Development of a Digital Audio Card for Recording and Playback on Personal Computers

By

ADEOLA, EPHRAIM OLUBUNMI
EEE/93/5401
B.Sc.(Hon.) Electrical Engineering, University of Lagos, 1992

A Thesis in

The Department of Electrical and Electronic Engineering

Submitted to

The School of Postgraduate Studies

towards the Degree of

Master of Engineering (M.Eng.) (Control Engineering with Computer Applications)

The Federal University of Technology,

Akure, Ondo State, Nigeria

March, 1997

CERTIFICATION

This is to certify that this thesis has been read and approved as meeting the requirements of the Department of Electrical and Electronic Engineering, School of Engineering and Engineering Technology, The Federal University of Technology, Akure for the award of Master of Engineering (Electrical and Electronic) Degree.

Dr. S. O. Falaki — 19/8/97
Major Supervisor — Date

J. O. Oni — 19/8/97
Co-Supervisor — Date

J. O. Oni — 19/8/97
Ag. Head of Department — Date

Dr. T S. Ibiyemi — Date
External Examiner

DEDICATION

This project is dedicated to the entire families of Chief & Mrs. L.O. Adeola.

DECLARATION

I hereby declare that this thesis is a record of my research work. It has never been presented nor accepted in any previous application for a higher degree.

All sources of information have been specifically acknowledged.

Signature: _____ Date: 17th June, '97

Adeola, E.O.

ACKNOWLEDGMENT

The duration of this research work spanned for a period longer than earlier projected. This is not due to too many failures, but because of the numerous challenges that cropped up along the line. I therefore give special thanks to the Almighty God, for seeing me through and whose "unmerited favour" I always enjoy.

My sincere appreciation goes to my Major Supervisor, Dr. S.O. Falaki, whose constructive criticisms were of immense help and for the provision of Borland C++ compiler used. I also acknowledge the important role played by my Co-Supervisor who incidentally is the present acting. Head of Department of Electrical & Electronic Engineering, FUTA, Engr. J.O. Oni, for the technical support in various capacities. Special thanks to the entire members of staff of the Department of Electrical & Electronic Engineering. FUTA for their various assistance and the use of the Departmental computer systems, during the course of this research work.

I am very grateful for the affection shown me by my parents, Chief L.O. Adeola and Mrs. C.M. Adeola and for their moral and financial supports. I also express my profound gratitude to my brothers (Vincent, Charles, James & Francis) and my sisters(Margaret & Francisca) - I love you all.

Worthy of mentioned are materials sourced from individuals, among which are ; Prof. L.O. Kehinde, Mr. E.O. Daramola, Engr. Pat Bodun, Kenny Ogunjobi Mr. S. Ajewole and the Organisation of Workkman Integral Networks.

ABSTRACT

The development of communication interfaces for personal computers and related equipment presents one of the most challenging problems in the electronic industry today. This research work will provide a high speed procedure of recording and playing back of audio signals at a relatively reduced cost. The developed system uses a personal computer to control and synchronize all the input/output (I/O) operations of the audio card. It will increase the versatility of the microcomputer systems and provide a suitable means of audio signal acquisition, storage and its retrieval.

The developed system consist of two sections; a software section and a hardware section. The hardware consist of a digital audio card circuitry interfaced to a personal computer via its parallel port. The circuit card was developed in a modular fashion comprising of three modules, namely; the recording module, the playback module and the power module. The recording module utilizes an 8-bit analog-to-digital converter IC, configured to accommodate a sampling frequency of approximately 44.1KHz, thus allowing the use of an economic reconstruction filter. The digitized audio signal could be stored in a floppy disc or the hard disc (depending on the specification in the software driver) for subsequent playback. The playback module uses an 8-bit digital-to-analog converter IC. The power module provides the four stabilized voltage levels; +15.0V, -15.0V, +5.0V and -5.0V.

A menu-driven software package written in C++ for the I/O control of the audio card was developed The software driver can broadly be divided into the following three modules; the menu module, the recording module and the playback module. A graphical display interface that displays the relative amplitude of the instantaneous audio signal (during recording or playback mode) is incorporated in the menu module. The recording module controls the sampling time of the ADC and the storage of the digitized signal, while the playback module controls the signal reconstruction frequency, the retrieval of the stored digital signal for subsequent transmission to the DAC circuit.

The software driver was used to carry out preliminary system performance tests in the computer hardware laboratory to simulate the recording and playback of baseband audio signal, using generated pseudo-random signal values. The developed audio card may be used for data storage, data archiving. A commercial application of the card is its use as repeaters; whereby audio signals are recorded and

playback at intervals. Digital repeaters may be used in a train station or a supermarket for advertisement. This will eliminate the rewinding time as in the case of analog ones, will result in reduced tape wear because the read head is contactless, and the playback will possess the same quality as in the

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER ONE

INTRODUCTION

The evolution of recording and reproduction of audio signals started in 1877 with the invention of the phonograph by T. A. Edison (Luc Baert, et. al., 1988). Since then, research and development to improve techniques have been geared towards the ultimate aim of recording and reproducing an audio signal faithfully; i.e. a high fidelity signal without introducing distortion or noise of any form.

With the introduction of the gramophone, a disc phonograph in 1893 by J. P. Berliner, a much better sound could be produced and reproduced easily. Further developments such as electric crystal pick-up in 1925 and broadcast AM radio stations in the 1930's evolved. Broadcast radio began its move from AM to FM, with consequent improvement of sound quality, and in the early 1960's stereo FM broadcasting became a reality. The basic media available then: tape, record, AM and FM broadcast, were all analog media.

In the 1960s and 1970, the professional open-reel tape recorders was the instrument used for both record production and for broadcast. However, due to its bulkiness and cost, the compact cassette, invented in 1963, made more impact in the homes for great number of purposes, such as recording speeches, dictation, taking notes for study, playing back etc. The invention has witnessed successive changes or improvement, from the standard playing (SP) 78 rpm record to the $33^{1}/_{3}$ rpm long-playing (LP) record, subsequently increasing the playing duration on each side from 2 minutes to 25 minutes respectively.

In spite of the spectacular improvement in analog technology, the original dynamic range is still seriously affected in the analog reproduction chain. Limitations are also posed to other factors affecting the system: frequency response, signal-to-noise ratio, distortion ..etc. exist simply due to the analog process involved. These reasons prompted research into digital techniques for audio reproduction, specifically pulse code modulation (PCM).

One of the problem is that the magnetic material of conventional analog tape recorders contains distortion components even before anything is actually recorded. Also the medium itself is non-linear,

i.e. recording and reproducing a signal with total accuracy is impossible. Hence, distortion is therefore built into every analog tape recorder. In PCM recording however, the original bit value pattern corresponding to the audio signal can be faithfully recovered, even if the recorded signal is distorted by the tape non-linearities and other causes, thus the audio signal itself.

The first public demonstration of PCM digital audio was in 1976 by Japan Broadcasting corporation (Eric D. Daniel and John R. Walkinson, 1988). The FM format having 12-bits, 2 channel, uses a 12-inch digital audio disc (DAD) read by laser, with a playing time of 30 minutes at 1800 rpm. The world first consumer digital audio processor was produced a year later with a playing time of 1 hour at 900 rpm. Recent development of the 90s include the portable compact disc (CD) player, car CD players, CD-ROM, and host of others.

The objective of this study is to develop a low cost, high-speed digital audio recording and playback system, utilizing the personal computer as the controller. The methodology involves the development of a digital audio card interface (software and hardware) that monitors, controls and also makes the input/output audio signal compatible with the I/O requirements of personal computers (PCs), so that, further processing on the signal can be achieved. The interface method of the card is via the parallel port. One advantage of the digital processing of analog signals is that factors which affect the transmission quality (such as noise, non-linearity, ,etc.) can be made arbitrarily low by proper dimensioning of the system. In addition, a digitally reproduced signal will exhibit the same basic qualities of the original signal.

Error detection and correction are very important and necessary processes in any digital communication system. Hence, digital audio systems use a complex method called Cross-Interleave Reed-Solomon Code (CIRC) to add a group of calculated bits to the data bits. These added bits enable the playback system to detect and in most cases, correct bit errors in the data words read out from the disc or tape.

Interfacing of peripherals to desktop computers can be achieved in two ways: either through the serial interface port or through the parallel interface port. Both techniques involve the transfer of data to/from the computer and the peripherals. The serial technique involves the transfer of data one bit at a time, over a single communication line to a receiver. While the parallel interface technique (employed in this research) involves the transfer of data bits in parallel, say eight bits (a byte) at a time. The serial

and parallel ports are found in virtually every microcomputer systems, which then makes their use to be worldwide. These ports are principally used for connecting input/output devices (I/O) such as keyboards, mouse, printers, visual display units, scanners and host of other sophisticated equipment. Hence, there are available standard connectors for each particular port, to allow for signal and pin compatibility and also to prevent wrong connections. The interface circuit uses a male 25-DB connector and plugged directly into the parallel port.

Considerable work was done by Dhananjay V. Gadre et. al., 1994, [illegible] National Instrument, Inc., 1996 in describing the techniques for the parallel port interfacing and the IEEE 488 interfacing respectively. The principles of these techniques are adequately presented in the literature review. It is possible to use the microcomputer and the various facilities provided in its basic input/output system (BIOS) to control computer external devices. It is this method that has been employed in this work to control the processing of the (analog and digital) audio signals.

1.0 Research Objectives

The research objectives are as enumerated below:

- To develop a low cost digital stereo audio card for DSA use on personal computers.
- To design a circuitry for digital recording of 16 bits per sample at sampling rate up to 44.1 KHz from analog audio sources.
- To produce a prototype card for the digital audio.
- To incorporate facilities for playing back of the digitally recorded audio signal.
- To develop a software driver for the audio card.
- To demonstrate how computer peripheral devices can be controlled via the parallel port.
- Exploration of the interface method using the serial interface which is a better alternative for long distance communication.

CHAPTER TWO

REVIEW OF LITERATURE

2.0 Introduction to Digital Signal Processing

Digital signal processing (DSP) started to gain popularity in the 1960s with the development of integrated circuits. Shown in figure 2.0 are the stages involved in a digital signal processing system. The most critical stage in the digital processing of analog signals are the conversion from analog to digital and vice versa. Although, the A/D and D/A conversions may seem relatively simple in principle, but they are technically speaking very difficult and may result in severe degradation of the original signal. As a result, this stage often generates a limiting factor that determines the overall system performance.

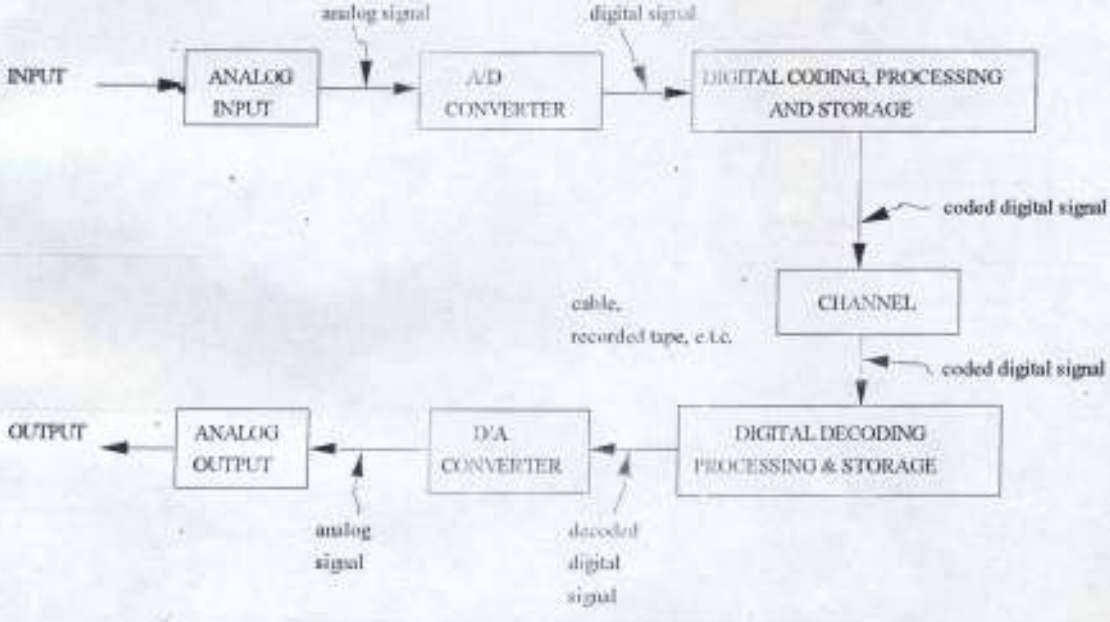
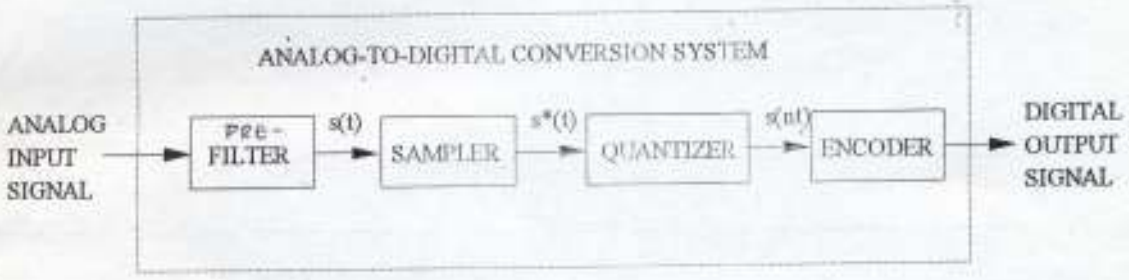


Figure 2.0 Basic digital signal processing system

Figure 2.1 is the block diagram depicting the processing sequence in an analog-to-digital conversion system.



s(t): analog signal - continuous in time, continuous in value.
s*(t): sampled signal - discrete in time, continuous in value.
s(nt): quantised signal - discrete in time, discrete in value.

Figure 2.1 Block diagram of an analog-to-digital conversion system.

The steps involved are described as follows:

- **Filtering** : this limits the analog bandwidth within the required frequency band.
- **Sampling** : the process of obtaining a discrete-time (digital)signal form the continuos time (analog) signal.
- **Quantisation:** the process of replacing a continuos-value signal by a finite (or discrete) signal of corresponding values.
- **Coding**: representation of the digitized signal in numeric form.

2.0.1 Filtering

A filter is an electronic device which is designed to pass signals in a selected band of frequencies. Two broad groups of filters are analog and digital filters. Analog filters may subsequently be grouped as passive filters (incorporating the passive elements, i.e. resistors, capacitors and inductors only) or active filters (in which the operational amplifier is of particular importance in their design).

Analog Filters

A passive, first-order, low-pass analog filter circuit and its frequency response is shown in fig. 2.2(a) and (b) respectively.

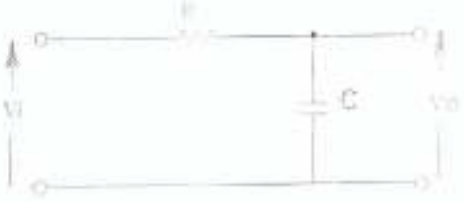


Fig. 2.2(a) Low-Pass Filter Circuit

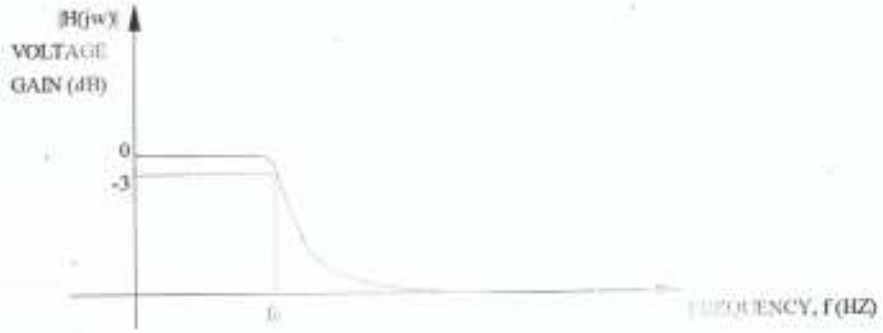


Fig. 2.2(b) Amplitude-Frequency Response of a Low-Pass Filter

Figure 2.2 Simple first order low-pass RC filter

The circuit incorporates a resistor and a capacitor and its transfer function is often described as a function of **S**, where **S** is the independent variable in the frequency domain. The transfer function of the low-pass filter circuit is given as follows:

$$H(s) = \frac{Vo(s)}{Vi(s)} = \frac{1}{1 + sRC} \quad \text{.....(2.0)}$$

$$H(j\omega) = H(s)\Big|_{s=j\omega} = \frac{1}{1 + j\omega RC} \quad \text{.....(2.1)}$$

where:

$\omega = 2\pi f$ is the angular frequency in radians per second

ω_c is the cut-off angular frequency of the filter also in Radian per second. Figure 2.2(a) shows the amplitude frequency response of the low-pass filter. The cut-off frequency of the filter f_c (in *Hertz)* can be calculated as follows:

$$\omega_c = 2\pi f_c = \frac{1}{RC} \qquad \text{.....(2.2)}$$

so that:

$$f_c = \frac{1}{2\pi RC} \qquad \text{.....(2.3)}$$

Other types of the analog filters includes: high-pass filters, band-pass filters, band-stop (or band-reject) filters, Butterworth filters, Chebychev filters e.t.c.

Digital Filters

A digital filter is a processing system which generates an output sequence, y(n), from an input sequence, x(n), whose I/O equation is of the general form:

$$y(n) = \sum_{i=0}^{M} a_i\, x(n-i) - \sum_{j=1}^{N} b_j\, y(n-j) \qquad \text{.....(2.4)}$$

where x(n) is the infinite series of numbers corresponding to the discrete-time signal obtained by sampling analog input signal x(t). Each value of x(n) corresponds to a sampling point at time $t = T_n$ for $-\alpha < n < +\alpha$. That is:

$$x(n) = \ldots,\ x(-2),\ x(-1),\ x(0),\ x(1),\ x(2),\ \ldots \qquad \text{.....(2.5)}$$

The coefficients a_o, a_1, ..., a_M and b_o, b_1, ..., b_N of eqn.(2.4) are constants which described the filter response. The filter is said to be recursive or cyclic whenever $N > 0$, which implies that past output samples are used in the calculation of the present output samples - see figure 2.3(a). However, when only the present and past input samples are used in the calculation of the present output sample, such a filter is said to be non-recursive or non-cyclic: because no past output

samples are involved in the calculation, the second term of eqn.(2.4) then becomes zero, since N is zero.

The block diagram of a non-recursive filter is illustrated in figure 2.3(b). The non-recursive filter is generally employed in digital audio system, such as the CPP - 102 compact disc player, that utilizes a 96^{th} order digital filter, which contains 96 multipliers and other devices that operates on 16-bit wide data word (Luc Baert, et. al., 1988). The constant coefficients are contained in a ROM look-up table.

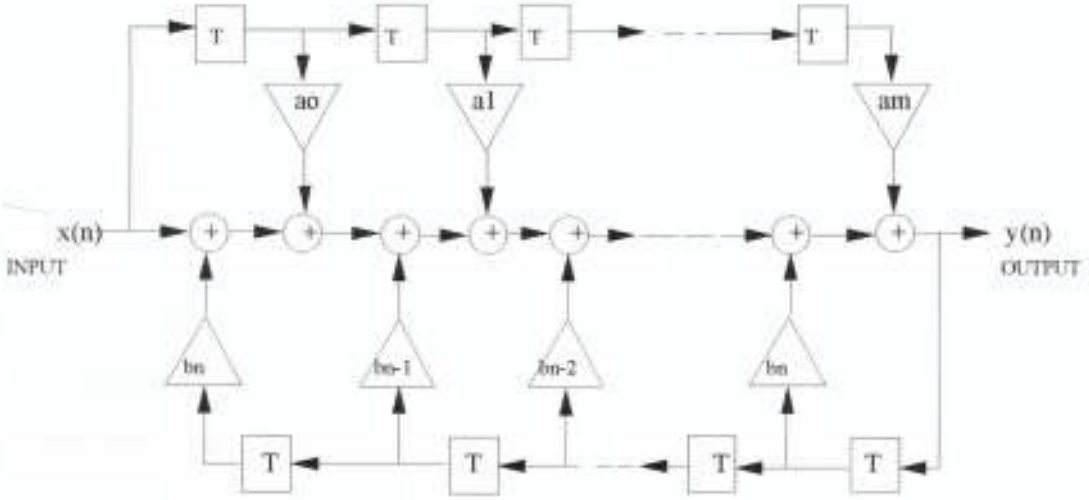


Fig. 2.3(a) Recursive Filter

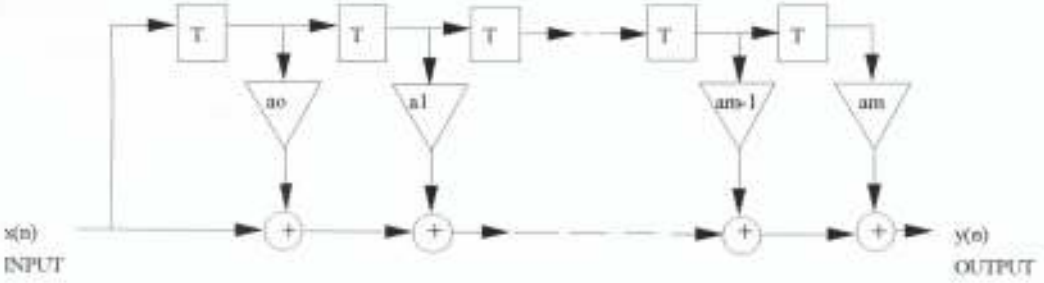


Fig. 2.3(b) Non-Recursive Filter

T ⇔ Unit delay(sampling period)
\+ ⇔ adder
Δ ⇔ Multiplier (coefficient = a_i or b_j)'

Figure 2.3 Digital Filter Block Diagrams

2.0.2 Sampling

In many applications, it is necessary to represent a continuous varying signal (i.e. analog signal) in terms of sampled values taken at appropriately spaced intervals. This fixed time interval between successive samples are called sampling intervals (t_s) and the whole process is referred to as *sampling*.

Although, the sampling operation may seem to introduce a rather drastic modification of the input signal (as it ignores all the signal changes that occur between the sampling points). However, it has been proved that the sampling process in principle removes no information whatsoever, as long as the sampling frequency is at least twice the maximum frequency of the input signal. This is referred to as *Nyquist Sampling Theorem* or *Shannon Sampling Theorem* (Dalghish R. L., 1987). The sampling process may be uniform, or random, or multi-rate as stated below:

- **Uniform sampling** ; sampling at regular time interval.
- **Random sampling** ; sampling instance are random with a particular distribution.
- **Multi-rate sampling** ; two simultaneous sampling operation with different time periods are carried out on the signal to produce the sampling output.

Figure 2.4(a) is the block diagram of a sampler while figures 2.4(b) and 2.4(c) illustrate an example of uniform sampling.

The Nyquist theorem can be verified by considering the frequency spectra of the input and output signals shown in figure 2.5. An analog signal $x_i(t)$, with maximum frequency f_{max} will have a spectrum having any form between 0 Hz and f_{max} Hz. Figure 2.5a(ii) shows the sampling signal $\delta_T(t)$, a unit impulse train having a fixed frequency f_S. The frequency spectrum, $\delta_T(f)$, can be represented by single lines occurring at f_S, $2f_S$, .. etc. as shown in figure 2.5b(ii). The sampling process is somewhat equivalent to a multiplication of $x_i(f)$ and $\delta_T(f)$ (i.e. convolution of $x_i(t)$ and $\delta_T(t)$). The spectrum of the resultant signal, figure 2.5b(iii), can be seen to contain the same spectrum as the analog signal, together with the repetitions of the spectrum modulated around the multiples of the sampling frequency. As a consequence, a low-pass filter (of the type in figure 2.2) can be completely isolated and thus can completely recover the analog signal.

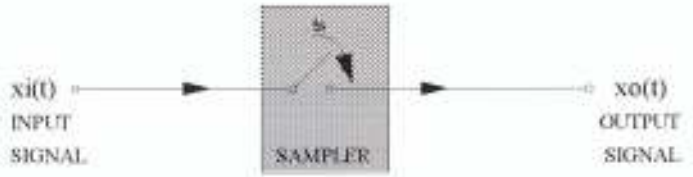


Fig. 2.4(a) Block diagram of a sampler

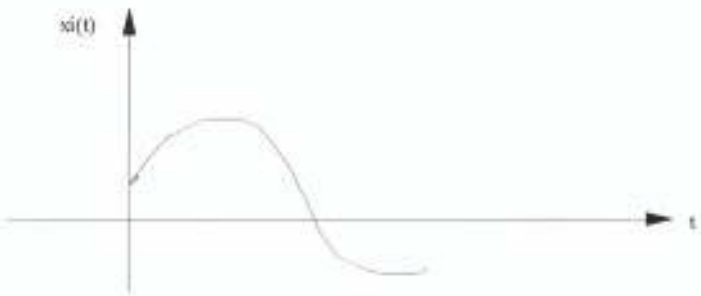


Fig. 2.4(b) Analog input signal $x_i(t)$

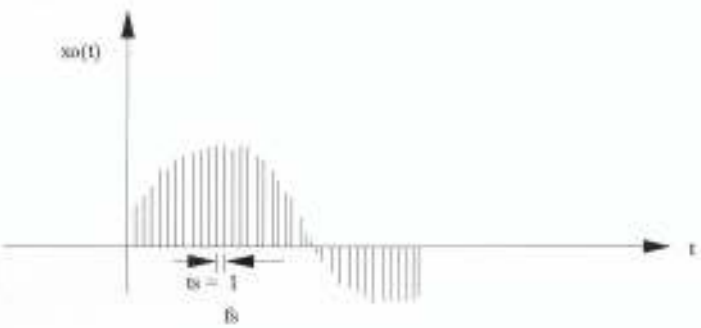
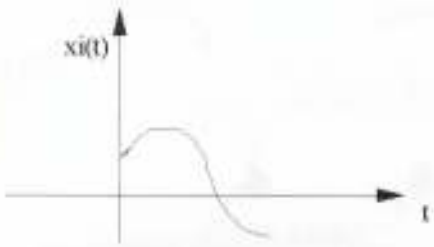


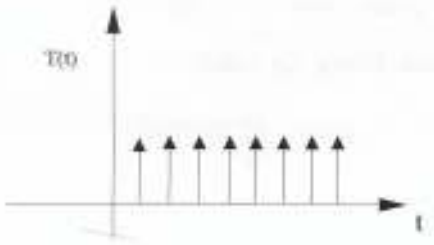
Fig. 2.4 (c) Output (sampled) signal $x_0(t)$

Figure 2.4 Illustatrion of uniform periodic sampling.

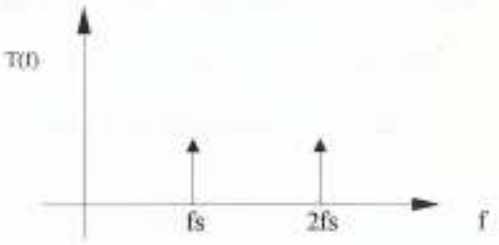


a(i) **Analog signal**

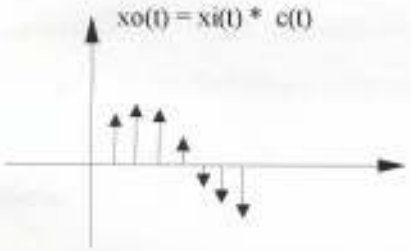
b(i) **Analog signal**



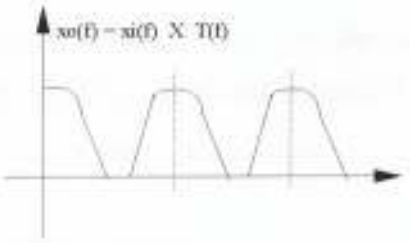
a(ii) **Impulse train signal $\delta_T(t)$**



b(i) **Impulse train signal $\delta_T(f)$**



a(iii) **Sampled signal**



b(iii) **Sampled signal**

Figure 2.5 Sampling processes in (a) time domain and (b) frequency domain.

Pragmatically, the sampling frequency f_s must be greater than $2f_{max}$, otherwise, the original spectrum will overlap with the modulated part of spectrum. This effect is called ***frequency foldover***, or ***frequency aliasing*** . The frequency aliasing can be prevented either by increasing the sampling frequency or by placing an anti-aliasing filter in front of the sampler (Charles L. Philips and H. Troy Nagle, 1990). The anti aliasing filter is a very sharp cut-off, low-pass filter that removes frequencies that are greater than $fs/_2$ (i.e. half the sampling frequency).

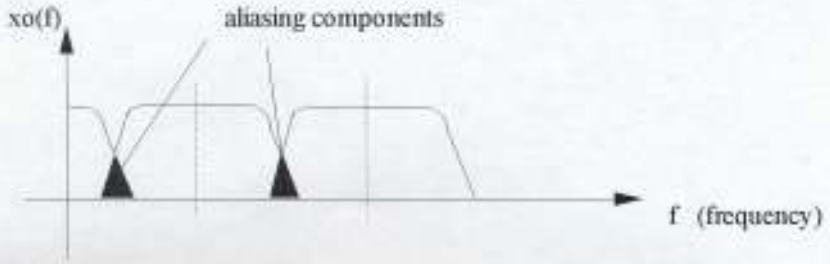


Figure 2.6 Aliasing phenomenon due to too low sampling frequency.

It is obvious that the selection of the sampling frequency in compliance with the Nyquist theorem is very important. But, since ideal low-pass filter do not exist, a certain safety margin must be incorporated in order to avoid any frequency higher than $fs/_2$ passing through the filter with insufficient attenuation. However, selecting too high a sampling rate would drastically increase the hardware cost.

For an audio signal with a typical bandwidth of 20Hz to 20KHz, the lowest sampling frequency which corresponds to the Nyquist rate is 40KHz. At this frequency, a very steep and, consequently very expensive reconstruction filter is required. Therefore a sampling frequency of approximately 44.1KHz is typically used, allowing the use of an economic reconstruction filter. An example of the frequency response of a reconstruction filter is illustrated in figure 2.7, which is flat until 20KHz but with sufficient attenuation at 22KHz to make possible aliasing components inaudible.

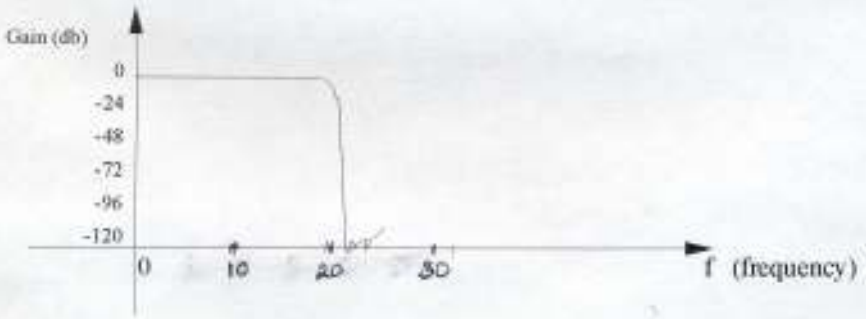
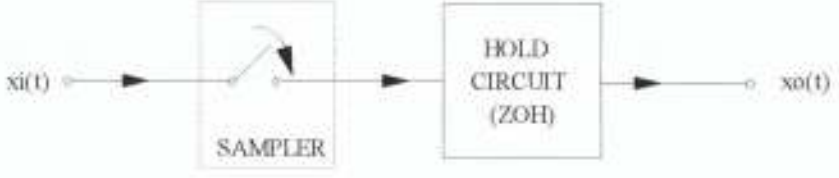


Figure 2.7 A reconstruction filter frequency characteristic

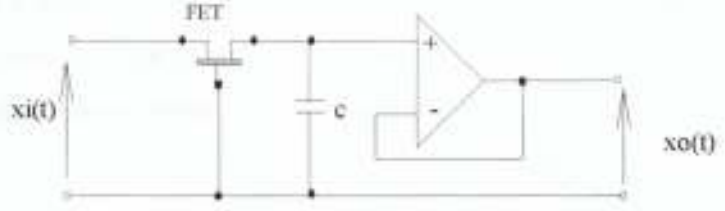
The standard sampling rate for compact disc is 44.1KHz. The same sampling rate is also used in the reproduction of commercial albums released on R-DAT and S-DAT cassettes (Luc Baert, et. al., 1988). 48 KHz is used for the highest-quality recording and playback systems.

Sample-and-Hold Circuits

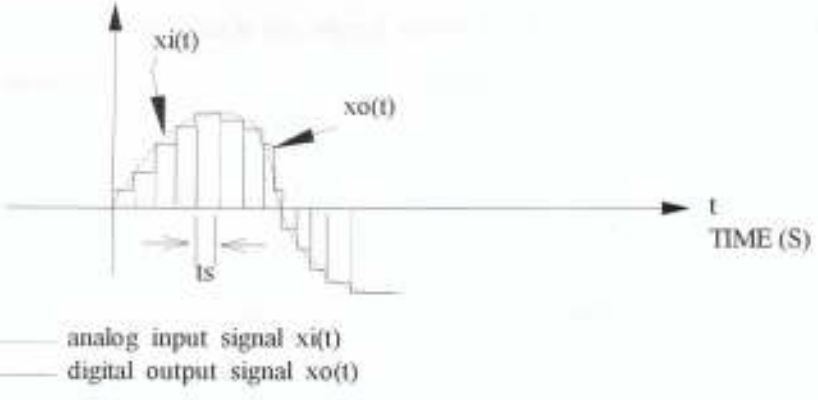
In applications such as digital audio recording whereby one needs to digitize rapidly changing signals, a severe problem which may occur is for the input analog signal to change before the completion of the analog to digital (A/D) conversion. This may present some error values in the output digital words. Such errors can be greatly reduced or totally eliminated by connecting a sample-and-hold circuit to the input of the A/D converter. It samples the analog signal and stores it for a time; during which the signal value can be converted by the A/D converter into a digital code. The principle of a sample-and-hold circuit is relatively simple as depicted fig. 2.8(c) for a sampler with zero order hold (ZOH).



(a) Block diagram of a Zero-Order-Hold (ZOH) sampler



(b) FET input sample-and-hold circuit



2.8(c) Input-Output characteristics

Figure 2.8 Sample-and-hold circuit

Prior to each A/D conversion, the electronic switch is briefly closed to allow the capacitor to charge up to the voltage at the input signal and then rapidly opened. After the signal is acquired on the capacitor, the ADC is signaled to perform a conversion on the stored sample. Therefore, each sample accurately represents the voltage on the input when the sample was taken, regardless of the rate at which the signal was changing. Sample-and-hold circuits are not only used in A/D conversion, but also in D/A conversion, to remove transient (***glitches***) from the output of the converter. In this case, the sample-and-hold circuit is usually refer to as ***deglitcher***.

2.0.3 Quantisation

Even after sampling, the output signal will still be continuous in value, however it is discrete in time. There is a need to make the sampled signal amplitude to be discrete in value by a process known as ***quantisation***.

In practical systems, the analog signal range is divided into a number of regions, and samples of the signal are assigned a certain value according to the region in which they fall. (See figure 2.9 for an example of a quantised signal, x(t)) These values will subsequently be coded accordingly when fed into an encoder. The example shows a bipolar system in which the input voltage can be either positive or negative, which is the normal case for audio signals. The coding system employed in figure 2.9 is sign and magnitude as shown in Table 2. The code is similar to the natural binary code but with an additional sign bit. The MSB is used as a sign bit; a sign bit of '0' represents positive values, while a '1' denotes that the value is negative (Ronald J. Tocci, 1980). The regions into which the signal range is diverted is called ***quantisation intervals***. A series of bits representing signal value corresponding to each quantisation interval is called a ***code word***.

Since any signal value in a certain quantisation interval are represented by the centre value of this interval, the process of quantisation is therefore a non-linear process and creates an error, referred to as ***quantisation error*** or ***round-off-error***.

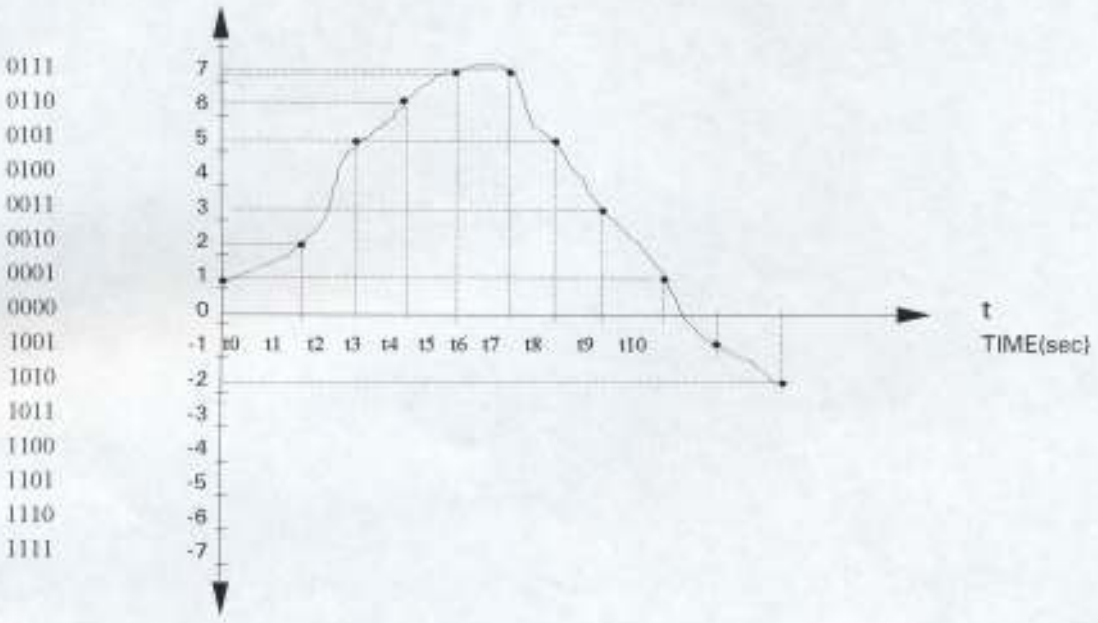


Figure 2.9 Quantised signal s(t)

TABLE 2.0 Quantised level coding in sign and magnitude form

SAMPLING POINTS	t0	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10
QUANTISED LEVEL	1	2	5	6	7	7	5	3	1	-1	-2
4-BIT ENCODED OUTPUT	0001	0010	0101	0110	0111	0111	0101	0011	0001	1001	1010

2.1 Coding Techniques

In principle any digital coding system can be adopted to represent the different discrete levels in A/D or D/A conversion, as long as they are properly defined. However, some are better in certain applications than the others. Two main group in existence are; the bipolar and unipolar codes.

Bipolar codes give information on both the magnitude and the sign of the signal, which makes them preferable for audio applications.

2.1.1 Unipolar Codes

Unipolar codes are restricted to the representation of positive values only. Examples of such codes areas follows:

- Natural binary codes:
- BCD code i.e. the well known 4-bit code in which the maximum value is 1001 (decimal 9) after which the code is reset to 0000
- Gray code which is advantageous in a situation whereby changing of only one bit per transition is important. This code is not often used for audio applications.

2.1.2 Bipolar Codes

These codes are similar to the unipolar natural binary code, but with the inclusion of a sign bit. Examples of such codes are as follows:

- **Sign and magnitude**: similar to the normal unipolar code but a sign bit to express its polarity is added. The MSB is normally used as the sign bit. Conventionally, a '0' at the MSB is used to depict positive values and a '1' for negative values as illustrated in Table 2.0 for the sampled signal of figure 2.9.

- **Offset Binary**: this is a natural binary code, but with zero at minus full scale (Oni, J.O., 1983).

- **One's complement**: here, negative values are represented by the one's complement of its absolute value. The one's compliment of a number can be obtained by interchanging 1's for 0's and vice versa.

- **Two's Complement**: arithmetically, two's complement of a binary value is obtained by adding a 1 to its 1's complement and disregarding any overflow. The code is very similar to the offset binary, but with the sign bit inverted.

2.2 Code Errors and Error Compensations

When digital signals are sent through a transmission channel or recorded and subsequently played back, different types of ***code errors*** can occur. Even the smallest error, say in the MSB, can cause unsatisfactory audible effect. Such results are entirely unacceptable in the high quality expected from digital audio, hence, the need for an effective error-detecting and error-correcting scheme. If correction fails, then error concealment (to cover-up the faulty information) will be carried out so that at least no audible disturbance will result.

The four major types of code errors, namely dropouts, jitters, intersymbol interference and noise are described as follows:

- **Dropouts:** they are caused by dust or scratches on the magnetic tape or CD surface or microscopic bubbles in the disc coating. Tape dropout may result in a relatively long-time errors called ***bursts***, in which long sequences of related data are lost together. It may also results into single random errors.
- **Jitters:** tape jitters cause random errors in the timing of defected bits and to some extent is unavoidable due to the tape transportation mechanism properties.
- **Intersymbol interference:** at very high bit rate, a detected pulse if wider than the original pulse will result in interference between adjacent bits. This is known as intersymbol interference or time crosstalk, and causes random errors.
- **Noise:** this may have similar effect as dropouts or result in random errors in the case of pulse noise.

Thus, the three basic areas involved in the effective treatment of errors are ***detection***, ***correction*** and ***concealment***.

2.2.1 Error Detection

Simple parity may be used to detect whether a data word contains an error. The simplest way is to add one extra bit, called parity bit, before transmission. The parity bit takes a value of 0 or 1, depending on the number 1's in the word itself. The transmitted word must also have the required

number of 1's, otherwise a transmission error is said to have been detected. Two systems are possible: odd parity and even parity checking.

- **Odd parity:** the added bit is so as to make the total number of 1's in the word to be odd. Two examples of odd parity bit and even parity bit inclusion are as follows :

[1101100] [1] ⇔ [11011001]

data word ↑ odd parity transmitted word

odd parity bit

[1011011] [0] ⇔ [10110110]

data word ↑

odd parity bit odd parity transmitted word

- **Even Parity:** the added bit must make the total number of 1's in the word to be even.

[1101100] [0] ⇔ [11011000]

data word ↑ even parity transmitted word

even parity bit

[1011011] [1] ⇔ [10110111]

data word ↑

even parity bit even parity transmitted word

Although, this is simple to achieve but it has two major short comings stated as follows:

- Even if an error is detected, there is no provision of knowing the faulty bit.
- If two bits of the same word is faulty, the errors compensate each other and no error will be detected.

These disadvantages are overcome by utilizing a more complex form of parity checking, such as the combinational parity checking. The parity code used for error detection in most digital systems today are cyclic redundancy check (CRC) codes (Dimitri Bertsekas and Robert Galleger, 1987).

2.2.2 Error Correction

The correction of binary data is ensured by using some form of block encoding scheme, incorporating another error-correcting bits, in a process called *redundancy*. These added bit are called redundant bits, and it is a variable in the definition of the code rate, such that the ratio:

$$\frac{\text{data} + \text{redundant data}}{\text{data}}$$

is known as the code rate. The two forms of error correcting codes illustrated in figure 2.10 areas follows:

- **Block codes**: data bits and error bits are arranged in separate blocks.
- **Convolution codes**: data and error bits are mixed within each block.

(a) Block code

(b) Convolution code

Figure 2.10 Format illustration block and convolution error correcting codes

Combinational (Horizontal/Vertical) Parity Checking

An example of a 16-bit binary word or message arranged in a 4 X 4 matrix using combinational (horizontal/vertical) parity checking scheme is given in Figure 2.11. This schemes is such that the bits of each binary word are arranged in matrix form. Then, to each row and column one or more bits is added to make the parity for that row or column odd (or even if even parity is required).

Then a final bit is added in the lower right-hand corner to make the last column odd which automatically makes the last row to be odd (Milos D. Ercegovac and Tonas Lang, 1991).

1011	0
1100	1
1011	0
1010	1
1001	1

Figure 2.11 Combinational parity checking

The entire array may be transmitted in sequence row by row (or column by column). If any error occur to one bit during transmission, parity check on the bits row and column will fail; the error will be found at the intersection, and consequently be corrected by inverting the bit. The entire array of 25 bits, of which 16 are data bits, form a code word. There are 9 (25 - 16) redundant bits. Most other error correcting codes are based on these ideas. The only limitation in this type of bit-by-bit error correction is that it may break down whenever the error occur in more than one bit. However, this problem is overcome in word-by-word error correcting coding scheme, such as crossword coding.

Crossword Code

The combinational parity checking just explained deals with error correction bits by bits. It is not uncommon in digital application for errors to come in bursts, with complete clusters of faulty bits, hence, such bursts or errors demands powerful error correcting codes. An of such code is the *crossword code*. A simple illustration of crossword coding is illustrated in figure 2.12. The crossword code utilizes a matrix scheme similar to the previous example, but it carries out its operation with the whole words instead of individual bits. It should be noted that decimal values are used to represent binary values or word for simplicity sake.

RECORDING

DATA [4] [6] [2] [9]

⇓

[4] [6] → [10] } Insertion of error correcting codewords which when added, make 20 both in the horizontal and vertical rows.

[2] [9] → [9]

↓ ↓

[14] [5]

⇓

[4] [6] [2] [9] [10] [9] [14] [5] ← Recording pattern on tape

PLAYBACK

[4] [8] [2] [9] [10] [9] [14] [5] Error occurred in the shaded code word

⇓

[4] [8] → [10] } = 22 It can be seen that the code word at the intersection of the row and column which added to 22 (and not 20) is

[2] [9] → [9] } = 20 incorrect. It must be in excess of 2 (i.e. 22 - 20), so the correct code word will be 6 (i.e. 8 -2).

↓ ↓

[14] [5]

20 22

PLAYBACK [4] [6] [2] [9] ← Playback pattern after correction and removal of redundant bits

Figure 2.12 Illustration of Crossword Code

CIRC Code and Other Codes

Many other error-correcting codes exist, but most digital audio systems use a complex method of coding called Cross-interleaved Reed-Solomon Code (CIRC) to add a group of calculated bits to the data bits. These added bits enable the playback system to detect and, in most cases, correct bit errors in the data words read out of tape or disc. CIRC is designed for correction of long burst errors. It also provides interpolation of the audio signal for the detected, but uncorrectable, long burst of errors. This interpolation of the signal, called error concealment, is one effective compromise solution for very long burst errors in audio and video applications (Arvind M Patel, 1988). However, error concealment is not useful for computer applications.

2.2.3 Error Concealment

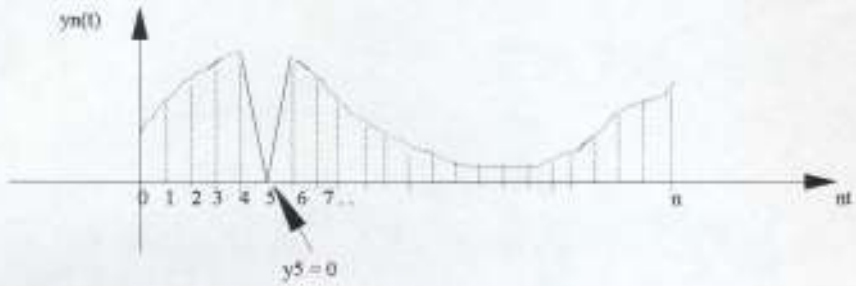
The technique which prevents any uncorrected code errors from affecting the quality of the reproduced sound is known as concealment. The four typical methods of error concealment (which includes muting, previous word holding, linear interpolation or averaging, and higher-order interpolation) are described as follows:

- **Muting:** a rough method of concealment whereby the erroneous word is muted, i.e. set to zero as depicted in figure 2.13(a).

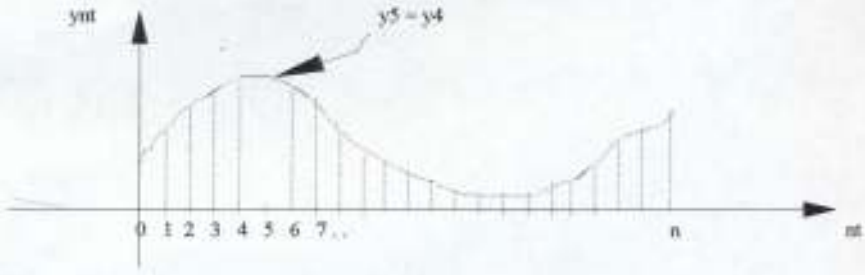
- **Previous Word Holding**: the value of the word before the erroneous word is used instead. This normally produces no audible difference, however, the result may be unsatisfactory at high frequency which employs high sampling frequency. Fig 2.13(b).

- **Linear Interpolation (Averaging):** the erroneous word is replaced by the average value of the proceeding and succeeding words. Fig. 2.13(c).

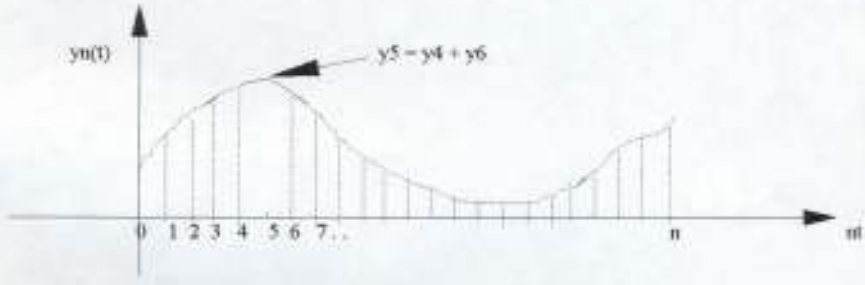
- **High-order polynomial Interpolation**: the erroneous word is estimated taking into account a number of preceding and succeeding words



(a) Muting



(b) Previous word holding



(c) Linear interpolation

Figure 2.13 Three types of error concealment with the error occurring at the sample y_5

Interleaving

In view of the high recording density of digital systems, dropouts leading to burst error could destroy many words simultaneously. Error correction in this situation may failed and concealment may be unsatisfactory. For this reason adjacent words are not written next to each other on the tape or disc, but at location sufficient distance apart to make sure that they can not suffer from the same dropout. In effect, this method converts possible long burst error into distributed error, which can easily be corrected.

The words are arranged in interleaved blocks by the process known as ***interleaving***. Figure 2.14 shows how interleaving can be used to effectively reduce bursts, affecting 4 code words, to 4 random errors. The bursts affected the consecutive code words, W2, W8, W11 and W3, placed in the second interleaved block is seen to reduce to random errors. The resulting random errors may then be treated using any of the earlier described methods.

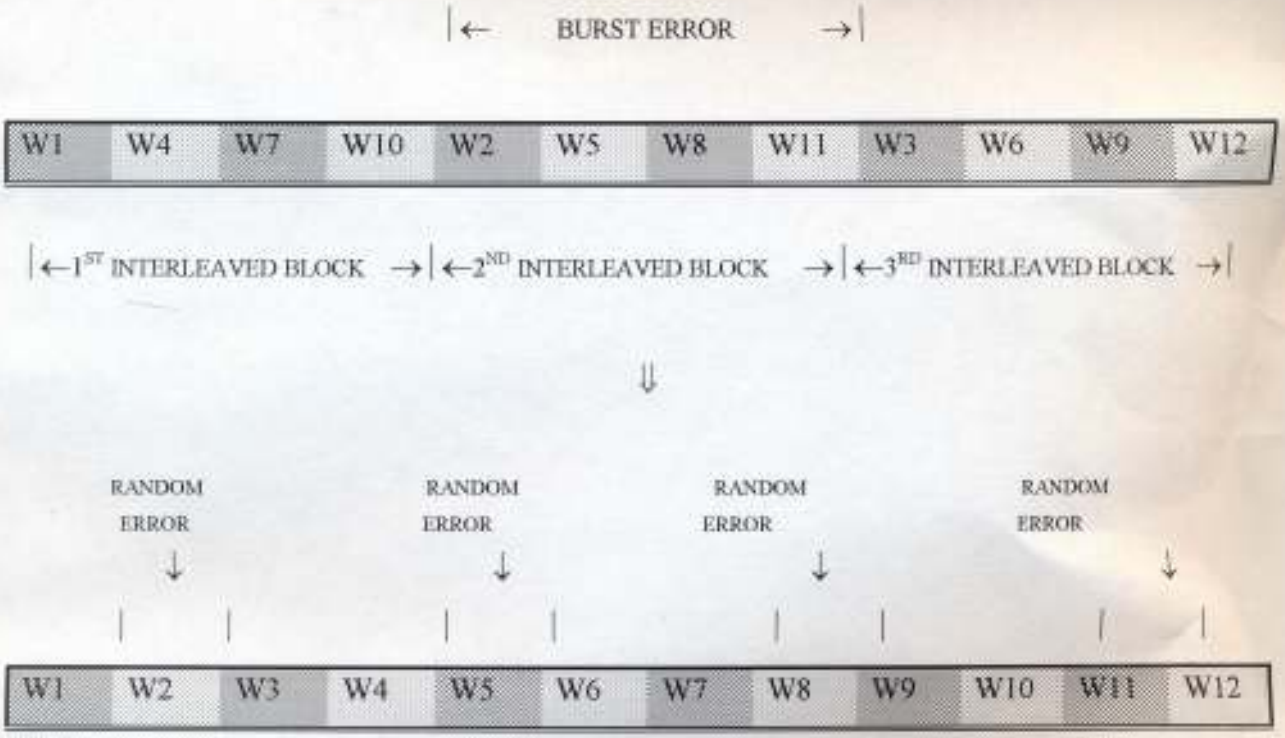


Fig. 2.14 Illustrating how interleaving of data effectively reduces burst error into random errors.

2.3 Conventional Use of the Personal Computer Ports

Personal computer ports are conventionally used for the connections of its I/O peripherals, such as keyboards, mouse, printers, visual display unit and host of others. Today, most Scientist and Engineers are using PCs' parallel and serial ports for laboratory research, industrial control, test and measurements. Obtaining special result from such PC based data acquisition system (DAQ) depends on each of the system's elements, such as the PC type, the transducers employed, the signal conditioning , the DAQ hardware and software capabilities.

2.4 Input/Output Organistaion

The bits comprising a data word within most microprocessors are transferred from point to point in parallel form, i.e. using one line for each bit. This mode is referred to as parallel communication. Whereas this allows very fast data transfer, it is not often economical to transmit data in this form, particularly over long distances. This is because special line drivers and receivers have to be used together with multicore cable. An alternative which is widely used is commonly called serial communication. It uses a transmitter to send data, one bit at a time, over a single communication line to a receiver. This simplifies interconnection with I/O devices, but at the expense of lowering the data transfer rates. Yet, the resulting rates are adequate for many I/O devices (such as the keyboard, the mouse, the bar code scanner, e.t.c.) that are often equipped with serial interface port for connecting the respective devices.

Before a communication can be established between any intelligent device and a CPU, there should be a way of addressing the device. In microcomputers, the scheme is to provide separate address spaces (of about 64K bytes) associated with I/O and status registers for the peripheral devices (Hamacher V. Carl, et. al., 1989). The assignment of these addresses to specific devices varies between PC models. In a situation whereby a CPU lacks a separate I/O address space, the I/O ports may be treated just as if they are memory locations. This scheme is referred to as memory mapped I/O. The following techniques are used for synchronizing data transfers are described as follows:

1. **Programmed I/O:** Programmed-controlled I/O involves the scanning of device status by the CPU under the control of internal programs. This requires continuous involvement of the CPU, though only a minimal amount of external hardware is needed to connect the peripheral devices to the computer. Examples of programmed I/O include are ***direct*** and ***polled I/Os***.
2. **Interrupt-Driven I/O**: data transfer is initiated by the device by sending an interrupt service request signal to the CPU, hence, eliminating the time wasted by the CPU during repetitive status check that is necessary in the case of programmed I/O.
3. **Direct Memory Access (DMA):** The direct memory access allows data flow directly to/from the memory. The DMA controller relieves the CPU of all I/O functions except for the initialization of the transfer parameters.

4. **Channel I/O (or I/O processors):** Due to the fact that it will be uneconomical to provide a separate DMA controller for each peripheral device, hence, the concept of the channel I/O came about as a means of providing a shared DMA controller for a number of devices. I/O channels are peripheral processors, which are computers in their own right, but their instruction sets are limited to I/O functions (Protopapas D. A., 1988).

Most dedicated I/O devices are TTL compatible and their output are capable of driving one TTL load only. The current sourcing capability is very low (tens of microamperes only) but the sinking capability is of the order of 1.6 mA (Vears R. E., 1992). These figures indicates that few peripheral devices can be directly interfaced without the aid of some form of electrical buffering. For most circuits, this generally consist of some form of current amplification and perhaps changes in voltage level.

2.5 Serial Interface Standards

Serial communication is a scheme whereby data are being transferred one bit at a time over a single communication line. It is usually employed for data transfer over long distances or when low rates are desired and it occurs either in synchronous or asynchronous format.

In asynchronous serial communication, data words, called characters, are transmitted independent of each other. Also, there may be variation in the time interval between successive data words. Whereas in synchronous communication format, data are transmitted in blocks instead of the character-by-character basis. A character may consist of up to 8 data bits, with an optional parity bit, all (braced) within a start bit and stop bit(s) as shown in figure 2.15. Synchronous communication can support higher data rates by keeping the receiver in synchronism with the transmitter. Hence, maintaining transmission during the idle time between successive frames. Synchronous protocols in which the building blocks of a frame are characters, are referred to as character-oriented protocol (COP). Although, it has a shortcoming when bit-oriented information (e.g. graphic information) are to be sent. The problem is overcame by using a bit-oriented protocol (BOP) (Protopapas D. A., 1988).

Some examples of serial communication standards include; RS-232, RS-422, RS-423, RS-485, RS-449 and others. The most widely used is the RS-232 which defines the interface between data terminal equipment (DTE) and data circuit-terminating equipment (DCE). Computers operate as DTE devices

and communicate with DCE devices, such as modems, printers and programmable instruments in general.

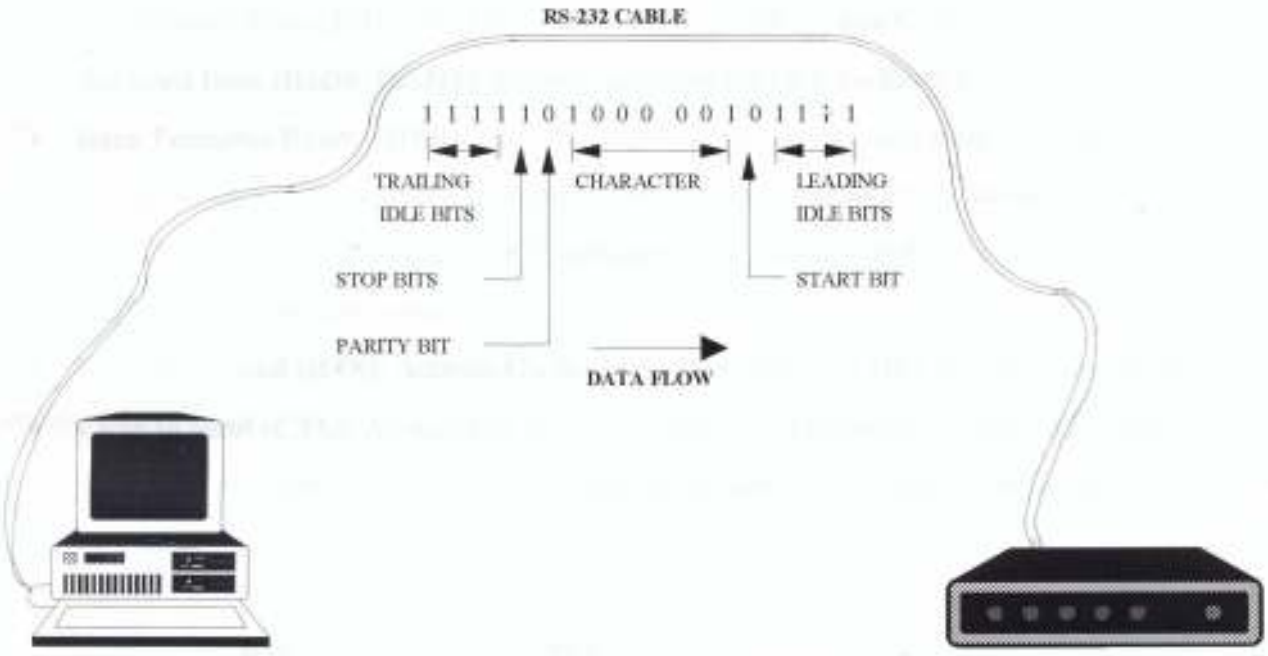


Figure 2.15 Example of serial communication (RS 232 configuration)

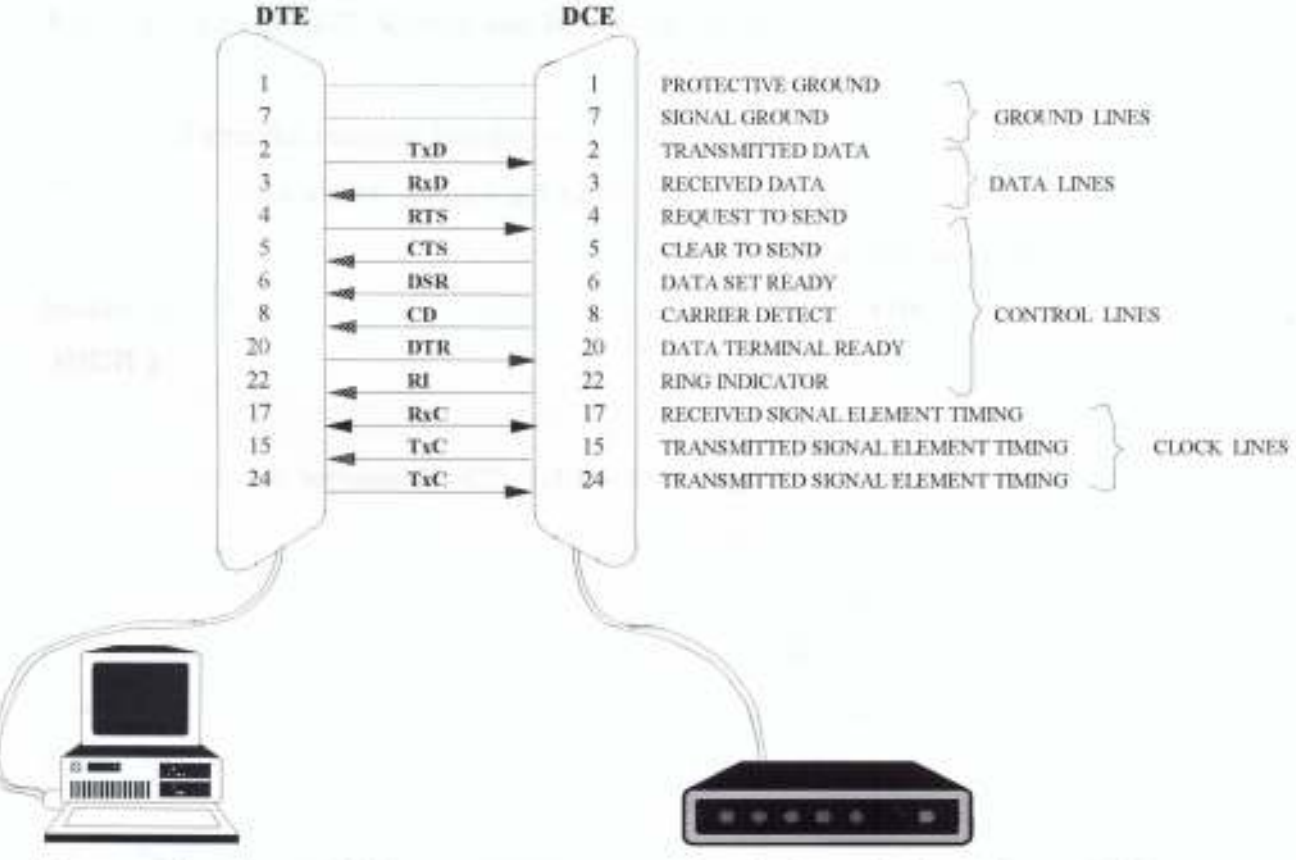
2.5.1 The EIA RS-232C

The EIA RS-232C stands for the Electronic Industries Association recommended standard 232 current version. It is virtually the most common standard in short distance communication. The majority of computer hardware and peripherals, such as the mouse and the keyboards have RS-232 interfaces for their connections. It defines a total of 20 lines, although a smaller subset can also be employed. The RS-232 has connectors that vary from 4, 9 and 25-pin configurations. Also indicated in the figure are the lines that are usually employed in the implementation of serial I/O communication when using the RS-232 protocol. The minimum line requirements for the serial interfacing are the four lines; transmitted data, received data, signal ground and protective ground.

Shown in figure 2.16 are the pneumonics used in practice for the signals carried by the lines. The communication pathway may be simplex (one transmitter), half duplex (asynchronous) or full duplex (simultaneous). The four lines (DTR, DSR, RTS and CTS) are used for hand shaking. So, these four

lines must all be active during synchronous data transmission. The functional characteristics of the lines are as follows:

- **Transmitted Data (TxD):** The DTE uses this line to transmit data to the DCE.
- **Received Data (RxD):** The DTE receives data from the DCE via this line.
- **Data Terminal Ready (DTR):** The DTE activates this line for indication of its in-service status so that the DCE can subsequently connects itself to the communication channel.
- **Data Set Ready (DSR):** The DCE activates this line to signify that it is connected to a communication line and is ready.
- **Request to Send (RTS):** Activated by the DTE to condition the DCE for data transmission.
- **Clear to Send (CTS):** Activated by the DCE to indicate its readiness for data transmission.
- **Ring Indicator (RI):** The DCE activates this line to indicates its reception of a ring signal through the common line.



Data terminal equipment (**DTE**) Data circuit-terminating equipment (**DCE**)

Fig 2.16 Typical EIA RS-232C Connector(25 - Pin 'D' type) with pin assignments.

- **Carrier Detect (CD):** Also called the received signal detector line, and is activated by the DCE to indicates its reception of a carrier signal suitable for modulation.
- **Transmitted Signal Element Timing (TxC):** Used by either DTE or DCE to convey a clock signal for synchronous purposes.
- **Received Signal Element Timing (RxC):** Used in synchronous communication by the DCE to provide a received clock to the DTE.
- **Signal Ground:** Common ground reference for all signal lines is established via this line.

The maximum data rate in bit per second (bps) supported by the RS 232C standard is 20Kbps over cable up to 50ft (John Litteler and John Maher, 1989) and the voltage range of -3V to -25V is used to represent a '***LOW***' while a '***HIGH***' is represented within the voltage range of +3V to +25V. But this voltage ranges are not usually been provided by the microcomputers which operate at the standard of +5V TTL/CMOS levels. Hence, an additional power supply and line drivers are often required.

2.5.2 The EIA RS-422, RS-423, and RS-449 Standards

The RS 422 specifies balanced line drivers (i.e. the use of two wires for each signal) with differential I/O line to provide a greater distance and higher data transfer rate than the RS-232. It also allows the interface to be implemented using the 5.0V supply from most microprocessor based equipment, because of its narrower transition region (-3.6V to -6.0V for a '**LOW**' and +3.6V to + 6.0V for a '**HIGH**').

The only difference between RS-422 and RS-423 is that RS-423 specifies an unbalanced electrical interface - i.e. single ended ones. RS 422 may allow a data rate of up to 2Mbps over about 200ft cable while RS-423 would only permit about 140Kbps. The RS-499 defines only the functional and the mechanical aspects of the serial interface, but with the RS-232, RS-422, and RS-423 compatibility in mind. So, while complying to 422 and 423 it can support higher data rates and cable lengths. RS-449 define a total of 30 interface lines (Protopapas D. A., 1988).

2.6 Parallel Interface Standards

The parallel interface port on the IBM (and compatibles) desktop computers is the easiest to understand. The port is used primarily to interface with printers. Most devices that connect to it are

wired at the factory to be compatible. The standard is to provide a DB 25-pin female connector on the computer end and a centronic style 36-pin (with only 25 pins wired) connector on the printer end.

Figure 2.17 shows a typical parallel port and the connector pin arrangements. Each parallel port may be viewed to actually consist of three independent ports - two output ports and one input port. The output ports consist of an 8-bit wide port, commonly called ***Data Port*** and a nibble (4-bit) - wide port referred to as the ***Control port***. The input port which is designated as the ***Status port*** is a byte wide, although its three lowest bits are unused. All these ports operate at the standard +5Volts TTL/CMOS levels.

Most I/O software drivers usually provide for the control of two parallel interface ports, conventionally labeled LPT1 and LPT2. The difference between these two ports is only in their I/O address specifications. The signal assignments of the DB 25-pin and the 36-pin centronic type parallel port connectors and the mapping of their pins is depicted in Table 2.0. (Cyrillic, 1995)

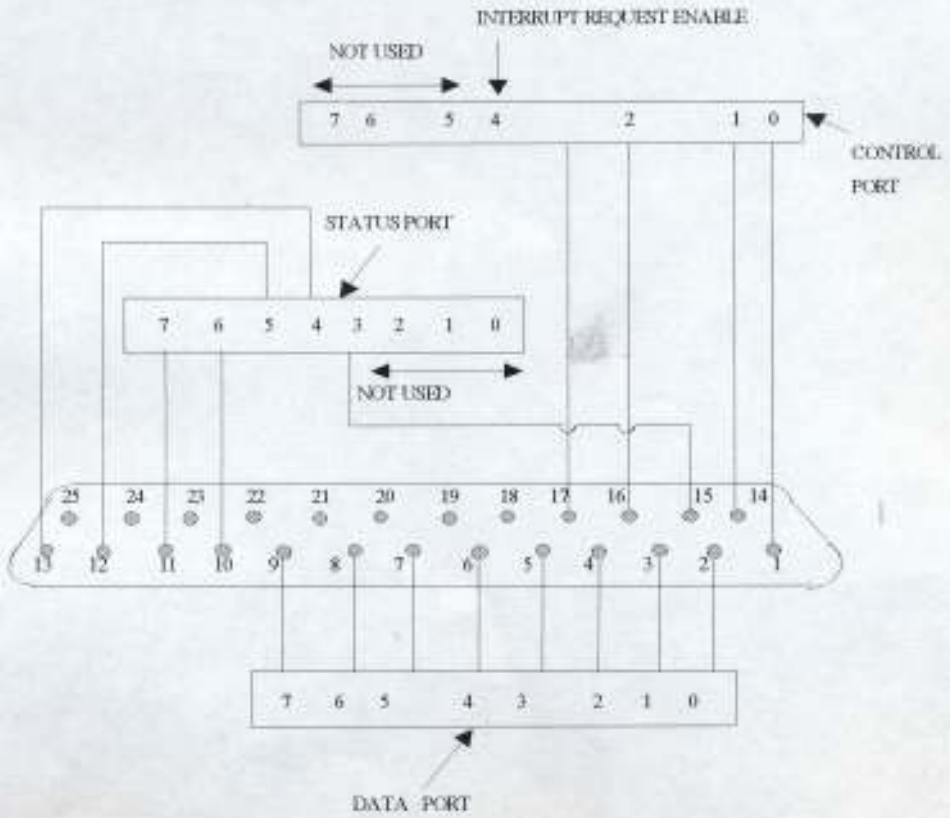


Figure 2.17 A typical parallel interface port

Table 2 : Signal Assignments of the (25-pin and 36-pin) Parallel Port Connectors

DB 25-pin	36-pin Centronic type	Register (port)	In/Out	Bit	Name	Function
1	1	control	Out	C_0*	STROBE	Signal by the computer to printer commanding that data be read.
2-9	2-9	data	Out	D_0 to D_7	DATA_0 to DATA_7	Bit of character sent to printer.
10	10	status	In	S_6	ACK	Signal by printer to computer to indicate data received.
11	11	status	In	S_7*	BUSY	Signal by printer to computer to indicate that printer is busy.
12	12	status	In	S_5	PAPER END	Signal by printer to computer to indicate paper out.
13	13	status	In	S_4	SELECT	
14	14	control	Out	C_1*	AUTO FEED	Signal by computer to printer that auto line feed is required.
15	32	status	In	S_3	ERROR	Signal by printer to computer that an error condition exist
16	31	control	Out	C_2	INITIATE	Signal by computer to instruct printer to reset itself
17	36	control	Out	C_3*	SLCT_IN	
18-25	16, 17, 19 to 30, 33				GROUND	GROUND
	15, 18, 34, 35	-	-	-	NC	Not used

* **Indicates that the signal bit is inverted.**

The list of the common I/O address scheme of the parallel port interfaces are specified in Table 3. The ports I/O addresses for a specific computer can be determined from its manual, or on the other hand from the operating system setup.

Table 3: Parallel port I/O addresses in hexadecimal

Parallel port type	PC (LPT1)	PC (LPT2)	PC/AT(LPT1)	PC/AT(LPT2)
Data port	378	3BC	378	278
Status port	379	3BD	379	279
Control port	37A	3BE	37A	27A

It should be noted that the signal bits of the DB 25-pin parallel port at pins 1, 11, 14, and 17, which are starred use inverted logic. The choice of computer language to be used is often a matter of personal preference. But, some languages provides the capability of linking in assembly modules, such as C and C++ computer languages.

2.6.1 The ANSI IEEE 488 Parallel Interface

The America national Standard Interface (ANSI) IEEE 4888 was generally accepted in 1975, although designed by Hewlett-Packard in 1965 (National Instrument, Inc., 1996). Hence, the IEEE 488 bus is also called Hewlett-Packard Interface Bus (HP-IB) or General Purpose Interface Bus (GPIB). The interface system consist of sixteen (16) signal lines and eight (8) ground return of shield drain lines. The 16 signal lines are grouped into eight data lines, three handshake lines and five interface management lines, as shown in figure 2.18. (Kehinde L. O. and Ojetayo T. K., 1992)

Data Lines

The eight data lines (DI01 - DIO8) carry both data and command messages. The state of pin 11, the Attention (ATN) determines whether is data or command. All commands and most data use the ASCII or ISO code set, so that the eighth bit (DIO8) is either unused or used for parity.

Fig. 2.18 IEEE 488 parallel Interface Connector and the Signal Assignments

Handshake Lines

The three handshake lines (NRFD, NDAC, and DAV) asynchronously control the message bytes transfer between devices. They also guarantee that message bytes on the data lines are sent and are received without transmission error.

- NRFD (not ready for data) line - Driven by all devices in the receiving mode to indicate whether it is ready or not ready to receive a message byte.
- NDAC (not data accepted) line - Driven by a receiving device to indicate whether it has accepted a message byte or not.
- DAV (data valid) line - This line is driven by a transmitting device to indicate whether data line signals are stable (i.e. valid) and can be accepted safely by devices.

Interface Management Lines

The five interface management lines (ATN, IFC, REN, SRQ, and EOI) manage the flow of information across the interface;

- ATN (Attention) line - Driven TRUE or FALSE to indicate whether the message bytes on the data lines are command or data messages respectively.

- IFC (interface clear) line - Used by the system controller to initialize the bus and become the ***controller in charge (CIC)***.
- REN (remote enable) - Used by the system controller to place devices in the remote or the local programming mode.
- SRQ (service request) - May be driven by any device to asynchronously request the service from the controller.
- EOI (end or identify) - Used for two purposes; by the talker to mark the end of a message string, and by the controller to tell devices to identify their purposes in a parallel poll.

The IEEE 488 bus uses negative logic with standard TTL levels. In order to achieve the high transfer rate for which the IEEE 488 was designed, the physical differences between the devices and the number of devices on the bus are limited. But for a fully loaded system with 15 devices and 15 meter of cable, transfer rates can reach 1.5 Megabits/second, using the ***high-speed GPIB handshake protocol*** (Protopapas D. A., 1988). In terms of compatibility, there are converter boxes that can transparently convert various protocols (such as RS-232, RS-422, SCPI, and centronic parallel) to the IEEE 488 interface in order to achieve interoperability of the devices.

2.7 Analog and Digital Interfaces

Most real world processes produce analog signal which vary continuously, such as change in temperature or pressure. It is not easy to manipulate, compare, calculate or retrieve data accurately using analog technology. However, digital computers can perform this task quickly on an unlimited mass of data and with high precision and speed. Thus, the need to interface analog and digital 'worlds'. Also, the storage requirement (cost per bit) in digital technology is drastically reduced.
Signal conditioning is required to condition the 'real-world' signals so as to meet the required I/O specifications of digital systems. An ADC can then take the conditioned analog signal as input to produce a digital output, thus allowing analog devices to communicate with digital computer. Similarly, the digital computer communicate with analog devices via the DAC.

2.7.1 Need for Signal Conditioning

The signals obtained from transducers and sensors are rarely suitable for direct processing by microcomputers, and in addition to analog-to-digital conversion, some form of preprocessing or signal

conditioning usually required. Signal conditioning may consist of simple amplification or scaling, but often takes the form of shaping and 'cleaning' of signals prior to processing. Hardware alone may be used for this purpose, but often a pure software solution is applicable. Simple signal conditioning may involve the processes outlined below:

- **Amplification;** This is the most common type of conditioning. Low-level signals which might sometimes due to attenuation need to be amplified in order to increase its resolution and reduce the noise. For the highest possible accuracy, the signal should be amplified so that the maximum voltage range of the conditioned signal equals the maximum input range of an ADC.
- **Linearisation;** The output signal from a sensor should be directly proportional to the applied stimuli, resulting in an ideal straight line characteristic. However, most sensors do not posses ideal characteristic and the departure from the straight line is called *'non-linearity'*. If the degree of the non-linearity is small it may be ignored, but where unacceptable error would result, there is a need for compensating for the non-linearity by a process known as *'linearisation'*. This may be achieved by hardware such as op-amp with feedback arranged to provide equal but opposite non-linearity. Alternatively, software routine may be used to provide similar compensation.
- **Isolation;** Isolation of transducer's signal from the computer is necessary for safety purposes. The system being monitored may contain high-voltage transients that could damage the computer. It may be used to achieve a sort of current buffering and impedance matching.
- **Filtering;** Filtering is a process which rejects all unwanted frequencies from the entire spectrum , thus selecting only wanted range of frequencies for further processing

2.7.2 Digital-to-Analogue Converters (DAC)

Basically, digital-to-analogue conversion is the process of taking a digitally coded value (such as binary or BCD) and converting it to a current or a voltage signal that is proportional to the digital value. Figures 2.19(a) and 2.19(b) shows the block diagram of a typical 4-bit DAC and its characteristics respectively. Most DACs use a BCD input code, where 4-bit code are used for each decimal digit. Some of the important characteristics of the DACs are its resolution, accuracy and operating speed .The resolution of the DAC may be defined in the following two ways:

(i) As the number of different analogue output that can be provided by the DAC. For n-bit DAC:

$$Resolution = 2^n \qquad \text{.....(2.6)}$$

(ii) Resolution is also defined as the ratio of a change in output voltage resulting from a change in 1 least significant bit (LSB) at the digital input. This is also refer to as the step size. Hence, for an n-bit DAC;

$$Resolution = \frac{Full\ scale\ output\ voltage\ (V_{FSO})}{2^n - 1} \quad \text{.....(2.7)}$$

So, from Fig. 2.19(a), where $n = 4$ digital inputs, equation. (2.6) implies that the output voltage V_O will have 16 (i.e. 2^4) different output values, that is 0 through 15. The input-output characteristic of figure. 2.19 (b) shows a 4-bit DAC with a full-scale output voltage V_{FSO} is 15 V, i.e. when all digital inputs are one. Thus, the resolution of this DAC according to the equation (2.7) is obtained as:

$$Resolution = \frac{15}{2^4 - 1} = 1\ V/LSB$$

Fig. 2.19(a) Block diagram

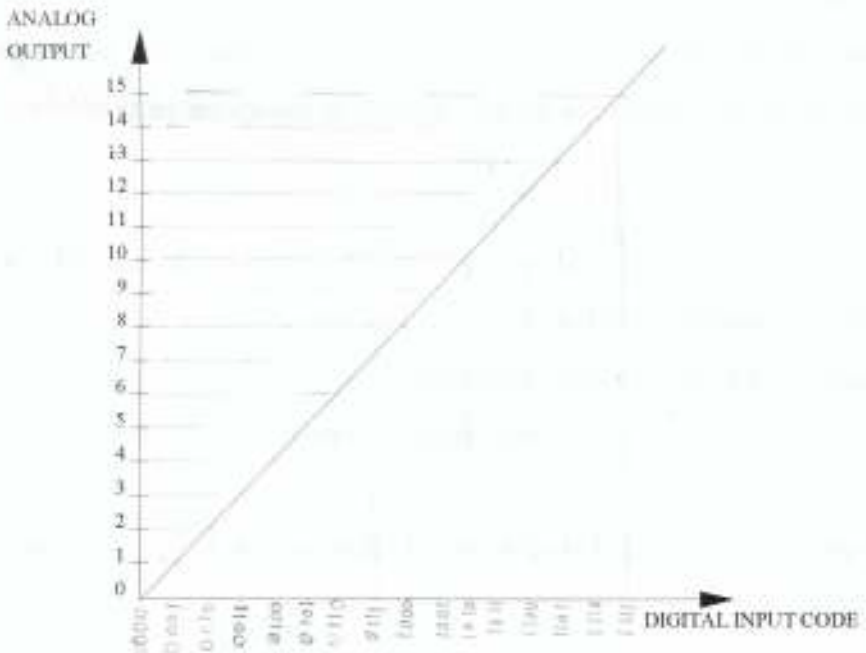


Fig. 2.19(b) Input-output characteristic

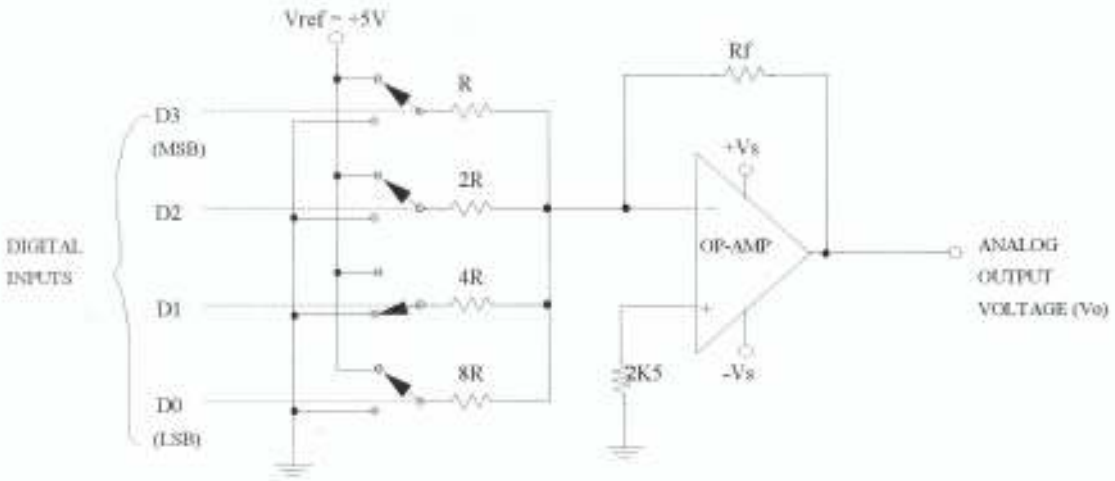


Figure 2.20 4-bit DAC circuit utilizing a binary weighted network

Input-Output (I/O) Equation of the DAC

The transfer function or the I/O equation of the DAC is obtained by multiplying the resolution by the decimal value of the digital code input. i.e.

V_O = *Resolution* X D .....(2.8)

where V_O is the analog output, the resolution is as defined in equation. (2.7), and D is the decimal equivalent of the digitally coded input. There are several methods and circuits for producing the D/A operation. Although, it is not important to be familiar with all the various schemes because they are available in an encapsulated package that do not require any circuit knowledge. Two of the schemes are ;

- The use of op-amp as a summing amplifier, whose output voltage represents a weighted sum of the digital inputs. An example of this type is shown in figure. 2.20 with input set to 1101.
- The use of resistance ladder network commonly called R-2R ladder network, in which case the output may be a current or a voltage output.

The mathematical expression for the amplifier output voltage V_O of the figure DAC is given as :

$V_O = - R_f/R \, (V_{D3} + 1/2V_{D2} + 1/4V_{D1} + 1/8V_{D0})$ .....(2.9)

Each of the digital input may be either a 0 or 5V. Assuming the value of the resistors are chosen such that the resistance ratio R_f/R is equal to 8/5 say, therefore eqn. (2.9) becomes :

$$V_O = -1/5\,(8V_{D3} + 4V_{D2} + 2V_{D1} + V_{DO}) \quad \text{.....(2.10)}$$

As an example when the input code $D_3D_2D_1D_0$ is 0101, i.e. decimal 5, then the analog output voltage will be:

$$V_O = -1/5(0 + 4/5 + 0 + 1/5) = -5\ V$$

Hence, the I/O characteristic of the DAC is similar to that shown in Fig. 2.19(b) and its resolution may be evaluated as:

$$\textit{Resolution} = \frac{\textit{Equivalent output voltage of } D_3D_2D_1D_0 = 11111}{2^4 - 1}$$

$$= \frac{15}{2^4 - 1} = 1\ \text{V/LSB}$$

DAC Specifications

Resolution is one of the important specifications of a DAC and two forms of equations for resolution are as stated in equations (2.6) and (2.7). Other DAC specifications include; accuracy, and the operating speed of the DAC.

Accuracy : The two most common accuracy specifications are the ***relative accuracy*** and the ***differential linearity. Relative accuracy*** is the maximum deviation of the DAC output from its expected (i.e. ideal) value (John B. Peatman, 1981). It is usually expressed as a percentage of the full scale (FS) output. A typical value is 0.01% FS. ***Differential linearity*** is the maximum deviation of the step size from the ideal step size. It is also expressed in % FS.

Operating Speed: DAC operating speed is usually expressed as the settling time, which is the maximum time taken by the output to change from zero to full scale when the digital input code changes from all 0's to all 1's. Typical value range from 1 to 20 μsec.

2.7.3 Analog-to-Digital Converter (ADC)s.

An ADC takes analog input voltage and after a certain period of time produces a digital output code which represents the analog input. Shown in Fig. 2.20(a) is the circuit symbol for a 4-bit ADC. Most ADC utilize a DAC as part of their circuitry. The general block diagram of a class of ADCs is illustrated in Fig. 2.21(b). It comprises of four sections: a DAC, a voltage comparator, a register, and a logic control unit. The logic control unit contains the circuitry that generates the proper sequence of operation for the *start* command pulse, which initiates the conversion process. The comparator is a special unit with two analog inputs and a digital output that switches states depending upon which of the analog input is greater. There are different types of ADCs in this class whose variations is only in the logic control unit. Examples are the digital-ramp and the successive approximation ADCs.

The digital-ramp ADC uses a binary counter and allow the clock to increment the counter one step at a time until $V'_a \geq V_a$. The resolution of the ADC depends upon that of the DAC utilized, and have a similar expression as that of the DAC. Hence, for an *n*-bit ADC:

$$Resolution = 2^n \quad \text{.....(2.11)}$$

ADC resolution may be defined as the ratio of a change in the value of the input voltage V_A needed to change the digital output by 1 LSB. i.e. for an *n*-bit ADC ;

$$Resolution = \frac{V_{FSI}}{2^n - 1} \quad \text{(Volts / LSB)} \quad \text{.....(2.12)}$$

where, V_{FSI} represents the full-scale input voltage. Therefore, the input-output equation of the ADC may be expressed as:

$$Digital\ output\ code\ (V_O) = Binary\ equivalent\ of\ D \quad \text{.....(2.13)}$$

where;

$$D = \frac{V_a}{Resolution} \quad \text{.....(2.14)}$$

The accuracy of the ADC is not dependent on its resolution, but on the accuracy of the circuit components utilized, such as, its comparator, DAC, resistors, and so on. The conversion time of the ADC depends upon the input voltage V_a, and the clock period, with a maximum of 2^n clock periods for an *n*-bit ADC.

The most widely used ADC is the successive approximation type, because of its much shorter conversion time, which is fixed and independent of the analog input. The symbol of 4-bit ADC is shown in figure 2.21(a), while the block diagram in figure 2.21(b) depicts that of a successive approximation type. It uses a register control unit, where as a digital ramp ADC will use a counter unit. Its logic is successive setting of the output bits to 1 or leaving it at 0, and comparing it with the analog input V_a, beginning with the MSB to the LSB.

Fig. 2.21(a) Symbol for a 4-bit ADC

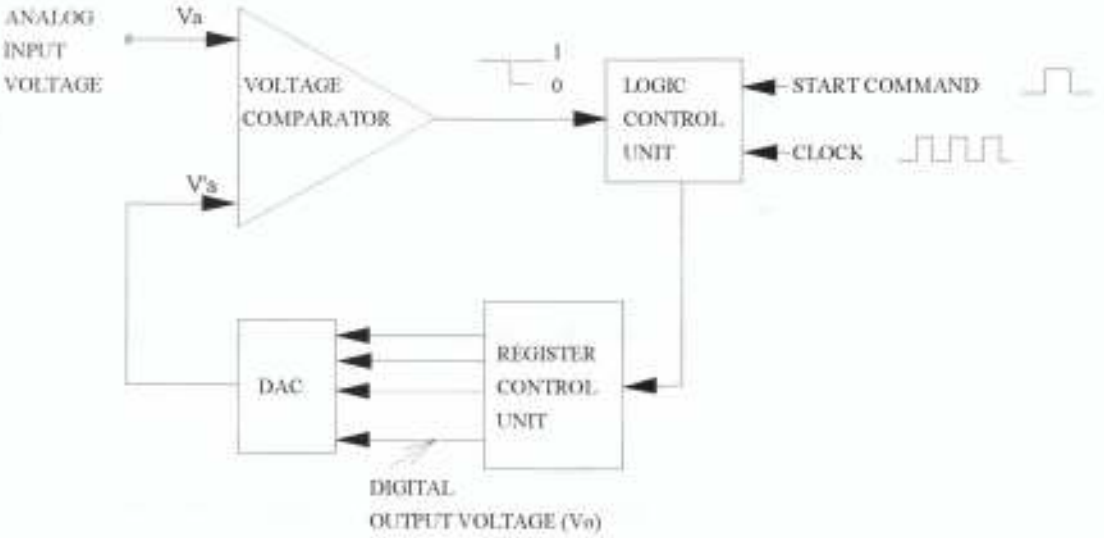


Fig. 2.21(b) Building blocks of an ADC

Figure 2.21 Analog-to-Digital Converter

The conversion operation logic of a successive approximation ADC requires n steps regardless of the value of V_a. Each step requires a clock period.

Quantisation Error of an ADC

The I/O characteristic of a 4-bit ADC in fig. 2.22 shows that the analog input 1.0V ±0.5V is converted to the digital code 0001. Hence, there is an unavoidable uncertainty about the exact value of the analog input. This uncertainty is specified as the quantisation error and has a value of ± 1/2LSB. The quantisation error can be observed by continuously sampling a time-varying analog signal with an ADC, converting it back to analog with a DAC, and tracking the difference between the two (Sergio Franco, 1988). The distortions due to the quantisation errors is often referred to as quantisation noise due to the random nature of these errors. The quantisation noise power is uniformly distributed throughout the range of frequencies spanned by the analog signal. It cannot therefore be removed by filtering the reconstructed analog signal. It has the properties of an added white noise component (Harold S. Stone, 1982).

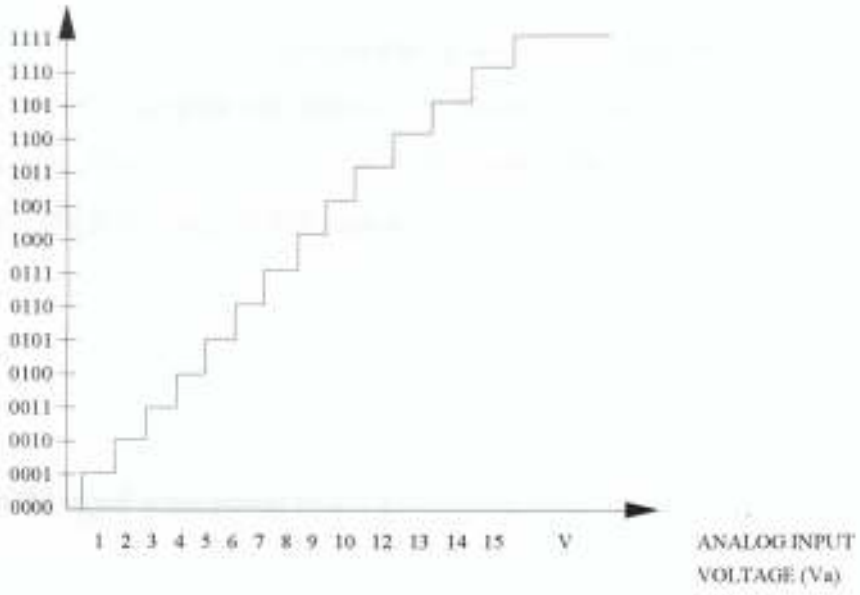


Figure 2.22 Input output characteristic of a 4-bit ADC

If an analog signal has a value V at an instance of sampling and the nearest quantised level is V_j, quantised error is due to $(V - V_j)$. In this process, anytime a sample occurs within the range $V_j - (\Delta V_j)/2$ to $V_j + (\Delta V_j)/2$, the code word corresponding to the j^{th} level is assigned to it. Associating a mean square error voltage $(V - V_j)^2$ with the j^{th} level, the mean square value of the quantisation noise associated with the continuous process may be computed as the sum of the mean square error voltages associated with each level. If the level separator are small compared with the total signal amplitude

range, then the relationship between the quantisation noise and the ΔV_j (becomes ΔV a constant, for linear quantisation) is as follows:

$$N_Q^2 = \frac{\Delta V^2}{12} \qquad \text{.....(2.15)}$$

where;

$$N_Q = \frac{\Delta V}{\sqrt{12}}$$

is the RMS value of the quantisation noise and ΔV is the separation of the quantised levels as written in equation (2.16) and n is the number bits of the ADC.

$$\Delta V = \frac{\text{full scale analog range}(V_{FSR})}{2^n} \qquad \text{.....(2.16)}$$

The expression for N_Q may be used to indicate the signal -to-noise ratio (SNR) to be expected when a sinusoidal analog signal is sampled and digitized. Considering a full scale sine wave whose peak-to-peak range spans the whole of the ADC used in the system, the peak value of such a full scale sine wave is $(2^n \Delta V)/_2$. The RMS value of full-scale sine wave is stated in equation (2.17).

$$(V_{RMS})_{\text{full-scale}} = \frac{(2^n \Delta V)/_2}{\sqrt{2}} = \frac{(2^{n-1} \Delta V)}{\sqrt{2}} \qquad \text{.....(2.17)}$$

Thus, the ratio of the signal noise power to the quantisation noise power is :

$$SNR = \frac{(V_{RMS})_{\text{full-scale}}}{N_Q} = \frac{(2^{n-1} \Delta V)/_{\sqrt{2}}}{\Delta V/_{\sqrt{12}}}$$

$$= \sqrt{(1.5)}\, 2^n \qquad \text{.....(2.18)}$$

Expressing SNR in decibel gives:

$$SNR_{dB} = 20 \log 10[\sqrt{(1.5)}\, 2^n] = 6.02n + 1.76 \text{ dB} \qquad \text{.....(2.19)}$$

The above analysis clearly shows that the SNR can be increased (i.e. reduction of the quantisation error) by increasing the number of bits n of the ADC, which will consequently reduce the step size, ΔV. Alternatively, the quantisation noise may be reduced by increasing the sampling frequency of the ADC. A 16-bit linear ADC is recommended as the ideal for the digitization of very high fidelity audio signal (Clayton G. B., 1982).

CHAPTER THREE

SYSTEM IMPLEMENTATION OF THE INTERFACE CARD

3.0 Introduction

This section covers the theory and circuit implementation of the integrated circuits (ICs) utilized in the interface card design. The circuits include: a tristate buffer LS231; the pre-amplifier and the power amplifier circuits utilizing the LM741 and LM386 ICs respectively. The microprocessor compatible ADC and DAC ICs, AD0804LCN and DAC0807LCN, are employed in the recording and the playback module respectively.

Two experimental interface circuits are presented in this chapter to demonstrate the data input-output handling capability of the computer via the parallel port. Also presented is the step-by-step analysis and design procedure of the three modules that make up the audio card; namely the recording module, the playback module and the regulated power supply module.

3.1 Experimental Interface Circuits Design & Testing

Two interface circuit experiments were carried out. The first is a light-emitting diodes (LEDs) interface circuit that was set up to study how the three ports that makes the micro-computer parallel port normally handle the transfer of data bits. The second is a seven-segment LED display interface circuit to further examine how to send data bits through the data port for the controlling of a simple electronic device.

3.1.1 LED Interface Circuit

Figure 3. 0(a).shows the LED interface circuit where '1' or '0' bit is indicated by the 'ON' and 'OFF' state of the LEDs respectively. An LED is connected via a current limiting resistor to each of the data port and control port signal lines. The five lower bit of the data port are looped back to the status port, in order to supply the signal to be received by the status lines via these data lines.

LEDs like the common diodes or P-N junction semiconductor devices has the special property of emitting light. This occurs under forward biased condition when electron and hole recombine. LED forward biased voltage (V_f) are typically about 2.0 Volts, with forward current (I_f) that is often less than 15mA . The required current limiting resistor may be calculated by considering the forward biased circuit shown in figure 3.0(b) with current requirement, I_f of about 10mA, say. Application of Kirchoff's Voltage Law (KVL) yields:

$$V_{CC} - V_f - I_f R_C = 0$$

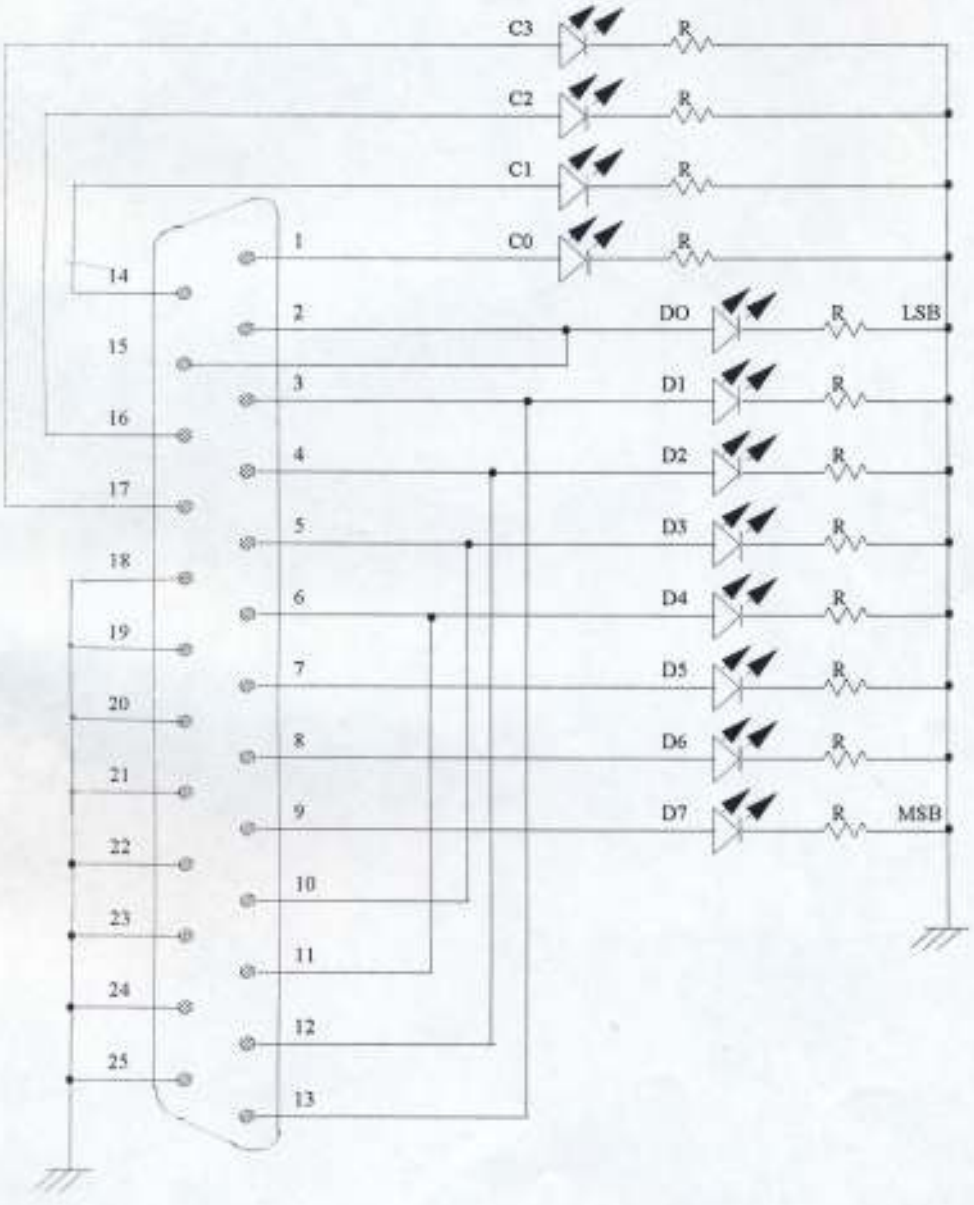


Figure 3.0(a) LED Interface Circuit

$$R_C = \frac{V_{CC} - V_f}{I_f}$$

$$R_C = \frac{5.0 - 2.0}{10 \times 10^{-3}} \quad R_C = 300\ \Omega$$ (A standard resistor of 330 Ω was chosen)

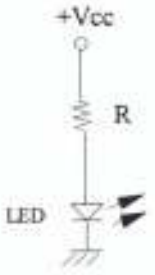


Figure 3.0(b) Forward biased LED Circuit

Sending Out Data Bits

The binary equivalent of the decimal numbers between 0 to 255 may be sent via the 8 - bit(1 Byte) wide data port, which will subsequently activate the respective LEDs. The range is 0 to 15 decimal when the bits are sent through the 4 - bit(1 nibble) - wide control port. It should be noted that three control port bits(C_0*, C_1* ,C_3*) which are starred uses inverted logic. This infers that a 'HIGH' represents a '0' , while a 'LOW' represents a '1'. Hence, these are taken into consideration when sending data bits via the control port. For example when the decimal number 1, with binary equivalent of 0001, is to be sent, the actual binary code generated and sent by the program is 1010, or 'A' in hexadecimal. Hence the first, third and fourth bits are inverted before been sent. To display decimal 2(i.e. 0010 binary) means that 1001 or '9' in hexadecimal will actually be sent, and so on.

Receiving Data Bits

The only input lines of the parallel port is the status port lines. Its address (is 379 in hexadecimal) must be specified in order to receive data via these lines. In fig. 3.0(a), pins 11, 10, 12, 13, and 15 which corresponds to the status lines are looped back to the five LSBs of the data port pins 6, 5, 4, 3, and 2 respectively. This enables the bits to be received via the status port while it is being sent through the data port lines. Both the decimal value and the binary equivalent of the received number will be displayed on the visual display unit(VDU) of the computer. The program for the flowchart in figure 3.1 is coded in C++ and the source code is in Appendix A2.

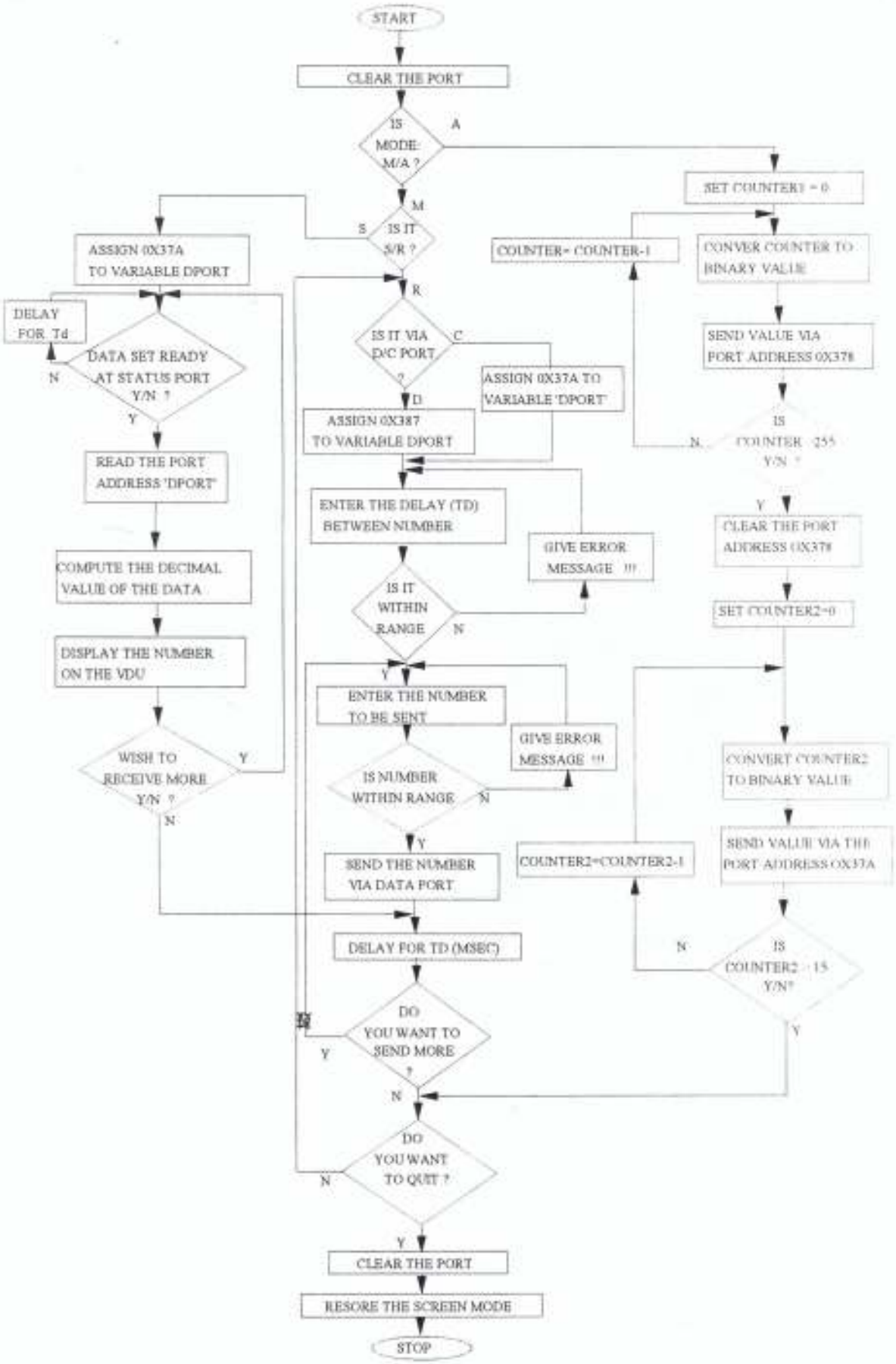


Figure 3.1 **Flowchart for the LED Interface Driver**

3.1.2 Seven-Segment Display Interface Circuit.

Display modules serves to output numerical, alphanumeric and even graphical information. The simplest type of them is the seven-segment character fonts that can be employed for the display of numerical symbols. By activating appropriate set of segments, it allows formation of any numerical symbol and a small number of alphabetic ones.

Other display modules include 16-segment character font, 5 X 7 dot matrix for numeric and alphabetic characters. The factor to be considered before selecting a display module type includes; the information content, the character size, the power consumption, the colour, the brightness, the overall physical size, the interface characteristics, cost and so on. Figure 3.2 shows the 7- segment character font interface circuit while Table 4.0 is the look-up table for the display of hexadecimal characters **0** to **F.**

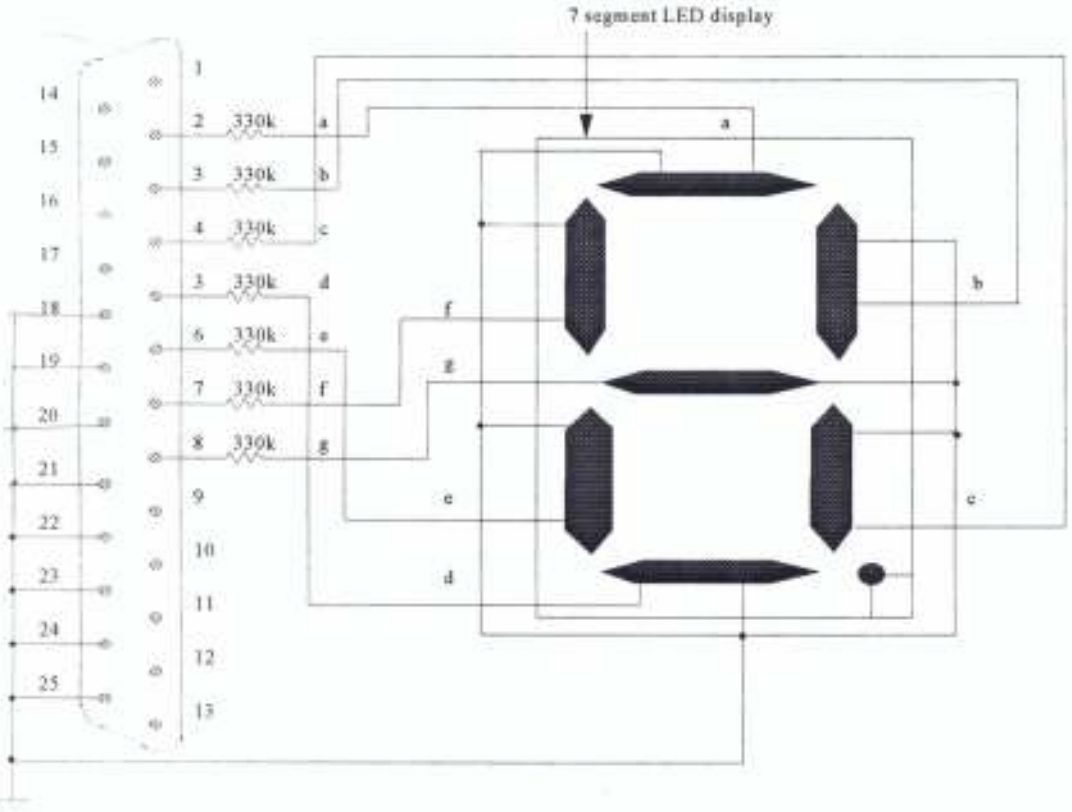


Figure 3.2 7-Segment Display Interface Circuit.

The LED segments are labeled *a* through *g*, so that appropriate binary code that will activate the respective hexadecimal characters could easily be formed. As an example, in order to display the

character **0**, the segments ***a, b, c, d, e*** and ***f*** must be ON. Hence, taking the LSB as g and the seventh bit as **a** , this is equivalent to the binary number 1111110 or the hexadecimal number 7E. Therefore by sending out the hexadecimal number 7E via the 8 - bit data port, the decimal character **0** will be displayed on the 7 - segment module.

The flowchart for the 7-segment drivers is shown in figure 3.3. Each of the software drivers for the interface experiments has the option to operate in either automatic or in manual mode. The automatic mode will automatically display the binary equivalent of 0 to 255 (via data port) or 0 to 15 (via control) on the LED interface, or hexadecimal 0 to F on the 7-segment display. But in the manual mode, the user has the option of entering the code to be displayed. The source code for the flowchart figure 3.3 can be found in appendix A2.

Table 4.0: Look-up Table for the 7-Segment Decoder

Decimal Number	Hex. Number	Binary Code a b a d e f g	Hex Code Equivalent
0	0	1 1 1 1 1 1 0	7 E
1	1	0 1 1 0 0 0 0	3 0
2	2	1 1 0 1 1 0 1	6 D
3	3	1 1 1 1 0 0 1	7 9
4	4	0 1 1 0 0 1 1	3 3
5	5	1 0 1 1 0 1 1	5 B
6	6	1 0 1 1 1 1 1	5 F
7	7	1 1 1 0 0 0 0	7 0
8	8	1 1 1 1 1 1 1	7 F
9	9	1 1 1 0 0 1 1	7 3
10	A	1 1 1 0 1 1 1	7 7
11	B	0 0 1 1 1 1 1	1 F
12	C	1 0 0 1 1 1 0	4 E
13	D	0 1 1 1 1 0 1	3 D
14	E	1 0 0 1 1 1 1	4 F
15	F	1 0 0 0 1 1 1	4 7

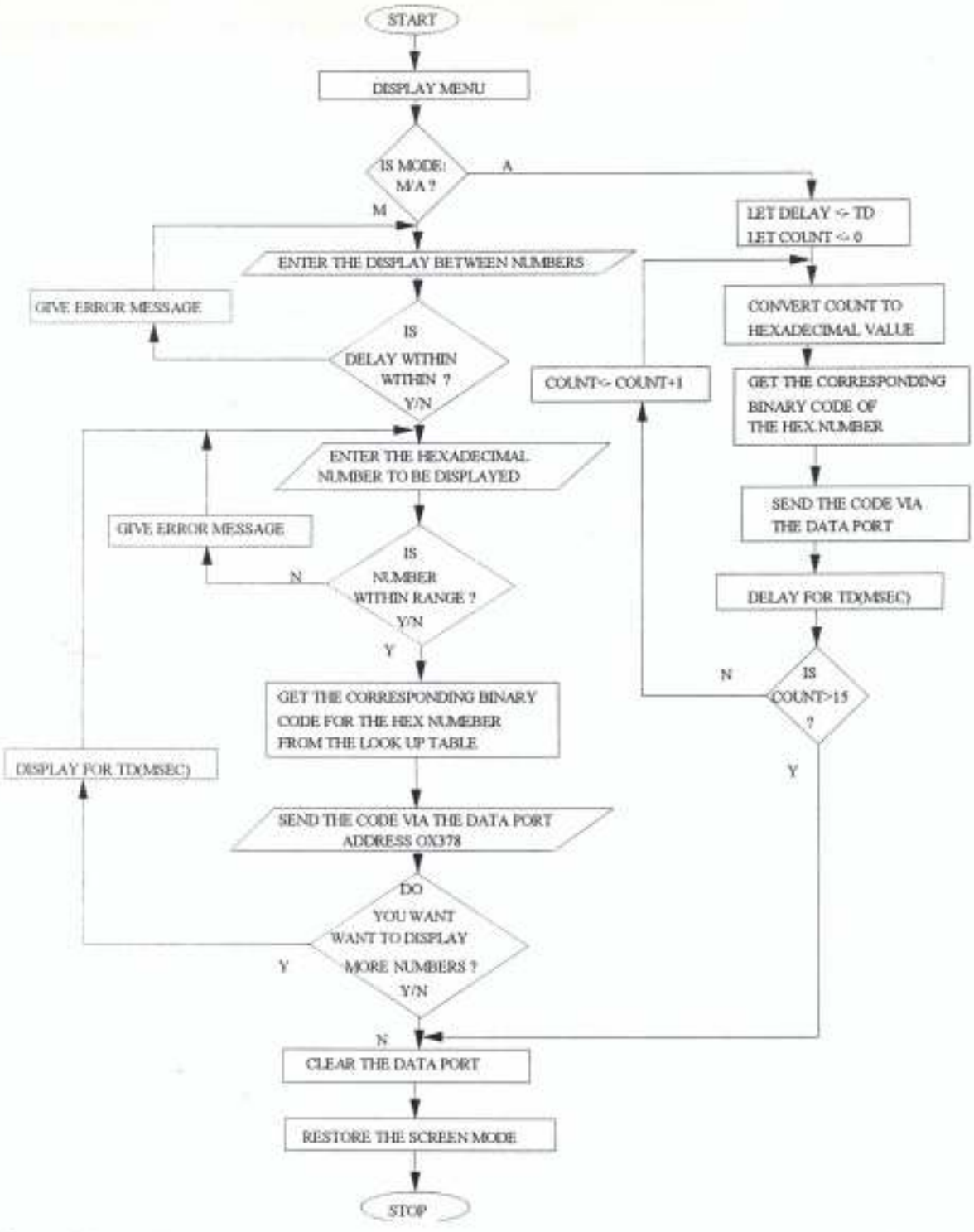


Figure 3.3 Flowchart to Activate any Hexadecimal Character on a 7-Segment Display

3.2 Digital Audio Card Hardware Developments.

Figure 3.4 is the block diagram of the digital audio circuit card. It performs the conversion of the analog audio signal into digital words, that can be stored in the computer memory and receives digital

control commands via the computer control port to synchronize the conversion process. The tristate switches are used to multiplexed the 8-bit output of the A/D converter to the four of the status port lines. The card also converts the digitally stored signal back to analog form through the D/A converter, amplify it via the op-amp before sending it via the speaker.

The stabilized voltage outputs utilized on the circuit card are fed from the regulated power supply unit. The four stabilized voltage outputs are: -5V, +5V, -15V and +15V. The port adapter used is the 36-pin Centronic male connector on the board, so that a detachable standard parallel printer cable, type MXI-1, could be used to connect the board to the computer port. The reverse conversion, i.e. the reconstruction of the digitally stored voltage signal in order to produce the equivalent analog signal, is also performed by this card.

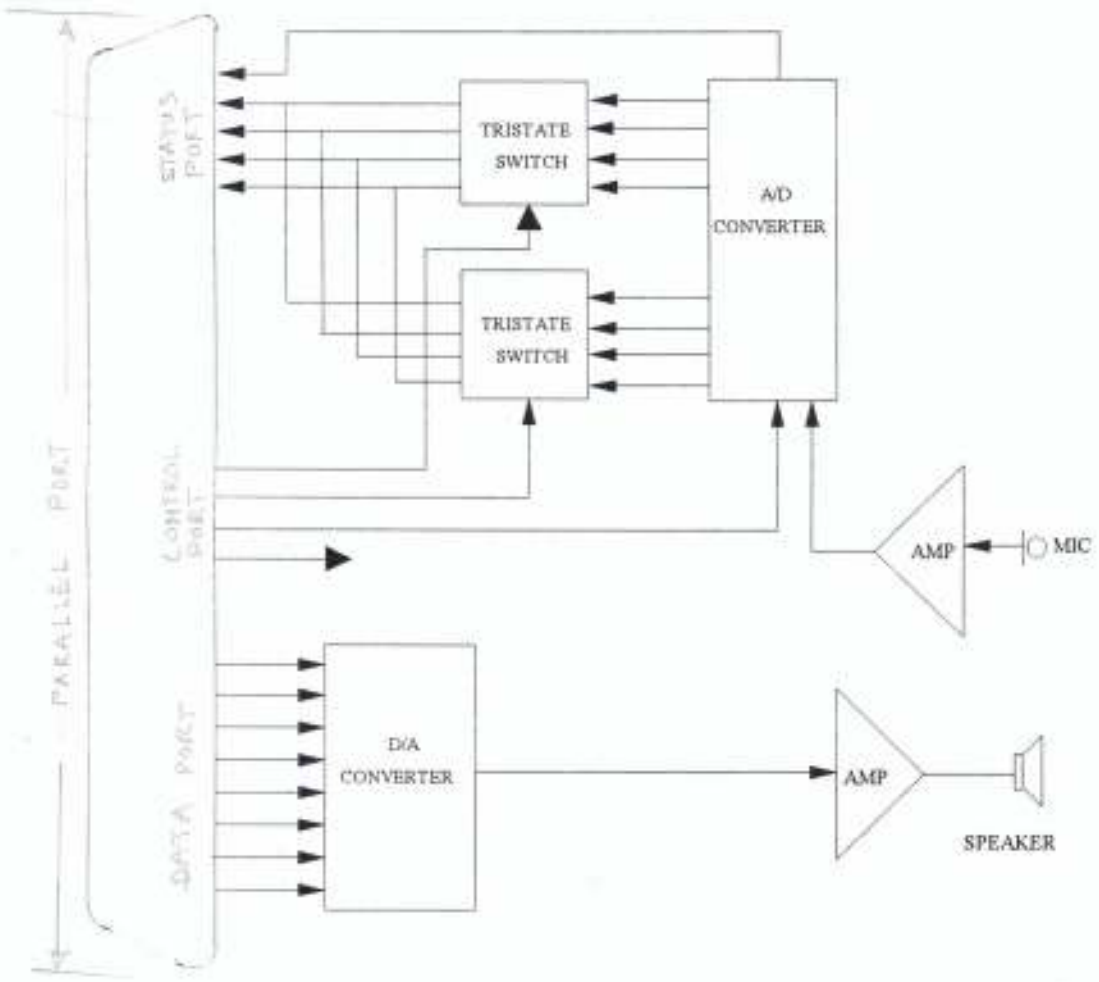


Figure 3.4 Block Diagram of the Digital Audio Card

3.2.1 Recording Module Circuit

The recording module circuit is illustrated in figure 3.5. It is an 8-bit, microprocessor controlled ADC circuit incorporating the ADC0804LCN IC. The AD0804LCN is a 20-pin IC with an on-chip clock generator and an external voltage reference. The IC is a CMOS 8-bit successive approximation A/D converter that uses a modified potentiometric ladder and it is designed to operate with the microcomputer compatibility, via its tristate outputs.

The A/D converter output will appear to the microprocessor as memory locations or I/O ports. The IC is wired for output range of -5.0V to +5.0V, although the actual output voltage range of the ADC circuit will be from -5.0V to +4.961V (i.e. the full-scale output minus 1 LSB).

Functional Description of the Circuit.

The ADC08 series operates on successive approximation principles where, analogue switches are closed sequentially by successive approximation logic until the analogue differential input voltage [$V_{in(+)}$ - $V_{in(-)}$] matches that derived from a tapped resistor string across the reference voltage. The MSB is tested first and after eight comparisons, an 8-bit binary code is transferred to the output latch.

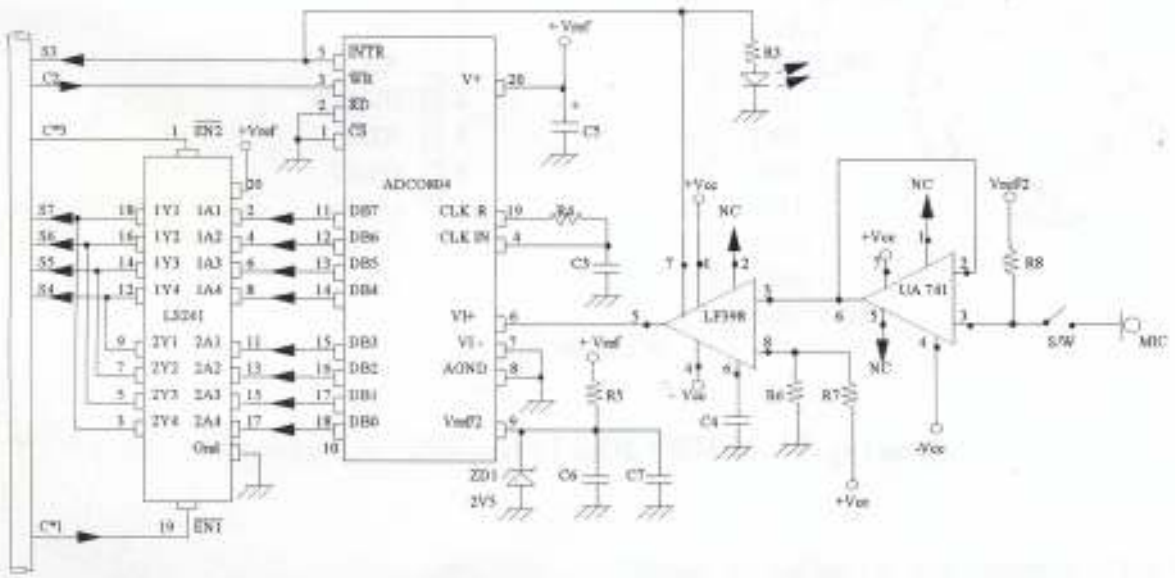


Figure 3.5 Recording module circuit

The digital logic terminals are as follows:

- the digital control inputs **CS**, **RD** and **WR**;
- the clock inputs **CLK IN** and **CLK R**;
- the interrupt output terminal **INTR**;
- the digital ground terminal **DGND**
- and the eight digital output terminals D_{B0} - D_{B7}.

The digital control inputs, **CS**, **RD** and **WR**, correspond to the ADC0804 pins 1, 2 and 3 respectively. They are all active low terminals and operate with the standard TTL voltage logic levels. The **WR** and **RD** input terminals are equivalent to the A/D converter start and output enable control terminals. The **CS** input is wired to the digital ground (i.e. low), so that the IC is permanently selected. The standard A/D start function is obtained by applying an active-low pulse to the **WR** input (pin 3) from the microprocessor control port line C_2.

The output enable function is achieved by an active-low pulse at the **RD** input (pin 2). The pin-out of the ADC0804 IC is shown in figure 3.6.

Figure 3.6 Pin-out of ADC0804 IC (Top view).

The normal operation of the circuit proceed as follows; on the high to low transition of the **WR**, the internal successive approximation register (SAR) latches. Consequently, the shift register stages are reset, and the **INTR** output terminal will be set to high. As long as both the **CS** and **WR** remain low,

the ADC will remain in the reset state. Conversion will start after at least one of the inputs (**WR** in this case) makes a low-to-high transition. The **INTR** will make a low-to-high transition, after the requisite number of clock pulses to complete the conversion. Hence, the **INTR** pin is used to signal the end of conversion via the status port line S_3 for storage purpose by the microprocessor. A **RD** operation (with **CS** low) will clear the **INTR** line again, and the three-state output latches will be enabled to make the 8-bit digital word transparent to the LS241 tristate IC. The output data latch is not updated if the conversion in progress is not complete. So, the data from the previous latch will remain in the three-state output latch during this period.

Digital Outputs

Pins 11 through 18 (D_{B0} - D_{B7}) of the ADC0804 are designated the digital data output terminals. These terminals are connected via two quad-tristate buffer (i.e. the tristate octal buffer LS241SN IC) to the upper four bits of the microprocessor status lines. The tristate enable signals (E_{N1} and E_{N2}) are controlled by the software driver via the microprocessor control lines C^*_3 and C^*_1 respectively. This connection is done so that the 8-bit output data presented by the A/D converter to the microprocessor can be determined via four lines of the status port.

The process of getting the data from the A/D converter is as follows; firstly C^*_1 is taken high and C^*_3 is taken low (i.e E_{N1} E_{N2} = 01), so that the 4-bit status port data inputs will correspond to the lower four bits of the A/D converter. Next, C^*_1 is taken low with a high on C^*_3 (i.e E_{N1} E_{N2} = 10) in order to store the upper four bits of the converter output. The situation whereby the two enable inputs, (E_{N1} and E_{N2}), are simultaneously taken low is prohibited. Consequently, this situation is disallowed according to the program logic. The normal resting position of the enable inputs is the presentation of logic 1 to each of the inputs (i.e E_{N1} E_{N2} = 11).

The digital interrupt output terminal will make a low to high transition after the requisite number of clock pulses to complete the conversion. This signal is used to interrupt the microprocessor via one of the status line, S_3. The processor will normally poll this line to confirm that the conversion is complete before accepting the data. A green LED, D1, is connected through the current limiting resistor R8 from the **INTR** terminal to the ground to indicate the current state of this terminal. The clock input, **CLK IN**

and **CLK R** (pins 4 and 9 respectively) of the ADC utilize an internal Schmitt trigger circuit. The clock is derived from an external RC network, R_4 C_3, added to provide the self-clocking of the A/D converter as depicted in figure 3.7.

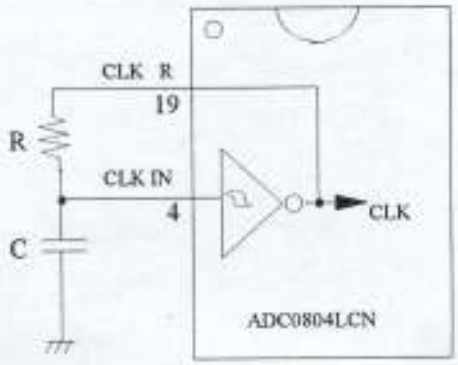


Figure 3.7 Self-clocking configuration of ADC0804

Where the clock frequency , f_{CLK}, is given as;

$$f_{CLK} \equiv \frac{1}{1.1\,R\,C} \qquad \text{.....} \qquad (3.0)$$

But in the circuit in figure 3.5, the resulting clock frequency , f_{CLK}, is obtained from eqn. 3.0 as;

$$f_{CLK} \equiv \frac{1}{1.1\,R_4\,C_3} \qquad \text{.....} \qquad (3.1)$$

Since $R_4 = 10\Omega$ and $C_3 = 150\mu F$, therefore

$$f_{CLK} \equiv \frac{1}{(1.1)\,(10 \times 10^{3})\,(150 \times 10^{-9})}$$

$$f_{CLK} \equiv 606060\ \text{Hz} \equiv 606.060\ \text{KHz}$$

In the alternative, the clock signal of the A/D converter may be derived from external source, such as the CPU clock.

Conversion time of the ADC

The conversion time is a function of the clock frequency f_{CLK} and the number of bits n of the ADC. In an n-bit conversion, the value of the n^{th} bit is found at the occurrence of the n^{th} clock pulse after the start conversion bit. This value is not fixed into the n^{th} register until the occurrence of the $(n+1)^{th}$ clock pulse. An n-bit conversion is thus completed in (n + 1) clock periods (Clayton G. B., 1982). The expression for the conversion time T_C is given as follows:

$$T_C = (n+1)\,T_{CLK} = \frac{n+1}{f_{CLK}} \qquad (3.2)$$

where: $f_{CLK} = \frac{1}{T_{CLK}}$ ...is the clock period

The conversion time of the 8-bit ADC used i.e. thus obtained as follows:

$$T_C = \frac{8+1}{606060\ \text{Hz}} = 14.85\ \mu\text{Sec}$$

Analog Inputs

The analog inputs are the differential voltage inputs, the analog ground and the reference voltage inputs. The ADC uses the differential voltage inputs, $V_{IN(+)}$ and $V_{IN(-)}$, which correspond to pins 6 and 7 of the IC. This is advantageous because of its common mode noise reduction. The leads of these analog inputs are kept as short as possible, in order to minimize stray signal pickup and electromagnetic interference (EMI), since EMI and undesired signal coupling to these inputs can cause system error. The differential voltage input are wired for an analog voltage input range of -5.0V to +5.0V. An alternative configuration for handling an analog input voltage range of ±10V is illustrated in figure 3.8.

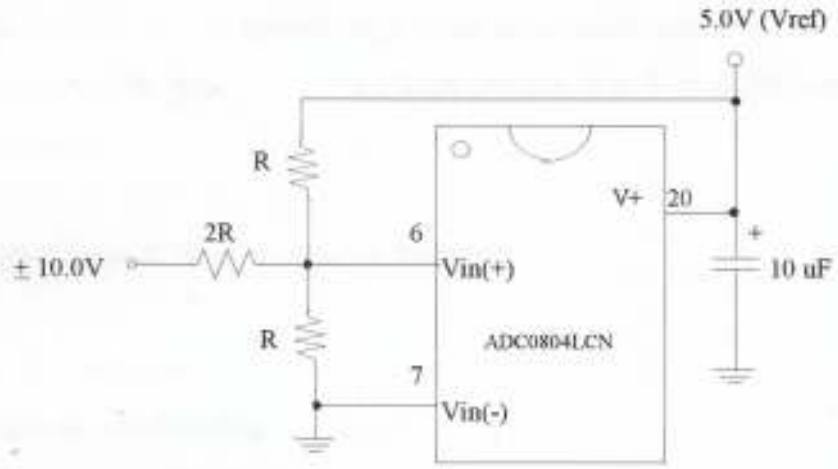


Figure 3.8 ADC0804 input terminal configuration for handling ±10V analogue voltage range

3.2.4 Playback Module Circuit

The playback module circuitry does the conversion of the stored binary data words back to analogue signal, filters and output the signal via the 8-Ω speaker. The playback frequency has been fixed in the software to approximately 8 kHz by using a delay time of 125 μsec between the sending of data words. Figure 3.9 shows the circuit diagram of the playback module.

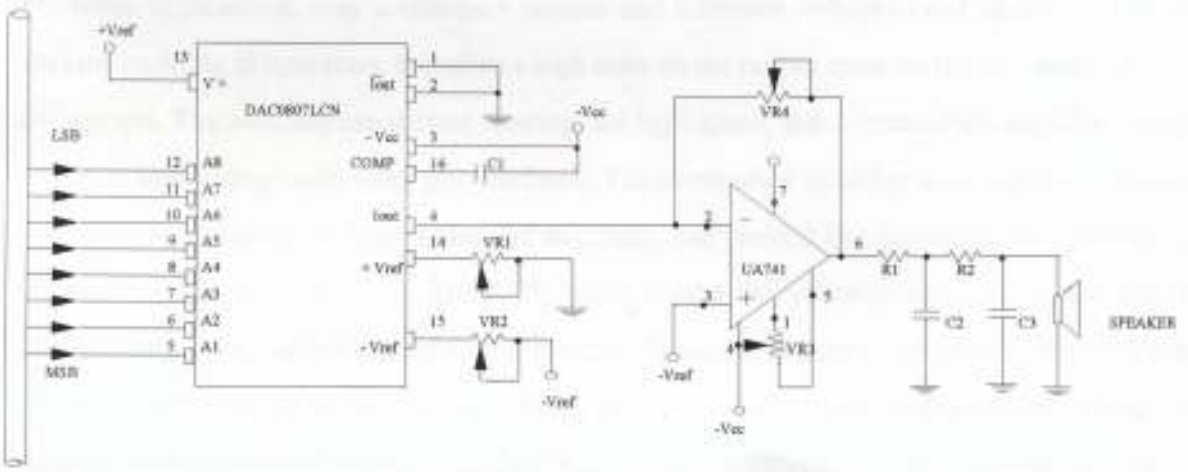


Figure 3.9 Playback module circuit

The playback module circuit uses a DAC08 series IC (DAC0807ICN), which is a series of 8-bit monolithic D/A converters that provides high speed performance with low cost. Conversion principle of the IC is such that the output current is a linear product of an 8-bit digital data word and an analog reference voltage.

Other application areas of the IC include the following:

- Tracking A/D converters
- Audio digitizing and decoding
- programmable power supplies
- A/D division
- Digital addition and subtraction
- Speech compression and expansion
- stepping motor drive
- Modems
- Servo motor, pen drivers and host of others.

The pin-out and the internal architecture of the DAC0807 IC are as shown in figure 3.10(a) and 3.10(b) respectively. The IC consist of reference current amplifier , an R-2R ladder, and 8 high-speed binary weighted electronic current switches.

For many applications, only a reference resistor and reference voltages need be added. The switches are non-inverting in operation, therefore a high state on the output turns on the specified output current component. The switches use current steering for high speed, and a termination amplifier consisting of an active load voltage with unity gain feedback. The termination amplifier holds parasitic capacitance of the ladder at a constant voltage during the switching, and provide low impedance termination of equal voltage for all legs of the ladder. The R-2R ladder divides the reference amplifier current into binarily-related components, which are fed to the switches. It should be noted that there is always a remainder current, which is equal to the least significant bit (LSB) that is shunt to the ground. Hence, the full-scale or maximum output current (I_f) is given as 255/256 of the reference amplifier current. i.e;

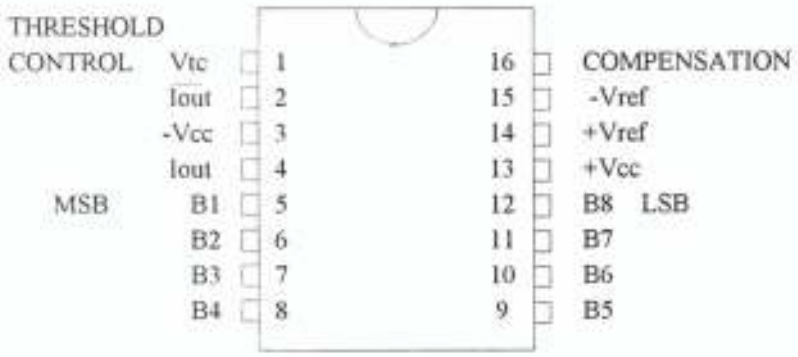


Figure 3.10(a) Pin-out of DAC0807 IC

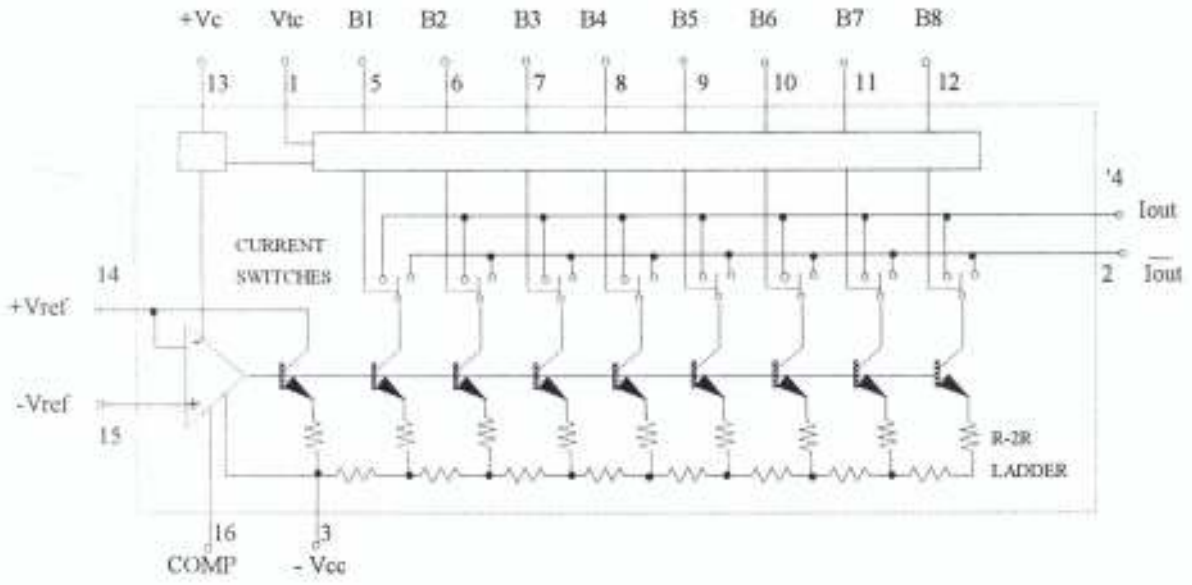


Figure 3.10(b) The Internal architecture of DAC0807 IC

$$I_{FS} = \frac{V_{REF.}}{R_2} \times \frac{255}{256}$$

Normally, the full-scale output current I_{FS} is set to equal to 1.992mA with reference current at 2.000mA. The currents I_{OUT} and I_{OUT} are complimentary. In other words, when I_{OUT} is 1.992mA, I_{OUT} will be 0.000mA. But, when I_{OUT} is 0.000mA, I_{OUT} will be 1.992mA. The relationship between this current and the I_{FS} is such that:

$$I_{FS} = I_{OUT} + \overline{I_{OUT}}$$

The (741) op-amp stage at the output of the ADC converts the output current into proportional voltage. The op-amp gives a low impedance output and the output swing is fixed to be ± 5.0V, with the use of resistor V_{R4}. For all the input codes where the MSB is 0, the output voltage will be negative. But, whenever the digital input has the MSB of 1 (from 10000000 to 11111111), the analog output becomes positive. This type of coding is referred to as offset binary. Using a D/A converter in this way enables true alternating signals to be generated at the analog output.

Functional Description

The reference amplifier input current must always flow into pin 4 regardless of the setup method or reference supply polarity. VR_2 is tied to a negative voltage corresponding to the minimum input level , due to the bipolar reference signals. The compensation capacitor C1 is tied to $-V_{CC}$, in order to increase the negative supply rejection. The negative reference voltage used is apply to the VR_2 while pin 14 grounded through VR_1. A high input impedance is the main advantage of this method. The negative voltage (-5.0V in this case) must be at least 3.0V above the - V_{CC} supply. Bipolar input signals may otherwise be handled by connecting VR_1 to a positive reference voltage equals to the peak positive input level at pin15.

Output Voltage Range

The output at pin 4 must always be at least 4.0V more positive that the voltage of the negative supply (pin 3), when the reference current is 2mA or less. It must also be at least 8.0V more positive than the negative supply in a situation whereby the reference current is within the range 2mA to 4mA. This becomes necessary to avoid saturation of the output transistors, which could cause serious degradation of accuracy. The offset null circuit utilizes the variable resistor V_{R3}, for adjusting the voltage level, so that zero output voltage will be obtained when all bits of the digital input are at logic 0.

The DAC 0807 is designed to operate with TTL-type inputs. However, interfacing with the other type of logic is possible by changing the voltage level applied to the threshold control, V_{LC}, pin 1. As mentioned earlier, the DAC0807 contains eight binary-weighted current switches, which are controlled by the TTl/CMOS logic signals applied to the data inputs A1 - A8. The output signal on pin 4 is a current which depend on the data word applied to the A1-A8 inputs. Since, there are eight data inputs,

this current can have any one of the 256 (i.e. 2^8) values. The equivalent voltage for the current would be obtained at the output of the current-to-voltage converter.

The DAC0807 causes a positive current I_o to flow into pin 4 as indicated in figure 3.10(b) and the op-amp output will feed back this current to the DAC0807 IC output via VR_4. In order to supply this positive current the output of the op-amp must be at a more positive voltage relative to its inverting input. Since the non-inverting input of the op-amp is connected to -5V, therefore, the op-amp will function so as to hold its inverting input at -5V also. This is due to the negative feedback supply through VR_4 to the inverting input. The current Io depends on the data words at the inputs, so that the output voltage V_o may be computed as follows:

$$V_o = \frac{V_{REF}}{R_{REF}} \cdot R_F \left(\frac{A_1}{2} + \frac{A_2}{4} + \frac{A_3}{8} + \frac{A_4}{16} + \frac{A_5}{32} + \frac{A_6}{64} + \frac{A_7}{128} + \frac{A_8}{256} \right)$$

$$V_o = \frac{V_{REF}}{R_{REF}} \cdot R_F \sum_{i=1}^{8} \left(A_i / 2_i \right)$$

Where A_1 is the MSB and A_8 is the LSB and each of A_1 to A_8 can assume a value of 1 or 0, depending on the logic level of each bit. For instance, if an input level of 00000000 is applied to the inputs, then the D/A converter output current will equally be zero. Hence, there will be no current through the feedback resistor. The output of the op-amp will then be at the same voltage level of -5V as its inverting input. But, as the inputs increases from 00000000 to 11111111, the voltage at the output of the op-amp will step from -5V to +4.961V. Although, the circuit is described as a ± 5V D/A converter, but the actual full-scale output voltage is +4.961V, rather than +5.0V. The reason is that 0.0V counts as one of the 256 levels. Hence, the full-scale output voltage for any D/A converter will always be 1 LSB less than the rated full-scale value, when the circuit is properly adjusted. The output equations of the A/D converter circuit may written as:

$$I_o = I_{REF} \sum_{i=1}^{8} \left[A_i / 2^i \right]$$

8

$$I_o = 2\,mA \sum_{i=1}^{8} [A_i / 2^i]$$

$$I_o = \sum_{i=1}^{8} [A_i / 2^{(i-1)}] \quad mA$$

But,

$$I_o = \frac{V_o - (-5)}{R_{REF}}$$

So that,

$V_o = I_o R_F - 5$ (V), where VR_4 is 5.0KΩ.

$V_o = 5(I_o - 1)$ (V),

Examples of expected analog outputs that will be produced for a few input data words are as follows:

For 00000000 input data word: $I_o = \sum_{i=1}^{8} [A_i / 2^{(i-1)}] = 0.0$ mA

$V_o = 5(0.0 - 1.0) = -5.0V$

For 00000001 input data word: $I_o = \sum_{i=1}^{8} [A_i / 2^{(i-1)}] = \frac{1}{128} = -7.8125mA$

$V_o = 5(\frac{1}{128} - 1) = -39.0625$ mV

For 10000000 input data word: $I_o = 1.0$ mA

$V_o = 5(1 - 1) = 0.0$ V

For 10000001 input data word: $I_o = \sum_{i=1}^{8} [A_i / 2^{(i-1)}] = 1 + \frac{1}{128} = 1.0078\ A$ (3.20)

$$V_o = 5\left(1 + \frac{1}{128} - 1\right) = 39.0625\ mV \quad (3.21)$$

Shown in figure 3.11 is the complete circuit of the digital audio card while Table 5 is the component list for the complete digital audio circuit.

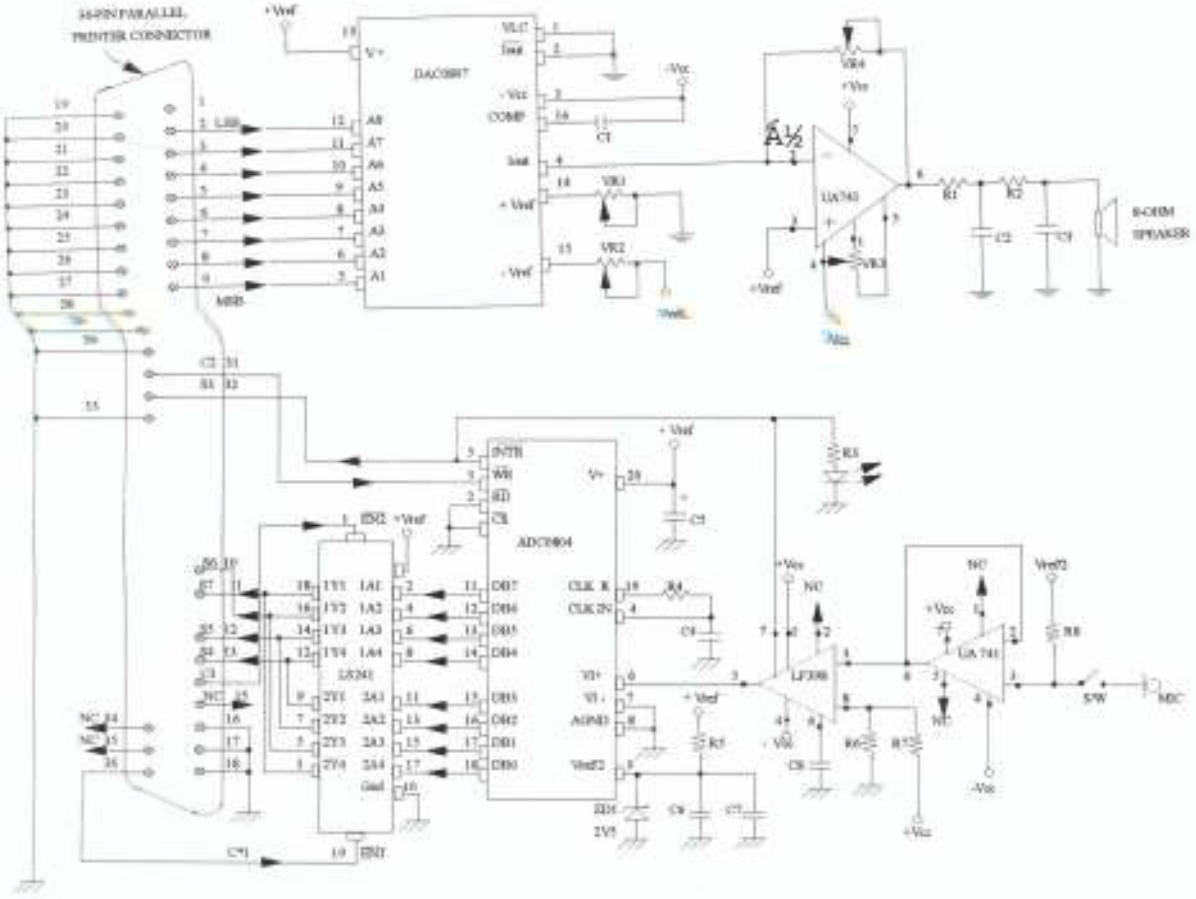


Figure 3.11 Complete Circuit Board of the Digital audio Card.

Table 5.0: Component Lists

CIRCUIT KEY	DESCRIPTION	VALUE
ADC	8-bit Analogue-to-Digital Converter	ADC0804LCN
C_1, C_2, C_6	Ceramic Capacitor	0.1 μF
C_2, C_3	Ceramic Capacitor	0.01 μF
C_4	Ceramic Capacitor	150 pF
C_7, C_9	Electrolytic Capacitor	10 μF, 16V
C_8	Ceramic Capacitor	0.001 μF
DAC	8-bit Digital-to-Analogue Converter	DAC0807LCN
LF398	Sample and Hold Amplifier	LF398
UA741	Operational Amplifier	UAL41CN
LS241	Tri-State Octal Buffer	SN74LS241
MIC	Microphone	
R_1, R_2	Resistor	1 KΩ
R_3	Resistor	330 Ω
R_4	Resistor	10 KΩ
R_5	Resistor	1 KΩ
R_6	Resistor	5.6 KΩ
R_7	Resistor	18 KΩ
R_8	Resistor	1 MΩ
SPEAKER	Speaker	8 Ω
S/W	Switch	SPST
V_{CC}	Supply Voltage	+15 V
VR_1, VR_2, VR_4	Variable Resistor	2.5 KΩ
VR_3	Variable Resistor	10 KΩ
V_{REF}	Reference Voltage	+5 V
ZD 2V5	Zener Diode	+2.5 V

3.3 Software Requirements, Development and Documentation

The choice of computer language is often dependent on the nature of the application. Hence, for the purpose of this research work, (Borland)C++ was chosen. C++ is an expanded version of C and it is an object oriented programming (OOP) language. The object oriented programming features in C++ allow programs to be structured for clarity, extensibility and ease of maintenance without loss of efficiency. It combines elements of high-level languages with the functionalism of assembly language. C++ allows the manipulation of bits, bytes and addresses, i.e. the basic elements with which the computer works. However, other high-level computer languages such as Basic, Pascal , or an assembly language may be used for the implementation of the software driver.

3.3.1 Software Requirements

The version of C++ used, for the compilation of the source code, is Borland C++ 3.1 Version produced in 1992 by Borland International Incorporation. This version requires at least 39 Megabytes(MB) freespace on the hard disk: 34 Megabytes available space in order to install all options and an additional 5 MB of workspace. However, the ***executable file*** of any syntax error-free program will automatically be generated after compilation. So, the compiler will not be necessary for subsequent execution of such programs. It has special in-built functions that can handle several activities, including data transfer between I/O devices and CPU. This special facilities made it possible to control the digital I/O without the need of incorporating any assembly language module(s), as the case may be in some other high level programming language.

3.3.2 Software Design: Development and Documentation

Three separate software packages were designed; the first two packages handle the two experimental interface circuits while the third is the software driver for the digital audio card. That of the interface experiment has been presented in section 3.1 along with its hardware design.

Figure 3.12 is a compressed flowchart for the digital audio card driver, where each of the processing blocks represent one or more subroutine programs. Each of the two processing blocks, "PLAYBACK THE FILE" and "RECORD FOR TR MINUTES" was designed as a separate module comprising of

several sub-functions. These constitute the two main modules, out of the three modules of the software driver. The source code for the software driver is attached to this write-up as appendix A3. The driver can broadly be classified into the following three modules :

- Menu module
- Recording module
- Playback module

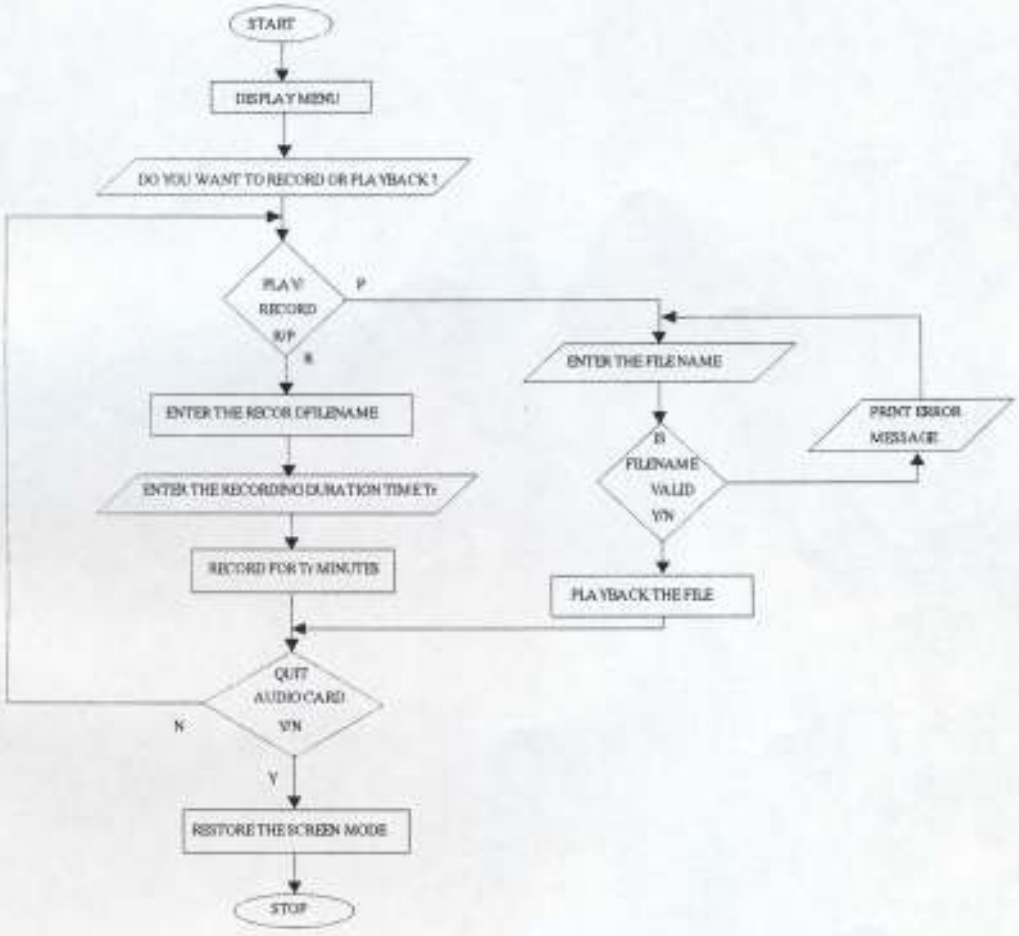


Figure 3.12 Main Flowchart of the Digital Audio Card Software Driver

The Menu module

The general feature of the menu module is illustrated in figure 3.13. The available options in the main menu include; **RECORDING**, **PLAYBACK**, **STOP**, or **QUIT**, depending on the desired mode in which one intend to operate. The graphical capability of C++ is greatly put into use in this module.

Among the in-built prototype functions employed in this module include; line-draw, line-fill, circle-draw, circle-fill, colour- select, square, rectangle, blinking, and lot more.

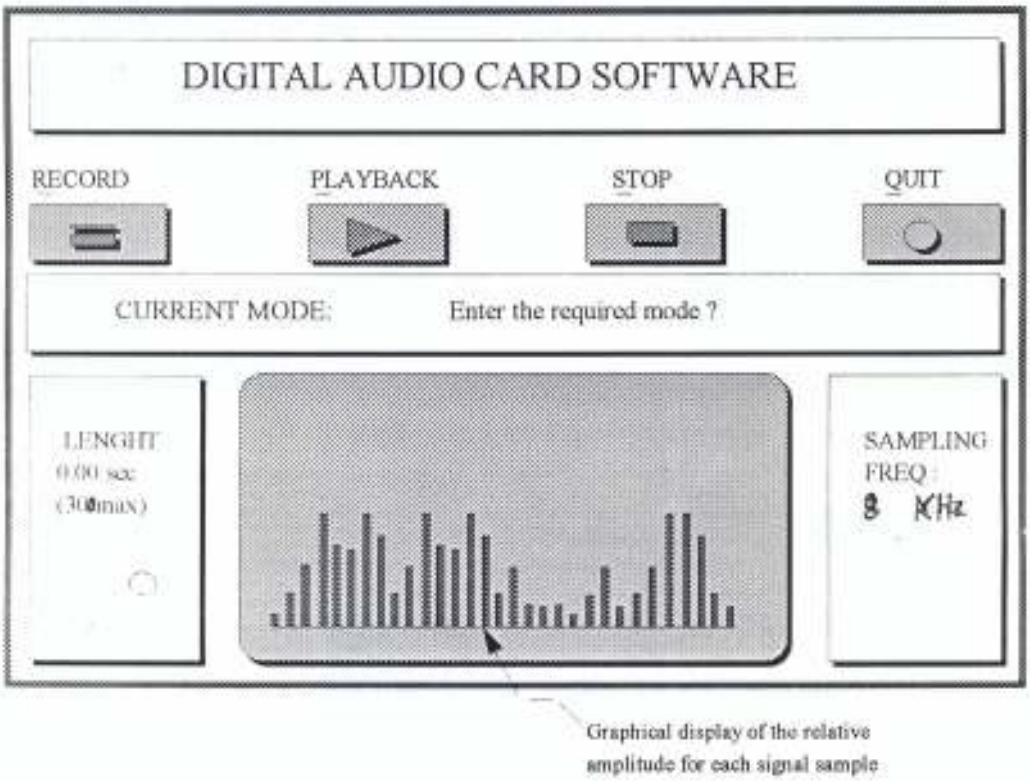


Figure 3.13 General Format of the Main Menu Module

Any of the first alphabets of these modes will effectively put the user in that mode. An error message will be generated if any of the users' input is unrealistic or out of range. An example is the situation whereby the input alphabet for the operating mode is neither of the four letters; "**P**", "**R**", "**Q**", or "**S**", or if the clock speed is unreasonably large or small or not a numeric value.

There is a graphical display section in this main menu, that displays the relative amplitude of successive samples for each voltage signal sample during recording or playback. This is similar to the signal amplitude graphical display unit of most digital audio tape player available nowadays.

The Recording Module

Shown in figure 3.14 is the compressed flowchart for the recording module. An input requirement in this recording module is the name of the file in which the user intend to store the digital signal. The file name should not be more than twelve characters according to the program logic.

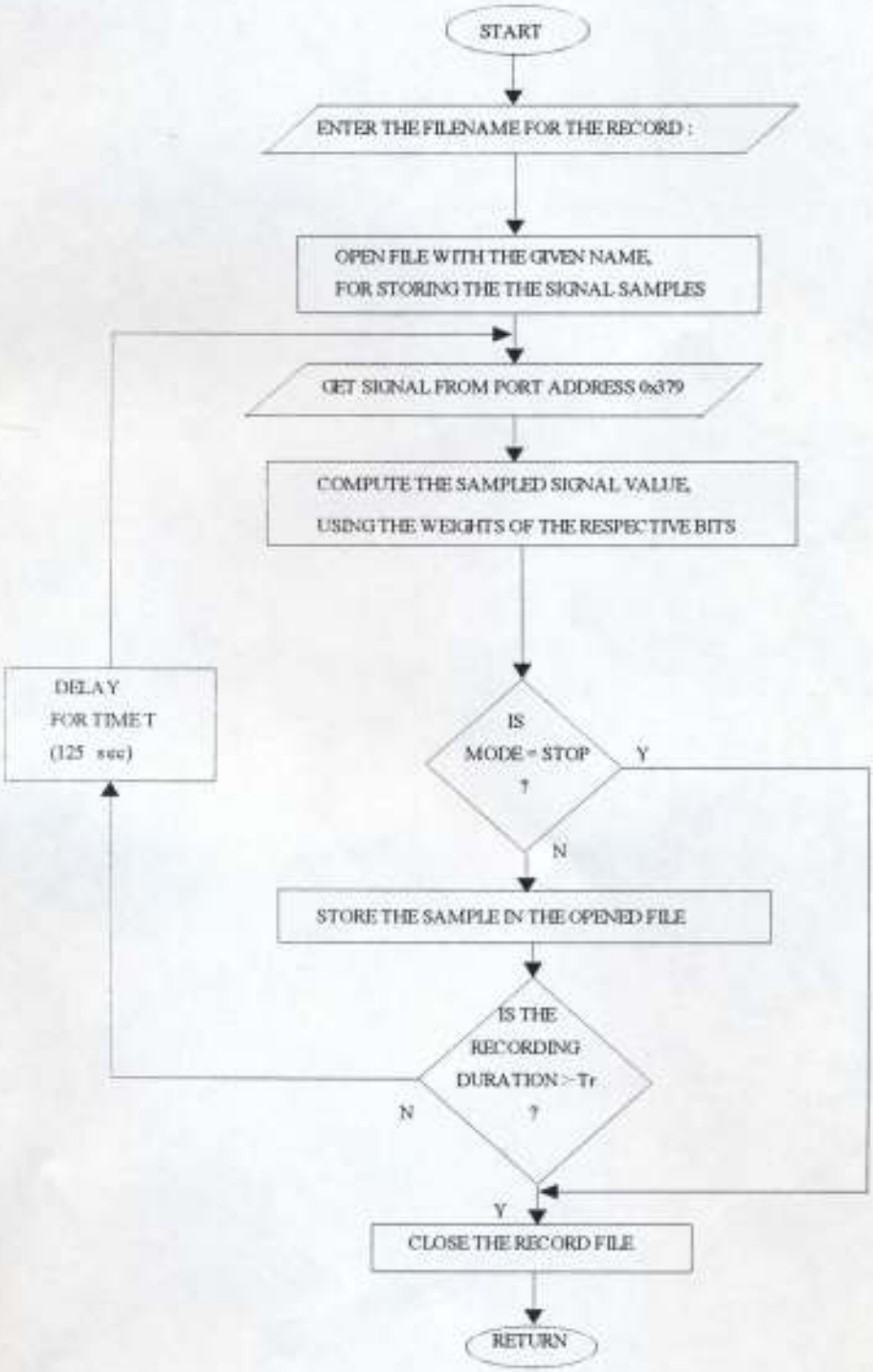


Figure 3.14 Flowchart for Recording Module

The ADC conversion time is much shorter than the the intersamples delay time, τ_D. The delay time is set to 125 microseconds, i.e. the interval between the issuance of successive **READ** (RD) command signal to the recording (or playback) circuit to be 8 KHz.

Summarily, the recording module main function is to obtain the digitized audio signal at the end of the ADC conversion process, compute the weighted value and store it into a file. This is done in two successive **READ** operations via the 4 upper bits of the status port. The weights of each bit' position is used to determine the decimal value of the sampled signal, hence its corresponding analogue value. The ADC will signify the end-of-conversion through its INTR pin, which is normally polled at intervals by this module.

The Playback Module

Figure 3.15 is the flowchart for the playback module. The only input requirement in this playback module is the name of the file in which the user intend to get or retrieve the stored. This section recombines the digital words and send them via the 8-bit wide data port. It does not requires the monitoring of the status of the DAC circuit, since the circuit conversion speed is known to be higher than that of the signal reconstruction speed.

3.5 Discussion and Observations

The conversion from analog to digital and from digital back to analog seem simple in principle, but in practice very difficult. The drift from analog systems toward digital systems is to achieve capabilities which otherwise may be impossible with analog ones. Hence, care must be taken to ensure that the ADC coding scheme is matched to that of the DAC utilized, in order to ensure that the coded analog signal is correctly decoded by the DAC. The best method (which has been employed in this research work) is through software technique. The bits are manipulated before storage in order to achieve a uniform coding for both the ADC and the DAC circuits.

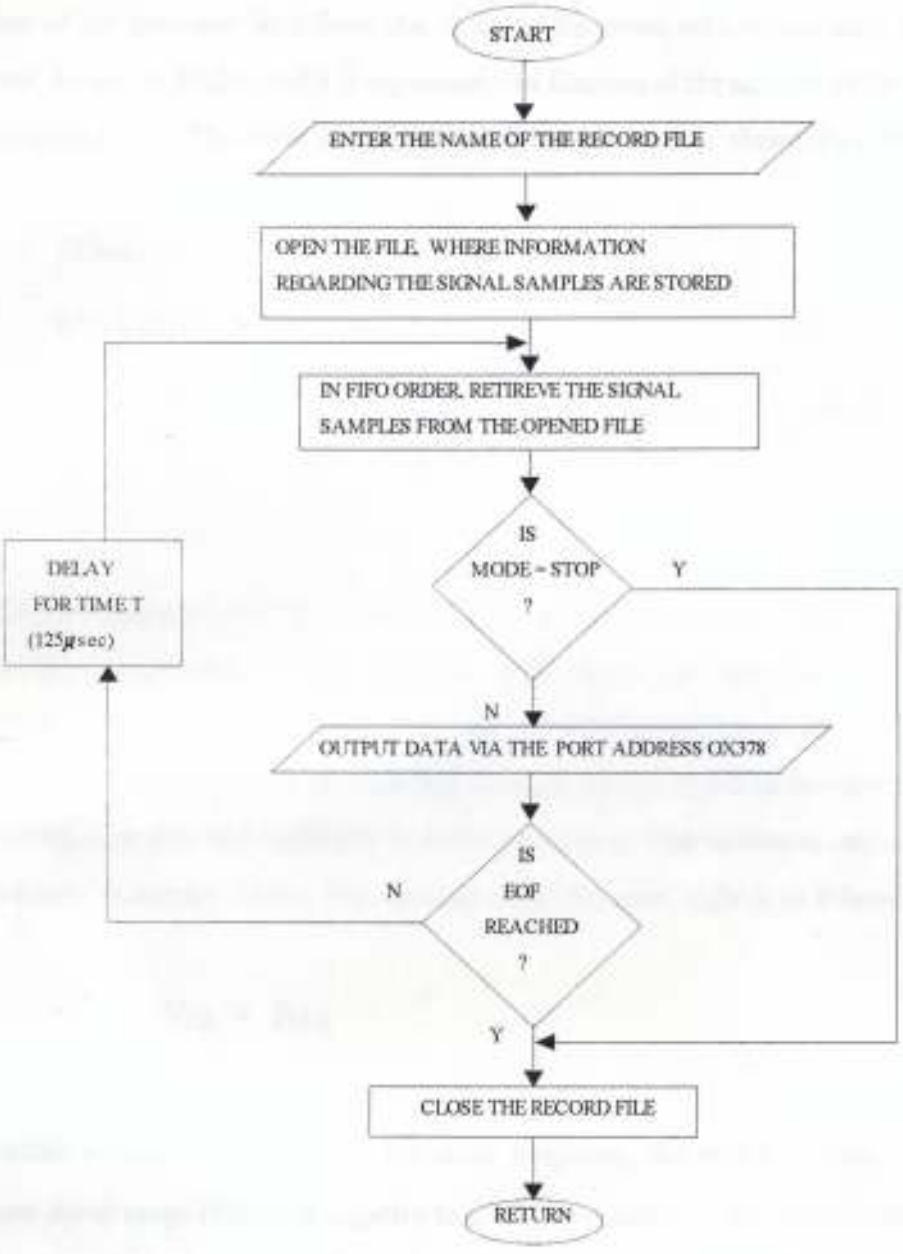


Figure 3.15 Flowchart for Playback Module

Some of the important charcteristics of the ADC that must be into consideration are its '*slew rate*' and '*dynamic range*'.

Dynamic range of an ADC

The dynamic range of an ADC is the ratio of the largest signal the converter can digitised to the smallest value it can resolve (Douglas V. Hall, 1989). Dynamic range (D_R), then, is a way of expressing

the resolution of the converter in a form that is useful for some comparison such as those in audio measurements. Its unit in decibel and it is expressed as a function of the number of bits n of the ADC as written in equation (3.3). The dynamic range of the 8-bit ADC is thus obtained as 48.16 dB.

$$D_R = 20 \log_{10} 2^n$$

$$= 20 n \log_{10} 2$$

$$= 6.02 n \quad \text{dB} \qquad \text{.....(3.22)}$$

Slew Rate

The ADC output changes by one bit (or one LSB in the case of a digital ramp) at the occurrence of each clock pulse. The maximum rate at which the ADC output can change is thus controlled by the clock frequency. If the analogue input rate of change exceeds this maximum rate, the digital output will no longer be accurate representation of the analogue signal. Hence, it will be necessary to incorporate a sample and hold circuit into such analogue to digital conversion. The maximum rate of change is called the loop *'slew rate'* (Commer, 1981). The equation of the slew rate is given as follows:

$$\text{slew rate} = \frac{V_{FSI}}{2^n} \times f_{CLK} \qquad \text{.....(3.23)}$$

where the variables f_{CLK}, n and V_{FSI} are the clock frequency, the number of bits and the full-scale analogue input signal range (full scale negative to full-scale positive in the case of a bipolar converter) of the ADC respectively. The slew rate limitation of a tracking converters sets an upper frequency limit to the full scale sinusoidal signal that can be accurately digitized. A full scale sinusoid may be represented by the following expression:

$$v = \frac{V_{FSI} \sin(2 \pi f t)}{2} \qquad \text{.....(3.24)}$$

The maximum rate of change of the sinusoid is obtained by differentiation as eqn(3.6), i.e.

$$\left| \frac{dv}{dt} \right|_{max.} = 2 \pi f \times \frac{V_{FSI}}{2} \qquad \text{.....(3.25)}$$

The maximum rate of change for which the slew rate is not exceeded is thus determine as follows:

$$f_{max} = \frac{f_{CLK}}{2^n \pi} \quad \text{.....(3. 6)}$$

The occurrence of overflow or underflow of the converter, which is the situation whereby the input signal exceeds the maximum quantisation level (positive $V_{FSI}/2$ or negative $V_{FSI}/2$) should be prevented This situation can be avoided by careful scaling of the input signal.

The initial problem encountered in the software design was to get the correct storing sequence of the binary code words received via the status port and also to send appropriate timing signals vial the control port. The results of the two interface circuit experiments (investigation of data I/O via the parallel port) were of immense help in overcoming these problems. The correct storing of the received data bit were then carried out as follows; the digital output of the ADC was read in two successions (of 4 bit each) via an octal buffer IC. The first read operation takes the 4 upper and the bit positions b_6, b_5 and b_4 were assigned the weights 64, 32 and 16 respectively while the MSB b_7 is used as the sign bit. The second read operation takes the 4 lower bits b_3, b_2, b_1 and b_0 with the associated weights 8, 4, 2 and 1 respectively. The value of the of the received digital signal is then stored as the sum of the two successive read operations.

The major problem encountered in the hardware design was in making the ADC IC to work according to the manufacturer specifications. The ADC was wired for bipolar input range of ±5.0V so that the expected value of the LSB is 39mV (i.e. $10V/2^8$). However, when its input was fed with predetermined analog voltage values, with each of its 8-bits digital outputs connected to an LED, the results obtained were not satisfactory. A single digital output sequence, 01111111, was obtained for all the values even before the application of start conversion pulse. Although, this problem was overcome when the digital ground was separated from the analog ground and the sampling frequency reduced, but digital output sequence still behaves erratically.

CHAPTER FOUR

CONCLUSION AND RECOMMENDATIONS

4.0 Conclusion

A digital audio card and its software driver has been developed for microcomputer systems. The card senses audio analogue signals through a microphone, amplify and store the signal after it has been digitised via an ADC. It also playback the signal by retrieving the digitally stored data, send it through a DAC, amplify and filter the resulting signal before routing it out via an 8-ohm speaker.

The incorporation of this developed digital audio card to a personal computer will increase its functionality at a relatively reduced cost. This include voice coding, recording and playback of audio signals and acquisition of such conditioned analogue signals within the audio signal spectrum and whose peak-to-peak amplitude value is within the ADC range. Digital audio recording and playback systems has drastically replaced the analogue ones because of its improved sound quality, reduced cost and achievements of capabilities which otherwise would be impossible with analogue systems. Apart from its use in domestic appliances, it also find applications in audio signal ***archiving*** and in ***repeaters***. One advantage of this digitally recorded audio signal is that its hardware uses contactless reading operation, hence less liable to friction and wearing.

In developed countries, message repeaters are used in many works of life , from hospitals , schools and factories to retail stores, public transportation services and broadcast services. The original repeater a person walking the door and repeating the message at or about the right time, was replaced by person sitting by microphone making announcement over a public address system. But with the introduction of tape recorders, message could be prerecorded and playback, manually or automatically (i.e. as intelligent repeaters) Some advantages of a digital audio tape repeater systems over the analogue ones include; no rewind time required, it provides solid state reproduction quality, it is more reliable, it is flexible and programmable, it reduces tape wear, it improves file packaging, and significantly improves error rate.

The developed system may be used in banks and also for crime detection as part of a voice recognition system that is gradually replacing recognition by signature, however, further processing of the signal would be required.

4.1 Recommendations: Suggestion For Future Research

The digital audio card was originally designed as a 16-bit system, but, due to scarcity of some of the hardware components (specifically the ADC and DAC ICs), this necessitates the redesigning for an 8-bit system. Although, this affected the resolution but the principle remains the same and the signal quality is still acceptable. So, the use of 16-bit DAC and ADC ICs will increase the word length of the stored data, hence, a better resolution and sound quality could be achieved.

The digital audio signal is stored directly without much processing of the data. Hence, the memory space has not been optimally utilized for high density data storage. Memory space management could be achieved by carrying out compression on the digitized data before being stored and performing the reverse operation (i.e. decompression) before playback. But care must be taken so that the signal quality, the effective use of the bandwidth, interopretability and cost effectiveness of the hardware is not compromised. Compression can be achieved by removing predictable, redundant or predetermined information from a signal.

REFERENCES

Asthana P. 1994. The lesson of Optical Storage. *IEEE Spectrum*. October, 1994. Vol. 31. Number 10. pp. 60 - 66

Baert L., Theunissen L and Vergult G., 1988. Digital Audio and Compact Disc Technology, Sony Service Centre (Europe), *Heinemann Professional Publishing Ltd., Oxford*. 253 pp.

Ballou G., 1996. Digital Audio Storage. *Sound and Video Contractor (S & VC)*. Vol. 14, Number 4, April 20. pp. 10 - 11

Bertsekas D. and Galleger R., 1987. Data Networks, *Prentice-Hall of India Private Limited, New Delhi*. 486 pp.

Carl H.V., Vranesic Z.G. and Zaky S G., 1989. Computer Organisation. *McGraw-Hill Kogaskusha Ltd., Tokyo*. 464 pp.

Clayton G. B., 1982. Data Converters. *The Macmillan Press Ltd., London*. 242pp.

Commer D.J., 1981. Electronic Design with Integrated Circuits. *Addison - Wesley Publishing Company Inc., Reading*. 216 pp.

Coughlin R.F. and Driscoll F.F. 1991. Operational Amplifiers and Linear Circuits. 4th. Edition. *Prentice-Hall International Inc., New Jersey*. 552 pp.

Cyrillic, 1995. *Peripheral Printer Operation Manual Specification*. RA Version, 122 pp.

Dalghish R. L., 1987. An Introduction to Control and Measurement with Computers. *Cambridge University Press, Cambridge*. 342 pp.

Daniel E.D. and Walkinson J.R., 1988. Audio Recording. In : *Magnetic recording Volume III: Video Audio and Instrumentation Recording*. (Denis Mee and Eric D. Daniel). McGraw-Hill Publishing Inc., New York. pp. 149 - 217

Ercegovac M.D. and Lang T., 1991. Digital systems and Hardware/Software Algorithms. *John Wiley and Sons, New York.* 838 pp.

Falaki S.O., 1974. *Design of an Introductory Digital System Laboratory.* M.Sc. (Computer Science) Thesis. University of Carlifonia, Los Angeles, USA.

Falaki S.O., 1981. *Design of Digital Filters, Using Incremental Changes in Processing.* Ph.D.. (Electrical Engineering) Dissertation. University of Lagos, Nigeria.

Franco S., 1988. Design with Operational Amplifiers and Analogue Integrated Circuits. *McGraw-Hill Publishing Inc., New York.* 636 pp.

Gadre, D.V., Upadhyay P.K. and Varna V.S. ,1994. Catching the Right Bus, Part 2: Using a Parallel Printer Adapter as An Inexpensive Interface. *Computer in Physics*, Vol. 8, No 1 Jan/Feb. pp. 45 - 50.

Hall D.V., 1989. Digital Circuit and Systems. *McGraw-Hill Publishing Inc., New York.* 554pp.

Kehinde L. O. and Ojetayo T. K., 1992. A Computer-Based Industrial Controller and Data Acquisition System.

Lee S.C., 1981. Digital Circuits and Logic Design. *Jay Pack Private Limited, New Delhi.* 594 pp.

Litteler J. and Maher J., 1989. Computers in the Laboratory: A student Guide to Microprocessor *Interfacing. Longman Scientific & Technical, Essex, England.* 385 pp.

Miller A., 1994. Data Communication: From Here to ATM. *IEEE Spectrum.* June, 1994. Vol. 31. Number 6. pp. 20 - 24

Murphy E.E. and Voelcher J., 1990. System Software: OOP a Hit. *IEEE Spectrum.* January, 1990. Vol. 27. Number 1. pp. 38 - 40

National Instrument Inc., 1996. *Instrumentation Reference and Catalogue: Test Measurement and Industrial Automation.* National Instrument, Inc., Texas pp. 2-12 - 2-13.

Oni, K.C., 1995. *Research Reporting: A Guide to Thesis Preparation for Agricultural Engineers.* 27 pp.

Oni, J.O., 1982. *Notes On Dynamic Systems.* Pennsylvania State University, USA. 48 pp.

Oni, J.O., 1983. *Notes On Switching Theory and Logic Design.* Polytechnic Institute of New York, NY., USA. 53 pp.

Patel A.M., 1988. Signal and Error Control Coding. In : *Magnetic recording Volume III: Video Audio and Instrumentation Recording.* (Denis Mee and Eric D. Daniel). McGraw-Hill Publishing Inc., New York. pp. 247 - 308

Peatman J.B., 1981. Digital Hardware Design. *McGraw-Hill Kogaskusha Ltd., Tokyo.* 438 pp.

Protopapas D. A., 1988. Microcomputer Hardware Design. *Prentice-Hall International, Inc., New Jersey* 264 pp.

Robert Sedgewick, 1983. Algorithms in C. *Addison-Wesley Publishing Co., Inc.*, Reading. 650 pp.

Robinson P.D., Klein P. and Ballou G., 1996. Digital Audio Storage: Applications in Digital Voice and Music Compression Technology. *Sound and Video Contractor (S & VC).* Vol. 14, Number 4, April 20. pp. 12 - 28

Schildt H., 1992. Teach Yourself C^{++}. *Osborne Mc Graw-Hill Inc,* 346 p.

Stone H.S., 1982. Microprocesor Interfacing. *Addison-Wesley Publishing Co., Inc.*, Reading 383 pp.

Tocci R.J. 1980. Digital Systems: Principles and Applications. *Prentice-Hall International Inc., London.* 540 pp.

Vears R. E., 1992. Microcomputer Interfacing. *Butterworth - Heinemann Ltd., Oxford.* 192 pp.

APPENDIX A1

```
#include "graphics.h"
#include "stdlib.h"
#include "conio.h"
#include"stdio.h"
#define D_PORT 0x378
#define S_PORT 0x379
#define C_PORT 0x37A
void demo_display(void);
void plot_LED(void);
red_s_port();
int L1,L2,L3,L4,num2,value;
char mod1,wish1;
main()
   {
   demo_display();
   plot_LED();
   outtextxy(15,400,"Press any key to exit ....");
   while(!kbhit())
       {
       sleep(1);
       }
   }

void demo_display(void)
   {
   struct viewporttype vp;
   int mode,driver;
   mode = DETECT;
   driver = DETECT;
   initgraph (&mode,&driver, " ");
   setcolor(5);
   settextstyle(TRIPLEX_FONT,HORIZ_DIR,5);
   outtextxy(5,100," COMPUTER INTERFACING VIA ");
   outtextxy(100,210," THE PARRALEL PORT");
   sleep(2);
   cleardevice();
   settextstyle(TRIPLEX_FONT,HORIZ_DIR,5);
   outtextxy(40,100,"LIGHT-EMITTING DIODE");
   outtextxy(50,210,"INTERFACE CIRCUIT");
   sleep(1);
   setcolor(4);
   settextstyle(TRIPLEX_FONT,HORIZ_DIR,5);
   outtextxy(120,300,"Developed by");
   outtextxy(130,400,"E.O. Adeola");
   sleep(1);
   cleardevice();
   }
```

```
void plot_LED(void)
    {
    struct viewporttype vp;
    int x,y, i, L1, L2;
    char mod, wish;
    char buffa[40];
    char numb[16] = { '0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'};
    getviewsettings(&vp);
    setfillstyle(SOLID_FILL,WHITE);
    setcolor(BLUE);
    bar(1,1,600,450); /* Draw a rectangle  */
    rectangle(1,1,600,450); /* Outline the rectangle  */
    rectangle(3,3,598,448);
    setcolor(BLACK);
    moveto(1,60);
    lineto(600,60);  /* Position cusor at the origin of the graph */
    setfillstyle(SOLID_FILL,BLUE);
    bar(4,4,597,59);
    setcolor(BLACK);
    settextstyle(TRIPLEX_FONT,HORIZ_DIR,3);
    outtextxy(30,15,"LED INTERFACE CIRCUIT EXPERIMENT");
    settextstyle(TRIPLEX_FONT,HORIZ_DIR,2);
    setcolor(RED);
    outtextxy(15,100,"Do you want operate Manually/Automatically (M/A)
?");
    mod = getch();
    mod = toupper(mod);
    switch(mod)
       {
       case('M'):
             {
    outtextxy(15,150,"Do you send or receive (S/R) ?");
    mod1 = getch();
    mod1 = toupper(mod1);
    switch(mod1)
       {
       case('S'):
             {
             outtextxy(15,200,"Do you wish to send via data/controlport
(D/C) ");
             wish1 = getch();
             wish1 = toupper(wish1);
             if(wish1 == 'D')
               {
                setfillstyle(SOLID_FILL,WHITE);
                bar(14,90,590,445);

L1:          outtextxy(15,150,"Enter any number between 0 and 255 :");
             setfillstyle(SOLID_FILL,RED);
             scanf("%d", &num2);
             gotoxy(10,60);
             printf("Sending the number %d via the data port",num2);
             outport(D_PORT,num2);
             bar(500,130,560,190);
             sleep(2);
             outtextxy(15,250,"Do you wish to display more (Y/N) ");
             wish = getch();
             wish = toupper(wish);
```

```
        if(wish == 'Y')
          {
           setfillstyle(SOLID_FILL,WHITE);
           bar(14,90,590,445);
           goto L1;
          }
        if(wish == 'N')
          {
          goto L2;
          }
        else
           {
           outtextxy(15,300,"ERROR !!!");
           outtextxy(15,350,"Enter Y for Yes and N for No");
           setfillstyle(SOLID_FILL,WHITE);
           bar(14,290,590,420);
           }
      }

L3:
      if(wish1 == 'C')
        {
         setfillstyle(SOLID_FILL,WHITE);
         bar(14,90,590,445);
        outtextxy(15,150,"Enter any number between 0 and 15 :");
        setfillstyle(SOLID_FILL,RED);
        bar(500,130,560,190);
        scanf("%d", &num2);
        gotoxy(10,60);
        printf("Sending the number %d via the control port",num2);
        outport(C_PORT,num2);
        sleep(2);
        outtextxy(15,250,"Do you wish to display more (Y/N) ");
        wish = getch();
        wish = toupper(wish);
        if(wish == 'Y')
          {
           setfillstyle(SOLID_FILL,WHITE);
           bar(14,90,590,445);
           goto L3;
          }
        if(wish == 'N')
          {
          goto L2;
          }
        else
           {
           outtextxy(15,300,"ERROR !!!");
           outtextxy(15,350,"Enter Y for Yes and N for No");
           setfillstyle(SOLID_FILL,WHITE);
           bar(14,290,590,420);
           }
      }
    else
           {
           outtextxy(15,300,"ERROR !!!");
           outtextxy(15,350,"Enter C for Control port or D for data
port");
```

```
                setfillstyle(SOLID_FILL,WHITE);
                bar(14,290,590,420);
                }
        break;
       }
     case('R'):
                {
                setfillstyle(SOLID_FILL,WHITE);
                bar(14,80,597,420);
                outtextxy(15,100,"Automatic operation");
                outtextxy(15,300,"receiving the values 0 - 31 via the
status port");
                for(i=0; i<=31; i++)
                      {
                      value = read_s_port0;
                      gotoxy(20,60);
                      printf("receiving %d via the status port,value");
                      }
                }   /* End of case 'R' */
     default:
            {
              setfillstyle(SOLID_FILL,WHITE);
              bar(5,65,597,445);
              outtextxy(15,300,'ERROR !!!");
              outtextxy(15,350,"Enter R for receive and S for sending
data");
              while(!kbhit())
                 {
                 delay(5);
                 }
              setfillstyle(SOLID_FILL,WHITE);
              bar(5,65,597,445);
              break;
            }
          }
              outtextxy(15,150,"Enter any number between 0 and 255 :");
              setfillstyle(SOLID_FILL,RED);
              bar(500,130,560,190);
              display_number();
              sleep(2);
              outtextxy(15,250,"Do you wish to display more (Y/N) ");
              wish = getch();
              wish = toupper(wish);
              if(wish == 'Y')
               {
                setfillstyle(SOLID_FILL,WHITE);
                bar(14,90,590,445);
                goto L1;
               }
              if(wish == 'N')
               {
               goto L2;
               }
              else
                {
                outtextxy(15,300,'ERROR !!!");
                outtextxy(15,350,"Enter Y for Yes and N for No");
                setfillstyle(SOLID_FILL,WHITE);
```

```
                bar(14,290,590,420);
                }
        break;
        }
         case('A');
                {
                setfillstyle(SOLID_FILL,WHITE);
                bar(14,80,597,420);
                outtextxy(15,100,"Automatic operation");
                outtextxy(15,300,"Sending the value 0 - 255 via the data
port");
                for(i=0; i<=255; i++)
                      {
                       outport(D_PORT,i);
                       delay(100);
                      }
                setfillstyle(SOLID_FILL,WHITE);
                bar(14,80,597,420);
                outtextxy(15,100,"Automatic operation");
                outtextxy(15,300,"Sending the value 0 - 15 via the control
port");
                for(i=0; i<=15; i++)
                      {
                      outport(C_PORT,i);
                      gotoxy(20,50);
                      printf("sending %d \n",i);
                      delay(100);
                      }
                }
         break;
         }   /* End of switch  */
       default:
              {
               setfillstyle(SOLID_FILL,WHITE);
               bar(5,65,597,445);
               outtextxy(15,300,"ERROR !!!");
               outtextxy(15,350,"Enter M for Manual and A for Auto");
               while(!kbhit())
                  {
                   delay(5);
                  }
               setfillstyle(SOLID_FILL,WHITE);
               bar(5,65,597,445);
               break;
              }
             }
L2:    return;
       }

/*     Module to read the upper 4 bits of the status port and assign the
       weights 1,2,4,8,16 to the bits b3,b4, b5,b6 and b7 respectively.
*/
read_s_port()
     {
      int bit[5];
      int weight2[5] = {
      1,2,4,8,16    };
      unsigned int w,bits,sum;
```

```
int i,j;
bits = inport(S_PORT);
for(i=0; i<=4; i++)
    {
    bit[i] = bits;
    }
j = 0;
sum = 0;
for(i=3; i<=7; i++)
   {
   bit[j] >>= i;
   if(i == 7)
     {
     w = (bit[j] & 0x1) ? 0:1;
     w*=weight2[j];
     }
   else{
      w = (bit[j] & 0x1) ? 1:0;
      w*= weight2[j];
       }
   sum+=w;
   j++;
   }
return(sum);
}
```

APPENDIX A2

```
#include "graphics.h"
#include "stdlib.h"
#include "conio.h"
#include"stdio.h"
#define D_PORT 0x378
#define S_PORT 0x379
#define C_PORT 0x37A
void demo_display(void);
void plot_seven_seg(void);
void display_number(void);
main()
   {
   demo_display();
   plot_seven_seg();
   outtextxy(15,400,"Press any key to exit . . . .");
   while(!kbhit())
       {
       sleep(1);
       }
   }

void demo_display(void)
   {
   struct viewporttype vp;
   int mode,driver;
   mode = DETECT;
   driver = DETECT;
   initgraph (&mode,&driver," ");
   setcolor(5);
   settextstyle(TRIPLEX_FONT,HORIZ_DIR,5);
   outtextxy(5,100," COMPUTER INTERFACING VIA ");
   outtextxy(100,210," THE PARRALEL PORT");
   sleep(2);
   cleardevice();
   settextstyle(TRIPLEX_FONT,HORIZ_DIR,5);
   outtextxy(40,100,"7 - SEGMENT DISPLAY");
   outtextxy(50,210,"INTERFACE CIRCUIT");
   sleep(1);
   setcolor(4);
   settextstyle(TRIPLEX_FONT,HORIZ_DIR,5);
   outtextxy(120,300,"Developed by");
   outtextxy(130,400,"E.O. Adeola");
   sleep(1);
   cleardevice();
   }

   void plot_seven_seg(void)
      {
      struct viewporttype vp;
      int x,y, i, L1, L2;
      char mod, wish;
      char buffa[40];
      char numb[16] = { '0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'};
      getviewsettings(&vp);
```

```
     getviewsettings(&vp);
     setfillstyle(SOLID_FILL,WHITE);
     setcolor(BLUE);
     bar(1,1,600,450); /* Draw a rectangle  */
     rectangle(1,1,600,450); /* Outline the rectangle  */
     rectangle(3,3,598,448);
     setcolor(BLUE);
     moveto(1,60);
     lineto(600,60);       /* Position  cusor at  the origin  of
the graph */
     setfillstyle(SOLID_FILL,GREEN);
     bar(4,4,597,59);
     setcolor(BLACK);
     settextstyle(TRIPLEX_FONT,HORIZ_DIR,3);
     outtextxy(30,15,"SEVEN    SEGEMENT    DISPLAY    INTERFACE
CIRCUIT");
     settextstyle(TRIPLEX_FONT,HORIZ_DIR,2);
     setcolor(RED);
L1:  outtextxy(15,100,"Do        you        want        operate
Manually/Automatically (M/A) ?");
     mod = getch();
     mod = toupper(mod);
     switch(mod)
         {
         case('M'):
                {
                outtextxy(15,150,"Enter any  Hex number  between
0 ,1 ,2 .... E ,F :");
                setfillstyle(SOLID_FILL,RED);
                bar(500,130,560,190);
                display_number();
                sleep(2);
                outtextxy(15,250,"Do  you wish  to display  more
(Y/N) ");
                wish = getch();
                wish = toupper(wish);
                if(wish == 'Y')
                  {
                setfillstyle(SOLID_FILL,WHITE);
                bar(14,90,590,445);
                goto L1;
                  }
                if(wish == 'N')
                  {
                  goto L2;
                  }
                else
                {
                outtextxy(15,300,"ERROR !!!");
                outtextxy(15,350,"Enter  Y  for  Yes  and N  for
No");
                setfillstyle(SOLID_FILL,WHITE);
                bar(14,290,590,420);
                }
       break;
       }
        case('A'):
                {
```

```
        setfillstyle(SOLID_FILL,WHITE);
        bar(14,80,597,420);
        outtextxy(15,100,"Automatic operation");
        for(i=0; i<=15; i++)
              {
  buffa[0] = numb[i];
switch(i)
   {
   case(0):
        {
        outport(D_PORT,0x7E);
        break;
        }
   case(1):
        {
        outport(D_PORT,0x30);
        break;
        }
   case(2):
        {
        outport(D_PORT,0x6D);
        break;
        }
   case(3):
        {
        outport(D_PORT,0x79);
        break;
        }
   case(4):
        {
        outport(D_PORT,0x33);
        break;
        }
   case(5):
        {
        outport(D_PORT,0x5B);
        break;
        }
   case(6):
        {
        outport(D_PORT,0x5F);
        break;
        }
   case(7):
        {
        outport(D_PORT,0x70);
        break;
        }
   case(8):
         {
         outport(D_PORT,0x7F);
         break;
         }
   case(9):
        {
        outport(D_PORT,0x73);
        break;
        }
```

```
            case(10):
                {
                outport(D_PORT,0x77);
                break;
                }
            case(11):
                {
                outport(D_PORT,0x1F);
                break;
                }
            case(12):
                {
                outport(D_PORT,0x4E);
                break;
                }
            case(13):
                {
                outport(D_PORT,0x3D);
                break;
                }
            case(14):
                {
                outport(D_PORT,0x4F);
                break;
                }
            case(15):
                {
                outport(D_PORT,0x47);
                break;
                }
          }     /* End of case 'A' */

          outtextxy(15, 200, buffa);
          setfillstyle(SOLID_FILL,WHITE);
          outtextxy(40,   200,"is   been   displayed   on   the
7-Segment module");
          sleep(2);
          bar(5,190,597,445);
        }
     break;
        }    /* End of switch  */
     default:
            {
              setfillstyle(SOLID_FILL,WHITE);
              bar(5,65,597,445);
              outtextxy(15,300,"ERROR !!!");
              outtextxy(15,350,"Enter M  for Manual  and A for
Auto");
              while(!kbhit())
               {
               delay(5);
               }
              setfillstyle(SOLID_FILL,WHITE);
              bar(5,65,597,445);
              break;
            }
          }
L2:  return;
```

```
}

display_number()
        {
        int L3;
        char buffer[4];
        char ch;
        ch = getch();
        ch = toupper(ch);

        switch(ch)
           {
           case('0'):
               {
               outport(D_PORT,0x7E);
               break;
               }
           case('1'):
               {
               outport(D_PORT,0x30);
               break;
               }
           case('2'):
               {
               outport(D_PORT,0x6D);
               break;
               }
           case('3'):
               {
               outport(D_PORT,0x79);
               break;
               }
           case('4'):
               {
               outport(D_PORT,0x33);
               break;
               }
           case('5'):
               {
               outport(D_PORT,0x5B);
               break;
               }
           case('6'):
               {
               outport(D_PORT,0x5F);
               break;
               }
           case('7'):
               {
               outport(D_PORT,0x70);
               break;
               }
           case('8'):
                {
                outport(D_PORT,0x7F);
                break;
                }
```

```
            case('9'):
                {
                outport(D_PORT,0x73);
                break;
                }
            case('A'):
                {
                outport(D_PORT,0x77);
                break;
                }
            case('B'):
                {
                outport(D_PORT,0x1F);
                break;
                }
            case('C');
                {
                outport(D_PORT,0x4E);
                break;
                }
            case('D');
                {
                outport(D_PORT,0x3D);
                break;
                }
            case('E');
                {
                outport(D_PORT,0x4F);
                break;
                }
            case('F');
                {
                outport(D_PORT,0x47);
                break;
                }
            default:
                {
                setfillstyle(SOLID_FILL,WHITE);
                bar(14,190,590,420);
                outtextxy(15,200,"ERROR !!!");
                outtextxy(15,250,"Only an  Hex number between  0
- F can be displayed");
                setfillstyle(SOLID_FILL,WHITE);
                sleep(2);
                bar(14,190,590,420);
                goto L3;
                }
            }
              setcolor(WHITE);
              buffer[0] = ch;
              outtextxy(513, 150, buffer);
              setcolor(RED);
              outtextxy(15, 200, buffer);
              outtextxy(40,  200,"is   been  displayed  on   the
7-Segment module");
              sleep(1);
              setfillstyle(SOLID_FILL,WHITE);
              bar(5,265,597,445);
```

```
L3:       return;
          }
```

APPENDIX A3

```
#include <io.h>
#include <stdio.h>
#include <fcntl.h>
#include <graphics.h>
#include <time.h>
#include <dos.h>
#include <ctype.h>
#include <conio.h>
#define D_PORT 0x378
#define S_PORT 0x379
#define C_PORT 0x37A
#define X1i 170.0
#define Y2 425.0
FILE *fp1,*fp;
void show(void);
void beep(void);
read_s_port1();
read_s_port2();
take_signal();
void arrow_left(int x1,int y1);
void arrow_right(int xr,int yr);
void ADC_SC(void);
void ADC_RD(void);
void clear_mode(void);
void record_mode(void);
play_audio_signal(int n_samples,int sampling_time);
record_audio_signal(int n_samples,int sampling_time);
unsigned int check_status();
void plot(int value,float x1,float x2);
int signbit, bit0_6;
int r[1],op,num,L1;
char *tpv;
char play_file[12];
float durat;
main()
     {
     struct viewporttype vp;
     time_t starttime;
     time_t endtime;
     char funtion;
     int mode,driver,duration,n_samples,max_samples, values;
     float x1,x2, sampling_t;
L1:  getviewsettings(&vp);
     mode = DETECT;
     driver = DETECT;
     initgraph(&mode,&driver," ");
     cleardevice();
     show();
     funtion = getch();
     clear_mode();
     funtion = toupper(funtion);
     sampling_t =0.125 ;
```

```
switch(funtion)
     {
     case('R'):
        {
        clear_mode(); /* Refresh the displayed mode on the screen */
        gotoxy(24,11);
        printf("Enter the duration(in minutes): ");
        scanf("%d",&duration);
        clear_mode();
        gotoxy(24,11);
        if(duration <= 5)
          {
          printf("Enter the filename for the recording:");
          scanf("%s",&play_file);
          clear_mode();
          outtextxy(220,152,"RECORDING");
          fp = fopen(play_file,"w");
          starttime=time(NULL);

          x1 = X11;
          x2 = X11 + 5;
          do
            {
            ADC_SC();
            max_samples = (duration * 60)/sampling_t;
            record_audio_signal(max_samples,sampling_t);
            endtime=time(NULL);
            durat=difftime(endtime,starttime);
            } while (durat <= duration*60);
        }
        else
          {
          clear_mode();
          gotoxy(24,11);
          printf("The duration time should not be more than 5 minutes
\n");
          }
        fclose(fp);
        goto L1;
        break;
        }
      case('P'):
        {
        gotoxy(24,11);
        printf("Enter the filename for the recording: ");
        scanf("%s",&play_file);
        clear_mode();
        op = open(play_file,O_RDONLY);
        clear_mode();
        outtextxy(220,152,"PLAYING");
        arrow_right(330,175);  /* display the symbol for recording */
x1 = X11;
x2 = X11 + 5;
values = 5;
        do
```

```
        {
        num = read(op,tpv,2);
        n_samples = atoi(tpv);

        clear_mode();
        gotoxy(24,11);
        printf("%d",n_samples);
        n_samples+=128;

        play_audio_signal(n_samples,sampling_t); */
        plot(n_samples,x1,x2);
        x1 = x2 + 2.0;
        x2 = x1 + 5;
        if(x2 > 430)
           {
           setfillstyle(SOLID_FILL,8);
           bar(160,200,430,430);
           x1 = X11;
           x2 = x1 + 5;
           }
        values ++;
        delay(sampling_t);
              }while(!EOF);  */
        } while (num != -1);
        close(op);
        goto L1;
        break;
        }
   case('Q'):  /* Quitting to the system  */
        {
        outtextxy(220,152,"Quit");
        beep();
        exit(1);
        break;
        }

   case('S'):  /* Stop the operation  */
        {
        clear_mode();
        outtextxy(220,152,"STOP");
        clear_mode();
        outtextxy(220,152,"Press any key ...");
        beep();
        sleep(2);
        goto L1;
        break;
        }


   default:
        {
        clear_mode();
        outtextxy(220,152,"ERROR !!!");
        beep();
        break;
        }
   }
sleep(2);
```

```
      return 0;
      }

play_audio_signal(int n,int sampling_time)
     {
     int  values;
     float x1,x2;
     x1 = X11;
     x2 = X11 + 5;
     values = 5;
     setfillstyle(SOLID_FILL,8);
     bar(160,200,430,430);

        plot(n,x1,x2);
        x1 = x2 + 2.0;
        x2 = x1 + 5;
        if(x2 > 430)
       {
       setfillstyle(SOLID_FILL,8);
       bar(160,200,430,430);
       x1 = X11;
       x2 = x1 + 5;
       }
        values ++;
        delay(sampling_time);
    return 0;
    }

void plot(int value, float x1, float x2)
     {
     int i;
     float y1;
     y1 = Y2 - (float)value/2.0;
     setcolor(14);
     for(i = Y2; i > y1; i--)
      {
      moveto(x1,i);
      lineto(x2,i);
      }
     return;
     }

void beep(void)
     {
     printf("%c",7);
     delay(100);
     }

read_s_port1() /* reading  the lower 4 bits  of the status port
*/
        {
        int bit[4];
        int weight1[4] = {
        1,2,4,8          };
        unsigned int w,bits,sum;
        int i,j;
        bits = inport(S_PORT);
        for(i=0; i<4; i++)
```

```
            {
            bit[i] = bits;
            }
        j = 0;
        sum = 0;
        for( i = 4; i<=7; i++)
            {
            bit[j] >>= i;

            if(i == 7)
              {
              w = (bit[j]  & 0x1) ? 0:1; /*  Get the value of pin
7 in inverted form */
              }
            else{
             w = (bit[j] & 0x1) ? 1:0;
             }
            w *= weight1[j];
            sum += w ;
            j++;
            }
        return(sum);
        }

/*    Module to  read the upper  4 bits of  the status  port and
      assign the weights 16, 32,  and 64 to the bits b4, b5, and
      b6 respectively while  b7, which is used  as the sign bit,
      is assigned the weight 128.
*/
read_s_port2()
       {
        int bit[4];
        int weight2[4] = {
        16,32,64,128      };
        unsigned int w,bits,sum;
        int i,j;
        bits = inport(S_PORT);
        for(i=0; i<=4; i++)
         {
         bit[i] = bits;
         }
        j = 0;
        sum = 0;
        for(i=4; i<=7; i++)
        {
        bit[j] >>= i;
        if(i == 7)
          {
          w = (bit[j] & 0x1) ? 0:1;
          signbit = w;        /* store the value of  pin 7 as the
sign bit */
          w*=weight2[j];
          }
        else{
            w = (bit[j] & 0x1) ? 1:0;
            w*= weight2[j];
             }
        sum+=w;
```

```
        j++;
        }
        return(sum);
        }

take_signal()
        {
        unsigned int check,bit0_3,bit4_6,bit0_6;
        check = check_status();
        if(check == 1)     /*  check = 1 indicates that   end of
conversion is reached */
          {
          ADC_RD();
          outport(C_PORT,0x6);  /* Enable  the lower  4 bits of
the tristate switch */
          bit0_3 = read_s_port1();
          ADC_RD();
          outport(C_PORT,0xA); /* Enable  the upper  4 bits  of
the tristate switch */
          bit4_6 = read_s_port2();
          ADC_RD();
          bit0_6 =  bit0_3  +  bit4_6;   /*  compute the  total
analog value of the signal */
          }
        else /*  To be executed if conversion is still going on
*/
            {
            gotoxy(24,11);
            printf("STATUS HIGH !!!");
            beep();
            clear_mode();
            }
        return(bit0_6);
        }

void ADC_SC(void)   /* Start conversion command for the ADC */
      {
      outport(C_PORT,0x0);
      return;
      }

void ADC_RD(void)     /* Read command of the ADC */
       {
       outport(C_PORT,0x4);
       return;
       }

/*         Module to check  the conversion status of  the ADC :
Monitors the INTR line
     of  the ADC via the  status port line  s3. It  returns a 1
for  end of conversion
     and a 0 otherwise.
*/
     unsigned int check_status()
          {
          unsigned int scheck,sbit3;
          sbit3 = inport(S_PORT);
          sbit3 >>= 3;
```

```
            scheck = (sbit3 & 0x1) ? 1:0;
            return(scheck);
          }

record_audio_signal(int n,int sampling_time)
        {
        int values,count, n_samples;
        float x1,x2;
        x1 = X11;
        x2 = x1 + 5;
        setfillstyle(SOLID_FILL,8);
        bar(160,200,430,430);
        values = 4;
        for(count = 0; count < n; count ++)
        {
/*         n_samples = take_signal();
        fprintf(fp,"%x",n_samples);
*/
        clear_mode();
        gotoxy(24,11);
        setcolor(RED);
        outtextxy(210,152,"RECORDING");
        setcolor(YELLOW);

        n_samples = values;

        plot(n_samples,x1,x2);
        x1 = x2 + 2.0;
        x2 = x1 +5;
        if(x2 > 430)
          {
          setfillstyle(SOLID_FILL,8);
          bar(160,200,430,430);
          x1 = X11;
          x2 = x1 + 5;
          }
        if(values > 300) values = 0;
        delay(sampling_time);
        values ++;
        }
        return;
        }

void show(void)
     {
     setfillstyle(SOLID_FILL,WHITE);
     bar(51,1,550,450);
     setfillstyle(SOLID_FILL,7);
     bar(53,3,548,60);
     bar(70,100,160,130);
     bar(190,100,280,130);
     bar(310,100,400,130);
     bar(430,100,520,130);
     bar(70,140,520,190);
     bar(70,200,150,430);
     bar(440,200,520,430);
     setcolor(BLACK);
     outtextxy(90,85,"RECORD        PLAYBACK          STOP
```

```
QUIT");
 outtextxy(90,88,"_                    _                    _
_    ");
 outtextxy(80,220,"LENGHT:
 SAMPLING");
 outtextxy(80,240,"0.00 Sec.
 FREQ:");
 outtextxy(80,260,"5 min Max
 8 kHz   ");
 outtextxy(80,162,"CURRENT MODE:");
 setcolor(RED);
 outtextxy(200,162,"Enter the required mode : ");
 setcolor(BLACK);
 rectangle(53,3,548,60);
rectangle(70,100,160,130);
rectangle(190,100,280,130);
rectangle(310,100,400,130);
rectangle(430,100,520,130);
rectangle(70,140,520,190);
rectangle(70,200,150,430);
rectangle(440,200,520,430);
rectangle(160,200,430,430);
arrow_right(230,125);
setfillstyle(SOLID_FILL,8);
 bar(160,200,430,430);
 bar(347,107,363,123);
 record_mode();
 circle(475,115,8);
 setcolor(RED);
 rectangle(51,1,550,450);
 rectangle(52,2,549,449);
 settextstyle(TRIPLEX_FONT,HORIZ_DIR,3);
 outtextxy(120,20,"DIGITAL AUDIO CARD SOFTWARE");
 sleep(5);
 return;
 }

 void arrow_left(int x1,int y1)
    {
    int i;
    for(i=16; i>=0; i-=2)
     {
     setcolor(8);
     moveto(x1,y1);
     lineto(x1,y1-i);
     x1-=2;
     y1-=1;
     }
    return;
    }

 void arrow_right(int xr,int yr)
    {
    int i;
    for(i=16; i>=0; i-=2)
    {
    setcolor(8);
    moveto(xr,yr);
```

```
        lineto(xr,yr-i);
        xr+=2;
        yr-=1;
        }
    return;
    }

void clear_mode(void)
 {
 setfillstyle(SOLID_FILL,7);
 bar(182,142,518,188);
 return;
 }
void record_mode(void)
  {
  setfillstyle(SOLID_FILL,8);
  bar(107,111,123,123);
  bar(107,107,123,109);
  return;
  }
```