THE COMMUNICATION COMPLEXITY OF VLSI CIRCUITS

Thomas Lengauer

A 82/14

This material was presented as a Lecture within the AMS Short Course
on Computer Communications,
Denver, Colorado, Jan.3-4, 1983

Fachbereich 10
Universität des Saarlandes
D-6600 Saarbrücken
West Germany

THE COMMUNICATION COMPLEXITY OF VLSI CIRCUITS

Thomas Lengauer

Abstract: Very Large Scale Integration (VLSI) is a quickly emerging discipline in Computer Science that also raises many theoretical questions. The concept of a VLSI computation is very much different from classical concepts of a (sequential) computation. A VLSI computation is performed by many switching elements (s.e.'s) that are laid out on the planar chip surface and connected among each other with wires. These s.e.'s can perform computations in parallel. The computation of any s.e. depends on data received over wires from other s.e.'s. Several measures of complexity are of interest in this context: The chip area A, i.e., the area of wires and s.e.'s, the computing time T and the switching energy E. Clearly there exist tradeoffs between A and T. In this paper we survey lower bounds on the combined complexity measure AT^2. The lower bounds are proved by accounting for the amount of work necessary just to communicate intermediate results between s.e.'s. In many cases the lower bounds we get are (asymptotically) tight, giving evidence for the fact that communication cost dominates the complexity of many VLSI computations.

TABLE OF CONTENTS:

1. INTRODUCTION

In recent years the area of Very Large Scale Integration (VLSI) which originally was reserved to electrical engineering gained increasing interest among computer scientists. The book of Mead and Conway ([MC80]) triggered a lot of activity in VLSI design at the computer science departments of the universities and also focussed the attention of theoretical computer scientists and mathematicians on the area.

Several authors proposed theoretical models for VLSI computations ([T80,BK81, CM81,LS81,LM81]). These models are designed to capture the essential characteristics of the VLSI technology such as for instance the planarity of the chip. On the other hand they abstract from extraneous technological detail. Their purpose is to facilitate general statements on VLSI computations, specifically, lower bounds on their complexity.

The main complexity measures studied are the chip area A and the computing time T. For A we measure the area on the chip surface that is taken up by switches and wires, the so-called *active* area ([T80,LM81]), resp., the area of the enclosing rectangle, the so-called *total* area ([BK81,CM81,LS81]). The first area measure is directly related to the chip yield during fabrication. The second area measure accounts for the total size of the chip. For our purposes both area measures are equivalent ([LM81]). The question how to measure time is more difficult to answer. Most authors only study synchronous circuits. Here a global clock counts the synchronous steps of the computation performed by the switching elements. [T80,BK81,LS81,LM81] interpret T as the number of clock cycles spent on the computation. Note that this is not a "physical" measure of time, since the length of a clock cycle may itself vary with the size of the chip. However, as long as the delay of signal propagation along wires is not significantly longer than the delay of the switching elements the number of clock cycles gives a good representation of the time spent, i.e., is asymptotically accurate. As the number of transistors on a chip increases this ceases to be the case, however. [CM81] introduce a physical time measure T by measuring time in seconds under the assumption that signals are propagated along wires at a constant speed. They get dramatically different lower bounds on circuit complexity. Not only are their lower bounds larger, as we expect, but the optimal chips according to their complexity measures differ significantly from the optimal chips if T is measured in clock cycles. This is, because [CM81] pay a penalty for long wires across which communication is expensive. The time measure of [CM81] may become technologically significant eventually, as the speed of light becomes the limiting factor in signal propagation on VLSI chips. In the meantime, however, other physical time measures may be more appropriate, such as the one introduced in [MC80] that is based on the capacitive properties of VLSI structures.

In this paper we will only count clock cycles in synchronous VLSI circuits. Our main observation, namely that the chip complexity is dominated by the cost of communication, holds for the other time measures as well. However, the time measure we consider has the most developed body of theory and thus is best suited to make our point.

Since synchronous circuits can be pipelined another complexity measure gains significance, namely the period P. A pipelined circuit receives input to a new problem to be solved before it has finished solving the old problem. P is the number of clock cycles between the first input to two consecutive problems (see Figure 1).

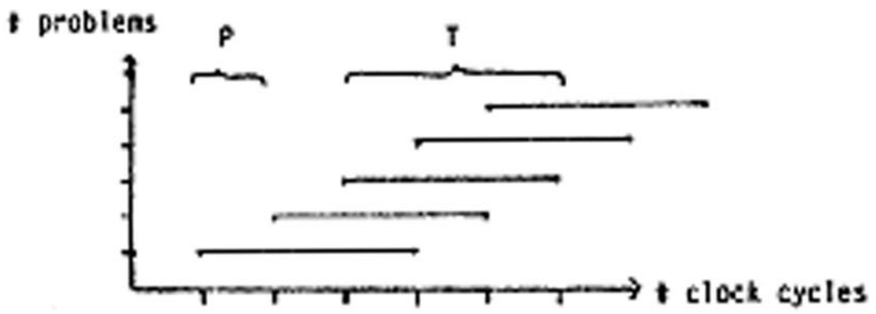

Figure 1

As a circuit solves many problems, i.e., a number n of problems such that nP>>T, P assume the role of T since the time T+(n-1)P to solve all n problems does not differ significantly from nP. We will not discuss P explicitly in this paper but just mention that the lower bounds we prove hold even if we substitute P for T and make the appropriate modifications in the relevant definitions.

Finally the switching energy E has been considered as a complexity measure for VLSI computations. The switching energy E is measured based on the following assumption:

Every unit of active chip area consumes one unit of switching energy each time it changes its state from 0 to 1 or vice versa.

This complexity measure is closely related to the energy dissipated when charging a wire. In technologies without high d.c. currents the switching energy dominates the total energy dissipation on the chip.

Lower bounds on A can be obtained in many cases ([VB0,LM81]). However, the respective arguments are based on measuring the storage capacity necessary for the computation and do not involve communication complexity. We therefore do not discuss these results here. Looking for lower bounds on T reduces to the same problem in Boolean circuit complexity, which is known to be very hard. A lower bound on E will be proved in Section 7 of this paper. It stems from [LM81].

The above complexity measures can be combined to form new interesting complexity measures. Clearly there is a tradeoff between A and T, since more available chip area allows greater on-chip concurrency. Thus the quantity AT is an interesting complexity measure. However, with respect to AT many circuits are optimal that we intuitively would not consider good VLSI chips. (An example is the bitserial adder.) This is, because concurrency in the computation is not given sufficient priority. We can make the complexity more sensitive to concurrency by giving the T component a heavier weight. This motivates the investigation of AT^2 as a complexity measure. [T80] has presented a proof method for deriving lower bounds on the AT^2 complexity of VLSI chips, and later authors have elaborated on it. This method will be the main subject of this paper. The method measures the amount of information that has to be exchanged between different components of the chip and relates it to the chip area and computing time. It only accounts for the cost of communication, not for the cost of computation or storage. The fact that many of the derived lower bounds are (asymptotically) tight shows that the complexity of VLSI computations is in many cases dominated by the cost for communicating intermediate results across the chip.

In Section 2 we present the model for synchronous VLSI circuits we are using. In Section 3 we give an intuitive outline of the method for proving AT^2 lower bounds. In Sections 4 to 6 we provide the formal tools for proving the lower bounds. Section 4 contains the geometric tool in the form of a separator theorem. Sections 5 and 6 provide the information theoretic tools for Boolean predicates resp. for Boolean functions with many outputs. Finally, in Section 7 we use the ideas developed in the earlier chapters to prove a lower bound on the switching energy consumed by VLSI chips.

2. THE VLSI MODEL

Our VLSI model is based on Boolean circuits. This choice is adequate also for modelling more general "multi-directional" VLSI structures, e.g., buses ([LM81]).

DEFINITION 1: A *chip* $\chi=(\Gamma,\Lambda,\Delta)$ consists of three structures:

a) The *circuit* Γ: A synchronous Boolean circuit with feedback and unbounded fan-in. Formally, this is a directed bipartite graph $\Gamma=(V,E)$ where V is partitioned into a set S of *switches* and a set W of *wires*. Here $S=P\cup G$ where P is a set of *ports* labelled in or out and G is a set of *gates* labelled and, or, nand, or nor. For $s\in S$ and $w\in W$, if $(s,w)\in E$ then w is called an *output* of s, if $(w,s)\in E$ then w is called an *input* of s. All gates have out-degree 1, all wires have in-degree 1. Each input port has one input and one output. Each output port has two inputs and no output. The "additional" input signal for the ports is an enable signal computed on the chip that activates the port.

b) The *layout* Λ: The layout maps every vertex in the circuit into a compact connected region in the plane. Furthermore, each point in the region lies inside some square of side length $\lambda>0$ that is completely contained in the region.

(This provision models the finite resolution of the fabrication process for VLSI chips.) Each point in the plane belongs to the interior of at most $\nu \geq 2$ regions. (The parameter ν is another fabrication process specific constant representing the number of functional *layers* on the chip. Since $\nu \geq 2$ also non-planar circuits Γ can be laid out.) Two regions touch exactly if the vertices they represent are neighbours in Γ. (We say that regions R_1 and R_2 touch if $R_1 \cap R_2 \neq \emptyset$ but $R_1^o \cap R_2^o = \emptyset$, where R_1^o, R_2^o are the interiors of R_1 resp. R_2.)

c) The *manual* Δ: The manual is a set of directions for the communication between the chip and its environment. It contains for every input port a sequence of numbers that identify the input bits that enter through this port. The sequence also determines the order in which the input bits enter. When its enable signal is raised the port requests the next input bit in the sequence to enter. Analogously, for each output port the manual contains a sequence of numbers identifying the output bits produced at that port and their order. When its enable signal is raised the port produces the next output bit in the sequence.

After defining all components of the chip we can define the operation of the circuit. To this end we associate with each port a word $w \in B^*$, $B = \{0,1\}$, that we call its *history*. Furthermore, we label each wire with an initial Boolean value from $B \cup \{X\}$. (X stands for the undefined Boolean value, $0 \wedge X = 0$, $1 \vee X = 1$, $0 \vee X = 1 \wedge X = X$.) Such a labelling we call a *state* of the circuit. The initial state is most often the completely undefined state. In the i-th cycle the circuit does the following. Each gate "reads" the values on all its input wires and computes the Boolean operation given by its label. The resulting value is put on its output wire. Each input port puts an X on its output wire if its enable signal is 0, otherwise it puts the ext bit from its history on its output wire. Each output port checks if its enable signal is 1, and if so it puts its other input at the end of its history. Thus input ports consume their histories and output ports produce them. All actions of the gates happen in parallel. Thus a new state is reached on which the (i+1)-st cycle of the computation is started.

Finally we discuss, what A, T and E mean in this context. The area A is the area of the set M of all points in the plane that belong to at least one region in the layout. (Thus we measure active area.) The time T is the number of steps taken by the circuit to produce the desired outputs in its output histories. The switching energy E is based on the area of the regions in the layout.

There has been some confusion about different kinds of manuals in the past. Therefore we discuss the issue here at greater length.

The manual used in Definition 1 is called *strongly where-oblivious* in [LM81]. Many authors use manuals of a different kind to prove the same lower bounds.

[LS81] call a manual where-oblivious (when-oblivious) if the port (cycle) at which an input bit is requested resp. an output bit is produced is independent of the input. In this paper we will also consider manuals which are both where- and when-oblivious. Such manuals simply list for each I/O bit the time and port at which it crosses the chip boundary. These data are independent of the input. (Note that strongly where-oblivious manuals are in general not when-oblivious.) If we restrict ourselves to manuals which are where- and when-oblivious then we do not have to require the I/O ports to be activated by enable signals which are produced by the circuit. We can instead assume that all ports are active all the time, and that they sometimes consume resp. produce bits which are insignificant for the problem to be solved. This assumption takes some computational burden off the chip, because it can "blindly" consume and produce bits without having to compute the knowledge on what data it is currently working. Even so, we can still prove the same lower bounds as for strongly where-oblivious chips. This is, because if the manual is where- and when-oblivious then even on "blind" chips solutions to different problems are always represented by different I/O histories. If the manual is strongly where-oblivious this is not the case, however, and the chip has to "know" what bits cross its boundary when and where. If this knowledge were not computed by the chip but could be found inside the manual, every problem could be solved by a trivial chip with a manual that contains all the computation. (The chip takes no input, runs for two cycles and produces a 0 in the first and a 1 in the second cycle at its unique output port. One of these two values is the unique output and the manual says which one it is, depending on the input.)

We distinguish two kinds of complexity analysis: Worst case analysis and average case analysis. In worst case analysis we ask for the greatest value of a complexity measure (A,T,E) for all possible input sequences of the same length. In average case analysis we average the complexity measure over all input sequences of the same length under the assumption that they are equally likely. For the area A worst case analysis and average case analysis coincide. For the switching energy E they may differ. For T they coincide if the manual is where- and when-oblivious, and they may differ, if the manual is strongly where-oblivious.

3. THE LOWER BOUND ARGUMENT

In this section we give an intuitive outline of the argument for proving lower bounds on the AT^2 complexity of VLSI chips.

Let us assume that we have a rectangular chip computing some Boolean predicate

$p:B^n \to B$. The chip has side lengths a and b, $a \le b$. Let us furthermore assume that the chip has n input ports and one output port, and that a unique I/O bit is assigned to each port. Clearly we can cut the chip into two halves L and R by a line C parallel to the sides of length a of the chip, such that about half the input ports lie on the side L resp. R. We first observe that the length of the cut is $\lg(C)=a \le \sqrt{A}$. The chip is where-oblivious and therefore we can also assign to each input bit a side among L,R through which it enters the chip, independently of the input. This partitions the input bits into two classes X and Y corresponding to the sides L and R. Let us assume that the output port lies on side R. If we now can show that in order to compute the output bit we have to know many, e.g. ω of the input bits in X we can argue as follows: Since the chip is planar the input bits in X can only be communicated to the side R across the cut C. By our layout model C can only be crossed by a number of wires that is proportional to its length. Thus during each cycle only $O(\lg(C))$ bits can be communicated across C. Since ω bits have to cross C this takes time $T=\Omega(\omega/\lg(C))$. By our bound on $\lg(C)$ we get $T=\Omega(\omega/\sqrt{A})$, or $AT^2=\Omega(\omega^2)$.

Thus the argument is divided into two parts. The first part is the proof of a geometric separator theorem that says that any well behaved (e.g. rectangular) region in the plane with area A representing the chip layout can be cut by a nicely behaved (e.g. straight) line into two "balanced" halves (e.g. with respect to the number of input ports in them). Furthermore the length of the cut line is small, i.e., $O(\sqrt{A})$. We call such a theorem a *separator theorem*. The notion of a separator is known from graph theory ([LT79]), and in non-trivial cases the methods from graph theory can be carried over to prove geometric separator theorems. We will prove a general separator theorem in Section 4 that allows the application of the lower bound argument for the case that we are interested in active area. The obvious separator theorem for rectangles contained in the above outline applies to the notion of total area.

The second part of the lower bound argument lies in the definition of a certain notion of "communication complexity" for the Boolean function p to be computed. Several different notions are possible here. We will discuss them in detail in Section 5 and 6. Intuitively, the communication complexity of a function p is the amount of information that has to be exchanged between two cooperating processors that each have a partial knowledge of the input and that want to aid each other in computing p. The communicating processors are in this case the two halves of the chip as divided by the cut C. The communication complexity of p then takes the part of the quantity ω in the above outline.

These ideas will be made more precise in the following sections.

4. A GEOMETRIC SEPARATOR THEOREM

This section gives a separator theorem for compact plane regions. The theorem is a special case of a more general separator theorem presented in [LN81]. The proof is patterned after the proof of a similar separator theorem for planar graphs given in [LT79].

THEOREM 1: Let M be a compact plane region. Let $p_1,\dots,p_r$ be r different points inside the interior of M of which n are specially "marked". Then M can be cut by a set C for three straight cuts into two compact sets M_ℓ and M_r such that:

1. $M_\ell \cup M_r = M \qquad M_\ell \cap M_r = C$
2. $\lg(C) \le 2\sqrt{A}$ where A is the area of M.
3. M_ℓ, M_r both have at most $2n/3$ marked points of $p_1,\dots,p_r$ in their interior and none of the points $p_1,\dots,p_r$ on their border.

PROOF: First we note that because the set $P=\{p_1,\dots,p_r\}$ is finite M can be put into a Cartesian coordinate system such that no line parallel to the x- or y-axis contains more than one point in P. We devise the following procedure for cutting M.

W.l.o.g. assume that $\lceil n/2\rceil$ of the marked points lie above the x-axis, $\lfloor n/2\rfloor$ of the marked points lie below the x-axis, and no point in P lies on the x-axis. Let $M_+ = \{(x,y)\in M; y\ge 0\}$ and $N_- = \{(x,y)\in M; y\le 0\}$. Let A_+ be the area of M_+ and A_- be the area of M_- $(A_+ + A_- = A)$. The method for cutting M has two steps:

<u>Step</u> 1: For any real ℓ, let $C_\ell = \{(x,y)\in M; y=\ell\}$. We choose two cuts of M parallel to the x-axis at distances $y=\ell_t>0$ and $y=\ell_b<0$. C_{ℓ_t} and C_{ℓ_b} cut M into three parts M_t, M_m, and M_b as suggested by Figure 2.

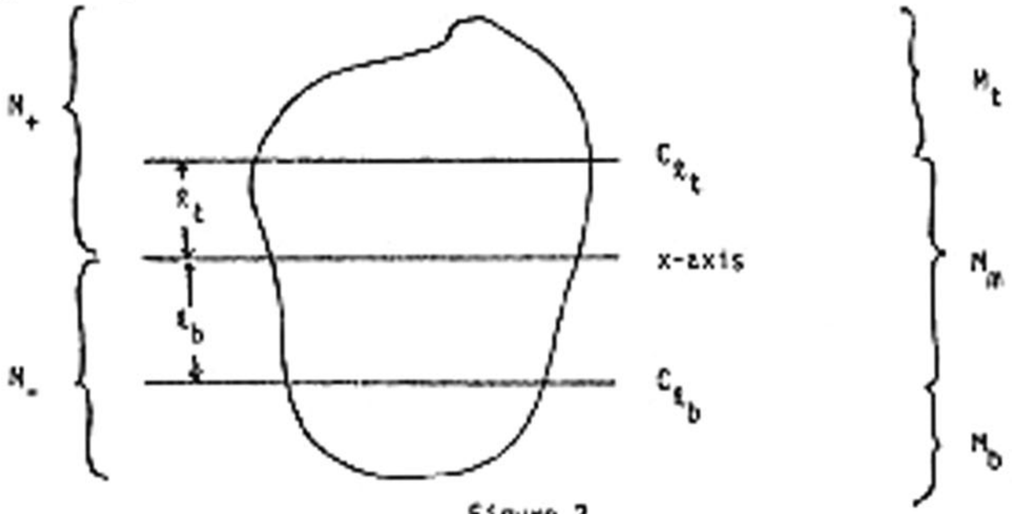

Figure 2

We choose ℓ_t such that $\lg(C_{\ell_t})+\ell_t \le \sqrt{2A_+}$ and $\ell_t \le \sqrt{2A_+}$. Analogously we choose ℓ_b such that $\lg(C_{\ell_b})-\ell_b \le \sqrt{2A_-}$ and $\ell_b \ge -\sqrt{2A_-}$. Such ℓ_t and ℓ_b can be found. For assume that such an ℓ_t does not exist. Then

$$A_+ = \int_0^\infty \lg(C_\ell)\, d\ell \ge \int_0^{\sqrt{2A_+}} \lg(C_\ell)\, d\ell > \int_0^{\sqrt{2A_+}} (\sqrt{2A_+}-\ell)\, d\ell = A_+$$

which is a contradiction. In fact we can assume that no point of P lies on C_{ℓ_t} since the set $\{\ell; 0<\ell\le\sqrt{2A_+}$ and $\lg(C_\ell)+\ell \le \sqrt{2A_+}\}$ must have a positive measure. An analogous argument applies to ℓ_b.

Now, if M_m contains no more than $2n/3$ marked points of P then we are done. Because M_t and M_b both have at most $n/2$ marked points of P we only have to choose M_ℓ to be the one of the three sets with the most marked points of P in it.

If, however, M_m has more than $2n/3$ marked points in it then we have to continue cutting.

<u>Step</u> 2: We bisect M_m with a line $x=\ell_v$ such that half the marked points in M_m are on either side on the line. More precisely, the cut consists of the segment $\{(x,y); x=\ell_v$ and $\ell_t\ge y\ge\ell_b\}$. Now clearly the half of M_m containing the most marked points of P will serve as M_ℓ.

The total length of the cut C is bounded by

$$\lg(C) \le \lg(C_{\ell_t}) + \lg(C_{\ell_b}) + (\ell_t-\ell_b) \le \sqrt{2A_+} + \sqrt{2A_-} \le \sqrt{2A_+} + \sqrt{2(A-A_+)} \le 2\sqrt{A}$$

since the function $z(x)=\sqrt{x}+\sqrt{1-x}$ has a maximum of $\sqrt{2}$ at $x=1/2$.□

5. THE COMMUNICATION COMPLEXITY OF BOOLEAN PREDICATES

As motivated in Section 3 we start from the following definition.

DEFINITION 2 ([Y79]): Let $p:X\times Y\to B$ be a Boolean predicate. We call X the set of left inputs and Y the set of right inputs. Let L and R be two computing agents one of which knows $x\in X$ and the other of which knows $y\in Y$. L and R want to compute $p(x,y)$ by exchanging information between each other. We are primarily interested in deterministic computations:

A *deterministic algorithm* is given by two reponse functions $\rho_L:X\times B^*\to B$ and $\rho_R:Y\times B^*\to B$ and the partial output function $a:B^*\to B$. For computing $p(x,y)$ L starts sending the bit $w_1=\rho_L(x,\varepsilon)$ to R. R responds with $w_2=\rho_R(y,w_1)$; L returns $w_3=\rho_L(x,w_2)$ and so on, until $w_1,\ldots,w_{k(x,y)}\in dom(a)$. At this point the computation stops with the result $a(w_1,\dots,w_{k(x,y)})=p(x,y)$. $k(x,y)$ is the length of the computation.

The deterministic two-way communication complexity of p is given by

$$C_{det}(f,L\leftrightarrow R) = \min_{A} \max_{\substack{x\in X\\ y\in Y}} k_A(x,y)$$

where A is any deterministic algorithm computing p and $k_A(x,y)$ is the length of the computation on input (x,y), if algorithm A is used.

For theoretical purposes it is often interesting to introduce non-determinism into the model. In an non-deterministic algorithm we are allowed guesses as to the outcome of certain computations. These guesses then only have to be verified. W.l.o.g. we can assume for non-deterministic algorithms that information is only sent from L to R. (L could guess all of the information sent from R and R just has to verify these guesses.)

DEFINITION 3: A non-deterministic (one-way) algorithm is given by a set $\rho_L(x)\in B$ for every x and by a response function $b:Y\times B^*\to B$. On input (x,y), L sends some word $w\in\rho_L(x)$ to R and R outputs b(y,w). In order for R to know when L has finished sending its information we assume $\rho_L(x)$ to be prefix-free.) The non-deterministic algorithm computes p if $p(x,y)=1$ exactly if there is a $w\in\rho_L(x)$ such that $b(y,w)=1$. The non-deterministic communication complexity of p is

$$C_{ndet}(f,L\to R) = \min_{A} \max_{\substack{x\in X\\ y\in Y\\ f(x,y)=1}} \min(|w|; w\in\rho_{L,A}(x) \text{ and } b_A(y,w)=1)$$

A non-deterministic algorithm is *unambiguous* if the last minimum in the above equation is taken over singletons (and can thus be omitted). We define

$$C_{unamb}(f,L\to R) = \min_{\substack{A\\ unamb}} \max_{\substack{x\in X\\ y\in Y\\ f(x,y)=1}} |w(x,y)|$$

where $w(x,y)$ is the unique w such that $w\in\rho_{L,A}(x)$ and $b(y,w)=1$.

A third class of algorithms which are of interest are "Las Vegas" algorithms. Las Vegas algorithms use internal randomization (coin flipping) to determine the result. However, the randomization does not affect the outcome of the computation but just its complexity.Las Vegas algorithms are not only of theoretical interest but of practical significance since the realization of true coin flipping devices in VLSI technologies seems in the realm of the possible.

DEFINITION 4: A Las Vegas algorithm is given by two response functions $\rho_L: X\times B^*\times B^t \to B$ and $\rho_R: Y\times B^*\times B^t \to B$ and the partial output function $a: B^* \to B$. We assume that both L and R first toss t coins to determine the third arguments t_L, t_R in the response functions, and then start a deterministic computation as in Definition 2. The computation ends when $(w_1,\dots,w_{k(x,y,t_L,t_R)}) \in dom(a)$. Its result is $a(w_1,\dots,w_k)$. The Las Vegas communication complexity of p is

$$C_{LV}(p,L\leftrightarrow R) = \min_{\substack{A\\ LV\text{-}alg}} \sum_{t_L,t_R\in B^t} k(x,y,t_L,t_R)/2^{2t}$$

We can also define a deterministic average case communication complexity of p by just averaging over all inputs instead of maximizing over them:

$$C_{ave}(p,L\leftrightarrow R) = \min_{\substack{A\\ det}} \sum_{x\in X, y\in Y} k_A(x,y)/(|X||Y|)$$

The relation of the quantities defined above to the AT^2 complexity of chips will be discussed later in this section. However, they are also of use for concerns outside the VLSI area. For instance, they apply to problems in one-tape Turing machine complexity. Indeed, they can be used as a basis for a theory of communication complexity, and as such we will consider them now. This kind of complexity theory has been studied in [Y79,Y81,MVB2].

We now study methods for finding lower bounds on the communication complexity of Boolean predicates. The methods consider the Boolean predicate p as an $|X|\times|Y|$ matrix P of zeroes and ones. The input x selects the row, y selects the column of the matrix. Thus $P_{xy}=p(x,y)$.

METHOD 1 (Crossing Sequences):

This method is an adaptation of the crossing sequence technique for showing lower time bounds for one-tape Turing machine complexity. It was first presented in our context in [S81]. It is based on the following theorem.

THEOREM 2: Consider the matrix P defining the Boolean predicate p. By permuting rows and columns independently put P into the form $\begin{pmatrix} I & B \\ A & C \end{pmatrix}$ where I is an $m\times m$ identity matrix and m is as large as possible. Then $C_{det}(p,L\leftrightarrow R)\geq \log m$.

PROOF: The proof is an application of the pigeon hole principle. Assume that $C_{det}(p,L\leftrightarrow R)<\log m$. Then there is a deterministic algorithm computing p that always exchanges less than log m bits. Thus there are two inputs (x_1,y_1) and (x_2,y_2) corresponding to diagonal elements in the identity matrix I of P for which the same information is exchanged between L and R. By Definition 2 the same information is also exchanged on inputs (x_1,y_2) resp. (x_2,y_1). Thus the algorithm yields the same result for all four input values. Since (x_1,y_2) and (x_2,y_1) are off the diagonal in J the algorithm must therefore yield incorrect results on some of the four input values.□

COROLLARY 1: The above theorem is still true if we substitute C_{ndet} or C_{unamb} for C_{det}.

PROOF: Analogous to the proof of Theorem 2.□

Corollary 1 is the manifestation for communication complexity of the well known fact that the crossing sequence technique is not able to differentiate between determinism and non-determinism.

Note that Method 1 is not strong enough to show average case lower bounds. This is, because we bound the communication complexity of a Boolean predicate from below by showing that it entails an identity predicate $id(x,y):=(x=y)$ for strings of some length. But identity predicates are easy to compute on the average. In fact, most often the inputs (x,y) of a identity predicate will not agree and this can be found out very fast. (Discrepancies will occur in the first bit half the time.) One easily shows that $C_{ave}(id,L\leftrightarrow R)\le 2$.

Although Corollary 1 may at first sight seem to show a strength of Method 1, it really exhibits a weakness. Non-deterministic lower bounds are in general weaker than deterministic lower bounds because non-determinism is a more powerful concept than determinism. Thus Method 1 will yield bad results for predicates that are difficult to compute deterministically but easy to compute non-deterministically. Sometimes one can get better results by complementing the predicate, i.e., by considering $\overline{p} = 1-p$ instead of p. $\overline{p}$ has deterministically the same communication complexity as p but may be harder non-deterministically. An example for such a predicate is again the identity predicate. By Corollary 1 we have for the identity predicate on strings of length n $C_{ndet}(id,L\leftrightarrow R)\ge n$. However, we can show that $C_{ndet}(\overline{id},L\rightarrow R)\le \log n + 1$. This is, because in order to show that $x\ne y$, L only has to guess one bit position in which x and y differ. It then sends the number of this position (log n bits) and the corresponding bit value of x (1 bit) to R. Thus $\overline{id}$ is easy to compute non-deterministically. However, we can still show a good deterministic lower bound on $C_{det}(\overline{id},L\leftrightarrow R)$ by applying Method 1 to its complement id, which is difficult to compute non-deterministically.

However, there are natural Boolean predicates p such that

$$C_{ndet}(p,L\rightarrow R),\ C_{ndet}(\overline{p},L\rightarrow R) \ll C_{det}(p,L\leftrightarrow R).$$

For such predicates Method 1 fails to prove good deterministic lower bounds. For the purpose of proving lower bounds in such cases a second method is presented.

METHOD 2 (Rank Lower Bound):

This method has been presented in [MM82].

DEFINITION 5: Let r be a ring and let $r^{(n,m)}$ be the set of $n\times m$ matrices over r. The rank of $A\in r^{(n,m)}$ over r is the minimum k such that A can be written as $A = C\cdot D$, where $C\in r^{(n,k)}$ and $D\in r^{(k,m)}$.

If r is a field the above definition coincides with the definition of matrix rank known from linear algebra. Method 2 is based on the following theorem.

THEOREM 3: Let p be a Boolean predicate, and let P be its associated matrix. Then $C_{det}(p,L\leftrightarrow R) \geq \log \text{rank}_N(P) \geq \log \text{rank}_r(P)$ where r is any field.

PROOF: The second inequality is known from algebra. For the proof of the first inequality we state the following lemma.

LEMMA 1: Let r be a ring, $A\in r^{(n,m)}$, $B\in r^{(n,k)}$, and $C\in r^{(k,m)}$. Then

$$\text{rank}_r((A\ B)) \leq \text{rank}_r(A) + \text{rank}_r(B)$$

$$\text{rank}_r\left(\begin{pmatrix}A\\C\end{pmatrix}\right) \leq \text{rank}_r(A) + \text{rank}_r(C)$$

PROOF: If $A=D_1\cdot E_1$ and $B=D_2\cdot E_2$ then

$$(A\ B) = (D_1\ D_2)\cdot\begin{pmatrix}E_1 & 0\\0 & E_2\end{pmatrix}.$$

The proof of the second inequality is analogous.□

Now consider any deterministic algorithm for computing p. Inductively on the length of $w\in B^*$ we define the matrix P_w as follows:

$|w|=0$: $P_\varepsilon := P$

$|w|>0$: If $|w|=2\ell$ then P_{w0} (P_{w1}) is obtained from P_w by selecting all rows x with $\rho_L(x,w)=0$ ($\rho_L(x,w)=1$). If $|w|=2\ell+1$ then P_{w0} (P_{w1}) is obtained from P_w by selecting all columns y with $\rho_R(y,w)=0$ ($\rho_R(y,w)=1$).

By Lemma 1 we have $\max\{\text{rank}_N(P_{w0}),\text{rank}_N(P_{w1})\}\geq \text{rank}_N(P_w)/2$. Moreover, if $w\in\text{dom}(a)$ then P_w must be monochromatic, i.e., all of the entries in P_w must have the same value $a(w)$, and thus $\text{rank}_N(P_w)\leq 1$ must hold. Thus there is a $w\in\text{dom}(a)$ such that $|w|\geq\log \text{rank}_N(P)$.□

In [MM82] it is shown that in fact $C_{unamb}(p,L\to R)=\lceil\log \text{rank}_N(P)\rceil$. On the other hand $C_{ndet}(p,L\to R)$ can be much smaller, as the following example shows.

DEFINITION 6: Let $n\in\mathbb{N}$ and $X=Y=\{0:2^n-1\}^n$. For $x=(x_1,\ldots,x_n)\in X$ and $y=(y_1,\ldots,y_n)\in Y$ let

$$p_1(x,y) = \begin{cases}1 & \text{if } x_i=y_i \text{ for some } i, 1\leq i\leq n\\ 0 & \text{otherwise}\end{cases}$$

The above definition is the result of modifying the identity predicate such that it and its complement are non-deterministically about equally hard. Instead of checking the identity of one pair of n-bit words we now check the identity of n pairs of n-bit words. If any pair is identical we output 1 else 0.

THEOREM 4: a) $C_{det}(p_1, L \leftrightarrow R) \geq n^2$

b) $C_{ndet}(p_1, L \rightarrow R) = O(n + \log n)$

c) $C_{ndet}(\overline{p_1}, L \rightarrow R) = O(n \log n)$

d) $C_{LV}(p, L \leftrightarrow R) = O(n(\log n)^2)$

e) $C_{ave}(p, L \leftrightarrow R) = O(n)$

PROOF: a) Since $P_1 \in B^{2^{(n^2)}}$ we only have to show that $\text{rank}_{GF(2)} P_1 \geq 2^{(n^2)}$, where GF(2) is the field of characteristic 2. Let $\oplus$ denote addition modulo 2. We transform the matrix $\overline{P_1}$ associated with $\overline{p_1}$ into the identity matrix of size $2^{(n^2)} \times 2^{(n^2)}$ by means of linear transformations.

LEMMA 2: Let $w_1, \ldots, w_n, y_1, \ldots, y_n \in [0:2^n - 1]$. Define

$$g(w_1, \ldots, w_n, y_1, \ldots, y_n) := \bigoplus_{\substack{x_1 \\ x_1 \ast w_1}} \cdots \bigoplus_{\substack{x_n \\ x_n \ast w_n}} \overline{p_1}(x_1, \ldots, x_n, y_1, \ldots, y_n)$$

Then $g(w_1, \ldots, w_n, y_1, \ldots, y_n) = id(w_1, \ldots, w_n; y_1, \ldots, y_n)$.

PROOF: Omitted.□

b) L guesses the number $i \in [1:n]$ such that $x_i = y_i$ and sends i and x_i to R. If $x_i = y_i$ then R outputs 1.

c) For each $i \in [1:n]$ L guesses a position $j_i \in [1:n]$ such that x_i and y_i differ in position j_i. L sends j_i and the corresponding bit of x_i to R. R verifies that the bits indeed differ.

d) See [MN82].

e) For $i = 1, \ldots, n$ L and R alternate sending bits of x_i resp. y_i until one bit is found in which x_i and y_i differ. Then L and R proceed with (x_{i+1}, y_{i+1}). If $x_i = y_i$ is verified then the computation stops immediately. This deterministic algorithm computes n independent identity predicates. We already noted that the average case communication complexity of the identity predicate is less than 2. Thus $C_{ave}(p_1, L \leftrightarrow R) \leq 2n$.□

Part e) of Theorem 4 shows that Method 2 is not strong enough to show average case lower bounds either.

With the above two methods for showing lower bounds on the communication complexity of Boolean predicates we are now able to devise strategies for making statements about the AT^2 complexity of chips computing such predicates.

DEFINITION 7: Let p be a Boolean predicate with n "marked" input bits. p is called *ω-separable* if for every partition of the input bits into sets X, Y such that both X and Y contain at most $2n/3$ of the marked input bits we have

$$C_{det}[p(x,y),L \leftrightarrow R] \geq \omega.$$

The following Theorem emerged over a series of papers ([T80,BK81,V80,S81, LS81,K82]).

THEOREM 5: Let p be an ω-separable Boolean predicate. Then $AT^2=\Omega(\omega^2)$ for every (where- and when-oblivious or strongly where-oblivious) chip computing p.

PROOF: Let $\chi=(\Gamma,\Lambda,\Delta)$ be a chip computing p. Let π be an input port and let R_π be its associated region in the layout Λ. Let the input bit x_i enter the chip through input port π. With each such x_i we associate a point p_i in the interior of R_π such that different points are associated with different input bits. For the purposes of the lower bound proof we will consider the bit x_i to enter the chip through point p_i. If x_i is a marked input bit we consider p_i a marked point.

We now apply Theorem 1. Thus we cut the layout area M into two halves M_ℓ and M_r, and induce a partition of the input bits as follows:

$$X = \{x_i;p_i \in M_\ell\} \qquad Y = \{x_i;p_i \in M_r\}.$$

Furthermore X,Y contain at most $2n/3$ marked inputs.

Since the cut C has a length of at most $2\sqrt{A}$, at most $h=O(\sqrt{A})$ regions of Λ associated with components of Γ can intersect C. This is, because by the VLSI model, only $O(1)$ regions can intersect any square of unit area in Λ.

Now consider the computation of p by χ. At each cycle we associate two values with C, a left and a right "crossing" value. The left (right) crossing value $v^\ell=(v_1^\ell,\ldots,v_h^\ell)$ $(v^r=(v_1^r,\ldots,v_h^r))$ contains a component v_i^ℓ (v_i^r) for each region R_i intersecting C. If R_i is a (<u>nand</u>,<u>nor</u>,<u>and</u>,<u>or</u>) gate then v_i^ℓ (v_i^r) is the (nand, nor,and,or) of all input values during the last cycle whose regions intersect M_ℓ (M_r). If R_i is a wire then v_i^ℓ (v_i^r) is the above value for its input gate. Ports act as <u>and</u>-gates in this context.

With these definitions the computation of p by χ can be regarded as a deterministic algorithm in the sense of Definition 2. The information exchanged between L and R are the crossing values. The left crossing values are sent from L to R, and the right crossing values are sent from R to L. After the last cycle L and R exchange two more bits which make the result of the computation known globally.

If the length of the VLSI computation is T the communication length of the associated deterministic algorithm is $O(Th)$. We get $Th=T\sqrt{A}=\Omega(\omega)$, or $AT^2=\Omega(\omega^2)$.□

Note that if we substitute C_{ave} for C_{det} in Definition 7 Theorem 5 yields an average case lower bound. However, as mentioned before we have no methods of proving good average case lower bounds on the communication complexity of Boolean predicates.

The identity predicate $id(x,y)$ that served as an example for the application of Method 1 is not $\Omega(n)$-separable because we can partition the input bits cleverly to assure that little information has to be exchanged between L and R. Specifically, if we choose $X=\{x_1,\ldots,x_{n/2},y_1,\ldots,y_{n/2}\}$ and Y accordingly L has to send just one bit to R, namely the bit $x_1=y_1 \wedge x_2=y_2 \wedge\ldots\wedge x_{n/2}=y_{n/2}$. We can modify the identity predicate to yield an $\Omega(n)$-separable Boolean predicate.

DEFINITION 8: Let $id_1(x,y,k)$ be a predicate of three inputs. x and y are n-bit strings $(n=2^m)$ and k is an m-bit string encoding a number $0\le k<n$. We define

$$id_1(x,y,k) = \begin{cases} 1 & \text{if for all } i=0,\ldots,n-1 \text{ we have } x_i = y_{(i+k) \bmod n} \\ 0 & \text{otherwise} \end{cases}$$

The predicate id_1 tests whether y is the same as x rotated cyclically by k positions.

THEOREM 6: With the choice of x and y as the marked input bits of id_1, id_1 is $\Omega(n)$-separable.

PROOF: The proof is an application of Theorem 10 in Section 6 to the cyclic shifts function defined in Section 6.□

By Theorem 5, for every chip computing id_1 we have $AT^2=\Omega(n^2)$.

We can modify the predicate p_1 in a similar way to carry Theorem 4 over to VLSI. In particular, this yields a predicate p_2 of N inputs such that for every chip computing p_2 we have $AT^2=\Omega(N^2)$. However, in a straightforward modification of the VLSI model to allow coin tossing on a chip there is a chip that computes p_2 with $AT^2=O(N^{3/2}\text{poly}(\log N))$. For details see [MM82].

6. THE COMMUNICATION COMPLEXITY OF BOOLEAN FUNCTIONS WITH MANY OUTPUTS

Boolean functions with many outputs do not fit the framework outlined in Section 5. This is, because just to put the output bits on the communication channel will cost a lot. But in VLSI an output bit does not have to be made known to the whole chip. Rather it just has to be known to the output port that produces it. Thus in multiple output functions output bits may be computed "locally" and we have to measure how much information has to be gathered from far away to compute each output bit.

Therefore when dividing the inputs up between L and R we assign sides to the output bits as well. The corresponding definition is the following.

DEFINITION 9: Let $f:X\times Y\to B^m$. Let I,J be a partition of the output bits. We assume that L has to compute I and R has to compute J.

A deterministic algorithm is again given by two response functions as in Definition 2. But now for each input $x\in X$ and output bit $i\in I$ we have a partial output function $a(x,i):B^*\to B$. Similarly, for each input $y\in Y$ and output bit $j\in J$ we have a partial output function $a(y,j):B^*\to B$. The computation proceeds as in Definition 2. To avoid conflicts we assume that if w is a prefix of w' and $a(x,i)(w)$ is defined then $a(x,i)(w')$ is defined and $a(x,i)(w')=a(x,i)(w)$. (Similarly for $a(y,j)$.) The computation stops as soon as $a(x,i)(w)$ is defined for all $i\in I$ and $a(y,j)(w)$ is defined for all $j\in J$. $C_{det}(f,L\leftrightarrow R)$ is defined as in Definition 2.

Definition 9 differs from Definition 2 in that the partial inputs x resp. y can help in determining an output. If m=1 then Definition 8 yields a notion of communication complexity which is no greater and at most 2 less than the communication complexity defined in Definition 2. This is, because once a side has computed the unique output bit it can make it globally known by transmitting it to the other side.

The concept of non-determinism does not make sense for multiple output functions. Las Vegas algorithms can be defined analogously to Definition 4.

There is one method for proving lower bounds on the communication complexity of multiple output functions.

METHOD 3 (Crossing Sequences for Multiple Output Functions):

DEFINITION 10: Let $f:X\times X\to B^m$ and let I,J be a partition of the output bits of f. f has *ω-flow* if there is a partial input $y\in Y$ such that f restricted to $X\times\{y\}$ in its domain and J in its range has more than $2^{\omega-1}$ different points in its range.

THEOREM 7: If f has ω-flow then $C_{det}(f,L\leftrightarrow R)\geq\omega$.

PROOF: The proof is similar to the proof of Theorem 2. Assume that there is a deterministic algorithm computing f that has a communication length of less than ω. Then for two inputs (x_1,y), (x_2,y) generating different output configurations in J the same communication sequence w is generated. Thus for some $j\in J$ $f_j(x_1,y)\neq f_j(x_2,y)$, but the algorithm computes $f_j(x_1,y)=a(y,j)(w)=f_j(x_2,y)$. Thus the algorithm does not compute f correctly, a contradiction.□

Theorem 7 can be applied to get lower bounds on the AT^2 complexity of chips computing multiple output functions. The corresponding definition and Theorem are the following.

DEFINITION 11 Let $f:B^n \to B^m$. Let A be a subset of the input bits and B be a subset of the output bits containing the "marked" I/O bits. f is called *ω-separable* if for all partitions of the I/O bits into sets X,Y such that both X and Y contain at most 2/3 of all marked I/O bits we have

$$C_{det}(f,L \leftrightarrow R) \geq \omega.$$

THEOREM 8: If $f:B^n \to B^m$ is ω-separable then $AT^2=\Omega(\omega^2)$ for every chip computing f.

PROOF: Analogous to the proof of Theorem 5.□

[Y80] gives an example of a class of functions to which Method 3 applies.

DEFINITION 12: Let $f(x_1,\dots,x_n,s_1,\dots,s_m)=(y_1,\dots,y_n)$ be a Boolean function. f computes a permutation group G on n elements if for all $g\in G$ there is an assignment $\sigma_1,\dots,\sigma_m$ to the $s_1,\dots,s_m$ such that $f(x_1,\dots,x_n,\sigma_1,\dots,\sigma_m)=(x_{g(1)},\dots,x_{g(n)})$ for all choices of $x_1,\dots,x_n$. We call $x_1,\dots,x_n$ the permutation inputs and $s_1,\dots,s_k$ the control inputs. f is called *transitive* of degree n if G is a transitive group, i.e., if for all $i,j=1,\dots,n$ there is a $g\in G$ such that $g(i)=j$.

The most straightforward example of a transitive function of degree n is the cyclic shift function $cs(x_1,\dots,x_n,s_1,\dots,s_m)=(y_1,\dots,y_n)$ where $n=2^m$ and the $s_1,\dots,s_m$ encode a number k, $0\leq k<n$ and $y_i=x_{(i+k) \bmod n}$. cs computes the transitive group of cyclic permutations. Other examples of transitive functions of degree $\Omega(n)$ are the multiplication of n-bit integers, the multiplication of three $\sqrt{n}\times\sqrt{n}$ matrices, the sorting of n numbers between 0 and n etc.

THEOREM 9: Let $f:B^{n+m}\to B^n$ be transitive of degree n. Then f is n/6-separable.

PROOF: Let G be the transitive group computed by f. The equivalence relation $g(i)=h(i)$ for fixed but arbitrary $i\in\{1,\dots,n\}$ divides G into n equivalence classes of size $|G|/n$. Let A be the set of all permutation input bits and let B be the set of all output bits. Let X,Y be any partition of $A\cup B$ such that $|X|,|Y|\leq 4n/3$. For each input bit i in X and output bit j in Y there are $|G|/n$ group elements $g\in G$ such that $g(i)=j$. Let w.l.o.g. X be no greater than Y and assume that X contains at least as many input bits as output bits. (The other cases can be argued similarly.) Let S be the set of input bits in X and S' be the set of output bits in Y. Then

$$|S|\cdot|S'| \geq n/3 \cdot n/2 = n^2/6.$$

For each of the pairs $(i,j)\in S\times S'$ there are $|G|/n$ group elements matching them. Since there are only a total of $|G|$ group elements there must be one element $g_0\in G$ realizing at least n/6 matchings between inputs in S and outputs in S'. The partial input y realizing the flow sets the control input bits in Y such together with appropriate assigments to the control input bits in X they encode this element g_0. The other input bits in Y are assigned arbitrarily.□

We can establish a relationship between Method 3 and Method 1.

DEFINITION 13: Let $f: X \to B^n$. The *characteristic predicate* of f is the predicate $p_f: X \times B^n \to B$ such that

$$p_f(x,y) = \begin{cases} 1 \text{ if } y=f(x) \\ 0 \text{ otherwise.} \end{cases}$$

The characteristic predicate of the cyclic shift function cs is the predicate id_l defined in Definition 8.

THEOREM 10: Let $f: X \times Y \to B^n$. Let p_f be the characteristic predicate of f. Let I,J be a partition of the output bits of f, and consider the induced partition of the input of p_f. (An input bit of p_f corresponding to an output f of f enters on the side on which the output bit is produced.) If f has ω-flow then p_f fulfills the premise of Theorem 2 with $m=2^{\omega}$.

PROOF: The rows and columns of the identity matrix are given by the 2^{ω} input configurations of p_f that correspond to the 2^{ω} input configurations of f generating all different output configurations in J, together with the correct guesses of all outputs.□

An application of Theorem 10 to the cyclic shift function cs proves Theorem 6.

Theorem 10 shows that as Method 1, Method 3 is also rather weak: If we can prove a lower bound on the communication complexity of f then we can prove the same non-deterministic lower bound on the communication complexity of p_f.

In general proving average case lower bounds on the AT^2-complexity of chips is hard. However, there are examples where it can be done. [TBD] proves an average case lower bound for the Discrete Fourier Transform and for Sorting in a modified VLSI model. We will now prove an average case lower bounds for the cyclic shift function. The proof is based on two observations. First we note that for the cyclic shift function every assignment of the control input bits implements exactly one group element. Second we observe that the worst case lower bound on the number of matchings realized simultaneously by a group element (see proof of Theorem 9) can be extended to an average case lower bound.

LEMMA 3: Let f be a transitive function of degree n computing the transitive group G. Then for each partition of the I/O bits of f as in Theorem 9 at least $|G|/11$ group elements realize each at least $n/12$ matchings across the cut.

PROOF: Otherwise at most $(|G|/11 - 1)n + (10|G|/11 + 1)(n/12 - 1) < n|G|/6$ matchings across the cut are realized in total.□

THEOREM 11: For any chip computing the cyclic shift function cs we have $AT^2=\Omega(n^2)$ on the average.

PROOF: Let C be the cut bisecting the chip. Let $g\in G$ be one of the n/11 group elements that realize n/12 matchings across the cut C. Consider the assignment to the control inputs that encodes g, and any assignment to the 11n/12 permutation inputs that are not matched across C. By varying the n/12 permutation input bits that are matched across C we have to generate $2^{n/12}$ different sequences of bits across C. The average length of each such bit sequence is thus at least

$$\sum_{0 \le j \le 2^{n/12}} j2^j / 2^{n/12} \ge n/12 - 2.$$

Thus the average length for all inputs whose control part encodes g is n/12 -2. This means that $C_{ave}(cs, L \leftrightarrow R) \ge (n/12 - 2)/11$. The rest follows as in the proof of Theorem 5.□

Note that by the argument known to us from Section 5 it can be shown that $C_{ave}(id_1, L \leftrightarrow R)=O(1)$ across any cut bisecting a chip computing id_1. Indeed, one can devise chips that compute id_1 with $AT^2=O(n\ \mathrm{poly}(\log n))$ on the average.

Method 3 also applies to many functions that are not transitive, e.g., matrix multiplication ([SS31]). In most cases the lower bounds thus proved can be (asymptotically) matched with tight upper bounds ([PV80, PV81]). This shows that indeed the communication complexity often dominates the AT^2 complexity of a VLSI chip.

7. A LOWER BOUND ON THE SWITCHING ENERGY OF VLSI CHIPS

In Section 6 we proved lower bounds on the AT^2 complexity of chips computing Boolean functions with multiple outputs by measuring the amount of information that has to flow across a cut bisecting the chip. We argued that each bit needs some time, namely one cycle, to cross the cut, and that because of the limited length of the cut only a limited number of bits can cross simultaneously. The switching energy essentially counts the number of state changes happening in the circuit. More precisely, the switching energy is defined by the following statement:

Each unit of chip area consumes one unit of switching energy every time it changes its state from 0 to 1 or vice versa.

Thus, if we want to prove a lower bound on the switching energy considering one cut is not enough. Many of the state changes could happen in circuit components that do not cross the cut. Therefore we must consider many cuts to cover a large part of the circuit. By proving a lower bound on the amount of information flowing across each cut, and summing over all cuts, we can account for many of the state changes happening in the circuit.

Since we have to consider many cuts at once it will not suffice to consider functions with a great communication complexity in the worst case. However, a property such as the one proved for transitive functions in Lemma 3 will do.

THEOREM 12 ([LM81]): Let f be a transitive function of degree n. Let E be the worst case switching energy consumed by any chip computing f. Let A be the (active) chip area and let T be the worst case computing time. Then

$$c_1AT^2 \geq ET \geq \frac{c_2n^2}{\log \frac{c_3AT^2}{n^2}} \geq 0$$

for appropriate constants $c_1,c_2,c_3>0$.

PROOF: Let $\chi=(\Gamma,A,\Delta)$ be any chip computing f. The upper bound on ET is trivial. We prove the lower bound in two steps. First we defined a set of rectilinear cuts through the chip. Second we count state changes in the circuit components crossing the cuts.

In the first step we cut the chip according to a technique devised by [T79]. To this end we overlay the chip with a square grid of mesh size λ. Then we define cuts $C_1,\ldots,C_Q$ bisecting the chip as shown in Figure 3.

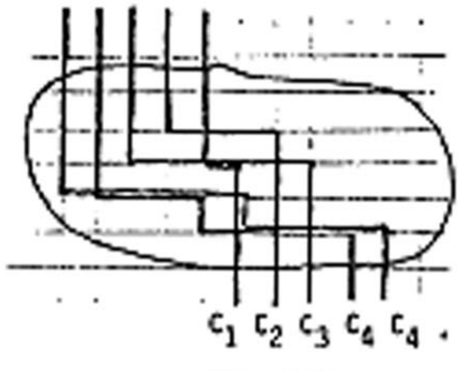

Figure 3

Hereby each cut consists of several sections as shown in Figure 4.

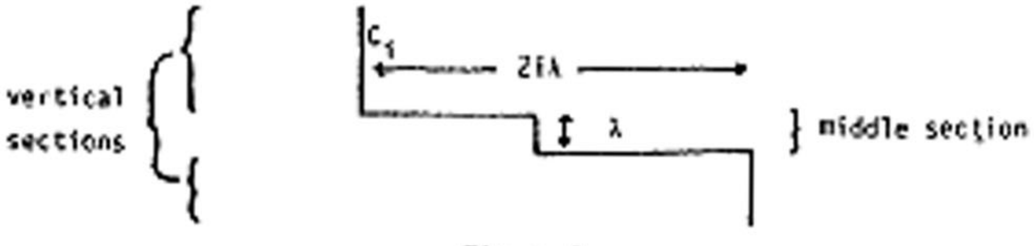

Figure 4

The middle section of cut C_i has a length of at most $(2i-1)\lambda$. The vertical sections of all cuts C_i are disjoint. Each cut induces a partition of the permutation input bits A into sets X,Y and of the output bits B into sets I,J such that $|X|+|I|, |Y|+|J| \leq 4n/3$. We will count state changes happening on the vertical section of the cuts. Let L_i be the number of circuit components crossing C_i and let ℓ_i be the number of circuit components crossing the vertical sections of C_i. Then $\ell_i \geq L_i - c_0 i$ for some appropriate constant $c_0 > 0$.

In the second step we start by noting that for each cut C_i $(1 \leq i \leq Q)$ by Lemma 3 there are $|G|/11$ group elements each realizing at least $n/12$ matchings across C_i. Therefore there has to be one group element $g_0 \in G$ realizing at least $n/12$ matchings across a set Φ of $Q/11$ cuts. We will prove a lower bound on the average case switching energy consumed for computing g_0 on any configuration of the permutation inputs. This is then also a lower bound for the worst case energy for computing f on any input.

Before we start the actual argument we discuss an encoding that expresses bit strings in terms of the state changes happening in them.

DEFINITION 10: Let w be an arbitrary bit string $w = 0^{\ell_1} 1^{\ell_2} 0^{\ell_3} \ldots {}^{\ell_t}$.

a) s(w) is the number of state changes in w, i.e. $s(w) = t$.

b) bin(w) is the bit string obtained from w after substituting each 0 with 00 and each 1 with 11.

c) compress(w) = 0 bin(ℓ_1) 01 bin(ℓ_2) 01 bin(ℓ_3) 01 ... 01 bin(ℓ_t).
If the first bit of w is a 1 so is the first bit of compress(w).

EXAMPLE: compress(00011) = 0 11 11 01 11 00

Lemma 4: $|\text{compress}(w)| \leq 4s(w) + 2s(w) \log \frac{w}{s(w)}$

PROOF: $|\text{compress}(w)| \leq 2s(w) + 2 \sum_{j=1}^{s(w)} (1 + \log \ell_j)$

$$\leq 4s(w) + 2 \sum_{j=1}^{s(w)} \log \ell_j$$

$$\leq 4s(w) + 2s(w) \log \frac{w}{s(w)} \quad .$$

Here the last inequality is derived using the well known fact that the geometric mean is no greater than the arithmetic mean.□

We now consider an arbitrary but fixed cut C_i. We choose an arbitrary but fixed assignment of the $11n/12$ permutation inputs that are not matched across C_i and let the $n/12$ permutation inputs matched across C_i vary in all $2^{n/12}$ possible ways. Thus we generate $2^{n/12}$ different bit sequences (w_{ij}, h_{ij}) across

C_i. Here w_{ij} summarizes all bits crossing C_i in the vertical sections and h_{ij} summarizes all bits crossing C_i in the middle section. $(1 \le j \le 2^{n/12})$.

LEMMA 5: let $\{w_1, \ldots, w_r\}$ be a set of r different bit strings. Then

$$\sum_{1 \le j \le r} |\text{compress}(w)| \ge (\lfloor \log r \rfloor - 2)\, r.$$

PROOF: The mapping $w \to \text{compress}(w)$ is injective. Thus

$$\sum_{1 \le j \le r} |\text{compress}(w)| \ge \sum_{1 \le j < \lfloor \log r \rfloor} j2^j + (r - 2^{\lfloor \log r \rfloor}) \lfloor \log r \rfloor \ge (\lfloor \log r \rfloor - 2) r. \square$$

Using Lemma 5 we now compute an upper bound on the length of compress(w_{ij}) averaged over all $1 \le j \le 2^{n/12}$.

LEMMA 6: $$\sum_{1 \le j \le 2^{n/12}} |\text{compress}(w_{ij})| \ge \left(\frac{n}{12} - c_0 iT - 3\right) 2^{n/12}.$$

PROOF: Since the length of the middle section of cut C_i is at most $(2i-1)\lambda$ there are at most $c_0 i$ circuit components that intersect the middle section of cut C_i. Thus there are at most t, $1 \le t \le 2^{c_0 iT}$, different sequences h_{ij}. For each such sequence there is a number u_t of different sequences w_{ij} such that (w_{ij}, h_{ij}) is generated by some input as described above. We have $U := \sum_{1 \le k \le t} u_k \ge 2^{n/12}$ and by Lemma 5

$$\begin{aligned} \sum_{1 \le j \le 2^{n/12}} |\text{compress}(w_{ij})| &\ge \sum_{1 \le k \le t} (\log u_k - 3) u_k \\ &\ge U \log(U/t) - 3U \\ &\ge 2^{n/12} (n/12 - c_0 iT - 3). \end{aligned}$$

Here we used the fact that $\sum_{1 \le k \le t} u_k \log u_k$ is minimum if for all k $u_k = U/t$. $\square$

Using the upper bound on $|\text{compress}(w)|$ derived in Lemma 4 we can now give a lower bound on the average number of state changes in all sequences w_{ij}. Let

$S_i = \sum_{1 \le j \le 2^{n/12}} s(w_{ij})/2^{n/12}$ be the average number of state changes occurring in any w_{ij} $(1 \le j \le 2^{n/12})$.

LEMMA 7: $(n/12 - c_0 iT - 3) \le 4S_i + 2S_i \log \frac{T\ell_i}{S_i}$.

PROOF: Apply Lemma 4 and Lemma 6 and observe that $\sum_{1 \le j \le 2^{n/12}} s(w_{ij}) \log \frac{T\ell_i}{s(w_{ij})}$ is maximum if for all j, $s(w_{ij}) = S_i$. $\square$

We now sum over all cuts C_i in Φ. Let $\ell := \sum_{i=1}^{Q/11} \ell_i$ and $S := \sum_{i=1}^{Q/11} S_i$. Thus S is the average number of state changes in the sequences w_{ij} across all cuts in Φ.

LEMMA 8: $\sum_{1\le i\le Q/11} (n/12 - c_0QT - 3) \le 4S + 2S \log (T\ell/S)$.

PROOF: We manipulate the formula in a way similar to the proof of Lemma 7.□

Choosing $Q=(n-36)/24c_0T$ yields:

LEMMA 9: For suitable constants $c_2, c_3 > 0$ and sufficiently large n we get

$$ST \ge \frac{c_2 n^2}{\log \frac{c_3 T^2 \ell}{n^2}} .$$

PROOF: Substituting the value for Q into Lemma 8 yields

$$\left(\frac{n-36}{24}\right)^2 \frac{1}{c_0T} \ge 4S \left(1 + \frac{1}{2} \log \frac{T\ell}{S}\right) . \qquad (*)$$

Thus

$$\log S \ge \log \left(\left(\frac{n-36}{24}\right)^2 \frac{1}{4c_0T}\right) - \log \left(1 + \frac{1}{2} \log \frac{T\ell}{S}\right) .$$

Applying the inequality $\log(1+x)\le x$ we get

$$\log S \ge 2 \log \left(\left(\frac{n-36}{24}\right)^2 \frac{1}{4c_0T} \right) - \log(T\ell) .$$

Substituting this into (*) proves the lemma.□

We are left with relating S to the switching energy and bounding ℓ from above. Both can be done with the following argument.

Let c be a component crossing the vertical section of cut C_i. It can be shown that we can charge to c an area of size $\Omega(1)$ inside the layout of component c such that areas charged to different components overlap at most v-fold. From this immediately follows that $\ell=O(A)$. Furthermore $S=O(E+A)$. (We have to add A here for the state changes in w_{ij} that can occur when swutching from one component crossing C_i to the next. Those state changes in w_{ij} do not correspond to quanta of switching energy dissipated on the chip.) Thus

$$ET \ge \frac{c_2 n^2}{\log \frac{c_3 A T^2}{n^2}} - c_4 A.$$

If we assume that on each circuit component there will be at least one state change, say, during initialization, then the last term can be omitted proving the theorem.□

Note that if the chip is AT^2 optimal, i.e., if $AT^2=O(n^2)$ then the above lower bound on E is tight up to a constant factor. This shows that AT^2 optimal chips use their computing resources to capacity (up to a constant factor).

REFERENCES

[BK81] R.P.Brent, H.T.Kung, "The Area-Time Complexity of Binary Multiplication," Journal ACM Vol.28 No.3 (1981), 521-534.

[CM81] B.Chazelle, L.Monier, "A Model of Computation for VLSI with Related Complexity Results," 13th Annual ACM-STOC Conference (1981), 318-325.

[K82] R.Kolla, "Untere Schranken für VLSI," Diplomarbeit, Fachbereich 10, Univ. d. Saarl., Saarbrücken,West Germany (1982).

[LM81] T.Lengauer, K.Mehlhorn, "The Complexity of VLSI Computations," CMU-Conference on VLSI Systems and Computations (1981), 89-99.

[LS81] R.J.Lipton, R.Sedgewick, "Lower Bounds for VLSI," 13th Annual ACM-STOC Conference (1981) 300-307.

[LT79] R.J.Lipton, R.E.Tarjan, "A Separator Theorem for Planar Graphs," SIAM J. Appl. Math. Vol.36 (1979), 177-189.

[MC80] C.Mead, L.Conway, Introduction to VLSI Systems, Addison Wesley, Reading, Mass. (1980).

[MM82] K.Mehlhorn, E.Meineche-Schmidt, "Las Vegas is Better Than Determinism in VLSI and Distributed Computing," 14th Annual ACM-STOC Conference (1982), 330-337.

[PV80] F.P.Preparata, J.Vuillemin, "Area-Time Optimal VLSI Networks for Multiplying Matrices," Inf. Proc. Let. Vol.11 No.2 (1980), 77-80.

[PV81] F.P.Preparata, J.Vuillemin, "Area-Time Optimal VLSI Networks for Computing Integer Multiplication and Discrete Fourier Transform," 8th Int. Colloq. on Automata Theory Languages and Programming (Springer Lecture Notes in Comp. Sci. No.115) (1981), 29-40.

[S81] J.E.Savage, "Planar Circuit Complexity and the Performance of VLSI Algorithms," CMU-Conference on VLSI Systems and Computations (1981), 61-68.

[T80] C.D.Thompson, A Complexity Theory For VLSI, Ph.D. Thesis, Dep. of Comp. Sci., Carnegie-Mellon University (1980).

[V80] J.Vuillemin, "A Combinatorial Limit to the Computing Power of VLSI Circuits," 21st Annual IEEE-FOCS Symposium (1980), 294-200.

[Y79] A.C.Yao, "Some Complexity Questions Related to Distributive Computing," 11th Annual ACM-STOC Conference (1979), 209-213.

[Y81] A.C.Yao, "The Entropic Limitations on VLSI Computations," 13th Annual ACM-STOC Conference (1981), 308-311.

FACHBEREICH 10
UNIVERSITAT DES SAARLANDES
6600 SAARBRÜCKEN
WEST GERMANY