1-1989

Lower Bounds on the Area of Finite-State Machines

M. J. Foster

Ronald I. Greenberg
Loyola University Chicago, Rgreen@luc.edu

M. J. Foster and Ronald I. Greenberg. Lower bounds on the area of finite-state machines. *Information Processing Letters*, 1988. To appear.

Published in Volume 30, Issue 1, 16 January 1989, pages 1-7.

Lower Bounds on the Area of Finite-State Machines

M. J. Foster
Computer Science Department
Columbia University
New York City 10027

Ronald I. Greenberg
MIT Laboratory For Computer Science
Cambridge, MA 02139

September 20, 1988

Index terms

Custom VLSI, layout algorithms, regular languages, finite-state machines, lower bounds, silicon compilers

Abstract

There are certain straightforward algorithms for laying out finite-state machines. This paper shows that these algorithms are optimal in the worst case for machines with fixed alphabets. That is, for any s and k, there is a deterministic finite-state machine with s states and k symbols such that *any* layout algorithm requires $\Omega(ks \lg s)$ area to lay out its realization. Similarly, any layout algorithm requires $\Omega(ks^2)$ area in the worst case for nondeterministic finite-state machines with s states and k symbols.

1 Introduction

To manage the complexity of designing large VLSI chips, automated layout algorithms, sometimes called silicon compilers, are required. These are computer programs that accept a high-level behavioral description of a chip and produce its layout. There are many layouts corresponding to any behavioral specification, some better than others according to various measures. For example, some layouts may be very compact so that the chip can be small, some layouts may be very fast, so that the chip can work quickly, or some may be very low-power. We would like to be sure that our layout algorithms produce good layouts according to the measure we select.

Regular languages and finite automata have proven to be useful models for a wide range of computations. Many chips and portions of chips are commonly modelled as finite automata, and several systems have been built for automatic layout of these machines. Given a state-table or algorithmic description of a finite-state machine, these systems attempt to produce a small, fast layout on a VLSI chip. Techniques for layout have included regular expression trees [5, 4], networks of PLA's [8], and multi-level standard-cell logic [3].

This paper shows that straightforward techniques for layout of regular languages are optimal in the worst case. Although some of the techniques that are used in practice can produce small layouts for some examples, there are many finite state machines for which the straightforward implementation (such as direct coding of the state table) is the best possible. In particular, we show that for any s and k, any algorithm for laying out a finite-state machine must use area $\Omega(ks \lg s)$ for some deterministic machine

This research was supported in part by the Defense Advanced Research Projects Agency under contract N00014–87–K–0825. M. Foster was partially supported by the National Science Foundation as a Presidential Young Investigator and by the Center for Telecommunications Research at Columbia University. R. Greenberg was supported in part by a Fannie and John Hertz Foundation Fellowship.

with s states and k symbols. Asymptotically, this is the area of the direct state table implementation. Similarly, we show that $\Omega(ks^2)$ area is required for some nondeterministic machine with s states and k symbols, which corresponds to a straightforward implementation of nondeterministic machines.

This paper is organized to highlight the ideas behind the lower bounds. We prove a simple theorem that displays the method, then extend it to cover more complicated cases. In Section 2 we describe the model of computation in which these lower bounds hold. Section 3 proves the lower bound on the area of a fixed-shape region for layout of deterministic finite-state machines with 2 character alphabets, and Section 4 proves a similar lower bound for nondeterministic machines. Section 5 extends the results to machines with larger alphabets. In Section 6 we show that allowing different machines to occupy regions of different shapes does not decrease the area required.

2 Area Lower Bounds in VLSI

Why prove a lower bound on area? Several types of lower bounds on the resources needed by VLSI chips have been proven [1, 2, 6, 7]. Most of these have been bounds on the product of chip area and some power of the running time; AT^2 is common. In this paper we prove a lower bound for area alone, without considering time. Finite-state computations differ from the computations considered by previous authors (context-free language recognition, sorting, DFT, etc.); the area required by a computation on a string of length n is independent of n. Computations on which bounds have previously been proven (including an area lower bound on string matching [6]) have required increasing area as the length of the input increases. The memory required by a finite state machine is fixed, no matter how long the input may be. Moreover, the time taken by a finite-state computation is simply the time to read the input. As the length of the input increases, then, the computation time increases proportionally while the area stays constant, so that the tightest AT^k lower bound on a finite-state computation on an input of size n is simply $\Omega(n^k)$. Therefore, we can characterize the implementation of a finite-state machine strictly in terms of its area.

To prove a lower bound on area, we need a model of computation that relates area to the problem to be solved. We will use a model similar to the one introduced by Thompson [7]. Any computational structure must be laid out on a grid of unit squares, corresponding to the minimum feature size of the implementation technology. Each of the unit squares may contain one or more types of conducting or insulating material in several layers. There is a finite number of possible conducting or insulating layers, and a unit square may contain some combination of these layers. The combination chosen for a particular unit square can be specified by a fixed number of bits, so the set of combinations chosen for an area A can be specified by a number of bits proportional to A. The set of layers chosen for each square of the chip determines its structure completely, which determines the computation performed. We have therefore proven the following fundamental lemma.

Lemma 1 *The computation performed by an area of size A can be specified with $\Theta(A)$ bits.*

In a typical CMOS implementation, for example, the unit squares are typically about one micron on a side, and each square may contain p or n-type diffusion, polysilicon, and one or two layers of metal. There may also be insulators present to isolate these layers, or the insulators may be removed to make a contact. There are about 60 combinations of conducting layers and insulators that make electrical sense, so the configuration of each unit square can be specified by 6 bits. Therefore, the computation performed within an area of m square microns can be specified by at most $6m$ bits. Of course, many of the possible combinations will not perform meaningful computations, since neighboring squares might violate design rules, or the layout as a whole might not be a working circuit. The constant of proportionality is therefore less than 6, but it is clear that the computation performed by a CMOS chip of area m can be specified by $\Theta(m)$ bits.

To prove a lower bound of area $\Omega(n)$ using this model, we show that the computation we want to perform requires n bits of specification. Suppose that we are given a chip of area A, together with a family of 2^n different computations, any one of which may be specified as the circuit to be built on the chip. Since

the computations are all different, each one requires a different structure for area A. The structure of area A is thus specified by n bits, so A must be $\Omega(n)$.

In using this proof technique, notice that the computations themselves must be different, not just the computational structures. If two different structures on the same chip produced the same behavior, an implementor could always use just one of them in implementing a computation. By counting behaviors rather than structures, we can be sure that we need a different chip for each different computation.

The results we obtain from this technique are worst case bounds. They show that at least one member of the family of 2^n computations will require area n. The best case could of course be considerably better. For example, we show below that the worst case area for laying out an s-state deterministic machine is $\Omega(s \lg s)$. Many s-state machines have considerably smaller area, requiring as little as $O(\lg s)$ area for the best layout. The bounds in this paper, while existentially tight, do not eliminate the search for clever implementations.

3 Deterministic Automata

There is a well-known method for simulating a deterministic finite automaton (DFA) with a program for a general-purpose computer. States of the machine are represented by integers, which are used as indices into a transition table. Position i of the table contains the next states for all input characters if the machine is in state i. If the program is in state i, it selects its next state from position i in the table, and uses that state to index into the table on the next character input.

This program structure can be implemented in hardware in a straightforward fashion using a PLA. The PLA for a machine with s states and 2 symbols in the alphabet will have $\lg s$ input and output columns to represent the states as well as an input line for the alphabet symbol. The number of rows (product terms) in the PLA will be $2s$, one for each combination of state and input symbol. The output state bits are computed by ORing the appropriate product terms into each state bit. The total area of this hardware realization of the finite-state machine is $O(s \lg s)$.

Of course, for many machines the PLA implementation is extremely wasteful of space. For example, an n-bit counter has 2^n states so that its PLA representation would require area $n \times 2^n$. A simple ripple-carry implementation of the same counter, however, requires only area n. A clever layout algorithm for finite-state machines might be able to recognize that a particular machine was a counter and use the smaller implementation. There are other tricks of this kind for other types of machines, such as machines that recognize patterns having short regular expressions [4, 5, 8]. Do all finite-state machines succumb to these kinds of techniques, or are there machines for which the obvious PLA layout is the best?

In the worst case, the PLA layout is the best that can be done. We can show this by producing a set of s^s different languages that can each be accepted by machines with s states. It is not sufficient to simply note that there are s^{2s} different state tables for s-state machines, since some of these state tables represent equivalent machines. A clever layout algorithm could implement equivalent machines in the same way. We must count languages or behaviors, not machines. Keeping this in mind, we can prove the following theorem, which provides a lower bound matching the upper bound for area of an s state DFA.

Theorem 2 *The amount of area required to implement an s state deterministic finite automaton is $\Omega(s \lg s)$ in the worst case.*

Proof. Consider the following class, F_s, of s state machines with input alphabet $\{0, 1\}$. Each machine has states numbered 0 to $s-1$, where state 0 can be distinguished from all others by observing the output of the machine; it can be thought of as a final state if the machine is used as a recognizer. The machines act identically in every state on input 0; specifically, if a machine is in state i and gets an input of 0, it goes to state $i+1 \bmod s$. On input 1, however, all possible behaviors are included among the machines in F_s. Since the transition on input 1 from each of the s states can be to any one of s states, there are s^s machines in F_s.

The machines in F_s represent different languages, so they can be distinguished by their behaviors. To see this, consider two different machines in F_s. There must be some state i for which a 1 input takes the machines to different states, say j and k. We can test for the presence of an edge from i to j by inputting zeros to put the machine into its distinguished state, then inputting the string $0^i 10^{s-j}$. The machine will be in its distinguished state at the end of this input if and only if there is a transition from state i to state j on input 1. Thus if two machines in F_s differ, there is a string accepted by one but not by the other.

The machines in F_s all behave differently, so any layout algorithm must produce different layouts for each one. Since there are s^s machines in this set, $s \lg s$ bits are required to specify one of the layouts. Therefore, an area A that can contain the layout of any of the machines in F_s must be of size $\Omega(s \lg s)$. ■

4 Nondeterministic Automata

Floyd and Ullman [4] presented a generic layout for a non-deterministic finite automata (NFA) that is analogous to the layout given above for deterministic machines. The only difference is that instead of encoding the state into $\lg s$ PLA inputs and outputs, one bit is used for each of the s states. After each input symbol is received, the PLA output corresponding to a given state is on if and only if the non-deterministic machine could be in that state. The area of the Floyd-Ullman implementation of an s-state NFA over a two-symbol alphabet is $O(s^2)$ since there are s input and output terms and at most $2s$ product terms.

Is there a better way to implement non-deterministic machines? In a proof similar to that for the deterministic case, we can show that $\Omega(s^2)$ is a lower bound on the layout area.

Theorem 3 *The amount of area required to implement an s state nondeterministic finite automaton is $\Omega(s^2)$ in the worst case.*

Proof. Once again, consider a set F_s of machines over the two-symbol alphabet $\{0, 1\}$. The states of each machine are numbered 0 to $s-1$, with state 0 distinguishable by the output of the machine. The set F_s consists of those machines such that the only 0-edge out of state i goes to state $i+1 \bmod s$. Every possible combination of 1-edges occurs in some machine in F_s. Since the transitions on input 1 from each state can be to any subset of the s states, F_s contains $(2^s)^s$, or 2^{s^2} machines.

The machines in F_s all recognize different languages, by the same argument used in the deterministic case. If two machines differ, there must be a 1-edge present in one machine that is not present in the other one. Thus we can distinguish different machines by testing for the presence of a single 1-edge. As above, to determine whether the 1-edge from state i to state j is present in a machine, we place the machine in state 0, then input the string $0^i 10^{s-j}$. Since the machines in F_s are deterministic on 0 inputs, the machine will be in state 0 after this input if and only if there is a 1-edge from i to j.

Since the machines in F_s represent different languages, they must all have different layouts. The specification of a non-deterministic machine with s states is therefore a specification of one of 2^{s^2} different layouts. By the arguments given in the preceding sections, the smallest chip that can contain any of these layouts has area $\Omega(s^2)$. ■

5 More Than Two Symbols

If a machine has more than two symbols in its alphabet, both the upper and lower bounds for layout area increase. An s-state deterministic machine with k symbols in the alphabet has a PLA implementation with area $O(ks \lg s)$. This is easy to see when k is $O(s)$. Then we use a construction just like the earlier one except that there are $\lg s + \lg k$ inputs in order to account for both the states and input symbols, and there are ks product terms. If k is not $O(s)$, then we can use an alternative construction which yields an area bound of $O((k + \lg s)s \lg s)$, which is $O(ks \lg s)$ for $k = \Omega(\lg s)$.

The alternative upper-bound construction is obtained by using a PLA in which the input symbol has been decoded, i.e., it is represented by turning on exactly one of k input lines. (It is actually desirable to input the complements of these signals to the PLA.) Each product term corresponds to the pairing of a state and an output state bit which is turned on by some transitions from that state. The product term includes the check that the input symbol is not one which fails to generate such a transition. Since there are $s \lg s$ product terms and $k + \lg s$ input terms, the area of the PLA is $O((k + \lg s)s \lg s)$.

To decode input symbols specified with $\lg k$ bits into the k-bit representation, we just need to add the area of a decoder, which is $k \lg k$. The decoder is certainly small enough if $\lg k$ is $O(s \lg s)$. If $\lg k$ is greater than $O(s \lg s)$, then some of the input symbols are extraneous. That is, there are only s^s complete state-transition behaviors which can be assigned to the input symbols, so if there is a larger number of input symbols, there must be symbols with identical behaviors. Thus it suffices to translate the $\lg k$ input signals into $s \lg s$ signals corresponding to the appropriate state-transition behavior in binary (requiring area $s \lg s \lg k$), and then decode these $s \lg s$ signals into the appropriate s^s signals in unary (requiring area $s^s s \lg s$, which is $O(ks \lg s)$ for $\lg k = \Omega(s \lg s)$).

The lower bound on deterministic machines with s states and k symbols is also $\Omega(ks \lg s)$, as we now show.

Theorem 4 *The amount of area required to implement an s state, k symbol DFA is $\Omega(ks \lg s)$ in the worst case.*

Proof. Consider the family F_s, extended so that each state has k outgoing edges instead of just two. For each of the s states, one of the outgoing edges (the 0-edge) is constrained, but each of the other $k - 1$ edges can go to any one of s next states. Each state therefore has s^{k-1} possibilities, and since all states are independent the number of possible machines is $(s^{k-1})^s$. The machines are all different, since the presence of any edge can be tested by experiments on the machine. The number of bits required to specify one of these machines, and thus the layout area required, is therefore $\Omega(ks \lg s)$. ■

The extension to k symbols is similar in the case of nondeterministic finite automata. The area of an s state NFA with k symbols is $O(ks^2)$. The straightforward modification of Floyd and Ullman's construction gives a PLA with $s + \lg k$ input terms and ks product terms for an area of $O((s + \lg k)ks)$, which is $O(ks^2)$ when s is $\Omega(\lg k)$. If s is not $\Omega(\lg k)$, we can use an alternative construction which yields an area bound of $O((k + s)s^2)$, which is $O(ks^2)$ when s is $O(k)$.

The alternative upper bound is obtained in the same fashion as described above for deterministic machines. When decoded alphabet symbols are input to the PLA, the number of inputs plus outputs is $O(k + s)$, and the number of product terms is s^2, for an area of $O((k + s)s^2)$. Again we can handle input symbols which have not been decoded with a decoder of area $k \lg k$. The area of the decoder is $O(ks^2)$ if s is $\Omega(\sqrt{\lg k})$, and otherwise we have redundant symbols. That is, there are only 2^{s^2} possible state-transition behaviors for each symbol. Thus the $\lg k$ input bits for the alphabet symbols can be translated into s^2 bits representing the state-transition behavior and then decoded into 2^{s^2} bits, all in area $s^2 \lg k + s^2 2^{s^2}$, which is $O(ks^2)$ for $s = O(\sqrt{\lg k})$.

As before, we obtain a lower bound which matches the upper bound on area.

Theorem 5 *The amount of area required to implement an s state, k symbol NFA is $\Omega(ks^2)$ in the worst case.*

Proof. We once again consider the family of machines F_s, in which 0-edges form a cycle. There are $(2^s)^{s(k-1)}$ machines in F_s since from each state, there are 2^s possible combinations of edges for each character. A similar proof to those above shows that these machines all recognize different languages, and thus must have different layouts, so the area lower bound for a non-deterministic machine with s states and a k-symbol alphabet is $\Omega(s^2 k)$. ■

6 Shape independence

The results which we have given so far hold in the case of laying out machines in a fixed region such as a square. The results depend only on the mild assumption that there is a minimum size grid, with each grid square having a finite number of possible compositions.

In fact, stronger assumptions are reasonable for VLSI technologies, and we can show that all the previous results hold even if we allow each machine to be implemented in a different area of arbitrary shape. In VLSI technologies, adjacent grid squares containing conducting material on the same layer are electrically connected, and there is no other relevance to the shape in which the grid squares are arranged as long as the design rules are obeyed. It is therefore possible to view a VLSI layout as a bounded-degree graph of grid squares of various compositions.

We extend our earlier results by invoking the following lemma, which states that $\Theta(A)$ bits suffice to specify all graphs of wire area A (i.e. A grid squares).

Lemma 6 *There are $2^{O(A)}$ VLSI layouts of wire area A.*

Proof. We begin by invoking a result of Thompson that a graph of wire area A must have a bisection width of $O(\sqrt{A})$ [7]. Then we can write a recurrence for $G(A)$, the number of graphs of wire area A as follows:

$$G(A) \leq [G(A/2)]^2[(A/2)^2]^{O(\sqrt{A})} .$$

The right side of the recurrence is an upper bound on the number of ways of forming two graphs of wire area at most $A/2$ and then forming connections across the bisection. Letting $H(A) = \lg G(A)$, we can rewrite the recurrence as

$$H(A) \leq 2H(A/2) + O(\sqrt{A} \lg A) .$$

Then noting, that $G(1)$ is a constant, we can solve the recurrence to obtain $H(A) = O(A)$, and $G(A) = 2^{O(A)}$. ∎

7 Conclusion

This paper has presented results on area lower bounds for finite state automata. We have shown that any algorithm for laying out finite automata must require as much area for some machines as do the most straightforward algorithms. Our results can be extended easily to volume lower bounds in 3D-VLSI circuits, as long as there is a minimum feature size.

Can our bounds be tightened? Our lower bounds are tight for only the worst case machines with a given number of states. However, there are finite-state machines with s states that can be laid out in considerably less than $s \lg s$ area. For example, an s-state counter can be laid out in area $\lg s$. What features of finite state machines let them have small layout area? It would be desirable to obtain a simple measure of complexity of finite state machines in terms of which we could obtain universally close upper and lower bounds on area.

Even if tighter bounds are found for some types of finite state machines, it is clear that almost all machines will require the area that our bounds specify. For example, consider the set of s-state deterministic machines that have layouts of area s. There are at most 2^s different machines that can be laid out in area s, so the fraction of s-state machines that can have area s layouts is at most $2^s/s^s$, which is asymptotically zero. Thus, almost all deterministic machines with s states require more than area s. Similar arguments show that, for any area lower bound $L(s)$ in this paper and for any function $f(s)$ such that

$$\lim_{s \to \infty} \frac{2^{f(s)}}{2^{L(s)}} = 0$$

almost all machines with s states require more than $f(s)$ area.

Tighter bounds than ours must depend upon machine features which are present in a vanishingly small proportion of s-state machines. Nonetheless, machines of practical interest may contain these features, so a search for tighter bounds is worthwhile.

Acknowledgements

Thanks to Lane Hemachandra of Columbia for stimulating discussions.

References

[1] R. P. Brent and L. M. Goldschlager. Area-time tradeoffs for VLSI circuits. In *Microelectronics '82*, pages 52–56, The Institution of Engineers, Australia, May 1982.

[2] R. P. Brent and H. T. Kung. On the area of binary tree layouts. *Information Processing Letters*, 11(1):46–48, August 1980.

[3] S. Devidas, H. Ma, A. R. Newton, and A. Sangiovani-Vincentelli. Mustang: state assignment of finite-state machines for optimal multi-level logic implementation. In *ICCAD-87*, pages 16–19, IEEE, November 1987.

[4] R. W. Floyd and J. D. Ullman. The compilation of regular expressions into integrated circuits. *JACM*, 29(3):603–622, July 1982.

[5] M. J. Foster. A specialized silicon compiler and programmable chip for language recognition. In N. G. Einspruch, editor, *VLSI Design*, chapter 5, pages 139–196, Academic Press, Orlando, Florida, 1986.

[6] O. Sykora and I. Vrto. Tight chip area lower bounds for string matching. *Information Processing Letters*, 26(3):117–119, November 1987.

[7] C. D. Thompson. *A Complexity Theory for VLSI.* PhD thesis, Carnegie-Mellon University, 1980.

[8] H. W. Trickey. Good layouts for pattern recognizers. *IEEE Transactions on Computers*, C-31(6):514–520, June 1982.