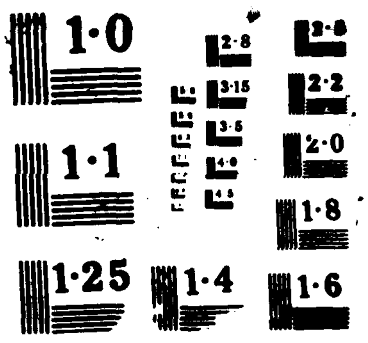

1·0
2·8
2·5
3·15
2·2
3·5
2·0
4·0
1·1
4·5
1·8
1·25
1·4
1·6

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

2

1. REPORT NUMBER ARO 20090.22-EL	2. GOVT ACCESSION NO. N/A	3. RECIPIENT'S CATALOG NUMBER N/A
4. TITLE (and Subtitle) EXPANDED VLSI ARCHITECTURES		5. TYPE OF REPORT & PERIOD COVERED Technical
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Robert Michael Owens, Mary Jane Irwin		8. CONTRACT OR GRANT NUMBER(s) DAAG29-83-K-0126
9. PERFORMING ORGANIZATION NAME AND ADDRESS Department of Computer Science, The Pennsylvania State University, University Park, PA 16802		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS U. S. Army Research Office, Post Office Box 12211, Research Triangle Park, NC 27709		12. REPORT DATE September 1984
		13. NUMBER OF PAGES 20 pages
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

NA

18. SUPPLEMENTARY NOTES

The view, opinions, and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This paper describes a non von Neumann architecture which also conforms to the requirements for VLSI implementation - expanded VLSI architectures. In expanded VLSI machines, more than O(n) processors are used to solve O(n) problems, where inexpensive (in terms of silicon area), fast processors have been added to simplify the processor interconnections. Expanded architectures are constructed by deriving algorithms which trade many of one type of operation like addition, for regularity of data movement. An expanded architecture for —

ARO 20090.22-EL

EXPANDED VLSI ARCHITECTURES

Robert Michael Owens
Mary Jane Irwin
Department of Computer Science
The Pennsylvania State University

CS-84-14 Sept. 1984

September 1984

This work is supported in part of the Office of Naval Research under Contract N00014-80-C-0517 and the Army Research Office under Contract DAAG29-83-K-0126.

87 9 9 266

EXPANDED VLSI ARCHITECTURES

ABSTRACT

This paper describes a non von Neumann architecture which also conforms to the requirements for VLSI implementation - *expanded VLSI architectures*. In expanded VLSI machines, more than $O(n)$ processors are used to solve $O(n)$ problems, where inexpensive (in terms of silicon area), fast processors have been added to simplify the processor interconnections. Expanded architectures are constructed by deriving algorithms which trade many of one type of operation, like addition, for regularity of data movement. An expanded architecture for the Discrete Fourier Transform problem is derived. Three operational components are described, each of which can be implemented in one (or a few) VLSI chips. Optimal measures for silicon area and processing time are primary concerns.

INTRODUCTION

Research in computer architecture in the last decade has been driven largely by the motivation to overcome the "von Neumann bottleneck." Many computer architects believe that the next generation of computers will be based on non-von Neumann architecture capable of exploiting VLSI processors [Tre82]. Some of these new architecture designs tailor the number of processors to match the size of the problem; e.g., $O(n)$ processors to solve $O(n)$ problems. Our paper describes a VLSI architecture which has many more than $O(n)$ processors where processor have been added to simplify the processor interconnection requirements. With an expanded number of processors, the goal is to design with lots of inexpensive (in terms of silicon area), fast processors which have simple interconnection requirements.

New advances in VLSI technology have opened up new horizons for the design of fast, reliable, and efficient special-purpose processors. Since it is now possible to lay hundreds of thousands of transistors on a single chip, efficient algorithms for very complex problems can be implemented in VLSI. These designs have to satisfy several general conditions, such as regular layout, simple control, modularity, and simple communication. The main criteria used by researchers to evaluate a VLSI design are the area of the chip and the time required to solve an instance of the problem. Theoretical lower bounds on the area and time have been obtained, together with new designs for many problems which meet these bounds. However, many of these designs seem to be difficult to implement on single chips because of the required complexity of the processing components. For example, if we consider the VLSI design outlined in [Pre83], n multipliers are required on a chip to compute the Discrete Fourier Transform of n points. However, with current technology, it is not possible to lay out more than a few multipliers on a single chip. Hence, the large number of chips which would be needed to solve any but the most trivially sized problem make the design impractical. We consider in this paper a VLSI design which is currently being implemented. Each processing component can be implemented in one (or a few) chips.

We introduce in the next section a sample expanded architecture. Using similar derivation techniques, we hope to be able to construct a family of expanded, very high speed VLSI arithmetic processors. Expanded architectures are based on algorithms which trade many of one type of operation (like addition) for regularity of data movement and speed. Since a one digit online adder is actually quite small, this trade off is not only feasible, but from the point of view of being able to construct hardware which is extremely fast, regular, and has minimal control, is very desirable. Expanded architectures are interesting in that fundamental data movement restrictions (like the amount of information which can be transferred across the boundary of an area) limits performance and nothing else. That is, even though the architecture seems to "throw hardware" at the problem, it does achieve the *Area* × *Time*^{2} lower bound of n^2 which holds for any VLSI processor [ChMo81]. Each component of the expanded VLSI machine can be implemented in one (or a few) chips.

THE EXPANDED VLSI ARCHITECTURE

To show the merits of our expanded architecture, we will consider the Discrete Fourier Transform (*DFT*) problem. The *DFT* of the n element vector $\vec{X}$ is the n element vector $\vec{Y}$ which satisfies $\vec{Y} = \mathbf{W}\vec{X}$, where $\mathbf{W}$ is a $n \times n$ matrix such that $\mathbf{W}_{ij} = w^{ij}$ and w is the n'*th* primitive root of unity. By reformulating the *DFT* equation into either of the forms below, an expanded architecture for the *DFT* can be derived in a straightforward manner.

$$\mathbf{Y} = \mathbf{S}_2 \mathbf{D}_2 \mathbf{C}_2 \left[\mathbf{S}_1 \mathbf{D}_1 \mathbf{C}_1 \mathbf{X} \right]^T \quad \text{(Prime Factors)}$$

or

$$Y = S_2 D_2 C_2 \left[A S_1 D_1 C_1 X \right]^T \quad \text{(Winograd)}$$

where D_1, D_2, A are diagonal arrays, S_1, S_2, C_1, and C_2 are arrays whose elements are either -1, 0, or 1, and X and Y are arrays whose elements in row major form are the elements of $\hat{X}$ and $\hat{Y}$ using Good's algorithm with prime factors reduction [KoPa77] or Winograd's small n algorithm [Win78]. For more background the reader is referred to [Ja84].

The expanded *DFT* machine is designed by constructing components which can compute SX (the summation component), DX (the scaling component), and X^T (the transpose component) and which can be interconnected in an efficient manner. The expanded machine structure to solve the *DFT* using the prime factors reduction is obtained by interconnecting components of these three types as indicated in Figure 1.

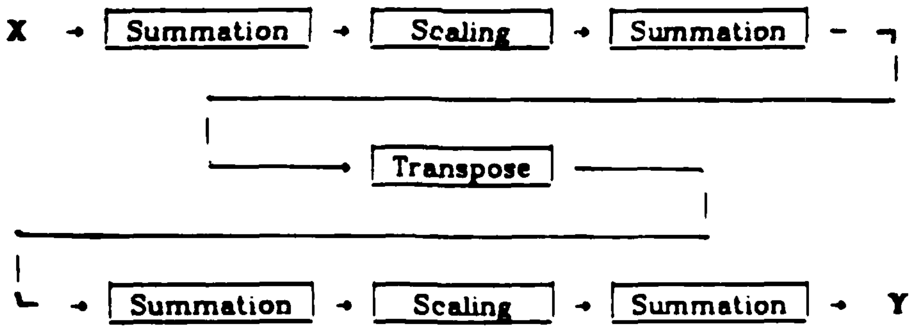


Figure 1. Expanded *DFT* Machine

To aid in the construction of the VLSI hardware, each of the components should have, where ever possible, the following properties.

1) The execution rate of each component should be constant and therefore independent of the mathematical and physical characteristics of its size. As we will see later, this rate can be made quite high.

2) Each component should be constructed by interconnecting in a regular way smaller elements of a few different types. The size of the

elements should be small and independent of the size of the problem. Furthermore, given the formal description, the design of a component should be straightforward.

3) The size of each component should be realistic and should satisfy VLSI circuit density and size constraints.

4) The logical and the physical input/output characteristics of the various components should be compatible so as to allow one component to be connected to another in a straightforward manner. Furthermore, the number of interconnections between components should be realistic.

The Summation Component

The mathematical operation performed by the summation component is given by $\mathbf{Z} = \mathbf{S}\mathbf{X}$ where

$$\mathbf{S} = \left[S_{i,j},\ 0 \le i < n_0,\ 0 \le j < n_1 \right] ,$$

$$\mathbf{X} = \left[X_{i,j},\ 0 \le i < n_1,\ 0 \le j < n_2 \right] ,$$

and

$$\mathbf{Z} = \left[Z_{i,j},\ 0 \le i < n_0,\ 0 \le j < n_2 \right] .$$

For the *DFT* problem, the elements of **S** are predefined and are either −1, 0, or 1. We will make use of this fact by building them into the summation component itself rather than supplying them to the component as the operation is performed. Being able to tailor the component in this manner will decrease the complexity and, hence, the size of the hardware needed to perform the calculation. The summation component will be constructed by interconnecting smaller elements of only three different types (addition, subtraction, and delay), which share the same rectangular shape. The area of the summation component is $O(n_1 n_0)$ and depends only on n_1, n_0, and the size of the elements. The area of each elements depends only on the precision p of the integer values being either added or subtracted. Each of these elements can be represented as shown in Figure 2.

$$\begin{tikzcd}
 & Z_{in} \arrow[d] & \\
X_{in} \arrow[r] & \boxed{S} \arrow[r] \arrow[d] & X_{out} \\
 & Z_{out} &
\end{tikzcd}$$

where

$X_{out} = X_{in}$ and

$Z_{out} = Z_{in} + X_{in}$ if $S_{i,j} = 1$

$Z_{out} = Z_{in} - X_{in}$ if $S_{i,j} = -1$

$Z_{out} = Z_{in}$ if $S_{i,j} = 0$

Figure 2. Summation Elements

These elements are interconnected as indicated in Figure 3 to form a summation component.

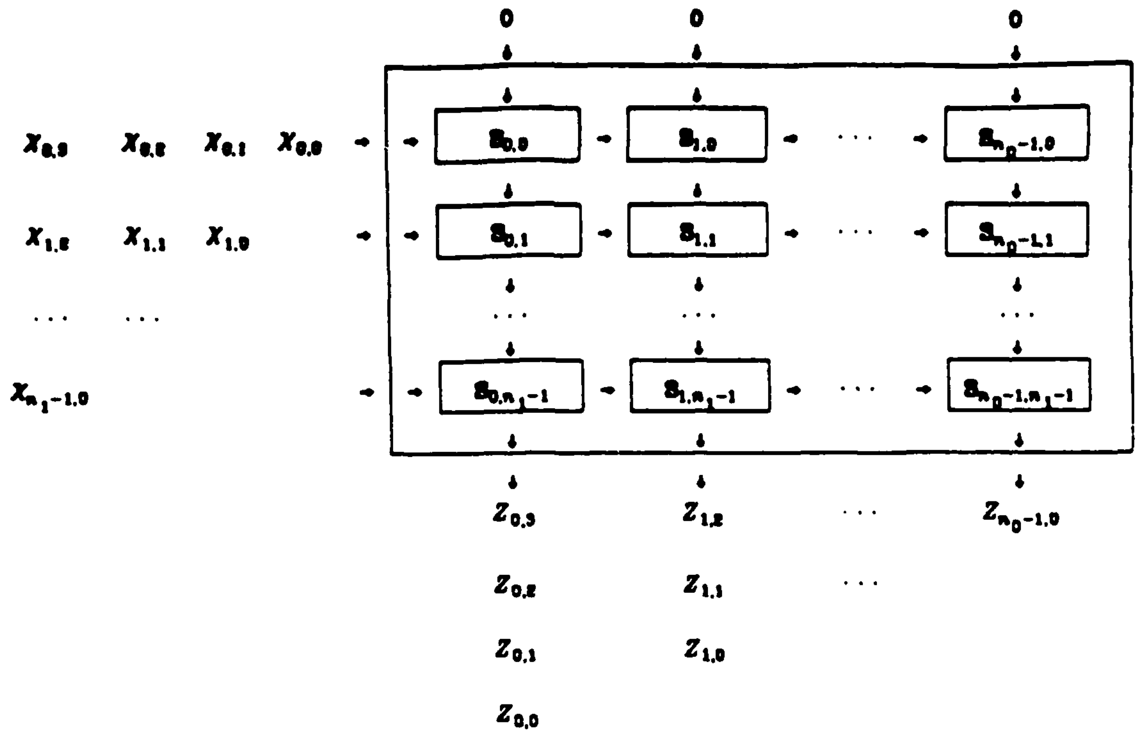

0
0
0
$X_{0,3}$ $X_{0,2}$ $X_{0,1}$ $X_{0,0}$
$X_{1,2}$ $X_{1,1}$ $X_{1,0}$
$X_{n_1-1,0}$
$S_{0,0}$
$S_{1,0}$
$S_{n_0-1,0}$
$S_{0,1}$
$S_{1,1}$
$S_{n_0-1,1}$
S_{0,n_1-1}
S_{1,n_1-1}
S_{n_0-1,n_1-1}
$Z_{0,3}$ $Z_{0,2}$ $Z_{0,1}$ $Z_{0,0}$
$Z_{1,2}$ $Z_{1,1}$ $Z_{1,0}$
$Z_{n_0-1,0}$

Figure 3. Summation Component

Element $S_{i,j}$, $0 \le i < n_0$, $0 \le j < n_1$, is either an addition element ($S_{i,j} = 1$) , a subtraction element ($S_{i,j} = -1$), or a delay element ($S_{i,j} = 0$). Note that the elements of the input array and the result array are supplied and generated in an element skewed manner. Assuming addition takes unit time, the time required to compute SX is $2(n_1 - 1) + n_2$.

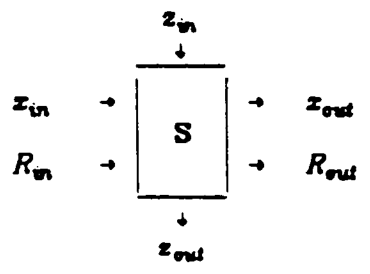
The principle difficulty in constructing the summation component in the manner just outlined is having to deal with the relatively large number of inputs and outputs, $2p\,(n_1 + n_0)$, and the relatively large physical size of a high speed, full precision adder. This difficulty can be illustrated by observing that our goal is to build *DFT* hardware which can handle problems of size at least 1024 sixteen bit numbers. This does not seem to be feasible with current or even foreseeable VLSI technology. Furthermore, the required size of the perimeter, $O(p\,(n_1 + n_0))$, (as defined by the input/output requirements of the summation component) does not fit well with its area requirements.

$O(p^2 n_1 n_0)$. This comment is based on the observation that if the perimeter requirement of a component exceeds the square root of the area requirement, then either area or time is probably wasted. For small precision, this will not be problem. However, for large precision it will be.

The obvious first choice to get around this difficulty would be to share (via time multiplexing) some smaller number of input and output interconnections and to share a fewer number of integer adders and subtractors. However, this approach would suffer not only the usual problem of having to map a larger problem to the available hardware but also a given element could no longer be tailored for the calculation it performs. Hence, we would would lose the size advantage we obtained by tailoring a given element to a given task (addition, subtraction, and delay). We would also lose the advantage of not having to supply the elements of S. To avoid the above problems, we propose to use digit online processing [TrEr77, IrOw83] where each value is transmitted in a digit serial manner. Digit online algorithms for arithmetic functions generate the jth digit of the result after having been supplied with up to only the first $(j + k)^{th}$ digits of the input operands, where k is a small integer corresponding to the digit online delay. By using digit online arithmetic, we are able to reduce the perimeter required from $O(p(n_1 + n_0))$ to $O(n_1 + n_0)$ and the area required from $O(p^2 n_1 n_0)$ to $O(n_1 n_c)$. Note that neither the perimeter nor the area required depend on p. The only disadvantage of using digit online arithmetic is that the digit online version of the summation component must be clocked p times for each time the word parallel version is clocked once. This may not be as much of a problem as it first appears, since the digit online version may be clocked at a faster rate than the word parallel version because its basic elements are simpler (bit adders versus word adders) and, therefore, faster [Gur84].

We could at this point elect either left directed digit online processing (working from least significant digit to most significant digit) or right directed digit online processing (working from most significant digit to least significant digit), as integer addition and subtraction can be done either way. If the construction of the summation component was the only consideration, we would elect to use left directed (conventional serial addition) as its digit online delay is lower (k = 0) and the complexity of the hardware necessary to perform the required calculations is smaller. However, as we will see, the scaling component requirements will force us to use right directed techniques. Algorithms for right directed, digit online processing have been developed for floating point comparison exchange, addition, subtraction, and multiplication (k = 1); for floating point division, square root, and the trigonometric functions of sine and cosine (k = 3); and for fixed point logarithm and exponentiation (k = 1) [Owe 80, Owe81].

The digit online summation element for $S_{ij} = 1$ can be represented as shown in Figure 4.



where

$$R_{out} = R_{in}$$

$$x_{out} = x_{in}$$

$h_j = x_{in} + z_{in} - b\, g_j$ where

g_j is chosen so that $-(b-1) \leq h_j \leq (b-1)$ and

$$z_{out} = h_{j-1} + g_j$$

Figure 4. Summation Digit Online Element

The flag input R_{in} is used to notify the element that the first digit of a operand will be supplied next. The digit values h_{j-1} and g_j are stored in digit registers in the summation element. The reset flag resets the value of $g_{p-1} = 0$ to complete the previous operation and resets the value of $h_0 = 0$ to start the next operation.

The number system we have decided to use in our implementation is octal with a maximally redundant digit set [Atk75] of $D_{max} = \{-7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7\}$ instead of the conventional digit set $D_{con} = \{0, 1, 2, 3, 4, 5, 6, 7\}$ However, the digits of predefined values of S_{ij} will be restricted to the minimally redundant set $D_{min} = \{-4, -3, -2, -1, 0, 1, 2, 3, 4\}$. In this number system, three representations for 28_{10} are 0034_8, $004\bar{1}_8$, and $01\bar{4}\bar{4}_8$ where $\bar{4} = -4$. An example of the right directed, digit online addition operation used in the summation element is shown below where $X_{in} = 3344_8$ and $Z_{in} = 3434_8$. Note that this example would cause a full length ripple carry in a conventional adder.

clock	x_{in}	z_{in}	g_j	h_j	z_{out}
R	•	•	0	0	•
1	3	3	0	6	0
2	3	4	1	-1	7
3	4	3	1	-1	0
4	4	4	1	0	0
R	•	•	0	0	0

The result generated by the example is $Z_{out} = 7000_8$ Note that there is a digit online delay of *one*. The functional description of the digit online subtraction and delay elements follows from the description of the addition element.

The digit online summation elements are interconnected as indicated in Figure 5 to form a summation component.

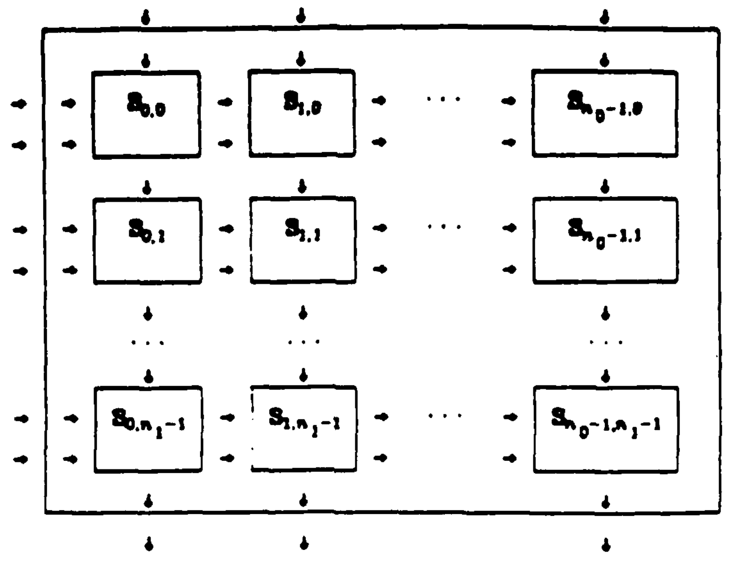


Figure 5. Digit Online Summation Component

It should be noted that the manner in which the input data is supplied to the summation component has changed because of the use of digit online elements. Figure 6 illustrates how the input data is now supplied to summation component.

$$\begin{array}{ccccccccccc}
(s_{0,1})_3 & (s_{0,1})_2 & (s_{0,1})_1 & (s_{0,1})_0 & (s_{0,0})_{p-1} & \cdots & (s_{0,0})_4 & (s_{0,0})_3 & (s_{0,0})_2 & (s_{0,0})_1 & (s_{0,0})_0 & \rightarrow \\
(s_{1,1})_1 & (s_{1,1})_0 & (s_{1,0})_{p-1} & \cdots & (s_{1,0})_4 & (s_{1,0})_3 & (s_{1,0})_2 & (s_{1,0})_1 & (s_{1,0})_0 & & & \rightarrow \\
(s_{2,0})_{p-1} & \cdots & (s_{2,0})_4 & (s_{2,0})_3 & (s_{2,0})_2 & (s_{2,0})_1 & (s_{2,0})_0 & & & & & \rightarrow \\
 & \cdots & \cdots & \cdots & \cdots & & & & & & & \\
(s_{n_1-1,0})_2 & (s_{n_1-1,0})_1 & (s_{n_1-1,0})_0 & & & & & & & & & \rightarrow
\end{array}$$

Figure 6. Digit Skewed Data

Note, that the elements of the input array and the result array are now supplied and generated in a digit skewed as well as element skewed manner.

The Scaling Component

The next component we will consider is the scaling component. The mathematical operation performed by this component is given by $\mathbf{Z} = \mathbf{D}\mathbf{X}$ where

$$\mathbf{D} = \left[D_{i,j},\ 0 \le i < n_0,\ 0 \le j < n_1 \right] \text{ and } i \ne j \Rightarrow D_{i,j} = 0$$

$$\mathbf{X} = \left[X_{i,j},\ 0 \le i < n_1,\ 0 \le j < n_2 \right] .$$

and

$$\mathbf{Z} = \left[Z_{i,j},\ 0 \le i < n_0,\ 0 \le j < n_2 \right] .$$

For the *DFT* problem, the elements along the major diagonal of **D** are predefined integer constants. As with the summation component, we will make use of this fact by building them into the scaling component itself rather than supplying them to the component as the operation is performed. This allows us to reduce the circuit complexity, the number of input/output connections, and, hence, the size of the component. The scaling component will be constructed by interconnecting only one type of element. The area of the scaling component is $O(n_0)$ and depends only n_0 and the size of the elements. The area of each element depends only on the precision p of the integer values being scaled. This element can be represented as shown in Figure 7.

$$\begin{tikzcd} X_{in} \arrow[r] & \boxed{D(Y)} \arrow[r] & Z_{out} \end{tikzcd}$$

where

$$Z_{out} = Y X_{in}$$

Figure 7. Scaling Element

These elements are interconnected as indicated in Figure 8 to form a scaling component.

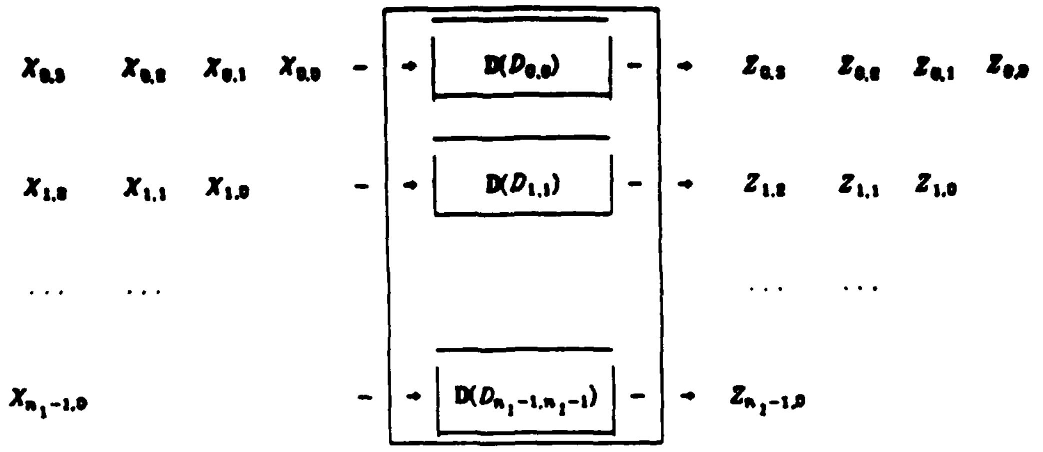
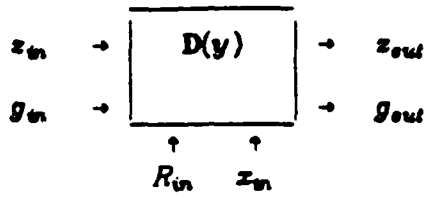

$X_{0,3}$ $X_{0,2}$ $X_{0,1}$ $X_{0,0}$
$D(D_{0,0})$
$Z_{0,3}$ $Z_{0,2}$ $Z_{0,1}$ $Z_{0,0}$
$X_{1,2}$ $X_{1,1}$ $X_{1,0}$
$D(D_{1,1})$
$Z_{1,2}$ $Z_{1,1}$ $Z_{1,0}$
$X_{n_1-1,0}$
$D(D_{n_1-1,n_2-1})$
$Z_{n_1-1,0}$

Figure 8. Scaling Component

Again because of the area difficulties discussed with respect to the summation component, we elect to use digit online arithmetic. This effect is even more dramatic in the scaling component because of the size of a full precision integer multiplier. However, while addition and subtraction can be done equally well left or right directed, multiplication is best done right directed. This occurs because of the following observations. The product of two p digit integer numbers is a $2p$ digit integer. In left directed processing the p digits of each input are supplied in a least significant digit to most significant digit order. More importantly , the $2p$ digits of the result are generated in a right to left order. The opposite holds for right directed arithmetic. Hence, if we use left directed arithmetic, we would have only the least p significant digits of the result, after supplying the p digits of each input. However, we are interested in the p most significant digits of the result. Hence, we could either clock the multiplier p more times or use a two stage multipler (the first part computes the p least significant digits which are, unfortunately, then discarded; the second part computes the p most significant digits). Two stages must be used so that we may keep up with the flow of input digits as they are supplied to the component. Neither of these two options is very palatable. However for right directed arithmetic, the p most significant digits of the result will be generated, after supplying the p digits of each input. These are the very digits we want. Hence, we have elected to use right directed arithmetic despite the modest increase in hardware costs.

Each of the digit online scaling elements can be represented as shown in Figure 9.



where

$$h_j = y\, x_{in} + z_{in} - b\, g_{out}$$

g_{out} is chosen such that $-(b-1) \leq h_j \leq (b-1)$ and

$$z_{out} = h_{j-1} + g_{in}$$

Figure 9. Scaling Digit Online Element

The flag input R_{in} is used to notify the element that the first digit of an operand will be supplied next. The single digit constant y and the digit value h_{j-1} are stored in digit registers in the scaling element. The reset flag resets the value $h_C = 0$ to start the next operation.

The digit online scaling elements are interconnected as indicated in Figure 10 to form a scaling component.

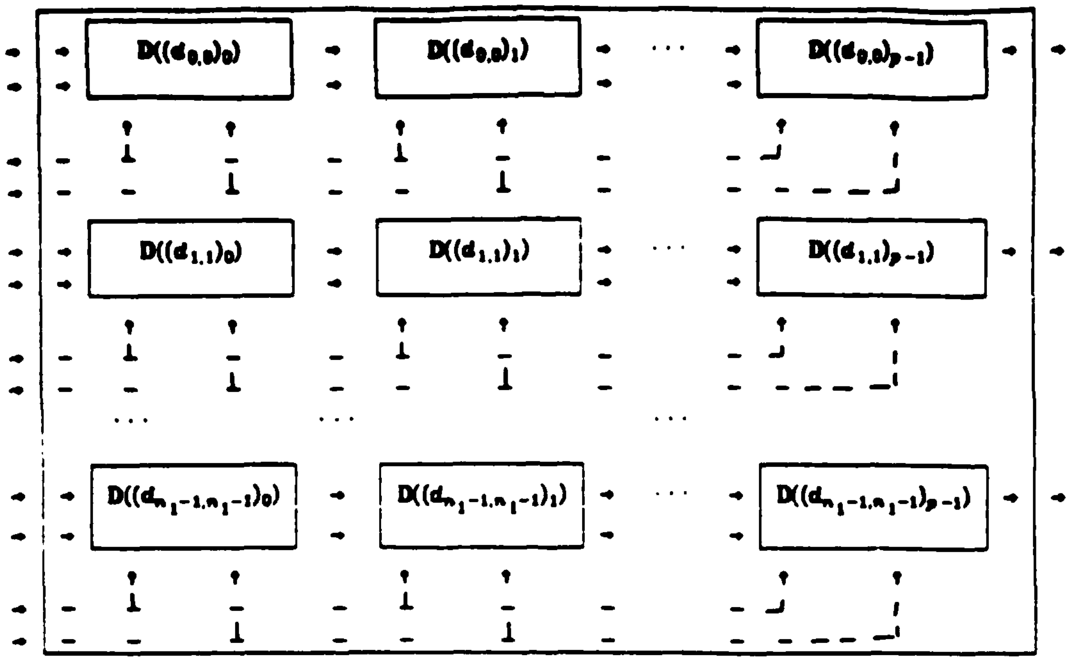


Figure 10. Digit Online Scaling Component

Note that each element holds one digit of the appropriate integer operand $D_{i,j}$. Thus, each element contains a digit multiplier to form $y\ x_{in}$, as well as a digit online adder. Like the summation component, it should be noted that the manner in which the input data is supplied to the scaling component has changed because of the use of digit online elements. However, the input/output requirements of the two are identical. Hence, the output of a summation component can be connected (with respect to logical considerations) directly to the input of a scaling component and vice versa.

The Transpose Component

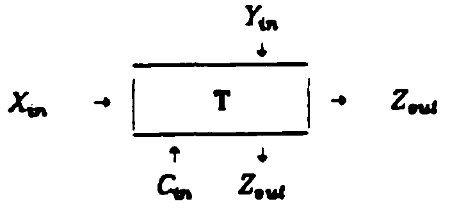
The last major component we will consider is the transpose component. Since this component performs no arithmetic operations on the data supplied to it, it is in some ways the simplest. However, since its data movement requirements are the most general, it is in other ways the most complicated. The mathematical operation performed by this component is $\mathbf{Z} = \mathbf{X}^T$, where

$$\mathbf{X} = \left\{ X_{i,j},\ 0 \le i < n_1,\ 0 \le j < n_0 \right\} .$$

and

$$\mathbf{Z} = \left\{ Z_{i,j},\ 0 \le i < n_0,\ 0 \le j < n_1 \right\} .$$

The transpose component will be constructed by interconnecting elements of only one type (storage) which have a rectangular shape. The area of the transpose component is $O(n_1 n_0)$ and depends only on n_1, n_0, and the size of the storage element. The area of the element depends on the precision p of the integer value being stored. Each of these elements can be represented as shown in Figure 11.



where

$$Z_{out} = \begin{cases} X_{in} & \text{if } C_{in} = 0 \\ Y_{in} & \text{otherwise} \end{cases}$$

Figure 11. Storage Element

These elements are interconnected as indicated in Figure 12 to form a transpose component.

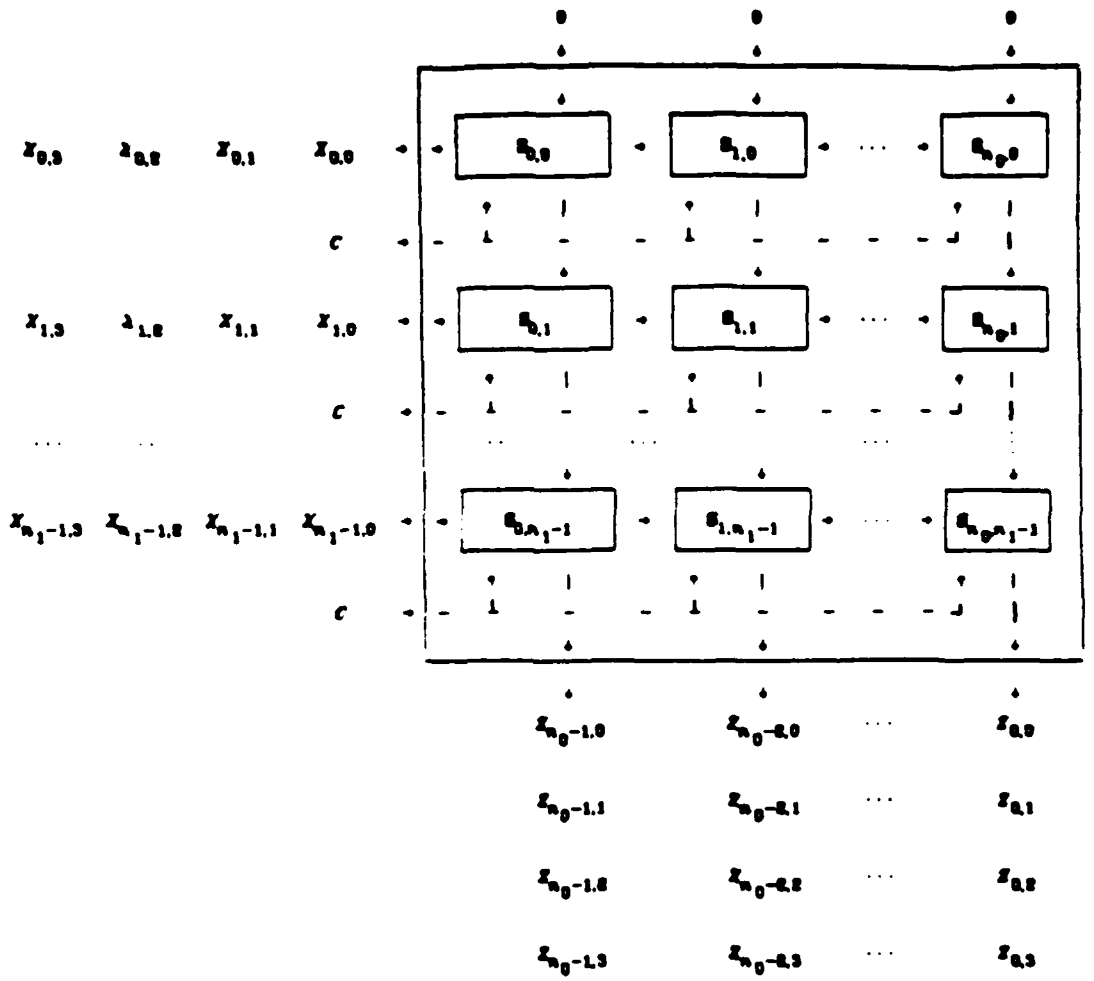

$x_{0,3}$ $x_{0,2}$ $x_{0,1}$ $x_{0,0}$
$x_{1,3}$ $x_{1,2}$ $x_{1,1}$ $x_{1,0}$
$x_{n_1-1,3}$ $x_{n_1-1,2}$ $x_{n_1-1,1}$ $x_{n_1-1,0}$
C
$s_{0,0}$ $s_{1,0}$ $s_{n_0,0}$
$s_{0,1}$ $s_{1,1}$ $s_{n_0,1}$
s_{0,n_1-1} s_{1,n_1-1} s_{n_0,n_1-1}
$z_{n_0-1,0}$ $z_{n_0-2,0}$ $z_{0,0}$
$z_{n_0-1,1}$ $z_{n_0-2,1}$ $z_{0,1}$
$z_{n_0-1,2}$ $z_{n_0-2,2}$ $z_{0,2}$
$z_{n_0-1,3}$ $z_{n_0-2,3}$ $z_{0,3}$

Figure 12. Transpose Component

By transferring X into the transpose component with $C_{in} = 0$ and transferring Z out with $C_{in} \neq 0$, the transpose can be performed. Note that elements of input array and the result array are not supplied and generated in an element skewed manner. Hence, the input/output characteristics of this component are not compatible with the other two components. This problem can be corrected by increasing the hardware area and cost, but a more desirable solution to this problem is needed.

CONCLUSIONS

An expanded VLSI architecture for solving the *DFT* was presented. Expanded architectures are constructed by deriving algorithms which trade many of one type of operation, like addition, for regularity of data movement. Three operational components were described, a summation component, a scaling component, and a transform component, each of which can be implemented in one (or a few) VLSI chips. Using the same expansion techniques, we are presently investigating other problem domains for which optimal expanded architectures can be derived. The basic components may have to be modified and augmented. However the goals of regularity of data movement, speed, and being able to implement each component in one (or a few) chips remain the same.

BIBLIOGRAPHY

Atk75 Atkins, D., "An Introduction to the Role of Redundancy in Computer Arithmetic," *Computer*, Vol. 8, No. 6, pp. 74-76, June 1975.

BoBr83 Bowen, B. A. and W. R. Brown, *VLSI Systems Design for Digital Signal Processing*, Volume 1, Prentice Hall, 1982.

ChMo81 Chazelle, B. and L. Monier, "A Model of Computation for VLSI with Related Complexity Results," *Proceeding of the 13th Annual Symposium on Theory of Computing*, ACM, pp. 318-325, May 1981.

Gur84 Gurney, D., "Investigation of Some Processors with Digit Serial I/O," M.S. Paper in progress, Department of Computer Science, Penn State University, August 1984.

IrOw83 Irwin, M. J. and R. M. Owens, "Fully Digit Online Networks," *IEEE Transactions on Computers* Vol. C-32, No. 4, pp. 402-406, April 1983.

Ja84 JaJa, J., "High-speed VLSI Networks for Computing the Discrete Fourier Transform," *Proceedings of the 1984 Conference on Advanced Research in VLSI*, pp. 11- 20, MIT, Boston, MA, January 1984.

KoPa77 Kolba, D. P. and I. W. Parks, "A Prime Factor FFT Algorithm using High-speed Convolution," *IEEE Transactions on Acoustic and Speech Signal Processing*, ASSP-25, pp. 281-294, 1977.

Pre83 Preparata, F. P., "A Mesh-connected Area-time Optimal VLSI Multiplier of Large Integers," *IEEE Transactions on Computers*, Vol C-32, No. 2, pp. 194-198, February 1983.

Owe80 Owens, R. M., "Digit On-Line Algorithms for Pipeline Architectures," Ph.D. Thesis, Department of Computer Science Technical Report CS-80-21, Penn State University, August 1980.

Owe81 Owens, R. M., "Compound Algorithms for Digit Online Arithmetic," *Proceedings of the Fifth Symposium on Computer Arithmetic*, Ann Arbor, MI, pp. 64-71, May 1981.

Tre82 Treleaven, P. C., "VLSI Processor Architectures," *Computer*, June 1982.

TrEr77 Trevedi, K. S. and M. D. Ercegovac, "On-line Algorithms for Division and Multiplication," *IEEE Transactions on Computers*, Vol. C-26, No. 7, pp. 681-687, July 1977.

Win78 Winograd, S., "On Computing the Discrete Fourier Transform," *Mathematics of Computation*, Vol. 32, pp. 175-199, 1978.