On the design of VLSI arrays for discrete Fourier transform

C.-M. Liu
C.-W. Jen

Indexing terms: VSLI arrays, Systolic algorithms, Discrete Fourier transform

Abstract: In this paper the design of VLSI arrays for discrete Fourier transform (DFT) is investigated through three topics: (i) algorithm exploitation, derivation and analysis, (ii) array realisation, and (iii) schemes to calculate arbitrarily long length DFT using a reasonable sized array. Four DFT systolic algorithms are examined and compared in terms of computing parallelism and computational complexity. Among the four algorithms, one is newly proposed. The new one exhibits much higher computing parallelism and lower computational complexity than the other three, but is applicable when the DFT length is prime. Based on the four algorithms, seven systolic arrays and seven two-level pipelined systolic arrays are devised. The outstanding features of these arrays are that the number of I/O channels is independent of the DFT length and the time overhead in manipulating consecutive data bundles are eliminated. Two schemes are presented to calculate long-length DFT using arrays with a reasonable number of processing elements. Performance of different algorithms, arrays and schemes is compared and summarised in six tables to serve as the selection criteria for different applications.

1 Introduction

Evolution in VLSI technology has propelled a review for the criteria of evaluating digital processing algorithms. The efficiency of an algorithm to be implemented by VLSI is based more on the degree of the communication complexity required between arithmetic elements rather than on the number of computations. Hence, the fact having been observed by many researchers [1–5] is that FFT (fast Fourier transform)-like algorithms that have been used extensively for their low numbers of multiplications are not well suited for VLSI implementation.

Systolic arrays [1, 2] can meet the increasing demands of processing speed and are suitable for VLSI implementation. Their attributes of parallel and pipeline processing are means to attain high computing power while structural regularity, modularity, and local interconnections give the feasibility of VLSI implementation. We refer to the paper in Reference 3 for the motivation of DFT systolic arrays over other architectures. Since systolic arrays designate one processing element (PE) as one pipeline stage, the computing parallelism can be obtained from the multiple PEs in an array. In the existing systolic arrays for DFT [2, 3, 6, 7], a PE should perform one complex multiplication and one complex addition. To implement such complicated arithmetic with VLSI hardware, the pipelined architecture for the arithmetic units (AUs) in interior PEs can attain higher computing parallelism. Two-level pipelined systolic arrays were presented with the idea of assigning pipeline stages into the AUs [8]. As a result, two-level pipelined systolic arrays can attain parallelism from the pipeline stages of AUs in an array and offer higher throughput and less computation time than systolic arrays. In this paper, the design of systolic arrays and two-level pipelined systolic arrays for DFT is investigated through algorithm derivation and analysis, VLSI array design, and schemes to calculate long-length DFT.

Since not every computing algorithm is suitable for VLSI array realisation, the exploitation and derivation of algorithms for VLSI arrays are persistently engaged. In the literature, there were three DFT algorithms [2, 3, 6] proposed for systolic array realisation [2, 3, 6, 7]. Also these algorithms were applied to calculate the 'discrete cosine transform' and 'discrete sine transform' [9, 10]. Although these algorithms offer about the same computation time and hardware cost when realised with systolic arrays, they should meet another review in computing parallelism when realised with two-level pipelined systolic arrays. In other words, these algorithms should face parallelism examination in AU level despite that their parallelisms in PE level are equivalent. In Section 2 we review the three systolic algorithms and propose a new one which especially suits for the realisation of two-level pipeline systolic array. The computing parallelism and computational complexity of these four algorithms are examined, showing that the new one exhibits much higher computing parallelism and lower computational complexity than the other three, but is applicable when the DFT length is prime. The performance of the four algorithms is summarised in Table 1.

The efficient design of a VLSI array for an algorithm should consider the parallelism in an algorithm and the required performance in different applications. In the literature there are systematical design procedures to design systolic arrays and two-level pipelined systolic arrays

Paper 8723G (E10), first received 16th April 1991 and in revised form 17th January 1992

C.-M. Liu is with the Department of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu, Taiwan, Republic of China

C.-W. Jen is with the Department of Electronics Engineering & Institute of Electronics, National Chiao Tung University, Hsinchu, Taiwan, Republic of China

from a systolic algorithm [11–21]. However, these methods do not have proper considerations in the reduction of I/O channels and I/O bandwidth, and in how to initialise arrays through limited input channels or pump results stored in PEs to output channels. These kind of problems affect the pipelining ability to arrays and degrade the associated performance. Such problems are defined as I/O problems in this paper. If the existing systolic arrays [2, 3, 6, 7] for the DFT are examined, they can be obtained by directly applying these systematical methods and have left the I/O problems unsolved. In Section 2, we show that the four systolic algorithms require a low number of input operands. This gives rise to the potential of solving the I/O problems. The arrays presented in this paper solve the I/O problems by adopting the scheme called tag control [22]. This scheme provides I/O channels with the controllability over the contents of the local registers in each PE and solves the I/O problems efficiently. In Section 3, seven systolic arrays and seven two-level pipelined systolic arrays are devised. The outstanding features of these arrays are that the numbers of I/O channels are independent of the DFT length, and the time overhead in manipulating consecutive data bundles is eliminated. These arrays are examined with various performance parameters and summarised in Tables 2–4. These tables provide a selection criteria for the feasible arrays in different applications. These tables also show that two-level pipelined systolic arrays have much better performance in the latency time, the throughput rate, and the average computation time than the corresponding systolic arrays.

The numbers of PEs in the VLSI arrays designed in Section 3 are proportional to the DFT length. When the DFT length is long, schemes to compute DFT by arrays with a reasonable number of PEs should be applied. The realization of the maximum parallelism with multiple PEs is one attribute to VLSI arrays. This attribute plus the features of local interconnection and simple control render the preference of DFT systolic algorithm over FFT algorithm. However, the maximum parallelism cannot be attained when the DFT length is too long for VLSI implementation. Under the circumstance, the calculation should be a combination of the parallel computations within the capability of small-size arrays and the serial computations fitting for long-length DFT. From the development of FFT-like algorithms [23–26], the required number of multiplications and additions can be reduced through serial computations. Hence, the increase of computing power through the parallel computations of VLSI arrays and the reduction of computational complexity through serial computations should be a promising approach to implement long-length DFT in VLSI circuits. In Section 4, two schemes based on this approach are presented to calculate long-length DFT with small-size arrays.

2 Four systolic DFT algorithms

The N-point DFT of an input sequence $x(0), x(1), \ldots, x(N-1)$ is defined as

$$y(k) = \sum_{i=0}^{N-1} x(i)W^{ik} \quad \text{for } k = 0, 1, \ldots, N-1 \tag{1}$$

where $W = \exp(-j2\pi/N)$. Eqn. 1 can be represented by the matrix-vector multiplication form as

$$\begin{bmatrix} y(0) \\ y(1) \\ \vdots \\ y(N-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & W & W^2 & \cdots & W^{N-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & W^{N-1} & W^{2(N-1)} & \cdots & W^{(N-1)(N-1)} \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ \vdots \\ x(N-1) \end{bmatrix} \tag{2}$$

If the matrix-vector multiplication is directly realised by linear systolic arrays, each PE should have one external connection (defined as I/O channel in this paper) to receive the twiddle factor W^{ik} at each time step. Such an array is not good when considering the routing complexity, the I/O bandwidth, and the pin limitation of a chip package. Observing the elements in the matrix of eqn. 2, the powers of W increases by a constant factor both along a column and a row. In the development of the three systolic algorithms in [2, 3, 6, 7], the constant increase in the powers is utilised so that the W^{ik} can be generated in the computing process and only N values of W^{ik} are applied from the external. The reduction in the number of operands W^{ik} provides the potential to design VLSI arrays with low I/O bandwidth and a low number of I/O channels. In Section 2.1, another systolic algorithm with higher computing parallelism and lower computational complexity is devised. These four algorithms will be represented with graphs, named dependence graphs (DGs). DGs are an efficient vehicle to exploit computing parallelism and design appropriate arrays as was indicated in the systematical design methods for systolic arrays [11, 13–17, 19] and two-level pipelined systolic arrays [20, 21]. In Section 2.2, four DGs are constructed for the four algorithms. The computing parallelism and computational complexity of these algorithms are examined based on these DGs. The results show that the proposed algorithm possesses much higher parallelism and lower complexity than the other three. However, the algorithm is applied specifically when the DFT length is prime. In Section 3, VLSI arrays are designed based on these four algorithms.

2.1 A new systolic DFT algorithm

Considering first an example of 5-point DFT, it can be represented as

$$\begin{bmatrix} y(0) \\ y(1) \\ y(2) \\ y(3) \\ y(4) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & W & W^2 & W^3 & W^4 \\ 1 & W^2 & W^4 & W^6 & W^8 \\ 1 & W^3 & W^6 & W^9 & W^{12} \\ 1 & W^4 & W^8 & W^{12} & W^{16} \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \\ x(4) \end{bmatrix} \tag{3}$$

Taking the periodicity property of $W^N = 1$ and permuting the input and output sequence, eqn. 3 has the form

$$\begin{bmatrix} y(0) \\ y(2) \\ y(4) \\ y(3) \\ y(1) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & W^2 & W & W^3 & W^4 \\ 1 & W^4 & W^2 & W & W^3 \\ 1 & W^3 & W^4 & W^2 & W \\ 1 & W^1 & W^3 & W^4 & W^2 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(3) \\ x(4) \\ x(2) \end{bmatrix} \tag{4}$$

The matrix in eqn. 4 has the property that the elements in the same diagonal line has the same value exclusive the 1s in the first row and the first column. The phenomenon was found by Rader [27], and will be utilised to develop a new systolic algorithm here.

If N is a prime number there exists some number π, not necessarily unique, such that there is a one-to-one mapping from the integers $i = 1, 2, \ldots, N-1$ to the integers $j = 1, 2, \ldots, N-1$ given by

$$j = \pi^i \text{ modulo } N \tag{5}$$

where the double parentheses in eqn. 6 denote modulo N arithmetic on the indices, that is

$$((\pi^i))_N = \pi^i \text{ modulo } N \tag{6}$$

In the following argument, π^i denotes the operation 'π^i modulo N' (for short). The fourier transform in eqn. 1 will be rewritten with i and k as the powers of a primitive element π. Because i and k take on the value zero which is not a power of π, the zero frequency components must be treated specially, that is

$$y(0) = \sum_{i=0}^{N-1} x(n) \tag{7}$$

$$y(k) = x(0) + \sum_{i=1}^{N-1} x(i)W^{ik} \quad \text{for } k = 1, 2, \ldots, N-1 \tag{8}$$

Replace i and k by π^i and π^k, eqn. 8 is rewritten as

$$y(\pi^k) = x(0) + \sum_{i=1}^{N-1} x(\pi^i)W^{\pi^{i+k}} \tag{9}$$

Set $j = N - i$. Eqn. 8 then becomes

$$y(\pi^k) = x(0) + \sum_{j=1}^{N-1} x(\pi^{-j})W^{\pi^{k-j}} \tag{10}$$

It can be shown that eqn. 4 is obtained from eqn. 10 by setting $N = 5$ and $\pi = 2$. The output vector is $[y(0), y(\pi^1), y(\pi^2), \ldots, y(\pi^N)]$ and the input vector is $[x(0), x(\pi^{-1}), x(\pi^{-2}), \ldots, x(\pi^{-N})]$. Observe that the superscript ik of W in eqn. 1 is now $(k-j)$ of W^π in eqn. 10. It is the $(k-j)$ term that leads to the same value in the same diagonal line of the matrix in eqn. 4. To help analyse the algorithm in the following subsection, this algorithm is expressed as the recursive form as follows

$$y(j, k') = y(j, \pi^k) = y(j-1, \pi^k) + x(\pi^{-j})W^{\pi^{k-j}} \tag{11}$$

where $y(j, \pi^k)$ indicates the value of (π^k)th or kth DFT sample at the jth recursive iteration. Here, $1 \leqslant k \leqslant N-1$, $1 \leqslant j \leqslant N-1$ and $y(0, \pi^k) = x(0)$. The DFT samples are obtained after $N-1$ iterations, that is

$$y(\pi^k) = y(N-1, \pi^k) \tag{12}$$

2.2 The algorithms analysis and comparisons

Based on the method in [15], the recursive form in eqns. 11 and 12 can be used to construct the DG for the presented algorithm in Fig. 4*a*. In the graph, the nodes represent the operations to be executed as described in Figs. 4*b* and *c*. The directed arcs mean the data dependence between two nodes; that is, the computed result from one node should be sent along an arc for operating in the other node. The DGs in Figs. 1, 2, and 3 are constructed for the three algorithms in [2, 3, 6]. The systolic algorithm illustrated in Fig. 1 was derived based on the Horner's rule [2], and was used to design the arrays in References 2 and 7. The algorithms illustrated in Fig. 2 and Fig. 3 are two other alternatives that were derived based on the required input sequence or two-dimensional DFT realisation. The algorithm in Fig. 3 propagates W^{-i} instead of the W^i in Fig. 1 and that in Fig. 2 propagates

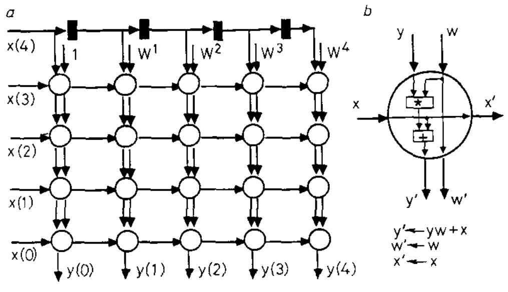


Fig. 1 *The algorithm in Reference 2*
a Dependence graph
b Functions of nodes

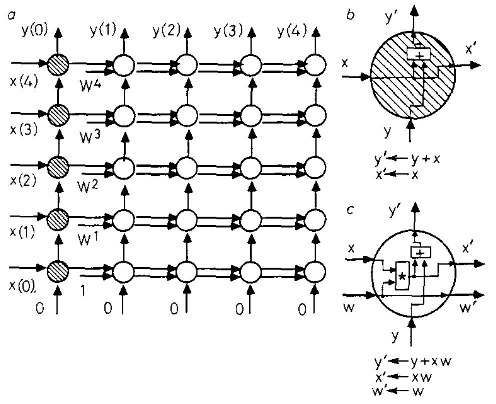


Fig. 2 *The algorithm in Reference 4*
a Dependence graph
b Functions of nodes
c Functions of dark nodes

W^i along another direction. The designed arrays in References 3, 6 and 9 are based on the two algorithms. One major distinction between Fig. 4 and others is that the two operands of each multiplication in Fig. 4 are obtained from transmitted data, instead of the iterated results from other operations. It will be shown that the distinction leads to the benefits in computational complexity and computing parallelism.

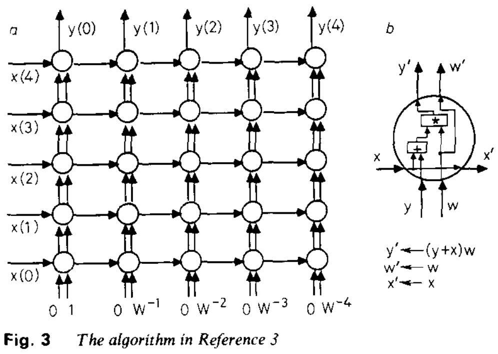


Fig. 3 *The algorithm in Reference 3*
a Dependence graph
b Functions of nodes

The computational complexity of the algorithms illustrated in References 2, 3, and 6 can be inspected from the node operations in the associated DGs. If input data $x(i)$ are complex, one complex multiplication in the nodes of these DGs can be computed by four real multiplications and two real additions typically. The first column in Table 1 lists the number of multiplications for these DGs when the input data are complex. If input data are real,

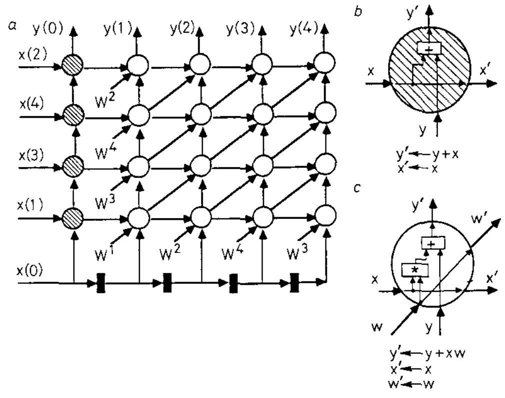


Fig. 4 *The presented algorithm*
a Dependence graph
b Functions of shaded nodes
c Functions of dark nodes

one multiplied operand in the node of Fig. 4*c* is real. Hence, one complex multiplication in Fig. 4*c* can be computed by two real multiplications. However, both operands of the multiplications in the nodes of other DGs are still complex numbers despite the input data being real. In this case, the number of real multiplications for Fig. 4*a* is $2(N-1)^2$ and those for Fig. 1*a*, Fig. 2*a*, or Fig. 3*a* are $O(4N^2)$ as illustrated in the second column of Table 1.

To examine the computing parallelism of systolic algorithms, a critical computing path is introduced. A critical computing path is the longest data flow path from input signal to output signal. The required time in such a path indicates the minimum required time for the computation of a systolic algorithm. In Fig. 4*a* the data flow path from $x(0)$ to $y(3)$ is the critical computing path. If systolic array realisation is considered, one PE will be one pipeline stage and the time for a signal to traverse a node in the DGs is assumed to be a system clock cycle labelled as T. The required time in the critical computing path is $(2N-2)T$. Similar analysis can be applied to Figs. 1–3 and the required time is listed in the third column of Table 1. If two-level pipelined systolic array realisation is adopted, the AUs in the PEs of array will be implemented with pipelined architectures and the required time of the critical path should be checked from all the AUs in a DG. Examining now the dark nodes in Fig. 4*c*, the operator from the input arc, labelled as y, to the output arc, labelled as y', is an addition. The operators for the other pairs of arcs labelled as x, x' and w, w' are null. This indicates that the critical computing path involves $(N-1)$ complex additions as well as one complex multiplication. This means that the massive multiplications in Fig. 4*a* can be computed in parallel with $(N-1)$ complex additions and one complex multiplication. Considering the difference between Fig. 4 and Figs. 1–3, the operators for the pair of arc y, y' in Figs. 1*b* and 2*b* are a multiplication and an addition, and the operator for the x, x' in Fig. 3*c* is a multiplication. The required time in the critical computing paths for the four DGs is listed in the fourth column in Table 1 with the assumption that a data signal traversing a node should take at least one unit of time. Also, the consumption time of a complex multiplication and an addition of these four algorithms are assumed to be equal so that the scrutiny of computing parallelism can be isolated from their variances in computational complexity. One interesting phenomenon is that the computing parallelism in Fig. 4 is much higher than that in Figs. 1–3 despite their similar functionality. From Table 1 it is obvious that the presented algorithm possesses higher computing parallelism and lower computational complexity than the other three but is applicable when N is prime.

3 VLSI array realisation

As described in the preceding section, the development of the systolic algorithms for the DFT can reduce the number of input operands W^{ik} when computing the DFT. The reduction provides the chances to design the VLSI arrays which can attain high computing power, low I/O band width, and low numbers of I/O channels. However, the systolic arrays in References 2, 3, 6, and 7 designed from these algorithms have not effectively utilised this property. As described in Section 1, there are systematical methods to design systolic arrays and two-level pipelined systolic arrays [11–21]. But these methods do not have proper considerations for the I/O problems. The systolic arrays in References 2, 3, 6, and 7 can be designed based on the systematical design methods and have left the I/O problems unsolved. In this section, seven systolic arrays and seven two-level pipelined systolic arrays are so devised; that is designed with particular attention given to the realisation of the parallelism in the algorithm and with care being given to the manipulation of I/O problems. The performance of these arrays is summarised in terms of various parameters in Tables 2–4.

3.1 Design of systolic arrays for DFT

The seven systolic arrays shown in Figs. 5–11 are based on the four DFT algorithms presented in the preceding

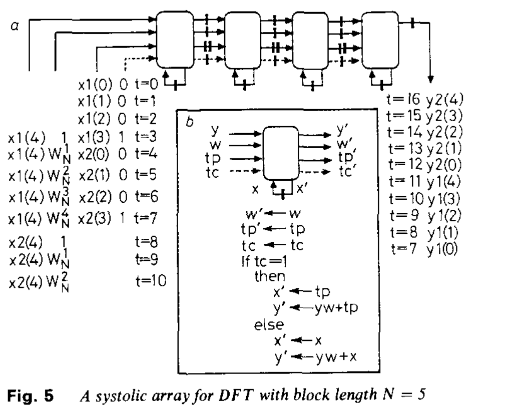
Table 1: Performance comparison of the four DGs (t_a = consumption time for complex addition and t_m = consumption time for complex multiplication)

DGs	Number of multiplications *for complex data*	Number of multiplications *for real data*	Required time in critical computing *path when systolic* array realisation is considered	Required time in critical computing *path when two-level* pipelined systolic array realisation is considered
Fig. 1	$4(N-1)N$	$4(N-1)N$	$(2N-2)T$	$(N-1)t_a + (N-1)t_m + (N-1)$
Fig. 2	$4(N-1)N$	$4(N-1)N$	$(2N-1)T$	$Nt_a + (N-1)t_m + 1$
Fig. 3	$4N^2$	$4N^2$	$(2N-1)T$	$Nt_a + Nt_m + (N-1)$
Fig. 4	$4(N-1)^2$	$2(N-1)^2$	$(2N-2)T$	$(N-1)t_a + t_m + (N-1)$

Table 2: Performance comparison of the systolic arrays (T = system clock cycle or the consumption time for one PE, L = wordlength, ACT = average computation time, N = block length, A_m = area of one multiplier, and A_a = area of one adder)

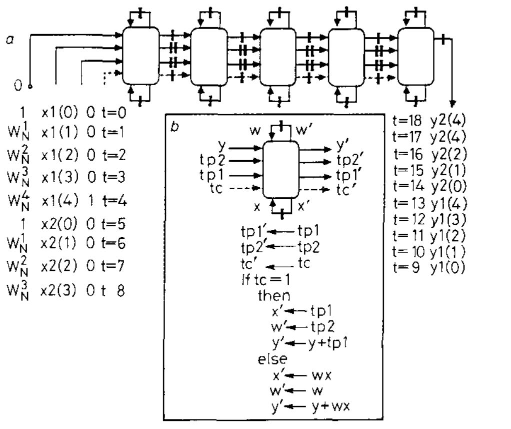
Arrays	Area complexity of PE (for real data)	Throughput rate	ACT	No. of PEs	Latency time	No. of I/O channels	Feature of I/O sequence
Fig. 5	$4A_m + 3A_a$	$1/T$	$(N-1)T$	$N-1$	$(3N-4)T$	$4L+1$	Natural order
Fig. 6	$4A_m + 3A_a$	$1/T$	NT	N	$(4N-3)T$	$3L+1$	Input sequence are reverse
Fig. 7	$4A_m + 4A_a$	$1/T$	NT	N	$(3N-2)T$	$3L+1$	Natural order
Fig. 8	$4A_m + 4A_a$	$1/T$	NT	N	$(3N-2)T$	$3L+1$	Output sequence are reverse
Fig. 9	$4A_m + 3A_a$	$1/T$	NT	N	$(3N-2)T$	$3L+1$	Input sequence are reverse
Fig. 10	$4A_m + 3A_a$	$1/T$	NT	N	$(4N-3)T$	$3L+1$	Natural order
Fig. 11	$2A_m + 2A_a$	$1/T$	$(N-1)T$	N	$(3N-4)T$	$6L+1$	I/O sequence are scrambled

section. These arrays have two distinctive features. First, input data, twiddle factors and computed results are piped in and drained out from the I/O channels at the extreme ends of a linear array. Second, 'tag control' is applied to control the contents of local registers and initialise arrays without an overhead in the average computation time. (The average computation time is the average time to finish one DFT calculation when consecutive DFT calculations are applied; it also indicates the minimum time interval between the first data of two consecutive DFT calculations piped in.) The performance of these seven arrays is illustrated in Table 2.



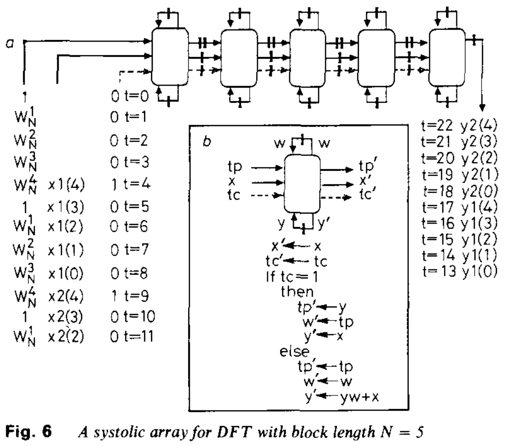
```
w' ← w
tp' ← tp
tc ← tc
If tc=1
then
    x' ← tp
    y' ← yw+tp
else
    x' ← x
    y' ← yw+x
```

Fig. 5 *A systolic array for DFT with block length N = 5*
a Time sequence of intput and output data are indicated in the same row of each data by 't = *'
b Functions of the PE in the array



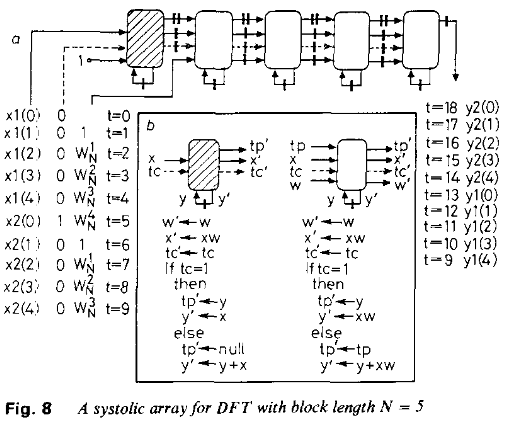
```
tp1' ← tp1
tp2' ← tp2
tc' ← tc
If tc=1
then
    x' ← tp1
    w' ← tp2
    y' ← y+tp1
else
    x' ← wx
    w' ← w
    y' ← y+wx
```

Fig. 7 *A systolic array for DFT with block length N = 5*
a Time sequence of input and output data are indicated in the same row of each data by 't = *'
b Functions of the PE in the array



```
x' ← x
tc' ← tc
If tc=1
then
    tp' ← y
    w' ← tp
    y' ← x
else
    tp' ← tp
    w' ← w
    y' ← yw+x
```

Fig. 6 *A systolic array for DFT with block length N = 5*
a Time sequence of input and output data are indicated in the same row of each data by 't = *'
b Functions of the PE in the array



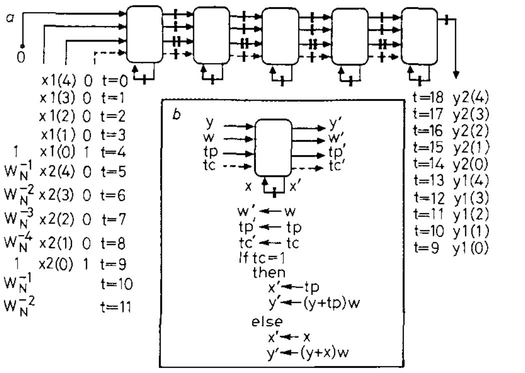
```
w' ← w
x' ← xw
tc' ← tc
If tc=1
then
    tp' ← y
    y' ← x
else
    tp' ← null
    y' ← y+x
```

```
w' ← w
x' ← xw
tc' ← tc
If tc=1
then
    tp' ← y
    y' ← xw
else
    tp' ← tp
    y' ← y+xw
```

Fig. 8 *A systolic array for DFT with block length N = 5*
a Time sequence of input and output data are indicated in the same row of each data by 't = *'
b Functions of the PE in the array

Consider now in detail the seven systolic arrays shown in Figs. 5–11. The arrays in Figs. 5 and 6 are derived from the algorithm illustrated in Fig. 1; the arrays in Figs. 7 and 8 from Fig. 2; the arrays in Figs. 9 and 10 from Fig. 3; and the array in Fig. 11 from Fig. 4. Multiple arrays can be derived from a DG and here we present two arrays for each DG in Figs. 1–3 and one array for the DG in Fig. 4. Other arrays can be exploited using the systematical design methods for systolic arrays [11–19]. The demonstration of these particular arrays (instead of others) is based on the performance in the number of PEs and computation time; other choices of array behave worse in this aspect. The designs of these arrays are based partly on the systematic design method given in References 11–19 and partly on the application of the tag control scheme explained in Reference 22.

In Figs. 5–11, $N = 5$ and consecutive DFT calculations are assumed. The first input and output data bundles are denoted by $x1(i)$ and $y1(i)$, and the second input and output data bundle are $x2(i)$ and $y2(i)$ and so on. Analysing for example the array in Fig. 11, input data $x(i)$ and twiddle factor, W^{ik} are piped in from the left most PE while output data $y(k)$ are drained out from the

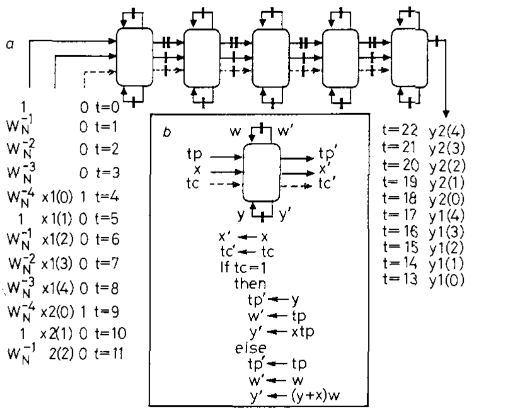


```
w' ← w
tp' ← tp
tc' ← tc
If tc=1
  then
    x' ← tp
    y' ← (y+tp)w
  else
    x' ← x
    y' ← (y+x)w
```

Fig. 9 *A systolic array for DFT with block length N = 5*

a Time sequence of input and output data are indicated in the same row of each data by '$t = *$'

b Functions of the PE in the array



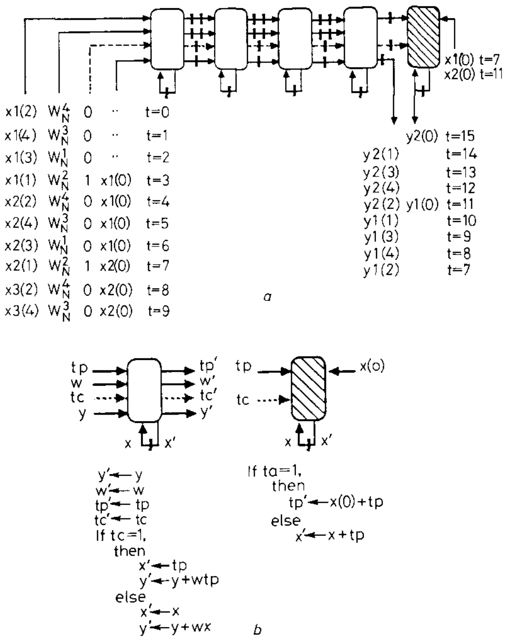
```
x' ← x
tc' ← tc
If tc=1
  then
    tp' ← y
    w' ← tp
    y' ← xtp
  else
    tp' ← tp
    w' ← w
    y' ← (y+x)w
```

Fig. 10 *A systolic array for DFT with block length N = 5*

a Time sequence of input and output data are indicated in the same row of each data by '$t = *$'

b Functions of the PE in the array

right most two PEs. The time instants for the input and output data sequence are also indicated in the same row of each data. Fig. 11*b* illustrates the functions of the PEs



```
y' ← y
w' ← w
tp' ← tp
tc' ← tc
If tc=1,
  then
    x' ← tp
    y' ← y+wtp
  else
    x' ← x
    y' ← y+wx
```

```
If ta=1,
  then
    tp' ← x(0)+tp
  else
    x' ← x+tp
```

Fig. 11 *A systolic array for DFT with block length N = 5*

a Time sequence of input and output data are indicated in the same row of each data by '$t = *$'

b Functions of the PEs in the array

in the array. Fig. 12 depicts the activity of the array at successive six clocks from $t = 3$ to $t = 8$, where the $yp^j(k')$ is the iterative result $y(j, k')$ in eqn. 11 of the pth data bundle. Each PE in the array has two additional links named 'tp' and 'tc' as depicted in Fig. 11*b*. Link 'tp' is used to carry data to appropriate PEs and the data in link 'tc' is to tell PEs when to load the data in link 'tp' into its local register. The data in 'tc' is a one-bit control signal, called 'tag control', and are used to tell PE when to perform suitable operations. Based on the control scheme, the data in the local register of each PE can be controlled appropriately from the input channels at the extreme ends of a linear array. The hardware overheads paid for the scheme in each PE is $(L + 1)$ one-bit links and about one demultiplexer. The time overhead is $(N - 1)T$ units of time, where T is the time period required for the operation in a PE. However, the time overhead can be skipped by overlapping the computation time of two consecutive DFT calculations. As depicted in Fig. 11, there is no extra time between the input of the first bundle $x1(i)$ and the second $x2(i)$, or between the output bundle $y1(k)$ and the bundle $y2(k)$. In other words, the tag control should give overhead to the latency time of a DFT problem and no overhead to the average computation time. (The latency time is the consumption time from the input of first data to the output of the final data for a DFT calculation.) The phenomenon can also be checked from the array activity in Fig. 12. From $t = 4$ to $t = 7$, the array calculates the first DFT problem by using $x1(i)$ and simultaneously brings $x2(i)$ for the second

DFT problem. Such concurrent computing favours the average computation time.

By the same analytical method just described, the other arrays in Figs. 5–10 can be checked. The one-bit control link 'tc' is used in these arrays either to control the contents of local registers or to assign suitable operations for PEs. The link 'tp' in the array is used to carry input data $x(i)$ or twiddle factor W^k from the left-most PE to appropriate ones, or carry output result $y(k)$ from the PEs to the right-most one. The 'tp' links in Figs. 6–8 are used to carry the twiddle factors and output data simultaneously. The performance of these arrays in the area complexity of a PE, average computation time, the number of PEs, latency time, the number of I/O channels and the features of I/O sequence is listed in Table 2 where the area complexity of a PE is approximated by the area required for the adders and multipliers in the white PEs of the associated array. The area complexity of a PE is evaluated by the required multiplications and additions in a PE. Considering for example the PE functions of Fig. 5*b*, this consists of one complex multiplication and one real addition. Since one complex multiplication can be realised by four multipliers and two adders, the complexity of a PE is $(4A_m + 3A_a)$, where A_m and A_a are the layout area required for one multiplier and one adder, respectively. Here, the area complexity is evaluated only when input data is real instead of complex because all the complexity of the arrays in Figs. 5–11 is $(4A_m + 4A_a)$ when input data are complex. For the array in Fig. 11 the input sequence is π^k for $k = 1, 2, \ldots, N$ and the output sequence is π^{-j} for $j = N, N - 1, \ldots, 1$. For comparison, it is assumed that the consumption time of a PE (or the system clock period) in all the arrays in Table 2 is T. If the required multiplications and adders in a PE are performed serially by one multiplier and one adder, the area complexity for all the PEs in these arrays will be equal although the clock period will be proportional to the area complexity listed in Table 2. That is, the area complexity and the system clock is a trade-off.

In addition to using the systematic methods given in References 11–19 and tag control to design systolic arrays in Figs. 5–11, another detail (or 'trick') is used to

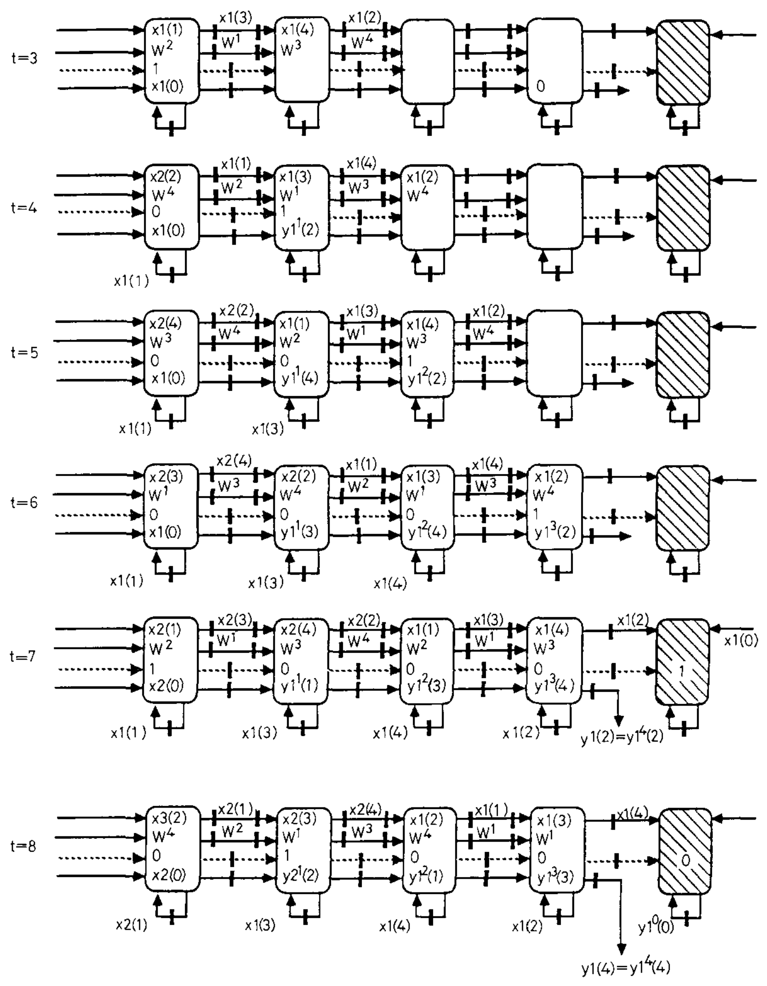


Fig. 12 *The activity of the systolic array in Fig. 11 at seven successive instants of time*

Table 3: Performance comparisons of the two-level pipelined systolic arrays (TLPSAs) (t_a = units of consumption time for the adders in a PE, t_m = units of consumption time for the multipliers in a PE, T' = units of consumption time for the multipliers and adders in a PE, L = DFT wordlength, ACT = average computation time, N = block length, A_m = area of one multiplier, and A_a = area of one adder

Arrays	Area complexity of PE (for real data)	Throughput rate	ACT	No. of PEs	Latency time	No. of I/O channels	Feature of I/O sequence
TLPSA for Fig. 5	$4A_m + 3A_a$	1	$N-1$	$N-1$	$(N-1)T' + (2N-3)$	$3L+1$	Natural order
TLPSA for Fig. 6	$4A_m + 3A_a$	$1/T'$	NT'	N	$(3N-2)T' + (N-1)$	$3L+1$	Input sequence are reverse
TLPSA for Fig. 7	$4A_m + 4A_a$	$1/t_m$	Nt_m	N	$NT' + (N-1)t_m$	$3L+1$	Natural order
TLPSA for Fig. 8	$4A_m + 4A_a$	$1/t_a$	Nt_a	N	$NT' + (N-1)t_a$	$3L+1$	Output sequence are reverse
TLPSA for Fig. 9	$4A_m + 3A_a$	1	N	N	$NT' + (2N-2)$	$3L+1$	Input sequence are reverse
TLPSA for Fig. 10	$4A_m + 3A_a$	$1/T'$	NT'	N	$(3N-2)T' + (N-1)$	$3L+1$	Natural order
TLPSA for Fig. 11	$2A_m + 2A_a$	1	$(N-1)$	N	$(N-1)t_a + t_m + (2N-4)$	$6L+1$	I/O sequence are scrambled

design the array in Fig. 11. This is also the reason why two arrays result from every DG in Figs. 1–3 but only one array from Fig. 4. Considering the DGs in Figs. 1–4, there are $(2N - 3)$ values of W^{ik} which will be supplied for the computation of Fig. 4 but only N values will be supplied for others. It seems that this may result in additional cost in computation time or the number of input channels. As shown in Fig. 11, it can be found that only one input channel is used for W^{ik} and the output data for N points of DFT can be obtained for every $(N - 1)$ clocks. How, then, can it be possible that the $2N - 3$ values can be transmitted in $(N - 1)$ clocks through one channel? The 'trick' used here is that the consequent sets of DFT computations can be overlapped so that, on average, only $(N - 1)$ values is needed for a DFT computation. For the array shown in Figs. 11 and 12, the W^{ik} values are piped in from $t = 0$ to $t = 6$ to compute the first data bundle $y1(i)$. Likewise, the second data bundle $y2(i)$ is computed by using the W^{ik} piped in from $t = 4$ to $t = 9$. It means that those W^{ik} values piped in from $t = 4$ to $t = 6$ are used to calculate both the first and the second output bundles. It is the overlap that avoids overhead in time or channels. The existent condition for the overlap comes from the cyclic property underpinning the modulo operation in eqn. 5, that is $\pi^{-(N-1-i)} = \pi^{i}$.

From Table 2 it can be seen that the throughput rate of all the arrays in Figs. 5–11 is equal. The average computation time, the number of PEs, and the number of I/O channels in Figs. 6–10 are equivalent. The latency time of the arrays in Figs. 6 and 10 are longer than the others; this comes from the time to initialise appropriate twiddle factors for arrays and to drain results to the boundary PE. The area complexity of PEs in Figs. 7 and 8 is larger than the others. The array in Fig. 5 has one less PE than all other arrays and provides a smaller latency time than those in Figs. 6–10. One special array which requires notice is that of Fig. 11: this array provides an area complexity of a PE with approximately one half that of the others. The benefit comes from the lower computational complexity of the associated algorithm presented in the preceding section. Table 2 provides a selection criteria for the feasible systolic arrays in different applications. Despite the variance among these arrays, all have two attributes when compared with others [2, 3, 6, 7]. First, the input data, twiddle factors and computed results are piped in and drained out from the I/O channels at the extreme ends of a linear array, and the number of channels is independent of DFT length. Secondly, the tag control is applied to control the contents of local registers and initialise arrays without any overhead in average computation time.

3.2 *Design of two-level pipelined systolic arrays for DFT*

The systolic arrays in the preceding subsection can be redesigned into two-level pipelined systolic arrays with performance as listed in Table 3. In this table, we assume the numbers of pipeline stages for an adder and a multiplier are t_a and t_m, respectively. If the basic pipeline period is one, t_a and t_m are also the consumption time of an adder and a multiplier. Also, T' is assumed to be the sum of t_a and t_m, i.e. $T' = t_a + t_m$. If the required time of the adder and the multiplier in two-level pipelined systolic arrays is compatible with that in systolic arrays, i.e. $T' \simeq T$, the performance of these two types of arrays can be compared from Tables 2 and 3. It can be seen that the redesigned arrays have the same performance as the original arrays in the area complexity, the number of PEs, the number of I/O channels, and the features of I/O sequence. But these two-level pipelined systolic arrays have much better performance in the throughput rate, the average computation time, and the latency time than the associated systolic arrays. If the last rows of Tables 2 and 3 are checked by substituting $N = 5$, $t_a = 2$, $t_m = 3$, and $T = 5$, the associated systolic array exhibits throughput rate being 1/5, latency time being 55, and average computation time being 20 while the associated two-level pipelined systolic array exhibits throughput 1, latency 18, and time 4. Fig. 13 is the two-level pipelined systolic arrays redesigned from Fig. 11 by setting t_a and t_m as 2 and 3, respectively. Table 4 list the required number of registers associated with each link in these redesigned arrays in terms of the parameters t_a and t_m. From tables

Table 4: The number of register elements required for each link in various arrays (t_a and t_m are, respectively, the number of pipeline stages or the units of consumption time for the adders and multipliers in a PE). *Note:* the link names are consistent with the names given in Fig. 5–11.

Arrays	Register numbers				
	x	y	w	tp	tc
TLPSA for Fig. 5	1	0	$t_a + t_m$	$t_a + t_m + 1$	$t_a + t_m$
TLPSA for Fig. 6	1	0	$t_a + t_m$	$t_a + t_m + 1$	1
TLPSA for Fig. 7	t_m	0	1	$t_a + t_m$	t_a
TLPSA for Fig. 8	t_m	0	1	$t_a + t_m$	t_m
TLPSA for Fig. 9	1	0	$t_a + t_m$	$t_a + t_m + 1$	$t_a + t_m$
TLPSA for Fig. 10	1	0	$t_a + t_m$	$t_a + t_m + 1$	1
TLPSA for Fig. 11	1	0	$t_a + 1$	$t_a + 1$	t_a

3 and 4, the desired two-level pipelined systolic arrays can be configured. Consider for example the array in Fig. 11, the five links in Fig. 11*b* are labelled as *x*, *y*, *w*, 'tp' and 'tc'. If t_a and t_m are set as 2 and 3, the number of delay elements associated with each links of the array in Fig. 13 is then obtained by substituting the values of t_a and t_m to the seventh row in Table 4. The time sequence of input data and computed result of the array can be configured from the performance prescriptions in Table 3.

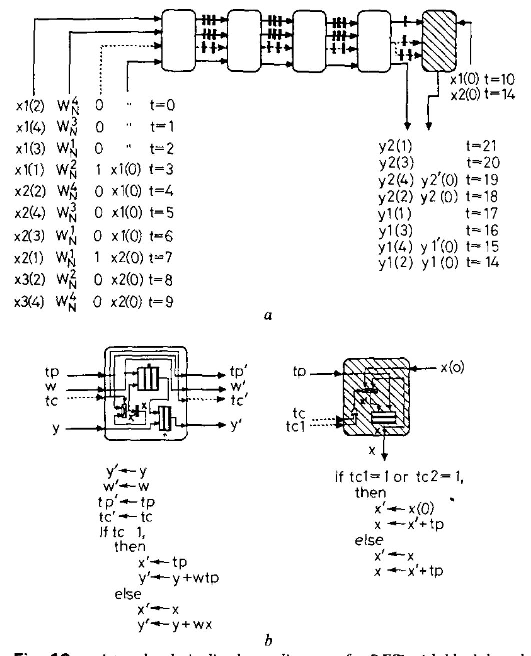


Fig. 13 *A two-level pipelined systolic array for DFT with block length N = 5*

a Time sequence of input and output data are indicated in the same row of each data by 't = *'
b Function of the PEs in the array

From the performance of these arrays in Table 3, the arrays redesigned from Figs. 5, 9 and 11 exhibit better performance in the throughput rate and the average computation time. Also, it can be seen that the array redesigned from Fig. 11 possesses higher speeds than the others. As listed in Table 1, the critical computing path for Fig. 4 is $(N-1)t_a + t_m + (N-1)$. Checked with the latency time $(N-1)t_a + t_m + (2N-4)$ in Table 3, the array needs an additional $(N-3)$ time units for initialisation. Similar checks can be made for other arrays to show that the two-level pipelined array in Table 3 can realise the maximum parallelism with some overheads in the initialisation of arrays.

Two specialties in the two-level pipelined systolic array redesigned from Fig. 11 are that the latency time is related with the computation time of only one multiplier and the average computation time is independent of the computation time of multipliers or adders. The two specialties indicate that the hardware cost required for the multiplier is much more important than the associated speed when the implementation of the two-level pipelined systolic array is considered. From the latency time, an interesting result is that the $(N-1)^2$ multiplications in Fig. 4 can be computed by $(N-1)$ multipliers in a linear array in a time related with just one multiplier. Obviously, the required time of $(N-1)^2$ multiplications has been suitably absorbed by the pipelined stages of AUs in $[N-1)t_a + t_m + (2N-4)]$ unit of time. The absorption results from the inherent parallelism of the computing algorithm.

One special point in Fig. 13 is that the data in link 'tp' are piped in every clock while the number of pipeline stages in an adder is two. Hence, the required *yp*(0) will be accumulated as two separated parts in the two pipeline stages of adder and extra manipulation is needed to add the *yp*'(0) and *yp*''(0) to obtain final *yp*(0). The argument can be extended to the general case where the number of pipeline stages in an adder is *q*. There will be *q* links, labelled as tc1, tc2, ..., tcq, between the white PE and the shaded PE. *yp*(0) will be obtained by adding *q* parts of data. The delay elements associated with the links of Fig. 13 are designed with the assumption that t_a is two and t_m is three. For a general consideration, the number of delay elements between white PEs is listed in Table 4 and the number of delay elements between a shaded PE and a white PE is one for link 'tp' and 1, 2, ..., *q* for links tc1, tc2, ..., tc*p*, respectively.

4 VLSI arrays for long-length DFT

The numbers of PEs in the VLSI arrays presented in Section 3 are equal to the DFT length *N* or *N* − 1. When the DFT length is long, the realisation of the DFT in an array with a reasonable number of PEs is necessary. In this section, two schemes are presented to consider this issue: Scheme 1 decomposes one DFT problem with length *N* into *P* DFT problems each with size Q and Scheme 2 factorises a one-dimensional DFT problem into a two-dimensional DFT problem. The factorised results can be efficiently computed by small-size arrays without sacrificing much time. Comparisons of the two schemes show that Scheme 1 requires smaller intermediate storage and lower control overhead while Scheme 2 gains benefits in computing speeds. The performance of the two schemes is summarised in Tables 5 and 6.

Table 5: Performance of scheme 1 (*t_a and t_m are, respectively,* the units of consumption time for the adders and multipliers in a PE; *T* = units of consumption of time for the multipliers and adders in a PE, and *N* = *PQ*)

Arrays	Throughput rate	*ACT*	Latency time	No. of I/O channels	No. of PE	Suggested *P*
Fig. 5	$1/PT$	PNT	$(PN+2Q-2)T$	$3L+1$	Q	$f(N)$
Fig. 7, Fig. 9	$1/PT$	PNT	$(PN+2Q-2)T$	$3L+1$	Q	2
TLPSA for Fig. 5	$1/P$	PN	$(PN+QT+Q-2)$	$3L+1$	Q	$f(N, T)$
TLPSA for Fig. 9	$1/P$	PN	$(PN+QT+Q-2)$	$3L+1$	Q	$(T+1)$
TLPSA for Fig. 7	$1/Pt_m$	PNt_m	(PNt_m+QT-t_m)	$3L+1$	Q	$(t_m+t_a)/t_m$

Table 6: Performance of scheme 2 ($N = n'n''$ for $n' \geqslant n''$)

Arrays	Throughput rate	*ACT*	Latency time	No. of I/O channels	No. of PE
Fig. 5	$1/T$	NT	$(2N + n' + n'' - 4)T + (n' - 1)T$	$(4L + 1)2$	$n' + n'' - 2$
Fig. 7, Fig. 8, Fig. 9	$1/T$	NT	$(2N + n' + n'' - 2)T + (n' - 1)T$	$(3L + 1)2$	$n' + n''$
Fig. 11	$N/(N - n')T$	$(N - n')T$	$(2N + n' - 4)T$	$(6L + 1)2$	$n' + n''$
TLPSA for Fig. 5	$1/T$	NT	$[2N + (n' - 1)T + (n'' - 1)T - 2] + (n' - 2)$	$(4L + 1)2$	$n' + n'' - 2$
TLPSA for Fig. 9	$1/T$	NT	$(2N + n'T + n''T - 2) + (n' - 1)$	$(3L + 1)2$	$n' + n''$
TLPSA for Fig. 7	$1/t_m$	Nt_m	$(2N + n')t_m + (n' + n'')t_a - t_m$	$(3L + 1)2$	$n' + n''$
TLPSA for Fig. 8	$1/t_a$	Nt_a	$(2N + n')t_a + (n' + n'')t_m - t_a$	$(3L + 1)2$	$n' + n''$
TLPSA for Fig. 11	$N/(N - n'')$	$(N - n'')$	$2(N - 1) + (n' - 1)(n' + n'' - 2)t_a + 2t_m$	$(6L + 1)2$	$n' + n''$

4.1 Scheme 1

The DFT problem is described as

$$y(k) = \mathrm{DFT}(x(i)) = \sum_{i=0}^{N-1} x(i)W^{ik} \tag{13}$$

One method of realising a DFT with length N in array with PE number Q is to decompose the DFT problem into P DFT problems each with length Q, where P is equal to N/Q. The idea can be achieved by replacing the index i in eqn. 13 by a coarse index and a vernier index as follows:

$$i = (i'' + Qi') \quad i' = 0, 1, \ldots, P - 1 \quad i'' = 0, 1, \ldots, Q - 1$$

Then,

$$y(k) = \sum_{i'=0}^{P-1} \sum_{i''=0}^{Q-1} x(i'' + Qi')W^{(i'' + Qi')k} \tag{14}$$

for $i' = 0, 1, \ldots, P - 1$; $i'' = 0, 1, \ldots, Q - 1$; and $k = 0, 1, \ldots, N - 1$. Eqn. 14 can then be expressed as P DFT problems each has length Q and N outputs as follows:

$$\begin{aligned} y(k) &= \sum_{i''=0}^{Q-1} x(i'')W^{(i'')k} + \sum_{i''=0}^{Q-1} x(i'' + Q)W^{(i'' + Q)k} \\ &\quad + \cdots + \sum_{i''=0}^{Q-1} x(i'' + PQ - Q)W^{(i'' + PQ - Q)k} \qquad (15) \\ &= y(k, 0) + W^{Qk}y(k, 1) + W^{2Qk}y(k, 2) \\ &\quad + \cdots + W^{(P-1)Qk}y(k, P - 1) \\ &= \sum_{i'=0}^{P-1} y(k, i')W^{Qi'k} \qquad (16) \end{aligned}$$

where

$$y(k, i') = \sum_{i''=0}^{Q-1} x(i'' + Qi')W^{i''k}$$

with $i' = 0, 1, \ldots, P - 1$; $k = 0, 1, \ldots, N - 1$. The P DFT problems; i.e. $y(k, i')$, can be serially computed by the arrays presented in preceding section, and the final result $y(k)$ is obtained by computing eqn. 16. For the arrays in Figs. 5, 7 and 9, the two-step computation can be efficiently computed by feeding the output links 'y'' and 'tc'' in the right-most PE to the input links 'y' and 'tc' in the leftmost PE through first-in first-out buffer (FIFO) and one demultiplexer. The size of the FIFO is $(N - Q)$.

Taking Fig. 5 as an example, eqn. 16 can be rewritten as

$$\begin{aligned} y(k) &= ((\ldots(y(k, P - 1)W^{Qk} + y(k, P - 2))W^{Qk} \\ &\quad + \cdots + y(k, 1))W^{Qk} + y(k, 0) \end{aligned} \tag{17}$$

The $y(k, i')$ can be serially computed by the array in Fig. 5 and the multiplication of W^{Qk} can be computed through the Q consequent multiplications of W^k in Q PEs. Fig. 14 depicts the modified version of the array in Fig. 5 for the realisation of long-length DFT. Fig. 14 has an extra link with FIFO to transmit the result and tag signal from the right most PE to the input link of the left most PE. As a result the tag signal can tell the demultiplexer in the left most part to transmit the $y(k, i')$ into the input link y of the left-most PE. It is such an arrangement that distributes the computations of eqn. 17 into the Q PEs without overheads in hardware and time, and the number of PEs is Q instead of $(Q - 1)$ as indicated by Fig. 5. A similar scheme may be applied to the arrays in Figs. 7 and 9 by using the same size of FIFO and demultiplexer. Based on the scheme, the systolic arrays in Figs. 5, 7 and 9 and the associated two-level pipelined systolic arrays can realise long-length DFT with performance as shown in Table 5. By replacing $P = 1$, and $Q = N$, it can be checked that these performances will be equal to those in Tables 2 and 3. In other words, the effect of the scheme is to reduce the number of PEs and increase the average computation time by a factor P. It should be noted that eqn. 16 results by replacing the index i in eqn. 13 by $(i'' + Qi')$. Such a replacement is suitable for the arrays in Figs. 5, 7 and 9 but not for the arrays in Figs. 6, 8 and 11. Observing the arrays in Figs. 5, 7 and 9, we find that all $x(i)$ must be stored in the PEs an one-to-one manner. Since index i is the pointer to $x(i)$ as in eqn. 13, the index i should be suitably partitioned as indicated by $(i'' + Qi')$ to reduce the number of PEs in these arrays. If the number of PEs in the arrays in Figs. 6 and 8 is to be reduced, index k should be similarly replaced by $(k'' + Qk')$. However, as we have pointed out, the scheme increases the average computation time by a factor P. From Table 3, the two-level pipelined systolic arrays of Figs. 6 and 8 is much worse than that of Figs. 5 and 9. Hence, we shall not

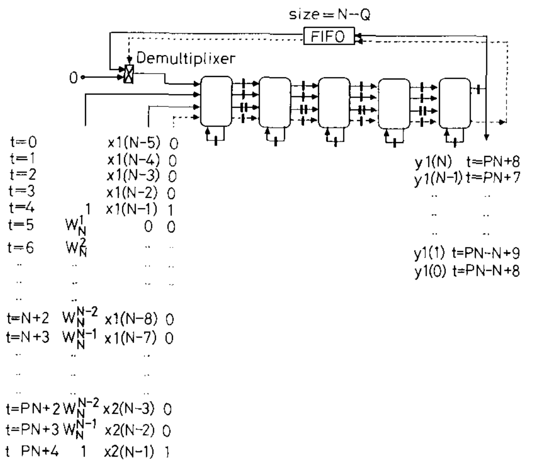


Fig. 14 *Modified array from that in Fig. 5 with number $Q = 5$ to realise DFT with blocklength $N = PQ$ using the scheme 1*

discuss such a replacement here. If the scheme is applied to the array in Fig. 11, it should also induce bad performance in the average computation time. As discussed in Section 3, the average computation time of the systolic array or two-level pipelined systolic array from Fig. 11 can be kept low by using the property, $\pi^{-(N-1-i)} = \pi^{i}$. The property should be useless if the arrays are to compute long-length DFT based on the scheme and the average computation time would degrade considerably. Because of the above reasons, the scheme is applied only to the arrays in Figs. 5, 7 and 9.

Comparing Tables 2 and 3 with Table 5, we find that scheme 1 induces an increase to the average computation time and reduce the throughput rate by a factor P. When the values of latency time in Tables 2 and 3 are compared with those in Table 5, the influence of scheme 1 on the latency time cannot be directly captured and some analysis should be applied. Consider, for example, the latency time $(PN + QT + Q - 2)$ in the fourth row of Table 5 and the time $(NT + (2N - 2))$ in the fifth row of Table 3, if scheme 1 is to have a decrease effect on latency time, then

$$(PN + QT + Q - 2) \leqslant [NT + (2N - 2)] \qquad (18)$$

and P should be

$$P \leqslant (T + 1) \qquad (19)$$

This shows that the feasible value of P is independent of N. Based on the analysis method, the entries in the fifth column of Table 5 are filled in. In Table 6, the $f(N)$ and $f(N, T)$ mean the function of N and function of N and T, respectively. It then follows that the scheme sometimes gives a positive effect on both computation time and PEs. As a result, the scheme can be undoubly used when the latency time, instead of average computation time or throughput rate, is the decisive requirement.

4.2 Scheme 2

Scheme 1 factorises a long-length DFT into multiple small-length DFT problems and then computes these problems serially by a small-size array. If the number of PEs is multiplied with the average computation time the result will be a constant value. So, the serial computing in scheme 1 has a tradeoff between the array size and average computation time. In other words, the parallelism and nearest neighbour interconnection which are attributes of VLSI array should blend the concepts of serial computing and temporal dependence when realising long-length DFT. If the serial computing is linked with the FFT algorithms which emphasise the reduction in the number of multiplications through serial computing, it may have some prominent effects on the performance. The scheme developed here is based on the idea.

The second scheme adopts the Good–Thomas factorisation [23, 24] in algorithm level and fits the factorised results into the arrays presented in Section 3. The derivation of the Good–Thomas algorithm is based on the Chinese remainder theorem for integers. It shows that when $N = n'n''$ and n' and n'' are relatively prime. Eqn. 13 can be rewritten as

$$y(k', k'') = \sum_{i''=0}^{n''-1} \beta^{i''k''} \sum_{i'=0}^{n'-1} x(i', i'')\gamma^{i'k'} \qquad (20)$$

where $\beta = \exp(-j2\pi/n'')$ and $\gamma = \exp(-j2\pi/n')$. Eqn. 20 is a two-dimensional DFT and can be computed serially through one-dimensional DFT process. The number of multiplication-adders is about $N(n' + n'')$ in the computing process. Compared with the N^2 in eqn. 13, eqn. 20 possesses simplified computational complexity.

If eqn. 20 is realised by two arrays with the numbers of PEs being n' (or $n' - 1$) and n'' (or $n'' - 1$), the computing procedure is given in Fig. 15a. The input data is first mapped to a two-dimensional form and fed into a linear array with size n'' (or $n'' - 1$) to compute the n' DFT problems serially. The computed result from the first array is then fed to the second array with size n' (or $n' - 1$) to compute the n'' DFT problems serially. The final two-dimensional result is mapped to one-dimensional form in original order. Fig. 15b shows the diagram of the arrangement for input data, two-dimensional input data, two-dimensinal result, and final natural order with n' and n'' are assumed to be five and three respectively. The performance of various VLSI arrays which realise the computing procedure in Fig. 15a is represented in Table 6 by neglecting the overhead in scrambling the input and output data and the communication of the two arrays. Compared with Tables 2 and 3, the scheme in Table 6 retains the average computation

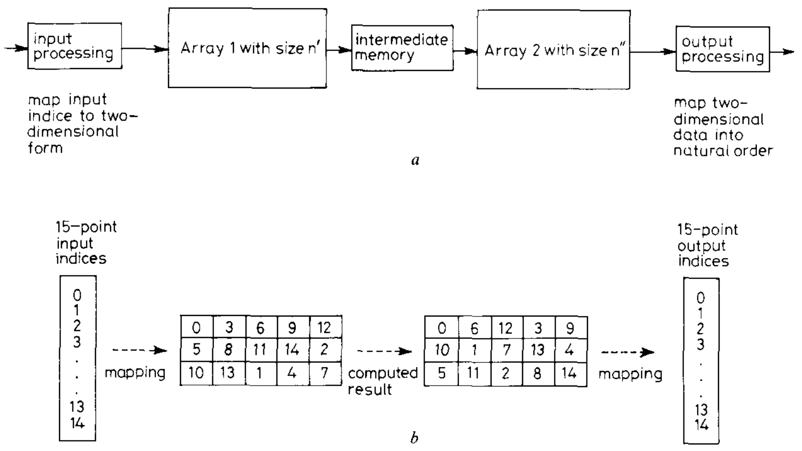


Fig. 15 *The computing procedure for DFT*
a Blocklength $N = n'n''$ using scheme 2 *b* Arrangement of input and output data in the input and output processing

time, gives small overhead to the latency time, doubles the number of I/O channels, and greatly reduces the number of PEs. One special case in Table 6 is that the two-level pipelined systolic array modified from Fig. 11 can reduce the average computation time. Comparing scheme 1 in Table 5 and scheme 2 in Table 6 by assuming $n' = P$ and $n'' = Q$, scheme 2 has superiority in throughput rate, average computation time, latency time but inferior in the number of I/O channels and PEs. The benefit of time for scheme 2 can be traced to be the reduction in the computational complexity. However, the points that should be noted is that the overheads in the scrambling indices, communicating two arrays and storing intermediate data, are neglected in Table 6. The overheads should be considered when an architecture system is built.

5 Conclusions

The design of VLSI arrays for DFT has been considered through three topics. In the first topic, four algorithms which are suitable for VLSI array realisation were considered. Among them, three algorithms were of interest for systolic array realisation; these are illustrated through 'dependence graphs' and a new systolic algorithm is proposed. The four algorithms were examined for functional parallelism and computational complexity. The results showed that the proposed algorithms has much higher computing parallelism and lower computational complexity than the other three. The benefits in parallelism and complexity were effectively exerted when the algorithm is realised with systolic arrays and two-level pipelined systolic arrays.

Secondly, seven systolic arrays and seven two-level pipelined systolic arrays were devised from the four algorithms. From analysis these arrays were shown to have short average computation time. Also, the required number of I/O channels in these arrays is independent of the number of PEs. The performance of these arrays is summarised in Tables 2 and 3.

The third topic considered is the scheme to calculate a long-length DFT problem using small-size arrays. Two schemes were presented with their performances listed in Tables 5 and 6. These schemes first factorise a one-dimensional DFT into two-dimensional DFT. Then the two-dimensional DFT is effectively calculated by using small-size arrays. Scheme 1 has been shown to be superior to scheme 2 in the number of I/O channels and PEs, but inferior in throughput rate, average computation time and latency time.

6 Acknowledgment

The authors wish to thank the anonymous reviewers for their helpful and constructive comments. This work was supported by DRPCC, Taiwan, R.O.C. under contract CS80-0210-D009-08.

7 References

1 KUNG, H.T.: 'Why systolic architectures?', *Comput. Mag.*, 1982, **15**, (1), pp. 37–45

2 KUNG, H.T.: 'Special purpose devices for signal and image processing: An opportunity in very large scale integration (VLSI)', *in* Proceedings of SPIE, (Real Time Signal Processing III), **241**, 1980, pp. 76–84

3 BERALDIN, J.A., ABOULNASR, T., and STEENAART, W.: 'Efficient one-dimensional systolic array realization of discrete Fourier transform', *IEEE Trans.*, 1989, **CAS-36**, (1), pp. 95–100

4 CURTIS, T.E., and WICKENDEN, J.T.: 'Hardware-based Fourier transforms: algorithms and architectures', *IEE Proc.*, 1983, **130F**, (5), pp. 423–432

5 THOMPSON, C.D.: 'Fourier transforms in VLSI', *IEEE Trans. Comput.*, 1983, **C-32**, (1), pp. 1047–1057

6 CHANG, L.W., and CHEN, M.Y.: 'A new systolic array for discrete Fourier transform', *IEEE Trans. ASSP*, 1988, **36**, (10), pp. 1665–1667

7 BAYOUMI, M.A., JULLIEN, G.A., and MILLER, W.C.: 'A VLSI array for computing the DFT based on RNS', *in* Proceedings of ICASSP 86, Tokyo, pp. 2147–2150

8 KUNG, H.T., and LAM, M.S.: 'Wafer scale integration and two-level pipelined implementations of systolic arrays', *J. Parallel and Distrib. Comput.*, 1984, pp. 32–64

9 CHO, N.I., and LEE, S.U.: 'DCT algorithms for VLSI parallel implementations', *IEEE Trans. ASSP*, 1990, **38**, (1), pp. 121–127

10 LEE, M.H., and YASUDA, Y.: 'New 2D systolic array algorithm for DCT/DST', *Electron. Lett.*, 7 Dec. 1989, pp. 1702–1703

11 QUINTON, P.: 'Automatic synthesis of systolic arrays from recurrence equations'. Proceedings of the 11th Annual Symposium on Computer Architecture, pp. 208–214.

12 LI, G.L., and WAH, B.W.: 'The design of optimal systolic arrays', *IEEE Trans. Comput.*, 1985, **C-34**, pp. 66–77

13 DELOSOME, J.M., and IPSEN, I.C.F.: 'An illustration of a methodology for the construction of efficient systolic architectures in VLSI'. Proceedings of the 2nd International Symposium on VLSI Technology, Systems and Applications, Taipei, 1985, pp. 268–273

14 CHEN, M.C.: 'A design methodology for synthesizing parallel algorithms and architectures', *J. Parallel & Distrib. Comput.*, Dec. 1986, pp. 461–491

15 KUNG, S.Y., LO, S.C., and LEWIS, P.S.: 'Optimum systolic design for transitive closure and shortest path problems', *IEEE Trans. Comput.*, 1987, **C-36**, (5), pp. 603–614

16 JAGADISH, H.V., RAO, S.K., and KAILATH, T.: 'Array architecture for iterative algorithms', *IEEE Proc.*, 1987, **75**, (9), pp. 1304–1321

17 RAO, S.K., and KAILATH, T.: 'Regular iterative algorithms and their implementation on processor arrays', *Proc. IEEE*, 1988, **76**, (3), pp. 256–269

18 FORTES, J.A.B., and MOLDOVAN, D.I.: 'Parallelism detection and algorithm transformation techniques useful for VLSI architecture design', *J. Parallel Distrib. Comput.*, 1985, pp. 277–301

19 LIU, C.-M., and JEN, C.-W.: 'On the design of algorithm-based fault-tolerant VLSI array processor', *IEE Proc.*, 1989, **136E**, (6), pp. 539–547

20 JEN, C.-W., and LIU, C.-M.: 'Two-level pipeline design for image resampling'. International Conference on ASSP, Glasgow, Scotland, 1989, pp. 2441–2444

21 LIU, C.-M., and JEN, C.-W.: 'Hierarchical synthesis of two-level pipelined systolic arrays', under revision in IEEE Trans. Circuits and Systems.

22 JEN, C.-W., and HSU, H.Y.: 'The design of a systolic array with tags input'. International Symposium on Circuits and Systems, Finland, 1988, pp. 2263–2266

23 GOOD, I.J.: 'The interaction algorithm and practical Fourier analysis', *J. Royal Statist. Soc.*, 1958, **B20**, pp. 361–375, and Addendum, 1960, **22**, pp. 372–375

24 THOMAS, L.H.: 'Using a computer to solve problems in physics', *in* 'Applications of Digital Computers' (Ginn and Co., Boston, MA, 1963).

25 WINOGRAD, S.: 'On computing the discrete Fourier transform'. Proceedings of the National Academy of Science, USA, 73, 1976, pp. 1005–1006

26 WINOGRAD, S.: 'On computing the discrete Fourier transform', *Math. Comput.*, 32, 1978, pp. 175–199

27 MEAD, C., and CONWAY, L.: 'Introduction to VLSI systems'. Addison-Wesley, pp. 274–275

28 RADER, C.M.: 'Discrete Fourier transforms when the number of data samples is prime', *Proc. IEEE*, 1968, pp. 1107–1108