SOME CONSIDERATIONS OF DIGITAL FILTER IMPLEMENTATION USING MICROPROCESSORS

by

JAY PURUSHOTHAMAN JAYAPALAN

B.Sc., University of Madras, 1971
B.E., Indian Institute of Science, 1974

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Electrical Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1976

Approved by:

N. Ahmed

Major Professor

To the memory of

Dr. Dale E. Kaufman

who developed my interest in microprocessors

and initiated this line of research.

THIS BOOK CONTAINS NUMEROUS PAGES WITH DIAGRAMS THAT ARE CROOKED COMPARED TO THE REST OF THE INFORMATION ON THE PAGE.

THIS IS AS RECEIVED FROM CUSTOMER.

ACKNOWLEDGMENT

I wish to thank Dr. N. Ahmed for his valuable guidance and suggestions at various stages of the research reported here and during the final preparation of this report. Thanks are also due to Dr. W. W. Koepsel and Dr. D. H. Lenhert for their support and assistance.

I wish also to thank Mrs. Patricia Stewart for typing this report.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Digital filters have been considered for various applications which include process control, guidance control, and data acquisition. Such filters are usually implemented via hardwired logic and involve the use of shift-registers, adders, multipliers and logic gates [1]. However, the hardware implementation of digital filters is yet a developing field. To this end, the main objective of the work explained in this report is to examine some aspects of implementing digital filters using microprocessors. There are several reasons for considering microprocessors for such implementations:

i) microprocessors are becoming increasingly powerful and yet their cost is reducing

ii) the number of components required is reduced

iii) there is a greater computational capability and flexibility.

A major disadvantage at present is that they are relatively slow. However, since the related technology is moving at a rapid pace, it is reasonable to expect that this disadvantage will be overcome in the relatively near future.

The effects of limited word length such as round off error and coefficient inaccuracy will still be present. However, implementation using microprocessors makes it possible to choose almost any desired word length rather than designing the filter to suit the commercially available hardware multipliers and adders with limited word lengths. For example, several 2 bit slices of Intel 3000 type microprocessors can be used to form a system

of almost any desired word length. Even among fixed word length processors there is a wide choice of word lengths to choose from.

The approach for implementing digital filters via microprocessors is introduced by considering a simple first order recursive filter represented by the transfer function,

$$H(Z) = \frac{Z}{Z + K} \qquad (1.1)$$

This filter has been implemented on a micro-computer system using the Intel 8008 microprocessor. Although Intel 8008 is one of the slowest microprocessors, it is considered for this study since hardware and software information pertaining to it are readily available. This is because it was one of the first microprocessors to have been available commercially. This system has been simulated on an IBM 370 system. Studies pertaining to the filter's frequency response characteristics using the simulator have been made. Interfacing units such as A/D converter have been included in the simulation. Some of the results obtained via the simulator have been verified on the actual microcomputer system.

Chapter 2 introduces some fundamentals pertaining to digital filtering. In Chapter 3 various factors related to digital design considerations are summarized. Conventional methods of digital filter implementation are discussed in Chapter 4, while implementation using microprocessors is considered in Chapter 5.

CHAPTER 2

FUNDAMENTALS OF DIGITAL FILTERS

2.1. Classification

Digital filters are generally classified into two categories as follows:

(i) recursive filters

(ii) non-recursive filters

A recursive filter is one which has an impulse response of infinite duration. For example

$$H(Z) = \frac{Z}{Z - 0.5} \qquad (2.1.1)$$

is a first-order recursive filter since its impulse response is given by

$$h(nT) = (0.5)^n \qquad n = 0,1,2,\ldots \qquad (2.1.2)$$

In contrast, a digital filter with an impulse response which is finite in duration is called a non-recursive filter. As an example, we have

$$H(Z) = (1/2)(1 + Z^{-1}), \qquad (2.1.3)$$

which has an impulse response given by

$$h(nT) = (1/2)[\delta(t) + \delta(t - T)] \qquad (2.1.4)$$

Generally, recursive filters are also known as infinite impulse response (IIR) filters and non-recursive filters as finite impulse response (FIR) filters. Non-recursive filters are implemented via the fast fourier transform (FFT), while recursive filters are implemented using delayed feedback which is readily achieved by means of storage registers.

2.2. Transfer Functions

The transfer function H(Z) of a digital filter is defined as the

ratio of the Z-transform of the output, Y(Z) to the Z-transform of the input, X(Z); i.e.,

$$H(Z) = \frac{Y(Z)}{X(Z)} \tag{2.2.1}$$

The most general recursive filter is a sampled-data linear system, whose present output is a linear combination of the past outputs as well as present and past samples of the input. The following difference equation represents the input-output characteristics of such a filter.

$$Y_n = \sum_{k=0}^{N} a_k X_{n-k} - \sum_{k=1}^{N} b_k Y_{n-k} \tag{2.2.2}$$

where $\{Y_n\}$ is the output sequence, $\{X_n\}$ is the input sequence and $\{a_k\}$ and $\{b_k\}$ are the filter coefficients. Therefore the transfer function of this general recursive filter is given by

$$H(Z) = \frac{\sum_{k=0}^{N} a_k Z^{-k}}{1 + \sum_{k=1}^{N} b_k Z^{-k}} \tag{2.2.3}$$

A non-recursive filter has outputs depending only on the present and past inputs. Since such filters do not involve feedback, they are inherently stable. The input-output characteristics of a general non-recursive filter is

$$Y_n = \sum_{k=0}^{N} a_k X_{n-k} \tag{2.2.4}$$

and its transfer function is

$$H(Z) = \sum_{k=0}^{N} a_k Z^{-k} \tag{2.2.5}$$

Implementation of the above types of filters described by Eqs. (3) and (5) involve widely different techniques. Recursive filters can be realized in

three basic forms. If the output is directly calculated using H(Z) as given by Eq. (3), then it is said to be realized in canonical or direct form. Although a canonic form realization requires a minimum number of delay elements for its realization, it suffers from severe accuracy problems due to accumulation of round off errors. These problems can be minimized by realizing higher order filters by either cascade or parallel forms consisting of first- and second-order sections. To this end, the numerator and denominator polynomials of Eq. (3) are expressed as

$$H(Z) = K \prod_{i=1}^{K_1} \hat{H}_{1i}(Z) \prod_{i=1}^{K_2} \hat{H}_{2i}(Z) \qquad (2.2.6)$$

where $\hat{H}_{1i}(Z) = \dfrac{1+\alpha_{1i} Z^{-1}}{1 + \beta_{1i} Z^{-1}}$, is the transfer function of a first-order section. $\hat{H}_{2i}(Z)$ represents a second-order section; i.e.,

$$\hat{H}_{2i}(Z) = \frac{1 + \alpha_{1i} Z^{-1} + \alpha_{2i} Z^{-2}}{1 + \beta_{1i} Z^{-1} + \beta_{2i} Z^{-2}} \qquad (2.2.7)$$

and $K_1 + K_2 = N$

The coefficients α's and β's are real. The corresponding realization is as in Fig. 2.1.

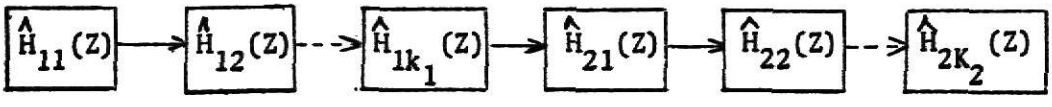


Fig. 2.1. Cascade form realization of recursive filters.

For parallel form realization we take the partial fraction expansion of the Eq. (3)

$$H(Z) = \gamma_0 + \sum_{i=1}^{K_1} H_{1i}(Z) + \sum_{i=1}^{K_2} H_{2i}(Z) \qquad (2.2.8)$$

where $H_{1i}(Z)$ is a first-order section of the form

$$H_{1i}(Z) = \frac{\gamma_{0i}}{1 + \beta_{1i} Z^{-1}} \qquad (2.2.9)$$

and $H_{2i}(Z)$ is a second-order section of the form

$$H_{2i}(Z) = \frac{\gamma_{0i} + \gamma_{1i} Z^{-1}}{1 + \beta_{1i} Z^{-1} + \beta_{2i} Z^{-2}} \qquad (2.2.10)$$

The parallel form realization is shown in Fig. 2.2.

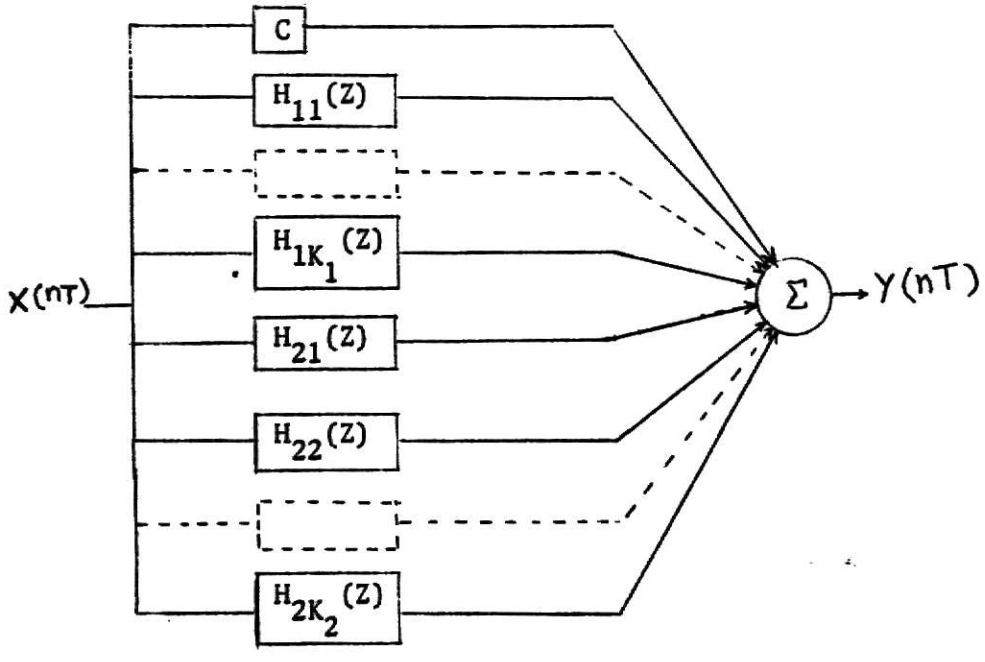


Fig. 2.2. Parallel form realization of recursive filters.

2.3. Number Representation

The two's complement representation of a binary number is most appropriate for digital filters since additions can be performed without any previous knowledge of signs or magnitudes, and with no corrections later as in one's complement notation. Two's complement arithmetic is tolerant to minor scaling errors. At times, even an overflow in the accumulator need not be catastrophic. The output during overflow will be in error but it recovers later on. This is due to an important property of two's complement arithmetic, which is called the cyclic property. For convenience, it will be assumed the signal has magnitude less than unity, which implies that only fractional numbers will be involved.

Now let us study the cyclic property of two's complement arithmetic. The transfer characteristics of a two's complement arithmetic unit is as shown in Fig. 2.3.

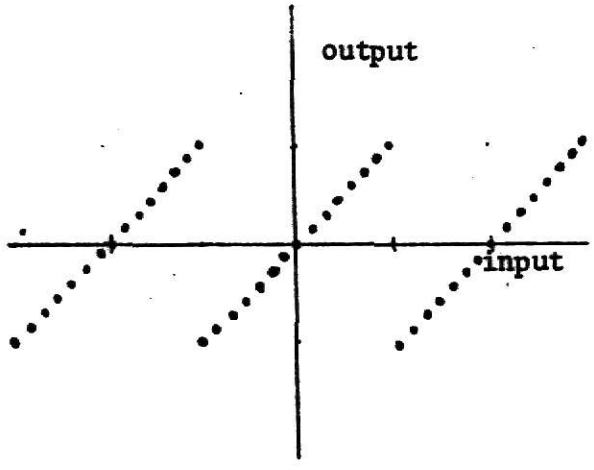


Fig. 2.3. Transfer characteristics of a two's complement binary unit.

This transfer characteristics can be explained by considering a two's complement system with two data bits and one sign bit with respect to a set of numbers on the real line in the interval $-1 \leqslant X < 1$, as shown in Fig. 2.4.

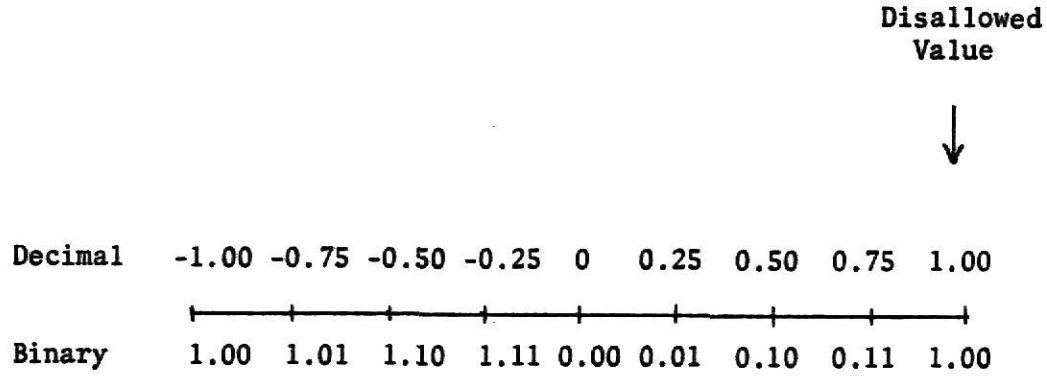


Fig. 2.4. Binary representation on real line.

Incrementing 0.11 we get 1.00. Hence the real line actually folds over in the two's complement binary arithmetic. This in essence is the cyclic property, which is further illustrated in Fig. 2.5.

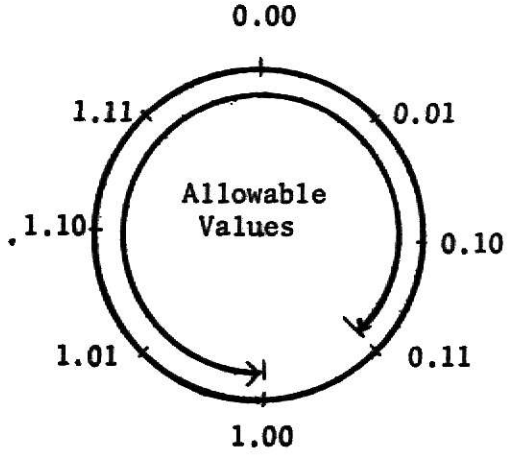


Fig. 2.5. Cyclic property of two's complement binary.

Overflow Considerations. The property that overflow is handled conveniently in a two's complement arithmetic is best illustrated by a simple example.

Consider the addition of 0.75 and 0.5.

$$\begin{array}{r} 0.10 \\ +\ 0.11 \\ \hline 1.01 \end{array} \quad \text{which is } -0.75.$$

This is clearly an error due to overflow. However, suppose the next number to be added is -0.5. Then we have

$$\begin{array}{r} 1.01 \\ +\ 1.10 \\ \hline 0.11 \end{array}$$

which yields 0.75, the result we would expect if 0.75, 0.5, and -0.5 are added. Thus overflow errors "settle down" as we proceed. In most cases it is difficult to represent a decimal fraction exactly in binary form. In such cases, truncation or rounding is necessary since only a finite number of binary digits (bits) are used for the representation. For a specified accuracy we now determine the number of bits required.

Determination of word length. Any decimal fraction consisting of finite digits is represented as

$$X = \sum_{n=1}^{N} d_n \cdot 10^n \qquad (2.3.1)$$

where d_n is the nth digit after the decimal point. Such a number has accuracy limits of

$$X - \Delta X \leq X \leq X + \Delta X \qquad (2.3.2)$$

where $X = 1/2 \cdot 10^{-N}$ (2.3.3)

We seek to approximate X by Y such that

$$Y = X \pm \Delta Y$$

where

$$Y = \sum_{m=1}^{M} b_m 2^{-m} \qquad (2.3.4)$$

and $\Delta Y \leq \Delta X,$ (2.3.5)

so that the accuracy will at least remain the same. Thus, when M bits are used, we obtain

$$Y = 1/2 \cdot 2^{-M} \tag{2.3.6}$$

Using Eqs. (3) and (6) in inequality (5) we obtain

$$1/2 \cdot 2^{-M} \leq 1/2 \cdot 10^{-N} \tag{2.3.7}$$

or $$M \geq N \log_2 10 \tag{2.3.8}$$

i.e., $$M \geq 3.3\ N \tag{2.3.9}$$

Therefore we need 3.3 N bits to represent a decimal fraction of N digits without any loss in accuracy.

To represent a number using a fixed number of bits we have to either truncate or round off the number. Thus, if

$$Y = \sum_{m=1}^{M} b_m 2^{-M}$$

is truncated to K bits, where M>K then

$$Y_T = \sum_{m=1}^{K} b_m 2^{-m} \tag{2.3.10}$$

To round Y to K bits we first calculate

$$Y' = Y + 2^{-(K+1)} = \sum_{m=1}^{M} a_m 2^{-m} \tag{2.3.11}$$

and then truncate to K bits to obtain

$$Y_K = Y'_T = \sum_{m=1}^{K} a_m 2^{-m} \tag{2.3.12}$$

Rounding as expected gives less error than truncation.

2.4. Data Converters

Since most signals are analog in nature, analog to digital (A/D) conversion is an important part of digital systems. Using an A/D converter,

an analog signal is sampled at regular intervals and then approximated to obtain a binary number of finite word length. Due to limited resolution resulting from the finite word length of the binary representation, quantization of input becomes necessary. This quantization can be represented as in Fig. 2.6. The function of the A/D converter is to decide on one of the quantized levels for each sample of input based on some minimum error criterion.

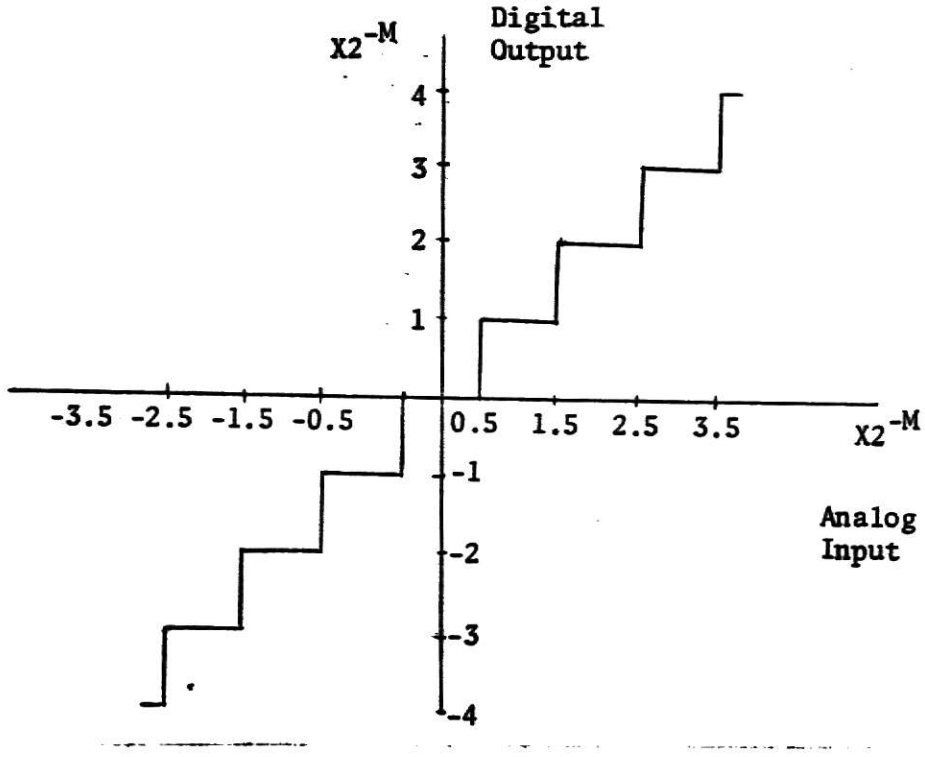


Fig. 2.6. Quantization in A/D converters.

Next, we calculate the error due to quantization. Consider any value of the input, V which lies between $1/2 \cdot 2^{-M}$ and $3/2 \cdot 2^{-M}$. This is approximated to the quantized level 2^{-M}. Assuming all the quantized levels are uniformly separated by $q = 2^{-M}$, we have in general any value that lies between $V_K - q/2$ to $V_K + q/2$ approximated to V_K, the Kth quantized level. The

corresponding mean square error introduced at level K is

$$\overline{(V - V_K)^2} = \int_{V_K - q/2}^{V_K + q/2} (V - V_K)^2 \, p(v)dv \qquad (2.4.1)$$

where p(v) is the probability density of the signal. Assuming that the signal is uniformly distributed in the quantization interval, we have

$$\overline{(V - V_K)^2} = \frac{q^3}{12} p(V_K) \qquad (2.4.2)$$

The quantization noise power is the sum of the mean square errors introduced at each level; hence

$$\overline{Nq^2} = K \, \overline{(V - V_K)^2} \qquad (2.4.3)$$

which yields

$$\overline{Nq^2} = \frac{q^2}{12} \sum_K P_K(V) \qquad (2.4.4)$$

where $P_K(V)$ is the discrete probability of the input appropriate to the Kth step and is given by

$$P_K(V) = \int_{V_K - q/2}^{V_K + q/2} p(V)dv = p(V_K) \cdot q \qquad (2.4.5)$$

Also,

$$\sum_K P_K(V) = 1$$

Thus the final result is

$$\overline{Nq^2} = \frac{q^2}{12} \qquad (2.4.6)$$

This result will be used in Sec. 3.3.

We observe that since $q = 2^{-M}$, Eq. (6) implies that the larger the word length M, the less the noise due to quantization.

Analog-to-Digital conversion techniques are well developed and it is possible to implement A/D conversion in a number of different ways such as successive approximation, single or dual slope conversion, conversion using a voltage controlled oscillator (VCO), parallel conversion, etc. Each method is chosen based on the requirements such as speed, accuracy and cost. For the purpose of discussion, the successive approximation A/D converters are considered. The basic structure of these is as shown in Fig. 2.7.

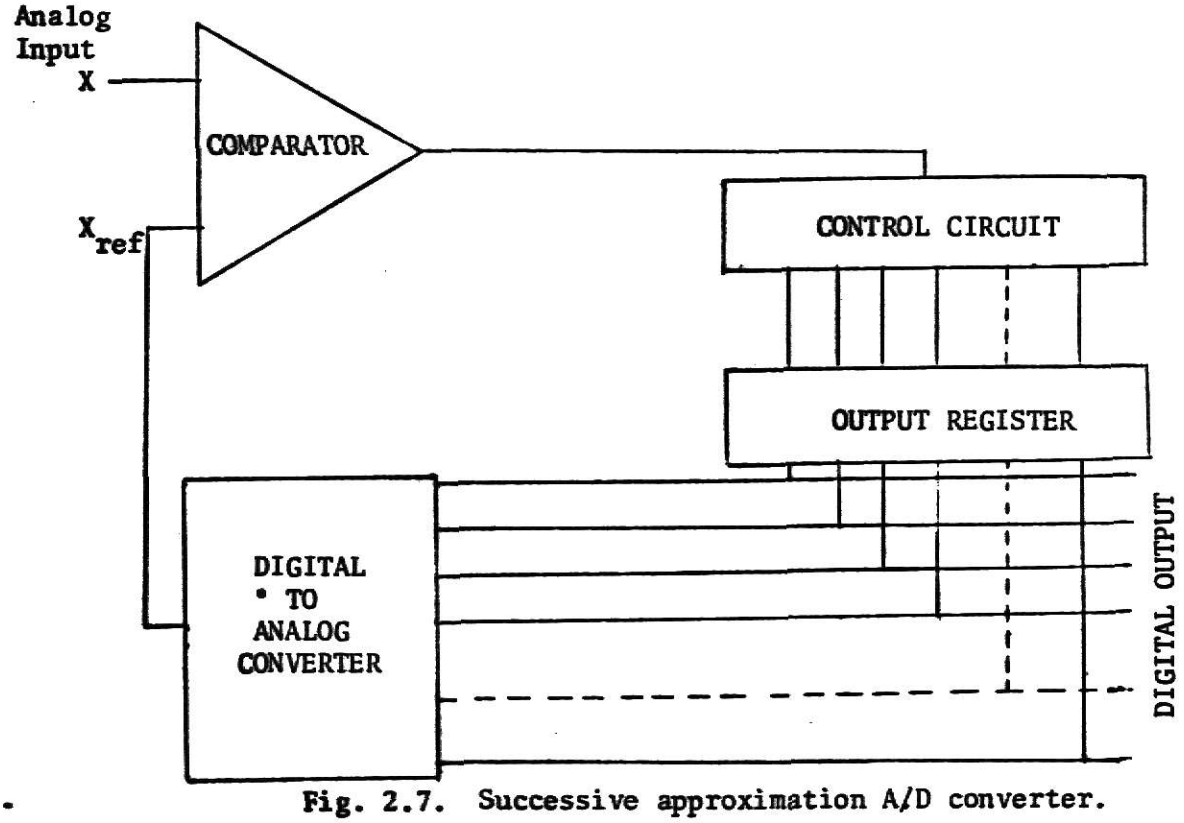


Fig. 2.7. Successive approximation A/D converter.

The successive approximation converter works as follows. At the start of the conversion, the control circuit places a '1' in the most significant bit (MSB) of the previously cleared output register. This corresponds exactly to the mid point in the input range of the converter. The D/A in the feedback converts the 100 . . . 0 to $X_{range}/2$. If X_{in} is greater than $X_{range}/2$ the output of the comparator is '1'. The control circuit reads this output and decides to keep the '1' in MSB if the comparator output is '1'. Otherwise it resets MSB to a '0'. Now the control circuit places a '1' in the next most significant bit and checks the output of the comparator. A decision on whether to leave the '1' in that bit of the output register or to clear it to '0' is taken based on the comparator output. Likewise, the conversion is done on a bit by bit basis (i.e. serial conversion) until the least significant bit (LSB) is reached. When the conversion is complete, the output register has the digital equivalent of the analog input. High precision and low cost are some of the features of this method. Parallel A/D conversion is faster but more expensive. For an M-bit conversion we need 2^{M-1} comparators and hence the high cost.

Digital-to-Analog conversion, on the other hand, is much simpler to implement. Very often resistor networks are employed in D/A converters. Again, there are several methods of realizing D/A conversion and since information on these is easily available we will not discuss this at length.

Having reviewed the basic concepts of digital filtering we are now ready to discuss the related design concepts in the next chapter.

CHAPTER 3

DIGITAL DESIGN CONSIDERATIONS

3.1. Introduction

In this chapter we will discuss some details related to the hardware implementation of digital filters. We shall assume fixed point binary computations in the implementations. Due to finite word lengths, problems such as round off accumulation, quantization and overflow arise. The main objectives of this chapter are as follows: (i) explain how a set of design criteria can be developed to determine the gain scaling factor to avoid overflow, and (ii) explain how one can choose optimum word lengths to reduce quantization and round off errors [1].

3.2. Gain Scaling

Fixed point arithmetic calls for constraints on inputs to the filter such that the computations are within the linear range. These constraints can be achieved via gain scaling. For example, the input to the adders must be scaled such that the outputs from the adders are always within $\pm$ 1. As we discussed in Sec. 2.2, digital filters of higher order can be constructed using basic blocks of first and second order filters. Thus we need to analyze only these two classes of filters. For simplicity and as a basic introduction to the concepts involved in hardware implementation, we will restrict our attention to the discussion of first order filters only.

A generalized first order recursive filter has a transfer function of the form,

$$H(Z) = K\frac{(1 - AZ^{-1})}{(1 - BZ^{-1})} \qquad (3.2.1)$$

The squared-magnitude frequency response, M of the above transfer function is

$$M = \left| H(Z) \right|^2_{Z = e^{jwT}}$$

$$= K^2 \frac{(1 - 2A \cos wT + A^2)}{(1 - 2B \cos wT + B^2)} \qquad (3.2.2)$$

where T is the sampling interval. The maximum and minimum of M occur when

$$\frac{dM}{d(wT)} = 0 \qquad \text{with} \quad \frac{d^2M}{d(wT)^2} < 0,$$

and

$$\frac{dM}{d(wT)} = 0 \qquad \text{with} \quad \frac{d^2M}{d(wT)^2} > 0$$

respectively.

Differentiating Eq. (2)

$$\frac{dM}{d(wT)} = K^2 \left[\frac{2 \sin wT \,(A - B)(1 - AB)}{(1 - 2B \cos wT + B^2)^2}\right] \qquad (3.2.3)$$

When we equate the above derivative to zero , we arrive at the following conditions:

(1) $\sin wT = 0$

which yields $w = n\pi f_s$

$$= \frac{n\, w_s}{2}$$

That is $w = nw_N$ (3.2.4)

W_s is the sampling frequency and $W_N = \frac{W_s}{2}$, is the Nyquist frequency.

(2) $A = B$ or $A = 1/B$ are the other two conditions for $\frac{dM}{d(wT)} = 0$. These are trivial and are not of interest to us.

Setting $wT = n\pi$, the second derivative of M becomes

$$\frac{d^2M}{d(wT)^2} = K^2 \frac{(-1)^n\, 2(A - B)\,(1 - AB)}{[1 + (-1)^{n+1} \cdot B]^4} \qquad (3.2.5)$$

When $(A - B)(1 - AB) < 0$, M_{max} occurs at $n = 0$, i.e., at $w = 0$. When $(A - B)(1 - AB) > 0$, M_{max} occurs at $n = 1$, i.e., at $w = w_N$

Let us assume $|A| \leq 1$ for convenience. Then we require $|B| < 1$ for stability. Two cases which arise (depending on the relative values of A and B) are illustrated in Fig. 3.1.

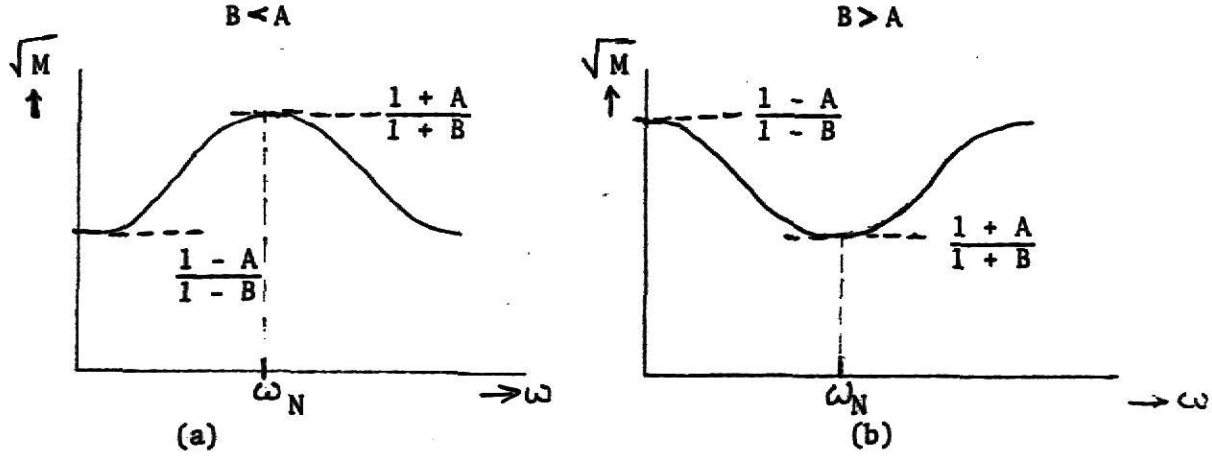


Fig. 3.1. Frequency response curves of first order recursive filter networks.

If the filter is implemented as in Fig. 3.2, then we should make sure there is no overflow in each of the adders. This could be controlled by the scale factor, K, which determines the gain. K must be just low enough that it avoids overflow, ensuring linearity and at the same time large enough that the dynamic range is not affected.

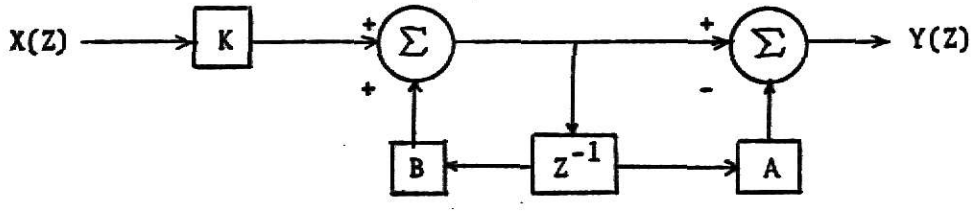


Fig. 3.2. A configuration for implementing a first order filter.

To avoid overflow in the first adder, we assume A = 0, so that we have only the first adder in the configuration. Then we have

$$M_{max} = K \left(\frac{1}{1-B} \right) \qquad B > 0 \qquad (3.2.6)$$

$$M_{max} = K \left(\frac{1}{1+B} \right) \qquad B < 0 \qquad (3.2.7)$$

Our aim is to restrict M_{max} to 1 by choosing K appropriately. This results in

$$K = 1 - B \qquad \text{for } B > 0 \qquad (3.2.8)$$

$$K = 1 + B \qquad \text{for } B < 0 \qquad (3.2.9)$$

However in general when $A \neq 0$ we have

$$M_{max} = K \frac{(1 - A)}{(1 - B)} \qquad (3.2.10)$$

$$\text{i.e., } K = \frac{1 - B}{1 - A} \qquad \text{for } B > A \qquad (3.2.11)$$

and

$$M_{max} = K \frac{(1 + A)}{(1 + B)} \qquad (3.2.12)$$

$$K = \frac{1 + B}{1 + A} \qquad \text{for } B < A \qquad (3.2.13)$$

Therefore, we take the minimum of two values to avoid overflow. In other words,

$$K = \min \left\{ (1 - B), \frac{1 - B}{1 - A} \right\} \qquad \text{for } B > A \qquad (3.2.14)$$

and

$$K = \min \left\{ (1 + B), \frac{1 + B}{1 + A} \right\} \qquad \text{for } B < A \qquad (3.2.15)$$

which is the desired gain scale factor for first order networks.

3.3. Word Lengths

In general, there are three word lengths which are important in the digital design of a filter. They are

(i) The word length, C of the A/D converter. This is the word length of data input to the filter. This determines, as discussed earlier, the quantization noise in the A/D converter.

(ii) The word length, M of data when it is processed through the digital filter. This is generally larger than the input word length, C. The output of the A/D is appended with (M - C) zeros at the end to decrease the effects of repeated truncation and/or rounding during the computation. M is known as the computational word length.

(iii) The word length, N of the filter coefficients. The value of N depends on how precisely the coefficients have to be represented.

Input data word length, C. Two main considerations determine the value of C. These are as follows:

(a) accuracy at the input is one of the two factors to be considered. Let the minimum detectable level of the signal, which is also known as the threshold of the signal, be designated by X_{th}. If the saturation level, which is the maximum analog input to the converter, is represented by X_{sat} then the total number of quantized levels, N, is given by

$$N = \text{smallest integer greater than } (X_{sat}/X_{th}) \tag{3.3.1}$$

The number of bits required to represent N levels is

$$C_1 = \log_2 N^* \text{ bits} \tag{3.3.2}$$

The input word length C is C_1 plus one sign bit. Therefore

$$C = 1 + C_1 = 1 + \log_2 \left(\frac{X_{sat}}{X_{th}}\right) \tag{3.3.3}$$

(b) as explained in Sec. 2.4, the quantization error which when treated as additive white noise has a noise power of

---

*All word lengths are rounded off to the smallest integer greater than the value obtained on the right hand side of the equations.

$$\overline{N_q^2} = \frac{q^2}{12} \tag{3.3.4}$$

where $q = \frac{X_{th}}{X_{sat}} = \frac{1}{N}$, which is the quantum step. From Eqs. (3) and (4), it follows that

$$\overline{N_q^2} = \frac{(2^{-C_1})^2}{12} = 2^{-2(C_1 + 1)} \cdot \frac{1}{3} \tag{3.3.5}$$

The noise figure, F, which is the signal to noise power ratio expressed in db is given by

$$F = 10 \log \frac{\overline{S^2}}{\overline{N_q^2}} \tag{3.3.6}$$

where $\overline{S^2}$ is the signal power. Thus Eq. (5) yields

$$F = 10 \log_{10} [3 \overline{S^2} 2^{2(C_1 + 1)}] \tag{3.3.7}$$

solving for $C = C_1 + 1$ we obtain

$$C = C_1 + 1 = \frac{F + 10 \log 3 \overline{S^2}}{20 \log_{10} 2} \tag{3.3.8}$$

It is reasonable to assume the input signal amplitude has a zero-mean Gaussian distribution with a standard deviation = 1/3. Then we have

$$\overline{S^2} = 1/9 \tag{3.3.9}$$

Substitution for $\overline{S^2}$ in Eq. (8) results in

$$C = \frac{F + 10 \log_{10} 3}{20 \log_{10} 2} \tag{3.3.10}$$

Thus the desired input word length, C, is taken as the maximum of the values dictated by Eqs. (2) and (10); i.e.,

$$C = \text{Max}\left\{(1 + \log_2 \frac{X_{sat}}{X_{th}}), (\frac{F + 10 \log_{10} 3}{20 \log_{10} 2})\right\} \qquad (3.3.11)$$

For example, if the input signal is limited to ± 10 V and the threshold voltage of 100 mV is assumed, then we have

$$C = 1 + \log_2 \frac{10}{0.1}$$

$$= 7.644$$

Rounded off to the nearest integer, C is 8. If the allowed quantization error gives a signal to noise ratio of 50 db, Eq. (11) gives

$$C = \frac{50 + 4.77}{6.02}$$

$$= 9.10$$

Now $C = 10$

Hence $C = \text{Max}\{8, 10\}$

$= 10$ bits

Computational Word Length. To precisely represent the product of two N bit numbers, we need 2N bits. However, in practice, we either truncate this or round it off to N bits. Rounding results in less error compared to truncation. It has been found that rounding is as advantageous as increasing the word length by one bit. The transfer characteristics of both truncation and round off are shown in Fig. 3.3 [1].

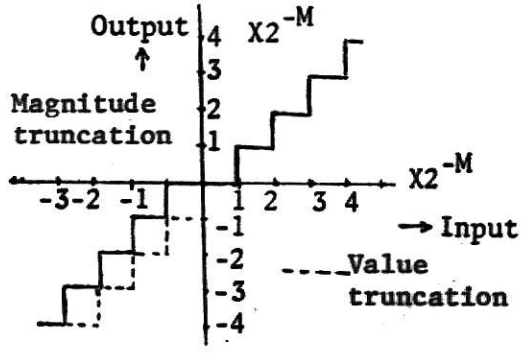


Fig. 3.3 (a). Input-output relation for truncating.

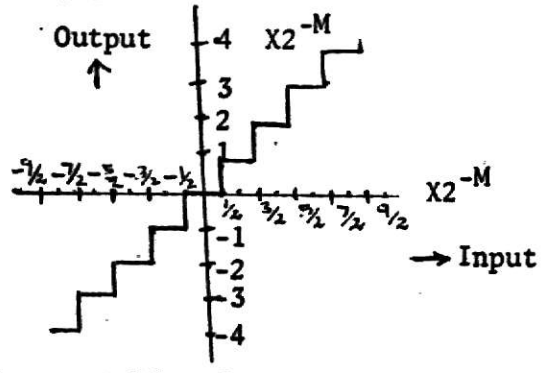


Fig. 3.3 (b). Input-output relation for rounding off.

These figures illustrate that both rounding and truncation are highly non-linear operations. However, the input-output characteristics resemble that of an A/D converter (Fig. 2.6) very closely. Hence these can also be treated as quantization error. It is also found that truncation or round off distortion can be assumed as an additive noise that is uncorrelated with quantization noise as well as with the input signal. Non-linearity introduced by the rounding off might affect the stability of the filter. It is instructive to first study how round off considerations dictate a particular minimum computational word length.

For the simple first-order filter $H(Z) = \frac{1}{1 - BZ^{-1}}$, the round off operation is represented by the quantizer, Q, as shown in Fig. 3.4 (a).

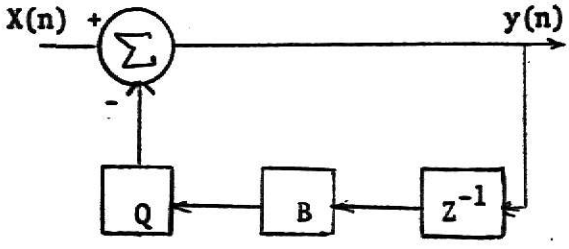


Fig. 3.4 (a)

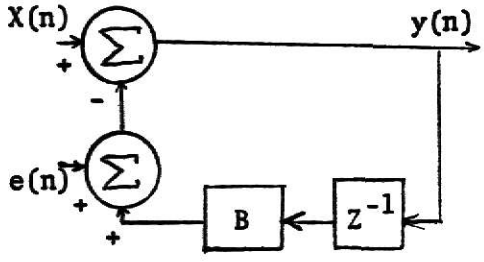


Fig. 3.4 (b)

Noise source equivalent of round off error.

The quantizer can be replaced by an equivalent noise source as shown in Fig. 3.4 (b). The noise input sequence, e(n) is assumed to be independent from sample to sample. We also assume that it is a zero-mean variable with a variance of σ_e^2. The output due to the noise, Y_e (n) is found convolving the input sequence with the impulse response, h (n) of the filter section from the noise source to the output.

$$Y_e(n) = \sum_{m=0}^{n} h(m)\ e(n-m) \qquad (3.3.12)$$

The variance of the round off noise at the output of the filter can be computed as

$$E\ [Y^2(n)] = E\ [\sum_{m=0}^{n} \sum_{k=0}^{n} h(m)\cdot h(K) e(n-m) e(n-K)] \qquad (3.3.13)$$

$$= \sum_{m=0}^{n} \sum_{k=0}^{n} h(m)\cdot h(K)\ E[e(n-m)e(n-K)] \qquad (3.3.14)$$

Since the noise sequence is assumed to be uncorrelated from sample to sample,

$$E\ [e(n-m)e(n-K)] = \sigma_e^2 \qquad m = K$$

$$= 0 \qquad m \neq K \qquad (3.3.15)$$

Then Eq. (14) becomes

$$E[Y^2(n)] = \sum_{m=0}^{n} h^2(m)\ \sigma_e^2 \qquad (3.3.16)$$

The steady state variance of the round off noise at the output, σ_n^2 is obtained from Eq. (16) as follows

$$\sigma_n^2 = \lim_{n \to \infty} E\ [Y^2(n)]$$

$$= \sigma_e^2 \sum_{m=0}^{\infty} h^2(m) \qquad (3.3.17)$$

The quantity $\sum_{m=0}^{\infty} h^2(m)$ is called the noise gain of the filter. It can be computed using Cauchy's residue theorem as follows

$$\text{Noise Gain, NG} = \sum_{m=0}^{\infty} h^2(m)$$

$$= \sum_{\substack{\text{residue inside} \\ \text{the unit circle}}} H(Z)\ H(Z^{-1})Z^{-1} \qquad (3.3.18)$$

If M computational data bits are used in the filter mechanization, the variance of the noise generated by a multiplier round off is

$$\overline{e_m^2} = \frac{(2^{-M})^2}{12}$$

$$= \frac{2^{-2(M+1)}}{3} \qquad (3.3.19)$$

$$\overline{e_o^2} = \frac{2^{-2(M+1)}}{3} \, NG \qquad (3.3.20)$$

If we assumed the same signal as in the case of the A/D converter, then the signal to noise ratio in the case of round off is

$$F = 10 \log \frac{\overline{S^2}}{\overline{e_0^2}} = 10 \log_{10} \left[\frac{2^{2(M+1)}}{3 \cdot NG}\right] \qquad (3.3.21)$$

Solving for M + 1, the required number of bits is given by

$$M + 1 = (0.8 + \frac{F}{6}) + \frac{10}{6} \log_{10} (NG) \qquad (3.3.22)$$

Example. For the first-order section represented by

$$H(Z) = \frac{1}{1 - BZ^{-1}}$$

$$= \frac{Z}{Z - B}$$

The impulse response

$$h(m) = Z^{-1}\left[H(Z)\right]$$

$$= B^m$$

Thus noise gain for the filter is obtained as

$$NG = \sum_{m=0}^{\infty} h^2(m)$$

$$= \sum_{m=0}^{\infty} B^{2m}$$

$$= \frac{1}{1 - B^2} \qquad (3.3.23)$$

If F is specified to be 35 db and B = 0.75, then we have

$$M + 1 = (0.8 + \frac{35}{6}) + \frac{10}{6} \log \frac{1}{1 - (0.75)^2}$$

$$= 7.23$$

Thus M + 1 = 8 meets the requirements.

Another factor that needs to be considered for determining M is the phenomenon of limit cycle oscillations. This phenomenon occurs only in infinite impulse response or recursive networks. This concept is explained by the following example. Consider the first-order filter

$$H(Z) = \frac{Z}{Z - 0.5},$$

whose impulse response is given by

$$h(nT) = (0.5)^n, \quad n = 0,1,2 \ldots \qquad (3.3.24)$$

The difference equation that represents the output of the filter is

$$Y(nT) = X(nT) + 0.5\,[Y(n-1)T] \qquad (3.3.25)$$

Suppose $X(0) = 0.75$

and $X(nT) = 0 \quad \text{for } n \geq 1 \qquad (3.3.26)$

Then $Y(nT) = 0.5\,Y\,[(n-1)T] \quad \text{for } n \geq 1 \qquad (3.3.27)$

i.e., $Y(T) = 0.5\,Y(0)$

$= 0.5 \times 0.75$

$= 0.375$

$$Y(2T) = 0.5\ Y(T)$$

$$= 0.5 \times 0.375$$

$$= 0.1875$$

similarly $Y(3T) = 0.09375$

$$Y(4T) = 0.046875$$

Clearly $\lim_{n \to \infty} Y(nT) = 0$

However, a practical digital filter implemented with input word length 3 and computational word length of 4 yields the following outputs on rounding.

$$K = 0.5 = 0.10_2$$

$$Y(T) = 0.5 \times 0.75$$

$$= 0.10 \times 0.110$$

$$= 0.01100$$

$$= 0.011 \quad \text{(after rounding)}$$

$$Y(2T) = 0.10 \times 0.011$$

$$= 0.00110$$

$$= 0.010$$

$$Y(3T) = 0.10 \times 0.010$$

$$= 0.00100$$

$$= 0.001$$

$$Y(4T) = 0.10 \times 0.001$$

$$= 0.00010$$

$$= 0.001$$

$$Y(5T) = 0.10 \times 0.001$$

$$= 0.00010$$

$$= 0.001$$

That is Y(nT) = 0.001, for all $n \geq 3$. Hence this output never reaches zero as it should. Similarly,

$$\text{for} \qquad H(Z) = \frac{Z}{Z + 0.5}$$

the output corresponding to an input as described by Eq. (2.6) oscillates between 0.001 and 1.111. Hence we have output with no input which is a clear indication of oscillations. The amplitude of the oscillations can be reduced by increasing word length, M.

It has been established [1] that the amplitude of the oscillations for a filter of the above type is given by

$$Y_L \leq \frac{(0.5)\,2^{-M}}{1 - |K|} \tag{3.3.28}$$

Filter Coefficient Word Length, N. The frequency characteristics of a filter is determined by the filter coefficients. The accuracy with which these coefficients are represented in binary form determines the quality of performance of the filter. Each coefficient K is restrained to lie between a minimum, K_{min} and a maximum, K_{max} values. These values depend on the tolerance of the filter performance dictated by the system in which it is used. If N bits are used to represent K we must make sure N is large enough such that

$$2^{-N} \leq K_{max} - K_{min}$$

In other words

$$N \geq \log_2 \frac{1}{K_{max} - K_{min}} \tag{3.3.29}$$

For a given accuracy Eq. (2.3.9) can also be used to determine N.

In this chapter we have seen how the fixed word length in digital filters affect the filter and how we can choose these word lengths

appropriately. Now we can proceed to look at some of the implementations using hard-wired logic before considering the problem of implementation using microprocessors.

CHAPTER 4

IMPLEMENTATION USING HARD-WIRED LOGIC

4.1. Conventional Methods

Once the word lengths, scale factors, etc., have been determined, the mechanization of first- and second-order filters is quite straightforward. When hard-wired logic is used, one can think of several configurations of implementing a filter. A detailed study related to the implementation of a first-order filter as given by Eq. (3.2.1) is considered next.

In Fig. 4.1 commonly used configuration is shown, where K is the gain scale factor. It follows that

$$Y_1(Z) = K\,X(Z) + B\,Y_1(Z)\,Z^{-1} \qquad (4.4.1)$$

$$Y(Z) = Y_1(Z) - AY_1(Z)\,Z^{-1}$$

$$= (1 - AZ^{-1})\,Y_1(Z) \qquad (4.1.2)$$

Substituting for $Y_1(Z)$ using Eq. (1) we obtain

$$Y(Z) = \frac{K(1 - AZ^{-1})}{(1 - BZ^{-1})}\,X(Z) \qquad (4.1.3)$$

Thus

$$H(Z) = \frac{K(1 - AZ^{-1})}{(1 - BZ^{-1})} \qquad (4.1.4)$$

The term Z^{-1} represents a delay of one sampling interval and is implemented as a storage register which stores the outputs and makes it available for processing when the next input appears. Adders are formed by using a chain of full adders with as many full adders as the number of bits. Multiplication can be done in several ways. The most common approach is through

shifting and adding successively in a register. Sometimes a table look up for multiplication is used when the coefficients are fixed. This speeds up the processing. The trade off here is between total storage necessary and computational time.

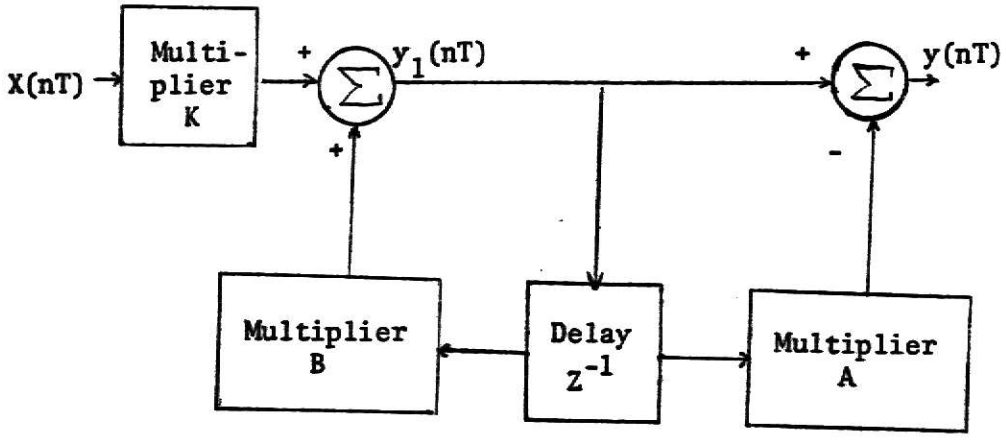


Fig. 4.1. A first-order filter implementation.

From Eq. (3.2.14) we have

$$K = \min\left[(1 - B), \left(\frac{1 - B}{1 - A}\right)\right] \quad \text{for } B > A \qquad (4.1.5)$$

Now, if $0 \leq A < 1$ then

$$K = 1 - B \qquad (4.1.6)$$

Hence the above configuration can be changed as shown in Fig. 4.2. Here a multiplier has been traded off for an adder.

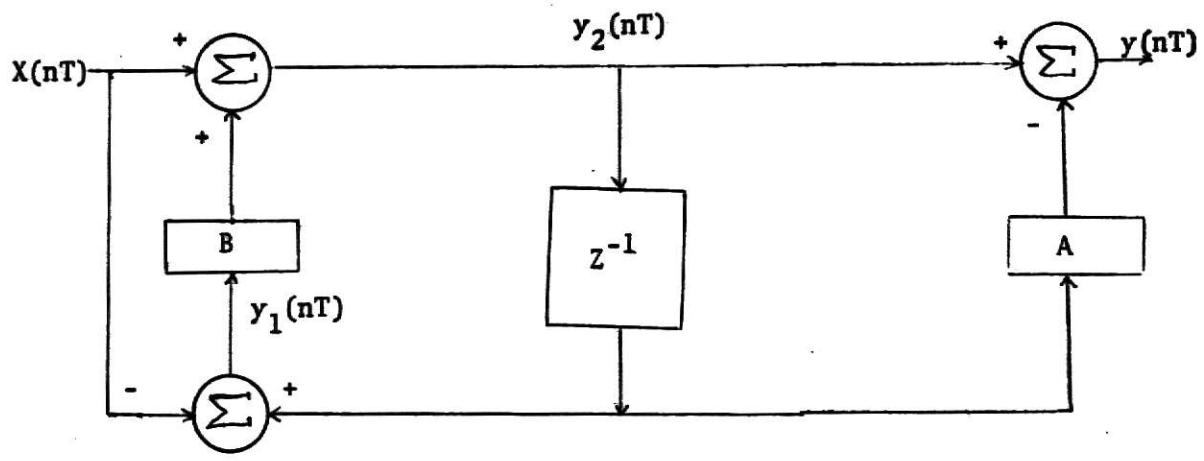


Fig. 4.2. An alternate implementation of the first-order filter.

From Fig. 4.2 it follows that

$$Y_1(Z) = Y_2(Z)\ Z^{-1} - X(Z) \tag{4.1.7}$$

$$Y_2(Z) = X(Z) + BY_1(Z)$$

$$= X(Z) + BY_2(Z)\cdot Z^{-1} - BX(Z) \tag{4.1.8}$$

$$\frac{Y_2(Z)}{X(Z)} = (1 - B)\ \frac{1}{1 - BZ^{-1}} \tag{4.1.9}$$

$$Y(Z) = Y_2(Z) - AY_2(Z)\cdot Z^{-1} \tag{4.1.10}$$

$$= (1-AZ^{-1})\ Y_2(Z) \tag{4.1.11}$$

Substituting for $Y_2(Z)$ results in

$$Y(Z) = (1 - B)\ \frac{(1 - AZ^{-1})}{(1 - BZ^{-1})}\ X(Z) \tag{4.1.12}$$

Thus

$$H(Z) = (1 - B)\frac{(1 - AZ^{-1})}{(1 - BZ^{-1})} \qquad (4.1.13)$$

The transfer function from the input to the output of the new adder is

$$\frac{Y_1(Z)}{X(Z)} = \frac{-1 - Z^{-1}}{1 - BZ^{-1}} \qquad (4.1.14)$$

The maximum of this transfer function occurs at the Nyquist frequency as in Fig. 3.1 (a) and its value is

$$M_{max} = \frac{2}{1 + B} \qquad (4.1.15)$$

Since $|B| < 1$, M_{max} is greater than unity. Therefore to avoid overflow in this adder we must be certain the input does not exceed $\frac{1 + B}{2}$.

In the special case when A = 0 the above two configurations become as shown in Fig. 4.3.

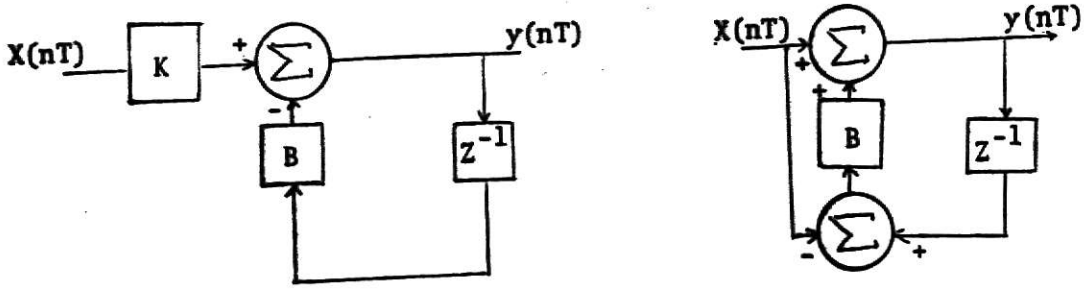


Fig. 4.3. First-order filter configuration with A=0.

4.2. A Recent Approach Using Read Only Memories (ROM's)

A recently suggested method involves storage of finite number of possible outcomes of an intermediate arithmetic operation and performing certain operations such as shifting and adding on these to obtain the next output sample. This method is easily explained using the following examples.

When numbers are represented in two's complement notation as explained in Sec. 2,3, any X, which is bounded by $\pm$ 1, is given by

$$X = -X^0 + \sum_{j=1}^{M} X^j \, 2^{-j} \tag{4.2.1}$$

Here X is represented by M + 1 bits as $X^0 \; X^1 \; X^2 \; . \; . \; . \; X^M$ where $\{X^j\}_{j = 0, 1 \; . \; . \; . \; M} = 0$ or 1. Recall that X^0 is the sign bit which is conventionally 0 for positive values and 1 for negative values of X. For the simple first order network represented by

$$H(Z) = \frac{1 - B}{1 - BZ^{-1}} \tag{4.2.2}$$

the nth output is given by

$$Y_n = (1 - B) \; X_n + BY_{n-1} \tag{4.2.3}$$

Substitution of Eq. (1) in Eq. (3) leads to

$$Y_n = (1 - B) \; [-X_n^0 + \sum_{j=1}^{M} X_n^j \, 2^{-j}]$$

$$+ B \; [-Y_{n-1}^0 + \sum_{j=1}^{M} Y_{n-1}^j \, 2^{-j}] \tag{4.2.4}$$

i.e.,
$$Y_n = \sum_{j=1}^{M} [2^{-j} \{(1 - B) \; X_n^j + BY_{n-1}^j\}] -$$

$$(1 - B) \; X_n^0 + BY_{n-1}^0 \tag{4.2.5}$$

We define a Boolean function, $\emptyset\ (x,y)$ as

$$\emptyset\ (x,y) = (1 - B)\ x + By \qquad (4.2.6)$$

where x and y are binary variables. Then Eq. (5) can be written as

$$Y_n = \sum_{j=1}^{M} 2^{-j}\ \emptyset\ (X_n^j\ Y_{n-1}^j) - \emptyset\ (X_n^o,\ Y_{n-1}^o) \qquad (4.2.7)$$

The function $\emptyset$ has only four possible values which are 0, 1-B, B and 1 represented in binary form as explained in Table 4.1. The corresponding 4 bit representation for $\emptyset$ when B = 0.75 is also included.

Table 4.1. Values of $\emptyset$ to be stored in a ROM.

X	Y	$\emptyset$	4 bit representation of $\emptyset$ when B = 0.75
0	0	0	0.000
1	0	1 - B	0.010
0	1	B	0.110
1	1	1	1.000

The implementation in this form is best understood by studying the following example.

Let $$X_n = X_n^0 \cdot X_n^1\ X_n^2\ X_n^3$$

$$= 0 \cdot 0\ 1\ 0$$

and $$Y_{n-1} = Y_{n-1}^0 \cdot Y_{n-1}^1\ Y_{n-1}^2\ Y_{n-1}^3$$

$$= 0 \cdot 1\ 0\ 1$$

Then Y_n according to Eq. (3) is given by

$$Y_n = 0.25\ X_n + 0.75 \cdot Y_{n-1}$$

$$= (0.010\ X\ 0.010) + (0.110\ X\ 0.101)$$

$$= 0.000100 + 0.011110$$

$$= 0.100010$$

which when rounded to 4 bits yields

$$Y_n = 0.100$$

We now calculate Y_n according to Eq. (7). From this equation it follows that

$$Y_n = \emptyset\ (X_n^3\ Y_{n-1}^3)\ 2^{-3} + \emptyset\ (X_n^2\ Y_{n-1}^2)\ 2^{-2} + \emptyset\ (X_n^1\ Y_{n-1}^1)\ 2^{-1} - \emptyset\ (X_n^0\ Y_{n-1}^0) \qquad (4.2.8)$$

for M = 3 as in this example. Substituting the given values of X_n^j and Y_{n-1}^j for j = 0,1,2 and 3 we obtain

$$Y_n = \emptyset\ (0,1)\ 2^{-3} + \emptyset\ (1,0)\ 2^{-2} + \emptyset\ (0,1)\ 2^{-1} - \emptyset\ (0,0)\ \ldots \qquad (4.2.9)$$

Using Table 4.1

$$Y_n = 0.11 \times 0.001 + 0.01 \times 0.01 + 0.11 \times 0.1 - 0.000$$

$$= 0.00011 + 0.0001 + 0.011$$

$$= 0.100010$$

When rounded, $Y_n = 0.100$ which is the same as above. We remark that the result has been obtained without any direct multiplication since multiplication by 2^{-j} in Eq. (9) is realized by shifting.

As a second example, let $X_n = 1.010$ and $Y_n = 1.000$. Then Eq. (3) yields

$$Y_n = (1.010 \times 0.010) + (1.000 \times 0.110)$$

$$= 1.110100 + 1.010000$$

$$= 1.000100$$

Again from Eq. (8) we obtain

$$Y_n = \emptyset\ (1,0)\ 2^{-2} - \emptyset\ (1,1)$$

$$= 0.010 \times 0.01 - 1.000$$

$$= 0.00010 + 1.000^*$$

$$= 1.00010$$

To implement the filter via Eq. (7) we use a ROM, with its contents being as specified in Table 4.2.

Table 4.2. Storage of $\emptyset$ in ROM

ROM address	Contents
0 0	$\emptyset\ (0,0)$
0 1	$\emptyset\ (0,1)$
1 0	$\emptyset\ (1,0)$
1 1	$\emptyset\ (1,1)$

We have X_n and Y_{n-1} in 4 bit shift registers as shown in Fig. 4.4.

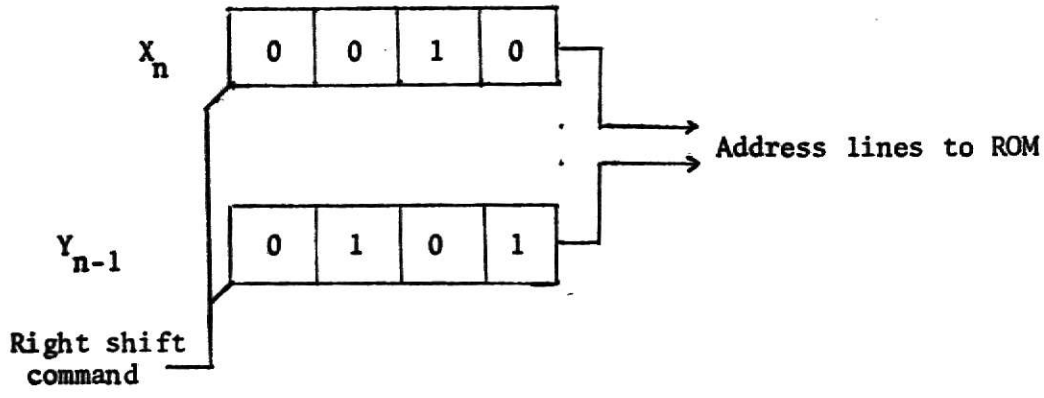


Fig. 4.4. Addressing the ROM.

*Note: 2's complement of 1.000 is 1.000

The least significant bits of these registers are used to address the ROM. We now consider the complete filter implementation as illustrated in Fig. 4.5. The contents of location addressed in ROM is brought out to register R_d. The first value taken from ROM is to be multiplied by 2^{-M}; see Eq. (7) for j = M. This is done by shifting the data in R_d, M places to the right and then adding it to the contents of R_o which is cleared at the start of the sequence of computations for each Y_n. The sum is stored in R_o. Next, the registers which hold X_n and Y_{n-1} are shifted one place to the right. The corresponding $\emptyset$ $(X_n^{M-1}\ Y_{n-1}^{M-1})$ is retrieved from the ROM and placed in R_d. The contents of R_d are shifted (M-1) places to the right and subsequently added to current contents of R_o. This process is repeated until j=0. For j=0 we need to subtract the value $\emptyset$ $(X_n^0\ ,\ Y_{n-1}^0)$ from the contents of R_o. The resulting contents of R_o is the desired output, Y_n.

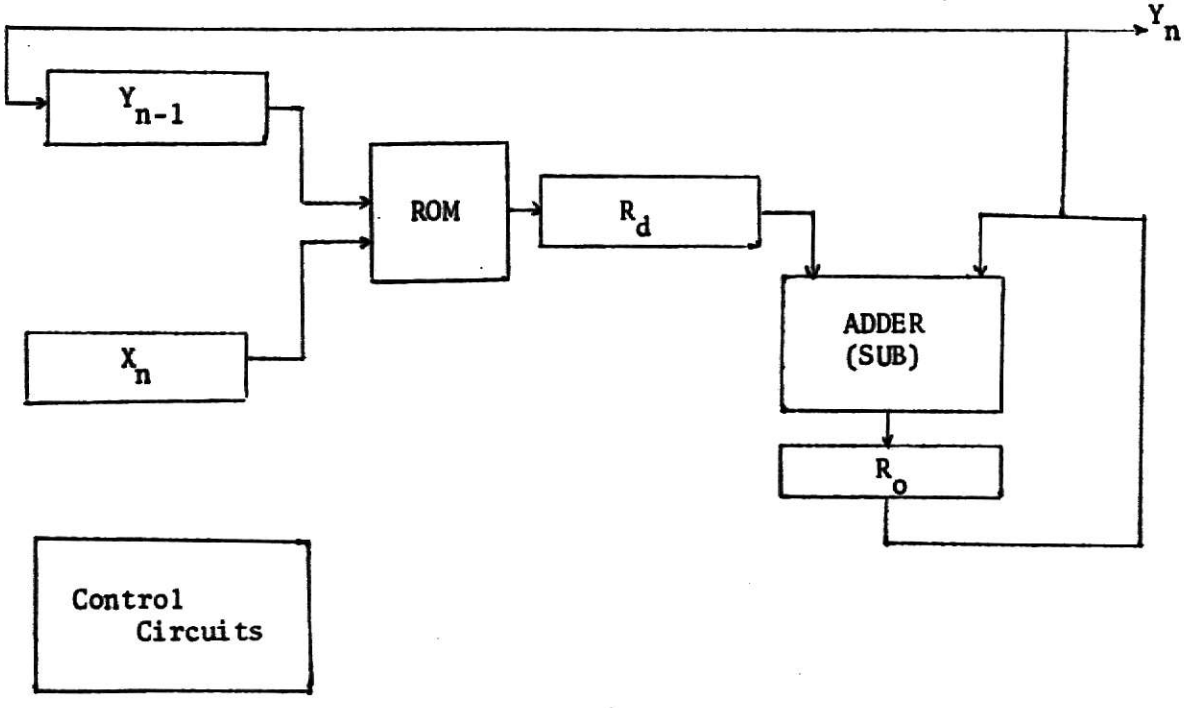


Fig. 4.5. First order filter with no multipliers.

The advantage of this method may not be obvious for the simple case considered above. The important implications of this method are apparent when higher order transfer functions are to be implemented. For example, consider

$$H(Z) = \frac{a_0 + a_1 Z^{-1} + a_2 Z^{-2}}{1 + b_1 Z^{-1} + b_2 Z^{-2}} \qquad (4.2.10)$$

Then

$$Y_n = a_0 X_n + a_1 X_{n-1} + a_2 X_{n-2} - b_1 Y_{n-1} - b_2 Y_{n-2} \qquad (4.2.11)$$

From Eq. (11) it follows that there are at least 5 multiplications to be carried out if this filter is implemented directly. However, using the method discussed above, we need only to store the Boolean function ϕ which is now a function of 5 binary variables and hence can have one of 32 different values.

$$\text{i.e.,} \quad \phi (x_1 x_2 x_3 y_1 y_2) = a_0 x_1 + a_1 x_2 + a_3 x_3 - b_1 y_1 - b_2 y_2 \qquad (4.2.12)$$

Hence Eq. (11) can be written as

$$Y_n = \sum_{j=1}^{M} 2^{-j} \phi (X_n^j, X_{n-1}^j, X_{n-2}^j, Y_{n-1}^j, Y_{n-2}^j) - \phi (X_n^0, X_{n-1}^0, X_{n-2}^0, Y_{n-1}^0, Y_{n-2}^0) \qquad (4.2.13)$$

As before the 32 possible values of ϕ are stored in a ROM. The related implementation is as shown in Fig. 4.6.

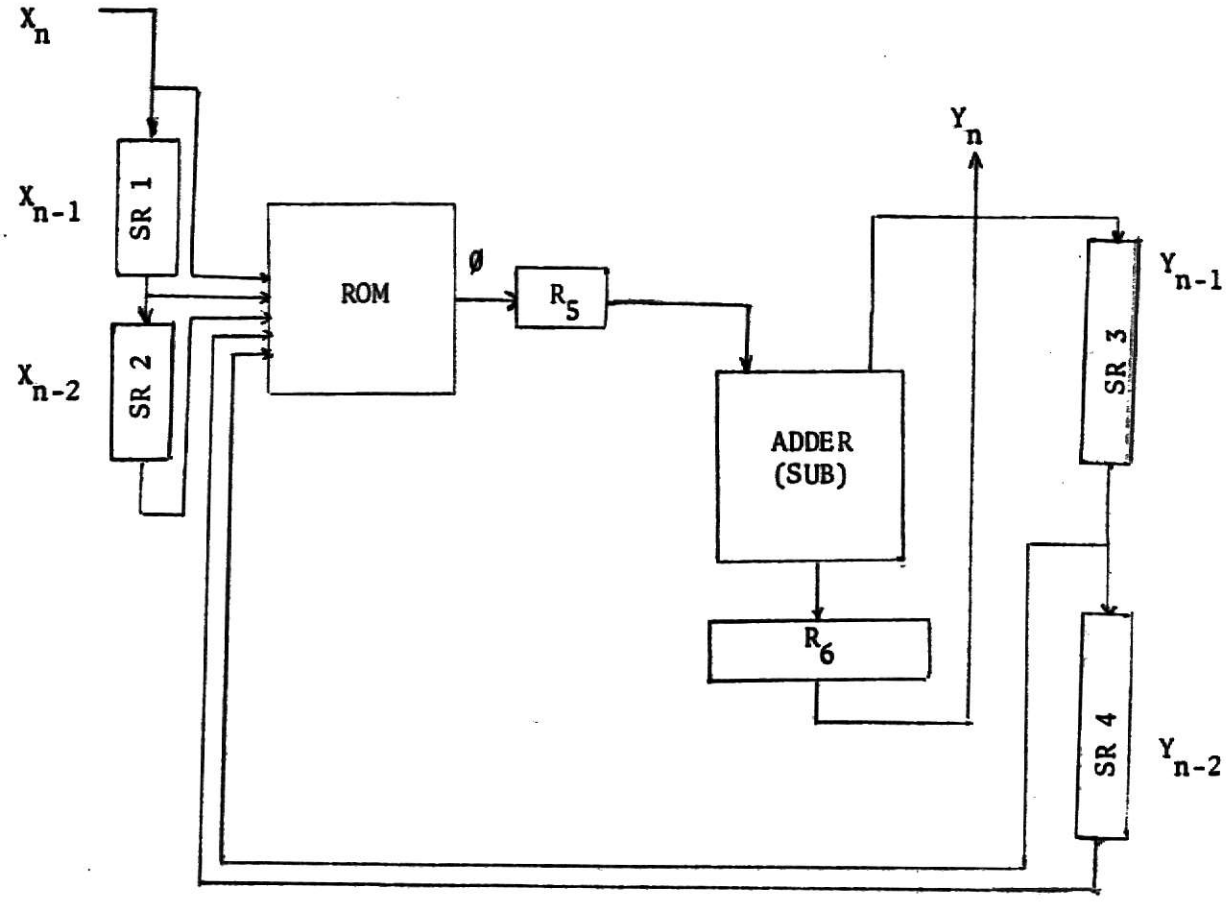


Fig. 4.6. Realization of second order network using the new approach.

The operation of the realization in Fig. 4.6 is similar to that considered earlier; see Fig. 4.5. The size of the ROM is larger in this case since 32 different values need to be stored. Hence five bits for addressing are required. If it is necessary to change the properties of the filter all we need to do is to change the contents of the ROM appropriately. This will lead to the realization of a second order filter with entirely different coefficients and characteristics.

The advantages of this method over the conventional method are as follows:

1. Greater speed of calculation
2. Flexibility
3. Low cost
4. Low power consumption

CHAPTER 5

IMPLEMENTATION USING MICROPROCESSORS

5.1. Introduction

With recent advances in microprocessor technology almost any digital control or processing system of superior performance can be designed with relative ease. Digital filter implementation as discussed in Chapter 4 involves arithmetic and logic units as well as some control circuitry. A digital filter should also be able to handle input from an A/D converter and output the result to a D/A converter. Under program control, a microprocessor can perform arithmetic and logic operations, handle input and output and interface memory. These capabilities directly match the needs that arise in realizing a digital filter. The main objective of the work reported here is to examine some aspects of implementing digital filters using microprocessors. A recursive low pass filter has been implemented using Intel 8008 microprocessor system. The choice of the processor was limited mainly by the availability when this study began. However, studies show that flexible and efficient filters can be implemented using the present day microprocessors. The main advantage of using microprocessors compared to hard-wired logic are as follows:

i) Low cost and complexity

ii) More computational and control capability of microprocessors.

Though the design procedures are at present new and different from conventional digital design techniques, with a little more familiarity these are expected to become simpler. An added advantage in using microprocessors

is that a multisection filter can be implemented with very little or no additional hardware than that required for a single section. For example, a filter that needs to be realized as a cascade of several second order filter sections can be implemented by repeatedly running a general second-order filter program, with necessary changes in the filter coefficients in a microprocessor.

In this chapter we will study the important characteristics of the microprocessor used, the design of the filter mentioned above, and the corresponding results.

5.2. Characteristics of the Intel 8008 Microprocessor

Intel 8008 is an 8 bit parallel processor. It is a single bus microprocessor which makes the systems built with this microprocessor relatively slow. It can execute a set of 48 instructions which includes data movement and manipulation, binary arithmetic and jump to subroutine. The microprocessor consists of the registers listed below.

Register	Function	Register Size
A	Accumulator	8 bits
B,C,D and E	Data storage registers	8 bits each
H and L	Data storage and memory address registers	8 bits each
PC	Program Counter	14 bits

PC is on top of a last-in-first-out (LIFO) memory stack which has a total of 8 registers with 14 bits each. These 14 bits permit direct addressing of 16 K words in the memory. The memory stack enables a maximum subroutine nesting of 7 levels. The flip-flops or flags are denoted as follows:

C - carry and borrow

S - sign

P - parity

Z - zero

Addressing Modes. Four addressing modes are available.

(i) Direct addressing: In this address mode the address of the operand is given in the instruction. This type of instructions are 3 bytes long.

(ii) Indirect addressing: Contents of memory registers H and L are used as the effective address. These are one byte instructions.

(iii) Implied addressing: These are inter-register and intra-register operations.

(iv) Immediate addressing (or Immediate data): These have the operand in the second byte of the instruction.

Intel 8008 can be interfaced to 8 input devices and 24 output devices. The set of instructions and their operations are explained in Appendix A. This directly reflects on the capabilities of the microprocessor. As one can note, arithmetic and logic operations as well as rotate data operation are possible only in the accumulator (i.e. register A).

5.3. Filter Design

The design process is discussed briefly in what follows.

Step 1. Determination of Coefficients. To design a single pole low pass filter with cut off frequency at $\eta_c = 0.046$ where η_c is the cut off frequency normalized with respect to the sampling frequency.

The general recursive first order filter has a transfer function

$$H(Z) = K\left(\frac{1 - AZ^{-1}}{1 - BZ^{-1}}\right) \qquad (5.3.1)$$

We assume that A = 0; then

$$H(Z) = \frac{K}{1 - BZ^{-1}} \qquad (5.3.2)$$

A steady-state frequency characteristic of the above filter is given by

$$|H(w)|^2 = \left|\frac{K}{1 - Be^{jwt}}\right|^2$$

$$= \left|\frac{K^2}{(1 - b\cos wT) - jB\sin wT}\right|^2$$

$$= \frac{K^2}{1 + B^2 - 2B\cos wT}$$

We have $wT = 2\pi f/f_s = 2\pi\eta$

Thus

$$|H(\eta)|^2 = \frac{K^2}{1 + B^2 - 2B\cos(2\pi\eta)}$$

For any K, the cut off frequency (i.e., the 3db point) is at $\eta = \eta_c$ = 0.046 when B is chosen to be equal to 0.75. Thus

$$H(Z) = \frac{KZ}{Z - 0.75} \qquad (5.3.3)$$

Step 2. Gain scale factor. We now need to determine the gain scale factor according to Eq. (3.2.14). Since A = 0, we have

$$K = 1 - B = 0.25 \qquad (5.3.4)$$

Hence

$$H(Z) = \frac{(1 - B)Z}{Z - B}$$

$$= \frac{0.25Z}{Z - 0.75} \qquad (5.3.5)$$

The input-output relation for the filter represented by Eq. 5 is

$$Y_n = 0.25\,X_n + 0.75\,Y_{n-1} \qquad n = 0,1,2\ldots \qquad (5.3.6)$$

where Y_n is the nth output.

In other words, we calculate the outputs respectively, as follows:

$$Y_0 = 0.25\ X_0$$

$$Y_1 = 0.25\ X_1 + 0.75\ Y_0$$

$$Y_2 = 0.25\ X_2 + 0.75\ Y_2, \text{ etc.} \qquad (5.3.7)$$

Step 3. Word lengths. The input word length is determined as explained in Sec. 3.3. If $\frac{X_{sat}}{X_{th}} = 100$, for the A/D converter, then at least $\log_2(100)$ bits are needed. Thus the desired input word length is 7 bits. Since the range of the A/D varies from $-X_{sat}$ to $+X_{sat}$, we need 8(=7 + 1) bits to represent all the quantized levels in two's complement for M. If the tolerable quantization noise at A/D output is specified, then Eq. (3.3.10) is used to calculate the input bits according to Eq. (3.3.11). For an 8 bit input word length, the quantization error results in a signal-to-noise ratio of 43 db which is obtained using Eq. (3.3.10).

A/D converters often use off set binary coding in which case the input-output relationship of the A/D conversion is as shown in Table (5.1), where q is the quantization interval.

Table 5.1.

Analog input	Digital output
X_{sat}	0 0 0 0 0 0 0 0
--------	--------
+q	0 1 1 1 1 1 1 1
0	1 0 0 0 0 0 0 0
-q	1 0 0 0 0 0 0 1
--------	--------
$-X_{sat}$	1 1 1 1 1 1 1 1

In such cases, two's complement representation is easily obtained by inverting the sign bit for all A/D outputs. However, note the sign change compared to Eq. (4.2.1).

Here X is given by

$$X = X^0 - \sum_{j=1}^{M} X^j \, 2^{-j} \qquad (5.3.8)$$

As explained in Sec. 3.3, the computational word length is determined according to Eq. (3.3.22), which is

$$M + 1 = (0.8 + \frac{F}{6}) + \frac{10}{6} \log_{10} (N.G) \qquad (5.3.9)$$

N.G, the noise gain of the filter is represented by Eq. (3.3.23) to be

$$NG = \frac{1 - B}{1 - B^2} = \frac{1}{1 + B} \qquad (5.3.10)$$

If the round off noise figure allowed is 40 db, then for B = 0.75 we obtain M + 1 = 8 using Eq. (9).

An Intel 8008 microprocessor was chosen for the purposes of this study. However, fixed word length microprocessors are currently available for different computational word lengths such as 4 bits, 8 bits, 12 bits and 16 bits. Two bit slices of Intel 3000 type can be used for any word length one may choose. For example, if a 10 bit computational word length is required, a machine using five Intel 3000 chips can be constructed.

The coefficient word length is generally determined as explained in Sec. 3.3. However, in this case it is chosen to be 8 bits.

Step 4. Determining the software and hardware needed.

Software: The microprocessor is programmed to do the computations needed depending on the particular configuration that is chosen. The configuration shown in Fig. 4.1.3 (b) has a limitation on the input. In other words, this configuration limits the input dynamic range as mentioned in Sec. 4.1. Therefore, the configuration in 4.1.3 (a) was chosen for implementing the filter. The algorithm for filtering is as follows.

(1) Test end of conversion, (EOC) output of the A/D, and if the conversion is complete, then read in the output of the A/D converter.

(2) Convert this data into two's complement form if it is not already represented in two's complement notation.

(3) Multiply each data point, X_n by (1 - B) = 0.25.

(4) Add it to the contents of a storage register SR, which holds B times the previous output, Y_{n-1}.

(5) The result is transferred to the output port where the D/A or any network that needs to process the output is connected.

(6) The result is also multiplied by B and stored in the storage

register SR, mentioned above.

(7) Repeat above steps.

The above algorithm can be flowcharted as in Fig. 5.1.

This algorithm translated into the assembly language of Intel 8008 is given in Appendix B. The filter was tested using the Intel 8008 simulator which is available on the IBM system of the Computing Center of Kansas State University. An 8 bit successive approximation A/D convertor and a bipolar D/A converter, both working with offset binary coding were simulated using the programs given in Appendix B.

Hardware: While using a microprocessor system, we could use just the minimum amount of hardware and peripherals needed. For example, to implement the filter at hand, we need a microprocessor, data converters, interface circuitry necessary to have just one input port and one output port, and less than a page of memory. In an 8008 system, the chip count may not be low. But, in the more recent microprocessors, the necessary hardware can be realized with two or three chips and the related data converters.

5.4. Simulation Results

To test the characteristics of the filter the following input signal was synthesized:

$$X(mT) = 1 + \sum_{n=1}^{16} \cos\,[2\pi\, n(m-1)\, T] + \sum_{n=1}^{16} \sin\,[2\pi\, n(m-1)\, T] \qquad (5.4.1)$$

$$m = 1,2, \ldots, 64$$

The spectrum of this signal is as shown in Fig. 5.2.

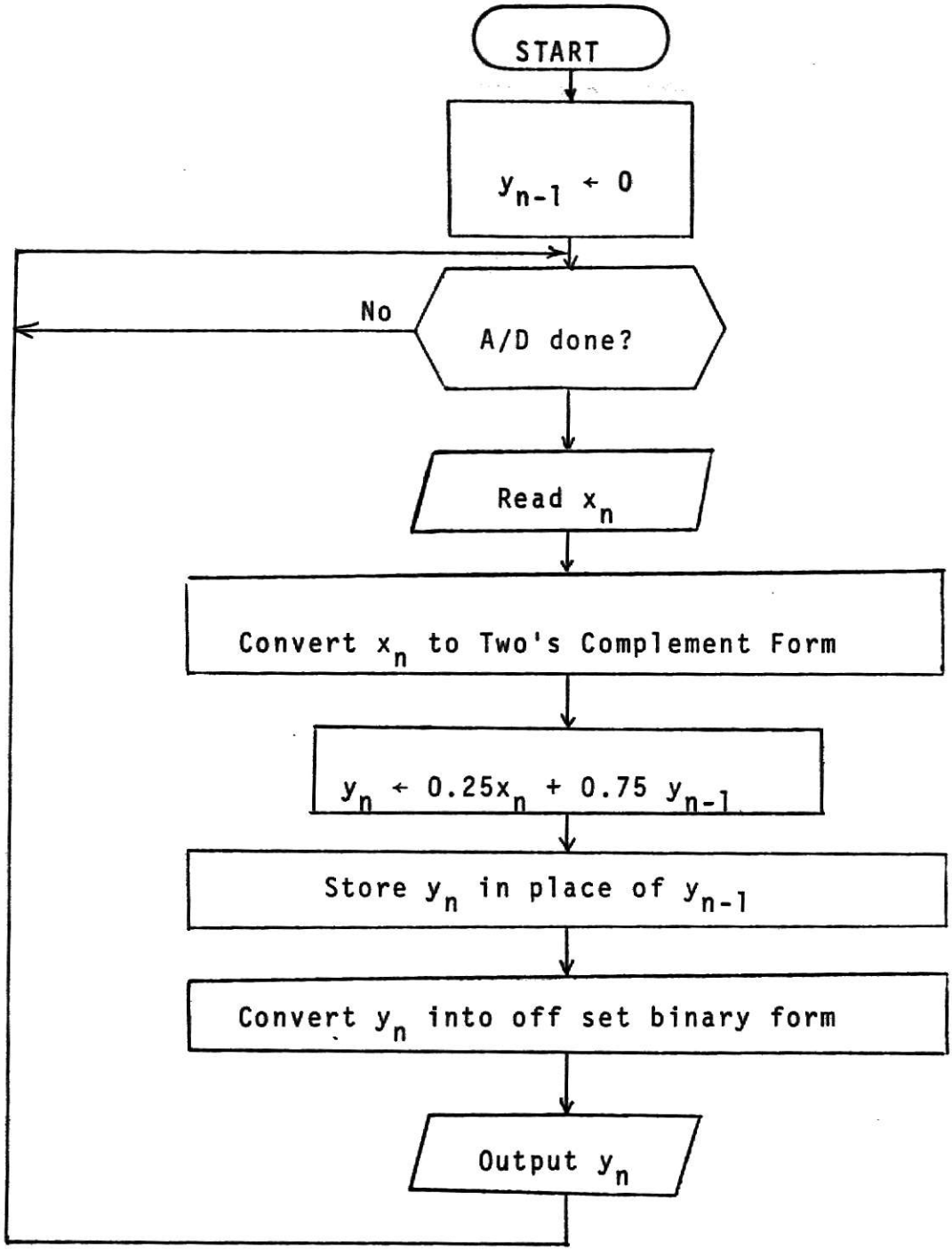


Fig. 5.1. The Filter Algorithm

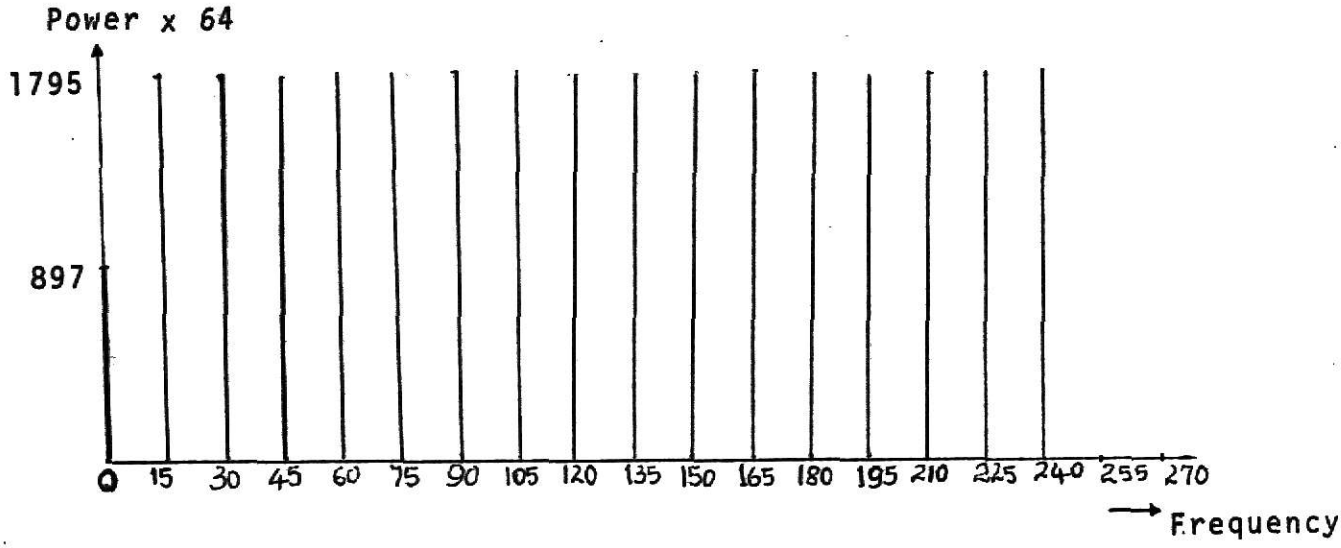


Fig. 5.2. Spectrum of input signal.

This signal was sampled at a rate of 980 samples/sec and the sample values were converted to digital format by the A/D simulator given in Appendix B. The "ideal case" output of the filter for 100 samples of the input, as given by Eq. (5.3.6), was calculated using IBM 370. By the term "ideal case" it is implied that each sampled value is represented by 32 bits (i.e., one word in the IBM 370 computer). The first 36 outputs were ignored to eliminate transients. The spectrum of the remaining 64 output values was calculated using discrete fourier transform. This spectrum is considered the ideal case with which the output spectrum of the Intel 8008 filter is compared. The output of the 8008 filter which is in digital format was processed through the D/A simulator program and then its spectrum was calculated. This comparison is shown in Fig. 5.3 and Fig. 5.4. After each multiplication, the product was truncated to 8 bits for the case illustrated in Fig. 5.3, while the products were rounded to 8 bits for the case illustrated in Fig. 5.4. As it was stressed earlier, rounding off yields better results compared

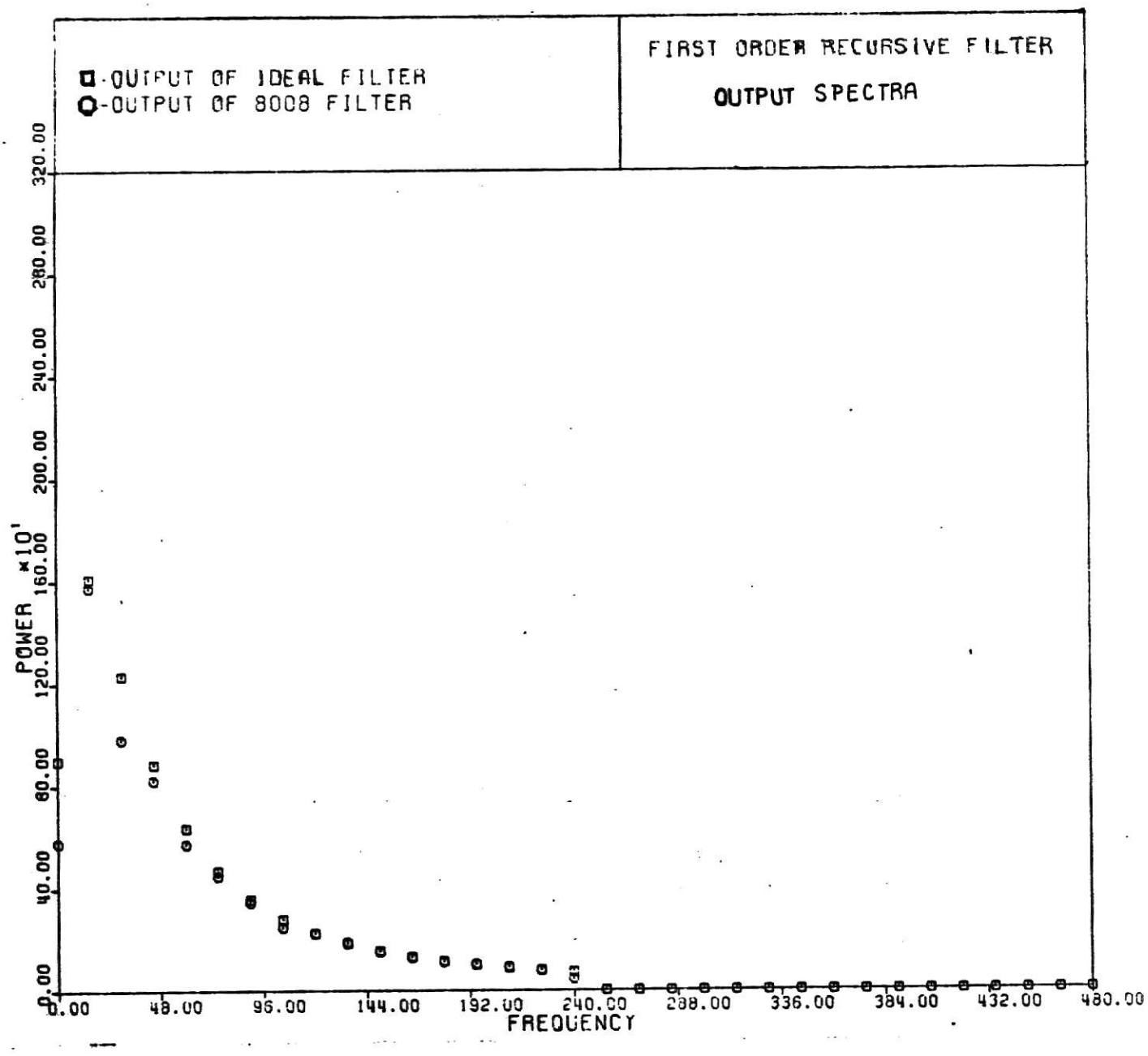


Fig. 5.3. Output spectra when products are truncated.

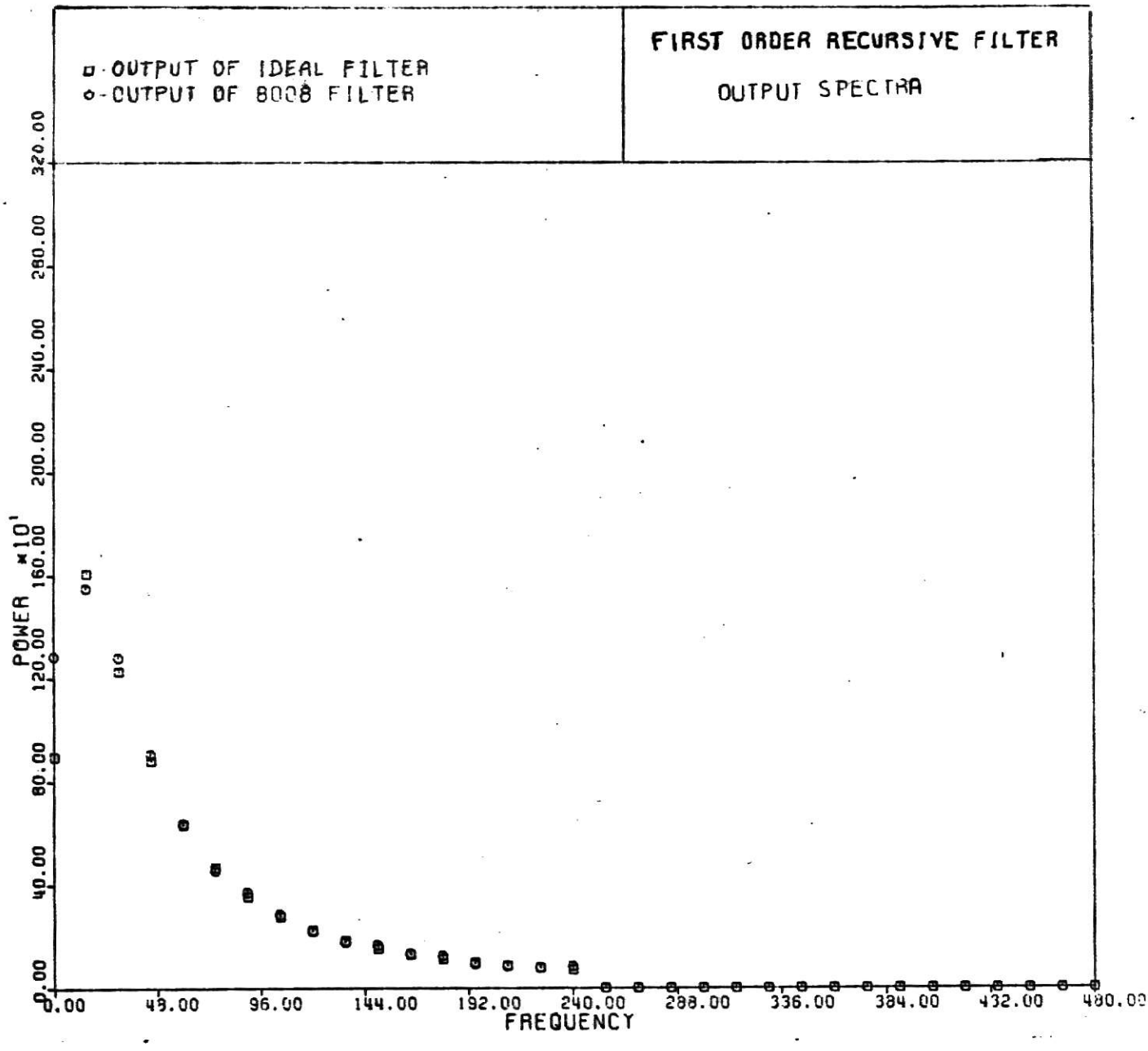


Fig. 5.4. Output spectra when products are rounded.

to truncation.

These results illustrate the accuracy that can be achieved even with word lengths limited to 8 bits. The difference in the D.C. level of the signal in the two cases illustrated above is yet to be explained completely. The fact that a change in one output recursively affects all the outputs thereafter hints towards this large difference.

Processing time. On the average the processing time for each input is approximately 5 m sec. This means that if one seeks real-time filtering the sampling rate is limited to 200 H_z. The present-day microprocessors are 5 to 10 times faster than Intel 8008 and hence one can increase the sampling rate. It is important to note that the processing time will be greater for second-order filters and hence the use of the configuration explained in Sec. 4.2 will considerably increase the speed of processing.

Currently available microprocessors with some of their characteristics are listed in Table 5.2. Cost of the microprocessor chips vary widely from time to time and the cost is also dependent on the source. Hence the cost is not listed in the table.

Table 5.2.

Microprocessor	Clock KH_z/Phases	Number of Instructions	Memory Address Capacity	Register Add Time M Sec	Remarks
Intel 8008-1	800/2	48	16 K	12.5	Slow
Intel 8080	2083/2	78	64 K	2	
Motorola M6800	1000/2	72	64 K	2	Arithmetic shift and rotate memory words operations can be useful in filter implementation
National IMP 16	715/4	59	64 K	2	
Rockwell PPS-8	56/2	90	16 K	4	

CHAPTER 6

CONCLUSIONS

Some aspects pertaining to the design and implementation of digital filters via microprocessors were presented in this report. The simulation results presented indicate that first-order filter with limited word length (e.g. 8 bits) realizations using microprocessors perform satisfactorily in the sense that their performance compares well with that of the corresponding ideal case. It is reasonable to assume that this would be the case for second order filters also.

Many improvements over the Intel 8008 are included in the microprocessors that are currently available. For example, Intel 8080 is ten times faster, while Motorola M6800 has more arithmetic and data manipulation capabilities. This means that the same algorithm can be implemented using lesser number of instructions as well as each instruction gets executed in a shorter duration.

Conventional implementation using hard-wired logic will continue to be preferred over implementation through microprocessors since hard-wired logic uses the conventional design techniques. When digital filters are to be used in large numbers such as in telephony then custom LSI chips will be preferred over microprocessors. Implementation using microprocessors is expected to become familiar in fields where complicated transfer functions are to be realized and where real-time filtering is of secondary importance. As outlined by S. D. Stearns [3] Butterworth filters can be realized using a convenient design technique. Such filters can be implemented through microprocessors with more ease than with conventional techniques.

Appendix A

The complete instruction set for Intel 8008 is explained in this Appendix.

A. Data Movement

No condition flip-flops are affected by these instructions.

Notations: M stands for Memory location specified by contents of H and L. r, r_1, r_2 stand for registers A, B, C, D, E, H and L unless otherwise specified. C(X) stands for contents of X. BB BBB BBB is 8 bit binary data.

No. of bytes	Instruction	Address mode	Description of Operation
1	MOV r_1, r_2	Implied	Load register r_1 with C(r_2)
1	MOV r, M	Indirect	Load register r with C(M)
1	MOV M, r	Indirect	Store C(r) in memory M
2	MVI r BB BBB BBB	Immediate	Load register r with data B . . . . B
2	MVI M BB BBB BBB	Immediate and Indirect	Load memory location M with data B . . . . B

B. Arithmetic and Logic

1	INR r	Implied	Increment C(register r) r ≠ A S,Z, and P are affected
1	DCR r	Implied	Decrement C(register r) Y ≠ A S, Z, and P are affected

All flag flipflops are affected in the following. These are possible only with Accumulator (register A).

1	ADD r	Implied	Add C(r) to C(A) Overflow sets carry
1	ADD M	Indirect	A←C(A) + C(M) Overflow sets carry
2	AD I BB BBB BBB	Immediate	A←C(A) + B . . . . B Overflow sets carry

The above three addressing modes are possible in the following instructions.

```
ADC r
ADC M      } Addition with carry          e.g. ADC r results in
ADI          overflow sets carry                A←C(A) + C(r) + (carry)
BBBBBBBB

SUB r
SUB M      } Subtraction                  e.g. SUI BB BBB BBB
SUI        } Underflow sets carry (borrow)    A← C(A) - BBBBBBBB
BBBBBBBB

SBB r
SBB M      } Subtract with borrow         e.g. SBB M results in
SBI BBBBBBBB Carry flip flop set by underflow A←C(A) - C(M) - (carry)
```

Logic operations affect all flipflops; they reset carry to zero.

AND	EX-OR	OR
ANA r	XRA r	ORA r
ANA M	XRA M	ORA M
ANI BBBBBBBB	XRI BBBBBBBB	ORI BBBBBBBB

eg. XRA M results in A ← C(A) $\oplus$ C(M)

Compare

```
CMP r      } Compare C(r) or C(M) or B . . . . B with C(A)
CMP M      } C(A) is not changed.  All flags are affected
CPI
BBBBBBBB
```

Zero flipflop set if C(A) = operand
Carry flipflop reset if C(A) < operand
Carry flipflop set if C(A) $\geq$ operand.

Rotate Accumulator. (one byte)

RLC	(carry)← A_7, A_{n+1} ← A_n, A_0 ← A_7	Set carry = A_7 rotate acc. left
RRC	(carry)← A_0, A_n ← A_{n+1}, A_7 ← A_0	Set carry = A_0 rotate acc. right
RAL	A_{n+1} ← A_n, (carry)← A_7, A_0 ← (carry)	Rotate acc. left through carry
RAR	A_n ← A_{n+1} (carry)← A_0, A_7 ← (carry)	Rotate acc. right through carry

C. Branching and Subroutines (3 bytes)

JMP	ADDR	Unconditional jump to ADDR which is a label or a 14 bit address.
JC	ADDR	Jump if carry is true
JZ	ADDR	Jump if zero is true
JM	ADDR	Jump if sign is true (negative or minus result)
JPE	ADDR	Jump if parity is true (even parity)

JNC, JNZ, JP, and JPO are opcodes for jump if corresponding flag is zero.

CALL ADDR	Unconditional call of Subroutine Starting in Address ADDR Push stack

CC, CZ, CM, CPE, CNC, CNZ, CP, CPO are conditional subroutine calls.
Similar to conditional jumps

RET	Unconditional return to main program (one byte) (i.e. POP Stack)

RC, RZ, RM, RPE are conditional returns if corresponding flag is true.

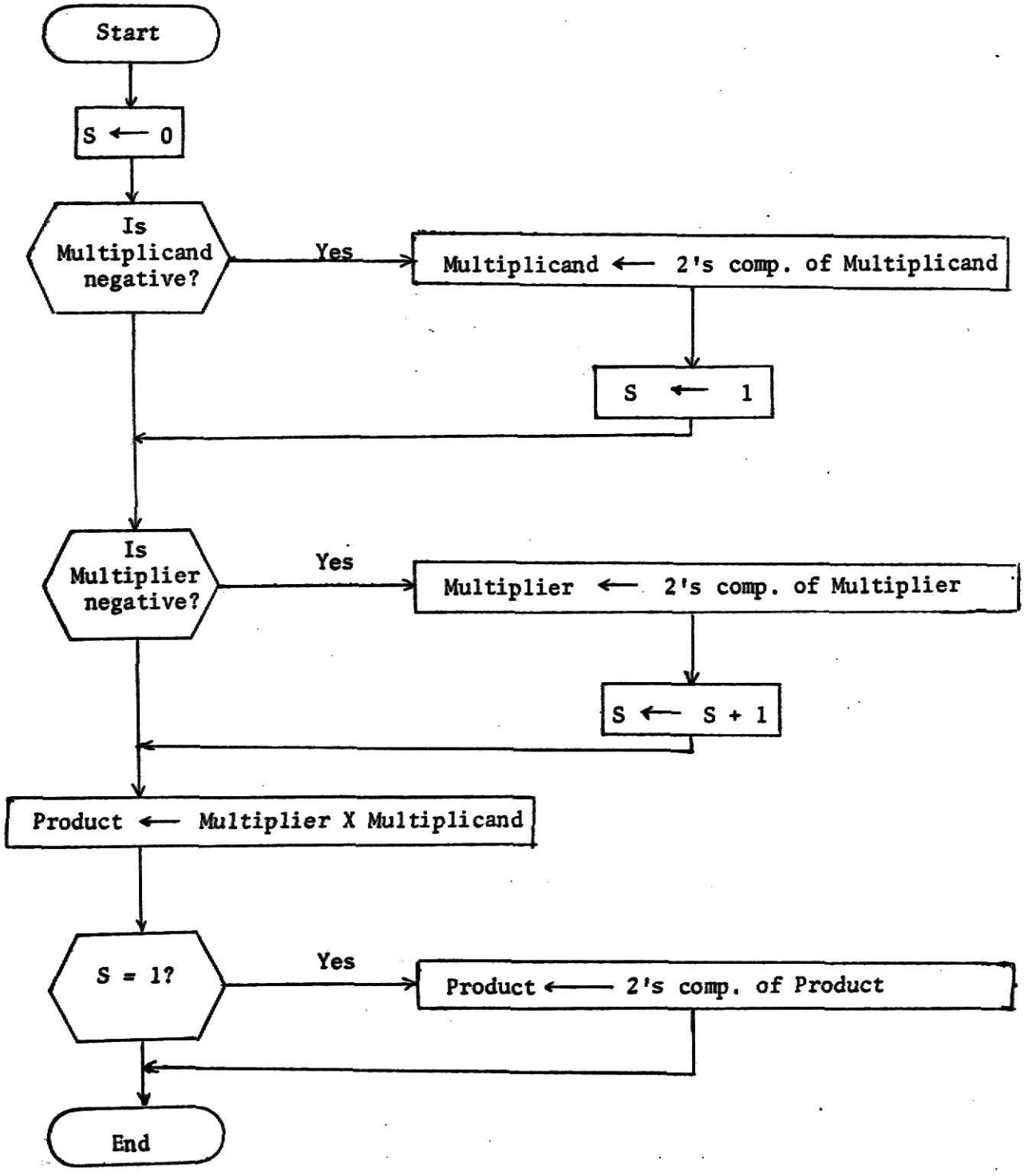
RNC, RNZ, RP and RPO are conditional returns if corresponding flag is false.

Machine instruction:

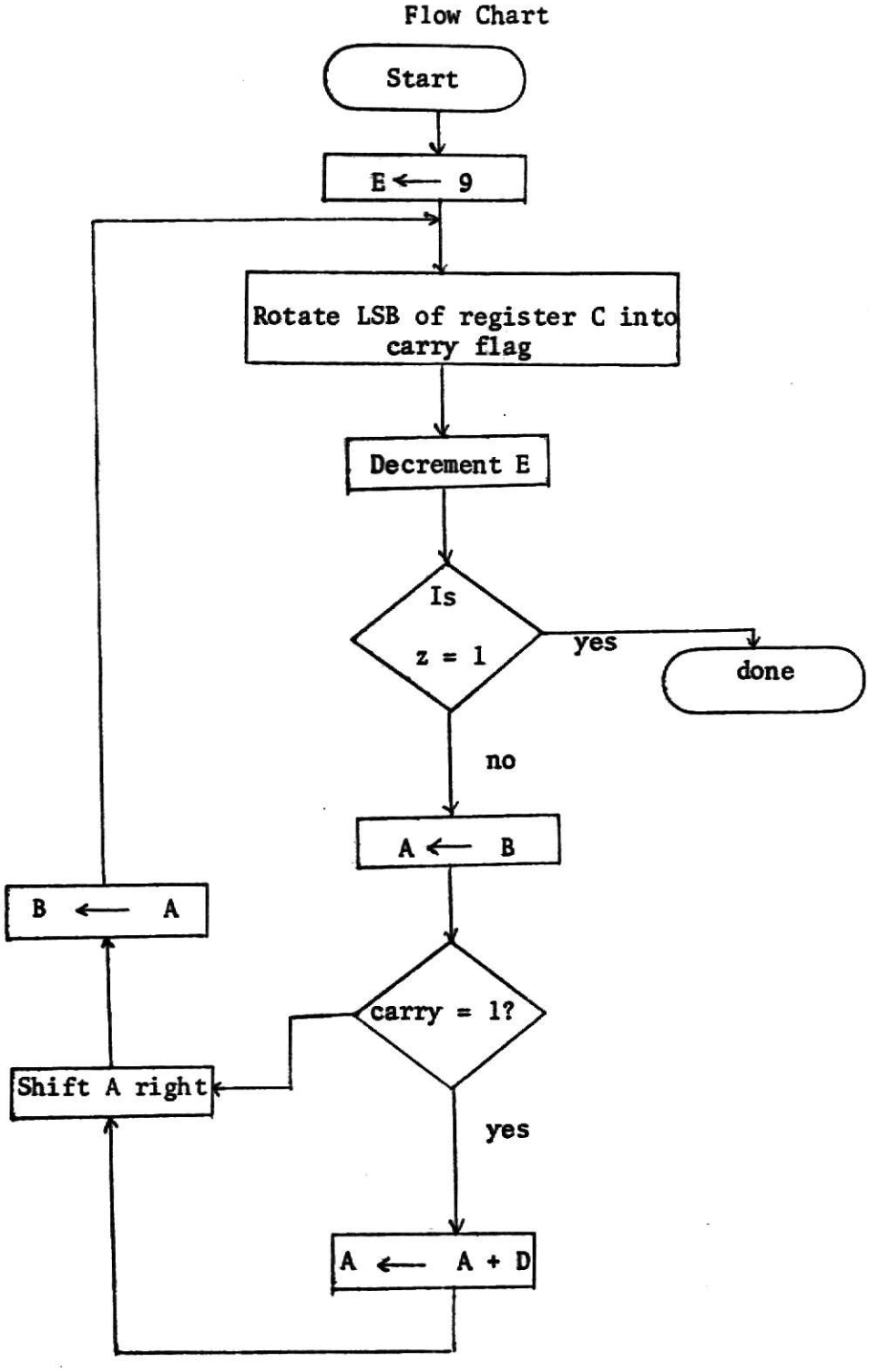
HLT causes execution to terminate.

Appendix B

B. 1. Two's Complement Multiplication: Multiplication in two's complement notation is done according to the following flow chart.



B.2. Subroutine for multiplying two 8 bit binary numbers using Intel 8008. This assumes that the multiplier is in register C and the multiplicand in Register D. The product is available in registers B and C at the end of execution.



Subroutine for multiplication (with rounding)

LABEL	OPCODE	OPERANDS (octal)	COMMENT
MULT	MVI	B,000	Clear B
	MVI	E,010	Load E with 8
MULTO	MOV	A, C	Bring multiplier
	RAR		Rotate LSB into carry
	MOV	C, A	Move shifted multiplier back to C
	DCR	E	Decrement contents of E
	JZ	DONE	If contents of E = 0 multiplication is done
	MOV	A, B	Move Partial Product into B
	JNC	MULT 1	Check LSB of multiplier which is now in carry
	ADD	D	Add multiplicand to Partial Product
MULT 1	RAR		Rotate Partial Product
	MOV	B, A	
	JMP	MULT 0	Go back to check next bit of multiplier
DONE	XRI	000	To set up sign flag appropriately
	JN	R00	Check MSB of least significant byte

```
          INR     B
R00       RET
```

B. 3. Filter Software

The filter is implemented through the following program. Here B_1 represents filter coefficient.

LABEL	OPCODE	OPERANDS (octal)	COMMENTS
Interrupt	RST		End of conversion from A/D restarts the processing by interrupt
	IN	0	Input data, X_n from device 0
	MVI	C,040	Load C with multiplier $(1 - B_1)$
	XRI	200	Change MSB of input to convert from offset to two's complement
	MOV	D, A	Move X_n into D
	JP	C00	Jump to 'multiply' subroutine
	MVI	L,000	Clear L
	INR	L	L is used as a mark register
	XRI	377	Complement contents of A.
	MOV	D, A	
	INR	D	Now two's complement of X_n is in D
C00	CAL	MULT	Call subroutine MULT. Returns rounded product in B
	MOV	A, B	
	DCR	L	
	JNZ	A00	Conditional branch depending on if X_n was complemented before multiplication

	XRI	377	Two's complement of Product
	ADI	001	
	ADD	H	Add B_1 Y_{n-1} to $(1-B_1)$ X_n
	MOV	H,A	Store this in H
	XRI	200	Convert Y_n to offset binary to suit D/A
	OUT	8	Output Y_n at port 8
	MOV	D,H	Bring C(H) into D
	MVI	C,140	Filter coefficient, B, is moved into C
	JN	C01	Jump if C(H) has MSB = 0
	XRI	177	Complement 7 LSB's since MSB is already complemented.
	MOV	D,A	
	INR	D	Completes 2's complement conversion
	MVI	L,000	Clear L
	INR	L	
CO 1	CAL	MULT	
	MOV	A, B	
	DCL		
	JZ	STR	
	XRI	377	
	ADI	001	
STR	MOV	H, A	Store B_1 Y_n in H; to be used as B_1 Y_{n-1} in next cycle 1
	HLT		

The machine is halted after processing. End of conversion, EOC signal from A/D jams into the microprocessor a restart instruction which

starts the processing again. The convert command of A/D is driven by the clock whose frequency is the sampling rate.

B.4. A/D Simulator. The following is a program written in Fortran which simulates a successive approximation A/D converter.

```
          READ 100, (X(M), M = 1, 100)
100       FORMAT (10F 8.6)
          READ 200, (J(I), I = 1,J)
C         J(I) ARE THE DIGITAL VALUES FOR
C         EACH BIT BEING ONE
C         DATA CARD FOR J(I) 100Ø 40Ø 20Ø 10ØØ 4ØØ 2ØØ 1
200       FORMAT    (7 I 3)
          DO 50    M = 1, 100
C         CONVERTS 100 DATA POINTS
          R = 5.0
C         R IS THE REFERENCE VOLTAGE
          IF (X(M))11, 11, 16
           L(M) = 200
12        DO 15    N = 1, 7
          IF (X(M) + R)  14,13,13
13        R = R - (5·/2· **N)
          GO TO 15
14        L(M) = L(M) + J(N)
          R = (R + 5/2 **N)
15        CONTINUE
          GO TO 50
```

```
16          L(M) = 0
            X(M) = X(M) - 10.0
            GO TO 12
50          CONTINUE
C           L(M)  CONTAINS THE DIGITAL OUTPUT
C           IN OCTAL FORM FOR THE 100 INPUTS.
```

B.5. D/A Simulator. D/A is simulated as follows:

```
C         X IS OUTPUT: M IS INPUT
            X = 0.0
            L = M - 200
            IF (L· GE·0)  M = L
            DO 2  N = 1, 7
C           J ARRAY IS SAME AS IN A/D CONVERTER
            I = M - J(N)
            IF (I·GE·0)  M = I
            IF (I)  2,1,1
1           X = X + 10/2 * *N
2           CONTINUE
            IF (L·LT·0)  X = 10 - X
            IF (L·GE·0)  X = -X
```

REFERENCES

[1] Current Digital Filter Signal Processing Techniques, National Engineering Consortium, Inc., Professional Growth in Engineering Seminar, 1974, vol. 2.

[2] L. R. Rabiner and B. Gold, Theory and Application of Digital Signal Processing, Prentice Hall, Inc., 1975.

[3] S. D. Stearns, Digital Signal Analysis, Hyden Book Company, Inc., 1975.

[4] A. Peled and B. Liu, "A New Hardware Realization of Digital Filters," IEEE Trans. Acoust., Speech, Signal Processing, vol. ASSP-22, pp. 456-461, Dec. 1974.

[5] B. Liu, "Effect of Finite Word Length on the Accuracy of Digital Filters - A Review," IEEE Trans. Circuit Theory, vol. CT-18, pp. 670-677, Nov. 1971.

[6] Microcomputers I: Basic Concepts and Applications, National Engineering Consortium, Inc., Professional Growth in Engineering Seminar, 1974, vol. 1.

SOME CONSIDERATIONS OF DIGITAL FILTER IMPLEMENTATION USING MICROPROCESSORS

by

JAY PURUSHOTHAMAN JAYAPALAN

B.Sc., University of Madras, 1971
B.E., Indian Institute of Science, 1974

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Electrical Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1976

Digital filters have been considered for various applications which include process control, guidance control, data acquisition, and digital signal processing. However, the hardware implementation of digital filters is yet a developing field. To this end the main objective of this report is to examine some aspects of implementing digital filters using microprocessors. The theory and practice of hardware implementation of digital filters is developed from the fundamentals assuming only the knowledge of Z transforms on the part of the reader.

Fundamentals pertaining to digital filtering, digital design considerations, and conventional methods of digital filter implementation are discussed.

Implementation of a simple first order recursive filter represented by the transfer function,

$$H(Z) = \frac{Z}{Z + K}$$

using Intel 8008 microprocessor has been simulated on IBM 370 system.

Studies pertaining to the filter's frequency response characteristics using the simulator have been reported here. Interfacing units such as A/D converter have been included in the simulation. Some of the results obtained via the simulator have been verified on the actual microcomputer system.

Currently available microprocessors which are characterized by their low cost, tremendous computing capabilities, and compactness appear to be one of the promising means for the implementation of digital filters.