STOC(Milwaukee 1981),300-307

Lower Bounds for VLSI

Richard J. Lipton †

Princeton University
Princeton, New Jersey 08544

Robert Sedgewick † †

Brown University
Providence, RI 02912

1. Introduction

Increased use of Very Large Scale Integration (VLSI) for the fabrication of digital circuits has led to increased interest in complexity results on the inherent VLSI difficulty of various problems. Lower bounds have been obtained for problems such as integer multiplication [1,2], matrix multiplication [7], sorting [8], and discrete Fourier transform [9], all within VLSI models similar to one originally developed by Thompson [8,9]. The lower bound results all pertain to a space-time trade-off measure that arises naturally within this model. In particular, for all the problems listed above, the results show that if A is the area used by a VLSI circuit to compute one of the n-input, n-output functions listed above, and T is the time required for the computation, then the bound

$$AT^2 > \Omega(n^2)$$

must hold. Vuillemin [10] has recently unified most of these results by showing that they share a simple mathematical structure. In this paper, we extend the model and the class of functions for which non-trivial bounds can be proved.

In Section 2, we give a more general model than has been proposed previously. In Section 3 we show how to reduce the derivation of lower bounds within the model to a problem in distributed computing similar to that proposed by Yao [11]. It is interesting to note that this problem can be solved by a "crossing sequence" technique similar to that commonly used to prove lower bounds for one-tape Turing machines [4].

In Section 4, we consider lower bounds for a number of *predicates*: n-input, 1-output functions (as contrasted with the n-input, n-output functions which have been studied previously). The techniques of Section 3 make it possible to prove that $AT^2 > \Omega(n^2)$ for a variety of predicates. Furthermore, we show that union, composition, and alphabet change can transform "easy" functions into "hard" ones, which makes the VLSI computational model quite different from classical ones where these operations are easy.

In Section 5, we show that previous lower bound results (for n-input, n-output functions) also apply even when the model is extended to allow nondeterminism, randomness, and multiple arrivials. The predicates of Section 4 are more sensitive: some are significantly easier, others are not affected by these extensions (but the lower bounds are more difficult to prove.) We also consider the affects of other variations in the model.

Finally, the full details of the results presented here will appear in the final version of this paper.

2. The Model

In this section, we will present the basic model of VLSI computation used in this paper. Since we are interested in proving lower bounds, we will allow the model great generality: it allows features which certainly are not even contemplated in the near future. But the generality of the model allows it to be specialized to incorporate the particular features of a much wider range of actual systems than allowed by previous models, and all the old lower bound still hold. In its generality, the model is in many ways simpler than previous ones, and admits cleaner lower bound proofs.

There are three main components: the *boolean function* f which is to be computed, *a synchronous circuit* C that computes f, and a *VLSI layout* V that realizes C.

We assume that C is a network of *wires* attached to each other and to *gates*. The gates of C can be "and", "or", or "not" gates of *arbitrary* fan-in and fan-out, or they can implement more general *decomposable* functions with arbitrary numbers of inputs and outputs. (These possibilities are dis-

† Supported in part by NSF #170C938

† † Supported in part by NSF

cussed further in the next section.) Such a circuit, which may have *feedback*, computes f provided there is an *input-output schedule* that decides how the inputs and outputs of f are mapped onto the input and output wires of C. In all previous work it has always been assumed that the input-output schedule satisfies:

(1) each input arrives once and each output leaves once;

(2) when and where this takes place is *data independent*.

We will, for now, assume (1) but we will not assume (2). The motivation for (1) is that otherwise we would be allowing the circuit "free" memory. However, (2) is an unnecessary restriction, and we will allow quite a larger class of input-output schedules.

Definition: An input-output schedule is *where-oblivious* (*when-oblivious)* provided where (when) the inputs arrive and the outputs leave is independent of the values of the data.

Thus, all previous models consider only when and where oblivious input-output schedules. Note, of the two types of schedules the first is the more common: often it is natural to assume that where an input goes is fixed; however, when it arrives may depend on a variety of things. For example, consider a micro- processor. As it fetches data from a RAM, where the inputs go is fixed and hence is data independent, but when inputs arrive depend on what addresses the micro-processor generates and when it generates them.

These new types of input-output schedules are not just of theoretical interest. Recently, Lipton and Valdes have proved:

Theorem: There is a where-oblivious VLSI layout for the transitive closure of graphs that has $AT^2 = 0(n^{3\ 2+\epsilon})$ (any $\epsilon > 0$).

It is an interesting open question whether or not such a layout can be made when-oblivious.

The crucial part of the model has to do with the assumptions imposed on the VLSI layout, so we will give a more formal definition:

Definition: A VLSI layout V is a (λ,μ_1,μ_2)*-layout* of the sequential circuit C if there is a map that assigns to each gate g (wire w) of C a closed connected region of the plane g^* (w^*) so that

(1) If w is an input or output wire of gate g, then g^* intersects w^*; and

(2) For each $\lambda \times \lambda$ square S of the plane,

  (a) at most μ_1 gates g map to regions g^* that intersect S, and

  (b) at most μ_2 wires w map to regions w^* that intersect S

We further assume that all of the g^* and w^* lie in a convex region R, the *region of the layout*.

Note that no assumption is made about the location of circuit inputs or outputs, or even how they are assigned to gates (except that the assignment must satisfy the schedule). Different inputs could be assigned to the same gates, at different times. For convenience we will call gates in the layout which have inputs or outputs that correspond to inputs or outputs in the circuit *input* or *output* gates. We will assume that circuits use all their inputs, and the parameter which describes the "size" of the circuit is n, the number of inputs.

As mentioned above, this definition is quite general and allows VLSI layouts that could never be fabricated: but the generality of the definition only reinforces the lower bounds that we will prove. Indeed, it is the restrictions in the definition that need to be examined carefully.

Clearly, it is reasonable to assume that gates and wires are connected regions: otherwise they could not carry electrical charge. The assumption that they are closed regions is purely a mathematical convenience with no practical ramifications. The assumption that R is convex could be relaxed somewhat, but it is a very modest restriction (VLSI layouts are always rectangles).

Condition (1) simply forces electrical connections to correspond to topological connections. Note carefully, however, that the converse is not true: g^* and w^* can intersect without any connection between g and w in C. This is of course possible in actual layouts, because multiple layers are allowed. We have chosen not to explicitly model which layer a gate or wire in on: gates and wires can be mapped to intersecting regions yet not be connected. The lower bound results can be proved even within this much simplified model.

Condition (2) is a direct result of the limits of VLSI fabrication. Think of the $\lambda \times \lambda$ square S as a "window" onto the plane. This condition ensures that in any such window we can only "see" a fixed number of gates and wires. This has two implications. First, it limits the number of layers used: No more than $\mu_1 + \mu_2$ layers can be used at the same point, because if they were, more than $\mu_1 + \mu_2$ gates or wires would intersect at that point, violating either (2a) or (2b). Second, condition (2) gives a bound on how tightly gates and wires can be packed. Note, however, there is no explicit bound on the size of any individual gate or wire. Again, this is unreasonable, but it only makes our lower bounds that much stronger.

As mentioned above, it is straightforward to specialize this model to less general models. For example, if we take $\mu_1 = 1$ and $\mu_2 = 2$, only allow gates to be on lattice points, and only allow wires to run along a lattice, our model becomes essentially the same as previous models. But by eliminating virtually all such geometric restrictions and allowing unlimited fan-in and fan-out, we are able to model a much richer class of actual VLSI layouts.

We are almost ready to begin to prove lower bounds on the VLSI complexity of functions, but we must first define our complexity measure. Let C be a circuit that computes f in time T and let V be a VLSI layout with area A that realizes C. Then central to all that follows is the assumption:

A and T measure the complexity of f

The justification for A is simple. In any of the current VLSI techniques the area of a VLSI layout is a critical parameter. As the area increases the difficulty in fabricating the layout also increases; hence, it is natural to require that A be kept as small as possible. Just as the number of gates of C was a reasonable cost measure in classic models of complexity, the area A now plays that role.

The justification for T is more complex. The VLSI layout V takes time $\tau \times T$ to compute f where τ is the time required for one clock period of the sequential circuit C. What is somewhat surprising is that we can assume that τ is *independent* of the layout V and only depends on the given technology. The key is that the time it takes to send a signal down a wire is not limited by the speed of light but by the parasitic capacitance of the wire. By increasing the size of the driving gate it is possible to keep τ independent of the layout and only effect the area by a constant factor. One further comment is in order: for lower bounds clearly the use of T is justified. It is only when we turn to upper bounds that we must carefully check the validity of T as a measure of speed. (See [2,8].)

As modeled here the central problem facing a VLSI designer is as follows:

Given a boolean function f;

Find a sequential circuit C that computes f in time T and a VLSI layout of C with area A so that A and T are "minimized"

Thus the key problem facing the VLSI designer is to minimize both the time and area of the VLSI layout. However, in all but the most trivial cases a designer is unlikely to be able to minimize both simultaneously. The designer will therefore have to minimize some derived function $u(A,T)$: here $u(A,T)$ is the *complexity* of a VLSI layout with area A and time T. There are many possible such functions.

Examples.

1. $u(A,T) \equiv 1$. This corresponds to a simple situation where any implementation is useful. Perhaps it models the way a beginning designer feels - if it works, its a success!

2. $u(A,T) \equiv A$. This corresponds roughly to the situation of the designer of small calculators. Here speed is unimportant - the whole key is size and hence cost.

3. $u(A,T) \equiv T$. This is "dual" to the last example. It corresponds to the situation in the design of supercomputers.

4. $u(A,T) \equiv AT$. This is a measure that weights both area and time. It is similar to the measures used previously in studying time and space in sequential computations.

5.

$$u(A,T) \equiv \begin{cases} 0, & \text{if } A \leqslant A_0 \text{ and } T \leqslant T_0 \\ 1, & \textit{otherwise} \end{cases}$$

This might correspond to the task of a real VLSI designer. If she meets her constraints, i.e., $A \leqslant A_0$ and $T \leqslant T_0$, then her design is a success; if not, then it is a failure.

We are faced with a crucial question: which complexity functions do we use? The way out of our embarrassment of riches is to add a few constraints on such functions, since not all complexity functions are equally interesting.

First, it is natural to assume that $u(A,T)$ is *monotone*, i.e.,

$$u(A,T) \leqslant u(A',T')$$

if $A \leqslant A'$ and $T \leqslant T'$. Surely it is reasonable to assume that less area or less time do not raise the complexity of a design.

Secondly, it is natural to assume that $u(A,T)$ is *rescalable*, i.e., there is a function $g(k,l)$ so that

$$u(kA,lT) = g(k,l)\, u(A,T)$$

for all positive k,l,A, and T. The intuitive meaning of this is that u depends on the choice of units used to measure A and T only up to a constant factor. Thus,

$$u(A,T) = cu(A',T')$$

for some constant c, if A and T are measured in cm^2 and seconds and A' and T' are measured in ft^2 and microseconds. It is important to note that physical laws are almost always rescalable in our sense. Since we wish to discover such basic laws for VLSI, we will only study such complexity functions. Continuing the last examples we observe that 1,2,3, and 4 are all rescalable while 5 is not.

Theorem: Let $u(A,T)$ be a monotone, rescalable map from pairs of positive number to non-negative ones. Then $u(A,T) = \alpha_0 A^{\alpha_1} T^{\alpha_2}$ for some $\alpha_0, \alpha_1, \alpha_2$.

This is an important theorem, since it allows us to focus all our attention on the complexity functions $\alpha_0 A^{\alpha_1} T^{\alpha_2}$. (We omit its proof which is based on simple functional analysis.)

Let us again summarize, in light of this theorem, the VLSI designer's problem:

Given a boolean function f and a complexity measure u (recall u is assumed to be monotone and rescalable).

Find a sequential circuit C that computes f in time T and has area A so that $u(A,T)$ is minimized.

By this theorem, we can assume that $u(A,T) = \alpha_0 A^{\alpha_1} T^{\alpha_2}$ for some $\alpha_0, \alpha_1, \alpha_2$. For the purpose of the minimization of A and T, there are only three classes of complexity measures:

(1) $u \equiv 0$;

(2) $u \equiv T$;

(3) $u \equiv AT^{\alpha}$ for some α.

By this we mean that given any $u_1(A,T) = \alpha_0 A^{\alpha_1} T^{\alpha_2}$ we can find a u_2 from one of these three classes so that

A,T makes $u_1(A,T)$ a minimum

if and only if

A,T also makes $u_2(A,T)$ a minimum.

Since we are primarily interested in A and T we will only study u from class three. Indeed AT^2 will play a special role.

3. Proof Techniques

The general technique used by Thompson and others to prove lower bounds for VLSI layouts involves a "cut theorem" relating area and time to the "information flow" across a line which divides the layout into two parts. Roughly, the theorem is proved as follows: Draw a line which divides the layout into two parts, with about half of the inputs to the circuit in each part. Some restrictions about the geometry of the layout and the nature of the inputs are needed to ensure that this can be done. Suppose that the line cuts through about ω wires (the line could also cut through gates, a complication which must be handled carefully): then under some assumptions about the geometry of the layout and the line, it is possible to show that $A > \Omega(\omega^2)$. Furthermore, the value ω can be thought of as a bound on the "information" that can flow across the boundary: if the total amount of information that must flow across the boundary during the entire computation is I, then the time taken must satisfy $T > I/\omega$. This leads immediately to the bound $AT^2 > \Omega(I^2)$. The proofs for specific problems are completed by showing that $I > \Omega(n)$ for any division of the circuit that puts half the inputs on either side of the dividing line.

This line of reasoning leads to lower bound arguments very similar to those used for one-tape Turing Machines [4]. Proving lower bounds for AT^2 is reduced to constructing appropriate sets of input assignments. The "planarity" restriction for VLSI is, in some sense, as severe as the one-tape restriction for Turing machines. The lower bound proofs do not carry through directly, however, because for VLSI the result must be proved for *any* possible division of the inputs into two halves, while for Turing machines a single division is inherent in the problem. For example, the set of strings xy with $x = y$ is "hard" for one-tape Turing Machines: they need time at least $\Omega(n^2)$. On the other hand, there is a VLSI layout for this predicate with AT^2 at most $O(n)$. The reason for this difference is clear: for some partitions a great deal of "information must flow" and for others very little is needed.

We will now make these notions more precise. Let f be a boolean function with n inputs. It is convenient to assume that n is even. The first step is to cover the layout for f with the smallest enclosing rectangle. It is easy to see that this at most doubles the area, and therefore only affects the AT^2 product by a constant factor.

The second step is to assume that the input-output schedule is where-oblivious. Because of this and the fact that each input arrives exactly once, we can assign inputs uniquely to input wires. Next construct a line parallel to the short side of the rectangle so that $l < n/2$ and $l + m > n/2$ where

l = the number of inputs that arrive to gates strictly to the left of this cut;

m = the number of inputs that arrive to gates that are hit by this cut.

Note carefully that the definition of these quantities are well defined since the input-output schedule is where-oblivious. One constructs this cut as follows: start on one of the short edge of the rectangle covering the layout (say the layout is oriented so that this is the left edge), then scan across until a point is reached where $l < n/2$ and $l + m > n/2$. (It is an easy geometric argument to prove that such a point must exist.)

The key now is to partition the inputs of f into two equal classes: L and R. All L's must arrive to input gates that are either to the left or hit by the cut; all R's must arrive to input gates that either to the right or hit by the cut. Since $l < n/2$ and $l + m > n/2$ this is clearly possible.

After this has been established, the two halves of the inputs can be treated as cooperating distributed algorithms as in [11]. The problem of computing lower bounds for AT^2 is then reduced to the problem of finding lower bounds for I, the amount of "information" that L and R must exchange in order to compute f. But this notion can be made precise.

Define a *crossing value* as including the following:

(1) for each wire hit by the cut, its value;

(2) for each wire input to a "not" gate which is hit by the cut, its value;

(3) for each "and" gate that is hit by the cut, the value of $u_1 \wedge \ldots \wedge u_r$ and $v_1 \wedge \ldots \wedge v_s$ where $u_1, \ldots, u_r$ are strictly to the left of the cut and $v_1, \ldots, v_s$ are either to the right of it or hit by it and $u_1, \ldots, u_r, v_1, \ldots, v_s$ are all the input wires to this gate.

(4) for each "or" gate that is hit by the cut, the value of $u_1 \vee \ldots \vee u_r$ and $v_1 \vee \ldots \vee v_s$ where $u_1, \ldots, u_r$ and $v_1, \ldots, v_s$ are as in (3).

A *crossing sequence* is then the sequence of T crossing values that appear during the history of the computation on some particular computation. Let w be the length of the cut. Note, $w^2 \leqslant A$.

Lemma 1: There are at most c^{wT} crossing sequences. (c is a constant that depends on the VLSI technology only.)

Proof. This follows directly from the definition of a crossing value and the definition of a VLSI layout.

$\square$

A few comments are in order. The definition of a crossing value was complicated by our desire to allow arbitrary fan-in "and" and "or" gates. If we replace part (3) of the definition of a crossing value by: (similarly for part (4))

for each "and" gate that is hit by the cut, the values of the input wires,

we would find that lemma 1 would be false! This is the case since a large fan-in gate can touch the cut in a very small area. One of the advantages of our model is that we do allow arbitrary fan-in while all previous models do not. It is important to allow this since current VLSI technologies do indeed support such gates, although such gates will grow in area as the number of inputs increases. Moreover, all the old lower bounds are still true for layouts with such gates.

Finally, we can allow even more complex gates with arbitrary fan-in. For example, suppose we wish to allow threshold functions:

$$x_1 + \ldots + x_k > m.$$

Then we can repair the definition of crossing value so the bound in lemma 1 becomes

$$c^{Tw \lg w}$$

This will only affect the lower bounds we obtain by a $\lg n$ factor. In general, a new family of gates can be allowed provided the definition of crossing values can be repaired so that there are not "too many" crossing sequences.

Just as in the analysis of one-tape Turing Machines the following is the key. Let x be an input: we use x_L to denote the input that equals x on the L part and are 0 elsewhere; we use x_R to denote the input that equals x on the R part and are 0 elsewhere. Note $x = x_L + x_R$ (usual addition).

Lemma 2: If x and y are inputs with the same crossing sequence, then

$$x_L + y_R$$

behaves the same as x on all output wires that are to the left of the cut.

A corresponding lemma is true for the right side of the layout. Both these lemmas are proved by a simple induction on the number of steps in the computation. Part (3) and (4) of the definition of a crossing value allow the induction to work correctly.

We can use lemmas 1 and 2 as follows to prove lower bounds on AT^2. Assume that we can show that there must be at least c^I crossing sequences. Then by lemma 1,

$$c^{wT} \geqslant c^I$$

and so $w^2T^2 \geqslant I^2$ and since $w^2 < A$ it follows that $AT^2 \geqslant I^2$. The key then is to use lemma 2 to get a lower bound on the number of crossing sequences needed. For the n-input, n-output functions of [10] it is easy to see that $I > \Omega(n)$. For predicates as considered in the next section we argue as follows:

Assume that output wire is to the left of the cut. Then construct a family of inputs $x_1,...,x_m$ with $m = 2^{\delta n}$ $(\delta > 0)$ so that each x_i has a unique L and R part and so all are accepted by the predicate. (There is a dual method when all are rejected.) Now assume that some way exists to do this in less than m crossing sequence. Then for some $i \neq j$, by the pigeon-hole principle,

$$(x_i)_L + (x_j)_R \qquad (*)$$

is accepted. But if we have constructed the family so that this is impossible, then we have proved that at least $2^{\delta n}$ crossing sequence are needed. We can even generalize this technique. Suppose we could do the predicate in only $2^{\epsilon n}$ crossing sequence $\epsilon < \delta$, then (*) would not only be true but would hold for many i,j pairs. Indeed, this technique is used in proving a number of our lower bounds.

While the above proof techniques are applicable to many problems, it must be noted that with one simple added restriction, the same type results can be proved in a trivial way (within a model that describes the way VLSI circuits are currently built.) The added restriction is that each input gate must touch the boundary of the region of the layout at some point (call such a layout a *boundary layout*). Then

Theorem: $AT^2 > \Omega(n^2)$ for boundary layouts whose bounding rectangle has sides that are in a constant ratio.

This is true for all existing layouts. Note, it assumes nothing about the input-output schedule: it need not even be where-oblivious. In other words, in order to do better than the n^2 bound (and require the intricate proof technique above), a VLSI layout must either be a non-boundary layout or it must be exceeding narrow (say lgn by n/lgn). Some real VLSI it appears, will be non-boundary layouts. In any event, it does offer a new technique for obtaining non-trivial lower bounds. For boundary layouts, we can show that

$$AT^2 > \Omega(In).$$

This can give good lower bounds for problems where $I > \Omega(n^2)$ may be too difficult to prove or even false! We actually use this technique in the next section.

4. Lower Bounds for Predicates

Previous lower bounds for VLSI have been for n-input, n-output functions. The proofs have depended on showing that a large amount of information must flow between the inputs and the outputs because each output must depend in some non-trivial way on all inputs. This type of proof obviously will not work when there is only one output. However, the proof technique of the previous section is more general than this and allows $AT^2 > \Omega(n^2)$ for a variety of predicates. Because we have obtained many such results we will only state them here and will only give a detailed proof of one of the more interesting ones. All lower bounds assume where-oblivious input-output schedules.

Selection/Equality testing: Given $2n$ input bits, divide them into two halves of n bits each. The first half is used for *selection:* it has $n/2$ zeros and $n/2$ ones. The value of the predicate is 1 if the $n/2$ bit number obtained by selecting those bits in the second half of the input at positions corresponding to the zero bits in the selection mask is equal to the $n/2$ bit number obtained in the same way from the one bit positions in the selection mask.

A deterministic context-free language: The value of the predicate is 1 if the inputs belong to the DCFL xcx^R, with x a word from $\{0,1\}^*$, but $*$'s can occur anywhere in the input and must be ignored. This problem can be encoded to be a predicate n binary strings for which $AT^2 > \Omega(n)$ (This result obtained with M. Harrison.)

Pattern Matching: Given a binary text string of $(1-\alpha)n$ bits and a pattern of αn bits, with $0 < \alpha < 1$, determine if the pattern occurs in the text.

Factor Verification: Given binary numbers x,y, and z of αn, βn, and $(1-\alpha-\beta)n$ (with $0 < \alpha,\beta < 1$) determine whether or not $xy = z$. A harder problem: given an n bit number x, determine whether or not $x = y^2$ for some integer y. (This result obtained with Ravi Kannan.)

The proof techniques in Section 3 allow $AT^2 > \Omega(n^2)$ lower bounds to be proved for all of the above problems, though each proof is detailed and requires some problem-specific facts. Typically the proofs break into two parts depending on whether the two parts of the input which arise naturally in the problem fall mainly on opposite sides of the dividing line (in which case information about the inputs must be transferred across the cut), or mainly on both sides of the dividing line (in which case information about intermediate computations must be transferred across the cut).

The tools of the previous section can also be used to prove weaker lower bounds for more difficult problems. For example, an interesting problem for which no non-trivial lower bound has previously been known is

Binary Determinant: Given a square binary matrix with a total of n bits, compute the value of its determinant (mod 2).

We have been able to show that the lower bound $AT^2 > n^{3/2}$ must hold for this problem, using the combined "boundary" argument mentioned at the end of the last section. This is a rare example of a lower bound which could not be obtained using the trivial argument for "boundary layouts" mentioned above. We believe the true bound to be n^2 for this problem, but the question remains open.

Another problem which remains open is *Primality:* Given an n-bit number, determine whether or not it is prime. This problem leads, after the "crossing sequence" argument has been applied, to an open conjecture in number theory, which seems difficult to prove, though likely to be true.

Finally, it is possible to prove some general results about combining simple functions that provide strong evidence that the VLSI complexity is unusual. In particular, we have n-input functions f and g which can be computed in $AT^2 = O(n)$ but which become difficult when combined according to the fundamental rules:

Union: The predicate which is 1 if either $f(x)$ or $g(x)$ is 1.

Composition: The function $f(g(x))$.

(Of course different functions are used for each of these results.)

These results seem to suggest that the "top-down" design philosophy of decomposing complicated functions into simpler ones, then combining the results, may not be a good strategy for VLSI: it may be harder to combine results than to compute them. The "integration" in VLSI may need to apply to algorithm design as well as fabrication technology. It is also possible to show that *alphabet change* can dramatically decrease the complexity. There is a function $f(x)$ with $AT^2 > \Omega(n^2)$ if inputs can take on values from $\{a,b,c,d\}$; but if these are represented as two bits, then $AT^2 = O(n)$. In the first case it is equivalent to allow the inputs to be represented as two bits but require them to be near each other on the layout. It is interesting to note that in Thompson [8] it is assumed that inputs to a FFT layout come from a non-binary alphabet. This result shows that such assumptions can affect the complexity greatly.

We will now prove that there are two f and g so that each can be done in $AT^2 = O(n)$ and yet their union requires $AT^2 > \Omega(n^2)$. But first we note that the existence of such functions allows us to easily construct F,G so that each is easy but whose composition is hard. To do this just define:

$G(x) = (x, g(x))$

and

$F(x,y) = f(x) \vee y.$

Then the point of these definitions is that $F(G(x)) = f(x) \vee g(x)$.

Lemma 1: For each n there is a n-vertex cubic graph G_n such that

(1) G_n is 3-edge colorable;

(2) for any partition of the vertices into two sets of size at least $n/3$ there are at least ϵn edges between these classes where ϵ is an absolute positive constant.

Part (2), of course, means that G_n is not easily separated. We will not give a detailed proof now: a full proof can be based on the probabilistic method of Erdos [3].

We plan to use G_n to construct our functions f and g. Let G_n be 3-edge colored with the colors red, green, and blue. Also, let us associate an input x_i with each vertex i of G_n. Then define the predicate S_α (α is red or green or blue) as follows:

S_α if and only if each edge (i,j) that is colored α has $x_i = x_j$.

Thus, S_{red} if and only if all the red edges have their end points equal.

Lemma 2: For each n and any α,β colors the predicate $S_\alpha \vee S_\beta$ can be done in $AT^2 = O(n)$.

Thus, the union of any two of these predicates is easy. (Note, this lemma uses one gate with large fan-in; without this the bound becomes $O(n lg^2 n)$.) The proof of our result depends then on:

Lemma 3: For each n, $S_{red} \vee S_{blue} \vee S_{green}$ requires $AT^2 > \Omega(n^2)$.

Clearly, once this is proved we are done: just let $f(x)$ be S_{red} and $g(x)$ be $S_{blue} \vee S_{green}$.

Proof of Lemma 3:

Consider a layout for $S_{red} \vee S_{blue} \vee S_{green}$. Construct the L,R partition as described in Section 3 (assume n even). Now by lemma 1, there are at least ϵn edges with one L end point and one R end point. Clearly, at least 1/3 of these are the same color: without loss of generality let us assume that it is red. Let these red edges form the set E.

We can now select (assuming n is large enough) one blue edge (b_1,b_2) and one green edge (g_1,g_2). Now fix these inputs so $b_1 \neq b_2$ and $g_1 \neq g_2$. Then $S_{red} \vee S_{blue} \vee S_{green}$ if and only if S_{red}. If we fix all those red edges (r_1,r_2) with both endpoints L or R, so that $r_1 = r_2$, then finally,

$S_{red} \vee S_{blue} \vee S_{green}$

if and only if

$x_i = x_j$ for all (i,j) in E.

Moreover, these x_i's are all unspecified and unrelated. Thus, the computation of $S_{red} \vee S_{blue} \vee S_{green}$ requires the checking of $\epsilon n/3$ distinct pairs of inputs for equality. Since in each such equality, one is L and one is R, a simple crossing sequence argument completes the proof of the lemma.

□

5. Extensions to the Model

In this section, we will discuss a number of extensions to the model of Section 3. The first two of these are suggested by the analogy with one-tape Turing Machines.

(a) Nondeterminism

A careful examination of our lower bounds shows that they still hold if the VLSI layouts are *nondeterministic*. While the idea of nondeterministic VLSI is fantasy, it is a comment on the basic nature of the lower bounds that they would hold even if nondeterministic VLSI layouts could be built. In a sense our lower bounds make no assumption on how things are computed on the left and right part of the layout, only about how they exchange information about their results. We will return to this later in this section.

(b) Randomness

In a similar vein, it is possible to prove that the lower bounds that have been proved for n-input, n-output functions also hold when *randomness* is allowed. This result is more difficult to prove, and it is more relevant to practical situations, since efficient algorithms which exploit randomness have recently been invented for some problems, and it is reasonable to contemplate the fabrication of VLSI circuits that implement randomness (probably more reliably than the "random number generators" in widespread use on general-purpose computers today).

The crux of this result is the following problem: can we use randomness to move an n-bit number from one place to another with less than 2^n crossing sequences. Note, randomness does help with the related but simpler problem: verify that two n-bit numbers are equal.

Now assume that randomness does help and it is possible to do it with $m < 2^n$ crossing sequence. Let

P_{ij} = *Prob* [on input i the j^{th} crossing sequence occurs]

($i = 1,...,2^n$ and $j = 1,...,m$). Clearly,

$$\sum_{i=1}^{m} P_{ij} = 1 \qquad (1)$$

for all $i = 1,...,2^n$. Also let

S_{kj} = *Prob* [the k^{th} crossing sequence outputs j]

($k = 1,...,m$ and $j = 1,...,2^n$).

By output we of course mean that j is transferred. Again, it is clear that

$$\sum_{j=1}^{2^n} S_{kj} = 1 \qquad (2)$$

for all $k = 1,...,m$. We insist that the probability of correctly answering is $\geq \delta > 1/2$. Thus,

$$\sum_{k=1}^{m} P_{ik} S_{ki} \geq \delta \qquad (3)$$

for all $i = 1,...,2^n$. We next claim that for each input j, there is a crossing sequence k so that $S_{kj} \geq \delta$. Fix an input j and assume that $S_{kj} < \delta$ for all k. Then by (1),

$$\sum_{k=1}^{m} P_{ik} S_{ki} < \delta$$

which contradicts (3); hence, our claim is true. Let $h(j)$ satisfy for all inputs j,

$$S_{h(j)j} \geq \delta; \qquad (4)$$

$h(j)$ exists by our claim. Since $m < 2^n$ by the pigeon-hole-principle it follows that

$$h(j_1) = h(j_2)$$

for some $j_1 \neq j_2$. Therefore, it follows that

$$\sum_{j=1}^{2^n} S_{kj} \geq S_{h(j_1)j_1} + S_{h(j_2)j_2} \geq 2\delta$$

where $k = h(j_1) = h(j_2)$; however, this contradicts (2) and it follows the assumption that $m < 2^n$ is false.

Thus, randomness does not help in transferring information from one place to another. However, the results of Section 4 on predicates are more sensitive. For example, the lower bound on the factor verification predicate does not hold when randomness is allowed. That such examples should exist is of course suggested by the analogy to one-tape Turing Machines. There, as here, randomness is sometimes more powerful than nondeterminism.

(c) Multi-arrivals

Still more realistic is the possibility of enhancing the efficiency of VLSI circuits by allowing for more general input-output schedules. If schedules are not where-oblivious, then it is possible to do the "hard" n-input, n-outputs functions in $AT^2 = 0(n)$. A more fruitful and realistic direction i to weaken one of our basic assumptions an allow inputs to arrive *multiple times*. Again, we can show that the lower bounds which have previously been proved for n-input, n-output functions still hold. The proof of this result is somewhat more complicated than our other VLSI lower bound proofs, because it requires multiple cuts of the VLSI layout.

This last extension is interesting since the results on union, composition, and alphabet change do *not* hold if such multiple arrivals of the inputs are allowed.

(d) Other Measures

So far, we have exclusively measured VLSI layouts by AT^2. What about other measures, such as AT^α? The following general meta principle turns out to be useful:

If $AT^2 > \Omega(n^2)$ is proved by the crossing sequence method for where-oblivious layouts, then $AT^\alpha > \Omega(n^{1+\alpha/2})$ for layouts that are also when-oblivious provided $0 \leqslant \alpha \leqslant 2$.

Thus, provided we are willing to assume that only when-oblivious input-output schedules are allowed, we can to see how well A can be traded-off for T.

Another class of measures for VLSI is the data rate D notion of Vuillemin. He shows for certain n-input, n-output functions that $A > \Omega(D^2)$. He claims that $AT^2 > \Omega(n^2)$ is *weaker*, but this is open as far as we can see. The following meta principle may also be of interest:

If $AT^2 > \Omega(n^2)$ is proved by the crossing sequence method, then $A > \Omega(D^2)$.

Thus, his results on data rate are really just results about AT^2 unless he can introduce a new proof technique.

We can also relate AT^2 to a notion of boolean circuit complexity. Let $P(f)$ denote the size of the smallest *planar* boolean circuit for f (see [6]). Then,

Theorem: $AT^2 > \Omega(P(f))$ for when and where oblivious layouts.

The proof of this is a simple one: we must just show how to simulation a VLSI layout of area A and time T by a boolean planar circuit. The only difficulty is that the layout may use feedback and this is not allowed in the planar circuit. Thus, the $\Omega(n^2)$ results obtained in [6] can be used to get lower bounds on AT^2. However, note that these results are weaker than those obtained here by crossing sequences. (This result had been obtained earlier by J. Savage.)

Finally, a word on the limitations of the crossing sequence method. Consider the problem of connectivity:

Given an $\sqrt{n} \times \sqrt{n}$ adjacency matrix of a graph (undirected).

Determine whether or not the graph is connected.

For awhile, we tried in vain to use the crossing sequence method to prove that this problem takes $AT^2 > \Omega(n^2)$. We then made two simple observations. Fix an arbitrary L,R partition of the entries of the input (the adjacency matrix):

(1) If the graph is connected, there is a *short* sequence of messages between L and R that can prove this.

(2) If the graph is disconnected, there is also a *short* sequence of messages between L and R that can prove this.

In (1) we have L *guess* a spanning tree and send it to R along with those edges he has; R then checks that he has all the remaining edges. In (2), we have L *guess* the two connected components. After checking that none of his edges join them, he sends the components to R for a similar check. Clearly, both require at most $0(\sqrt{n}\ lgn)$ bits.

The key then is to see that this means that there is a *non-deterministic* VLSI layout with $AT^2 = o(n^2)$ for both connectivity and its complement. Since the crossing sequence method cannot distinguish deterministic from non-deterministic layouts, we have proved:

Either

$AT^2 > \Omega(n^2)$ for connectivity by a *new proof technique*

or

$$AT^2 = o(n^2).$$

It has turned out, especially among graph properties, that this is not an isolated phenomenon. We can prove a similar faith for the following predicates:

has a cycle

planarity

strong connectivity

2-colorability

has a perfect matching

and others. Thus, each of these and their complements can be recognized with $AT^2 = o(n^2)$ by non-deterministic VLSI layouts. Either $AT^2 > \Omega(n^2)$ by a new proof technique or $AT^2 = o(n^2)$ for a real VLSI layout. For connectivity, as we pointed out in Section 2, the answer is known; for the rest it remains open.

6. Acknowledgements

We wish to thank J. Valdes and B. Arden for the helpful comments on this work. We also thank R. Kannan and M. Harrison and A. Odlyzko for allowing us to include results obtained earlier with them. Finally, we thank the Princeton students of "511" for their helpful comments.

7. References

[1] H. Abelson and P. Andreae, "Information Transfer and Area-Time Tradeoffs for VLSI Multiplication," *CACM* 23 pp. 20-23 (January 1980).

[2] R. P. Brent and H. T. Kung, "The Area-Time Complexity of Binary Multiplication," Report No. CMU-CS-79-136, Dept. of Comp. Sci., Carnegie-Mellon U., Pittsburgh, Penn. (July, 1979).

[3] P. Erdos and J. Spencer, *Probabilistic Methods in Combinatorics* Academic Press, 1974.

[4] F. C. Hennie, "On-line Turing Machine Computations," *IEEE Transactions on Electronic Computers* EC-15 pp. 35-44 (1966).

[5] J. E. Hopcroft and J. D. Ullman, *Formal Languages and Their Relation to Automata*, Addison-Wesley (1969).

[6] R. J. Lipton and R. E. Tarjan, "Applications of a Planar Separator Theorem," *SIAM J. of Computing* August 1980, vol. 9 no. 3 pp. 615-627.

[7] J. E. Savage, "Area-Time Tradeoffs for Matrix Multiplication and Related Problems in VLSI Models," *Proc. 17th Allerton Conf. on Communication, Control, and Computing*, pp. 670-676 (Oct. 10-12, 1979).

[8] C. D. Thompson, "A Complexity Theory for VLSI," Report No. CMU-CS-, Dept. of Comp. Sci., Carnegie-Mellon U., Pittsburgh, Penn. (Sept. 16, 1979).

[9] C. D. Thompson, "Area-Time Complexity for VLSI," *Procs. 11th ACM Ann. Symp. Th. Comp.*, pp. 81-88 (April 1979).

[10] J. Vuillemin, "A Combinatorial Limit to the Computing Power of VLSI Circuits," *Procs. 21st Ann. Symp. on Foundations of Comp. Sci.*, pp. 294-300 (Oct. 12-14, 1980).

[11] A. C. Yao, "Some Complexity Questions Related to Distributed Computing," *Procs. 11th ACM Ann. Symp. Th. Comp.*, pp. 209-213 (May 1979).