

To the Graduate Council:

I am submitting herewith a thesis written by Tyrone Reid, entitled "An Analysis and Implementation of State Space Digital Filters Free of Limit Cycles." 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.

Bruce W. Bomar
Bruce W. Bomar, Major Professor

We have read this thesis
and recommend its acceptance:

Roy D Joseph
L. Montgomery Smith

Accepted for the Council:

Lee Minkel
Associate Vice Chancellor
and Dean of The Graduate School

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a Master's degree at The University of Tennessee, Knoxville, I agree that the Library shall make it available to borrowers under rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of the source is made.

Permission for extensive quotation from or reproduction of this thesis may be granted by my major professor, or in his absence, by the Head of Interlibrary Services when, in the opinion of either, the proposed use of the material is for scholarly purposes. Any copying or use of the material in this thesis for financial gain shall not be allowed without my written permission.

Signature Tyrone Reid

Date 4/26/93

**AN ANALYSIS AND IMPLEMENTATION
OF STATE SPACE DIGITAL FILTERS FREE
OF LIMIT CYCLES**

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Tyrone Reid

May 1993

Dedicated
to my parents
Mr. and Mrs. Willie J. Reid

ACKNOWLEDGMENTS

I would like to express my sincere appreciation to my advisor and major professor Dr. Bruce W. Bomar. His guidance, suggestions, and advice were vital to the success of this thesis. Appreciation is also due Dr. Roy Joseph and Dr. L. Montgomery Smith for serving on my thesis committee.

I would also like to thank the National Consortium for Graduate Degrees for Minorities in Engineering and Science, Inc. (GEM) for their support during the course of this study.

ABSTRACT

The focus of this thesis is the elimination of constant-input limit cycles in fixed-point implementation of digital filters. Methods are discussed that produce state-space realizations free of limit cycles under constant-input conditions from realizations free of limit cycles under zero-input conditions.

The first issue that was addressed is whether a realization free of limit cycles under zero-input conditions can have limit cycles under constant-input conditions. Computer simulations were presented to show that realizations free of zero-input limit cycles can have constant-input limit cycles.

Four transfer functions which have constant-input limit cycles were selected for subsequent study. Implementations of these transfer functions on a Motorola DSP56000/1 family of digital signal processors were considered. By implementing these transfer functions on a Motorola DSP56000/1 simulator, it was verified that constant-input limit cycles are eliminated from the four transfer functions. To measure the computational cost of eliminating limit cycles, the limit-cycle-free implementations were compared to implementations that support limit cycles. The comparison was based on the number of processor instruction cycles required per sample processed.

TABLE OF CONTENTS

CHAPTER	PAGE
I. INTRODUCTION.....	1
II. STATE-SPACE REALIZATION	7
III. ELIMINATING LIMIT CYCLES	
1. First-order section	15
2. Second-order section	19
IV. DEVELOPMENT OF EXAMPLES OF LIMIT CYCLES	
1. PC Programs for finding limit cycles	32
2. Transfer functions and implementations	37
V. DSP PROGRAMS FOR DIGITAL FILTERS	
FREE OF LIMIT CYCLES	44
1. First-order section	46
2. Second-order section	47
3. Nth-order filter (5th EXAMPLE)	49
4. Experimental results	51
VI. COMPUTATIONAL LOADING	
1. COMPUTATION CYCLES REQUIRED.....	52
2. COMPUTATION OF IMPLEMENTATION WITH	
LIMIT CYCLES	54
VII. SUMMARY AND CONCLUSIONS	56
REFERENCES	58

APPENDIXES.....	61
A. PROGRAM LISTINGS OF LIMITCY	62
B. PROGRAM LISTINGS OF MYT	65
C. PROGRAM LISTINGS OF SCALELI	68
D. PROGRAM LISTINGS OF IFILNM	69
E. PROGRAM LISTINGS OF COEFFIR	70
F. PROGRAM LISTINGS OF COEFSEC	71
G. PROGRAM LISTINGS OF FIRST	73
H. PROGRAM LISTINGS OF DCGAIN	75
I. PROGRAM LISTINGS OF FIFTH	77
J. PROGRAM LISTINGS OF RFIL.....	81
VITA	83

LIST OF TABLES

TABLES	PAGE
1. Coefficients for $H_1(z)$	38
2. Coefficients for $H_2(z)$	39
3. Coefficients for $H_3(z)$	40
4. Coefficients for $H_4(z)$	41
5. Coefficients for $H_5(z)$	42
6. Coefficients for $H_6(z)$	43

CHAPTER I

INTRODUCTION

Limit cycles occur in recursive digital filters when finite precision arithmetic is used to perform the arithmetic operations required. When recursive digital filters are implemented, errors due to finite precision are unavoidable and results in overflow, truncation, and roundoff errors.

There are two basic types of limit cycles that occur in the implementation of recursive digital filters when realized by finite arithmetic: limit cycles due to multiplication roundoff and limit cycles due to adder overflow. Multiplication roundoff is caused by rounding or truncating products to the nearest representable value with a finite number of bits, which creates nonlinearities in the filter. Overflow is caused by the inability of fixed-point arithmetic to express a sum greater than some full scale value creating severe nonlinearities.

For stable digital filters implemented with infinite-precision arithmetic, if the input becomes zero and remains zero, the output will also decay asymptotically toward zero. If the same filters are implemented with finite-precision arithmetic the output may continue to oscillate indefinitely even though the input remains zero [1]. This phenomenon is referred to as a zero-input limit cycle. Zero-input limit

cycles are caused by either the nonlinear quantizers in the feedback loop of the filters or by addition overflows in the finite length registers.

If the input to a fixed-point digital filter is a constant, internal errors are highly correlated. Thus, for recursive filters there may not be a unique steady-state output for a constant input [2]. The region in which there is no-unique steady-state output is known as a deadband. The output is dependent upon where the state first encounters the deadband region. Constant-input limit cycles are caused by the rounding (or truncating) of the lowest-order bits while the input remains constant. This rounding or truncating of the lowest-order bits may create nonlinearities in the filter causing the output to oscillate within the deadband.

Limit cycles are generally undesirable since they give rise to oscillator behavior when the filter is designed for a constant steady-state response. For example, suppose a speech signal is sampled, filtered by a digital filter, and then converted back to an acoustic signal using a D/A converter. In this case it would be very undesirable for the filter to enter a limit cycle since this would produce an unwanted audible tone [1]. Another place where limit cycles are undesirable is a situation where a constant reference is applied to a control system. In this case due to the limit cycle the desired constant output would not be obtained.

Limit cycles are not only influenced by the type of input

sequence but also by the filter realization or structure, the location of the filter poles and the filter's initial state. In the past, techniques have been introduced to eliminate limit cycles. Many papers have been written using a state-space realization [3]-[6] as opposed to other realizations or structures such as the direct form realization. Bose and Brown [6] established that by placing restrictions on the location of poles that various realizations are free of limit cycles. Singh [5] used saturation arithmetic to guarantee the nonexistence of zero-input overflow limit cycles. Other techniques such as controlled rounding [7], [8], Lyapunov's direct method [9], and magnitude truncation [3], [4], [7], [10] have also been may be used to suppress zero-input limit cycles.

This thesis deals with the elimination of constant-input limit cycles in fixed-point digital filters. In [11] a method was presented that eliminates overflow limit cycles and zero-input limit cycles in fixed-point state-space digital filters. This method is based on: 1) a proper choice of the realization coefficients and 2) using magnitude truncation rather than rounding of the filter calculations. This method will subsequently be referred to as the Mills method. In [4], [12], [13] methods were proposed for modifying the Mills method so that the resulting realizations can be proven free of constant-input limit cycles. This thesis first consolidates the results of [4], [12], [13] so that the

reader is familiar with the proposed methods of eliminating limit cycles.

Although it is possible to prove that the realizations of [4], [12], [13] will not have constant-input limit cycles, it has not been shown that such limit cycles do indeed exist if only the Mills method, without the complications of the methods of [4], [12], [13], is employed. Therefore, the first question addressed by the research in this thesis is whether the methods of [4], [12], [13] are really necessary. To answer this question, computer programs were written to search for filter transfer functions that have constant-input limit cycles even though the Mills method is employed. Many transfer functions were found that do have constant-input limit cycles despite using the Mills method. Four of these transfer functions were selected at random for subsequent work. The search programs were also used to confirm that zero-input limit cycles do not exist using the Mills method.

Since the techniques of [4], [12], [13] are necessary, implementations of these methods on the Motorola DSP56000/1 family of digital signal processors were considered. These implementations were first shown to eliminate constant-input limit cycles for the four representative transfer functions. This was accomplished by running the implementations on a Motorola DSP56000/1 simulator. Then, to provide designers with information on the computational cost of eliminating limit cycles, the implementations were compared to those that

do not eliminate limit cycles. The required number of processor instruction cycles per sample processed were compared.

In Chapter II , the background equations for state-space realizations are introduced. It also discusses how realization coefficients are determined and scaled. Furthermore, Chapter II shows that any transfer function can be separated into a sum of second order sections for even order filters and a sum of second order sections plus one first order section for odd order filters. Chapter III discusses how first and second order sections can be made free of both zero and constant input limit cycles. A procedure for determining the realization coefficients and some implementation considerations are included in Chapter III. Chapter IV gives an explanation of the computer programs used to search for filter transfer functions that have constant-input limit cycles and introduce some example transfer functions that have constant-input limit cycles. Chapter V deals with the implementation of constant-input limit-cycle-free digital filters on the Motorola DSP56000/1 processor. Chapter VI discusses the computational loading required to implement the limit-cycle-free realizations on the Motorola digital signal processor and concludes by comparing the number of computational cycles required for a limit-cycle-free second order section with a second order filter that does not eliminate limit cycles. Conclusions and recommendations are

presented in Chapter VII.

CHAPTER II

STATE SPACE REALIZATION

A state space realization for a digital filter gives more than the set of difference equations for implementing the filter. It also provides a full information concerning the input/output relationship of the filter and also information on the internal state of the filter. The first state space equation gives the output signal in terms of the state variables and the second state space equation relates the state variables to the input signal. The state variables give information about the internal state of the filter. Thus, a state space realization provides more information than merely an input/output description.

Consider an N^{th} order direct form II realization

$$\begin{aligned} y(n) &= du(n) + \sum_{i=1}^N \alpha_i x(n-i+1) \\ x(n+1) &= u(n) - \sum_{i=1}^N \beta_i x(n-i+1). \end{aligned} \tag{2.1}$$

where $y(n)$ is the output, $x(n)$ is the state variable sequence, and $u(n)$ is the input. The general state space realization for the above equations can be written in matrix form as

$$\begin{bmatrix} x_1(n+1) \\ x_2(n+1) \\ x_3(n+1) \\ \vdots \\ x_{N-1}(n+1) \\ x_N(n+1) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & \dots & 0 & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 0 & 1 \\ -\beta_N & -\beta_{N-1} & -\beta_{N-2} & \dots & -\beta_2 & -\beta_1 \end{bmatrix} \begin{bmatrix} x_1(n) \\ x_2(n) \\ x_3(n) \\ \vdots \\ x_{N-1}(n) \\ x_N(n) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} u(n) \quad (2.2)$$

where $x_1(n) \dots x_N(n)$ are the elements of the state vector and $x_N(n)=x(n)$, $x_{N-1}(n)=x(n-1)$, $x_{N-2}(n)=x(n-2), \dots$. The output equation is given by

$$y(n) = [\alpha_N \ \alpha_{N-1} \ \alpha_{N-2} \ \dots \ \alpha_2 \ \alpha_1] \begin{bmatrix} x_1(n) \\ x_2(n) \\ x_3(n) \\ \vdots \\ x_{N-1}(n) \\ x_N(n) \end{bmatrix} + du(n) \quad (2.3)$$

where $y(n)$ is the output. The state equation and output equation can also be written as

$$\begin{aligned} \mathbf{x}(n+1) &= \mathbf{A}\mathbf{x}(n) + \mathbf{b}u(n) \\ y(n) &= \mathbf{c}^t \mathbf{x}(n) + du(n) \end{aligned} \quad (2.4)$$

where

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & \dots & 0 & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 0 & 1 \\ -\beta_N & -\beta_{N-1} & -\beta_{N-2} & \dots & -\beta_2 & -\beta_1 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ \vdots \\ \vdots \\ 0 \\ 1 \end{bmatrix}, \quad (2.5)$$

$$\mathbf{x}(n) = \begin{bmatrix} x_1(n) \\ x_2(n) \\ x_3(n) \\ \vdots \\ \vdots \\ \vdots \\ x_{N-1}(n) \\ x_N(n) \end{bmatrix}, \quad \mathbf{c} = \begin{bmatrix} \alpha_N \\ \alpha_{N-1} \\ \alpha_{N-2} \\ \vdots \\ \vdots \\ \vdots \\ \alpha_2 \\ \alpha_1 \end{bmatrix}, \quad (2.6)$$

and d is a scalar constant.

The state variables for a given filter are not unique. The freedom to select different forms of the same filter can be useful in the implementation of filters. Alternative realizations can be produced by transforming the state vector $\mathbf{x}(n)$ into a new state vector $\mathbf{x}'(n)$ by the linear transformation

$$\mathbf{x}'(n) = \mathbf{T}\mathbf{x}(n) \quad (2.7)$$

where $\mathbf{T}$ is any nonsingular $N \times N$ matrix. From equations (2.4) and (2.7) we have

$$\begin{aligned} \mathbf{x}'(n+1) &= \mathbf{A}'\mathbf{x}'(n) + \mathbf{b}'u(n) \\ y(n) &= \mathbf{c}'^t\mathbf{x}'(n) + d'u(n). \end{aligned} \quad (2.8)$$

The state matrices for the transformed filter are seen to be

$$\begin{aligned} \mathbf{A}' &= \mathbf{TAT}^{-1} \\ \mathbf{b}' &= \mathbf{Tb} \\ \mathbf{c}'^t &= \mathbf{c}^t\mathbf{T}^{-1} \\ d' &= d. \end{aligned} \quad (2.9)$$

The state space realization of equations (2.4) and (2.8) both have the same transfer function

$$H(z) = \mathbf{c}^t(z\mathbf{I} - \mathbf{A})^{-1}\mathbf{b} + d \quad (2.10)$$

where $\mathbf{I}$ is the $N \times N$ identity matrix.

Limit cycle behavior is only affected by the poles of a filter. Limit cycles occur because the state variables do not decay to a fixed value. The state-variable feedback relation that gives rise to limit cycles is defined by the $\mathbf{A}$ matrix. The determinant of $z\mathbf{I} - \mathbf{A}$ is the denominator of the transfer function. Therefore, the denominator (recursive) part of the transfer function determines limit cycle behavior.

Implementing an N^{th} order digital filter on a digital signal processor will cause roundoff errors which may produce limit cycles. By properly choosing $\mathbf{T}$ it is possible to suppress the occurrence of zero-input limit cycles and minimize roundoff noise but this results in nontrivial values for all elements in $\{\mathbf{A}', \mathbf{b}', \mathbf{c}', d'\}$. Since $\mathbf{A}'$ is $N \times N$ and

b' and c' are $N \times 1$, this produces $N^2 + 2N + 1$ multiplies. A more efficient approach is to perform a partial fraction expansion on the N^{th} order transfer function $H(z)$ to get

$$H(z) = d + \sum_{i=1}^q H_i(z) \quad (2.11)$$

where $q = N/2$ for even-order filters and $q = (N+1)/2$ for odd-order filters. By performing the partial fraction expansion, all $H_i(z)$ will be second-order of the form

$$H_i(z) = \frac{\alpha_1^{(i)} z + \alpha_2^{(i)}}{z^2 + \beta_1^{(i)} z + \beta_2^{(i)}} \quad (2.12)$$

except for one first-order section

$$H_i(z) = \frac{\alpha^{(i)}}{z + \beta^{(i)}} \quad (2.13)$$

in odd-order filters. This method requires $4N$ multiplies for odd-order filters and $4N + 1$ multiplies for even-order filters.

Often digital filter design techniques give $H(z)$ in the form

$$H(z) = G \frac{N_1(z)}{D_1(z)} \cdot \frac{N_2(z)}{D_2(z)} \cdots \frac{N_q(z)}{D_q(z)} \quad (2.14)$$

where

$$\begin{aligned} D_1(z) &= (z - z_1)(z - z_1^*), & z_1 &= x_1 + jy_1 \\ D_2(z) &= (z - z_2)(z - z_2^*), & z_2 &= x_2 + jy_2 \\ &\vdots & & \\ D_q(z) &= (z - z_q)(z - z_q^*), & z_q &= x_q + jy_q. \end{aligned} \quad (2.15)$$

Partial fraction expansion gives $H(z)$ in the

form

$$H(z) = d + \frac{\alpha_1^{(1)} z + \alpha_2^{(1)}}{D_1(z)} + \frac{\alpha_1^{(2)} z + \alpha_2^{(2)}}{D_2(z)} + \dots + \frac{\alpha_1^{(q)} z + \alpha_2^{(q)}}{D_q(z)}. \quad (2.16)$$

Multiplying both forms by $D_1(z_1)$ and equating gives

$$\begin{aligned} GN_1(z_1) \frac{N_2(z_1)}{D_2(z_1)} \dots \frac{N_q(z_1)}{D_q(z_1)} &= dD_1(z_1) + \alpha_1^{(1)} z + \alpha_2^{(1)} \\ &+ D_1(z_1) \frac{\alpha_1^{(2)} z + \alpha_2^{(2)}}{D_2(z)} \\ &+ \dots + D_1(z_1) \frac{\alpha_1^{(q)} z + \alpha_2^{(q)}}{D_q(z)} \end{aligned} \quad (2.17)$$

Thus, to find $\alpha_1^{(1)}$ and $\alpha_2^{(1)}$ we evaluate

$$GN_1(z_1) \frac{N_2(z_1)}{D_2(z_1)} \dots \frac{N_q(z_1)}{D_q(z_1)} = u_1 + jv_1. \quad (2.18)$$

Then

$$\alpha_1^{(1)} (x_1 + jy_1) + \alpha_2^{(1)} = u_1 + jv_1 \quad (2.19)$$

and

$$\begin{aligned} \alpha_1^{(1)} &= v_1 / y_1 \\ \alpha_2^{(1)} &= u_1 - \alpha_1^{(1)} x_1. \end{aligned} \quad (2.20)$$

To find $\alpha_1^{(2)}$ and $\alpha_2^{(2)}$ we evaluate

$$G \frac{N_1(z_2)}{D_1(z_2)} N_2(z_2) \frac{N_3(z_2)}{D_3(z_2)} \dots \frac{N_q(z_2)}{D_q(z_2)} = u_2 + jv_2, \quad (2.21)$$

then

$$\alpha_1^{(2)} = v_2 / y_2$$

(2.22)

$$\alpha_2^{(2)} = u_2 - \alpha_1^{(2)} x_2.$$

This process is repeated until all $\alpha_1^{(i)}$ and $\alpha_2^{(i)}$ values are obtained. After all of the α values are determined, d can be obtained by substituting any value for z which is not a zero or pole into both forms of the equation.

It is necessary for a state-space realization to be scaled to insure that a full scale input signal gives no more than a full scale signal at the state variable nodes. Otherwise, the state variables may overflow and produce distortion and overflow limit cycles. For a realization to possess balanced L_p scaling [14] it is necessary that

$$\|F_i(z)\|_p = 1, \quad i=1, 2, \dots, n \quad (2.23)$$

where $F_i(z)$ is the transfer function from the filter input to the i^{th} state variable node, $\|\cdot\|_p$ represents the L_p norm and

$$F(z) = \frac{X(z)}{U(z)} = (zI - A)^{-1}b = \begin{bmatrix} F_1(z) \\ F_2(z) \\ \cdot \\ \cdot \\ F_n(z) \end{bmatrix}. \quad (2.24)$$

In the important case where $p=\infty$,

$$\|F_i(z)\|_\infty = \max_\omega |F_i(e^{j\omega})|. \quad (2.25)$$

Thus, the L_∞ -norm scales the realization for a peak frequency response of unity from the filter input to each state

variable. This guarantees that overflow will not happen for sinusoidal inputs. In the remainder of the thesis, L_∞ -norm scaling will be used.

CHAPTER III

ELIMINATING LIMIT CYCLES

This chapter illustrates how first and second-order state-space realizations can be implemented without constant-input limit cycles. It also shows that zero-input limit cycles can be eliminated more easily than constant-input limit cycles. Derivations from [11] and [13] are combined to give a complete process for eliminating both zero-input and constant-input limit cycles.

First-Order Section

As shown in Chapter II, a first-order section can be written as

$$H(z) = \frac{\alpha}{z + \beta}. \quad (3.1)$$

This will be realized by the first-order state equation

$$\begin{aligned} y(n) &= cx(n) \\ x(n+1) &= ax(n) + bu(n) \end{aligned} \quad (3.2)$$

where $x(n)$ is the state variable and $\{ a, b, c \}$ are the realization coefficients. From (2.10), equation (3.2) has the transfer function

$$H(z) = \frac{Y(z)}{U(z)} = \frac{bc}{z - a}. \quad (3.3)$$

Equating (3.1) and (3.3) gives

$$\begin{aligned} a &= -\beta \\ bc &= \alpha. \end{aligned} \tag{3.4}$$

Proper choice of b will eliminate limit cycles and scale the realization. Then, the value of c can be chosen for equality in (3.4).

Limit cycles can be eliminated from a first order transfer function by choosing

$$b = \lambda(a-1) \tag{3.5}$$

where λ is a scaling multiplier which will be used to scale the realization. Using (3.5) in (3.2) the state equation becomes

$$\begin{aligned} y(n) &= cx(n) \\ x(n+1) &= a[x(n) + \lambda u(n)] - \lambda u(n) \end{aligned} \tag{3.6}$$

To prevent limit cycles in the implementation, (3.6) is implemented on a digital signal processor as [13]

$$\begin{aligned} y(n) &= q_r[cx(n)] \\ \hat{u}(n) &= q_r[\lambda u(n)] \\ x(n+1) &= q_m\{ a[x(n) + \hat{u}(n)] \} - \hat{u}(n) \end{aligned} \tag{3.7}$$

where $q_m\{\}$ represents quantization by magnitude truncation (closest representable value with fewer bits and a smaller magnitude) and $q_r\{\}$ represents quantization by rounding (closest representable value with fewer bits).

The state update equation for $u(n) = u_0 = \text{constant}$ can be written as

$$x(n+1) + \hat{u}_0 = q_m\{a[x(n) + \hat{u}_0]\} \quad (3.8)$$

where $\hat{u}_0 = q_r\{\lambda u_0\}$. Letting

$$\hat{x}(n) = x(n) + \hat{u}_0 \quad (3.9)$$

(3.8) can be written as

$$\hat{x}(n+1) = q_m\{a\hat{x}(n)\}. \quad (3.10)$$

Notice that this has the form of a zero-input state update equation. Limit cycles will not occur if $x(n)$ in (3.9) goes to $-\hat{u}_0$ as $\hat{x}(n)$ decays to zero in response to the constant input. For $\hat{x}(n)$ to decay to zero it is sufficient that

$$|\hat{x}(n+1)| < |\hat{x}(n)| \quad (3.11)$$

However $|q_m\{p\}| \leq |p|$ and, for a stable filter, $|a| < 1$, so

$$\begin{aligned} |\hat{x}(n+1)| &= |q_m\{a\hat{x}(n)\}| \leq |a\hat{x}(n)| \\ &\leq |a| |\hat{x}(n)| \\ &< |\hat{x}(n)| \end{aligned} \quad (3.12)$$

so constant-input limit cycles can not exist.

Notice that since (3.10) has the form of a zero-input state-update equation, the conventional state update equation of (3.2), using magnitude truncation, will have no zero-input limit cycles. Thus, to eliminate zero-input limit cycles it is not necessary to use the more complicated state update equation (3.7). However, to mathematically guarantee no constant-input limit cycles (3.7) must be used. It remains to be shown in the next chapter that constant input limit cycles

do exist when the conventional state update equation is used instead of (3.7).

To prevent state variable overflow, λ will be chosen so that the frequency response from the input to the state variable is unity (L_∞ -norm scaling). From (2.24), (3.4) and (3.5)

$$F(z) = \frac{X(z)}{Y(z)} = \frac{b}{z+\beta} = \frac{-\lambda(\beta+1)}{z+\beta}. \quad (3.13)$$

The square of the magnitude response is then

$$|F(e^{jw})|^2 = \frac{\lambda^2(1+\beta)^2}{1+2\beta\cos(w)+\beta^2}. \quad (3.14)$$

The peak value of the magnitude response is

$$|F(e^{jw})|_{\max} = \frac{\lambda(1+\beta)}{1-|\beta|} = 1 \quad (3.15)$$

giving

$$\lambda = \frac{1-|\beta|}{1+\beta}. \quad (3.16)$$

From (3.4), (3.5) and (3.16), the coefficients for realizing a first order section without constant-input limit cycles using (3.7) are

$$\begin{aligned} a &= -\beta \\ b &= |\beta| - 1 \\ c &= \frac{\alpha}{|\beta| - 1} \\ \lambda &= \frac{1-|\beta|}{1+\beta}. \end{aligned} \quad (3.17)$$

Second Order Sections

Consider the realization of a second-order transfer function with complex conjugate poles, $p, p^* = re^{\pm j\theta}$, of the form

$$H(z) = \frac{Y(z)}{X(z)} = d + \frac{\alpha_1 z + \alpha_2}{z^2 + \beta_1 z + \beta_2} \quad (3.18)$$

where $\beta_1 = 2r \cos \theta$ and $\beta_2 = r^2$. This transfer function can be realized by the state equation of (2.14) with

$$\begin{aligned} \mathbf{A} &= \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \\ \mathbf{b} &= \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \\ \mathbf{c} &= \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} \end{aligned} \quad (3.19)$$

By comparing (3.18) and (2.10), the following relations between the transfer function coefficients in (3.18) and the realization coefficients of $\{\mathbf{A}, \mathbf{b}, \mathbf{c}\}$ are obtained [15]:

$$\begin{aligned} \alpha_1 &= c_1 b_1 + c_2 b_2, \\ \alpha_2 &= c_1 b_2 a_{12} + c_2 b_1 a_{21} - c_1 b_1 a_{22} - c_2 b_2 a_{11}, \\ \beta_1 &= -(a_{11} + a_{22}), \\ \beta_2 &= a_{11} a_{22} - a_{12} a_{21}. \end{aligned} \quad (3.20)$$

$\mathbf{b}$ can be chosen to eliminate limit cycles and scale the realization. Then, $\mathbf{c}$ can be chosen for equality in (3.20).

To eliminate limit cycles in a second order section, choose

$$\mathbf{b} = \lambda (\mathbf{A} - \mathbf{I}) \mathbf{v} \quad (3.21)$$

where

$$\mathbf{v} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad (3.22)$$

The scaling multiplier λ will be chosen to scale the realization.

Equation (3.21) is equivalent to

$$\mathbf{b} = \lambda \begin{bmatrix} a_{11} - 1 \\ a_{21} \end{bmatrix}. \quad (3.23)$$

Substituting (3.21) into (2.4), the state equation becomes

$$y(n) = \mathbf{c}^t \mathbf{x}(n) \quad (3.24)$$

$$\mathbf{x}(n+1) = \mathbf{A}[\mathbf{x}(n) + \lambda u(n) \mathbf{v}] - \lambda u(n) \mathbf{v}.$$

To eliminate constant-input limit cycles, this is implemented on a digital signal processor as [13]

$$\begin{aligned} y(n) &= q_r \{ \mathbf{c}^t \mathbf{x}(n) \} \\ \hat{\mathbf{u}}(n) &= q_r \{ \lambda u(n) \mathbf{v} \} \\ \mathbf{x}(n+1) &= q_m \{ \mathbf{A}[\mathbf{x}(n) + \hat{\mathbf{u}}(n)] \} - \hat{\mathbf{u}}(n) \end{aligned} \quad (3.25)$$

For $u(n) = u_0 = \text{constant}$, the state update equation of (3.25) can be written

$$\mathbf{x}(n+1) + \hat{\mathbf{u}}_0 = q_m \{ \mathbf{A}[\mathbf{x}(n) + \hat{\mathbf{u}}_0] \} \quad (3.26)$$

where $\hat{\mathbf{u}}_0 = q_r \{ \lambda u_0 \mathbf{v} \}$. Letting

$$\hat{x}(n) = x(n) + \hat{u}_0 \quad (3.27)$$

equation (3.26) can be written

$$\hat{x}(n+1) = q_m \{A[\hat{x}(n)]\}. \quad (3.28)$$

Notice that this has the form of a zero-input state-update equation. If $\hat{x}(n)$ decays to zero in response to the constant input, then $x(n)$ goes to $-\hat{u}_0$ and limit cycles will not occur.

A sufficient condition for $\hat{x}(n)$ to decay to zero using magnitude truncation was developed in [11] to show how zero-input limit cycles and overflow oscillations could be eliminated. $\hat{x}(n)$ will decay to zero if

$$||\hat{x}(n+1)|| < ||\hat{x}(n)|| \quad (3.29)$$

where

$$||\hat{x}(n)|| = [\hat{x}^T D \hat{x}]^{1/2}, \quad D = \begin{bmatrix} d_1^2 & 0 \\ 0 & d_2^2 \end{bmatrix}, \quad (3.30)$$

and d_1^2 and d_2^2 are strictly positive. This will happen if

$$||A\hat{x}(n)|| < ||\hat{x}(n)|| \quad (3.31)$$

since

$$|q_m(v)| \leq |v|, \quad (3.32)$$

$$||q_m(A\hat{x}(n))|| \leq ||A\hat{x}(n)|| \quad (3.33)$$

$$||q_m(A\hat{x}(n))|| \leq ||A\hat{x}(n)|| < ||\hat{x}(n)||. \quad (3.34)$$

Therefore, the requirement for no limit cycles is

$$||A\hat{x}(n)|| < ||\hat{x}(n)|| \quad (3.35)$$

$$||A\hat{x}(n)||^2 < ||\hat{x}(n)||^2 \quad (3.36)$$

$$||A\hat{x}(n)||^2 - ||\hat{x}(n)||^2 < 0 \quad (3.37)$$

$$\hat{x}^t(n) D \hat{x}(n) - \hat{x}^t(n) A^t D A \hat{x}(n) > 0 \quad (3.38)$$

$$\hat{x}^t(n) [D - A^t D A] \hat{x}(n) > 0 \quad (3.39)$$

for all $\hat{x}(n)$. Thus, it is necessary and sufficient that the matrix

$$D - A^t D A \quad (3.40)$$

be positive definite.

In the following, the conditions for

$$\hat{x}^t(n) [D - A^t D A] \hat{x}(n) > 0 \quad (3.41)$$

for a second-order section are developed. Let

$$\hat{x} = Tq \quad (3.42)$$

where

$$T = \begin{bmatrix} \frac{1}{d_1} & 0 \\ 0 & \frac{1}{d_2} \end{bmatrix} \quad (3.43)$$

and

$$T^{-2} = \begin{bmatrix} d_1^2 & 0 \\ 0 & d_2^2 \end{bmatrix} = D \quad (3.44)$$

Now in equation (3.42) q can be chosen to be any vector. Therefore, we have not restricted the choices for x . The corresponding q for any x is

$$q = T^{-1}x = \begin{bmatrix} d_1 & 0 \\ 0 & d_2 \end{bmatrix} x. \quad (3.45)$$

Therefore equation (3.41) can be rewritten as

$$(Tq)^t (D - A^t D A) Tq > 0 \quad (3.46)$$

$$q^t T^t (T^{-2} - A^t T^{-2} A) Tq > 0 \quad (3.47)$$

$$q^t [I - (T^{-1} A T)^t (T^{-1} A T)] q > 0 \quad (3.48)$$

$$q^t (I - V^t V) q > 0 \quad (3.49)$$

where

$$V = T^{-1} A T = \begin{bmatrix} a_{11} & a_{12} \frac{d_1}{d_2} \\ a_{21} \frac{d_2}{d_1} & a_{22} \end{bmatrix} \quad (3.50)$$

Next, we need $I - M$ to be positive definite where

$$M = V^t V = \begin{bmatrix} m_{11} & m_{12} \\ m_{12} & m_{22} \end{bmatrix}. \quad (3.51)$$

For a 2x2 matrix to be positive definite it is necessary and sufficient that

$$\begin{aligned}\lambda_1 + \lambda_2 &> 0 \\ \lambda_1 \lambda_2 &> 0\end{aligned}\tag{3.52}$$

where λ_1 and λ_2 are the matrix eigenvalues. If λ_1 and λ_2 are the eigenvalues of $I - M$, we require

$$\begin{aligned}\text{Trace}(I-M) &= \lambda_1 + \lambda_2 > 0 \\ \text{Det}(I-M) &= \lambda_1 \lambda_2 > 0.\end{aligned}\tag{3.53}$$

Examining the $\text{Det}(I-M)$,

$$\begin{aligned}\text{Det}(I-M) &= \text{Det} \begin{bmatrix} 1-m_{11} & -m_{12} \\ -m_{21} & 1-m_{22} \end{bmatrix} \\ &= 1 - m_{11} - m_{22} + m_{11}m_{22} - m_{12}m_{21} \\ &= 1 - \text{Trace}(M) + \text{Det}(M).\end{aligned}\tag{3.54}$$

From (3.50) it can be seen that V has the same eigenvalues and determinant as A . Thus, since

$$M = V^t V \tag{3.55}$$

then

$$\text{Det}(M) = \text{Det}(V^t) \text{Det}(V) = [\text{Det}(V)]^2 = [\text{Det}(A)]^2. \tag{3.56}$$

Hence,

$$\text{Det}(I-M) = 1 + [\text{Det}(A)]^2 - \text{Trace}(M). \tag{3.57}$$

For stable filters with complex poles

$$\text{Det}(A) = pp^* = |p|^2 < 1. \tag{3.58}$$

For real poles

$$\text{Det}(\mathbf{A}) = p_1 p_2 < 1 \quad (3.59)$$

where $p_1 < 1$ and $p_2 < 1$. Therefore, in general

$$\begin{aligned} \text{Det}(\mathbf{A}) &< 1 \\ [\text{Det}(\mathbf{A})]^2 &< 1 \\ 1 + [\text{Det}(\mathbf{A})]^2 &< 2 \end{aligned} \quad (3.60)$$

so

$$\text{Det}(\mathbf{I}-\mathbf{M}) < 2 - \text{Trace}(\mathbf{M}). \quad (3.61)$$

Now, examining the $\text{Trace}(\mathbf{I}-\mathbf{M})$:

$$\begin{aligned} \text{Trace}(\mathbf{I}-\mathbf{M}) &= \text{Trace} \begin{pmatrix} 1-m_{11} & -m_{12} \\ -m_{21} & 1-m_{22} \end{pmatrix} \\ &= 2 - (m_{11} + m_{22}) \\ &= 2 - \text{Trace}(\mathbf{M}) \\ &> \text{Det}(\mathbf{I}-\mathbf{M}) \end{aligned} \quad (3.62)$$

Since $\text{Trace}(\mathbf{I}-\mathbf{M}) > \text{Det}(\mathbf{I}-\mathbf{M})$ we need only to insure that

$$\text{Det}(\mathbf{I}-\mathbf{M}) > 0 \quad (3.63)$$

to satisfy (3.52) and (3.53). If (3.63) is true then the trace must be positive.

Continuing the analysis of $\text{Det}(\mathbf{I}-\mathbf{M})$:

$$\text{Det}(\mathbf{I}-\mathbf{M}) = 1 + [\text{Det}(\mathbf{A})]^2 - \text{Trace}(\mathbf{M}) \quad (3.64)$$

Since the $\text{Det}(\mathbf{A})$ is not a function of d_1 and d_2 we will choose d_1 and d_2 to minimize $\text{Trace}(\mathbf{M})$ and hence maximize $\text{Det}(\mathbf{I}-\mathbf{M})$. If this maximized value of $\text{Det}(\mathbf{I}-\mathbf{M})$ is greater than zero, then

there is a d_1 and d_2 for which $D-A'DA$ is positive definite.

Letting

$$\tau = \frac{d_2}{d_1} > 0 \quad (3.65)$$

gives, from (3.50) and (3.51),

$$\text{Trace}(M) = a_{11}^2 + a_{21}^2 \tau^2 + a_{22}^2 + a_{12}^2 \frac{1}{\tau^2} \quad (3.66)$$

The $\text{Trace}(M)$ is positive and it approaches infinity as τ approaches either zero or infinity. Therefore,

$$\frac{d\text{Trace}(M)}{d\tau} = 2a_{21}^2 \tau - 2a_{12}^2 \tau^{-3} = 0$$

$$\tau^4 = \frac{a_{12}^2}{a_{21}^2} \quad (3.67)$$

$$\tau^2 = \left| \frac{a_{12}}{a_{21}} \right|$$

minimizes $\text{Trace}(M)$ and

$$\begin{aligned} \text{Trace}(M) |_{\min} &= a_{11}^2 + a_{21}^2 \left| \frac{a_{12}}{a_{21}} \right| + a_{22}^2 + a_{12}^2 \left| \frac{a_{21}}{a_{12}} \right| \\ &= a_{11}^2 + |a_{21}|^2 \frac{|a_{12}|}{|a_{21}|} + a_{22}^2 + |a_{12}|^2 \frac{|a_{21}|}{|a_{12}|} \\ &= a_{11}^2 + |a_{21}| |a_{12}| + a_{22}^2 + |a_{12}| |a_{21}| \\ &= a_{11}^2 + a_{22}^2 + 2|a_{12}a_{21}|. \end{aligned} \quad (3.68)$$

This gives

$$\text{Det}(I-M) \leq 1 + [\text{Det}(A)]^2 - a_{11}^2 - a_{22}^2 - 2|a_{12}a_{21}|, \quad (3.69)$$

so for a D to exist for which $D-A'DA$ is positive definite, it

is necessary that

$$1 + [\text{Det}(\mathbf{A})]^2 - a_{11}^2 - a_{22}^2 - 2|a_{12}a_{21}| > 0. \quad (3.70)$$

The analysis for both real and complex poles could be continued but it is assumed that if real poles are present they will be implemented as two first-order sections. Therefore, the focus for second-order sections is the case of a pair of complex conjugate poles. In this case, if $D(z)$ is the transfer function denominator polynomial then

$$\begin{aligned} D(z) &= \text{Det}(z\mathbf{I} - \mathbf{A}) \\ &= \begin{vmatrix} z - a_{11} & -a_{12} \\ -a_{21} & z - a_{22} \end{vmatrix} \\ &= z^2 - (a_{11} + a_{22})z + a_{11}a_{22} - a_{12}a_{21}. \end{aligned} \quad (3.71)$$

In order to have complex poles from a quadratic equation it is necessary for

$$\begin{aligned} (a_{11} + a_{22})^2 - 4(a_{11}a_{22} - a_{12}a_{21}) &< 0 \\ a_{11}^2 + a_{22}^2 + 2a_{11}a_{22} - 4a_{11}a_{22} + 4a_{12}a_{21} &< 0 \\ a_{11}^2 + a_{22}^2 - 2a_{11}a_{22} + 4a_{12}a_{21} &< 0 \\ (a_{11} - a_{22})^2 + 4a_{12}a_{21} &< 0. \end{aligned} \quad (3.72)$$

Therefore, for complex-conjugate poles $a_{12}a_{21} < 0$ and

$$|a_{12}a_{22}| = -a_{12}a_{21} \quad (3.73)$$

giving

$$\begin{aligned}
1 + [\text{Det}(\mathbf{A})]^2 - a_{11}^2 - a_{22}^2 + 2a_{12}a_{21} &> 0 \\
[1 - \text{Det}(\mathbf{A})]^2 + 2\text{Det}(\mathbf{A}) - a_{11}^2 - a_{22}^2 + 2a_{12}a_{21} &> 0 \\
[1 - \text{Det}(\mathbf{A})]^2 + 2a_{11}a_{22} - 2a_{12}a_{21} - a_{11}^2 - a_{22}^2 + 2a_{12}a_{21} &> 0 \\
[1 - \text{Det}(\mathbf{A})]^2 - (a_{11} - a_{22})^2 &> 0 \quad (3.74) \\
[1 - \text{Det}(\mathbf{A})]^2 &> (a_{11} - a_{22})^2 \\
1 - \text{Det}(\mathbf{A}) &> |a_{11} - a_{22}|Z \\
\text{Det}(\mathbf{A}) + |a_{11} - a_{22}| &< 1
\end{aligned}$$

Since

$$\text{Det}(\mathbf{A}) = |p|^2 \quad (3.75)$$

and

$$p = re^{j\theta} \quad (3.76)$$

where r is the pole radius. Therefore,

$$|a_{11} - a_{22}| + r^2 < 1. \quad (3.77)$$

Thus, choosing

$$a_{11} = a_{22} \quad (3.78)$$

and using magnitude truncation means that limit cycles will not exist for any stable second order section.

Notice that since (3.28) has the form of a zero-input state update equation, if $a_{11} = a_{22}$ then the conventional state update equation of (2.4) using magnitude truncation, without the complications of (3.25), will have no zero-input limit cycles, as explained in [11]. However, to mathematically guarantee no constant-input limit cycles, (3.25) must be used.

It remains to be shown in the next chapter that constant-input limit cycles can exist when the conventional state update equation of (2.4) with magnitude truncation is used.

To determine the realization coefficients in terms of the transfer function coefficients, (3.78) is substituted into (3.20) giving

$$\begin{aligned} a_{11} &= a_{22} = -\frac{\beta_1}{2} \\ a_{12}a_{21} &= \left(\frac{\beta_1}{2}\right)^2 - \beta_2 \end{aligned} \quad (3.79)$$

Next, the values of λ and a_{21} are chosen so that the peak values of the frequency responses from the input to the state variables are unity.

The transfer functions from the input to the state variables are, from (2.24),

$$\begin{aligned} F_1(z) &= \frac{b_1 z + (a_{12}b_2 - a_{22}b_1)}{z^2 + \beta_1 z + \beta_2} \\ F_2(z) &= \frac{b_2 z + (a_{21}b_1 - a_{11}b_2)}{z^2 + \beta_1 z + \beta_2} \end{aligned} \quad (3.80)$$

Substituting (3.23) and (3.79) into (3.80) gives

$$\begin{aligned} F_1(z) &= \lambda \left[\frac{-\left(1 + \frac{\beta_1}{2}\right)z - \left(\frac{\beta_1}{2} + \beta_2\right)}{z^2 + \beta_1 z + \beta_2} \right] = \lambda R_1(z) \\ F_2(z) &= \lambda a_{21} \left[\frac{z-1}{z^2 + \beta_1 z + \beta_2} \right] = \lambda a_{21} R_2(z) \end{aligned} \quad (3.81)$$

where

$$R_1(z) = \frac{-\left(1 + \frac{\beta_1}{2}\right)z - \left(\frac{\beta_1}{2} + \beta_2\right)}{z^2 + \beta_1 z + \beta_2} \quad (3.82)$$

$$R_2(z) = \frac{z-1}{z^2 + \beta_1 z + \beta_2}$$

To scale the state variables it is required that

$$|F_1(e^{j\omega})|_{\max} = |F_2(e^{j\omega})|_{\max} = 1. \quad (3.83)$$

This can be achieved by requiring

$$\lambda = \frac{1}{|R_1(e^{j\omega})|_{\max}} \quad (3.84)$$

$$a_{21} = \frac{1}{\lambda |R_2(e^{j\omega})|_{\max}}$$

Since both $R_1(z)$ and $R_2(z)$ have the form

$$R(z) = \frac{\gamma_1 z + \gamma_2}{z^2 + \beta_1 z + \beta_2} \quad (3.85)$$

their peak response can be determined from [16]

$$|R(e^{j\omega})|_{\max} = \sqrt{\frac{\gamma_1^2 + \gamma_2^2 + 2\gamma_1\gamma_2\epsilon}{4\beta_2[(\epsilon - q_2)^2 + q_3]}}$$

$$\epsilon = \begin{cases} \text{Sat}(q_2), & \gamma_1\gamma_2 = 0 \\ \text{Sat}\left\{q_1 \left[-1 + \sqrt{\left(1 + \frac{q_2}{q_1}\right)^2 + \frac{q_3}{q_1^2}} \right] \right\}, & \gamma_1\gamma_2 \neq 0 \end{cases}$$

$$\text{Sat}(x) = \begin{cases} 1, & x > 1 \\ x, & -1 \leq x \leq 1 \\ -1, & x < -1 \end{cases} \quad (3.86)$$

$$q_1 = \frac{\gamma_1^2 + \gamma_2^2}{2\gamma_1\gamma_2}, \quad \gamma_1\gamma_2 \neq 0$$

$$q_2 = \frac{-\beta_1(1 + \beta_2)}{4\beta_2}$$

$$q_3 = \left(1 - \frac{\beta_1^2}{4\beta_2}\right) \frac{(1 - \beta_2)^2}{4\beta_2}$$

From (3.79)

$$a_{12} = \frac{\left(\frac{\beta_1}{2}\right)^2 - \beta_2}{a_{21}}. \quad (3.87)$$

Substituting the known values of **A** and **b** into (3.20) gives

$$c_1 = \frac{-(\alpha_1 + \alpha_2)}{\lambda(1 + \beta_1 + \beta_2)}$$

$$c_2 = \frac{1}{\lambda a_{21}} \left[\alpha_1 - \frac{\left(1 + \frac{\beta_1}{2}\right)(\alpha_1 + \alpha_2)}{1 + \beta_1 + \beta_2} \right]. \quad (3.88)$$

CHAPTER IV

DEVELOPMENT OF EXAMPLES OF LIMIT CYCLES

As was stated earlier, the Mills method presented in [11] eliminates zero-input limit cycles in fixed-point state-space digital filters. In [4], [12], [13] methods were proposed for modifying the Mills method so that the resulting realizations are free of constant-input limit cycles. Now it must be shown that constant-input limit cycles exist in a realization using the Mills method without the modification of [4], [12], or [13]. To do this, computer programs were written to search for transfer functions with constant-input limit cycles in a realization free of zero-input limit cycles employing only the Mills method. These programs only considered all pole filters since the zeros do not affect limit cycle behavior.

PC Programs for finding limit cycles

FORTTRAN programs, LIMITCY (see appendix A) and MYT (see appendix B) were used to search for second order digital filters with constant-input limit cycles even though the Mills method had been used to eliminate zero-input limit cycles. These programs were run on an IBM compatible computer using Microsoft Fortran version 5.1. The program LIMITCY searches

for transfer functions with constant-input limit cycles. The program MYT determines if a particular transfer function has constant-input limit cycles. It should be noted that the programs LIMITCY and MYT did not find any second order digital filters with zero input limit cycles.

The programs LIMITCY and MYT use two different number representations, integer and fractional. The integer representation has a range of $-32768 \leq x < 32768$ whereas the fractional representation has a range of $-1 \leq x < 1$. The only difference in their two representations is the location of the decimal point. In the following discussion the initial states are given in fractional representation but limit cycles are shown in integer representation to make it easier to see the limit cycle. When represented as integers, the actual fractional values are found by dividing by 32768.

The program LIMITCY uses a do loop statement to select the pole radius from 0.9 to 0.999 in increments of 0.001 and another do loop statement takes the selected value of the radius and scans a grid from 1° to 179° by steps of 1° for the value of the pole angle. The program MYT requires the values of β_1 and β_2 from a transfer function of the form in (3.18) to be entered and internally computes the values of the pole radius and angle.

For each radius, angle pair, the main program calculates a realization with all real-valued coefficients. This realization is

$$A = \begin{bmatrix} -\frac{\beta_1}{2} & 1 \\ \left(\frac{\beta_1}{2}\right)^2 - \beta_2 & -\frac{\beta_1}{2} \end{bmatrix} \quad (4.1)$$

$$b = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad (4.2)$$

$$c = \begin{bmatrix} \alpha_2 - \frac{\alpha_1 \beta_1}{2} \\ \alpha_1 \end{bmatrix} \quad (4.3)$$

In this realization $a_{11} = a_{22}$ so zero-input limit cycles do not exist with a conventional state-update equation and magnitude truncation. Then, the main program calls the subroutine SCALELI (see appendix C) which calculates the L_∞ norm using a technique developed by Bomar and Joseph [16]. Upon returning from subroutine SCALELI, the main program rounds each realization coefficient and the input to the nearest representable value with fewer bits. In this experiment the input was selected to be 0.01. Next, the program enters another do loop. Within this loop random values for the state variables are selected and scaled to the nearest integer. Then the main program calls the subroutine IFILNM (see appendix D) which moves the state vector a thousand samples ahead without computing the outputs. It was found that moving the state vector ahead a thousand samples was an adequate number of samples to allow the state vector to settle

to a constant if there were no limit cycles. When the program returns from subroutine IFILNM, one of the updated state variables is stored in memory. Then the program re-enters subroutine IFILNM to move the state vector further ahead. Next, the main program performs a comparison between the last updated state variable and the updated state variable stored in memory. If they are equal the program returns to the beginning of the do loop and selects another pair of random state variables. Otherwise, limit cycles exist for this radius, theta, and input. If this happens, the program will repeatedly enter subroutine IFILNM in a loop, advancing the state vector one sample each time. The values of the state variables , x_1 and x_2 , are printed in an output file so their limit cycle pattern can be observed. A block diagram of LIMITCY is given in figure 1 to illustrate the procedure.

On running the program LIMITCY, many transfer functions were found that have constant input limit cycles for many different nonzero states and initial inputs. Four of these will be identified in subsequent sections.

Coefficient Programs

Since constant-input limit cycles were found it was necessary to realize filters using (3.7) and (3.25). Programs were written to calculate the realization coefficients using (3.17), and (3.79)-(3.88). FORTRAN programs, COEFFIR and COEFSEC (see appendix E and F) were written to calculate the

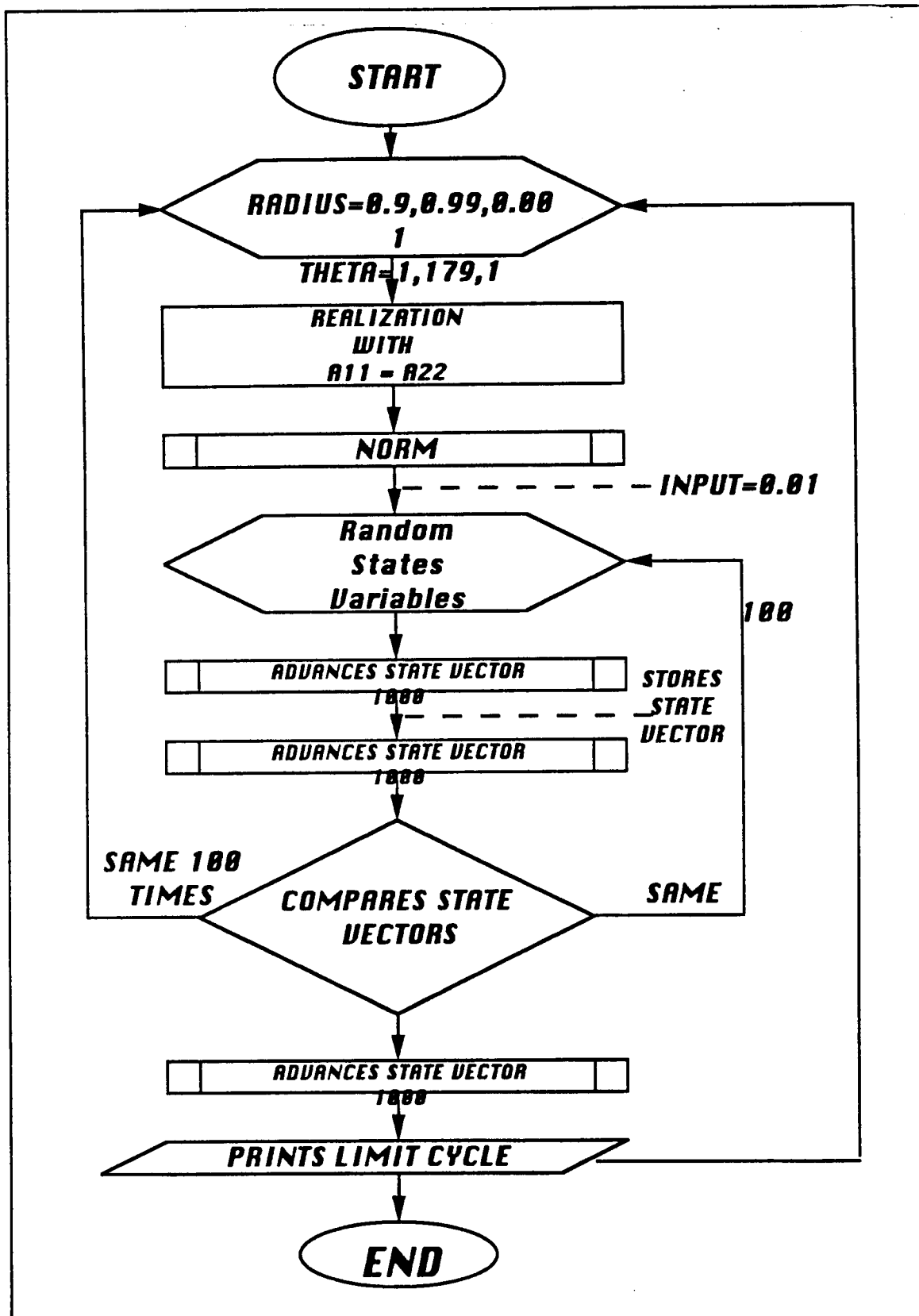


Figure 1: Flowchart for LIMITCY program

state space realizations for the first and second order sections presented in this thesis. The programs were run on an IBM compatible computer using Microsoft FORTRAN version 5.1. COEFFIR calculates the coefficient values of the first order section. This program requires a keyboard entry of β and α from a transfer function of the form in (3.1). It uses equations (3.17) to calculate the coefficients and stores the values of the coefficients into an output file, FIRST.OUT. COEFSEC calculates the state space coefficients for second order transfer function of the form in equation (3.18). COEFSEC reads the parameters of the transfer function from a data file. The program implements equations (3.79), (3.82), (3.84), (3.86), (3.87), and (3.88) to calculate the coefficients. A section of this program also scales the value of c , so its elements are less than one, if necessary. Coefficients less than one are often required for implementation on a digital signal processor. COEFSEC writes the coefficients into a output file.

Transfer functions and implementations

Many transfer functions were found to have constant input limit cycles for many nonzero inputs and initial states. Four of these were selected at random for subsequent work.

The first transfer function was found using the program LIMITCY. It has a peak response of unity. The transfer

function is

$$H_1(z) = \frac{1}{z^2 - 1.707709z + 0.9025}.$$

The output of LIMITCY gives the following sequence of the state variables for one period

x_1^i {71, 72, 72, 72, 72, 71, 70, 69, 69, 69, 69, 70}

x_2^i {26, 26, 25, 24, 23, 23, 23, 23, 24, 25, 26, 26}

when the initial states were

$x_1^f = 0.8337919$

$x_2^f = -0.1149269.$

Using the equations developed in Chapter III, this transfer function can be implemented free of constant-input limit cycles using the coefficients in Table 1.

Table 1. Coefficients for $H_1(z)$

GAM	=	0.09702133
c1	=	-0.94487214
c2	=	-0.99951607
a11	=	0.8538545
a12	=	-0.1834613
a21	=	0.9453357
a22	=	0.8538545
scale	=	28

The second transfer function was also found using the program LIMITCY. The value of α_1 was set to zero and the value of α_2 was selected in such away to give the transfer function a dc gain of one. The transfer function is

$$H_2(z) = \frac{0.456837}{z^2 - 1.493787z + 0.950624}$$

The output of LIMITCY gives the following sequence for the state variables for one period

x_1^i {22,28,31,30,25,19,14,13,16}

x_2^i {18,16,11,5,1,1,5,11,16}

when the initial states were

$x_1^f = 0.9999694$

$x_2^f = 0.9998168.$

Using the equations developed in Chapter III, this transfer function can be implemented free of constant-input limit cycles using the coefficients in Table 2.

Table 2. Coefficients for $H_2(z)$

GAM	=	0.04935468
c1	=	-0.92097772
c2	=	-0.94500136
a11	=	0.7468935
a12	=	-0.4030196
a21	=	0.9745782
a22	=	0.7468935
scale	=	11

The third transfer function was found to have constant-input limit cycles by using the MYT program. This transfer function is a second order lowpass Chebyshev digital filter with passband edge of 0.35 and passband deviation of 0.45 db. The transfer function is

$$H_3(z) = \frac{0.566831z + 0.335598}{z^2 + 1.023505z + 0.421856}$$

The output of MYT gives the following sequence of the state variables for one period

x_1^i {47,47,48}

x_2^i {110,111,110}

when the initial states were

$x_1^f = 0.9999694$

$x_2^f = 0.9998168.$

Using the equations developed in Chapter III, this transfer function can be implemented free of constant-input limit cycles using the coefficients in Table 3.

Table 3. Coefficients for $H_3(z)$

GAM	=	0.5458228
c1	=	-0.6761116
c2	=	0.2365448
a11	=	-0.5117525
a12	=	-0.2608725
a21	=	0.6131939
a22	=	-0.5117525
scale	=	1

The fourth transfer function was also selected by the program MYT. This transfer function is a fifth order highpass Chebyshev digital filter with passband edge of 0.14 and passband deviation of 0.5 db. The transfer function is

$$H(z) = \frac{0.013547642z^2 - 0.047596084z + 0.028850883}{z^5 - 1.589611z^4 + 1.682092707z^3 - 0.713784839z^2 + 0.241565521z + 0.047634177}$$

This transfer function can be separated into two second order sections and one first order section. The first second order section is

$$H_4(z) = \frac{-0.148479z + 0.406006}{z^2 - 0.52427z + 0.433577}$$

The output of MYT gives the following sequence of the state variables for one period

$$x_1^i \quad \{186, 186, 187, 187\}$$

$$x_2^i \quad \{149, 150, 150, 149\}$$

when the initial states were

$$x_1^f = 0.9999694$$

$$x_2^f = 0.9998168.$$

Using the equations developed in Chapter III, this transfer function can be implemented free of constant-input limit cycles using the coefficients in Table 4.

Table 4. Coefficients for $H_4(z)$

GAM	=	0.5622541
c1	=	-0.2518445
c2	=	-0.6455710
a11	=	0.2621350
a12	=	-0.5582184
a21	=	0.6536193
a22	=	0.2621350
scale	=	1

The other second-order section is

$$H_5(z) = \frac{0.16149z - 0.125769}{z^2 - 1.195305z + 0.8453360}.$$

This section yields the following sequence for the state variables for one period

$$x_1^i \quad \{60, 59, 58, 57, 57, 58, 59, 60\}$$

$$x_2^i \quad \{31, 30, 30, 31, 32, 33, 33, 32\}$$

when the initial states were

$$x_1^f = 0.9999694$$

$$x_2^f = 0.9998168.$$

Using the equations developed in Chapter III, this transfer function can be implemented free of constant-input limit cycles using the coefficients in Table 5.

Table 5. Coefficients for $H_5(z)$

GAM	=	0.1543282
c1	=	-0.3560773
c2	=	1.801147
a11	=	0.5976525
a12	=	-0.5320842
a21	=	0.9174253
a22	=	0.5976525
scale	=	1

The first order section is

$$H_6(z) = \frac{-0.565006}{z + 0.129964}$$

when the initial state for this transfer function was

$$x_1^f = 0.4.$$

Using the equations developed in Chapter III, this transfer

function can be implemented free of constant-input limit cycles using the coefficients in Table 6.

Table 6. Copefficients for $H_6(z)$

a	=	-0.1299639940261841
b	=	-0.8700360059738159
c	=	0.6494053278318468
lam	=	0.7699679021397694

CHAPTER V

DSP PROGRAMS FOR DIGITAL FILTERS FREE OF LIMIT CYCLES

Motorola DSP56000/1 programs were developed to implement the four transfer functions discussed in Chapter IV. To verify proper operation, the programs were run on an IBM compatible computer using Motorola DSP56000/1 Digital Signal Processor simulator software. The programs were written in DSP56000/1 assembly language.

Motorola DSP56000/1 Digital Signal Processor

Motorola DSP56000/1 Digital Signal Processor has two types of data ALU registers, input registers and accumulator registers. There are four 24-bit input registers X1, X0, Y1, and Y0. The six data accumulator registers: A2, A1, A0, B2, B1, and B0 form two general purpose 56-bit accumulators, A and B. The A-Accumulator is the concatenation of registers A2:A1:A0 and the B-Accumulator is the concatenation of registers B2:B1:B0. Registers A1, A0, B1, and B0 are each 24-bits wide, whereas registers A2 and B2 are each 8-bits wide.

Also, the Motorola DSP56000/1 Digital Signal Processor has three types of address generation unit registers: address registers, offset registers, and modifier registers.

Registers R0-R7 are address registers, N0-N7 are offset registers, and M0-M7 are modifier registers. Registers R0-R7, N0-N7 and M0-M7 are each 16-bits wide. Registers R0-R7 are used as pointers for retrieving data from and storing data to memory locations. There are two separate memory banks, X and Y. Data is arranged in the X and Y banks from 0 - FFFF (hexadecimal).

Magnitude truncation

In each of the Motorola DSP56000/1 programs in the appendices magnitude truncation was used rather than rounding in the filter's state-variable update calculations. To perform magnitude truncation on the Motorola DSP56000/1 Digital Signal Processor, the number to be truncated is stored in the 56-bit A-Accumulator. The first statement in the magnitude truncation portion of the programs is a NEG A A,X0 statement. This statement takes the two's complement of the value in the A accumulator and stores the 56-bit result back into the A-Accumulator. This statement simultaneously stores the upper 24-bits of the A-Accumulator (A1) into the 24-bit X0 register truncating the value originally in the A-Accumulator. Next, the processor moves the value of zero into the lower 24-bits of the A-Accumulator (A2) thus truncating the two's complement of the original value. Once again the two's complement is taken on the value in the A-Accumulator. Then, the statement TGT X0,A tests whether the value in the A-

Accumulator is positive. If the value in the A-Accumulator is greater than zero the value in the X0 register is transferred to the A-Accumulator. Otherwise, the value in the A-Accumulator is the correct magnitude truncated value and the X0 register contents are not transferred to A. This procedure performs magnitude truncation as follows.

When the number for which magnitude truncation is to be performed is negative, this approach first takes the two's complement of this negative value. The resulting positive number is magnitude truncated by placing zeros in the lower order bits of the A-Accumulator and the two's complement is taken on the result. Hence the number is once again a negative number in the A-Accumulator and is the correct magnitude truncated value. If the number on which magnitude truncation is to be performed is positive, the correct value is stored in the X0 register. This is due to the fact that truncation is performed when the 56-bit number stored in the A-Accumulator is moved to the 24-bit X0 register. In this case the value of X0 would be moved into the A-Accumulator by the TGT X0,A instruction.

First-order

The assembly program FIRST (see appendix G) was used to implement first-order sections on the Motorola DSP56000/1 digital signal processor. The program FIRST uses the output of COEFFFIR.FOR as its state-space realization coefficients.

These coefficients are stored in X-memory locations under the label GU. FIRST uses the R2 address register and the M2 modifier register to transport the coefficients within the do-loop portion of the program. The value of the initial state variable is stored in Y-memory under label TR. The R6 address register and the M6 modifier register are used to transport the initial state variable prior to starting the do-loop portion of the program. The program FIRST has a do-loop within which several operations are performed. Within this loop, the state variable is carried in X1. The first operation is reading the input from an input file (or an input port). Then, there are several statements within the do-loop which implement equation (3.7). The first MPYR instruction scales the input and rounds. Next, the other MPYR instruction calculates and rounds the output. Then the MPY and MAC instructions calculate the unquantized updated state variable. Next, magnitude truncation is performed on this unquantized updated state variable. Finally, the scaled input is subtracted to obtain the updated state variable which is moved to X1. Following the state variable update, the output is stored into an output file (or it goes to an output port).

Second-order

The assembly program DCGAIN (see appendix H) was used to realize a second order section on Motorola's DSP56000/1 digital signal processor. This program implements equation

(3.25). It uses the data from Table 2 for the state space realization coefficients. The program has also been used with data from other tables like Tables 1 and 3. The coefficients are stored in X-memory locations under the label GK. The program uses the R0 address register and the M0 modifier register to transport the coefficients within the do-loop portion of the program. The values of the initial state variables are stored in Y-memory under label TK. The R4 address register and the M4 modifier register are used to transport the state variables within the do-loop portion of the program and also to return the updated state variables to the Y-memory location labeled TK.

As in FIRST, each these programs have a do-loop which performs several operations. The first operation within the do-loop is the reading of the input from an input file. The MPYR statement forms $u(n)$ according to (3.25). Next, the MPY and MACR statements calculate and round the output as shown in equation (3.25). Then, the next three statements (MPY, MAC, MAC) calculate the value of the first unquantized updated state variable. Following those three statements, there are four statements (NEG, MOVE, NEG, and TGT) which perform magnitude truncation on the first unquantized updated state variable. Then the SUB instruction subtracts the scaled input from the quantized value to obtain the updated state variable. This value of the updated state variable is stored in the Y-memory location under the label TK. The next three statements

(MPY, MAC, and MAC) perform the calculation for the other unquantized updated state variable. In the next four statements (NEG, MOVE, NEG, and TGT), magnitude truncation is performed on the unquantized updated state variable to obtain a quantized updated state variable. This updated state variable is also stored in the Y-memory location under the label TR. Now the output previously calculated is multiplied by a scaling factor to obtain the true output. Finally, this output is stored into an output file.

Nth-order filter (5th example)

A Nth order realization is either a combination of second order sections for even order filters or a combination of second order sections and one first order section for odd order filters. The assembly program FIFTH (see appendix I) implements the fourth transfer function discussed in Chapter IV. Since this transfer function was fifth-order, the realization has two second order sections and one first order section. FIFTH uses the data from Tables 4, 5, and 6 for the state-space realization coefficients of the first second-order section, the other second-order section and the first order section, respectively.

The coefficients from Tables 4, 5, and 6 are stored in X-memory locations under the label GK, GG, and GU, respectively. The program FIFTH uses R0, R1, and R2 address registers and M0, M1, and M2 modifier registers to transport the

coefficients within the do-loop portion of the program. The values of the initial state variables are stored in Y-memory under the labels TK, TT, and TR. The R4, R5, R6 address registers and the M4, M5, and M6 modifier registers are used to transport the state variables within the do-loop portion of the program and also to return the updated state variables to the Y-memory location labeled TK, TT, and TR respectively. The Y-memory location labeled TY is used to stored the value of the input once it has been read in. The R7 address register and M7 modifier register are used to transport the value of the input into and out of the Y-memory location labeled TY.

Within the program FIFTH there is a do-loop where the values of the two second order sections and the first order section are calculated. The two second order sections are exactly same as the second order programs described above with one exception. The only difference is that the output is stored in a Y-memory location to be retrieved later rather than in an output file. Similarly, the only difference in the first order section in the program FIFTH and the first order section discussed above is that the output is not stored in an output file. For the first order section the output is not stored. The output of the first order section is added to the sum of the outputs produced by the two second order sections. This value is the output of the fifth order transfer function and it is the value stored into an output

file.

Experimental results

By running the computer program DCGAIN with the coefficients from table 1-3 and FIFTH, the experimental results were obtained. These programs verified that if the methods in [4], [12], [13] are employed in an implementation, constant-input limit cycles can be eliminated. The outputs of each of these programs yielded the expected outputs which were previously calculated. The expected output is the dc gain times the input. For the inputs 0.1, 0.5, and 0.85, each program was run with the initial state variables as indicated on each program. For each value of the input, these programs were successful in eliminating constant-input limit cycles from the four transfer functions discussed in Chapter IV. For the second order program DCGAIN an impulse response was applied as the input to the realization. The output of the program was compared to the output of a FORTRAN program which implements a realization of the same second order filter. Agreement was obtained between the outputs of the FORTRAN program and DCGAIN for the coefficients of tables 1-3.

CHAPTER VI

COMPUTATIONAL LOADING

Computation Cycles Required

In the design of digital filters to be implemented on a digital signal processor the number of computational cycles can be very important. In this chapter, the actual number of computational cycles required for the first, second and fifth order transfer functions is determined. Also, to provide information on the computational cost of eliminating limit cycles, the second order implementations are compared to a second order implementation that does not eliminate limit cycles.

The number of computational cycles is based on the number of processor instruction cycles required per sample processed. The Motorola data book gives a listing of the number of oscillator clock cycles each instruction requires. This is always an even number which can be divided by two to obtain the number of instruction cycles. An illustration will be given to show how the required computational cycles are determined in the first-order section.

First-order

For a first order section implemented on a digital signal processor using equation (3.7), twelve instruction cycles are

required. From the Motorola data book each instruction (MOVE, MPYR, MPY, MAC, NEG, TGT, SUB) requires two oscillator clock cycles. There are four MOVE, two MPYR, two NEG, one MPY, one MAC, one TGT, and one SUB. Thus, the first order section requires twenty four oscillator clock cycles. To determine the number of instruction cycles, the number of oscillator clock cycles is divided by two. Hence, the result is twelve instruction cycles.

Second-order

For a second order section to be implemented on a digital signal processor using equation (3.25), twenty one instruction cycles were required. There are four MOVE, one MYPR, four MPY, one MACR, four MAC, four NEG, two TGT, one SUB. Hence it required twenty one instruction cycles to implement a second order section.

Nth-order

As it was stated in Chapter V an Nth order realization is a combination of second order sections for even order filters and a combination of second order sections plus one first order section for odd order filters. For the fifth order filter in chapter IV fifty nine instruction cycles were required.

Computation of implementation with limit cycles

In this study another second order realization was implemented on a Motorola digital signal processor. This realization was taken from a paper by Bomar [15]. In that paper the realization was referred to as a type III structure. The equations for the type III structure are

$$a_{11}=a_{22}=-\frac{\beta_1}{2} \quad (6.1)$$

$$a_{12}=\sqrt{1+\left(\frac{\beta_1}{2}\right)^2\left(\frac{\beta_2-3}{\beta_2+1}\right)} \quad (6.2)$$

$$a_{21}=\frac{\left(\frac{\beta_1}{2}\right)^2-\beta_2}{a_{12}} \quad (6.3)$$

$$b_1=0 \quad (6.4)$$

$$b_2=\sqrt{\frac{(1-\beta_2)[(1+\beta_2)^2-\beta_1^2]}{(1+\beta_2)\left[1+\left(\frac{\beta_1}{2}\right)^2\right]-\beta_1^2}} \quad (6.5)$$

$$c_1=\frac{\alpha_2+a_{11}\alpha_1}{a_{12}b_2} \quad (6.6)$$

$$c_2=\frac{\alpha_1}{b_2} \quad (6.7)$$

Notice that $a_{11} = a_{22}$, which as discussed in Chapter III guarantees zero-input limit cycles will not exist for any

stable second order section if magnitude truncation is used. This realization was implemented using the program, RFIL (see appendix J). In this program the state variables were rounded instead of using magnitude truncation since rounding takes less processing time to implement. The number of cycles required to implement the type III realization using RFIL with rounding was twelve instruction cycles. Thus, it requires nine additional instruction cycles to implement a realization free of constant input limit cycles.

CHAPTER VII

SUMMARY AND CONCLUSIONS

Although it is possible to prove that the realizations of [4], [12], [13] will not have constant-input limit cycles, it had not been shown that such limit cycles exist if only the Mills method [11] is employed without the complications of the methods of [4], [12], [13]. Computer programs were written to search for filter transfer functions that have constant-input limit cycles when only the Mills method is employed. Using these computer programs, many transfer functions were found that do have constant-input limit cycles despite using the Mills method. Four of these transfer functions were selected for subsequent work. The search programs were also used to confirm that zero-input limit cycles do not exist using the Mills method, as predicted by the theory.

Since the methods of [4], [12], [13] are necessary, implementations of the methods on the Motorola DSP56000/1 family of digital signal processors were considered. By running the implementations on a Motorola DSP56000/1 simulator, it was shown that the four transfer functions are free of limit cycles under constant-input conditions.

To provide designers with information on the computational cost of eliminating limit cycles, the DSP56000/1 limit-cycle-free implementations were compared to DSP56000/1

implementations that may support limit cycles. The comparison was made based on the number of processor instruction cycles required per sample processed. For the second-order case, this comparison indicated that the implementation that may produce limit cycles requires nine less instruction cycles than the implementation which eliminates limit cycles, out of a total of twenty one instruction cycles. This is considered an acceptable increase in computation to eliminate limit cycles.

REFERENCES

REFERENCES

- [1] A. V. Openheim and R. W. Schafer, *Discrete-Time Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1989.
- [2] R. A. Roberts and C. T. Mullis, *Digital Signal Processing*. Massachusetts: Addison-Wesley Publishing Company, 1987.
- [3] P. P. Vaidyanathan and V. Liu, "An Improved Sufficient Condition for Absence of Limit Cycles in Digital Filters," *IEEE Trans. on Circuits and Systems*, vol. CAS-34, No. 3, pp. 319-322, March 1987.
- [4] P. Dinz and A. Antoniou, "More Economical State-Space Digital-Filter Structures Which are Free of Constant-Input Limit Cycles," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. ASSP-34, No. 4, pp. 807-815, August 1986.
- [5] V. Singh, "Elimination of Overflow Oscillations in Fixed-Point State-Space Digital Filters Using Saturation Arithmetic," *IEEE Trans. on Circuits and Systems*, vol. 37, No. 6, pp. 814-818, June 1990.
- [6] T. Bose and D. P. Brown, "Limit Cycles Due to Roundoff In State-Space Digital Filters," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. 38 No. 8, pp. 1460-1462, August 1990.
- [7] H. J. Butterweck, A. C. P. Van Meer, and G. Verkroost, "New Second-Order Digital Filter Sections Without Limit Cycles," *IEEE Trans. on Circuits and Systems*, vol. CAS-31, pp. 141-146 February 1984.
- [8] G. Verkroost, "A General Second-Order Digital Filter with Controlled Rounding to Exclude Limit Cycles for Constant Input Signals," *IEEE Trans. on Circuits and Systems*, vol. CAS-24, No. 8, pp. 428-431, August 1977.
- [9] S. R. Parker and S. F. Hess, "Limit-Cycle Oscillations in Digital Filters," *IEEE Trans. on Circuits and Systems*, vol. CT-18, No. 6, pp. 687-697, November 1971.
- [10] E. S. K. Liu and L. E. Turner, "Stability, Dynamic Range, and Roundoff Noise in a New Second-Order Recursive Digital Filter," *IEEE Trans. on Circuits and Systems*, vol. CAS-30, pp. 815-821, Nov. 1983.
- [11] W. L. Mills, C. T. Mullis, and R. A. Roberts, "Digital filter realizations without overflow oscillations," *IEEE*

Trans. Acoust., Speech, Signal Processing, vol. ASSP-26, pp. 334-338, Aug. 1978.

- [12] M. Kawamata and T. Higuchi, "A Systematic Approach to Synthesis of Limit Cycles-Free Digital Filters," *IEEE Trans. on Acoustics, Speech and Signal Processing*, vol. ASSP 31 No. 1 February 1983.
- [13] B. W. Bomar, "Low-roundoff-noise Limit-cycle-free implementation of recursive transfer functions on a fixed-point digital signal processor," *IEEE Transactions on Industrial Electronics*, to appear.
- [14] B. W. Bomar, ".On the Design of Second-Order State-Space Digital Filter Sections," *IEEE Trans. on Circuits and Systems*, vol. 36 No. 4, pp. 542-552, April 1989.
- [15] B. W. Bomar, "New Second-Order State-Space Structures for Realizing Low Roundoff Noise Digital Filters," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, Vol. ASSP-33 No. 1, pp. 106-110, February 1985.
- [16] B. W. Bomar and R. D. Joseph, "Calculation of L_∞ Norms for Scaling Second-Order State-Space Digital Filter Sections," *IEEE Trans. on Circuits and Systems*, vol. CAS-34, No. 8, pp. 983-984, August 1987.

APPENDIXES

APPENDIX A

```

C*****
C   Program name:  LIMITCY.FOR
C
C   This program is used to search for a filters with
C   constant-input limit cycles
C*****

```

```

C       R=RADIUS
C       T= THETA
C
C       IC(1)=U=input
C       IC(2)=X1 state variable (returns updated state
C       variable)
C       IC(3)=X2 state variable (returns updated state
C       variable)
C       IC(4)=C1
C       IC(5)=C2
C       IC(6)=A11
C       IC(7)=A12
C       IC(8)=B1
C       IC(9)=A21
C       IC(10)=A22
C       IC(11)=B2
C       IC(12)=N

```

```

PARAMETER (DR=3.141592653589793/180.0)
DIMENSION A(2,2),B(2),C(2)
INTEGER*2 IC(12)
F=32768.0
N=1000
ISEED=-1
DO IR=90,99

```

```

C
C   RADIUS
C
C       R=0.9+0.001*FLOAT(IR)
C       DO IT=1,179
C           WRITE(*,*) R,T

```

```

C
C   THETA
C
C       T=DR*FLOAT(IT)
C       C1=COS(T)
C       S1=SIN(T)

```

```

C

```

```

C   REAL VALUED REALIZATION COEFFICIENTS
C
      A(1,1)=R*C1
      A(1,2)=1.0
      A(2,1)=-R*R*S1*S1
      A(2,2)=A(1,1)
      B(1)=0.0
      B(2)=1.0
      C(1)=1.0
      C(2)=0.0
C
C   SUBROUTINE SCALELI   CALCULATES THE L NORM USING BOMAR
C   AND JOSEPH TECHNIQUE
C
      CALL SCALELI(A,B,C)
C
C   ROUNDS REALIZATION COEFFICIENTS TO THE NEAREST INTEGER
C
      IC(6)=NINT(A(1,1)*F)
      IC(7)=NINT(A(1,2)*F)
      IC(8)=NINT(B(1)*F)
      IC(9)=NINT(A(2,1)*F)
      IC(10)=NINT(A(2,2)*F)
      IC(11)=NINT(B(2)*F)
C
C   INPUT, ROUNDS INPUT TO NEAREST INTEGER
C
      U=0.01
      IC(1)=NINT(U*F)
C
C   DO LOOP TO SELECT RANDOM STATE VARIABLES AND ROUNDS TO
C   NEAREST INTEGER
C
      DO IS=1,100
        X1=2.0*RND(ISEED)-1.0
        X2=2.0*RND(ISEED)-1.0
        IC(2)=NINT(X1*F)
        IC(3)=NINT(X2*F)
        IC(12)=N
C
C   SUBROUTINE IFILNM MOVE STATE VECTOR N SAMPLES AHEAD
C   WITHOUT COMPUTING THE OUTPUT
C
      CALL IFILNM(IC)
C
C   STORES ONE STATE VARIABLE
C
      IX1=IC(2)
      IC(12)=N
C
C   SUBROUTINE IFILNM
C

```

```

      CALL IFILNM(IC)
C
C   TEST FOR LIMIT CYCLES
      IF (IC(2) .NE. IX1) THEN
C
C   CALCULATES THE VALUES FOR BETA  1 AND BETA 2
C
      B1=-2*R*COS(T)
      B2=R*R
C
C   PRINTS BETA 1, BETA 2, AND STATE VARIABLES
C
      WRITE(*,*) B1, B2,X1,X2
      PAUSE
      IC(12)=1
      DO I=1,200
C
C   SUBROUTINE IFILNM
C
      CALL IFILNM(IC)
C
C   PRINTS STATE VARIABLES
C
      WRITE(*,*) IC(2),IC(3)
      ENDDO
      GO TO 1
      ENDIF
      ENDDO
1    ENDDO
      ENDDO
      ENDDO
      STOP
      END

```

APPENDIX B

```

C*****
C   Program name:  MYT.FOR
C
C   This program is used to find if a
C   given a transfer function has limit cycles.
C*****

C       R=RADIUS
C       T= THETA
C
C       IC(1)=U=input
C       IC(2)=X1 state variable (returns updated state
C       variable)
C       IC(3)=X2 state variable (returns updated state
C       variable)
C       IC(4)=C1
C       IC(5)=C2
C       IC(6)=A11
C       IC(7)=A12
C       IC(8)=B1
C       IC(9)=A21
C       IC(10)=A22
C       IC(11)=B2
C       IC(12)=N

      DIMENSION A(2,2),B(2),C(2)
      INTEGER*2 IC(12)
      F=32768.0
      N=1000
      ISEED=-1

      WRITE(6,*)'Enter beta 1 for a second order section'
      READ(5,*)B1
      WRITE(6,*)'B1=',B1

      WRITE(6,*)'Enter beta 2 for a second order section'
      READ(5,*)B2
      WRITE(6,*)'B2=',B2

C
C   RADIUS
C
C       R=sqrt(B2)
C
C   THETA
C
C       T = ACOS(-B1 / (2*R))

```

```

WRITE(*,*) R,T

C1=COS(T)
S1=SIN(T)
C
C REAL VALUED REALIZATION COEFFICIENTS
C
      A(1,1)=R*C1
      A(1,2)=1.0
      A(2,1)=-R*R*S1*S1
      A(2,2)=A(1,1)
      B(1)=0.0
      B(2)=1.0
      C(1)=1.0
      C(2)=0.0
C
C SUBROUTINE SCALELI CALCULATES THE L NORM USING BOMAR
C AND JOSEPH TECHNIQUE
C
      CALL SCALELI(A,B,C)
C
C ROUNDS REALIZATION COEFFICIENTS TO THE NEAREST INTEGER
C
      IC(6)=NINT(A(1,1)*F)
      IC(7)=NINT(A(1,2)*F)
      IC(8)=NINT(B(1)*F)
      IC(9)=NINT(A(2,1)*F)
      IC(10)=NINT(A(2,2)*F)
      IC(11)=NINT(B(2)*F)
C
C INPUT, ROUNDS INPUT TO NEAREST INTEGER
C
      U=0.01
      IC(1)=NINT(U*F)
C
C DO LOOP TO SELECT RANDOM STATE VARIABLES AND ROUNDS TO
C NEAREST INTEGER
C
      DO IS=1,100
      X1=2.0*RND(ISEED)-1.0
      X2=2.0*RND(ISEED)-1.0
      IC(2)=NINT(X1*F)
      IC(3)=NINT(X2*F)
      IC(12)=N
C
C SUBROUTINE IFILNM MOVE STATE VECTOR N SAMPLES AHEAD
C WITHOUT COMPUTING THE OUTPUT
C
      CALL IFILNM(IC)
C
C STORES ONE STATE VARIABLE

```

```

C          IX1=IC(2)
          IC(12)=N
C
C  SUBROUTINE IFILNM
C
C          CALL IFILNM(IC)
C
C  TEST FOR LIMIT CYCLES
C          IF (IC(2) .NE. IX1) THEN
C
C          PRINTS BETA 1, BETA 2, AND STATE VARIABLES
C
C          WRITE(*,*) B1, B2,X1,X2
C          PAUSE
C          IC(12)=1
C          DO I=1,200
C
C          SUBROUTINE IFILNM
C
C          CALL IFILNM(IC)
C
C          PRINTS STATE VARIABLES
C
C          WRITE(*,*) IC(2),IC(3)
C          ENDDO
C          GO TO 1
C          ENDIF
1      ENDDO
      STOP
      END

```

APPENDIX C

```

C*****
C   Program name:  SCALELI.FOR
C
C   Subroutine used by LIMITCY and MYT
C*****

```

```

SUBROUTINE SCALELI(A,B,C)
DIMENSION A(2,2),B(2),C(2)
B1=-(A(1,1)+A(2,2))
B2=A(1,1)*A(2,2)-A(1,2)*A(2,1)
C3=-0.25*B1*(1.0+B2)/B2           ! Xi
C4=0.25*(1.0-0.25*B1*B2/B2)*
&      ((1.0-B2)**2)/B2           ! Mu (eq. 10e)
C5=A(1,2)*B(2)-A(2,2)*B(1)       ! Alpha2 for F1(z)
C6=A(2,1)*B(1)-A(1,1)*B(2)       ! Alpha2 for F2(z)
D1=2.0*B(1)*C5/(B(1)*B(1)+C5*C5) ! 1/Delta for F1(z)
D2=2.0*B(2)*C6/(B(2)*B(2)+C6*C6) ! 1/Delta for F2(z)
IF (D1 .NE. 0.0) E1=(-1.0+SQRT((1.0+D1*C3)**2+
&      D1*D1*C4))/D1
IF (D1 .EQ. 0.0) E1=C3
IF (D2 .NE. 0.0) E2=(-1.0+SQRT((1.0+D2*C3)**2+
&      D2*D2*C4))/D2
IF (D2 .EQ. 0.0) E2=C3
IF (E1 .GT. 1.0) E1=1.0           ! E1=epsilon1
IF (E1 .LT. -1.0) E1=-1.0
IF (E2 .GT. 1.0) E2=1.0           ! E2=epsilon2
IF (E2 .LT. -1.0) E2=-1.0
D1=4.0*B2*((E1-C3)**2+C4)
D2=4.0*B2*((E2-C3)**2+C4)
S1=SQRT((B(1)*B(1)+C5*C5+2.0*B(1)*C5*E1)/D1)
S2=SQRT((B(2)*B(2)+C6*C6+2.0*B(2)*C6*E2)/D2)
A(1,2)=(S2/S1)*A(1,2)
A(2,1)=(S1/S2)*A(2,1)
B(1)=B(1)/S1
B(2)=B(2)/S2
C(1)=S1*C(1)
C(2)=S2*C(2)
RETURN
END

```

APPENDIX D

```

C*****
C   Program name: IFILNM.FOR
C
C   Subroutine used in LIMITCY program
C*****

C
C   MOVE STATE VECTOR N SAMPLES AHEAD WITHOUT COMPUTING
C   OUTPUTS
C
C   SUBROUTINE IFILNM(IC)
C   INTEGER*2 IC(12),IW(2),INX1
C   INTEGER*4 ISUM
C   EQUIVALENCE (IW(1),ISUM)
C
C   IC(1)=U=input
C   IC(2)=X1 state variable (returns updated state
C   variable)
C   IC(3)=X2 state variable (returns updated state
C   variable)
C   IC(4)=C1
C   IC(5)=C2
C   IC(6)=A11
C   IC(7)=A12
C   IC(8)=B1
C   IC(9)=A21
C   IC(10)=A22
C   IC(11)=B2
C   IC(12)=N
C
C   DO I=1,IC(12)
C
C
C       ISUM=2*(IC(6)*IC(2)+IC(7)*IC(3)+IC(8)*IC(1))
C       INX1=IW(2)
C       IF ((ISUM .LT. 0) .AND. (IW(1) .NE. 0))
C INX1=INX1+1
C       ISUM=2*(IC(9)*IC(2)+IC(10)*IC(3)+IC(11)*IC(1))
C       IC(3)=IW(2)
C       IF ((ISUM .LT. 0) .AND. (IW(1) .NE. 0))
C IC(3)=IC(3)+1
C       IC(2)=INX1
C   ENDDO
C   RETURN
C   END

```

APPENDIX E

```

C*****
C   Program name:  COEFFIR.FOR
C
C   THIS PROGRAM CALCULATES THE STATE SPACE REALIZATION FOR
C   FIRST ORDER SECTIONS
C*****
      DOUBLE PRECISION A,B,C,LAM

      OPEN(UNIT=6,FILE='FIRST.OUT')

      WRITE(6,*)'Enter beta for a first order section'
      READ(5,*)BETA
      WRITE(6,*)'B1=',BETA

      WRITE(6,*)'Enter alpha for a first order section'
      READ(5,*)ALPHA
      WRITE(6,*)'A1=',ALPHA
C
C   EQUATION (3.17)
C
      a= - beta
      b= abs(beta) -1
      c = ALPHA /(abs(beta) -1)
      lam= (1-abs(beta)) / ( 1 + beta )
C
C   WRITES THE OUTPUT TO FIRST.OUT
C
      write(6,*)'a=',a
      write(6,*)'b=',b
      write(6,*)'c=',c
      write(6,*)'lam=',lam

c   close(unit=6)
      stop
      end

```

APPENDIX F

```

C*****
C   Program name:  COEFSEC.FOR
C
C   THIS PROGRAM CALCULATES THE STATE-SPACE COEFFICIENTS
C   FOR A SECOND-ORDER DIGITAL FILTER
C*****

      DIMENSION A(2,2),B(2),C(2)
      OPEN(UNIT=6,FILE='LAST.OUT')
      OPEN(UNIT=7,FILE='TXT1.TXT')
      READ(7,*) B1, B2,A1,A2
      WRITE(6,*) 'B1',B1
      WRITE(6,*) 'B2',B2
      WRITE(6,*) 'A1',A1,'A2',A2

C
C   EQUATIONS (4.1) - (4.3)
C
      A(1,1)=-B1/2
      A(2,2)=A(1,1)
      A(1,2)=1.0
      A(2,1)=(B1/2)**2-B2
      B(1)=0.0
      B(2)=1.0
      C(1)=1.0
      C(2)=0.0
      SIG=SQRT(B2-(B1/2)**2)
      C3=-0.25*B1*(1.0+B2)/B2           ! Xi (eq. 10d)
      C4=0.25*(1.0-0.25*B1*B1/B2)*
&          ((1.0-B2)**2)/B2           ! Mu (eq. 10e)
      C5=A(1,2)*B(2)-A(2,2)*B(1)       ! Alpha2 for F1(z)
      C6=A(2,1)*B(1)-A(1,1)*B(2)       ! Alpha2 for F2(z)
      D1=2.0*B(1)*C5/(B(1)*B(1)+C5*C5)  ! 1/Delta for F1(z)
      D2=2.0*B(2)*C6/(B(2)*B(2)+C6*C6)  ! 1/Delta for F2(z)
      IF (D1 .NE. 0.0) E1=(-1.0+SQRT((1.0+D1*C3)**2+D1*D1*
&          C4))/D1
      IF (D1 .EQ. 0.0) E1=C3
      IF (D2 .NE. 0.0) E2=(-1.0+SQRT((1.0+D2*C3)**2+D2*D2*
&          C4))/D2
      IF (D2 .EQ. 0.0) E2=C3
      IF (E1 .GT. 1.0) E1=1.0           ! E1=epsilon1 (eq.18)
      IF (E1 .LT. -1.0) E1=-1.0
      IF (E2 .GT. 1.0) E2=1.0           ! E2=epsilon2 (eq.18)
      IF (E2 .LT. -1.0) E2=-1.0
      D1=4.0*B2*((E1-C3)**2+C4)
      D2=4.0*B2*((E2-C3)**2+C4)
      S1=SQRT((B(1)*B(1)+C5*C5+2.0*B(1)*C5*E1)/D1)
      S2=SQRT((B(2)*B(2)+C6*C6+2.0*B(2)*C6*E2)/D2)
      DEL=S1/S2

C
C   EQUATION (3.84)

```

```

C      A(2,1)=SIG*DEL
C
C      EQUATION (3.87)
C
C      A(1,2)=-SIG/DEL
C      GAM=1/S2
C      C(1)=-((A1+A2)/(1+B1+B2))
C      C(2)=(1/A(2,1))*(A1-((1+0.5*B1)*A1+A2)/(1+B1+B2))
C
C      EQUATION (3.88)
C
C      C(1)=C(1)*S2
C      C(2)=C(2)*S2
C
C      SCALES C(1) AND C(2) IF NECESSARY
C
C      IF(C(1).LT.1.0.AND.C(2).LT.1.0) GOTO 9
C      PP=ABS(C(1))
C      PPP=ABS(C(2))
C      IF (PP .GT. PPP) GOTO 3
C      QPPP=MOD(PPP,2)
C      IF (QPPP .EQ. 0) PP=(PPP+2)
C      PP=PPP+1
C      GOTO 6
3      QPP=MOD(PP,2)
C      IF (QPP .EQ. 0) PP=(PP+2)
C      PP=PP+1
C
6      C(1)=C(1)/PP
C      C(2)=C(2)/PP
C
C      CALCULATES THE SCALING FACTOR
C
C      SCALE=PP/2
C      GOTO 11
C
9      SCALE=1.0
11     WRITE(6,*)'GAM=',GAM
C      WRITE(6,*)'C1=',C(1)
C      WRITE(6,*)'C2=',C(2)
C      WRITE(6,*)'A11=',A(1,1)
C      WRITE(6,*)'A12=',A(1,2)
C      WRITE(6,*)'A21=',A(2,1)
C      WRITE(6,*)'A22=',A(2,2)
C      WRITE(6,*)'SCALE=',SCALE
C      CLOSE(UNIT=6)
C      CLOSE(UNIT=7)
C
C      STOP
C      END

```

APPENDIX G

```

;*****
;   Program name:  FIRST.ASM
;
;   This program calculates the output of a first-order
;   digital filter using state space coefficients
;   calculated by COEFFIR.FOR.
;*****

```

$$y(n) = q_r[cx(n)]$$

$$u(n) = q_r[\lambda u(n)]$$

$$x(n+1) = q_m\{ a[x(n) + \tilde{u}(n)] \} - \tilde{u}(n)$$

```

;   y(n) is the output
;   u(n) is the input
;   x(n+1) is the next state variable
;   q_r{} represents quatization by rounding
;   q_m{} represents quatization by magnitude truncation

```

$$H(z) = \frac{-0.565006}{z+0.129964}$$

```

      OPT      CEX
      ORG      X:$0

```

```

;
;   STATE SPACE COEFFICIENTS TAKEN FROM THE OUTPUT OF
;   FIRST.FOR
;

```

GU

```

      DC      0.7699679021397694
      DC      0.6494053278318468
      DC      -0.1299639940261841

```

```

      ORG      Y:$0

```

```

;
;   STATE VARIABLE
;
TR

```

```

                DC      0.4                      ;x1

                ORG      P:$0

                JMP      <START

;
;    MODULAR REGISTERS USED TO TRANSPORT THE COEFFICIENTS
;

START          MOVE     #GU,R2
                MOVE     #2,M2
                MOVE     #TR,R6
                MOVE     #0,M6

                NOP
                CLR A    X:(R2)+,X0                ;GAMMA
                MOVE     Y:(R6)+,X1                ;X1

;
;    DO LOOP USED TO CALCULATE THE NEXT STATE VARIABLE AND
;    THE OUTPUT
;
                DO      #0,LOOP

                MOVE     Y:$2,Y0
                MPYR     X0,Y0,A    X:(R2)+,X0      ;LAM*U
                MPYR     X0,X1,B    X:(R2)+,X0      A,Y0    ;C*X,A
                MPY      X0,X1,A      ;A*X,B
                MAC      X0,Y0,A    b,y1            ;A*LAM*U

;
;    PERFORMS MAGNITUDE TRUNCATION
;

                NEG      A          A,X0
                MOVE     #0,A0
                NEG      A
                TGT      X0,A
                SUB      Y0,A    X:(R2)+,X0          ;-LAM*U
                MOVE     A,X1

;
;    STORES THE OUTPUT INTO AN OUTPUT FILE
;
                MOVE     B1,Y:$3
LOOP
                END

```

APPENDIX H

```

;*****
; Program name: DCGAIN.ASM
;
;    $x_1(N+1) = q\{a_{11}x_1(N) + a_{12}x_2(N) + a_{11}U(N)\} - U(N)$ 
;    $x_2(N+1) = q\{a_{21}x_1(N) + a_{22}x_2(N) + a_{21}U(N)\}$ 
;    $Y(N) = qr\{c_1x_1(N) + c_2x_2(N)\}$ 
;    $U(N) = qr\{GAMMA*U(N)\}$ 
;   q---MAGNITUDE TRUNCATION
;   qr---ROUNDING
;*****

```

	OPT	CEX	
	ORG	X:\$0	
GK			
	DC	0.04935468	;GAMMA
	DC	-0.92097772	;c ₁
	DC	-0.94500136	;c ₂
	DC	0.7468935	;a ₁₁
	DC	-0.4030196	;a ₁₂
	DC	0.9745782	;a ₂₁
	DC	0.7468935	;a ₂₂
	DC	\$00000B	;scale

	ORG	Y:\$0	
TK			
	DC	0.9999694	;x ₁
	DC	0.9998168	;x ₂
	ORG	P:\$0	
	JMP	<START	
START	MOVE	#GK,R0	
	MOVE	#7,M0	
	MOVE	#TK,R4	
	MOVE	#1,M4	
	NOP		
	CLR A	X: (R0)+,X0	;GAMMA
	MOVE	Y: (R4)+,X1	;x ₁
	DO	#0,LOOP	
	INPUT		
	MOVE	Y:\$2,Y0	
	MULTIPLIES INPUT BY A SCALING FACTOR		
	MPYR	X0,Y0,A X: (R0)+,X0 Y: (R4)-,Y1	;GAM*U
	CALCULATES AN UNSCALED OUTPUT		

```

;
    MPY      X0,X1,B      X:(R0)+,X0      ;C1*x1
    MACR     X0,Y1,B      X:(R0)+,X0      A,Y0      ;C2*x2
;
;
CALCULATES AN UNQUANTIZED UPDATED STATE VARIABLE
;
    MPY      X0,X1,A      ;a11*x1
    MAC      X0,Y0,A      X:(R0)+,X0      ;a11*
    MAC      X0,Y1,A      ;a12*x2
;
;
PERFORMS MAGNITUDE TRUNCATION ON UPDATED STATE VARIABLE
;
    NEG      A            A,X0
    MOVE     #0,A0
    NEG      A
    TGT      X0,A
;
;
SUBTRACTS SCALED INPUT FROM UPDATED STATE VARIABLE
;
    SUB      Y0,A          X:(R0)+,X0      ;-GAM*U
;
;
CALCULATES AN UNQUANTIZED UPDATED STATE VARIABLE
;
    MPY      X0,X1,A      A,Y:(R4)      ;a21*x1
    MAC      X0,Y0,A      X:(R0)+,X0      B,Y0      ;a21*U
    MAC      X0,Y1,A      Y:(R4)+,X1      ;a22*x2
;
;
PERFORMS MAGNITUDE TRUNCATION ON UPDATED STATE VARIABLE
;
    NEG      A            A,Y1
    MOVE     #0,A0
    NEG      A            X:(R0)+,X0      ;SCALE
    TGT      Y1,A
;
;
SCALES THE OUTPUT
;
    MPY      Y0,X0,B      A,Y:(R4)      X:(R0)+,X0
;
;
STORES THE OUTPUT IN AN OUTPUT FILE
;
    MOVE     b0,Y:$3      ;OUTPUT
LOOP
    END

```

APPENDIX I

```

;*****
;   Program name:  FIFTH.ASM
;
;   This program calculate the output of the fifth order
;   digital filter discussed in Chapter IV using state
;   space realization coefficients in tables 4, 5, 6
;
;   Both second order section have limit cycles
;*****

```

$$H(z) = \frac{0.013547642z^2 - 0.047596084z + 0.028850883}{z^5 - 1.589611z^4 + 1.682092707z^3 - 0.713784839z^2 + 0.241565521z + 0.047634177}$$

```

OPT      CEX
ORG      X:$0

```

$$H(z) = \frac{-0.148479z + 0.406006}{z^2 - 0.52427z + 0.433577}$$

GK

```

DC      -0.2518445      ;c1
DC      -0.6455710      ;c2
DC      0.5622541       ;gamma
DC      0.2621350       ;a11
DC      -0.5582184      ;a12
DC      0.6536193       ;a21
DC      0.2621350       ;a22
DC      $000001

```

$$H(z) = \frac{0.16149z - 0.125769}{z^2 - 1.195305z + 0.8453360}$$

GG

```

DC      0.1543282
DC      -0.1780386
DC      0.9005735

```

$$H(z) = \frac{-0.565006}{z+0.129964}$$

```

MPY      X0,X1,B      X: (R0)+,X0      Y: (R4)-,Y1      ;C1*X1; C2,U
MACR     X0,Y1,B      X: (R0)+,X0      Y: (R7),Y0      ;C2*X2; A11
MPYR     X0,Y0,A      X: (R0)+,X0      ;GAM*U; C1,X2
MPY      X0,X1,A      A,Y0      ;A11*X1
MAC      X0,Y0,A      X: (R0)+,X0      ;A11*U; A12,X2
MAC      X0,Y1,A      ;A12*X2
NEG      A      A,X0
MOVE     #0,A0
NEG      A
TGT      X0,A
SUB      Y0,A      X: (R0)+,X0      ;-GAMMA*U; A21

MPY      X0,X1,A      A,Y: (R4)+      ;A21*X1; STORES X1
MAC      X0,Y0,A      X: (R0)+,X0      B,Y0      ;A21*GAMMA*U,A22 OUTPUT

MAC      X0,Y1,A      Y: (R5)+,X1      ;A22*X2;NEXT X1
NEG      A      A,Y1
MOVE     #0,A0
NEG      A      X: (R0)+,X0      ;SCALE
TGT      Y1,A
MPY      Y0,X0,B      A,Y: (R4)-      X: (R1)+,X0
;SCALE*OUTPUT,STORES X2 GAM2
MOVE     b0,y:$80      ;OUTPUT

;;;;;;;;;;;;;
;      CALCULATES THE OUTPUT OF A SECOND ORDER SECTION
;;;;;;;;;;;;;

MOVE     Y: (R7),Y0
MPYR     X0,Y0,A      X: (R1)+,X0      Y: (R5)-,Y1      ;GAM*U; C1,X2

MPY      X0,X1,B      X: (R1)+,X0      ;C1*X1; C2,U
MACR     X0,Y1,B      X: (R1)+,X0      A,Y0      ;C2*X2; A11
MPY      X0,X1,A      ;A11*X1
MAC      X0,Y0,A      X: (R1)+,X0      ;A11*U; A12,X2
MAC      X0,Y1,A      ;A12*X2
NEG      A      A,X0
MOVE     #0,A0
NEG      A
TGT      X0,A
SUB      Y0,A      X: (R1)+,X0      ;-GAMMA*U; A21

MPY      X0,X1,A      A,Y: (R5)+      ;A21*X1; STORES X1
MAC      X0,Y0,A      X: (R1)+,X0      B,Y0      ;A21*GAMMA*U,A22 OUTPUT

MAC      X0,Y1,A      Y: (R6),X1      ;A22*X2;
NEG      A      A,Y1
MOVE     #0,A0
NEG      A      X: (R1)+,X0      ;SCALE
TGT      Y1,A
MPY      Y0,X0,B      A,Y: (R5)-      X: (R2)+,X0
;SCALE*OUTPUT,STORES X2

```


APPENDIX J

```

;*****
;   Program name:  RFIL.ASM
;
;   This program calculates the output of a second order
;   digital filter using a state space realization with
;   all=a22
;*****

```

```

                                OPT   CEX
                                ORG   X:$0

GK                                DC   -0.07187541455251370      ;C1
                                DC   0.3029353689105536        ;C2
                                DC   0.5976524949073792        ;a11
                                DC   0.7635031715032031        ;a12
                                DC   0.0                        ;b1
                                DC   -0.6393523085997628        ;a21
                                DC   0.5976524949073792        ;a22
                                DC   0.5330839839355270        ;b2

                                ORG   Y:$0

TK                                DC   -0.042435                ;x1
                                DC   -0.7690                    ;x2

                                ORG   P:$0
                                JMP   <START

START                            MOVE   #GK,R0
                                MOVE   #7,M0
                                MOVE   #TK,R4
                                MOVE   #1,M4
                                NOP
                                MOVE   X:(R0)+,X0      Y:(R4)+,Y0
                                DO   #0,LOOP

                                MOVE   Y:$2,Y1

;
;   CALCULATE THE OUTPUT
;
                                MPY   X0,Y0,A      X:(R0)+,X0      Y:(R4)+,Y0 ;C1*x1
                                MAC   X0,Y0,A      X:(R0)+,X0      Y:(R4)+,Y0 ;C2*x2
                                MOVE   A,Y:$3

```

```

;
;
;
    CALCULATES THE NEXT STATE VARIABLE FOR  $x_1$ 

    MPY  X0,Y0,A      X:(R0)+,X0      Y:(R4)+,Y0      ; $a_{11} * x_1$ 

    MAC  X0,Y0,A      X:(R0)+,X0
    MACR X0,Y1,A      X:(R0)+,X0      Y:(R4)+,Y0      ; $a_{12} * x_2$ 
                                           ; $b_1 * U$ 

;
;
;
    CALCULATES THE NEXT STATE VARIABLE FOR  $x_2$ 

    MPY  X0,Y0,B      X:(R0)+,X0      Y:(R4)+,Y0      ; $a_{21} * x_1$ 
    MAC  X0,Y0,B      X:(R0)+,X0      ; $a_{22} * x_2$ 

    MACR X0,Y1,B      A1,Y:(R4)+
    MOVE B1,Y:(R4)-
    MOVE X:(R0)+,X0    Y:(R4)+,Y0      ; $b_2 * U$ 

LOOP

    END

```

VITA

Tyrone Reid was born on February 20, 1969. He attended school in Jones County, Georgia and graduated from Jones County High School in June, 1987. From June, 1987 to May 1991, he attended Tuskegee University in Tuskegee, Alabama and obtained a Bachelor of Science degree in Electrical Engineering. In August 1991, he enrolled in The University of Tennessee Space Institute in Tullahoma, TN. He was awarded the Master of Science degree in Electrical Engineering in May, 1993.