* waiting for the operator to start the program */
           start_record = *start_record_ptr;
           k=0;
           for(l=0;l<2*delay;l++)
             {
             while((int_read = *amelia_intm_ptr & INTM_XMT) != INTM_XMT);
             if((int_read = *amelia_sts_ptr & BLOCK_MASK_SEND) != BLOCK_MASK_SEND)
               {
               *ch_A_ptr = 0;
               *ch_B_ptr = 0;
               }
             else
               {
               *ch_A_ptr = sync_array[k];
               *ch_B_ptr = (sync_array[k++])<<16;
               }
             if(k==AES_BLOCK_SIZE) k=0;
             }
          } /* end of waiting for the operator */
/*************************** create header if desired ***************/
          if(header==READY)
          { /* this is the loop to put header on the beginning of the DAT */
            l=0;
            while(l<AES_BLOCK_SIZE)
              {
                    while((int_read = *amelia_intm_ptr & INTM_XMT) != INTM_XMT);
              if((int_read = *amelia_sts_ptr & BLOCK_MASK_SEND) != BLOCK_MASK_SEND)
                  {
                  *ch_A_ptr = READY_A;
                  *ch_B_ptr = READY_B;
                  }
              else  {
                  *ch_A_ptr = sync_array[l];
                  *ch_B_ptr = (sync_array[l++])<<16;
                  }
              } /* end of BLOCK_SIZE while */
            l=0;
            while(l<AES_BLOCK_SIZE)
              {
                while((int_read = *amelia_intm_ptr & INTM_XMT) != INTM_XMT);
                if((int_read = *amelia_sts_ptr & BLOCK_MASK_SEND) != BLOCK_MASK_SEND)
                  {
                  *ch_A_ptr = 0;
                  *ch_B_ptr = 0;
                  }
                else
                  {
                  *ch_A_ptr = sync_array[l];
                  *ch_B_ptr = (sync_array[l++])<<16;
                  }
              } /* end of BLOCK_SIZE while */
          } /****************************************** End of header if */

/* now ready to start fill up the memory */
          switch(frame_flag)
            {
              case NOT_READY:  /* the frames are locally made */
                  /* LOOP FOREVER POLLING INTERRUPTS */
                  l=0;
                  while (1 != 0)
```

```
                    {
                      while((int_read = *amelia_intm_ptr & INTM_XMT) != INTM_XMT);
                       if((int_read = *amelia_sts_ptr & BLOCK_MASK_SEND) !=
BLOCK_MASK_SEND)
                          {
                          *ch_A_ptr = *current_read_ptr; /* copy channel A to out */
                          *ch_B_ptr = (*current_read_ptr++)<<16;/*copy cnannel B to out */
                          samp_count += 1;
                         }
                        else
                         {
                         *ch_A_ptr = sync_array[l];
                         *ch_B_ptr = (sync_array[l++])<<16;
                         if(l==AES_BLOCK_SIZE) l=0;
                          }
/*************  ********** constantly check to see if current buffer is full */
                       if(samp_count == DP_RAM_BUFFER_LENGTH)
                         {
                           switch(current_buffer)
                                 { /* toggle between buffers */
                                 case BUFFA:
                                 current_read_ptr = (long *)DP_RAM_BUFFER_B_BASE_ADDRESS;
                                 samp_count = 0;
                                 *current_status_ptr = READY;
                                 current_status_ptr = buffB_status_ptr;
                                 *current_status_ptr = NOT_READY;
                                 current_buffer = BUFFB;
                                 break;
                                 case BUFFB:
                                 current_read_ptr = (long *)DP_RAM_BUFFER_A_BASE_ADDRESS;
                                 samp_count = 0;
                                 *current_status_ptr = READY;
                                 current_status_ptr = buffA_status_ptr;
                                 *current_status_ptr = NOT_READY;
                                 current_buffer = BUFFA;
                                 break;
                                 } /* end of switch */
                         } /* end samp_count if */
                    } /******* end while 1!=0  *************/
                  break;



              case READY :

/*                   flag = NOT_READY;
                    *amelia_intm_ptr = INTM_XMT;
                    *amelia_intm_ptr = 0;
                    *amelia_intm_ptr = INTM_XMT;
                    *amelia_config_ptr = CNFG_DATA; */

/*                  int_read = *amelia_intm_ptr;*/ /* Dummy read to clear int */
/*                  *ch_A_ptr = 0;
                    *ch_B_ptr = 0; */

/****************************** then synchronies on the second frame */
/*                  while (flag != READY)
                    {
                     while((int_read = *amelia_intm_ptr & INTM_XMT) != INTM_XMT);
                    if((int_read = *amelia_sts_ptr & BLOCK_MASK_SEND) == BLOCK_MASK_SEND)
                    { flag = READY; }
                    *ch_A_ptr = 0;
                    *ch_B_ptr = 0;
                    } */
                    /* LOOP FOREVER POLLING INTERRUPTS */
                    while (1 != 0)
                      {

                        while((int_read = *amelia_intm_ptr & INTM_XMT) != INTM_XMT);
                        *ch_A_ptr = *current_read_ptr; /* copy channel A to out */
                        *ch_B_ptr = (*current_read_ptr++)<<16; /* copy channel B to out */
                        samp_count += 2; /* advancing samp_count ahead to gain time */
                        while((int_read = *amelia_intm_ptr & INTM_XMT) != INTM_XMT);
                        *ch_A_ptr = *current_read_ptr; /* copy channel A to out */
                        *ch_B_ptr = (*current_read_ptr++)<<16; /* copy channel B to out */
```

```
/************************** constantly check to see if current buffer is full */
                        if(samp_count == DP_RAM_BUFFER_LENGTH)
                          {
                            switch(current_buffer)
                                  { /* toggle between buffers */
                                  case BUFFA:
                                  current_read_ptr = (long *)DP_RAM_BUFFER_B_BASE_ADDRESS;
                                  samp_count = 0;
                                  *current_status_ptr = READY;
                                  current_status_ptr = buffB_status_ptr;
                                  *current_status_ptr = NOT_READY;
                                  current_buffer = BUFFB;
                                  break;
                                  case BUFFB:
                                  current_read_ptr = (long *)DP_RAM_BUFFER_A_BASE_ADDRESS;
                                  samp_count = 0;
                                  *current_status_ptr = READY;
                                  current_status_ptr = buffA_status_ptr;
                                  *current_status_ptr = NOT_READY;
                                  current_buffer = BUFFA;
                                  break;
                                  } /* end of switch */
                          } /* end samp_count if */
                      } /* end of while 1!=0 */
                    break;
              } /* end of frame_flag switch  */
            } /* end of else to data transfer */
        break;
    } /* end of mode switch whether 16 or 32 bits */
}/************* end of main ********/
```

LIST OF REFERENCES

[1] Ken C. Pohlmann, *Principles of Digital Audio*, second edition, SAMS Publishing, Carmel, Indiana, 1992.

[2] Robert Finger and T. Nakanishi, "Developments In R-DAT Data Recorders," IEEE Mass Data Storage Conference Proceedings pp. 35-37. Monterey, California, 1990.

[3] Albert S. Hoagland, *Digital Magnetic Recording*, John Wiley & Sons Inc., 1963.

[4] *The Academic American Encyclopedia* (Electronic Version), Grolier, Inc., Danbury, CT, 1992.

[5] Steve Oualline, *Practical C Programming*, O'Reilly & Associates, Inc., Sebastopol, California, 1993.

[6] *TMS320 Floating Point DSP Optimizing C Compiler*, User's Guide, Texas Instruments Inc., 1991.

[7] Ken C. Pholmann, "DAT in Depth," *Electronics* Australia, pp. 14-18, April 1988.

[8] W.S. Hodgkiss and J. C. Nickles, "Real Time Data Management in a UNIX Network Environment," IEEE Oceans and Seas Conference Proceedings pp. 294-297, 1990.

[9] John W. Einberger, "CD as a Mass Storage Device," IEEE Mass Data Storage Conference Proceedings, pp. 125-129, Monterey, California, 1988.

[10] *SDSP/C30D SBUS BOARD*, Technical Reference Manual, SPECTRUM Signal Processing Inc., Version 1.01, September 1992.

[11] *SDSP/C30D SBUS BOARD*, User Guide, SPECTRUM Signal Processing Inc., Version 1.01, September 1992.

[12] *DM/D24AES DIGITAL AUDIO MODULE*, User's Manual, SPECTRUM Signal Processing Inc., Version 1.00, March 1993.

[13] *TMS320 Floating Point DSP Assembly Language Tools*, User's Guide, Texas Instruments Inc., 1991.

BIBLIOGRAPHY

Brycer, Bernard B., *Digital Magnetic Tape Recording Principles and Computer Application*, Hayden Book Company Inc., New York, 1965.

Hipson, Peter, *Advanced C*, SAMS publishing, 1992.

Jacson Bruce, A User's Guide to Digital Audio Interconnects, Mix, October 1992.

Kelly-Bootle, Stan, *Understanding UNIX*, SYBEX Inc., 1992.

Kernighan, Brian W. and Ritchie, Dennis M., *The C Programming Language*. second edition, Prentice Hall, 1988.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria VA 22304-6145	2
2.	Dudley Knox Library, Code 52 Naval Postgraduate School Monterey, CA 93943-5101	2
3.	Chairman, Code EC Electrical and Computer Engineering Department Naval Postgraduate School Monterey, CA 93943-5121	1
4.	Charles W. Therrien, Code EC/Ti Electrical and Computer Engineering Department Naval Postgraduate School Monterey, CA 93943-5121	6
5.	Murali Tummala, Code EC/Tu Electrical and computer Engineering Department Naval Postgraduate School Monterey, CA 93943-5121	1
6.	Monique P. Fargues, Code EC/Fa Electrical and Computer Engineering Department Naval Postgraduate School Monterey, CA 93943-5121	1
7.	Israeli Defense Navy Attache Embassy of Israel 3514 International Drive N.W. Washington, D.C. 20008	4
8.	Arie Gal-Gartentlaub 24a Haagana St. Kiryat Motzkin, 26372 Israel.	2