A Two-Level Pipelined Implementation of Direct–Form Recursive Filters

Zhou B. B. and Brent R. P.
Computer Sciences Laboratory
Australian National University
Canberra, Australia

Report TR-CS-88-06
April 1988

Abstract

A stabilized parallel algorithm for direct-form recursive filters is obtained using a new method of derivation in the Z domain. The algorithm is regular and modular, so very efficient VLSI architectures can be constructed to implement it. The degree of parallelism in these implementations can be chosen freely, and is not restricted to be a power of two.

1. Introduction

Recursive filtering is one of the most important techniques in digital signal processing. To achieve high-sampling rate computation of this kind of problems, a lot of efforts has been made in the past a few years, for example, [1][5][6][7] and [10]. The two-level pipeline, first introduced by Kung and his colleagues[2][3], is a good method for not only achieving high-sampling rate computation, but also reducing a great amount of area in VLSI implementation. An efficient pipelined, or two-level pipelined, architecture for the recursive filters of state-variable form was recently introduced by Parhi and Messerschmitt[7]. They use first the look-ahead computation to obtain the pipelined algorithm and then the decomposition technique to reduce the number of multiplications in the pipelined algorithm. They also show that the block (or parallel) processing can be combined with the pipelining to further increase the sampling rate.

The look-ahead computation concept may be applied to the direct-form implementation of recursive filters to achieve parallel computation[8][9][11][12]. However, it can cause numerical instability if the effect of finite wordlength is taken into consideration. Thus this paper will introduce a stablized algorithm for computing direct-form recursive filters. It leads to a more efficient two-level pipelined structure. (Although the pipeline and parallel processing is achievable, we only deal with the pipelining in this paper.) We shall see that this algorithm can save a number of multiplications and that the corresponding structure is more regular than the one described in [7]. Thus it is more suitable for VLSI implementation.

E-mail addresses: {bing,rpb}@cslab.anu.edu.au
 rpb104tr typeset using TEX

In Section 2, an example is given to show that the look-ahead computation may cause numerical instability in the direct-form implementation of recursive filters. Therefore, we derive a stablized algorithm and then the corresponding two-level pipelined structure in Section 3 and 4, respectively. Section 5 analyses the stability and compexity of the derived algorithm. In Section 6 we compare our method with one in [7].

2. Applying The Look-Ahead Computation To Direct-Form Recursive Filters

In the look-ahead technique, the given recursion is iterated as many times as desired to create the necessary concurrency and then the concurrency created can be used to obtain pipelined and/or parallel implementation of recursive systems[7]. In the following, we give an example to show that this concept may cause numercal instability in direct-form recursive filters if the effect of finite wordlength in practice is considered.

A second-order direct-form recursive filter can be expressed as

$$y_i = \sum_{j=0}^{2} w_j x_{i-j} + \sum_{j=1}^{2} r_j y_{i-j} \tag{1}$$

where w_j and r_j are coefficients and x_i and y_i are input and output, respectively.

According to (1), we write y_{i-1} explicitly as

$$y_{i-1} = \sum_{j=0}^{2} w_j x_{i-1-j} + \sum_{j=1}^{2} r_j y_{i-1-j} \tag{2}$$

Letting $j' = j + 1$, then

$$y_{i-1} = \sum_{j'=1}^{3} w_{j'-1} x_{i-j'} + \sum_{j'=2}^{3} r_{j'-1} y_{i-j'} \tag{3}$$

To apply the look-ahead computation concept, we substitute (3) into (1). Then

$$\begin{aligned} y_i &= \sum_{j=0}^{2} w_j x_{i-j} + r_1 y_{i-1} + r_2 y_{i-2} \\ &= \sum_{j=0}^{2} w_j x_{i-j} + r_1 (\sum_{j=1}^{3} w_{j-1} x_{i-j} + \sum_{J=2}^{3} r_{j-1} y_{i-j}) + r_2 y_{i-2} \\ &= \sum_{j=0}^{3} w'_j x_{i-j} + \sum_{j=2}^{3} r'_j y_{i-j} \end{aligned} \tag{4}$$

where $w'_0 = w_0$, $w'_1 = w_1 + r_1 w_0$, $w'_2 = w_2 + r_1 w_1$, $w'_3 = r_1 w_2$, $r'_2 = r_1^2 + r_2$ and $r'_3 = r_1 r_2$.

Since y_i in (4) depends upon y_{i-2} and y_{i-3}, two outputs can be computed simultanously.

To analyse the stability of the modified algorithm, we transform (4) into the Z domain as

$$Y(z) = \sum_{j=0}^{3} w'_j X(z) z^{-j} + \sum_{j=2}^{3} r'_j Y(z) z^{-j} \tag{5}$$

We then obtain the impulse response function as

$$\begin{aligned} H'(z) = \frac{Y(z)}{X(z)} &= \frac{\sum_{j=0}^{3} w'_j z^{-j}}{1 - \sum_{j=2}^{3} r'_j z^{-j}} \\ &= \frac{w_0 + (w_1 + r_1 w_0) z^{-1} + (w_2 + r_1 w_1) z^{-2} + r_1 w_2 z^{-3}}{1 - (r_1^2 + r_2) z^{-2} - r_1 r_2 z^{-3}} \\ &= \frac{(w_0 + w_1 z^{-1} + w_2 z^{-2})(1 + r_1 z^{-1})}{(1 - r_1 z^{-1} - r_2 z^{-2})(1 + r_1 z^{-1})} \\ &= H(z) \frac{1 + r_1 z^{-1}}{1 + r_1 z^{-1}} \end{aligned} \tag{6}$$

where $H(z)$ and $H'(z)$ are the impulse response function before and after the modification, respectively.

From (6) we see that, to obtain a parallel algorithm, we have multiplied both numerator and denominator of the original impulse response by a factor $1 + r_1 z^{-1}$. Suppose that the two poles of $H(z)$ are in the unit circle, say, 0.7 and 0.8. The denominator of $H(z)$ can be given as $1 - 1.5z^{-1} - 0.56z^{-2}$, that is, $r_1 = 1.5$ and $r_2 = 0.56$. However, the root of $1 + r_1 z^{-1}$ is -1.5, which is outside the unit circle. If the effect of finite wordlength in practice is taken into consideration, this will definitely cause instability.

3. Algorithm

In this section, we derive a new parallel algorithm. This algorithm is not only cost-effective in VLSI implementation, but is guaranteed to be stable. if the original (serial) algorithm is stable.

The impulse response of an N^{th} order recursive filter, $H(z)$, can be expressed as

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{j=0}^{N} w_j z^{-j}}{1 - \sum_{j=1}^{N} r_j z^{-j}} \tag{7}$$

where $X(z)$ and $Y(z)$ represent the Z transformations of input and output, respectively.

We introduce an $N \times N$ matrix B as follows

$$B = \begin{pmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ r_N & r_{N-1} & r_{N-2} & \dots & r_1 \end{pmatrix} \tag{8}$$

In the matrix B, the elements on the first superdiagonal are all equal to one and the j^{th} element on the last row is r_j, the coefficient of the denominator in (7), but all other elements are equal to zero.

In the following lemma, we prove that, by using the matrix B, the denominator of the impulse response in (7) can be defined as a matrix form.

Lemma 1: Given B as that in (8), we then have

$$1 - \sum_{j=1}^{N} r_j z^{-j} = \det(I - Bz^{-1}) \tag{9}$$

where $\det(x)$ denotes the determinant of the matrix x.

Proof : First , if $N = 2$, then

$$\begin{aligned} I - Bz^{-1} &= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} - \begin{pmatrix} 0 & 1 \\ r_2 & r_1 \end{pmatrix} z^{-1} \\ &= \begin{pmatrix} 1 & -z^{-1} \\ -r_2 z^{-1} & 1 - r_1 z^{-1} \end{pmatrix} \end{aligned}$$

We have

$$\det(I - Bz^{-1}) = 1 - r_1 z^{-1} - r_2 z^{-2} = 1 - \sum_{j=1}^{2} r_j z^{-j}$$

By induction, assume that (9) holds for $N = K - 1$, that is,

$$\begin{aligned} \det(I - Bz^{-1}) &= \det \begin{pmatrix} 1 & z^{-1} & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & -z^{-1} \\ -r_{K-1} z^{-1} & -r_{K-2} z^{-1} & \dots & -r_2 z^{-1} & 1 - r_1 z^{-1} \end{pmatrix} \\ &= 1 - \sum_{j-1}^{K-1} r_j z^{-j} \end{aligned} \tag{10}$$

For $N = K$, we have

$$I - Bz^{-1} = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{pmatrix} - \begin{pmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ r_K & r_{K-1} & r_{K-2} & \dots & r_1 \end{pmatrix} z^{-1}$$
$$= \begin{pmatrix} 1 & -z^{-1} & 0 & \dots & 0 \\ 0 & 1 & -z^{-1} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -r_K z^{-1} & -r_{K-1}z^{-1} & -r_{K-2}z^{-1} & \dots & 1 - r_1 z^{-1} \end{pmatrix} \tag{11}$$

By expanding $\det(I - Bz^{-1})$ with respect to the first column, we have

$$\det(I - Bz^{-1}) = \det \begin{pmatrix} 1 & -z^{-1} & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ -r_{K-1}z^{-1} & -r_{K-2}z^{-1} & \dots & -r_2 z^{-1} & 1 - r_1 z^{-1} \end{pmatrix} +$$
$$+ (-1)^{K+1}(-r_K z^{-1})\det \begin{pmatrix} -z^{-1} & 0 \dots & 0 & 0 \\ 1 & -z^{-1} & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & -z^{-1} \end{pmatrix} \tag{12}$$

The second cofactor of B on the right-hand side of (12) is a banded triangular matrix, and we have

$$\det \begin{pmatrix} -z^{-1} & 0 & \dots & 0 & 0 \\ 1 & -z^{-1} & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & -z^{-1} \end{pmatrix} = (-z^{-1})^{K-1} \tag{13}$$

Substitute (10) and (13) into (12), we then obtain

$$\begin{aligned} \det(I - Bz^{-1}) &= 1 - \sum_{j=1}^{K-1} r_j z^{-j} + (-1)^{K+1}(-r_K z^{-1})(-z^{-1})^{K-1} \\ &= 1 - \sum_{j=1}^{K-1} r_j z^{-j} + (-1)^{2K+1} r_K z^{-K} \\ &= 1 - \sum_{j=1}^{K} r_j z^{-j} \end{aligned}$$

In order to obtain a parallel algorithm, we make the following modification. Multiplying both numerator and denominator of (7) by a factor $\det(\sum_{j=0}^{M-1} B^j z^{-j})$, giving

$$
\begin{aligned}
H(z) &= \frac{(\sum_{j=0}^{N} w_j z^{-j}) \det(\sum_{j=0}^{M-1} B^j z^{-j})}{(1 - \sum_{j=1}^{N} r_j z^{-j}) \det(\sum_{j=0}^{M-1} B^j z^{-j})} \\
&= \frac{(\sum_{j=0}^{N} w_j z^{-j}) \det(\sum_{j=0}^{M-1} B^j z^{-j})}{\det(I - Bz^{-1}) \det(\sum_{j=0}^{M-1} B^j z^{-j})} \\
&= \frac{(\sum_{j=0}^{N} w_j z^{-j}) \det(\sum_{j=0}^{M-1} B^j z^{-j})}{\det((I - Bz^{-1})(\sum_{j=0}^{M-1} B^j z^{-j}))}
\end{aligned}
\tag{14}
$$

It is known that

$$
(I - Bz^{-1})(\sum_{j=0}^{M-1} B^j z^{-j}) = I - B^M z^{-M} \tag{15}
$$

Substituting (15) into (14), we obtain

$$
H(z) = \frac{(\sum_{j=0}^{N} w_j z^{-j}) \det(\sum_{j=0}^{M-1} B^j z^{-j})}{\det(I - B^M z^{-M})} \tag{16}
$$

To analyse the above modified algorithm, we divide (16) into two parts:

$$
H(z) = \frac{Y(z)}{U(z)} \frac{U(z)}{X(z)} = H_2(z) H_1(z) \tag{17}
$$

In the above equation, $U(z)$ is an intermediate variable, $H_2(z)$ and $H_1(z)$ are the Z transformations of the impulse response of recursive convolution and linear convolution, respectively, which are defined as follows

$$
H_2(z) = \frac{1}{\det(I - B^M z^{-M})} \tag{18}
$$

and

$$
H_1(z) = (\sum_{j=0}^{N} w_j z^{-j}) \det(\sum_{j=0}^{M-1} B^j z^{-j}) \tag{19}
$$

3.1. Recursive convolution

We first analyze the recursive convolution part $H_2(z)$. We show that the denominator of $H_2(z)$ is an NM^{th} order polynomial with only $N+1$ terms and that there are N multiplications required for implementing $H_2(z)$.

Lemma 2. Suppose that B is an $N \times N$ matrix, $\det(I - B^M z^{-M})$ can be expressed as an NM^{th} order polynomial with only $N+1$ terms, that is,

$$\det(I - B^M z^{-M}) = 1 - \sum_{j=1}^{N} b_j z^{-jM} \tag{20}$$

where b_j is a combination of some elements in B^M.

Proof : Since B is an $N \times N$ matrix, the product of M matrices B is also an $N \times N$ matrix. Letting $\lambda = z^M$, then

$$\det(I - B^M z^{-M}) = \det(I - B^M \lambda^{-1}) \tag{21}$$

from Lemma 1, we have

$$\det(I - B^M \lambda^{-1}) = 1 - \sum_{j=1}^{N} b_j \lambda^{-j} \tag{22}$$

where b_j is a combination of some elements in B^M.

Substituting z^M for λ in (22), we then obtain

$$\det(I - B^M z^{-M}) = 1 - \sum_{j=1}^{N} b_j z^{-jM}$$

Form (18) and (20), we have

$$H(z) = \frac{Y(z)}{U(z)} = \frac{1}{1 - \sum_{j=1}^{N} b_j z^{-jM}} \tag{23}$$

Converting (23) into the time domain, we obtain

$$y_n = \sum_{j=1}^{N} b_j y_{n-jM} + u_n \tag{24}$$

It is easy to see, from (24), that the number of multiplications required for computing the recursive convolution by the modified algorithm is N. Because y_n in (24) depends only on y_{n-jM} for $j = 1$ to N, M outputs can be computed simultanously. That is why we call our modified algorithm a parallel algorithm.

In the following, we give an example with $N = 2$.

Suppose that $B^M = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$. Then

$$I - B^M z^{-M} = \begin{pmatrix} 1 - b_{11}z^{-M} & -b_{12}z^{-M} \\ -b_{21}z^{-M} & 1 - b_{22}z^{-M} \end{pmatrix} \tag{25}$$

We have

$$\begin{aligned} \det(I - B^M z^{-M}) &= (1 - b_{11}z^{-M})(1 - b_{22}z^{-M}) - b_{12}b_{21}z^{-2M} \\ &= 1 - (b_{11} + b_{22})z^{-M} - (b_{12}b_{21} - b_{11}b_{22})z^{-2M} \\ &= 1 - \mathrm{tr}(B^M)z^{-M} - (-\det(B^M))z^{-2M} \\ &= 1 - \sum_{j=1}^{2} b_j z^{-jM} \end{aligned} \tag{26}$$

where $\mathrm{tr}(B^M)$ denotes the trace of B^M, $b_1 = \mathrm{tr}(B^M)$ and $b_2 = -\det(B^M)$.

Since $H_2(z) = \frac{Y(z)}{U(z)} = \frac{1}{1-\sum_{j=1}^{2} b_j z^{-jM}}$, then

$$y_n = b_1 y_{n-M} + b_2 y_{n-2M} + u_n \tag{27}$$

3.2. Linear convolution

We now analyse the linear convolution part.

Lemma 3. Suppose B is an $N \times N$ matrix, $\det(\sum_{j=0}^{M-1} B^j z^{-j})$ is an $N(M-1)^{th}$ polynomial.

Proof : From Lemma 2, $\det(I - B^M z^{-M})$ is an NM^{th} order polynomial. We have also known, from Lemma 1, that $\det(I - Bz^{-1})$ is an N^{th} roder polynomial. However, we have $\det(I - B^M z^{-M}) = \det(I - Bz^{-1})\det(\sum_{j=0}^{M-1} B^j z^{-j})$. The order of $\det(\sum_{j=0}^{M-1} B^j z^{-j})$ must be $NM - N = N(M-1)$.

Since $\det(\sum_{j=0}^{M-1} B^j z^{-j})$ is an $N(M-1)^{th}$ order polynomial, $N(M-1)$ multiplications are required for implementing it. Therefore, extra $N(M-1)$ multiplications have been introduced in the modified algorithm. It is not very practical if either N or M is large. By using the decomposition technique[7], however, this number of multiplications can greatly be reduced. In [7], the decomposition technique was used only in the case when M is a power of two, that is, $M = 2^K$. The following four lemmas extend the use of this technique to more general cases.

Lemma 4. If $M = m_1 m_2$, where m_1 and m_2 are integers, then $\sum_{j=0}^{M-1} B^j z^{-j}$ can be expressed as

$$\sum_{j=0}^{M-1} B^j z^{-j} = (\sum_{j=0}^{m_2-1} (Bz^{-1})^{jm_1})(\sum_{j=0}^{m_1-1} (Bz^{-1})^j) \tag{28}$$

where $B^0 = I$.

Proof : We arrange M terms of $\sum_{j=0}^{M-1} B^j z^{-j}$ into m_2 groups with m_1 terms each as follows

$$\begin{aligned}\sum_{j=0}^{M-1} B^j z^{-j} = {} & [I + Bz^{-1} + \cdots + (Bz^{-1})^{m_1-1}] + \\ & + [(Bz^{-1})^{m_1} + (Bz^{-1})^{m_1+1} + \cdots + (Bz^{-1})^{2m_1-1}] + \cdots \\ & + [(Bz^{-1})^{(m_2-1)m_1} + (Bz^{-1})^{(m_2-1)m_1+1} + \cdots + (Bz^{-1})^{m_2 m_1-1}]\end{aligned} \tag{29}$$

Letting

$$Q = I + Bz_{-1} + \cdots + (Bz^{-1})^{m_1-1} = \sum_{j=0}^{m_1-1} (Bz^{-1})^j \tag{30}$$

for the i^{th} group in (29) we have

$$(Bz^{-1})^{(i-1)m_1} + (Bz^{-1})^{(i-1)m_1+1} + \cdots + (Bz^{-1})^{im_1-1} = (Bz^{-1})^{(i-1)m_1} Q \tag{31}$$

We then obtain

$$\begin{aligned}\sum_{j=0}^{M-1} B^j z^{-j} &= Q + (Bz^{-1})^{m_1} Q + \cdots + (Bz^{-1})^{(m_2-1)m_1} Q \\ &= [I + (Bz^{-1})^{m_1} + \cdots + (Bz^{-1})^{(m_2-1)m_1}] Q \\ &= (\sum_{j=0}^{m_2-1} (Bz^{-1})^{jm_1}) Q \\ &= (\sum_{j=0}^{m_2-1} (Bz^{-1})^{jm_1})(\sum_{j=0}^{m_1-1} (Bz^{-1})^j)\end{aligned}$$

Lemma 5. If $M = \prod_{k=1}^{K} m_k$, where m_k is an integer number, then

$$\sum_{j=0}^{M-1} B^j z^{-j} = \prod_{k=1}^{K} (\sum_{j=0}^{m_k-1} (Bz^{-1})^{j \prod_{i=1}^{k-1} m_i}) \tag{32}$$

where $B^0 = I$ and m_i's are not necessarily distinct.

Proof : It follows by induction on K from Lemma 4.

We give an example with $M = 12$. Since 12 can be expressed as a product of three prime numbers 3, 2 and 2, we have

$$\begin{aligned}\sum_{j=0}^{11} B^j z^{-j} = {} & (I + Bz^{-1} + B^2 z^{-2}) + (B^3 z^{-3} + B^4 z^{-4} + B^5 z^{-5}) + \\ & + (B^6 z^{-6} + B^7 z^{-7} + B^8 z^{-8}) + (B^9 z^{-9} + B^{10} z^{-10} + B^{11} z^{-11}) \\ = {} & (I + Bz^{-1} + B^2 z^{-2})((I + B^3 z^{-3}) + (B^6 z^{-6} + B^9 z^{-9})) \\ = {} & (I + Bz^{-1} + B^2 z^{-2})(I + B^3 z^{-3})(I + B^6 z^{-6})\end{aligned} \tag{33}$$

From the above example, it is easy to see that a large polynomial has been decomposed into a product of three small polynomials. Then $\sum_{j=0}^{11} B^j z^{-j}$ can be implemented on a three-stage cascaded structure with only $4N$ multiplications, instead of $11N$ multiplications, where we suppose B is an $N \times N$ matrix. This reduction of the number of multiplications can be formally expressed by the following two lemmas.

Lemma 6. Suppose that B is an $N \times N$ matrix and q and p are constants, then

$$\det(\sum_{j=0}^{q-1}(Bz^{-1})^{jp}) = 1 + \sum_{j=1}^{(q-1)N} b_j z^{-jp} \tag{34}$$

where $B^0 = I$ and b_j is a combination of some elements in $B^p, \cdots B^{(q-1)p}$.

Proof : Letting $z^p = \lambda$, then

$$\det(\sum_{j=0}^{q-1}(Bz^{-1})^{jp}) = \det(\sum_{j=0}^{q-1} B^{jp}\lambda^{-j}) \tag{35}$$

From Lemma 3, we know that $\det(\sum_{j=0}^{q-1} B^{jp}\lambda^{-j})$ is an $N(q-1)^{th}$ order polynomial. It can then be expressed as

$$\det(\sum_{j=0}^{q-1} B^{jp}\lambda^{-j}) = \sum_{j=0}^{(q-1)N} b_j \lambda^{-j} \tag{36}$$

where b_j is a combination of some elements in $B^p, \cdots B^{(q-1)p}$.

Since the coefficient of λ^0 is one both in $\det(I - B^M\lambda^{-M})$ and in $\det(I - B\lambda^{-1})$ (see Lemma 1 and 2), the coefficient of λ^0 in (36) must also be one. Then

$$\det(\sum_{j=0}^{q-1} B^{jp}\lambda^{-j}) = 1 + \sum_{j=1}^{(q-1)N} b_j \lambda^{-j} \tag{37}$$

Replacing λ by z^p in (37), we then obtain

$$\det(\sum_{j=0}^{q-1}(Bz^{-1})^{jp}) = 1 + \sum_{j=0}^{(q-1)N} b_j z^{-jp}$$

We see, from Lemma 6, that there are only $N(q-1)$ multiplications required to implement $\det(\sum_{j=0}^{q-1}(Bz^{-1})^{jp})$ although it is an $Np(q-1)^{th}$ order polynomial. By extending this result, we have Lemma 7.

Lemma 7. If $M = \prod_{k=1}^{K} m_k$, there are $N(\sum_{k=1}^{K} m_k - K)$ multiplications required for implementing $\det(\sum_{j=0}^{M-1} B^j z^{-j})$ by using the decomposition technique, where B is an $N \times N$ matrix.

Proof : From Lemma 5, we may have

$$
\begin{aligned}
\det\left(\sum_{j=0}^{M-1} B^j z^{-j}\right) &= \det\left(\prod_{k=1}^{K}\left(\sum_{j=0}^{m_k-1} (Bz^{-1})^{j\prod_{i=1}^{k-1} m_i}\right)\right) \\
&= \prod_{k=1}^{K} \det\left(\sum_{j=0}^{m_k-1} (Bz^{-1})^{j\prod_{i=1}^{k-1} m_i}\right)
\end{aligned}
\tag{38}
$$

We see, from the above equation, that $\det(\sum_{j=0}^{M-1} B^j z^{-j})$ can be expressed as a product of K small polynomials. It can then be implemented on a K-stage cascaded structure. From Lemma 6, however, there are $N(m_k - 1)$ multiplications required in the k^{th} stage. For K stages, the total number of multiplications is then $\sum_{k=1}^{K} N(m_k - 1)$ or $N(\sum_{k=1}^{K} m_k - K)$.

4. Structure

In this section, we give the pipelined structure associated with the algorithm that we derived in the previous section. We first introduce the structure for the recursive convolution and show that our modified algorithm can be easily used for pipelining. Then we describe the structure for the linear convolution. Since there is no feed-back in the linear convolution, we can use a uni-directional structure. The uni-directional structures can have as many second-level pipelined stages as desired[2]. Therefore, the two structures can be easily matched with the same sampling rate.

4.1. Recursive convolution

For convenience, we rewrite (24) as follows

$$
y_n = \sum_{j=1}^{N} b_j y_{n-jM} + u_n \tag{39}
$$

It is easy to construct a structure for computing this algorithm, as shown in Fig. 1†. Since y_n depends only on y_{n-jM} for $j = 1, 2, \cdots N$, NM delays are evenly distributed in the system. By using the cut theorem[2][4], a two-level pipeline with M stages can easily be obtained. Therefore the sampling rate is M times higher than that of the structure for computing the unmodified algorithm.

† Figures are omitted from this version.

4.2. Linear convolution

From (19), (34) and (38), the impulse response of linear convolution in the modified algorithm $H_1(z)$ can be expressed as a product of $K+1$ polynomials as

$$H_1(z) = (\sum_{j=0}^{N} w_j z^{-j}) \prod_{k=1}^{K} (1 + \sum_{j=1}^{(m_k-1)N} b_j^{(k)} z^{-j \prod_{i=1}^{k-1} m_i}) \tag{40}$$

A $K+1$ stage cascaded structure may then be used for implementing this algorithm. In the following, we only consider one stage, that is,

$$\frac{U(z)}{X(z)} = 1 + \sum_{j=1}^{(q-1)N} b_j z^{-jp} \tag{41}$$

where q and p is a constant and $X(z)$ and $U(z)$ are the Z transformations of input and output, respectively.

Converting (41) into the time domain, we have

$$u_n = x_n + \sum_{j=1}^{(q-1)N} b_j x_{n-jp} \tag{42}$$

The above equation can then be computed by using an uni-directional array, as shown in Fig. 2. Since the structure in Fig. 2 is uni-directional, as mentioned before it can have as many second-level pipelined stages as desired. Therefore, we can easily make Fig. 2 match Fig. 1 with the same sampling rate.

5. Stability and Complexity

We rewrite (16) as

$$
\begin{aligned}
H(z) &= \frac{(\sum_{j=0}^{N} w_j z^{-j})\det(\sum_{j=0}^{M-1} B^j z^{-j})}{\det(I - B^M z^{-M})} \\
&= \frac{z^{-N}(\sum_{j=0}^{N} w_j z^{N-j})\det(z^{-(M-1)}(\sum_{j=0}^{M-1} B^J z^{M-1-j}))}{\det(z^{-M}(Iz^M - B^M))}
\end{aligned}
\tag{43}
$$

Since we have assumed that B is an $N \times N$ matrix, then

$$
\det(z^{-M}(Iz^M - B^M)) = z^{-NM}\det(Iz^M - B^M) \tag{44}
$$

and

$$
\det(z^{-(M-1)}(\sum_{j=0}^{M-1} B^j z^{M-1-j})) = z^{-N(M-1)}\det(\sum_{j=1}^{M-1} B^j z^{M-1-j}) \tag{45}
$$

Substituting (44) and (45) into (43), we then have

$$
H(z) = \frac{(\sum_{j=0}^{N} w_j z^{N-j})\det(\sum_{j=1}^{M-1} B^j z^{M-1-j})}{\det(Iz^M - B^M)} \tag{46}
$$

Suppose that the original algorithm before the modification is stable. Then the roots of $\det(Iz - B)$ are all in the unit circle. This means that the eigenvalues of B, z_i's, are all in the unit circle. It is clear that the eigenvalues of B^M, which are z_i^M's, are also in the unit circle and closer to the origin than their corresponding z_i's. The stability of our modified algorithm is obvious.

In the previous discussion, we have shown that, after the modification, the pipelined structure increases the sampling rate M times. Here we compute the number of multiplications required in the structure for an N^{th} order recursive filter.

Suppose that B is an $N \times N$ matrix and $M = \prod_{k=1}^{K} m_k$, where m_k is a integer number. We know, from Lemma 2, that N multiplications are required for implementing $\det(I - B^M z^{-M})$, and from Lemma 7, that $N(\sum_{k=1}^{K} m_k - K)$ multiplications for $\det(\sum_{j=0}^{M-1} B^j z^{-j})$. It is easy to see that, for implementing $\sum_{j=0}^{N} w_j z^{-j}$, we need $N + 1$ multiplications. Therefore, the total number of multiplications required for an N^{th} order recursive filter is

$$
N + N(\sum_{k=1}^{K} m_k - K) + N + 1 = N(\sum_{k=1}^{K} m_k - K + 2) + 1 \tag{47}
$$

When m_k is not a prime number and can be expressed as a product of some prime numbers, the small polynomial on the right-hand side of (38) can be further decomposed. Therefore, in the best case when $M = \prod_{k=1}^{K'} p_k$ where p_k is a prime number, the total number of multiplications can be reduced to $N(\sum_{k=1}^{K'} p_k - K' + 2) + 1$. If M is a power of two, then the total number of multiplications is

$$
N + N\log_2 M + N + 1 = N(\log_2 M + 2) + 1 \tag{48}
$$

6. Comparison

In this section, we compare our pipelined algorithm and structure with those described in [7]. In [7], the authors use the look-ahead computation to obtain a pipelined algorithm for a first-order recursive filter of state-variable form and then the decomposition technique to reduce the number of multiplications involved in that algorithm. The method can of course be extended to high-order cases. For convenience, we briefly describe this method below.

The state-variable form of an N^{th} order recursive filter can be expressed in two equations, that is, the state update equation

$$X(n+1) = AX(n) + Bu(n) \tag{49}$$

and the output equation

$$y(n) = C^T X(n) + du(n) \tag{50}$$

where A is an $N \times N$ matrix, B and C are $N \times 1$ vectors, d is a scalar, $X(n)$ represents an $N \times 1$ state-variable vector and $u(n)$ and $y(n)$ are input and output, respectively.

In the above two equations, only the state update equation contains recursion. To obtain a pipelined algorithm, the look-ahead computation needs to be applied to that equation. Iterating (49) itself M' times, we have

$$\begin{aligned} X(n+M') &= A^{M'}X(n) + A^{M'-1}Bu(n) + \cdots + Bu(n+M'-1) \\ &= A^{M'}X(n) + \sum_{j=0}^{M'-1} A^{M'-1-j}Bu(n+j) \end{aligned} \tag{51}$$

where $A^0 = I$.

Since it can be precomputed, $A^{M'-1-j}B$ is just an $N \times 1$ vector. Therefore, to compute the sum on the right-hand side of (51) requires NM' multiplications. However, this number of multiplications can be reduced by using the decomposition technique. It is described as follows.

Transforming (51) into the Z domain,then

$$X(z)z^{M'} = A^{M'}X(z) + \sum_{j=0}^{M'-1} A^{M'-1-j}BU(z)z^j \tag{52}$$

or

$$\frac{X(z)}{U(z)} = \frac{\sum_{j=0}^{M'-1} A^{M'-1-j}Bz^{j-M'}}{I - z^{-M'}A^{M'}} \tag{53}$$

where $X(z)$ and $U(z)$ are the Z transformations of state-variable vector and input, respectively.

For the numerator of (53), we have

$$\begin{aligned} \sum_{j=0}^{M'-1} A^{M'-1-j}Bz^{j-M'} &= (\sum_{j=0}^{M'-1} A^{M'-1-j}z^{-(M'-1-j)})Bz^{-1} \\ &= (\sum_{j=0}^{M'-1} A^j z^{-j})Bz^{-1} \end{aligned} \tag{54}$$

From Lemma 5, $\sum_{j=0}^{M'-1} A^j z^{-j}$ can be expressed as a product form. Suppose, for simplicity, that M' is a power of two. Then

$$\begin{aligned}\sum_{j=0}^{M'-1} A^{M'-1-j} B z^{j-M'} &= (\prod_{k=1}^{\log_2 M'} (\sum_{j=0}^{1} (Az^{-1})^{j2^{k-1}}))Bz^{-1} \\ &= (I + Az^{-1})(I + A^2 z^{-2}) \cdots (I + A^{M'/2} z^{-M'/2}) B z^{-1}\end{aligned} \tag{55}$$

From (50), (53) and (55), we can calculate the total number of multiplications required in the pipelined implementation of an N^{th} order recursive filter in the state-variable form.

First we assume that A is an $N \times N$ full matrix. To implement the recursive part of (53), $I - z^{-M'} A^{M'}$, N^2 multiplications are required because it involves matrix-vector multiplications. In order to obtain an M stage two-level pipeline, however, M' should be equal to NM. This is because we have to distribute M' delays evenly to all basic processing elements in the system. (It can easily be proved by drawing a diagram and then using the cut theorem.) To implement the non-recursive part of (53), that is, the equation in (55), we need $\log_2 M' + 1$ cascaded stages. However, it requires N^2 multiplications to implement $I + A^i z^{-i}$. Then the number of multiplications for this computation is

$$N^2 \log_2 M' + N = N^2 \log_2(NM) + N \tag{56}$$

To compute the output equation, there are $N+1$ multiplications required. Therefore, the total number of multiplications is

$$N^2 + N^2 \log_2(NM) + N + N + 1 = N^2(\log_2(NM) + 1) + 2N + 1 \tag{57}$$

This number is much greater than that in our direct-form implementation, which is $N(\log_2 M + 2) + 1$.

The total number of multiplications can be decreased if A is not a full matrix. We consider the best case when A is a block-diagonal matrix with block size 2 by 2. (Note that A can not be a diagonal matrix in most cases. Otherwise complex numbers will appear in the computation.) For implementing the recusive part of (53), there are $2N$ multiplications required. M' is now only $2M$ and $2N$ multiplications are required for implementing $I + A^i z^{-i}$ in the non-recursive equation of (55). Therefore, the total number of multiplications becomes

$$2N + 2N \log_2(2M) + N + N + 1 = 2N(\log_2 M + 3) + 1 \tag{58}$$

This number is still greater than that in our implementation.

We can see, from (48) and (58), that the best case in the state-variable form implementation still requires twice the number of multiplications as our direct-form implementation. Furthermore, our structure is more regular. Thus it is more suitable for VLSI implementation.

7. Conclusions

Although the look-ahead computation is a good technique for obtaining the pipeline and/or parallel algorithm of an N^{th} order recursive filter in state-variable form, it may cause instability in the direct-form implementation. In this paper, we have used a new method for deriving a stablized algorithm of the direct-form recursive filter. This algorithm is not only guaranteed to be stable, but also leads directly to an efficient pipelined structure.(This algorithm can also leads to an efficient pipeline and parallel architecture[13].) We have shown that, to achieve the same sampling rate, our direct-form implementation of an N^{th} order recursive filter requires less multiplications than the state-variable form countpart. Our structure is also more regular and modular. Therefore, it is more suitable for VLSI implementation.

9. Postscript

We have recently learned that Parhi K. K. and Messerschmitt D. G.[14] have obtained a similar result using a different approach. The disadvantage of their method is that the decomposition technique can be applied only when M is a power of two. However, our method does not have this limitation.

9. References

[1] Barnes, C. W. and Shinnaka, S., Block shift invariance and block implementation of discrete-time filters, *IEEE Trans. Circuits Syst.*, vol. CAS-27, Aug. 1980, pp. 667-672.

[2] Kung, H. T. and Lam, M. S., Wafer-scale integration and two-level pipelined implementations of systolic arrays, *Journal of Parallel and Distributed Computing*, vol. 1, 1984, pp. 32-63.

[3] Kung, H. T., Ruane, L. M. and Yen, D. W. L., A two-level pipelined systolic array for convolutions, *in Proc. CMU Conference VLSI Syst. Comp.*, 1981, pp. 255-264.

[4] Kung, S. Y., On supercomputing with systolic/wavefront array processors, *Proc. IEEE*, vol. 72, No. 7, July 1984, pp. 867-884.

[5] Lu. H. H., Lee, E. A. and Messerschmitt, D. G., Fast recursive filters through transformation to block state variable form, *IEEE Trans. Circuits Syst.*, vol. CAS-32, Nov. 1985, pp.1119-1129.

[6] Nikias, C. L., Fast block data processing via new IIR digital filter structure, *IEEE Trans. Acoustics, Speech, Signal Process.*, vol. 32, No. 4, Aug. 1984.

[7] Parhi, K. K. and Messerschmitt, D. G., Concurrent cellular VLSI adaptive filter architectures, *IEEE Trans. Circuits Syst.*, vol. 10, Oct. 1987, pp. 1141-1151.

[8] Robert, Y. and Tchuente, M., An efficient systolic array for the 1D recursive convolution problem, *Journal of VLSI and Computer Systems*, vol. 1, No. 4, pp. 398-407.

[9] Wu, C. W. and Cappello, R., Application-specific CAD of high throughput IIR filters, *in Proc. Thirty-Second IEEE Computer Society International Conference*, Feb. 1987, pp. 302-305.

[10] Zeman, J. and Linsgren, A. G., Fast digital filters with low round-off noise, *IEEE Trans. Circuits Syst.*, vol. CAS-28, July 1981, pp.716-723.

[11] Zhou, B. B. and Brent, R. P., An efficient architecture for solving the recursive convolution equation with high throughput, *in Proc. 1st IASTED International Symposium on Signal Processing and Its Applications*, Aug. 1987, pp. 771-775.

[12] Zhou, B. B. and Brent, R. P., A high throughput systolic implementation of the second order recursive filter, *in Proc. IEEE Int. Conf. Acoustics, Speech, Signal Process.*, Apr. 1988, to appear.

[13] Zhou, B. B. and Brent, R. P., A stablized parallel algorithm for the direct-form recursive filter, Tech. Report, Comput. Sci. Lab., Australian National Uni., Australia, 1988.

[14] Parhi, K. K. and Messerschmitt, D. G., Pielined VLSI recursive filter architectures using scttered look-ahead and decomposition, *in Proc. IEEE Int. Conf. Acoustics, Speech, Signal Process.*, Apr. 1988, to appear.