[4] V. J. Mathews, "Adaptive polynomial filters," *IEEE Signal Processing Mag.*, pp. 10–26, July 1991.
[5] E. Karlsson, "Adaptive polynomial lattice filters," in *Proc. ICASSP-91* (Toronto), Apr. 1991.
[6] S. Karaboyas and N. Kalouptsidis, "Efficient adaptive algorithms for ARX identification," *IEEE Trans. Signal Processing*, vol. 39, no. 3, pp. 571–582, Mar. 1991.
[7] G. V. Moustakides and S. Theodoridis, "Fast Newton transversal filters—A new class of adaptive estimation algorithms," *IEEE Trans. Signal Processing*, vol. 39, no. 10, pp. 2184–2193, Oct. 1991.
[8] S. Theodoridis, G. V. Moustakides, and K. Berberidis, "A class of fast adaptive algorithms for multichannel system identification and signal processing," *CTI Tech. Rep. #930936*, Sept. 1993.

An Efficient CORDIC Array Structure for the Implementation of Discrete Cosine Transform

Yu Hen Hu and Zhenyang Wu

***Abstract*—We propose a novel implementation of the discrete cosine transform (DCT) and the inverse DCT (IDCT) algorithms using a CORDIC (COordinate Rotation DIgital Computer)-based systolic processor array structure. First, we reformulate an N-point DCT or IDCT algorithm into a rotation formulation which makes it suitable for CORDIC processor implementation. We then propose to use a pipelined CORDIC processor as the basic building block to construct 1-D and 2-D systolic-type processor arrays to speed up the DCT and IDCT computation. Due to the proposed novel rotation formulation, we achieve 100% processor utilization in both 1-D and 2-D configurations. Furthermore, we show that for the 2-D configurations, the same data processing throughput rate can be maintained as long as the processor array dimensions are increased linearly with N. Neither the algorithm formulation or the array configuration need to be modified. Hence, the proposed parallel architecture is *scalable* to the problem size. These desirable features make this novel implementation compare favorably to previously proposed DCT implementations.**

I. Introduction

In this correspondence, we present an efficient implementation of the discrete cosine transform (DCT) algorithm [1] and its inverse (IDCT) using a CORDIC processor array structure. DCT has been incorporated into image compression standards such as JPEG, MPEG, and CCITT H261. It has also found many applications in speech coding and realization of filter banks for frequency-division and time-division multiplexer (FDM-TDM) systems. Due to its increasing importance, numerous attempts have been made to accelerate the DCT computation in order to facilitate real time, high-throughput implementation [2]. One family of approaches is to derive fast DCT algorithms [7]–[12] by reducing the number of multiplications needed in the formulation. Yet another family of approaches [3]–[6] has focused on using hardware implementation of DCT with parallel or pipelined VLSI array structures [13]. While most of these proposed implementations are based on the multiply-and-add-type arithmetic units, some [5] have reported implementations using a special arithmetic unit called CORDIC.

CORDIC (COordinate Rotation DIgital Computer) is a rotation-based arithmetic algorithm which is particularly efficient for the evaluation of fast transformation algorithms such as DFT (discrete Fourier transform), FFT (fast Fourier transform) [15], and DHT (discrete Hartly transform) [16]. In this correspondence, we will propose new formulations of both the DCT and the IDCT algorithms to facilitate very efficient implementation using CORDIC processor array structures. Compared to the previous result [5], our implementations require only local data communication, have simple, regular array structures, and are linearly scalable.

Manuscript received March 10, 1993; revised May 26, 1994. The associate editor coordinating the review of this paper and approving it for publication was Prof. Daniel Fuhrmann.

Y. H. Hu is with the Department of Electrical and Computer Engineering, University of Wisconsin-Madison, Madison, WI 53706 USA.
Z. Wu is with the Department of Radio Engineering, Southeast University, Nanjing 210018, People's Republic of China.

IEEE Log Number 9406835.

II. Vector Rotation Formulation of DCT and IDCT Algorithm

Given a real-valued sequence $\{x(n);\ 0 \le n \le N-1\}$, the DCT of $\{x(n)\}$ is defined by

$$X(k) = \frac{2}{N} c(k) \sum_{n=0}^{N-1} x(n) \cos\left[\frac{\pi(2n+1)k}{2N}\right] \qquad 0 \le k \le N-1 \tag{1}$$

and the IDCT of an N-point real-valued sequence $\{X(k); 0 \le k \le N-1\}$ is defined by

$$x(n) = \sum_{k=0}^{N-1} c(k) X(k) \cos\left[\frac{\pi(2n+1)k}{2N}\right], \quad 0 \le n \le N-1 \tag{2}$$

where $c(0) = \frac{1}{\sqrt{2}}$, and $c(k) = 1$ for $1 \le k \le N-1$. Since $\frac{2}{N}$ is a scaling factor which can easily be computed if N is a power of 2, we need only to compute $\hat{X}(k) = NX(k)/2$. Let us define

$$V(k) = \sum_{n=0}^{N-1} x(n) \exp\left[j\frac{\pi(2n+1)k}{2N}\right], \quad 0 \le k \le N-1. \tag{3}$$

Clearly, $\hat{X}(k) = \mathrm{Re}\{V(k)\}$ for $k \ge 1$, and $\hat{X}(0) = \frac{1}{\sqrt{2}}V(0)$. Assuming that N is an even number, our strategy is to decompose $V(k)$ such that

$$V(k) = V_e(k) + V_0(k) \tag{4}$$

where

$$V_e(k) = \sum_{n=0}^{N/2-1} x(2n) \exp\left[j\frac{\pi(4n+1)k}{2N}\right] \tag{5}$$

and

$$V_0(k) = \sum_{m=0}^{N/2-1} x(2m+1) \exp\left[j\frac{\pi(4m+3)k}{2N}\right]. \tag{6}$$

The following relations can be verified easily: $\mathrm{Re}\{V_e(N-k)\} = \mathrm{Im}\{V_e(k)\}$ and $\mathrm{Re}\{V_0(N-k)\} = -\mathrm{Im}\{V_0(k)\}$. Substitute $m =$

$N/2 - n - 1$ into (6), we have

$$V_0^*(k) = \sum_{m=0}^{N/2-1} x(2m+1)\exp\left[-j\frac{\pi(4m+3)k}{2N}\right] = \sum_{n=0}^{N/2-1} x(N-2n-1)\exp[-j\pi k]\exp\left[j\frac{\pi(4n+1)k}{2N}\right]. \quad (7)$$

Therefore, we have the following DCT computing algorithm:

$$\tilde{X}(k) + j\tilde{X}(N-k) = V_e(k) + V_0^*(k) \quad 1 \le k \le N/2$$
$$= \sum_{n=0}^{N/2-1} \{x(2n) + (-1)^k x(N-2n-1)\} \times \exp\left[j\frac{\pi(4n+1)k}{2N}\right] \quad (8)$$

$$\tilde{X}(0) = \frac{1}{\sqrt{2}}\sum_{n=0}^{N-1} x(n) = \mathrm{Re}\left\{\sum_{n=0}^{N/2-1}[x(2n) + x(N-2n-1)]\exp[j\pi/4]\right\}. \quad (9)$$

In (8) and (9), the N-point DCT can be obtained using $(N/2)^2 + N/2$ complex number multiplications. In particular, $\tilde{X}(k)$ and $\tilde{X}(N-k)$ are simultaneously computed as the real part and imaginary part of the results.

Similar to the DCT case, for IDCT we can also decompose $x(n)$ into

$$x(n) = \tilde{x}_e(n) + \tilde{x}_0(n) \quad (10)$$

where

$$\tilde{x}_e(n) = \sum_{k=0}^{N/2-1} c(k)X(2k)\cos\left[\frac{\pi(2n+1)k}{N}\right] \quad (11)$$

and

$$\tilde{x}_0(n) = \sum_{k=0}^{N/2-1} X(2k+1)\cos\left[\frac{\pi(2n+1)(2k+1)}{2N}\right]. \quad (12)$$

It is not difficult to verify that

$$x(N-n-1) = \tilde{x}_e(n) - \tilde{x}_o(n). \quad (13)$$

If $N/2$ is also an even number, (11) and (12) can be decomposed as follows:

$$\tilde{x}_e(n) = X(0)\cos[\pi/4] + \sum_{k=1}^{N/4-1} X(2k)\cos\left[\frac{\pi(2n+1)k}{N}\right] + \sum_{k=1}^{N/4-1} X(N-2k)\cos\left[\frac{\pi(2n+1)(N-2k)}{2N}\right] + X(N/2)\cos\left[\frac{\pi(2n+1)}{4}\right]$$
$$= \mathrm{Re}\left\{X(0)\exp\left[\frac{j\pi}{4}\right] + X(N/2)\cos\left[\frac{\pi(2n+1)}{4}\right] + \sum_{k=1}^{N/4-1}[X(2k) - jX(N-2k)] \times \exp\left[(-1)^n j\frac{\pi(2n+1)k}{N}\right]\right\} \quad (14)$$

$$\tilde{x}_0(n) = \sum_{k=0}^{N/4-1} X(2K+1)\cos\left[\frac{\pi(2n+1)(2k+1)}{2N}\right] + \sum_{k=0}^{N/4-1} X(N-2k-1) \times \cos\left[\frac{\pi(2n+1)(N-2k-1)}{2N}\right]$$
$$= \mathrm{Re}\left\{\sum_{k=0}^{N/4-1}[X(2k+1) - jX(N-2k-1)] \times \exp\left[(-1)^n j\frac{\pi(2n+1)(2k+1)}{2N}\right]\right\}. \quad (15)$$

On the other hand, if $N/2$ is an odd number, we decompose (11) and (12) as follows:

$$\tilde{x}_e(n) = \mathrm{Re}\left\{X(0)\cos[\pi/4] + \sum_{k=1}^{(N-2)/4}[X(2k) - jX(N-2k)] \times \exp\left[j(-1)^n\frac{\pi(2n+1)k}{N}\right]\right\} \quad (16)$$

$$\tilde{x}_0(n) = \mathrm{Re}\left\{\sum_{k=0}^{(N-6)/4}[X(2k+1) - jX(N-2k-1)] \times \exp\left[j(-1)^n\frac{\pi(2n+1)(2k+1)}{2N}\right] + X(N/2)\exp\left[j\frac{\pi(2n+1)}{4}\right]\right\}. \quad (17)$$

Substituting the above equations into (10) and (13) yields the inverse DCT algorithm below:

$$x(n) = \mathrm{Re}\left\{X(0)\exp[j\pi/4] + \sum_{k=1}^{N/2-1}[X(k) - jX(N-k)] \times \exp\left[(-1)^n j\frac{\pi(2n+1)k}{2N}\right] + X(N/2)\exp\left[j\frac{\pi(2n+1)}{4}\right]\right\}$$
$$0 \le n \le \frac{N}{2} - 1 \quad (18)$$

$$x(N-n-1) = \mathrm{Re}\left\{X(0)\exp[j\pi/4] + \sum_{k=1}^{N/2-1}(-1)^k[X(k) - jX(N-k)] \times \exp\left[(-1)^n j\frac{\pi(2n+1)k}{2N}\right] + (-1)^{N/2}X(N/2) \times \exp\left[j\frac{\pi(2n+1)}{4}\right]\right\}$$
$$0 \le n \le \frac{N}{2} - 1. \quad (19)$$

Similar to the DCT formulation, (18) and (19) are computed using complex multiplications. Careful comparison of these two equations also reveals that $x(N-n-1)$ can be computed using the same complex multiplication results used to compute $x(n)$. The only difference is in the summation of the final results.

III. CORDIC Array Structure Implementation of DCT and IDCT

Consider the multiplications of two complex numbers $x+jy$ and $r(\cos\theta + j\sin\theta)$. The result, $u+jv$, can be obtained by evaluating the final coordinate after rotating a 2×1 vector $[x \quad y]^t$ through an angle θ and then scaled by a factor r. This is accomplished in CORDIC via a three-phase procedure [14]: angle conversion, vector rotation, and scaling:

CORDIC Implementation of Complex Number Multiplication

```
/* CORDIC angle conversion */
Initialization: $z_0 = \theta$
For $i = 0$ to $b-1$ Do
    $\mu_i = \text{sign}(z_i)$;   /* $\mu_i = 1$ if $z_i > 0$,
    and
    $\mu_i = -1$ if $z_i < 0$. */
        $z_i + 1 = z_i - \mu_i \tan^{-1} 2^{-i}$;   (20)
/* CORDIC vector rotation */
Initialization: $[x_0 \quad y_0]^t = [x \quad y]^t$.
For $i = 0$ to $b-1$ Do
        $x_{i+1} = x_i - \mu_i y_i 2^{-i}$   (21a)
        $y_{i+1} = y_i + \mu_i x_i 2^{-i}$   (21b)
/* Scaling operation */
        $u = \frac{r}{K} x_b; \quad v = \frac{r}{K} y_b$   (22)
```

During the angle conversion phase, the angle θ is represented as the sum of a nonincreasing sequence of elementary rotation angles $\{\tan^{-1} 2^{-i}; 0 \le i \le b-1\}$ such that

$$\theta = \sum_{i=0}^{b-1} \mu_i \tan^{-1} 2^{-i}. \tag{23}$$

In (23), the set of parameters $\mu_i (= \pm 1)$ constitutes an *implicit representation* of θ, and b is the number of bits in the internal register. In DCT and IDCT, θ is known. Hence, angle conversion can be performed in advance. The scaling factor $K = \prod_{i=0}^{b-1} \cos(\mu_i \tan^{-1} 2^{-i})$ will be a constant and independent of μ_i as long as $|\mu_i| = 1$. Hence K can be computed in advance as well. Moreover, in DCT and IDCT, $r = 1$. Thus, r/K will be a known constant. Multiplication by a known constant can be computed very efficiently using *multiplier recoding* [14].

In view of the efficient implementation of the complex multiplication using a CORDIC processor, Wu *et al.* have proposed [5] a CORDIC-based architecture to implement the DCT algorithm. In this previous implementation, $N/2$ CORDIC rotations are computed in parallel with the *broadcasting* of each input $x(n)$ to all $N/2$ CORDIC processors. The intermediate results are routed to $N/2$ set of dual accumulators via a *globally* interconnected exchange network. The output are computed in parallel once all the input $x(n)$ are fed into the network sequentially. This design is a serial-in-parallel-out computing scheme which requires both global synchronization of $N/2$ CORDIC processors and global interconnections to broadcast the input data and to exchange intermediate results. When N becomes large, it may suffer excessive communication overhead.

Based on the new algorithm developed in (8), (9), (18), and (19), we propose to implement the DCT algorithm using different CORDIC processor array structures. Toward this goal, we first transform the proposed rotation algorithms in Section II into a localized recursive formulation in which no global data communication is needed. Let us define $\phi(n,0) = \theta(n,0) = \pi/4$, and for $k \ge 1$

$$\phi(n,k) = \frac{\pi(4n+1)k}{2N}$$

and

$$\theta(n,k) = \frac{\pi(2n+1)k}{2N}. \tag{24}$$

Then the locally recursive DCT and IDCT algorithms can be formulated as follows:

Locally Recursive Formulation of the Vector Rotation DCT Algorithm

```
Initiation: Given $\{x_f(n,0) = x(2n),$
        $x_r(n,0) = x(N-2n-1);$
        $0 \le n \le N/2 - 1\}, V(0,k) = 0.$
for $k = 0$ to $\frac{N}{2}$,
    for $n = 0$ to $\frac{N}{2} - 1$,
        $V(n+1,k) = V(n,k) + (x_f(n,k) - (-1)^k x_r(n,k)) \times K \times \exp[j\phi(n,k)]$   (25)
    end /* the factor $K$ means the CORDIC scaling operation
        is not performed now */
        $x_f(n,k+1) = x_f(n,k); x_r(n,k+1) = x_r(n,k)$
    Output: $X(k) + jX(N-k) = V(\frac{N}{2},k)/K$
        /* scaling is performed at the end */
end
```

Locally Recursive Formulation of Vector Rotation IDCT Algorithm

```
Initiation: $X_f(0,k) = X(k), 0 \le k \le N/2.$
        $X_r(0,k) = X(N-k), 0 \le k < N/2.$
        $X_r(0,N/2) = 0, U(n,0) = V(n,0) = 0.$
for $n = 0$ to $\frac{N}{2} - 1$
    for $k = 0$ to $\frac{N}{2} - 1$
        $W(n,k) = (X_f(n,k) + jX_r(n,k)) \times K \times \exp[(-1)^n \theta(n,k)]$   (26)
        $U(n,k+1) = U(n,k) + \text{Re}\{W(n,k)\}$   (27a)
        $V(n,k+1) = V(n,k) + (-1)^k \times \text{Re}\{W(n,k)\}$   (27b)
    end
        $X_f(n+1,k) = X_f(n,k),$
        $X_r(n+1,k) = X_r(n,k),$
    output: $x(n) = U(n,N/2)/K,$
        $x(N-n-1) = V(n,N/2)/K$
end
```

In the kth iteration, there are three types of operations: complex number multiplications, complex number additions, and data transmission. For convenience, we shall use T_{cm}, T_{ca}, and T_r to represent

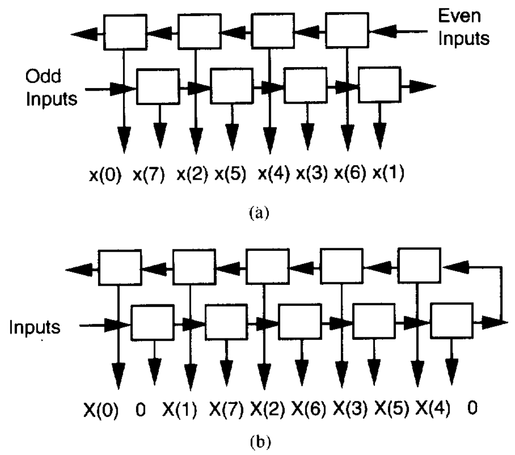
Even Inputs
Odd Inputs
x(0) x(7) x(2) x(5) x(4) x(3) x(6) x(1)
(a)
Inputs
X(0) 0 X(1) X(7) X(2) X(6) X(3) X(5) X(4) 0
(b)

Fig. 1. Data input FIFO buffers for 2-D DCT and IDCT CORDIC array: (a) Data input FIFO buffer for 2-D DCT CORDIC array; (b) data input FIFO buffer for 2-D IDCT CORDIC array.

the time duration needed to complete each of these operations, respectively. Usually $T_{\rm cm} \cong bT_{\rm ca}$ and $T_{\rm ca} \geq T_r$ where b is the number CORDIC iterations, which is also roughly equal to the number of bits in the internal registers. For convenience, we shall assume $T_{\rm ca} = T_r = T$ and $T_{\rm cm} = bT$ in later discussion. We will also assume the time taken to perform the scaling operation (multiplying a known number $1/K$) is rT. Taking advantage of the discrepancies of the computing speed, we can optimize the performance using minimum hardware.

In both the DCT and IDCT cases, the input data order needs to be scrambled as described in the above formulations. However, unlike the shuffling network used in [5], our scheme can easily be realized with two FIFO (first-in-first-out) buffers accepting specified portions of the input sequence at two opposing directions, as depicted in Fig. 1. In the DCT case (Fig. 1(a)), odd and even entries will enter separate FIFO's. In the IDCT case (Fig. 1(b)), the first half of the N-point sequence will enter the first FIFO and the second half will enter the other FIFO. Our scheme thus assumes serial data is first reordered and buffered in the two FIFO's and then are fed to the processor array in parallel. Since there are many N-point DCT's or IDCT's to be performed sequentially in the processor array, the time taken to load the data to these FIFO's can overlap the computation of the DCT or IDCT in the processor array of the preceding set of data. Thus the data input overhead is negligible.

A. 2-D Array Configuration

Depicted in Fig. 2 is a 2-D pipelined CORDIC processor array using $(N/2+1)(N/2)$ processors, which is a direct realization of (25) for an 8-point DCT problem. Each CORDIC operation is performed by a pipelined CORDIC processor array with b cascaded functional units as depicted in Fig. 2(a). Each rectangle contains two adders and a shifter which realize a basic CORDIC iteration for a particular μ_i. This will take T time units each. As a result, even each CORDIC rotation needs bT time units to complete in this processor, b different rotations can be performed concurrently in the same processor in a pipelined fashion. Hence the data processing throughput rate is T time units per CORDIC rotation operation. The data $x_f(n,k)$ and $x_r(n,k)$ are propagated vertically through a single buffer, which also incurs T time units delay per stage. The summation operation is performed by yet another function unit which also takes T time units delay. In Fig. 2(b), the 2-D array configuration along with

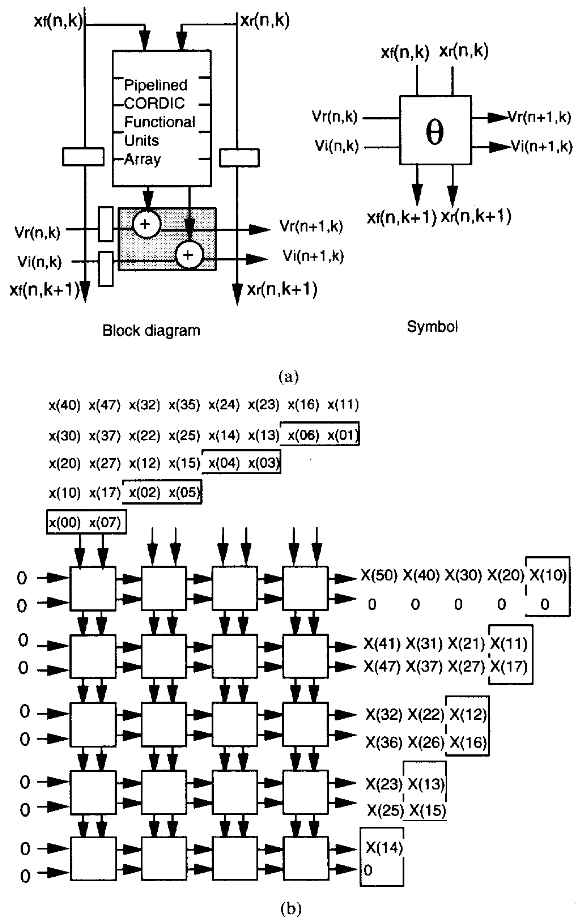

Fig. 2. (a) Function definition of processing element on pipelined CORDIC processor array; (b) 2-D pipelined CORDIC processor array for 8-point DCT. (The input data in the boxes belong to the same set of data. The indices are different from (a). Here, the first index refers to different data sets, the second index is the index within the same data set (8 points). The output scaling unit is not shown here.)

the data input/output patterns are depicted. We assume the data are stored in the input buffers as decried earlier. Hence, successive DCT computations can be executed concurrently in different portions of the same 2-D array. The average throughput rate is T time units per N-point DCT. The latency (time duration between the first input arrives and the first output available) is $(N/2 + b + N/2 + r)T$ time units. In this expression, the first $NT/2$ time units accounts for the input data rearrangement, and bT is the time taken for a CORDIC operation. The second $NT/2$ time units account for the $N/2$ summations, and rT is the time taken to perform scaling at the end. From the above discussion, it is clear that as the size of the DCT N increases, the throughput rate can remain the same as long as the size of this 2-D array ($N/2$ by $(N/2+1)$) is increased accordingly.

In Fig. 3, we show a 2-D pipelined CORDIC processor array for the implementation of an 8-point IDCT. It differs from the 2-D DCT array in several aspects. First, the input data reordering buffer is different (see the above discussion), and the function of the pipelined CORDIC processor is slightly different. With fully pipelined operations, it is not difficult to see that this 2-D array has the same performance in terms of both throughput rate and latency as the 2-D DCT array.

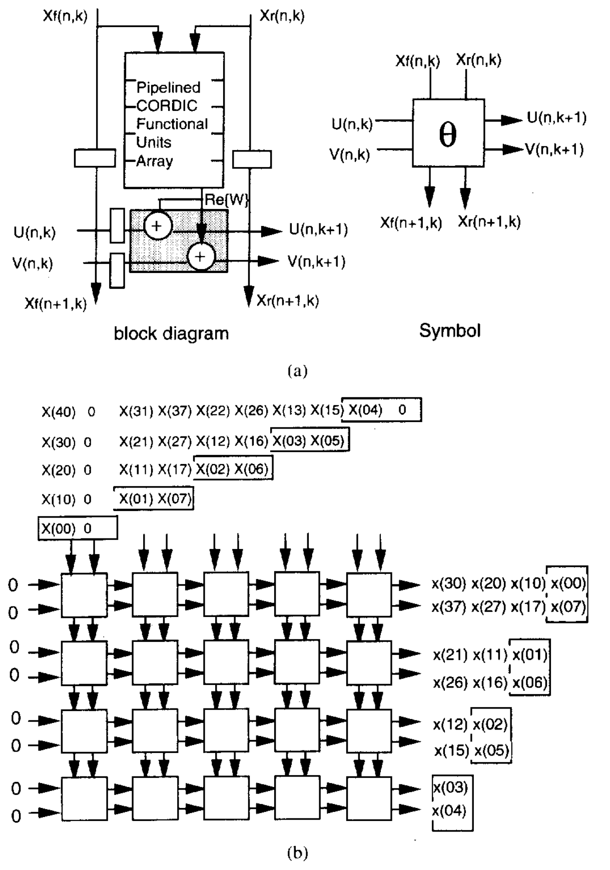

Fig. 3. (a) 2-D pipelined CORDIC processor array for 8-point IDCT; (b) function definition of processing element on pipelined CORDIC processor array. (The input data in the boxes belong to the same set of data. The indices are different from (a). Here, the first index refers to different data sets, the second index is the index within the same data set (8 points). Output scaling processors are not shown.)

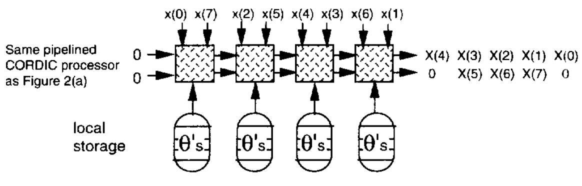
x(0) x(7) x(2) x(5) x(4) x(3) x(6) x(1)
Same pipelined CORDIC processor as Figure 2(a)
X(4) X(3) X(2) X(1) X(0)
0 X(5) X(6) X(7) 0
local storage
θ's

Fig. 4. 1-D pipelined CORDIC processor array for eight-point DCT.

B. 1-D Array Configuration

The 2-D array is a direct implementation of the original data dependency graph. If the data throughput constraint is less demanding, we may devise a 1-D processor array such as depicted in Fig. 4 by projecting the data dependence graph along the vertical direction[1]. In this configuration, an N point DCT (or IDCT) can be computed every $NT/2$ time units using $(N/2+1)$ pipelined CORDIC processors. Successive iterations can share the same array. Hence, 100% processor utilization is accomplished. A 1-D array for IDCT is depicted in Fig. 5.

[1]The systolic array synthesis method of projecting a graph along a specific direction on the index space is described in detail in [13].

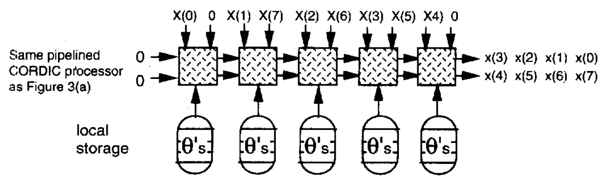
X(0) 0 X(1) X(7) X(2) X(6) X(3) X(5) X4) 0
Same pipelined CORDIC processor as Figure 3(a)
x(3) x(2) x(1) x(0)
x(4) x(5) x(6) x(7)
local storage
θ's

Fig. 5. 1-D pipelined CORDIC processor array for eight-point IDCT.

C. Discussion

1) So far, we have neglected the time needed to execute the scaling operation in the CORDIC algorithm. Fortunately, since both DCT and IDCT involve only complex multiplication and accumulation operations, it is *not* necessary to perform the scaling operation after each complex multiplication. Instead, the scaling operation can be accrued and performed only once at the end. Moreover, in our derivation of the DCT algorithm, we have postponed the multiplication of $2/N$. This factor can be combined with the CORDIC scaling factor $1/K$ so that it will not cost any extra computation overhead. We note that this *delayed-scaling* strategy has previously been proposed for FFT implementation [18].
2) A brief comparison with other existing results: In [3], the DCT is realized using the inverse DFT (IDFT) algorithm followed by multiplication operations. It requires two types of processors. Both [3] and [5] require approximately twice the number of complex multiplications compared to our algorithm. Chang and Wu [6] derived a 1-D systolic processor array in which each PE contains two real multipliers. Although this structure uses only real multiplications, its throughput rate is slower than our 1-D array, and it needs $N-1$ PE's to evaluate an N-point DCT. The structure proposed in [4] requires the size of the DCT, N, to be the product of a set of prime numbers. In our formulation, N needs only to be an even number.

IV. Conclusion

We have presented novel rotation-based formulations for DCT and IDCT algorithms. These new formulations require $\frac{N}{2}(\frac{N}{2}+1)$ complex multiplications, and facilitate efficient CORDIC processors implementation. Both 2-D and 1-D pipelined CORDIC array structures have been presented. The proposed parallel structures consist of a locally-connected module configured as a regular array, and are linearly scalable to handle large value of N.

References

[1] N. Ahmed, T. Natarajan, and K. R. Rao, "Discrete cosine transform," *IEEE Trans. Comput.*, vol. 23, pp. 90–93, Jan. 1974.
[2] K. R. Rao and P. Yip, *Discrete Cosine Transform.* New York: Academic, 1990, pp. 48–82.
[3] M. H. Lee, "On computing 2-D systolic algorithm for discrete cosine transform," *IEEE Trans. Circuits Syst.*, vol. 37, no. 10, pp. 1321–1323, Oct. 1990.
[4] C. Chakrabarti and J. JáJá, "Systolic architecture for the computation of the discrete Hartley and discrete cosine transform based on prime factor decomposition," *IEEE Trans. Comput.*, vol. 39, no. 11, pp. 1359–1368, Nov. 1990.
[5] J.-L. Wu and W.-J. Duh, "A novel concurrent architecture to implement discrete cosine transform based on index partitions," *Int. J. Electron.*, vol. 68, no. 2, pp. 165–174, 1990.
[6] L.-W. Chang and M.-C. Wu, "A unified systolic array for discrete cosine and sine transforms," *IEEE Trans. Signal Processing*, vol. 39, no. 1, pp. 192–194, 1991.
[7] W. H. Chen, C. H. Smith, and S. C. Fralick, "A fast computational algorithm for the discrete cosine transform," *IEEE Trans. Commun.*, vol. 25, pp. 1004–1009, Sept. 1977, .

[8] W. Kou and J. W. Mark, "A new look at DCT-type transforms," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 37, pp. 1899–1908, Dec. 1989

[9] B. G. Lee, "A new algorithm to compute the discrete cosine transform," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP–32, no. 6, pp. 1243–1244, Dec. 1984.

[10] W. Lee, "A new algorithm to compute the DCT and its inverse," *IEEE Trans. Signal Processing*, vol. 39, no. 6, pp. 1305–1313, June 1991.

[11] E. Feig and S. Winograd, "Fast algorithm for discrete cosine trasform," *IEEE Trans. Signal Processing*, vol. 40, no. 9, pp. 2174–219, Sept. 1992.

[12] H. S. Hou, "A fast recursive algorithm for computing the discrete cosine transform," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 35, no. 10, pp. 1455–1461, 1987.

[13] S. Y. Kung, *VLSI Array Processors*. Englewood Cliffs, NJ: Prentice-Hall, 1988.

[14] Y. H. Hu, "CORDIC-based VLSI architecture for digital signal processing," *IEEE Signal Processing Mag.*, vol. 9, no. 3, pp. 16–35, July 1992.

[15] A. M. Despain, "Fourier transform computers using CORDIC iterations," *IEEE Trans. Comput.*, vol. 23, pp. 993–1001, Oct. 1974.

[16] L. W. Chang and S. W. Lee, "Systolic arrays for the discrete Hartly transform," *IEEE Trans. Signal Processing*, vol. 39, no. 11, pp. 2411–2418, Nov. 1991.

[17] E. Deprettere, P. Dewilde, and R. Udo, "Pipelined CORDIC architecture for fast VLSI filtering and array processing," in *IEEE ICASSP*, 1984, 41A, pp. 61–64.

[18] A. M. Despain, "Very fast Fourier transform algorithms hardware for implementation," *IEEE Trans. Comput.*, pp. 333–341, May 1979.

Unequal-Length Multichannel δ_b-Levinson and Schur Type RLS Algorithms

Xiaqi Liu and H. (Howard) Fan

***Abstract*—In this correspondence, δ_b-operator-based unequal-length multichannel Levinson and Schur-type RLS algorithms are developed which have the potential of improved numerical behavior for fast-sampled or ill-conditioned input data. They provide computational improvement over the overparameterization, or the zero padding approach using the existing equal-length multichannel δ_b algorithms when an unequal length multichannel case is considered.**

I. Introduction

Recently, δ-operator-based Levinson and Schur algorithms have been developed which show numerical advantages over the traditional q-operator-based algorithms for fast sampled or ill-conditioned data [1]–[3]. The δ_b-operator[1]-based Levinson and Schur-type RLS algorithms developed in [3] may be used in on-line adaptive signal processing applications. But only equal length multichannel algorithms have been proposed in [3]. They may be used in unequal

Manuscript received July 26, 1993; revised June 15, 1994. This work was supported by U.S. Office of Naval Research under Grant N00014-90-J-1017. The associate editor coordinating the review of this paper and approving it for publication was Dr. James Zeidler.

X. Liu, formerly with the Department of Electrical and Computer Engineering, University of Cincinnati, is now with Adaptive Technology, Inc., Syracuse, NY 13208-2940 USA.

H. Fan is with the Department of Electrical and Computer Engineering, University of Cincinnati, Cincinnati, OH 45221 USA.

IEEE Log Number 9406914.

[1] The backward delta operator δ_b is defined as $\delta_b = (1 - q^{-1})/\Delta$ where q^{-1} is the backward shift operator and Δ is a scaling factor.

TABLE I
PRIMARY MODULE OF MULTICHANNEL δ_b-LEVINSON-TYPE RLS ALGORITHM

$$
\begin{aligned}
\Gamma^e_{\mathbf{m}_i}(t) &= -R^{-1}_{\mathbf{m}_i-i}(t-1)C^T_{\mathbf{m}_i-i}(t)\\
\Gamma^r_{\mathbf{m}_i}(t) &= -E^{-1}_{\mathbf{m}_i-i}(t)C_{\mathbf{m}_i-i}(t)\\
E_{\mathbf{m}_i}(t) &= E_{\mathbf{m}_i-i}(t) + C_{\mathbf{m}_i-i}(t)\Gamma^e_{\mathbf{m}_i}(t)\\
R_{\mathbf{m}_i}(t) &= R_{\mathbf{m}_i-i}(t-1) + C^T_{\mathbf{m}_i-i}(t)\Gamma^r_{\mathbf{m}_i}(t)\\
\boldsymbol{\beta}_{\mathbf{m}_i}(t) &= \mathcal{S}^T_{\mathbf{m}_i}\begin{bmatrix}\boldsymbol{\beta}_{\mathbf{m}_i-i}(t)\\ \mathbf{0}\end{bmatrix} + \left(\mathcal{S}^T_{\mathbf{m}_i}\begin{bmatrix}\boldsymbol{\beta}'_{\mathbf{m}_i-i}(t-1)\\ \mathbf{0}\end{bmatrix}\right.\\
&\qquad \left. - \Delta\mathcal{T}^T_{\mathbf{m}_i}\begin{bmatrix}\mathbf{0}\\ \boldsymbol{\beta}'_{\mathbf{m}_i-i}(t-1)\end{bmatrix}\right)\Gamma^e_{\mathbf{m}_i}(t)\\
\boldsymbol{\beta}'_{\mathbf{m}_i}(t) &= \mathcal{S}^T_{\mathbf{m}_i}\begin{bmatrix}\boldsymbol{\beta}'_{\mathbf{m}_i-i}(t-1)\\ \mathbf{0}\end{bmatrix} - \Delta\mathcal{T}^T_{\mathbf{m}_i}\begin{bmatrix}\mathbf{0}\\ \boldsymbol{\beta}'_{\mathbf{m}_i-i}(t-1)\end{bmatrix}\\
&\qquad -\mathcal{S}^T_{\mathbf{m}_i}\begin{bmatrix}\boldsymbol{\beta}_{\mathbf{m}_i-i}(t)\\ \mathbf{0}\end{bmatrix}\Gamma^r_{\mathbf{m}_i}(t)\\
C_{\mathbf{m}_i}(t) &= X^T_{i,t}Z_{\mathbf{m}_i+i,t-1}\boldsymbol{\beta}'_{\mathbf{m}_i}(t-1)
\end{aligned}
$$

length multichannel cases using the overparameterization or the zero padding approach. However, the unequal length multichannel LS algorithms provide computational efficiency over the zero padding approach [4]. In this correspondence, the equal length multichannel δ_b-Levinson and Schur type RLS algorithms in [3] are extended to more general unequal length multichannel cases. It will cover the situation when the channels have unequal order filters. Levinson and Schur-type RLS algorithms have been developed for this situation based on the traditional q-operator [4]. We now develop a δ_b-operator version. A transformation method similar to that of [3] will be used to transform the q-domain algorithms [4] to the δ_b-domain.

Suppose we have k input channels $x_1(t), x_2(t), \ldots, x_k(t)$ and each channel contains different channel length (order) n_i, $1 \le i \le k$. Then, the multichannel forward and backward linear prediction models are

$$
\begin{aligned}
\mathbf{e}_{k,t} &= [X_{k,t},\quad X_{\mathbf{n}_k,t-1}]\begin{bmatrix} I\\ A_{\mathbf{n}_k}(t)\end{bmatrix}\\
&= X_{\mathbf{n}_k+k,t}\mathcal{T}^T_{\mathbf{n}_k+k}\begin{bmatrix} I\\ A_{\mathbf{n}_k}(t)\end{bmatrix} \qquad (1.1)\\
\mathbf{r}_{k,t} &= [X_{\mathbf{n}_k,t},\quad X_{k,t-\mathbf{n}_k}]\begin{bmatrix} B_{\mathbf{n}_k}(t)\\ I\end{bmatrix}\\
&= X_{\mathbf{n}_k+k,t}\mathcal{S}^T_{\mathbf{n}_k+k}\begin{bmatrix} B_{\mathbf{n}_k}(t)\\ I\end{bmatrix} \qquad (1.2)
\end{aligned}
$$

and the multiindex $\mathbf{n}_k$ *is defined as*

$$
\begin{aligned}
\mathbf{n}_k &= [n_1, n_2, \ldots, n_k]\\
\mathbf{n}_k + k &= [n_1 + 1, n_2 + 1, \ldots, n_k + 1].
\end{aligned}
$$

Here $\mathcal{T}_{\mathbf{n}_k}$ and $\mathcal{S}_{\mathbf{n}_k}$ are the permutation matrices [4], $A_{\mathbf{n}_k}(t)$ and $B_{\mathbf{n}_k}(t)$ are multichannel forward and backward prediction parameter matrices with dimension $(\sum_{i=1}^{k} n_i) \times k$. The notations in (1.1) and (1.2) are defined as

$$
\begin{aligned}
X_{k,t} &= [\mathbf{x}_{1,t}, \ldots, \mathbf{x}_{k,t}]\\
X_{k,t-\mathbf{n}_k} &= [\mathbf{x}_{1,t-n_1}, \ldots, \mathbf{x}_{k,t-n_k}]\\
X_{\mathbf{n}_k,t} &= [X_{n_1,t}, \ldots, X_{n_k,t}]\\
X_{n_i,t} &= [\mathbf{x}_{i,t}, \ldots, \mathbf{x}_{i,t-n_i+1}].
\end{aligned}
$$