COMPUTATION OF DISCRETE FOURIER TRANSFORMS OVER FINITE ABELIAN GROUPS USING PIPELINED AND SYSTOLIC ARRAY ARCHITECTURES*

M. G. KARPOVSKY**, E. A. TRACHTENBERG,*** T. ROZINER**

** BOSTON UNIVERSITY, COLLEGE OF ENGINEERING, BOSTON, MA 02215
*** DREXEL UNIVERSITY, DEPT. OF ELECTRICAL AND COMPUTER ENGINEERING, PHILADELPHIA, PA 19104

ABSTRACT

The paper discusses two methods for computation of Generalized Discrete Fourier Transforms (GDFT's) over finite groups in a multiprocessor environment. We note that GDFT is simply the multidimensional DFT if the underlying group is Abelian. We use a precise algorithm for GDFT computation by a multiprocessor SIMD system with local non-shared memory. Tradeoffs between hardware complexity and computation time are established for unibus and complete communication networks. Bounds on the number of operations, data transfers and communication links are presented. The GDFT algorithm is then implemented using a pipelined architecture. Bounds are given for numbers of operations, data transfers, delay of the whole pipeline and clock period for unibus and complete communication networks. In order to use standard systolic arrays for GDFT computation,it is necessary to consider its direct computation instead of the sequential FFT-like algorithm used for the pipelined architecture. In case of Abelian groups we discuss such efficient systolic array implementations of GDFT which can be made adoptable to the structure of the underlying group, and we estimate their overall speed and hardware complexity.

1. INTRODUCTION

Recently, there has been an increasing interest in implementing different types of algorithms on systolic array structures, because these architectures are extremely suitable for VSLI technology [4-6]. Discrete Fourier Transform (DFT) is widely used in signal and image processing in the analysis, synthesis and optimization of linear time invariant systems [1-3]. However, the corresponding fast butterfly-like algorithm is inherently sequential. That restricts full utilization of parallel computations and systolic array architectures for real time applications. In [1] a method was presented for an optimal implementation of Generalized Discrete Fourier Tranform (GDFT) algorithms over finite groups in a multiprocessor environment. A SIMD multiprocesor system with local non-shared memory was used and tradeofs were established between computation time and hardware complexity for unibus and complete communication networks. In the next section we present that algorithm together with bounds on the number of operations, data transfers and communication links. It is shown that using non-Abelian (e.g. quaternion) groups results in the optimal (i.e. fastest) GDFT performance. In section 3 the GDFT algorithm is implemented using a pipelined architecture. Bounds are given for numbers of operations, data transfers, the delay of the whole pipeline and the clock period for unibus and complete communication networks. In section 4 we discuss implementation of two-dimensional GDFT (when the underlying group is a direct product of two cyclic groups) by a systolic array. Such an array consists of identical processing elements (PEs), where only the neighboring PEs are allowed to communicate with each other. The PEs are identical in the number of input/output lines as well as in their internal structure. In order to use standard systolic arrays, it is necessary to consider direct GDFT computation instead of the sequential FFT-like algorithm used for the pipelined architecture. Also in that section we consider an approach to efficient systolic array implementations of GDFT which can be made adoptable to the strucutre of the underlying group $G=G_1xG_2$, where G_1,G_2 are arbitrary Abelian groups.

* The work was supported by the Office of Naval Research Under Naval Research Contract N00014-87-K-0735 and by the National Science Foundation under Grant ECS-8512748.

2. COMPUTATION OF GENERALIZED DISCRETE FOURIER TRANSFORM (GDFT).

Given a set {0,1,...,N-1} of N integers, let G be an arbitrary finite group of order N imposed on it. Let R_ω be the ω^{th} irreducible unitary representation of G of dimensional d_ω. The set $\{R_\omega\} = G$ of all these representations is called the dual object of G and its elements constitute an orthogonal basis in the space $\{f: G \to \mathbb{C}\}$, where $\mathbb{C}$ is the field of complex numbers. The direct and inverse Fourier transform over G are defined as follows

$$\hat{f}(\omega) = N^{-1} d_\omega \sum_{i \in G} f(i)\, R_\omega(i^{-1}),\ R_\omega \in \hat{G};\quad f(i) = \sum_{R_\omega \in \hat{G}} \mathrm{Trace}\,(\hat{f}(\omega)\ R_\omega(i)),\ i \in G; \qquad (1)$$

where $\hat{f}(\omega)$ is a $(d_\omega \times d_\omega)$ - matrix over C.

The transform (1) has the usual properties of linearity, group transalation, convolution, Plancherel and Poisson theorems, etc., [7]. Without loss of generality, consider the computation of the direct transform only. We note that the algorithm described in [7] computes (1) in at most $N\sum_{j=0}^{m-1} n_j$ computer operations using N memory locations for intermediate data storage in case when G is a direct product of groups G_j (of order n_j), $G = \prod_j G_j$, $0 \le j \le m-1$, $N = n_0 n_1 \ldots n_{m-1}$. We are going to consider a precise fast algorithm [1] for computing (1). Naturally, (see Table 2) it is always much faster than the upper bound discussed above.

Let $x = (x_0, x_1, \ldots, x_{m-1}) \in G$, $x_j \in G_j$, $0 \le j \le 1$. For $R_\omega \in \hat{G}$ we have $R_\omega(x) = \bigotimes_{j=1}^{m=1} R_{\omega_j}(x_j)$, where $R_{\omega_j} \in \hat{G}_j$, $\otimes$ stands for Kronecker product of matrices and $\omega = (\omega_0, \omega_1, \ldots, \omega_{m-1})$.

In that case we have by (1):

$$\hat{f}(\omega) = \hat{f}(\omega_0, \omega_1, \ldots, \omega_{m-1}) = \frac{d_\omega}{N} \sum_{x_0} \sum_{x_1} \cdots \sum_{x_{m-1}} f(x_0, x_1, \ldots, x_{m-1}) \bigotimes_{j=0}^{m-1} R_{\omega_j}(x_j^{-1})$$

$$= \frac{d_\omega}{N} \sum_{x_{m-1}} \left(\cdots \left(\sum_{x_1} \left(\sum_{x_0} \left(f(x_0, x_1, \ldots, x_{m-1}) \otimes R_0(x_0^{-1}) \otimes \ldots \right) \otimes R_{m-1}(x_{m-1}^{-1}) \right) \right) \right). \qquad (2)$$

The calculation of spectrum $\hat{f}(\omega)$ is performed in m steps (Step j over G_j, $0 \le j \le m-1$). For Step j: $f_{j+1}(\omega_0, \omega_1, \ldots \omega_j, x_{j+1} \ldots x_{m-1}) = \sum_{x_j} f_j(\omega_0, \omega_1, \ldots, \omega_{j-1}, x_j, \ldots, x_{m-1}) \otimes R_{\omega_j}(x_j^{-1})$.

The algorithm, described in [1] is based on that m-step procedure. In step j, N/n_j subsets of which contain n_j elements of input data f(0), f(1),...,f(N-1) are used for the butterfly algorithm of the constituent group G_j. Denote the number of operations needed to perform the algorithm of G_j on a single input data set (one out of N/n_j) as $L(G_j)$. (In general, the algorithm is characterized by a pair of numbers - the number of additions/subtractions and the number of multiplications. We shall assume $L(G_j)$ is the equivalent number of additions). A variety of examples of butterfly algorithms is given in [1]. The numbers L(G) for some groups are given in Table 1, where C_N stands for the cyclic group of order N with respect to addition mod N, S_3 denotes the symmetric of order 6, and Q_2 denotes the quaternion group of order 8.

Groups	C_2	C_3	C_4	C_8	S_3	Q_3
Additions	2	7	8	24	14	20
multiplications	-	4	-	10	4	-

Table 1: Computation of Fourier transform, Single Processor

To perform the spectrum calculation over $G = G_0 \times G_1 \times \ldots \times G_{m-1}$ by a single processor, L(G) operations are needed:

$$L(G) = \sum_{j=0}^{m-1} L(G_j) \prod_{\substack{i=0 \\ i \ne 0}}^{m-1} n_i = N \sum_{j=0}^{m-1} \frac{L(G_j)}{N_j} \qquad (3)$$

In Table 2 below we summarize and compare computation of the Fourier Transform (1), (2) using the direct method, the upper bound algorithm [7] and the precise fast algorithm [1].

Group / Algorithm		C_8	C_3^2	S_3	Q_2	$S_3 \times Q_2$	$Q_2 \times Q_2$
N^2	adds.	64	64	36	64	2304	4096
	multpl.	64	0	36	0	2304	0
$N \sum_j n_j$	adds.	48	48	n.a.	n.a.	672	1024
	multpl.	48	0	n.a.	n.a.	48	0
L(G) by (3)	adds.	24	24	14	20	336	384
	multpl.	10	0	4	0	32	0

Table 2: Comparison of Fourier Transform Algorithms Single Processor

Consider n>1 processor working simultaneously on the ordered data f(0), f(1),...,f(N-1), $x \in G$, $x=(x_0,x_1,...,x_{m-1})$, $x_j \in G_j$; , $0 \le j \le m-1$. The order is established by the following rule: x_0 is the most significant position and x_{m-1} is the least significant position and the carry over in position j is modulo n_j, where n_j is the order of G_j. In that case, by (3)

$$L^{(n)}(G) \approx \frac{L(G)}{n} = \frac{N}{n} \sum_{j=0}^{m-1} \frac{L^{(n)}(G_j)}{n_j}. \qquad (4)$$

However, for the number of processors n>1, the data transfers among the processors are needed (at least in one of the algorithm steps). Therefore, the total time $T^{(n)}(G)$ needed for spectrum calculation over G with n processors is:

$$T^{(n)}(G) = L^{(n)}(G).\, t_a + M^{(n)}(G).\, t_c, \qquad (5)$$

where t_a is the time needed for one addition/subtraction, t_c is the time needed for one data transfer, and $M^{(n)}(G)$ is the total number of transfers needed for spectrum calculation. The value of $M^{(n)}(G)$ is not only a function of n and of the group structure for a fixed N; it depends also on the type of the processor communication network. We will consider two "extreme" communication networks: the taxi and unibus connections. In a taxi network, each processor is allowed to communicate to any other processor during the communication time t_c. The network requires a large number of links and $M^{(n)}(G)$ is, of course, minimized. In a unibus network all processors communicate through a single bus. Hence during the communication time t_c only one data transfer is allowed between two selected processors. We assume that:

(i) Each processor has local memory but there is no global shared memory.

(ii) There are no idle processors at any step $j \in \{0,1,...,m-1\}$ of the algorithm which means that $n.n_j$ divides N for all j's.

(iii) The ordered input data f(0), f(1),...,f(N-1) are entered into local memories of n processors in blocks: the first N/n elements are given to the 1st processor, the next N/n elements to the 2nd, etc.

(iv) At any stage j of the algorithm, any set of n_j elements is delivered to a single processor, $j \in \{0,1,..., m-1\}$. Even though that requirement is not necessary for executing the algorithm, it maybe shown [1] that its violation results in drastic increase in both the number of transfers and the number of total computations.

Under these assumptions, (5) was evaluated analytically for an arbitrary group in [1] and the decrease in execution time was calculated due to use of n processors as compared to a single processor. The relative speed-up due to n processors depends on the ratio t_c/t_a which characterizes the speed of the bus. It was shown in [1] that the bus must be sufficiently fast ($t_c << t_a$) to have an affect on the increase in the number of processors. For example, for $G=Q_2 \times Q_2$ and for a unibus connection, for k=1.5 doubling the number of processors (from 4 to 8) results in only about 18% of overall execution time decrease. Similar results were obtained for any other group.

3. PIPELINED IMPLEMENTATION OF FOURIER TRANSFORM

Consider a pipelined implemention of the Fourier transform on $G = G_0 \times G_1 \times ... \times G_{m-1}$ by multiple processors. The algorithm (2) is implemented under the assumptions (i) - (iv). The pipeline is an array of $n=m \cdot N$ processors depicted below. The "width" of the array is N and N data points f(0),...f(N-1) are entered simultaneously from the left into the 1st stage of N processors that compute the Fourier Transform over G_0. The "length" of the array is m since there are m stages in the algorithm.

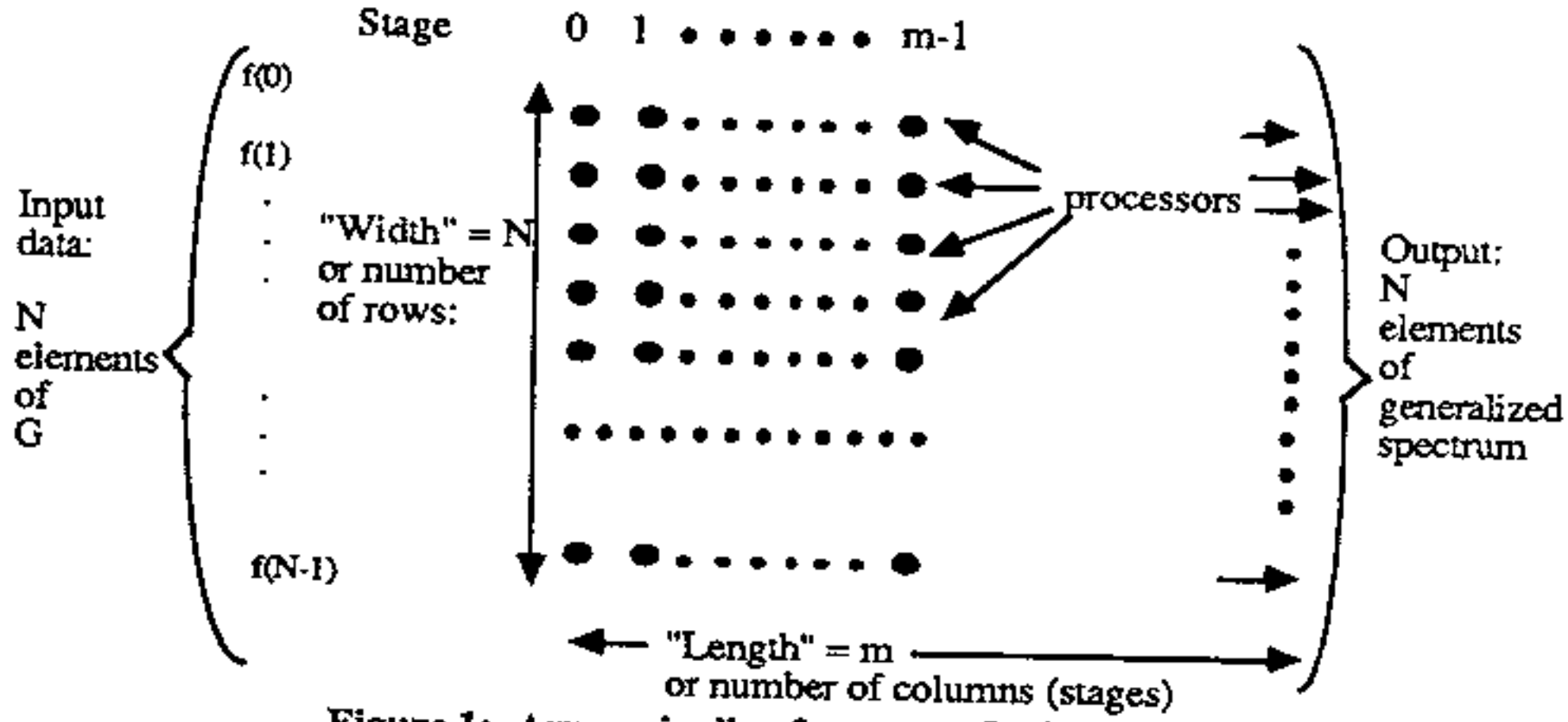

Figure 1: Array pipeline for group $G=G_0xG_1x...G_{m-1}$

At the stage j ($0 \le j \le m-1$), the intermediate spectrum on G_j is computed by the jth column of processors. Each processor in column j adds n_j numbers. It also may have to do some multiplications, depending on the G_j, see e.g. Table 1. Since $R_{\omega_0}(x) = 1$ for any $x \in G_j$, the number of multiplications is not larger than n_j-1 for each processor. It can therefore be shown that the total number of operations (both additions and multiplications) and of transfers satisfies

$$\sum_{j=0}^{m-1} (n_j-1) \le L^{(m \cdot N)}(G) \le 2 \sum_{j=0}^{m-1} (n_j-1), \quad M^{(m \cdot N)}(G) = N \sum_{j=0}^{m-1} n_j \qquad (6)$$

The delay of the whole array pipeline (i.e. time from the moment of input of N elements of data up to the moment of obtaining the N elements of the spectrum output) is given for the unibus connection as:

$$D_{unibus}^{(m \cdot N)}(G) = T_{unibus}^{(m \cdot N)}(G) \le \sum_{j=0}^{m-1} (n_j-1) \cdot (t_a + t_m) + N \sum_{j=1}^{m-1} n_j \cdot t_c, \qquad (7)$$

where t_a is one addition time, t_m is one multiplication time and t_c is the communication time. In case of a taxi connection, when all processors can communicate simultaneously, the number of transfers for stage j is n_j, $1 \le j \le m-1$ because each of the N processors at that stage transfers its output to n_j different processors of the next stage. No transfers are needed at stage 0. The delay is evaluated as follows:

$$D_{taxi}^{(mN)}(G) = T_{taxi}^{(mN)}(G) \le \sum_{j=0}^{m-1} (n_j-1) \cdot (t_a + t_m) + \left(\sum_{j=1}^{m-1} n_j \right) \cdot t_c . \qquad (8)$$

Note that the number of communication links among n processors in case of taxi connection is $N \sum_{j=1}^{m-1} n_j$. The evaluation of the number of operations and of the pipeline delay do not take into account the normalization of the spectrum which can be done after the last stage.

Example: $G = C_4 x C_4$, m=2, N=16, $n_0=n_1=4$. The number of processors n = 32. It follows by (1) that for C_4 we have: $\hat{f}(0) = f(0)+f(1)+f(2)+f(3)$; $\hat{f}(1) = f(0)+if(1)-f(2)-if(3)$; $\hat{f}(2) = f(0)-f(1)+f(2)-f(3)$; $\hat{f}(3) = f(0)-if(1)-f(2)+if(3)$, $i = \sqrt{-1}$. At Stage 0, the data f(0) to f(3) is broadcast to processors 0 to 3, data f(4) to f(7) is broadcast to processors 4 to 7, ... , data f(12) to f(15) is broadcast to processors 12 to 15. Also, at Stage 0 three additions/subtractions are performed simultaneously in each of the 16 processors (we do not take into account multiplication by $\pm i$) as follows:

in Processor 0: $\hat{f}(0) = f(0)+f(1)+f(2)+f(3)$; in Processor 1: $\hat{f}(1) = f(0)+if(1)-f(2)-if(3);...;$
in Processor 4: $\hat{f}(4) = f(4)+f(5)+f(6)+f(7);...;$ in Processor 15: $\hat{f}(15) = f(12)-if(13)-f(14)+if(15).$
At stage 1, the input data for each processor are chosen "skipping over $n_0=4$" that is, each of the 16 processors from stage 0 performs four transfers.

Processors at Stage 1	16	...	19	20	...	23	...	28		31
Transfers from Processors at Stage 0	0,4,8,12	...	0,4,812	1,5,9,15	...	1,5,9,13	...	3,7,11,15	...	3,7,11,15

In each Processor of Stage 1, the number of additions/substractions is also $n_1-1=3$. The total number of operations is $\sum_{j=0}^{1}(n_j-1) = 3\cdot 2 = 6$ and the total number of communications is $16\cdot 4 = 64 = N\cdot n$, for a unibus and $4=n_1$ for a taxi connection. The pipeline delay is $6t_a+64t_c$ for the unibus and $6t_a+4t_c$ for the taxi connection. The number of required communication links for the taxi connection is $N\cdot n_1=64$.

In conclusion, we note that the clock period of the array pipeline is:

$$\tau_{unibus} \le \max_j \{ (n_j-1)\cdot(t_a+t_m) + N\cdot n_j\cdot t_c \}, \tau_{taxi} \le \max_j \{ (n_j-1)\cdot(t_a+t_m) + n_j\cdot t_c \}. \quad (9)$$

To avoid having bottlenecks in the pipeline, it is reasonable to form G as a direct product of groups of the same type.

4. IMPLEMENTATION OF TWO-DIMENSIONAL DFT BY A SYSTOLIC ARRAY.

Let $G = C_N x C_N$. Then $R_{(i_1,i_0)}(l_1,l_0) = W_N^{i_1l_1+i_0l_1}$, $W_N = \exp(2\pi\sqrt{-1}/N)$ and by (1):

$$\hat{f}(i_1,i_0) = \sum_{l_1=0}^{N-1}\sum_{l_0=0}^{N-1} f(l_1,l_0)\, W_N^{i_1l_1+i_0l_0} = \sum_{l_1=0}^{N-1} P(l_1,i_0) W_N^{i_1l_1} \quad (10)$$

$$\text{where } P(l_1,i_0) = \sum_{l_0=0}^{N-1} f(l_1,l_0)\, W_N^{i_0l_0}. \quad (11)$$

It can be shown that algorithm (10), (11) can be written in the form of a FORTRAN - like program as follows.

Procedure 1.

DO 10 $k_1 = 0,N-1$

DO 10 $k_2 = 0,N-1$

$P(k_1, k_2, -1) = 0$ (initialize)

I
DO 20 $k_3 = 0, N-1$

$f(k_1, -1, k_3) = f(k_1, k_3)$

$f(k_1, k_2, k_3) = f(k_1, k_2-1, k_3)$

$P(k_1, k_2, k_3) = P(k_1, k_2, k_3-1) + f(k_1, k_2, k_3)\, W_N^{k_2k_3}$

20 CONTINUE

II
DO 30 $k_3 = N, 2N-1$

$\hat{f}(k_3-N, k_2, -1) = 0$ (initialize)

$P(k_1, k_2, k_3) = P(k_1, k_2, k_3-1)$

$\hat{f}(k_3-N, k_2, k_1) = \hat{f}(k_3-N, k_2, k_1-1) + P(k_1, k_2, k_3)\, W_N^{k_1k_3}$

30 CONTINUE

10 CONTINUE

This program may be obtained according to the accepted method [6] and it is suitable for systolic/semi-systolic implementation described below (see Figure 2). Note that in part I the index k_2 stands for the number of steps of accumulation of $f(\cdot, \cdot)$ in the right hand side of (10) whereas

in II the index k_1 has the same role for the spectrum $\hat{f}(\cdot, \cdot)$ in the left hand side of (10). The

dependence matrix for the Procedure 1 is

$$D = \begin{matrix} f & P & \hat{f} \\ \end{matrix} \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{matrix} k_1 \\ k_2 \\ k_3 \end{matrix}$$

The method developed in [4,6] is based on the idea of a nonsingular space-time mapping T of the index set (k_1,k_2,k_3) which maps a given problem to many possible systolic designs. That is, $\hat{D} = TD$; $(t\ s_0\ s_1)^T = T(k_1\ k_2\ k_3)^T$, $\hat{D}$ is the new dependence matrix and $(t\ s_0\ s_1)^T$ is the new index set, where t is the time coordinate with integer values ≥ 0; s_0, s_1 - the row and the column number, respectively, of the processing element $PE_{s_0 s_1}$, in the systolic array. One possible design for the (NxN) DFT requires N^2 $PE_{s_0 s_1}$; f moves through the array by s_0 ("north to south"), P accumulates in the PEs during N array clocks and $\hat{f}$ moves by s_1 ("west to east").

during N array clocks and $\hat{f}$ moves by s_1 ("west to east"). And for this design we have

$$\hat{D} = \begin{matrix} f & P & \hat{f} \\ \end{matrix} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{matrix} t \\ s_0 \\ s_1 \end{matrix} ; \qquad T = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} ; \ t = k_1+k_2+k_3 ; \quad s_0 = k_2 ; \ s_1 = k_1. \qquad (12)$$

Figure 2 illustrates the performance of the array for N = 2.

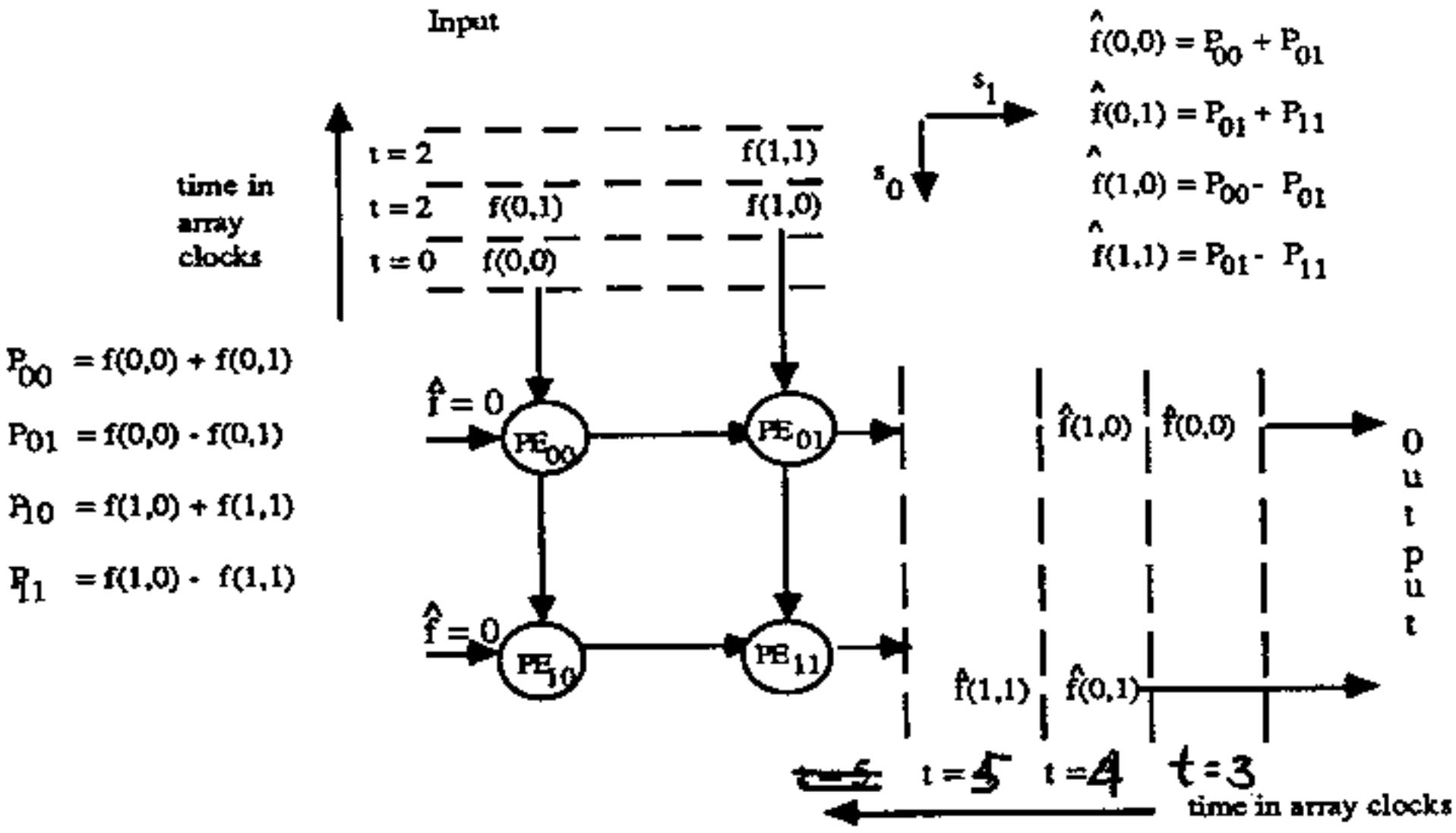

Figure 2: Systolic array of 4 PEs for 2x2 DFT

Each PE of the systolic array is a "double" multiply-add cell that calculates in parallel P – and $\hat{f}$ - values. The $PE_{s_0 s_1}$ is activated in the beginning of the array operation on array clock number $(s_0 + s_1)$ (the first clock is referred to as clock 0). During the first N active clocks, the PE performs only one multiply-add operation accumulating value of $P_{s_1 s_0}$. After that, the calculated value of $P_{s_1 s_0}$ is stored in the internal storage register during next N clocks; on each one of these clocks, the stored value of $P_{s_1 s_0}$ is used in multiply-add operations generating the values of the spectrum elements. In other words, after the first N active clocks, two operations occur in parallel in each PE: multiply-add operation with input data f and multiply-add operation with the stored value of $P_{s_1 s_0}$, to calculate in N steps the values of the spectrum. Figure 3 shows the general structure of the PE for the systolic array described above. The PE has the built-in counter, to calculate "the local time" modulo N, that is, to control the proper timing of the values $P_{s_1 s_0}$ to be accumulated, stored, or changed to a new value. Note that instead of the multipliers operating with the powers of W_N, the CORDIC rotation modules [8] can be used since the multiplications performed are rotations in the comlex plane. The CORDIC module uses only addition/subtraction and shift operations; having the same throughput and complexity as the complex multiplier, it occupies less space in a silicon chip. However since a standard CORDIC module performs rotations in the range of (-90°,90°) only, additional hardware would be needed for the rotation angles outside this range. The hardware complexity of one PE (in terms of the equivalent number of two-input gates) for the general case of N^2 DFT ($N = 2^n$ and the data lines

have 2r bits) can be evaluated as:

$$L_{PE} = L_{PE}(n,r) = 6n^2 - 3n + 2^{n+1}(n-1) + 4r(2^n-1) + 48r^2 + 100r + 294\left\lceil\frac{r}{2}\right\rceil + 158\left\lceil\frac{n}{4}\right\rceil,\ n \geq 4 \quad (13)$$

The total complexity of the array of N^2 PE's is $L(N^2) = 2^{2n} L_{PE}$; the clock of the array is $C=(416 + 3n)\cdot t_g$, where t_g is the average delay of one gate; the total hardware x time complexity is $H = L(N^2) \times C \times N$. Note that the number of I/O pins of one PE is 4n + 12r for 2r - bit data lines, where $n = \log_2 N$. Table 3 shows these complexity parameters for r=8 (16-bit data lines).

n	N^2	L_{PE}	L	C	H	Number of pins per PE (data lines only)
4	256	5866	1.50×10^6	428	1.03×10^{10}	112
5	1024	6747	6.91×10^6	431	9.52×10^{10}	116
6	4096	8218	3.36×10^7	434	9.35×10^{11}	120

Table 4: Complexity of Systolic Arrays for NxN DFT, $n=\log_2 N$

Note that the throughput of such an array is one spectrum set (N^2 values) per N array clocks.

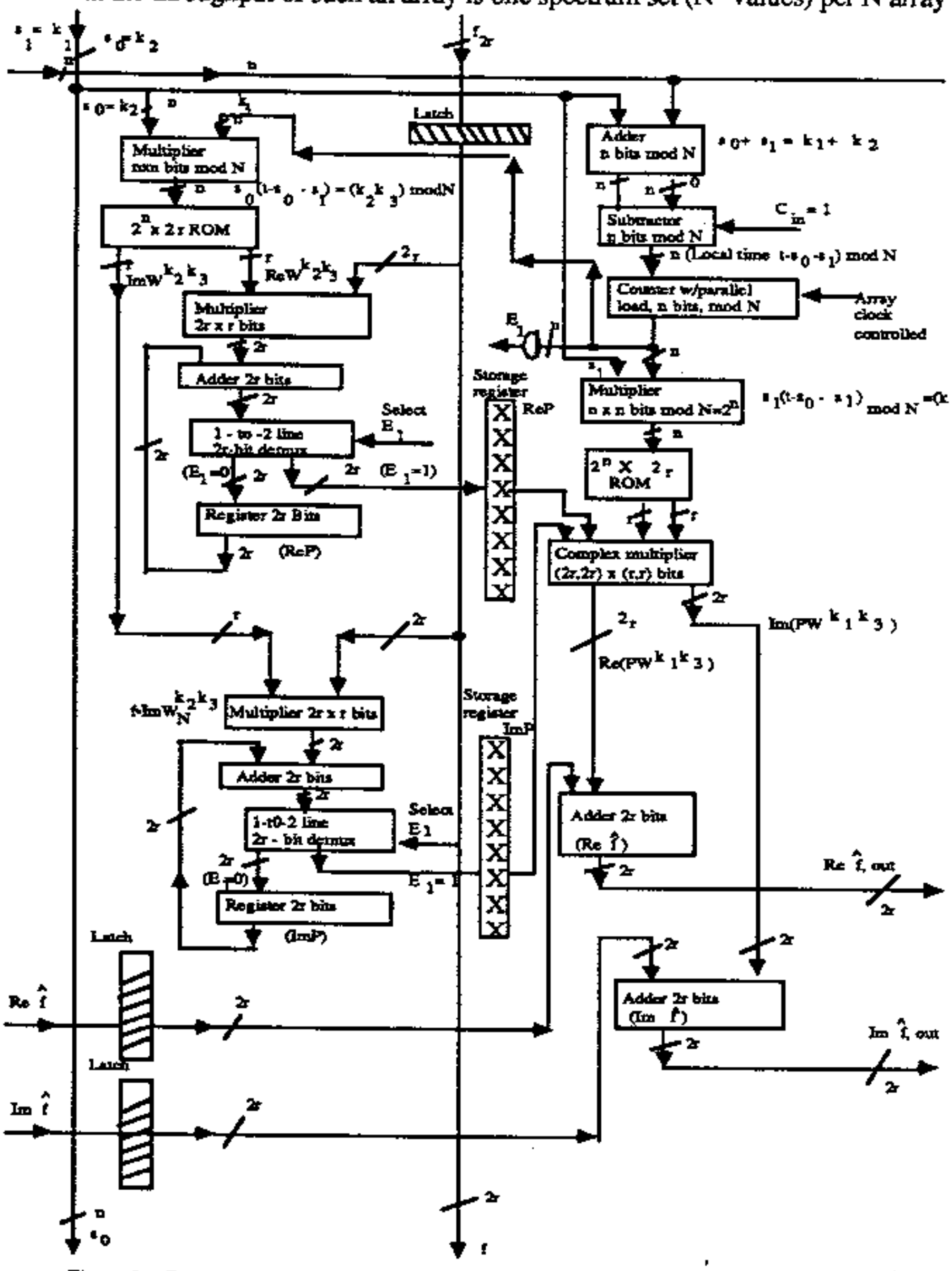

Figure 3: General structure of a PE for NxN systolic array (f moves by s_0, P accumulates, $\hat{f}$ moves by s_1)

Finally we outline our approach to the adaptation of the systolic array to any given group G. This can be achieved by adopting the PEs. We will illustrate our approach by a design of a systolic array which can be adapted to any group $G=G_1 \times G_2$, where $G_1, G_2 \in \{C_1, C_2, C_{2^2}, \ldots, C_{2^{k-1}}\}$ and $k=2^\alpha-1$. The design for a processing element is given in Figure 4.

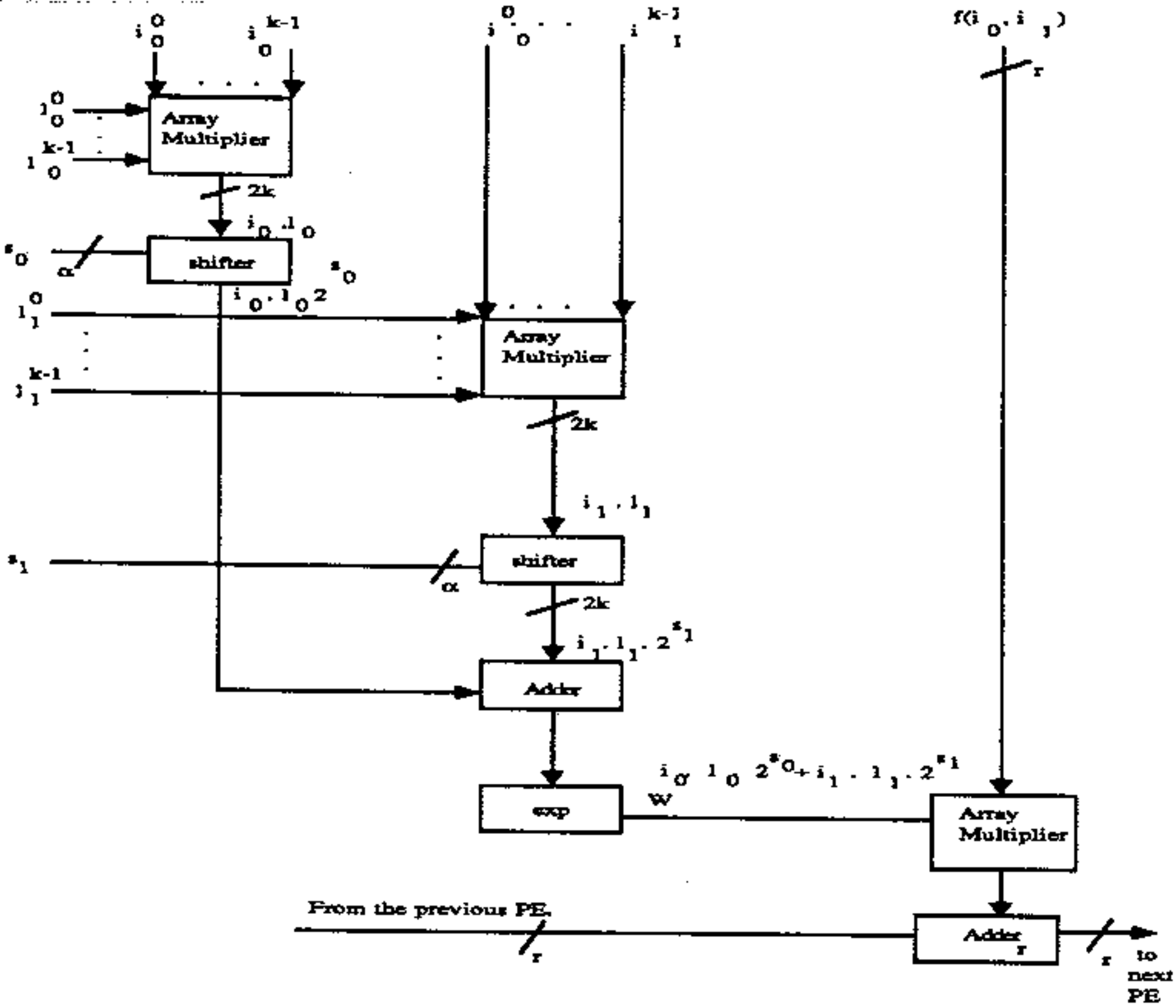

Figure 4: Adaptive Processing Element, $G=G_1 \times G_2$.

For this design $i = (i_0,i_1)$, $i_0 \in G_0$, $i_1 \in G_1$, $l = (l_0,l_1)$, $l_0 \in G_0$, $l_1 \in G_1$, $i_0 = (i_0^0,\ldots,i_0^{k-1})$, $i_1 = (i_1^0,\ldots,i_1^{k-1})$, $l_0 = (l_0^0,\ldots,l_0^{k-1})$, $l_1 = (l_1^0,\ldots,l_1^{k-1})$, $i_0^s, i_1^s, l_0^s, l_1^s \in \{0,1\}$, $W = \exp(2\pi \cdot 2^{-k+1}\sqrt{-1})$, and a shifter with the left input p computes a product by 2^p. Total number of input/output lines for a processing element is $4k+\log_2(k+1) + 3r$ ($\alpha=\log_2(k+1)$, r- is the number of bits in data). To adapt this processing element to $G = C_2a \times C_2b$, where $a,b \in \{0,1,\ldots,k-1\}$, one should set $i_0^0=\ldots=i_0^{k-1-a}=l_0^0=\ldots=l_0^{k-a-1}=0$, $i_1^0=\ldots=i_1^{k-b-1}=l_1^0=\ldots=l_1^{k-b-1}=0$ and $s_0=k-a-1$, $s_1=k-b-1$.

A similar adaptation technique can be used in case when G is product of more than two groups.

5. References

1. T. Roziner, M. Karpovsky and E. A. Trachtenberg, "Fast Fourier Transforms over Finite Groups by Multiprocessor Systems," IEEE Trans. on ASSP, (To Appear, September, 1989).

2. E. A. Trachtenberg, "Fault Tolerant Computing and Reliable Communication: A Unified Approach," Information and Computation, 79, 3,pp. 257-279, December, 1988.

3. E. A. Trachtenberg and M. G. Karpovsky, "Optimal Varying Dyadic Structure Models of Time Invariant Systems," Proc. 1988 Int'l Symp. on Circuits and Systems, pp. 1111-1114, Espoo, Findland, June, 1988.

4. D.I. Moldovan, J. A. B. Fortes. "Partitioning and Mapping Algorithms into Fixed Size systolic arrays", IEEE Trans. Comp., vol C-35, 1, pp. 1-13, Jan. 1986.

5. W. L. Miranker and A. Winkler. "Space-time Representation of Computational Structures", Computing, vol. 32, pp.sssdd 93-114, 1984.

6. D. I. Moldovan. "On the Design of Algorithms for VLSI Systolic Arrays". proc. IEEE, vol. 71, 1, pp. 113-120, Jan. 1983.

7. E. A.Trachtenberg, M.G. Karpovsky, "Fourier Transforms Over Finite Groups for Error Detection and Error Correction in Computation Channels". Information and Control, 40, 3, pp. 335-358, March 1979.

8. A. M. Despain. "Fourier Transform Computers Using CORDIC Iterations". IEEE Trans. Comp., vol. C-23, 10, Oct. 1974, pp. 993-1001.