Computationally Efficient Systolic Architecture for Computing the Discrete Fourier Transform

J. Greg Nash, *Senior Member, IEEE*

***Abstract*—A new high-performance systolic architecture for calculating the discrete Fourier transform (DFT) is described which is based on two levels of transform factorization. One level uses an index remapping that converts the direct transform into structured sets of arithmetically simple four-point transforms. Another level adds a row/column decomposition of the DFT. The architecture supports transform lengths that are not powers of two or based on products of coprime numbers. Compared to previous systolic implementations, the architecture is computationally more efficient and uses less hardware. It provides low latency as well as high throughput, and can do both one- and two-dimensional DFTs. An automated computer-aided design tool was used to find latency and throughput optimal designs that matched the target field programmable gate array structure and functionality.**

***Index Terms*—Computation, fast algorithms, parallel processing architectures, systolic and wavefront architectures, transforms.**

I. Introduction

THE discrete Fourier transform (DFT) is of central importance to most domains of signal processing: telecommunications (orthogonal frequency division multiplexing for wireless local area networks), radar (synthetic aperture radar, pulse compression, range-Doppler imaging), antenna arrays (frequency domain beamforming), navigation (GPS), acoustics, seismic analysis (nuclear event detection), speech processing (spectrograms), biomedical signal analysis (spectral analysis), multimedia, image processing (X-rays, magnetic resonance imaging, ultrasound), spectroscopy, and sonar (LOFARgram) [1], [2].

Since many of these applications are either "real-time" or involve very large data sets, it is not surprising that special purpose parallel circuitry for computing the DFT has been intensively studied. Most such circuit implementations to date have been based on the fast Fourier transform (FFT), using either a fixed radix $2^s \geq 2$ or variations with mixed or split radices [3]–[6]. A common characteristic of these circuits is that the number of samples N must be a power of two, which limits the number of reachable values of N and their spacing uniformity. However, this limitation on N is not always a natural choice for the application at hand. For example, a recently proposed HDTV transmission standard is based on a number of subcarriers (3780) that is not a power of two [7]. In this case, a 3780-point FFT circuit was developed that provided better overall system performance even though it was slower and used 33% more memory than a 4096 split-radix pipelined FFT circuit. Thus, there is a strong rationale for developing an efficient, high performance, parameterized (scalable) circuit that is suitable for applications in need of DFT sizes that are not necessarily a power of two.

In Section II, previous work related to relevant parallel computation of the FFT is summarized with particular reference to systolic implementations. In Section III, a mathematical framework that decomposes the one-dimensional (1-D) DFT into four-point transforms is developed and leads to a new matrix-based representation of the 1-D DFT. Section IV provides a description and analysis of a new direct "base-4" systolic implementation of the 1-D DFT based on the matrix expression derived in Section III. In Section V, a block row/column factorization technique is presented that can be used to increase computationally efficiency while avoiding the need for a separate transposition step. Throughput and latency are discussed in Sections VI and VII, which analyze computational efficiency and compare the base-4 circuit to other systolic and FFT implementations.

II. Related Work

A variety of systolic array designs have been proposed for the direct computation of the DFT. Linear arrays have been based on matrix-vector multiplication [8], [9], Horner's rule [10]–[13], RNS arithmetic [14], and four-point DFTs [15]. These designs offer architectural simplicity and will typically work for any N, but because they are based on a direct algorithm implementation of the DFT for which the number of arithmetic operations per DFT is $O(N^2)$, they are computationally less efficient for longer transforms and not directly suited to performing two-dimensional (2-D) DFTs.

Alternatively, a 2-D systolic array of size $N_1 \times N_2$ can more efficiently compute a 1-D DFT with the limitation on N that it be expressed as two cofactors, $N = N_1 N_2$. In this case the well-known row/column method can be used to do the transform: N_1 DFTs of length N_2, then an N-element twiddle factor multiplication (the twiddle multiplication can be avoided if N_1 and N_2 are coprime), and finally N_2 transforms of length N_1. The row/column transforms for this architecture are computed directly, each in $O(N_i)$ time. In this case the number of arithmetic operations per DFT is reduced from that above to $O(N(N_1 + N_2))$ [16]. These designs have been based on efficient use of properly interconnected sets of 1-D arrays [17] and index transformations leading to triple matrix products [18]–[23]. While these 2-D systolic designs are fast and well suited to VLSI implementation, hardware requirements can be prohibitive due to

Manuscript received July 15, 2004; revised November 21, 2004. This work was supported in part by the Defense Advanced Projects Research Agency under Contracts DAAH01-96-C-R135 and DAAH01-97-C-R107. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Sergios Theodoridis.

The author is with Centar, Los Angeles, CA 90077 USA (e-mail: jgregnash@centar.net).

Digital Object Identifier 10.1109/TSP.2005.859216

the need for having one processing element (PE) per transform point. Although "partitioning" strategies exist [8], by the time the number of multipliers are reduced sufficiently, designs either no longer possess useful systolic attributes or a complex and large memory structure is needed for buffering.

Computing efficiency can be further improved if N is highly composite, i.e., $N = N_1 N_2 \ldots N_q$. If these factors are coprime, several architectures based on the prime factor algorithm (PFA) [24] have been proposed [7], [15], [25], [26], each consisting of a cascade of "small-N" Winograd FFT modules with commutators in between stages. One such PFA design shows that computational efficiency is comparable to a Cooley–Tukey type split-radix design, although latency and memory usage are not as good [7]. One important disadvantage to such PFA implementations is the complexity and irregularity of the designs, in that they require use of several different small FFT modules, indexing circuitry is in general more complicated, and there can be limitations on handling errors due to finite register lengths [7]. Also, use of the PFA alone specifically excludes transforms that are a power of two.

Compared to the architectures above, the base-4 algorithm has the same multiplicative complexity as the 2-D systolic arrays, but the "constant factor" is $\times 10$ less and its circuit implementation uses at least $\times 64$ fewer complex multipliers. It is also much more regular and simple than the PFA-based architectures. It can compute the DFT for any N-point sequence divisible by 256, so that more points are reachable compared to a power of two algorithm and these points are uniformly spaced. Additionally, it can be used to compute 2-D DFTs as well, and any base-4 implementation can be programmed to compute any allowed DFT size as long as memory resources are adequate.

III. Base-4 DFT Matrix Expression

The observation that use of a decimation in frequency and time leads to a DFT coefficient matrix consisting of a regular array of 4×4 matrices has been made before in the context of building an efficient 64-point pipelined FFT [27]. However, that analysis was restricted to producing matrix expressions for only two transform sizes ($N = 16$ and $N = 64$) and its recursive form best suits a pipelined FFT architecture. The analysis here makes use of this same coefficient matrix regularity, but from the point of view of building systolic arrays. This leads to a matrix expression applicable to any valid value of N and one that shows explicitly the role of 4-point transforms. (An alternative derivation is also possible based on bit-reversing permutation matrices [28].)

The DFT is defined as

$$Z(k) = \sum_{n=0}^{N-1} W_N^{nk}\, X(n) \tag{1}$$

where $X(n)$ are the time-domain input values, $Z(k)$ are the frequency-domain outputs, and $W_N = e^{-j(2\pi/N)}$. In matrix terms (1) can be represented as

$$Z = CX \tag{2}$$

where C is a coefficient matrix containing elements $W_N^{nk} = e^{-j(2\pi nk/N)}$. If N can be factored as $N = N_1 N_2$, then using the reindexings $n = n_1 + N_1 n_2$ and $k = k_1 + N_1 k_2$ with $n_1 = 0, 1, \ldots, N_1 - 1$, $k_1 = 0, 1, \ldots, N_1 - 1$, $n_2 = 0, 1, \ldots, N_2 - 1$, $k_2 = 0, 1, \ldots, N_2 - 1$, (1) becomes

$$\begin{aligned} &Z(k_1 + N_1 k_2) \\ &= \sum_{n_1=0}^{N_1-1} \left(W_N^{n_1 k_1} \sum_{n_2=0}^{N_2-1} W_{N_2}^{n_2 k_1}\, W_{N_2}^{n_2 k_2 N_1}\, X(n_1 + N_1 n_2) \right) W_{N_2}^{n_1 k_2}. \end{aligned} \tag{3}$$

This expression can be considerably simplified by imposing the restriction that N_1/N_2 be an integer value so that $W_{N_2}^{n_2 k_2 N_1} = e^{-j(2\pi n_2 k_2 N_1/N_2)} = 1$. For any particular value of n_1, k_1, the value of the inner parenthesis in (3) $Y(k_1, n_1)$ can be evaluated from the dot product

$$\begin{aligned} Y(k_1, n_1) = W_N^{n_1 k_1} \begin{bmatrix} W_{N_2}^0 & W_{N_2}^{k_1} & W_{N_2}^{2k_1} \cdots W_{N_2}^{(N_2-1)k_1} \end{bmatrix} \\ \times \begin{bmatrix} X(n_1) \\ X(n_1 + N_1) \\ X(n_1 + 2N_1) \\ \cdot \\ \cdot \\ \cdot \\ X(n_1 + (N_2 - 1)N_1) \end{bmatrix} \end{aligned}$$

so that (3) becomes

$$Z(k_1 + N_1 k_2) = \sum_{n_1=0}^{N_1-1} Y(k_1, n_1)\, W_{N_2}^{n_1 k_2}. \tag{4}$$

All the N_1^2 dot product values $Y(k_1, n_1)$ can be collected in the $N_1 \times N_1$ matrix Y_b by performing the matrix multiplication

$$Y_b = W_M \bullet C_{M1} X_b \tag{5}$$

where W_M is an $N_1 \times N_1$ matrix with elements $W_M[k_1, n_1] = W_N^{n_1 k_1}$, C_{M1} is an $N_1 \times N_2$ coefficient matrix with elements $C_{M1}[k_1, n_2] = W_{N_2}^{n_2 k_1}$, X_b is an $N_2 \times N_1$ matrix with elements $X_b[n_2, n_1] = X(n_1 + N_1 n_2)$, Y_b is an $N_1 \times N_1$ matrix with elements $Y_b[k_1, n_1] = Y(k_1, n_1)$, and "$\bullet$" means element-by-element multiply.

In a similar way for a particular k_1, k_2, the corresponding $Z(k_1 + N_1 k_2)$ can be calculated from the dot product

$$\begin{aligned} &Z(k_1 + N_1 k_2) \\ &= \begin{bmatrix} W_{N_2}^0 & W_{N_2}^{k_2} & W_{N_2}^{2k_2} & \cdots & W_{N_2}^{(N_1-1)k_2} \end{bmatrix} \begin{bmatrix} Y(k_1, 0) \\ Y(k_1, 1) \\ Y(k_1, 2) \\ \cdot \\ \cdot \\ Y(k_1, N_1 - 1) \end{bmatrix} \end{aligned} \tag{6}$$

and by collecting the dot products as before, a matrix expression for calculating Z is obtained as

$$\begin{aligned} Y_b &= W_M \bullet C_{M1}\, X_b \\ Z_b &= C_{M2}\, Y_b^t \end{aligned} \tag{7}$$

where C_{M2} is an $N_2 \times N_1$ coefficient matrix with elements $C_{M2}[k_2, n_1] = W_{N_2}^{n_1 k_2}$ and Z_b is an $N_2 \times N_1$ matrix containing the transform outputs $Z_b[k_2, k_1] = Z(k_1 + k_2 N_1)$.

The character of (7) is determined largely by the value of N_2 or the "base" b ($b = N_2$) because this sets the reachable values of N and the structure of the coefficient matrices C_{M1} and C_{M2}. In (7), C_{M1} and C_{M2} contain N/b^2 submatrices $C_B = [c_1|c_2|\ldots|c_b]^t$ with the form $C_{M1} = [C_B^t|C_B^t|\ldots]^t$ and $C_{M2} = [C_B|C_B|\ldots]$ due to the periodicity of W_{N_2}. Also, values of N are constrained to be integer multiples of b^2, since it was assumed in (3) that N_1/N_2 is an integer. Although the choice of b is application dependent, here only "base-4" ($b = 4$) designs are considered because this choice provides good architectures that are arithmetically efficient. This selection results in

$$c_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix},\ c_2 = \begin{bmatrix} 1 \\ -j \\ -1 \\ j \end{bmatrix},\ c_3 = \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \end{bmatrix},\ c_4 = \begin{bmatrix} 1 \\ j \\ -1 \\ -j \end{bmatrix}$$

and

$$C_B = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \tag{8}$$

where C_B in (8) is the coefficient matrix for a four-point DFT and also describes a radix-4 decimation in time butterfly. In (7), Y_b can be seen as resulting from a series of 4-point transforms of a bit-reversed input X followed by a twiddle multiplication and Z_b is obtained from summations of the results of 4-point transforms of Y_b^t. (Even though $b = 4$, the symbol b will be used throughout to emphasize its architectural origin and to avoid confusion with regular constants.)

By comparing (7) with (2), the computational advantages of the base-4 form are readily evident. In (7), the matrix products $C_{M1}X$ and $C_{M2}Y^t$ involve only addition/subtraction because the elements of C_{M1} and C_{M2} contain only ± 1 or $\pm j$, whereas the product CX in (2) requires complex multiplications. Also, the size of the coefficient matrix W_M in (7) is $(N/b) \times (N/b)$ versus the $N \times N$ size of C in (2); consequently, the number of overall direct multiplications in (7) is reduced by b^2 compared to (2). Note that for systolic implementations, a distribution of the elements $C_{M1}[i, j]$ and $C_{M2}[i, j]$ does not impose significant bandwidth requirements because full complex numbers are not used.

IV. Direct Form DFT Architecture

This section describes two direct form systolic arrays suitable for calculating a single DFT based on (7). Since there are a many systolic designs that can be derived from (7), a mapping tool, Symbolic Parallel Algorithm Development Environment (SPADE), was used to find optimal designs [29]. The main constraint imposed in making a design choice was that the systolic array architecture matches that of recent generations of field-programmable gate arrays (FPGA). FPGAs have uniformly distributed logic, memory, and routing resources so that circuit performance, area, and power dissipation depend critically on obtaining a good match [31], [32]. More details on the mapping exercise and use of constraints within SPADE can be found in [28] and [30].

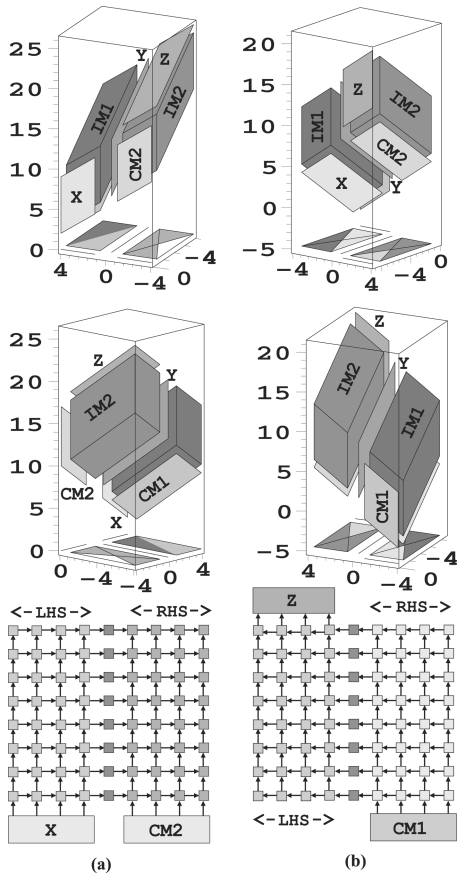


Fig. 1. Two different space-time views of each of two systolic architectures are shown with their corresponding PE arrays beneath ($N = 32$). The variable structures in the space-time view are also shown projected to a constant time plane. Each variable is labeled according to the input code (9) so that its position can be seen explicitly. The variable WM has been mapped to the same locations as Y and is not shown. The variables "IM1" and "IM2" are intermediate variables created in SPADE to perform running sums associated with the "add" function in (9). Each PE array consists of a left-hand-side (LHS) and right-hand-side (RHS) $N/b \times b$ PE adder array with an N/b linear array of multipliers in between.

SPADE found only two unique FPGA-compatible systolic arrays that are throughput and latency optimal, shown in Fig. 1(a) and (b). The array structures, variable names, and data movement associated with Fig. 1 are best understood by rewriting (7) using SPADE's input representation

```
for j to N/b do
  for k to N/b do
    Y[j,k]:=WM[j,k]* add
      (CM1 [j,i]*X[i,k],i=1..b);
  od;
  for k to b do
    Z[k,j]:=add(CM2[k,i]*Y[j,i],i=1..N/b);
  od
od;
```

where the "add" function is a summation over the index i and the subscript "b" has been removed. SPADE creates two new variables from (9), IM1[i, j, k] = CM1[j, i] ∗ X[i, k] and

TABLE I
TRANSFORMATIONS FOR TWO SYSTOLIC ARRAY DESIGNS SHOWN IN FIG. 1

Name	Fig. 1a		Fig. 1b	
	T	t	T	t
X	$\begin{bmatrix} 1 & 1 \\ -1 & 0 \\ 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ b+1 \\ 0 \end{bmatrix}$	$\begin{bmatrix} -1 & 1 \\ -1 & 0 \\ 0 & -1 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$
CM1	$\begin{bmatrix} 1 & 1 \\ 0 & -1 \\ -1 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ b+1 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 1 & -1 \\ 0 & -1 \\ 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$
IM1	$\begin{bmatrix} 1 & 1 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ b+1 \\ 0 \end{bmatrix}$	$\begin{bmatrix} -1 & 1 & 1 \\ -1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$
Y	$\begin{bmatrix} 1 & 1 \\ 0 & 0 \\ -1 & 0 \end{bmatrix}$	$\begin{bmatrix} b+1 \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 \\ 0 & 0 \\ 0 & -1 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$
IM2	$\begin{bmatrix} 1 & 1 & -1 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix}$	$\begin{bmatrix} 2b+2 \\ -b-1 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \\ -1 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$
CM2	$\begin{bmatrix} -1 & 1 \\ 1 & 0 \\ 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 2b+2 \\ -b-1 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & -1 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$
Z	$\begin{bmatrix} -1 & 1 \\ 1 & 0 \\ 0 & -1 \end{bmatrix}$	$\begin{bmatrix} 4b+3 \\ -b-1 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 2b+1 \\ 0 \\ -2b-1 \end{bmatrix}$

$\mathrm{IM2[i,j,k]} = \mathrm{CM2[k,i]} * \mathrm{Y[j,i]}$, which facilitate performing the addition using a running sum method. The SPADE simulator was used to verify correct systolic operation of each solution.

For each algorithm variable x in (9), SPADE finds an affine transformation $T(x)$ such that $T(x) = T_x(A_x(I)) + t_x$, where T_x is a matrix, t_x is a vector, and $A_x(I)$ is the affine indexing function for algorithm variable x and index set I. Here $T()$ represents a "mapping" from one index set $\{i, j, k\}$ to another index set $\{t, x, y\}$, where the latter index set includes one index representing time with the remaining indexes representing spatial coordinates. These transformation matrices T, t for each algorithm variable in (9) are shown in Table I for the designs in Fig. 1. For example, algorithm variable CM2 in (9) has an indexing function

$$A_{\mathrm{CM2}}(I) = A_{\mathrm{CM2}} I + a_{\mathrm{CM2}} : A_{\mathrm{CM2}} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

$$a_{\mathrm{CM2}} = \begin{bmatrix} 0 \\ 0 \end{bmatrix},\ I = \begin{bmatrix} i \\ j \\ k \end{bmatrix}$$

and therefore the space-time transformation $T(CM2)$ for the systolic array in Fig. 1(a) becomes

$$T(\mathrm{CM2}) = \begin{bmatrix} -1 & 1 \\ 1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} i \\ j \\ k \end{bmatrix} + \begin{bmatrix} -1 & 1 \\ 1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 2b+2 \\ -b-1 \\ 0 \end{bmatrix} = \begin{bmatrix} i-k+2b+2 \\ k-b-1 \\ 0 \end{bmatrix}.$$

TABLE II
VECTORS [time, x, y]^t FOR EACH DEPENDENCY IN SOLUTIONS FIG. 1(A) AND (B)

Solution	Y → IM1	IM1 → CM1	IM1 → X
Fig. 1a	[1 −1 0]	[1 0 0]	[1 0 −1]
Fig. 1b	[1 1 0]	[1 0 −1]	[1 0 0]
Solution	**IM2 → CM2**	**IM2 → Y**	**Z → IM2**
Fig. 1a	[1 0 −1]	[1 −1 0]	[1 0 0]
Fig. 1b	[1 0 0]	[1 1 0]	[1 0 −1]

Computations get mapped to the space-time location of the result variable in a statement. The projection of the space-time structure along the time axis produces a systolic array after a PE is associated with each grid coordinate [Fig. 1 (bottom)]. In addition to the outputs T, t for each of the algorithm variables, SPADE computes a set of vectors indicating the direction of data flow between PEs for each dependency in the algorithm (9), and these are shown in Table II. The terminology "a → b" in Table II means that b is an argument used to compute a. The data flow unit vector is $[time, x, y]^t$, where each unit of time corresponds to a PE computation cycle. Data flow directions are shown only for the PE arrays [Fig. 1 (bottom)]. Note that the data flow for some dependencies indicates no spatial (x/y) movement; this means that that the argument and result variable remain in the same PE. The combined set of transformations, data flow vectors, index ranges, and PE computations completely specify the operation and signal flow graph of the systolic array. The space-time algorithm structure is best viewed as it is rotated using a three-dimensional real-time rendering program found in many graphics programs. This visualization, along with knowledge of data flow vectors, is a good alternative to the usual systolic descriptions in terms of data movement snapshots on a reduced sized problem.

Each of the two array designs is composed of a linear array of N/b multipliers with a $N/b \times b$ mesh array of adder/subtractors on each side.[1] Consequently, the array scales in only one direction with the transform size N. (In order to avoid confusion with true 2-D systolic arrays that scale in both directions, it will be referred to as a "linear" array in the remaining text.) This structure matches directly that of FPGAs with their embedded linear arrays of hardwired multipliers and mesh connected arrays of logic cells on both sides, each cell being composed of logic, a register, and fast adder circuitry; consequently, this makes FPGA implementations area and routing efficient. Such an architecture/hardware match is particularly important because interconnections are the biggest contributor to power consumption and delay [31], [32] in this technology. The multipliers generate the products involving WM and the adder/subtractors perform the CM1 X and CM2 Y^t products in (9). The main difference between the two designs is that in the design of Fig. 1(a), the input X is in a memory at the edge of the array and the transform result Z resides internally after processing, whereas for the design of Fig. 1(b), the input data X is internal and the output Z is collected in a memory at the edge of the array.

The corresponding functional operation of each PE is depicted in Fig. 2. As can be seen, for each design there are three

[1]Multipliers, adder/subtractors, memories, and registers operate on complex data.

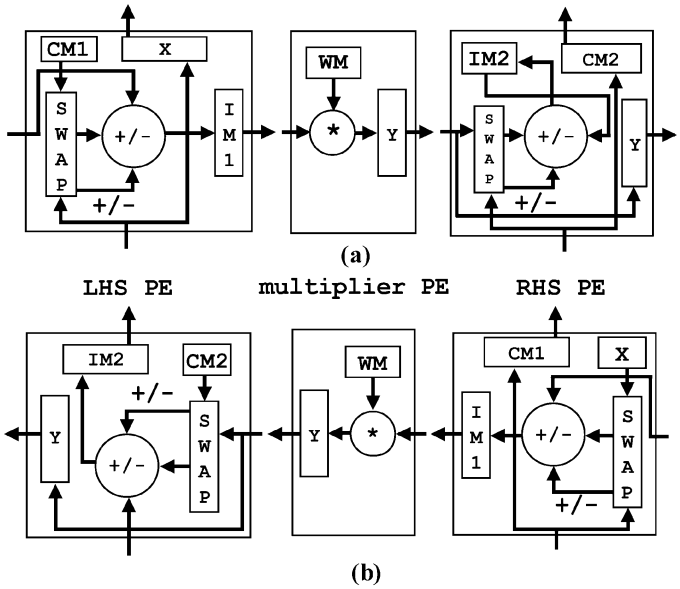


Fig. 2. PE functional operation for design (a) in Fig. 1(a) and (b) in Fig. 1(b). The "SWAP" logic block is used to switch real and imaginary parts of X when multiplying by j and to set the PE for addition or subtraction.

different PE types, one for the LHS adder/subtractor array, one for the multiplier array, and one for the RHS adder/subtractor array. Only the multiplier PE has an amount of memory that depends on the transform size; each multiplier PE stores one row of the matrix WM. All LHS and RHS PEs perform an addition or subtraction operation and contain nominally two registers. If a PE input does not come from an adjacent PE output or a memory structure at the array edge, then that input value is zero. In both LHS PEs there is a small register that stores one of the values $\{\pm 1, \pm j\}$ for CM1 [solution Fig. 1(a)] or CM2 [solution Fig. 1(b)]. In both RHS PEs the coefficient values CM1 and CM2 propagate from PE to PE. The PE function "SWAP" examines the matrix input from either CM1 or CM2, exchanges the real and imaginary parts of the other input to the swap function as necessary, and finally sends this result along with a control signal "$+/-$" to the adder/subtractor. The variable IM1 accumulates the products CM1 [j, i] X [i, k] and Y [j, k] becomes the value in (9) of IM1 when it arrives at the multiplier PE. The variable IM2 accumulates the products CM2 [k, i] Y [j, i] and becomes the value Z [k, j] in (9) at the end of processing. In both the LHS and RHS arrays, the matrix elements of X and Y propagate unchanged from PE to PE.

The designs in Fig. 1 each have a latency of $2N/b + b + 4$ cycles and a throughput of $N/b+1$ cycles per DFT. The assignment of the same cycle time to PEs that do addition/subtraction and to PEs that do multiplication is consistent because in the FPGA target hardware the multipliers are hardwired. Consequently, their speed approximately matches that of the configurable logic used to perform addition/subtraction.

V. Row/Column Factorization

A. Introduction

Although the designs in Fig. 1(a) and (b) possess arithmetic and architectural efficiencies compared to other linear systolic arrays, they are still direct DFT implementations because the number of computations is $O(N^2)$. To reduce the overall number of computations, another factorization $N = N_3 N_4$ is applied to the computation using the traditional row/column approach. This factorization requires computation of two sets of DFTs, N_4 transforms of length N_3 (referred to as "column" transforms), and N_3 transforms of length N_4 (referred to as "row" transforms). Each set of transforms is computed using the direct base-4 architecture described in Section IV. In between column and row transforms, it is necessary to multiply each of the N points by a corresponding twiddle factor $W_N^{i,k}$, $i = 0, 1, \ldots, N_3 - 1$, $k = 0, 1, \ldots N_4 - 1$, using the multiplier PE of Fig. 2 [22]. (Without the twiddle multiplication, a 2-D DFT is performed.) The approach followed here will be to use just one systolic array to do the column transforms, the row transforms, and the twiddle multiplications. This constraint leads to several architectural options, two of which are described below.

B. Combined Architecture for Square Factorizations ($N_3 = N_4$)

Both the systolic designs from Fig. 1 are necessary for this factorization: that in Fig. 1(a) for the column transforms and that in Fig. 1(b) for the row transforms. In this way the column stage inputs X and row stage outputs Z are appropriately external to the array to facilitate I/O. Also, the column stage PE outputs are internal just as are the row stage inputs as seen in Fig. 1(a) and (b), a mapping which permits the row stage to immediately follow the column stage without requiring any extra internal data shuffling. The main architectural addition to a combined array would be a problem size dependent amount of internal memory with which to store the column stage DFT results and coefficient storage in each multiplier PE for the twiddle coefficients. The positioning and sequencing of the DFTs are best understood from the space-time view shown in Fig. 3 for a computation with $N_3 = N_4 = 64$. In this example, each of the two stages of the computation computes 64 successive DFTs, although only two DFT iterations are shown for each of the two stages. Here it can be seen that successive column stage and row stage DFT iterations can be directly and efficiently "stacked" on top of each other in time to geometrically fill the space-time computing volume.

A complete architecture that includes column, twiddle multiplication, and row stage operations will then be a combination of the two designs in Fig. 1. Because the array structure for both designs is the same and each PE requires very similar functionality, only a small amount of additional circuitry for PE data routing and control is necessary. However, it is important that overall data organization and flow for the combined architectures be consistent. For example, the value of an element of CM1 inside a particular PE in Fig. 1(a) must be the same as the element of CM2 in the corresponding PE in Fig. 1(b). Although the two architectures in Fig. 1 are unique, SPADE found 16 different variations of each, equivalent to ways of using the same architecture to do the same computation but with different orderings and placement of the variables involved. The two particular choices of transformations and dependence vectors in Tables I and II were chosen to provide the appropriate "match" of data

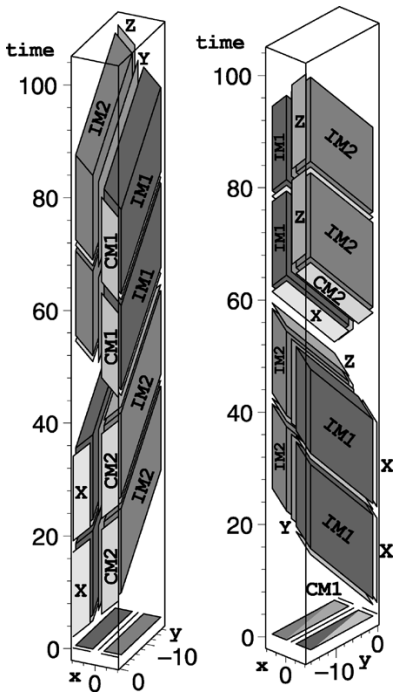


Fig. 3. SPADE generated space-time view of successively "stacked" DFT iterations for performing row/column factorization, where $N = 4096$ and $N_3 = N_4 = 64$. Only two of 64 DFT iterations each for column transforms ("bottom" two iterations) and row transforms ("top" two iterations) are shown. (The twiddle multiplications by $W_N^{i,k}, i = 0, 1 \ldots N_3 - 1, k = 0, 1 \ldots N_4 - 1$ are not shown, as SPADE does not generate these.)

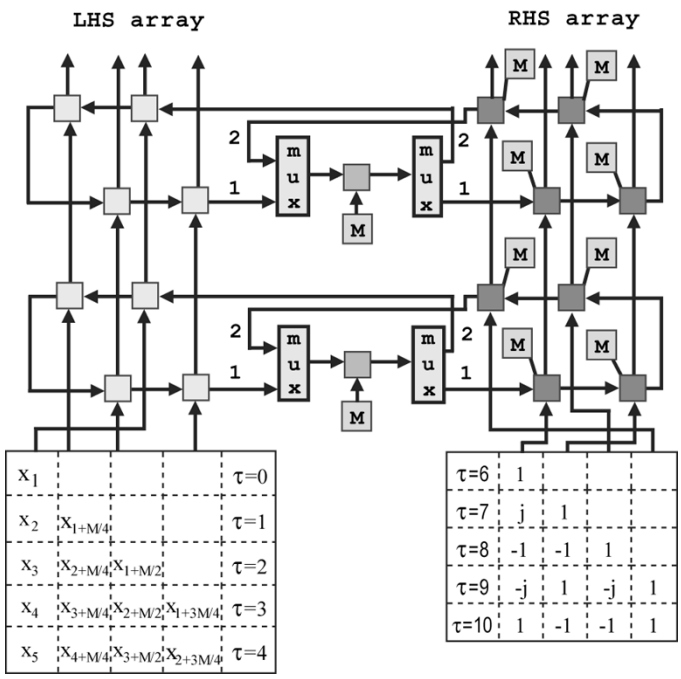


Fig. 4. Architecture created by combination of array and interconnection elements from Fig. 1(a) and (b). Only the bottom two PE rows of the LHS and RHS arrays are shown. Memory elements are labeled "M." Array inputs X at different times τ are shown for a column DFT of length $M = N_3$, along with corresponding inputs for CM2.

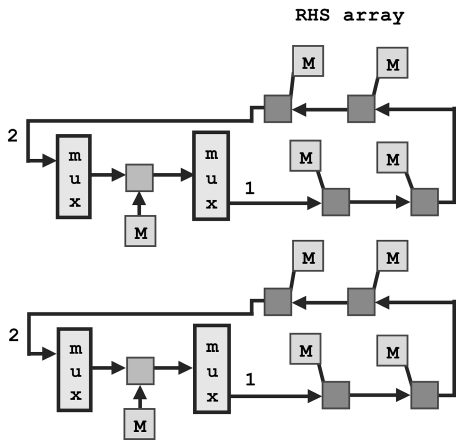


Fig. 5. Data path from Fig. 4 used during the twiddle updates in between column and row DFTs. Here, the LHS array multiplexer uses path "2," while that on the RHS uses path "1." (Only the bottom two PE rows of the LHS and RHS arrays are shown.)

structures in order that the combined design places data in consistent PE locations.

The biggest issue in combining the two architectures of Fig. 1 is that of accommodating the need for bidirectional connections between PEs in the "east-west" direction. This is especially true for FPGA target hardware, which in general provides little support for such structures. An alternative approach based on a modified toroidal structure allows one way east-west PE connections and still maintains locality as shown in Fig. 4. This figure shows that the PEs at each end of both the LHS and RHS arrays of Fig. 1(a) and (b) are directly connected to the multiplier PE through two multiplexers. The path labeled "1" performs the east-west connections for the Fig. 1(a) design and the path labeled "2" does the same for the Fig. 1(b) design. In this combined design, columns of RHS/LHS PEs that are furthest from the multiplier PE in Fig. 1(a) become logically and physically adjacent the multiplier PE in the design Fig. 1(b).

In Fig. 4, the memory elements that are necessary to store the values of the column transforms and twiddle multiplications are shown explicitly. The aggregate memory required in the RHS array is the number of transform points N with each of the N_3 RHS PEs containing room for $N/N_3 = N_4$ elements. For the multiplier PE, the attached memory needs to accommodate WM and the N twiddle coefficients $W_N^{i,k}$, $i = 0, 1 \ldots N_3 - 1$, $k = 0, 1 \ldots N_4 - 1$. Because these PE memories are uniformly distributed, they can be implemented using FPGA embedded memories and their relatively small size makes them inherently fast and low power.

The data path within the combined architecture of Fig. 4 that is used to do the twiddle multiplication in between the row and column transforms is shown in Fig. 5. Here it can be seen that only the RHS array is used to do this update. It is natural for the multiplier PE to be constructed as a b stage pipeline, matching the b pipeline stages of the RHS array PEs. In this way every four cycles a new set of four twiddle-updated coefficients are written while a new set of coefficients to be updated are read (assuming memories in the RHS array are dual ported). The total computation time for this twiddle step using the N_3/b multiplier PEs is then $N_3 N_4/(N_3/b) + b = b(N_4 + 1)$ cycles.

C. Matrix Transpose

In the above discussion, it has been assumed that each row DFT input X is available in the correct format prior to the row

stage DFTs. However, after the sequence of column DFTs, the outputs [Z in RHS array of Fig. 1(a)] are not suitably formatted to begin the row DFTs. This problem is illustrated in Fig. 6(a), which shows that the position in the RHS array [Fig. 1(a)] of transform coefficient outputs Z for a sequence of 32 32-point column transforms is the same for all 32 DFTs. Therefore, each PE stores the same coefficient number for each of the DFTs. However, each row DFT requires the same coefficient number from each of the column DFTs; therefore the desired data format for the sequence of inputs [X in RHS array of Fig. 1(b)] to the row transforms is as shown in Fig. 6(b). This formatting issue is well known and is usually dealt with by performing a data transposition; however, such an additional step would considerably reduce the overall throughput. Alternatively, here it is possible to do the transposition "on-the-fly" during the course of the column DFTs. This can be accomplished by a systolic shift of the rows or columns of the matrices CM1, CM2, and WM as the DFTs are performed. For example, if the rows of an argument matrix A in the product $C = AB$ are circularly shifted down, then the rows of the result C are shifted down the same way.

The concept of on-the-fly transposition during the column DFT stage is best demonstrated by altering the code in (9) to include N_4 successive column DFTs of size N_3 with the matrix shift steps made explicit to give

```
for n to N4 do
  for j to N3/b do
    for k to N3/b do
      Y[j,k]:=WM[j,k]*add
        (CM1[j,i]*X[n,i,k],i=1..b)
    od;
    for k to b do
      Z[n,k,j]:=add
        (CM2[k,i]*Y[j,i],i=1..N3/b)
    od;
  od;
  WM := matrix_shift(WM, "down");
  CM1 := matrix_shift(CM1, "down");
  if n mod(N4/4)=0 then
    CM2 := matrix_shift(CM2, "down")
  fi;
od;                                          (10)
```

where the procedure "matrix_shift()" does a circular shift on the argument matrices in the direction indicated (there are several straightforward ways to implement circular shifts in hardware). Here each use of the procedure on WM and CM1 causes a shift of Z output by one column position to the right, and on CM2 causes the rows of Z to be shifted downward by one. After all N_4 column stage DFTs are completed, the rotated matrices have returned to their original position. (The SPADE simulator was used to verify correct systolic operations with these shifts incorporated for both column and row stage transforms.) The Z^t outputs, using this shift scheme for a sequence of 32 32-point column stage DFTs ($N_3 = N_4 = 32$), are then as shown in Fig. 7. It demonstrates that the positions of the output coefficients obtained from the column stage DFTs change for each

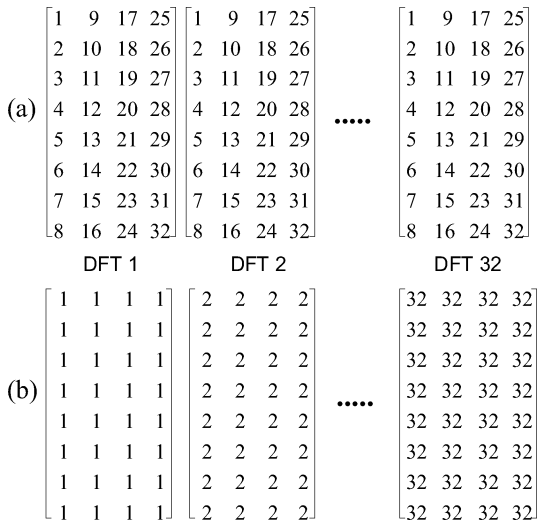


Fig. 6. (a) Locations of DFT coefficient numbers after each 32-point column DFT in PEs forming the RHS array of Fig. 1(a) (coefficients are numbered 1 to 32 and the arrays shown represent the successive outputs Z^t). (b) Desired DFT coefficient number locations in PEs of the RHS array in Fig. 1(b) prior to starting row stage DFTs.

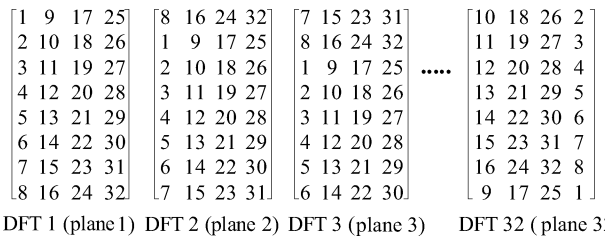


Fig. 7. PE locations [Z^t in RHS array in Fig. 1(a)] of successive 32-point column stage DFT coefficient outputs after matrix shifts shown in (10). Each RHS PE has 32 memory storage locations to store these results. The same memory location in all RHS PEs correspond to a plane of data.

of the N_4 DFTs in a way that all column coefficients are now available in desired PE locations shown in Fig. 6(b); however, each coefficient is stored in a different RHS memory location as indicated by the memory "planes" shown in Fig. 7. Therefore, each PE will need a different memory address to access the column element it is required to supply for the first row DFT; however, each subsequent value can be obtained by just incrementing this address by "1" modulo the memory size N_4. During the sequence of row stage DFTs, a similar set of matrix rotations as those in (10) is used to return each of the N_3 DFT outputs to the usual bit reversed form.

D. Combined Architecture for Nonsquare Factorizations ($N_3 \neq N_4$)

For this more general factorization, the array topologies from Fig. 1(a) and (b) cannot be used as before because the column and row DFT transform sizes are different. A variety of approaches are possible that can accommodate this case and a choice depends on performance requirements, transform sizes, and target technologies. Here a design is described for which regularity and modularity are the architectural priorities. Assuming the $N_3 > N_4$, the basic approach is to use the Fig. 1(a) architecture with length N_3/b to perform the N_4 column stage DFT transforms each of length N_3 as described in Section V-B.

1	9	17	25	25	1	9	17	17	25	1	9	9	17	25	1
2	10	18	26	26	2	10	18	18	26	2	10	10	18	26	2
3	11	19	27	27	3	11	19	19	27	3	11	11	19	27	3
4	12	20	28	28	4	12	20	20	28	4	12	12	20	28	4
5	13	21	29	29	5	13	21	21	29	5	13	13	21	29	5
6	14	22	30	30	6	14	22	22	30	6	14	14	22	30	6
7	15	23	31	31	7	15	23	23	31	7	15	15	23	31	7
8	16	24	32	32	8	16	24	24	32	8	16	16	24	32	8
(planes 1-8)				(planes 9-16)				(planes 17-24)				(planes 25-32)			

Fig. 8. PE locations [Z^t in RHS array in Fig. 1(a)] of successive 32-point column DFT coefficient outputs without matrix rotations that move data in north-south direction. The same memory location in all RHS PEs corresponds to a plane of output data.

Then for the row stage transforms the length N_3/b array structure is used to emulate the architecture of Fig. 1(b) with length N_4/b. This requires adding PE memory to the LHS array if sequentially ordered output is desired and requires modified processing to perform the N_3 row transforms each of length N_4. The modified processing uses different PE rows of the length N_3/b array to each independently execute b of the row stage DFTs. The details of how this is accomplished are described in the remainder of this section.

In Section V-C, it was shown that on-the-fly permutations could be used to distribute input data elements of each row DFT uniformly across the RHS of the array in Fig. 1(b). Alternatively, inputs for a row DFT could be distributed so that they reside only within one of the N_3/b east-west PE rows as shown in Fig. 8. In this case since there are N_3 row DFTs that need to be performed and there are N_3/b PE rows, each RHS PE row will contain inputs for b different row DFTs. For example, from Fig. 8, it can be seen that the first PE row will have transform input data for rows 1, 9, 17, and 25 (assuming $N_3 = N_4 = 32$).

With all DFT row data confined to specific PE rows, each PE row can function separately as a linear systolic "subarray" to perform the b row DFTs stored there. Each linear systolic subarray can be thought of as being created by projecting an array like that in Fig. 1(b) of length N_4/b along the north-south axis. The linear subarray shown in Fig. 9(a) then emulates the virtual array of the length N_4/b as shown in Fig. 9(b). Here, at any time τ the PEs in the linear subarray [Fig. 9(a)] perform the same operations as the virtual PEs [Fig. 9(b)] that are along the dashed line corresponding to that value of τ. In this case processing starts ($\tau = 1$) on the far RHS PE in the linear subarray [Fig. 9(a)] and corresponds the operation of the lower right PE in the array of Fig. 9(b). By time $\tau = 9$ the entire linear subarray of Fig. 9(a) will be active. At time $\tau = 13$ an element of Z will have been generated by the LHS PE adjacent to the multiplier PE and stored in the memory of Fig. 9(a). Each time cycle after that another element of Z will be generated by another LHS PE.

The sequence of operations starts in the computation plane just one time unit above and parallel to the X and CM2 planes in Fig. 1(b). At time $\tau = 9$ the next plane above this one is started in the far RHS PE of Fig. 9(a) and computations continue using a new row of coefficients from the edge memory CM1. The computations continue in this way executing plane-by-plane the

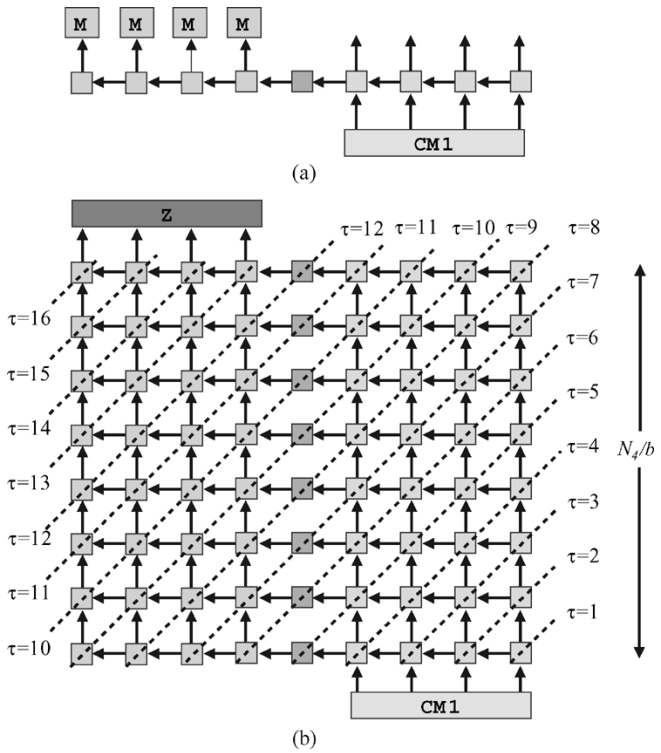


Fig. 9. (a) Linear systolic subarray corresponding to one PE row of architecture from Fig. 1(a) and (b), the virtual array of Fig. 1(b) which it emulates. The dashed lines specify which virtual PEs of array (b) are active in linear subarray (a) at time τ.

space-time computations shown in Fig. 1(b) until the DFT is completed. For example the linear subarray corresponding to PE row 1 of the architecture in Fig. 1(a) (top row in Fig. 8) will successively perform row DFTs number 1, 9, 17, and 25 (assuming $N_3 = N_4 = 32$). The second PE row of the N_3/b length array will start its operation as a linear subarray one cycle after the first linear subarray and will compute row DFTs 2, 10, 18, 26, and so forth until all N_3 row stage DFTs have been completed. The transformations in Tables I and II plus the ordering shown in Fig. 9(b) completely define the operation of each of the N_3/b linear systolic subarrays in the combined architecture. Additionally, the same volumes of space-time are used as for the $N_3 = N_4$ architecture of Section V-B.

The main architectural change from the array structure shown in Fig. 4 is that each LHS linear subarray must now have its own output memories as shown in Fig. 9(a), assuming an ordered serial output is desired. The architectural advantage of this structure compared to that described previously ($N_3 = N_4$) is that the single Z memory shown in Fig. 1(b) has now been distributed among the LHS row PE slices, making the structure more modular and hence easier to scale and partition. This also makes the architecture more suited to FPGA hardware, which typically has few large memories but an abundance of small memory options. The individual PE architectures of the linear subarray are shown in Fig. 10 and are almost identical to than in Fig. 2(b), the difference being that IM2 accumulates its result in the same PE. In summary the combined array structure consists of an array of length N_3/b that does the column transforms as shown in Fig. 1(a), but uses the N_3/b PE rows as N_3/b linear

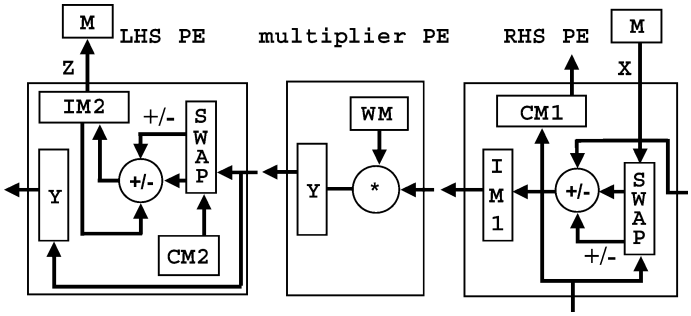


Fig. 10. PE functional operation for linear subarray design in Fig. 9(a).

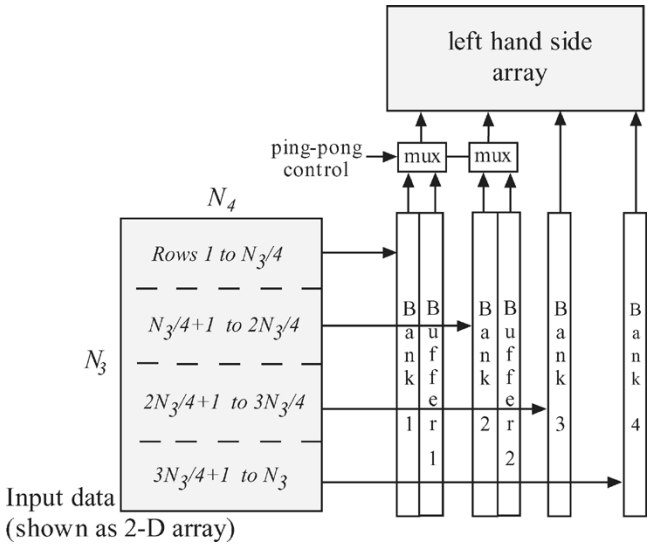


Fig. 11. An example of buffering used to achieve reordering of input data assuming input data arrives serially and row-wise from the source array shown. Each input row is routed to one of four banks of input memory feeding the four PE columns depending upon which of the four quadrants that row resides. Each data point in a row is written into a memory bank with a stride of $N_3/4$. When reading from these banks during column stage operations, input data is routed to buffer memories forming a ping-pong style input for some banks. For a particular column in the input data array above, the ordering of data out of the four memory banks to the LHS array is shown in Fig. 4. Each of the four memory banks is of size $N/4$.

subarrays each emulating the operations of a Fig. 1(b) length array of length N_4/b.

For DSP-based applications requiring continuous in-order serial input and output, the base-4 circuit needs reordering buffers at both input and output. The simplest such memory would be a traditional "ping/pong" memory buffer arrangement (Fig. 11). In this case during the column stage computation, the LHS array would read input from a memory of size N, while new serial input data is being written to other buffers of total size $f*N$ with $f \leq 1$. Because the relative time associated with reading input during column stage computation varies depending on the ratio N_3/N_4, f will vary from 0.4 to 1.0. Output data can be written into the LHS PE memories (Fig. 10) so that the aggregate output memory forms an $N_3 \times N_4$ array and the transform result (stored as $N_3 \times N_4$ elements) can be read out sequentially in column major order. More specifically, the memory in each LHS PE row contains four rows of this $N_3 \times N_4$ output array, and the memory in each LHS PE column contains $N_4/4$ columns of this output array. The buffering is necessary because four columns of the output array are being written simultaneously to the LHS memories, but can only be read out sequentially. Register transfer level (RTL) simulations have been performed to verify this continuous serial I/O model on DFT sizes 256 to 8192 points using input buffers that vary from $1.5N$ to $2N$ and an output buffer of fixed size N.

E. Accuracy and Partitioning

Given the target FPGA hardware and base-4 architecture, the most natural strategy for providing suitable accuracy is to use a block floating point strategy. Today's embedded multiplier FPGA hardware is typically pipelined and able to run at full clock speeds to support any mantissa length up to 18 bits, which would be adequate precision for most applications. Because the array structures in Fig. 1 are linear and regular, it is possible to implement a block floating point capability in a way that doesn't involve any global or problem-size dependent routing lines. This capability arises because all of the processing can be done in the east-west direction even though the array structure scales with problem size along the north-south axis. For example, in the architecture for $N_3 \neq N_4$ (Section V-D) all the data movement and processing during the row stage computation is done in the linear subarrays independently. Also, in the column processing stage the only north-south activity is the movement of X up the array and it propagates unchanged. Therefore, these linear subarrays can each have their own independent identical block floating point circuitry.

For the same reasons partitioning is very straightforward as a way to match the throughput of an application with the throughput of the underlying architecture without altering clock rates. Because only the input data X flows up unmodified through the LHS array, any horizontal section of the entire structure is capable of doing any of the processing. For example, half the array in Fig. 1 could be used with two separate passes of the input data X to do the column DFTs. It would only be necessary to make sure memory sizes are adequate and that the correct WM coefficients are used. Consequently, any array size can be used to do any allowed transform size given adequate memory resources.

VI. Throughput and Latency

Throughput can be determined from the sum of the component operations: column DFTs, twiddle multiplication, and row DFTs. In addition Fig. 3 shows that there is a processing delay associated with the switch from solution of Fig. 1(a) and (b). This delay in switching from column to row processing can be seen from Fig. 3 is twice the time to traverse the east-west direction of the array or $6b$. As noted in Section IV, the throughput of the direct base-4 implementation of a DFT is $N/b+1$, as reflected by the SPADE generated stacked DFTs in the space-time diagram of Fig. 3. However, this solution was obtained with the constraint that there be no overlap in space-time of input planes with regions of computation, specifically the polytopes associated with IM1 and IM2 in (9). This constraint was added only to clearly delineate regions of space-time associated with the different variables in (9). If this constraint is eliminated, then no "time slot" is necessary to accommodate the Z/CM1 [Fig. 1(a)]

or X/CM2 [Fig. 1(b)] planes and the throughput becomes N/b instead of $N/b+1$. Since the combined processing requires N_3 column DFTs of length N_4 and N_4 row DFTs of length N_3, the overall throughput *Thrpt* expressed as cycles per transform is

$$\text{Thrpt} = \underbrace{N_4\left(\frac{N_3}{b}\right)}_{\text{columnDFTs}} + \underbrace{b(N_4+1)}_{\text{twiddlemultiplication}} + \underbrace{N_3\left(\frac{N_4}{b}\right)}_{\text{rowDFTs}} + \underbrace{6b}_{\text{row/coldelay}} \quad (11)$$

for the architecture with $N_3 = N_4$ (Section V-B). The throughput for the architecture with $N_3 \neq N_4$ ($N_3 > N_4$), (Section V-D) is the same, except for the row stage DFTs. Since N_4/b is the throughput for computing a single row stage DFT with N_4/b PE rows, using just a single PE row to do the computation will require $(N_4/b)^2$ cycles. Therefore, with each PE row computing b DFTs

$$\text{Thrpt} = \underbrace{N_4\left(\frac{N_3}{b}\right)}_{\text{columnDFTs}} + \underbrace{b(N_4+1)}_{\text{twiddlemultiplication}} + \underbrace{b\left(\frac{N_4}{b}\right)^2}_{\text{rowDFTs}} + \underbrace{6b}_{\text{row/coldelay}} . \quad (12)$$

The throughput during the column and row DFTs is optimal because the speedup over a serial computation of (9) is equal to the number of PE array elements. For example, in (9) there are $N_4(N_3/b)^2$ multiplications associated with the column stage; so with N_3/b multiplier PEs, the optimal throughput is $N_4(N_3/b)$ the same value seen in (11) and (12). Similarly, in (9) there are a total of $2bN_4(N_3/b)^2$ column DFT additions when implemented as a running sum, so the optimal throughput with LHS and RHS arrays of adder/subtractors each of size $(N_3/b)\times b$ is again $N_4(N_3/b)$ and similarly for the design of Section V-D. These optimal throughputs are to be expected since each point within space-time bounds of the mapped variables performs a computation each cycle. The throughput associated with twiddle multiplications is also optimal because, the multiplier PEs operate at 100% efficiency after the initial b stage pipeline delay.

The computational latency L is the number of clock cycles it takes to do the first in a series of identical DFT calculations assuming input data is available in the required format. Consequently, it is obtained by adding to the throughput the number of cycles necessary to "fill" the pipeline. From Fig. 1(a) it can be seen that the maximum length data path in the array is the time to travel the length of the array (N_3/b cycles). Consequently

$$L = \frac{N_3}{b} + \text{Thrpt}. \quad (13)$$

If input data is only available in serial order, one data element per clock cycle, then there would be an additional latency delay of at least $3N/4$ because the circuit needs access to element $3N/4$ at the beginning of computation.

The throughputs and latencies for a variety of transform sizes calculated using (12) and (13) are shown in Table III, along with the corresponding hardware used. A nonpower-of-two 2816-point transform for which $N_3 \gg N_4$ is included as well. An RTL behavioral simulation was performed to verify to within a few control dependent clock cycles all values in the Table III. A similar RTL simulation was performed for a 1024-point transform based on the combined array architecture with $N_3 = N_4$ described in Section V-B to verify (11).

TABLE III
THROUGHPUT, COMPUTATIONAL LATENCY AND HARDWARE USED IN BASE-4 CIRCUIT

Points	N3	N4	Throughput (cycles/DFT)	Latency (cycles/DFT)	Mults	Adders
512	32	16	284	292	8	64
1024	32	32	668	676	8	64
2048	64	32	924	940	16	128
2816	176	16	860	904	44	352
8192	128	64	3356	3388	32	256

VII. COMPARISONS

A. Computational Efficiency

As explained in Section II, a variety of systolic array designs, characterized by use of uniform arrays of PEs, have been proposed for computation of the DFT. The most common are of two types, linear arrays that require $O(N^2)$ multiplications and the more computationally efficient 2-D arrays that require $O(N(N_3+N_4))$ multiplications (typically $N(N_3+N_4+1)$ [19], [22]) based on use of a row/column factorization $N = N_3N_4$. The base-4 design has the same asymptotic complexity as the 2-D arrays, but the "constant" factor is much smaller. This can be seen from (7) which shows that the number of multiplications for a single base-4 DFT is equal to the number of elements of W_M. For the N_4 column DFTs W_M is $N_3/4\times N_3/4$ and for the N_3 row transforms W_M is $N_4/4\times N_4/4$, so the ratio of 2-D systolic to base-4 multiplications, including the twiddle multiplication, is

$$16\frac{N_3+N_4+1}{N_3+N_4+16}$$

which is always greater than ten and asymptotically approaches 16. Similarly, there are $N(N_3+N_4-2)$ additions in a 2-D systolic computation [19] and $4(N_i/4)^2$ array additions for a DFT of size N_i for each multiplication by C_{Mi}. Consequently the ratio of 2-D systolic to base-4 additions is

$$2\frac{N_3+N_4-2}{N_3+N_4}$$

which is approximately equal to two.

B. Circuit Architecture Comparisons

Table IV provides a parameterized list of some of the important implementation characteristics of a variety of parallel circuits for computing the DFT. Because previous systolic architectures typically require one multiplier-adder combination per transform point, the hardware requirements for all but the smallest transforms are prohibitive. However, the base-4 design needs only $N_3/4$ multipliers and consequently, since

TABLE IV
COMPARISON OF IMPLEMENTATION PARAMETERS FOR LINEAR SYSTOLIC, 2-D SYSTOLIC, BASE-4, AND PIPELINED FFT CIRCUITS (n IS AN INTEGER > 0). THE NUMBER OF SPLIT-RADIX FFT COMPLEX MULTIPLICATIONS COMES FROM [33]

Architecture	Multipliers	Adders	Data Memory	Total Number of Multiplications	Throughput (cycles/DFT)	Latency (cycles/DFT)	Limits on N
1-D Systolic	N	N	$2N$	N^2	N	$2N-1$	none
2-D Systolic [19][22]	N	N	$2N$	$N(N_3+N_4+1)$	N_3+N_4+1	$2(N_3+N_4)-1$	$N=N_3N_4$
Base-4	$N_3/4$	$2N_3$	$N+5N_3$	$\frac{N}{16}(N_3+N_4+16)$	$N/4+N_4^2/4+4N_4+28$	$N/4+N_4^2/4+4N_4+N_3/4+28$	$N=256n$
Pipelined FFT [4]	$\log_4 N-1$	$4\log_4 N$	$N-1$	$\frac{N(3n-8)-(-1)^n}{9}+1$	N	$N+2\log_2(N)$	$N=2^n$

$N/(N_3/4)=4N_4$ and $N_4 \geq 16$, the base-4 design will always use at least $\times 64$ fewer multipliers than previous 2-D systolic arrays. Similarly, with $2N_3$ adders (Fig. 1), $N/(2N_3)=N_4/2$, and therefore the base-4 design will always use at least $\times 8$ fewer adders than 2-D systolic arrays. Since there are $\times 8$ more adders than multipliers in the base-4 design, the granularity of the design is inherently finer and therefore there is a potential for easier design, simpler partitioning and reduced wiring issues compared to 2-D systolic arrays, especially in the context of FPGA hardware. Also, linear and 2-D systolic arrays typically require $2N$ words for data in uniformly distributed PE registers, whereas N of the total $N+5N_3$ words of base-4 PE memory is of the more desirable RAM form. Finally, the computational latency of the base-4 design is inherently low because it uses a block-based systolic approach that reuses the same low latency architecture to do all the row and column DFTs.

Also included as a measure of relative circuit efficiency is a recent split-radix pipelined FFT that has multiplicative complexity advantages over other fixed radix and mixed radix pipelined FFT implementations [4]. Compared to a base-4 design for $N = 1024$, the pipelined FFT has $\sim 50\%$ worse throughput and latency, but uses 50% fewer multipliers and performs $\sim 50\%$ fewer multiplications. Circuits based on the PFA are not included in Table IV since there are few of them and they are not inherently easy to parameterize [7], [25], [26].

The data memory values shown in Table IV do not include input or output buffers for data reordering, as their use varies with application. For DSP-based applications requiring continuous in-order serial input and output, the base-4 and 2-D systolic circuits need reordering buffers at both input and output, whereas a pipelined FFT can be designed needing only an output buffer [34].

VIII. CONCLUSION

A parallel base-4 architecture is described that provides computationally efficient, high performance execution of the DFT for all transform sizes N which are divisible by 256. Past systolic DFT implementations are significantly less computationally efficient and require much more hardware in circuit implementations. Previously reported pipelined FFTs are limited to values of N that are a power of two and pipelined PFA approaches are limited to values of N that are composite with coprime factors. Additionally, the fine-grained, adder-centric, regular, locally connected (single global clock), linear base-4 structure makes efficacious use of the underlying target FPGA hardware and will also minimize logic delays and associated power consumption in the FPGA routing network. A good FPGA match is expected to provide higher clock rates than more coarse grained pipelined FFT/PFA designs. The base-4 architecture also supports execution of 2-D DFTs using the same circuit and its modularity allows simple partitioning strategies to match architecture/application throughputs. Other base-4 FFT designs are possible and these depend on the application requirements and target hardware. A timing analysis of a partially populated 16-bit fixed point base-4 circuit implemented using a recent generation FPGA chip (Altera Stratix, speed grade 3) showed that a clock speed of ~300 MHz could be expected, leading to a 1024-point complex DFT throughput of approximately 2.2 μsec, a number that compares favorably with recent FFT hardware, e.g., 34 μs for a radix-2^2 design [6].

REFERENCES

[1] R. N. Bracewell, *The Fourier Transform and its Applications*, 3rd ed. New York: McGraw-Hill, 1999.

[2] E. O. Brigham, *Fast Fourier Transform and its Applications*. Englewood Cliffs, NJ: Prentice-Hall, 1988.

[3] C.-H. Chang, C.-L. Wang, and Y.-T. Chang, "Efficient VLSI architectures for fast computation of the discrete Fourier transform and its inverse," *IEEE Trans. Signal Process.*, vol. 48, pp. 3206–3216, Nov. 2000.

[4] W.-C. Yeh and C.-W. Jen, "High-speed and low-power split-radix FFT," *IEEE Trans. Signal Process.*, vol. 51, pp. 864–874, Mar. 2003.

[5] K. Maharatna, E. Grass, and U. Jagdhold, "A 64-point Fourier transform chip for high-speed wireless LAN application using OFDM," *IEEE J. Solid-State Circuits*, vol. 39, pp. 484–493, Mar. 2004.

[6] S. He and M. Torkelson, "Design and implementation of a 1024-point pipeline FFT processor," in *Proc. IEEE Custom Integrated Circuits Conf.*, 1998, pp. 131–134.

[7] Z.-X. Yang, Y.-P. Hu, C.-Y. Pan, and L. Yang, "Design of a 3780-point IFFT processor for TDS-OFDM," *IEEE Trans. Broadcast.*, vol. 48, pp. 57–61, Mar. 2002.

[8] S. Y. Kung, *VLSI Array Processors*. Englewood Cliffs, NJ: Prentice-Hall, 1988, pp. 138–139.

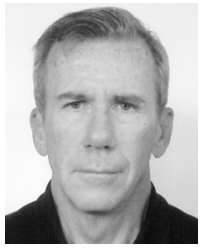
[9] H. T. Kung and C. E. Leiserson, "Systolic arrays (for VLSI)," in *Symp. Sparse Matrix Computations*, 1978, pp. 256–282.

[10] G. H. Allen, P. B. Denyer, and D. Renshaw, "A bit serial linear array DFT," in *Proc. ICASSP*, 1984, pp. 41A1.1–41A1.4.

[11] D. C. Kar and V. V. B. Rao, "A new systolic realization for the discrete Fourier transform," *IEEE Trans. Signal Process.*, vol. 41, pp. 2008–2010, 1993.

[12] J. A. Beraldin, T. Aboulnasr, and W. Steenaart, "Efficient 1-D systolic array realization for the discrete Fourier transform," *IEEE Trans. Circuits Syst.*, vol. 36, pp. 95–100, 1989.

[13] H. T. Kung, "Special-purpose devices for signal and image processing: An opportunity in VLSI," in *Proc. SPIE*, vol. 241, Real-time signal processing III, Jul. 1980, pp. 76–84.
[14] M. A. Bayoumi, G. A. Jullien, and W. C. Miller, "A VLSI array for computing the DFT based on RNS," in *Proc. Int. Conf. Acoustics, Speech, Signal Processing*, 1986, pp. 2147–2150.
[15] K. J. Jones, "High-throughput, reduced hardware systolic solution to prime factor discrete Fourier transform algorithm," *Proc. Inst. Elect. Eng. E*, vol. 137, no. 3, pp. 191–196, May 1990.
[16] E. Chu and A. George, *Inside the FFT Black Box*. Boca Raton, FL: CRC Press, 2000.
[17] C. N. Zhang and D. Y. Y. Yun, "Multidimensional systolic networks for discrete Fourier transform," in *Proc. 11th Int. Symp. Computer Arch.*, 1984, pp. 215–222.
[18] J. L. Aravena, "Triple matrix product architecture for fast signal processing," *IEEE Trans. Circuits Syst.*, vol. 35, no. 1, pp. 119–122, 1988.
[19] S. He and M. Torkelson, "A systolic array implementation of common factor algorithm to compute DFT," in *Proc. Int. Symp. Parallel Architectures, Algorithms and Networks*, Kanazawa, Japan, 1994, pp. 374–381.
[20] S. Peng, I. Sedukhin, and S. Sedukhin, "Design of array processors for 2-D discrete Fourier transform," *IEICE Trans. Inform. Syst.*, vol. E80-D, no. 4, pp. 455–465, Apr. 1997.
[21] W. Marwood and A. P. Clarke, "Matrix product machine and the Fourier transform," *Proc. Inst. Elect. Eng. G*, vol. 137, no. 4, pp. 295–301, Aug. 1990.
[22] H. Lim and E. E. Swartzlander, "Multidimensional systolic arrays for the implementation of discrete Fourier transforms," *IEEE Trans. Signal Process.*, vol. 47, no. 5, pp. 1359–1370, 1999.
[23] N. Ling and M. A. Bayoumi, "The design and implementation of multidimensional systolic arrays for DSP applications," in *Proc. Int. Conf. Acoustics, Speech, Signal Processing*, 1989, pp. 1142–1145.
[24] D. P. Kolba and T. W. Parks, "A prime factor FFT algorithm using high-speed convolution," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-25, pp. 281–294, Aug. 1977.
[25] B. Arambepola, "Discrete Fourier transform processor based on the prime factor algorithm," *Proc. Inst. Elect. Eng. G*, vol. 130, no. 4, pp. 138–144, Aug. 1983.
[26] R. M. Owens and J. Ja'ja', "A VLSI chip for the Winograd/prime-factor algorithm to computer the discrete Fourier transform," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-34, pp. 979–989, Aug. 1986.
[27] C. C. W. Hui, T. J. Ding, J. V. McCanny, and R. F. Woods, "A 64-point Fourier transform chip for motion compensation using phase correlation," *J. Solid State Circuits*, vol. 31, no. 11, pp. 1751–1761, 1996.
[28] J. G. Nash, "Hardware efficient base-4 systolic architecture for computing the discrete Fourier transform," in *Proc. IEEE Workshop Signal Processing Systems*, 2002, pp. 87–92.
[29] ——, "Automatic generation of systolic array designs for reconfigurable computing," in *Proc. Int. Conf. Engineering Reconfigurable Systems and Algorithms (ERSA 02)*, 2002, pp. 176–182.
[30] ——, "CAD tool for exploring latency optimal systolic array designs," in *Proc. SPIE ITCom*, vol. 4867, 2002, pp. 8–19.
[31] A. Singh and M. Marek-Sadowska, "Efficient circuit clustering for area and power reduction in FPGAs," in *Proc. 10th ACM Int. Symp. FGPAs*, Feb. 2002, pp. 59–66.
[32] L. Shang, A. S. Kaviani, and K. Bathala, "Dynamic power consumption in Virtex™ FPGA family," in *Proc. 10th ACM Int. Symp. FGPAs*, Feb. 2002, pp. 157–164.
[33] P. Duhamel, "Algorithms meeting the lower bounds on the multiplicative complexity of length-2^n DFTs and their connection with practical algorithms," *IEEE Trans. Acoustics, Speech, Signal Processing*, vol. 38, pp. 1504–1511, Sep. 1990.
[34] S. He and M. Torkelson, "A new approach to pipeline FFT processor," in *Proc. 10th Int. Parallel Processing Symp.*, 1996, pp. 766–770.



J. Greg Nash (S'72–M'75–SM'91) was born in Houston, TX, in 1945. He received the B.S.E. degree in basic engineering from Princeton University, Princeton, NJ, in 1968, the M.S. and Ph.D. degrees in electrical engineering from the University of California at Los Angeles (UCLA) in 1970 and 1974, respectively.

He is President of Centar, Los Angeles, a company engaged in designing signal processing IP and building CAD tools for mapping algorithms onto fine grained, application specific parallel circuits. Previously, he was with Hughes Research Laboratories where he was in charge of a group developing parallel algorithms, architectures and prototype computers for Hughes embedded military signal and image processing systems. At HRL he received two best paper awards and the Hughes Group Patent award. He has eight issued patents and has published over 60 scientific papers.

Dr. Nash has been active professionally, serving as Chairman of the IEEE VLSI Signal Processing Committee from 1986 to 1988, editor of the *Journal of VLSI Signal Processing* and on the organizational committees for more than ten signal/image processing workshops.