論文 95-2-2-8

블럭상태공간식을 이용한 효율적인 2차원 디지탈 신호처리기

正會員 鄭 在 吉*, 正會員 張 鍾 煥**

An Efficient 2-D Digital Signal Processor Based Upon The Block State Space Representation

Jae Gil Jeong*, Jong Whan Jang** Regular Members

要 約

공간영역 2차원 IIR 디지탈 필터와 2차원 FIR 필터의 효율적인 구현 방안을 제안하였다. 2차원 필터의 블럭상태공간 함수를 이용하여 computational primitive를 유도하고, 유도된 computational primitive를 프로세서의 arithmetic unit의 구현에 사용함으로써 효율적인 2차원 디지탈 신호처리 시스템의 구현을 가능하게 하였으며, 2차원 디지탈 신호의 실시간 또는 준 실시간 처리가 가능한 다중프로세서 구조를 제안하였다. 또한, 블럭의 크기에 따른 각 프로세서의 throughput과 효율을 분석하여, 급속히 발전되는 VLSI기술로 인하여 높아진 집적도를 효율적으로 활용하기 위한 방안을 제시하였다.

ABSTRACT

This paper presents an efficient implementation of the spatial domain 2-D IIR and FIR filters. We derived computational primitives based upon a block state space representation. The computational primitive can be implemented as an arithmetic unit for a programmable processor, which can be used for the efficient implementation of a 2-D digital signal processing system. We present a multiprocessor implementation for a real-time or near real-time 2-D digital filter. We compared several implementations based upon the block processing with various block sizes for analyzing the throughput and efficiency for each processor. This analysis provides the information which can be used to improve the utilization of the available silicon area provided by the advanced VLSI technologies.

* 배재대학교 전자공학과
Dept. of Physics, Pai Chai University
** 배재대학교 정보통신공학과
Dept. of Infor. and Comm. Eng., Pai Chai University
論文番號 : 94239
接受日字 : 1994年 9月 5日

Ⅰ. INTRODUCTION

Real-time processing of the very large two-dimensional (2-D) data with the use of the high order filter requires a tremendous speed for the computations. During recent years, a number of approaches have been considered to achieve real-time digital filtering. Among them, Kim and Alexander [1] presented an architecture based upon a computational primitive. They derived the computational primitive from the state space representation of 2-D IIR filter. Their system does not implement the state space equation itself; rather they use the state space equation to decompose the algorithm to implement a modular architecture with a low number of regular data transfers. Zhang and Steenaart [2] presented a high speed architecture for recursive filters based upon block processing and local and global speed-up. Their architecture actually implements the state space equation for the 2-D recursive filter. Therefore, this architecture can provide the advantages of the block state space implementation such as minimum round-off error, absence of overflow oscillations, and low coefficient sensitivity [3]. However, it requires local data broadcasting which is undesirable for the VLSI implementation, especially for high order filters.

It is straightforward to exploit the concurrency of nonrecursive filters. Thus, they can be efficiently implemented with high parallelism. However, recursive filters can only be implemented in a sequential manner. This obviously imposes an upper limit the effective use of a large number of processing elements [4].

Lu, Lee, and Messerschmitt [5] proposed several systolic architectures for block implemented 2-D FIR and IIR digital filters which offer considerably higher sampling and throughput rates as compared with the single processing element. However, in order to utilize such advantages the state matrix in the block state-space equation should be transformed to a triangular or a quasi-triangular form using unitary or orthogonal similarity transformations. Parhi and Messerschmitt [6] introduced a look-ahead computation scheme for parallel implementation of the block state space equation. Azimi-Sadjadi and Rostampour [4] proposed parallel and pipelined architectures for 2-D recursive and nonrecursive block digital filters. Their architectures achieve high throughput with a large number of processing elements, which is proportional to the block size. However, these architectures require relatively complex custom designed processing elements and require a large number of block data transfers between the processing elements. Therefore, the effectiveness of these implementations depends upon the high speed communications between the processing elements. Thus, the performance of the communication channel may limit the overall system performance rather than the performance of the processing element itself.

In this paper, we present the extension of the implementation of 2-D IIR filter as presented by Kim and Alexander [1]. We present a real-time 2-D IIR and FIR filter implementation based upon the computational primitive which is derived from the block state space representation of the 2-D spatial domain digital filter.

Ⅱ. COMPUTATIONAL PRIMITIVE

A 2-D digital filter can be modeled as a discrete linear shift invariant (DLSI) system and can be implemented efficiently using the state space approach. A general order 2-D DLSI system with a quarter plane support can be represented by the finite difference equation as given by:

$$g(m,n)=\sum_{i=0}^{M}\sum_{j=0}^{N}a(i,j)f(m-i,n-j)-\sum_{\substack{i=0\\i+j>0}}^{M}\sum_{j=0}^{N}b(i,j)g(m-i,n-j) \quad (1)$$

The parameters a(i, j) and b(i, j) are coefficients which determine the characteristics of the system. If all of the b(i, j) are equal to zero, then we have an FIR filter. Otherwise, Eq.(1) represents an IIR filter. By taking the z-transform of Eq.(1), we can write the relationship between the transform of the input sequence, $F(z_1,z_2)$ and the transform of the output sequence, $G(z_1,z_2)$ as follows:

$$G(z_1,z_2)=a(0,0)F(z_1,z_2)+\sum_{i=0}^{M}\Big[\sum_{\substack{j=0\\i+j>0}}^{N}[a(i,j)F(z_1,z_2)-b(i,j)G(z_1,z_2)]z_1^{-i}\Big]z_2^{-j} \quad (2)$$

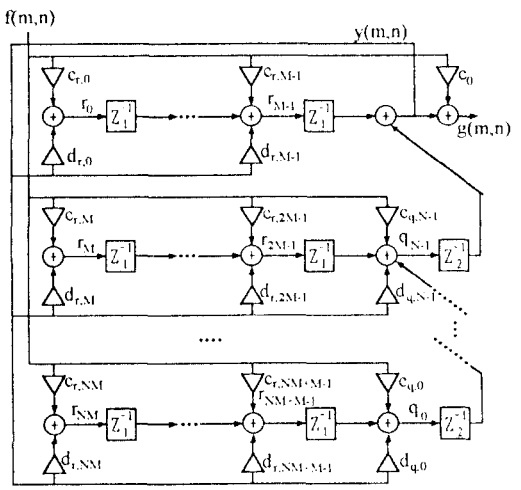

Fig. 1. A block diagram for the general order 2-D DLSI system

From this equation, we can obtain a block diagram for the 2-D DLSI system shown in figure 1. In this figure, there are two kinds of delay elements. z_1^{-1} is the pixel delay and z_2^{-1} is the line delay. We assigned the horizontal state variable r as the input to each z_1^{-1} delay and the vertical state variable q as the input to each z_2^{-1} delay. From this diagram, we can obtain the following state space equations [7]:

$$g(m,n)=c_0f(m,n)+y(m,n) \quad (3)$$

$$r_k(m,n)=c_{r,k}f(m,n)+d_{r,k}y(m,n)+r_{k-1}(m-1,n) \quad (4)$$

$$q_k(m,n)=c_{q,k}f(m,n)+d_{q,k}y(m,n)+r_{M(N+1-k)-1}(m-1,n)+q_{k-1}(m,n-1) \quad (5)$$

where

$c_0=a(0,0)$

$c_{r,k}=a(M-i,j)-a(0,0)\times b(M-i,j)$: $d_{r,k}=-b(M-i,j)$ for $0\le i\le M-1$, $0\le j\le N$, and $k=i+jM$

$c_{q,k}=a(0,N-j)-a(0,0)\times b(0,N-j)$: $d_{q,k}=-b(0,N-j)$ for $1\le j\le N$ and $k=j$

$y(m,n)=r_{M-1}(m-1,n)+q_{N-1}(m,n-1)$

$r_k(m,n)=0$ for $k<0$, $m<0$, or $n<0$

$q_k(m,n)=0$ for $k<0$, $m<0$ or $n<0$

The modified coefficients c's and d's are obtained from the a's and b's, and y is a temporary variable defined for simplification. If we use blocks of length L in the horizontal direction, we can obtain the block state space equations as followings:

Since the system is discrete linear shift invariant, we can obtain following equations from Eq.(4).

$r_{k-1}(m-1,n)=c_{r,k-1}f(m-1,n)+d_{r,k-1}y(m-1,n)+r_{k-2}(m-2,n)$

$r_{k-2}(m-2,n)=c_{r,k-2}f(m-2,n)+d_{r,k-2}y(m-2,n)+r_{k-3}(m-3,n)$

$\vdots$

$r_{k-L+1}(m-L+1)=c_{r,k-L+1}f(m-L+1,n)+d_{r,k-L+1}y(m-L+1,n)+r_{k-L}(m-L,n)$

Then, recursively substitute $r_{k-1}(m-1,n)$ through $r_{k-L+1}(m-L+1,n)$ to the above scalar state space equations until all the equations are expressed with $r(m-L,n)$. With this, we can obtain the following block state space equations.

$$R_k(m,n)=AF(m,n)+BY(m,n)+\overline{R}_{k-L}(m-L,n) \quad (6)$$

$$G(m,n)=c_oF(m,n)+Y(m,n) \quad (7)$$

$$r_k(m,n)=C_rF(m,n)+D_rY(m,n)+r_{k-L}(m-L,n) \quad (8)$$

$$Q_k(m,n)=C_qF(m,n)+D_qY(m,n)+\overline{R}_{M(N+1-k)-L}(m-L,n)+Q_{k-1}(m,n-1) \quad (9)$$

Where

$$Y(m,n)=C_yF(m,n)+D_yY(m,n)+\overline{R}_{M-L}(m-L,n)+Q_{N-1}(m,n-1) \quad (10)$$

and

$F(m,n)=[f(m,n)\ f(m-1,n)\cdots f(m-L+1,n)]^T$,

$G(m,n)=[g(m,n)\ g(m-1,n)\cdots g(m-L+1,n)]^T$,

$R_k(m,n)=[r_k(m,n)\ r_k(m-1,n)\cdots r_k(m-L+1,n)]^T$,

$\overline{R}_k(m,n)=[r_k(m,n)\ r_{k+1}(m,n)\cdots r_{k+L-1}(m,n)]^T$,

$Q_k(m,n)=[q_k(m,n)\ q_k(m-1,n)\cdots q_k(m-L+1,n)]^T$,

$Y(m,n)=[y(m,n)\ y(m-1,n)\cdots y(m-L+1,n)]^T$,

$$A=\begin{bmatrix} c_{r,k} & c_{r,k-1} & \cdots & c_{r,k-L+1} \\ 0 & c_{r,k} & \cdots & c_{r,k-L+2} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & c_{r,k} \end{bmatrix},$$

$$B=\begin{bmatrix} d_{r,k} & d_{r,k-1} & \cdots & d_{r,k-L+1} \\ 0 & d_{r,k} & \cdots & d_{r,k-L+2} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & d_{r,k} \end{bmatrix},$$

$$C_r=[\,c_{r,k}\ c_{r,k-1}\ \cdots\ c_{r,k-L+1}\,],$$

$$D_r=[\,d_{r,k}\ d_{r,k-1}\ \cdots\ d_{r,k-L+1}\,],$$

$$C_y=\begin{bmatrix} 0 & c_{r,M-1} & \cdots & c_{r,M-L+1} \\ 0 & 0 & \cdots & c_{r,M-L+2} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix},$$

$$D_y=\begin{bmatrix} 0 & d_{r,M-1} & \cdots & d_{r,M-L+1} \\ 0 & 0 & \cdots & d_{r,M-L+2} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix},$$

$$C_q=\begin{bmatrix} c_{q,k} & c_{r,M(N+1-k)-1} & \cdots & c_{r,M(N+1-k)-L+1} \\ 0 & c_{q,k} & \cdots & c_{r,M(N+1-k)-L+2} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & c_{q,k} \end{bmatrix},$$

$$D_q=\begin{bmatrix} d_{q,k} & d_{r,M(N+1-k)-1} & \cdots & d_{r,M(N+1-k)-L+1} \\ 0 & d_{q,k} & \cdots & d_{r,M(N+1-k)-L+2} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & d_{q,k} \end{bmatrix}.$$

Note that Eq. (10) is computable since all of the diagonal elements of D_y are equal to zeros. The equations, Eq. (7) to Eq. (9), are used to design the system. Among them, the computation of vertical state variable in Eq. (9) is the most complex. If we build a processor which can compute vertical state variable in a single cycle, we can compute each output or state variable from Eq. (7) to Eq. (9) in a single cycle. Therefore, we define this as a computational primitive for the processor. It requires 2L multiplications and 2L+1 additions.

For the 2-D FIR filter, b(i,j) in Eq. (1) and Eq. (2) are zeros. Therefore, the equations for 2-D FIR filters are given by:

$$G(m,n)=c_oF(m,n)+\overline{R}_{M-L}(m-L,n)+Q_{N-1}(m,n-1) \quad (11)$$

$$r_k(m,n)=C_rF(m,n)+r_{k-L}(m-L,n) \quad (12)$$

$$Q_k(m,n)=C_qF(m,n)+\overline{R}_{M(N+1-k)-L}(m-L,n)+Q_{k-1}(m,n-1) \quad (13)$$

As you can see from the above equations, we can define a computational primitive for the 2-D FIR filter with a block size of L that requires L multiplications and L+1 additions. Thus, a 2-D FIR filter with a block size of 2L and a 2-D IIR filter with a block size of L have the same computational primitive. Therefore, both 2-D FIR and IIR filter can be implemented efficiently with the same computational primitive.

If we implement a processor based on one specific computational primitive, the other algorithm, which has a different computational requirement, can not be implemented efficiently with this processor. We can achieve higher efficiency by finding a common computational primitive for both 2-D IIR and FIR filters.

We derived a common computational primitive from a block state space representation for the 2-D DLSI system. This approach has many benefits such as reducing the number of computational cycles required for each algorithm and reduc-

ing the number of processors required for a real-time implementation. However, it increases the complexity of the processor. Today's VLSI implementation and design technology are developing very rapidly. In a near future, we can implement more complex circuits in a single DSP chip.

Ⅲ. IMPLEMENTATION

The overall multiprocessor implementation is shown in figure 2. In this implementation, we define one row of data as one data block; we assign the first row of data to the first processor, the second row of data to the second processor, and so on. Generally, the number of processors is much smaller than the number of rows. If the number of processors is four, then we assign the fifth row of data to the first processor, the sixth row of data to the second processor, and so on. With this implementation scheme, horizontal state variables are not required to be transferred and vertical state variables are required to be transferred to the nearest neighbors. Therefore, the amounts of data transfers are reduced and only local data transfers are required.

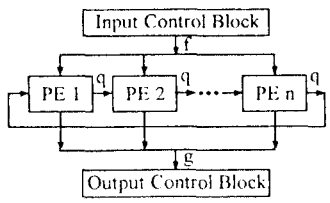

Fig. 2. A block diagram of the multiprocessor implementation

The use of this scheme permits us to achieve real-time throughput with a given number of processors. Each processor, as shown in figure 2, processes one row of data. The timing diagram shown in figure 3 explains how we can achieve real-time throughput with multiple processors with this implementation scheme. This figure was based on the assumption that the input data interval is 127 nsec. This rate is equivalent to the 30 frames per second of 512×512 video. We also assume that the processor can compute one computational primitive in 100 nsec, and that four equations need to be computed.

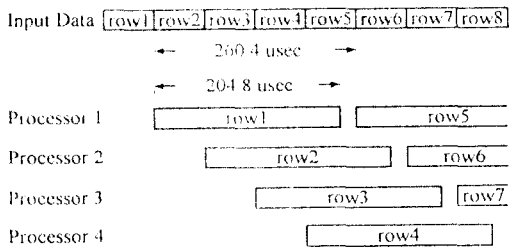

Fig. 3. The timing diagram for the example

As you can see in the figure, processor 1 starts processing as soon as the first row of data is available. During the processing, the processor computes the vertical state variables, q. They will be used for the processing of the next row of data. When the second row of data is available, processor 2 can start the processing immediately because the required vertical state variables are already available. Processor 1 starts the processing of fifth row of data as soon as the input data is available because the resource (processor) and all the required data (input data and vertical state variables) are available. After processor 4 starts the processing, every processor is simultaneously computing as long as input data is available. Therefore, we can achieve real-time processing with four processors.

All the processors are connected in a linear array with data communications only with the input control block, the output control block, and nearest neighbors. Data communications between the processors are always first-in first-out and buffered. If data is available, the processor operates at its full speed. Since the processors operate asynchronously, speedup is essentially linear as

we add additional processors.

For the general M×N-th order system based on the state space implementation with a block size of one, we need (M+1)(N+1) computations for each input data. Therefore, we can achieve real-time throughput with (M+1)(N+1) processors if each processor can compute one computational primitive in a single cycle. The number of processors can be greatly reduced by using the block state space implementation. If we use the block size of two, the required number of processors for the M×N-th order system becomes (M+2)(N+1)/2. Thus, the number of processors is reduced by almost one half for the high order system.

For the analysis, let T_1 be the number of cycles to compute Eq.(1) with a single processor that can compute one multiplication or one addition in a single cycle. Direct implementation of the M×N-th order 2-D IIR filter requires 2(M+1)(N+1)-1 multiplications and 2(M+1)(N+1)-2 additions. Thus, T_1 is equal to 4(M+1)(N+1)-3 for the direct implementation. Let P be the number of these processors used in a multiprocessor system and let T_p be the cycles required to process one pixel with a single processor that can compute the computational primitive in a single cycle. Note that T_p is also the number of state equations to be computed per pixel plus one (for computing the output). For the M×N-th order 2-D IIR and FIR filter with a block size of L, the numbers of equations to be implemented for each pixel are as follows : L for the outputs, M(N+1) for the horizontal state variables, and LN for the vertical state variables. Thus, a total (M+L)(N+1) equations have to be computed to obtain L outputs. Therefore, $T_p=(M+L)(N+1)/L$. The maximum possible speedup and efficiency can be defined as $S_p=T_1/T_p$ and $E_p=S_p/P$, respectively. Table 1 and Table 2 give the comparisons of the performance of N×N-th order 2-D IIR and FIR filters for various block sizes, respectively. As you can see from these tables, the higher efficiency can be obtained with a larger block size for the high order filter.

Table 1. Performance of 2-D IIR filters for various block sizes

Block size		1	2	4	8
P		5	9	17	33
Order=2	T_p	9	6	4.5	3.75
T_1=33	S_p	3.67	5.50	7.33	8.80
	E_p	0.73	0.61	0.43	0.27
Order=8	T_p	81	45	27	18
T_1=321	S_p	3.96	7.13	11.89	17.83
	E_p	0.79	0.79	0.70	0.54
Order=32	T_p	1089	561	297	165
T_1=4353	S_p	4.00	7.76	14.66	26.38
	E_p	0.80	0.86	0.89	0.80
Order=128	T_p	16641	8385	4257	2193
T_1=66561	S_p	4.00	7.94	15.64	30.35
	E_p	0.80	0.88	0.92	0.91
Order=512	T_p	263196	131841	66177	33345
T_1=1052673	S_p	4.00	7.98	15.91	31.57
	E_p	0.80	0.89	0.94	0.96

Table 2. Performance of 2-D FIR filters for various block sizes

Block size		1	2	4	8
P		3	5	9	17
Order=2	T_p	9	6	4.5	3.75
T_1=17	S_p	1.89	2.83	3.78	4.53
	E_p	0.63	0.57	0.54	0.27
Order=8	T_p	81	45	27	18
T_1=161	S_p	1.99	3.58	5.96	8.94
	E_p	0.66	0.72	0.66	0.53
Order=32	T_p	1089	561	297	165
T_1=2177	S_p	2.00	3.88	7.33	13.19
	E_p	0.87	0.78	0.81	0.78
Order=128	T_p	16641	8385	4257	2193
T_1=33281	S_p	2.00	3.97	7.82	15.18
	E_p	0.87	0.79	0.87	0.89
Order=512	T_p	263196	131841	66177	33345
T_1=526337	S_p	2.00	3.99	7.95	15.78
	E_p	0.87	0.80	0.88	0.93

Ⅳ. COMPARISON

We consider speedup and efficiency to be appro-

priate measures for the performance of a multiprocessor system. With a given multiprocessor architecture, we can achieve an almost linear speedup as the number of processors increases. Therefore, the combination of the hardware complexity for each processing element and the throughput provided by a single processing element can determine the performance of the overall multiprocessor system. There are two ways to utilize VLSI technology to improve the performance of a multiprocessor system. One method is to implement many simple processing elements on a single VLSI chip. The other method is to implement a smaller number of more complex processing elements on a single VLSI chip. The maximum number of processing elements that can be placed on a single VLSI chip is determined by the hardware complexity for each processing element and the available VLSI technology.

In order to evaluate the tradeoffs between hardware complexity, algorithm complexity and throughput, we selected four computational primitives as shown in figure 4 for comparison. These are as follows:

CP-1 One multiplication with an accumulation that is a generic primitive for the 1-D FIR filter and is used widely for commercial, programmable DSPs,

CP-2 One multiplication with two additions that is a primitive for the 2-D FIR filter with a block size of one,

CP-3 Two multiplications and three additions that is a primitive for the 2-D IIR filter with a block size of one or a primitive for the 2-D FIR filter with a block size of two. This primitive was suggested by Kim and Alexander[1], and

CP-4 Four multiplications and five additions that is a primitive for the 2-D IIR filter with a block size of two or a primitive for the 2-D FIR filter with a block size of four.

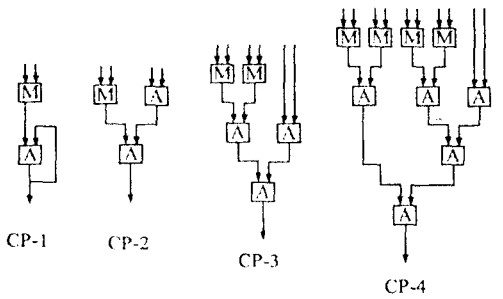

Fig. 4. Computational primitives for the implementation of 2-D digital filter (A : Adder, M : Multiplier)

The throughput comparisons between the implementations with various block sizes and various computational primitives are shown in table 3 and table 4. As you can see from these tables, the maximum throughput for the 2-D IIR filter can be obtained from CP-4 with block processing with a block size of 2 and the maximum throughput for the 2-D FIR filter can be obtained from CP-4 with block processing with a block size of 4. We can conclude that the maximum throughput can be obtained with block processing with a matching block size when a processor based on the computational primitive derived from the largest block size is used. That is, the suggested implementation can provide higher throughput than the implementation suggested by Kim and Alexander [1] and the implementation based upon CP-1 such as commercial DSPs. However, the maximum implementable size for the computational primitive is limited because of the available VLSI technology and other constraints such as design cost and time.

In order to obtain a performance measure for the efficiency, we defined the hardware complexity for computational primitives as follows : 1 for CP-1 and CP-2, 2 for CP-3, and 4 for CP-4. We based these definitions on the number of multipliers required for each primitive because multi-

Table 3. Throughput for the N×N-th order 2-D IIR filter

	Block Size=1	Block Size=2	Block Size=2
CP-1	$\frac{1}{(3N^2+7N+3)(t_m+t_a)}$	$\frac{2}{(5N^2+15N+8)(t_m+t_a)}$	$\frac{4}{(9N^2+37N+24)(t_m+t_a)}$
CP-2	$\frac{1}{(2N^2+4N+3)(t_m+t_a)}$	$\frac{1}{(2N^2+5N+2)(t_m+t_a)}$	$\frac{1}{(2N^2+7N+4)(t_m+t_a)}$
CP-3	$\frac{1}{(N^2+2N+1)(t_m+2t_a)}$	$\frac{2}{(2N^2+5N+3)(t_m+2t_a)}$	$\frac{2}{(2N^2+7N+5)(t_m+2t_a)}$
CP-4	$\frac{1}{(N^2+2N+1)(t_m+4t_a)}$	$\frac{2}{(N^2+3N+2)(t_m+4t_a)}$	$\frac{2}{(N^2+4N+3)(t_m+4t_a)}$

Table 4. Throughput for the N×N-th order 2-D FIR filter

	Block Size=1	Block Size=2	Block Size=2
CP-1	$\frac{1}{(2N^2+5N+3)(t_m+t_a)}$	$\frac{2}{(3N^2+10N+7)(t_m+t_a)}$	$\frac{4}{(5N^2+23N+18)(t_m+t_a)}$
CP-2	$\frac{1}{(N^2+2N+1)(t_m+t_a)}$	$\frac{2}{(2N^2+5N+3)(t_m+t_a)}$	$\frac{2}{(2N^2+7N+5)(t_m+t_a)}$
CP-3	$\frac{1}{(N^2+2N+1)(t_m+2t_a)}$	$\frac{2}{(N^2+3N+2)(t_m+2t_a)}$	$\frac{2}{(N^2+4N+3)(t_m+2t_a)}$
CP-4	$\frac{1}{(N^2+2N+1)(t_m+4t_a)}$	$\frac{2}{(N^2+3N+2)(t_m+4t_a)}$	$\frac{4}{(N^2+5N+4)(t_m+4t_a)}$

pliers occupy significantly larger silicon area than adders. We can use the hardware complexity and the throughput to compare the efficiency of the implementations. We define the efficiency as the ratio of the throughput and the hardware complexity. When we use twice the hardware, we expect to achieve twice the throughput for the same efficiency. When we consider the hardware complexity for the computational primitive, the most efficient implementation is obtained with the use of scalar processing with a processor based on CP-3 for the 2-D IIR filter and with scalar processing with a processor based on CP-2 for the 2-D FIR filter.

Another important measure of the hardware complexity is the amount of memory required for each processor to implement a given application. The required buffer sizes for the I/O buffer are the same for all implementations according to our implementation scheme that requires one row of data to be stored in this buffer. Also, the required buffer sizes for horizontal state variables are the same for all implementations. That is, each processor requires N(N+1) words for the N ×N-th order filter. The required buffer size for the vertical state variables is L×N×W words where L is the block size and W is the number of pixels per row. The required buffer sizes for the coefficients are different for various implementations. They are shown in table 5 and 6 for the different block sizes and different computational primitives. The total buffer size requirement for each processor is as follows : $(LN+1)\times W+N(N+1)+\alpha$ words where N is the order of a filter, L is the block size, W is the number of pixels per row, and α is the required buffer size for coefficients. As a result, the required buffer sizes for all implementations based upon the four computational primitives are not significantly different. The required buffer size depends on the order of the system and the block size rather than on which computational primitive is used.

Table 5. Coefficient buffer requirements for the N×N-th order 2-D IIR filter

	Block Size=1	Block Size=2	Block Size=2
CP-1	$3N^2+7N+3$	$5N^2+15N+8$	$9N^2+37N+24$
CP-2	$2N^2+4N+1$	$4N^2+10N+4$	$8N^2+28N+16$
CP-3	$2N^2+4N+2$	$4N^2+10N+6$	$8N^2+28N+20$
CP-4	$4N^2+8N+4$	$4N^2+12N+8$	$8N^2+32N+24$

Table 6. Coefficient buffer requirements for the N×N-th order 2-D FIR filter

	Block Size=1	Block Size=2	Block Size=2
CP-1	$2N^2+5N+3$	$3N^2+10N+7$	$5N^2+23N+18$
CP-2	N^2+2N+1	$2N^2+5N+3$	$4N^2+14N+10$
CP-3	$2N^2+4N+2$	$2N^2+6N+4$	$4N^2+16N+12$
CP-4	$4N^2+8N+4$	$4N^2+12N+8$	$4N^2+20N+16$

When we consider these storage requirements,

hardware complexity of the computational primitive, and throughput, we can determine which implementation is the most efficient in terms of silicon area and throughput. The most efficient implementation for the 2-D FIR filter is obtained from the block processing with a block size of four with a processor based upon CP-4 and the most efficient implementation for the 2-D IIR filter is obtained from the block processing with a block size of two with a processor based upon CP-4.

Therefore, we can conclude that the highest efficiency can be obtained from the block processing with a matching block size when the processor, which is based on the computational primitive derived from the largest possible block size, is used.

This conclusion means that the suggested implementation based upon the block state space representation can provide higher efficiency than the implementation presented by Kim and Alexander [1]. Also the processor based upon the computational primitive, which is obtained from the block state space equation, can provide better efficiency in terms of throughput and hardware complexity than many other DSPs those are based on CP-1.

V. CONCLUSION

In this paper, we presented an efficient processor structure for spatial domain 2-D IIR and FIR filters. This structure is based upon the computational primitive that is derived by using the block processing concept. We presented a multiprocessor system with data partitioning [8] to achieve real-time operation. With this system, we can achieve almost linear speedup as we add additional processors until we reach real-time operation.

As we described in this paper, we can achieve high efficiency with a moderate hardware complexity by using the processor based upon the computational primitive derived from the block state space representation with a matching block size. An attractive feature of the approach presented in this paper is that it can be easily extended to higher order system or higher dimensional algorithms.

REFERENCES

1. J. H. Kim and W. E. Alexander, "A multiprocessor architecture for 2-D digital filter," IEEE Trans. Computer, vol. C-36, no. 7, pp. 876-884, 1987.
2. Z. Y. Zhang and W. Steenaart, "High speed architectures for two-dimensional state-space recursive filtering," IEEE Trans. Circuits Syst., vol. CAS-37, no. 6, pp. 831-836, 1990.
3. C. J. Ju and W. E. Alexander, "Block realization of multidimensional IIR digital filters and its finite word effects," IEEE Trans. Circuits Syst., vol. CAS-34, no. 9, pp. 1030-1044, 1987.
4. M. R. Azimi-Sadjadi and A. R. Rostampour, "Parallel and pipeline architectures for 2-D block processing," IEEE Trans. Circuits Syst., vol. CAS-36, no. 3, pp. 443-448, 1989.
5. H. Lu, E. A. Lee, and D. G. Messerschmitt, "Fast recursive filtering with multiple slow processing elements," IEEE Trans. Circuits Syst., vol. CAS-32, no. 11, pp. 1119-1129, 1985.
6. K. K. Parhi and D. G. Messerschmitt, "Block digital filtering via incremental block-state structure," in Proc. IEEE Int. Symp. on Circuits Syst., pp. 645-648, Philadelphia, PA, May 1987.
7. S. M. Park et al., "A novel VLSI architecture for the real-time implementation of 2-D

signal processing systems," Proc. IEEE Int. Conf. Comp. Design : VLSI in computers and Processors, Oct. 1988.

8. J. G. Jeong and J. W. Jang, "A multi-processor implementation of the real time digital filter," KITE Journal of Electronics Engineering, vol. 4, no. 1A, Jul. 1993.

鄭 在 吉(Jae Gil Jeong) 정회원

1980년 2월 : 한양대학교 공대 전자공학과 졸업 (공학사)
1987년 5월 : 미국 North Carolina State University 전기 및 컴퓨터공학과 졸업 (공학석사)
1991년 8월 : 미국 North Carolina State University 전기 및 컴퓨터공학과 졸업 (공학박사)
1979년 12월~1985년 6월 : 국방과학연구소 연구원
1991년 8월~1992년 8월 : 한국전자통신연구소 선임연구원
1992년 9월~현재 : 배재대학교 전자공학과 조교수

張 鍾 煥(Jong Whan Jang) 정회원

1979년 2월 : 한양대학교 공대 전자통신공학과 졸업(공학사)
1986년 5월 : 미국 North Carolina State University 전기 및 컴퓨터공학과 졸업(공학석사)
1990년 12월 : 미국 North Carolina State University 전기 및 컴퓨터공학과 졸업 (공학박사)
1990년 5월~현재 : 배재대학교 정보통신공학과 조교수
1991년 3월~현재 : 배재대학교 전자계산소장