

the system; each of them can be dynamically reconfigured to form the desired topologies to better match task graphs, without interfering with each other. To facilitate the design, we first studied the switching capability of the baseline network, and then presented a number of rules of avoiding connection conflicts. These design rules are so versatile that plenty of flexibility is provided in the design of the procedures. As a result of this, most of commonly used topologies, such as loop, mesh, and binary tree, can be achieved with the procedures.

To reconfigure a subsystem to form a desired topology, the corresponding procedures are invoked with identical parameters and executed simultaneously on all the processors of the subsystem. In addition, the execution time of these procedures is independent of subsystem size. The reconfiguration of a subsystem, as a consequence, can be accomplished in constant time.

REFERENCES

- [1] W. Li, C. Wang, and M. Lavin, "Structured process: A new language attribute for better interaction of parallel architecture and algorithm," in *Proc. 1985 Int. Conf. Parallel Processing*, 1985, pp. 247-254.
- [2] L. Snyder, "Introduction to the configurable highly parallel computer," *Computer*, pp. 47-56, Jan. 1982.
- [3] C. Davis, et al., "Reconfigurable multicomputer network for very fast real-time application," in *Proc. NCC 1982*, pp. 167-184.
- [4] D. DeGroot, "Partitioning job structure for SW-Banyan networks," in *Proc. Int. Conf. Parallel Processing*, 1983, pp. 106-113.
- [5] A. Fiat, A. Shmir, and E. Shapiro, "Polymorphic array: An architecture for a programmable systolic machine," in *Proc. Int. Conf. Parallel Processing*, 1985, pp. 112-117.
- [6] C. Wu and T. Y. Feng, "On a class of multistage interconnection networks," *IEEE Trans. Comput.*, vol. C-31, pp. 694-702, Aug. 1980.
- [7] W. Lin and C. Wu, "An object-based resource management mechanism for a high-performance distributed computing system-star," *COMPSAC 1984*, pp. 208-216.
- [8] S. Yalamanchili and J. Aggarwal, "Reconfiguration strategies for parallel architectures," *Computer*, pp. 44-60, Dec. 1985.
- [9] C. Wu, et al., "Prototype of star architecture—a status report," in *Proc. 1985 NCC*, pp. 191-202.
- [10] B. Arden and H. Lee "Analysis of chordal ring network," *IEEE Trans. Comput.*, vol. C-30, pp. 291-295, Apr. 1981.

Dual Systolic Architectures for VLSI Digital Signal Processing Systems

G. E. BRIDGES, W. PRIES, R. D. MCLEOD, M. YUNIK, P. G. GULAK, AND H. C. CARD

Abstract—This correspondence presents a linear systolic array for the implementation of digital signal processing systems based upon matrix-vector multiplication algorithms where the matrix elements can be computed from their row and column indexes. Haar, Walsh, and the discrete Fourier transforms are solved using this approach. The method presented enables the n^2 matrix elements to be computed *in situ* directly from the $2n$ matrix indexes. Thus, performance comparable to known systolic matrix-vector multipliers is achieved using only constant I/O bandwidth, rather than $O(n)$ bandwidth required in the more general case. A generalized method is given for the development of recursively

Manuscript received February 26, 1985; revised December 7, 1985. This work was supported by the Natural Sciences and Engineering Research Council of Canada.

The authors are with the Department of Electrical Engineering, University of Manitoba, Winnipeg, Man. R3T 2N2 Canada.
IEEE Log Number 8610079.

formed matrices and specifically the VLSI implementation of the Haar and Walsh transforms.

Index Terms—Logic design, matrix multiplication algorithms, parallel computation, systolic architectures, VLSI.

I. INTRODUCTION

Recently proposed VLSI architectures attempt to incorporate high degrees of parallel, pipeline, systolic or hierarchical techniques. Several VLSI computational and design models have been proposed and a variety of metrics have been derived for their performance evaluation [1]-[3]. Perhaps the most common are the VLSI grid model and the AT^2 metric. However, aside from theoretical metrics there are design constraints or practical limits imposed by a developing VLSI technology; these include packaging as well as limits upon the maximum integration complexity.

In this correspondence, we illustrate the implementation of digital signal processing (DSP) systems which are based upon matrix multiplication algorithms. Addressing the problem of physical limitations, a dual systolic architecture is proposed as a method of implementation. The presentation is organized into six sections. Subsequent to the introduction, the dual systolic architecture concept is described with Section III focusing on the development of the architecture for a specific class of problems defined by recursively formed matrices. It is then shown in Section IV how these concepts are used in the implementation of the Haar and Walsh transforms and in Section V, issues related to the practical implementation of DSP problems are discussed. The extension of the architecture to other more general systems, such as the discrete Fourier transform is presented in Section VI. Finally, the conclusions generated by this work are presented.

II. DUAL SYSTOLIC ARCHITECTURE

An architecture based on the systolic array processor was chosen as the method of implementing the desired matrix multiplication algorithms. It should be mentioned that similar algorithms could be implemented on both shuffle-exchange graphs (SE) and cube-connected cycle graphs (CCC) [4]-[6]. I/O limitations and matrix coefficient generation complexity, however, limit the physical realization of the matrix operations on either the SE or CCC architectures using existing technology. On the other hand, the proposed systolic array is capable of performing $O(n)$ computations for each I/O operation. This property is desirable in practice because the I/O bandwidth with the outside world is a major limiting factor for achieving high performance [7].

The recurrence relation for systolic matrix multiplication of order n , $\bar{y} = T\bar{x}$ with $T = [t_{ij}]$, is

$$y_i^{(0)} = 0$$

$$y_i^{(j+1)} = y_i^{(j)} + t_{ij}x_j$$

$$y_i = y_i^{(n+1)} \quad (1)$$

where $\bar{y}, \bar{x} \in$ binary R^n space. The inherent difficulty encountered when implementing the systolic architecture is the method by which the matrix coefficients t_{ij} are generated and passed to the inner product step processors (IPSP) of the array as shown in Fig. 1. A major bottleneck occurs with the generation and presentation of the matrix coefficients to their respective IPSP's. This task is a fact which is often neglected in many proposed architectures and may require a substantial amount of hardware.

For the DSP problems addressed in this correspondence, where the matrix coefficients are functions of the matrix indexes, a dual systolic architecture [8] is proposed as a method of generating the appropriate elements to be passed to the inner product step processors as shown in Fig. 2. The top array of matrix element coprocessors (MECP) are

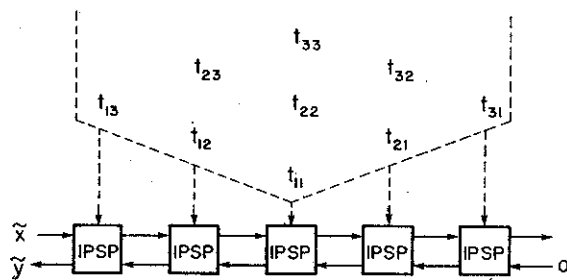


Fig. 1. Systolic array for matrix multiplication of order $n = 3$.

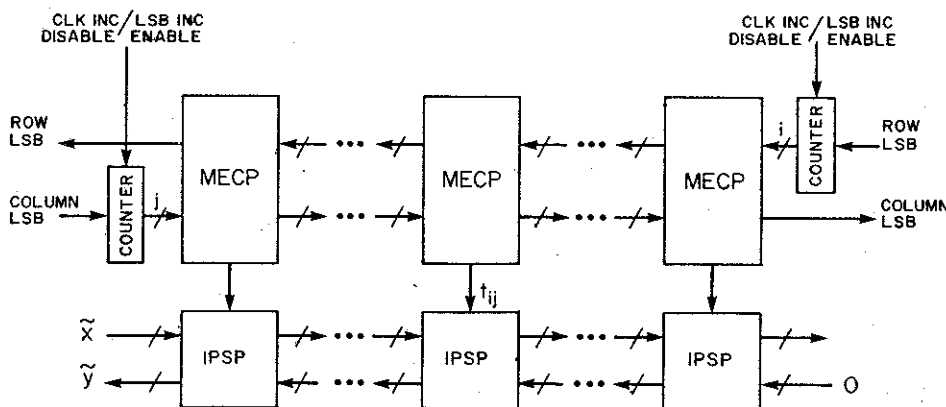


Fig. 2. Schematic of the dual systolic array processor (DSAP).

used to generate the coefficients t_{ij} and any other control signals, with the second array or backbone of IPSP's being used, as before, to perform the matrix multiplication operations. The matrix element indexes (i, j) are generated by counters at both ends of the array. These indexes propagate along the matrix element coprocessor array in parallel with the input and accumulating output vectors which are simultaneously being transmitted along the backbone array.

The concept of using a dual systolic array is similar to the two-level pipelined systolic array architecture utilizing the CMU WARP processor [7]. The matrix coefficients used in the WARP processor are either obtained from a data RAM or are passed in from the outside world. The proposed dual systolic architecture, on the other hand, generates the desired coefficients at each cell site directly from the matrix indexes. This concept has the advantage of flexibility; however, is only practical for simple applications since the complexity of each individual MECP can increase dramatically when complicated coefficient generation is required.

III. GENERALIZED RECURSIVE IMPLEMENTATION

Using the dual systolic architecture shown in Fig. 2, a given matrix multiplication problem can be solved for which the matrix coefficients are derived from the matrix indexes. This large problem set has wide applications in the digital signal processing area such as in the implementation of the discrete Fourier transform, convolutions, or finite impulse response filters. In the n -point discrete Fourier transform, for example, the matrix coefficients are derived as $t_{ij} = \omega^{(i-1)(j-1)}$ where $\omega =$ primitive n th root of unity and $\omega, t_{ij} \in C^n$. As already mentioned however, for many of these general cases the complexity required for implementation of the matrix element coprocessor and the inner product step processor becomes complicated and area consuming. Thus, only limited-point transforms can be performed on a single chip using present VLSI technology. A more complete discussion of this limitation, as well as the discrete Fourier transform with respect to dual systolic architectures will be provided in Section VI. Problem classes where this architecture is practical using present technology will be presented next.

In the remainder of this section we will address a specific class of

digital signal processing problems, namely the implementation of binary- and ternary-based matrices whose coefficients can be generated recursively. Digital signal processing systems such as the Haar [9] and Walsh [10] transformations are examples which fall into this class of problems. This emphasis stems from the research in the literature directed toward the representation of logic functions by finite orthogonal series and their use for the analysis and synthesis of digital systems [11]-[13]. The Haar and Walsh transforms are particularly suited for the spectral representations of Boolean functions and are utilized in many problems such as the determination of network fault signatures [14], [15]. With respect to VLSI design, the major attraction of the restriction to a ternary number set $\{-1, 0, +1\}$ is that this allows the use of an extremely simple IPSP since only the SUBTRACT, NO-OPERATION, and ADD operations are required. The further extension of this problem class to the p -valued based number set where p is prime, is by the system of Chrestenson functions [16].

The matrix multiplication of order $n = 2^N, N = 0, 1, 2, \dots$ will be defined in the usual manner as $\tilde{y} = T^N \tilde{x}$ where $T^N, \tilde{y}, \tilde{x} \in$ binary R^n space and $\tilde{y} = \{y_i | i = 1, 2, \dots, n\}, \tilde{x} = \{x_j | j = 1, 2, \dots, n\}$. Here T^N is the desired $n \times n$ transformation matrix with elements denoted as $T^N = \{t_{ij}^N | i = 1, 2, \dots, n, j = 1, 2, \dots, n\}$. The elements t_{ij}^N are the coefficients passed to the inner product step processors of Fig. 2. Note that for the class of problems discussed here, the transformation matrix T^N can be recursively generated. Using this fact, the matrix T^N is formed from combinations of the next lower order matrix T^{N-1} , then T^{N-2} , and so on, until only the initial 1×1 matrix T^0 is required. The rules defining the recursion combinations determine the type of transform desired, Haar, Walsh, etc. In this discussion, the general form for recursively generated matrices will be developed using the notation

$$[T^k] = \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} \quad (2)$$

where $A_{r_k c_k} = f(T^{k-1}, T^{k-1}, \alpha(r_k, c_k), r_k, c_k)$ and $\alpha(r_k, c_k) \in \{-1, 0, +1\}; r_k, c_k \in \{0, 1\}$. Thus, a transformation matrix of order $n = 2^N$ can be generated iteratively, $k = 1, 2, \dots, N$ using the form

defined in (2). For the k th iteration, the matrices $A_{r_k c_k}$ are formed as a function of the base matrix T^{k-1} , the unit matrix I^{k-1} , and the constant $\alpha \in \{-1, 0, +1\}$. T^{k-1} is the resultant $2^{k-1} \times 2^{k-1}$ matrix of the $(k-1)$ th iteration step. The function forming the matrix $A_{r_k c_k}$ is determined from the binary row and column operators r_k and c_k and defined by the desired recursive nature of the matrix. One example of a recursively formed matrix is the Haar transformation matrix which is defined by the function

$$A_{r_k c_k} = \begin{cases} T^{k-1}, \alpha = 1; & r_k = 0, c_k = 0 \\ T^{k-1}, \alpha = 1; & r_k = 0, c_k = 1 \\ I^{k-1}, \alpha = 1; & r_k = 1, c_k = 0 \\ I^{k-1}, \alpha = -1; & r_k = 1, c_k = 1 \end{cases} \quad (3)$$

with $[T^0] = [1]$. Thus, using (3)

$$[T^k] = \begin{bmatrix} T^{k-1} & T^{k-1} \\ I^{k-1} & -I^{k-1} \end{bmatrix}$$

from which the Haar transform of order $n = 2^3(N = 3)$ can be formed recursively as

$$[T^1] = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad [T^2] = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \quad (4)$$

$$[H^3] = [T^3] = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 & 0 & 1 & 0 & -1 \\ 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \end{bmatrix}$$

Examination of the recursion (2) shows that the control signal passing the matrix element t_{ij} to the inner product step processor in Fig. 2 can be derived directly from the binary representations of the matrix row i and column j indexes. Let the row and column indexes be represented in binary form as

$$\begin{aligned} i &= [r_N, r_{N-1}, \dots, r_k, \dots, r_3, r_2, r_1] \\ \text{MSB} & & \text{LSB} \\ j &= [c_N, c_{N-1}, \dots, c_k, \dots, c_3, c_2, c_1] \end{aligned} \quad (5)$$

where $n = 2^N$ is the order of the matrix with r_k and c_k being the binary digits of i and j , respectively. Thus, the control signal t_{ij}^N can be produced through the iterative process as follows:

$$\begin{aligned} t_{ij}^1 &= f(t_{ij}^0, i_0^0, \alpha(r_1, c_1), r_1, c_1) \\ &\vdots \\ t_{ij}^k &= f(t_{ij}^{k-1}, i_0^{k-1}, \alpha(r_k, c_k), r_k, c_k) \\ &\vdots \\ t_{ij}^N &= f(t_{ij}^{N-1}, i_0^{N-1}, \alpha(r_N, c_N), r_N, c_N) \end{aligned} \quad (6)$$

where k corresponds to the k th iteration step with $k = 1, 2, \dots, N$. Here i_0^k is a unity matrix control signal indicating whether $I_{ij}^{k-1} = 0$ or $I_{ij}^{k-1} = 1$. Note that the initial condition t_{ij}^0 must also be defined in the process, and is simply the value of the 1×1 matrix T^0 .

Using the process described in (6) a method of implementation can

now be developed. First, a ternary number set $\{-1, 0, +1\}$ will be defined which will be implemented using a binary basis. A two-bit control signal V as defined in Table I will be chosen to represent the appropriate ternary operation. Thus, the control signal produced by each of the matrix element coprocessors in Fig. 2 and passed to the IPSP will be a two-bit control word $t_{ij}^N = [v_1^N, v_0^N]$ representing the operations, ADD, NO-OP, or SUBTRACT. This control word will be generated iteratively from the defining function given in (2), using the row and column indexes i and j . The process will be implemented using the matrix element coprocessor design shown in Fig. 3.

In the process shown in Fig. 3, the signal $t_{ij}^0 = [v_1^0, v_0^0]$ is initialized and passed recursively through the series of common function blocks so that the control word t_{ij}^N is produced, and subsequently passed to the IPSP. All the common function blocks are physically the same, with only the appropriate row and column bits (r_k, c_k) entering them determining the value of the control signal t_{ij}^k at each iteration step. For an order n multiplication, $\log(n)$ common function blocks are required.

Each of the common function blocks of the MECF performs the iteration function forming $A_{r_k c_k}$ as described by (2). The layout of the function block $[T^k]$ can be implemented using five logical cells as shown in Fig. 4, each performing a specific task during the iteration. The inputs to the function block are the row and column bits (r_k, c_k), a unity matrix signal bit i_0^{k-1} , as well as the control word t_{ij}^{k-1} from the previous function block. The specific tasks of the five cells are: 1) generation of the unity matrix identifier bit i_0^{k-1} , 2) generation of the control signal w indicating whether i_0^{k-1} or t_{ij}^{k-1} is required, 3) generation of the constant multiplier α , 4) a switch cell implementing the control signal w , and 5) performing the multiplication by α . All five cells are easily implemented using a minimum of logic. Three of the cells (Fig. 4, cells 1, 4, 5) are independent of the recursion function. The remaining two cells (2, 3), which generate the w and α signals, are determined by the required recursion function using the row and column indexes r_k and c_k . These last two cells are thus defined by the desired problem class (Walsh, Haar, etc.).

Generation of the unity matrix elements for I^{k-1} are derived from the row and column indexes by

$$i_0^{k-1} = 1 \cdot \overline{(r_1 \oplus c_1)} \cdot \overline{(r_2 \oplus c_2)} \cdot \dots \cdot \overline{(r_{k-1} \oplus c_{k-1})}, \quad (7)$$

$$i_1^{k-1} = 0$$

where using ternary logic for the control signal $I_{ij}^{k-1} = [i_1^{k-1}, i_0^{k-1}]$. The logic satisfies the definition set up in Table I. Thus, the implementation of the cell (1) requires only a single bit from the unity cell of the previous function block $[T^{k-1}]$ as shown in Fig. 5(a). The switch cell (4) implementing the control signal w , which indicates whether I_{ij}^{k-1} or the previous control word t_{ij}^{k-1} is required, can be designed using pass transistor logic as shown in Fig. 5(b). Note that i_1^{k-1} is grounded (logical "0") in the unity matrix cell. The cell (5) performing the multiplication of the intermediate word V' by the constant multiplier α is derived from the expressions

$$\begin{aligned} v_1^k &= \alpha_1 \oplus v_1' \\ v_0^k &= \alpha_0 \cdot v_0' \end{aligned} \quad (8)$$

and is implemented as shown in Fig. 5(c).

The remaining cells (2, 3), used for the generation of the control signals w and α , are dependent on the desired recursion function. The implementation of the Haar and Walsh transforms will be used as examples to complete the discussion.

IV. HAAR AND WALSH TRANSFORMS

The Haar transform is defined as $\bar{y} = H\bar{x}$ where $H = [h_{ij}]$ with $h_{ij} \in \{-1, 0, +1\}$. Haar functions form a complete orthogonal basis in

TABLE I
CONTROL WORD OPERATION ASSIGNMENT^a

V		Operation	
v_1	v_0		
X	0	0	(NOP)
0	1	+1	(ADD)
X	0	0	(NOP)
1	1	-1	(SUB)

^a X = DON'T CARE state.

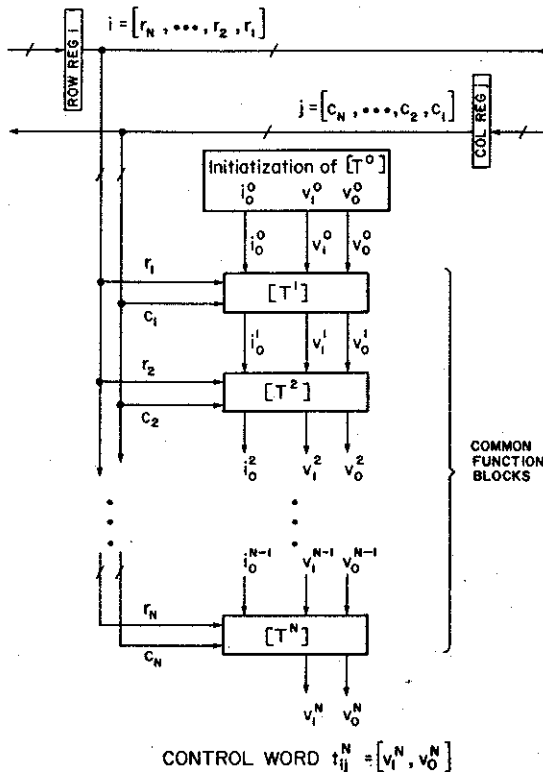


Fig. 3. Matrix element coprocessor (MECP) layout containing N common function blocks. Inputs are the matrix indexes (i, j). Output is the matrix coefficient t_{ij} to be passed to an IPSP.

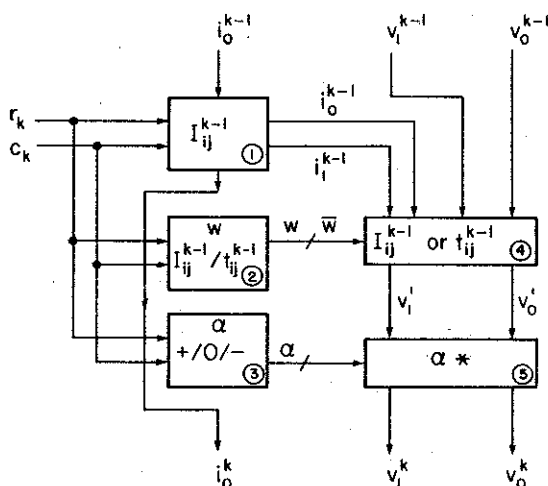


Fig. 4. Common function block designed using five logical cells.

binary R^n space. It should be pointed out that although Haar functions are not normalized, the inverse transform is simply $H^{-1} = [\lambda_i]H^T$, where $[\lambda_i]$ represents a diagonal matrix. The natural ordered Haar matrix [9] is generated recursively using the rule

$$[H^k] = \begin{bmatrix} H^{k-1} & H^{k-1} \\ I^{k-1} & -I^{k-1} \end{bmatrix} \quad (9)$$

with $[H^0] = [1]$.

The recursive generation process for of the Haar matrix has previously been shown in (4). The defining function (2) generating the forming matrix elements $A_{r_k c_k}$ of $[H^k]$ was also defined in (3) as

$$A_{r_k c_k} = f(H^{k-1}, I^{k-1}, \alpha(r_k, c_k), r_k, c_k) = \begin{cases} H^{k-1}; & r_k=0, c_k=0 \\ H^{k-1}; & r_k=0, c_k=1 \\ I^{k-1}; & r_k=1, c_k=0 \\ -I^{k-1}; & r_k=1, c_k=1 \end{cases} \quad (10)$$

Now, using (10), the control signal w generated by cell (2) of Fig. 4, and the constant multiplier $\alpha(r_k, c_k)$ generated by cell (3) of Fig. 4, can now be defined as shown in Table II. Thus, for the case of the Haar transformation, the remaining two cells (2, 3) of the common function block can be implemented as shown in Fig. 6. Note again that only these two cells are dependent on the problem class. Now using the cells designed in Figs. 5 and 6, the common function block for performing the Haar transformation can be implemented as shown in Fig. 7. Note that some further simplification of the design can be made since all possible signals are not required.

Another example selected to illustrate the design methodology is the Walsh transform. The Walsh transform is defined as $\bar{y} = W\bar{x}$ where $W = [w_{ij}]$ and $w_{ij} \in \{-1, +1\}$. Walsh functions form a complete orthonormal basis in binary R^n space and also have the advantage over Haar functions that the inverse Walsh matrix $W^{-1} = W/n$. For natural ordering, the Walsh matrix is generated [17] using the rule

$$[W^k] = \begin{bmatrix} W^{k-1} & W^{k-1} \\ W^{k-1} & -W^{k-1} \end{bmatrix} \quad (11)$$

with $[W^0] = [1]$.

The Walsh matrix of order $n = 2^3(N = 3)$ can be generated as follows:

$$[W^1] = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad [W^2] = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

$$[W^3] = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix} \quad (12)$$

Here, the defining function generating the matrix elements $A_{r_k c_k}$ of $[W^k]$ are defined as

$$A_{r_k c_k} = f(W^{k-1}, I^{k-1}, \alpha(r_k, c_k), r_k, c_k) = \begin{cases} W^{k-1}; & r_k=0, c_k=0 \\ W^{k-1}; & r_k=0, c_k=1 \\ W^{k-1}; & r_k=1, c_k=0 \\ -W^{k-1}; & r_k=1, c_k=1 \end{cases} \quad (13)$$

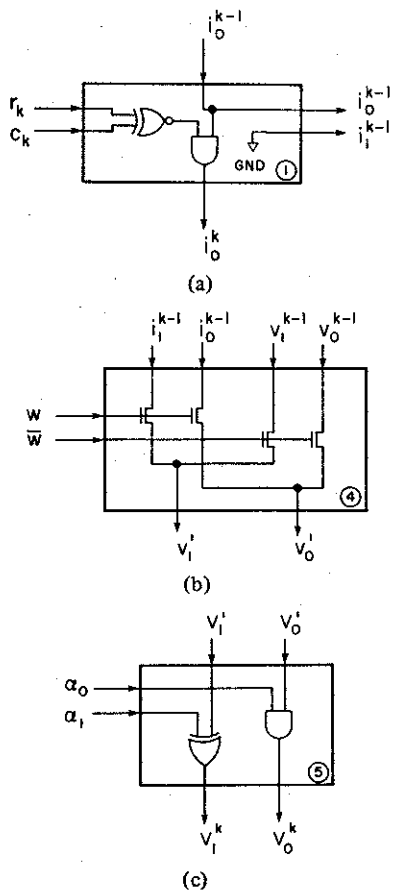


Fig. 5. (a) Unity matrix cell (1) design. (b) Switch cell (4) design. (c) Multiplication cell (5) design.

TABLE II
DEFINITION OF w AND α FOR THE HAAR TRANSFORM

r_k	c_k	w	α	α_0
0	0	0	0	1
0	1	0	0	1
1	0	1	0	1
1	1	1	1	1

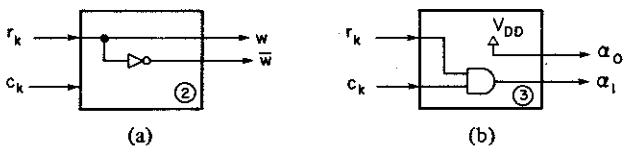


Fig. 6. Cells (2, 3) for generating (a) the control signal w and (b) the constant multiplier α , respectively, for the case of the Haar transform.

Using (13), the control signal w and the constant multiplier α can then be defined as in Table III.

The two remaining cells (2, 3) of the common function block of Fig. 4 can then be implemented as shown in Fig. 8 for the Walsh transform. Note that the implementation of the control signal w is trivial due to the fact that the unity matrix I^{k-1} is not present in the Walsh generating function (11). In fact, since the number base used to form the set of Walsh functions is binary $w_{ij} \in \{-1, +1\}$, only a one-bit control word $i_j^k = [v_1^k]$ is required. The control bit v_0^k can thus be eliminated from the design of the common function block in Fig.

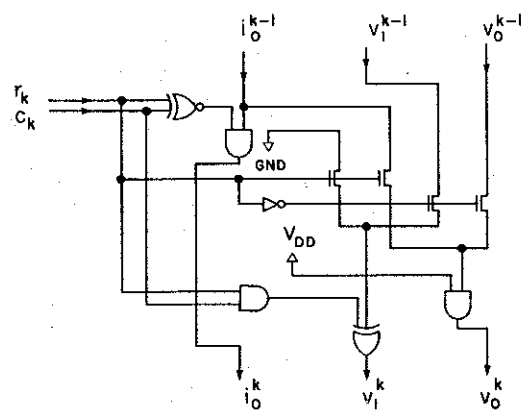


Fig. 7. Haar transform function block.

TABLE III
DEFINITION OF w AND α FOR THE WALSH TRANSFORM

r_k	c_k	w	α	α_0
0	0	0	0	1
0	1	0	0	1
1	0	0	0	1
1	1	0	1	1

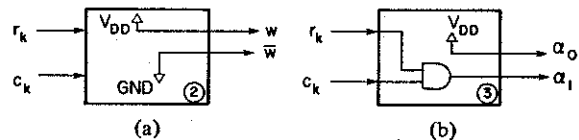


Fig. 8. Cells (2, 3) for generating (a) the control signal w and (b) the constant multiplier α , respectively, for the Walsh transform.

4. The simplified implementation of the matrix element coprocessor for the Walsh transform, taking these considerations into account, is shown in Fig. 9 for an order $n = 2^4 (N = 4)$.

V. DISCUSSION

The proposed dual systolic architecture satisfies most of the criteria for VLSI realizability [18]. The design makes use of each data input item, that is, the input and output data streams are pipelined and used by all processors. Moreover, successive matrix multiplication problems can also be pipelined with the addition of limited control logic. With respect to complexity, the different cells are few in number and of simple topology, which greatly reduces design and implementation costs. The data and control flow is simple and regular and the only global communication aside from power and ground is the system clock. The systolic design uses extensive concurrency; for an n -point operation, an average of $n/2$ processors are computing simultaneously. Furthermore, the major advantages of this design to be stressed is that the dual systolic array is capable of performing $O(n)$ computations for each I/O operation and the problem of delivering the matrix element coefficients to the appropriate step processor has been eliminated.

Unlike many other matrix multiplication architectures, the dual systolic array is easily cascadable. This advantage is important since present single-chip technology constraints will not pose a strict limitation on the transform size that is required. Thus, a 1024-point transform can be achieved in a technology allowing only 256-point transform modules. Cascading of the dual systolic array is accomplished by properly connecting the index counters of the individual

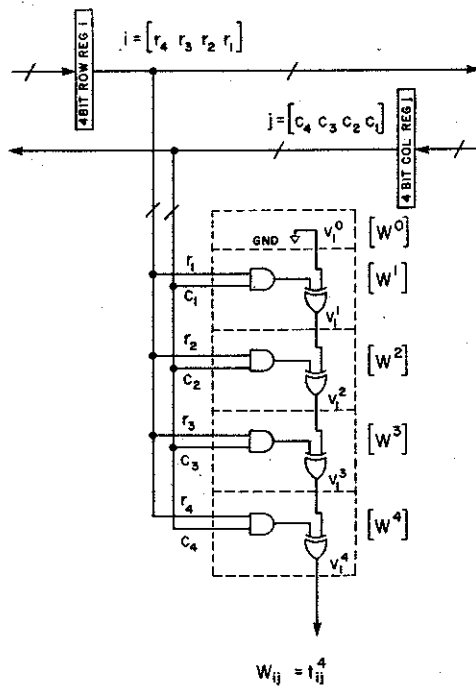


Fig. 9. MECP for implementation of the Walsh transform of order $n = 2^4$.

successive systolic array modules. Connection is made so that the index counter of a given array is activated by the lowest order bit of the preceding module's row or column register. Thus, many individual dual systolic array modules, shown in Fig. 2, can be cascaded as in Fig. 10.

As an example which represents the practical constraints for VLSI implementation, an order $n = 2^N$ point transform will be discussed. The design requires $2n - 1$ cells, each having a matrix element coprocessor and an inner product step processor. The matrix element coprocessor in the coefficient generation array (see Fig. 2) requires two N -bit registers for storing the row and column indexes. The logic required to generate the matrix coefficients requires N common function blocks, each of which can be implemented using 5 gates in the case of the Haar transform or 2 gates in the case of the Walsh transform (see Figs. 7 and 9, respectively). The systolic matrix multiplication inner product step processors each require a data register and an accumulation register, with the accumulation register being N bits wider than the associated data register. Also required is the logic to implement the matrix coefficient operation; addition, subtraction or no-operation in the case of the Haar transform; addition or subtraction in the case of the Walsh transform. The dual systolic architecture has a regular topology and a well-defined control algorithm. There is very little interconnection between cells and thus, for large problem instances the architecture is suited for wafer-scale integration. Though this architecture has an unpleasant aspect ratio, for transform lengths of interest, the array can easily be folded into a practical topology since there are no complicated intercell wire connections and the I/O interface is only at the array ends. Finally, the design methodology outlined allows for the straightforward construction of a silicon assembler or compiler for the automated design of dual systolic architectures.

VI. EXTENSION TO OTHER SYSTEMS

The dual systolic architecture discussed in this correspondence has been introduced for the solution of matrix multiplication algorithms where the matrix elements are well defined functions of the matrix indexes. The specific class of digital signal processing systems considered were those based upon ternary transformation matrices

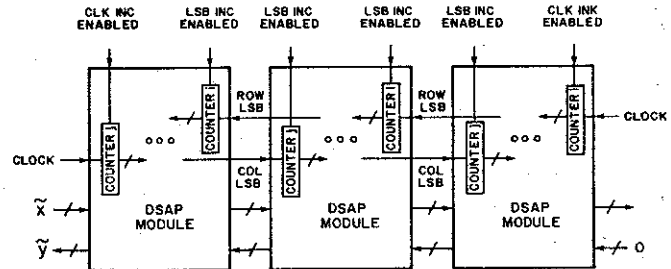


Fig. 10. Cascaded dual systolic array processor modules.

and whose generation could be performed recursively. These systems were examined because they allowed simple design implementation since complex operations were not involved in either the matrix element generation or in the matrix multiplication. The extension to this work will be to suggest alternative dual systolic architectures that are not as restricted to specific DSP problems as the class previously discussed. In turn, however, these architectures will involve more complicated logic for their implementation.

In the dual systolic architecture developed, the matrix indexes were derived directly from the row and column indexes as they were transmitted along the array of matrix element coprocessors as was shown in Fig. 2. The indexes were generated by counters at both ends of the array. In general, however, the matrix indexes themselves need not be transmitted along the array, but any desired vector containing either data or control signals can be passed. Also, the vector can be generated externally from memory (in effect acting as a microprogram control store) as well as internally. Some of these features have been suggested in architectures employing the CMU WARP processor [7]. As a further extension, however, the architecture of the MECP need not be restricted to only passing the vector data, as some operation can be performed on the data as they are passed along the array.

In general, the matrix multiplication in the dual systolic array processor (DSAP) will perform the operation $\tilde{y}_n = C_{n \times n} \tilde{x}_n$. However, to directly implement this matrix operation in the DSAP, the elements of the matrix $C_{n \times n}$ to be generated by the MECP will be formed by the vectors \tilde{a}_n and \tilde{b}_n such that

$$C_{n \times n} = \tilde{a}_n^T \cdot \tilde{b}_n = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_n \end{bmatrix} \odot [b_1 b_2 b_3 \cdots b_n] \quad (14)$$

The outer-product operator \odot is defined as any implementable algorithm in a MECP where \tilde{a}_n and \tilde{b}_n are vectors passed along the array from opposing ends of the processor.

To demonstrate some of these alternative concepts, an MECP architecture will be proposed to perform the discrete Fourier transform (DFT). As previously stated, this architecture will require a more complicated IPSP processor than used previously since the algorithm requires complex number multiplication. Furthermore, the straightforward transformation method requires $O(n^2)$ operations to solve an n -point DFT, while the fast Fourier transform method requires only $O(n \log n)$ operations [19]. However, this example was chosen to demonstrate the flexibility of the dual systolic design since the DFT is both well known and its matrix elements can be determined from the row and column indexes.

The n -point discrete Fourier transform is defined as $\tilde{y} = F\tilde{x}$ where $F = [f_{ij}]$. The complex matrix coefficients are derived from $f_{ij} = \omega^{(j-1)(i-1)}$ where $\omega =$ primitive n th root of unity and $f_{ij}, \omega \in C^n$.

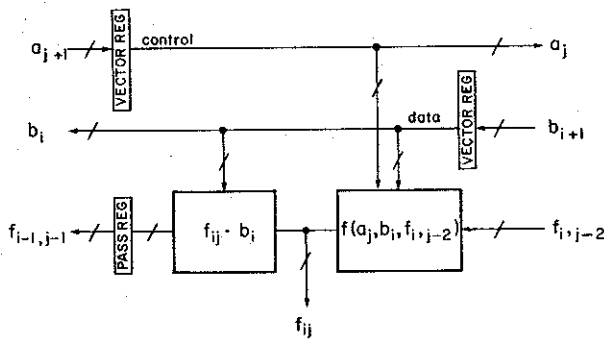


Fig. 11. MECP layout for the discrete Fourier transform.

For order $n = 2^2(N = 2)$, the transformation matrix is given as

$$F = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega & \omega^2 & \omega^3 \\ 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^3 & \omega^6 & \omega^9 \end{bmatrix} \quad (15)$$

where ω is the complex primitive root of $\omega = \sqrt[4]{1}$. The matrix coefficients f_{ij} can be formed using the MECP architecture shown in Fig. 11. Here the vectors \vec{a} and \vec{b} are input vectors of length n , and are clocked into the array at $1/2$ the rate of the data being processed in the array. The elements of the vector $\vec{a} = [a_j]$ are simply two bit control words $a_i \in \{0, 1, 2\}$ indicating the start and finish of the input data \vec{x} . The vector $\vec{b} = [1 \ \omega^2 \ \omega^3 \ \dots \ \omega^{n-1}]$ contains the roots of unity. The matrix coefficient f_{ij} is passed in parallel with the vector \vec{b} and is formed from the function $f(a_j, b_i, f_{i,j-2})$ defined as

$$f_{ij} = f(a_j, b_i, f_{i,j-2}) = \begin{cases} 0, & a_j = 1, b_i = 0 \\ 0, & a_j = 0 \\ 1, & a_j = 1, b_i \neq 0 \\ f_{i,j-2}, & a_j = 2 \end{cases} \quad (16)$$

An example of the coefficient flow for a transform of order $n = 4$ is shown in Fig. 12. For this case, $2n - 1 = 7$ processors are required and the input vectors \vec{a} and \vec{b} are given by

$$\vec{a} = [1 \ 2 \ 2 \ 2] \\ \vec{b} = [1 \ \omega \ \omega^2 \ \omega^3]. \quad (17)$$

VII. CONCLUSIONS

In this correspondence we have demonstrated a novel method for generating the matrix coefficients for digital signal processing systems based upon matrix multiplication algorithms. The cases discussed were the implementation of problems where the matrix coefficients are derived from simple operations on their indexes; specific examples being the Haar and Walsh transforms.

The major advantage of using the dual systolic architecture was that the problem of presenting the transformation matrix coefficients to the inner product step processors has been solved. The coefficients are generated directly in each cell, thus eliminating the necessity to pump them into the array from outside the chip. In addition, the architectures discussed not only adhere to most of the criteria for VLSI design but also are inherently cascadable to any problem size instance. Extension of the proposed architecture to include other DSP problems, such as the discrete Fourier transform, was also presented, the main objective being to demonstrate the flexibility and generality of the concept to a wide variety of possible architectures. At present,

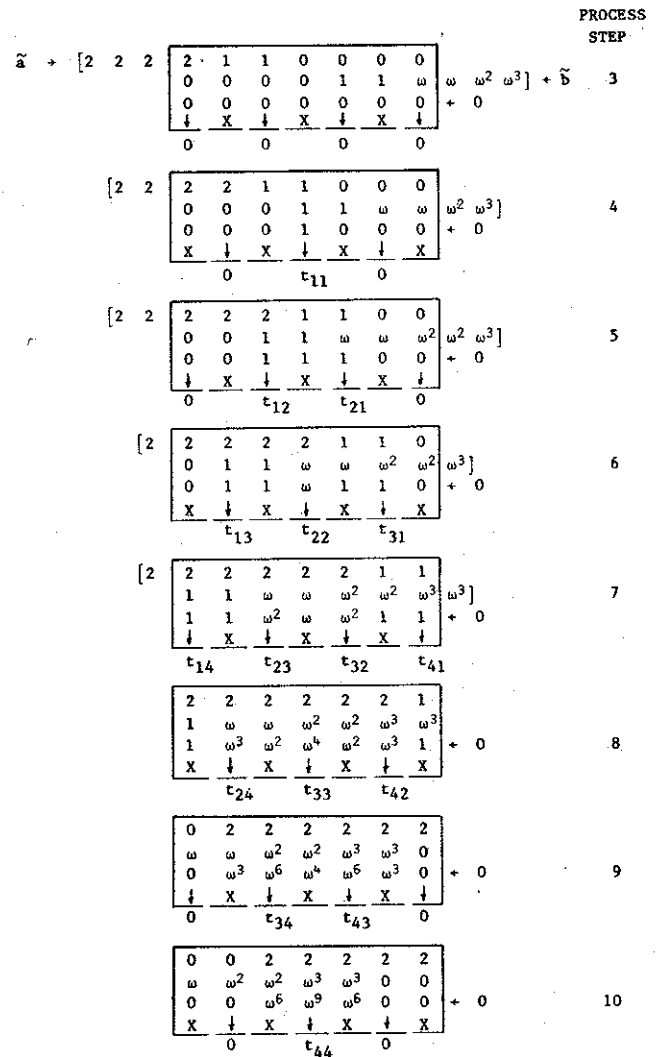


Fig. 12. MECP coefficient flow for a discrete Fourier transform of order $n = 4$. "1" indicates the transmission of a coefficient to an operating IPSP. "X" indicates an IPSP in the resting state.

however, the application to problems requiring only simple operations, such as the Haar and Walsh transform, is practicable. Further effort is continuing to adapt several other DSP algorithms for implementation by a dual systolic architecture as well as in developing a systematic design methodology for coefficient generation.

ACKNOWLEDGMENT

The suggestions made by the referees are appreciated.

REFERENCES

- [1] C. D. Thompson, "Area-time complexity for VLSI," in *Proc. 11th Annu. ACM Symp. Theory of Comput.*, 1975, pp. 81-88.
- [2] G. Bilardi and F. P. Preparata, "An architecture for bitonic sorting with optimal VLSI performance," *IEEE Trans. Comput.*, vol. C-33, July 1984.
- [3] J. D. Ullman, *Computational Aspects of VLSI*. Rockville, MD: Computer Science, 1984.
- [4] H. S. Stone, "Parallel processing with the perfect shuffle," *IEEE Trans. Comput.*, vol. C-20, pp. 153-161, Feb. 1971.
- [5] F. P. Preparata and J. Vuilleman, "The cube-connected cycles: A versatile network for parallel computation," *Commun. Ass. Comput. Mach.*, vol. 24, pp. 300-304, May 1981.
- [6] C. D. Thompson, Ph.D. dissertation, Dep. Comput. Sci., Carnegie-Mellon Univ., Pittsburgh, PA, Aug. 1980.

- [7] H. T. Kung, "Systolic algorithms for the CMU WARP processor," in *Proc. 7th Int. Conf. Pattern Recognition*, Montreal, Canada, July 1984, pp. 570-577.
- [8] M. Yunik, W. Pries, R. D. McLeod, G. E. Bridges, P. G. Gulak, and H. C. Card, "Dual systolic architectures for digital signal processing in VLSI," *1984 Canadian Conf. VLSI*, Edmonton, Canada, Oct. 1984.
- [9] N. Ahmed and K. R. Rao, "Cooley-Turkey-type algorithm for the Haar transform," *Electron. Lett.*, vol. 9, no. 12, pp. 276-278, June 1973.
- [10] J. L. Walsh, "A closed set of normal orthogonal functions," *Amer. J. Math.*, vol. 55, pp. 5-24, 1923.
- [11] N. Ahmed and K. R. Rao, *Orthogonal Transforms for Digital Processing*. New York: Springer, 1975.
- [12] M. G. Karpovsky, *Finite Orthogonal Series in the Design of Digital Devices*. New York: Wiley, 1976.
- [13] S. L. Hurst, *The Logical Processing of Digital Signals*. New York: Crane Russak, 1978.
- [14] D. M. Miller and J. C. Muzio, "Spectral fault signatures for single stuck-at faults in combinational networks," *IEEE Trans. Comput.*, vol. C-33, pp. 765-769, Aug. 1984.
- [15] H. C. Andrews and K. L. Caspari, "A generalized technique for spectral analysis," *IEEE Trans. Comput.*, vol. C-19, pp. 16-25, Jan. 1970.
- [16] H. E. Chrestenson, "A class of generalized Walsh functions," *Pacific J. Math.*, vol. 5, pp. 17-31, 1955.
- [17] N. Ahmed, H. N. Schriber, and P. V. Lopresti, "On notation and definition of terms related to a class of complete orthonormal functions," *IEEE Trans. Electromagn. Compat.*, vol. EMC-15, no. 2, pp. 75-80, May 1973.
- [18] H. T. Kung, "Why systolic architectures?" *IEEE Computer*, pp. 37-46, Jan. 1982.
- [19] L. R. Rabiner and B. Gold, *Theory and Application of Digital Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1975.

A VLSI Solution to the Vertical Segment Visibility Problem

E. LODI AND L. PAGLI

Abstract—We present a parallel algorithm to solve the visibility problem among n vertical segments in a plane, which can be implemented on a VLSI chip arranged as a mesh of trees. Our algorithm determines all the pairs of segments that "see" each other in time $O(\log n)$; while the fastest sequential algorithm requires $O(n \log n)$. A lower bound to the area-time complexity of this problem of $\Omega(n^2 \log^2 n)$ is also derived.

Index Terms—Area-time complexity, computational geometry, mesh of trees, parallel algorithm, VLSI.

I. INTRODUCTION

A relevant problem in VLSI circuit layout is called the "visibility problem." It arises in the compaction of stick diagrams where all components are bounded by orthogonal segments, and it is necessary to determine the "visibility pairs" from among all pairs of vertical (or horizontal) segments [1].

Let us confine our study to vertical segments. Formally, letting S_0, \dots, S_{n-1} be a set of disjoint vertical segments (each S_i includes its extreme points), two segments S_i, S_j constitute a visibility pair if there exists a horizontal line intersecting S_i, S_j , and not intersecting any other segment S_k which lays between S_i, S_j (see Fig. 1).

Among the known results on this problem, we have that the

Manuscript received February 12, 1985; revised July 23, 1985. This work was supported by a grant from the Ministero della Pubblica Istruzione, Italy.

The authors are with the Department of Information Science, University of Pisa, 56100-Pisa, Italy.

IEEE Log Number 8610080.

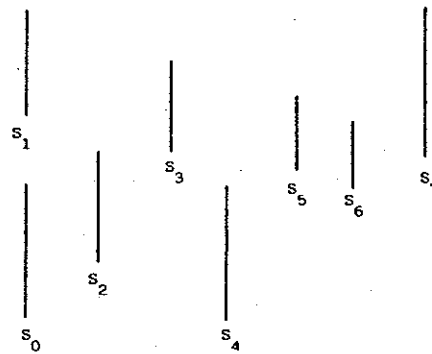


Fig. 1. A visibility problem. Some of the visibility pairs are $S_0, S_1, S_2, S_3, S_4, S_5, S_6,$ and S_7 .

number of visibility pairs is at most $3n - 6$; the fastest known sequential algorithm determines all visibility pairs in time $O(n \log n)$; a lower bound on this time of $\Omega(n \log n)$ has been found for an arbitrary sequence S_0, \dots, S_{n-1} . While no bound higher than $\Omega(n)$ is known when the sequences of x and y coordinates of the endpoints of the S_i 's have been previously sorted (this may be the case in practical applications [1]).

In this correspondence we present a parallel solution to the visibility problem, which can be implemented by means of the proper VLSI chip. First we determine a lower bound to our problem in terms of AT^2 (area-time complexity), by evaluating the "information transfer" of any chip which solves the problem. For this purpose we will show that the visibility problem is equivalent to the one of string equality. This lower bound is an improvement of the one already determined in [2]. Second, we present the VLSI implementation of an algorithm which determines all the visibility pairs of n segments in $O(\log n)$ time units. The VLSI model of computation is the one introduced by Thompson [3] whose basic assumption is that the different processors belonging to a chip work synchronously being driven by the same clock, distributed through the chip, and the transmission time is independent from the wire length. In our proposal the processors are interconnected as a "mesh of trees."

We recall that an $m \times n$ mesh of trees (called *MT*) is a set of $m \times n$ nodes (called leaves) arranged as a $m \times n$ array [4]. The nodes in the i th row ($i = 1, \dots, m$) are connected by a complete binary tree as well as the nodes of the j th column ($j = 1, \dots, n$) so there are $m + n$ trees whose roots constitute the external accesses of the chip (see Fig. 2). The *MT* interconnection is a powerful tool for its multiplex-demultiplex capabilities. For instance, n bits sent in parallel through the n column tree roots reach all the $m \times n$ leaves in $\log m$ time units. Finally we discuss the design of the chip and we conclude that both leaves and internal nodes of the *MT* are elementary processors of constant (small) size, performing elementary operations; the communication channels between nodes are still of constant bandwidth. We assume also that the transmission is allowed from each father node to each son node and vice versa at each clock time. Such mesh of trees can be optimally embedded in a rectangular area of $O(mn \cdot \log m \cdot \log n)$ [4].

II. LOWER BOUND

An AT^2 lower bound to the area-time complexity can be determined by evaluating the "information transfer" I of the problem, that is the minimum number of bits crossing the circuit partitioned in two halves during a worst case computation. It is also known that the square of the information transfer constitutes a lower bound to the AT^2 complexity [3]. We will show that a restricted visibility problem of n segments is equivalent to the one of string equality of length of $O(n \log n)$.

The following lower bound holds in a suitable VLSI model of computation whose basic assumptions are that the chip is synchro-