Programmable Narrowband Acoustic Analyzer

PROJECT WORK

SUBMITTED BY:

Divakar P.C.

Padmakumar N.

Raja Ganapathy M.

Santhosh Unni

Under the Guidance of: Ms. A. Vasuki. M.E.,

Submitted in partial fulfilment of the requirements for the award of the Degree of

Bachelor of Engineering

***in Electronics and Communication Engineering** of the*

***Bharathiar University**, Coimbatore.*

Department of Electronics and Communication Engineering

Kumaraguru College of Technology

Coimbatore - 641 006

1995-96

Kumaraguru College of Technology

Coimbatore - 641 006.

Department of Electronics and Communication Engineering

Certificate

This is to Certify that the Report entitled

"Programmable Narrowband Acoustic Analyzer"

has been submitted by

Mr...

in partial fulfilment of the requirements for the award of Degree of Bachelor of Engineering in the Electronics and Communication Engineering Branch of the Bharathiar University, Coimbatore - 641 006 during the academic year 1995 - 96.

26/3/96
(Guide)

26/3
(Head of Department)

Certified that the Candidate was Examined by us in the Project Work Viva-Voce Examination held on ____ . ______ and the University Register Number is ______________

9/4
(Internal Examiner)

(External Examiner)

Cable : PROPULSION
Telex : 0435-242 LPSC IN
Fax : 72686
भारत सरकार
अन्तरिक्ष विभाग
द्रव नोदन प्रणाली केन्द्र
वलियमला
तिरुवनन्तपुरम - 695 547

सत्यमेव जयते

Telephone:

GOVERNMENT OF INDIA
DEPARTMENT OF SPACE
LIQUID PROPULSION SYSTEMS CENTRE
VALIAMALA P. O.
TRIVANDRUM - 695 547, INDIA.

CERTIFICATE

This is to certify that this is a bonafide record of the project work on PROGRAMMABLE NARROWBAND ACOUSTIC ANALYZER" done by the following four students of KUMARAGURU COLLEGE OF TECHNOLOGY ,COIMBATORE ,

1. DIVAKAR P.C 2. PADMAKUMAR N
3. RAJAGANAPATHY M 4. SANTHOSH UNNI

in partial fulfilment of the requirements of Bachelor of Engineering (Electronics & Communication) , of BHARATHIAR UNIVERSITY , COIMBATORE at the INDIAN SPACE RESEARCH ORGANISATION (I.S.R.O), VALIAMALA, TRIVANDRUM under my guidance.

S. Usha Devi

S.USHA DEVI
Engr.SE
Electronics Section
CSCG/LPSC
ISRO.

N. Sridharan dhas

Approved by: Sri .N. Sridharan Das,
Group Director
CSCG/LPSC
ISRO.

LIQUID PROPULSION SYSTEMS CENTRE
ISRO
VALIAMALA-695 547

Dedicated to

"Our Beloved Parents"

Acknowledgement

We are eternally indebted to our Head of the Department, Prof. M. Ramasamy M.E.; M.I.S.T.E.; M.I.E.E(USA); M.I.E; C.Eng.(I) and faculty guide Ms.A.Vasuki M.E., for their constant encouragement, critical suggestions and valuable guidance during the course of the project.

We would like to express our gratitude to our beloved Principal Dr.S.Subramanian M.Sc(Engg.); Ph.D; S.M.IEEE,for his kind patronage.

Our heartfelt thanks to Mr.N. Sridharan Das, Group Director,CSCG, LPSC, Mr. Anguswamy, Head-BSD, VSSC, and Mr. Gopakumar, Engineer-in-charge, LPSC for granting us permission to undertake the project at their esteemed organisation.

We also thank our institutional guide Smt. S. Usha Devi, Engr. CSCG, LPSC for her guidance in our work.

We are also solemnly indebted to Mr. A.R.Krishnan, Engr., BSD, VSSC, without whose overwhelming support this project would not have been a reality.

Last but not least, we would like to extend our sincere thanks to all the faculty members of the Department of Electronics and Communication Engineering and our great friends for their strong support and inspiration throughout the development of the project .

Synopsis...

SYNOPSIS

The title " Programmable Narrowband Acoustic Analyzer" clearly manifests the concept of frequency spectrum analysis of acoustic signals by the construction of narrowband filters on a simulator package by suitable programming. Narrowband spectral analysis is the process of discriminating between input frequencies in very small frequency increments across a specified frequency band of interest. To realize the analyzer, the entire audio range is divided into octaves. In each of these octaves, Infinite Impulse Response (IIR) or Finite Impulse Response (FIR) filter are constructed. The response of each of these filters in their respective octaves collectively gives the frequency spectrum analysis of the signal. The filters are designed using standard PC based filter design software and implemented on hardware using Digital Signal Processor namely, ADSP 2100 from M/S Analog Devices Inc. The codes are written using Assembler of ADSP - 2100.

Contents...

CONTENTS

Chapter 1

Introduction

1.INTRODUCTION

An Acoustic analyzer performs the spectral analysis of the signals applied to it. Analysis is achieved by the process of filtering, by which the frequency spectrum of the signal can be modified, reshaped or manipulated according to some desired specification. It may also entail alternating a range of frequency components, rejecting or isolating one specific frequency component etc. Frequency components of very small increments of the signal under test can be resolved with better resolution by the construction of narrowband filter in each of the octaves, that collectively comprises the entire audio range.

Inherent properties of digital filters namely accuracy, reliability, flexibility & sharp cutoff frequencies scores a direct advantage over analog filters. An overview of the DSP fundamentals is dealt in chapter 2.

The Elliptic type of IIR filter is chosen because of the lower order exhibited in design & faster transition from the passband to stopband. Chapter -3 explains the design of the elliptic IIR filter.

Software simulation of a processor is the process of executing programs written in the instruction set of a target processor on a host computer. The simulator goes through the operation cycle of the computer, keeps track of the contents of all registers, flags and memory locations. When implemented as an interactive facility it is the best design tool. The host system is IBM PC Compatible. The programs are created using a standard editor. Debugging facilities like display of registers and memory contents, display of flags are provided. The error messages displayed are user friendly. One such Digital Signal Processor Simulator namely ADSP - 2100 is used in this project work as design & debugging tool before final physical implementation of the filter banks on the ADSP - 2100 Emulator. Chapter 4 briefs on the features & architecture of ADSP - 2100 simulator.

The proper approach and implementation of the filter bank using ADSP - 2100 instruction set is enlisted in Chapter -5.

Chapter 2

Basic Concepts

2.BASIC CONCEPTS

For the successful implementation of the project the knowledgeof the following entailed particulars is a pre-requisite.

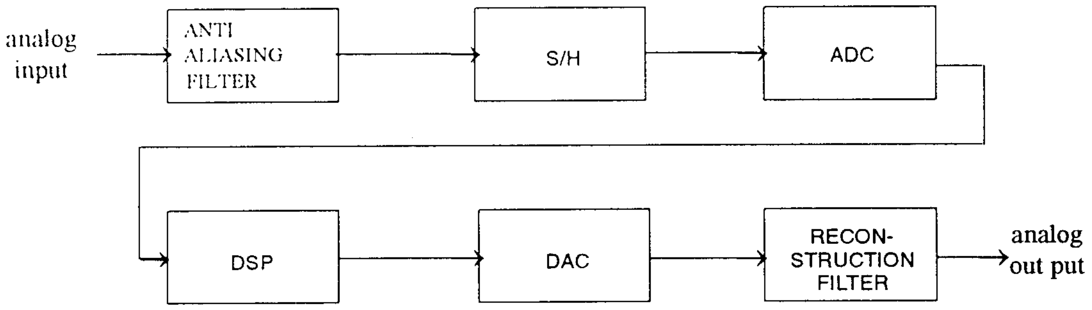
2.1 REVIEW OF DIGITAL SIGNAL PROCESSING:

DSP is any digital operation performed on a digital input sequence. The concept of DSP can be revealed by defining each of the words. Signal is a function of a set of independant variables. The signal carrying some kind of information available for observation. By processing we mean operating in some fashion on a signal to extract some useful information. The word Digital implies that the processing is done with a digital computer or special purpose digital hardware. Unlike data processing DSP has to process with real time analog signals.

Digital processing is stable, predictable and repeatable. There is little degradation due to critical interaction and the basic overall structure remains fixed allowing changes via programming. The design can be simulated using software and can be easily modified, thereby reducing product development time

Another advantage of DSP is the property that the complexity of the processing becomes independent of the hardware. Because of the flexibility of digital computers, it is useful to simulate a signal processing system in a digital computer before implementing it in analog hardware. Thus a new signal processing algorithm or system is evolved.

Block Diagram of a Typical DSP System



The prefilter is designed to reduce the effects of out of band interference signals. Interference signals could be extraneous noise or higher frequency signals that when sampled produce lower frequency signals. The ADC is a device which will, upon command give a binary code word corresponding to the input analog sample. The next stage, viz DSP deals with the processing of the digital output from the ADC. Digital filtering of the ADC samples is undertaken in this project. Digital filter is basically an algorithm that produces an output sequence from the input sequence. The DAC is a device that operates on a sequence of input code words to produce a continuous output signal. The output of the DAC is smoothed by reconstruction filter to produce the desired output.

2.2 DEFINITIONS OF IMPORTANT TERMINOLOGIES

2.21 Octave :-

An Octave is defined as a successive doubling in frequency away from the reference point (eg. 100,200, 400, 800 ....).

2.22 Octave Band :-

An Octave Band is one that has a lower band frequency that is one half the upper band frequency.

2.23 Octaveband Digital filtering :-

Translation of the centre of the frequency band to be analyzed to the base band.This is advantageous as the same approach can be used for multiple processing bands (quadrative demodulation followed by lowpass filters) as the control structure for each band is identical and changes are required only for demodulation frequency and lowpass filter BW.

With well defined Octave bands along with BW and sampling rate proportionally reduced, the filtering strings would all be the same.

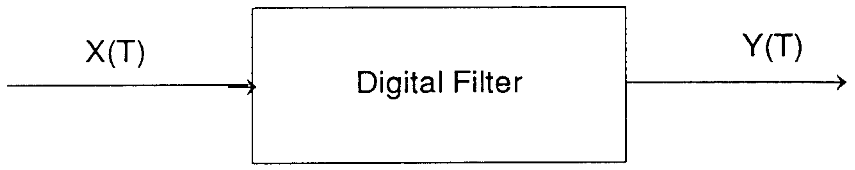
2.24 One -Twelfth Octave Analyzer :-

Each Octave in the spectrum of the signal under consideration (Audio range in this case) is resolved into twelve equal bands. Analysis of the signal components in each of these twelve bands constitute the One-Twelfth Octave Analyzer. Such an analysis is realized by the construction of twelve- narrow band pass filters in each of the twelve bands.

2.3 DIGITAL FILTERS:-

Filtering is a process by which the frequency spectrum of a signal can be modified, reshaped or manipulated according to some desired specifications and it may entail amplifying or attenuating a range of frequency components, rejecting ;or isolating one specific frequency component etc. Filtering is carried out to eliminate contaminations (noise), signal distortion to resolve signals into their components, to band limit signals and to convert discrete signals into continuous signals, as in electro-cardiogram processing, pattern recognition etc.

A digital filter is a digital system that can be used to filter discrete time signals. A band limited continuous time signal can be transformed into discrete time signal ~~can be transformed into discrete time signa~~l by means of sampling (ref. fig.2.3a). Conversely, a discrete time signal so generated can be used to regenerate the original continuous time signal by virtue of sampling theorem. Hence hardware digital filters can be used to perform real time filtering tasks. The input X(T) forms the excitation and the output Y(T) is the response of the filter.



Y(T) = R X(T)

Where R= Transfer function of the Digital Filter.

Fig. 2.3a

The Digital Filter or a discrete time system is characterised by a set of difference equations in time domain which can be transferred to the Z-plane using the Z-transform. The transfer function of a digital filter in the Z-domain can be specified as

$$H(Z) = \frac{Y(Z)}{X(Z)}$$

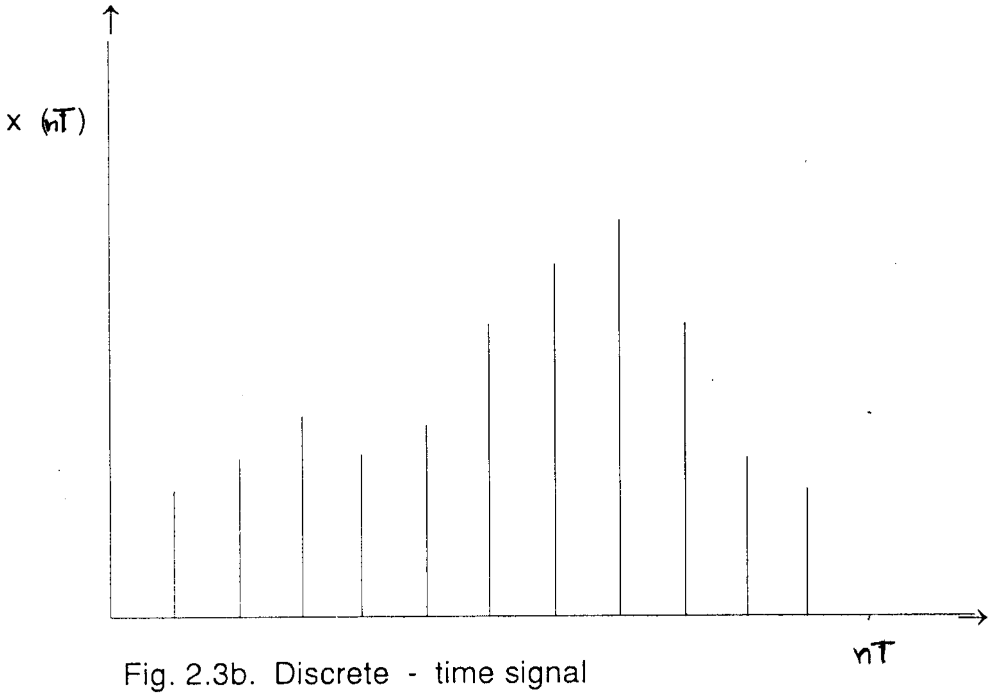


Fig. 2.3b. Discrete - time signal

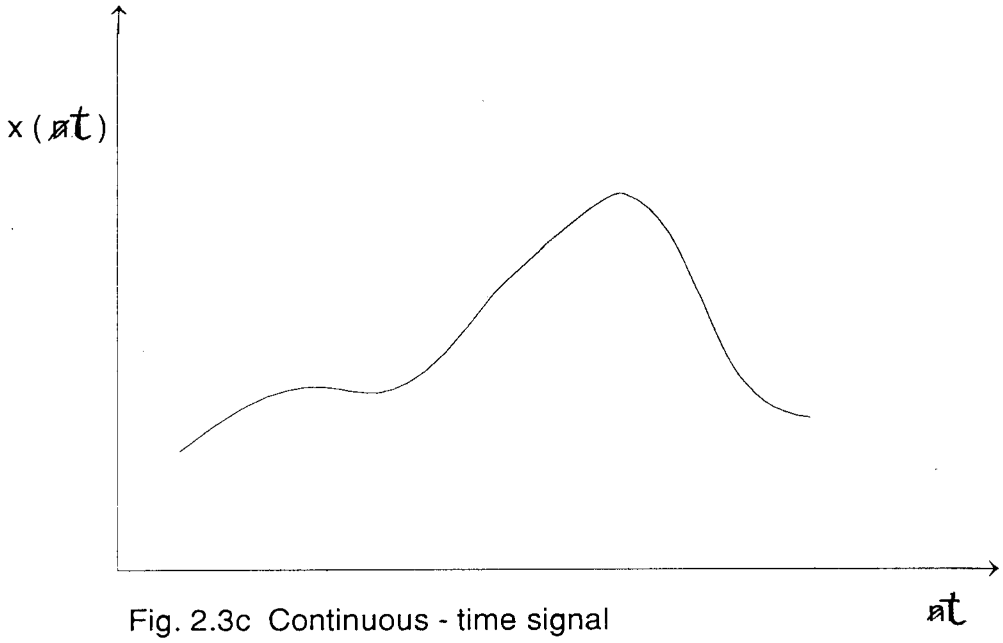


Fig. 2.3c Continuous - time signal

Where X(Z)= Z-Transform of the Input Excitation

Y(Z) = Z-Transform of the Output

$$X(Z) = \sum_{n=-\infty}^{\infty} X(T)\, Z^{-n}$$

where Z is a complex variable

$$= r\, e^{i\theta}$$

2.31 Advantage of Digital Filters :-

The primary advantage is the flexibility exhibited by digital filters as it is easier to change software than to change solder-joints and tracks on a PCB once it is in production. Implementations of standard filter components used in analog filtering viz, resistors have some capacitance and inductance, inductors have some resistance and capacitance & so on could be confined to very lesser limit by digital filters. Digital filters provide a much better performance at very low frequencies unlike size, weight, and cost problems encountered by using very large resistors,capacitors and inductors.

2.32 Basic Digital Filter Types :-

The digital filters are classified according to the difference equation as recursive filters or the Infinite Impulse Response (IIR) filter and the non-recursive filter or the Finite Impulse Response (FIR) filter.

2.32.1 FIR Filters:-

If the unit sample response of a linear shift invariant system is of finite duration, the system is said to be a Finite Impulse Response (FIR) system. FIR filter is also known as non-recursive filter.

For the non-recursive filter the response depends only upon the present input and a finite number of past input samples. The relation between the output and the input sequence is given as $y(n) = F[x(n), x(n-1), ----]$.

If we consider linearity and time invariance, then y(n) is given as

$$y(n) = \sum_{l=0}^{\infty} A_l\, x(n-l)$$

where A_l s are constants.

If causality is taken into account with the criteria as

$$x_1(n) = x_2\,(n) \text{ for } n \leq l$$

$$x_1(n) = x_2(n) \text{ for } n > l$$

$$\text{then } y(n) = \sum_{l=0}^{\infty} A_l\, x(n-l)$$

$$\text{If } x(n) = 0 \text{ for } n < 0 \text{ and bk for } k > n.$$

$$\begin{aligned}\text{then } y(n) &= \sum_{l=0}^{n} A_l\, x(n-l) + \sum_{l=n+1}^{\infty} A_l\, x(n-l)\\ &= \sum_{l=0}^{n} A_l\, x(n-l) + \sum_{l=n+1}^{\infty} A_l\, x(n-l)\\ &= \sum_{l=0}^{n} A_l\, x(n-l).\end{aligned}$$

Thus a linear, causal, time-invariant FIR filter can be represented by nth order linear differential equation where "n" is the order of the filter. This filter is a special kind of IIR filter.

Advantages of FIR filter :-

1. FIR filter with exactly linear phase can be easily designed.
2. Efficient realization of FIR filter exists as both recursive and non-recursive structures.
3. FIR filter realized non-recursively; ie, by direct convolution are always stable.
4. Round off noise can be made small for non-recursive realization of FIR filters.

Disadvantages of FIR filters :-

1. A large value of N, the impulse response duration is required to adequately approximate sharp cut-off filters. Hence large amount of processing is required to realize such filters.

2. Delay of linear phase FIR fliters need not always be an integer number of samples leading to some problems in some processing applications.

Techniques employed to obtain FIR filter from Analog filter

Any one of the following stated methods can be used to obtain an FIR filter from an analog filter.

1. Window method
2. Equiripple method
3. Frequency sampling method
4. Least squares method
5. FFT method

FIR filter can have only direct form of structure.

2.32.2 IIR Filters

If the unit sample response of a linear system is of infinite duration the system is said to be Infinite Impulse Response (IIR) filter. IIR filter is also known as recursive filter.

The output of a casual, linear, time-invariant discrete time system is

$$y(n) = \sum_{l=0}^{m} x(n-l) - \sum_{l=0}^{n} b_l y(n-l)$$

ie. the response of the IIR filter is a function of current and past input samples and past output samples.

Advantages of IIR filter :-

For a given order N, highly selective recursive digital filters can be designed. ie; they are computationally efficient.

Disadvantages of IIR filters :-

Proper attention should be paid to stability, parasitic phenomena and phase non-linearity.

2.33 IIR Filter Approximation :-

Recursive digital filters can be designed using any one of the following analog filter approximations.

- * Butterworth approximation
- * Chebyshev approximation
- * Bessel approximation
- * Elliptic approximation

2.33.1 Butterworth Approximation :-

Basic concepts of the Taylor's series approximation to the desired frequency response is employed here. The measure of approximation is the number of terms in the Taylor's series expansion of the actual frequency response that can be made equal to those of the desired frequency response. The optimal solution will have the maximum number of terms equal.

Taylor's series is a power series expansion of the form

$$F(w) = K_0 + K_1 (w) + K_1 W^2 + ----$$

where

$$K_0 = F(0)$$

$$K_1 = \frac{dF(w)}{dw} \quad w = o$$

$$K_2 = \frac{1}{2}\frac{d^2F(w)}{dw^2} \quad w = o$$

The Taylors series approximation gives a maximally flat passband region.The approximation and normalization is given by

$$F(jw) = \frac{1}{1+w^{2n}}$$

Characteristics of Butterworth filter.

- * Smooth gain response
- * Reasonably good phase response
- * Poor attenuation vs. filter order

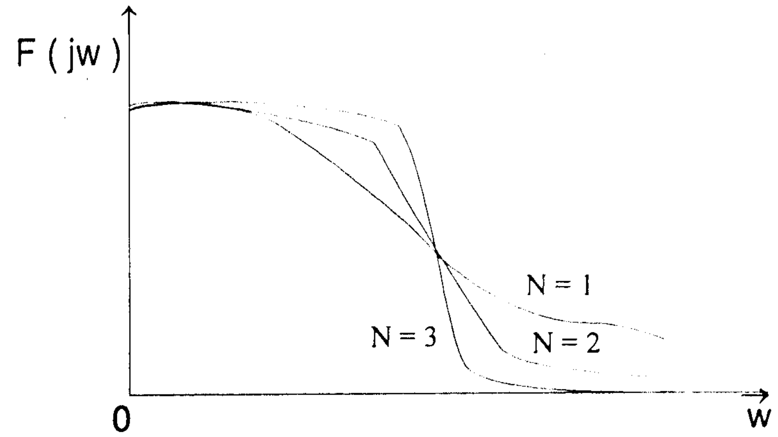


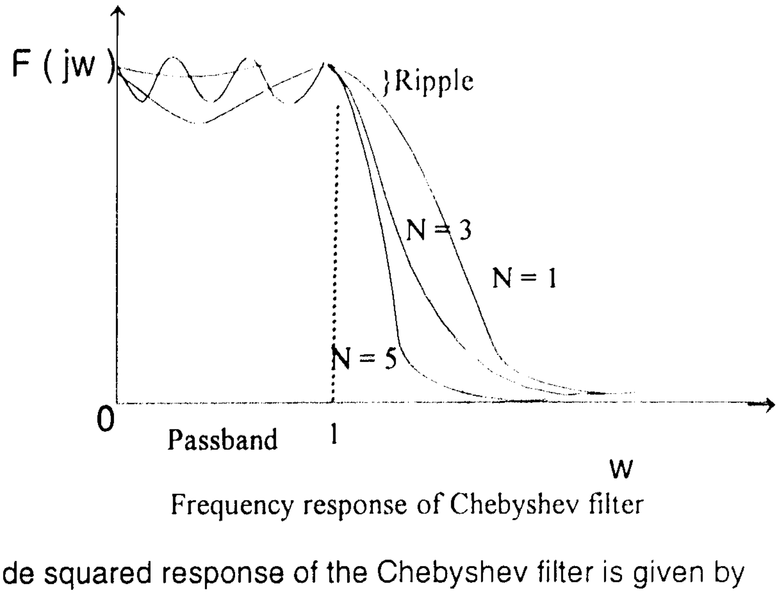
Fig.2.33.1 Freq. response of Butterworth filter

2.33.2 Chebyshev Approximation :-

This method exhibits two forms. Chebyshev Type - 1 Approximation has equiripple behaviour in the passband and Chebyshev Type - 2 Approximation has equiripple behaviour in the stop band.

Chebyshev Type - 1

For the IIR filter the Chebyshev error is minimized over the passband and a Taylor's series approximation at w = ∞



Frequency response of Chebyshev filter

Magnitude squared response of the Chebyshev filter is given by

$$F(jw)^2 = \frac{1}{1 + E^2C^2_N(w)}$$

where $C_N(w)$ is an Nth order Chebyshev polynomial

E = control ripple size.

Characteristics of Chebyshev Type -1 filter :-

- * Ripple in Passband
- * Smooth Stopband

The slope of the response near the transition from passband to stopband at w = 1 becomes steeper as both the order and the allowed passband and error ripples increases.

Chebyshev filter Type - 2 or inverse Chebyshev :-

It has a flat response at w = 0, which is the passband and maximum allowable response in the stopband.

Characteristics

- * Ripple in stopband
- * Smooth passband

The rate of drop off near the transition from passband to stopband is similar to the Chebyshev Type-1 filter.

General characteristics of Chebyshev filters

- * Poor phase response
- * Good attenuation vs filter order

2.33.3 Elliptic Approximation :-

Elliptic approximation is sometimes called as Caver filter or Rational Chebyshev filter. It is a combination of Chebyshev and Taylor's series in the passband and stopband. Elliptic filter usually gives the lowest order filter for a given set of specifications.

Characteristics of Elliptic filter :-

- * Maximum variation or ripple in passband
- * Maximum variation or ripple in stopband
- * Excellent attenuation vs filter order

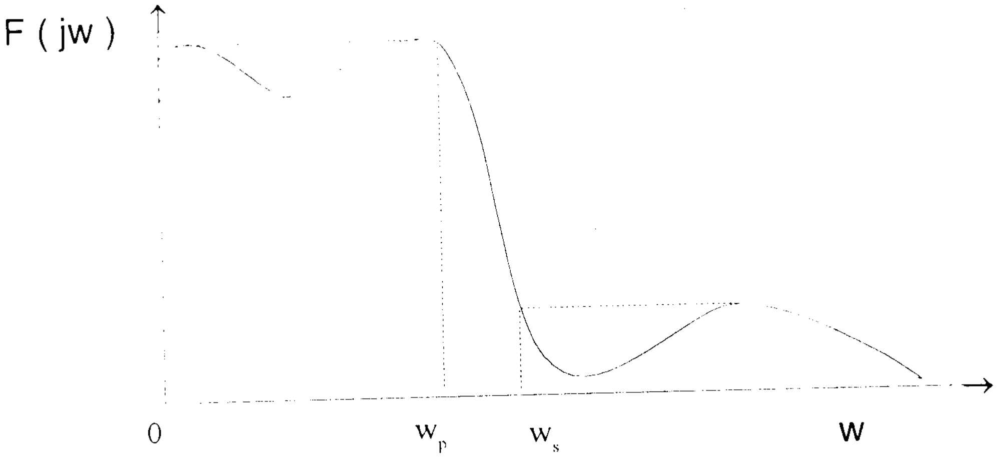


Fig. 2.33.3 Frequency response of Ellptic filter

- * Very poor phase response
- * Optimum choicce when phase response not critical
- * Higher transition rate

2.33.4 Bessel Approximation :-

Bessel filter is a filter whose group delay or the equivalent phase shift is a non-linear function of frequency. It is characterized by a maximally flat group delay. Bessel filters are of the all pole-type.

Characteristics Bessel Filter :-

- * Smooth gain response
- * Good phase response
- * Very poor attenuation vs filter order

2.34 IIR Filter Structure :-

IIR filters can have any one of the following structures :

1. Parallel
2. Cascade
3. Lattice
4. Direct form

Gain scaling is necessary in the case of IIR filters while it is usually unnecessary in the case of FIR filter.

2.35. Choice of IIR type of Filter :-

IIR filters are the best choice when non-linear phase characteristics can be tolerated ie, when only magnitude characteristics is of interest. Hence in the design of the Narrowband Acoustic Analyzer IIR type of filter is chosen as only the magnitude response rather than the phase response is of prime importance.

2.36. Techniques Employed to Obtain IIR filter from Analog Filter :-

Any one of the following enlisted methods can be used to obtain IIR filter from analog filter.

1. Bilinear Transformation
2. Impulse Invariant transformation
3. Least Squares transformation
4. Direct Method

2.36.1 Choice of Bilinear Transformation method in Design :-

The bilinear transformation method is chosen as it is very convenient when the filter specification is one of the four standard types of frequency selective responses. The discrete transfer function Hd(Z) will have the same characteristics as an analog prototype transfer function Hp(S') ; ie, it will have the same passband ripple and stopband attenuation, the only difference being the frequency at which a feature in the response occurs. This is because the continuous system frequency response spans the range Zero to Infinity,whereas in the discrete filter the same magnitude variation must fit into the span Zero to Fs/2 This is termed as Frequency Warping.

The steps involved in the design of Hd(Z) using the bilinear transformation method are as follows :-

1. Specify Hc(S) in the form of a mask
2. Derive the mask for Hd(Z)

3. Find the equivalent mask for Hp(S')

4. Find the poles and zerosof Hp(S') using one of the standard approximations (Elliptic in this case)

5. Derive Hd(Z) from Hp(S')

Chapter 3.

Design of Digital Filters

3.DESIGN OF DIGITAL FILTER

3.1 CHOICE OF ELLIPTIC TYPE OF IIR FILTER

Filter properties like excellent attenuation vs. filter order and the optimum choice when the phase response is not critical paved the way for the selection of Elliptic type of IIR filter in direct comparison with the filter properties exhibited by Butterworth and Chebyshev type of filters.

The transfer function in the Z - domain of a second order biquad IIR filter is

$$H(Z) = \frac{Y(Z)}{X(Z)}$$

$$= (B_0 + B_1Z^{-1} + B_2Z^{-2}) / (1 + A_1Z^{-1} + A_2Z^{-2})$$

where A_1, A_2, B_0, B_1 and B_2 are the coefficients that determine the desired impulse response of the system H (Z).

The corresponding difference equation for a biquad section is

$$Y(n) = B_0X(n) + B_1X(n-1) + B_2X(n-2) - A_1Y(n-1) - A_2Y(n-2).$$

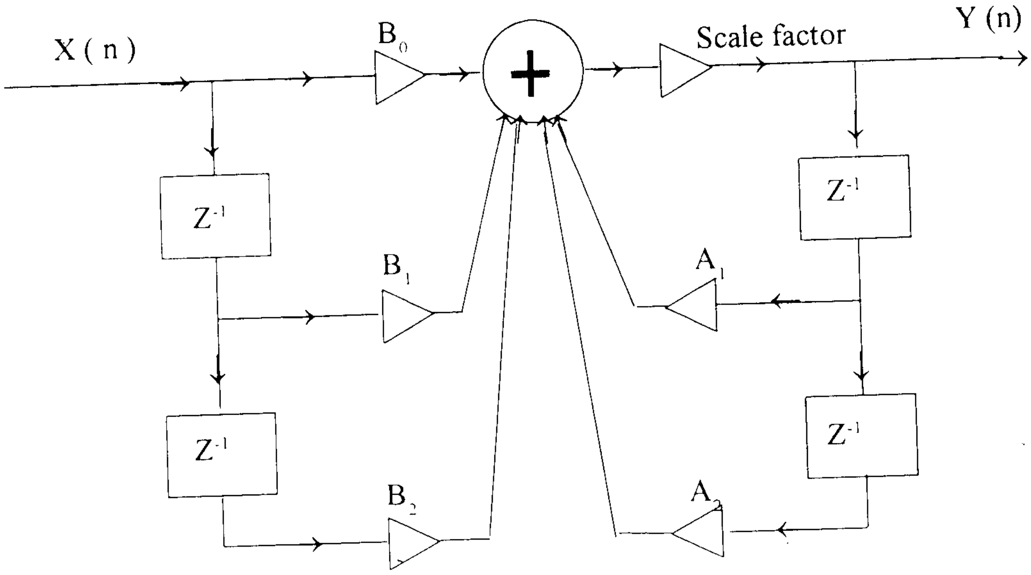


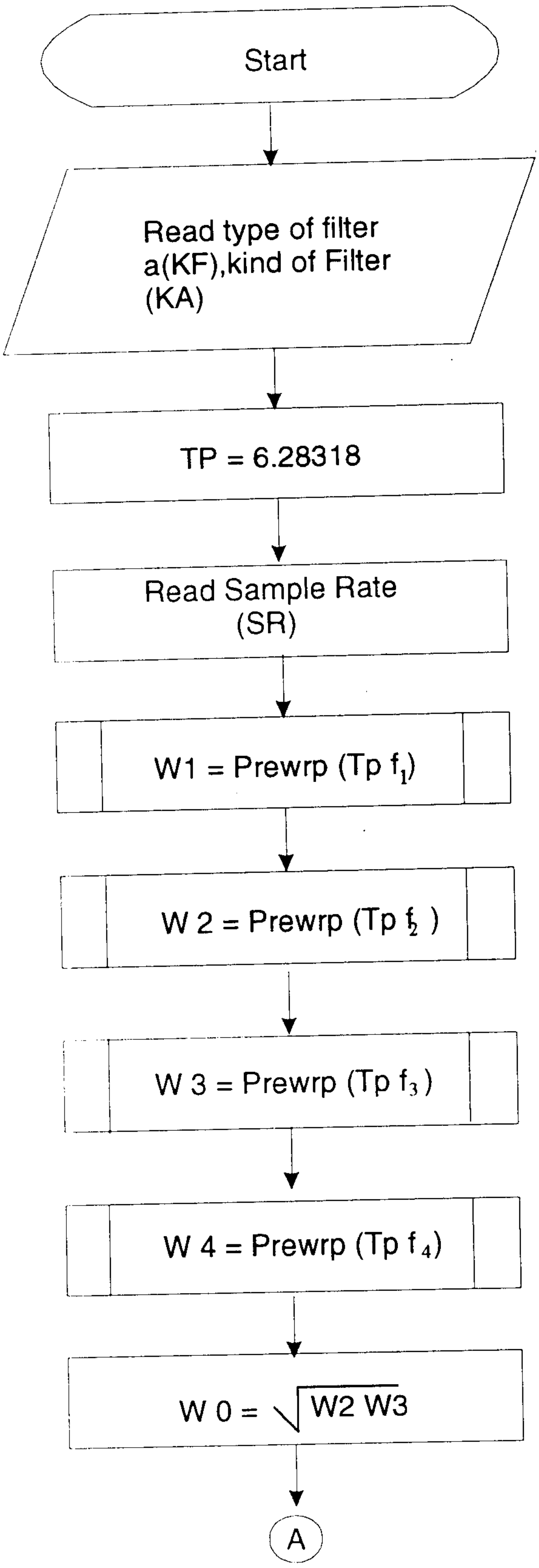
Fig.3.1a

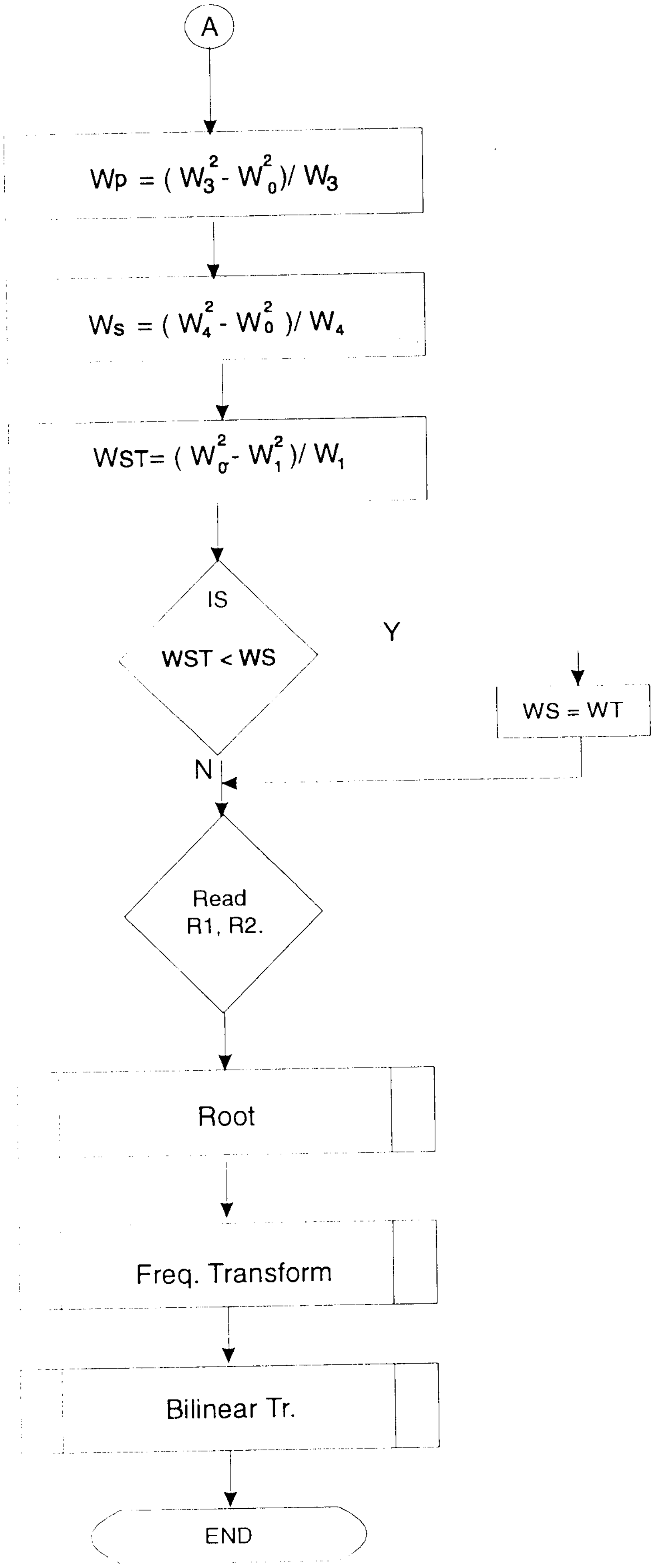
3.2 INPUT SPECIFICATION FOR THE DESIGN OF THE FILTER:-

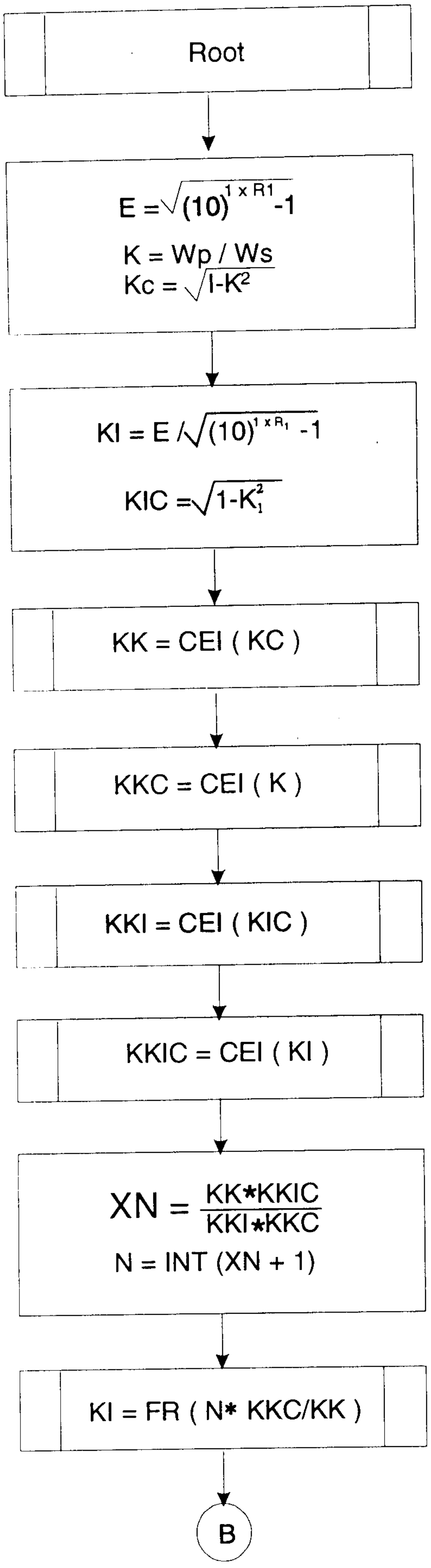
The various input specifications that are to be furnished to obtain the filter coefficients are : -

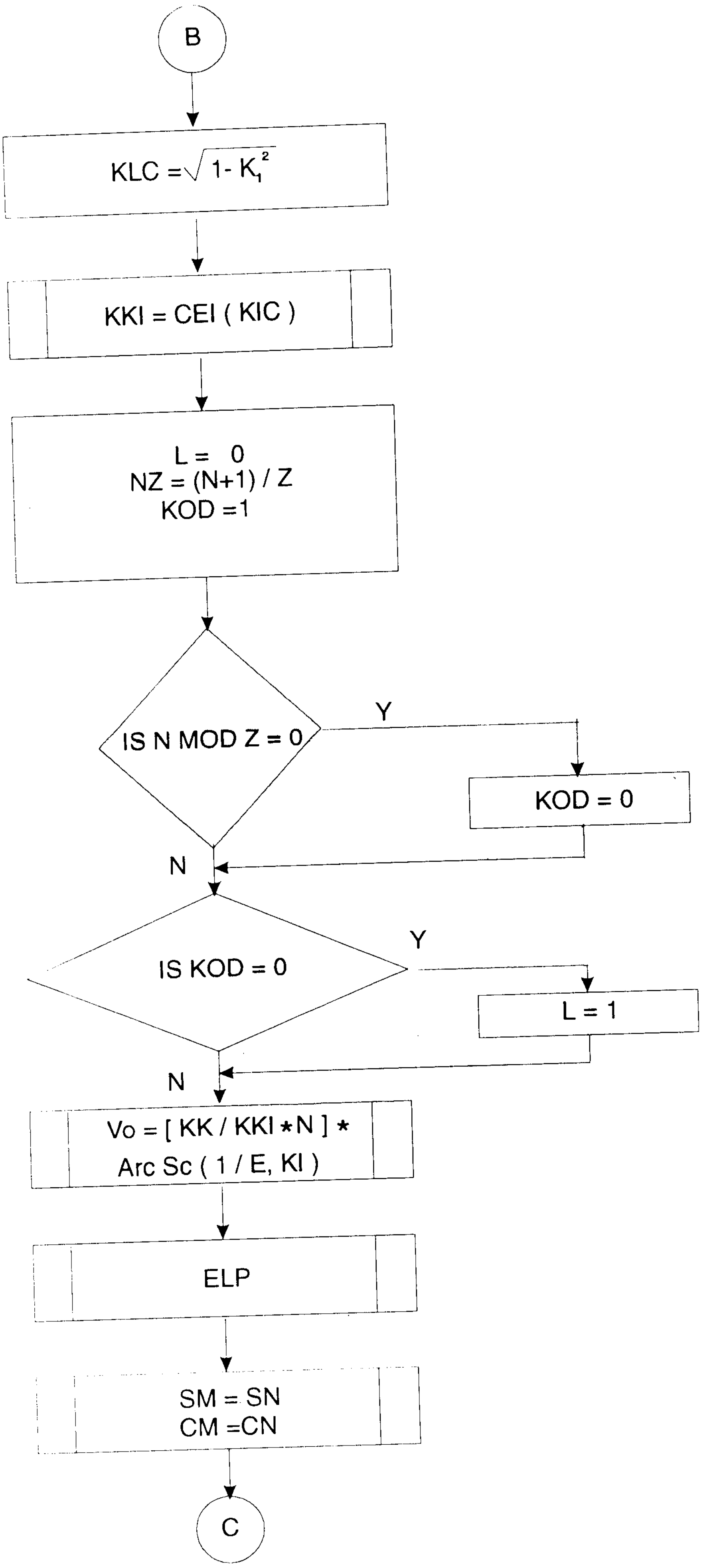
1. Passband Ripple in dB
2. Stopband Ripple in dB
3. Passband Cutoff Frequencies
4. Stopband Cutoff Frequencies
5. Sampling Frequency
6. Filter Structure
7. Filter Order desired.

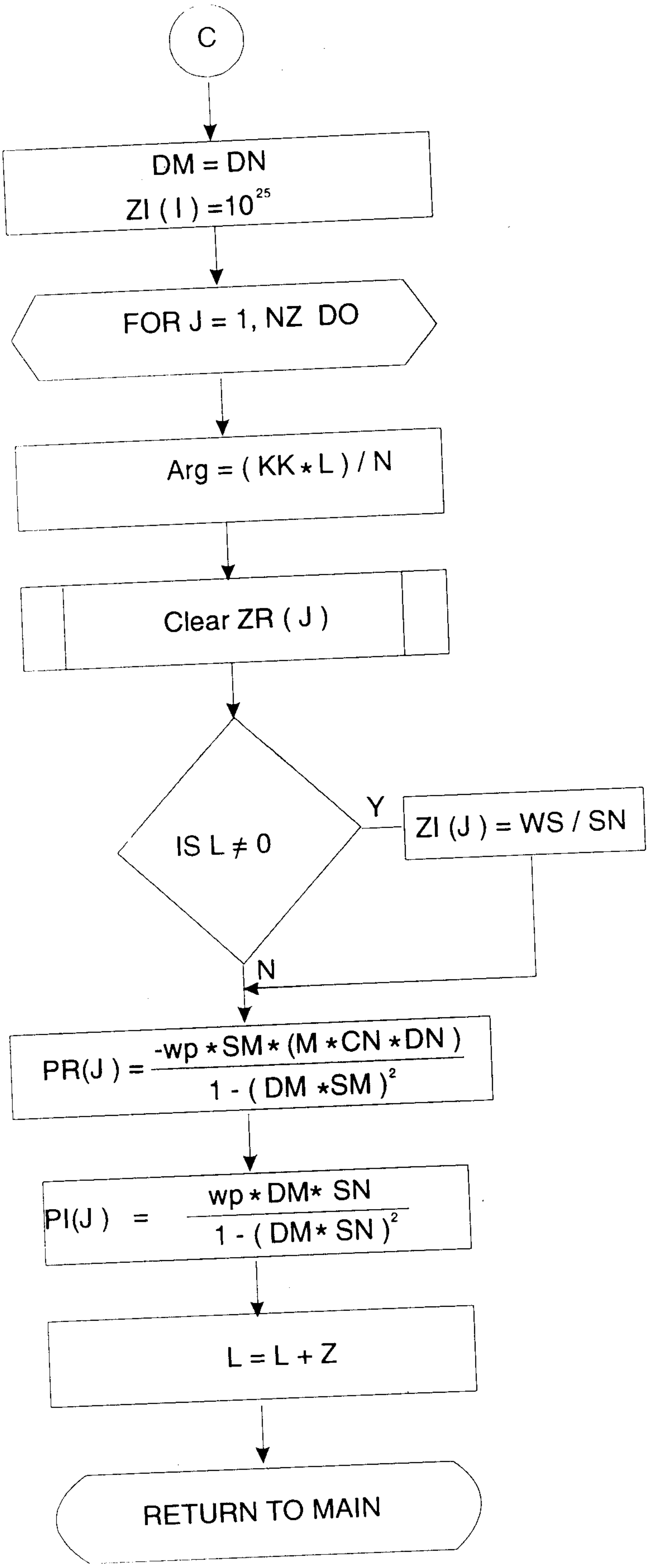
3.3 Design of Elliptic Type IIR Filter- Flowchart

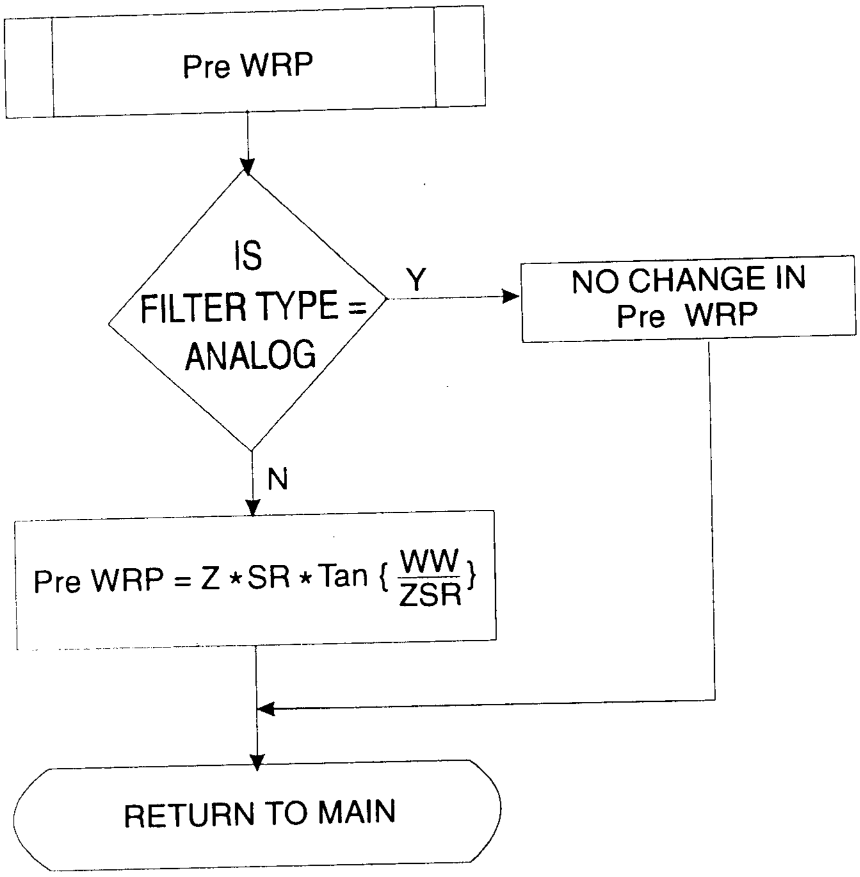


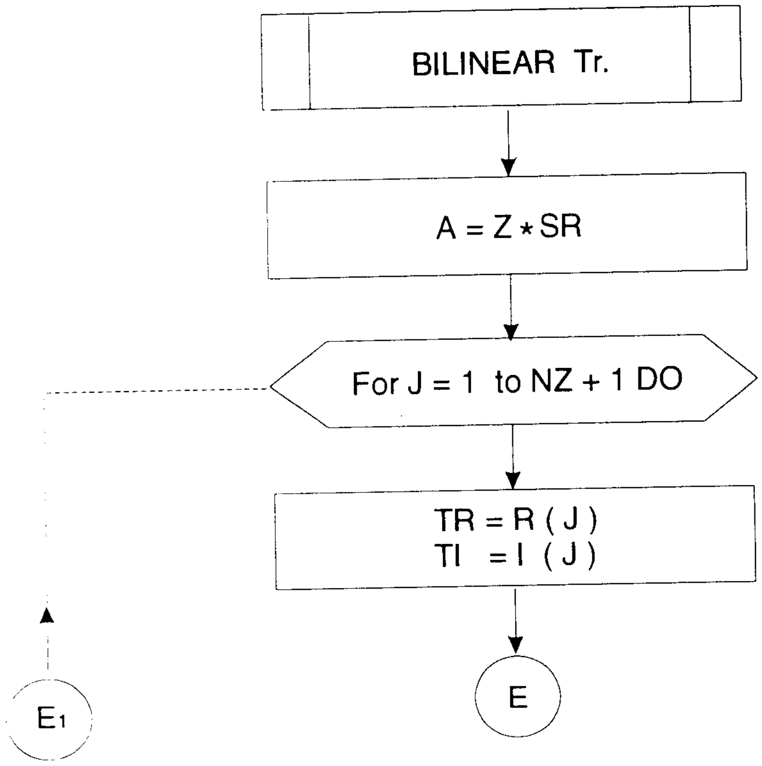

A
$Wp = (W_3^2 - W_0^2)/ W_3$
$Ws = (W_4^2 - W_0^2)/ W_4$
$WST = (W_\sigma^2 - W_1^2)/ W_1$
IS
WST < WS
Y
WS = WT
N
Read
R1, R2.
Root
Freq. Transform
Bilinear Tr.
END

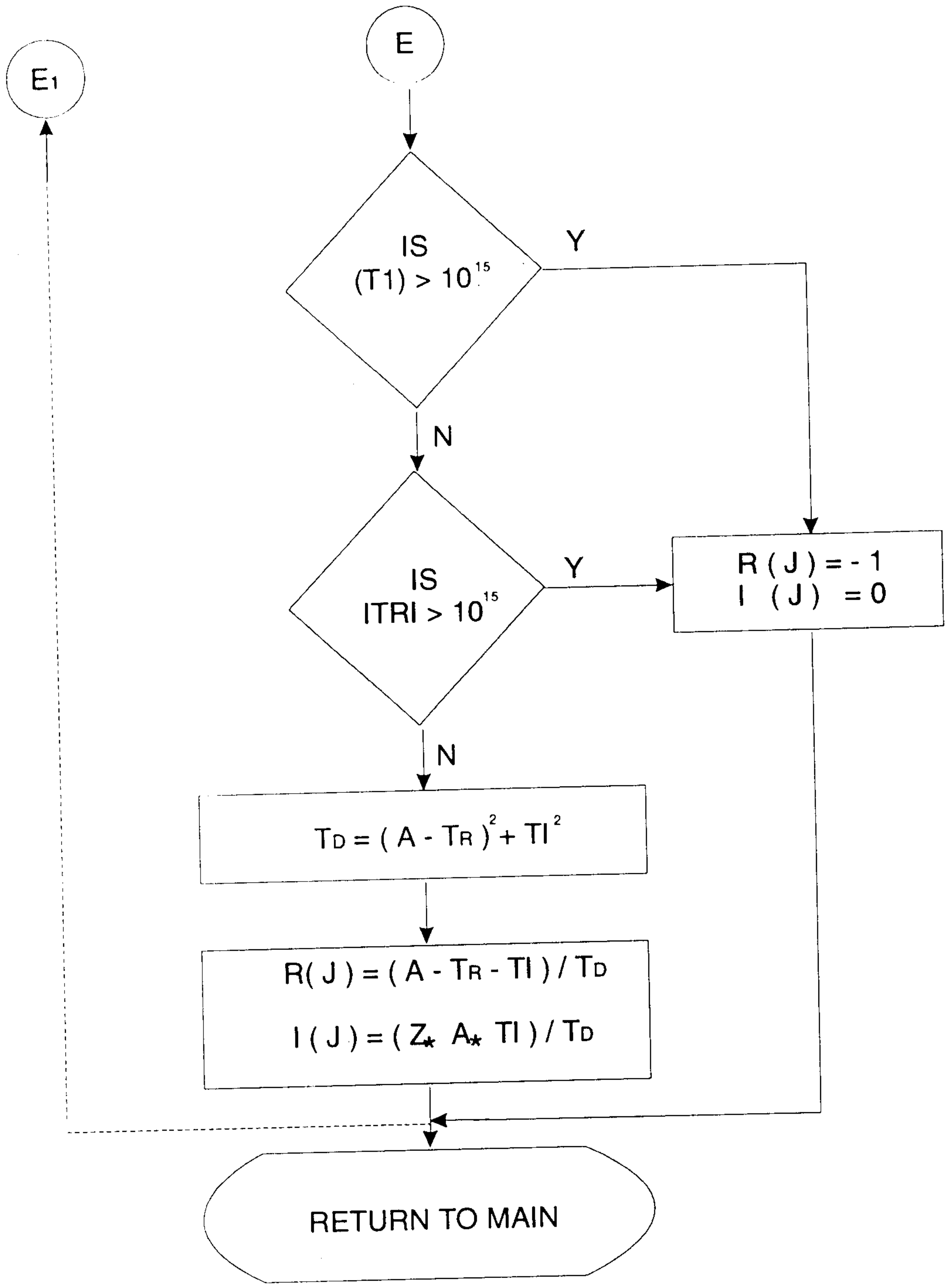

Root
$E = \sqrt{(10)^{1 \times R1} - 1}$
$K = Wp / Ws$
$Kc = \sqrt{l - K^2}$
$Kl = E / \sqrt{(10)^{1 \times R_1} - 1}$
$KlC = \sqrt{1 - K_1^2}$
KK = CEl (KC)
KKC = CEl (K)
KKl = CEl (KlC)
KKlC = CEl (Kl)
$XN = \frac{KK*KKlC}{KKl*KKC}$
N = INT (XN + 1)
Kl = FR (N* KKC/KK)
B

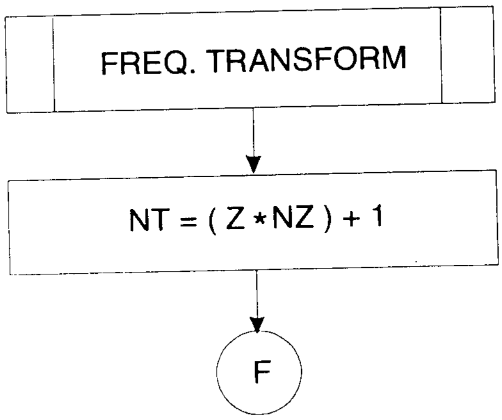

B
KLC = $\sqrt{1- K_1^2}$
KKI = CEI (KIC)
L = 0
NZ = (N+1) / Z
KOD =1
IS N MOD Z = 0
Y
KOD = 0
N
IS KOD = 0
Y
L = 1
N
Vo = [KK / KKI * N] *
Arc Sc (1 / E, KI)
ELP
SM = SN
CM =CN
C

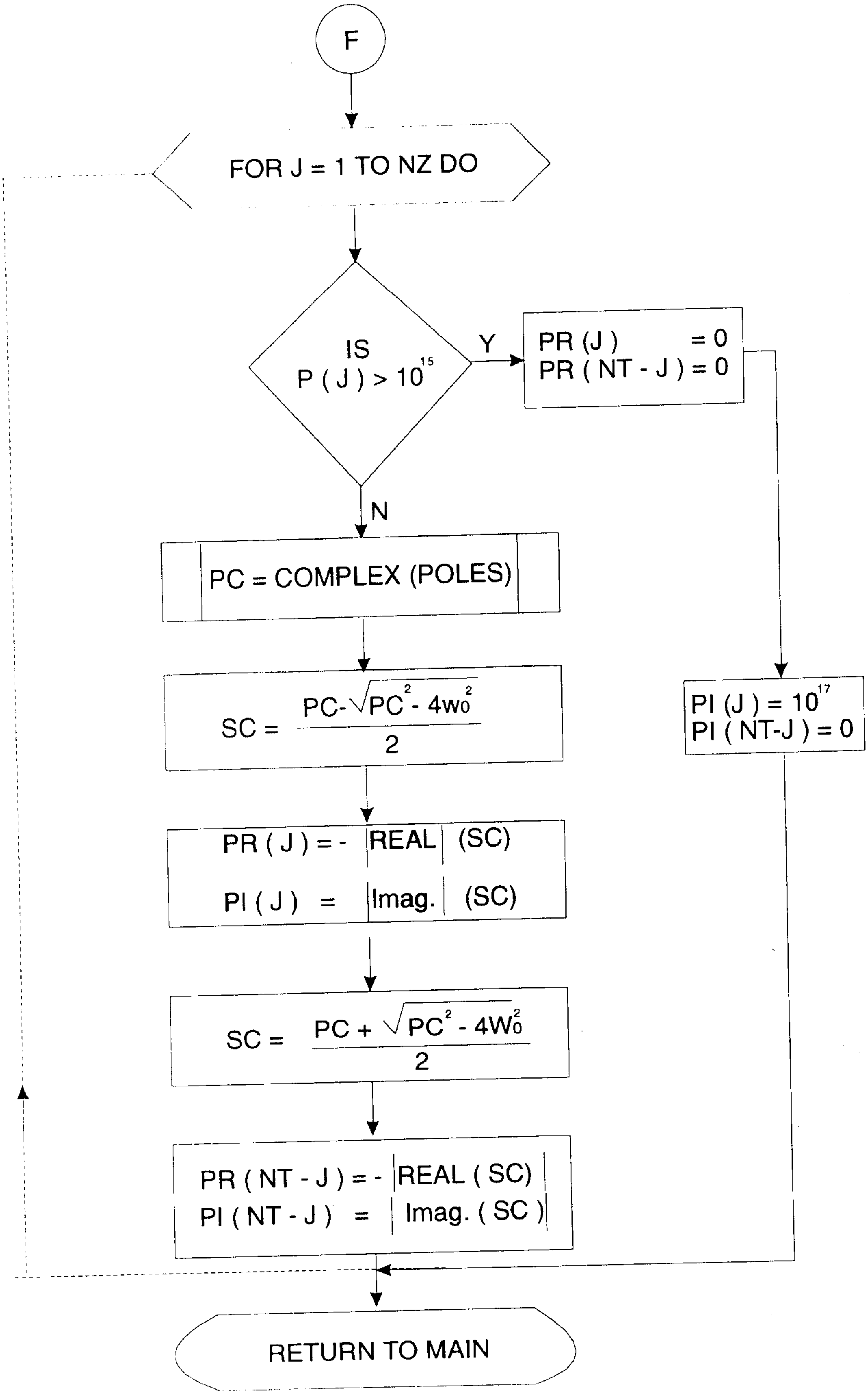

C
DM = DN
ZI (I) =10^{25}
FOR J = 1, NZ DO
Arg = (KK * L) / N
Clear ZR (J)
IS L ≠ 0
Y
ZI (J) = WS / SN
N
PR(J) = $\frac{-wp * SM * (M * CN * DN)}{1 - (DM * SM)^2}$
PI(J) = $\frac{wp * DM * SN}{1 - (DM * SN)^2}$
L = L + Z
RETURN TO MAIN

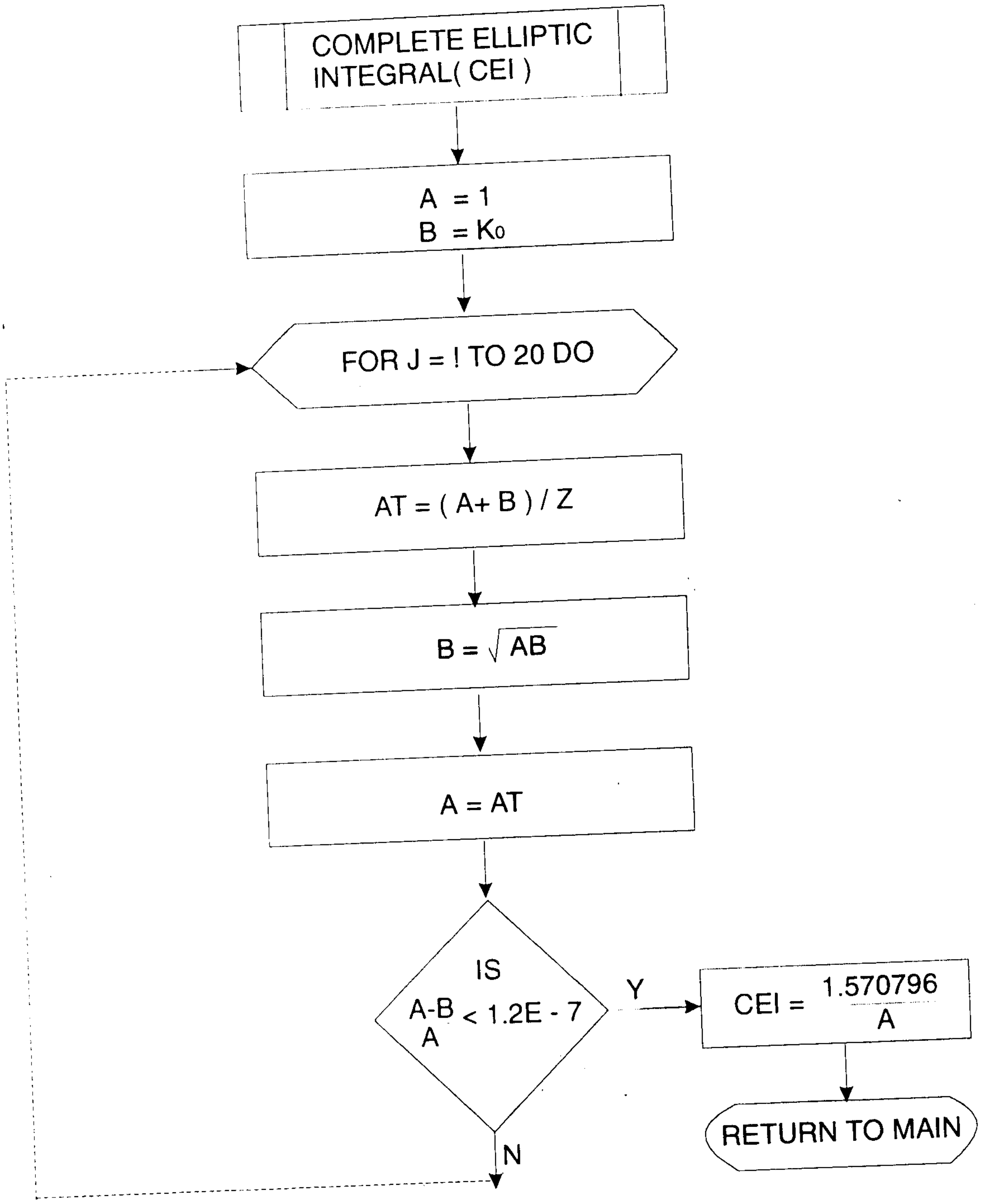

Pre WRP
IS FILTER TYPE = ANALOG
Y
NO CHANGE IN Pre WRP
N
Pre WRP = Z * SR * Tan { $\frac{WW}{ZSR}$ }
RETURN TO MAIN

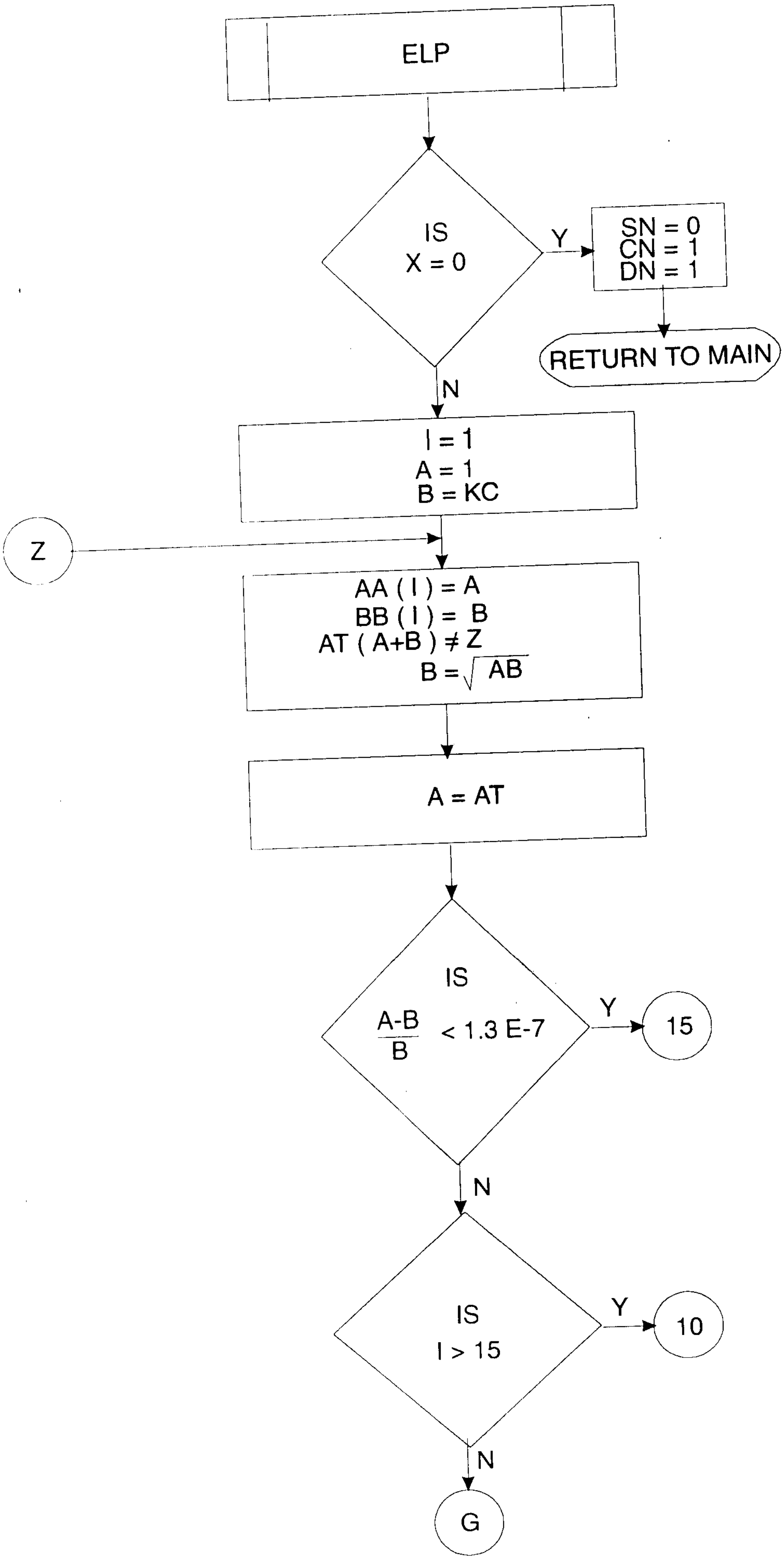

BILINEAR Tr.
A = Z * SR
For J = 1 to NZ + 1 DO
TR = R (J)
TI = I (J)
E
E1

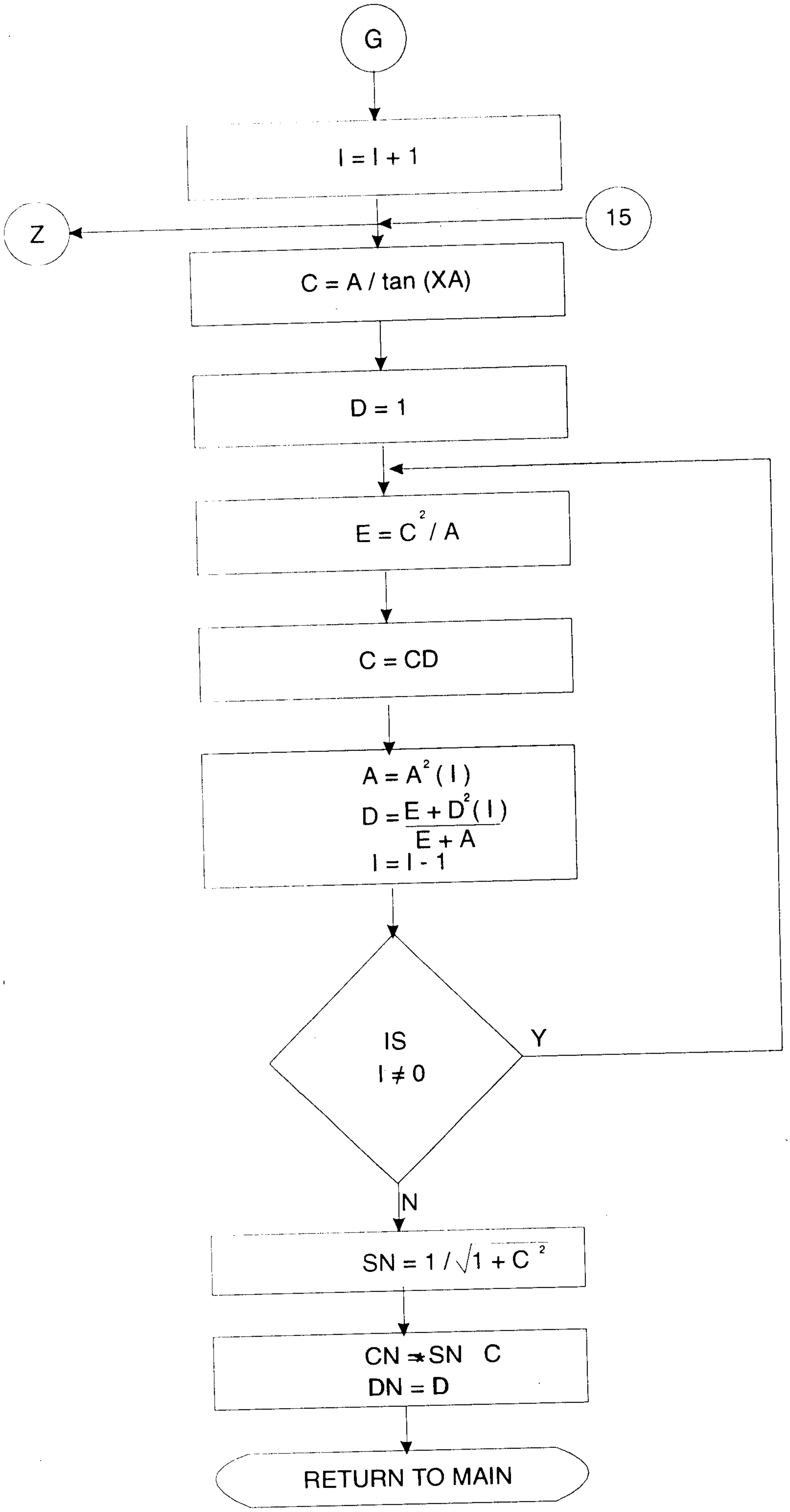

E
E1
IS
(T1) > 10^15
Y
N
IS
ITRI > 10^15
Y
R (J) = - 1
I (J) = 0
N
TD = (A - TR)^2 + TI^2
R(J) = (A - TR - TI) / TD
I (J) = (Z* A* TI) / TD
RETURN TO MAIN


FREQ. TRANSFORM
NT = (Z * NZ) + 1
F

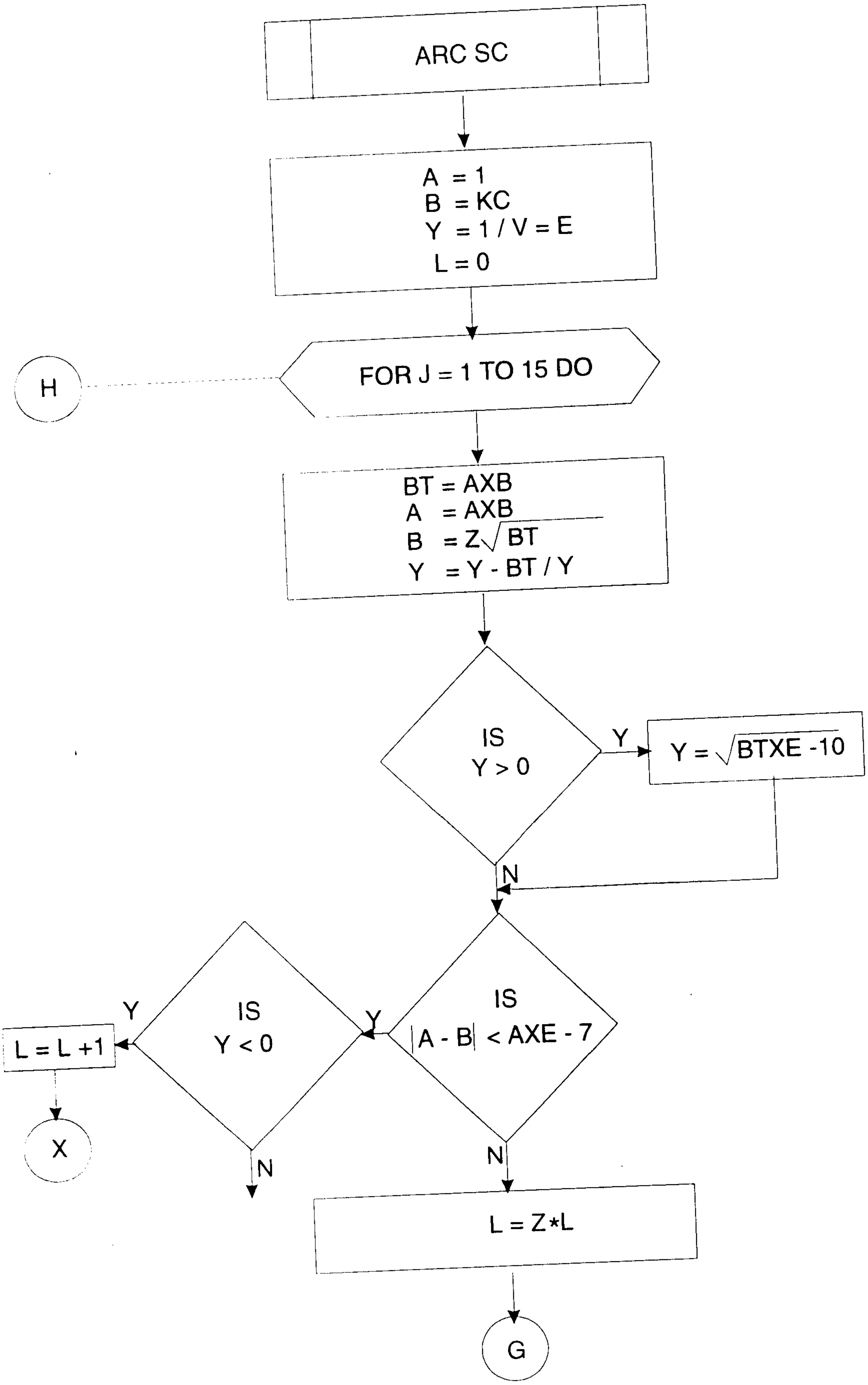

F
FOR J = 1 TO NZ DO
IS P (J) > 10^{15}
Y
N
PR (J) = 0
PR (NT - J) = 0
PC = COMPLEX (POLES)
SC = $\frac{PC-\sqrt{PC^2 - 4w_0^2}}{2}$
PI (J) = 10^{17}
PI (NT-J) = 0
PR (J) = - |REAL| (SC)
PI (J) = |Imag.| (SC)
SC = $\frac{PC + \sqrt{PC^2 - 4W_0^2}}{2}$
PR (NT - J) = - |REAL (SC)|
PI (NT - J) = | Imag. (SC)|
RETURN TO MAIN

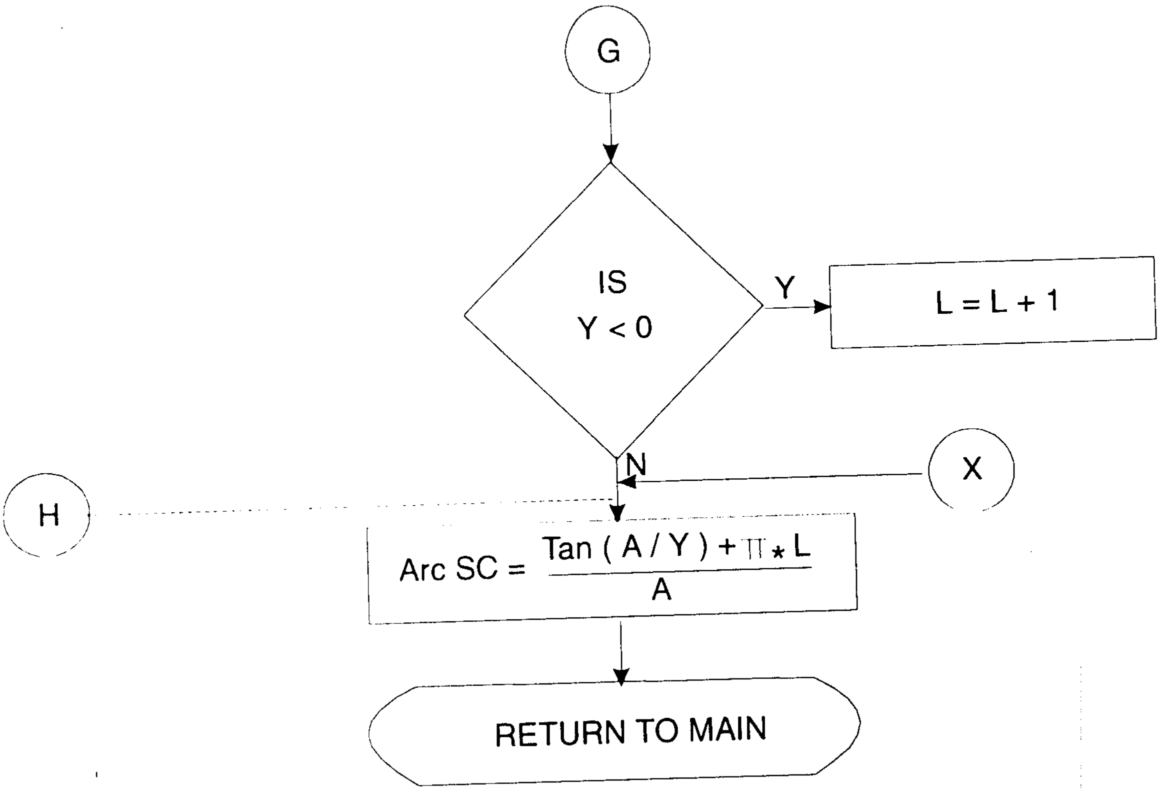

COMPLETE ELLIPTIC
INTEGRAL(CEI)
A = 1
B = K0
FOR J = ! TO 20 DO
AT = (A+ B) / Z
B = √ AB
A = AT
IS
A-B / A < 1.2E - 7
Y
N
CEI = 1.570796 / A
RETURN TO MAIN


ELP
IS
X = 0
Y
SN = 0
CN = 1
DN = 1
RETURN TO MAIN
N
I = 1
A = 1
B = KC
Z
AA (I) = A
BB (I) = B
AT (A+B) ≠ Z
B = √ AB
A = AT
IS
A-B / B < 1.3 E-7
Y
15
N
IS
I > 15
Y
10
N
G


G
I = I + 1
15
Z
C = A / tan (XA)
D = 1
$E = C^2 / A$
C = CD
$A = A^2 (I)$
$D = \frac{E + D^2(I)}{E + A}$
I = I - 1
IS
I ≠ 0
Y
N
$SN = 1 / \sqrt{1 + C^2}$
CN = SN C
DN = D
RETURN TO MAIN

p


ARC SC
A = 1
B = KC
Y = 1 / V = E
L = 0
H
FOR J = 1 TO 15 DO
BT = AXB
A = AXB
B = Z√BT
Y = Y - BT / Y
IS
Y > 0
Y
Y = √BTXE -10
N
IS
|A - B| < AXE - 7
Y
IS
Y < 0
Y
L = L +1
X
N
N
L = Z*L
G


G
IS
Y < 0
Y
L = L + 1
N
X
H
Arc SC = Tan (A / Y) + π * L / A
RETURN TO MAIN

For the digital filter, bilinear transformation is used for the conversion from analog to digital type and the sampling rate is entered. The passband and the stopband edges are entered in Hz. The passband ripple is the total difference between the maximum and minimum frequency responses over the passband. The band edges are converted from Hz to radian. The frequencies are prewarped by the formula

$$V_0 = (2 / T) \tan (w_0 T / 2)$$

This function is performed by Prewarp Subroutine. The pole and zero locations are calculated ; ie,Subroutine Root.The parameter E in the equation

$$F(jw)^2 = \sqrt{\frac{1}{1 + E^2 g^2(w)}}$$ is calculated by

$$E = \sqrt{2S_1 - S_1^2 / 1 - 2S_1 + S_1^2}$$

$$= \sqrt{10^{a/10} - 1}$$

where S_1 = passband ripple

a = passband ripple in dB

The order N is next determined from

$$N > \frac{KK_2'}{K'K_1}$$

that which requires the calculation of fair elliptic integrals .

Modulus K is calculated from the equation

$$K_1' = \sqrt{1 - K_1^2}$$

$$K' = \sqrt{1 - K^2}$$

The complementary Modulus K' is calculated from K by the equation

$$K = Wp / Ws$$

The second Modulus K1 is defined as

$$G(w) = Sn \{ nS_n^{-1} (W,R), R1 \}$$

The complete elliptic integrals are calculated in the function CEI( ). The order is calculated from the equation $N \geq KK2' / K'K2$ which calculates the new value of K1 to satisfy the equation $N = KK1' / K'K1$. This calculation is done in the functionFK ( ) by using the ratio of the power series expansions. V_0 is calculated from the input specifications using the equation

$V_0 \quad = \quad K / NK_1 Sc^{-1} (\frac{1}{E}, K1)$and requires an inverse elliptic target function which is calculated in ARCSC ( ). The elliptical sin. cos and dn functions are calculated in the subroutine ELP ( ). The zero locations are calculated from

$$Wzi = \frac{\pm 1}{KSn\ (iK / \ N, K}$$

and pole location from

$$Spi = \frac{Cn\ dn\ Sn'\ Cn' + j\ Sn\ dn'}{1 - dn^2\ (\ Sn'\ )^2}$$

where $Sn \quad = \quad Sn\ [\ (Ki, N)\ K]$

$Cn \quad = \quad Cn\ [\ (Ki / N), K\ ]$

$dn \quad = \quad dn\ [\ (Ki / N), K\ ]$

$Sn' \quad = \quad Sn\ (\ V_0, K'\ )$

$Cn' \quad = \quad Cn\ (V_0, K'\ )$

$dn' \quad = \quad dn\ (V_0, K'\ )$

These are scaled by WP to give the proper passband and stop-band edges. Frequency transformation of $S = \frac{P \pm \sqrt{P^2 - 4W_0^2}}{Z}$ is carried out in the subroutine FRIQXFm ( ). The root location for the pre-warped prototype analog filters are transferred to the digital filter root location by the bilinear transformation. Such a transformation is carried out in the subroutine BLT ( ) which uses the equation $S = \left(\frac{Z}{T}\right)\left(\frac{Z-1}{Z+1}\right)$

Chapter 4

ADSP - 2100 System

4. ADSP - 2100 SYSTEM

4.1 Introduction

ADSP - 2100 is a programmable single chip microprocessor optimised for Digital Signal Processing and other high-speed numeric processing applications. The ADSP - 2100 development system is a complete set of software and hardware development tools. Cross software system aids the software design and the real time hardware emulator to facilitate the debug cycle. Cross software system runs on the IBM PC under PC DOS.

The emulator provides the same basic information as the simulator but is based on working hardware. The emulator may uncover software errors as well as hardware timing problems. After debugging with the simulator, the emulator may be used with the target system and its own on-board program memory.After debugging with emulator and simulator the PROM splitter translates the excecutable image file (Linker output) into a file that is compatible with a PROM burner. The required system is developed by burning the ADSP-2100 code into the PROM.

The ADSP-2100 development flowchart is shown in fig.4.1a.

4.2 ADSP - 2100 ARCHITECTURE

The ADSP - 2100 processor contains three full-function and independent computational units : namely an arithmetic / logic unit, a multiplier / accumulator and a barrel shifter. The computational unit processes 16 - bit data directly and provides for multiprecision computation. (fig. 4.2a)

Two dedicated data address generators and a complete program sequencer supply addresses. The sequencer supports single - cycle conditional branching and executes program loops with zero overhead.

Dual address generators allow the processor to output simultaneous addresses for dual operand fetches. Together the sequencer and data address generators allow computational operations to execute with maximum efficiency. The ADSP-2100 family uses a modified Harvard architecture in which data memory stores data, and proogram memory stores both instructions and data. Able to store data in both program and data memory, ADSP - 2100 processors are capable of fetching two operands on the same instruction cycle.

The internal components are supported by five internal buses.

- Program Memory Address (PMA) bus
- Program Memory Data (PMD) bus
- Data memory Address (DMA) bus
- Data Memory Data (DMD) bus
- Result (R) bus (which interconnects the computational units)

On the ADSP -2100, the four memory busses are extended off-chip for direct connection to external memories.

The program memory data (PMD) bus serves primarily to transfer instructions from off-chip memory to the internal instruction register. Instructions are fetched and loaded into the instruction register during one processor cycle; they execute during the following cycle while the next instruction is being fetched. The instruction register introduces a single level of pipelining in the program flow. Instructions loaded into the instruction register are also written into the cache memory.

The next instruction address is generated by the program sequencer depending on the current instruction and internal status. This address is placed on the program memory address (PMA) bus. The program sequencer uses features such as conditional branching, loop counters and zero-overhead looping to minimize program flow overhead. The program memory address (PMA) bus is 14 bits wide, allowing direct access to upto 16K words of data. The state of the PMDA pin distinguishes between code and data access of program memory. The program memory data (PMD) bus, like the processor's instruction words, is 24 bits wide.

The data memory address (DMA) bus is 14 bits wide allowing direct access of upto 16 K words of data. The data memory data (DMD) bus provides a path for the contents of any register in the processor to be transferred to any other register, or to any external data memory location, in a single cycle. The data memory address can come from two sources: an absolute value specified in the instruction code (direct addressing) or the output of a data address generator (indirect addressing). Only indirect addressing is supported for data fetches via the program memory bus.

The program memory data bus can also be used to transfer data to and from the computational units through direct paths or via the PMD-DMD bus exchange unit. The PMD-DMD bus exchange unit permits data to be passed from one bus to the other. It contains hardware to overcome the 8-bit width discrepancy between the two buses when necessary.

Each computational unit contains a set of dedicated input and output registers. Computational operations generally take their operands from input registers and load the result onto an output register. The registers act as a stopover point for data between the external memory and the computational circuitry, effectively introducing one pipeline level on input and one level on output. The computational units are arranged side by side rather than in cascade. To avoid excessive delays when a series of different operations are performed, the internal result (R) bus allows any of the output registers to be used directly (without delay) as the input to another computation.

For a wide variety of calcuations, it is desirable to fetch two operands at the same time, one from data memory and one from program memory. Fetching data from program memory, however, makes it impossible to fetch the next instruction from program memory on the same cycle; an additional cycle would be required. To avoid this overhead, the ADSP-2100 incorporates an instruction cache which holds sixteen words. The benefit of the cache architecture is most apparent when excecuting a program loop that can be totally contained in the cache memory. In this situation, the ADSP-2100 works like a three-bus system with an instruction fetch and two operand fetches taking place at the same time. Many algorithms are readily coded in loops of sixteen instructions or less because of the parallelism and high-level syntax of the ADSP-2100 assembly language.

Every instruction loaded into the instruction register is also written in to cache memory. As additional instructions are fetched, they overwrite the current contents of cache on a circular fashion. When the current instruction does a program memory data access, the cache automatically sources the instruction register if its contents are valid. Operation of the cache is completely transparent to user.

There are two independent data address generators (DAGs). As a pair, they allow the simultaneous fetch of data stored in program and in data memory for executing dual-operand instructions in a single cycle. One data address generator (DAG1) can supply address to the data memory only; the other (DAG2) can supply address to either the data memory or the program memory. Each DAG can handle linear addressing as well as modulo addressing for circular buffers.

With the multiple bus structure, the ADSP-2100 supports a high degree of operational parallelism. In a single cycle, the ADSP-2100 can fetch an instruction, compute the next instruction address, perform one or two data transfers, update one or two address pointers and perform a computation. Every instruction executes in a single cycle.

Figure 4.2b is a simplified representation of the ADSP-2100 in a system context.The figure shows the two external memories used by the processor. Program memory stores instructions and is also used to store data. Data memory stores only data. The data memory address space may be shared with memory-mapped peripherals, if desired. Both memories may be accessed by external devices, such as a system host, if desired.

4.3 CROSS SOFTWARE

Cross software system consists of the following modules

- System Builder
- Assembler
- Linker
- Simulator
- PROM Splitter
- C - Compiler

4.31 SYSTEM BUILDER :-

The ADSP-2100 system builder translates the target hardware environment into the form used by the rest of the cross software system. This module facilitates to specify the amount of RAM and ROM available, the allocation of program and data memory and any memory mapped I/O ports for the target environment.

The desired system is specified in a system specification source file (.SYS) using the system builder directives. The system builder reads this file and generates two output files namely the architecture descrption file (. ACH) and a listing file (.BLD). The architecture description file passes information about the target hardware to the linker and simulator (fig.4.31a)

An example of system specification source file is annexed.

4.32 ASSEMBLER

The assembler translates source code modules into object code modules. An assembler source code module can be created using the ADSP-2100 assembly langauge and defined variables data buffers and symbolic constants using assembler directives. Separately assembled modules are later linked together to form a running system.

The ADSP-2100 assembler reads a source code file (.DSP) and generates four output files with the same root name : an object file (.OBJ), a code file (.CDE), an initialisation file (.INT) and a list file (.LST) (fig4.32a). The object file, code file and initialisation file are passed to the linker. The object file contains the information on the memory allocation and symbol declarations. The code file contains instruction of codes with unresolved symbols marked. The initialisation file contains initialisation information for data buffers. The list file (optional) is for documentation and message passing.Using assembly directives in the source code files, other source code files and the linker can be informed of the initialisation data files in the assembly processor. These files are read by assembler and processed together with the original source file.

4.33 LINKER

The ADSP-2100 linker generates a complete executable program modules which was assembled separately. The linker reads the target hardware information to determine placements of code and data segments. Using the information passed from the system builder, the linker determines the placement of relocatable code and data segments. The assembler output files together with initialisation data files and architecture description files are used by the linker.

The linker may create three files (fig.4.33a). The PM / DM memory image file (.EXE) contains the actual program and data memory images after the linkage. This file is used by the simulator, emulator, evaluation board and is passed to the PROM Splitter to prepare data file for a PROM burner. The optional map listing file (.MAP) assists in interpreting the result of the linkage. The optional debug symbol table file (.SYM) lists all symbols encountered by the linker, their absolute values and their scope of reference. This file is used by simulator, emulator and evaluation board.

4.34 SIMULATOR

The ADSP-2100 simulator provides an easy way to verify ADSP-2100 based system designs prior to hardware develoøpment. The simulator is interactive and screen oriented. The simulator can assemble and disassemble the ADSP-2100 instruction set making full use of symbols in the program.

The simulator reads the architecture description file (.ACH), the memory image file (.SYM) output by the linker (fig.4.34a). The architecture description file is input to the simulator to start an ADSP-2100 simulation session by configuring the simulated system as per the target architecture .

Using the upload / download files (.DAT) the user can load data memory images to simulate data generated by external devices to the simulator and later upload simulator processed data to a file for analysis. The simulator requires an input and an output to simulate a single bidirectional hardware port. The simulator can simulate upto eight memory mapped I /O devices and four interrupts.

The simulator command summary is annexed.

4.35 PROM SPLITTER

The ADSP-2100 PROM Splitter extracts the address information and the contents of the ROM portion of the PM / DM memory image file (.EXE) and formats the extracted images for uploading to PROM burners (fig.4.35a).

Commonly available PROM burners have eight bit wide data input. The PROM splitter separates the memory images into byte - wide format. It creates three one-byte wide PROM image files for the 24-bit program memory and two one-byte wide PROM image files for 16-bit data memory. Both program and data memory can also be optionally output as a single stream one-byte wide files.

4.36 C - COMPILER

ADSP- 2100 C- language system allows programmers familiar with C - programming language to write and compile C - programs to be run on ADSP - 2100. The C - compiler supports in - line assembly code conditional compilation, include files and conforms to the draft ANSI standard with some exceptions. The system consists of two primary modules, the pre processor and the compiler. The preprocessor reads directives begining with the pound sign (#) and takes the actions necessary to resolve them. The compiler reads the output of the preprocessor and produces ADSP - 2100 source code (fig.4.36a). This ADSP - 2100 source code is then assembled with the ADSP - 2100 assembler and is later linked with ADSP - 2100 linker.

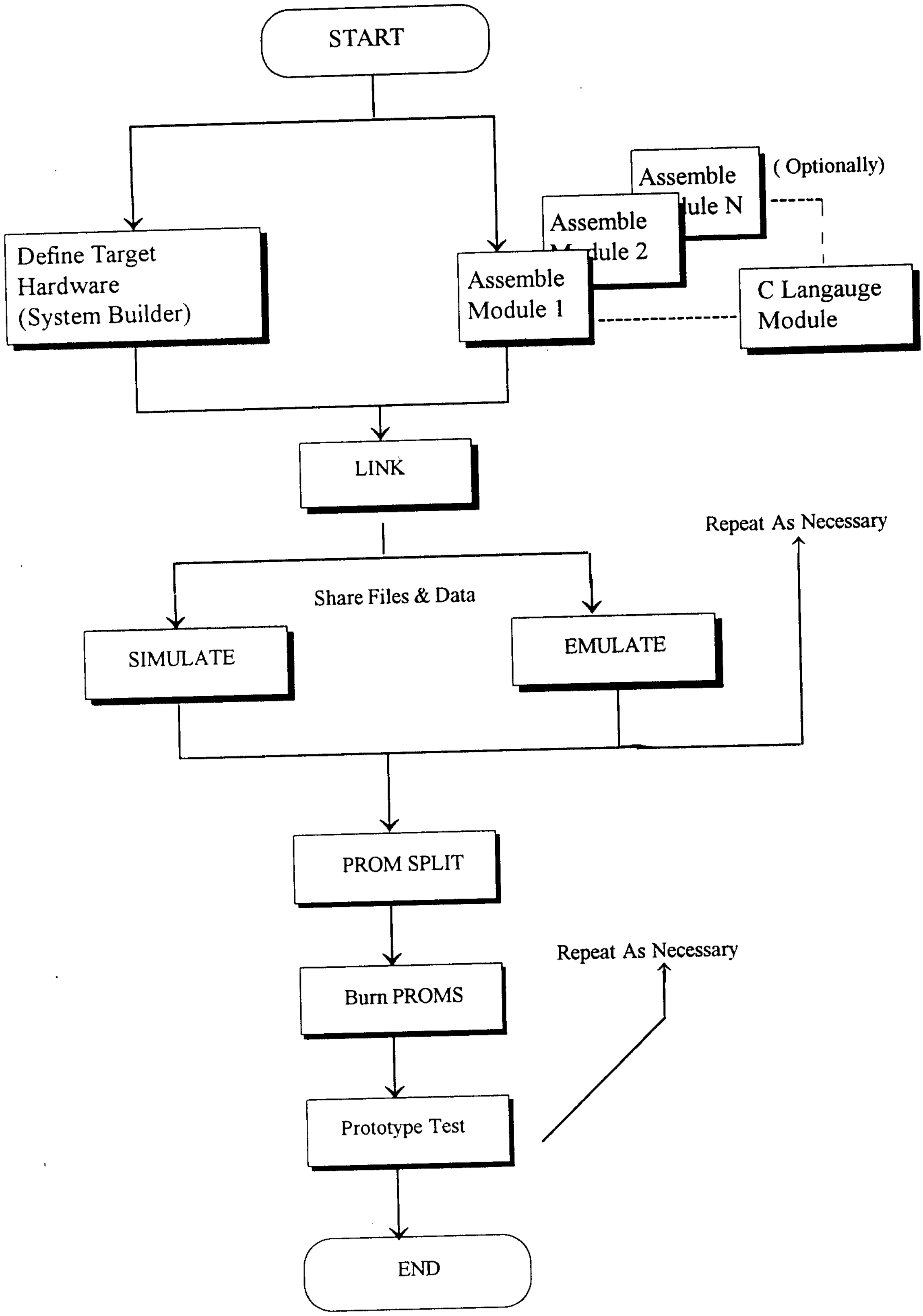

START
Define Target Hardware (System Builder)
Assemble Module 1
Assemble Module 2
Assemble Module N
(Optionally)
C Langauge Module
LINK
Share Files & Data
SIMULATE
EMULATE
Repeat As Necessary
PROM SPLIT
Burn PROMS
Prototype Test
Repeat As Necessary
END

Fig.4.1a

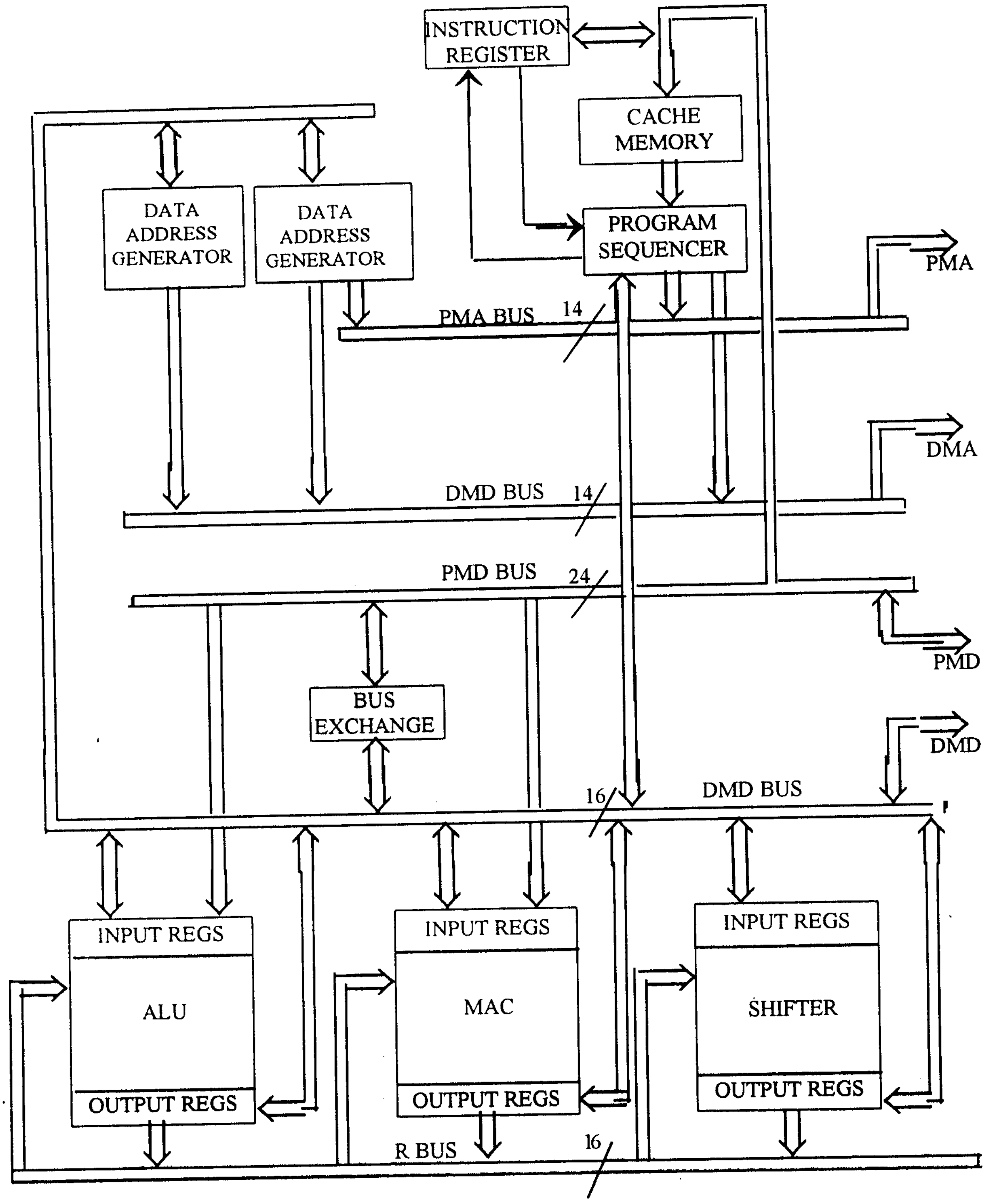

INSTRUCTION REGISTER
CACHE MEMORY
DATA ADDRESS GENERATOR
DATA ADDRESS GENERATOR
PROGRAM SEQUENCER
PMA
PMA BUS
14
DMA
DMD BUS
14
PMD BUS
24
PMD
BUS EXCHANGE
DMD
16
DMD BUS
INPUT REGS
ALU
OUTPUT REGS
INPUT REGS
MAC
OUTPUT REGS
INPUT REGS
SHIFTER
OUTPUT REGS
R BUS
16

Fig.4.2a

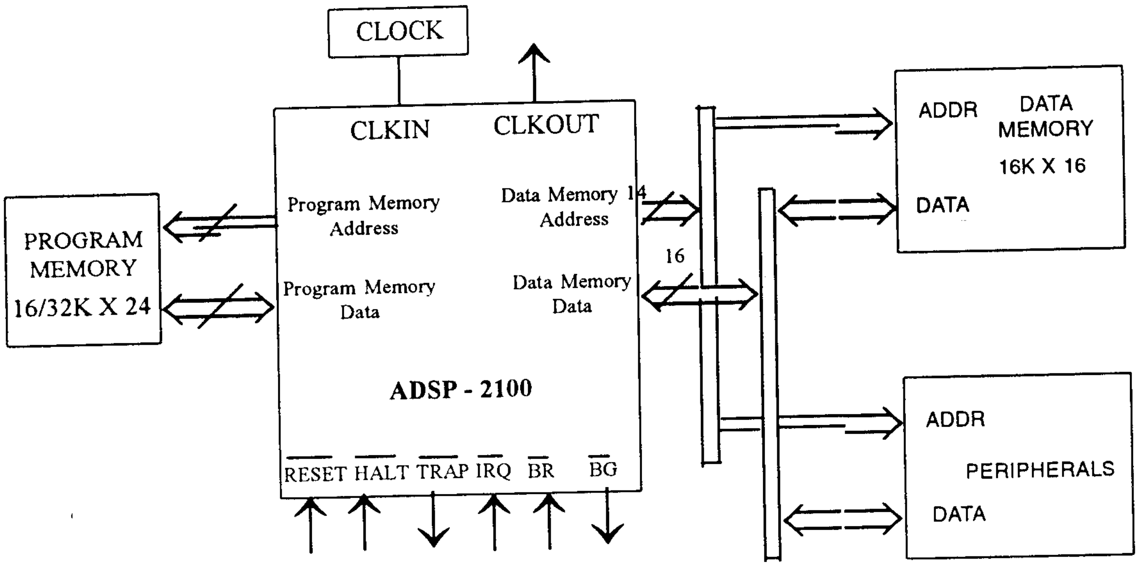

CLOCK
CLKIN
CLKOUT
PROGRAM MEMORY 16/32K X 24
Program Memory Address
Program Memory Data
Data Memory Address
14
16
Data Memory Data
ADSP - 2100
RESET
HALT
TRAP
IRQ
BR
BG
ADDR
DATA MEMORY 16K X 16
DATA
ADDR
PERIPHERALS
DATA

Fig.4.2b

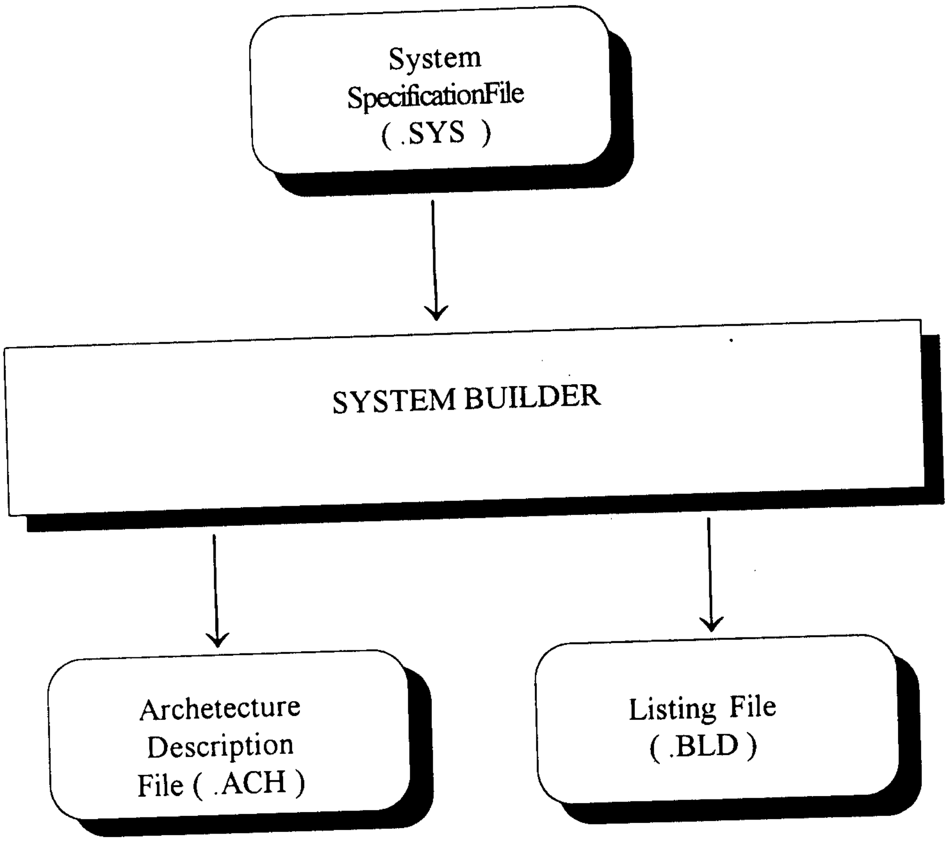

System
SpecificationFile
(.SYS)
SYSTEM BUILDER
Archetecture
Description
File (.ACH)
Listing File
(.BLD)

Fig.4.31a

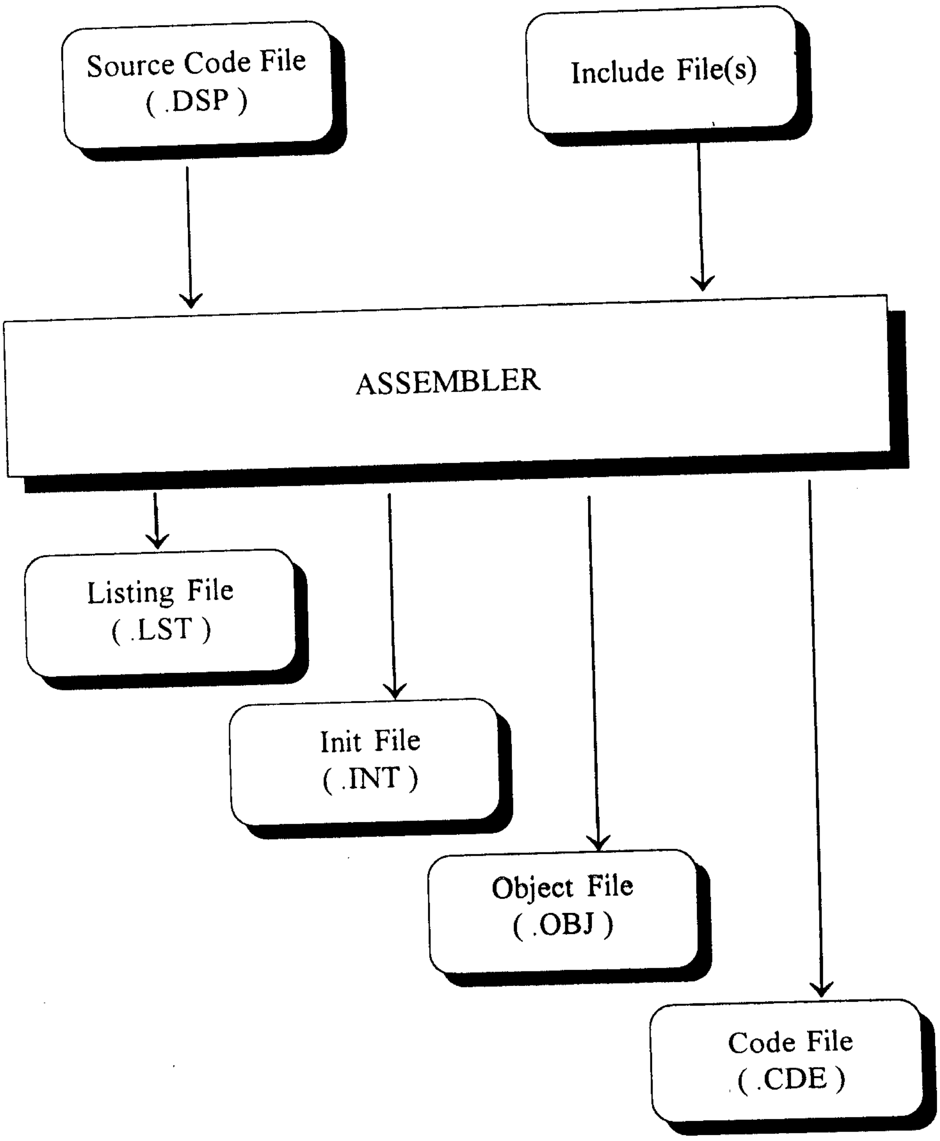

Source Code File
(.DSP)
Include File(s)
ASSEMBLER
Listing File
(.LST)
Init File
(.INT)
Object File
(.OBJ)
Code File
(.CDE)

Fig.4.32a

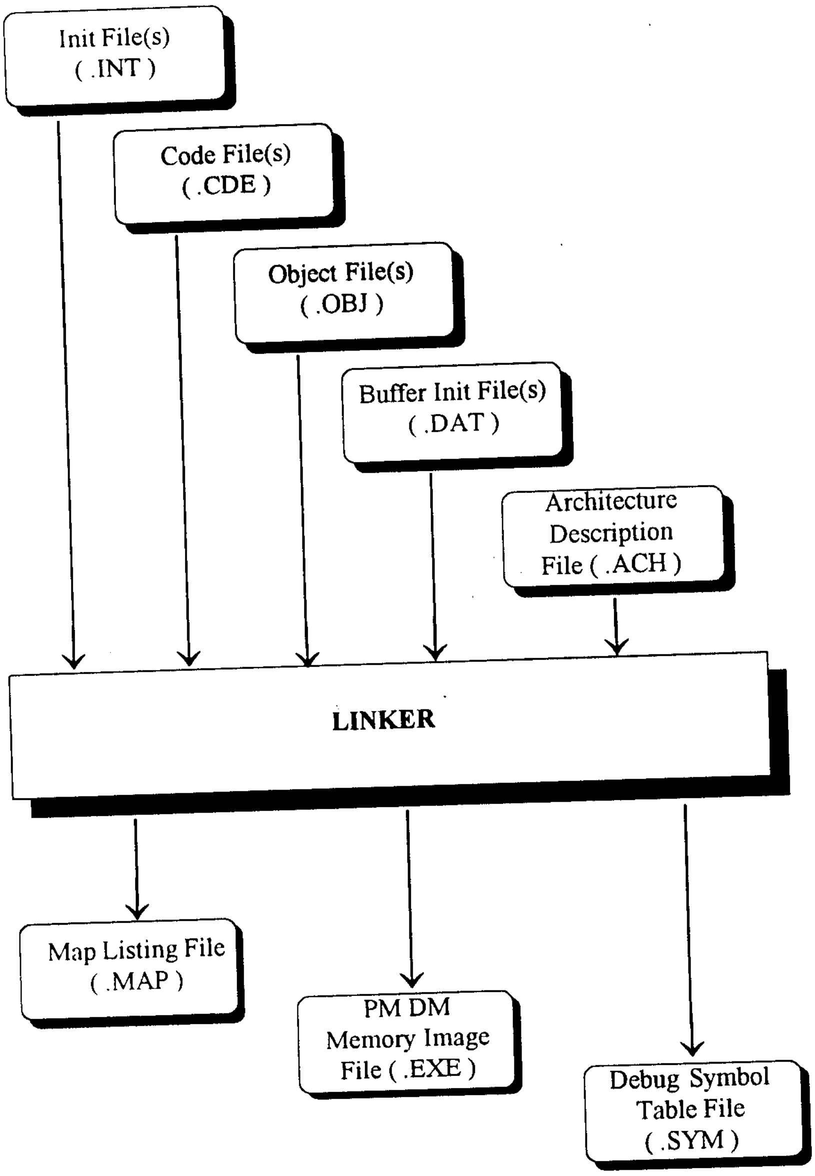

Init File(s)
(.INT)
Code File(s)
(.CDE)
Object File(s)
(.OBJ)
Buffer Init File(s)
(.DAT)
Architecture
Description
File (.ACH)
LINKER
Map Listing File
(.MAP)
PM DM
Memory Image
File (.EXE)
Debug Symbol
Table File
(.SYM)

Fig.4.33a

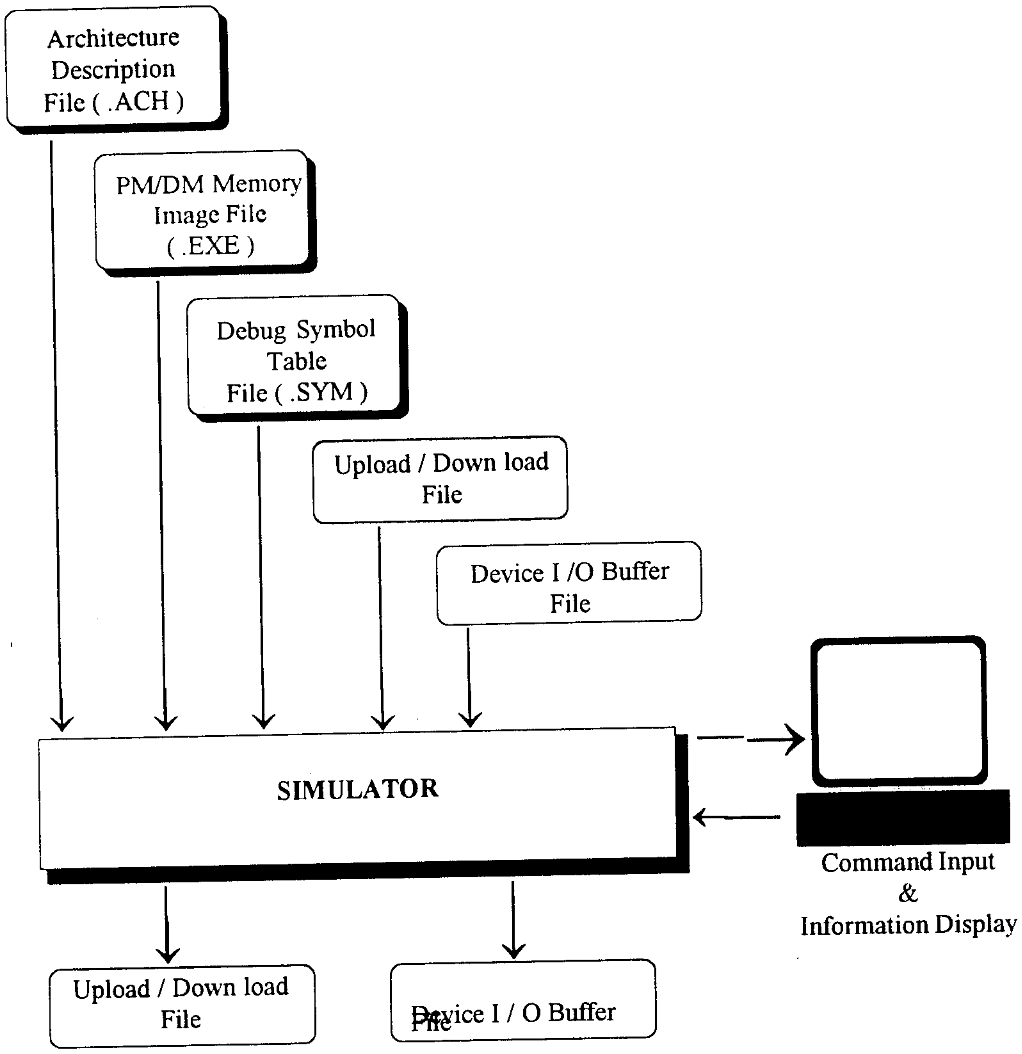

Architecture
Description
File (.ACH)
PM/DM Memory
Image File
(.EXE)
Debug Symbol
Table
File (.SYM)
Upload / Down load
File
Device I /O Buffer
File
SIMULATOR
Command Input
&
Information Display
Upload / Down load
File
Device I / O Buffer
File

Fig.4.34a

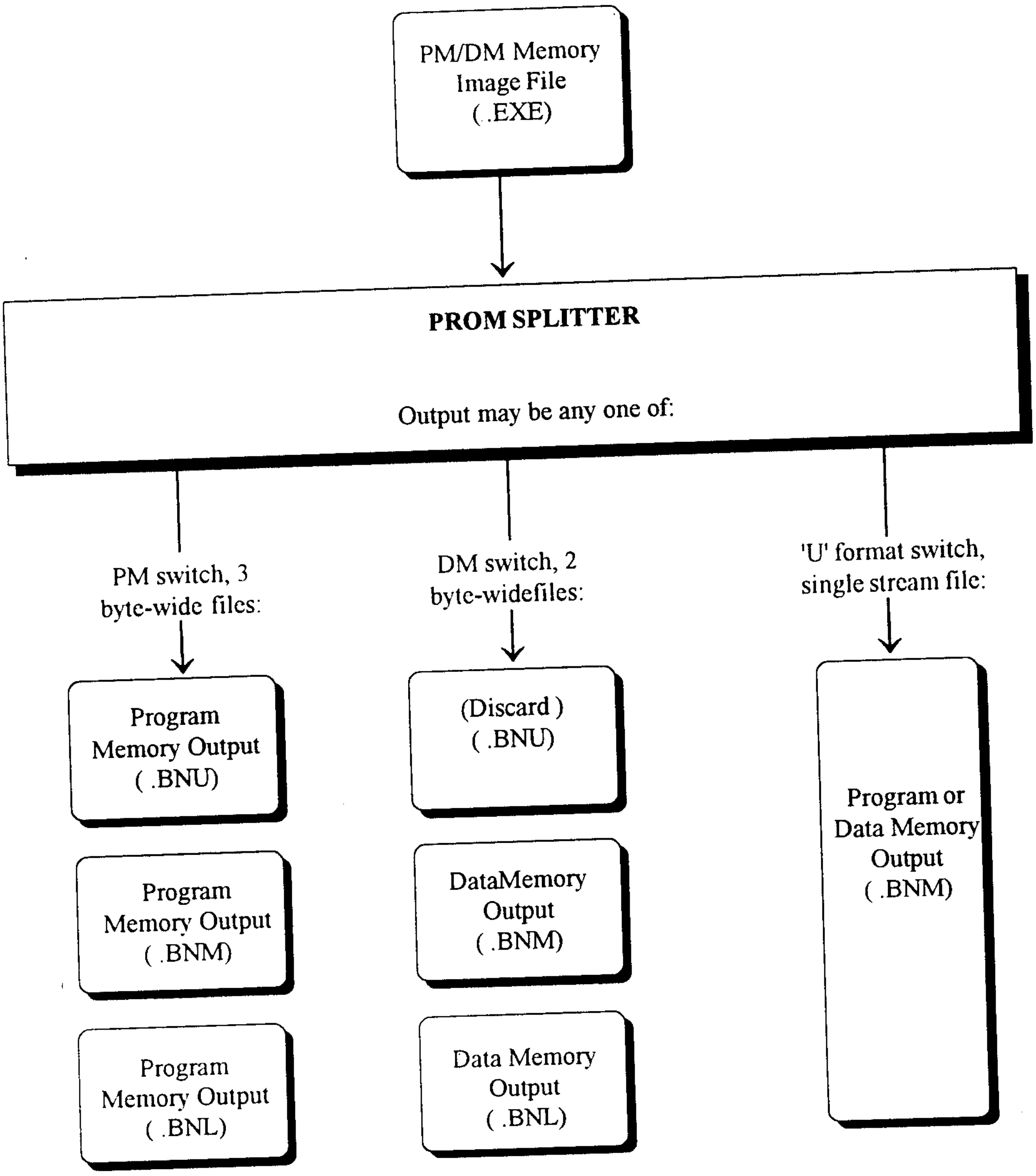

PM/DM Memory
Image File
(.EXE)
PROM SPLITTER
Output may be any one of:
PM switch, 3
byte-wide files:
DM switch, 2
byte-widefiles:
'U' format switch,
single stream file:
Program
Memory Output
(.BNU)
Program
Memory Output
(.BNM)
Program
Memory Output
(.BNL)
(Discard)
(.BNU)
DataMemory
Output
(.BNM)
Data Memory
Output
(.BNL)
Program or
Data Memory
Output
(.BNM)

Fig.4.35a

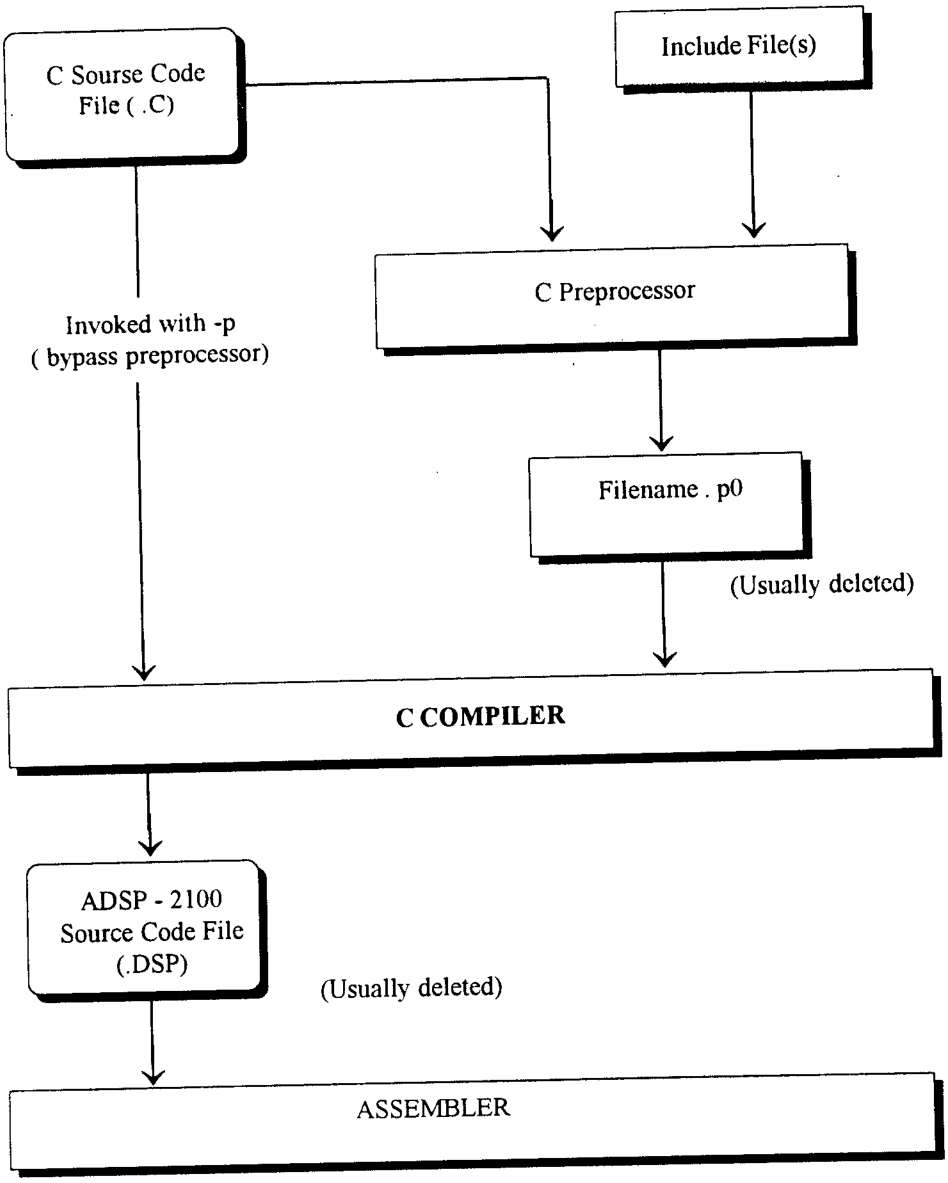

C Sourse Code
File (.C)
Include File(s)
C Preprocessor
Invoked with -p
(bypass preprocessor)
Filename . p0
(Usually deleted)
C COMPILER
ADSP - 2100
Source Code File
(.DSP)
(Usually deleted)
ASSEMBLER

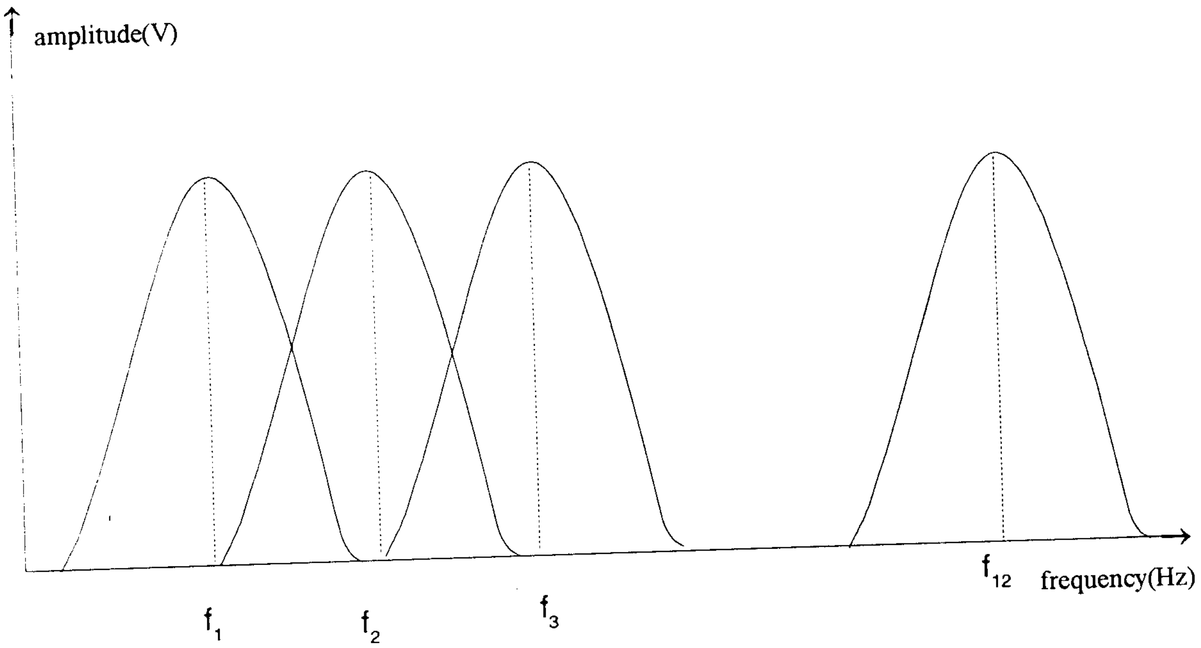
Fig.4.36a

Chapter 5

Practical Implementation

5 PRACTICAL IMPLEMENTATION

The implementation of the DSP system outlined in the previous chapter is as follows. The entire audio range is divided into several octaves. On each of the octaves twelve narrow bandpass filters are to be constructed with the appropriate centre frequencies. Twelve filters are implemented in each octave for better resolution.



All the filters are of elliptic type as the inherent properties namely higher transition rate and decreased order (for the given specification) exhibited by them is advantageous.

The design of the filters found using FDAS software is based upon the following specifications that are to be furnished.

* Pass band ripple in dB

* Stop band ripple in dB

*Pass band cut off frequencies

* Stop band cut off frequencies

* Sampling frequency

* Filter structure

* Filter order desired

* Transformation (Bilinear Transformation in our case)

The designed filters in the octave (1KHz - 2 KHz)is enclosed at the end of this chapter.

The front end of the physical system contains the amplifier, antialiasing filter, sample and hold followed by the ADC. The ADC that would suit our requirements is AD 7878 produced by Analog Devices Inc. Some of the features of the mentioned ADC are as follows.

* Produces DSP interface comprising Track/Hold amplifier with 2 μs aquisition time.

* 7 μs A/D conversions

* 3V Zener reference

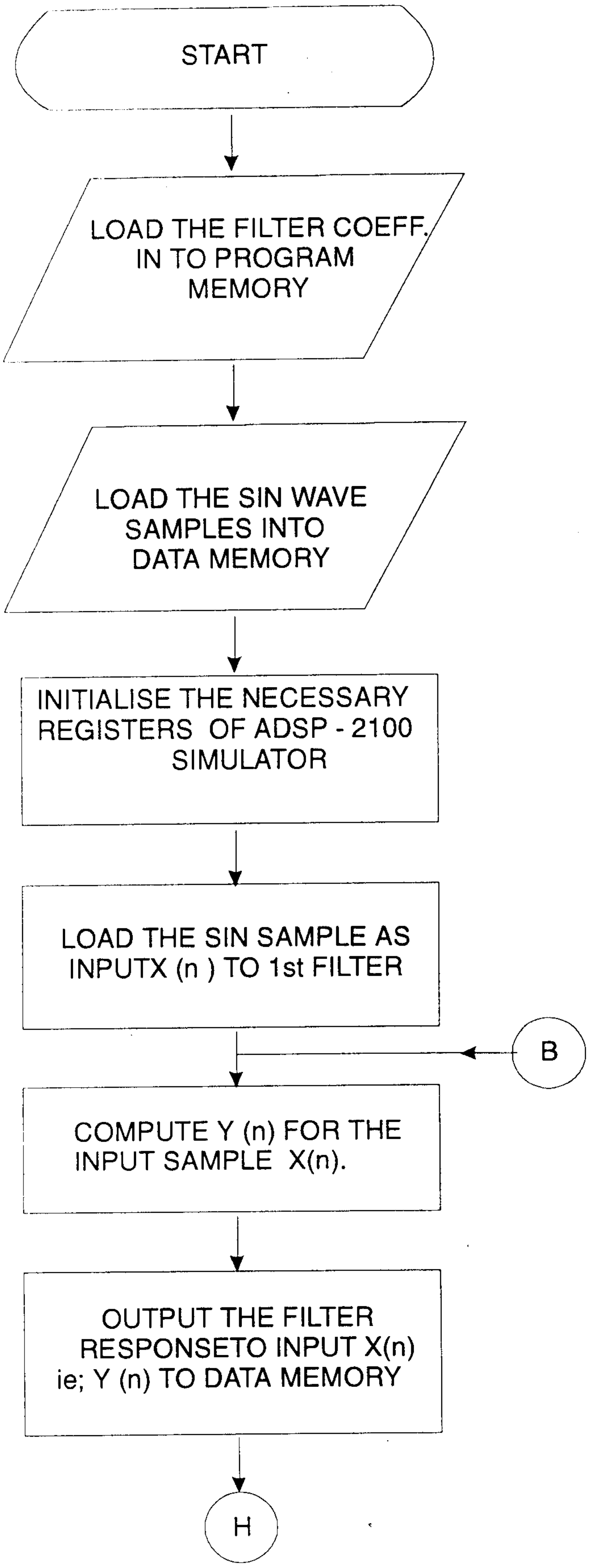
* 8 word FIFO

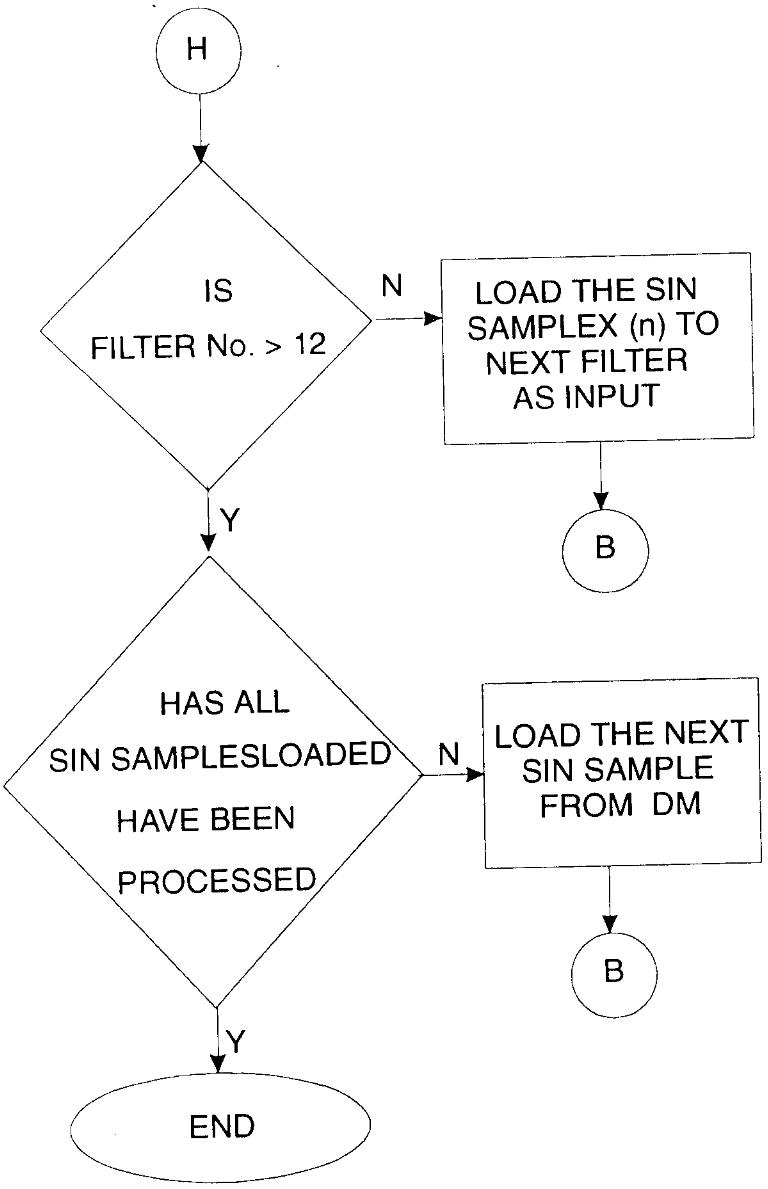
*72 dB SNR at 10 KHz input frequency

* Lower power, 60mW

AD 7878 is a fast complete 12 bit A/D converter with a versatile DSP interface consisting of an 8 word FIFO memory and associated control logic. The FIFO memory allows upto 8 samples to be digitised before the microprocessor is required to service the A/D converter. The ADC can connect full power signals upto 50 KHz. It's fabricated using linear compatable CMOS technology (LC^2 MOS).

5.1 THE IMPLEMENTATION OF THE FILTER BANK FOR THE OCTAVE RANGE 1KHz - 2KHz ON ADSP 2100


START
LOAD THE FILTER COEFF. IN TO PROGRAM MEMORY
LOAD THE SIN WAVE SAMPLES INTO DATA MEMORY
INITIALISE THE NECESSARY REGISTERS OF ADSP - 2100 SIMULATOR
LOAD THE SIN SAMPLE AS INPUTX (n) TO 1st FILTER
B
COMPUTE Y (n) FOR THE INPUT SAMPLE X(n).
OUTPUT THE FILTER RESPONSETO INPUT X(n) ie; Y (n) TO DATA MEMORY
H


H
IS
FILTER No. > 12
N
LOAD THE SIN SAMPLEX (n) TO NEXT FILTER AS INPUT
B
Y
HAS ALL
SIN SAMPLESLOADED
HAVE BEEN
PROCESSED
N
LOAD THE NEXT SIN SAMPLE FROM DM
B
Y
END

DESIGN OF DIGITAL FILTERS USING FDAS SOFTWARE

FILTER - 1

FILTER TYPE	BAND PASS		
ANALOG FILTER TYPE	ELLIPTIC		
PASSBAND RIPPLE IN -dB	-.9700		
STOPBAND RIPPLE IN -dB	- 35.0000		
PASSBAND CUTOFF FREQUENCIES	1049.00	1071.00	HERTZ
STOPBAND CUTOFF FREQUENCIES	1000.00	1120.00	HERTZ

SAMPLING FREQUENCY 8000.00 HERTZ

FILTER ORDER: 8

FILTER DESIGN METHOD: BILINEAR TRANSFOMATION

COEFFICIENTS OF Hd (Z) ARE QUANTIZED TO 24 BITS

QUANTATION TYPE : FIXED POINT FRACTIONAL

COEFFICIENTS SCALED FOR CASCADE FORM 1 (TRANSPOSE FORM)

DIGITAL TRANSFER FUNCTION Hd (Z)

NUMERATOR COEFFICIENTS			DENOMINATOR COEFFICIENTS		
Z**2 TERM	Z TERM	CONST TERM	Z**2 TERM	Z TERM	CONST TERM
.1187498569	-.1705991030	.1187498569	.50000	-.6667202711	.486792160
.1056386232	-.1314325333	.1056386232	.50000	-.6604263783	.4860887527
.1987540722	-.2700380087	.2044421434	.50000	-.6640146971	.4952365160
.4187879562	-.5744217634	.4307816029	.50000	-.6756117344	.4953037500

INITIAL GAIN 1.00000000

DESIGN OF DIGITAL FILTERS USING FDAS SOFTWARE

FILTER - 2

FILTER TYPE	BAND PASS		
ANALOG FILTER TYPE	ELLIPTIC		
PASSBAND RIPPLE IN -dB	-.9700		
STOPBAND RIPPLE IN -dB	- 35.0000		
PASSBAND CUTOFF FREQUENCIES	1110.00	1131.00	HERTZ
STOPBAND CUTOFF FREQUENCIES	1060.00	1190.00	HERTZ

SAMPLING FREQUENCY 8000.00 HERTZ

FILTER ORDER: 8

FILTER DESIGN METHOD: BILINEAR TRANSFOMATION

COEFFICIENTS OF Hd (Z) ARE QUANTIZED TO 24 BITS

QUANTATION TYPE : FIXED POINT FRACTIONAL

COEFFICIENTS SCALED FOR CASCADE FORM 1 (TRANSPOSE FORM)

DIGITAL TRANSFER FUNCTION Hd (Z)

NUMERATOR COEFFICIENTS			DENOMINATOR COEFFICIENTS		
**********************************			********************************		
Z**2 TERM	Z TERM	CONST TERM	Z**2 TERM	Z TERM	CONST TERM
.0969252586	-.1127341986	.0969252586	.50000	-.6254034042	.4866142273
.1082003117	-.1486263275	.1082003117	.50000	-.6317175627	.4866814613
.1978498697	-.2542793751	.2032922506	.50000	-.6284706593	.4954482317
.4165401459	-.5406733751	.4279391766	.50000	-.6399798393	.4955039024
	INITIAL GAIN 1.00000000				

DESIGN OF DIGITAL FILTERS USING FDAS SOFTWARE

FILTER - 3

FILTER TYPE	BAND PASS		
ANALOG FILTER TYPE	ELLIPTIC		
PASSBAND RIPPLE IN -dB	-.9700		
STOPBAND RIPPLE IN -dB	- 35.0000		
PASSBAND CUTOFF FREQUENCIES	1179.00	1201.00	HERTZ
STOPBAND CUTOFF FREQUENCIES	1120.00	1260.00	HERTZ

SAMPLING FREQUENCY 8000.00 HERTZ

FILTER ORDER: 8

FILTER DESIGN METHOD: BILINEAR TRANSFOMATION

COEFFICIENTS OF Hd (Z) ARE QUANTIZED TO 24 BITS

QUANTATION TYPE : FIXED POINT FRACTIONAL

COEFFICIENTS SCALED FOR CASCADE FORM 1 (TRANSPOSE FORM)

DIGITAL TRANSFER FUNCTION Hd (Z)

NUMERATOR COEFFICIENTS			DENOMINATOR COEFFICIENTS		
Z**2 TERM	Z TERM	CONST TERM	Z**2 TERM	Z TERM	CONST TERM
.0927652121	-.0982555151	0927652121	.50000	-.5822414160	.4857035875
.0989769697	-.1290367842	.0989769697	.50000	-.5895521641	.4864659309
.2021353245	-.2423114777	.2080850601	.50000	-.5850080252	.4952297211
.4111014605	-.4982172251	.4225386381	.50000	-.5975703001	.4952843189

INITIAL GAIN 1.00000000

DESIGN OF DIGITAL FILTERS USING FDAS SOFTWARE

FILTER - 4

FILTER TYPE	BAND PASS		
ANALOG FILTER TYPE	ELLIPTIC		
PASSBAND RIPPLE IN -dB	-.9700		
STOPBAND RIPPLE IN -dB	- 35.0000		
PASSBAND CUTOFF FREQUENCIES	1249.00	1271.00	HERTZ
STOPBAND CUTOFF FREQUENCIES	1190.00	1330.00	HERTZ

SAMPLING FREQUENCY 8000.00 HERTZ

FILTER ORDER: 8

FILTER DESIGN METHOD: BILINEAR TRANSFOMATION

COEFFICIENTS OF Hd (Z) ARE QUANTIZED TO 24 BITS

QUANTATION TYPE : FIXED POINT FRACTIONAL

COEFFICIENTS SCALED FOR CASCADE FORM 1 (TRANSPOSE FORM)

DIGITAL TRANSFER FUNCTION Hd (Z)

NUMERATOR COEFFICIENTS			DENOMINATOR COEFFICIENTS		
**********************************			********************************		
Z**2 TERM	Z TERM	CONST TERM	Z**2 TERM	Z TERM	CONST TERM
.0998744965	-.1217470169	.098744965	.50000	-.5448803902	.4860804081
.0916150808	-.0883606672	.0916150808	.50000	-.5377646685	.4860874414
.1983801126	-.2194705009	.2040580511	.50000	-.5398828983	.4952325821
.4185771942	-.4692115784	.4305636883	.50000	-.5529286861	.4952812195

INITIAL GAIN 1.00000000

DESIGN OF DIGITAL FILTERS USING FDAS SOFTWARE

FILTER - 5

```
FILTER TYPE                        BAND PASS
ANALOG FILTER TYPE                 ELLIPTIC
PASSBAND RIPPLE IN  -dB                -.9700
STOPBAND RIPPLE IN  -dB              - 35.0000
PASSBAND CUTOFF FREQUENCIES           1319.00      1341.00      HERTZ

STOPBAND CUTOFF  FREQUENCIES          1260.00      1410.00      HERTZ

SAMPLING FREQUENCY              8000.00 HERTZ
FILTER ORDER:               8
FILTER DESIGN METHOD:  BILINEAR TRANSFOMATION

COEFFICIENTS OF Hd (Z) ARE QUANTIZED TO  24 BITS
QUANTATION TYPE : FIXED POINT FRACTIONAL
COEFFICIENTS SCALED FOR CASCADE FORM 1 (TRANSPOSE FORM)
```

DIGITAL TRAN$\overset{F}{S}$ER FUNCTION Hd (Z)

```
      NUMERATOR COEFFICIENTS                DENOMINATOR COEFFICIENTS
**********************************    ********************************

Z**2 TERM     Z TERM     CONST TERM    Z**2 TERM    Z TERM       CONST TERM
.1974189281  -.2239594460 .1974189281   1.00000    -.9971808195  .9713920355
.1739912033  -.1500303745 .1739912033   1.00000    -.9832397699  .9729471207
.4129800797  -.4173489809 .4244630337   1.00000    -.9862560034  .9904664755
.4008204937  -.4114615917 .4126249552    .50000    -.5066126585  .4952760935
       INITIAL GAIN   1.00000000
```

DESIGN OF DIGITAL FILTERS USING FDAS SOFTWARE

FILTER - 6

FILTER TYPE	BAND PASS		
ANALOG FILTER TYPE	ELLIPTIC		
PASSBAND RIPPLE IN -dB	-.9700		
STOPBAND RIPPLE IN -dB	- 35.0000		
PASSBAND CUTOFF FREQUENCIES	1399.00	1421.00	HERTZ
STOPBAND CUTOFF FREQUENCIES	1330.00	1500.00	HERTZ

SAMPLING FREQUENCY 8000.00 HERTZ

FILTER ORDER: 8

FILTER DESIGN METHOD: BILINEAR TRANSFOMATION

COEFFICIENTS OF Hd (Z) ARE QUANTIZED TO 24 BITS

QUANTATION TYPE : FIXED POINT FRACTIONAL

COEFFICIENTS SCALED FOR CASCADE FORM 1 (TRANSPOSE FORM)

DIGITAL TRANSFER FUNCTION Hd (Z)

NUMERATOR COEFFICIENTS			DENOMINATOR COEFFICIENTS		
Z**2 TERM	Z TERM	CONST TERM	Z**2 TERM	Z TERM	CONST TERM
.1725628376	-.1807830334	.1725628376	1.00000	-.8884516954	.9713981152
.1523896456	-.1106227636	.1523896456	1.00000	-.8739098310	.9727409218
.4117527008	-.3698422909	.4232041836	1.00000	-.8757653236	.9904569387
.8031131029	-.7345362902	.8267599344	1.00000	-.9036093950	.9905308485

INITIAL GAIN 1.00000000

DESIGN OF DIGITAL FILTERS USING FDAS SOFTWARE

FILTER - 7

FILTER TYPE	BAND PASS		
ANALOG FILTER TYPE	ELLIPTIC		
PASSBAND RIPPLE IN -dB	-.9700		
STOPBAND RIPPLE IN -dB	- 35.0000		
PASSBAND CUTOFF FREQUENCIES	1489.00	1511.00	HERTZ
STOPBAND CUTOFF FREQUENCIES	1410.00	1580.00	HERTZ

SAMPLING FREQUENCY 8000.00 HERTZ

FILTER ORDER: 8

FILTER DESIGN METHOD: BILINEAR TRANSFOMATION

COEFFICIENTS OF Hd (Z) ARE QUANTIZED TO 24 BITS

QUANTATION TYPE : FIXED POINT FRACTIONAL

COEFFICIENTS SCALED FOR CASCADE FORM 1 (TRANSPOSE FORM)

DIGITAL TRAN$\underset{\wedge}{\overset{F}{S}}$ER FUNCTION Hd (Z)

NUMERATOR COEFFICIENTS			DENOMINATOR COEFFICIENTS		
Z**2 TERM	Z TERM	CONST TERM	Z**2 TERM	Z TERM	CONST TERM
.1707973480	-.1573051214	.1707973480	1.00000	-.7625004153	.9721635580
.1625005007	-.0971939564	.1625005007	1.00000	-.7467643023	.9721726179
.3965697289	-.3046208620	.4079210758	1.00000	-.7473347187	.9904662371
.8362491131	-.6558982134	.8601938486	1.00000	-.7760899067	.9905273914

INITIAL GAIN 1.00000000

DESIGN OF DIGITAL FILTERS USING FDAS SOFTWARE

FILTER - 8

FILTER TYPE	BAND PASS		
ANALOG FILTER TYPE	ELLIPTIC		
PASSBAND RIPPLE IN -dB	-.9700		
STOPBAND RIPPLE IN -dB	- 35.0000		
PASSBAND CUTOFF FREQUENCIES	1579.00	1591.00	HERTZ
STOPBAND CUTOFF FREQUENCIES	1500.00	1680.00	HERTZ

SAMPLING FREQUENCY 8000.00 HERTZ

FILTER ORDER: 8

FILTER DESIGN METHOD: BILINEAR TRANSFOMATION

COEFFICIENTS OF Hd (Z) ARE QUANTIZED TO 24 BITS

QUANTATION TYPE : FIXED POINT FRACTIONAL

COEFFICIENTS SCALED FOR CASCADE FORM 1 (TRANSPOSE FORM)

DIGITAL TRANSFER FUNCTION Hd (Z)

NUMERATOR COEFFICIENTS			DENOMINATOR COEFFICIENTS		
************************************			**********************************		
Z**2 TERM	Z TERM	CONST TERM	Z**2 TERM	Z TERM	CONST TERM
0903590918	-.0736408234	.0903590918	1.00000	-.6396229267	.9838179350
0867491961	-.0394047499	.0867491961	1.00000	-.6307668686	.9838179350
.3762413250	-.2389159203	.3787705898	1.00000	-.6307026148	.9947761297
.8091123104	-.5260392427	.8224207163	1.00000	-.6467466354	.9947910309

INITIAL GAIN 1.00000000

DESIGN OF DIGITAL FILTERS USING FDAS SOFTWARE

FILTER - 9

FILTER TYPE	BAND PASS		
ANALOG FILTER TYPE	ELLIPTIC		
PASSBAND RIPPLE IN -dB	-.9700		
STOPBAND RIPPLE IN -dB	- 35.0000		
PASSBAND CUTOFF FREQUENCIES	1669.00	1691.00	HERTZ
STOPBAND CUTOFF FREQUENCIES	1580.00	1780.00	HERTZ

SAMPLING FREQUENCY 8000.00 HERTZ

FILTER ORDER: 8

FILTER DESIGN METHOD: BILINEAR TRANSFOMATION

COEFFICIENTS OF Hd (Z) ARE QUANTIZED TO 24 BITS

QUANTATION TYPE : FIXED POINT FRACTIONAL

COEFFICIENTS SCALED FOR CASCADE FORM 1 (TRANSPOSE FORM)

DIGITAL TRANSFER FUNCTION Hd (Z)

NUMERATOR COEFFICIENTS			DENOMINATOR COEFFICIENTS		
Z**2 TERM	Z TERM	CONST TERM	Z**2 TERM	Z TERM	CONST TERM
.1276077032	-.0356633663	.1276077032	1.00000	-.4821540117	.9721678495
.1386144161	-.0975039005	.1386144161	1.00000	-.4986554384	.9721678495
.3966150284	-.1967042685	.4079697132	1.00000	-.4799643755	.9904586077
.8345972300	-.4280911684	.8584908247	1.00000	-.5100358725	.9904966354

INITIAL GAIN 1.00000000

DESIGN OF DIGITAL FILTERS USING FDAS SOFTWARE

FILTER - 10

FILTER TYPE	BAND PASS		
ANALOG FILTER TYPE	ELLIPTIC		
PASSBAND RIPPLE IN -dB	-.9700		
STOPBAND RIPPLE IN -dB	- 35.0000		
PASSBAND CUTOFF FREQUENCIES	1769.00	1791.00	HERTZ
STOPBAND CUTOFF FREQUENCIES	1680.00	1890.00	HERTZ

SAMPLING FREQUENCY 8000.00 HERTZ

FILTER ORDER: 8

FILTER DESIGN METHOD: BILINEAR TRANSFOMATION

COEFFICIENTS OF Hd (Z) ARE QUANTIZED TO 24 BITS

QUANTATION TYPE : FIXED POINT FRACTIONAL

COEFFICIENTS SCALED FOR CASCADE FORM 1 (TRANSPOSE FORM)

DIGITAL TRANSFER FUNCTION Hd (Z)

NUMERATOR COEFFICIENTS **********************************			DENOMINATOR COEFFICIENTS ********************************		
Z**2 TERM	Z TERM	CONST TERM	Z**2 TERM	Z TERM	CONST TERM
.1311998367	-.0733093023	.1311998367	1.00000	-.3474264145	.9721659422
.1304328442	-.0157139301	.1304328442	1.00000	-.3306442499	.9721698761
.3969492912	-.1350061893	.4083126783	1.00000	-.3269248009	.9904636145
.8337723017	-.2979681492	.8576438427	1.00000	-.3575009108	.9904893637

INITIAL GAIN 1.00000000

DESIGN OF DIGITAL FILTERS USING FDAS SOFTWARE

FILTER - 11

FILTER TYPE	BAND PASS		
ANALOG FILTER TYPE	ELLIPTIC		
PASSBAND RIPPLE IN -dB	-.9700		
STOPBAND RIPPLE IN -dB	- 35.0000		
PASSBAND CUTOFF FREQUENCIES	1871.00	1901.00	HERTZ
STOPBAND CUTOFF FREQUENCIES	1780.00	2000.00	HERTZ

SAMPLING FREQUENCY 8000.00 HERTZ

FILTER ORDER: 8

FILTER DESIGN METHOD: BILINEAR TRANSFOMATION

COEFFICIENTS OF Hd (Z) ARE QUANTIZED TO 24 BITS

QUANTATION TYPE : FIXED POINT FRACTIONAL

COEFFICIENTS SCALED FOR CASCADE FORM 1 (TRANSPOSE FORM)

DIGITAL TRANSFER FUNCTION Hd (Z)

NUMERATOR COEFFICIENTS			DENOMINATOR COEFFICIENTS		
**********************************			*********************************		
Z**2 TERM	Z TERM	CONST TERM	Z**2 TERM	Z TERM	CONST TERM
.1613601446	-.0092558861	.1613601446	1.00000	-.1640924215	.9630602598
.1653113365	-.0677983761	.1653113365	1.00000	-.1871836185	.9636925459
.4077255726	-.0694708824	.4233645201	1.00000	-.1565809250	.9870532751
.8529294729	-.1656694412	.8850638866	1.00000	-.1987537146	.9870779514

INITIAL GAIN 1.00000000

DESIGN OF DIGITAL FILTERS USING FDAS SOFTWARE

FILTER - 12

FILTER TYPE	BAND PASS		
ANALOG FILTER TYPE	ELLIPTIC		
PASSBAND RIPPLE IN -dB	-.9700		
STOPBAND RIPPLE IN -dB	- 35.0000		
PASSBAND CUTOFF FREQUENCIES	1979.00	2011.00	HERTZ
STOPBAND CUTOFF FREQUENCIES	1890.00	2060.00	HERTZ

SAMPLING FREQUENCY 8000.00 HERTZ

FILTER ORDER: 8

FILTER DESIGN METHOD: BILINEAR TRANSFOMATION

COEFFICIENTS OF Hd (Z) ARE QUANTIZED TO 24 BITS

QUANTATION TYPE : FIXED POINT FRACTIONAL

COEFFICIENTS SCALED FOR CASCADE FORM 1 (TRANSPOSE FORM)

DIGITAL TRANSFER FUNCTION Hd (Z)

NUMERATOR COEFFICIENTS			DENOMINATOR COEFFICIENTS		
*********************************			*******************************		
Z**2 TERM	Z TERM	CONST TERM	Z**2 TERM	Z TERM	CONST TERM
.2873008251	-.0433356762	.2873008251	1.00000	-.0198489428	.9607744217
.2819958925	.0381282568	.2819958925	1.00000	.0047878027	.9616966248
.4241057634	.0021114349	.4409974813	1.00000	.0150345564	.9863766432
.8312094212	-.0171722174	.8651452065	1.00000	-.0306346416	.9863778353
	INITIAL GAIN	1.00000000			

C PROGRAM TO CONVERT FILTER COEFFICIENTS INTO THEIR CORRESPONDING HEXADECIMAL VALUES

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>
main()
 {
    FILE *fout;
    double res,val;     float x[256];
    int n;
    long int temp;
    printf("Enter the coefficients : \n");
    for(n=0;n<240;++n)
      {
           scanf("%f",&x[n]);
      }
     fout=fopen("convert.out","wt");
     printf("Output file - convert.out\n");
     for (n=0;n<240;n++)
            {
              res=32767.0 * x[n];
              temp=res;
              fprintf(fout,"%04x\n",(int)temp);
             }

       fclose(fout);
   }
```

C PROGRAM TO GENERATE SINE SAMPLES

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>
void readval(float *value);
main( )
 {
   FILE *fout;
   double res,val;
   float fs,fin,amp,no_pt,n;
   long int temp;
   printf(“Enter sampling freq : “);
   readval(&fs);
   printf(“Enter signal freq : “);
   readval(&fin);
   printf(“Enter amp value (max 1) : “);
   readval(&amp);
   printf(“Enter no. of points : “);
   readval(&no_pt);
          if((fout=fopen(“sine.out”,”wt”)) == NULL);
          {
//       printf(“Cannot open file ‘sine.out’.\n”);
//        exit(0);
          }
    printf(“Output file - sine.out\n”);
    for (n=0;n<no_pt;n++)
          {
            val=2.0 * 3.141592654 * n * fin /fs;
            res=32767.0 * amp * sin(val);
            temp=res;
            fprintf(fout,”%04x\n”,temp);
          }

     fclose(fout);

   }

   void readval(float *value)
   {
    char amp[10];
    gets(amp);
    while(!strlen(amp))
           {
           printf(“\n Error! Enter again : “);
           gets(amp);
           }
     sscanf(amp,”%f”,value);
     flushall( );

    }
```

****

FILTER BANK IMPLEMENTATION FOR THE OCTAVE RANGE 1KHz - 2KHz

****

```
.module octaveanalyzer;

.var/dm/ram/abs=0x0300                  sin[256];

.var/pm/ram/abs=0x4096                  coeff[240];

.init     sin:<sinew.out>;

.init     coeff:<convert.out>;

.entry twvlfilt;

               rti;
               rti;
               rti;
               rti;

twvlfilt:  i0=0;
               ay1=0;
               ar=0;
               sr0=0;
               sr1=0;
               mx0=0;
               my0=0;
               mr=0;
               i4=4096;
               i7=700;
               m7=1;
               m4=1;
               m0=1;
               ax0=0;
               l0=250;
               l4=250;
               l7=250;
               ax1=0;
               ay0=0;
               i5=300;
               l5=256;
               m5=1;
               cntr=256;


               do loop until ce;


               { 1st FILTER OF THE OCTAVE }
```

```
i0=0;
i4=4096;
ax0=dm(i5,m5);
dm(i0,m0)=ax0;
i0=0;
ax0=0;

{ 2nd  order biquad }

mx0=dm(i0,m0),my0=pm(i4,m4);
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);


mx0=dm(i0,m0), my0=pm(i4,m4);
ax0=sr0;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;


mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 -  ay0;
```

```
mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 - ay0;

i0=3;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=3;
dm(i0,m0)=ar;
i0=1;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=0;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=5;
dm(i0,m0)=ar;

{ 4th order biquad }

i0=5;
i4=4101;
mx0=dm(i0,m0),my0=pm(i4,m4);
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);

mx0=dm(i0,m0), my0=pm(i4,m4);
ax0=sr0;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
```

```
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 -  ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 - ay0;

i0=8;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=8;
dm(i0,m0)=ar;
i0=6;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=5;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;


{ 6th order biquad }

i0=10;
dm(i0,m0)=ar;
i0=10;
i4=4106;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
```

```
sr=lshift mr0 by -1(lo);

mx0=dm(i0,m0), my0=pm(i4,m4);
ax0=sr0;
ay0=my0;
ax1=px;
ar=ax1 + ay0;
ay0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ay0=my0;
ax1=px;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ay0=my0;
ax1=px;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 -  ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ay0=my0;
ax1=px;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 - ay0;

i0=13;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=13;
dm(i0,m0)=ar;
i0=11;
ax0=dm(i0,m0);
```

```
dm(i0,m0)=ax0;
i0=10;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;

{ 8th order biquad }

i0=15;
dm(i0,m0)=ar;
i0=15;
i4=4111;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);

mx0=dm(i0,m0), my0=pm(i4,m4);
ax0=sr0;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax0 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 -  ay0;
```

```
mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax0 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 - ay0;
dm(i7,m7)=ar;

i0=18;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=18;
dm(i0,m0)=ar;
i0=16;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=15;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;


{ 2nd FILTER OF THE OCTAVE }


i0=0;
ax0=dm(i0,m0);
i0=20;
dm(i0,m0)=ax0;
i0=20;
i4=4116;
ax0=0;

{ 2nd order biquad }

mx0=dm(i0,m0),my0=pm(i4,m4);
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);


mx0=dm(i0,m0), my0=pm(i4,m4);
ax0=sr0;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
```

```
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;


mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 -  ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 - ay0;

i0=23;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=23;
dm(i0,m0)=ar;
i0=21;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=20;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=25;
dm(i0,m0)=ar;


i0=25;
```

```
i4=4121;

{ 4th order biquad }

mx0=dm(i0,m0),my0=pm(i4,m4);
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);

mx0=dm(i0,m0), my0=pm(i4,m4);
ax0=sr0;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 -  ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
```

```
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 - ay0;

i0=28;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=28;
dm(i0,m0)=ar;
i0=26;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=25;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;


{ 6th order biquad }

i0=30;
dm(i0,m0)=ar;
i0=30;
i4=4126;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);

mx0=dm(i0,m0), my0=pm(i4,m4);
ax0=sr0;
ay0=my0;
ax1=px;
ar=ax1 + ay0;
ay0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ay0=my0;
ax1=px;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;
```

```
mx0=dm(i0,m0),my0=pm(i4,m4);
 ax0=ar;
 ay0=my0;
 ax1=px;
 ar=ax1 + ay0;
 my0=ar;
 px=0;
 mr=mx0 * my0(uu);
 sr=lshift mr0 by -1(lo);
 ay0=sr0;
 ar=ax0 -  ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
 ax0=ar;
 ay0=my0;
 ax1=px;
 ar=ax1 + ay0;
 my0=ar;
 px=0;
 mr=mx0 * my0(uu);
 sr=lshift mr0 by -1(lo);
 ay0=sr0;
 ar=ax0 - ay0;

 i0=33;
 ax0=dm(i0,m0);
 dm(i0,m0)=ax0;
 i0=33;
 dm(i0,m0)=ar;
 i0=31;
 ax0=dm(i0,m0);
 dm(i0,m0)=ax0;
 i0=30;
 ax0=dm(i0,m0);
 dm(i0,m0)=ax0;

 { 8th order biquad }

 i0=35;
 dm(i0,m0)=ar;
 i0=35;
 i4=4131;

 mx0=dm(i0,m0),my0=pm(i4,m4);
  ax1=px;
  ay0=my0;
  ar=ax1 + ay0;
  my0=ar;
  px=0;
  mr=mx0 * my0(uu);
  sr=lshift mr0 by -1(lo);

 mx0=dm(i0,m0), my0=pm(i4,m4);
  ax0=sr0;
  ax1=px;
  ay0=my0;
  ar=ax1 + ay0;
```

```
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax0 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 -  ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax0 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 - ay0;
dm(i7,m7)=ar;

i0=38;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=38;
dm(i0,m0)=ar;
i0=36;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=35;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
```

```
{ 3rd FILTER OF THE OCTAVE }


i0=0;
ax0=dm(i0,m0);
i0=40;
dm(i0,m0)=ax0;
i0=40;
i4=4136;
ax0=0;

{ 2nd order biquad }

mx0=dm(i0,m0),my0=pm(i4,m4);
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);


mx0=dm(i0,m0), my0=pm(i4,m4);
ax0=sr0;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;


mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
```

```
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 -  ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 - ay0;

i0=43;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=43;
dm(i0,m0)=ar;
i0=41;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=40;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=45;
dm(i0,m0)=ar;


i0=45;
i4=4141;

{ 4th order biquad }

mx0=dm(i0,m0),my0=pm(i4,m4);
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);

mx0=dm(i0,m0), my0=pm(i4,m4);
ax0=sr0;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;
```

```
mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 -  ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 - ay0;

i0=48;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=48;
dm(i0,m0)=ar;
i0=46;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=45;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;


{ 6th order biquad }

i0=50;
dm(i0,m0)=ar;
i0=50;
i4=4146;

mx0=dm(i0,m0),my0=pm(i4,m4);
```

```
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);

mx0=dm(i0,m0), my0=pm(i4,m4);
ax0=sr0;
ay0=my0;
ax1=px;
ar=ax1 + ay0;
ay0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ay0=my0;
ax1=px;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ay0=my0;
ax1=px;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 -  ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ay0=my0;
ax1=px;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 - ay0;

i0=53;
```

```
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=53;
dm(i0,m0)=ar;
i0=51;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=50;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;

{ 8th order biquad }

i0=55;
dm(i0,m0)=ar;
i0=55;
i4=4151;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);

mx0=dm(i0,m0), my0=pm(i4,m4);
ax0=sr0;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax0 + ay0;
my0=ar;
```

```
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 -  ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax0 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 - ay0;
dm(i7,m7)=ar;

i0=58;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=58;
dm(i0,m0)=ar;
i0=56;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=55;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;



{ 4th FILTER OF THE OCTAVE }

i0=0;
ax0=dm(i0,m0);
i0=60;
dm(i0,m0)=ax0;
i0=60;
i4=4156;
ax0=0;

{ 2nd order biquad }

mx0=dm(i0,m0),my0=pm(i4,m4);
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);


mx0=dm(i0,m0), my0=pm(i4,m4);
```

```
ax0=sr0;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;


mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 -  ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 - ay0;

i0=63;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=63;
dm(i0,m0)=ar;
i0=61;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=60;
```

```
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=65;
dm(i0,m0)=ar;


i0=65;
i4=4161;

{ 4th order biquad }

mx0=dm(i0,m0),my0=pm(i4,m4);
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);

mx0=dm(i0,m0), my0=pm(i4,m4);
ax0=sr0;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 -  ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
```

```
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 - ay0;

i0=68;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=68;
dm(i0,m0)=ar;
i0=66;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=65;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;


{ 6th order biquad }

i0=70;
dm(i0,m0)=ar;
i0=70;
i4=4166;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);

mx0=dm(i0,m0), my0=pm(i4,m4);
ax0=sr0;
ay0=my0;
ax1=px;
ar=ax1 + ay0;
ay0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ay0=my0;
ax1=px;
ar=ax1 + ay0;
```

```
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ay0=my0;
ax1=px;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 -  ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ay0=my0;
ax1=px;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 - ay0;

i0=73;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=73;
dm(i0,m0)=ar;
i0=71;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=70;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;

{ 8th order biquad  }

i0=75;
dm(i0,m0)=ar;
i0=75;
i4=4171;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
```

```
sr=lshift mr0 by -1(lo);

mx0=dm(i0,m0), my0=pm(i4,m4);
ax0=sr0;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax0 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 -  ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax0 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 - ay0;
dm(i7,m7)=ar;

i0=78;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=78;
dm(i0,m0)=ar;
i0=76;
```

```
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=75;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;



{ 5th FILTER OF THE OCTAVE }

i0=0;
ax0=dm(i0,m0);
i0=80;
dm(i0,m0)=ax0;
i0=80;
i4=4176;
ax0=0;

{ 2nd order biquad }

mx0=dm(i0,m0),my0=pm(i4,m4);
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);


mx0=dm(i0,m0), my0=pm(i4,m4);
ax0=sr0;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;


mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
```

```
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 -  ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 - ay0;

i0=83;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=83;
dm(i0,m0)=ar;
i0=81;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=80;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=85;
dm(i0,m0)=ar;


i0=85;
i4=4181;

{ 4th order biquad }

mx0=dm(i0,m0),my0=pm(i4,m4);
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);

mx0=dm(i0,m0), my0=pm(i4,m4);
ax0=sr0;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
```

```
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 -  ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 - ay0;

i0=88;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=88;
dm(i0,m0)=ar;
i0=86;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=85;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;


{ 6th order biquad }
```

```
i0=90;
dm(i0,m0)=ar;
i0=90;
i4=4186;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);

mx0=dm(i0,m0), my0=pm(i4,m4);
ax0=sr0;
ay0=my0;
ax1=px;
ar=ax1 + ay0;
ay0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ay0=my0;
ax1=px;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ay0=my0;
ax1=px;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 -  ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ay0=my0;
ax1=px;
ar=ax1 + ay0;
my0=ar;
```

```
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 - ay0;

i0=93;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=93;
dm(i0,m0)=ar;
i0=91;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=90;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;

{ 8th order biquad }

i0=95;
dm(i0,m0)=ar;
i0=95;
i4=4191;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);

mx0=dm(i0,m0), my0=pm(i4,m4);
ax0=sr0;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;
```

```
mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax0 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 -  ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax0 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 - ay0;
dm(i7,m7)=ar;

i0=98;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=98;
dm(i0,m0)=ar;
i0=96;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=95;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;



{ 6th FILTER OF THE OCTAVE  }

i0=0;
ax0=dm(i0,m0);
i0=100;
dm(i0,m0)=ax0;
i0=100;
i4=4196;
ax0=0;

{ 2nd order biquad }

mx0=dm(i0,m0),my0=pm(i4,m4);
ax1=px;
ay0=my0;
ar=ax1 + ay0;
```

```
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);


mx0=dm(i0,m0), my0=pm(i4,m4);
ax0=sr0;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;


mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 -  ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 - ay0;

i0=103;
ax0=dm(i0,m0);
```

```
dm(i0,m0)=ax0;
i0=103;
dm(i0,m0)=ar;
i0=101;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=100;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=105;
dm(i0,m0)=ar;


i0=105;
i4=4201;

{ 4th order biquad }

mx0=dm(i0,m0),my0=pm(i4,m4);
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);

mx0=dm(i0,m0), my0=pm(i4,m4);
ax0=sr0;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
```

```
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 -  ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 - ay0;

i0=108;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=108;
dm(i0,m0)=ar;
i0=106;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=105;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;


{ 6th order biquad }

i0=110;
dm(i0,m0)=ar;
i0=110;
i4=4206;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);

mx0=dm(i0,m0), my0=pm(i4,m4);
ax0=sr0;
ay0=my0;
ax1=px;
ar=ax1 + ay0;
ay0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
```

```
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ay0=my0;
ax1=px;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ay0=my0;
ax1=px;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 -  ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ay0=my0;
ax1=px;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 - ay0;

i0=113;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=113;
dm(i0,m0)=ar;
i0=111;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=110;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;

{ 8th order biquad }

i0=115;
dm(i0,m0)=ar;
i0=115;
i4=4211;
```

```
mx0=dm(i0,m0),my0=pm(i4,m4);
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);

mx0=dm(i0,m0), my0=pm(i4,m4);
ax0=sr0;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax0 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 -  ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax0 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 - ay0;
dm(i7,m7)=ar;
```

```
i0=118;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=118;
dm(i0,m0)=ar;
i0=116;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=115;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;



{ 7th FILTER OF THE OCTAVE }

i0=0;
ax0=dm(i0,m0);
i0=120;
dm(i0,m0)=ax0;
i0=120;
i4=4216;
ax0=0;

{ 2nd order biquad }

mx0=dm(i0,m0),my0=pm(i4,m4);
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);


mx0=dm(i0,m0), my0=pm(i4,m4);
ax0=sr0;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;


mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
```

```
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 -  ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 - ay0;

i0=123;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=123;
dm(i0,m0)=ar;
i0=121;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=120;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=125;
dm(i0,m0)=ar;


i0=125;
i4=4221;

{ 4th order biquad }

mx0=dm(i0,m0),my0=pm(i4,m4);
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
```

```
sr=lshift mr0 by -1(lo);

mx0=dm(i0,m0), my0=pm(i4,m4);
ax0=sr0;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 -  ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 - ay0;

i0=128;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=128;
dm(i0,m0)=ar;
i0=126;
ax0=dm(i0,m0);
```

```
dm(i0,m0)=ax0;
i0=125;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;


{ 6th order biquad }

i0=130;
dm(i0,m0)=ar;
i0=130;
i4=4226;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);

mx0=dm(i0,m0), my0=pm(i4,m4);
ax0=sr0;
ay0=my0;
ax1=px;
ar=ax1 + ay0;
ay0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ay0=my0;
ax1=px;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ay0=my0;
ax1=px;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 -  ay0;
```

```
mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ay0=my0;
ax1=px;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 - ay0;

i0=133;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=133;
dm(i0,m0)=ar;
i0=131;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=130;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;

{ 8th order biquad }

i0=135;
dm(i0,m0)=ar;
i0=135;
i4=4231;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);

mx0=dm(i0,m0), my0=pm(i4,m4);
ax0=sr0;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
```

```
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax0 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 -  ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax0 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 - ay0;
dm(i7,m7)=ar;

i0=138;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=138;
dm(i0,m0)=ar;
i0=136;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=135;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;


{ 8th FILTER OF THE OCTAVE }


i0=0;
ax0=dm(i0,m0);
i0=140;
dm(i0,m0)=ax0;
i0=140;
i4=4236;
ax0=0;
```

```
{ 2nd order biquad }

mx0=dm(i0,m0),my0=pm(i4,m4);
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);


mx0=dm(i0,m0), my0=pm(i4,m4);
ax0=sr0;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;


mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 -  ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
```

```
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 - ay0;

i0=143;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=143;
dm(i0,m0)=ar;
i0=141;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=140;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=145;
dm(i0,m0)=ar;


i0=145;
i4=4241;

{ 4th order biquad }

mx0=dm(i0,m0),my0=pm(i4,m4);
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);

mx0=dm(i0,m0), my0=pm(i4,m4);
ax0=sr0;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;
```

```
mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 -  ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 - ay0;

i0=148;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=148;
dm(i0,m0)=ar;
i0=146;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=145;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;


{ 6th order biquad }

i0=150;
dm(i0,m0)=ar;
i0=150;
i4=4246;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);

mx0=dm(i0,m0), my0=pm(i4,m4);
ax0=sr0;
ay0=my0;
```

```
ax1=px;
ar=ax1 + ay0;
ay0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ay0=my0;
ax1=px;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ay0=my0;
ax1=px;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 -  ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ay0=my0;
ax1=px;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 - ay0;

i0=153;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=153;
dm(i0,m0)=ar;
i0=151;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=150;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
```

```
{ 8th order biquad }

i0=155;
dm(i0,m0)=ar;
i0=155;
i4=4251;

mx0=dm(i0,m0),my0=pm(i4,m4);
 ax1=px;
 ay0=my0;
 ar=ax1 + ay0;
 my0=ar;
 px=0;
 mr=mx0 * my0(uu);
 sr=lshift mr0 by -1(lo);

mx0=dm(i0,m0), my0=pm(i4,m4);
 ax0=sr0;
 ax1=px;
 ay0=my0;
 ar=ax1 + ay0;
 my0=ar;
 px=0;
 mr=mx0 * my0(uu);
 sr=lshift mr0 by -1(lo);
 ay0=sr0;
 ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
 ax0=ar;
 ax1=px;
 ay0=my0;
 ar=ax1 + ay0;
 my0=ar;
 px=0;
 mr=mx0 * my0(uu);
 sr=lshift mr0 by -1(lo);
 ay0=sr0;
 ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
 ax0=ar;
 ax1=px;
 ay0=my0;
 ar=ax0 + ay0;
 my0=ar;
 px=0;
 mr=mx0 * my0(uu);
 sr=lshift mr0 by -1(lo);
 ay0=sr0;
 ar=ax0 -  ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
 ax0=ar;
 ax1=px;
 ay0=my0;
 ar=ax0 + ay0;
```

```
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 - ay0;
dm(i7,m7)=ar;

i0=158;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=158;
dm(i0,m0)=ar;
i0=156;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=155;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;


{ 9th FILTER OF THE OCTAVE }

i0=0;
ax0=dm(i0,m0);
i0=160;
ax0=dm(i0,m0);
i0=160;
i4=4256;
ax0=0;

{ 2nd order biquad }

mx0=dm(i0,m0),my0=pm(i4,m4);
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);


mx0=dm(i0,m0), my0=pm(i4,m4);
ax0=sr0;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;


mx0=dm(i0,m0),my0=pm(i4,m4);
```

```
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 -  ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 - ay0;

i0=163;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=163;
dm(i0,m0)=ar;
i0=161;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=160;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=165;
dm(i0,m0)=ar;
i0=165;
i4=4261;

{ 4th order biquad }

mx0=dm(i0,m0),my0=pm(i4,m4);
ax1=px;
ay0=my0;
ar=ax1 + ay0;
```

```
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);

mx0=dm(i0,m0), my0=pm(i4,m4);
ax0=sr0;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 -  ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 - ay0;

i0=168;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=168;
```

```
dm(i0,m0)=ar;
i0=166;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=165;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;


{ 6th order biquad }

i0=170;
dm(i0,m0)=ar;
i0=170;
i4=4266;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);

mx0=dm(i0,m0), my0=pm(i4,m4);
ax0=sr0;
ay0=my0;
ax1=px;
ar=ax1 + ay0;
ay0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ay0=my0;
ax1=px;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ay0=my0;
ax1=px;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
```

```
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 -  ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ay0=my0;
ax1=px;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 - ay0;

i0=173;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=173;
dm(i0,m0)=ar;
i0=171;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=170;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;

{ 8th order biquad  }

i0=175;
dm(i0,m0)=ar;
i0=175;
i4=4271;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);

mx0=dm(i0,m0), my0=pm(i4,m4);
ax0=sr0;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
```

```
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax0 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 -  ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax0 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 - ay0;
dm(i7,m7)=ar;

i0=178;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=178;
dm(i0,m0)=ar;
i0=176;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=175;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;


{ 10th FILTER OF THE OCTAVE }

i0=0;
ax0=dm(i0,m0);
i0=180;
dm(i0,m0)=ax0;
i0=180;
i4=4276;
```

```
ax0=0;

{ 2nd order biquad }

mx0=dm(i0,m0),my0=pm(i4,m4);
 ax1=px;
 ay0=my0;
 ar=ax1 + ay0;
 my0=ar;
 px=0;
 mr=mx0 * my0(uu);
 sr=lshift mr0 by -1(lo);


mx0=dm(i0,m0), my0=pm(i4,m4);
 ax0=sr0;
 ax1=px;
 ay0=my0;
 ar=ax1 + ay0;
 my0=ar;
 px=0;
 mr=mx0 * my0(uu);
 sr=lshift mr0 by -1(lo);
  ay0=sr0;
  ar=ax0 + ay0;


mx0=dm(i0,m0),my0=pm(i4,m4);
 ax0=ar;
 ax1=px;
 ay0=my0;
 ar=ax1 + ay0;
 my0=ar;
 px=0;
 mr=mx0 * my0(uu);
 sr=lshift mr0 by -1(lo);
  ay0=sr0;
  ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
 ax0=ar;
 ax1=px;
 ay0=my0;
 ar=ax1 + ay0;
 my0=ar;
 px=0;
 mr=mx0 * my0(uu);
 sr=lshift mr0 by -1(lo);
  ay0=sr0;
  ar=ax0 -  ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
 ax0=ar;
 ax1=px;
 ay0=my0;
 ar=ax1 + ay0;
 my0=ar;
```

```
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 - ay0;

i0=183;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=183;
dm(i0,m0)=ar;
i0=181;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=180;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=185;
dm(i0,m0)=ar;


i0=185;
i4=4281;

{ 4th order biquad }

mx0=dm(i0,m0),my0=pm(i4,m4);
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);

mx0=dm(i0,m0), my0=pm(i4,m4);
ax0=sr0;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
```

```
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 -  ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 - ay0;

i0=188;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=188;
dm(i0,m0)=ar;
i0=186;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=185;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;


{ 6th order biquad }

i0=190;
dm(i0,m0)=ar;
i0=190;
i4=4286;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);

mx0=dm(i0,m0), my0=pm(i4,m4);
ax0=sr0;
```

```
ay0=my0;
ax1=px;
ar=ax1 + ay0;
ay0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ay0=my0;
ax1=px;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ay0=my0;
ax1=px;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 -  ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ay0=my0;
ax1=px;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 - ay0;

i0=193;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=193;
dm(i0,m0)=ar;
i0=191;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=190;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
```

```
{ 8th order biquad }

i0=195;
dm(i0,m0)=ar;
i0=195;
i4=4291;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);

mx0=dm(i0,m0), my0=pm(i4,m4);
ax0=sr0;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax0 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 -  ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
```

```
ar=ax0 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 - ay0;
dm(i7,m7)=ar;

i0=198;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=198;
dm(i0,m0)=ar;
i0=196;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=195;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;

{ 11th FILTER OF THE OCTAVE }

i0=0;
ax0=dm(i0,m0);
i0=200;
dm(i0,m0)=ax0;
i0=200;
i4=4296;
ax0=0;

{ 2nd order biquad }

mx0=dm(i0,m0),my0=pm(i4,m4);
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);


mx0=dm(i0,m0), my0=pm(i4,m4);
ax0=sr0;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;
```

```
mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 -  ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 - ay0;

i0=203;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=203;
dm(i0,m0)=ar;
i0=201;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=200;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=205;
dm(i0,m0)=ar;


i0=205;
i4=4301;

{ 4th order biquad }

mx0=dm(i0,m0),my0=pm(i4,m4);
```

```
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);

mx0=dm(i0,m0), my0=pm(i4,m4);
ax0=sr0;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 -  ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 - ay0;

i0=208;
```

```
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=208;
dm(i0,m0)=ar;
i0=206;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=205;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;


{ 6th order biquad }

i0=210;
dm(i0,m0)=ar;
i0=210;
i4=4306;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);

mx0=dm(i0,m0), my0=pm(i4,m4);
ax0=sr0;
ay0=my0;
ax1=px;
ar=ax1 + ay0;
ay0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ay0=my0;
ax1=px;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ay0=my0;
ax1=px;
ar=ax1 + ay0;
```

```
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 -  ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ay0=my0;
ax1=px;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 - ay0;

i0=213;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=213;
dm(i0,m0)=ar;
i0=211;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=210;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;

{ 8th order biquad }

i0=215;
dm(i0,m0)=ar;
i0=215;
i4=4311;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);

mx0=dm(i0,m0), my0=pm(i4,m4);
ax0=sr0;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
```

```
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax0 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 -  ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax0 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 - ay0;
dm(i7,m7)=ar;

i0=218;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=218;
dm(i0,m0)=ar;
i0=216;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=215;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;


{ 12th FILTER OF THE OCTAVE }

i0=0;
ax0=dm(i0,m0);
i0=220;
```

```
dm(i0,m0)=ax0;
 i0=220;
 i4=4316;
 ax0=0;

 { 2nd order biquad }

mx0=dm(i0,m0),my0=pm(i4,m4);
 ax1=px;
 ay0=my0;
 ar=ax1 + ay0;
 my0=ar;
 px=0;
 mr=mx0 * my0(uu);
 sr=lshift mr0 by -1(lo);


mx0=dm(i0,m0), my0=pm(i4,m4);
 ax0=sr0;
 ax1=px;
 ay0=my0;
 ar=ax1 + ay0;
 my0=ar;
 px=0;
 mr=mx0 * my0(uu);
 sr=lshift mr0 by -1(lo);
 ay0=sr0;
 ar=ax0 + ay0;


mx0=dm(i0,m0),my0=pm(i4,m4);
 ax0=ar;
 ax1=px;
 ay0=my0;
 ar=ax1 + ay0;
 my0=ar;
 px=0;
 mr=mx0 * my0(uu);
 sr=lshift mr0 by -1(lo);
 ay0=sr0;
 ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
 ax0=ar;
 ax1=px;
 ay0=my0;
 ar=ax1 + ay0;
 my0=ar;
 px=0;
 mr=mx0 * my0(uu);
 sr=lshift mr0 by -1(lo);
 ay0=sr0;
 ar=ax0 -  ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
 ax0=ar;
 ax1=px;
```

```
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 - ay0;

i0=223;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=223;
dm(i0,m0)=ar;
i0=221;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=220;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=225;
dm(i0,m0)=ar;


i0=225;
i4=4321;

{ 4th order biquad }

mx0=dm(i0,m0),my0=pm(i4,m4);
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);

mx0=dm(i0,m0), my0=pm(i4,m4);
ax0=sr0;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
```

```
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 -  ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax0=ar;
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
ay0=sr0;
ar=ax0 - ay0;

i0=228;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=228;
dm(i0,m0)=ar;
i0=226;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;
i0=225;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;


{ 6th order biquad }

i0=230;
dm(i0,m0)=ar;
i0=230;
i4=4326;

mx0=dm(i0,m0),my0=pm(i4,m4);
ax1=px;
ay0=my0;
ar=ax1 + ay0;
my0=ar;
px=0;
mr=mx0 * my0(uu);
sr=lshift mr0 by -1(lo);
```

```
mx0=dm(i0,m0), my0=pm(i4,m4);
 ax0=sr0;
 ay0=my0;
 ax1=px;
 ar=ax1 + ay0;
 ay0=ar;
 px=0;
 mr=mx0 * my0(uu);
 sr=lshift mr0 by -1(lo);
 ay0=sr0;
 ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
 ax0=ar;
 ay0=my0;
 ax1=px;
 ar=ax1 + ay0;
 my0=ar;
 px=0;
 mr=mx0 * my0(uu);
 sr=lshift mr0 by -1(lo);
 ay0=sr0;
 ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
 ax0=ar;
 ay0=my0;
 ax1=px;
 ar=ax1 + ay0;
 my0=ar;
 px=0;
 mr=mx0 * my0(uu);
 sr=lshift mr0 by -1(lo);
 ay0=sr0;
 ar=ax0 -  ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
 ax0=ar;
 ay0=my0;
 ax1=px;
 ar=ax1 + ay0;
 my0=ar;
 px=0;
 mr=mx0 * my0(uu);
 sr=lshift mr0 by -1(lo);
 ay0=sr0;
 ar=ax0 - ay0;

 i0=233;
 ax0=dm(i0,m0);
 dm(i0,m0)=ax0;
 i0=233;
 dm(i0,m0)=ar;
 i0=231;
 ax0=dm(i0,m0);
 dm(i0,m0)=ax0;
```

```
i0=230;
ax0=dm(i0,m0);
dm(i0,m0)=ax0;

{ 8th order biquad }

i0=235;
dm(i0,m0)=ar;
i0=235;
i4=4331;

mx0=dm(i0,m0),my0=pm(i4,m4);
 ax1=px;
 ay0=my0;
 ar=ax1 + ay0;
 my0=ar;
 px=0;
 mr=mx0 * my0(uu);
 sr=lshift mr0 by -1(lo);

mx0=dm(i0,m0), my0=pm(i4,m4);
 ax0=sr0;
 ax1=px;
 ay0=my0;
 ar=ax1 + ay0;
 my0=ar;
 px=0;
 mr=mx0 * my0(uu);
 sr=lshift mr0 by -1(lo);
 ay0=sr0;
 ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
 ax0=ar;
 ax1=px;
 ay0=my0;
 ar=ax1 + ay0;
 my0=ar;
 px=0;
 mr=mx0 * my0(uu);
 sr=lshift mr0 by -1(lo);
 ay0=sr0;
 ar=ax0 + ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
 ax0=ar;
 ax1=px;
 ay0=my0;
 ar=ax0 + ay0;
 my0=ar;
 px=0;
 mr=mx0 * my0(uu);
 sr=lshift mr0 by -1(lo);
 ay0=sr0;
 ar=ax0 -  ay0;

mx0=dm(i0,m0),my0=pm(i4,m4);
```

```
        ax0=ar;
        ax1=px;
        ay0=my0;
        ar=ax0 + ay0;
        my0=ar;
        px=0;
        mr=mx0 * my0(uu);
        sr=lshift mr0 by -1(lo);
        ay0=sr0;
        ar=ax0 - ay0;
        dm(i7,m7)=ar;

        i0=238;
        ax0=dm(i0,m0);
        dm(i0,m0)=ax0;
        i0=238;
        dm(i0,m0)=ar;
        i0=236;
        ax0=dm(i0,m0);
        dm(i0,m0)=ax0;
        i0=235;
        ax0=dm(i0,m0);
loop:   dm(i0,m0)=ax0;

        rts;
.endmod;
```

Chapter 6

Conclusion

6.CONCLUSION

The filter bank implemented in the octave range (1KHZ - 2KHz) can be extended to the entire audio range. The analysis of each of the octaves in the audio range is realised through the construction of low pass filters for each of these octaves. These lowpass filters are beneficial in reducing the overall processing time for the entire acoustic range. This system can also be realised on ADSP 2100 hardware board adequately supported with necessary software.

The power spectral analyzer of such a kind finds its application in the following thrust areas.

* Noise measurement in industries.

* Sonic boom measurement in laboratory equipments.

* Vibration and Acoustic measurements in onboard and ground systems in space communication.

* Evaluation of acoustic strength of relic buildings.

Bibliography

BIBLIOGRAPHY

1. A.V. Oppenheim and Ronald. W. Schafer.
"Digital signal processing". Englewood Cliffs, N.J. Prentice Hall , 1975

2. L.R. Rabiner and B. Gold.
" Theory and application of digital signal processing ".
Englewood Cliffs, N.J. Prentice Hall, 1975.

3. David J. Defatta, Joseph C. Lucas, William . S. Hodgkiss.
" Digital signal processing - a system design approach". John Wiley & Sons, 1988.

4. Andrew Bateman & Warren Yates.
"Digital signal processing design".A.H. Wheeler & Co Ltd. ,1988.

5. "Digital signal processing applications using the ADSP - 2100 family by the Applications engineering staff off Analog Devices Inc. DSP division; 1990.

Annexure

SYSTEM SPECIFICATION (.SYS) FILE

```
.SYSTEM iir_system;

.ADSP2100;

.MMAP1;

.SEG/PM/ABS=0/RAM/CODE program_mem[4096];

.SEG/PM/ABS=4096/RAM/DATA coeff_storage[256];

.SEG/DM/ABS=0/RAM/DATA delay_line[300];

.PORT/ABS=16382 ad_sample;

.PORT/ABS=16383 da_data;


.ENDSYS;
```

ADSP 2100 Command Summary.

❊ To run the system builder,
```
>BLD 21 [ - SWITCHES ] filename . SYS
```

❊ To run the assembler
```
> ASM 21 [ -SWITCHES]  filename .DSP {When using assembly code programs }
>CC21 [ -  SWITCHES]  filename . C        { When using  C programing }
```

❊ To run the linker
```
> LD 21 [ - SWITCHES] filename.
```

❊ To run the simulator
```
>SIM 2100 [ - SWITCHES] filename .ACH
```

Command Window Command Summary:-

❊ To load a program into program memory

```
> L 'filename'
```

❊ To inspect a memory Location / Register

```
> ?  Address / Register
```

❊ To alter a memory location

```
> E Address expression.
```

❊ To execute in single step [or multiple steps]

```
> S [number of steps]
```

❊ To execute the entire program in single step

```
> G
```

❊ To halt execution

```
> H
```

- To plot memory

```
> PL range decimation
```

- To temperarily exit to operating system

```
>SH
```

- To quit the simulator

```
>Q  {with verification from user}

>  QUIT {without verification from user}
```

ANALOG DEVICES

LC^2MOS Complete 12-Bit 100kHz Sampling ADC with DSP Interface

AD7878

FEATURES

Complete ADC with DSP Interface, Comprising:
- Track/Hold Amplifier with 2μs Acquisition Time
- 7μs A/D Converter
- 3V Zener Reference
- 8-Word FIFO and Interface Logic

72dB SNR at 10kHz Input Frequency

Interfaces to High Speed DSP Processors, e.g., ADSP-2100, TMS32010, TMS32020

41ns max Data Access Time

Low Power, 60mW typ

APPLICATIONS

Digital Signal Processing

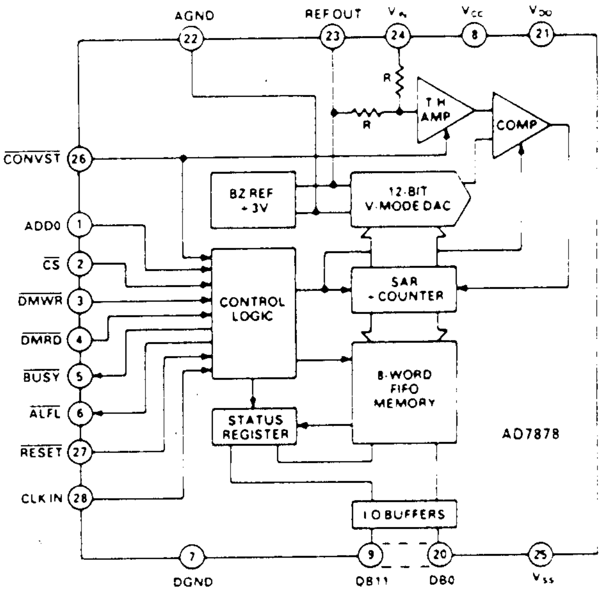
Speech Recognition and Synthesis

Spectrum Analysis

High Speed Modems

DSP Servo Control

AD7878 FUNCTIONAL BLOCK DIAGRAM



ENERAL DESCRIPTION

he AD7878 is a fast complete 12-bit A/D converter with a rsatile DSP interface consisting of an 8-word, first-in, first-out IFO) memory and associated control logic.

he FIFO memory allows up to eight samples to be digitized fore the microprocessor is required to service the A/D converter. he eight words can then be read out of the FIFO at maximum croprocessor speed. A fast data access time of 41ns allows rect interfacing to DSP processors and high speed 16-bit croprocessors.

on-chip status/control register allows the user to program the ective length of the FIFO and contains the FIFO out of age, FIFO empty and FIFO word count information.

e analog input of the AD7878 has a bipolar range of ±3V. e AD7878 can convert full power signals up to 50kHz and is ly specified for dynamic parameters such as signal-to-noise io and harmonic distortion.

e AD7878 is fabricated in Linear Compatible CMOS ²MOS), an advanced, mixed technology process that combines cision bipolar circuits with low power CMOS logic. The part vailable in four package styles, 28-pin plastic and hermetic l-in-line package (DIP), leadless ceramic chip carrier (LCCC) lastic leaded chip carrier (PLCC).

PRODUCT HIGHLIGHTS

1. Complete A/D Function with DSP Interface
   The AD7878 provides the complete function for digitizing ac signals to 12-bit accuracy. The part features an on-chip track/hold, on-chip reference and 12-bit A/D converter. The additional feature of an 8-word FIFO reduces the high software overheads associated with servicing interrupts in DSP processors.
2. Dynamic Specifications for DSP Users
   The AD7878 is fully specified and tested for ac parameters, including signal-to-noise ratio, harmonic distortion and intermodulation distortion. Key digital timing parameters are also tested and specified over the full operating temperature range.
3. Fast Microprocessor Interface
   Data access time of 41ns is the fastest ever achieved in a monolithic A/D converter and makes the AD7878 compatible with all modern 16-bit microprocessors and digital signal processors.

ECIFICATIONS (V_{DD} = +5V ±5%, V_{CC} = +5V ±5%, V_{SS} = −5V ±5%, AGND = DGND = 0V, f_{CLK} = 8MHz. All Specifications T_{min} to T_{max} unless otherwise noted.)

neter	J, A Versions[1]	K, L, B Versions	S Version	Units	Test Conditions/Comments
AMIC PERFORMANCE[2]					
al-to-Noise Ratio (SNR)[3] @ 25°C	70	72	70	dB min	V_{IN} = 10kHz sine wave, f_{SAMPLE} = 100kHz
in to T_{max}	70	71	70	dB min	Typically 71.5dB for 0<V_{IN}<50kHz
al Harmonic Distortion (THD)	−80	−80	−80	dB max	V_{IN} = 10kHz sine wave, f_{SAMPLE} = 100kHz Typically −86dB for 0<V_{IN}<50kHz
k Harmonic or Spurious Noise	−80	−80	−80	dB max	V_{IN} = 10kHz, f_{SAMPLE} = 100kHz Typically −86dB for 0<V_{IN}<50kHz
ermodulation Distortion (IMD)					
Second Order Terms	−80	−80	−80	dB max	fa = 9kHz, fb = 9.5kHz, f_{SAMPLE} = 50kHz
Third Order Terms	−80	−80	−80	dB max	fa = 9kHz, fb = 9.5kHz, f_{SAMPLE} = 50kHz
ack Hold Acquisition Time	2	2	2	μs max	See Throughput Rate section.
ACCURACY					
solution	12	12	12	Bits	
imum Resolution for which No Missing Codes are Guaranteed	12	12	12	Bits	
ative Accuracy	±1/2	±1/4	±1/2	LSB typ	
ferential Nonlinearity	±1/2	±1/2	±1/2	LSB typ	
olar Zero Error	±6	±6	±6	LSB max	
sitive Full Scale Error[4]	±6	±6	±6	LSB max	
gative Full Scale Error[4]	±6	±6	±6	LSB max	
LOG INPUT					
ut Voltage Range	±3	±3	±3	Volts	
ut Current	±550	±550	±550	μA max	
ERENCE OUTPUT[5]					
F OUT	3	3	3	V nom	
F OUT Error @ 25°C	±10	±10	±10	mV max	
T_{min} to T_{max}	±15	±15	±15	mV max	
ference Load Sensitivity ΔREF OUT ΔI	±1	±1	±1	mV max	Reference Load Current change (0−500μA). Reference Load should not be changed during conversion.
IC INPUTS					
ut High Voltage, V_{INH}	+2.4	+2.4	+2.4	V min	V_{CC} = +5V ±5%
ut Low Voltage, V_{INL}	+0.8	+0.8	+0.8	V max	V_{CC} = +5V ±5%
ut Current, I_{IN}	±10	±10	±10	μA max	V_{IN} = 0 to V_{CC}
ut Capacitance, C_{IN}[6]	10	10	10	pF max	
IC OUTPUTS					
put High Voltage, V_{OH}	+2.7	+2.7	+2.7	V min	I_{SOURCE} = 40μA
put Low Voltage, V_{OL}	+0.4	+0.4	+0.4	V max	I_{SINK} = 1.6mA
B11 − DB0					
Floating State Leakage Current	±10	±10	±10	±10	μA max
Floating State Output Capacitance[6]	15	15	15	15	pF max
NVERSION TIME	7/7.125	7/7.125	7/7.125	μs min/μs max	Assuming no external Read/Write operations
	7/9.250	7/9.250	7/9.250	μs min/μs max	Assuming 17 external Read/Write operations See Internal Comparator Timing section
WER REQUIREMENTS					
	+5	+5	+5	V nom	±5% for specified performance
	+5	+5	+5	V nom	±5% for specified performance
	−5	−5	−5	V nom	±5% for specified performance
	13	13	13	mA max	$\overline{CS}$ = $\overline{DMWR}$ = $\overline{DMRD}$ = 5V
	100	100	100	μA max	$\overline{CS}$ = $\overline{DMWR}$ = $\overline{DMRD}$ = 5V
	6	6	6	mA max	$\overline{CS}$ = $\overline{DMWR}$ = $\overline{DMRD}$ = 5V
wer Dissipation	95.5	95.5	95.5	mW max	Typically 60mW

es

[1] perature range as follows:
K, L versions: 0 to +70°C
B versions: −25°C to +85°C
rsion: −55°C to +125°C

[2] +5V. See Dynamic Specifications section.

[3] R calculation includes distortion and noise components.

[4] ured with respect to the Internal Reference.

[5] apacitive Loads greater than 50pF a series resistor is required (see Internal Reference section).

[6] e tested @ 25°C to ensure compliance.

ications subject to change without notice.

IMING CHARACTERISTICS[1] ($V_{DD} = 5V \pm 5\%, V_{CC} = 5V \pm 5\%, V_{SS} = -5V \pm 5\%$)

ameter	Limit at T_{min}, T_{max} (L Grade)	Limit at T_{min}, T_{max} (J,K,A,B Grades)	Limit at T_{min}, T_{max} (S Grade)	Units	Conditions/Comments
	65	65	75	ns max	CLK IN to $\overline{\text{BUSY}}$ Low Propagation Delay
	65	65	75	ns max	CLK IN to $\overline{\text{BUSY}}$ High Propagation Delay
	2 CLK IN cycles	2 CLK IN cycles	2 CLK IN cycles	min	$\overline{\text{CONVST}}$ Pulse Width
	0	0	0	ns min	$\overline{\text{CS}}$ to $\overline{\text{DMRD}}$/REGISTER ENABLE Setup Time
	0	0	0	ns min	$\overline{\text{CS}}$ to $\overline{\text{DMRD}}$/REGISTER ENABLE Hold Time
	45	**60**	**60**	ns min	$\overline{\text{DMRD}}$ Pulse Width
	50	50	50	µs max	
	16	16	16	ns min	ADD0 to $\overline{\text{DMRD}}$/REGISTER ENABLE Setup Time
	0	0	0	ns min	ADD0 to $\overline{\text{DMRD}}$/REGISTER ENABLE Hold Time
	41	57	57	ns min	Data Access Time after $\overline{\text{DMRD}}$
	5	5	5	ns min	Bus Relinquish Time
	45	45	45	ns max	
	42	42	55	ns min	REGISTER ENABLE Pulse Width
	50	50	50	µs max	
	20	20	30	ns min	Data Valid to REGISTER ENABLE Setup Time
	10	10	10	ns min	Data Hold Time after REGISTER ENABLE
	41	57	57	ns min	Data Access Time after $\overline{\text{BUSY}}$

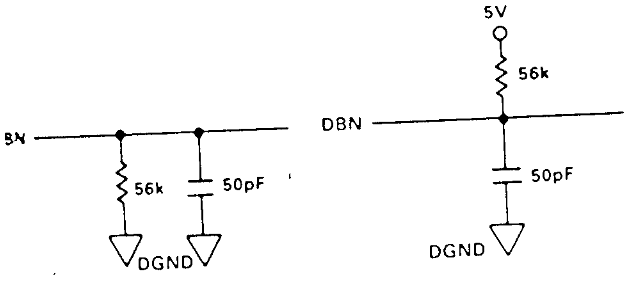
DTES

[1]iming Specifications in bold print are 100% production tested. All other times are sample tested at +25°C to ensure compliance. All input signals are specified with = tf = 5ns (10% to 90% of 5V) and timed from a voltage lêvel of 1.6V.

and t_{14} are measured with the load circuits of Figure 1 and defined as the time required for an output to cross 0.8V or 2.4V.

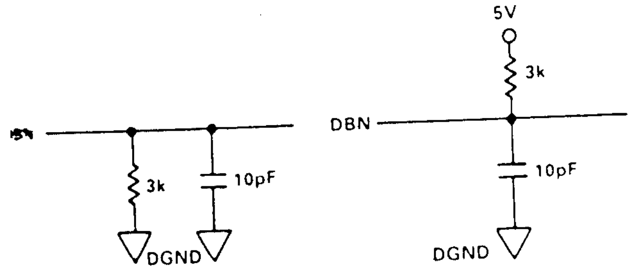
is defined as the time required for the data lines to change 0.5V when loaded with the circuits of Figure 2.

ecifications subject to change without notice.



a. High-Z to V_{OH} b. High-Z to V_{OL}

Figure 1. Load Circuits for Access Time



a. V_{OH} to High-Z b. V_{OL} to High-Z

Figure 2. Load Circuits for Output Float Delay

ABSOLUTE MAXIMUM RATINGS*

(T_A = +25°C unless otherwise stated)

V_{DD} to DGND	−0.3V to +7V
V_{CC} to DGND	−0.3V to +7V
V_{SS} to DGND	+0.3V to −7V
V_{DD} to V_{CC}	−0.3V to +0.3V
AGND to DGND	−0.3V to V_{DD} +0.3V
V_{IN} to AGND	−15V to +15V
REF OUT to AGND	0 to V_{DD}
Digital Inputs to DGND CLK IN, $\overline{\text{DMWR}}$, $\overline{\text{DMRD}}$, $\overline{\text{RESET}}$, $\overline{\text{CS}}$, $\overline{\text{CONVST}}$, ADD0	−0.3V to V_{DD} +0.3V
Digital Outputs to DGND $\overline{\text{ALFL}}$, $\overline{\text{BUSY}}$	−0.3V to V_{DD} +0.3V
Data Pins DB11 − DB0	−0.3V to V_{DD} +0.3V
Operating Temperature Range J, K, L Versions	0 to +70°C
A, B Versions	−25°C to +85°C
S Version	−55°C to +125°C
Storage Temperature Range	−65°C to +150°C
Lead Temperature (Soldering, 10secs)	+300°C
Power Dissipation (Any Package) to +75°C	1000mW
Derates above +75°C by	10mW/°C

*Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

AUTION

ESD electrostatic discharge) sensitive device. The digital control inputs are diode protect-
however, permanent damage may occur on unconnected devices subject to high energy
ectrostatic fields. Unused devices must be stored in conductive foam or shunts. The protective
should be discharged to the destination socket before devices are removed.

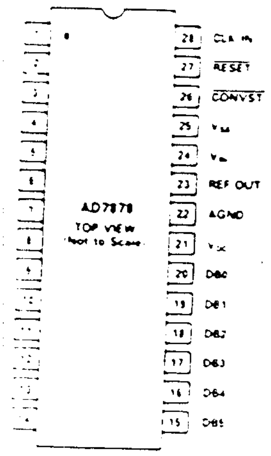
WARNING!
ESD SENSITIVE DEVICE

PIN FUNCTION DESCRIPTION

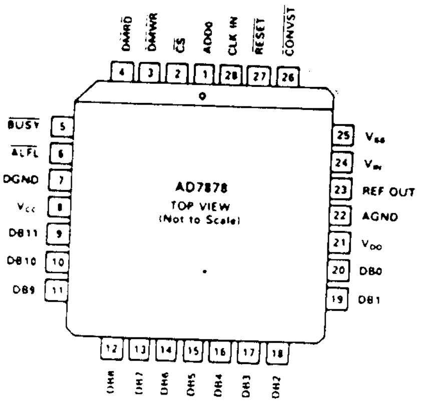
Pin Number	Pin Mnemonic	Function
	ADD0	Address Input. This control input determines whether the word placed on the output data bus during a read operation is a data word from the FIFO RAM or the contents of the status/control register. A logic low accesses the data word from Location 0 of the FIFO while a logic high selects the contents of the register (see Status/Control Register section).
	$\overline{\text{CS}}$	Chip Select. Active low logic input. The device is selected when this input is active.
	$\overline{\text{DMWR}}$	Data Memory Write. Active low logic input. $\overline{\text{DMWR}}$ is used in conjunction with $\overline{\text{CS}}$ low and ADD0 high to write data to the status control register. Corresponds to $\overline{\text{DMWR}}$ (ADSP-2100), R/$\overline{\text{W}}$ (MC68000, TMS32020), $\overline{\text{WE}}$ (TMS32010).
	$\overline{\text{DMRD}}$	Data Memory READ. Active low logic input. $\overline{\text{DMRD}}$ is used in conjunction with $\overline{\text{CS}}$ low to enable the three-state output buffers. Corresponds directly to $\overline{\text{DMRD}}$ (ADSP-2100), $\overline{\text{DEN}}$ (TMS32010).
	$\overline{\text{BUSY}}$	Active low logic output. This output goes low when the ADC receives a $\overline{\text{CONVST}}$ pulse and remains low until the track/hold has gone into its hold mode. The three-state drivers of the AD7878 can be disabled while the $\overline{\text{BUSY}}$ signal is low (see Extended READ/WRITE section). This is achieved by writing a logic 0 to DB5 ($\overline{\text{DISO}}$) of the status/control register. Writing a logic 1 to DB5 of the status control register allows data to be accessed from the AD7878 while $\overline{\text{BUSY}}$ is low.
	$\overline{\text{ALFL}}$	FIFO Almost Full. A logic low indicates that the word count (i.e., number of conversion results) in the FIFO memory has reached the programmed word count in the status/control register. $\overline{\text{ALFL}}$ is updated at the end of each conversion. The $\overline{\text{ALFL}}$ output is reset to a logic high when a word is read from the FIFO memory. It can also be set high by writing a logic 1 to DB7 ($\overline{\text{ENAF}}$) of the status/control register.
	DGND	Digital Ground. Ground reference for digital circuitry.
	V_{CC}	Digital supply voltage, + 5V ± 5%. Positive supply voltage for digital circuitry.
	DB11	Data Bit 11 (MSB). Three-state TTL output. Coding for the data words in FIFO RAM is 2s complement.
	DB10-DB5	Data Bit 10 to Data Bit 5. Three-state TTL input/outputs.
	DB4-DB1	Data Bit 4 to Data Bit 1. Three-state TTL outputs.
	DB0	Data Bit 0 (LSB). Three-state TTL output.
	V_{DD}	Analog positive supply voltage, + 5V ± 5%.
	AGND	Analog Ground. Ground reference for track/hold, reference and DAC.
	REF OUT	Voltage Reference Output. The internal 3V analog reference is provided at this pin. The external load capability of the reference is 500μA.
	V_{IN}	Analog Input. Analog input range is ± 3V.
	V_{SS}	Analog negative supply voltage, − 5V ± 5%.
	$\overline{\text{CONVST}}$	Convert Start. Logic input. A low to high transition on this input puts the track/hold into its hold mode and starts conversion. The $\overline{\text{CONVST}}$ input is asynchronous to CLK IN and independent of $\overline{\text{CS}}$, $\overline{\text{DMWR}}$ and $\overline{\text{DMRD}}$.
	$\overline{\text{RESET}}$	Reset. Active low logic input. A logic low sets the words in FIFO memory to 1000 0000 0000 and resets the $\overline{\text{ALFL}}$ output and status/control register.
	CLK IN	Clock Input. TTL-compatible logic input. Used as the clock source for the A/D converter. The mark-space ratio of this clock can vary from 35/65 to 65/35.

PIN CONFIGURATIONS

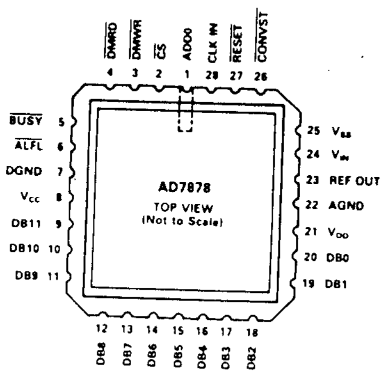
DIP



PLCC



LCCC



STATUS/CONTROL REGISTER

The status/control register serves the dual function of providing control and monitoring the status of the FIFO memory. This register is directly accessible through the data bus (DB11 – DB0) with a read or write operation while ADD0 is high. A write operation to the status/control register provides control for the $\overline{\text{ALFL}}$ output, bus interface and FIFO counter reset. This is normally done on power-up initialization. The FIFO memory address pointer is incremented after each conversion and compared with a preprogrammed count in the status/control register. When this preprogrammed count is reached, the $\overline{\text{ALFL}}$ output is asserted if the $\overline{\text{ENAF}}$ control bit is set to zero. This $\overline{\text{ALFL}}$ can be used to interrupt the microprocessor after any predetermined number of conversions (between 1 and 8). The status of the address pointer along with sample overrange and $\overline{\text{ALFL}}$ status n be accessed at any time by reading the status/control register. Note, reading the status/control register does not cause any ernal data movement in the FIFO memory. Status information a particular word should be read from the status register ore the data word is read from the FIFO memory.

ATUS/CONTROL REGISTER FUNCTION SCRIPTION

11 ($\overline{\text{ALFL}}$)

ost Full Flag, Read only. This the same as Pin 6 ($\overline{\text{ALFL}}$ out) status. A logic low indicates that the word count in the O memory has reached the preprogrammed count in bit tions DB10 – DB8. $\overline{\text{ALFL}}$ is updated at the end of ersion.

0 – DB8 (AFC2 – AFC0)

ost Full Word Count, Read/Write. The count value determines number of words in the FIFO memory which will cause L to be set. When the FIFO word count equals the programmed count in these three bits, then both the $\overline{\text{ALFL}}$ output DB11 of the status register are set to a logic low. For example, a code of 011 is written to these bits, $\overline{\text{ALFL}}$ is set when ion 0 through Location 3 of the FIFO memory contains data. AFC2 is the most significant bit of the word count. The count value can be read back if required.

DB7 ($\overline{\text{ENAF}}$)

Enable Almost Full, Read/Write. Writing a 1 to this bit disables the $\overline{\text{ALFL}}$ output and status register bit DB11.

DB6 (FOVR/RESET)

FIFO Overrun/RESET, Read/Write. Reading a 1 from this bit indicates that at least one sample has been discarded because the FIFO memory is full. When the FIFO is full (i.e., contains eight words) any further conversion results will be lost. Writing a 1 to this bit causes a system RESET as per the $\overline{\text{RESET}}$ input (Pin 27).

DB5 (FOOR/$\overline{\text{DISO}}$)

FIFO Out of RANGE/Disable Outputs, Read/Write. Reading a 1 from this bit indicates that at least one sample in the FIFO memory is out of range. Writing a 0 to this bit prevents the data bus from becoming active while $\overline{\text{BUSY}}$ is low regardless of the state of $\overline{\text{CS}}$ and $\overline{\text{DMRD}}$.

DB4 (FEMP)

FIFO Empty, Read Only. Reading a 1 indicates that there are no samples in the FIFO memory. When the FIFO is empty the internal ripple-down effects of the FIFO are disabled and further reads will continue to access the last valid data word in Location 0.

DB3 (SOOR)

Sample out of Range, Read Only. Reading a 1 indicates that the next sample to be read is out of range, i.e., the sample in Location 0 of the FIFO.

DB2 – DB0 (FCN2 – FCN0)

FIFO Word Count, Read Only. The value read from these bits indicates the number of samples in the FIFO memory. For example, reading 011 from these bits indicates that Location 0 through Location 3 contains valid data. Note, reading all 0s indicates that there is either one word or no word in the FIFO memory; in this case the FIFO Empty determines if there is no word in memory. FCN2 is the most significant bit.

T LOCATION	DB11	DB10	DB9	DB8	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
ATUS INFORMATION (READ)	$\overline{\text{ALFL}}$	AFC2	AFC1	AFC0	$\overline{\text{ENAF}}$	FOVR	FOOR	FEMP	SOOR	FCN2	FCN1	FCN0
NTROL FUNCTION (WRITE)	X	AFC2	AFC1	AFC0	$\overline{\text{ENAF}}$	RESET	$\overline{\text{DISO}}$	X	X	X	X	X
SET STATUS	1	0	0	0	0	0	0	1	0	0	0	0

DON'T CARE

Table I. Status Control Bit Function Description

ORDERING INFORMATION[1]

Signal-to-Noise Ratio	Data Access Time	Temperature Range and Package Options[2]		
		0 to +70°C Plastic DIP (N-28)	−25°C to +85°C Hermetic[3] DIP (Q-28)	−55°C to +125°C Hermetic[3] DIP (Q-28)
70 dB 72 dB 72 dB	57 ns 57 ns 41 ns	AD7878JN AD7878KN AD7878LN	AD7878AQ AD7878BQ	AD7878SQ
		PLCC[4] (P-28A)		LCCC[5] (E-28A)
70 dB 72 dB 72 dB	57 ns 57 ns 41 ns	AD7878JP AD7878KP AD7878LP		AD7878SE

NOTES

[1] To order MIL-STD-883, Class B processed parts, add/883B to part number. Contact our local sales office for military data sheet

[2] See Section 14 for package outline information.

[3] Analog Devices reserves the right to ship either ceramic (D-28) packages or cerdip (Q-28) hermetic packages

[4] PLCC: Plastic Leaded Chip Carrier

[5] LCCC: Leadless Ceramic Chip Carrier. Available to 883B processing only

INTERNAL FIFO MEMORY

The internal FIFO memory of the AD7878 consists of eight memory locations. Each word in memory contains 13 bits of information – 12 bits of data from the conversion result and one additional bit which contains information as to whether the 12-bit result is out of range or not. A block diagram of the AD7878 FIFO architecture is shown in Figure 3.

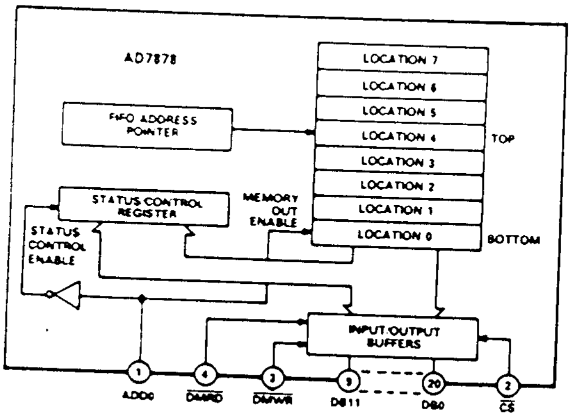


Figure 3. Internal FIFO Architecture

The conversion result is gathered in the successive approximation register (SAR) during conversion. At the end of conversion this result is transferred to the FIFO memory. The FIFO address pointer always points to the top of memory, i.e., the uppermost location which contains valid data. The pointer is incremented after each conversion. A read operation from the FIFO memory accesses data from the bottom of the FIFO, i.e., Location 0. On completion of the read operation each data word moves down one location and the address pointer is decremented by one. Therefore, each conversion result from the SAR enters at the top of memory, propagates down with successive reads until it reaches Location 0 from where it can be accessed by a microprocessor read operation.

The transfer of information from the SAR to the FIFO occurs in synchronization with the AD7878 input clock (CLK IN). The propagation of data words down the FIFO is also synchronous with this clock. As a result, a read operation to obtain data from the FIFO must also be synchronous with CLK IN to avoid Read/Write conflicts in the FIFO (i.e., reading from FIFO Location 0 while it is being updated). This requires that the microprocessor clock and the AD7878 CLK IN are derived from the same source.

INTERNAL COMPARATOR TIMING

The ADC clock, which is applied to CLK IN, controls the successive approximation A/D conversion process. This clock is internally divided by four to yield a bit trial cycle time of 500ns min (CLK IN = 8MHz clock). Each bit decision occurs 25ns after the rising edge of this divided clock. The bit decision is latched by the rising edge of an internal comparator strobe signal. There are 12 bit decisions, as in a normal successive approximation routine, and one extra decision which checks if the input sample is out of range. In a normal successive approximation A/D converter, reading data from the device during conversion can upset the conversion in progress. This is due to on-chip transients, generated by charging or discharging the data bus, concurrent with a bit decision. The scheme outlined below and shown in Figure 4 describes how the AD7878 overcomes this problem.

The internal comparator strobe on the AD7878 is gated with both $\overline{DMRD}$ and $\overline{DMWR}$ so that if a read or write operation occurs when a bit decision is about to be made, the bit decision point is deferred by one CLK IN cycle. In other words, if $\overline{DMRD}$ or $\overline{DMWR}$ goes low (with $\overline{CS}$ low) at any time during the CLK IN low-time immediately prior to the comparator strobing edge (t_{LOW} of Figure 4), the bit trial is suspended for a clock cycle. This makes sure that the bit decision is latched at a time when the AD7878 is not attempting to charge or discharge the data bus, thereby ensuring that no spurious transients occur internally near a bit decision point.

The decision point slippage mechanism is shown in Figure 4 for the MSB decision. Normally, the MSB decision occurs 25ns after the fourth rising CLK IN edge after $\overline{CONVST}$ goes high. However, in the timing diagram of Figure 4, $\overline{CS}$ and $\overline{DMRD}$ or $\overline{DMWR}$ are low in the time period t_{LOW} prior to the MSB decision point on the fourth rising edge. This causes the internal comparator strobe to be slipped to the fifth rising clock edge. The AD7878 will again check during a period t_{LOW} prior to this fifth rising clock edge; and if the $\overline{CS}$ and $\overline{DMRD}$ or $\overline{DMWR}$ are still low, the bit decision point will be slipped a further clock cycle.

The conversion time for the ADC normally consists of the 13 bit trials described above and one extra internal clock cycle during which data is written from the SAR to the FIFO. For an 8MHz input clock this results in a conversion time of 7µs. However, the software routine which services the AD7878 has the potential to read 16 times from the device during conversion – 8 reads from the FIFO and 8 reads from the status/control register. It also has the potential to write once to the status/control register. If these 17 (16 read plus 1 write) operations all occur during t_{LOW} time periods, it will cause the conversion time to slip by 17 CLK IN cycles. Therefore, if read or write operations can occur during t_{LOW} periods, it means that the conversion time for the ADC can vary from 7µs to 9.12µs (assuming 8MHz CLK IN). This calculation assumes that there is a slippage of one CLK IN cycle for each read or write operation.

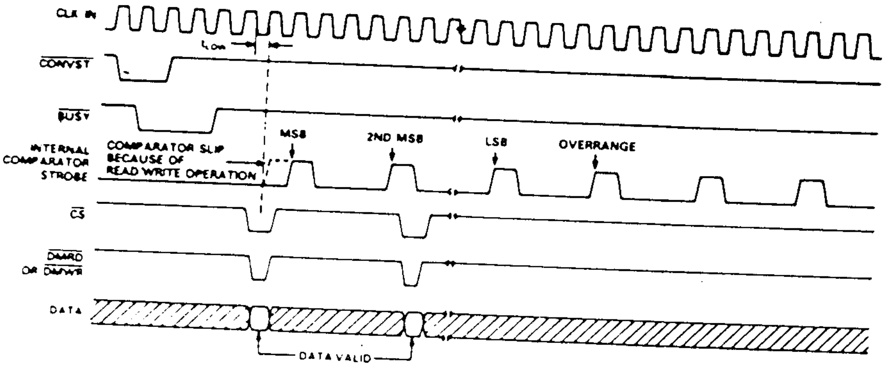


Figure 4. Operational Timing Diagram

INITIATING A CONVERSION

Conversion is initiated on the AD7878 by asserting the $\overline{\text{CONVST}}$ input. This $\overline{\text{CONVST}}$ input is an asynchronous input which is independent of either the ADC or DSP clocks. This is essential for applications where precise sampling in time is important. In these applications the signal sampling must occur at exactly equal intervals to minimize errors due to sampling uncertainty or jitter. In these cases the $\overline{\text{CONVST}}$ input is driven from a timer or some precise clock source. On receipt of a $\overline{\text{CONVST}}$ pulse, the AD7878 acknowledges by taking the $\overline{\text{BUSY}}$ output low. This $\overline{\text{BUSY}}$ output can be used to ensure no bus activity while the track/hold goes from track to hold mode (see Extended Read/Write section). The $\overline{\text{CONVST}}$ input must stay low for at least two CLK IN periods. The track/hold amplifier switches from the track to hold mode on the rising edge of $\overline{\text{CONVST}}$ and conversion is also initiated at this point. The $\overline{\text{BUSY}}$ output returns high after the $\overline{\text{CONVST}}$ input goes high and the ADC begins its successive approximation routine. Once conversion has been initiated another conversion start should not be attempted until the full conversion cycle has been completed. Figure 5 shows the timing diagram for the conversion start.

In applications where precise sampling is not critical, the $\overline{\text{CONVST}}$ pulse can be generated from a microprocessor $\overline{\text{WR}}$ or $\overline{\text{RD}}$ line gated with a decoded address (different to the AD7878 $\overline{\text{CS}}$ address). Note that the $\overline{\text{CONVST}}$ pulse width must be a minimum of two AD7878 CLK IN cycles.

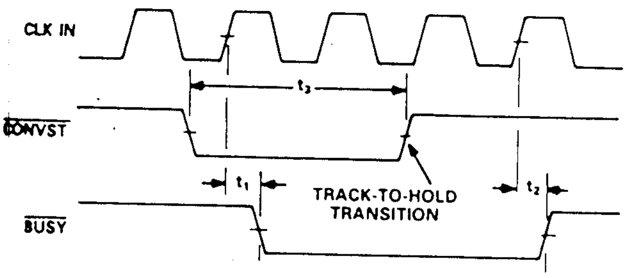


Figure 5. Conversion Start Timing Diagram

READ/WRITE OPERATIONS

The AD7878 read/write operations consist of reading from the FIFO memory and reading and writing from the status/control register. These operations are controlled by the $\overline{\text{CS}}$, $\overline{\text{DMRD}}$, $\overline{\text{DMWR}}$ and ADD0 logic inputs. A description of these operations is given in the following sections. In addition to the basic read/write operations there is an extended read/write operation. This can occur if a read/write operation occurs during a $\overline{\text{CONVST}}$ pulse. The extended read/write is intended for use with microprocessors which can be driven into a WAIT state and the scheme is recommended for applications where an external timer controls the $\overline{\text{CONVST}}$ input asynchronously to the microprocessor read/write operations.

Basic Read Operation

Figure 6 shows the timing diagram for a basic read operation on the AD7878. $\overline{\text{CS}}$ and $\overline{\text{DMRD}}$ going low accesses data from either the status/control register or the FIFO memory. A read operation with ADD0 low accesses data from the FIFO while a read with ADD0 high accesses data from the status/control register.

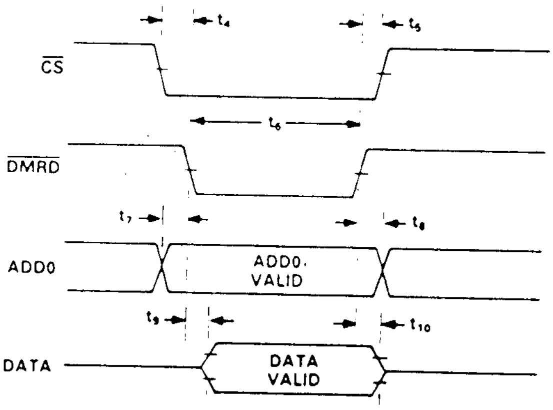


Figure 6. Basic Read Operation

Basic Write Operation

A basic write operation to the AD7878 status control register consists of bringing $\overline{\text{CS}}$ and $\overline{\text{DMWR}}$ low with ADD0 high. Internally these signals are gated with CLK IN to provide an internal REGISTER ENABLE signal (see Figure 7). The pulse width of this REGISTER ENABLE signal is effectively the overlap between the CLK IN low time and the $\overline{\text{DMWR}}$ pulse. This may result in shorter write pulse widths, data setup times and data hold times than those given by the microprocessor. The timing on the AD7878 timing diagram of Figure 8 is therefore given with respect to the internal REGISTER ENABLE signal rather than the $\overline{\text{DMWR}}$ signal.

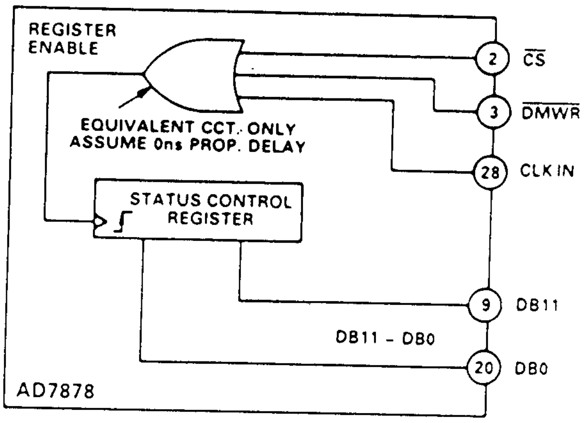
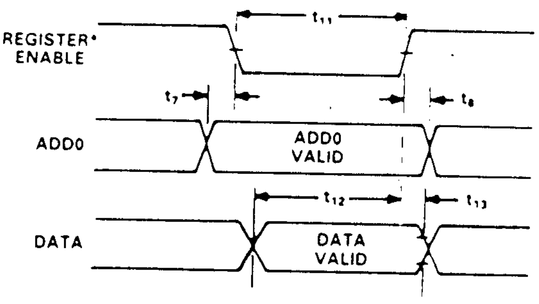


Figure 7. $\overline{\text{DMWR}}$ Internal Logic



*REGISTER ENABLE = $\overline{\text{CS}}$ + $\overline{\text{DMWR}}$ + CLK IN

Figure 8. Basic Write Operation

Extended Read/Write Operation

As described earlier, a read/write operation to the AD7878 can cause spurious on-chip transients. Should these transients occur while the track/hold is going from track to hold mode it may result in an incorrect value of V_{IN} being held by the track/hold amplifier. Because the $\overline{\text{CONVST}}$ input has asynchronous capability, a read/write operation could occur while $\overline{\text{CONVST}}$ is low. The AD7878 allows the read/write operation to occur but has the facility to disable its three-state drivers so that there is no data bus activity and hence no transients while the track/hold goes from track to hold.

Writing a logic 0 to DB5 ($\overline{\text{DISO}}$) of the status/control register prevents the output latches from being enabled while the AD7878 $\overline{\text{BUSY}}$ signal is low. If a microprocessor read/write operation can occur during the $\overline{\text{BUSY}}$ low time, the $\overline{\text{BUSY}}$ should be gated with $\overline{\text{CS}}$ of the AD7878 and this gated signal used to stretch the instruction cycle using DMACK (ADSP-2100), READY (TMS32020) or $\overline{\text{DTACK}}$ (68000).

When $\overline{\text{CONVST}}$ goes low the AD7878 acknowledges by bringing $\overline{\text{BUSY}}$ low on the next rising edge of CLK IN. With a logic 0 in DB5, the AD7878 data bus cannot now be enabled. If a read/write operation now occurs, the $\overline{\text{BUSY}}$ and $\overline{\text{CS}}$ gated signal drives the microprocessor into a WAIT state, thereby extending the read/write operation. $\overline{\text{BUSY}}$ goes high on the second rising edge of CLK IN after $\overline{\text{CONVST}}$ goes high. The AD7878 data outputs are now enabled and the microprocessor is released from its WAIT state, allowing it to complete its read/write operation to the AD7878.

The microprocessor cycle time for the read/write operation is extended by the $\overline{\text{CONVST}}$ pulse width plus two CLK IN periods worst case. This is the maximum length of time for which $\overline{\text{BUSY}}$ can be low. Assuming a $\overline{\text{CONVST}}$ pulse width of two CLK IN periods and an 8MHz CLK IN, the instruction cycle is extended by 500ns maximum. Figure 9 shows the timing diagram for an extended read operation. In a similar manner, a write operation will be extended if it occurs during a $\overline{\text{CONVST}}$ pulse.

For processors which cannot be forced into a WAIT state, writing a logic 1 into DB5 of the status/control register allows the output latches to be enabled while $\overline{\text{BUSY}}$ is low. In this case $\overline{\text{BUSY}}$ still goes low as before, but it would not be used to stretch the read/write cycle and the instruction cycle continues as normal (see Figures 6 and 8).

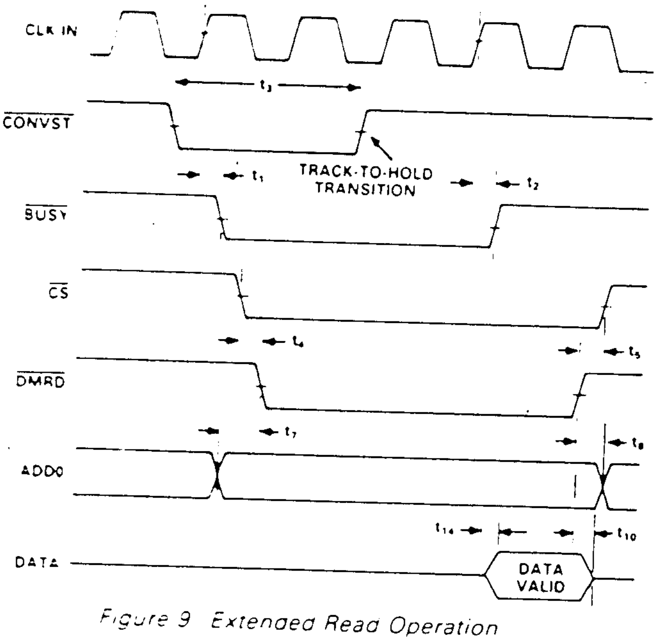


Figure 9 Extended Read Operation

AD7878 DYNAMIC SPECIFICATIONS

The AD7878 is specified and 100% tested for dynamic performance specifications rather than traditional dc specifications such as Integral and Differential Nonlinearity. These ac specifications provide information on the AD7878's effect on the spectral content of the input signal. Hence the parameters for which the AD7878 is specified include SNR, Harmonic Distortion, Intermodulation Distortion and Peak Harmonics. These terms are discussed in more detail in the following sections.

Signal-to-Noise Ratio (SNR)

SNR is the measured signal-to-noise ratio at the output of the ADC. The signal is the rms magnitude of the fundamental. Noise is the rms sum of all the nonfundamental signals (excluding dc) up to half the sampling frequency (fs/2). SNR is dependent upon the number of quantization levels used in the digitization process; the more levels, the smaller the quantization noise. The theoretical signal-to-noise ratio for a sine wave input is given by

$$\text{SNR} = (6.02\text{N} + 1.76)\text{dB} \; . \; . \; . \; . \; . \; (1)$$

where N is the number of bits. Thus for an ideal 12-bit converter, SNR = 74dB.

The output spectrum from the ADC is evaluated by applying a sine-wave signal of very low distortion to the V_{IN} input which is sampled at a 100kHz sampling rate. A Fast Fourier Transform (FFT) plot is generated from which the SNR data can be obtained. Figure 10 shows a typical 2048 point FFT plot of the AD7878KN with an input signal of 25kHz and a sampling frequency of 100kHz. The SNR obtained from this graph is 72.6dB. It should be noted that the harmonics are included in the SNR calculation.

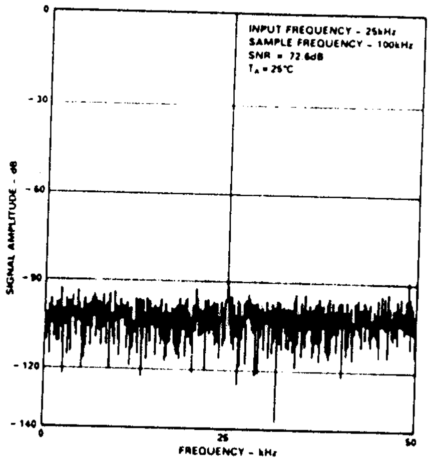


Figure 10. AD7878 FFT Plot

Effective Number of Bits

The formula given in (1) relates the SNR to the number of bits. Rewriting the formula, as in (2), it is possible to get a measure of performance expressed in effective number of bits (N). The effective number of bits for a device can be calculated directly from its measured SNR.

$$\text{N} = \frac{\text{SNR} - 1.76}{6.02} \; . \; . \; . \; (2)$$

Figure 11 shows a typical plot of effective number of bits versus frequency for an AD7878KN with a sampling frequency of 100kHz. The effective number of bits typically falls between 11.7 and 11.85 corresponding to SNR figures of 72.2 and 73.1dB.

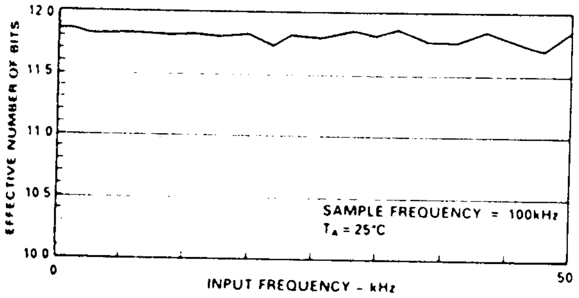


Figure 11. Effective Number of Bits vs. Frequency

Harmonic Distortion

Harmonic Distortion is the ratio of the rms sum of harmonics to the fundamental. For the AD7878, Total Harmonic Distortion (THD) is defined as:

$$\text{THD} = 20 \text{ Log } \frac{\sqrt{(V_2^2 + V_3^2 + V_4^2 + V_5^2 + V_6^2)}}{V_1}$$

where V_1 is the rms amplitude of the fundamental and V_2, V_3, V_4, V_5 and V_6 are the rms amplitudes of the second to the sixth harmonic. The THD is also derived from the FFT plot of the ADC output spectrum.

Intermodulation Distortion

With inputs consisting of sine waves at two frequencies, fa and fb, any active device with nonlinearities will create distortion products at sum and difference frequencies of mfa ± nfb where m,n = 0,1,2,3 . . . , etc. Intermodulation terms are those for which neither m or n are equal to zero. For example, the second order terms include (fa + fb) and (fa − fb) while the third order terms include (2fa + fb), (2fa − fb), (fa + 2fb) and (fa − 2fb).

Using the CCIF standard, where two input frequencies near the top end of the input bandwidth are used, the second and third order terms are of different significance. The second order terms are usually distanced in frequency from the original sine waves, while the third order terms are usually at a frequency close to the input frequencies. As a result, the second and third order terms are specified separately. The calculation of the intermodulation distortion is as per the THD specification where it is the ratio of the rms sum of the individual distortion products to the rms amplitude of the fundamental expressed in dBs.

Intermodulation distortion is calculated using an FFT algorithm but in this case the input consists of two equal amplitude, low distortion sine waves. Figure 12 shows a typical IMD plot for the AD7878.

Peak Harmonic or Spurious Noise

Peak harmonic or spurious noise is defined as the ratio of the rms value of the next largest component in the ADC output spectrum (up to fs/2 and excluding dc) to the rms value of the fundamental. Normally, the value of this specification will be determined by the largest harmonic in the spectrum, but for parts where the harmonics are buried in the noise floor the largest peak will be a noise peak.

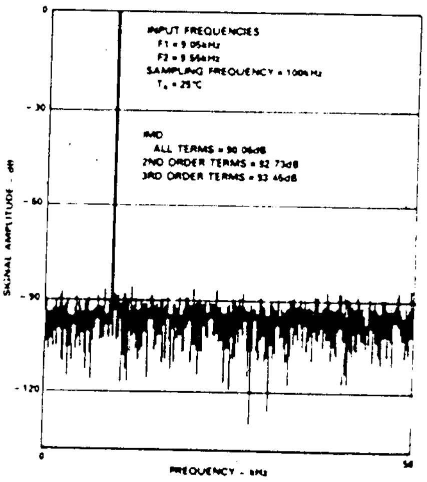


Figure 12. AD7878 IMD Plot

Histogram Plot

When a sine wave of a specified frequency is applied to the V_{IN} input of the AD7878 and several million samples are taken, it is possible to plot a histogram showing the frequency of occurrence of each of the 4096 ADC codes. If a particular step is wider than the ideal 1LSB width, then the code associated with that step will accumulate more counts than for the code for an ideal step. Likewise, a step narrower than ideal will have fewer counts. Missing codes are easily seen in the histogram plot because a missing code means zero counts for a particular code. Large spikes in the plot indicate large differential nonlinearity.

Figure 13 shows a histogram plot for the AD7878KN with a sampling frequency of 100kHz and an input frequency of 25kHz. For a sine-wave input, a perfect ADC would produce a cusp probability density function described by the equation:

$$p\,(V) = \frac{1}{\pi\sqrt{(A^2 - V^2)}}$$

where A is the peak amplitude of the sine wave and p(V) the probability of occurrence at a voltage V. The histogram plot of Figure 13 corresponds very well with this cusp shape. The absence of large spikes in this plot indicates small dynamic differential nonlinearity (the largest spike in the plot represents less than 1/4 LSB of DNL error). The AD7878 has no missing codes under these conditions since no code records zero counts.

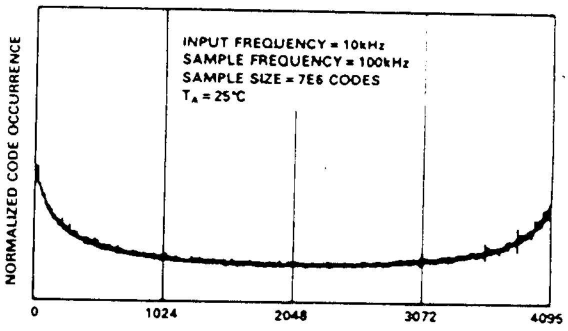


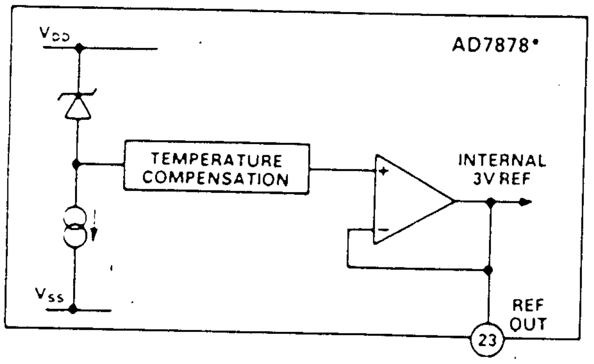
Figure 13. AD7878 Histogram Plot

CONVERSION TIMING

The track-and-hold on the AD7878 goes from track to hold mode on the rising edge of $\overline{CONVST}$ and the value of V_{IN} at this point is the value which will be converted. However, the conversion actually starts on the next rising edge of CLK IN after $\overline{CONVST}$ goes high. If $\overline{CONVST}$ goes high within approximately 30ns prior to a rising edge of CLK IN, that CLK IN edge will not be seen as the first CLK IN edge of the conversion process, and conversion will not actually start until one CLK IN cycle later. As a result, the conversion time (from $\overline{CONVST}$ to FIFO update will vary by one clock cycle depending on the relationship between $\overline{CONVST}$ and CLK IN. A conversion cycle normally consists of 56 CLK IN cycles (assuming no read/write operations) which corresponds to a 7µs conversion time. If $\overline{CONVST}$ goes high within 30ns prior to a rising edge of CLK IN, the conversion time will consist of 57 CLK IN cycles, i.e., 7.125µs. This effect does not cause track hold jitter.

INTERNAL REFERENCE

The AD7878 has an on-chip temperature compensated buried Zener reference (see Figure 14) which is factory trimmed to 3V ± 1%. Internally it provides both the DAC reference and the dc bias required for bipolar operation. The reference output is available (REF OUT) and is capable of providing up to 500µA to an external load.



*ADDITIONAL PINS OMITTED FOR CLARITY

Figure 14. AD7878 Reference Circuit

The maximum recommended capacitance on REF OUT for normal operation is 50pF. If the reference is required for use external to the AD7878 it should be decoupled with a 200Ω resistor in series with a parallel combination of a 10µF tantalum capacitor and a 0.1µF ceramic capacitor. These decoupling components are required to remove voltage spikes caused by the AD7878's internal operation.

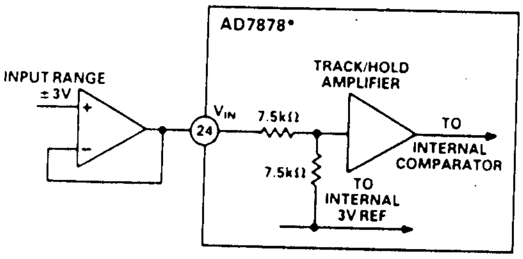
TRACK-AND-HOLD AMPLIFIER

The track-and-hold amplifier on the analog input of the AD7878 allows the ADC to accurately convert an input sine wave of 6V peak-peak amplitude to 12-bit accuracy. The input bandwidth of the track hold amplifier is much greater than the Nyquist rate of the ADC even when operated at its minimum conversion time. The 0.1dB cutoff frequency occurs typically at 500kHz. The track hold amplifier acquires an input signal to 12-bit accuracy in less than 2µs.

The operation of the track hold amplifier is transparent to the user. The track hold amplifier goes from its tracking mode to its hold mode at the start of conversion on the rising edge of $\overline{CONVST}$ and returns to track mode at the end of conversion.

ANALOG INPUT

Figure 15 shows the AD7878 analog input. The analog input range is ±3V into an input resistance of typically 15kΩ. The designed code transitions occur midway between successive integer LSB values (i.e., 1/2LSB, 3/2LSBs, 5/2LSBs . . . . FS – 3/2LSBs). The output code is 2s complement binary wit 1LSB = FS/4096 = 6V/4096 = 1.46mV. The ideal input/out transfer function is shown in Figure 16.



*ADDITIONAL PINS OMITTED FOR CLARITY

Figure 15. AD7878 Analog Input

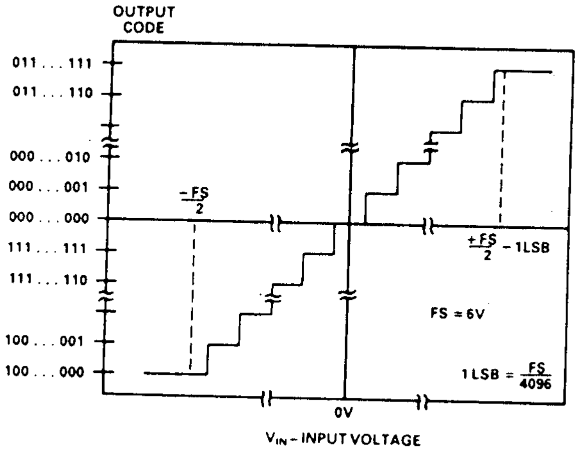


Figure 16. Input/Output Transfer Function

OFFSET AND FULL-SCALE ADJUSTMENT

In most Digital Signal Processing (DSP) applications offset and full-scale error have little or no effect on system performance. Offset error can always be eliminated in the analog domain by ac coupling. Full-scale error effect is linear and does not cause problems as long as the input signal is within the full dynamic range of the ADC. Some applications may require that the inpui signal span the full analog input dynamic range and accordingly offset and full-scale error will have to be adjusted to zero.

Where adjustment is required offset must be adjusted before full-scale error. This is achieved by trimming the offset of the op amp driving the analog input of the AD7878 while the input voltage is 1/2 LSB below ground. The trim procedure is as follows: apply a voltage of – 0.73mV (– 1/2 LSB) at V_I and adjust the op amp offset voltage until the ADC output code flickers between 1111 1111 1111 and 0000 0000 0000.

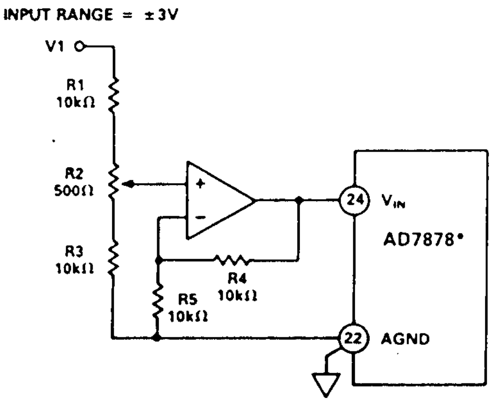
Gain error can be adjusted at either the first code transition (ADC negative full scale) or the last code transition (ADC positive full scale). The trim procedures for both cases are as follows:

Positive Full-Scale Adjust

Apply a voltage of 2.9978V (FS/2 – 3/2LSBs) at V_I. Adjust R2 until the ADC output code flickers between 0111 1111 1110 and 0111 1111 1111.

Negative Full-Scale Adjust

Apply a voltage of $-2.9993V$ ($-FS/2 + 1/2$ LSB) at V_I and adjust R2 until the ADC output code flickers between 1000 0000 0000 and 1000 0000 0001.



*ADDITIONAL PINS OMITTED FOR CLARITY.

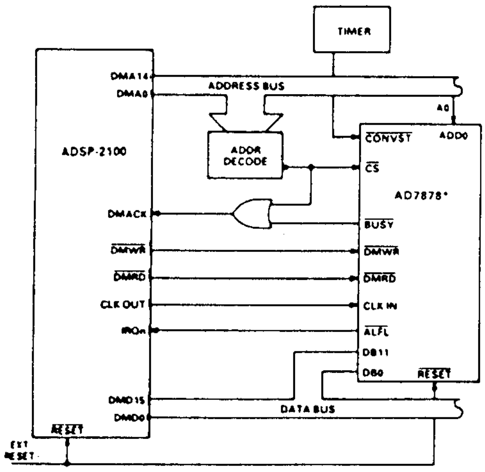
Figure 17. AD7878 Full-Scale Adjust Circuit

MICROPROCESSOR INTERFACING

The AD7878 high speed bus timing allows direct interfacing to DSP processors. Due to the complexity of the AD7878 internal logic, only synchronous interfacing is allowed. This means that the ADC clock must be the same as or a derivative of the processor clock. Suitable processor interfaces are shown in Figures 18 to 21.

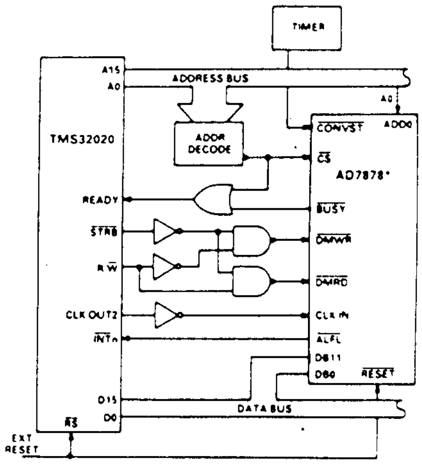
AD7878 – ADSP-2100/TMS32010/TMS32020

All three interfaces use an external timer for conversion control. This allows the ADC to sample the analog input asynchronously to the microprocessor. The AD7878 $\overline{\text{ALFL}}$ output interrupts the processor when the FIFO preprogrammed word count is reached. The processor then reads the conversion results from the AD7878 internal FIFO memory.



*ADDITIONAL PINS OMITTED FOR CLARITY

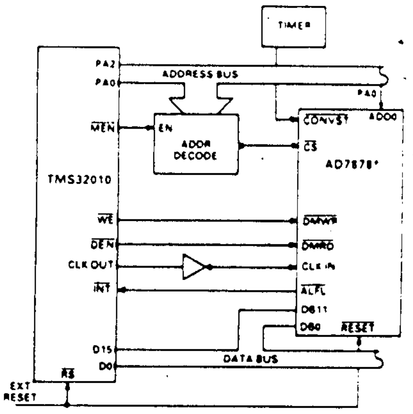
Figure 18. AD7878–ADSP-2100 Interface



*ADDITIONAL PINS OMITTED FOR CLARITY

Figure 19. AD7878 – TMS32020 Interface

The interfaces to the ADSP-2100 and the TMS32020 gate the AD7878 $\overline{\text{CS}}$ and the $\overline{\text{BUSY}}$ to provide a signal which drives the processor into a wait state if a read/write operation to the ADC is attempted while the ADC track/hold amplifier is going from the track to the hold mode. This avoids digital feedthrough to the analog circuitry. The TMS32020 does not have separate $\overline{\text{RD}}$ and $\overline{\text{WR}}$ outputs to drive the AD7878 $\overline{\text{DMWR}}$ and $\overline{\text{DMRD}}$ inputs. These are generated from the processor $\overline{\text{STRB}}$ and R/$\overline{\text{W}}$ outputs with the addition of some logic gates.



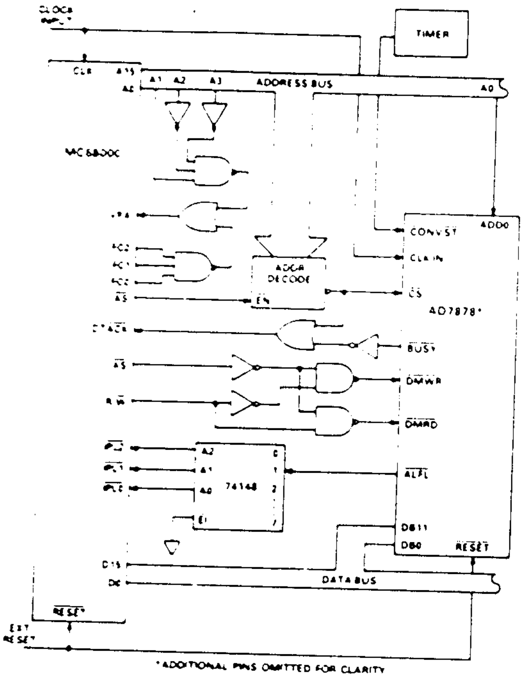
*ADDITIONAL PINS OMITTED FOR CLARITY

Figure 20. AD7878 – TMS32010 Interface

AD7878 – MC68000

This interface also uses an external timer for conversion control as described for the previous three interfaces. It is discussed separately because it needs extra logic due to the nature of its interrupts. The MC68000 has eight levels of external interrupt. When interrupting this processor one of these levels (0 to 7) has to be encoded onto the $\overline{\text{IPL2}}$ – $\overline{\text{IPL0}}$ inputs. This is achieved with a 74148 encoder in Figure 21, (interrupt Level 1 is taken for example purposes only). The MC68000 places this interrupt level on address bits A3 to A1 at the start of the interrupt service routine. Additional logic is used to decode this interrupt level on the address bus and the FC2 – FC0 outputs to generate a $\overline{\text{VPA}}$ signal for the MC68000. This results in an autovectored interrupt, the start address for the service routine must be loaded into the appropriate auto vector location during initialization. For further information on the 68000 interrupts consult the 68000 users manual.

The MC68000 $\overline{\text{AS}}$ and R/$\overline{\text{W}}$ outputs are used to generate separate $\overline{\text{DMWR}}$ and $\overline{\text{DMRD}}$ inputs for the AD7878. As with the previous three interfaces described earlier, WAIT states are inserted if a read/write operation is attempted while the track/hold amplifier is going from the track to the hold mode.



*ADDITIONAL PINS OMITTED FOR CLARITY

Figure 21. AD7878 – MC68000 Interface

Typical AD7878 Microprocessor Operating Sequence

After power up or reset the status control register is initialized by writing to the AD7878. This enables the $\overline{\text{ALFL}}$ output if required for a microprocessor interrupt and sets the effective word length of the FIFO memory. The processor now executes the main body of the program while waiting for an ADC interrupt. This interrupt will occur when the preprogrammed number of samples are collected in the FIFO memory. The interrupt service routine first interrogates DB5(FOOR) of the status/control register to determine if any sample in the FIFO memory is out of range. If all data samples are valid then the program proceeds to read the FIFO memory. If, on the other hand, at least one sample is out of range then an overrange routine is called.

There are many actions which can be taken by the out of range routine, the selection of which is application dependent. One option is to ignore all the current samples residing in the FIFO memory, reinitialize the status control register and return to the main body of the program. Another option is to check the individual out of range status of each word in the FIFO memory and discard the invalid ones. The underrange or overrange status of each word can also be determined and the analog input adjusted accordingly before returning to the main program.

Note there is no need to check the out of range status if the analog input is always assured to be within range.

THROUGHPUT RATE

The AD7878 has a maximum specified throughput rate (sample rate) of 100kHz. This is a worst case test condition and specifications apply for reduced sampling rates provided the Nyquist criterion is obeyed. The throughput rate must take into account ADC $\overline{\text{CONVST}}$ pulse width, ADC conversion time and the track/hold amplifier acquisition time. The time required for each of these tasks is shown in Table II for a selection of DSP processors. Since the ADC clock has to be synchronized to the microprocessor clock, the conversion time depends on the microprocessor used. In addition, time must be allowed for reading data from the AD7878. If this task is performed during the track/hold amplifier acquisition period then it does not impact on the overall throughput rate. However, if the read operations occur during a conversion, then they may stretch the conversion time and reduce the track/hold amplifier acquisition time. The track/hold amplifier requires a minimum of 2µs to operate to specification. The time required to read from the AD7878 depends on the number of FIFO memory locations to be read and the software organization.

As an example, consider an application using the ADSP-2100 and the AD7878 with a throughput rate of 100kHz. The time required for the $\overline{\text{CONVST}}$ pulse and the ADC conversion is 7.375µs. This leaves 2.625µs for the track/hold acquisition time and for reading the ADC (both operations occurring in parallel). The ADSP-2100, when operating from a 32MHz clock, has an instruction cycle of 125ns and an interrupt response time of 500ns. This allows adequate time to perform 16 read operations within the time budget allowed.

	$\overline{\text{CONVST}}$ Pulse Width	Conversion Time	T/H Acquisition Time
Number of Clock Cycles	2 min	57 max	Non-Applicable
ADSP-2100[1]	250ns min	7.125µs max	2µs min
TMS32010[2]	400ns min	11.14µs max	2µs min
TMS32020[2]	400ns min	11.14µs max	2µs min

NOTES
[1] ADSP-2100 Clock Freq. = 32MHz
[2] TMS320XX Clock Freq. = 20MHz

Table II. AD7878 Throughput Rate

APPLICATION HINTS

Good printed circuit board (PCB) layout is as important as the overall circuit design itself in achieving high speed A/D performance. The AD7878 is required to make bit decisions on an LSB size of 1.465mV. To achieve this, the designer has to be conscious of noise both in the ADC itself and in the preceding analog circuitry. Switching mode power supplies are not recommended as the switching spikes will feed through to the comparator causing noisy code transitions. Other causes of concern are ground loops and digital feedthrough from microprocessors. These are factors which influence any ADC, and a proper PCB layout which minimizes these effects is essential for best performance.

LAYOUT HINTS

Ensure that the layout for the printed circuit board has the digital and analog signal lines separated as much as possible. Take care not to run any digital track alongside an analog signal track. Guard (screen) the analog input with AGND.

Establish a single point analog ground (star ground) separate from the logic system ground at Pin 22 (AGND) or as close as possible to the AD7878 as shown in Figure 22. Connect all other grounds and Pin 7 (AD7878 DGND) to this single analog ground point. Do not connect any other digital grounds to this analog ground point. Low impedance analog and digital power supply common returns are essential to low noise operation of the ADC, so make the foil width for these tracks as wide as possible. The use of ground planes minimizes impedance paths and also guards the analog circuitry from digital noise. The circuit layout of Figures 25 and 26 have both analog and digital ground planes which are kept separated and only joined together at the AD7878 AGND pin.

NOISE

Keep the input signal leads to V_{IN} and signal return leads from AGND (Pin 22) as short as possible to minimize input noise coupling. In applications where this is not possible use a shielded cable between the source and the ADC. Reduce the ground circuit impedance as much as possible since any potential difference in grounds between the signal source and the ADC appears as an error voltage in series with the input signal.

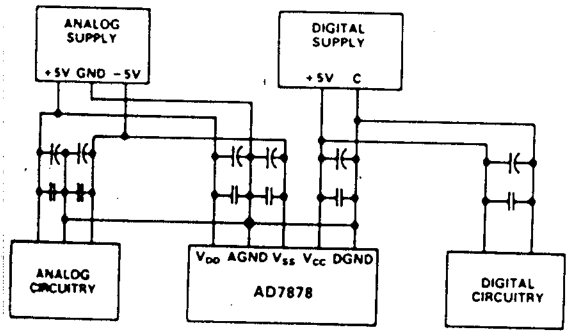


Figure 22. Power Supply Grounding Practice

DATA ACQUISITION BOARD

Figure 23 shows the AD7878 in a data acquisition circuit which will interface directly to either the ADSP-2100, TMS32010 or the TMS32020. The corresponding printed circuit board (PCB) layout and silkscreen are shown in Figures 24 to 26.

The only additional component required for a full data acquisition system is an antialiasing filter. There is a component grid provided near the analog input on the PCB which may be used for such a filter or any other conditioning circuitry. To facilitate this option, a wire link (labelled LK1 on the PCB) is required on the analog input track. This link connects the input signal to either the component grid or directly to the buffer amplifier driving the AD7878 analog input.

Microprocessor connections to the PCB can be made by either of two ways:

96-contact (3 ROW) Eurocard connector.
26-contact (2 ROW) IDC connector.

The 96-contact Eurocard connector is directly compatible with the ADSP-2100 Evaluation Board Prototype Expansion Connector. The expansion connector on the ADSP-2100 has eight decoded enable outputs labelled $\overline{ECE8}$ to $\overline{ECE1}$. $\overline{ECE6}$ is used to drive the AD7878 $\overline{CS}$ input on the data acquisition board. To avoid selecting onboard RAM sockets at the same time, LK6 on the ADSP-2100 board must be removed. In addition, the expansion connector on the ADSP-2100 has four interrupts labelled $\overline{EIRQ3}$ to $\overline{EIRQ0}$. The AD7878 $\overline{ALFL}$ output connects to $\overline{EIRQ0}$. The AD7878 and ADSP-2100 data lines are aligned for left justified data transfer.

The 26-way IDC connector contains all the necessary contacts for both the TMS32010 and TMS32020. There are two switches on the data acquisition board that must be set to enable the appropriate interface configuration (see Table III). The interface connections for the TMS32010/32020 and IDC signal contact numbers are shown in Table IV and Figure 23. Note the AD7878 $\overline{CS}$ input must be decoded from the address bus prior to the AD7878 evaluation board for the TMS320XX interfaces.

Connections to the analog input (V_{IN}) and the $\overline{CONVST}$ input are via two BNC sockets labelled SKT1 and SKT2 on the silk-screen. If the $\overline{CONVST}$ input is derived from either the microprocessor or ADC clock, the effects of clock noise coupling will be reduced.

SWITCH SETTING

Microprocessor	SW1	SW2
ADSP-2100	A	A
TMS32010	B	A
TMS32020	B	B

Table III. AD7878 PCB Switch Settings

POWER SUPPLY CONNECTIONS

The PCB requires two analog supplies and one 5V digital supply. Connections to the analog supplies are made directly to the PCB as shown on the silk screen in Figure 24. The connections are labelled V+ and V− and the range for both of these supplies is 12V to 15V. Connection to the 5V digital supply is made through either of the two microprocessor connectors. The +5V and −5V analog power supplies required by the AD7878 are generated from two voltage regulators on the V+ and V− power supply inputs (IC3 and IC4 in Figure 23).

COMPONENT LIST

IC1	AD711 Op Amp
IC2	AD7878 Analog-to-Digital Converter
IC3	MC78L05 5V Regulator
IC4	MC79L05 −5V Regulator
IC5*	74HC00 Quad NAND Gate
IC6*	74HC04 Hex Inverter
IC7	74HC02 Quad NOR Gate
SW1	Single Pole Double Throw
SW2	Double Pole Double Throw
LK1	Wire Link for Analog Input
C1, C3, C5, C7, C9 C11, C13, C15	10μF Capacitors
C2, C4, C6, C8, C10 C12, C14, C16	0.1μF Capacitors
R1*, R2*	10kΩ Resistors
SKT1, SKT2	BNC Sockets
SKT3	26-Contact (2 Row) IDC Connector
SKT4	96-Contact (3 Row) Eurocard Connector

*Not required for ADSP-2100 Interface