THE COMPLEXITY OF VLSI CIRCUITS FOR ARBITRARY BOOLEAN FUNCTIONS

M.R. Kramer and J. van Leeuwen

RUU-CS-82-7

May 1982

Rijksuniversiteit Utrecht

Vakgroep informatica

Princetonplein 5
Postbus 80.002
3508 TA Utrecht
Telefoon 030–531454
The Netherlands

THE COMPLEXITY OF VLSI CIRCUITS FOR ARBITRARY BOOLEAN FUNCTIONS

M. R. Kramer and J. van Leeuwen

Technical Report RUU-CS-82-7

May 1982

Department of Computer Science
University of Utrecht
P.O. Box 80.002, 3508 TA Utrecht
The Netherlands

THE COMPLEXITY OF VLSI CIRCUITS FOR ARBITRARY BOOLEAN FUNCTIONS

M.R. Kramer and J. van Leeuwen

Department of Computer Science, University of Utrecht
P.O. Box 80.002, 3508 TA Utrecht, the Netherlands

Abstract. It is well-known that all Boolean functions of n variables can be computed by a logic circuit with $O(2^n/n)$ gates (Lupanov's theorem) and that there exist Boolean functions of n variables which require logic circuits of this size (Shannon's theorem). In this paper we prove the corresponding, but different results for Boolean functions computed by VLSI circuits, using Thompson's model of a VLSI chip. First we provide a simple proof of Lupanov's theorem and point out why the resulting construction of a (logic) circuit is not suited for VLSI. As a proper analog of Lupanov's theorem for VLSI we prove that all Boolean functions of n variables can be computed by a VLSI circuit of $O(2^n)$ area and period n. We also prove that all Boolean functions can be computed by a VLSI circuit of $O(2^n)$ area and period 1. As an analog to Shannon's theorem we prove that there exist Boolean functions of n variables for which every (convex) VLSI chip must have $\Omega(2^n)$ area.

Keywords and phrases: logic circuit, Boolean function, Lupanov's theorem, Shannon's theorem, VLSI, chip, area, period.

1. Introduction.

The design and analysis of logic circuits for Boolean functions has long been of interest to engineers and computer scientists. (See e.g. Savage [5] for a good account of the known theory.) In recent years the study of circuits has entered the centre of attention again, as advances in LSI- and

VLSI-technology have made it feasible to design and implement much larger circuits of basic switching elements than ever before on a small chip of silicon. (See e.g. Mead and Conway [3] for technical details.) The requirements for VLSI circuits are not as relaxed as they are for logic circuits. For example, a VLSI circuit must be embedded in the plane and occupy only a small amount of area, counting what it takes both for gates and wires. Also there can be no unbounded fan-in or fan-out, as basic switching elements can handle at most 4 wires for signal routing each in all common physical designs. In this paper we study the quantitative effect of these differences on the "circuit complexity" of arbitrary Boolean functions.

It is well-known that all Boolean functions of n variables can be computed by a logic circuit of $O(2^n/n)$ gates (Lupanov [2]) and that there exist Boolean functions of n variables for which any logic circuit actually requires $\Omega(2^n/n)$ gates (Shannon [6]). We shall prove that suitable analogs of these results hold when VLSI circuits are used, but with the $2^n/n$ replaced by 2^n in the bounding expressions.

To prove upper- and lowerbounds on the VLSI complexity of Boolean functions we adopt Thompson's simplified model of a VLSI chip ([7]). Essentially a chip is a bounded convex region in the plane, with the plane divided into unit size cells as in the regular grid (see figure 1). Each cell may contain one basic switching element or (at most) two crossing wires. Wires connect processing elements, and are constraint to run in strictly horizontal or vertical directions. (Wires can bend from one direction into another in any cell, but can only cross and never run on top of one another.) The area of a chip is defined as the area of the grid that it occupies, which is the (finite) number of unit size cells needed to lay out its circuit and to route all wires. (Because we as-

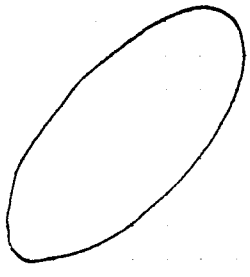


figure 1

sume the chip to be convex, a small proportion of unoccupied cells may be included in the count.) There are various notions of "computation time" for VLSI circuits. We simply assume, like Thompson, that all switching elements act synchronously and that signals take unit time to propagate over any single wire regardless its length. The period of a circuit is the minimum interval of time that must be between consecutive sets of input data when the circuit is pipelined. We use the standard notations: A for area, T for time, and P for period.

The paper is organised as follows. In section 2 we recall some necessary concepts from the theory of Boolean functions and provide a simple proof of Lupanov's theorem. It is included because the standard proof in the literature (cf. Savage [5], pp. 116-119) seems overly complex, and for having the opportunity to point out why the resulting construction of a logic circuit is not suited for VLSI. In section 3 we prove that all Boolean functions of n variables can be computed by a VLSI-circuit of $O(2^n)$ area and period n. By a different construction we prove that the period can in fact be reduced to 1, at the expense of only a constant factor in extra area. In section 4 we prove that there exist Boolean functions of n variables for which any (convex) VLSI circuit requires $\Omega(2^n)$ area.

2. Preliminaries, and a classroom proof of Lupanov's theorem.

This section gives a virtually self-contained summary of the notions from the theory of Boolean functions required for an understanding of this paper.

A minterm in $x_1, \ldots, x_n$ is any expression of the form $x_1^{\sigma_1} \ldots x_n^{\sigma_n}$ with $\sigma_i \in \{0,1\}$ $(1 \le i \le n)$, where $x_i^0 \equiv \overline{x}_i$ and $x_i^1 \equiv x_i$. Observe that $x_1^{\sigma_1} \ldots x_n^{\sigma_n} \equiv 1$ if and only if $x_i \equiv \sigma_i$ for all i. If $f(\sigma_1, \ldots, \sigma_n) = 1$, then $x_1^{\sigma_1} \ldots x_n^{\sigma_n}$ is called a minterm for f. It is well-known (and easy to prove) that <u>every Boolean function f is the sum of its minterms</u>. As there are only 2^n distinct minterm expressions in all, the resulting expression

for f is necessarily finite. Clearly a Boolean function f is uniquely determined by the choice of its set of minterms. As there are 2^{2^n} distinct sets of minterms, there are 2^{2^n} distinct Boolean functions of n variables.

The representation by minterms can be rephrased in computational term. Consider the lexicographic tree of minterms, modified in such a way that the nodes "compute" the proper x_i or $\bar{x}_i$ and "and" it to the partial term values in x_1 to x_{i-1} $(1 \le i \le n)$. The resulting tree will be called

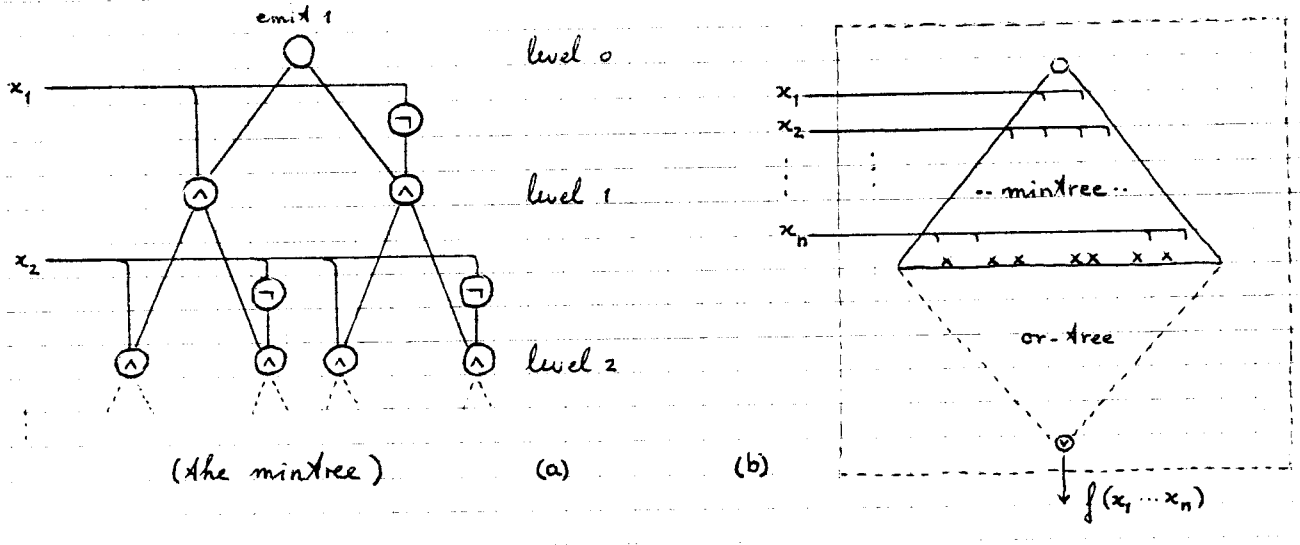


figure 2

the (general) mintree for n variables and is shown in figure 2(a). It actually is a circuit, with $2^i + 2^{i-1}$ logic gates in the i^{th} level and hence $\sum_1^n (2^i + 2^{i-1}) = 3(2^n - 1)$ gates total. Every leaf of the mintree is easily seen to correspond to a unique minterm in x_1 to x_n. By the earlier representation theorem it follows that every Boolean function of n variables corresponds to a selection of leaves of the mintree. Let the selected leaves be marked (by $\times$, see figure 2.b) and suppose another tree circuit is added that sums ("or-s") the computed values at the marked leaves. The resulting circuit is shown in figure 2(b) and clearly is a logic circuit of $O(2^n)$ gates for computing any Boolean function by the proper masking of components. We shall ignore the "or-" tree for our further

analysis and concentrate on the mintree part. By optimizing it we shall be able to derive Lupanov's theorem ([2]) in a simple way.

Let f be an arbitrary Boolean function in x_1 to x_n and let T be the mintree with marked leaves representing f. Cut T at level s, and

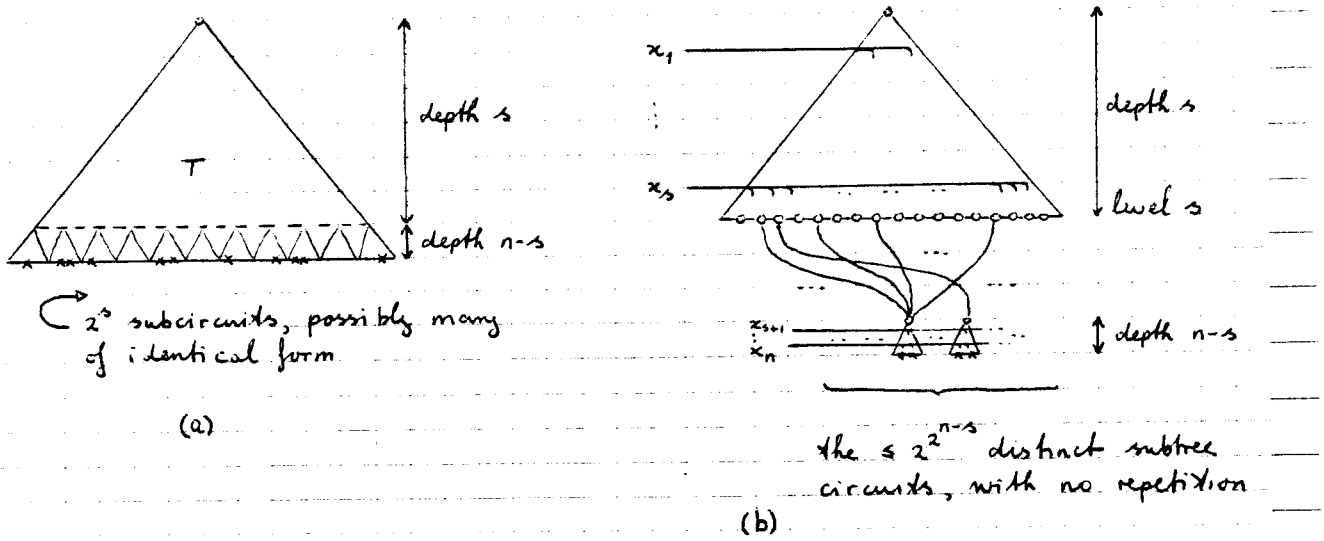


figure 3

consider the 2^s subcircuits (subtrees) attached to the nodes in this level (The value of s will be fixed later, but will be "almost equal" to n.) As circuits with input x_{s+1} to x_n the subtrees are identical, but they may differ through their marking. Because the subtrees are small but large in number, many must have the same marking and (hence) perform essentially the same computation. Thus, to save gates, we might as well split the subcircuits off from T and weed out all duplicates, and connect every intermediate node of level s to the proper and unique copy of the subcircuit that was originally attached to it. (Note that this introduces "unlimited" fan-in into the circuit, as many wires may get connected to the same top-gate of a subtree representative.) See figure 3 (a)(b) for a pictorial illustration of this optimization of T. When phrased in terms of Boolean functions rather than circuits in x_1 to x_s and x_{s+1} to x_n

the construction is known as <u>the "Lupanov decomposition"</u> of f. Observe that every subcircuit of depth $n-s$ has 2^{n-s} leaves, and thus there are $\leq 2^{2^{n-s}}$ distinct subcircuits if we take all possible markings into account. We can now obtain Lupanov's theorem by the proper choice of s.

<u>Theorem</u> 2.1 (Lupanov, 1958) Every Boolean function of n variables can be computed by a logic circuit of $O(2^n/n)$ gates, with no unlimited fan-in's.

<u>Proof</u>

Count the number of gates in the Lupanov decomposition of f's circuit. There are $3.(2^s-1)$ gates in the top part. Next there are $\leq 2^{2^{n-s}}$ small tree circuits with $3.(2^{n-s}-1)$ gates each. Suppose i_j nodes from the original level s are connected to the j^{th} subtree circuit ($1 \leq j \leq 2^{2^{n-s}}$). To avoid the unlimited fan-in, we can combine them through an or-tree with $i_j - 1$ gates. This accounts for an extra of $\sum_j (i_j - 1) < \sum_j i_j = 2^s$ gates. Finally the results at the marked leaves must be summed in an or-tree to obtain $f(x_1 \cdots x_n)$ as output. Because we optimized the lower part of the mintree, there may actually be fewer marked nodes in the subcircuits altogether that still (correctly) represent f. Note that each of the $2^{2^{n-s}}$ subtrees can be paired to one with a complementary marking, to contribute a total of 2^{n-s} marked leaves. Thus there are at most $\frac{1}{2}\, 2^{n-s}.\, 2^{2^{n-s}}$ marked leaves, and the or-tree summing for f needs to have no more gates than this. The total number of gates in the circuit thus is bounded by

$$3.(2^s-1) + 3.(2^{n-s}-1)\, 2^{2^{n-s}} + 2^s + \tfrac{1}{2}\, 2^{n-s}.\, 2^{2^{n-s}} \leq$$

$$\leq 4.2^s + 3\tfrac{1}{2}.\, 2^{n-s}.\, 2^{2^{n-s}}$$

$$\leq 4.\left(2^s + 2^{n-s}.\, 2^{2^{n-s}}\right)$$

Now choose s (integer) such that $2^{n-s} \leq n - \alpha \log n \leq 2^{n-s+1}$, for some α to be fixed later. It follows that $2^s \leq 2 \cdot 2^n/n - \alpha \log n$ and $2^{n-s} \cdot 2^{2^{n-s}} \leq$ $\leq (n - \alpha \log n) \cdot 2^n/n^\alpha \leq 2^n/n^{\alpha-1}$. Thus the number of gates in the circuit is bounded by

$$4 \cdot \left(2 \cdot 2^n/n - \alpha \log n + 2^n/n^{\alpha-1} \right) =$$

$$= 2^n/n \cdot \left(8 \cdot n/n - \alpha \log n + 4/n^{\alpha-2} \right) =$$

$$= O(2^n/n)$$

, for any $\alpha \geq 2$. □

(There are several further savings that can bring down the constant factor in the $O(2^n/n)$ term to about 1.) While the proof is quick and simple and of interest in its own right, it has been included as a prelude to the construction of an area/time-efficient circuit for VLSI. An appraisal (and rejection) of Lupanov's construction for the purposes of VLSI will be given in the next section.

3. Area/time-efficient VLSI circuits for Boolean functions.

We now return to the representation of a Boolean function f by a min-tree (cf. figure 2.b) but change our point of view to VLSI. We adopt all conventions as laid down in Thompson's model except that, for the moment, we will allow unlimited fan-outs by free "tapping" of a wire. The main issues are not gate counts, but area and time. The reason for representing f by a tree-based circuit will now become apparent. Trees

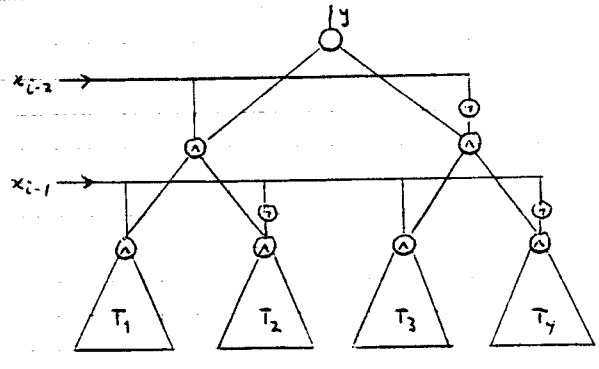


figure 4

are among the graphs that are easiest to lay out (see e.g. Thompson [7] and only require "linear" area. The circuit lay-outs presented in this section all derive from the common H-pattern design for (perfect) binary trees originally proposed by Mead and Rem [4].

Consider the representation of an arbitrary Boolean function f as suggested in fig 2 (b). For reasons of convenience we assume n even. The circuit of fig 2 (b) has a recursive structure that can be exploited to obtain an area-efficient lay-out. The essential idea is shown in figure 4. Build up the tree from the lowest level by adding two variables at the time, with the necessary logic to gate them into the four identical (but perhaps differently marked) subtrees that are needed on the variables x_i to x_n. Figure 5 shows the suggested lay-out, using the recursive assump-

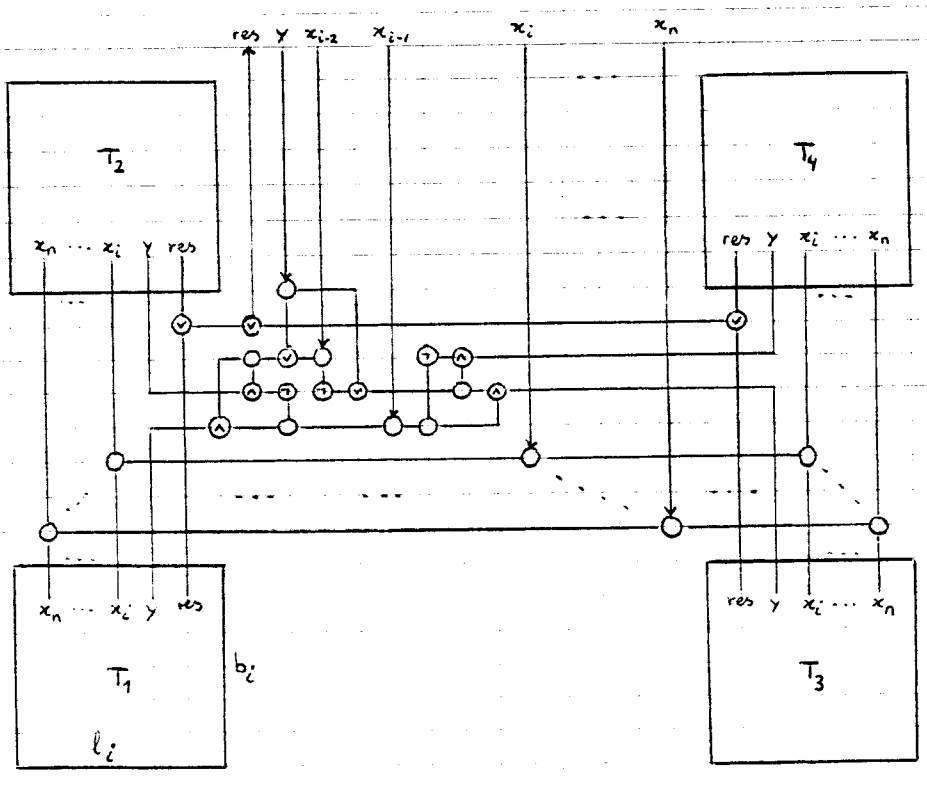


figure 5.

tion that area-efficient embeddings of $T_1, \ldots, T_4$ have been obtained in the same manner. Observe that an additional wire "res" is used to gate the result from the marked leaves back. (Two lay-outs are actually mirrored, to

align wires properly for connection. The o-cells serve to "tap" signals with no assumed time-loss.) The circuit C with inputs x_1 to x_n so obtained computes f and outputs $f(x_1, \cdots, x_n)$ on wire "res".

Proposition 3.1 Every Boolean function of n variables can be computed by a VLSI-circuit of $O(2^n)$ area, with a compute time and a period of both $O(n)$ when unlimited fan-outs with no time-loss are permitted.

Proof.

Consider the circuit $C = C_1$ obtained through the recursive construction just described. Let the i^{th} intermediate circuit C_i (starting with i equal to $n-1$) fit in a rectangle of size $b_i \times l_i$. From figure 5 we conclude that $b_{i-2} = 2b_i + (n-i) + 5$ and $l_{i-2} = 2l_i + (n-i) + 9$. (The precise additive constants are not important; smaller ones can be achieved.) Rewrite these equations as $\{b_{i-2} + (n-i+2+7)\} = 2\{b_i + (n-i+7)\}$ and $\{l_{i-2} + (n-i+2+11)\} = 2\{l_i + (n-i+11)\}$ respectively, then it easily follows that $b_1 = O(2^{n/2} - n)$ and $l_1 = O(2^{n/2} - n)$. Hence C_1 occupies $O(2^n)$ area.

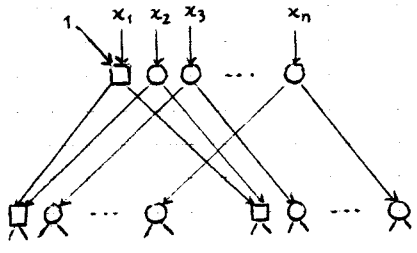
The time and period of the circuit very much depend on our assumption of unlimited fan-out of the input lines. It is easily seen from figure 2(b) that the compute time of the circuit is about $2n$: it takes n units of time for the signals to go down the mintree to reach the leaves, and another n to sum the results in the or-tree. Note that it takes n time units before an entire tuple x_1 to x_n is used and that, from this point of view, the period is precisely n. (The or-tree is trivially pipelineable.) On the other hand, after an x_1 (x_2, ..) of one tuple is used one can immediately input the x_1 (x_2, ..) of the next tuple and thus achieve a "period 1" under the skewed input convention. □

Proposition 3.1 proves the point that the mintree is not unreasonable for VLSI purposes. We shall soon improve it to achieve a period $O(1)$ with

no special assumptions.

The construction of circuit C suggests that a Lupanov-type optimization, while lowering the gate count, does not immediately lead to desirable effects for VLSI: (i) the many subtree circuits that Lupanov optimizes are all in the lowest part of the mintree and (hence) uniformly spread over the chip, and it would require much extra area and long wires to take the circuits "out" and set them apart for general access, (ii) by the same token it would lead to many more wire-crossings, which does not enhance the practicality for VLSI production and (iii) the unlimited fan-in (or: the or-trees that simulate it) is not equal at all place and (thus creates severe problems of synchronization if the circuit is to be pipelined. Redundancy has never been an issue in logic circuits, but it may be very attractive and important to have in VLSI circuits!

We shall now rid ourselves of the assumption of unlimited fan-out (which is not permitted by the strict rules of Thompson's model) and also improve the bound on the period. Rather than input the x_1 to x_n one after the other at separate (consecutive) levels of the mintree, we propose to input them simultaneously in parallel at the top and distribute the signals (again simultaneously in parallel) down the tree while the computation of partial minterms in x_1 to x_i $(i \to n)$ is taking place along the way. The idea is given in figure 6, where □-cells are used to represent the subcircuit that appends a next x_i to the computed terms at level i. Recursively continuing it leads to another tree that computes all minterms at its leaves, the so-called parallel mintree. It is clear that every Boolean function f of n variables can be represented by marking the leaves of the parallel mintree corresponding to its minterms.



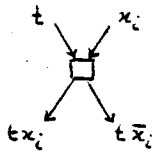


figure 6

The parallel mintree again has a nice recursive structure that makes it possible to apply the H-pattern for obtaining an area-efficient lay-out (see figure 7). The technique is very similar as for proposition 3.1 and consists of adding two variables x_{i-1} and x_{i-2} at a time (for $i \to 1$) while combining four recursively obtained subcircuits in x_i to x_n. The

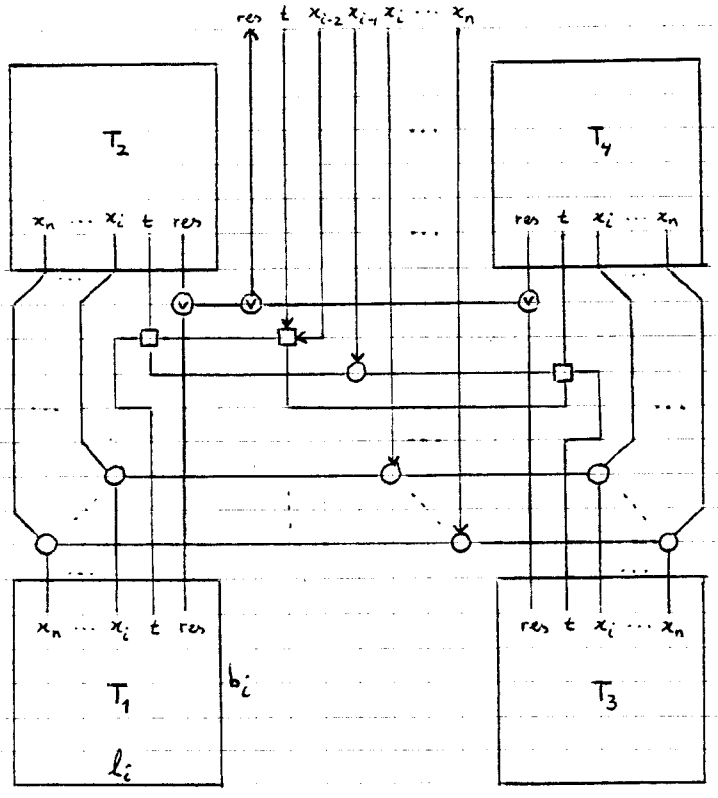


figure 7

□-cells in figure 7 actually are small circuits of some bounded size. The design uses a "t-" wire for accumulating partial minterms on the way down the tree, and a "res-" wire for summing the results from the marked leaves.

Theorem 3.2 Every Boolean function of n variables can be computed by a VLSI-circuit of $O(2^n)$ area, with a compute time of $O(n)$ and a period of $O(1)$.

Proof.

Let $C = C_1$ be the resulting VLSI-circuit for f obtained by laying out the parallel mintree as suggested. Let the i^{th} intermediate circuit C_i o the recursive construction (starting with i equal to $n-1$) fit in a rec tangle of size $b_i \times l_i$. From figure 7 we conclude that $b_{i-2} = 2b_i + (n-i + 9$ and $l_{i-2} = 2l_i + (n-i) + 8$. (The additive constants are likely to be a bit larger, because the detailed circuits for the $\square$-cells require mo than unit area.) As in the proof of proposition 3.2 it easily follows tha $b_1 = O(2^{n/2} - n)$ and $l_1 = O(2^{n/2} - n)$ and (hence) that C occupies no more than $O(2^n)$ area.

The time bound of $O(n)$ for computations with C is obvious. The period of $O(1)$ follows because C can process an entire input tuple in parallel as a single wavefront going down, with the possibility of sequencing consecutive wavefronts at unit time distance! $\square$

4. Optimal area/time bounds for Boolean VLSI-circuits

For particular Boolean functions f of n variables the bounds of theorem 3.2 may not be best possible. (See e.g. Thompson [7] for many examples of essentially Boolean functions that require only $O(n)$ or $O(n^2)$ area, or $O(\log n)$ time.) The question is rather how tight the bounds are if the theorem is taken as a uniform statement for the entire class of Boolean functions of n variables. This leads to the usual type of "worst-case" optimality problems which we shall here fully answer. It is clear that the $O(1)$ bound on the period of circuits can not be improved. We proceed to show that none of the other bounds in theorem 3.2 can be essentially improved either.

Theorem 4.1 There are Boolean functions of n variables such that any circuit to compute them requires $\Omega(n)$ time on some inputs.

Proof.

The result follows by combining some known results from the theory of Boolean functions. Let $T = T(n)$ be a (uniform) bound on the computing

time of the "fastest" circuits for the Boolean functions of n variables. (T exists and is $O(n)$.) Consider a Boolean function f and a circuit C that computes f within time T on any input. Note that all gates in C have out-degree 1 except the 0-cells, whose task it is to duplicate ("split") signals to multiple destinations. Tracing all possible computation "paths" backward from the output gate, we now construct a binary tree of gates in the following manner. (The construction relates to a similar one in Savage [5], p. 27.) Level 0 consists of just the single output gate of C. This gate also serves as the root of the tree. After level i has been formed, level i+1 is construed in the following manner. Every gate of level i corresponds to a gate of C and thus has one (when it is a $\neg$-gate) or two (when it is a $\vee$-gate or a $\wedge$-gate) predecessors in C. The one or two predecessors are copied and made into the son(s) of the gate and, thus proceeding for all gates in level i, level i+1 of the tree is formed. If a predecessor happens to be a 0-cell, then we do not include it but trace back further to determine the actual gate whose signal it is distributing and make this gate to the actual predecessor in level i+1. (Note that in this way many gates of C may arise multifold in a same level as distinct nodes.) Input gates are "terminal" in this process, and labeled by the proper x_i. We continue until T levels are constructed (or the procedure stopped because it reached a level of only "terminal" gates). Rewriting the "tree" into a "linear expression in x_1 to x_n", it follows that the effect of C (i.e., the function f) can be described by a formula of size $O(2^T)$. By a version of Shannon's theorem (see Savage [5], thm 3.4.2) however, "most" Boolean functions of n variables have a minimum formula size of $\Omega(2^n/\log n)$. Hence $T = \Omega(n)$. □

Corollary 4.2 Let f be a Boolean function with formula size L. Then any (logical or VLSI-) circuit that computes f requires $\Omega(\log L)$ time on some inputs.

A natural question is whether the $O(2^n)$ bound on the area requirement for general Boolean functions cannot be improved. We shall prove that the bound is optimal in a broad sense. To begin with, a simple argument will do if we make the (very restrictive!) assumption that the area bound A must be such that all Boolean functions of n variables can be computed by a circuit that fits on a fixed chip of area A (i.e., in a region of this area that is equal for all functions). Note that the upper-bound construction of theorem 3.2 gives rise to circuits for all f that satisfy this assumption with $A = O(2^n)$.

Proposition 4.3 Suppose all Boolean functions of n variables can be computed by circuits that fit on a fixed chip of area A. Then necessarily $A = \Omega(2^n)$

Proof

Recall that a chip was defined to be a (simple and connected) region of the unit grid. Although unimportant for this argument, we assume that the chip is at least one cell wide everywhere. Assume there is a fixed chip (region) R of area A such that for all Boolean functions f of n variables there is a circuit to compute it that can be embedded in R. According to Thompson's model each cell of R can be occupied by at most one of finitely many different constructs, say c altogether. (The constructs are a bounded variety of simple processing elements or gates, and a likewise bounded variety of wire patterns.) Thus at most c^A different circuits can be embedded in R. On the other hand, the number of distinct Boolean functions of n variables is 2^{2^n}. It follows that $c^A \geq 2^{2^n}$, or $A = \Omega(2^n)$. □

The assumption that there be a "general chip" is way too strong to measure the area requirement of Boolean functions. We shall thus allow that circuits are fitted on their own, optimal area chips. To be realistic we require that the chips under consideration are convex.

(This is a very common constraint, see e.g. Brent and Kung [1].) By the following two lemmas we shall be able to "mold" every circuit that fits on a convex chip of area A into a (permissible) form that fits in a standard (iso-oriented) rectangle on the grid of area $O(A)$.

Lemma 4.4 Every convex chip of area A can be enclosed by a (tilted) rectangular box of area $2A$.

Proof.

(The easy argument can be derived from e.g. Yaglom and Yaglom [8] solution to problem 120 a.) Let R be a bounded convex region of area A (see figure 8). Let AB be the longest chord. The tangents to R at A and B are perpendicular to AB. Now "enclose" R by drawing the (two) lines of support that are parallel to AB. Let the lines touch on R at C and D (see figure 8). In figure 8 is indicated that R is now enclosed by $\square EFGH$, formed by the four intersecting tangents. By convexity $\square ADBC \subseteq R$ and (hence) $\square ADBC$ has area $\leq A$. On the other hand, by geometric reasoning it follows immediately that $\square ABHE = 2 \cdot \triangle ABC$ and (symmetrically) that $\square ABGF = 2 \cdot \triangle ABD$ and (hence) that $\square EFGH = 2 \cdot \square ADBC \leq 2A$. $\square$

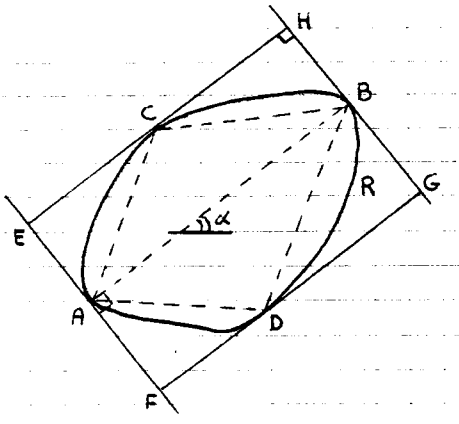


figure 8

Lemma 4.5 Let circuit C be embedded on the unit grid within a rectangle of area A that is tilted by angle α ($\alpha \leq \pi/4$). Then C can be embedded on the unit grid within an iso-oriented rectangle of area bounded by $(1 + 2\tan\alpha - \tan^2\alpha)A \leq 2A$.

Proof.

Let C be embedded on the grid such that it is enclosed by a rectangle of size $l \times w$ ($l \geq w$ and $l \cdot w = A$) whose longest side makes an

angle α with the positive x-axis of the grid (see e.g. figure 8). It is no restriction to assume that $0 \le \alpha \le \pi/4$, for otherwise the argument below would hold for a properly reflected or rotated version of the problem. We also assume that no cell cut by the boundary of the rectangle is used by C.

Divide the rectangle (and correspondingly, the circuit) into parts A_1, A_2 and B as shown in figure 9. Parts A_1 and A_2 will be left as they are, but are "padded" to iso-oriented rectangular boxes of size $w/\cos\alpha \times w\sin\alpha$ each (see figure 9). Part B consists of $l\cdot\cos\alpha - w\cdot\sin\alpha$ columns of $w/\cos\alpha$ cells each. Because $\alpha \le \pi/4$, each column can differ by at most one cell at either end from the immediately preceding column (see figure 10). It means that we can bring B down to an iso-oriented form by shifting each of its columns downwards by at most one cell with respect to the immediately preceding column, resulting in a circuit drawing as shown in figure 11. Clearly one cannot just shift a column with respect to its preceding column, because one must be able to bend (i.e., route) the wires that cross from one column into the next accordingly. Figure 12 shows that this can be done within the constraints of Thompson's model, provided that we include a spacing of one

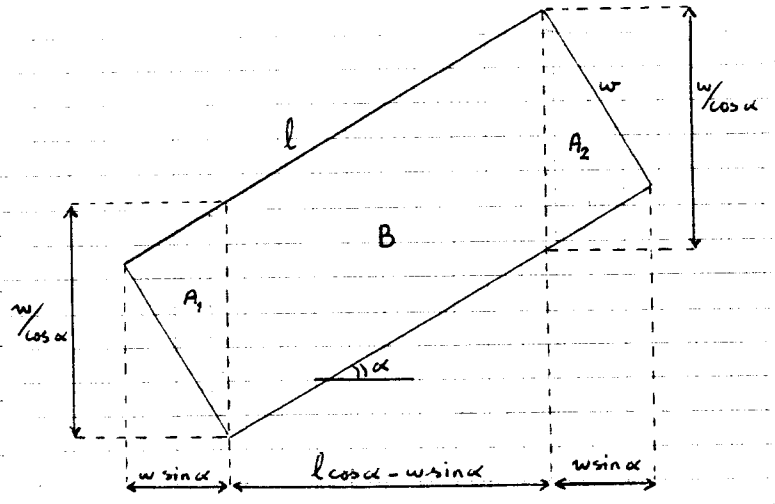


figure 9

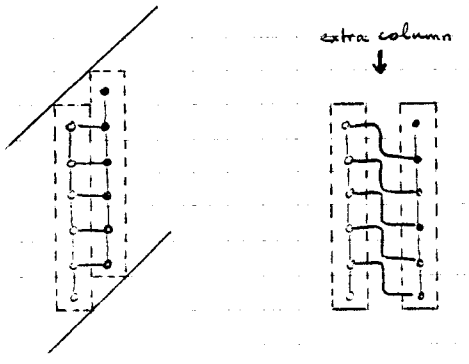


figure 10

extra column in between every pair of columns. This explains that in

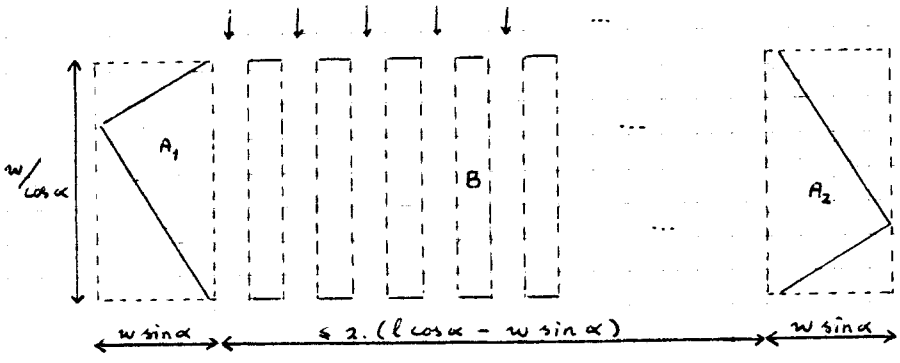


figure 11

figure 11 the length of the B-part is at most doubled. Thus circuit C appears embedded in an iso-oriented rectangle of width $w/\cos\alpha$ and length about $2.l.\cos\alpha$, and hence area $2.l.w = 2A$. By being a little more accurate, one can see that a down-shift over one cell (hence, the need for an extra column) is required only once in every $1/\tan\alpha$ columns. It follows that the length of B can remain at $(1+\tan\alpha)$ times the original. Hence C fits in an iso-oriented rectangle of size $(1+\tan\alpha).l\cos\alpha + (1-\tan\alpha).w\sin\alpha$ by $w/\cos\alpha$, which has an area bounded by $(1+2\tan\alpha-\tan^2\alpha)A$. □

We conclude that the convexity assumption in VLSI-theory actually degenerates and does not go much beyond assuming that chips are rectangular!

Corollary 4.6 Every circuit that fits on a convex chip of area A can be embedded in an ordinary iso-oriented rectangle of area $\leq 4A$.

Proof.

By applying lemma 4.5 to the embedding obtained from lemma 4.4. □

(The "aspect ratio" in VLSI-design is defined as the ratio between the shortest and the longest dimension of the chip, $\sigma = w/l$. Note that the aspect ratio of the iso-oriented embedding obtained in lemma 4.5 is ≈

$\frac{w}{\cos\alpha} / (1+\tan\alpha)\, l\cos\alpha + (1-\tan\alpha)\, w\sin\alpha \leq \frac{w}{\cos\alpha} / (1+\tan\alpha)\, l\cos\alpha = \frac{\sigma}{\cos^2\alpha + \sin\alpha\cos\alpha}$

which is at worst $\approx \sigma \cdot \frac{1}{2}\cos^2\alpha$ and thus bounded at a value between σ and $\frac{1}{2}\sigma$.)

The main result proves the desired lowerbound on the area-requirement for general Boolean functions.

Theorem 4.7 There are Boolean functions of n variables such that every convex chip for them requires $\Omega(2^n)$ area.

Proof.

Suppose that all Boolean functions of n variables can be computed by convex chips of area $\leq A$. By corollary 4.6 it follows that we may as well assume that all Boolean functions of n variables can be computed by circuits that fit in an iso-oriented rectangle of area $\leq 4A$. Now note that there are at most $\sqrt{4A} = 2\sqrt{A}$ different (i.e., non-isomorphi) iso-oriented rectangles of area $\leq 4A$ to consider for circuit embeddings, and that each of these rectangles can have at most c^{4A} different circuits. (The constant c is as in the proof of proposition 4.3.) It follows that necessarily $2\sqrt{A} \cdot c^{4A} \geq 2^{2^n}$, and (hence) that $A = \Omega(2^n)$. □

5. References

[1] Brent, R.P. and H.T. Kung, The area-time complexity of binary multiplication, J. ACM 28 (1981) 521-534.

[2] Lupanov, O.B., A method of circuit synthesis, Izv. V.U.Z. Radiofiz. 1 (1958) 120-140.

[3] Mead, C.A. and L.A. Conway, Introduction to VLSI systems, Addison-Wesley, Reading, Mass., 1980.

[4] Mead, C.A. and M. Rem, Cost and performance of VLSI computing structures, IEEE J. Solid State Circuits SC-14 (1979) 455-462.

[5] Savage, J. E., The complexity of computing, John Wiley & Sons, New York, NY., 1976.

[6] Shannon, C.E., The synthesis of two-terminal switching circuits, Bell Syst. Techn. J. 28 (1949) 59-98.

[7] Thompson, C.D., A complexity theory for VLSI, (Ph. D. Thesis), Techn. Rep. CMU-CS-80-140, Dept of Computer Science, Carnegie-Mellon University, Pittsburgh, PA., 1980.

[8] Yaglom, A.M. and I.M. Yaglom, Challenging mathematical problems with elementary solutions, Holden-Day, San Francisco, CA., 1967.