A VLSI Layout for a Pipelined Dadda Multiplier

PETER R. CAPPELLO and KENNETH STEIGLITZ
Princeton University

Parallel counters (unary-to-binary converters) are the principal component of a Dadda multiplier. We specify a design first for a pipelined parallel counter, and then for a complete multiplier. As a result of its structural regularity, the layout is suitable for use in a VLSI implementation.

We analyze the complexity of the resulting design using a VLSI model of computation, showing that it is optimal with respect to both its period and latency. In this sense the design compares favorably with other recent VLSI multiplier designs.

Categories and Subject Descriptors: B.2.1 **[Arithmetic and Logic Structures]**: Design Styles—*parallel; pipeline*; C.5.4 **[Computer System Implementation]**: VLSI Systems

General Terms: Algorithms, Design, Performance

Additional Key Words and Phrases: multiplier, VLSI, layout, complexity

1. INTRODUCTION

There has been considerable interest in applying VLSI technology to the task of integer multiplication [21]. Lower bounds for this problem have been presented on the area [24], area-time product [1, 4], and data rate [24]. Upper bounds have also appeared for the area-time product (e.g., [4, 13, 17, 18]).

These bounds are asymptotic, and before we go further we should point out that it is important to consider carefully the meaning of such bounds in our context, and especially the effect of using either the Fast Fourier Transform (FFT) or the Discrete Fourier Transform (DFT) as a subalgorithm for multiplication. Designs that do use the DFT or FFT can have good asymptotic properties, but are generally practical only for very long word lengths. Upper bounds obtained this way are viewed as being mainly of theoretical interest. The design

This work was supported in part by the National Science Foundation under Grant ECS-7916292, and in part by the U.S. Army Research Office, Durham, North Carolina, under Grant DAAG29-79-C-0024.

Authors' addresses: P. R. Cappello, Dept. of Computer Science, University of California, Santa Barbara, CA 93106; K. Steiglitz, Dept. of Electrical Engineering and Computer Science, Princeton University, Princeton, NJ 08544.

This paper was originally submitted for publication in the *Communications of the ACM*. The CACM department editor responsible for the paper was Duncan H. Lawrie. Both the authors and editor kindly agreed to publish the paper in the *ACM Transactions on Computer Systems*.

proposed in this paper does not use the FFT or DFT, and appears to be quite practical in the sense that the constants in the asymptotic upper bounds are relatively small.

The following remark appears in [4]: "We do not know if there is any practical design having $AT^{2\alpha} = o(n^{1+2\alpha})$ for $\alpha \in [0, 1]$." Brent and Kung go on to present a design and show that their lower bound is tight to within log factors, but they deem their design to be primarily of theoretical interest because it multiplies via the DFT. Preparata and Vuillemin [18] give an AT^2 optimal ($O(n^2)$) multiplier using the cube-connected cycles topology; it uses the FFT. Preparata [17] improves this theoretical result by achieving the AT^2 lower bound with a simpler topology: the two-dimensional mesh. This design is interesting because such a mesh can be embedded in the plane without long edges, but the design uses the DFT. It *may* be practical for sufficiently large integers. Jackson, Kaiser, and McDonald [11] and Lyon [14], on the other hand, describe a practical multiplier, but its area is $O(n)$ and its time is $O(n)$ so that its $AT^{2\alpha} = O(n^{1+2\alpha}) \neq o(n^{1+2\alpha})$.

Luk, in direct response to the remark of Brent and Kung for $\alpha = 1$, presents a multiplier layout with $T = O(\log^2 n)$ [13]. He states: "The AT^2 measure of this multiplier layout is nearly optimal, being $O(n^2 \log^4 n)$; so it answers the question posed by Brent-Kung that the existence of a practical multiplier having $AT^2 = o(n^3)$ measure." The multiplier design presented by Luk is recursive, performing an n-bit multiply via four $n/2$-bit multiplies and two additions. Its AT^2 complexity is $O(n^2 \log^6 n)$. His design employs the shuffle-exchange network. Reference was made to an alternate design where an n-bit multiply would be obtained by recursively performing three $n/2$-bit multiplies (see [2]). Luk stated that this design had an AT^2 complexity of $O(n^2 \log^4 n)$. It was not presented, however, because the layout is less regular.

The purpose of this paper is to present and analyze a VLSI layout for a multiplier that also has $AT^2 = o(n^3)$. The model we use here is the one used by [4, 13, 17, 18], the *synchronous* model of VLSI [3, 4, 23]. The design we present has time and period that are asymptotically optimal ($T = O(\log n)$ and $P = O(1)$); its AT^2 measure is $O(n^2 \log^3 n)$. Because its period is $O(1)$, it is potentially useful in applications that require both fast response and high throughput. A VLSI measure that favors fast response and high throughput without ignoring area is AP^2T^2. Under this measure, a lower bound for n-bit multiplication is $\Omega(n^2 \log^2 n)$, and the scheme presented has complexity $O(n^2 \log^3 n)$. Moreover, the architecture falls under the rubric of a Dadda scheme [8]. Such architectures are already used in some high-speed computers, attesting to their practicality in demanding situations. The layouts given for the parallel counter and overall multiplier are structurally regular, and so are suitable for a VLSI implementation. Table I summarizes the practicality and asymptotics of the designs mentioned above.

The remainder of the paper is organized as follows. Section 2 defines some terms used in the paper. Section 3 presents a design for a parallel counter (which we call a unary-to-binary converter). In Section 4 we review the Dadda scheme for K-ary addition. Then we present a Dadda design for K-ary addition that employs the unary-to-binary design presented in the previous section. In Section 5 we present a layout for a multiplier using the K-ary adder of Section 4. This is followed by some historical and concluding remarks.

Table I. Summmary of Multiplier Designs

Design	Note	Period	Time	Area	AP^2T^2
Lower bound (Ω)	—	1	$\log n$	n	$n^2\log^2 n$
[4]	DFT	$n^{1/2}\log n$	$n^{1/2}\log n$	$n \log n$	$n^3\log^5 n$
[18]	FFT	$\log^2 n$	$\log^2 n$	$(n/\log n)^2$	$n^2\log^4 n$
[17]	DFT	$n^{1/2}$	$n^{1/2}$	n	n^3
[14]	—	n	n	n	n^5
[15]	—	1	n	n^2	n^4
[13]	—	1	$\log^2 n$	$n^2\log^2 n$	$n^2\log^6 n$
This paper	—	1	$\log n$	$n^2\log n$	$n^2\log^3 n$

Note: Designs that multiply via a DFT computation are generally regarded as being practical only for large values of n.

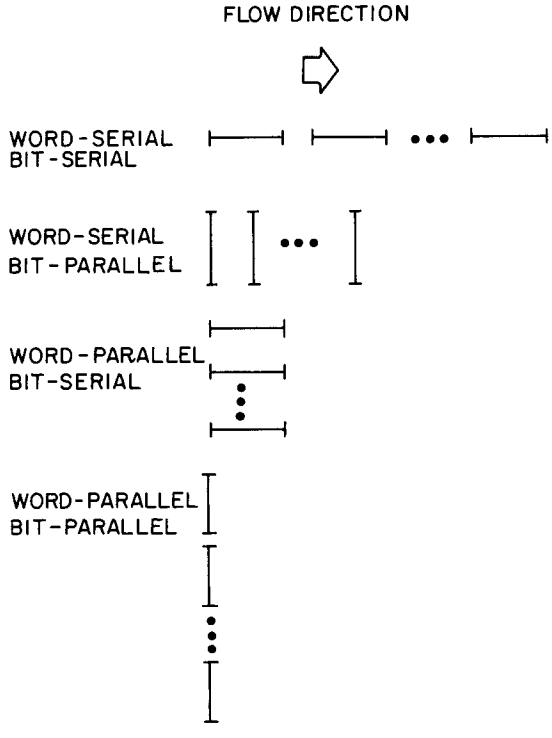

Fig. 1. Use of the following terms: word-serial, word-parallel, bit-serial, and bit-parallel.

2. TERMS

In this section we introduce some terminology. The terms word-serial, word-parallel, bit-serial, and bit-parallel will be used in the obvious ways, as shown in Figure 1.

We deal with classes of functions (and circuits) that are parameterized by a vector, π. For example, when we consider addition of K operands of word size B,

the parameter vector is $\pi = (K, B)$. Asymptotic complexity will be measured with respect to a parameter vector, π, throughout this paper.

We want to make clear important distinctions among three time measures of interest:

Definition. The *functional latency* (or time [24]) of a circuit is the amount of time separating the appearance of the first input bit on some port from the appearance of the last output bit on some port, for one computation of the function, f, denoted T_f. This corresponds to the usual use of the term *speed of operation.* A "100-nanosecond" multiplier, for example, means that 100 nanoseconds elapses between the appearance of the first input bit of the multiplicands and the last output bit of the product. The term does not take pipelining into account.

In order to define a circuit's *cycle time* we appeal to a finite state machine model of computation [10]. A circuit C can be defined as follows:

$$C = (Q, \Sigma, \Delta, \delta, q_0)$$

where

Q is a set of states,
Σ is an input alphabet,
Δ is an output alphabet,
q_0 is distinguished as the machine's initial state, and
δ is a (state transition) function that maps a state and an input to a new state and an output, $\delta: Q \times \Sigma \rightarrow Q \times \Delta$.

Definition. The *cycle time* of a circuit is T_δ (the latency of the circuit with respect to its transition function, δ).

In some circuit architectures, the cycle time depends on the *size* of the function being implemented. For example, an n-bit array multiplier [19] may have cycle time, $\tau = n\tau_c + n\tau_s$, where τ_c and τ_s are the carry and sum bit times of a 1-bit full adder. Such a circuit architecture has a cycle time of $O(n)$.

Definition. The *functional period* of a circuit is the number of cycles separating corresponding bits of successive inputs (outputs) of function f, denoted P_f. Period is the reciprocal of throughput rate. This term does take pipelining into account. Note that $P_\delta = T_\delta$; δ is an indivisible function with respect to these time measures.

Definition. A circuit is *completely pipelined* with respect to function f when $P_f = 1$ and its cycle time is $O(1)$ (see Proposition 2 of [6]).

Note. We consider in this paper only completely pipelined circuits ($P_f = O(1)$ seconds).

For example, in our B-bit (parallel) multiplier, $\Sigma = \{0, 1\}^{2B}$ and $\Delta = \{0, 1\}^{2B}$. In our K-ary B-bit (parallel) adder, $\Sigma = \{0, 1\}^{KB}$ and $\Delta = \{0, 1\}^{B+\log K}$. (See [15] for a variant of the array multiplier that is pipelined). Thus in these architectures, any combinational logic that can be performed in one cycle has depth $O(1)$. The chip's actual bandwidth is then within a constant factor of its maximum bandwidth as determined by the I/O delay of the technology.

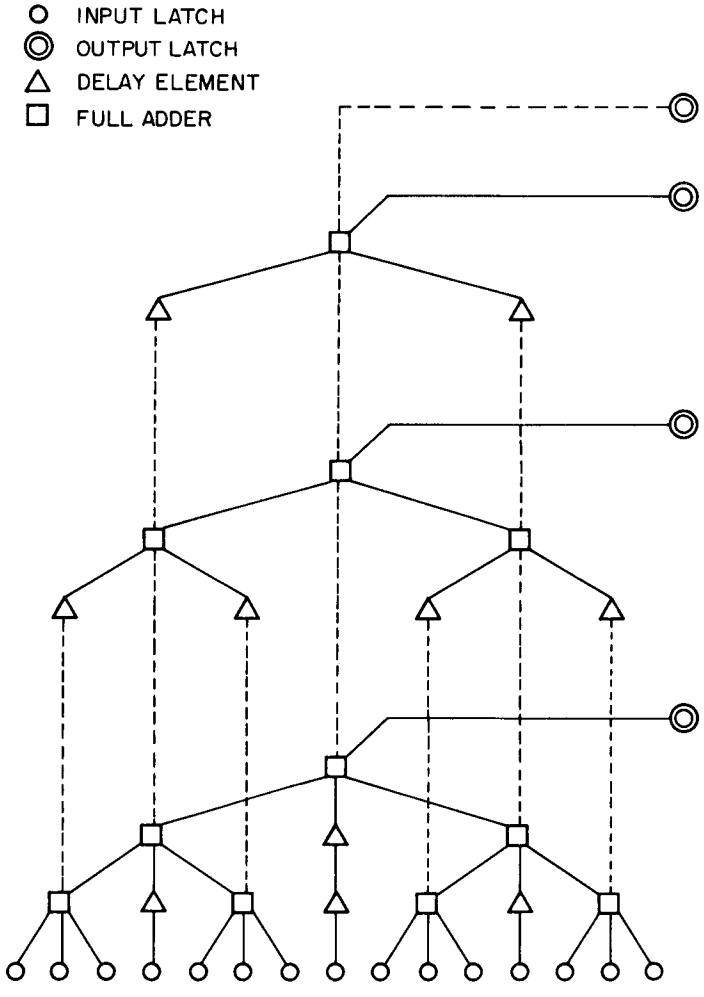

Fig. 2. A unary-to-binary conversion (UBC) structure for $B = 15$.

3. PARALLEL COUNTING

Parallel counting lies at the heart of all the algorithms presented in this paper. In this section we present a parallel counting scheme. First, we state the problem, which we refer to as *unary-to-binary conversion* (UBC).

Problem: Unary-to-binary conversion (UBC)
Input: B bits
Output: $\lfloor \log B \rfloor + 1$ bits—the binary representation of the number of 1 values among the input bits

We now give the UBC design.[1] Figure 2 illustrates a structure for UBC. Each square (□) in the figure represents a latched 1-bit full adder (FA). The triangles

[1] We assume $B = 2^K - 1$ for some integer K without loss of generality as far as asymptotic complexity is concerned.

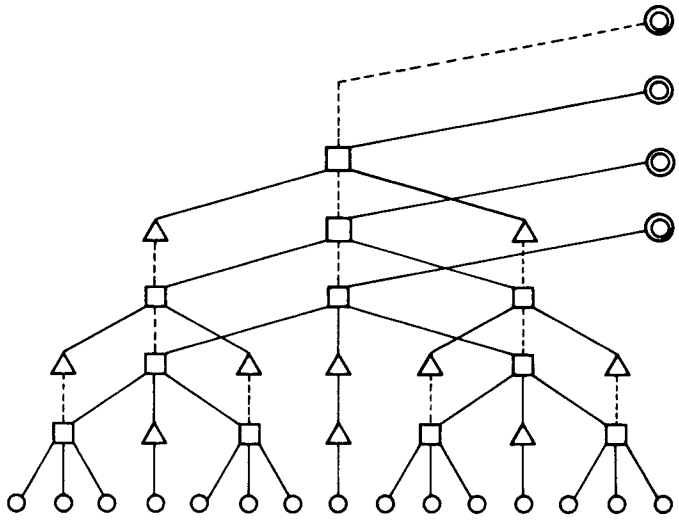

Fig. 3. A more compact UBC structure for $B = 15$.

(Δ) are delay elements. The computation proceeds as follows. Input enters the leaves of the *bottom* tree. The *dashed* output line transmits the carry bit (as depicted in the figure). The *solid* output line transmits the sum bit. Thus, the sum bit transmitted from the root node of the *bottom* tree is the low-order bit of the answer. The dashed lines emanating from the bottom tree, the carry bit lines, enter a second, smaller tree. The sum bit transmitted from that tree's root node is the next-to-low-order bit of the answer, and so on. The number of such trees needed to perform the entire UBC is log n. If the design was such that one tree finished its computation before its successor tree began, then the time complexity

$$T(n) = \sum_{i=\log((n+1)/2)}^{1} i = O(\log^2 n).$$

If the structure had the layout of Figure 2, then the area complexity

$$A(n) = n \times \left[\sum_{i=\log((n+1)/2)}^{1} i \right] = O(n \log^2 n).$$

The trees nest into each other, however, as depicted in Figure 3, and so do the computations. We do not have to finish computing in one tree before starting in the next. Thus,

$$T_{\text{UBC}}(n) = \log((n+1)/2) + \log((n+1)/4) = O(\log n)$$

$$A_{\text{UBC}}(n) = n[\log((n+1)/2) + \log((n+1)/4)] = O(n \log n).$$

The structure, in fact, has an $A(n) = O(n)$ layout since it has an $O(\log n)$-separator (see [12]). Such a layout would be useless though; it would not place the inputs on the layout boundary. Figure 4 illustrates the UBC design with one set of input bits. Each successively higher level in the structure gets values at successively later cycles (as indicated by the cycle labels of Figure 4). This

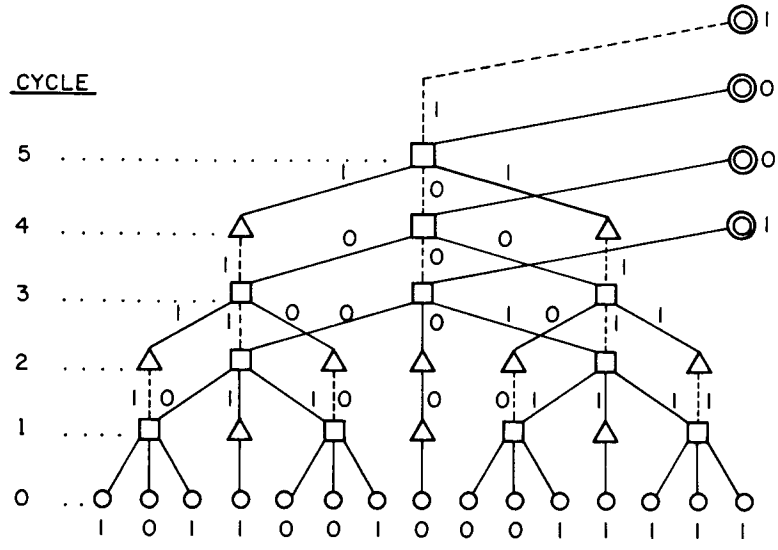

Fig. 4. An example computation on a UBC-network for $B = 15$.

```
{/* The algorithm starts with a pile of the K numbers to be added. */
Do sequentially from low order columns to high order columns:
    {
    Add the digits in the column;
    Post the low order digit of this sum in the sum digit for this column;
    Post the remaining (carry) digits in the appropriate columns;
    /* to be included in the summation of those columns */
    }
}
```

Fig. 5. Elementary school addition algorithm.

illustrates, as well, an important aspect of the design: it is pipelined, so a new set of inputs can be accepted every cycle.

The design has complexity $AP^2T^2_{\text{UBC}}(n) = O(n \log^3 n)$. Using the techniques of [24] we argue that $AP^2_{\text{UBC}}(n) = \Omega(n)$. $T_{\text{UBC}}(n) = \Omega(\log n)$ due to bounded fan-in. So $AP^2T^2_{\text{UBC}}(n) = \Omega(n \log^2 n)$.

In sum, the design is within a log factor of asymptotic optimality and has a simple, regular structure.

4. *K*-ARY ADDITION

In this section we present a K-ary adder (for adding K B-bit numbers) that is based on UBC. We begin by reviewing the K-ary addition algorithm taught in elementary school, given in Figure 5. The algorithm presented is a parallelized version of this.

A parallelized version of this algorithm (reported in [8]) proceeds roughly as follows.

First we do in *parallel* the following (for all columns):

1. Add the "digits" in the column.
2. Distribute the log K "digits" to the appropriate columns.

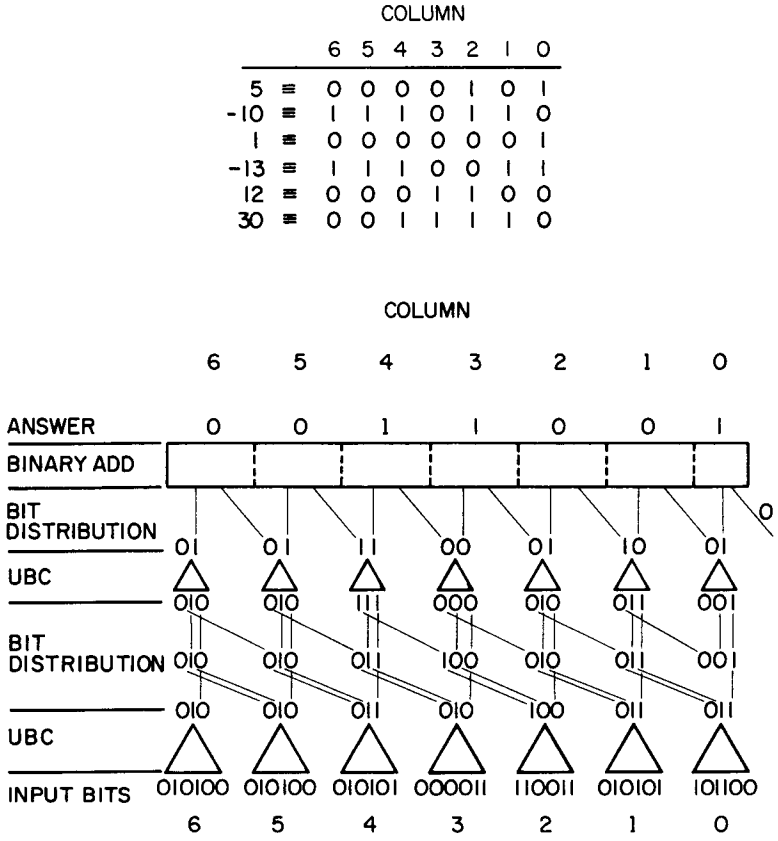

Fig. 6. K-ary addition for $K = 6$, $B = 7$, sum $= 25_{10} \equiv 011001_2$.

```
{
repeat until columns contain 2 bits
    do for all columns
        {
        perform UBC on the column;
        Distribute the sum bits;  /* create new columns */
        }
sum ← carry-lookahead binary-addition;
}
```

Fig. 7. A parallel K-ary addition algorithm.

These "digits" now constitute new columns to be added. We have reduced our K-ary addition problem to a log K-ary addition problem. The process is repeated until a binary addition problem remains. We use a carry-lookahead binary adder for the final addition.

We propose that a VLSI implementation of this algorithm employ the UBC-network design of the previous section, and the adder of Brent and Kung [5]. Figure 6 shows the structure and an example. Figure 7 is a restatement of the algorithm that assumes a binary number representation.

We now analyze the period, latency, and area complexity of this design using the *synchronous* model of VLSI computation [3, 4].

Since the UBCs and binary adder [5] are completely pipelined, the entire K-ary B-bit adder is completely pipelined: $P_{K\text{-add}} = 1$ *cycle.*

The latency of this algorithm is the sum of the time to do the following.

1. Repeat block, which is the sum of the time for doing UBCs and bit distribution ("column" recreation).
2. B-bit parallel binary addition.

The asymptotic latency is

$$\begin{aligned} T_{\text{add}}(K, B) &= O\left(\sum_{i=1}^{\beta(K)} \log \underbrace{\quad}_{i} \log K\right) && \text{(UBC)} \\ &\quad + \sum_{i=1}^{\beta(K)} O(1) && \text{(Bit distribution)} \\ &\quad + O(\log B) && \text{(Binary addition)} \\ &= O(\log KB) \end{aligned}$$

where $\beta(n)$ is a function that grows slower than $\log n$, but is not a constant. See Appendix.

The area complexity is taken as the product of the width and length of the structure:

$$\begin{aligned} A_{\text{add}}(K, B) &= O(KB) && \text{(Base)} \\ &\quad \times \left[O\left(\sum_{i=1}^{\beta(K)} \log \underbrace{\quad}_{i} \log K\right) \right. && \text{(UBC)} \\ &\quad + O\left(\sum_{i=1}^{\beta(K)} \log \underbrace{\quad}_{i} \log K\right) && \text{(Bit distribution)} \\ &\quad \left. + O(\log B) \right] && \text{(Binary addition)} \\ &= O(KB \log KB) \end{aligned}$$

The design has complexity $AP^2T^2_{\text{add}}(K, B) = O(KB \log^3 KB)$. Using the techniques of [24], we argue that $AP^2_{\text{add}}(K, B) = \Omega(KB)$. $T_{\text{add}}(K, B) = \Omega(\log KB)$ due to bounded fan-in. So $AP^2T^2_{\text{add}}(K, B) = \Omega(KB \log^2 KB)$.

Again, the design is within a log factor of asymptotic optimality and has a simple, regular structure.

5. BINARY MULTIPLICATION

In this section we present a binary multiplier. The algorithm, based on K-ary addition, is suggested by the following definition of binary multiplication.[2]

$$x \times y = \sum_{b=0}^{B-1} 2^b \times x_b \times y \tag{1}$$

[2] We work with positive numbers without loss of generality, assuming two's complement arithmetic. The structures described here require a small modification to compute a product with $2B$ correct bits.

$$
\begin{array}{cccc}
y_3 \times x_3 & y_2 \times x_3 & y_1 \times x_3 & y_0 \times x_3 \\
 & y_3 \times x_2 & y_2 \times x_2 & y_1 \times x_2 \quad y_0 \times x_2 \\
 & & y_3 \times x_1 & y_2 \times x_1 \quad y_1 \times x_1 \quad y_0 \times x_1 \\
 & & & y_3 \times x_0 \quad y_2 \times x_0 \quad y_1 \times x_0 \quad y_0 \times x_0
\end{array}
$$

Fig. 8. The summands of eq. (1).

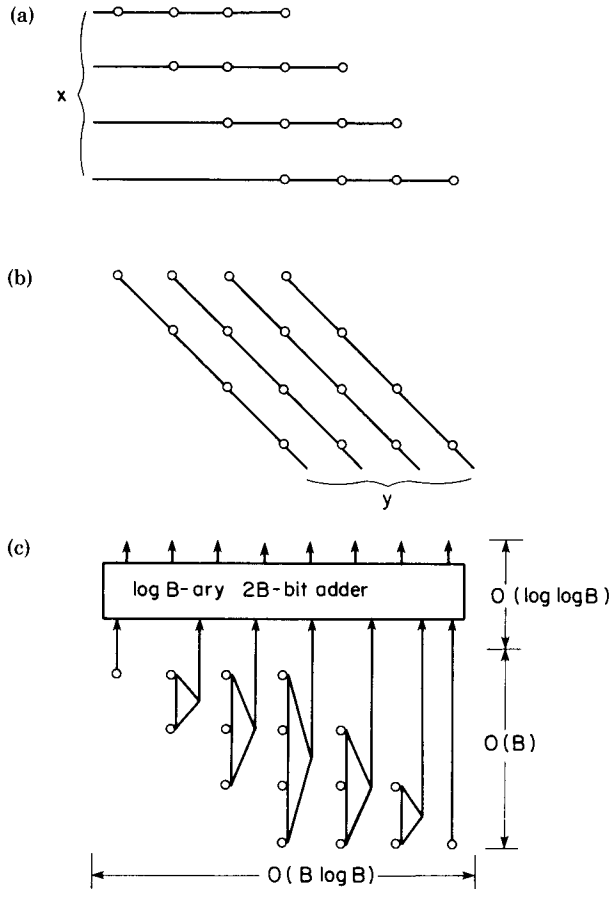

Fig. 9. The multiplication structure is the superposition of the above structures: (a) a structure for distributing x values ($B = 4$); (b) a structure for distributing y values ($B = 4$); (c) a structure for performing the B-ary addition ($B = 4$).

Figure 8 depicts the summands of eq. 1 for $B = 4$. The multiplication algorithm is simple:

1. Distribute x and y values.
2. B-ary add the resulting pile of numbers.

Figure 9(a, b, c) illustrates the structures for distributing x values, y values, and B-ary addition. The multiplication structure is the superposition of these structures.

We now analyze the VLSI complexity of this UBC-network Dadda multiplier with respect to its period, latency, and area. Since the input distribution and AND operation can be completely pipelined into the B-ary B-bit adder which is also completely pipelined, the entire multiplier is completely pipelined: $P_{\text{mult}} = 1$ *cycle.*

The latency of this algorithm is the sum of the time to do the following.

1. Distribute x and y values.
2. AND(x_i, y_j) for every i, j.
3. B-ary add the result.

Thus the asymptotic latency, dominated by the UBC-network latency, is

$$T_{\text{mult}}(B) = O(1) + O(1) + O(\log B) = O(\log B)$$

The asymptotic area complexity, also dominated by the UBC-network areas, is

$$A_{\text{mult}}(B) = O(B^2 \log B)$$

as indicated by Figure 9c.

The design has complexity $AP^2T^2_{\text{mult}}(B) = O(B^2 \log^3 B)$. Regarding VLSI lower bounds we note the following. Vuillemin has shown [24] that:

1. Integer multiplication is a transitive function, and
2. Any circuit computing a transitive function at data rate D must have wire area $A_w \geq a_w \times D^2$, for some technology-dependent constant a_w.

Since the period $P = B/D$ where B is the number of input bits and D is the rate at which they are read in, we have that for transitive functions

$$AP^2(B) = \Omega(B^2).$$

$T_{\text{mult}}(B) = \Omega(\log B)$ due to bounded fan-in. So $AP^2T^2_{\text{mult}}(B) = \Omega(B^2 \log^2 B)$.

As in the UBC and K-ary adder the design is within a log factor of asymptotic optimality and has a simple, regular structure.

6. HISTORICAL COMMENT

We would like to place this VLSI design in some historical perspective. *Gelosia* multiplication methods are those that

1. Construct a two-dimensional array of componentwise products, and
2. Take the B-ary sum of these component products.

The gelosia method (grating method) for multiplication was apparently quite popular during the fifteenth and sixteenth centuries. The method, illustrated in Figure 10, was "probably first developed in India, for it appears in a commentary on the *Lilavati* of Bhaskara (1114 to ca. 1185) and in other Hindu works" [9].

A *Dadda* multiplication scheme is a gelosia multiplication scheme where the "diagonal" K-ary addition is performed in parallel, as presented in Section 4. That is, the K-ary addition problem is reduced to a log k-ary addition problem,

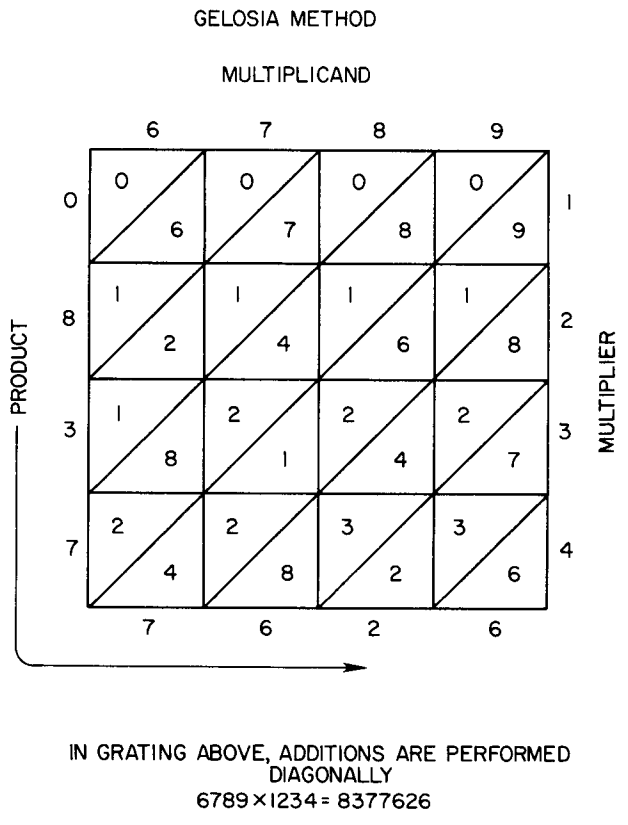

Fig. 10. Gelosia method for multiplication (after [9]). (In the grating illustrated above, additions are performed diagonally.

and that is reduced to a loglog k-ary addition problem, and so on, until a binary addition problem remains. In order to achieve asymptotic optimality with respect to period and latency, this approach requires performing the following:

1. Unary-to-binary conversions asymptotically optimally, and
2. Final binary addition asymptotically optimally.

The adder in [5] does the latter, the UBC-network of Section 3 does the former. And, what is important for VLSI implementation, they do them with simple, regular, compact layouts.

7. DISCUSSION

VLSI techniques are such that the time has now or will soon come when the performance characteristics of a multiplier can be matched to its intended application. For applications where neither fast response nor high throughput are crucial, the multiplier in [11, 14] may be a good match; it has an asymptotically minimal area and a simple, regular structure. For applications where high throughput is crucial but fast response is not, the design in [15] provides an

$AP^2(B)$ asymptotically optimal multiplier. This simple hex-connected array of latched 1-bit full adders is also practical.

We believe the UBC-network Dadda design presented here is a practical design for applications that require both short latency and high throughput. It is *flexible* as well, in that

1. Latency can be minimized by eliminating all latches and delay elements, yielding a purely combinational multiplier of circuit depth (hence cycle time) $O(\log B)$,
2. Throughput can be maximized by latching the UBC-networks at every level within the UBC-network, as presented in this paper, and
3. Some trade-off between latency and throughput is possible by latching every k level within a UBC-network (for $1 \leq k \leq O(\log B)$).

As long as the UBC-networks are latched every O(1) levels, the design is asymptotically optimal in both period and latency.

Finally, it may be that:

1. B is large.
2. The technology is such that propagation delay over a wire of length l is asymptotically greater than $\Omega(l^{1/2}/\log l)$, a situation not modeled properly by the synchronous VLSI model (e.g., [7, 20]).
3. High throughput is not as important as short latency.

For these applications, the design given in [17] merits consideration due to its two-dimensional mesh topology and $T(B) = O(B^{1/2})$ latency.

Digital signal processing is one area which stands to gain a great deal from VLSI implementation of arithmetic, both with small period (for throughput) and small latency (for fast adaptation, for example). Many such applications involve K-ary addition and can benefit from the Dadda approach. One important example is the *second-order section* of an Infinite Impulse Response (IIR) filter. This computation can be defined as follows [16]:

$$y_k \leftarrow a_0 x_k + a_1 x_{k-1} + a_2 x_{k-2} - b_1 y_{k-1} - b_2 y_{k-2}$$

The Dadda-like design for this "sum of products" involves using:

1. The gelosia method to create bit arrays for the five products, and
2. A UBC-network Dadda scheme to sum them *directly*, computing y_k (which Swartzlander calls *merged arithmetic* [22]).

Figure 11 illustrates the use of a UBC-network Dadda scheme in the design of a second-order section. The top part of that figure shows the *distribution* of the x and y bits, and the bottom part shows the *collection* of the results using UBC networks in a way analogous to Fig. 9c. Note that this structure can *not* be completely pipelined, because we need y_k to compute y_{k+1}. The latency of O(log B) is asymptotically as small as possible, so one cannot build a second-order section with higher asymptotic throughput. Using strictly combinational

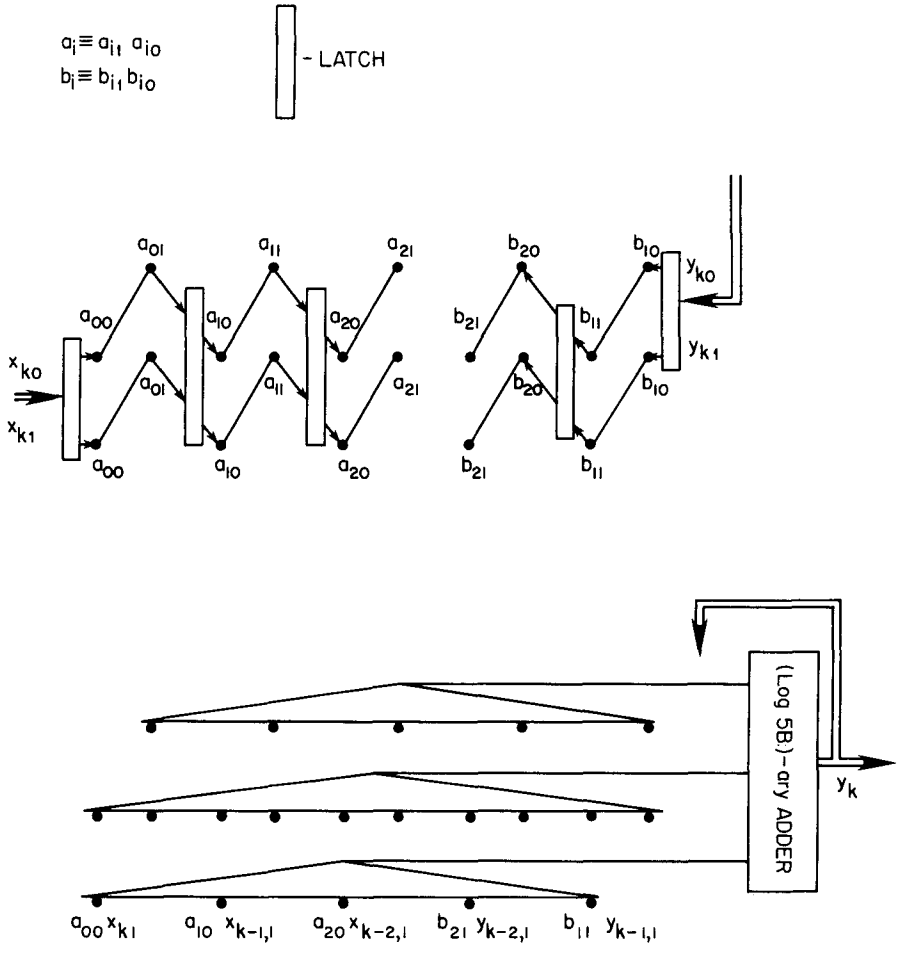

Fig. 11. The superposition of these structures is a UBC-network Dadda scheme for a second-order section ($B = 2$).

logic (no latches and no delays) would simplify the control and reduce latency. The cycle time would then be $O(\log B)$ seconds, as would the latency.

Figures 12 and 13 show similar designs for convolution and fixed-matrix vector product. In these problems there is no feedback, and the structures are completely pipelined. Table II summarizes the complexity of several functions designed this way.

Two points are worthy of emphasis here. First, the latency of these designs is in all cases asymptotically optimal. It is on the order of the *logarithm* of the input size. Second, these designs are *completely pipelined* (with the exception of the second-order section). That is, the designs are such that a new set of input data can be accepted every cycle and that the cycle time is *independent* of the parameters of the function (e.g., B-bits in the case of multiplication) being implemented.

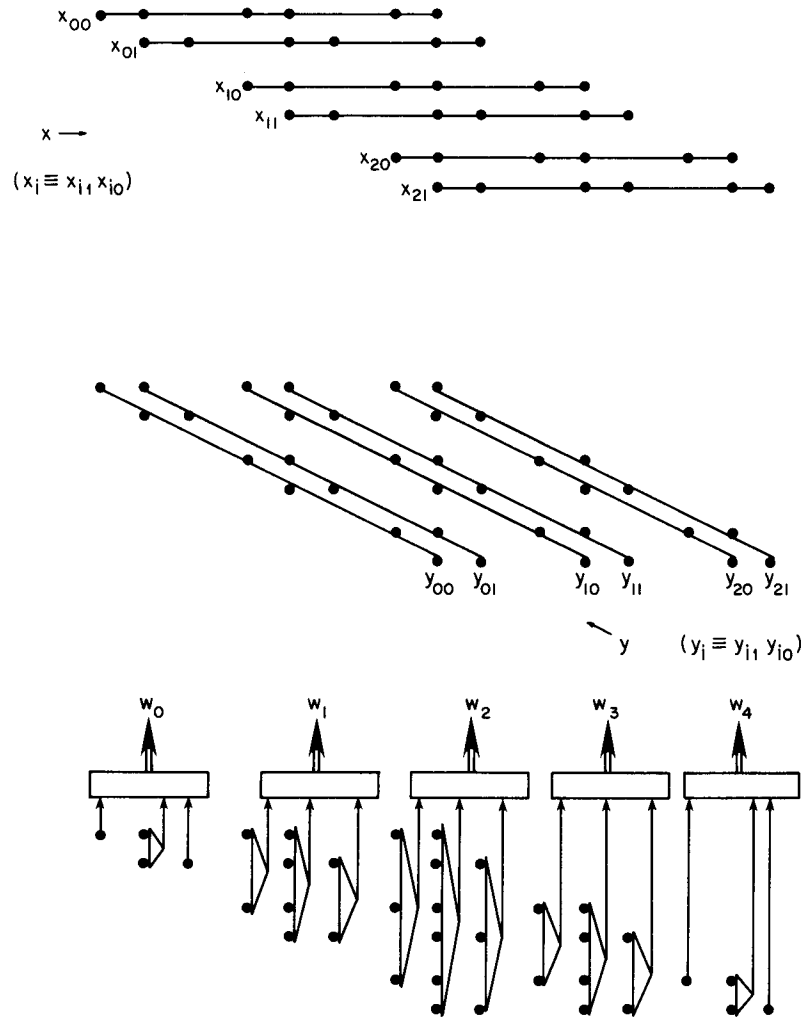

Fig. 12. The superposition of these structures is a UBC-network Dadda scheme for convolution ($K = 3, B = 2$).

8. CONCLUSIONS

We have presented a design for completely pipelined multiplication, and we have analyzed its period, latency, and area complexities using a VLSI model of computation. Such a multiplier is useful for applications where both short latency and high throughput are very important; its complexity is within one log factor of asymptotic optimality with respect to an appropriate measure: AP^2T^2. Its latency and period are asymptotically optimal. The cell types are simple, being no more complex than 1-bit full adders, and the layout has a regular structure. Consequently, we feel that the design is practical.

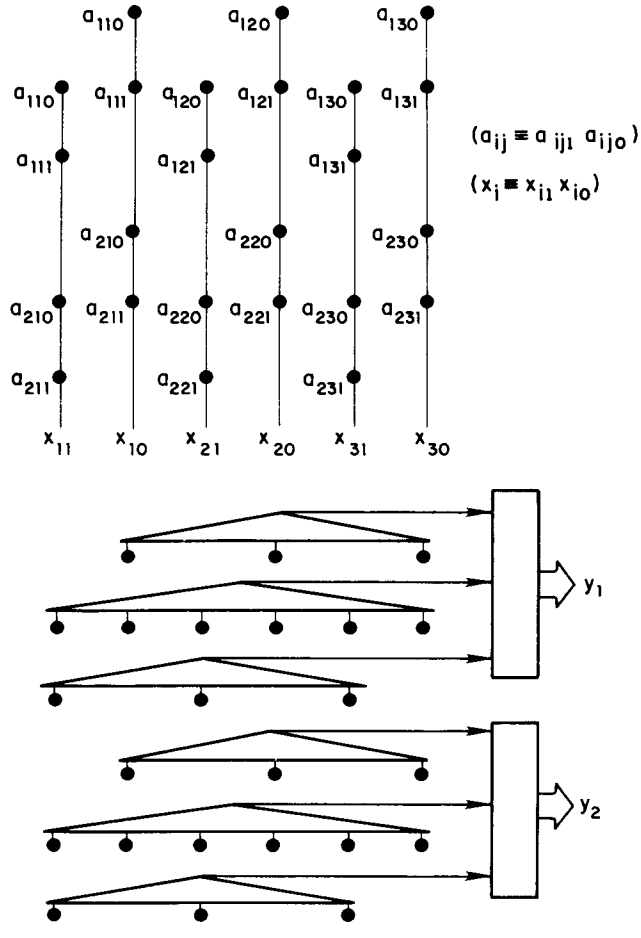

Fig. 13. The superposition of these structures is a UBC-network Dadda scheme for fixed-matrix product (the matrix is 2×3, $B = 2$).

Table II. Asymptotic Growth Using UBC-Network Dadda Scheme Design

	Area Bound	AP^2T^2 Bounds	
Function	Upper area	Lower	Upper
Integer product	$B^2 \log B$	$B^2 \log^2 B$	$B^2 \log^3 B$
Fixed-matrix vector product	$K^2B^2 \log KB$	$(KB)^2 \log^2 KB$	$(KB)^2 \log^3 KB$
Convolution	$K^2B^2 \log KB$	$(KB)^2 \log^2 KB$	$(KB)^2 \log^3 KB$
Matrix product	$K^4B^2 \log KB$	$K^2(KB)^2 \log^2 KB$	$K^2(KB)^2 \log^3 KB$

Table III. The function $\beta(n)$

n	$\beta(n)$
2	0
3	1
4–7	2
8–127	3
$128\text{-}2^{128}-1$	4

APPENDIX

$\beta(n)$ is defined as follows.

$$\beta(n) = \begin{cases} 0 & \text{if } n = 2 \\ 1 + \beta(\lfloor \log n \rfloor) & \text{if } n > 2 \end{cases}$$

It is essentially the same as $G(n)$ [2]:

$$G(n) = \begin{cases} 0 & \text{if } n = 1 \\ 1 + G(\lceil \log n \rceil) & \text{if } n > 1 \end{cases}$$

For practical purposes, $\beta(n)$ is less than 5, as can be seen from Table III.

ACKNOWLEDGMENT

We thank Peter Reusens for his many helpful suggestions on an earlier version of this paper. They improved the result considerably. We also benefited from the constructive criticism of the referees.

REFERENCES

1. ABELSON, H., AND ANDREAE, P. Information transfer and area-time tradeoffs for VLSI multiplication. *Commun. ACM 23*, 1 (Jan. 1980), 20–23.
2. AHO, A. V., HOPCROFT, J. E., AND ULLMAN, J. D. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, Mass., 1974.
3. BILARDI, G., PRACCHI, M., AND PREPARATA, F. P. A critique and an appraisal of VLSI models of computation. In *VLSI Systems and Computations*, edited by H. T. Kung, Bob Sproull, and Guy Steele, Computer Science Press, Rockville, Md., 1981.
4. BRENT, R. P., AND KUNG, H. T. The area-time complexity of binary multiplication. *J. ACM 28*, 3 (July 1981), 521–534.
5. BRENT, R. P., AND KUNG, H. T. The chip complexity of binary arithmetic. In *Proc. 12th Annual ACM Symp. Theory of Computing*, (Los Angeles, Cal., April 28–30, 1980), ACM, New York, pp. 190–200.
6. CAPPELLO, P. R., AND STEIGLITZ, K. Completely pipelined architectures for digital signal processing. Tech. Rep. 303, Dept. of Electrical Engineering and Computer Science, Princeton Univ., Princeton, N.J., June 1982.
7. CHAZELLE, B., AND MONIER, L. A model of computation for VLSI with related complexity results. In *Proc. 13th Annual Symp. on Theory of Computing*, (Milwaukee, Wisconsin, May 11–13, 1981), ACM, New York, pp. 318–325.
8. DADDA, L. Some schemes for parallel multipliers. *Alta Frequenza 34*, (May 1965), 349–356. Reprinted in *Computer Design Development*, E. E. Swartzlander, Jr., Ed., Hayden Book, Rochelle Park, N.J., 1976.
9. EVES, H. W. *Mathematical Circles Revisited.* Prindle, Weber & Schmidt, Boston, Mass., 1971.
10. HOPCROFT, J. E., AND ULLMAN, J. D. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, Mass., 1979.

11. JACKSON, L. B., KAISER, S. F., AND MCDONALD, H. S. An approach to the implementation of digital filters. *IEEE Trans. Audio and Electro-Acoustic, AU-16* (Sept. 1968), 413–421.
12. LEISERSON, C. E. Area-efficient graph layouts (for VLSI). In *Proc. IEEE 21st Annual Symp. Foundations of Computer Science.* Syracuse, N.Y., 1980.
13. LUK, W. K. A regular layout for parallel multiplier of $O(\log^2 n)$ time. In *VLSI Systems and Computations*, H. T. Kung, Bob Sproull, and Guy Steele, Eds., Computer Science Press, Rockville, Md., 1981.
14. LYON, R. F. Two's complement pipeline multipliers. *IEEE Trans. Comm., COM-24*, 4 (April 1976), 418–425.
15. MCCANNY, J. V., MCWHIRTER, J. G., ROBERTS, J. B. G., DAY, D. J., AND THORP, T. L. Bit level systolic arrays. In *Proc. 15th Asilomar Conf. on Circuits, Systems, & Computers*, Nov. 1981.
16. OPPENHEIM, A. V., AND SCHAFER, R. W. *Digital Signal Processing.* Prentice-Hall, Englewood Cliffs, N.J., 1975.
17. PREPARATA, F. P. A mesh-connected area-time optimal VLSI integer multiplier. In *VLSI Systems and Computations*, H. T. Kung, Bob Sproull, and Guy Steele, Eds., Computer Science Press, Rockville, Md., 1981.
18. PREPARATA, F. P., AND VUILLEMIN, J. E. The cube-connected-cycles: A versatile network for parallel computation. *Commun. ACM 24*, 5 (May 1981), 300–309.
19. RABINER, L. R., AND GOLD, B. *Theory and Application of Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, N.J., 1975.
20. RAMACHANDRAN, V. On driving many long wires in a VLSI layout. Tech. Rep. 300, Department of Electrical Engineering and Computer Science, Princeton, Univ., Princeton, N.J., April 1982.
21. REUSENS, P., KU, W., AND MAO, Y. Fixed-point high-speed multipliers in VLSI. In *VLSI Systems and Computations*, H. T. Kung, Bob Sproull, and Guy Steele, Eds., Computer Science Press, Rockville, Md., 1981.
22. SWARTZLANDER, E. E., JR., Merged arithmetic for signal processing. In *Proc. IEEE 4th Symp. Computer Arithmetic*, Santa Monica, Calif., 1978.
23. THOMPSON, C. D. Area-time complexity for VLSI. In *Proc. 11th Annual Symp. Theory of Computing*, (Atlanta, Georgia, April 30–May 2, 1979), ACM, New York, pp. 81–88.
24. VUILLEMIN, J. A combinatorial limit to the computing power of VLSI circuits. In *Proc. IEEE 21st Annual Symposium of Foundations of Computer Science.* Syracuse, N.Y., 1980.

Received October 1981; revised June 1982; accepted November 1982