To the Graduate Council:

I am submitting herewith a thesis written by James E. Phillips entitled "Distributed Arithmetic Implementation of a Reduced Roundoff Error Filter." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

James C. Hung, Major Professor

We have read this thesis
and recommend its acceptance:

J M Bailey

Donald W. Bouldin

Accepted for the Council:

The Graduate School

DISTRIBUTED ARITHMETIC IMPLEMENTATION OF A REDUCED ROUNDOFF ERROR FILTER

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

James E. Phillips

June 1984

ABSTRACT

The applicability of a hardware efficient realization of the distributed arithmetic approach for implementing digital filters was the concern of this study. To this end, 2 sixth order Butterworth filters were implemented. The first had a cutoff frequency that was 20% of the Nyquist frequency while the second had a cutoff frequency that was 2% of the Nyquist frequency. Each filter was implemented as 3 cascaded second order sections. The output roundoff error of the filters was reduced by choosing coefficients for each second order section so that the noise due to that section was minimum. Ordering of the cascaded sections was also chosen to reduce the output roundoff error. The procedures for defining filter coefficients and ordering the second order sections are presented and applied to the implemented filters.

The resulting filters were evaluated and shown to accurately reproduce the desired transfer function. The cost of the 20% implementation was compared to that of 3 alternative implementations. The distributed arithmetic approach proved cheaper than the other alternatives. In addition, the output roundoff error of the implemented filters was estimated and found to be 29 times less than the output roundoff error of a comparable minimum multiplier filter. Thus, the implemented filters were concluded to be a cost effective implementation of reduced roundoff error filters.

TABLE OF CONTENTS

CHAPTER PAGE

LIST OF TABLES

TABLE PAGE

LIST OF FIGURES

FIGURE PAGE

FIGURE PAGE

CHAPTER I

INTRODUCTION

Before the introduction of digital systems, signal processing was accomplished by analog techniques. The errors associated with representing system variables were negligible because system variables could take on a continuous range of values. In digital systems, however, system variables and coefficients are represented by finite length words. Therefore the variables and coefficients can assume only discrete values which lead to errors arising from coefficient quantization and computation roundoff as well as nonideal behavior such as overflow oscillations and limit cycles.

Coefficient quantization errors are errors in the filter output resulting from quantizing the filter's coefficients. Quantizing of the coefficients results in a finite number of realizable pole and zero locations in the vicinity of the desired locations. The realizable and desired pole and zero locations may not match; so, the realizable poles and zeros closest to those desired are generally chosen for the filter. Low coefficient sensitive structures help reduce the error due to coefficient quantization.

Roundoff errors occur as a result of quantizing stored filter variables. These errors are modelled by injecting, at the point of rounding, a uniformly distributed white noise signal having a zero mean and a variance, σ^2. This variance can be shown to be:

$$\sigma^2=\varepsilon^2/12 \tag{1.1}$$

where ε is the quantization step size. For normalized filters (all

variable values < 1) utilizing 2's complement arithmetic:

$$\varepsilon=1/2^{n-1} \tag{1.2}$$

where n is the number of bits in the storage word.

When the value of a filter variable becomes larger than representable, overflow occurs. Overflow presents large nonlinear transients to the filter. If the filter response to such a transient does not decay to zero, the output oscillates and the filter is said to support overflow oscillation. Filters that are otherwise stable can exhibit this phenomenon. Overflow oscillation characteristics of a filter are known to depend on the filter's pole-zero location and upon the filter's structure, but the relation between overflow oscillation and filter structure is not well known. Nevertheless, certain structures have been found that do not support the phenomenon of overflow oscillations. Thus, overflow oscillations can be prevented by selecting one of these structures or by severely scaling the filter to render the overflow condition impossible.

Limit cycles occur when the input is changed to a constant value and the output either sustains an erroneous constant value or oscillates about the predicted value. Limit cycle characteristics of a filter are influenced by pole-zero location, filter structure and roundoff technique (truncation or rounding).

Initial approaches for designing digital filters concentrated on reducing the number of multiplications required to realize the filter. The resulting minimum multiplier structures are known as canonical forms. These forms reduce the cost of the filter and might also be expected to reduce the roundoff error since fewer roundoffs are

required. This expectation, however, has been unfounded. For example the direct form state-space structure of a second order filter shown in Figure 1.1 is a practical minimum multiplier structure. This direct form structure, however, is notorious for having large roundoff errors.

A growing interest has emerged in recent years in determining the requirements necessary for a digital filter that would possess minimum roundoff error. Attention has been largely restricted to state-space structures.

Approaches for minimizing the roundoff for these structures are beginning to emerge and show capabilities of achieving the minimum possible roundoff error [1], [2]. These minimum roundoff error structures, however, require many more multiplies than canonical structures [3]. For an nth order filter, having a single input and a single output, the minimum roundoff error state-space structure requires $(n+1)^2$ multiplies [3] compared to 2n+1 multiplies for the canonical form [4]. Therefore minimum noise structures rapidly become more complex than the canonical form. An alternative approach which reduces rather than minimizes roundoff error in higher order filters is to cascade or parallel lower order sections that have been individually and independently optimized with respect to roundoff error [2], [3]. For example, a sixth order filter can be realized by cascading 3 second order sections for which the roundoff error has been individually minimized. Such filter structures, called sectionally optimum filters [3], provide a good compromise between minimum roundoff error filters and filter complexity.

One second order filter structure has been found that possesses

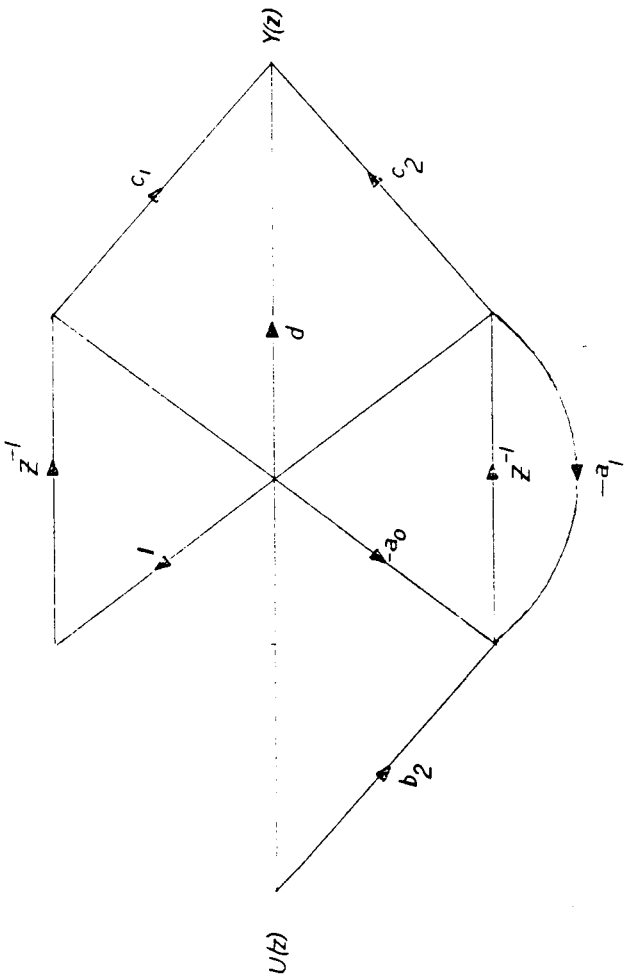


Figure 1.1. Direct form state-space structure of a second order filter.

low coefficient sensitivity, minimum roundoff error (for the given structure), and does not support overflow oscillation [3]. The structure for this second order filter is shown in Figure 1.2. It requires 9 multiplies which is 3 more than the number of multiplies required by the direct form of Figure 1.1. Techniques for determining structure parameters from the desired transfer function, H(z), have also been presented in the literature [3]. This filter structure is ideal for developing a sectionally optimum filter.

A proposed method for implementing digital filters utilizes the so called distributed arithmetic approach. In this approach precalculated partial products are placed in a look up table in memory, recalled at appropriate times, and summed in order to mimic the process of multiplication. In this manner expensive multiplier chips can be replaced with memory chips. Because memory requirements increase exponentially with filter order, the distributed arithmetic approach provides an effective implementation in terms of cost/speed trade-offs for lower order filters. Arjmand and Roberts [5] have found the distributed arithmetic approach for implementing a second order digital filter whose variables were represented by 12 bit words to be up to 5 times more efficient than implementations using multiplier chips.

The goal of this work was to establish the applicability of the distributed arithmetic approach to implementing reduced roundoff noise filters. To this end, 2 sixth order low pass Butterworth filters whose structures consist of 3 cascaded second order sections having the structure of Figure 1.2 were implemented and evaluated. One filter had a bandwidth of 20% of the Nyquist frequency while the other had a

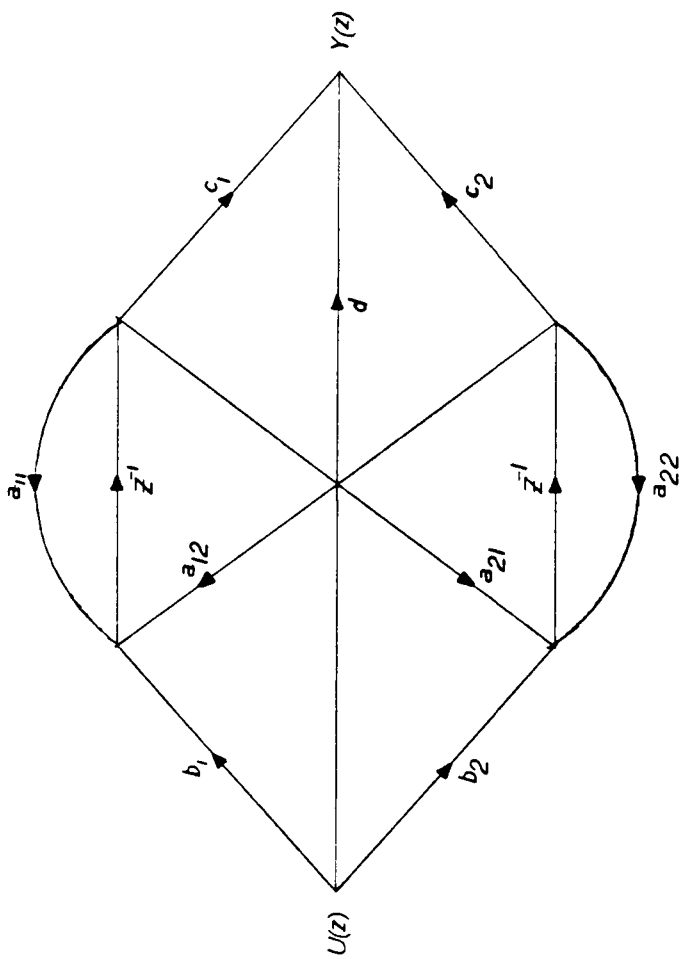


Figure 1.2. Minimum roundoff error, state-space, second order filter structure.

bandwidth of 2% of the Nyquist frequency. The filter's performance was evaluated by comparing the observed response with the predicted response. The operating filter was used to establish cost performance which was then compared to some selected alternative implementations.

CHAPTER II

DIGITAL FILTER STRUCTURES

Digital filters having low roundoff noise have recently been developed using state-space structures. State-space formulations of digital filters are known to be able to obtain the minimum achievable roundoff error when properly designed. These minimum roundoff error filters, however, rapidly become complex as the filter order increases. Alternatively, filters implemented by cascading individually and independently optimized, second order, state-space sections have been proposed. Such structures reduce the roundoff noise of the filter. In this chapter, the development of a minimum roundoff noise, second order, state-space section is reviewed. In addition, the direct form filter is derived to provide a benchmark for the comparison of filter complexity. The direct form was chosen for this comparison since it provides a minimum multiplier practical filter.

General Second Order Digital Filter

The general, minimum delay element, state variable description of a second order system with a single input and a single output is given as:

$$\mathbf{x}(k+1) = \mathbf{A}\mathbf{x}(k)+\mathbf{b}u(k) \tag{2.1}$$

$$y(k) = \mathbf{c}^{T}\mathbf{x}(k)+du(k) \tag{2.2}$$

where $\mathbf{x}(k)$ is a 2x1 vector representing the state variables at time k, $\mathbf{A}$ is a 2x2 system matrix, $\mathbf{b}$ is a 2x1 distribution vector, $\mathbf{c}$ is a 2x1 output vector, u(k) is the scaler input at time k, d is a scaler, and

y(k) is the scaler output at time k. The system is assumed to be time invariant; that is, **A**, **b**, **c** and d are independent of k. These equations are equivalent to the set of simultaneous difference equations:

$$x_1(k+1) = a_{11}x_1(k)+a_{12}x_2(k)+b_1u(k) \tag{2.3}$$

$$x_2(k+1) = a_{21}x_1(k)+a_{22}x_2(k)+b_2u(k) \tag{2.4}$$

$$y(k) = c_1x_1(k)+c_2x_2(k)+du(k). \tag{2.5}$$

Taking the Z transform of each equation gives:

$$zX_1(z) = a_{11}X_1(z)+a_{12}X_2(z)+b_1U(z) \tag{2.6}$$

$$zX_2(z) = a_{21}X_1(z)+a_{22}X_2(z)+b_2U(z) \tag{2.7}$$

$$Y(z) = c_1X_1(z)+c_2X_2(z)+dU(z). \tag{2.8}$$

A signal flow graph corresponding to this state variable representation is given in Figure 2.1.

The Direct Form Filter

The direct form filter is obtained by balance scaling the controllability canonical form. One controllability canonical form is derived by transforming the state variable equations, (2.1) and (2.2), by the transformation [6]:

$$\mathbf{x} = \mathbf{P}^{-1}\mathbf{w} \tag{2.9}$$

where **P** is the matrix,

$$\mathbf{P} = \begin{bmatrix} \mathbf{e}_n^T \\ \mathbf{e}_n^T\mathbf{A} \\ \cdot \\ \cdot \\ \cdot \\ \mathbf{e}_n^T\mathbf{A}^{n-1} \end{bmatrix}. \tag{2.10}$$

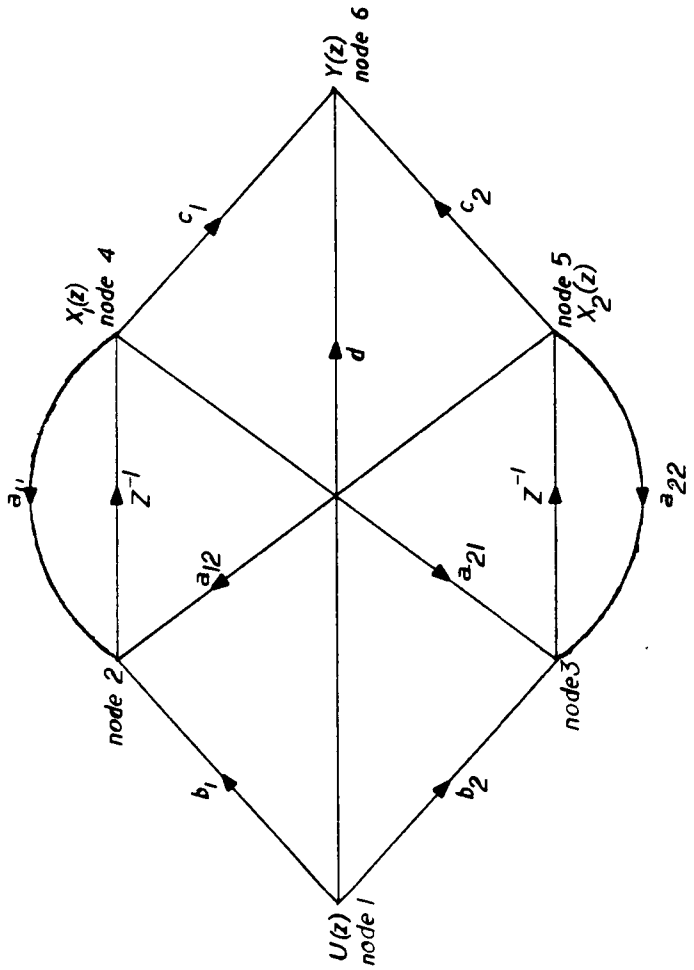


Figure 2.1. General, minimum delay element, second order state-space filter.

The vector $\mathbf{e}_n^T$ is obtained from the controllability matrix,

$$\mathbf{M} = [\mathbf{b} \quad \mathbf{Ab}]. \tag{2.11}$$

Assuming the system is completely controllable, then $\mathbf{M}$ is nonsingular and $\mathbf{M}^{-1}$ exists. For a second order filter $\mathbf{M}^{-1}$ may be represented as:

$$\mathbf{M}^{-1} = \begin{bmatrix} \mathbf{e}_1^T \\ \mathbf{e}_2^T \end{bmatrix} \tag{2.12}$$

where $\mathbf{e}_1$ and $\mathbf{e}_2$ are 2x1 column vectors. For this case $\mathbf{e}_n^T$ becomes $\mathbf{e}_2^T$ and $\mathbf{P}$ is:

$$\mathbf{P} = \begin{bmatrix} \mathbf{e}_2^T \\ \mathbf{e}_2^T\mathbf{A} \end{bmatrix}. \tag{2.13}$$

The equations resulting from substituting (2.9) into (2.1) and (2.2) are:

$$\mathbf{P}^{-1}\mathbf{w}(k+1) = \mathbf{A}\mathbf{P}^{-1}\mathbf{w}(k)+\mathbf{b}u(k) \tag{2.14}$$

$$y(k) = \mathbf{c}^T\mathbf{P}^{-1}\mathbf{w}(k)+du(k). \tag{2.15}$$

Equation (2.14) can also be written as:

$$\mathbf{w}(k+1) = \mathbf{P}\mathbf{A}\mathbf{P}^{-1}\mathbf{w}(k)+\mathbf{P}\mathbf{b}u(k). \tag{2.16}$$

Using (2.11), (2.12) and the definition of the inverse matrix,

$$\begin{bmatrix} \mathbf{e}_1^T \\ \mathbf{e}_2^T \end{bmatrix} [\mathbf{b} \ \mathbf{Ab}] = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}. \tag{2.17}$$

$\mathbf{Pb}$, therefore, can be seen from (2.17) to be:

$$\mathbf{Pb} = \begin{bmatrix} \mathbf{e}_2^T \\ \mathbf{e}_2^T\mathbf{A} \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \tag{2.18}$$

$\mathbf{PAP}^{-1}$ can also be deduced using (2.17). First $\mathbf{PAP}^{-1}$ can be written as $[\mathbf{PA}]\mathbf{P}^{-1}$. But $\mathbf{P}^{-1}$ may be represented as $[\mathbf{f}_1 \quad \mathbf{f}_2]$. Then,

$$\mathbf{PAP}^{-1} = \begin{bmatrix} \mathbf{e}_2^T\mathbf{A} \\ \mathbf{e}_2^T\mathbf{A}^2 \end{bmatrix} [\mathbf{f}_1 \quad \mathbf{f}_2]. \tag{2.19}$$

Using the definition of an inverse matrix:

$$\mathbf{P}\mathbf{P}^{-1} = \begin{bmatrix} \mathbf{e}_2^T \\ \mathbf{e}_2^T\mathbf{A} \end{bmatrix} [\mathbf{f}_1 \quad \mathbf{f}_2] = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{2.20}$$

from which $\mathbf{e}_2^T\mathbf{A}\mathbf{f}_1$ equals 0 and $\mathbf{e}_2^T\mathbf{A}\mathbf{f}_2$ equals 1.

Therefore,

$$\mathbf{P}\mathbf{A}\mathbf{P}^{-1} = \begin{bmatrix} 0 & 1 \\ \mathbf{e}_2^T\mathbf{A}^2\mathbf{f}_1 & \mathbf{e}_2^T\mathbf{A}^2\mathbf{f}_2 \end{bmatrix}. \tag{2.21}$$

Let

$$-a_0 = \mathbf{e}_2^T\mathbf{A}^2\mathbf{f}_1 \tag{2.22}$$

$$-a_1 = \mathbf{e}_2^T\mathbf{A}^2\mathbf{f}_2. \tag{2.23}$$

Then

$$\mathbf{P}\mathbf{A}\mathbf{P}^{-1} = \begin{bmatrix} 0 & 1 \\ -a_0 & -a_1 \end{bmatrix}. \tag{2.24}$$

Substituting (2.18) and (2.24) into (2.16) yields:

$$\begin{bmatrix} w_1(k+1) \\ w_2(k+1) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -a_0 & -a_1 \end{bmatrix} \begin{bmatrix} w_1(k) \\ w_2(k) \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(k). \tag{2.25}$$

The corresponding output equation is:

$$y(k) = \begin{bmatrix} c_1^* & c_2^* \end{bmatrix} \begin{bmatrix} w_1(k) \\ w_2(k) \end{bmatrix} + du(k) \tag{2.26}$$

where

$$[c_1^* \quad c_2^*] = \mathbf{c}^T\mathbf{P}^{-1}. \tag{2.27}$$

Multiplying the matrices gives the following simultaneous system of difference equations:

$$w_1(k+1) = w_2(k) \tag{2.28}$$

$$w_2(k+1) = -a_0 w_1(k) - a_1 w_2(k) + u(k) \tag{2.29}$$

$$y(k) = c_1^* w_1(k) + c_2^* w_2(k) + du(k). \tag{2.30}$$

Taking the Z transform of the equations yields:

$$zW_1(z) = W_2(z) \tag{2.31}$$

$$zW_2(z) = -a_0 W_1(z) - a_1 W_2(z) + U(z) \tag{2.32}$$

$$Y(z) = c_1^* W_1(z) + c_2^* W_2(z) + dU(z). \tag{2.33}$$

From these results the signal flow graph of Figure 2.2 is derived.

The direct form filter is obtained by scaling the controllability canonical form. For this discussion, balance scaling using the L_2 norm is used. The L_2 norm, denoted by $//h//_2$ is defined by:

$$//h//_2^2 = \sum_{i=1}^{\infty} /h_i/^2 \tag{2.34}$$

where h_i is the unit pulse response between the nodes of interest. For the controllability canonical structure, the filter needs to be scaled so that state variables at nodes 2 and 3 do not overflow. To scale node 2, the incoming branches to node 2 are divided by the square of the L_2 norm from the input to node 2, $//f_1//_2^2$, while the outgoing branches are multiplied by $//f_1//_2^2$ as shown in Figure 2.3. The procedure at node 3 is identical except that the square of the L_2 norm from the input to node 3, $//f_2//_2^2$, is used. The resulting filter is shown in Figure 2.4. From Figure 2.2, $//f_1//_2^2$ and $//f_2//_2^2$ are seen to be equal since nodes 2 and 3 are connected by a delay element and a unity gain branch. Therefore,

$$//f_1//_2^2 = //f_2//_2^2 = \Gamma. \tag{2.35}$$

Now denoting $1/\Gamma$ by $\bar{b}_2$, $c_1^*\Gamma$ by $\bar{c}_1^*$ and $c_2^*\Gamma$ by $\bar{c}_2^*$ gives the filter of Figure 2.5. This filter structure is the popular direct form. The direct form is notorious for possessing large roundoff noise. It

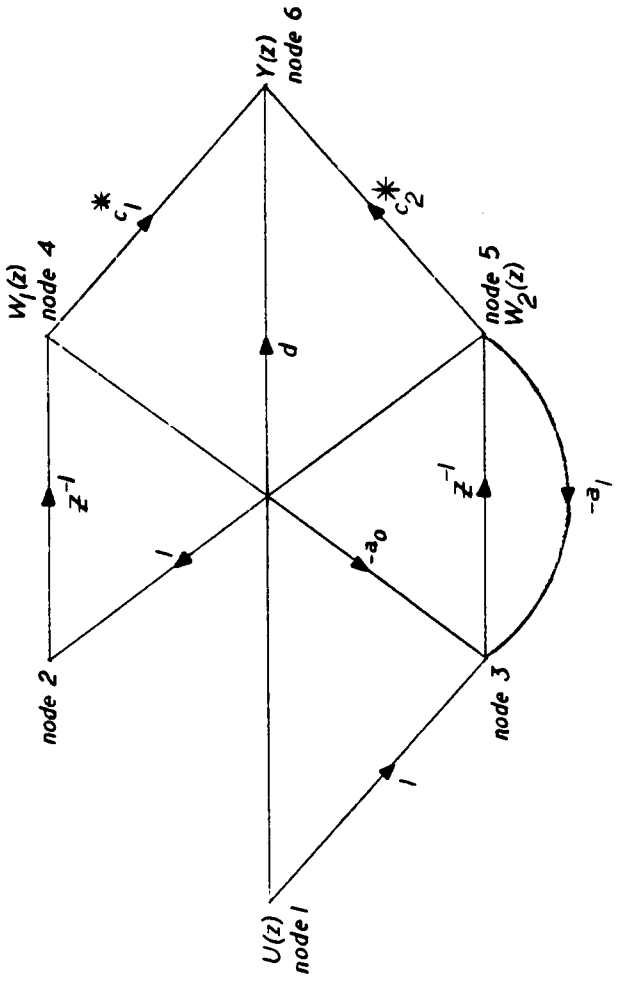


Figure 2.2. Controllability canonical form of the second order filter.

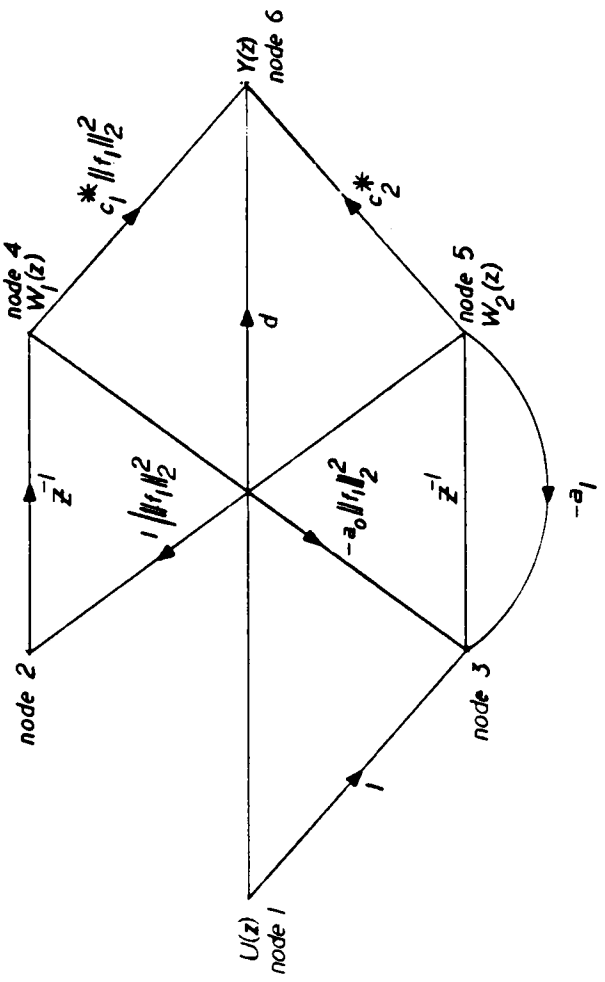


Figure 2.3. Controllability canonical form of the second order filter scaled at node 2.

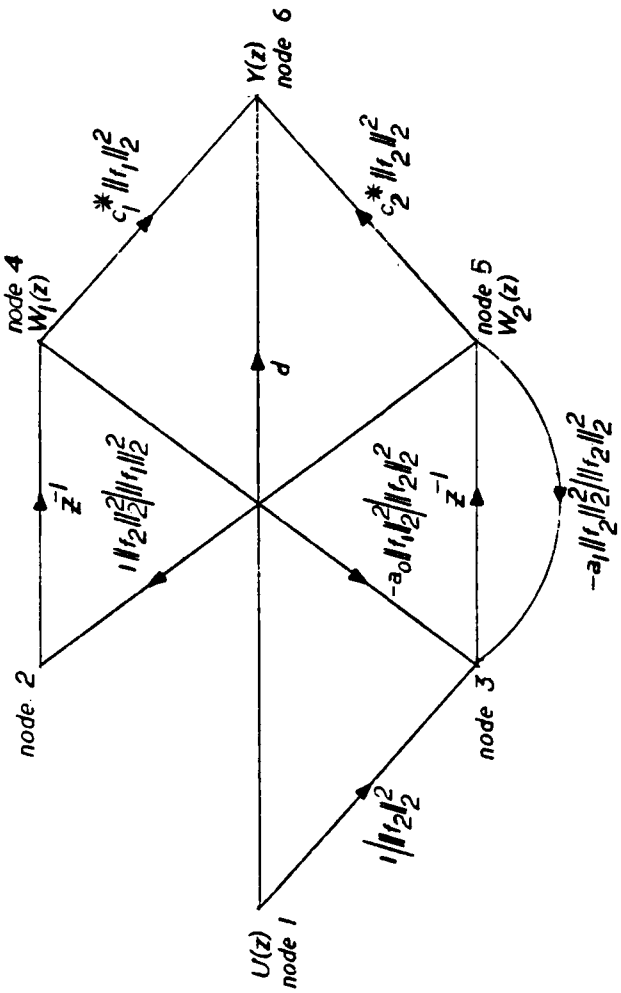


Figure 2.4. Controllability canonical form of the second order filter scaled at nodes 2 and 3.

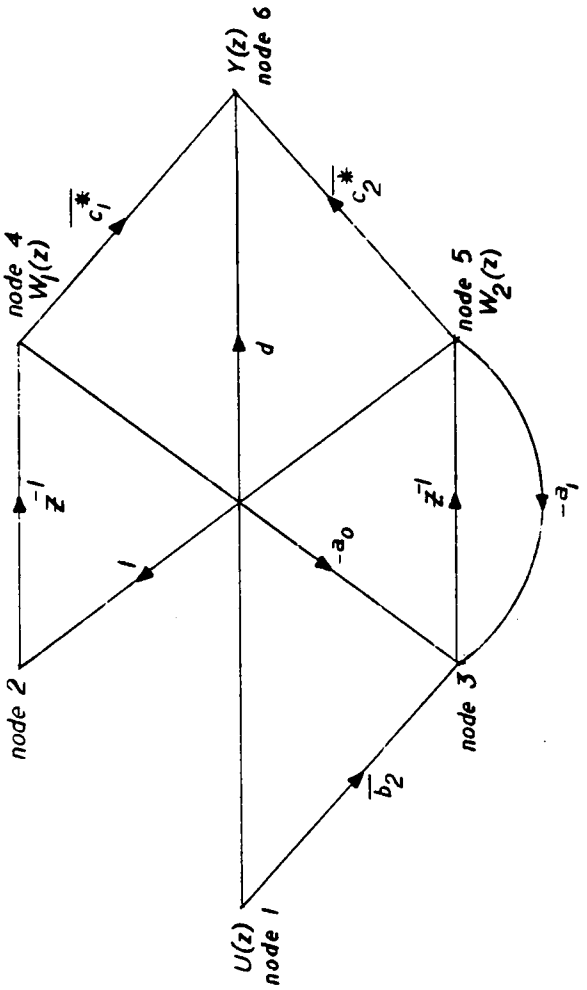


Figure 2.5. Direct form of the second order filter.

also supports overflow oscillations and possesses large coefficient sensitivity.

The number of multiplies required to implement the second order direct form is 6. This is 1 larger than the number of multiplies required by the controllability canonical form. Since scaling in general increases the number of multiplies required to implement a filter, the direct form filter structure of Figure 2.5 is a practical minimum multiplier structure.

Second Order Minimum Roundoff Error Filter

The determination of the minimum roundoff error, second order, state-space filter requires a derivation of the output noise expression for the filter followed by minimization of this roundoff noise. For this determination, the roundoff error is modelled as noise injected at the points at which rounding occurs as shown in Figure 2.6. In this model, the roundoff error is represented as uniformly distributed, zero mean white noise inputs having a variance of

$$\sigma^2 = \varepsilon^2 \nu / 12 \tag{2.36}$$

where ε is the quantization step size and ν is the number of quantizations at the error source per calculation. For a filter using 2's complement arithmetic, normalized to a full scale of unity and having m binary digits (bits) at each node,

$$\varepsilon = 1/2^{m-1}. \tag{2.37}$$

Also,

$$\nu = 1 \tag{2.38}$$

if quantization occurs following addition. Therefore,

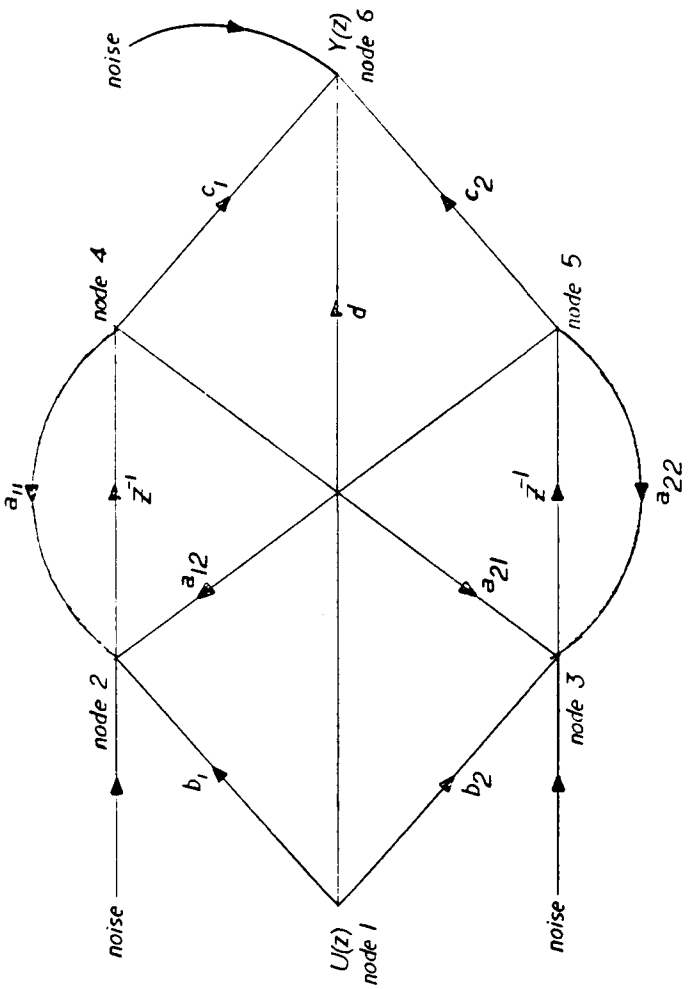


Figure 2.6. Model used for estimating the roundoff error of the general second order, state-space filter.

$$\sigma^2 = \varepsilon^2/12. \tag{2.39}$$

The above model is good if the overflow condition does not occur at a variable node. This condition implies that the filter has been scaled. In the following discussion, scaled filter parameters are denoted by primes.

Let $g_1'(k)$, and $g_2'(k)$ represent the unit pulse responses at time k from the state variables, x_1 and x_2, respectively to the output of a second order, state-space, balance scaled filter. Also, assume that each filter variable is represented by equal bit length words. Then for the noise model given above, the output noise can be shown to be:

$$\sigma_0^2 = \sigma_N^2[\sum_{k=0}^{\infty} g_1'^2(k) + \sum_{k=0}^{\infty} g_2'^2(k)+1] \tag{2.40}$$

where σ_0^2 is the output noise variance and σ_N^2 is the noise variance due to rounding at x_1, x_2 and the output. The first two terms in the right side of (2.40) are the square of the L_2 norms from nodes 2 and 3 respectively to node 6 of Figure 2.6. They are denoted by $//g_1'//_2^2$ and $//g_2'//_2^2$. Using this notation, (2.40) can be written as:

$$\sigma_0^2 = \sigma_N^2[//g_1'//_2^2 + //g_2'//_2^2 + 1]. \tag{2.41}$$

Since the variance provides a measure of power, (2.41) gives a measure of the output random noise power due to roundoff at the filter variables. This output noise power is to be minimized to obtain the minimum roundoff error filter. Therefore a state-space, second order, scaled filter where g_1' and g_2' minimize the roundoff noise at the output is desired.

Consider a filter that is balance scaled using the L_2 norm. Such a filter has unit pulse responses from x_1 and x_2 to the output of:

$$g_1' = g_1//f_1//_2^2 \tag{2.42}$$

and,

$$g_2' = g_2//f_2//_2^2 \tag{2.43}$$

where f_1 and f_2 are the unit pulse responses from the input to x_1 and x_2 respectively of the unscaled filter. Substituting (2.42) and (2.43) into (2.41) gives:

$$\sigma_0^2 = \sigma_N^2[//g_1//_2^2//f_1//_2^2+//g_2//_2^2//f_2//_2^2+1]. \tag{2.44}$$

Two matrices **K** and **W** can be defined for the state-space, second order structures. **K** is defined to have elements:

$$k_{ij} = \sum_{k=0}^{\infty} f_i(k)f_j(k) \tag{2.45}$$

while **W** has elements:

$$w_{ij} = \sum_{k=0}^{\infty} g_i(k)g_j(k) \tag{2.46}$$

where i and j may be 1 or 2. The diagonal elements of these matrices are the L_2 norms:

$$k_{ii} = //f_i//_2^2 \tag{2.47}$$

and,

$$w_{ii} = //g_i//_2^2. \tag{2.48}$$

Equation (2.44) can, therefore, be written as:

$$\sigma_0^2 = \sigma_N^2[k_{11}w_{11}+k_{22}w_{22}+1]. \tag{2.49}$$

The use of matrices **K** and **W** allow the minimum achievable output noise variance (power) to be determined. Mullis and Roberts [1] have shown that:

$$1/n \sum_{i=1}^{n} k_{ii}w_{ii} \geq [1/2 \sum_{i=1}^{n} \mu_i]^2 \tag{2.50}$$

where n is the order of **K** and **W**, and μ_i, i=1 to n, are the eigenvalues of the matrix **KW**. For the second order filter n=2 and (2.50) gives:

$$1/2[k_{11}w_{11}+k_{22}w_{22}] \geq [1/2(\mu_1+\mu_2)]^2 \tag{2.51}$$

or,

$$k_{11}w_{11}+k_{22}w_{22} \geq 2[1/2(\mu_1+\mu_2)]^2. \tag{2.52}$$

The minimum output error is obtained when equality occurs in (2.52). Therefore,

$$\sigma^2_{0min} = \sigma^2_N\{2[1/2(\mu_1+\mu_2)]^2+1\}. \tag{2.53}$$

Since the eigenvalues of the matrix **KW** are invariant under state variable coordinate transformation, the minimum output noise power can be obtained from **KW** even when the **K** and **W** matrices used to obtain **KW** do not produce equality in (2.52).

The minimum roundoff error filter can be realized by finding filter parameters that produce equality in (2.52) subject to the constraints that the filter be scaled and that the state variable description realize the desired transfer function. The existence of a minimum roundoff error filter has been guaranteed by Mullis and Roberts [1]. They have found that a transformation **T** exists such that (2.52) is satisfied with equality. In addition necessary and sufficient conditions for achievement of this equality have been found to be:

$$\mathbf{W}' = \mathbf{DK}'\mathbf{D} \tag{2.54}$$

where **D** is a diagonal matrix and,

$$k'_{ii}w'_{ii} = k'_{jj}w'_{jj} \tag{2.55}$$

for all i and j.

The necessary and sufficient conditions of (2.54) and (2.55) can

be translated into conditions for the system matrix **A**, distribution matrix **b**, and output matrix **c**. Recall that in the above discussion the filter was constrained to be scaled. Balanced scaling using the L_2 norm can be obtained by transforming the system coordinates with the transformation **Q** such that **Q** is diagonal and,

$$q_{ii} = \sqrt{k_{ii}}. \tag{2.56}$$

The **K** matrix following a coordinate transformation **T** can be shown to be:

$$\mathbf{K}_{trans} = \mathbf{T}^{-1}\mathbf{K}_{orig}(\mathbf{T}^{T})^{-1} \tag{2.57}$$

where $\mathbf{K}_{orig}$ is the **K** matrix before the transformation and $\mathbf{K}_{trans}$ is the **K** matrix after the transformation. Therefore,

$$\mathbf{K}' = \mathbf{Q}^{-1}\mathbf{K}(\mathbf{Q}^{T})^{-1}. \tag{2.58}$$

Equations (2.56) and (2.58) give:

$$k'_{ii} = 1 \tag{2.59}$$

for i=1,2. Substituting this result into (2.55) gives:

$$w'_{ii} = w'_{jj}. \tag{2.60}$$

These last two results can be used to rewrite (2.54) as:

$$\begin{bmatrix} a & b \\ c & a \end{bmatrix} = \begin{bmatrix} d_1 & 0 \\ 0 & d_2 \end{bmatrix}\begin{bmatrix} 1 & e \\ f & 1 \end{bmatrix}\begin{bmatrix} d_1 & 0 \\ 0 & d_2 \end{bmatrix} \tag{2.61}$$

which reduces to:

$$\begin{bmatrix} a & b \\ c & a \end{bmatrix} = \begin{bmatrix} d_1^2 & ed_1d_2 \\ fd_1d_2 & d_2^2 \end{bmatrix}. \tag{2.62}$$

Equation (2.62) is satisfied by choosing $d_1=d_2$. This gives:

$$\mathbf{D} = \rho\mathbf{I} \tag{2.63}$$

where ρ is a scaler and **I** represents the multiplicative identity matrix. Substituting (2.63) into (2.54) gives:

$$\mathbf{W}' = \rho^2\mathbf{K}'. \tag{2.64}$$

$\mathbf{K}'$ is a symmetrical second order matrix with equal diagonal elements. Such a matrix is invariant when multiplied on the left and right by $\mathbf{N}$ where:

$$\mathbf{N} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}. \tag{2.65}$$

Equation (2.64), therefore, can be written as:

$$\mathbf{W}' = \rho^2\mathbf{N}\mathbf{K}'\mathbf{N}. \tag{2.66}$$

The unit pulse responses f_1' and f_2' can be determined from the state variable equation. Let $\mathbf{f}'$ be defined as:

$$\mathbf{f}'(k) = \begin{bmatrix} f_1'(k) \\ f_2'(k) \end{bmatrix}. \tag{2.67}$$

Then substituting the unit pulse sequence {1,0,0,...} into (2.1) gives:

$$\mathbf{f}'(k) = \mathbf{A}'^{k-1}\mathbf{b}'. \tag{2.68}$$

In terms of $\mathbf{f}'(k)$, $\mathbf{K}'$ may be written as:

$$\mathbf{K}' = \sum_{k=0}^{\infty} \mathbf{f}'(k)\mathbf{f}'^{T}(k). \tag{2.69}$$

Substituting (2.68) into (2.69) produces:

$$\mathbf{K}' = \sum_{k=0}^{\infty} (\mathbf{A}'^{k-1}\mathbf{b}')(\mathbf{A}'^{k-1}\mathbf{b}')^{T}. \tag{2.70}$$

Similarly a $\mathbf{g}'^{T}(k)$ can be defined as:

$$\mathbf{g}'^{T}(k) = \begin{bmatrix} g_1'(k) \\ g_2'(k) \end{bmatrix}. \tag{2.71}$$

Applying a unit pulse sequence to the state variables of a state-space structure gives:

$$\mathbf{g}'(k) = \mathbf{c}'^{T}\mathbf{A}'^{k-1}. \tag{2.72}$$

In terms of $g'(k)$, W' is:

$$W' = \sum_{k=0}^{\infty} g'^{T}(k)g'(k). \tag{2.73}$$

Substituting (2.72) into (2.73) gives:

$$W' = \sum_{k=0}^{\infty} (c'^{T}A'^{k-1})^{T}(c'^{T}A'^{k-1}). \tag{2.74}$$

Equations (2.70) and (2.74) can now be substituted into (2.66) to give:

$$\sum_{k=0}^{\infty}\left[(A'^{k-1})^{T}c'\right]\left[c'^{T}A'^{k-1}\right] = \sum_{k=0}^{\infty}\left[NA'^{k-1}N^{-1}Nb'\rho\right]\left[\rho b'^{T}NN^{-1}(A'^{k-1})^{T}N\right]. \tag{2.75}$$

From (2.75), A' proves to be:

$$A'^{T} = NA'N^{-1}. \tag{2.76}$$

Using N, as defined by (2.65), and carrying out the matrix multiplication gives:

$$\begin{bmatrix} a'_{11} & a'_{21} \\ a'_{12} & a'_{22} \end{bmatrix} = \begin{bmatrix} a'_{22} & a'_{21} \\ a'_{12} & a'_{11} \end{bmatrix}. \tag{2.77}$$

By equality of matrices,

$$a'_{11} = a'_{22} \tag{2.78}$$

for a minimum roundoff error, L_2 norm balanced scaled, second order filter.

Also, from (2.75), c' can be shown as:

$$c' = \rho Nb'. \tag{2.79}$$

Since b' can be represented as:

$$b' = \begin{bmatrix} b'_1 \\ b'_2 \end{bmatrix}, \tag{2.80}$$

(2.79) becomes:

$$\begin{bmatrix} c_1' \\ c_2' \end{bmatrix} = \rho \begin{bmatrix} b_2' \\ b_1' \end{bmatrix}. \tag{2.81}$$

Again, using equality of matrices,

$$c_1'/b_2' = c_2'/b_1' \tag{2.82}$$

or,

$$c_1'b_1' = c_2'b_2'. \tag{2.83}$$

Therefore, for a second order, L_2 norm balanced scaled, minimum roundoff error filter, the necessary and sufficient conditions of (2.54) and (2.55) reduce to:

1) equivalent diagonal elements of the system matrix and

2) $\mathbf{b}'$ and $\mathbf{c}'$ related by (2.83).

The procedure for obtaining a second order, minimum roundoff error filter having L_2 norm scaling and a transfer function H(z) is as follows. First a state variable representation of the given transfer function must be obtained (ie. **A**, **b**, **c**, and d must be specified). The state variable representation must satisfy:

$$\gamma_1 = c_1b_1+c_2b_2 \tag{2.84}$$

$$\gamma_2 = c_1b_2a_{12}+c_2b_1a_{21}-c_1b_1a_{22}-c_2b_2a_{11} \tag{2.85}$$

$$\beta_2 = a_{11}a_{22}-a_{12}a_{21} \tag{2.86}$$

$$\beta_1 = -(a_{22}+a_{11}) \tag{2.87}$$

$$d = d \tag{2.88}$$

when the transfer function H(z) has the form:

$$H(z) = (\gamma_2 z^{-2}+\gamma_1 z^{-1})/(\beta_2 z^{-2}+\beta_1 z^{-1}+1)+d. \tag{2.89}$$

Once the state variable description is determined, a transformation is

desired such that the resultant filter is balance scaled using the L_2 norm, and (2.78) and (2.83) are satisfied. Two approaches for obtaining the suitable state variable equations have been presented in the literature [2], [3]. For one approach, scaling of the filter is inherent in the equations. These equations with inherent scaling are presented in Table 2.1.

The resulting second order filter has several desirable characteristics other than minimum roundoff error. Firstly the filter has low coefficient sensitivity which means that the error from coefficient quantization is reduced. Secondly, this filter does not support overflow oscillation. The 9 multiplies required to implement the filter, however, is larger than the 6 required for the direct form.

Table 2.1. Equations for Deriving Filter Parameters from a Given Transfer Function H(z).

DESCRIPTION	EQUATION	
Transfer Function	$H(z) = \dfrac{Q(z)}{P(z)} = \dfrac{q_2 z^2 + q_1 z + q_0}{z^2 - 2\alpha z + \alpha^2 + \beta^2}$	2.90
Poles of H(z)	$\lambda_* = \alpha + j\beta$ $\lambda^* = \alpha - j\beta$	2.91 2.92
δ	δ chosen for desired overflow probability	
γ	$\gamma = \lvert Q(\lambda^*)\rvert/\beta$	2.93
ϕ	$\phi = \mathrm{ARG}[-Q(\lambda^*)] = \mathrm{ARG}[Q(\lambda^*)] + \pi$	2.94
γ_1, γ_2	$\gamma_1 = (1/2)\left[\dfrac{1}{1-\lvert\lambda\rvert^2} - \left\lvert\dfrac{1}{1-\lambda^2}\right\rvert\right]$	2.95
	$\gamma_2 = (1/2)\left[\dfrac{1}{1-\lvert\lambda\rvert^2} + \left\lvert\dfrac{1}{1-\lambda^2}\right\rvert\right]$	2.96
θ	$\theta = -(1/2)\mathrm{ARG}(1-\lambda^2)$	2.97
d_1, d_2	$d_1^2 = \delta^2[\gamma_1\cos^2(\theta-\phi/2) + \gamma_2\sin^2(\theta-\phi/2)]$	2.98
	$d_2^2 = \delta^2[\gamma_1\sin^2(\theta-\phi/2) + \gamma_2\cos^2(\theta-\phi/2)]$	2.99
A	$\mathbf{A} = \begin{bmatrix} \alpha & -\beta(d_2/d_1) \\ \beta(d_1/d_2) & \alpha \end{bmatrix}$	2.100
b	$\mathbf{b} = \begin{bmatrix} \sin(\phi/2)/d_1 \\ \cos(\phi/2)/d_2 \end{bmatrix}$	2.101
$\mathbf{c}^T$	$\mathbf{c}^T = \gamma[d_1\cos(\phi/2) \quad d_2\sin(\phi/2)]$	2.102
d	$d = d$	2.103

CHAPTER III

REDUCED ROUNDOFF NOISE SIXTH ORDER LOW PASS BUTTERWORTH FILTER

In the previous chapter the technique for determining coefficients to implement second order minimum roundoff noise filters realizing a given transfer function were presented. In this chapter, this technique is applied to the transfer functions of 2 sixth order Butterworth filters. The procedure consists of factoring each filter into 3 second order sections by grouping pairs of complex conjugate poles. To these second order sections, the technique of Table 2.1 is applied to determine the desired filter coefficients. The resulting second order sections can then be cascaded to provide the sixth order filter. The resulting filter is sectionally optimized and provides reduced roundoff noise.

Sixth Order Butterworth Filter Transfer Function

The transfer function of a sixth order Butterworth filter having a cutoff frequency of ω_0 is:

$$H(s/\omega_0) = 1/(ABC) \tag{3.1}$$

where,

$$A = [s/\omega_0 - e^{j(0.58)\pi}][s/\omega_0 - e^{j(1.42)\pi}] \tag{3.2}$$

$$B = [s/\omega_0 - e^{j(0.75)\pi}][s/\omega_0 - e^{j(1.25)\pi}] \tag{3.3}$$

$$C = [s/\omega_0 - e^{j(0.92)\pi}][s/\omega_0 - e^{j(1.08)\pi}]. \tag{3.4}$$

This analog filter can be converted to a digital filter by using the Bilinear transformation:

$$s = [2/T][(z-1)/(z+1)]. \tag{3.5}$$

T represents the sampling period and is related to the sampling frequency by:

$$T = 1/f_s. \tag{3.6}$$

Since the Bilinear transformation nonlinearly maps frequencies in the s-plane to frequencies in the z-plane, the cutoff frequency of the filter must be prewarped so that the desired cutoff frequency is realized. This prewarping is accomplished by:

$$a = (2/T)\tan(\omega_0 T/2). \tag{3.7}$$

For a Butterworth filter in which ω_0 is 2% of the Nyquist frequency,

$$\omega_0 = (0.02\omega_s)/2. \tag{3.8}$$

The prewarped frequency, a, becomes:

$$a = 6.29\text{x}10^{-2} f_s. \tag{3.9}$$

Let ζ be defined by:

$$\zeta = s/a. \tag{3.10}$$

Then ζ is easily shown to be the following function of z:

$$\zeta(z) = 31.8(z-1)/(z+1). \tag{3.11}$$

The Bilinear representation of equation (3.1) is found by substituting $\zeta(z)$ for s/ω_0 in that equation. With this substitution,

$$H(z) = 1/(DEF) \tag{3.12}$$

where,

$$D = [\zeta(z)-e^{j(0.58)\pi}][\zeta(z)-e^{j(1.42)\pi}] \tag{3.13}$$

$$E = [\zeta(z)-e^{j(0.75)\pi}][\zeta(z)-e^{j(1.25)\pi}] \tag{3.14}$$

$$F = [\zeta(z)-e^{j(0.92)\pi}][\zeta(z)-e^{j(1.08)\pi}]. \tag{3.15}$$

The pulse transfer function H(z) is easily factored into second order sections with real coefficients by grouping complex conjugate pairs. Therefore:

$$H_1(z) = 1/D \tag{3.16}$$

$$H_2(z) = 1/E \tag{3.17}$$

$$H_3(z) = 1/F. \tag{3.18}$$

Equation (3.11) can be used to write equations (3.16)-(3.18) as:

$$H_1(z)_{2\%}=[1/(1.03\times10^{3})][(z+1)^2/(z^2-1.964z+.9680)] \tag{3.19}$$

$$H_2(z)_{2\%}=[1/(1.06\times10^{3})][(z+1)^2/(z^2-1.911z+.9150)] \tag{3.20}$$

$$H_3(z)_{2\%}=[1/(1.08\times10^{3})][(z+1)^2/(z^2-1.882z+.8856)]. \tag{3.21}$$

Applying the identical procedure to a 20% cutoff frequency filter gives:

$$H_1(z)_{20\%}=[1/(12.07)][(z+1)^2/(z^2-1.404z+.7359)] \tag{3.22}$$

$$H_2(z)_{20\%}=[1/(14.82)][(z+1)^2/(z^2-1.143z+.4128)] \tag{3.23}$$

$$H_3(z)_{20\%}=[1/(16.42)][(z+1)^2/(z^2-1.032z+.2757)]. \tag{3.24}$$

Filter Coefficients for a Sixth Order Low Pass Butterworth Filter

The equations of Table 2.1 (pg. 28) can be applied to the 2% and 20% bandwidth Butterworth filter pulse transfer functions to determine the filter coefficients for the corresponding minimum noise filters. The application of these equations to a single second order section of the 2% filter is presented in this section. The method is easily extended to the other second order sections. The results for both filters are then given.

Consider (3.19) assuming the gain term to be unity; that is, consider:

$$H_1(z) = (z^2+2z+1)/(z^2-1.964z+.9680). \tag{3.25}$$

Comparing this with the form of the transfer function in Table 2.1 the following relationships can be established:

$$2\alpha = 1.964 \tag{3.26}$$

or,

$$\alpha = .9821 \tag{3.27}$$

and,

$$\alpha^2+\beta^2 = .9680 \tag{3.28}$$

from which β is expressed as:

$$\beta = 5.968\text{x}10^{-2}. \tag{3.29}$$

Therefore, the complex conjugate poles are:

$$\lambda = z_1 = .9821+j(.05968) \tag{3.30}$$

$$\lambda^* = z_2 = .9821-j(.05968). \tag{3.31}$$

From (2.93) of Table 2.1 (pg. 28):

$$\gamma = |Q(\lambda^*)|/\beta \tag{3.32}$$

where,

$$Q(z) = (z+1)^2. \tag{3.33}$$

Equations (3.31) - (3.33) indicate that:

$$\gamma = 65.89. \tag{3.34}$$

The next step is to determine ϕ, the argument of $-Q(\lambda^*)$. But,

$$-Q(\lambda^*) = -\{[.9821-j(.05968)]+1\}^2 \tag{3.35}$$

or,

$$-Q(\lambda^*) = -3.925+.2366j. \tag{3.36}$$

ϕ is the argument of (3.36); therefore,

$$\phi = 3.081 \text{ radians}. \tag{3.37}$$

Next, γ_1, γ_2 and θ are determined and used to find d_1 and d_2. To determine γ_1 and γ_2, the terms $1/[1-|\lambda|^2]$ and $|1/(1-\lambda^2)|$ must be determined. But λ is given in equation (3.30); therefore,

$$1/(1-|\lambda|^2) = 1/(1-|.9821+.05968j|^2) \tag{3.38}$$

$$1/(1-|\lambda|^2) = 1/\{1-[(.9821)^2+(.05968)^2]\} \quad (3.39)$$

$$1/(1-|\lambda|^2) = 31.27 \quad (3.40)$$

and,

$$|1/(1-\lambda^2)| = 1/|1-(.9821+.05968j)^2| \quad (3.41)$$

$$|1/(1-\lambda^2)| = 1/|1-.9821^2-2(.9821)(.05968)j+(.05968)^2| \quad (3.42)$$

$$|1/(1-\lambda^2)| = 1/|3.911x10^{-2}-.1172j| \quad (3.43)$$

$$|1/(1-\lambda^2)| = 8.092. \quad (3.44)$$

From these results and (2.95) and (2.96) of Table 2.1 (pg. 28),

$$\gamma_1 = (1/2)(31.27-8.092) = 11.59 \quad (3.45)$$

and,

$$\gamma_2 = (1/2)(31.27+8.092) = 19.68. \quad (3.46)$$

θ can also be easily found by:

$$\theta = -(1/2)\arg(1-\lambda^2) \quad (3.47)$$

$$\theta = -(1/2)\arg(3.911x10^{-2}-.1172j) \quad (3.48)$$

$$\theta = .6244 \text{ radians.} \quad (3.49)$$

For a scaling parameter, δ, equal 5, then d_1^2 and d_2^2 are determined from (2.98) and (2.99) of Table 2.1 and values for γ_1, γ_2, and ϕ to be:

$$d_1^2 = 417.0 \quad (3.50)$$

and,

$$d_2^2 = 364.7. \quad (3.51)$$

All the parameters required for calculating the filter coefficients of a second order minimum roundoff error filter realizing $H_1(z)$ have been determined. The procedure for calculating these coefficients requires the appropriate substitution of values derived above into equations of Table 2.1 for the system matrix **A**, the distribution

vector **b**, and the output vector **c**. Applying this procedure gives:

$$\mathbf{A}'_{1_{2\%}} = \begin{bmatrix} 0.9821 & -0.05581 \\ 0.06382 & 0.9821 \end{bmatrix} \tag{3.52}$$

$$\mathbf{b}'_{1_{2\%}} = \begin{bmatrix} 0.04895 \\ 0.001576 \end{bmatrix} \tag{3.53}$$

$$\mathbf{c}'^{T}_{1_{2\%}} = [0.03843 \quad 1.193]. \tag{3.54}$$

The above procedure was applied to $H_2(z)$ and $H_3(z)$ to give:

$$\mathbf{A}'_{2_{2\%}} = \begin{bmatrix} 0.9556 & -0.02454 \\ 0.07363 & 0.9556 \end{bmatrix} \tag{3.55}$$

$$\mathbf{b}'_{2_{2\%}} = \begin{bmatrix} 0.06732 \\ 0.002534 \end{bmatrix} \tag{3.56}$$

$$\mathbf{c}'^{T}_{2_{2\%}} = [0.02757 \quad 0.7321] \tag{3.57}$$

and,

$$\mathbf{A}'_{3_{2\%}} = \begin{bmatrix} 0.9410 & -0.002853 \\ 0.08231 & 0.9410 \end{bmatrix} \tag{3.58}$$

$$\mathbf{b}'_{3_{2\%}} = \begin{bmatrix} 0.06880 \\ 0.002917 \end{bmatrix} \tag{3.59}$$

$$\mathbf{c}'^{T}_{3_{2\%}} = [0.02677 \quad 0.6314]. \tag{3.60}$$

The procedure was also applied to the 20% Butterworth filter with δ equal to 1. The resulting state variable coefficients were:

$$\mathbf{A}'_{1_{20\%}} = \begin{bmatrix} 0.7022 & -0.4608 \\ 0.5270 & 0.7022 \end{bmatrix} \tag{3.61}$$

$$\mathbf{b}'_{1_{20\%}} = \begin{bmatrix} 0.6758 \\ 0.2092 \end{bmatrix} \tag{3.62}$$

$$\mathbf{c}'^{T}_{1_{20\%}} = [0.1759 \quad 0.5681] \tag{3.63}$$

$$\mathbf{A}'_{2_{20\%}} = \begin{bmatrix} 0.5715 & -0.1695 \\ 0.5085 & 0.5715 \end{bmatrix} \tag{3.64}$$

$$\mathbf{b}'_{2_{20\%}} = \begin{bmatrix} 0.8698 \\ 0.2815 \end{bmatrix} \tag{3.65}$$

$$\mathbf{c}'^{T}_{2_{20\%}} = [0.1262 \quad 0.3899] \tag{3.66}$$

and,

$$\mathbf{A}'_{3_{20\%}} = \begin{bmatrix} 0.5160 & -0.01806 \\ 0.5213 & 0.5160 \end{bmatrix} \tag{3.67}$$

$$\mathbf{b}'_{3_{20\%}} = \begin{bmatrix} 0.8639 \\ 0.2970 \end{bmatrix} \tag{3.68}$$

$$\mathbf{c}'^{T}_{3_{20\%}} = [0.1225 \quad 0.3564]. \tag{3.69}$$

The sixth order low pass Butterworth filters described above are shown in Figure 3.1 for a cutoff frequency of 2% of the Nyquist frequency and Figure 3.2 for a cutoff frequency of 20% of the Nyquist frequency.

Second order sections cascaded to produce the sixth order filters of Figures 3.1 and 3.2 were arranged to reduce the output roundoff noise. Reduction of the output roundoff noise can be accomplished by reducing the output noise variance. This output noise variance can be

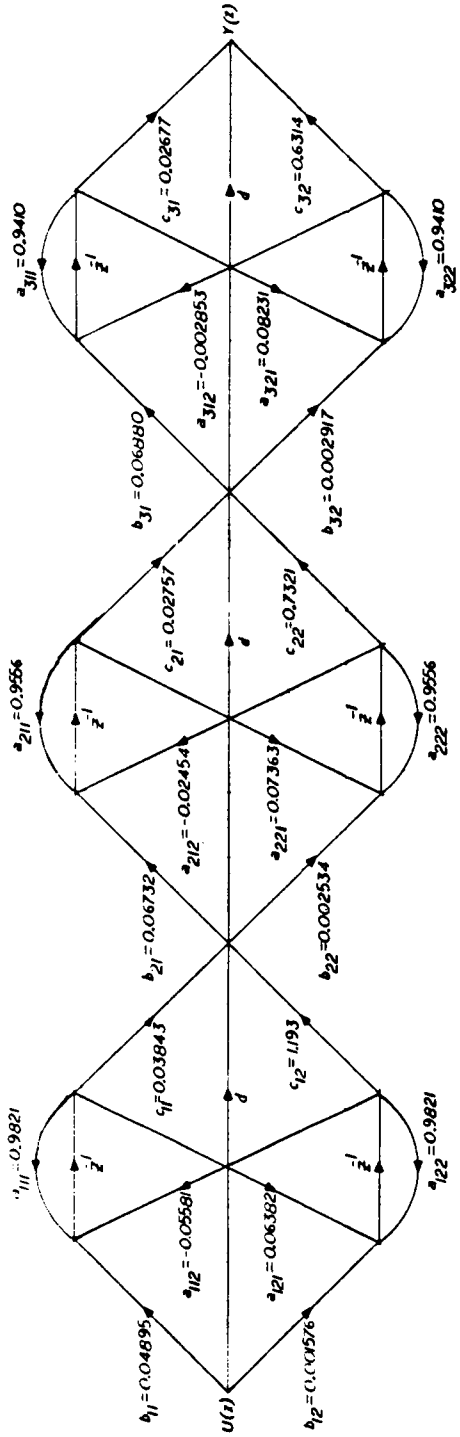


Figure 3.1. Reduced roundoff error filter for a sixth order low pass Butterworth filter with a cutoff frequency of 2% of the Nyquist frequency.

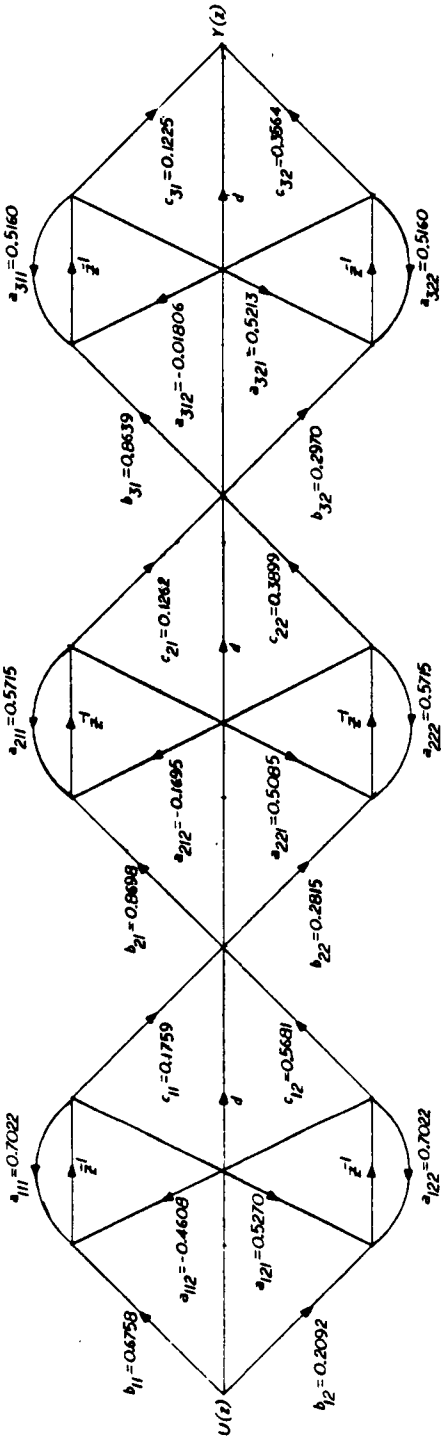


Figure 3.2. Reduced roundoff error filter for a sixth order low pass Butterworth filter with a cutoff frequency of 20% of the Nyquist frequency.

estimated, assuming the output roundoff noise is white, by applying the equation:

$$\sigma_y^2 \cong \sigma_1^2 //H_2(z)H_3(z)//_2^2 + \sigma_2^2 //H_3(z)//_2^2 + \sigma_3^2 \tag{3.70}$$

where σ_1, σ_2 and σ_3 are the noise variance of each isolated section. The output noise variance was reduced by choosing the sections such that $//H_2(z)H_3(z)//_2^2$ and $//H_3(z)//_2^2$ were minimized. $//H_2(z)H_3(z)//_2^2$ and $//H_3(z)//_2^2$ were calculated by applying Parseval's Theorem to all possible permutations for ordering the second order sections. The results of these calculations are given in Table 3.1 for the 2% filter and Table 3.2 for the 20% filter. From these results, the selected ordering of the cascaded sections for the two filters can be seen to reduce the output roundoff noise of each sixth order filter.

Table 3.1. L_2 Norms for all Combinations of the Second Order Sections for the 2% Filter.

Order of Filter Section	$//H_2(z)H_3(z)//_2^2$	$//H_3(z)//_2^2$
$H_{1_{2\%}} H_{2_{2\%}} H_{3_{2\%}}$	0.0135	0.0169
$H_{1_{2\%}} H_{3_{2\%}} H_{2_{2\%}}$	0.0135	0.0224
$H_{2_{2\%}} H_{1_{2\%}} H_{3_{2\%}}$	0.0255	0.0169
$H_{2_{2\%}} H_{3_{2\%}} H_{1_{2\%}}$	0.0255	0.0580
$H_{3_{2\%}} H_{1_{2\%}} H_{2_{2\%}}$	0.0371	0.0224
$H_{3_{2\%}} H_{2_{2\%}} H_{1_{2\%}}$	0.0371	0.0580

Table 3.2. L_2 Norms for all Combinations of the Second Order Sections for the 20% Filter.

Order of Filter Section	$//H_2(z)H_3(z)//_2^2$	$//H_3(z)//_2^2$
$H_{1_{20\%}} H_{2_{20\%}} H_{3_{20\%}}$	0.1820	0.2076
$H_{1_{20\%}} H_{3_{20\%}} H_{2_{20\%}}$	0.1820	0.2296
$H_{2_{20\%}} H_{1_{20\%}} H_{3_{20\%}}$	0.2357	0.2076
$H_{2_{20\%}} H_{3_{20\%}} H_{1_{20\%}}$	0.2357	0.4087
$H_{3_{20\%}} H_{1_{20\%}} H_{2_{20\%}}$	0.2860	0.2296
$H_{3_{20\%}} H_{2_{20\%}} H_{1_{20\%}}$	0.2860	0.4087

CHAPTER IV

A DISTRIBUTED ARITHMETIC APPROACH FOR IMPLEMENTING THE SIXTH ORDER BUTTERWORTH FILTER

A sixth order, reduced roundoff error, low pass Butterworth filter was presented in the previous chapter. This filter was to be implemented using the so called distributive arithmetic approach. In this chapter, the basis of the distributive arithmetic approach is developed and applied to a typical second order section of the filters given in Figures 3.1 and 3.2. The hardware required to implement the distributive arithmetic for the sixth order Butterworth filter is developed in general terms. The distributive arithmetic approach is shown to require the storage of specific functional values in memory and the control of data flow and digital chip functions by a control unit. Determination of the function values to be stored in memory, however, are deferred to a later chapter as is the design of the control unit.

Distributed Arithmetic

The signal flow graph of the digital filters to be implemented are shown in Figures 3.1 and 3.2. For the present discussion consider a single second order section which is shown in its general form in Figure 2.1 (pg. 10). Nodes 2, 3 and 6 of this figure represent variables which must be calculated. A typical node to be calculated in a state variable representation is shown in Figure 4.1. In this figure n_s represents the node to be calculated. It is fed from 1 nodes each having a transmittance a_i, $\{i=0,1,2,\ldots 1\}$. The variable at n_s is:

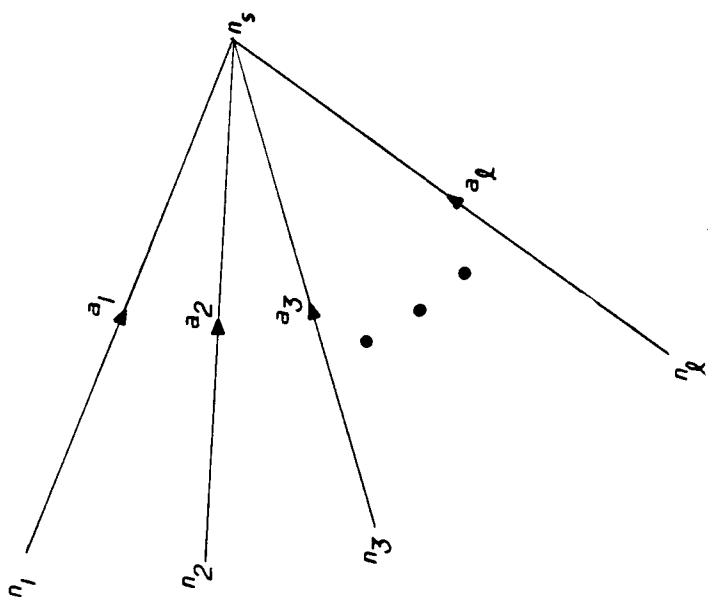


Figure 4.1. Typical state variable filter node.

$$v_s(k) = \sum_{i=1}^{1} a_i v_i(k) \tag{4.1}$$

where $v_s(k)$ represents the value of node n_s and $v_i(k)$ represents the value of node i at time k. Consider that each variable has a word length of m bits and is normalized so that the value of each variable is less than 1. Also assume that the variables are represented by 2's complement representation. Then,

$$v_s(k) = -v_{s0}2^0+v_{s(-1)}2^{-1}+\ldots+v_{s(-(m-1))}2^{-(m-1)} \tag{4.2}$$

$$v_s(k) = \sum_{j=1}^{m-1} v_{s(-j)}2^{-j}-v_{s0}2^0 \tag{4.3}$$

and

$$v_i(k) = -v_{i0}2^0+v_{i(-1)}2^{-1}+\ldots+v_{i(-(m-1))}2^{-(m-1)} \tag{4.4}$$

$$v_i(k) = \sum_{j=1}^{m-1} v_{i(-j)}2^{-j}-v_{i0}2^0. \tag{4.5}$$

Substituting equation (4.5) into (4.1) yields:

$$v_s(k) = \sum_{i=1}^{1} a_i\left(\sum_{j=1}^{m-1} v_{i(-j)}2^{-j}-v_{i0}2^0\right) \tag{4.6}$$

which can be written as:

$$v_s(k) = \sum_{j=1}^{m-1}\left(\sum_{i=1}^{1} a_i v_{i(-j)}\right)2^{-j}-\left(\sum_{i=1}^{1} a_i v_{i0}\right)2^0. \tag{4.7}$$

A function F(j) can be defined as:

$$F(j) = \begin{cases} \sum_{i=1}^{1} a_i v_{i(-j)} & j = 1,2,\ldots,m-1 \\ \sum_{i=1}^{1} a_i v_{i0} & j = 0 \end{cases} \tag{4.8}$$

Equation (4.7) then becomes:

$$v_s(k) = \sum_{j=1}^{m-1} F(j)2^{-j} - F(0). \tag{4.9}$$

Consider a single term $F(j)2^{-j}$ used in the calculation of $v_s(k)$. Since multiplication of F(j) by 2^{-j} can be accomplished by right shifting F(j) j times, acquiring F(j) is of particular interest. In the second order filter to be implemented each node to be calculated is fed by 3 branches so that 1 equals 3. Therefore:

$$F(j) = a_1 v_{1(-j)} + a_2 v_{2(-j)} + a_3 v_{3(-j)}. \tag{4.10}$$

While $v_{1(-j)}$, $v_{2(-j)}$ and $v_{3(-j)}$ may either be 0 or 1, a_1, a_2 and a_3 are constants. Therefore, F(j) may have 2^3 or 8 different values. Assume that the value of F(j) for $v_{1(-j)}$, $v_{2(-j)}$ and $v_{3(-j)}$ all zero is calculated and stored in a memory location addressed 000. Similarly assume the value of F(j) for $v_{1(-j)}$, $v_{2(-j)}$ both zero and $v_{3(-j)}$ equal 1 is calculated and stored in location 001 and so on for all permutations of $v_{1(-j)}$, $v_{2(-j)}$ and $v_{3(-j)}$. The appropriate value of F(j) could then be accessed by the circuit of Figure 4.2.

Each function value F(j) accessed by Figure 4.2 must be combined according to equation (4.9) to determine $v_s(k)$. Assume that the speed requirements of the filter allow the additions and subtraction of equation (4.9) to be accomplished serially. The calculation of $v_s(k)$ can be accomplished by the hardware of Figure 4.3. A total of m clock pulses are required to determine $v_s(k)$. Initially latch B is reset and F(j) for j equal m-1 is fed to side A. The add/$\overline{\text{sub}}$ line is high so that F(m-1) appears at the output. The output is fed back to the inputs to latch B and hardwired in such a manner that the result is right shifted 1 digit with the most significant bit (MSB) duplicated

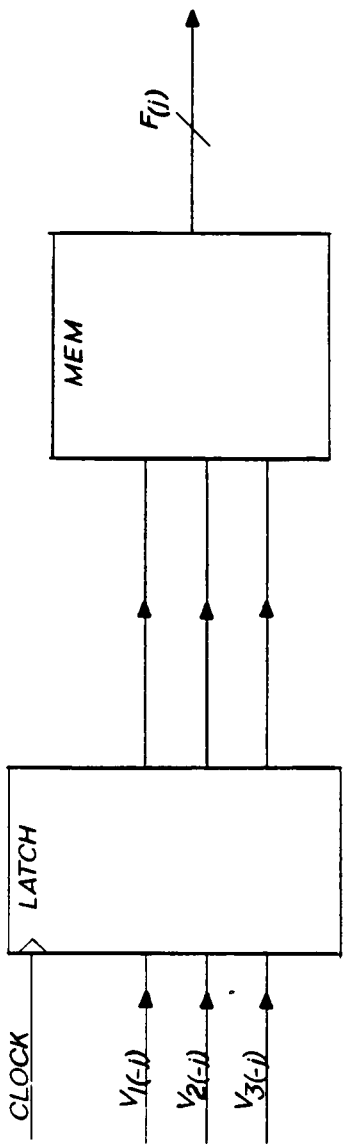


Figure 4.2. Circuit to access partial products F(j) used in the calculation of a filter node.

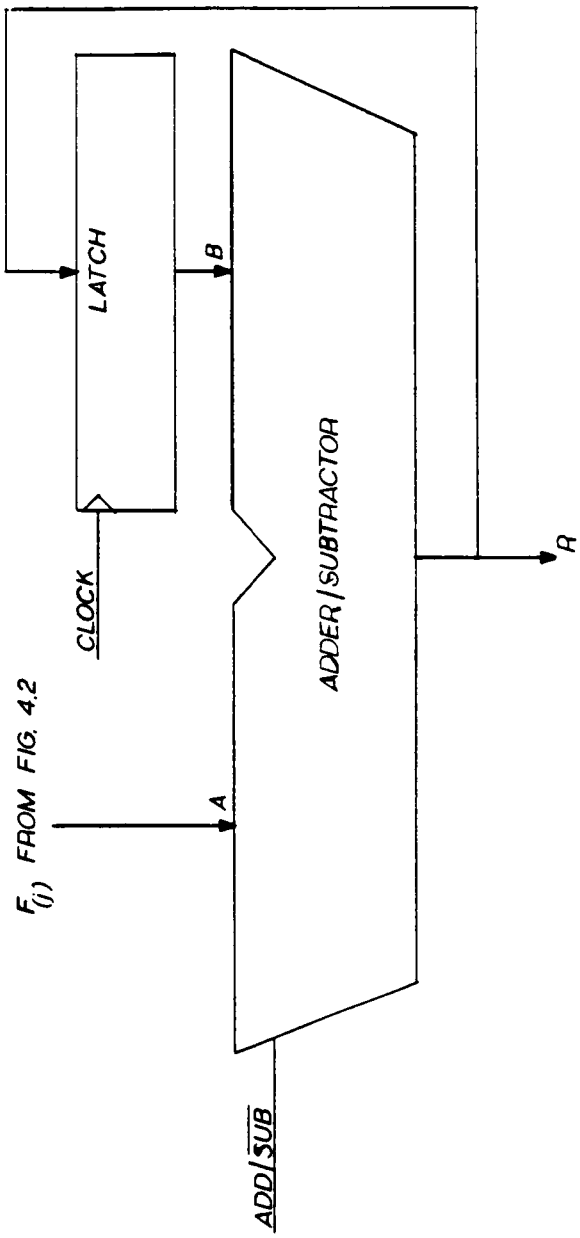


Figure 4.3. Hardware for combining partial products F(j) used in the calculation of a filter node.

and the least significant bit (LSB) lost as shown in Figure 4.4. On the next clock pulse the output is latched into B, and F(j) for j equal m-2 is fed to A. The result is:

$$R = F(m-2)+F(m-1)2^{-1}. \tag{4.11}$$

This process is continued through j equal 1 at which point

$$R = F(1)+F(2)2^{-1}+\ldots+F(m-2)2^{-(m-3)}+F(m-1)2^{-(m-2)}. \tag{4.12}$$

On the final clock, pulse A becomes F(0), B becomes

$$B = F(1)2^{-1}+F(2)2^{-2}+\ldots+F(m-2)2^{-(m-2)}+F(m-1)2^{-(m-1)} \tag{4.13}$$

and the add/$\overline{\text{sub}}$ line is low. Thus the result following the clock pulse is:

$$R = -F(0)+F(1)2^{-1}+F(2)2^{-2}+\ldots +F(m-2)2^{-(m-2)}+F(m-1)2^{-(m-1)} \tag{4.14}$$

$$R = \sum_{j=1}^{m-1} F(j)2^{-j}-F(0). \tag{4.15}$$

$$R = v_s(k). \tag{4.16}$$

The result is the value at the desired node n_s.

Distributed arithmetic provides a mechanism for efficiently calculating the value of a variable which is the sum of many terms each having an implied multiplication. It consists of distributing the computation over the bits of variables in the computation, defining a function which has a finite number of possible values, using the bits of the variables to determine the proper function value and right shifting and adding the function values to obtain the result. The hardware required is summarized in Figure 4.5.

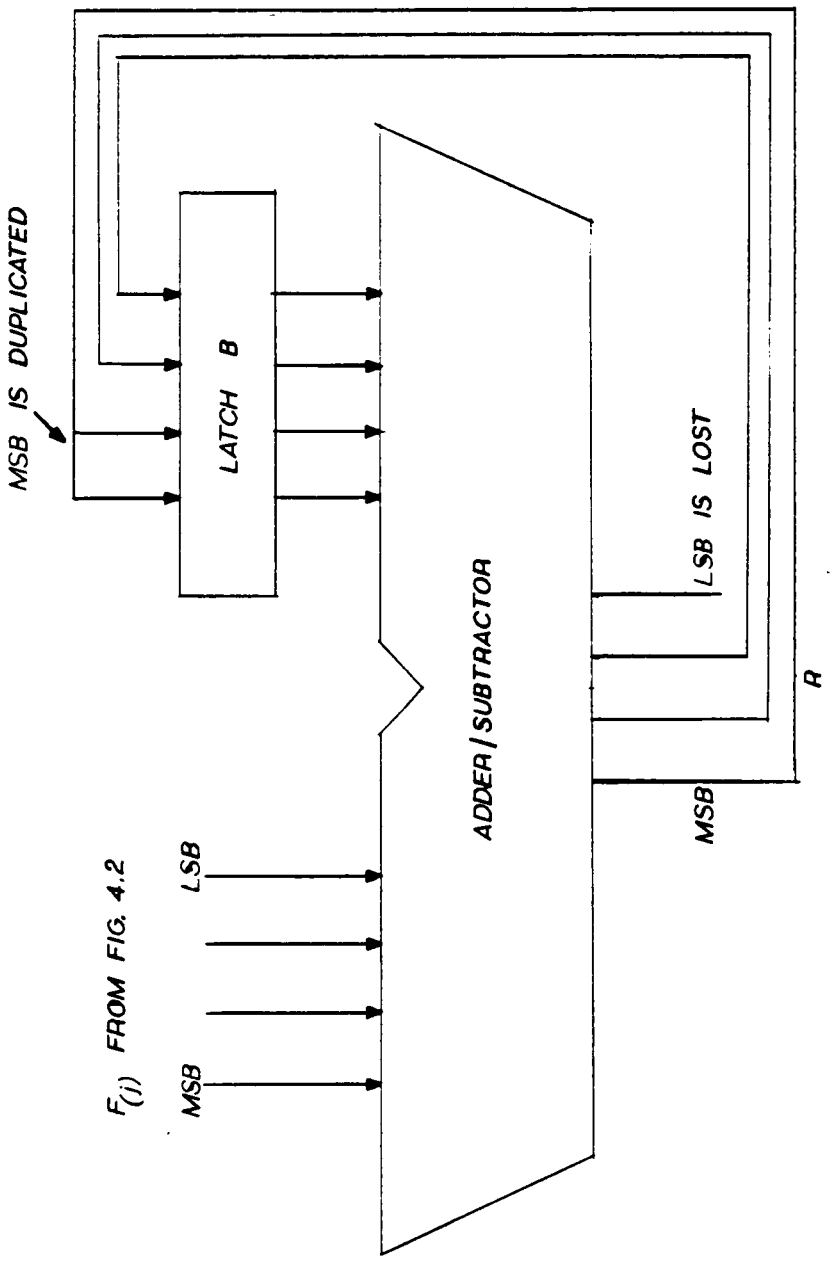


Figure 4.4. Wiring for feeding intermediate results to side B of ALU.

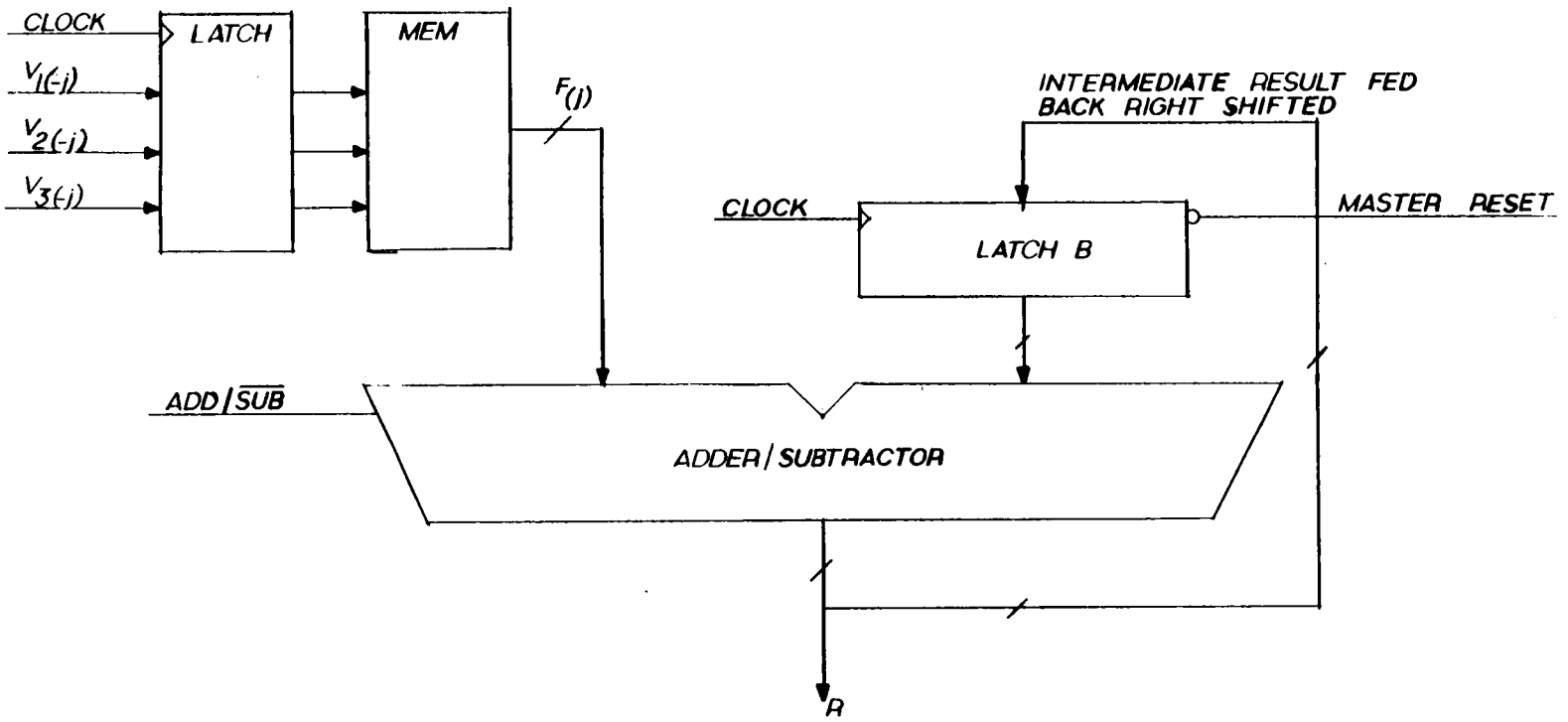


Figure 4.5. Summary of hardware for calculating a single filter node.

Digital Filter Implementation Using Distributed Arithmetic

The calculation by distributed arithmetic of a variable at a node having 1 incoming branches was discussed in the previous section and the implementation for 1 equal to 3 was presented. In this section the implementation is expanded to calculate the required variables for the digital filters given in Figures 3.1 and 3.2 (pgs. 36,37). For the moment consider only the first second order section of either Figure 3.1 or 3.2 which is shown in Figure 4.6. Note that for this section there are 3 nodes, n_{12}, n_{13} and n_{16} which are fed by 3 branches and must be calculated as in the previous section. All of these nodes are fed from nodes n_{11}, n_{14}, and n_{15} but through branches having different transmittances. These results mean that 3 separate functions denoted $F_{1i}(j)$ where i can be 1,2, or 3 with each having 8 possible values must be defined to calculate the values of nodes n_{12}, n_{13} and n_{16}. Also note that the values of n_{14} and n_{15} are values at n_{12} and n_{13} respectively delayed by one delay element. Therefore, the feeding node values would be $v_{12}(k)$, $v_{13}(k)$ and $v_{11}(k+1)$ for calculating the node values $v_{12}(k+1)$, $v_{13}(k+1)$, and $v_{16}(k+1)$ at time k+1. If the -jth bit is designated by $v_{12(-j)}(k)$, $v_{13(-j)}(k)$ and $v_{11(-j)}(k+1)$, the functions $F_{1i}(j)$ are defined as:

$$F_{11}(j) = d_1 v_{11(-j)}(k+1)+c_{11} v_{12(-j)}(k)+c_{12} v_{13(-j)}(k) \qquad (4.17)$$

$$F_{12}(j) = b_{11} v_{11(-j)}(k+1)+a_{111} v_{12(-j)}(k)+a_{112} v_{13(-j)}(k) \qquad (4.18)$$

$$F_{13}(j) = b_{12} v_{11(-j)}(k+1)+a_{121} v_{12(-j)}(k)+a_{122} v_{13(-j)}(k) \qquad (4.19)$$

where the subscripts 1, 2, and 3 on the function $F_{1i}(j)$ designates the calculation of nodes n_{16}, n_{12}, and n_{13} respectively. A total of 24 memory locations is needed to store the possible values of $F_{1i}(j)$ for i equal 1, 2, or 3.

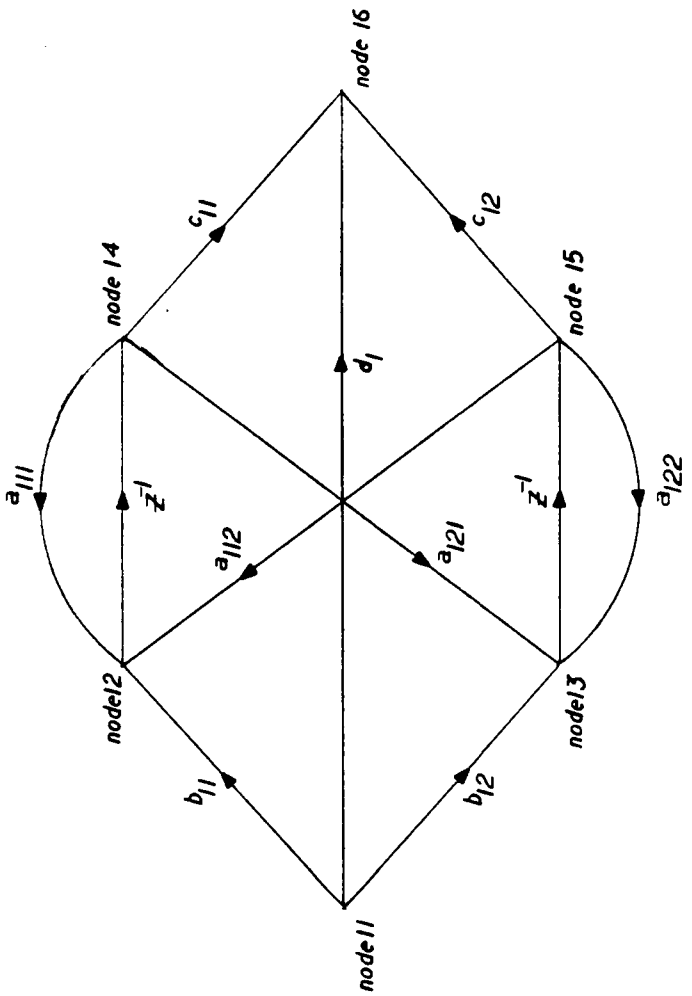


Figure 4.6. First second order filter section.

The sixth order Butterworth filters of Figure 3.1 and 3.2 (pgs. 36,37) have 3 second order sections. The results for the first second order section can be generalized as:

$$F_{n1}(j) = d_n v_{n1(-j)}^{(k+1)} + c_{n1} v_{n2(-j)}^{(k)} + c_{n2} v_{n3(-j)}^{(k)} \quad (4.20)$$

$$F_{n2}(j) = b_{n1} v_{n1(-j)}^{(k+1)} + a_{n11} v_{n2(-j)}^{(k)} + a_{n12} v_{n3(-j)}^{(k)} \quad (4.21)$$

$$F_{n3}(j) = b_{n2} v_{n1(-j)}^{(k+1)} + a_{n21} v_{n2(-j)}^{(k)} + a_{n22} v_{n3(-j)}^{(k)} \quad (4.22)$$

where n designates the filter section. For 3 filter sections, 72 memory locations are needed to store possible function values.

The circuit of Figure 4.5 must be modified as shown in Figure 4.7 to implement the sixth order Butterworth filter. The modifications include an expanded memory which requires 4 additional address lines, $A_3 - A_6$. These lines, which designate the section and node corresponding to the function $F_{ni}(j)$ to be accessed, can be supplied by a control unit for the filter.

The design of the control unit will be considered in a later chapter. The immediate goal is to develop hardware that can be used to determine addresses $A_0 - A_2$. For any node the addresses are created from bits feeding the node to be calculated (see section on Distributed Arithmetic). For the sixth order Butterworth filter of Figure 3.1 or 3.2, these bits are $v_{n1(-j)}^{(k+1)}$, $v_{n2(-j)}^{(k)}$, and $v_{n3(-j)}^{(k)}$. Recall that calculation of a node variable by distributed arithmetic begins at the LSB and proceeds to the MSB. The bits of the appropriate variables must be supplied to address lines $A_0 - A_2$ in this order. This can be accomplished by using right shift registers to store variables at feeding nodes with the LSB of the shift registers wired to $A_0 - A_2$. This modification is shown in Figure 4.8.

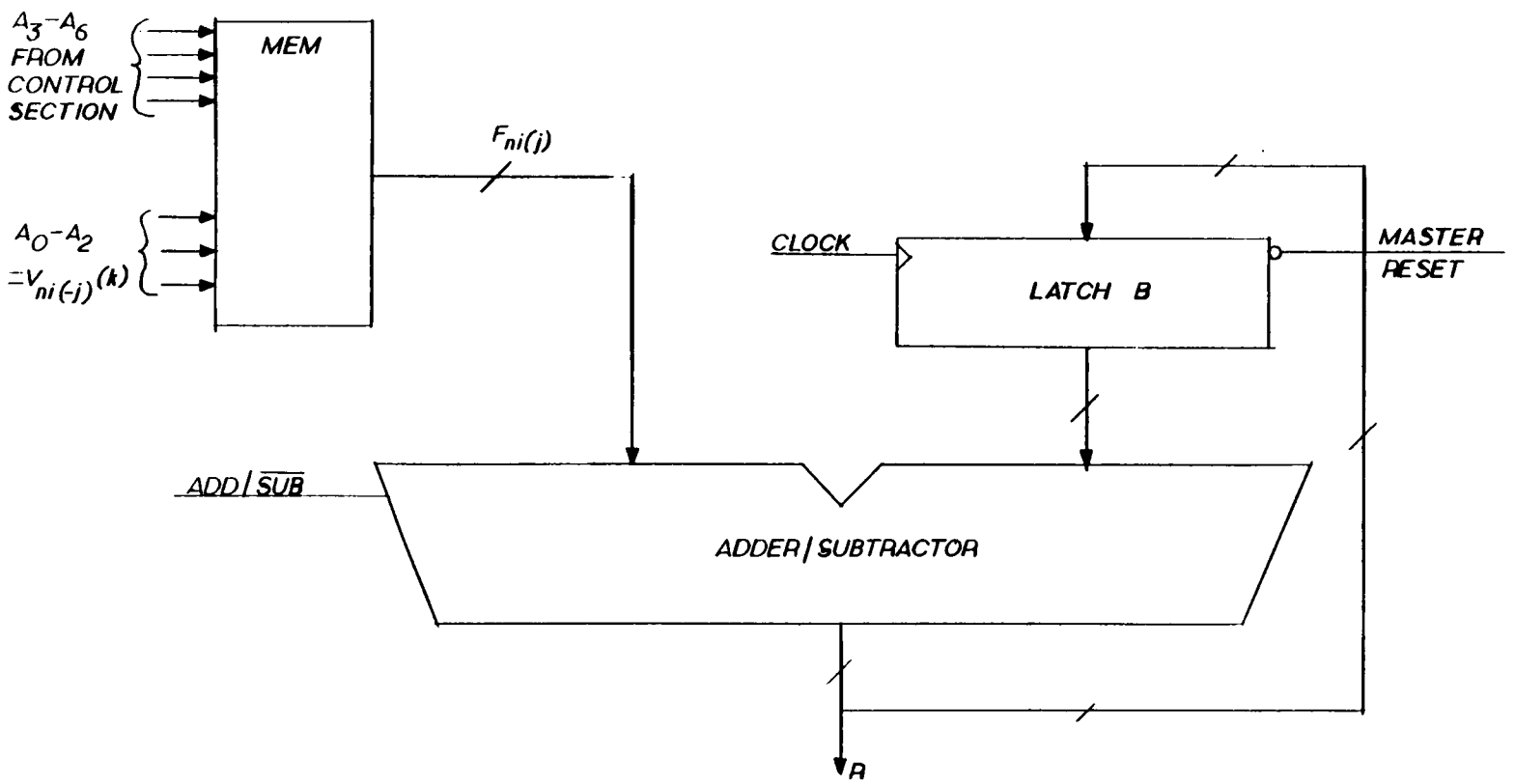


Figure 4.7. Hardware for calculating the 9 nodes of a sixth order Butterworth filter.

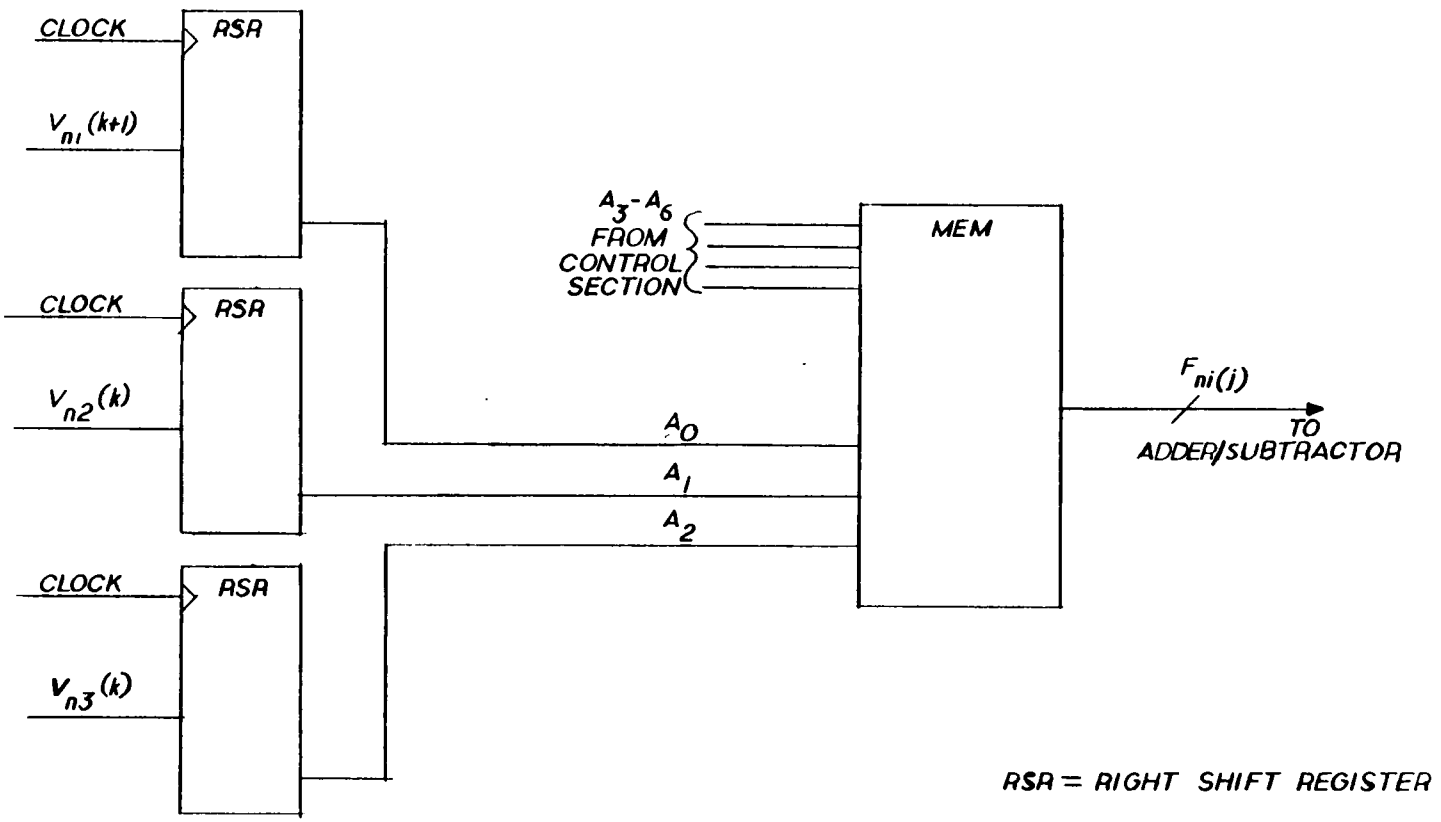


Figure 4.8. Circuitry for providing A0 - A2 to memory holding partial products.

Variables stored in the shift registers are required for the calculation of all 3 nodes of each second order section. The values can be retained in the shift registers by feeding the LSB from a shift register back to its MSB during a shift. The variable then continuously cycles through the shift register. By inserting a 2 to 1 multiplexer, the MSB of the shift register can be fed with its LSB or a bit from a different location. This mechanism can be used to recycle the values of feeding nodes of one section or to load values of the feeding nodes of a subsequent section while calculating the last node of the current section. The expanded hardware to access $F_{ni}(j)$ is shown in Figure 4.9.

Implementing the distributed arithmetic calculations for the digital filter can be completed by generating the next variable to be stored into the shift registers. The next variable to be used in accessing $F_{ni}(j)$ is the node values, v_{n2} and v_{n3}, calculated during the last filter cycle (at time k), and the value v_{n1} calculated from the current filter cycle (time k+1). The next variable is obtainable by passing the result from the arithmetic logic unit (ALU) through delay latches which ultimately feed the shift registers that supply A_0 - A_2. The appropriate circuit is shown in Figure 4.10. Variables $v_{n1}(k+1)$ are represented by shift registers 1 and 9, variables $v_{n2}(k)$ are represented by shift registers 5-8, and variables $v_{n3}(k)$ are represented by shift registers 2-4. Understanding of the circuit is perhaps best construed by tracing the data through a filter cycle. Consider the initiation of a calculation of the filter variables

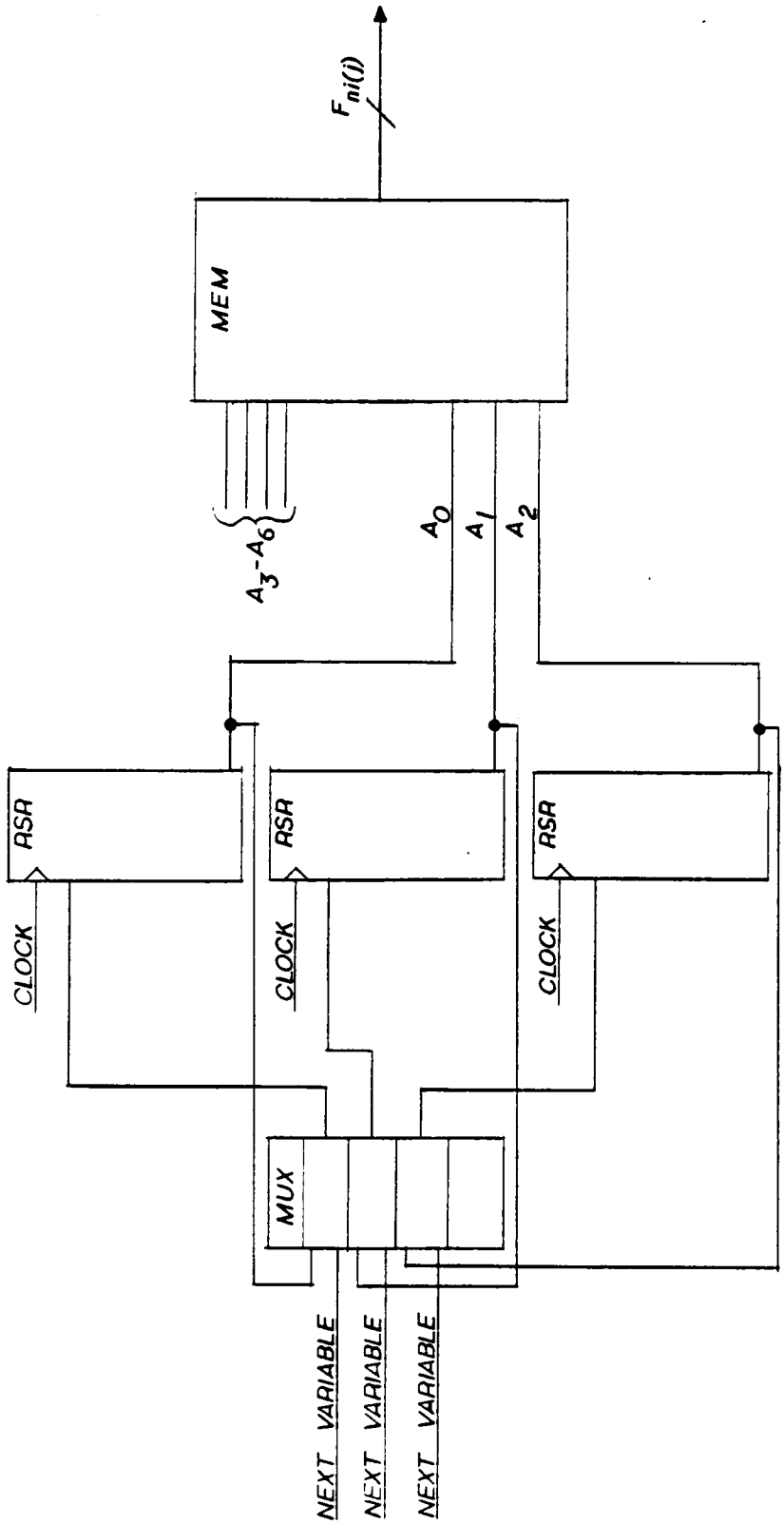


Figure 4.9. Multiplexer routes the appropriate bits to the shift registers.

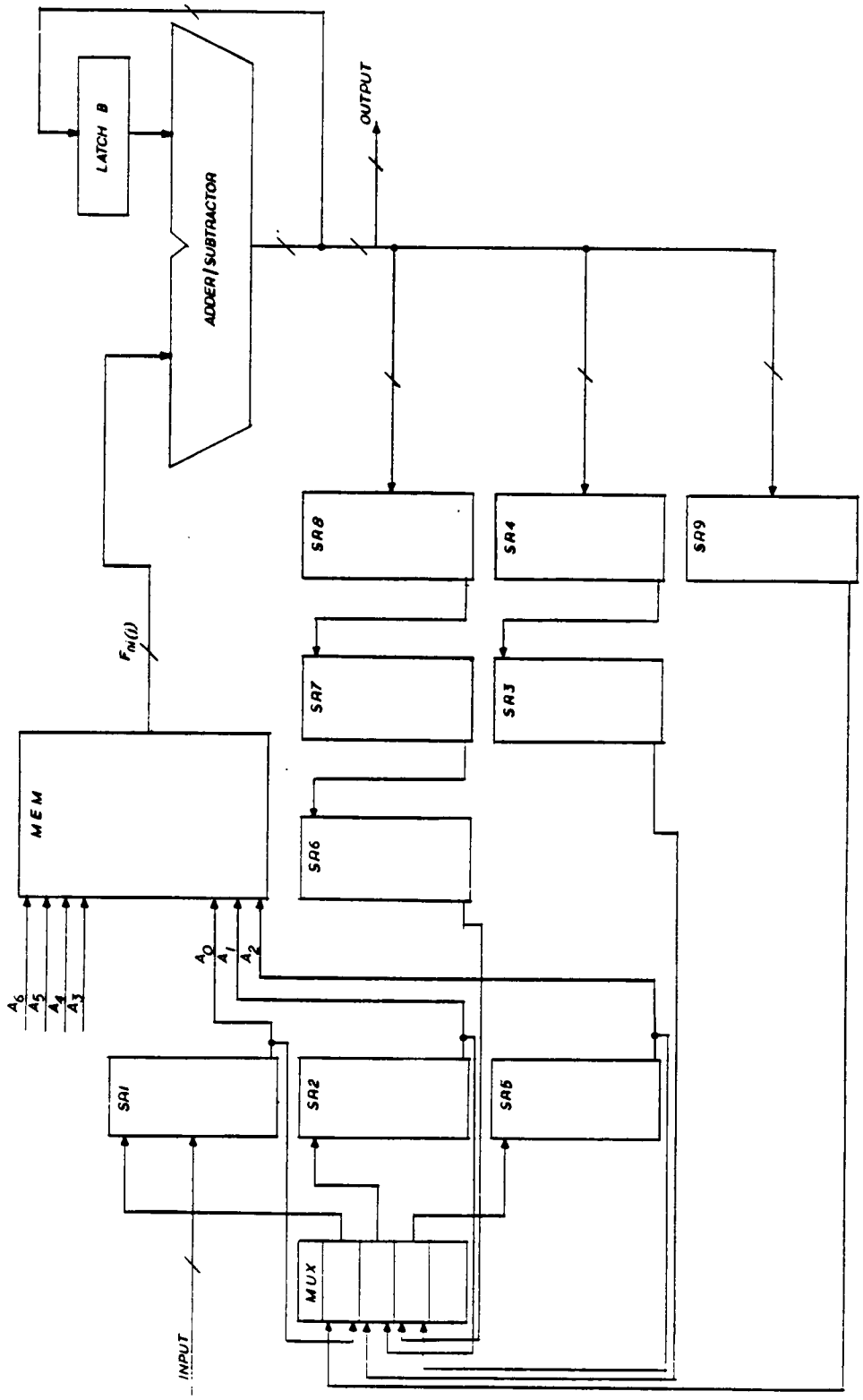


Figure 4.10. Circuit to calculate filter variable of sixth order Butterworth filter.

at time k+1. The cycle can be initiated by calculating the variable at node n_{12}, n_{13}, or n_{16}. Select node n_{16} as the initial node to be calculated. At this point the shift registers contain the following information: 1) shift register 1 (SR1) has been parallel loaded with the filter input; 2) SR8 and SR9 contain no information of interest; 3) SR2, SR3 and SR4 contain $v_{13}(k)$, $v_{23}(k)$ and $v_{33}(k)$ respectively; and SR5, SR6 and SR7 contain $v_{12}(k)$, $v_{22}(k)$ and $v_{32}(k)$ respectively. Address lines A_0 - A_2 are $v_{11(-(m-1))}(k+1)$, $v_{12(-(m-1))}(k)$ and $v_{13(-(m-1))}(k)$ respectively. Assuming the control unit is designating the function $F_{11}(j)$, the value accessed by memory is $F_{11}(m-1)$ as desired. The multiplexer is set so that the LSB of SR1, SR2 and SR5 are fed back to the MSB of SR1, SR2, and SR5 respectively. During the next m clock periods, address lines A_3 - A_6 remain constant while accessing function $F_{11}(j)$. At the end of m clocks, the result from the ALU, R, is $v_{21}(k+1)$ (see the Distributed Arithmetic section). This value is stored in SR9. At this point SR1-SR8 are the same as when the calculation was initiated and SR9 contains $v_{21}(k+1)$.

The control lines are then changed to access $F_{12}(j)$ and the multiplexer continues to cycle the bits of SR1, SR2 and SR5. Operation is as above with R after m clock periods being $v_{12}(k+1)$ which is stored in SR8. Now, SR1-SR7 and SR9 contain the same information as at the start of the calculation of $v_{12}(k+1)$.

Next the control lines access $F_{13}(j)$. Following this calculation, the variables $v_{11}(k+1)$, $v_{12}(k)$ and $v_{13}(k)$ will not be required in SR1, SR2 and SR5; therefore, the multiplexer is set to shift the variables for calculating nodes in the second stage of the filter into SR1, SR2

and SR5 as variable $v_{13}(k+1)$ is calculated. The variables required for calculation of stage 2 are $v_{23}(k)$, $v_{22}(k)$ and $v_{21}(k+1)$ which are stored in SR3, SR6 and SR9 respectively. Therefore, these registers are set to shift during the next m clock periods. In addition the remaining registers are also set to shift, thus preparing the filter for calculations of stage 3. After the m clock periods, R which is $v_{13}(k+1)$ is loaded into SR4. Shift register values are then as follows: SR1 contains $v_{21}(k+1)$; SR9 and SR8 contain no information of interest; SR2, SR3 and SR4 contain $v_{23}(k)$, $v_{33}(k)$, and $v_{13}(k+1)$ respectively; and SR5, SR6 and SR7 contain $v_{22}(k)$, $v_{32}(k)$, and $v_{12}(k+1)$. Verification that repeated application of the above sequence accomplishes the task of placing the appropriate variable in registers SR1, SR2, and SR5 is possible. Therefore, if the appropriate values are assigned to A_3 - A_6 by the control unit, the filter variables are obtained.

Control of Digital Filter Hardware

The basic hardware for calculating filter parameters for the digital filter is presented in Figure 4.10. Control lines must be provided to allow the orderly sequence of data passage and chip functions required to carry out these calculations. An algorithmic state machine (ASM) was used to supply this control. The ASM consists of memory in the form of an EPROM (or PROM) having selected outputs fed back to its address lines to control the next state of the machine. Additional output lines are provided to control the filter.

The memory of an ASM must be programmed to provide the required control lines. The programming of the ASM is deferred to Chapter V in

which detailed documentation of the filter is presented. The basic ASM hardware, however, is shown in Figure 4.11.

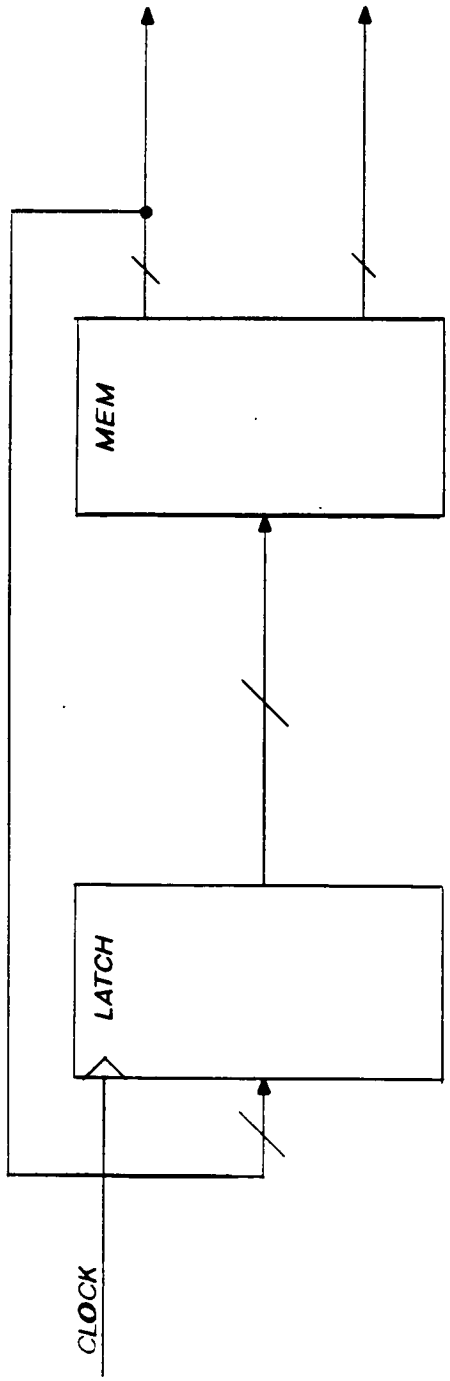


Figure 4.11. ASM hardware.

CHAPTER V

DISTRIBUTED ARITHMETIC IMPLEMENTATION OF TWO SIXTH ORDER LOW PASS BUTTERWORTH FILTERS

In this chapter the distributed arithmetic implementation of the reduced roundoff error Butterworth filters developed in Chapter III are presented. Each filter was to possess a 1 MHz clock and accept an analog sine wave input. Initially both filters were implemented using 8 bit length words for variable representation. The resulting 20% bandwidth filter behaved as expected with the output following sinusoidal inputs having frequencies within the filter's bandwidth. The output of the 2% filter, however, did not follow sinusoidal inputs. These results were explained by the choices of the scaling parameter δ. The appropriate δ for the 20% filter was 1 while δ for the 2% filter was 5 (pgs.33,34). Since the roundoff error increases with δ(all other factors equal), a higher roundoff error in the 2% filter could explain the poor output. The variable length for the 2% filter, therefore, was increased to 12 bits to reduce the roundoff error. After this increase, the 2% filter behaved as expected.

Because of the extra bits in the 2% filter's implementation, the 2% filter was more complicated than the 20% filter. Therefore the implementation of the 20% filter is first presented; then modifications to realize the 2% filter are presented. In both cases, the hardware is presented followed by a development of the software.

Implementation of the 20% of Nyquist Frequency Bandwidth Filter

Hardware. The hardware required to implement the 20% bandwidth filter could be divided logically into 5 major sections. These sections are shown in Figure 5.1 along with the necessary communication between sections. These sections are the input, output, clock, control, and computational sections. The clock section provided a 1 MHz signal for synchronizing events in the computational and control sections. The control section, in turn, provided signals for function synchronization in the input, output, and computational sections. Analog signals applied to the filter were converted into an 8 bit 2's complement representation in the input section. This 8 bit 2's complement signal was then passed to the computational section where the digital filter algorithm operated on the input to produce an 8 bit 2's complement output. The output section converted this 8 bit output into an analog signal. Detailed circuit diagrams and functional descriptions of each section are presented in this chapter.

Clock section. The clock section provided the filter with a 1 MHz square wave for synchronization. This square wave was provided by a 74LS124 voltage controlled oscillator excited with a 1 MHz crystal. Logical ANDing of the 1 MHz square wave with an enable signal allowed the filter to be turned on or off. The enable signal was provided by a switch debounced with a 74LS74 D flip-flop. With the switch in the on position, the output of the D flip-flop was high enabling the 1 MHz square wave to propagate through the AND gate. When in the off position, the output of the D flip-flop was low, thus maintaining the

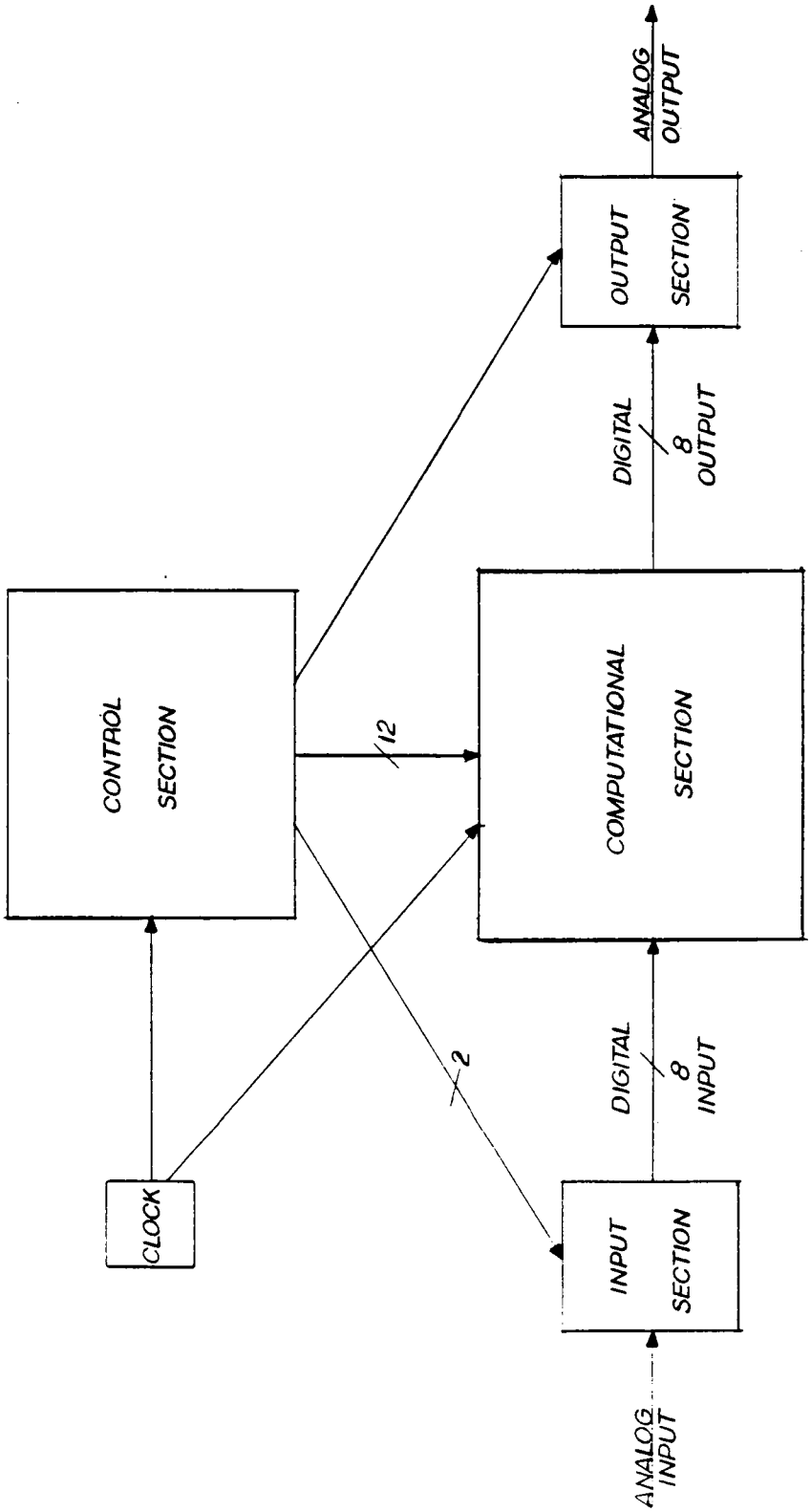


Figure 5.1. Major filter sections.

output of the AND gate at the low level. Paralleling 2 AND gates was used to improve the fan out. The circuit is shown in Figure 5.2.

Input section. The input section sampled an analog input and converted the sampled value to an 8 bit 2's complement representation. This function was provided by a commercial sample-and-hold chip, the AD582, in conjunction with a commercial analog-to-digital converter (ADC) chip, the AD570. The AD582 and AD570 were configured to convert a ±5V signal into 00 to FF hexadecimal (represented as 00H and FFH). This configuration is shown in Figure 5.3. From the ADC a 00H corresponded to -5V, 80H corresponded to 0V, and FFH corresponded to +5V. The inverter gate was included to complement the MSB of the ADC. This produced the 2's complement representation in which 80H corresponded to -5V, 00H corresponded to 0V, and 7FH corresponded to +5V.

Outputs from the AD570 ADC were buffered with a 74LS244 octal buffer. This buffering was required because the AD570 is slow compared to the clock period. The AD570 has a typical conversion time of 25μs measured from the lowering of the B/$\overline{C}$ pin. The 1 MHz clock however provided a 1μs window for the computation section to read the input. Thus, the B/$\overline{C}$ pin had to be lowered at least 26 clock periods before the computational section was instructed to read the output from the input section. Since the output from the ADC was to be placed on a data buss, the 74LS244 octal buffer was required to prevent buss contention by the AD570 and the shift registers which shared the buss. This buffer has a maximum propagation delay of 30 ns from the enable to the output. Therefore, with the 74LS244 disabled, the AD570 could be

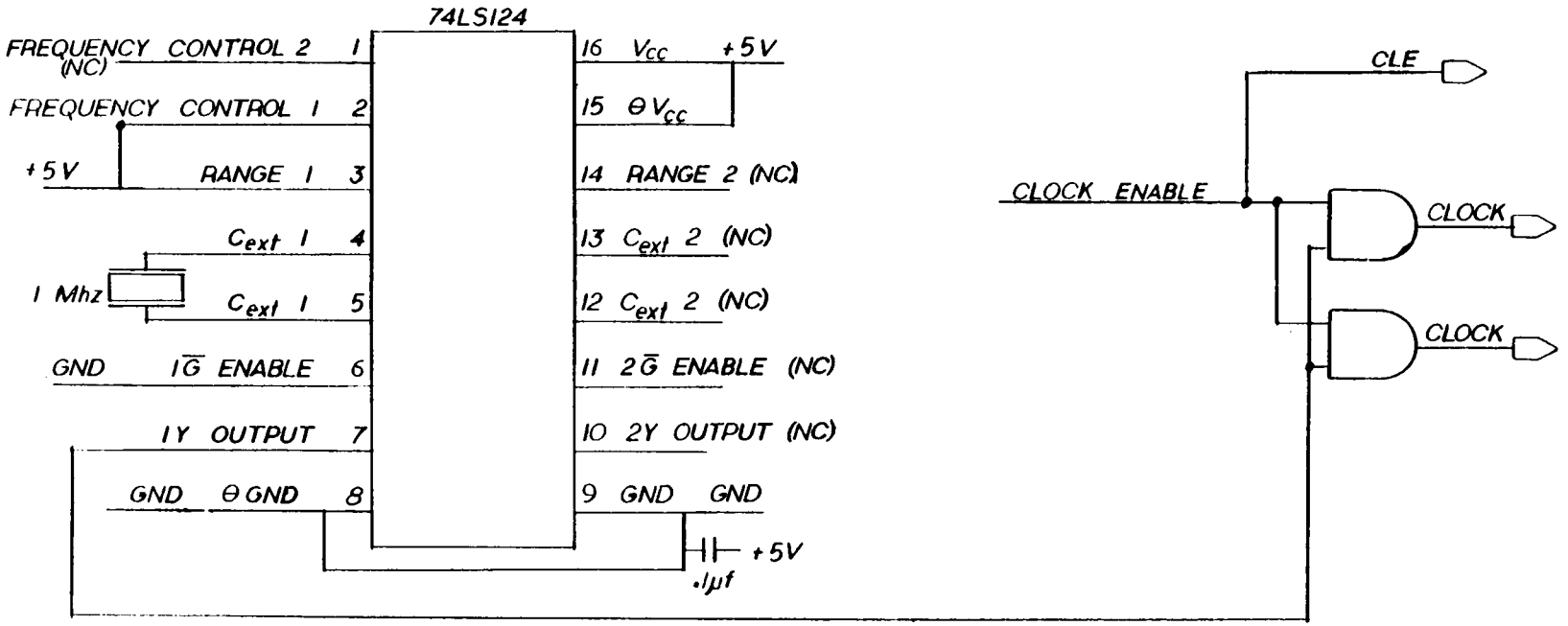


Figure 5.2. Clock section.

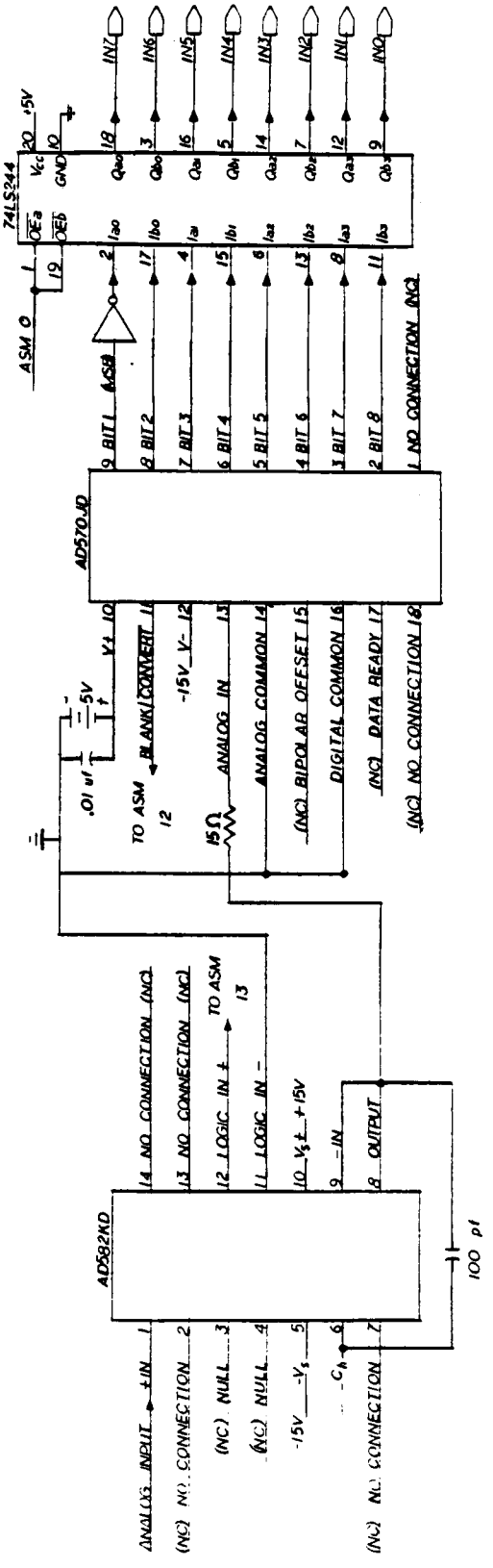


Figure 5.3. Input section.

instructed to convert the input long before the read cycle. When the computational section was ready to receive the data, the 74LS244 was enabled, thereby presenting the data from the input section to the computational section within 30 ns. This data was then latched by a shift register in the computational section at the next low to high clock transition. After this clock transition, the buffer was disabled and returned to its high impedance state preventing buss contention.

Output section. The results emerging from the computational section were converted to an analog signal by the output section. This conversion was accomplished with an 8 bit commercial digital to analog converter (DAC), the AD558. The AD558 was wired as shown in Figure 5.4. Line ASM7 was connected to the $\overline{CE}$ pin of the DAC. A low to high transition on this line caused data from the computational section (R2-R9) to be latched into the DAC. The inverter on data line R2 was used to convert the 2's complement representation from the computational section into an unsigned base 2 representation for the DAC. ASM7 was fed to the DAC from the control section. The control line, ASM7, and the data buss had to satisfy the following 3 timing constraints for the DAC to latch data reliably: 1) data hold time of 10 ns, 2) data setup time of 100 ns, and 3) minimum low pulse width of 100 ns. These constraints will be considered in the discussion of the control section.

Computational section. The distributed arithmetic approach for calculating filter variables was discussed in Chapter IV. An outline of the hardware required to perform these calculations was also

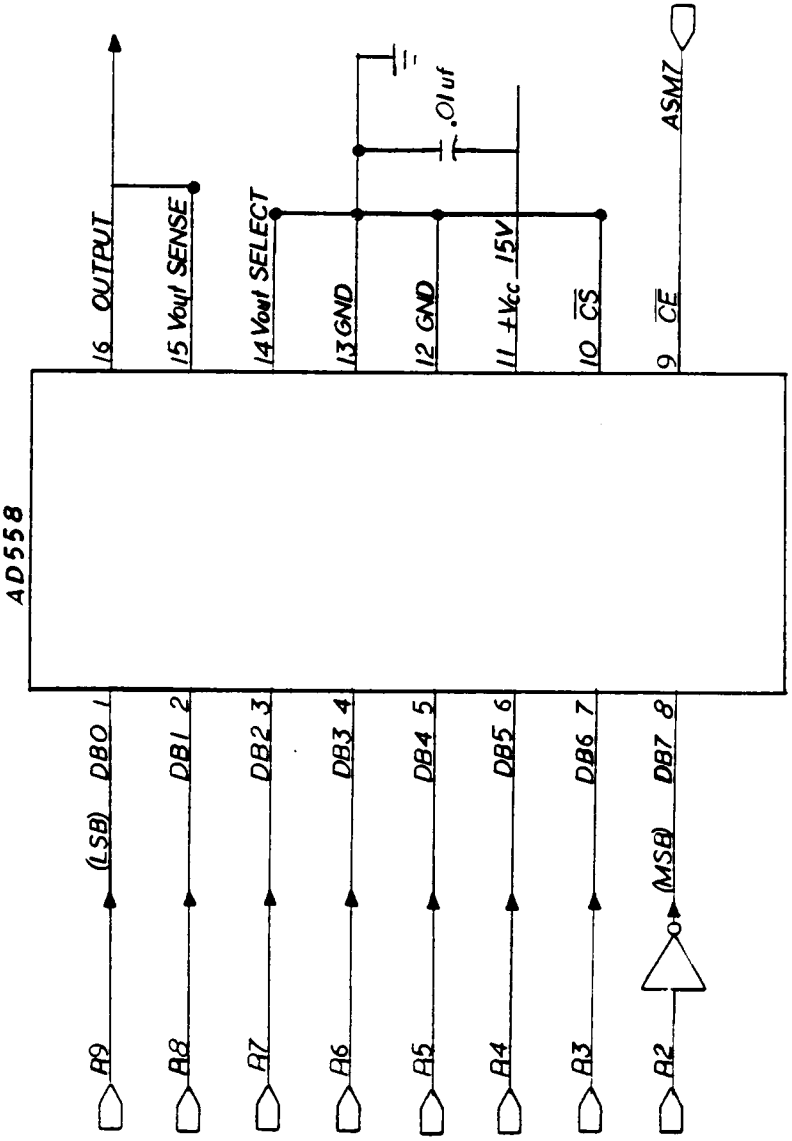


Figure 5.4. Output section.

developed. In this section, the hardware and connections used for calculating filter variables for the 20% Butterworth filter are presented in detail.

The circuit diagram for the computational section is shown in Figure 5.5. In this diagram, U1-U9 are 8 bit shift registers corresponding to SR1-SR9 respectively in Chapter IV. Modules U10-U12 are octal buffers, U13 is a quad 2 to 1 multiplexer, U14 and U15 are octal edge triggered latches, U16-U18 are ALU's and U19-U20 are EPROMS holding the partial products for the filter calculation. When the filter was turned off, the clock enable signal (CLE) discussed in the clock section was used to asynchronously reset the shift registers and latches. Therefore, when the filter was turned on, the filter proceeded through the 72 step sequence of the ASM, which calculated all 9 filter variables, with all shift registers and the latches containing zeros. During this time, the input section sampled the input and converted it into an 8 bit, 2's complement representation so that at the beginning of the next 72 step sequence the sampled input could be parallel loaded into U1. The filter operated on this data as discussed in Chapter IV by using the bits of appropriate filter variables to address lookup values in memory. The lookup values were combined to produce the desired filter variable. When the output variable was calculated, the data was latched into the DAC of the output section where the 2's complement representation was converted to an analog signal.

Filter calculations were performed using fixed point arithmetic with the assumption that filter variables were always scaled to be less

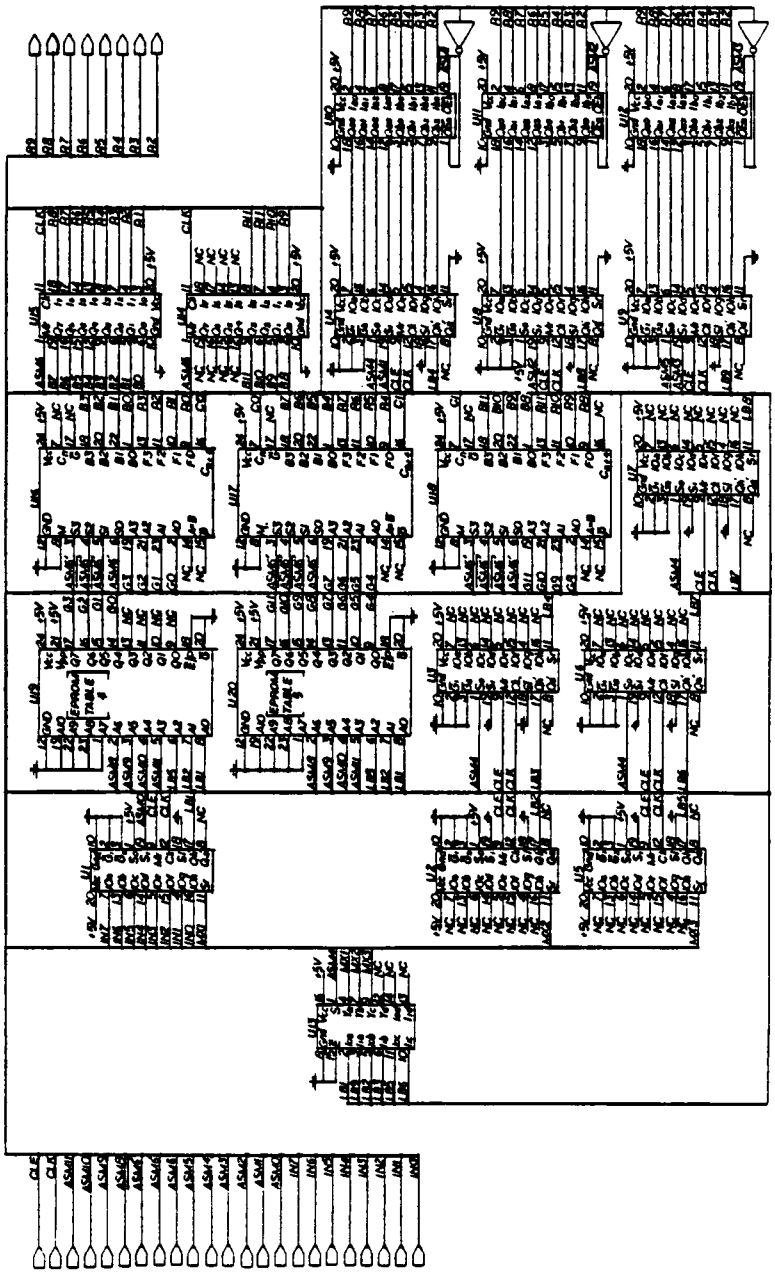


Figure 5.5. Computational section of 20% filter.

than 1. For the distributed arithmetic approach in which several additions and 1 subtraction were required to determine a filter variable, intermediate results from the additions were not guaranteed to remain less than 1.0. Additional bits in the ALU, U16-U18, were required to handle the largest expected intermediate value. Since the ALU 74LS181 chips used in the filter were 4 bits wide, 3 of them were paralleled giving a total of 12 bits to the arithmetic logic unit. The data, represented in the ALU as XXX.XXXXXXXXX, allowed for intermediate values as large as 3.996.

Two 74LS273's, U14 and U15, were used as register B of the ALU. The 12 bit result from the ALU was latched into the B register and hardwired so that the result was right shifted 1 bit with the MSB duplicated. Right shifting the output from the ALU divided each intermediate result by 2 as discussed in Chapter IV. Two MM2716 EPROM's, U19 and U20, were used as the function memory to provide 12 bit functional values to the A side of the ALU. These functional values were stored in the format of the ALU as XXX.XXXXXXXXX.

The remainder of the computational section was 8 bits wide. Nine shift registers, 3 buffers, and a quad 2 to 1 multiplexer comprised the remainder of the computational section. The 9 shift registers, U1-U9, were 74LS299's while the quad 2 to 1 multiplexer, U13, was a 74LS157. The 3 74LS244 buffers, U10-U12, were required to buffer the multiplexed I/O ports of shift registers 4, 8, and 9 from the outputs of the ALU. Otherwise, when these shift registers were in the shift mode, the outputs of the ALU would be driving outputs of the shift register. Thus, a total of 20 modules were required in the computational section.

Control section. The control section shown in Figure 5.6 provided signals to control the sequence of logic functions to be performed by the hardware, as well as, timing requirements for these functions. These control signals are provided by an ASM which consisted of a 74LS273 latch and three MM2716 EPROM's. Some of the outputs of the memory were fed back to the 74LS273 to be latched on the next clock pulse. These outputs provided the next address for the memory. For the 20% filter, 72 clock pulses were required to calculate all filter variables by the distributed arithmetic approach. Therefore 72 sequential steps or memory locations were needed for filter control data. Seven address bits were required to address these 72 locations.

An additional 13 outputs were used to control filter functions. Of these 13 outputs, 11 could be provided as direct outputs of the ASM. In the computational section of the filter, the memory and ALU acted as asynchronous devices while the shift registers and register B acted as synchronous devices. The synchronous devices were synchronized by the clock. Each clock period began immediately following a low to high transition on the clock and ended immediately after the next low to high transition. Control signals from the ASM informed the synchronous devices as to the function they were to fulfill upon the next low to high transition of the clock. The asynchronous devices were also informed as to the data to display or the functions to perform during the clock period. Data to the synchronous devices must become valid before the low to high transition of the clock and meet the setup and hold times of the device.

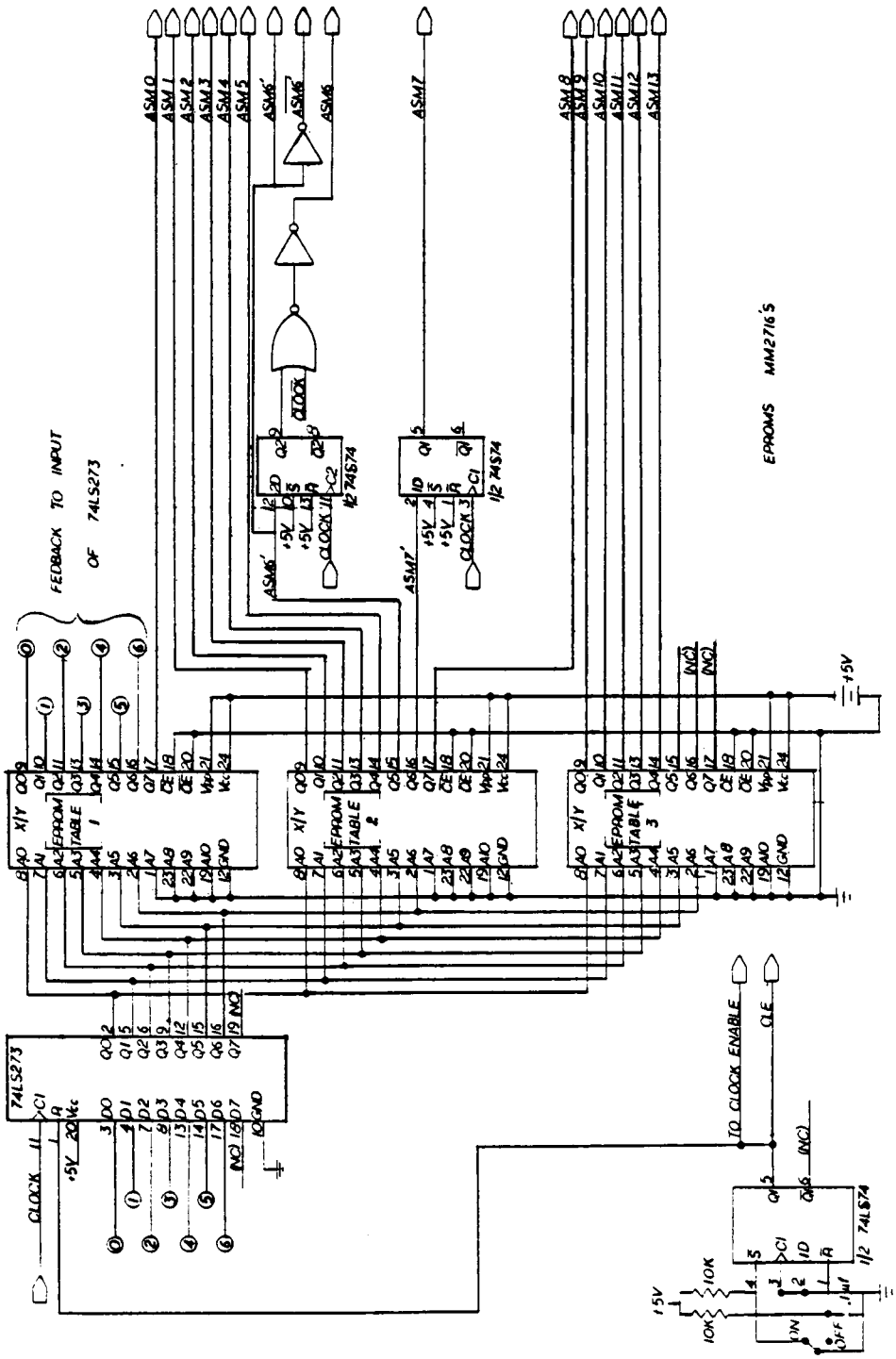


Figure 5.6. Control section.

Two signals, ASM6' and ASM7', however, had to be reshaped to meet critical timing criteria. One control line requiring shaping provided the control signal to asynchronously reset latch B. Consider the last clock period of a calculation of a filter variable, for example, clock period 56. During this period the ALU must be instructed to subtract. After the data from the memory has settled, the output from the ALU would become the valid filter variable. The appropriate shift register, either U4, U8 or U9, meanwhile would be instructed to parallel load the resulting variable value on the next positive clock edge. Calculation of the next filter variable would begin during the next clock cycle, for example, the 57th clock period. Sometime between latching the valid value into the appropriate shift register and the next positive clock edge, register B must be reset. However, if the ASM instructed register B to reset during the next clock period, for example, clock period 57, then register B would continue to reset at the 57th positive clock edge rather than latch the initial results for the calculation of the next filter variable. The B register, therefore, should be instructed to reset only for the first half of the next clock period so that it could also latch the ALU results on the 57th clock transition. The above statements are illustrated in Figure 5.7.

The results of Figure 5.7 can be used to develop the desired control signal. The truth table for the desired signal,

ASMR'	CLOCK	ASMR
0	0	1
0	1	0
1	0	1
1	1	1

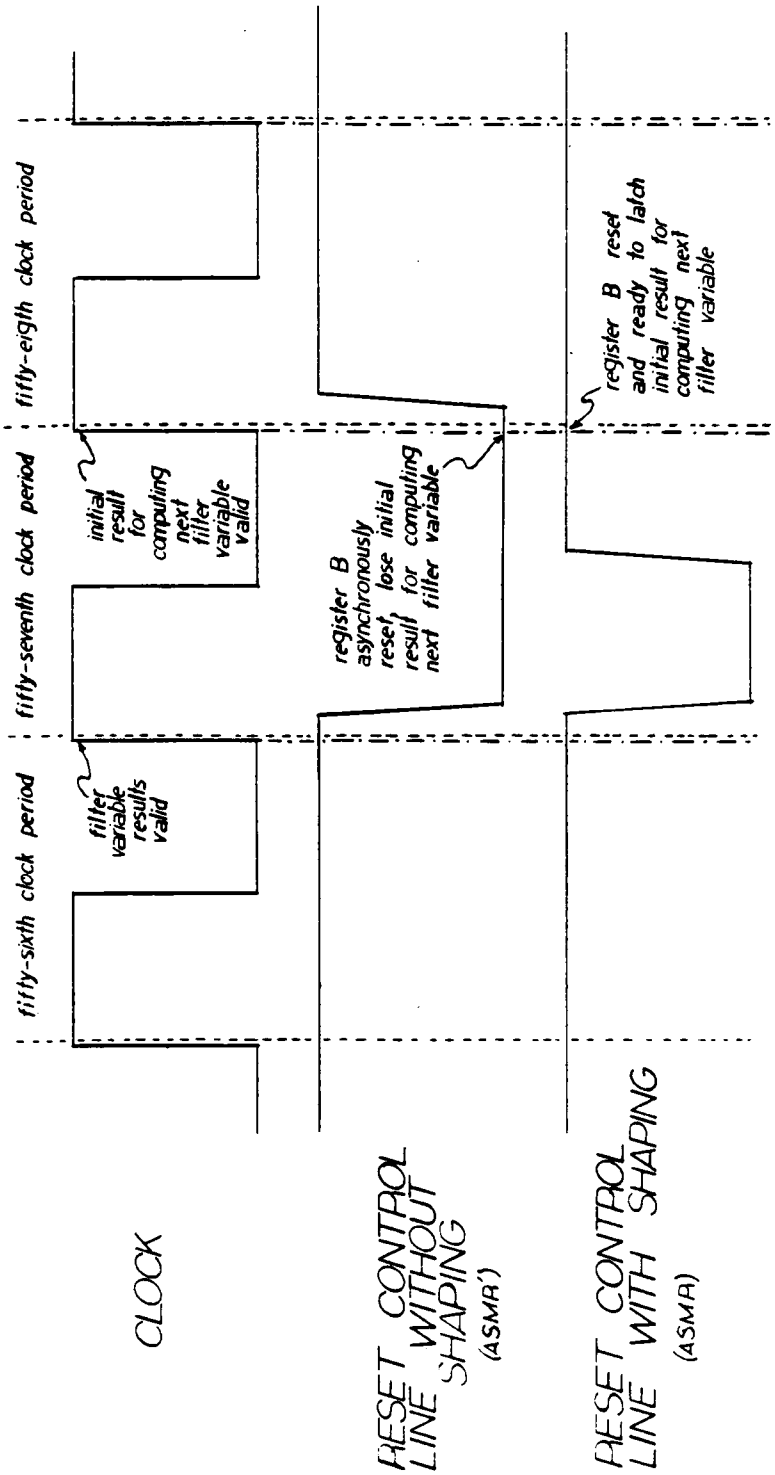
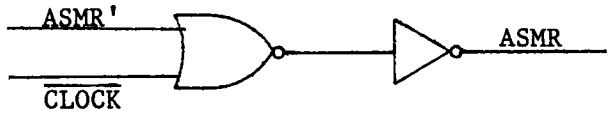


Figure 5.7. Timing for resetting register B.

can be used to derive the relationship between ASMR, ASMR' and CLOCK which is:

$$ASMR = \overline{CLOCK} + ASMR' = \overline{\overline{CLOCK} + ASMR'} \qquad (5.1)$$

where ASMR' represents the signal from the ASM, and ASMR represents the reshaped signal. The circuit realizing this relationship is:



A new timing problem, however, would develop if this circuit was used. Refer again to Figure 5.7. Some time after the low to high clock transition at the end of the 56th clock period, ASMR' would go low. This lowering of ASMR' would cause ASMR to fall as desired. Halfway through the clock period, the clock would also go low bringing ASMR high. To this point the operation would proceed as desired. At the 57th positive clock edge, however, the operation would falter. When the clock returned to the high state, ASMR' would not immediately rise to the high state. Therefore ASMR would return low for a short time till ASMR' returned high. This transition would produce a glitch on the asynchronous reset of register B and erase the data latched during the clock transition. This glitch would occur because the delay from the positive clock edge to the positive edge of ASMR' would be longer than the delay from the positive clock edge to the inverters high to low transition.

The above problem was easily overcome with the use of a fast D flip-flop. The ASMR' pulse was advanced by 1 clock period and fed to a 74S74 D flip-flop to delay the pulse to the original clock period.

ASMR' then corresponded to ASM6'. The nor-not circuit was connected to the output of this D flip-flop; these connections are shown in Figure 5.6 (pg. 73). Referring again to Figure 5.7, ASM6' went low during the 56th clock period. However, the D input of the flip-flop was high on the 55th clock transition, therefore, its output was high during the 56th period. At the 56th positive clock edge, ASM6' was low so that the D flip-flop output went low during the 57th clock period. This caused the output of the nor-not circuit to go low for the first half of the 57th clock period as before. At the 57th positive clock edge, however, the current circuit behaved differently than the previous circuit. At the time of this transition ASM6' was high so that the octal D flip-flop output went high with a maximum delay of 9 ns. The 74LS04 inverter, however, required 15 ns maximum for the transition. Therefore the ASM6' high transition to the nor gate preceded the low transition of the $\overline{\text{CLOCK}}$ pulse precluding the undesirable glitch. The resulting hardware is shown on ASM6 of Figure 5.6.

A similar timing problem occurred on the signal line controlling latching of data into the DAC. The control line providing this signal was ASM7'. With ASM7' connected to pin $\overline{\text{CE}}$ of the DAC, a low to high transition on ASM7' instructed the DAC to latch the data at its input. For this data to be latched reliably, it had to meet the DAC's requirements of a 100 ns setup time and a 10 ns hold time. Also the signal to $\overline{\text{CE}}$ needed a minimum strobe pulse width of 100 ns. Assume that ASM7', an output of the ASM memory, was directly connected to $\overline{\text{CE}}$ of the DAC. In this case the ALU was instructed to subtract during the clock period in which the filter output became valid. This output was

valid on the following positive clock edge; therefore, ASM7' instructed the DAC to latch the data at this time. To provide this signal, ASM7' was low during the clock period when the filter output became valid and went high during the next clock period. However, the ASM memory took around 100 ns to make the transition. By this time, the master reset of register B was active and the data from the ALU was no longer valid. In other words, the hold time required by the DAC was not met.

To overcome this timing violation, the positive edge of $\overline{CE}$ needed to occur before the reset signal to register B became active. Referring to Figure 5.6 (pg. 73), the propagation delay from the clock transition to reset signal for register B (ASM6) was 39 ns maximum. The propagation delay for register B was 27 ns maximum, and the propagation delay for the 74LS181 ALU chips was 38 ns maximum. Therefore, the total delay to the ALU outputs from the clock transition was 110 ns maximum. The setup and hold times for the DAC were met using a scheme similar to that for ASM6. The ASM7' pulse was advanced 1 clock period. This output of the ASM was connected to the D input of a 74S74 D flip-flop. This flip-flop delayed the pulse to the desired clock period; however, the propagation delay from the clock transition to the transition on the output of the D flip-flop was reduced to that of the 74S74 which was 9 ns maximum. Since this delay was less than the 110 ns delay to the ALU outputs, the hold time of 10 ns was easily met. Therefore, valid data was latched into the DAC.

Software. Two sets of memory had to be programmed to obtain an operating filter. These memory sets were the memory for the ASM

(filter control) and the memory containing the functional values used in the distributed arithmetic calculations. Programming of the ASM memory to provide filter control is first presented followed by a presentation of the contents that must be programmed into the function memory.

Normally ASM's are programmed by using ASM flow charts. For the digital filter, the ASM proceeded repeatedly through 72 sequential steps; no branching was required. Therefore, rather than using an ASM flow chart, the ASM memory was programmed by developing a table of the desired chip functions for each of the 72 steps of the sequence. These functions were translated into signals required to select the proper chip functions for each clock period. The signals were compared and combined when possible to reduce the number of outputs required from the ASM.

Table 5.1 presents the functions that each chip must perform for the 72 clock periods required to calculate filter variables. These functions coincide with the function sequence described in Chapter IV on the use of distributed arithmetic for calculating filter variables. The signals required to realize these functions are presented in Table 5.2.

In addition to the outputs of Table 5.2, outputs for controlling the sequence of the ASM must be included in the ASM. Since the ASM was to sequence through the 72 steps without branching, these outputs formed an arithmetic sequence with a difference between terms of unity. The revised ASM signals are presented in Table 5.3. The entries of Table 5.3 were converted to hexadecimal notation for the programming of

Table 5.1. Functions to be Performed to Calculate Filter Variables of 20% Filter.

FILTER STATE	SHIFT REGISTER 1 (74LS 299)	SHIFT REGISTER 2 (74LS 299)	SHIFT REGISTER 3 (74LS 299)	SHIFT REGISTER 4 (74LS 299)	SHIFT REGISTER 5 (74LS 299)	SHIFT REGISTER 6 (74LS 299)	SHIFT REGISTER 7 (74LS 299)	SHIFT REGISTER 8 (74LS 299)	SHIFT REGISTER 9 (74LS 299)	A L U (74LS 181)	OCTAL D FLIP FLOPS (74LS 273)	MUX FOR SHIFT REG 1 (74LS 157)	MUX FOR SHIFT REG 2 (74LS 157)	MUX FOR SHIFT REG 3 (74LS 157)	SAMPLE HOLD (LOGIC)	ANALOG TO DIGITAL (ADC80)	DIGITAL TO ANALOG (ADC80)	OCTAL BUFFER 1 (74LS 244)	OCTAL BUFFER 4 (74LS 244)	OCTAL BUFFER 8 (74LS 244)	OCTAL BUFFER 9 (74LS 244)
0	P.L.	S.R.	S.R.	P.L.	S.R.	S.R.	S.R.	SR/H	SR/H	SUB	STORE	1	1	1	N.	B.	H.	E	E	E	E
1	S.R.	S.R.	H.	H.	S.R.	H.	H.	SR/H	SR/H	ADD	CLEAR	1	1	1	SAMPLE	B.	H.	E	E	E	E
2	S.R.	S.R.	H.	H.	S.R.	H.	H.	SR/H	SR/H	ADD	STORE	1	1	1	SAMPLE	B.	H.	E	E	E	E
3	S.R.	S.R.	H.	H.	S.R.	H.	H.	SR/H	SR/H	ADD	STORE	1	1	1	SAMPLE	B.	H.	E	E	E	E
4	S.R.	S.R.	H.	H.	S.R.	H.	H.	SR/H	SR/H	ADD	STORE	1	1	1	SAMPLE	B.	H.	E	E	E	E
5	S.R.	S.R.	H.	H.	S.R.	H.	H.	SR/H	SR/H	ADD	STORE	1	1	1	SAMPLE	B.	H.	E	E	E	E
6	S.R.	S.R.	H.	H.	S.R.	H.	H.	SR/H	SR/H	ADD	STORE	1	1	1	SAMPLE	B.	H.	E	E	E	E
7	S.R.	S.R.	H.	H.	S.R.	H.	H.	SR/H	SR/H	ADD	STORE	1	1	1	SAMPLE	B.	H.	E	E	E	E
8	S.R.	S.R.	H.	H.	S.R.	H.	H.	SR/H	R.L.	SUB	STORE	1	1	1	SAMPLE	B.	H.	E	E	E	E
9	S.R.	S.R.	H.	H.	S.R.	H.	H.	SR/H	H.	ADD	CLEAR	1	1	1	SAMPLE	B.	H.	E	E	E	E
10	S.R.	S.R.	H.	H.	S.R.	H.	H.	SR/H	H.	ADD	STORE	1	1	1	SAMPLE	B.	H.	E	E	E	E
11	S.R.	S.R.	H.	H.	S.R.	H.	H.	SR/H	H.	ADD	STORE	1	1	1	SAMPLE	B.	H.	E	E	E	E
12	S.R.	S.R.	H.	H.	S.R.	H.	H.	SR/H	H.	ADD	STORE	1	1	1	SAMPLE	B.	H.	E	E	E	E
13	S.R.	S.R.	H.	H.	S.R.	H.	H.	SR/H	H.	ADD	STORE	1	1	1	SAMPLE	B.	H.	E	E	E	E
14	S.R.	S.R.	H.	H.	S.R.	H.	H.	SR/H	H.	ADD	STORE	1	1	1	SAMPLE	B.	H.	E	E	E	E
15	S.R.	S.R.	H.	H.	S.R.	H.	H.	SR/H	H.	ADD	STORE	1	1	1	SAMPLE	B.	H.	E	E	E	E
16	S.R.	S.R.	H.	H.	S.R.	H.	H.	R.L.	H.	SUB	STORE	1	1	1	SAMPLE	B.	H.	E	E	E	E
17	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	ADD	CLEAR	1	1	1	SAMPLE	B.	H.	E	E	E	E
18	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	ADD	STORE	1	1	1	SAMPLE	B.	H.	E	E	E	E
19	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	ADD	STORE	1	1	1	SAMPLE	B.	H.	E	E	E	E
20	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	ADD	STORE	1	1	1	SAMPLE	B.	H.	E	E	E	E
21	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	ADD	STORE	1	1	1	SAMPLE	B.	H.	E	E	E	E
22	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	ADD	STORE	1	1	1	SAMPLE	B.	H.	E	E	E	E
23	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	ADD	STORE	1	1	1	SAMPLE	B.	H.	E	E	E	E
24	S.R.	S.R.	S.R.	R.L.	S.R.	S.R.	S.R.	SR/H	SR/H	SUB	STORE	1	1	1	SAMPLE	B.	H.	E	E	E	E
25	S.R.	S.R.	H.	H.	S.R.	H.	H.	SR/H	SR/H	ADD	CLEAR	1	1	1	SAMPLE	B.	H.	E	E	E	E
26	S.R.	S.R.	H.	H.	S.R.	H.	H.	SR/H	SR/H	ADD	STORE	1	1	1	SAMPLE	B.	H.	E	E	E	E
27	S.R.	S.R.	H.	H.	S.R.	H.	H.	SR/H	SR/H	ADD	STORE	1	1	1	SAMPLE	B.	H.	E	E	E	E
28	S.R.	S.R.	H.	H.	S.R.	H.	H.	SR/H	SR/H	ADD	STORE	1	1	1	SAMPLE	B.	H.	E	E	E	E
29	S.R.	S.R.	H.	H.	S.R.	H.	H.	SR/H	SR/H	ADD	STORE	1	1	1	SAMPLE	B.	H.	E	E	E	E
30	S.R.	S.R.	H.	H.	S.R.	H.	H.	SR/H	SR/H	ADD	STORE	1	1	1	SAMPLE	B.	H.	E	E	E	E
31	S.R.	S.R.	H.	H.	S.R.	H.	H.	SR/H	SR/H	ADD	STORE	1	1	1	H.	B.	H.	E	E	E	E
32	S.R.	S.R.	H.	H.	S.R.	H.	H.	SR/H	R.L.	SUB	STORE	1	1	1	H.	C.	H.	E	E	E	E
33	S.R.	S.R.	H.	H.	S.R.	H.	H.	SR/H	H.	ADD	CLEAR	1	1	1	H.	C.	H.	E	E	E	E
34	S.R.	S.R.	H.	H.	S.R.	H.	H.	SR/H	H.	ADD	STORE	1	1	1	H.	C.	H.	E	E	E	E
35	S.R.	S.R.	H.	H.	S.R.	H.	H.	SR/H	H.	ADD	STORE	1	1	1	H.	C.	H.	E	E	E	E
36	S.R.	S.R.	H.	H.	S.R.	H.	H.	SR/H	H.	ADD	STORE	1	1	1	H.	C.	H.	E	E	E	E
37	S.R.	S.R.	H.	H.	S.R.	H.	H.	SR/H	H.	ADD	STORE	1	1	1	H.	C.	H.	E	E	E	E
38	S.R.	S.R.	H.	H.	S.R.	H.	H.	SR/H	H.	ADD	STORE	1	1	1	H.	C.	H.	E	E	E	E
39	S.R.	S.R.	H.	H.	S.R.	H.	H.	SR/H	H.	ADD	STORE	1	1	1	H.	C.	H.	E	E	E	E
40	S.R.	S.R.	H.	H.	S.R.	H.	H.	R.L.	H.	SUB	STORE	1	1	1	H.	C.	H.	E	E	E	E
41	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	ADD	CLEAR	1	1	1	H.	C.	H.	E	E	E	E
42	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	ADD	STORE	1	1	1	H.	C.	H.	E	E	E	E
43	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	ADD	STORE	1	1	1	H.	C.	H.	E	E	E	E
44	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	ADD	STORE	1	1	1	H.	C.	H.	E	E	E	E
45	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	ADD	STORE	1	1	1	H.	C.	H.	E	E	E	E
46	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	ADD	STORE	1	1	1	H.	C.	H.	E	E	E	E
47	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	ADD	STORE	1	1	1	H.	C.	H.	E	E	E	E
48	S.R.	S.R.	S.R.	R.L.	S.R.	S.R.	S.R.	SR/H	SR/H	SUB	STORE	1	1	1	H.	C.	H.	E	E	E	E
49	S.R.	S.R.	H.	H.	S.R.	H.	H.	SR/H	SR/H	ADD	CLEAR	1	1	1	H.	C.	H.	E	E	E	E
50	S.R.	S.R.	H.	H.	S.R.	H.	H.	SR/H	SR/H	ADD	STORE	1	1	1	H.	C.	H.	E	E	E	E
51	S.R.	S.R.	H.	H.	S.R.	H.	H.	SR/H	SR/H	ADD	STORE	1	1	1	H.	C.	H.	E	E	E	E
52	S.R.	S.R.	H.	H.	S.R.	H.	H.	SR/H	SR/H	ADD	STORE	1	1	1	H.	C.	H.	E	E	E	E
53	S.R.	S.R.	H.	H.	S.R.	H.	H.	SR/H	SR/H	ADD	STORE	1	1	1	H.	C.	H.	E	E	E	E
54	S.R.	S.R.	H.	H.	S.R.	H.	H.	SR/H	SR/H	ADD	STORE	1	1	1	H.	C.	H.	E	E	E	E
55	S.R.	S.R.	H.	H.	S.R.	H.	H.	SR/H	SR/H	ADD	STORE	1	1	1	H.	C.	H.	E	E	E	E
56	S.R.	S.R.	H.	H.	S.R.	H.	H.	SR/H	SR/H	SUB	STORE	1	1	1	H.	C.	LATCH	E	E	E	E
57	S.R.	S.R.	H.	H.	S.R.	H.	H.	SR/H	SR/H	ADD	CLEAR	1	1	1	H.	C.	H.	E	E	E	E
58	S.R.	S.R.	H.	H.	S.R.	H.	H.	SR/H	SR/H	ADD	STORE	1	1	1	H.	C.	H.	E	E	E	E
59	S.R.	S.R.	H.	H.	S.R.	H.	H.	SR/H	SR/H	ADD	STORE	1	1	1	H.	C.	H.	E	E	E	E
60	S.R.	S.R.	H.	H.	S.R.	H.	H.	SR/H	SR/H	ADD	STORE	1	1	1	H.	C.	H.	E	E	E	E
61	S.R.	S.R.	H.	H.	S.R.	H.	H.	SR/H	SR/H	ADD	STORE	1	1	1	H.	C.	H.	E	E	E	E
62	S.R.	S.R.	H.	H.	S.R.	H.	H.	SR/H	SR/H	ADD	STORE	1	1	1	H.	C.	H.	E	E	E	E
63	S.R.	S.R.	H.	H.	S.R.	H.	H.	SR/H	SR/H	ADD	STORE	1	1	1	H.	C.	H.	E	E	E	E
64	S.R.	S.R.	H.	H.	S.R.	H.	H.	R.L.	SR/H	SUB	STORE	1	1	1	H.	C.	H.	E	E	E	E
65	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	SR/H	ADD	CLEAR	1	1	1	H.	C.	H.	E	E	E	E
66	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	SR/H	ADD	STORE	1	1	1	H.	C.	H.	E	E	E	E
67	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	SR/H	ADD	STORE	1	1	1	H.	C.	H.	E	E	E	E
68	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	SR/H	ADD	STORE	1	1	1	H.	C.	H.	E	E	E	E
69	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	SR/H	ADD	STORE	1	1	1	H.	C.	H.	E	E	E	E
70	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	SR/H	ADD	STORE	1	1	1	H.	C.	H.	E	E	E	E
71	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	S.R.	SR/H	ADD	STORE	1	1	1	H.	C.	H.	E	E	E	E

S.R. = SHIFT RIGHT S.R/H = SHIFT RIGHT OR HOLD B. = BLANK E = ENABLE
H. = HOLD P.L. = PARALLEL LOAD C. = CONVERT E = NOT ENABLE

Table 5.2. Signals Required to Control the Computational Section.

ASM'S CURRENT STATE	ASM 0 to SI of SR1	ASM 1 to SI of SR4	ASM 2 to SI of SR8	ASM 3 to SI of SR9	ASM 4 to SO of SR 3,4&7	ASM 5 to SO of SR9	ASM 6 to additive and reset of latch B	ASM 7 CE pin to DAC	ASM 8 to A6 of functional rom	ASM 9 to A5 of functional rom	ASM 10 to A4 of functional rom	ASM 11 to A3 of functional rom	ASM 12 to SC of ADC	ASM 13 to SH of sample and hold
0	0	0	0	0	0	0	1	1	0	0	0	0	1	0
1	0	0	0	0	0	0	1	1	0	0	0	0	1	0
2	0	0	0	0	0	0	1	1	0	0	0	0	1	0
3	0	0	0	0	0	0	1	1	0	0	0	0	1	0
4	0	0	0	0	0	0	1	1	0	0	0	0	1	0
5	0	0	0	0	0	0	1	1	0	0	0	0	1	0
6	0	0	0	0	0	0	1	1	0	0	0	0	1	0
7	0	0	0	1	0	1	0	1	0	0	0	0	1	0
8	0	0	0	0	0	0	1	1	0	0	0	0	1	0
9	0	0	0	0	0	0	1	1	0	0	0	1	1	0
10	0	0	0	0	0	0	1	1	0	0	0	1	1	0
11	0	0	0	0	0	0	1	1	0	0	0	1	1	0
12	0	0	0	0	0	0	1	1	0	0	0	1	1	0
13	0	0	0	0	0	0	1	1	0	0	0	1	1	0
14	0	0	0	0	0	0	1	1	0	0	0	1	1	0
15	0	0	1	0	0	0	0	1	0	0	0	1	1	0
16	0	0	0	0	1	1	1	1	0	0	1	0	1	0
17	0	0	0	0	1	1	1	1	0	0	1	0	1	0
18	0	0	0	0	1	1	1	1	0	0	1	0	1	0
19	0	0	0	0	1	1	1	1	0	0	1	0	1	0
20	0	0	0	0	1	1	1	1	0	0	1	0	1	0
21	0	0	0	0	1	1	1	1	0	0	1	0	1	0
22	0	0	0	0	1	1	1	1	0	0	1	0	1	0
23	0	1	0	0	1	1	0	1	0	0	1	0	1	0
24	0	0	0	0	0	0	1	1	0	0	1	1	1	0
25	0	0	0	0	0	0	1	1	0	0	1	1	1	0
26	0	0	0	0	0	0	1	1	0	0	1	1	1	0
27	0	0	0	0	0	0	1	1	0	0	1	1	1	0
28	0	0	0	0	0	0	1	1	0	0	1	1	1	0
29	0	0	0	0	0	0	1	1	0	0	1	1	1	0
30	0	0	0	0	0	0	1	1	0	0	1	1	1	1
31	0	0	0	1	0	1	0	1	0	0	1	1	0	1
32	0	0	0	0	0	0	1	1	0	1	0	0	0	1
33	0	0	0	0	0	0	1	1	0	1	0	0	0	1
34	0	0	0	0	0	0	1	1	0	1	0	0	0	1
35	0	0	0	0	0	0	1	1	0	1	0	0	0	1
36	0	0	0	0	0	0	1	1	0	1	0	0	0	1
37	0	0	0	0	0	0	1	1	0	1	0	0	0	1
38	0	0	0	0	0	0	1	1	0	1	0	0	0	1
39	0	0	1	0	0	0	0	1	0	1	0	0	0	1
40	0	0	0	0	1	1	1	1	0	1	0	1	0	1
41	0	0	0	0	1	1	1	1	0	1	0	1	0	1
42	0	0	0	0	1	1	1	1	0	1	0	1	0	1
43	0	0	0	0	1	1	1	1	0	1	0	1	0	1
44	0	0	0	0	1	1	1	1	0	1	0	1	0	1
45	0	0	0	0	1	1	1	1	0	1	0	1	0	1
46	0	0	0	0	1	1	1	1	0	1	0	1	0	1
47	0	1	0	0	1	1	0	1	0	1	0	1	0	1
48	0	0	0	0	0	0	1	1	0	1	1	0	0	1
49	0	0	0	0	0	0	1	1	0	1	1	0	0	1
50	0	0	0	0	0	0	1	1	0	1	1	0	0	1
51	0	0	0	0	0	0	1	1	0	1	1	0	0	1
52	0	0	0	0	0	0	1	1	0	1	1	0	0	1
53	0	0	0	0	0	0	1	1	0	1	1	0	0	1
54	0	0	0	0	0	0	1	1	0	1	1	0	0	1
55	0	0	0	0	0	0	0	0	0	1	1	0	0	1
56	0	0	0	0	0	0	1	1	0	1	1	1	0	1
57	0	0	0	0	0	0	1	1	0	1	1	1	0	1
58	0	0	0	0	0	0	1	1	0	1	1	1	0	1
59	0	0	0	0	0	0	1	1	0	1	1	1	0	1
60	0	1	0	0	0	0	1	1	0	1	1	1	0	1
61	0	0	0	0	0	0	1	1	0	1	1	1	0	1
62	0	0	0	0	0	0	1	1	0	1	1	1	0	1
63	0	0	1	0	0	0	0	1	0	1	1	1	0	1
64	0	0	0	0	1	0	1	1	1	0	0	0	0	1
65	0	0	0	0	1	0	1	1	1	0	0	0	0	1
66	0	0	0	0	1	0	1	1	1	0	0	0	0	1
67	0	0	0	0	1	0	1	1	1	0	0	0	0	1
68	0	0	0	0	1	0	1	1	1	0	0	0	0	1
69	0	0	0	0	1	0	1	1	1	0	0	0	0	1
70	0	0	0	0	1	0	1	1	1	0	0	0	0	1
71	1	1	0	0	1	0	0	1	1	0	0	0	0	1

Table 5.3. Required ASM Signals.

CURRENT STATE	ASM 13	ASM 12	ASM 11	ASM 10	ASM 9	ASM 8	ASM 7	ASM 6	ASM 5	ASM 4	ASM 3	ASM 2	ASM 1	ASM 0	Y_0, MSB Next State	Y_1, MSB Next State	Y_2, MSB Next State	Y_3, 2nd LSB Next State	Y_4, 2nd LSB Next State	Y_5, 1st LSB Next State	Y_6, LSB Next State
0	0	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1
1	0	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0
2	0	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	1
3	0	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0
4	0	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1
5	0	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1	0
6	0	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1
7	0	1	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	0	0
8	0	1	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1
9	0	1	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1	0
10	0	1	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1
11	0	1	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0
12	0	1	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1
13	0	1	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1	0
14	0	1	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1
15	0	1	1	0	0	0	1	0	0	0	0	0	1	0	0	0	0	1	0	0	0
16	0	1	0	1	0	0	1	1	1	1	1	1	0	0	0	0	0	0	1	0	1
17	0	1	0	1	0	0	1	1	1	1	1	1	0	0	0	0	0	1	0	0	0
18	0	1	0	1	0	0	1	1	1	1	1	1	0	0	0	0	0	1	0	1	1
19	0	1	0	1	0	0	1	1	1	1	1	1	0	0	0	0	0	1	0	0	1
20	0	1	0	1	0	0	1	1	1	1	1	1	0	0	0	0	0	1	0	1	1
21	0	1	0	1	0	0	1	1	1	1	1	1	0	0	0	0	0	1	0	1	0
22	0	1	0	1	0	0	1	1	1	1	1	1	0	0	0	0	0	1	0	1	1
23	0	1	0	1	0	0	1	0	1	1	1	1	0	0	0	0	1	0	0	0	0
24	0	1	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1
25	0	1	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	1	0
26	0	1	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	1	1
27	0	1	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	1	1	0	0
28	0	1	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	1	1	1	1
29	0	1	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	1	1	1	0
30	1	1	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	1	1	1	1
31	1	0	1	1	0	0	1	0	1	0	1	0	0	0	0	0	1	0	0	0	0
32	1	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0	1
33	1	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0
34	1	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0	1
35	1	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	1	0	0	1	0
36	1	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0	1
37	1	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	1	0	0	1	0
38	1	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	1	0	0	1	1
39	1	0	0	0	1	0	1	0	0	0	0	0	1	0	0	0	1	0	0	0	0
40	1	0	1	0	1	0	1	1	1	1	1	1	0	0	0	0	1	0	0	0	1
41	1	0	1	0	1	0	1	1	1	1	1	1	0	0	0	0	1	0	0	1	0
42	1	0	1	0	1	0	1	1	1	1	1	1	0	0	0	0	1	0	0	1	1
43	1	0	1	0	1	0	1	1	1	1	1	1	0	0	0	0	1	0	1	0	0
44	1	0	1	0	1	0	1	1	1	1	1	1	0	0	0	0	1	0	1	0	1
45	1	0	1	0	1	0	1	1	1	1	1	1	0	0	0	0	1	0	1	1	0
46	1	0	1	0	1	0	1	1	1	1	1	1	0	0	0	0	1	0	1	1	1
47	1	0	1	0	1	0	1	0	1	1	1	1	0	0	1	0	1	1	0	0	0
48	1	0	0	1	1	0	1	1	0	0	0	0	0	0	0	0	1	1	0	0	1
49	1	0	0	1	1	0	1	1	0	0	0	0	0	0	0	0	1	1	0	1	0
50	1	0	0	1	1	0	1	1	0	0	0	0	0	0	0	0	1	1	0	1	1
51	1	0	0	1	1	0	1	1	0	0	0	0	0	0	0	0	1	1	0	0	0
52	1	0	0	1	1	0	1	1	0	0	0	0	0	0	0	0	1	1	0	1	1
53	1	0	0	1	1	0	1	1	0	0	0	0	0	0	0	0	1	1	1	0	0
54	1	0	0	1	1	0	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1
55	1	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	1	1	1	0	0
56	1	0	1	1	1	0	1	1	1	1	0	0	0	0	0	0	1	1	1	0	1
57	1	0	1	1	1	0	1	1	1	1	0	0	0	0	0	0	1	1	1	0	1
58	1	0	1	1	1	0	1	1	1	1	0	0	0	0	0	0	1	1	1	1	0
59	1	0	1	1	1	0	1	1	1	1	0	0	0	0	0	0	1	1	1	1	1
60	1	0	1	1	1	0	1	1	1	1	0	0	0	0	0	0	1	1	1	1	0
61	1	0	1	1	1	0	1	1	1	1	0	0	0	0	0	0	1	1	1	1	1
62	1	0	1	1	1	0	1	1	1	1	0	0	0	0	0	0	1	1	1	1	1
63	1	0	1	1	1	0	1	1	1	1	0	0	0	0	0	0	1	1	1	1	0
64	1	0	1	0	1	0	1	1	1	1	0	0	0	0	0	0	1	1	1	1	1
65	1	0	1	0	1	0	1	1	1	1	0	0	0	0	0	0	1	1	1	1	0
66	1	0	1	0	1	0	1	1	1	1	0	0	0	0	0	0	1	1	1	1	1
67	1	0	1	0	1	0	1	1	1	1	0	0	0	0	0	0	1	1	1	1	0
68	1	0	1	0	1	0	1	1	1	1	0	0	0	0	0	0	1	1	1	1	1
69	1	0	1	0	1	0	1	1	1	1	0	0	0	0	0	0	1	1	1	1	0
70	1	0	0	0	1	0	1	1	1	1	0	0	0	0	0	0	1	1	1	1	1
71	1	0	0	0	1	0	1	1	1	1	0	0	0	0	0	0	1	1	1	1	0

EPROM's used for the ASM. The tables for these EPROM's are shown in Tables 5.4-5.6.

Functional values for the distributed arithmetic approach were stored in the function memory. An equation for the desired function value was developed in Chapter IV. This equation was:

$$f_1(j) = a_1v_1(-j) + a_2v_2(-j) + a_3v_3(-j) \tag{4.8}$$

where a_i was the transmittance of the ith branch and $v_i(-j)$ was the -jth bit of the value of the ith node. Now consider node 12 of Figure 4.6 (pg. 50). The 3 transmittances into this node are b_{11}, a_{112}, and a_{111}. The corresponding nodes feeding node 12 are node 11, node 15, and node 14. Therefore the jth functional value would be:

$$f_{12}(j) = b_{11}v_{11}(-j) + a_{112}v_{15}(-j) + a_{111}v_{14}(-j). \tag{5.2}$$

There were, however, 8 possible combinations of $v_{11}(-j)$, $v_{15}(-j)$ and $v_{14}(-j)$ in binary representation. One of these was $v_{11}(-j) = 0$, $v_{15}(-j) = 0$ and $v_{14}(-j) = 0$. In this case,

$$f_{12}(j) = 0. \tag{5.3}$$

Another possibility was $v_{11}(-j) = 1$, $v_{15}(-j) = 0$ and $v_{14}(-j) = 0$ which would give:

$$f_{12}(j) = b_{11}. \tag{5.4}$$

A list of the possible combinations of $v_{11}(-j)$, $v_{15}(-j)$ and $v_{14}(-j)$ along with the corresponding value for $f_{12}(j)$ is given in Table 5.7. Here the values for b_{11}, a_{112} and a_{111} in Figure 3.2 (pg.37) have been used to determine the value of $f_{12}(j)$. In a similar manner, functional values for each node to be calculated in the filter were generated. These values are shown in Table 5.8. The results were then converted to 2's complement representation corresponding to the format of the

Table 5.4. EPROM Table 1 for 20% Filter.

ASM STATE	EPROM CONTENTS	ASM STATE	EPROM CONTENTS	ASM STATE	EPROM CONTENTS
00	01	18	19	30	31
01	02	19	1A	31	32
02	03	1A	1B	32	33
03	04	1B	1C	33	34
04	05	1C	1D	34	35
05	06	1D	1E	35	36
06	07	1E	1F	36	37
07	08	1F	20	37	38
08	09	20	21	38	39
09	0A	21	22	39	3A
0A	0B	22	23	3A	3B
0B	0C	23	24	3B	3C
0C	0D	24	25	3C	3D
0D	0E	25	26	3D	3E
0E	0F	26	27	3E	3F
0F	10	27	28	3F	40
10	11	28	29	40	41
11	12	29	2A	41	42
12	13	2A	2B	42	43
13	14	2B	2C	43	44
14	15	2C	2D	44	45
15	16	2D	2E	45	46
16	17	2E	2F	46	47
17	18	2F	30	47	30

Table 5.5. EPROM Table 2 for 20% Filter.

ASM STATE	EPROM CONTENTS	ASM STATE	EPROM CONTENTS	ASM STATE	EPROM CONTENTS
00	60	18	60	30	60
01	60	19	60	31	60
02	60	1A	60	32	60
03	60	1B	60	33	60
04	60	1C	60	34	60
05	60	1D	60	35	60
06	60	1E	60	36	60
07	54	1F	54	37	00
08	60	20	60	38	60
09	60	21	60	39	60
0A	60	22	60	3A	60
0B	60	23	60	3B	60
0C	60	24	60	3C	60
0D	60	25	60	3D	60
0E	60	26	60	3E	60
0F	42	27	42	3F	42
10	78	28	78	40	E8
11	78	29	78	41	E8
12	78	2A	78	42	E8
13	78	2B	78	43	E8
14	78	2C	78	44	E8
15	78	2D	78	45	E8
16	78	2E	78	46	E8
17	59	2F	59	47	C9

Table 5.6. EPROM Table 3 for 20% Filter.

ASM STATE	EPROM CONTENTS	ASM STATE	EPROM CONTENTS	ASM STATE	EPROM CONTENTS
00	08	18	0E	30	13
01	08	19	0E	31	13
02	08	1A	0E	32	13
03	08	1B	0E	33	13
04	08	1C	0E	34	13
05	08	1D	0E	35	13
06	08	1E	1E	36	13
07	08	1F	16	37	13
08	0C	20	11	38	17
09	0C	21	11	39	17
0A	0C	22	11	3A	17
0B	0C	23	11	3B	17
0C	0C	24	11	3C	17
0D	0C	25	11	3D	17
0E	0C	26	11	3E	17
0F	0C	27	11	3F	17
10	0A	28	15	40	10
11	0A	29	15	41	10
12	0A	2A	15	42	10
13	0A	2B	15	43	10
14	0A	2C	15	44	10
15	0A	2D	15	45	10
16	0A	2E	15	46	10
17	0A	2F	15	47	10

Table 5.7. Partial Products for Calculating Node 12.

$v_{11}(-J)$	$v_{14}(-J)$	$v_{15}(-J)$	$f_{12}(j)$
0	0	0	0
0	0	1	a_{112}
0	1	0	a_{111}
0	1	1	$a_{112} + a_{111}$
1	0	0	b_{11}
1	0	1	$b_{11} + a_{112}$
1	1	0	$b_{11} + a_{111}$
1	1	1	$b_{11} + a_{111} + a_{112}$

Table 5.8. Partial Products for Calculating all Nodes of the 20% Filter.

FUNCTION	POSSIBLE FUNCTION VALUE	FUNCTION	POSSIBLE FUNCTION VALUE	FUNCTION	POSSIBLE FUNCTION VALUE
$F_{11}(j)$	0	$F_{21}(j)$	0	$F_{31}(j)$	0
$F_{11}(j)$	0.0698	$F_{21}(j)$	0.0698	$F_{31}(j)$	0.0698
$F_{11}(j)$	0.1759	$F_{21}(j)$	0.1262	$F_{31}(j)$	0.1225
$F_{11}(j)$	0.2457	$F_{21}(j)$	0.1960	$F_{31}(j)$	0.1924
$F_{11}(j)$	0.5681	$F_{21}(j)$	0.3899	$F_{31}(j)$	0.3564
$F_{11}(j)$	0.6380	$F_{21}(j)$	0.4597	$F_{31}(j)$	0.4262
$F_{11}(j)$	0.7440	$F_{21}(j)$	0.5161	$F_{31}(j)$	0.4790
$F_{11}(j)$	0.8138	$F_{21}(j)$	0.5859	$F_{31}(j)$	0.5488
$F_{12}(j)$	0	$F_{22}(j)$	0	$F_{32}(j)$	0
$F_{12}(j)$	0.6758	$F_{22}(j)$	0.8698	$F_{32}(j)$	0.8639
$F_{12}(j)$	0.7022	$F_{22}(j)$	0.5715	$F_{32}(j)$	0.5160
$F_{12}(j)$	1.3780	$F_{22}(j)$	1.4413	$F_{32}(j)$	1.3799
$F_{12}(j)$	-0.4608	$F_{22}(j)$	-0.1695	$F_{32}(j)$	-0.0181
$F_{12}(j)$	0.2150	$F_{22}(j)$	0.7003	$F_{32}(j)$	0.8458
$F_{12}(j)$	0.2414	$F_{22}(j)$	0.4020	$F_{32}(j)$	0.4980
$F_{12}(j)$	0.9172	$F_{22}(j)$	1.2718	$F_{32}(j)$	1.3619
$F_{13}(j)$	0	$F_{23}(j)$	0	$F_{33}(j)$	0
$F_{13}(j)$	0.2092	$F_{23}(j)$	0.2815	$F_{33}(j)$	0.2970
$F_{13}(j)$	0.5270	$F_{23}(j)$	0.5085	$F_{33}(j)$	0.5213
$F_{13}(j)$	0.7362	$F_{23}(j)$	0.7900	$F_{33}(j)$	0.8183
$F_{13}(j)$	0.7022	$F_{23}(j)$	0.5715	$F_{33}(j)$	0.5160
$F_{13}(j)$	0.9114	$F_{23}(j)$	0.8529	$F_{33}(j)$	0.8131
$F_{13}(j)$	1.2292	$F_{23}(j)$	1.0800	$F_{33}(j)$	1.0373
$F_{13}(j)$	1.4384	$F_{23}(j)$	1.3615	$F_{33}(j)$	1.3343

filter ALU (ie. XXX.XXXXXXXXX) as shown in Table 5.9. The 2's complement representation was then converted to hexadecimal representation to produce the EPROM tables for function memory shown in Table 5.10 and 5.11.

Implementation of the 2% of Nyquist Frequency Bandwidth Filter

Hardware. The hardware realization of the 20% filter, with minor modifications, was used to implement the 2% filter. The modifications required were: 1) an increase in the shift register width to 12 bits and 2) an increase in the ALU width to 16 bits. These modifications, which occurred in the computational section, are shown in Figure 5.8. The clock, input, output, and control sections were identical to those of the 20% filter.

To increase the width of the shift registers, 4 bit shift registers were cascaded with the 74LS299's. The 4 bit shift registers chosen for this purpose were the 74LS194's. The mode select inputs for the 74LS194 and 74LS299 were identical; therefore, the control lines used to control the 74LS299's were used to control the 74LS194's. The 74LS194's, however, did not need buffers since their inputs and outputs are separate pins rather than multiplexed pins.

Development of a 12 bit filter implies that a 12 bit ADC should be used to sample the data. An 8 bit ADC, however, was all that was available. Therefore, the 8 bit ADC was used to establish the upper 8 bits of the digital signal input. The inputs of the 74LS194 which comprised the lower 4 bits of SR1 were tied to ground. Thus, when SR1 parallel loaded the signal input, the upper 8 bits were established by

Table 5.9. Two's Complement Representation of Partial Products.

FUNCTION	FUNCTION EPROM LOCATION	BIT 8	BIT 7	BIT 6	BIT -1	BIT -2	BIT -3	BIT -4	BIT -5	BIT -6	BIT -7	BIT -8	BIT -9
$F_{00}(i)$	00	0	0	0	0	0	0	0	0	0	0	0	0
$F_{01}(i)$	01	0	0	0	0	0	0	1	0	0	0	0	0
$F_{02}(i)$	02	0	0	0	0	0	1	0	1	1	0	1	0
$F_{03}(i)$	03	0	0	0	0	0	1	1	1	1	1	1	0
$F_{04}(i)$	04	0	0	0	1	0	0	1	0	0	0	1	1
$F_{05}(i)$	05	0	0	0	1	0	1	0	0	0	1	1	1
$F_{06}(i)$	06	0	0	0	1	0	1	1	1	1	1	0	1
$F_{07}(i)$	07	0	0	0	1	1	0	1	0	0	0	0	1
$F_{08}(i)$	08	0	0	0	0	0	0	0	0	0	0	0	0
$F_{09}(i)$	09	0	0	0	1	0	1	0	1	1	0	1	0
$F_{0A}(i)$	0A	0	0	0	1	0	1	1	0	1	0	0	0
$F_{0B}(i)$	0B	0	0	1	0	1	1	0	0	0	0	1	0
$F_{0C}(i)$	0C	1	1	1	1	0	0	0	1	0	1	0	0
$F_{0D}(i)$	0D	0	0	0	0	0	1	1	0	1	1	1	0
$F_{0E}(i)$	0E	0	0	0	0	0	1	1	1	1	1	0	0
$F_{0F}(i)$	0F	0	0	0	1	1	1	0	1	0	0	0	0
$F_{10}(i)$	10	0	0	0	0	0	0	0	0	0	0	0	0
$F_{11}(i)$	11	0	0	0	0	0	0	1	1	0	1	1	0
$F_{12}(i)$	12	0	0	0	1	0	0	0	0	1	1	1	0
$F_{13}(i)$	13	0	0	0	1	0	1	1	1	1	0	0	1
$F_{14}(i)$	14	0	0	0	1	0	1	1	0	1	0	0	0
$F_{15}(i)$	15	0	0	0	1	1	1	0	1	0	0	1	1
$F_{16}(i)$	16	0	0	1	0	0	1	1	1	0	1	0	1
$F_{17}(i)$	17	0	0	1	0	1	1	1	0	0	0	0	0
$F_{18}(i)$	18	0	0	0	0	0	0	0	0	0	0	0	0
$F_{19}(i)$	19	0	0	0	0	0	0	1	0	0	1	0	0
$F_{1A}(i)$	1A	0	0	0	0	0	1	0	0	0	0	0	0
$F_{1B}(i)$	1B	0	0	0	0	0	1	1	0	0	1	0	0
$F_{1C}(i)$	1C	0	0	0	0	1	0	1	0	1	0	0	0
$F_{1D}(i)$	1D	0	0	0	0	1	0	1	1	0	0	1	0
$F_{1E}(i)$	1E	0	0	0	1	0	0	0	1	0	0	1	0
$F_{1F}(i)$	1F	0	0	0	1	0	0	1	0	0	0	1	0
$F_{20}(i)$	20	0	0	0	0	0	0	0	0	0	0	0	0
$F_{21}(i)$	21	0	0	0	1	1	0	1	1	1	1	0	1
$F_{22}(i)$	22	0	0	0	1	0	0	1	0	0	1	0	1
$F_{23}(i)$	23	0	0	1	0	1	1	1	0	0	0	1	0
$F_{24}(i)$	24	1	1	1	1	1	0	1	0	1	1	1	1
$F_{25}(i)$	25	0	0	0	1	1	1	1	0	0	1	1	1
$F_{26}(i)$	26	0	0	0	0	1	1	0	0	1	1	1	0
$F_{27}(i)$	27	0	0	1	0	1	0	0	0	1	0	1	1
$F_{28}(i)$	28	0	0	0	0	0	0	0	0	0	1	0	0
$F_{29}(i)$	29	0	0	0	0	1	0	0	1	0	0	0	0
$F_{2A}(i)$	2A	0	0	0	1	0	0	0	0	0	0	1	0
$F_{2B}(i)$	2B	0	0	0	1	1	0	0	1	0	1	1	1
$F_{2C}(i)$	2C	0	0	0	1	0	0	1	0	0	0	1	1
$F_{2D}(i)$	2D	0	0	0	1	1	0	1	1	0	1	1	1
$F_{2E}(i)$	2E	0	0	1	0	0	0	1	0	0	1	0	1
$F_{2F}(i)$	2F	0	0	1	0	1	0	1	1	1	0	1	1
$F_{30}(i)$	30	0	0	0	0	0	0	0	0	0	0	0	0
$F_{31}(i)$	31	0	0	0	0	0	0	1	1	1	0	1	1
$F_{32}(i)$	32	0	0	0	0	0	0	1	1	1	1	1	1
$F_{33}(i)$	33	0	0	0	0	0	1	0	0	0	1	1	1
$F_{34}(i)$	34	0	0	0	0	0	1	1	1	1	0	1	1
$F_{35}(i)$	35	0	0	0	0	1	0	1	0	0	1	0	1
$F_{36}(i)$	36	0	0	0	0	1	0	1	1	1	0	0	1
$F_{37}(i)$	37	0	0	0	0	1	1	0	0	1	0	0	1
$F_{38}(i)$	38	0	0	0	0	1	1	0	1	0	0	1	1
$F_{39}(i)$	39	0	0	0	0	1	1	1	0	0	1	0	1
$F_{3A}(i)$	3A	0	0	0	1	0	0	0	0	1	0	1	1
$F_{3B}(i)$	3B	0	0	0	1	0	0	1	1	0	0	0	1
$F_{3C}(i)$	3C	0	0	0	1	0	1	0	1	1	1	1	1
$F_{3D}(i)$	3D	0	0	0	1	0	1	1	0	1	0	1	1
$F_{3E}(i)$	3E	0	0	0	1	0	1	1	1	0	1	0	1
$F_{3F}(i)$	3F	0	0	0	1	1	0	1	0	1	0	1	1
$F_{40}(i)$	40	0	0	0	0	0	0	0	0	0	0	0	0
$F_{41}(i)$	41	0	0	0	0	0	0	1	0	1	1	1	1
$F_{42}(i)$	42	0	0	0	0	1	0	1	1	0	0	1	1
$F_{43}(i)$	43	0	0	0	0	1	0	1	1	1	1	0	1
$F_{44}(i)$	44	0	0	0	0	1	1	0	0	1	0	1	1
$F_{45}(i)$	45	0	0	0	0	1	1	0	1	0	1	0	1
$F_{46}(i)$	46	0	0	0	0	1	1	1	0	1	0	1	1
$F_{47}(i)$	47	0	0	0	0	1	1	1	1	0	1	0	1

Table 5.10. EPROM Table 4 for 20% Filter.

FUNCTION EPROM LOCATION	POSSIBLE FUNCTION VALUE	FUNCTION EROM LOCATION	POSSIBLE FUNCTION VALUE	FUNCTION EROM LOCATION	POSSIBLE FUNCTION VALUE
00	0X	18	0X	30	0X
01	4X	19	4X	31	4X
02	AX	1A	1X	32	FX
03	EX	1B	4X	33	3X
04	3X	1C	8X	34	7X
05	7X	1D	BX	35	AX
06	DX	1E	8X	36	5X
07	1X	1F	CX	37	9X
08	0X	20	0X	38	0X
09	AX	21	DX	39	AX
0A	8X	22	5X	3A	8X
0B	2X	23	2X	3B	3X
0C	4X	24	9X	3C	7X
0D	EX	25	7X	3D	1X
0E	CX	26	EX	3E	FX
0F	6X	27	BX	3F	9X
10	0X	28	0X	40	0X
11	BX	29	0X	41	8X
12	DX	2A	4X	42	BX
13	9X	2B	5X	43	3X
14	8X	2C	5X	44	8X
15	3X	2D	5X	45	0X
16	5X	2E	9X	46	3X
17	0X	2F	9X	47	BX

Table 5.11. EPROM Table 5 for 20% Filter.

FUNCTION EPROM LOCATION	POSSIBLE FUNCTION VALUE	FUNCTION EPROM LOCATION	POSSIBLE FUNCTION VALUE	FUNCTION EPROM LOCATION	POSSIBLE FUNCTION VALUE
00	00	18	00	30	00
01	02	19	02	31	02
02	05	1A	04	32	03
03	07	1B	06	33	06
04	12	1C	0C	34	0B
05	14	1D	0E	35	0D
06	17	1E	10	36	0F
07	1A	1F	12	37	11
08	00	20	00	38	00
09	15	21	1B	39	1B
0A	16	22	12	3A	10
0B	2C	23	2E	3B	2C
0C	F1	24	FA	3C	FF
0D	06	25	16	3D	1B
0E	07	26	0C	3E	0F
0F	1D	27	28	3F	2B
10	00	28	00	40	00
11	06	29	09	41	09
12	10	2A	10	42	10
13	17	2B	19	43	1A
14	16	2C	12	44	10
15	1D	2D	1B	45	1A
16	27	2E	22	46	21
17	2E	2F	2B	47	2A

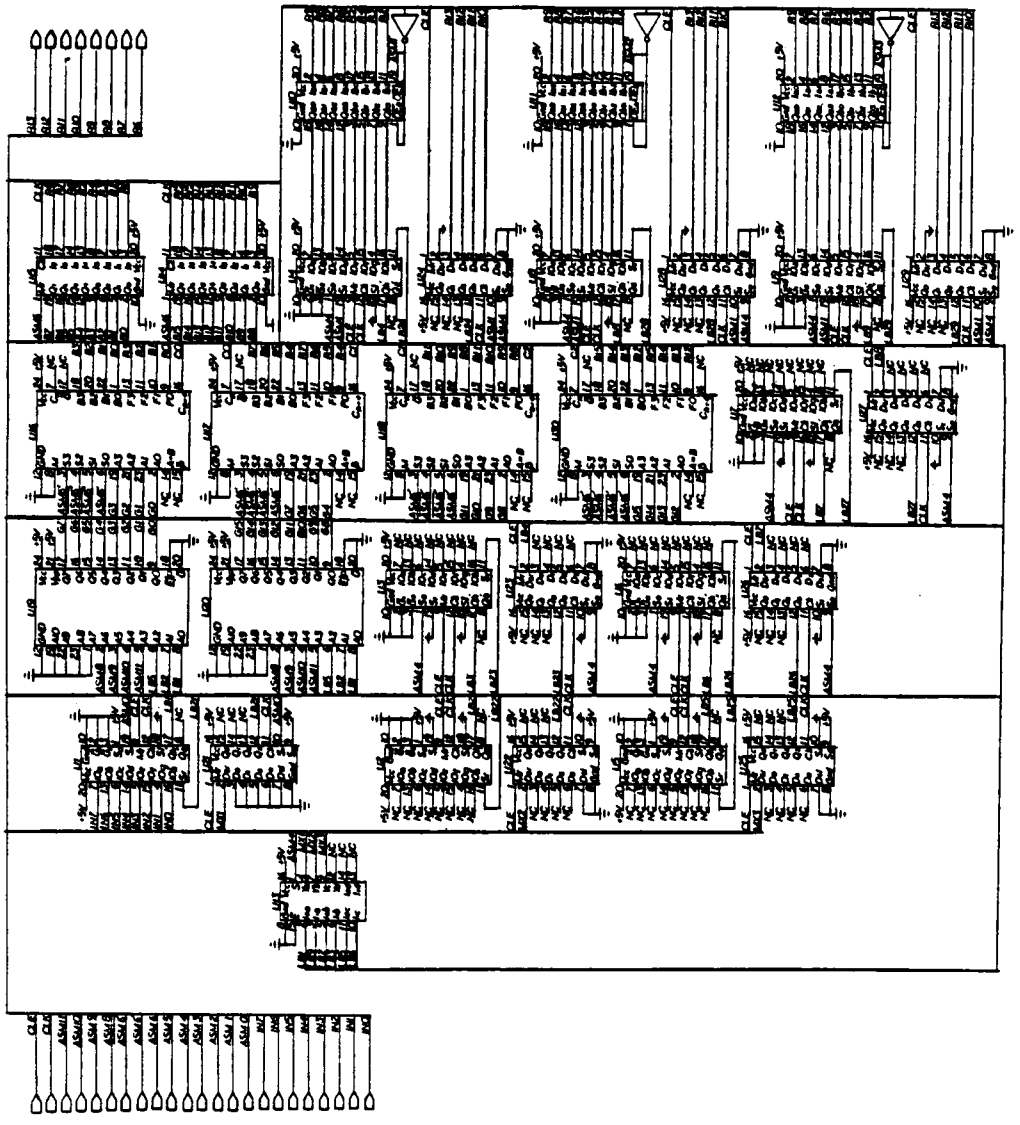


Figure 5.8. Computational section of 2% filter.

the ADC and the lower 4 bits were 0 which mimicked a 12 bit input.

As was the case for the 20% filter, the ALU for the 2% filter had to be wider than the filter variables to handle intermediate variable overflow. Thus an additional 74LS181 was used making the ALU 16 bits wide. A 16 bit wide function memory and B register were also required. The bits of the ALU were configured as XXX.XXXXXXXXXXXX. Functional values, therefore, were stored in memory in this format.

Software. Increasing the shift register widths from 8 bits to 12 bits also increased the clock cycles required to calculate a filter variable from 8 cycles to 12 cycles. Therefore, calculation of the 9 filter variables in the 2% filter required 108 clock cycles. As a result the program for the ASM memory had to be increased by including 4 additional right shifts for SR1 (modules U1 and U21), SR2 (modules U2 and U22) and SR5 (modules U5 and U25) during the calculation of each filter variable. The EPROM tables for the 20% filter, Tables 5.4 - 5.6 (pgs. 84-86), were easily revised to accommodate this requirement. The first 8 entries of these tables correspond to the calculation of the first filter variable. The entries are 086001, 086002, 086003, ..., 086007 and 085408. The first 7 values begin with 0860 and end with the increasing sequence of 01, 02, 03, 04, 05, 06, and 07. The increasing sequence produced the next address of the ASM while the constant 0860 produced the filter control signals. Since 4 additional shifts were required by the 2% filter, 4 additional 0860's were required to be programmed into the ASM memory for the 2% filter. Thus, the first 12 bits became 086001, 086002, ..., 08600B, 08540C. The 0854 control

signal and subsequent control signals slipped 4 addresses to accommodate the additional shifts. This was done for each variable calculation to produce Tables 5.12 - 5.14, the tables for the ASM EPROM's used to control the 2% filter.

EPROM tables for the function memory were created in the same manner as EPROM tables for the 20% filter. Filter constants from Figure 3.1 (pg. 36), however, were used in place of those from Figure 3.2 (pg. 37). The resulting EPROM tables are given in Tables 5.15 - 5.16.

Table 5.12. EPROM Table 1 for 2% Filter.

ASM STATE	EPROM CONTENTS	ASM STATE	EPROM CONTENTS	ASM STATE	EPROM CONTENTS	ASM STATE	EPROM CONTENTS
00	01	1B	1C	36	37	51	52
01	02	1C	1D	37	38	52	53
02	03	1D	1E	38	39	53	54
03	04	1E	1F	39	3A	54	55
04	05	1F	20	3A	3B	55	56
05	06	20	21	3B	3C	56	57
06	07	21	22	3C	3D	57	58
07	08	22	23	3D	3E	58	59
08	09	23	24	3E	3F	59	5A
09	0A	24	25	3F	40	5A	5B
0A	0B	25	26	40	41	5B	5C
0B	0C	26	27	41	42	5C	5D
0C	0D	27	28	42	43	5D	5E
0D	0E	28	29	43	44	5E	5F
0E	0F	29	2A	44	45	5F	60
0F	10	2A	2B	45	46	60	61
10	11	2B	2C	46	47	61	62
11	12	2C	2D	47	48	62	63
12	13	2D	2E	48	49	63	64
13	14	2E	2F	49	4A	64	65
14	15	2F	30	4A	4B	65	66
15	16	30	31	4B	4C	66	67
16	17	31	32	4C	4D	67	68
17	18	32	33	4D	4E	68	69
18	19	33	34	4E	4F	69	6A
19	1A	34	35	4F	50	6A	6B
1A	1B	35	36	50	51	6B	80

Table 5.13. EPROM Table 2 for 2% Filter.

ASM STATE	EPROM CONTENTS	ASM STATE	EPROM CONTENTS	ASM STATE	EPROM CONTENTS	ASM STATE	EPROM CONTENTS
00	60	1B	78	36	60	51	60
01	60	1C	78	37	60	52	60
02	60	1D	78	38	60	53	00
03	60	1E	78	39	60	54	60
04	60	1F	78	3A	60	55	60
05	60	20	78	3B	42	56	60
06	60	21	78	3C	78	57	60
07	60	22	78	3D	78	58	60
08	60	23	59	3E	78	59	60
09	60	24	60	3F	78	5A	60
0A	60	25	60	40	78	5B	60
0B	54	26	60	41	78	5C	60
0C	60	27	60	42	78	5D	60
0D	60	28	60	43	78	5E	60
0E	60	29	60	44	78	5F	42
0F	60	2A	60	45	78	60	E8
10	60	2B	60	46	78	61	E8
11	60	2C	60	47	59	62	E8
12	60	2D	60	48	60	63	E8
13	60	2E	60	49	60	64	E8
14	60	2F	54	4A	60	65	E8
15	60	30	60	4B	60	66	E8
16	60	31	60	4C	60	67	E8
17	42	32	60	4D	60	68	E8
18	78	33	60	4E	60	69	E8
19	78	34	60	4F	60	6A	E8
1A	78	35	60	50	60	6B	C9

Table 5.14. EPROM Table 3 for 2% Filter.

ASM STATE	EPROM CONTENTS	ASM STATE	EPROM CONTENTS	ASM STATE	EPROM CONTENTS	ASM STATE	EPROM CONTENTS
00	08	1B	0A	36	11	51	13
01	08	1C	0A	37	11	52	13
02	08	1D	0A	38	11	53	13
03	08	1E	0A	39	11	54	17
04	08	1F	0A	3A	11	55	17
05	08	20	0A	3B	11	56	17
06	08	21	0A	3C	15	57	17
07	08	22	0A	3D	15	58	17
08	08	23	0A	3E	15	59	17
09	08	24	0E	3F	15	5A	17
0A	08	25	0E	40	15	5B	17
0B	08	26	0E	41	15	5C	17
0C	0C	27	0E	42	15	5D	17
0D	0C	28	0E	43	15	5E	17
0E	0C	29	0E	44	15	5F	17
0F	0C	2A	0E	45	15	60	10
10	0C	2B	0E	46	15	61	10
11	0C	2C	0E	47	15	62	10
12	0C	2D	0E	48	13	63	10
13	0C	2E	1E	49	13	64	10
14	0C	2F	16	4A	13	65	10
15	0C	30	11	4B	13	66	10
16	0C	31	11	4C	13	67	10
17	0C	32	11	4D	13	68	10
18	0A	33	11	4E	13	69	10
19	0A	34	11	4F	13	6A	10
1A	0A	35	11	50	13	6B	10

Table 5.15. EPROM Table 4 for 2% Filter.

FUNCTION EPROM LOCATION	POSSIBLE FUNCTION VALUE	FUNCTION EPROM LOCATION	POSSIBLE FUNCTION VALUE	FUNCTION EPROM LOCATION	POSSIBLE FUNCTION VALUE
00	00	18	C0	30	00
01	08	19	08	31	08
02	3B	1A	E2	32	DC
03	43	1B	EA	33	E3
04	29	1C	6D	34	35
05	31	1D	75	35	3D
06	64	1E	4F	36	11
07	6C	1F	57	37	19
08	00	20	00	38	C0
09	91	21	28	39	34
0A	6E	22	95	3A	1D
0B	FE	23	BC	3B	51
0C	36	24	37	3C	E9
0D	C7	25	5E	3D	1D
0E	A4	26	CB	3E	06
0F	34	27	F2	3F	39
10	C0	28	C0	40	00
11	0D	29	15	41	18
12	0C	2A	5B	42	A2
13	19	2B	70	43	BA
14	6E	2C	95	44	1D
15	7B	2D	A9	45	35
16	79	2E	EF	46	BF
17	86	2F	04	47	D7

Table 5.16. EPROM Table 5 for 2% Filter.

FUNCTION EPROM LOCATION	POSSIBLE FUNCTION VALUE	FUNCTION EPROM LOCATION	POSSIBLE FUNCTION VALUE	FUNCTION EPROM LOCATION	POSSIBLE FUNCTION VALUE
00	00	18	00	30	00
01	00	19	00	31	00
02	01	1A	00	32	00
03	01	1B	00	33	00
04	26	1C	17	34	14
05	26	1D	17	35	14
06	27	1E	18	36	15
07	27	1F	18	37	15
08	00	20	00	38	00
09	01	21	02	39	02
0A	1F	22	1E	3A	1E
0B	20	23	20	3B	20
0C	FE	24	FF	3C	FF
0D	FF	25	01	3D	02
0E	1D	26	1D	3E	1E
0F	1F	27	1F	3F	20
10	00	28	00	40	00
11	00	29	00	41	00
12	02	2A	02	42	02
13	02	2B	02	43	02
14	1F	2C	1E	44	1E
15	1F	2D	1E	45	1E
16	21	2E	20	46	20
17	21	2F	21	47	20

CHAPTER VI

PERFORMANCE OF SIXTH ORDER BUTTERWORTH FILTERS IMPLEMENTED WITH DISTRIBUTED ARITHMETIC

The implemented filters were evaluated in 3 areas. First, each filter's output was compared with the predicted response. By making this comparison, the filters were shown to achieve the desired transfer functions. Second, the cost of the distributed arithmetic implementation was compared with three alternatives. This comparison demonstrated the appeal of the distributed arithmetic approach. Finally, the roundoff error of the filters was estimated.

Filter Performance

Qualitative considerations. For a low pass Butterworth filter, the output due to a sinusoidal input should be a delayed sinusoid of identical frequency. If the frequency of the input is in the passband, the magnitude of the output should approximately equal the magnitude of the input multiplied by the filter gain. Outside the passband the magnitude should roll off at a rate that is dependent upon the filter order. Since the filter gain for the implemented filters was unity, a sinusoid of identical frequency and approximately the same magnitude as the input should appear at the output of the filter if:

1) the frequency of the input is in the passband, and
2) the magnitude of the input does not produce overflow within the filter.

As the frequency of the input increases, the phase shift between the

input and output should increase, nevertheless, no distortion should appear in the output.

The response of the 20% Butterworth filter is shown in Figure 6.1 for several inputs. In this figure the bottom trace corresponds to filter inputs while the top trace corresponds to filter outputs for each photograph except 6.1a in which the positions of the trace are reversed. The vertical deflection is set at 1V/division for 6.1a - 6.1d while the horizontal deflections are 50 msec/division, 5 msec/ division, 0.5 msec/ division and 0.2 msec/division respectively. From these settings, the frequency of the inputs can be calculated to be 5Hz, 50Hz, 500Hz and 900Hz. The frequency of the output can also be seen as identical to the input frequency in all cases. As the frequency increases, however, the output begins to lag the input until at 500Hz the output and input are 180^{o} out of phase. At 900Hz the output lags the input by more than 180^{o}. Additionally, the output is free of distortion as desired.

The magnitude of the output is approximately equal to the magnitude of the input for 5Hz, 50Hz and 500Hz, but it is attenuated at 900Hz. For the 20% filter running at 1MHz, the input was sampled once every 72 clock cycles. Therefore the sampling rate f_s was:

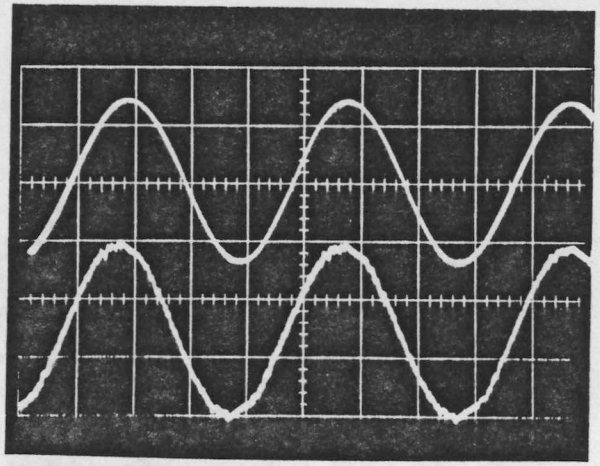
$$f_{s_{20\%}} = 1\text{MHz}/72 \tag{6.1}$$

which produced a Nyquist frequency f_n of:

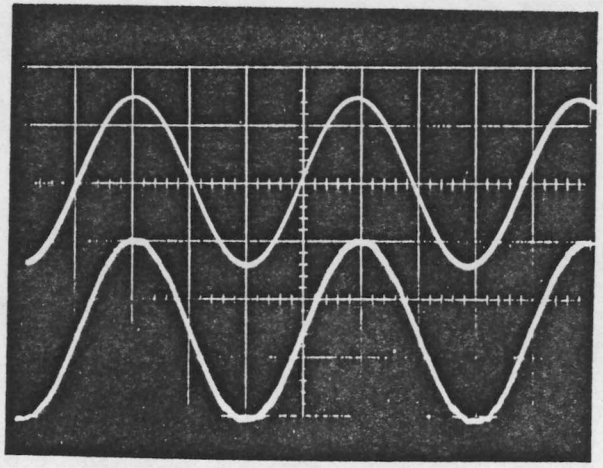
$$f_n = \frac{f_s}{2} = \frac{(1\text{MHz})}{2(72)} = 6944\text{Hz}. \tag{6.2}$$

Since the bandwidth was 20% of the Nyquist frequency, the bandwidth was:

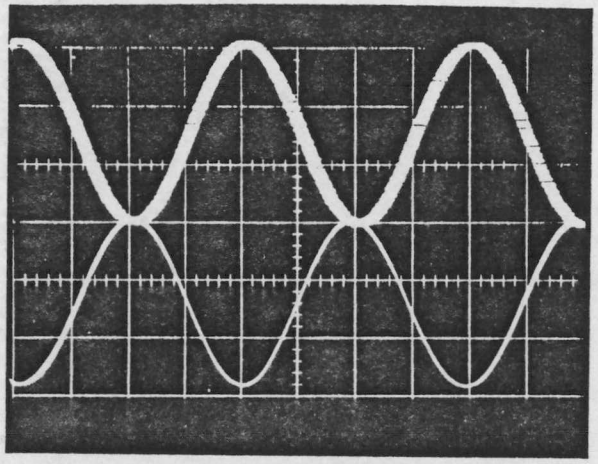
$$\text{B.W.} = .20(6944\text{Hz}) = 1389\text{Hz}. \tag{6.3}$$



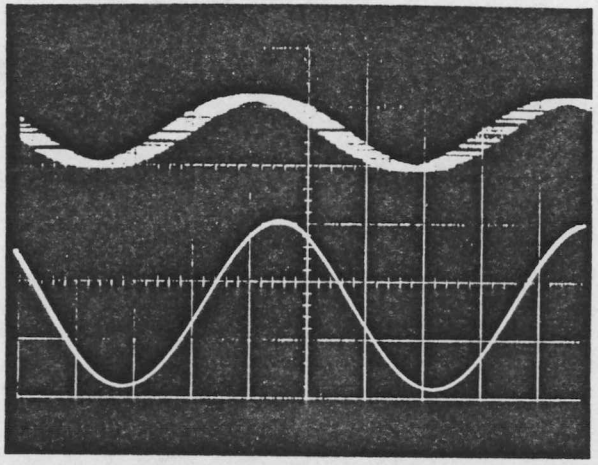
a. Response to a 5Hz input.



b. Response to a 50 Hz input.



c. Response to a 500Hz input.



d. Response to a 900 Hz input.

Figure 6.1. Response of 20% Butterworth filter for 4 inputs.

Therefore, the attenuation was due to the input frequency of 900Hz approaching the filter bandwidth. Since this attenuation was expected, and the output was free of distortion while exhibiting the anticipated magnitude characteristics, the filter was qualitatively performing correctly. Similar results were noted for the 2% filter with the exception that the magnitude began attenuating at a lower frequency since the bandwidth was 92.6Hz.

Quantitative analysis. To insure that the filter implementations were realizing the desired transfer functions, the magnitude and phase of the desired filter transfer function were compared to the measured magnitude and phase for both filters. The desired transfer functions for the 2% and 20% filters were given in Chapter III, pgs. 30,31. The magnitude and phase of these transfer functions were calculated. The results are shown in Table 6.1 for the 20% filter and in Table 6.2 for the 2% filter.

Bode plots of the measured and calculated magnitude and phase are shown in Figure 6.2 for the 20% filter and in Figure 6.3 for the 2% filter. From these plots the measured magnitude and phase can be seen to agree with the calculated values. For the 20% filter, the magnitude agreement was within 4dB while for the 2% filter the agreement was within 1.28dB. Discrepancies between the measured and calculated values were maximum when the slope was large while being negligible when the slope was small. These discrepancies, therefore, were attributed to errors in the frequency of the input that was used to measure the magnitude response. A similar correspondence between the calculated and measured phase supported this conclusion. As in the

Table 6.1. Magnitude and Phase of 20% Filter.

Frequency Hz	/H(w)/ dB	Phase o
5Hz	8.69×10^{-9}	-1.34
10Hz	1.74×10^{-8}	-2.68
20Hz	2.61×10^{-8}	-5.36
30Hz	2.61×10^{-8}	-8.04
40Hz	8.69×10^{-9}	-10.72
50Hz	4.34×10^{-8}	-13.41
60Hz	4.34×10^{-8}	-16.09
70Hz	-1.30×10^{-8}	-18.78
80Hz	1.74×10^{-8}	-21.47
90Hz	5.21×10^{-8}	-24.17
100Hz	4.34×10^{-8}	-26.86
200Hz	-1.37×10^{-7}	-54.11
300Hz	-2.42×10^{-5}	-82.18
400Hz	-7.98×10^{-4}	-111.65
500Hz	-1.23×10^{-2}	-143.46
600Hz	-1.16×10^{-1}	-179.38
700Hz	-7.44×10^{-1}	-222.08
800Hz	-3.06	-270.67
900Hz	-7.56	-314.52
1000Hz	-12.98	-347.31
1100Hz	-18.40	———
1200Hz	-23.57	———
1300Hz	-28.49	———
2000Hz	-58.73	———

Table 6.2. Magnitude and Phase of 2% Filter.

Frequency Hz	/H(w)/ dB	Phase o
5Hz	$-1.08\text{x}10^{-4}$	-11.95
10Hz	$-1.06\text{x}10^{-4}$	-23.93
20Hz	$-9.92\text{x}10^{-5}$	-48.08
30Hz	$-9.40\text{x}10^{-5}$	-72.67
40Hz	$-2.55\text{x}10^{-4}$	-97.99
50Hz	$-2.72\text{x}10^{-3}$	-124.47
60Hz	$-2.37\text{x}10^{-2}$	-152.81
70Hz	$-1.49\text{x}10^{-1}$	-184.18
80Hz	$-6.93\text{x}10^{-1}$	-220.07
90Hz	-2.33	-259.75
100Hz	-5.47	-297.37
110Hz	-9.50	-327.74
120Hz	-13.71	-350.96
130Hz	-17.77	-369.03
140Hz	-21.60	-383.55
150Hz	-25.18	-395.55
160Hz	———	———

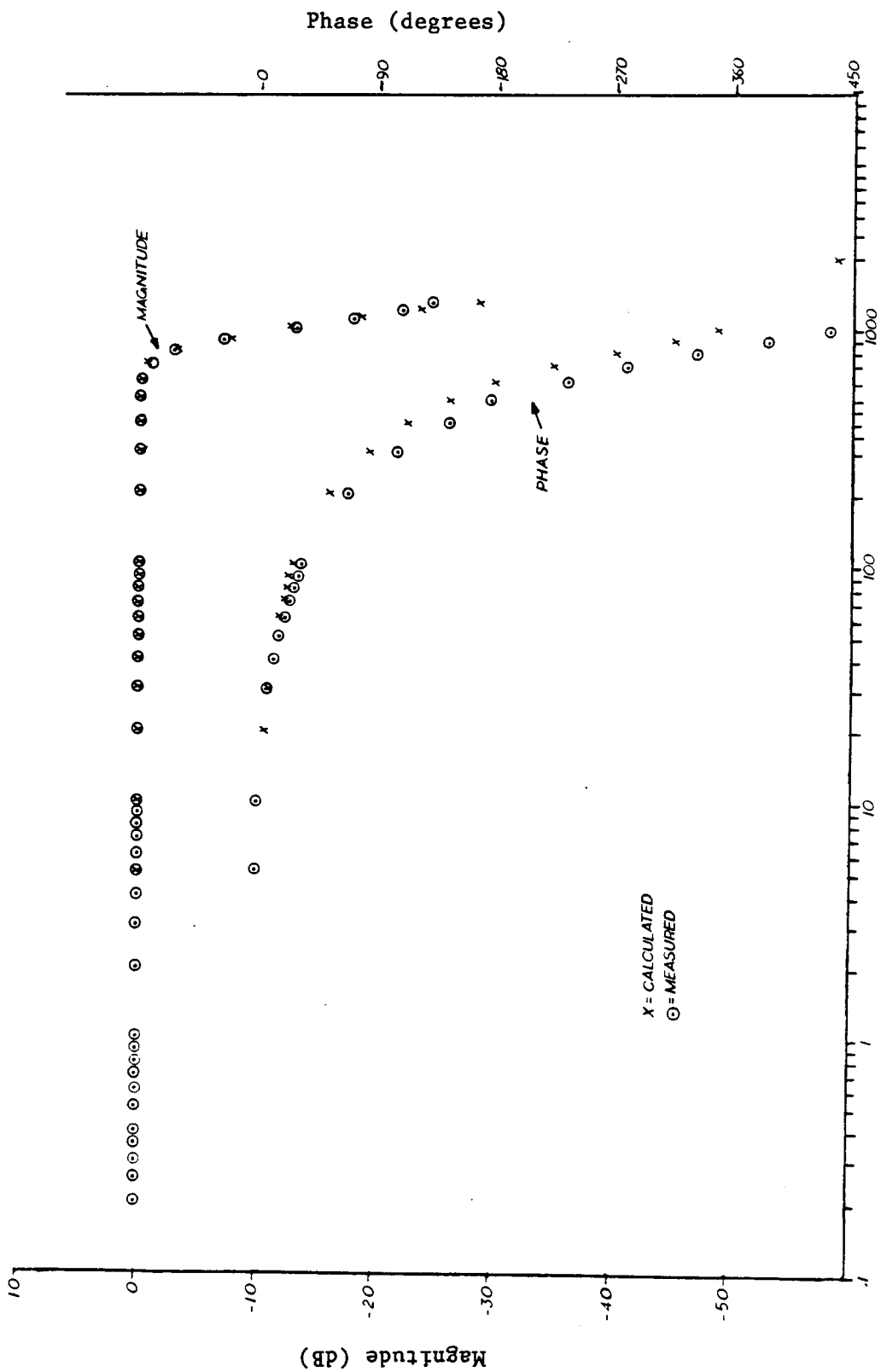


Figure 6.2. Bode plots of magnitude and phase for 20% filter.

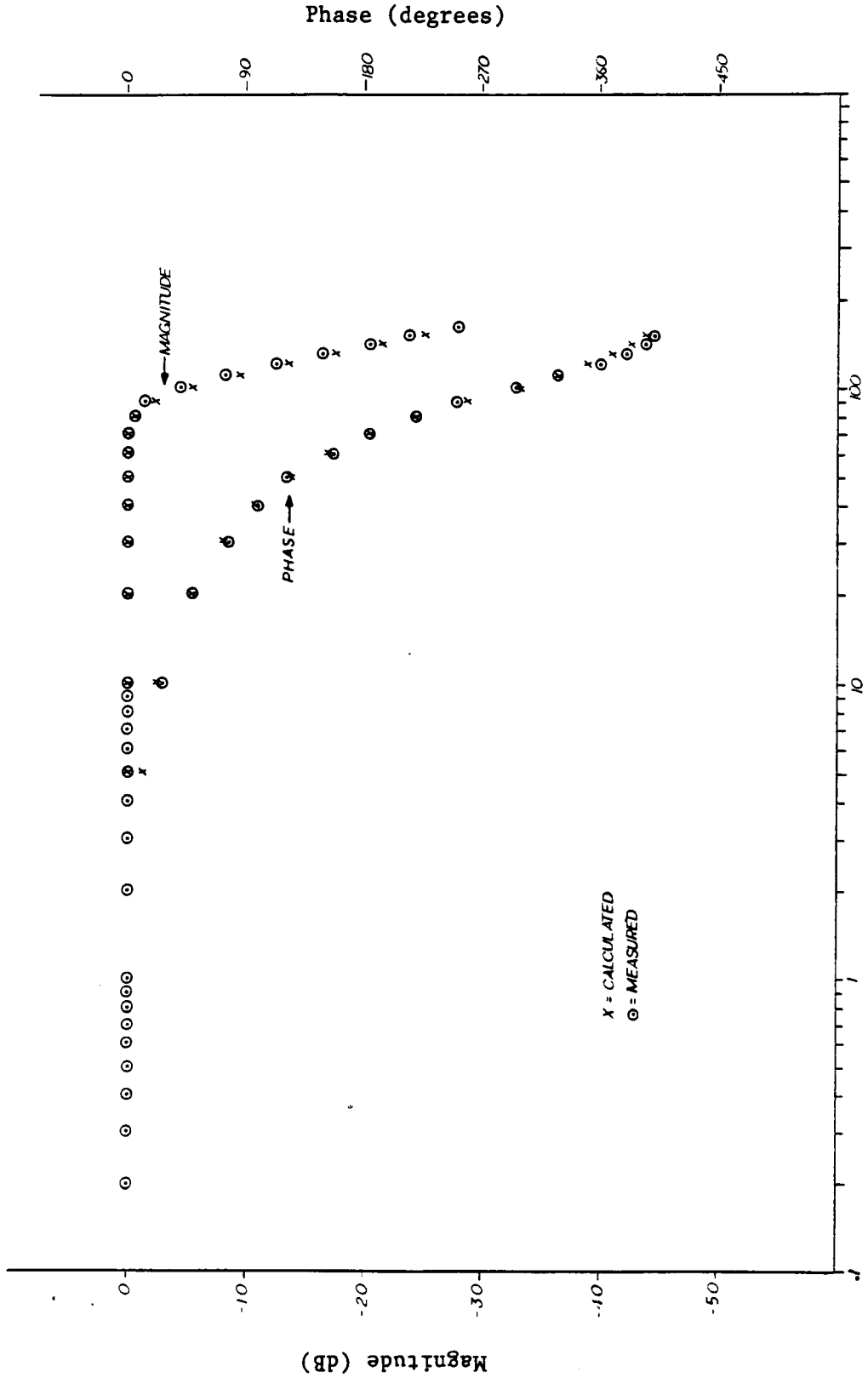


Figure 6.3. Bode plots of magnitude and phase for 2% filter.

magnitude plot, the calculated and measured phases agreed closely when the phase changed slowly with frequency, but began to disagree somewhat when the slope increased. Therefore, the implemented filter realized the desired transfer function within the measurement error of approximately 2%. The agreement between the measured and calculated magnitude and phase responses of the filters substantiates the conclusion that the filters were performing properly.

Comparison of the Distributed Arithmetic Implementation Cost to the Cost of Other Implementations.

One major criteria in choosing a digital filter implementation is its cost. In this section, the cost of several possible implementations for the 20% Butterworth filter are compared to establish the feasibility of the distributed arithmetic implementation. The cost of the different implementations consists of component costs, power supply costs, printed circuit board costs and manufacturing costs. Manufacturing costs, however, were not considered in this comparison since they do not reflect a basic difference in the costs of the various implementations. Also, cost estimates were pursued only for those parts of the filter where a cost difference between the implementations would occur.

Any estimate of cost will vary depending upon a number of factors. One of these factors is the volume in which the product is to be produced. Volumes not only affect the price of each individual module, but may also dictate what component will be used. One example of this is mask programmable ROMs. In the manufacture of mask programmable ROMs, the vendor charges a one time mask fee for producing the mask.

This charge is small if a large number of ROMs are to be purchased. If, however, a small quantity of ROMs is required, the one time charge makes ROMs an expensive choice. For large volume production, therefore, a ROM would be the component of choice. For small volume production, however, an EPROM or PROM would be the component of choice. In the following cost estimates, ROMs were chosen to provide non-volatile memory. Since all of the alternatives required non-volatile memory, the mask charge was assumed to be equal for all alternatives and was, therefore, neglected. The choice of ROMs implies that a large volume of the filter was to be produced. Volume quotes for the components, however, were not available. For this reason manufacturer suggested prices found in trade magazines were used for LSI components. For MSI and SSI components prices from mail order companies were used in the comparison. Though these prices apply to small volume purchases, they are sufficient for this comparison since the purpose is to establish the feasibility of the distributed arithmetic implementation of digital filters.

For the comparison, the distributed arithmetic implementation of the 20% Butterworth filter presented in Chapter V was compared to 3 alternate implementations. All of the alternatives used 8 bit implementations as did the distributed arithmetic implementation. Differences in the implementations occurred in the computational section; therefore, only the computational sections of the alternatives are considered.

The first alternative implementation is presented in Figure 6.4. In this implementation an 8 bit parallel multiplier/accumulator is used

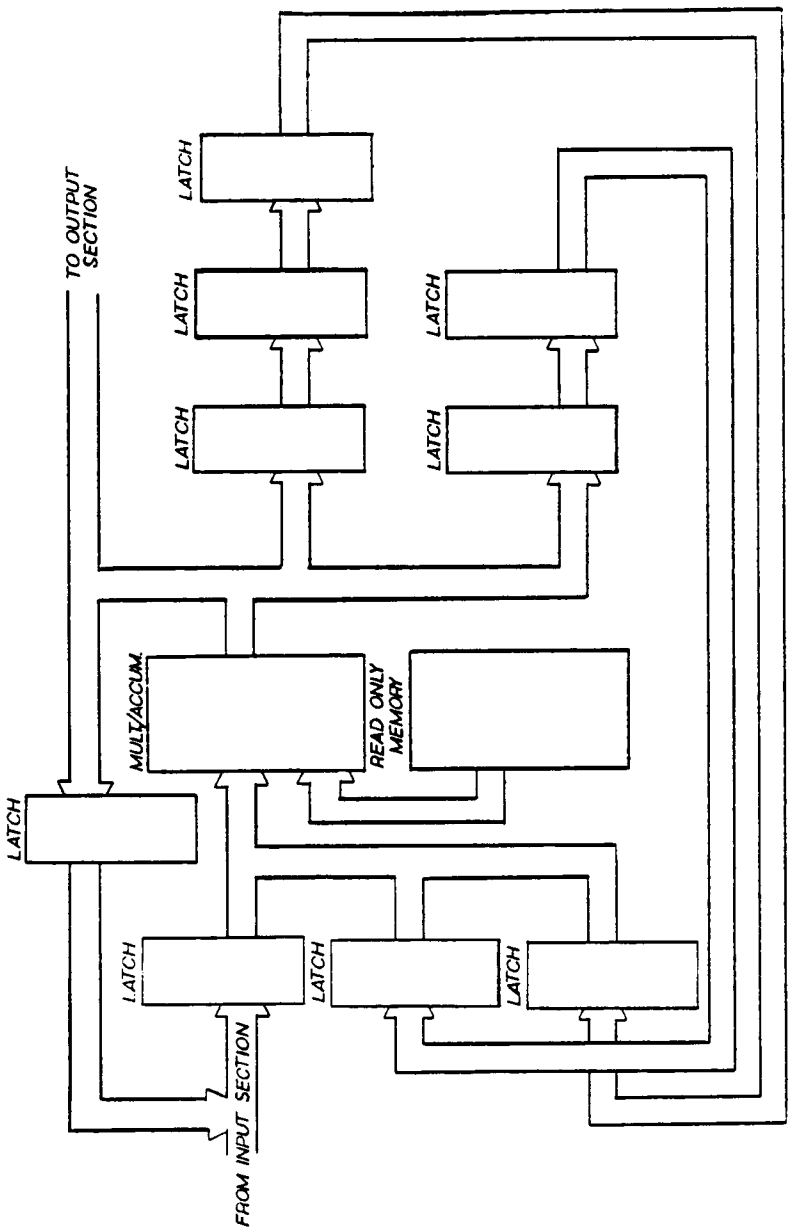


Figure 6.4. Alternative 1.

to calculate the inner product required by the digital filter algorithm. For this comparison, the ADSP-1008 CMOS multiplier/accumulator was chosen. This multiplier is capable of performing a multiply/accumulate function in 150ns maximum while producing a 16 bit product. Since the filter was scaled so that the magnitude of its variables were less than 1, only the 8 LSBs were used as the result.

The 74LS273 latches shown in Figure 6.4 were used to store filter variables and to provide a mechanism for presenting these variables to the multiplier/accumulator in the order required by the filter algorithm. These latches perform a similar function as the 9 shift registers used in the distributed arithmetic implementation; however, instead of addressing partial products stored in ROM, the bits of latches U1, U2 and U5 are fed to one side of the multiplier as a multiplicand. The other multiplicand, the filter coefficients, is provided to the multiplier by a ROM. For the sixth order Butterworth filter, this ROM must be 27 bytes deep.

The second alternative, shown in Figure 6.5, is a variation of alternative 1. As in alternative 1, an 8 bit parallel multiplier/accumulator is used to calculate the inner product for the filter algorithm. The same multiplier/accumulator used in alternative 1 was chosen for alternative 2. As in alternative 1, a ROM was used to provide the filter coefficient multiplicand to one side of the multiplier/accumulator. The latches used in alternative 1 to provide the filter variable multiplicand to the multiplier, however, have been replaced with an 8 bit wide RAM. This RAM must be 18 bytes deep. These 18 bytes are divided into 2 groups. Each group contains 9 bytes,

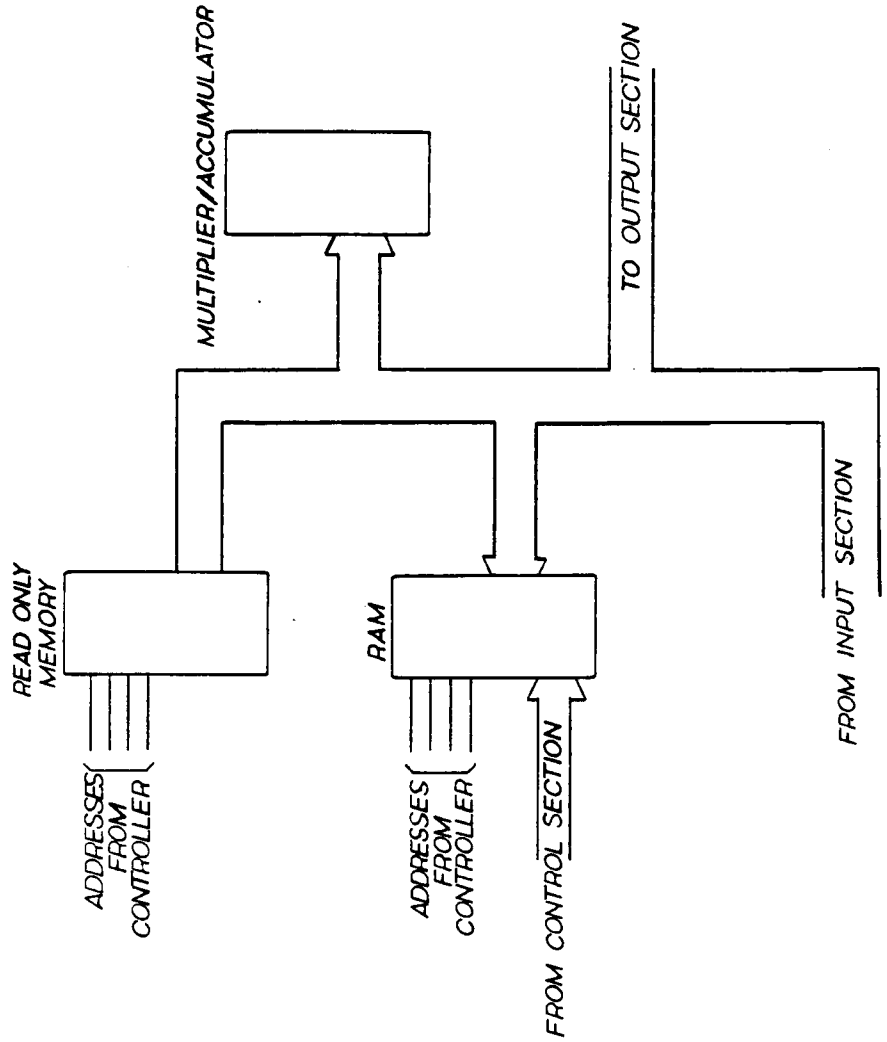


Figure 6.5. Alternative 2.

the number of variables contained in the filter. As the variables in one group are used to calculate a new set of variables, the new set is stored in the other group. Otherwise, as each new variable is calculated, the old variable would be lost. The old variable, however, is required to calculate other new variables.

Alternative 3, shown in Figure 6.6, makes use of a digital signal processor. One such processor is the TMS 32010 manufactured by Texas Instruments. This processor with 3K bytes of on chip ROM, 288 bytes of on chip static RAM, a 16 bit parallel multiplier and 32 bit ALU contains all the essentials for implementing the computational section of a digital filter. Therefore, the computational section for this alternative consists of only this module. The calculation of the filter algorithm is similar to alternative 2 except that programming the algorithm consists of writing a program for the TMS 32010 rather than programming an algorithmic state machine. This alternative, however, may violate the basic assumption that the other blocks of the digital filter are similar. The use of the TMS 32010 could conceivably make the control section of the filter much simpler, thereby producing some cost saving. Also, this processor provides for a 16 bit filter though only 8 bits are being used.

The comparison of these alternatives was made by estimating the board area and power of each alternative, converting these estimates into a cost and adding the results to the cost of the modules to obtain a total cost. The parameters used to make these estimates are given in Tables 6.3 - 6.6. A summary of the board area requirements for each alternative is given in Table 6.7. The total board area was obtained

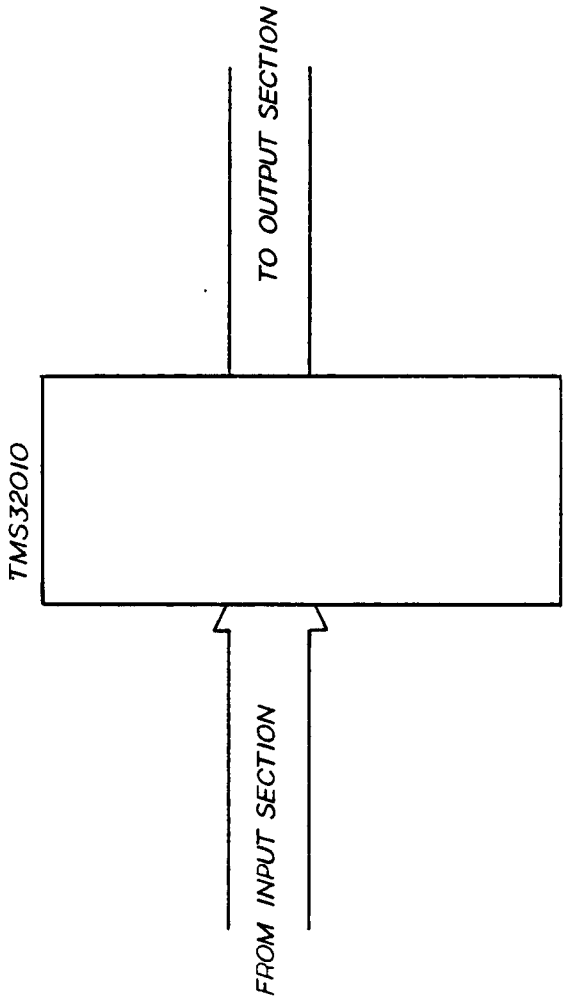


Figure 6.6. Alternative 3.

Table 6.3. Parameters for Estimating the Computational Section Costs for the Distributed Arithmetic Implementation

Module	Quantity	Cost (ea.) $	Supply Current Typical I_T (ea.) mA	Supply Current maximum I_M (ea.) mA	Board Area (ea.) in^2
ROM[1]	2	8.00	70	125	.80
74LS273	2	1.50	17	27	.33
74LS299	9	1.75	33	53	.33
74LS181	3	2.15	21	36	.69
74LS157	1	0.65	9.7	16	.23
74LS244	3	1.29	32	54	.33

[1] A 72 x 8 ROM would be required; a 2K x 8 ROM was used in the comparison because of its availability.

Table 6.4. Parameters for Estimating the Computational Section Costs for Alternative 1.

Module	Quantity	Cost (ea.) $	Supply Current Typical I_T (ea.) mA	Supply Current maximum I_M (ea.) mA	Board Area (ea.) in^2
ROM[1]	1	8.00	70	125	.80
Mult./Acc.	1	80.00	20	30	1.32
74LS273	9	1.50	17	27	.33

[1] A 27 x 8 ROM would be required; a 2K x 8 ROM was used in the comparison because of its availability.

Table 6.5. Parameters for Estimating the Computational Section Costs for Alternative 2.

Module	Quantity	Cost (ea.) $	Supply Current Typical I_T (ea.) mA	Supply Current maximum I_M (ea.) mA	Board Area (ea.) in^2
ROM[1]	1	8.00	~~70~~	125	.80
RAM	1	5.00	30	60	.69
Mult./Acc.	1	80.00	20	30	1.32

[1] A 27 x 8 ROM would be required; a 2K x 8 ROM was used in the comparison because of its availability.

Table 6.6. Parameters for Estimating the Computational Section Costs for Alternative 3.

Module	Quantity	Cost (ea.) $	Supply Current Typical I_T (ea.) mA	Supply Current maximum I_M (ea.) mA	Board Area (ea.) in^2
TMS32010	1	120.00	190	—	1.32

Table 6.7. Board Area Requirements of the Computational Section for Several Implementations.

Alternative	Total Chip Area in^2	Estimated Board Area in^2	Estimated Board Costs @ \$.20 per in^2 \$
Distributed Arithmetic	8.5	17.0	3.40
1	5.1	10.2	2.04
2	2.8	5.6	1.12
3	1.32	2.6	.53

by multiplying the module area by 2 to account for wiring the circuit. The costs were estimated by assuming the cost of a double sided printed circuit board is \$0.20 per in^2. The results indicate that alternative 3 requires the minimum amount of board area at 2.64 in^2 and a cost of \$0.53. Next comes alternative 2 at \$5.62 in^2 and a cost of \$1.12. Third is alternative 1 at 10.2 in^2 and a cost of \$2.04, while last is distributed arithmetic at 17.0 in^2 and \$3.40.

The power supply requirements for each alternative are summarized in Table 6.8. The total typical power supply current, $I_{T_{total}}$, for each alternative was obtained by adding the typical current required by the modules used in that alternative. The maximum current is the 3σ limit. The formula used to obtain the total maximum current, $I_{M_{total}}$, is:

$$I_{M_{total}} = \sum_{i=1}^{N} kI_T + 3\sqrt{\sum_{i=1}^{N} k\frac{(I_{Mi} - I_{Ti})^2}{3}} \qquad (6.4)$$

or,

$$I_{M_{total}} = \sum_{i=1}^{N} kI_T + \sqrt{\sum_{i=1}^{N} k[I_{Mi} - I_{Ti}]^2} \; . \qquad (6.5)$$

The maximum current was used to determine the power supply costs for each alternative. This estimate was made by deriving a cost versus current curve from the costs of Acopian power supplies. This curve was normalized to a power supply cost of \$30.00 for a 1 amp supply. The required current was then substituted into the cost curve and the resulting estimate obtained. From these estimates, alternative 2 requires the minimum current requiring 120 mA typical and 183 mA maximum. The estimated cost of such a supply is \$19.30. Alternative 3 requires the next lowest power at 190 mA typical. The maximum current

Table 6.8. Power Supply Requirements of the Computational Sections for Several Implementations.

Alternative	Supply Current Typical I_T (ea.) mA	Supply Current Maximum I_M (ea.) mA	Estimated Power Supply Cost $
Distributed Arithmetic	640	750	27.00
1	203	266	20.40
2	120	183	19.30
3	190	250[1]	20.20

[1] Estimated.

for the digital signal processor was not available, therefore, the maximum was estimated from other processor data to be 250 mA. Using this estimate, the power supply cost is $20.20 for this alternative. The typical and maximum currents required by alternative 1 were 203 and 266 mA respectively. The cost for this requirement was estimated as $20.40. The power supply current required by the distributed arithmetic approach is much higher than for any of these alternatives. The requirement was 640 mA typical and 750 mA maximum. The cost of this requirement was estimated as $27.00.

Thus far, the distributed arithmetic approach requiring more board area and more supply current than the other alternatives has not been competitive. The distributed arithmetic approach, however, gains a lot of ground when the module costs are considered. The estimates of module costs are presented in Table 6.9. The cost of the modules used in the distributed arithmetic approach is $45.72 which is much lower than the next closest alternative, alternative 2, at $93.00. Alternative 1 follows at $101.50 while alternative 3 is last at $120.00.

The board cost, power supply cost and module cost were added to give the total estimated cost for each alternative. These results are summarized in Table 6.10. The distributed arithmetic approach is the least expensive approach at $76.12 which is much lower than the next closest alternative, alternative 1, at $111.94. Alternative 2 is third at $113.42 while the digital signal processor approach is the most costly at $140.73.

To help put these results in perspective, the speed of each alternative was estimated. The speed was measured in terms of the

Table 6.9. Module Cost of the Computational Section for Several Implementations.

Alternative	Module Cost $
Distributed Arithmetic	45.72
1	101.50
2	93.00
3	120.00

Table 6.10. Cost Comparison of the Computational Section for Several Implementations.

Alternative	Cost $
Distributed Arithmetic	76.12
1	111.94
2	113.42
3	140.73

maximum sampling frequency allowed by each approach. For the distributed arithmetic approach, the maximum sampling frequency is dependent upon the access time from addresses valid for the function memory and upon the ALU propagation delay. For 3 cascaded 74LS181s, the maximum ALU propagation delay was 96ns. Assuming the ROM has a maximum access time of 150ns, the minimum cycle time is 246ns. For this case the maximum sampling frequency is:

$$f_s = \frac{1}{(246\ \frac{ns}{cycle})(72\ \frac{cycles}{sample})} = 56500\ \frac{samples}{sec}\ . \qquad (6.6)$$

If a ROM with a 200ns access time is used, the resulting sampling frequency is approximately 47000 samples/sec; while a ROM with a 250ns access time would allow 40000 samples/sec. The estimated cost was determined for a 200ns access time; therefore, the comparison is based upon a 47000 sample/sec speed. For alternative 1, the maximum multiplication cycle time is 150ns. Approximately 11 cycles would by required to calculate one filter variable. The filter has 9 such variables; therefore, the maximum sample rate is estimated as:

$$f_s = \frac{1}{(150\ \frac{ns}{cycle})(11\ \frac{cycles}{variable})(9\ \frac{variables}{sample})} = 67000\ \frac{samples}{sec}\ . \qquad (6.7)$$

The sampling rate for alternative 2 is identical to that of alternative 1. For the digital signal processor, 5 million instructions can be performed per second giving a 200ns cycle time. Calculation of the variables in each second order section would require approximately 34 instructions. Since 3 second order sections are used in the filter, the maximum sampling frequency is:

$$f_s = \frac{1}{3(34)(200\text{ns})} = 49000 \ \frac{\text{samples}}{\text{sec}} . \qquad (6.8)$$

Therefore alternative 1 and 2 are slightly faster than alternative 3 and the distributed arithmetic approach using ROMs with a 200ns access time.

The cost and speed of the alternatives can be incorporated into a single measure of performance by dividing the maximum sampling rate by the filter costs. Such a calculation accords the following results: 1) distributed arithmetic approach yields 617; 2) alternative 1 yields 599; 3) alternative 2 yields 591; and 4) alternative 3 yields 348. Since the higher the performance measure the more desirable the approach, the advantage of the distributed arithmetic is evident.

Output Noise of the Implemented Filters

The output roundoff error for the 2% and 20% filters is estimated in this section. The estimate is made assuming that the generated output roundoff noise is white. With this assumption, the output roundoff noise can be estimated from:

$$\begin{aligned}\sigma_0^2 = {} & \sigma_1^2 \|g_{11}\|_2^2 \|h_2 h_3\|_2^2 + \sigma_2^2 \|g_{12}\|_2^2 \|h_2 h_3\|_2^2 \\ & + \sigma_3^2 \|h_2 h_3\|_2^2 + \sigma_4^2 \|g_{21}\|_2^2 \|h_3\|_2^2 + \sigma_5^2 \|g_{22}\|_2^2 \|h_3\|_2^2 \\ & + \sigma_6^2 \|h_3\|_2^2 + \sigma_7^2 \|g_{31}\|_2^2 + \sigma_8^2 \|g_{32}\|_2^2 + \sigma_9^2 . \end{aligned} \qquad (6.9)$$

where σ_0^2 is the variance of the output roundoff noise, $\sigma_1^2 - \sigma_9^2$ are the roundoff variances introduced by rounding at the locations shown in Figure 6.7 and the other parameters are the L_2 norms defined in an earlier chapter. The cross correlation terms are not included since the output noise was assumed to be white. Mills, Mullis and Roberts [3], however, have shown that this is a good assumption.

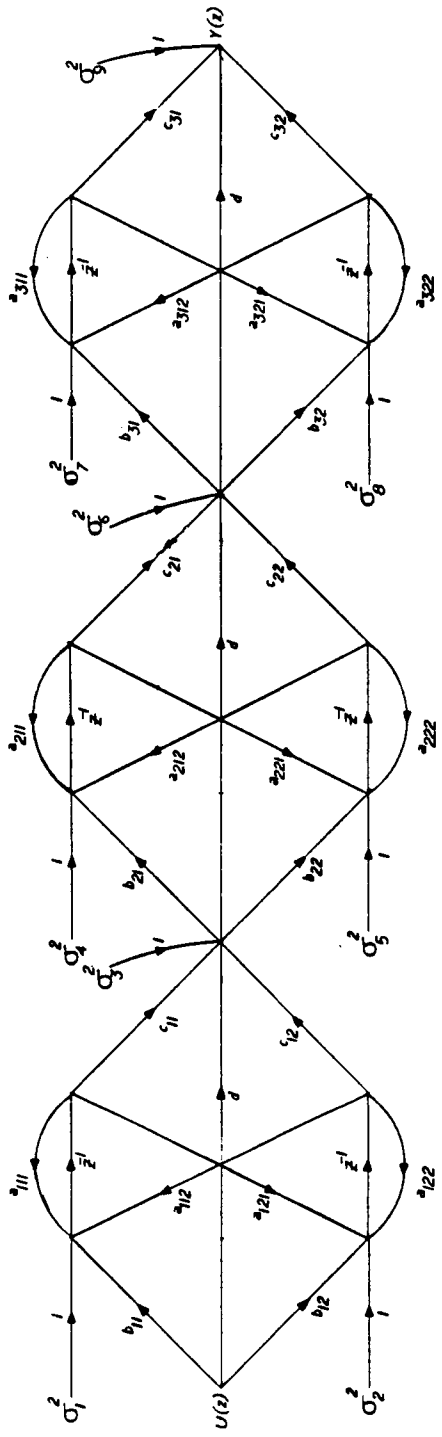


Figure 6.7. Model for estimating the output roundoff error of the 20% and 2% filters.

The variances $\sigma_1^2 - \sigma_9^2$ are found from (2.36)(pg. 18). Since all filter variables have equal word widths, $\sigma_1^2 - \sigma_9^2$ are equal for both filters. Kammeyer [7] has shown that for the distributed arithmetic approach the maximum roundoff may be estimated by using a ν of 4/3 in (2.36). For the 20% filter ε was:

$$\varepsilon = 2^{-7} \tag{6.10}$$

while for the 2% filter ε was:

$$\varepsilon = 2^{-11}. \tag{6.11}$$

Thus for the 20% filter, $\sigma_1^2 - \sigma_9^2$ were $(4/3)\frac{(2^{-7})^2}{12}$ and for the 2% filter $\sigma_1^2 - \sigma_9^2$ were $(4/3)\frac{(2^{-11})^2}{12}$.

The L_2 norms used in the calculations for each filter were calculated using Parseval's theorem. The results of these calculations are given for both filters in Table 6.11. From the results for the 20% filter given in Table 6.11 and $\sigma_1^2 - \sigma_9^2$ for the same filter given above, the output roundoff error for the 20% filter can be estimated as:

$$\sigma_{0_{20\%}}^2 = 1.4\text{x}10^{-5}. \tag{6.12}$$

Similarly from the results for the 2% filter given in Table 6.11 and $\sigma_1^2 - \sigma_9^2$ given above for the same filter, the output roundoff error for the 2% filter can be estimated as:

$$\sigma_{0_{2\%}}^2 = 2.27\text{x}10^{-7}. \tag{6.13}$$

Since ν for a normal multiplier implementation is 1, if rounding occurs after addition, the error due to the distributed arithmetic implementation is at most 1.33 times as large as that using a multiplier approach.

Table 6.11. L_2 Norms Used to Calculate the Output Roundoff Noise of the 20% and 2% Filters.

L_2 Norm	Value for the 2% Filter	Value for the 20% Filter
$\|\|g_{11}\|\|_2^2$	23.78	.7077
$\|\|g_{12}\|\|_2^2$	23.78	.7077
$\|\|h_2h_3\|\|_2^2$	0.0135	.1820
$\|\|g_{21}\|\|_2^2$	4.73	.2009
$\|\|g_{22}\|\|_2^2$	4.73	.2009
$\|\|h_3\|\|_2^2$	0.0169	.2076
$\|\|g_{31}\|\|_2^2$	3.37	.1702
$\|\|g_{32}\|\|_2^2$	3.37	.1702

The roundoff error for the implemented filters should be less than that for a filter implemented by cascading three direct form sections. The roundoff error was calculated for the 20% filter implemented using the direct form and found to be 4.02×10^{-4} for an 8 bit implementation. This roundoff error is 29 times higher than the roundoff error in the implemented filter. These results demonstrate the effectiveness of the reduced roundoff error filter in reducing the roundoff error.

Summary

The predominant goal of this work was to demonstrate the applicability of the distributed arithmetic approach to implementing digital filter algorithms. Therefore a hardware efficient implementation of the distributed arithmetic approach was used to implement two sixth order Butterworth filters. In this implementation, each sixth order filter was divided into 3 second order sections which were cascaded to provide the filter. The hardware required to implement one of the second order sections was shared to calculate the variables of the other sections.

The implemented filters were sixth order Butterworth filters, one of which had a cutoff frequency which was 20% of the Nyquist frequency, the other a cutoff frequency which was 2% of Nyquist frequency. An 8 bit implementation of the 20% filter produced the desired transfer function. A 12 bit ALU and function memory, however, were required for this 8 bit implementation. A 12 bit implementation was required for the 2% filter. In this case a 16 bit ALU and function memory were required. With these implementations, the filters implement the

required transfer functions within the measurement error of 2%. No visible distortion was evident at the output.

Comparison of the distributed arithmetic implementation with 3 alternate implementations revealed its economic advantage. The 3 alternatives included 2 multiplier approaches and 1 digital signal processor approach. The results indicate that the costs of the computational section of the distributed arithmetic implementation were 68% of the costs of a multiplier approach in which latches are used to delay filter variables, 67% of the costs of a multiplier approach in which RAM is used to delay filter variables, and 54% of the costs of the digital signal processor approach. The lower costs prevail even though the computational section of the distributed arithmetic approach requires more board area and current than any of the alternatives. The costs savings primarily reflect the high costs of multiplier chips and digital signal processors.

The filter coefficients for the implemented filters were chosen to reduce the output roundoff error of the filters. The use of 3 cascaded second order state-space sections simplified the hardware but sacrificed the achievement of minimum possible roundoff error. By minimizing each second order section independently and ordering the sections to minimize the noise gain, the reduced output roundoff error was obtained. The techniques for deriving the filter coefficients of a second order section were presented and applied to the 20% and 2% Butterworth filter sections.

The maximum output roundoff noise for the 20% and 2% filters was estimated. For the 20% filter with an 8 bit implementation, the output

roundoff error was estimated as 1.4×10^{-5}. For the 2% filter using a 12 bit implementation, the output roundoff error was estimated as 2.27×10^{-7}. The roundoff error using the distributed arithmetic approach is a maximum of 1.33 times that for an implementation using multipliers with rounding following addition. The estimated roundoff error of the 20% filter was 29 times lower than an equivalent filter implemented using direct form second order sections. Thus the implemented filters represent a cost effective compromise between reducing roundoff error and reducing the filter costs.

List of References

LIST OF REFERENCES

LIST OF REFERENCES

[1] C. T. Mullis and R. A. Roberts, "Synthesis of Minimum Roundoff Noise Fixed-Point Digital Filters," IEEE Trans. Circuits Syst., Vol. CAS-23, pp. 551-561, Sept. 1976.

[2] Leland B. Jackson, Allen G. Lindgren, and Young Kim, "Optimal Synthesis of Second Order State-Space Structures for Digital Filters," IEEE Trans. Circuits Syst., Vol. CAS-26, pp. 149-153, March 1979.

[3] William L. Mills, Clifford T. Mullis, and Richard A. Roberts, "Low Roundoff Noise and Normal Realizations of Fixed Point IIR Digital Filters," IEEE Trans. Acoust., Speech, Signal Processing, Vol. ASSP-29, No. 4, pp. 893-903, August 1981.

[4] Bruce W. Bomar, "A Framework for the Comparison, Analysis and Design of Digital Filter Structures," Dissertation Proposal, University of Tennessee, Knoxville, p. 59, May 1982.

[5] M. Arjmand and R. A. Roberts, "Efficient Hardware Implementations for Fixed Point Digital Filters," Proc. 1980 Int. Symp. Circuits System., Houston, Texas, pp. 1092-1096, April 1980.

[6] David G. Luenberger, Introduction to Dynamic Systems., New York: John Wiley and Sons; 1979, p. 293.

[7] K. D. Kammeyer, "Quantization Error Analysis of the Distributed Arithmetic, " IEEE Trans. Circuits Syst., Vol. CAS-24, No. 12, pp. 681-689, December 1977.

APPENDIX

APPENDIX

SELF TEST

Self testing and diagnostics have recently become an integral part of equipment design. In the past, hardware has been relatively expensive and labor relatively cheap. Today, however, the decreasing costs of digital hardware, especially microprocessors and memory chips, make the added hardware and design costs for self test less expensive than the labor costs required to debug a piece of equipment. The additional memory area of the 2KX8 EPROM's used in the digital filter's ASM and function memory can be used to provide self-test capability for the computational section of the digital filter. In this appendix, one possible self-test approach is outlined for the 20% bandwidth Butterworth filter.

The approach that is presented is a divide and conquer scheme. Dividing the filter into 4 or 5 sections that can be tested separately or using previously tested sections facilitates programming the ASM for self-test and analyzing the results. The sections would include 3 shift register bank sections, an EPROM, ALU, B register section and an EPROM, ALU, buffer section.

Testing the filter would begin by testing the shift register banks. Shift register bank 1 would consist of U9, U1, and one of the quad 2 input MUX's (MUX1a). Shift register bank 2 would be composed of U4, U3, U2, and another of the quad 2 input MUX's (MUX1b). U8, U7, U6, U5, and a third unit of the quad 2 input MUX (MUX1c) would comprise shift register bank 3. Assume that the filter has been reset so that

all bits of the filter contain 0's. At this time the ASM would apply a 1 to pin 11 of U4, U8, and U9 for 1 clock period. Meanwhile the shift registers would be instructed to shift right. On the next clock edge, a 1 would be shifted into the MSB of shift registers U4, U8, and U9. The ASM would then return pin 11 of U4, U8, and U9 to 0 and instruct all shift registers to shift right for the next 16 clock transitions. During these clock periods, MUX1 would be configured by the ASM to supply LB9 to output MX1, LB3 to MX2, and LB6 to MX3. The 1 applied to pin 11 of U4, U8, and U9 would ripple through the shift register banks appearing at Q'_h of SR1 at the end of the 16th clock transition.

Now assume that Q'_h of U1, U2, and U5 are connected to an 8 to 1 MUX (MUX2) whose output is an input to the ASM as shown in Figure A.1. Three outputs of the ASM dedicated as inputs to S_0, S_1, and S_2 of MUX2, select one of MUX2's 8 inputs as an input to the ASM. In this way, the ASM could monitor one of 8 inputs from the filter. Finally assume that Q'_h of U1 is connected to I_0, Q'_h of U2 is connected to I_1, and Q'_h of U5 is connected to I_2. With this configuration, S_0, S_1, and S_2 outputs of the ASM would be 0 for the 16 clock periods following the application of a 1 to pin 11 of U4, U8, and U9. The output of MUX2 should be 0 until after the 16th clock transition at which time the output should be 1 for 1 clock cycle. During these 16 clock cycles, the ASM monitors Q'_h of U1 as it proceeds through the self-test program. If an error occurs, an errant address would be seen by the ASM. The contents of all possible errant addresses could contain information to turn on an LED to indicate that shift register bank section 1 is bad and also hang up the ASM at that address. If these errant addresses are not

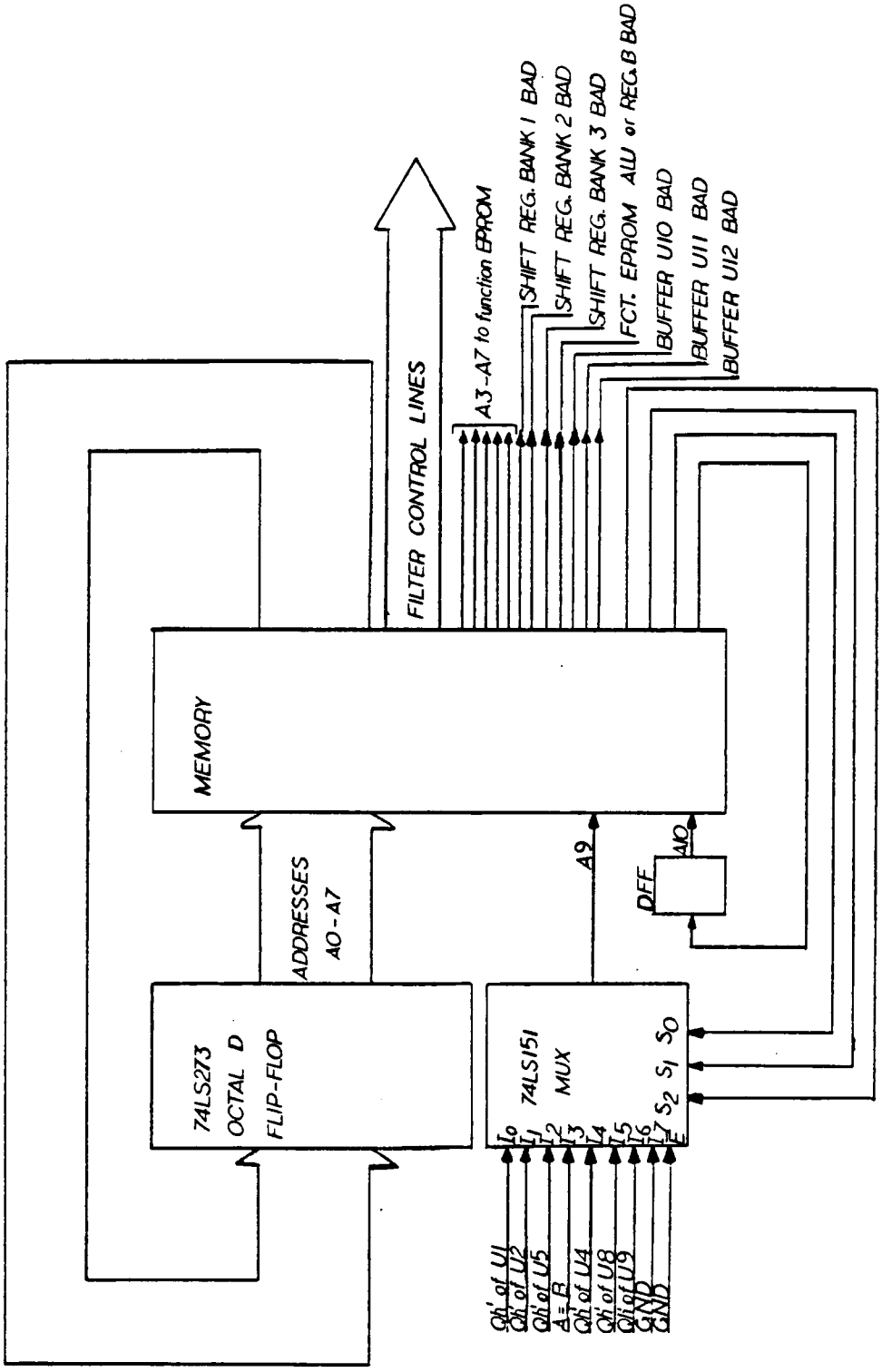


Figure A.1. ASM modified for self test.

encountered, the ASM would select I_1 of MUX2 after the 16th clock transition allowing the ASM to monitor Q'_h of U2. For the next 7 clock periods, MUX2's output should be 0, but after the 24th clock transition the output should become 1. Again, an errant address to the ASM could turn on an LED indicating that shift register bank 2 is bad and hang up the ASM at the errant address. As before, barring an encounter of an errant address, the ASM would continue to proceed through the self-test program. A similar situation would apply for shift register bank 3. After the 24th clock transition, input I_2 of MUX2 would be selected to allow the ASM to monitor Q'_h of U5. For the next 7 clock cycles, Q'_h of U5 should be 0, but after the 32nd clock transition, the output should become 1 for 1 clock cycle. As before, if an errant address is encountered, and LED indicating a bad shift register bank 3 lights up and the ASM hangs up at the errant address. If an errant address is not encountered, the self-test algorithm proceeds. A flow chart presenting the self-test for the shift registers banks is shown in Figure A.2.

Caution must be exercised when programming the ASM for self-test to ensure that errant locations as well as expected locations of the ASM are reserved for the proper control data. To demonstrate the significance of this statement assume that 1 input from the filter is being monitored by the ASM. Also, for simplicity, assume that only a 4 address ASM is required. This situation is shown in Figure A.3. The program that the machine is designed to execute resides in addresses 0000 to 0011. The self-test algorithm resides in the upper addresses 1000 to 1111. The input to be monitored by the test program is

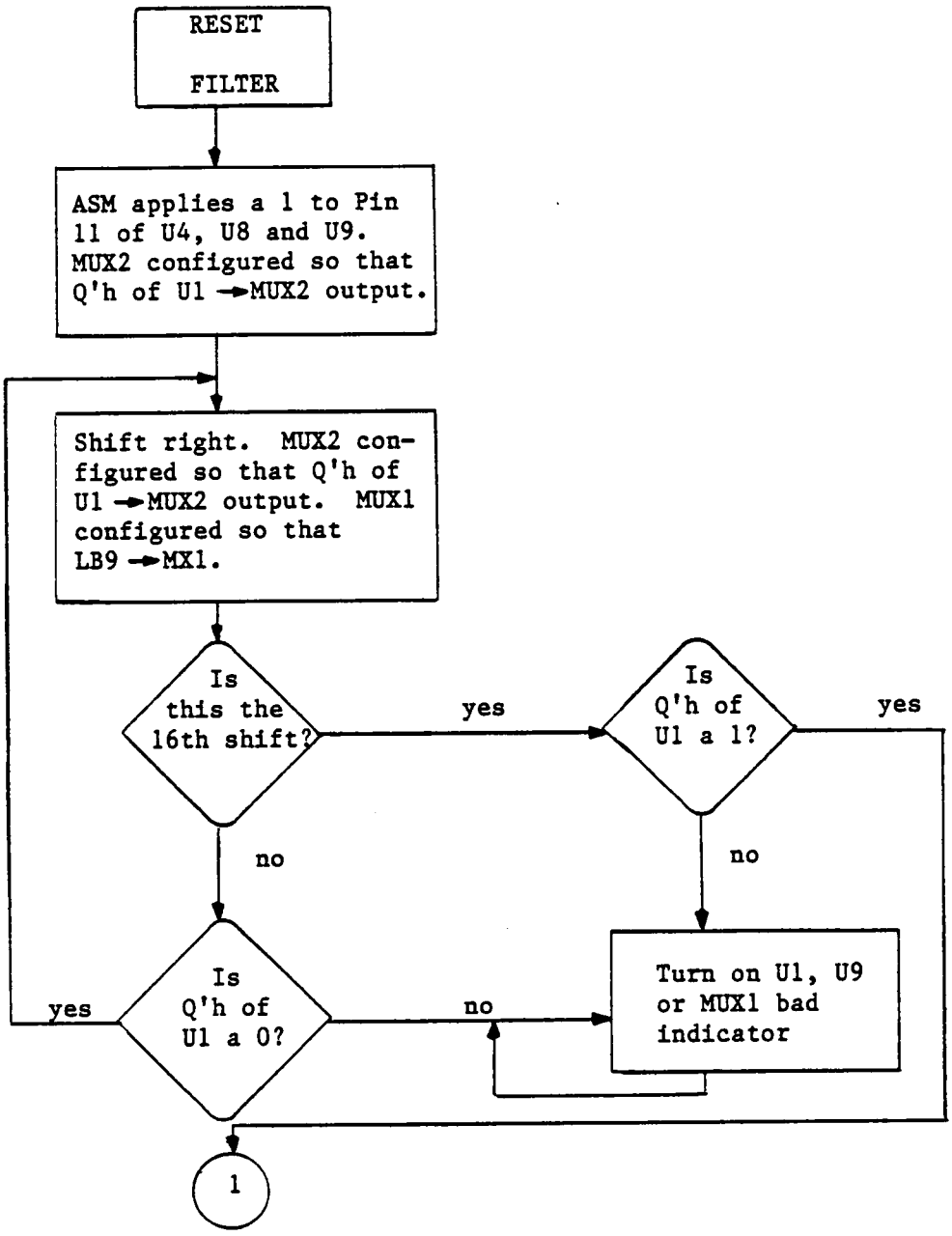


Figure A.2. Flow chart for shift registers self test.

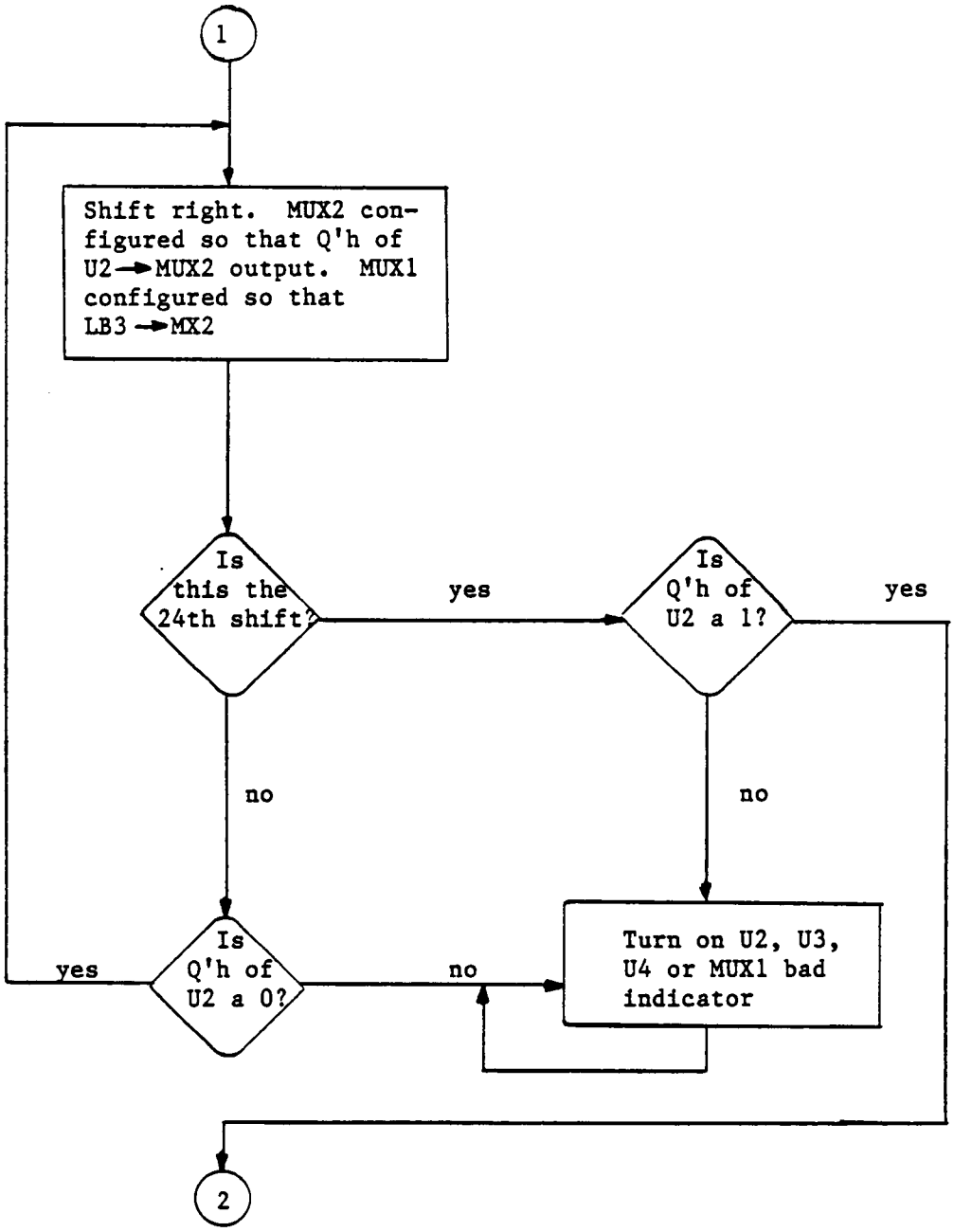


Figure A.2. (Continued)

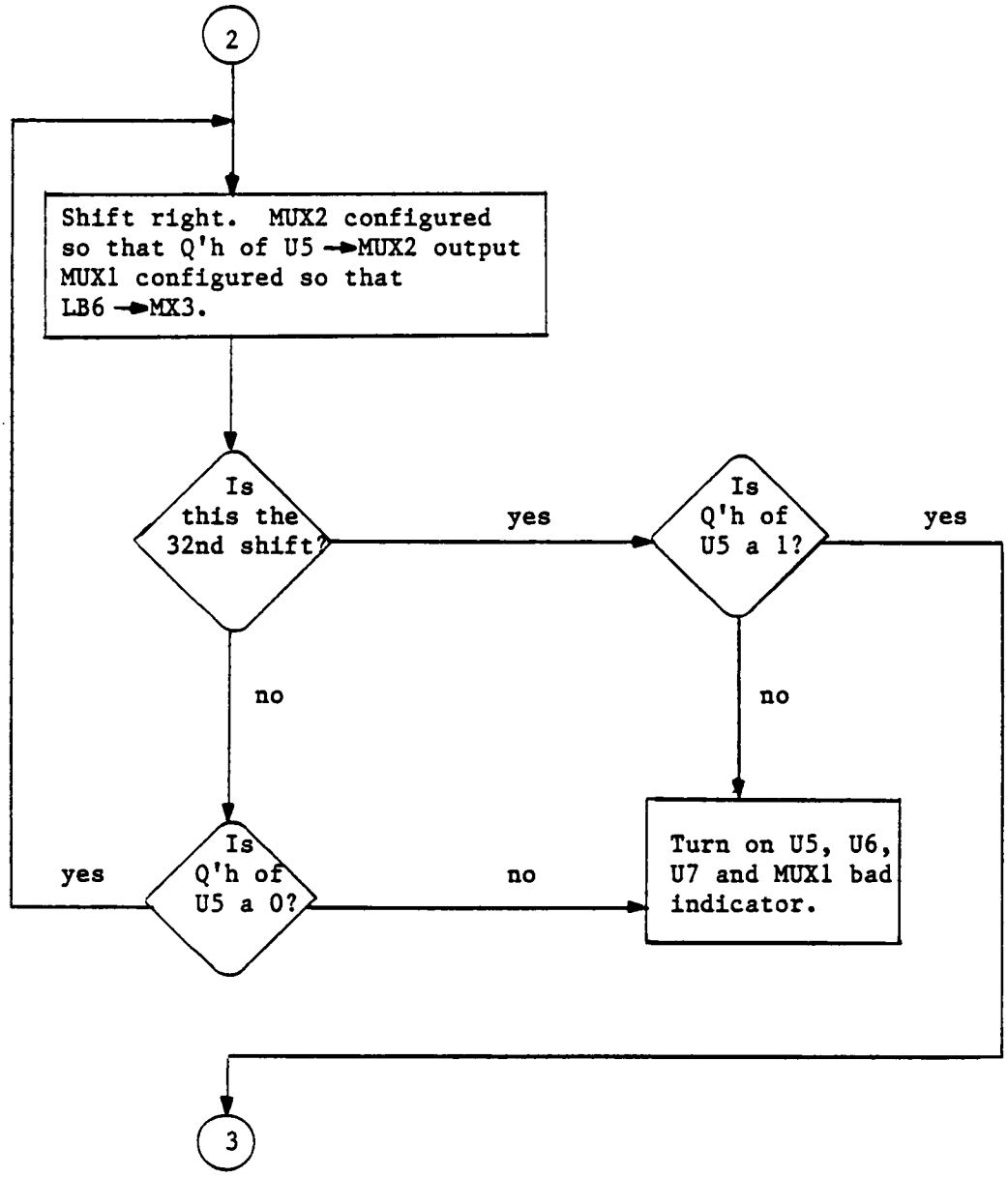


Figure A.2. (Continued)

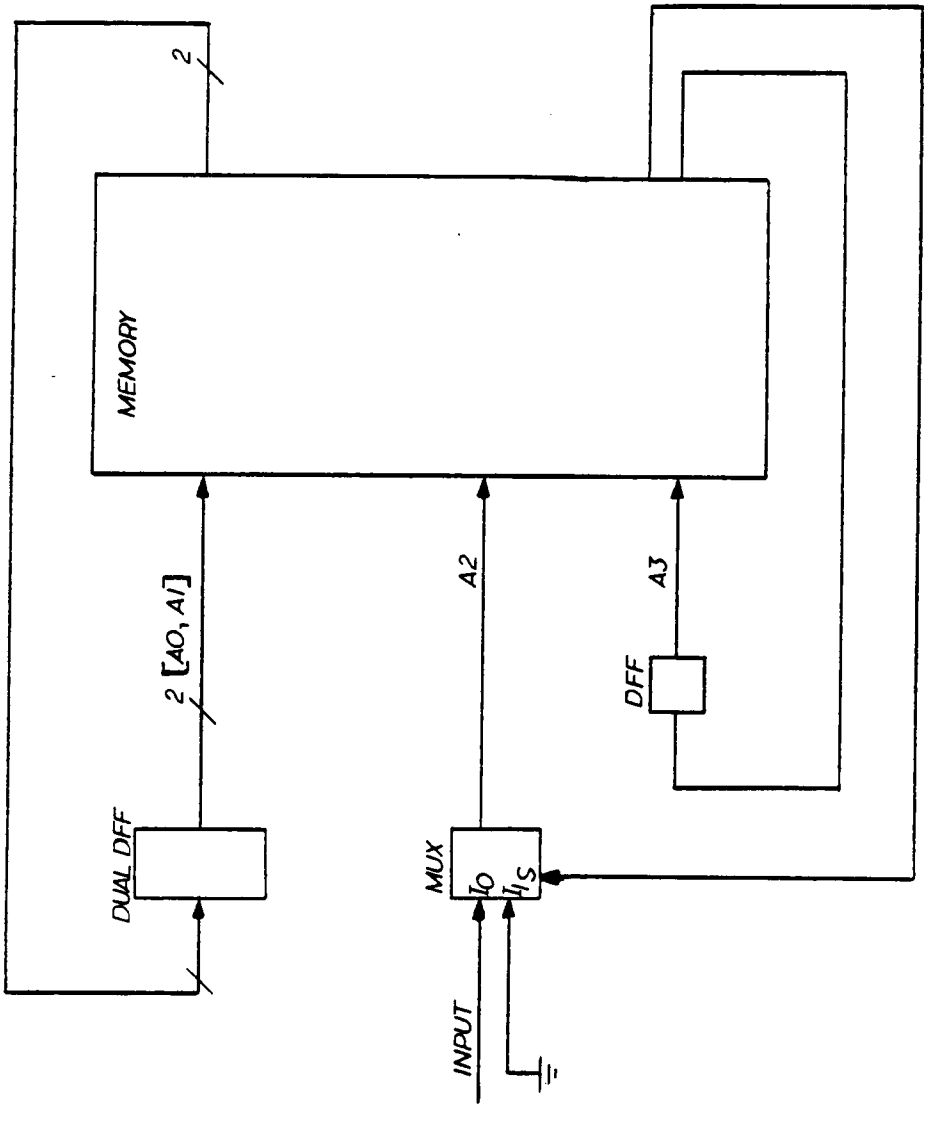


Figure A.3. Simplified ASM with self test capability.

connected to address A2. Also assume that when the ASM is in state 1010, it is instructed to monitor the input. The next state of the ASM will be 1111 assuming that A2 is expected to be a 1. The actual state to which the ASM will proceed, however, will be 1011 or 1111 depending upon the state of A2. Both locations must contain relevant control information for the possible outcomes of the input. The location 1111 would contain information allowing the self-test algorithm to proceed while the location 1011 would contain information to indicate that A2 was erroneous.

If the filter has performed as expected in the above tests, the assumption of properly functioning shift registers can be made. These functioning shift registers can be used along with the ASM to test the function EPROM's, ALU and B register. In this test the shift registers and 5 outputs from the ASM could be used to address the function EPROM's. Data at the addressed locations would be added to or subtracted from data in the B register by the ALU. The results would be latched into register B. After a few additions and subtractions, a location in the function EPROM containing the expected results of the additions and subtractions would be addressed putting the expected results into side A of the ALU. Side A would then be compared with the actual results contained in register B. If the data are equal, the ALU produces a signal which would be monitored by the ASM. If the signal is detected when expected, the self-test algorithm continues. If the signal does not occur when expected, the ASM hangs up and produces an indication that the function EPROM's, ALU or register B are bad.

The above scheme would be carried out in the following manner.

First register B of the filter is reset. The lower 3 address bits of the function EPROM's, A0 - A2, are placed on pin 11 of U4, U8, and U9 by the ASM. These address bits are shifted through the shift register banks ultimately appearing at Q'_h of U1, U2, and U5. An appropriate bit string must be placed on pin 11 of U4, U8, and U9 to obtain the proper addresses. One address bit string producing the desired results would be:

011000110011100100001111011000110011100100001111...

This bit string produces the lower address bit nibble sequence:

X000	X000	X000
X001	X100	X010
X011	X101	X110
X010	X001	X100
X110	X011	X101
X100	X010	X001
X101	X110	X011
X111	X111	X111.

The address bits A3 - A7 are to be supplied directly by outputs of the ASM as in the normal filter mode. A valid address then appears at the function EPROM, 32 clock transitions after the lower 3 bits are first placed on pin 11 of U4, U8, and U9. The address bit string, however, can begin immediately after the 1 was placed on pin 11 of U4, U8 and U9 for testing the shift registers. Thus, the addresses for this test would be available immediately after the 3rd shift register bank was tested.

The above address bit string will not address locations of the function EPROM sequentially, but this fact does not pose a problem since the function EPROM's would be programmed in the nibble sequence that will occur. Supplying the data of the addressed location to side

A of the ALU, which is set to add or subtract by the ASM control lines, causes data from the EPROM's to be added to or subtracted from data residing in register B. Register B then latches the results of each addition and subtraction right shifted 1 position (the inputs to register B are hardwired to perform this right shift) and in turn presents the latched data to side B of the ALU for the next addition or subtraction. After the additions and subtractions have been completed and the result latched in register B, the ALU is instructed to subtract. The EPROM is again addressed; however, the data in the addressed location contains the value expected to be in register B. If the data in register B is the anticipated result, the inputs to both sides of the ALU are equivalent. The ALU acknowledges this by placing a 1 on the A = B output. Since the A = B outputs are open collector, they can be and-tied to one of the 8 inputs of MUX2 which is configured by the ASM to pass the A = B output to the ASM. In this manner, the ASM can monitor the A = B output. If A and B are equivalent, the resulting ASM state allows the self-test algorithm to continue. If, however, A and B are not equivalent, the unexpected state is programmed to hang up the ASM and turn on an indicator announcing that the EPROM, ALU or B register is bad. Figure A.4 presents a flowchart for this test.

The self-test algorithm is completed by providing a means to test the 74LS244 buffers for the inputs to shift registers U4, U8, and U9. To test these buffers, data can be transferred from the function EPROM's through the ALU and buffers to U4, U8, and U9. Shift registers U4, U8, and U9 would then be instructed to latch the information from

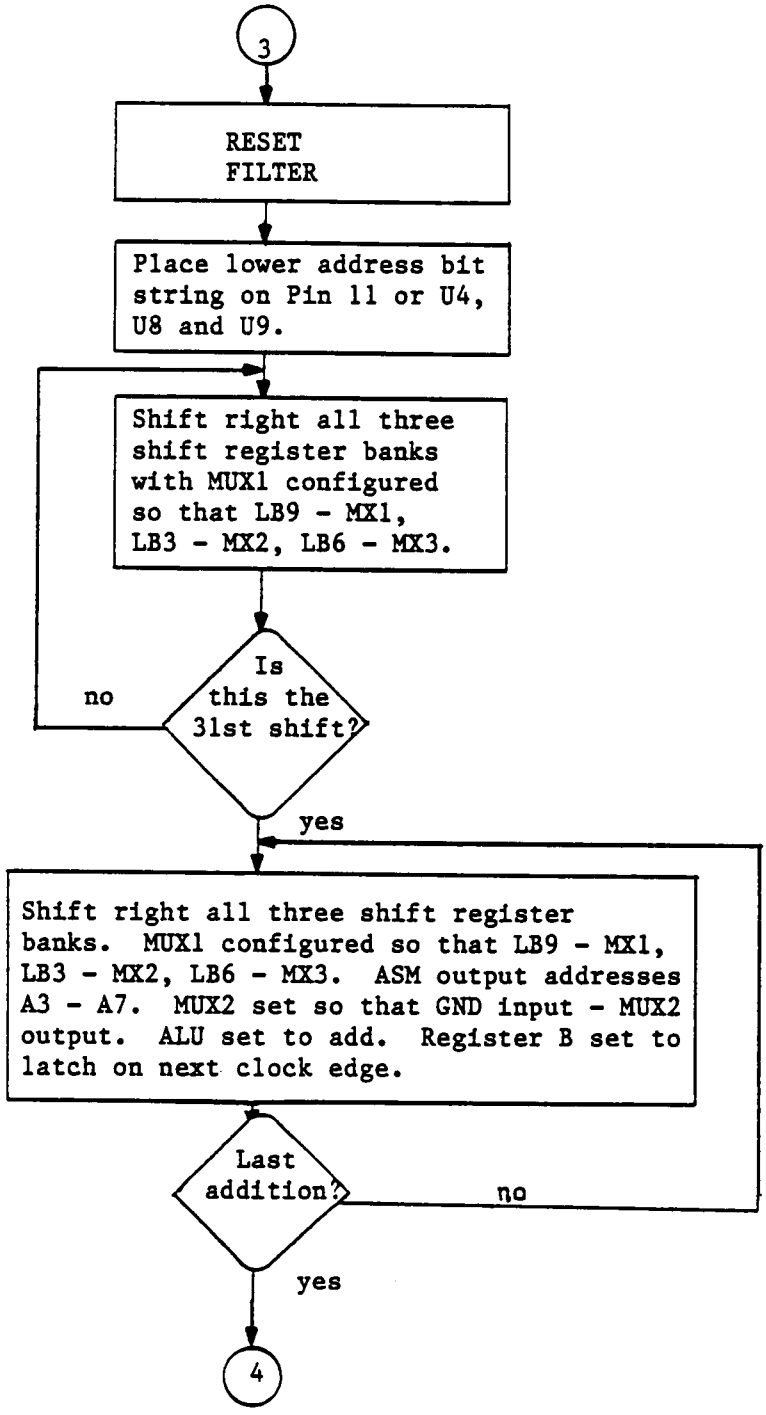


Figure A.4. Flow chart for EPROM, ALU, and B register self test.

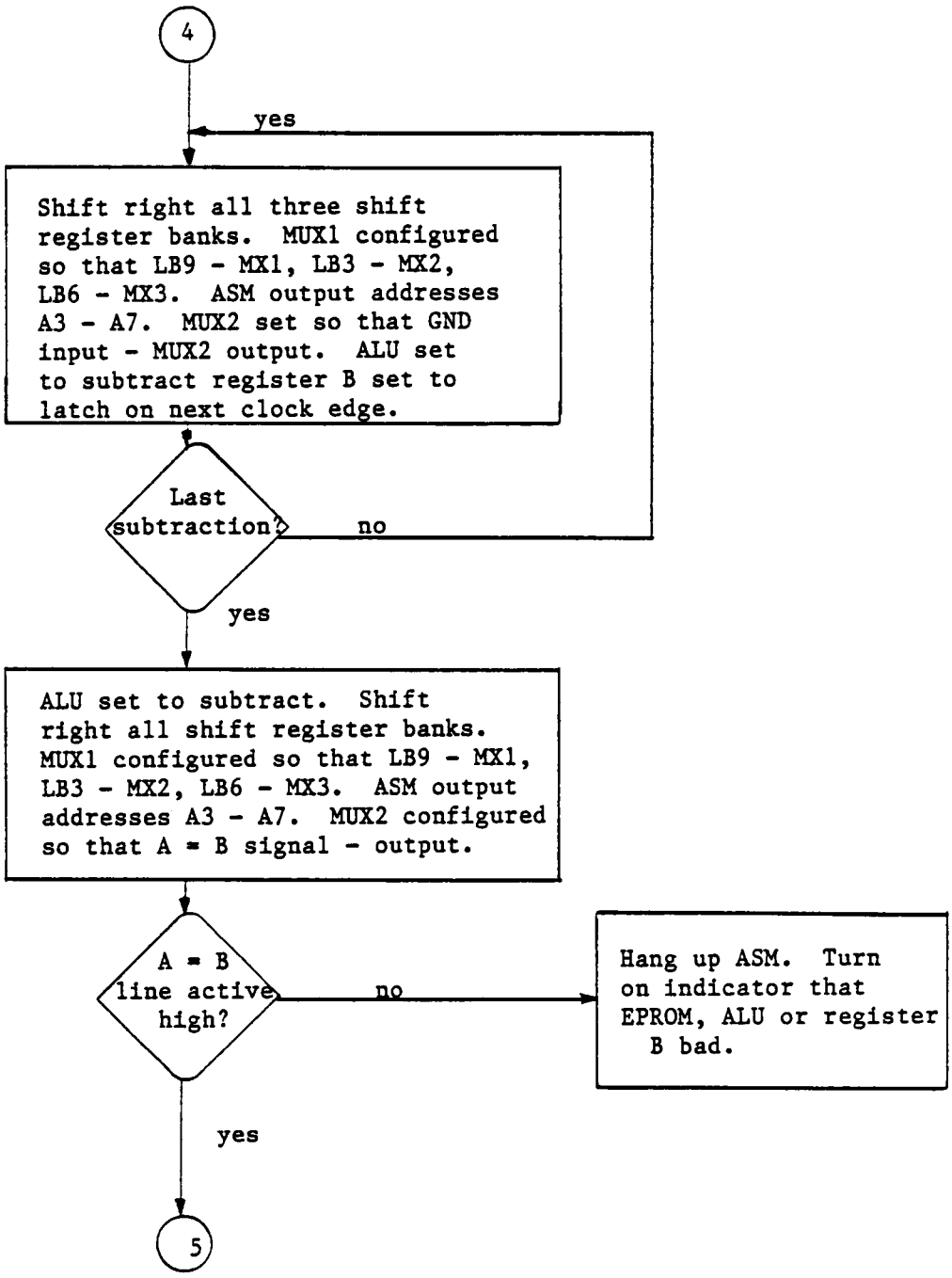


Figure A.4. (Continued)

the buffers. If the buffers are working properly the data latched into the shift registers should be identical to bits R2 - R9 from the ALU which in turn should be identical to G2 - G9 of the function EPROM. Assuming the function EPROM has been programmed with a known value, the value that should be in U4, U8, and U9 would be a known value. The shift registers could then be instructed one at a time by the ASM to shift their contents right 8 times. The output of the shift registers would be fed to MUX2 allowing the contents of the shift registers to be monitored by the ASM. If the contents of the shift registers are proper, the algorithm proceeds through the test; however, if the contents are erroneous, the ASM hangs up and indicates that the buffer is bad or that U4, U8 and U9 erroneously latched the data.

Implementation of the above scheme is similar to the scheme for checking the function EPROM, ALU, and B register. First register B of the filter is reset. Since the shift register banks have not been reset, the address sequence from the previous test remains valid. The location that is addressed, which is programmed with a known value, will output its contents to the ALU. The ALU, meanwhile, will be set to the addition mode by the ASM. Since register B is reset to zero, the output of the ALU will be equal to the output from the function EPROM. Bits R2 - R9 of the ALU output are fed to the buffers which pass the information to shift registers U4, U8 and U9. These shift registers latch the input on the next clock transition. Once the data is latched, shift register U4 shifts its data right 8 times while shift registers U8 and U9 hold. Q'_h of U4 is fed to one of the inputs of MUX2 which presents this input to the ASM. In this manner the ASM

monitors the data latched into shift register U4. If an erroneous bit occurs at the output of the MUX, the ASM indicates an error in buffer U10 or shift register U4's parallel load function. The ASM hangs up in this state. If no erroneous bits appear the self-test algorithm continues.

Upon the continuation of the self-test algorithm, shift register U8 is instructed to shift right 8 times. As for shift register U4, Q'_h of U8 is fed to one of the inputs of MUX2. While U8 is shifting right, MUX2 passes U8's output to the ASM which monitors the data in shift register U8. Shift register U9 then shifts right 8 times with its output monitored by the ASM. If while monitoring the data in U8 or U9 the ASM sees an erroneous bit, the ASM hangs up and indicates that the appropriate buffer is bad. If the filter passes all these tests, another set of data can be passed from the function EPROM to the shift registers and the test repeated. Four sets of data that might be used to test the buffers are 00H, FFH, AAH, and 55H. If the filter passes all these tests, then the computational section of the filter is deemed fit to run. The filter algorithm of the ASM EPROM is called and the filter begins to perform its function. A flow chart for testing the buffers is presented in Figure A.5.

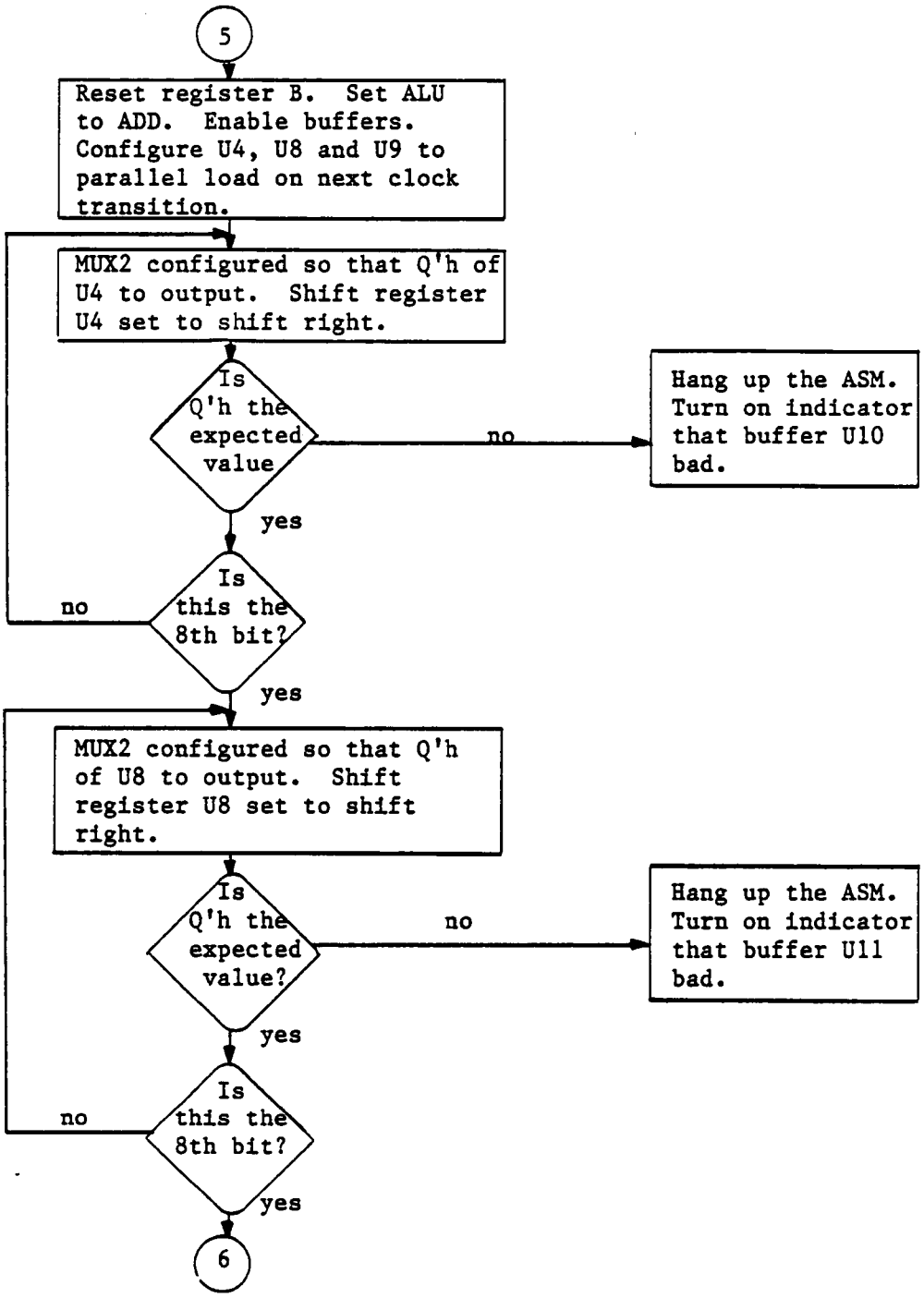


Figure A.5. Flow chart for buffers self test.

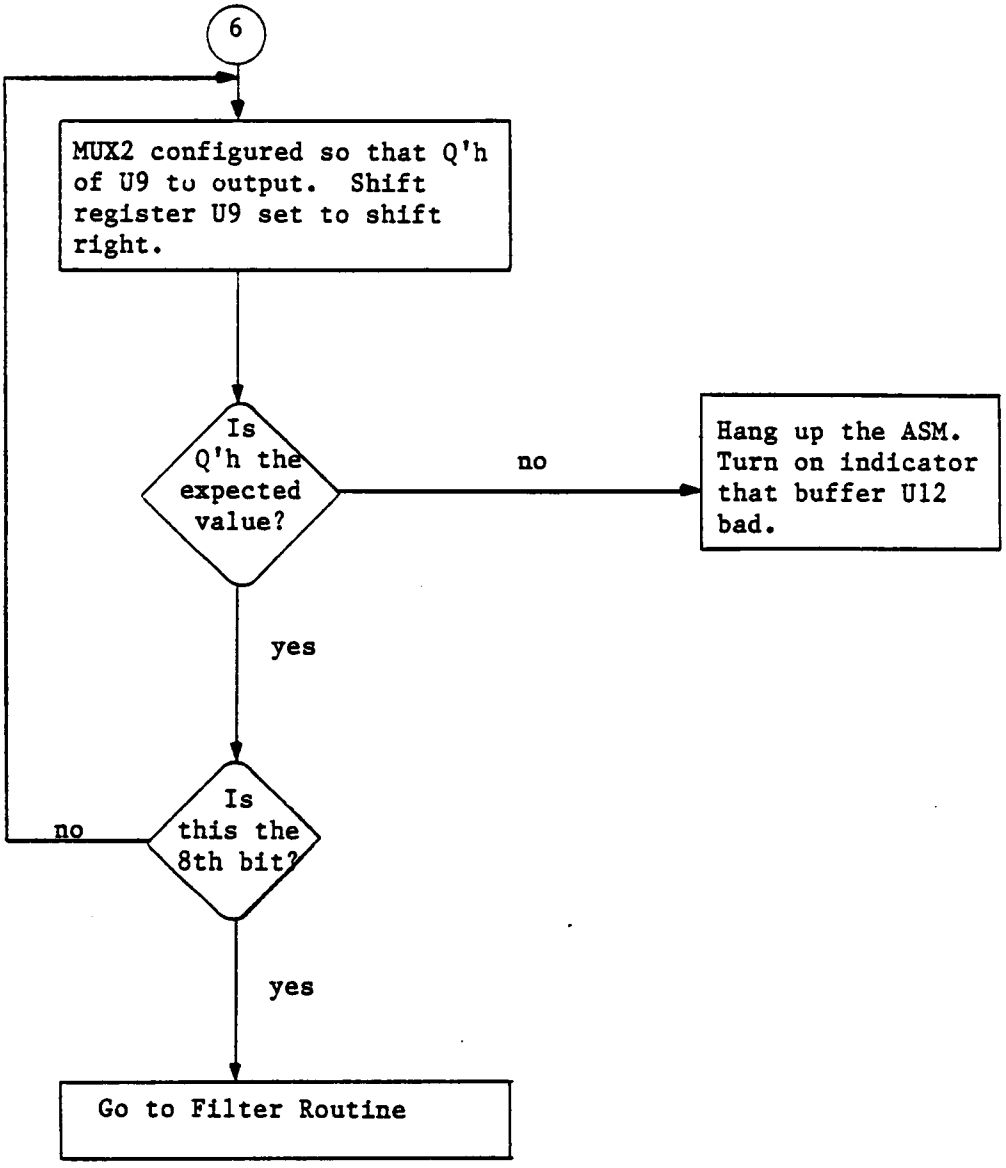


Figure A.5. (Continued)

VITA

James Edward Phillips was born in Winston-Salem, North Carolina on February 19, 1952. He attended elementary school in Kernersville, North Carolina and was graduated from East Forsythe High School in June 1970. The following September he entered North Carolina State University and in May 1974 he received a Bachelor of Science degree in Nuclear Engineering. In the fall of 1974, he entered The University of North Carolina at Chapel Hill to pursue a Master's degree in Environmental Science. This degree was awarded in December 1977.

After working at Oak Ridge National Laboratory in the Health and Safety Research Division for six years, he entered the Graduate School of The University of Tennessee, Knoxville, in January 1981. He received a Master's degree with a major in Electrical Engineering in the Spring of 1984.

The author is a member of the Institute of Electrical and Electronics Engineers and Tau Beta Pi Honor Society. Awards or honors awarded are Outstanding Senior in Nuclear Engineering, 1974, Reynolds Alumni Scholarship, Atomic Energy Commission Fellowship, 1975-1976, and Eagle Scout. Mr. Phillips is currently employed with International Business Machines.

He is married to the former Cynthia Warren of Chattanooga, Tennessee, and they live in Lexington, Kentucky.