On the Complexity of VLSI Implementations and Graph Representations of Boolean Functions with Application to Integer Multiplication*

Randal E. Bryant
Carnegie Mellon University

July 24, 1998

Abstract

This paper presents lower bound results on Boolean function complexity under two different models. The first is an abstraction of tradeoffs between chip area and speed in very large scale integrated (VLSI) circuits. The second is the ordered binary decision diagram (OBDD) representation used as a data structure for symbolically representing and manipulating Boolean functions. These lower bounds demonstrate the fundamental limitations of VLSI as an implementation medium, and OBDDs as a data structure. They also lend insight into what properties of a Boolean function lead to high complexity under these models.

Related techniques can be used to prove lower bounds in the two models. That is, the same technique used to prove that any VLSI implementation of a single output Boolean function has area-time complexity $AT^2 = \Omega(n^2)$ also proves that any OBDD representation of the function has $\Omega(c^n)$ vertices for some $c > 1$. The converse is not true, however. There are functions for which any OBDD representation is of exponential size, but there is a VLSI implementation of complexity $AT^2 = O(n^{1+\epsilon})$.

Consider an integer multiplier for word size n with outputs numbered 0 (least significant) through $2n-1$ (most significant). For the Boolean function representing either output $i-1$ or output $2n-i-1$, where $1 \leq i \leq n$, the following lower bounds are proved: (1) any VLSI implementation must have $AT^2 = \Omega(i^2)$, and (2) any OBDD representation must have $\Omega(1.09^i)$ vertices.

Index Terms—Binary decision diagrams, Boolean functions, VLSI complexity, multiplication

*This research was supported in part by the Defense Advanced Research Projects Agency, ARPA Order Number 4976, and in part by the Semiconductor Research Corporation, contract number 88-DC-068.

1. Introduction

This paper examines Boolean function complexity under two different models. The first model is an abstraction of digital logic as implemented in very large scale integrated (VLSI) circuit technology. This model provides a framework by which one can analyze potential trade-offs between chip area and computation time in chip designs.

The second model is the ordered binary decision diagram (OBDD) representation [4]. A binary decision diagram [2] represents a Boolean function as a directed acyclic graph with each vertex labeled by a Boolean variable. In an *ordered* binary decision diagram, the vertex labels occur in the same order along all directed paths. This representation has many desirable algorithmic properties. It has proved to work well as a data structure for symbolically representing and manipulating Boolean functions.

In measuring the complexity of Boolean functions, we study a family of related functions, where each member of the family is characterized by a size parameter n. An example of such a family is the set of integer multipliers, where n indicates the word size. The complexity of the family is then expressed in terms of an asymptotic formula in n. Under the VLSI model, a lower bound on the complexity of function is expressed in terms of a formula relating the chip area A, and the computation time T. Under the OBDD model, complexity is expressed in terms of the minimum number of graph vertices for any possible ordering of the input variables.

Note that proving a lower bound for a specific family of functions is a much different task than earlier work proving lower bounds on realizations of arbitrary Boolean functions of n variables. This more classical work, dating to Shannon [10], exploits the fact that there are 2^{2^n} Boolean functions of n variables. Hence, for any circuit model where Boolean functions are realized by networks of primitive elements, all but a vanishingly small fraction of the functions require an exponential number of elements. Such lower bounds serve mainly to indicate that only a small fraction of the possible Boolean functions of n variables have any practical significance whatsoever. On the other hand, if we can prove an undesirable lower bound for an important family of Boolean functions, such as those representing the outputs of an integer multiplier, then we have demonstrated a true weakness in the representation.

Similar techniques can prove that certain families of functions have high complexity under the two models considered in this paper. In particular, Lipton and Sedgewick developed a technique to prove that a single output function has complexity $AT^2 = \Omega(n^2)$ in the VLSI model [7]. We show that a weakened form of this argument can be used to prove that any OBDD representation of the function has at least $\Omega(c^n)$ vertices for some $c > 1$. This "weakening" is strict—there are functions that have efficient VLSI implementations, i.e., $AT^2 = O(n^{1+\epsilon})$, but no polynomial-sized OBDD representation.

Consider an integer multiplier for word size n with outputs numbered 0 (least significant) to $2n-1$ (most significant). We apply Lipton and Sedgewick's technique to the Boolean function representing either output $i-1$ or output $2n-i-1$, where $1 \leq i \leq n$. We show that any VLSI implementation of this function must have complexity $AT^2 = \Omega(i^2)$, while any OBDD must have $\Omega(1.09^i)$ vertices. This is a stronger result than previous VLSI lower bounds for integer multiplication [1, 3], where a lower bound of $\Omega(n^2)$ was proved for any VLSI implementation of the entire multiplier. Our result shows that computing the single output $n-1$ requires almost as much effort as computing the entire product. Our result also confirms a conjecture made by the author [4] that the Boolean functions describing integer multiplication cannot be represented efficiently as OBDD's.

2. Background

2.1. VLSI Complexity Theory

In 1979, Thompson introduced a model of computational complexity based on the properties of VLSI circuits [11]. In this model, and its various refinements [7, 9, 12], computation is viewed as the task of evaluating a Boolean function on a set of Boolean input values. The input values are supplied at designated locations on the chip, and some time later the output value or values appear at their designated locations. Thompson's model highlights the difficulty of communicating information across the two-dimensional surface of an integrated circuit chip. Complexity is measured in terms of the relation between the chip area A and the computation time T. Chip area is expressed in in units λ^2, where λ represents the minimum spacing between a pair of wires. Computation time is expressed in units τ, denoting the time required to propagate a signal along a wire.

For a family of functions parameterized by a "problem size" n, their VLSI complexity is expressed in terms of asymptotic bounds on the area and time required as a function of n. For example, both Abelson and Andreae [1], as well as Brent and Kung [3] showed that any VLSI implementation of an integer multiplier for word size n must have area A and time T satisfying $AT = \Omega(n^2)$. That is, if we consider increasingly larger word sizes, the area and time must scale up such that their AT^2 product grows quadratically.

These proofs rely on techniques first applied by Thompson to prove a lower bound on circuits computing an n-input discrete Fourier transform. The idea of these proofs is to show that if we consider any imaginary line drawn across the chip such that half of the inputs occur on either side, then αn bits of information must cross that line in order to correctly compute the function, for some constant $\alpha > 0$. Thus, if the chip is w units wide, we must have $wT \geq \alpha n$. This property holds across either dimension of a rectangular chip, and hence we can assume that $A \geq w^2$. Combining these yields an area-time lower bound $AT^2 = \Omega(n^2)$. These arguments hold even if some of the inputs are supplied to a single point on the chip sequentially. The only required assumption is that the circuit be "where-oblivious" [7], i.e., each input is supplied only once and at a location independent of its value. An imaginary line can then be used to partition the inputs into two sets, where inputs arriving at points intersected by the line can be placed in either set to equalize their sizes. This proof technique is similar to one developed by Yao [14] for proving lower bounds on communication requirements in a distributed system. However, Yao's method assumed a particular partitioning of the input set, whereas the VLSI argument requires showing high information transfer for any partitioning of the input set into two equal halves.

Observe that any chip computing a function that depends on n inputs must have $AT^2 \geq AT \geq n$ in order to supply the inputs to the chip. Furthermore, the Thompson argument cannot prove a lower bound stronger than $AT^2 = \Omega(n^2)$, since at worst all input values must be communicated across the chip. Thus, the range of bounds proved in the VLSI complexity model is quite limited.

Most of the work on area-time lower bounds has been for multiple output Boolean functions, such as integer multiplication. To prove a lower bound for a multiple output function, one need only prove that some of the inputs can be assigned values 0 and 1 such that a sizable fraction of the remaining input values must be transferred to outputs on the other side of the chip. Lipton and Sedgewick [7] refined the proof technique so that it could be applied to single output functions. Their method requires showing that a large amount of information about the input values on the two sides of the chip must be combined to produce the correct output value. They showed that various decision problems (i.e., Boolean functions where output 1 denotes "yes", and output 0 denotes "no") also have quadratic area-time lower bounds.

Previous arguments that integer multiplication has high VLSI complexity break down when considering individual multiplier outputs. For example, both published lower bound proofs for integer multiplication rely on the observation that multiplication is the same as shifting when one of the arguments has a single input equal to 1. That is, for a multiplier with inputs $A = \{a_{n-1}, \ldots, a_0\}$ and $B = \{b_{n-1}, \ldots, b_0\}$, if input b_j is set to 1, while all other inputs in B are set to 0, then output k will equal input a_{k-j}, for $j \leq k < j + n$. They show that for any partitioning of the chip that the splits the A inputs into two equal halves, a shift value j can be chosen such that for at least $n/4$ values of k, output k occurs on the opposite side from input $k - j$. Thus, at least $n/4$ bits of information must be transferred across the partition to correctly compute all of the output values. Any single output of such a shifter, however, can be implemented efficiently in VLSI. For example, under the restriction that only one of the inputs in B is set to 1, output $n-1$ of the shifter (or multiplier) is given by the Boolean expression

$$a_0 \cdot b_{n-1} + a_1 \cdot b_{n-2} + \cdots + a_{n-1} \cdot b_0. \tag{1}$$

This can be implemented with area $A = O(n)$ and time $T = O(\log n)$ by a set of AND gates and a tree of OR gates. Thus, establishing a lower bound on a single multiplier output requires a more detailed consideration of the properties of multiplication.

2.2. Ordered Boolean Decision Diagrams

In 1986, the author introduced a graph structure for representing Boolean functions [4]. Boolean functions are represented by a restricted class of binary decision diagrams (BDD) [2]. A BDD is a directed acyclic graph in which each vertex is labeled by a Boolean variable and has outgoing arcs labeled 0 and 1. Each arc leads either to another vertex or to a terminal label F or T. The value of the function for a given assignment to the inputs is determined by traversing from the root down to a terminal label, each time following the arc corresponding to the value assigned to the variable specified by the vertex label. The value of the function then equals 1 if the terminal label equals T, and 0 if the terminal label label equals F.

In the theoretical computer science literature, the term "Branching Program" is used instead of "Binary Decision Diagram". Several lower bound results have been proved about a restricted class of graphs called "1-time branching programs" [13]. Under this model, a single variable can appear at most once along any path from the root to a terminal vertex, but variables may appear in any order. Our ordering requirement is more restrictive than the 1-time requirement. Proving lower bounds on 1-time branching programs involves a totally different, and far more difficult proof technique than is the case for OBDD's.

In an *ordered* binary decision diagram, all vertex labels must occur according to a total ordering of the variables. That is, for a set of input variables X, we assign each variable an "index", according to a bijection $\pi: X \rightarrow \{1, \ldots, |X|\}$. If some vertex in the graph is labeled with variable a, and one of its children is labeled with variable b, then we must have $\pi(a) < \pi(b)$. In discussing a variable ordering, we will specify it by the sequence of variables in ascending order. That is, the variable ordering given by π is written as the sequence $[\pi^{-1}(1), \pi^{-1}(2), \ldots, \pi^{-1}(|X|)]$. Figure 1 shows an example of an OBDD for the function $\mathrm{a1} \cdot \mathrm{b1} + \mathrm{a2} \cdot \mathrm{b2} + \mathrm{a3} \cdot \mathrm{b3}$. In this graph, the variables are ordered $[\mathrm{a1}, \mathrm{b1}, \mathrm{a2}, \mathrm{b2}, \mathrm{a3}, \mathrm{b3}]$.

OBDD's have many desirable properties for symbolic Boolean manipulation. In particular, for a given variable ordering π, the smallest OBDD for a function is unique. Any OBDD can be reduced to such a canonical form by a simple, efficient algorithm. Furthermore, when a set of functions is represented by canonical OBDD's for a single ordering π, there are efficient algorithms for such operations as: determining an input assignment that satisfies a function, testing two functions for equivalence, and combining two functions according to a Boolean operation to produce a graph for the resulting function.

Experience has shown that many of the Boolean functions commonly encountered in digital logic designs have efficient (i.e., small) representations as OBDD's. For example, the function of Equation 1 can be represented by an OBDD of $2n$ vertices, for variable ordering $[a_0, b_{n-1}, a_1, b_{n-2}, \ldots, a_{n-1}, b_0]$. The size of an OBDD for a function, however, can be quite sensitive to the variable ordering chosen. For example, Figure 2 shows the OBDD for the same function as in Figure 1, but for variable ordering $[\mathrm{a1}, \mathrm{a2}, \mathrm{a3}, \mathrm{b1}, \mathrm{b2}, \mathrm{b3}]$. If we generalize the function of Figures 1 and 2 to a function over $2n$ variables, then the size of the OBDD representation can range between $2n$ and $2^{n+1} - 2$, depending on the variable ordering.

This sensitivity to variable ordering has prompted several efforts at developing strategies for choosing a good ordering. Recently, several heuristic methods have been published [5, 8] that produce good results for a number of gate-level combinational circuit benchmarks. These researchers were able to generate OBDD representations for the functions computed by combinational circuits ranging up to one with 207 inputs, 108 outputs, and 3719 gates.

Our interest in this paper is in investigating those functions that do not have an efficient OBDD representation regardless of the variable ordering. By studying the properties of these functions, we gain a better understanding of the strengths and weaknesses of the

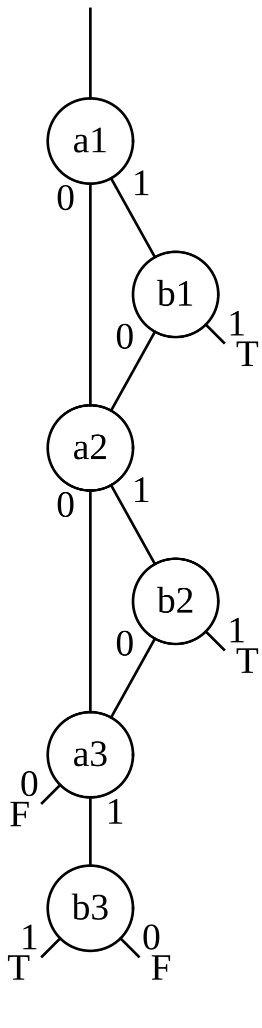

Figure 1: **OBDD with a Good Choice of Variable Ordering.**

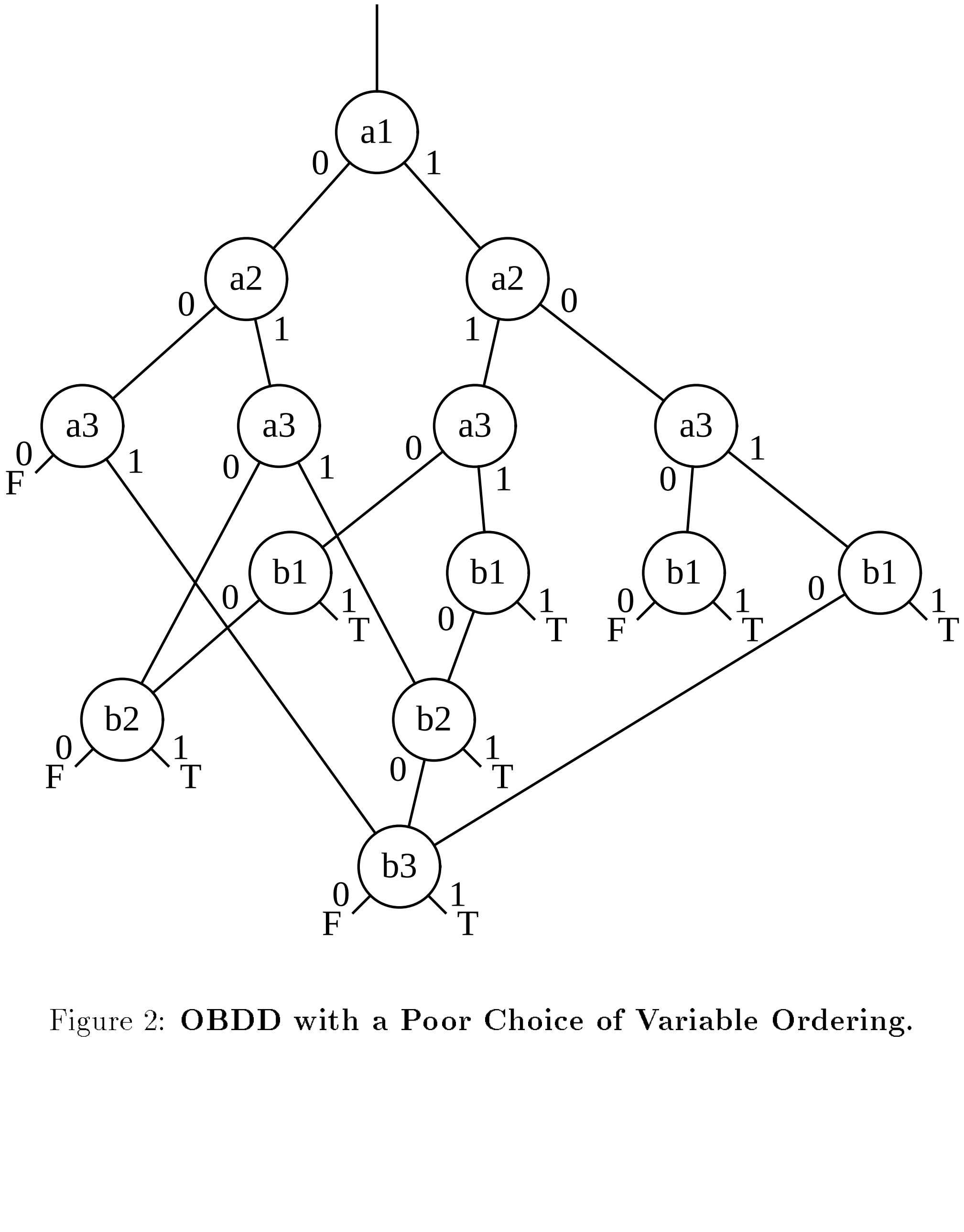

Figure 2: **OBDD with a Poor Choice of Variable Ordering.**

OBDD representation. Observe that the lower bounds we prove for OBDD representations of functions are far more severe than the VLSI lower bounds. If the OBDD complexity scales exponentially in n, then our algorithms will be practical only for small values of n.

As a step in this direction, the author [4] proved the following result for integer multiplication: for any variable ordering π, there is some i, where $0 \leq i < 2n$ such that the function corresponding to output i of an n-bit multiplier requires an OBDD with $\Omega(1.09^n)$ vertices. This left open the possibility that if one were allowed to choose a different ordering for each output, all of the outputs could be represented efficiently as OBDD's. This paper eliminates that possibility.

3. Techniques for Proving Lower Bounds

Let f denote a family of single output Boolean functions parameterized by the problem size n. The set of inputs is denoted by the input set X. Define an *input assignment* $x: X \rightarrow \{0,1\}$ as an assignment of Boolean values to the inputs. Given an input assignment x, the resulting output value is denoted $f(x) \in \{0,1\}$.

Our lower bound arguments for both VLSI and OBDD complexity have the same general form. Some subset of the inputs $Y \subseteq X$ is designated as the set of "key" inputs, $|Y|$, and a real-valued balance parameter ω between 0 and 1 is specified. Most often $\omega = 0.5$. Define a *balanced partitioning* as any partitioning of X into subsets L and R such that the fraction in L equals ω. That is,

$$\lfloor \omega |Y| \rfloor \quad \leq \quad |Y \cap L| \quad \leq \quad \lceil \omega |Y| \rceil$$

Proving a lower bound involves showing that for every balanced partition, computing f requires that a large amount of information about the assignments to inputs in L must be combined with a large amount of information about the assignments to inputs in R. In the case of a VLSI circuit, this high information transfer implies that either many wires or much time must be allocated to communicate values across the chip. In the case of an OBDD, this high information transfer implies that the graph must have many arcs crossing from vertices labeled by variables in L to vertices labeled by variables in R. For both types of lower bound arguments, we establish a lower bound on information transfer by creating a "fooling set" for the input partition, i.e., a set of input assignments no two elements of which can be computed by communicating the same information across the partition.

For a particular partitioning, define a *left* (respectively, *right*) input assignment $l: L \rightarrow \{0,1\}$ (resp., $r: R \rightarrow \{0,1\}$) as an assignment of Boolean values to the inputs in L (resp., R). Let $l \cdot r$ denote the complete input assignment resulting from left input assignment l and right input assignment r.

For VLSI, a fooling set for partition (L, R) is a set, $\mathcal{A}_{\mathrm{VLSI}}(L, R)$ of input assignments having the following properties:

> There is some $z \in \{0,1\}$, such that $f(x) = z$ for all $x \in \mathcal{A}_{\mathrm{VLSI}}$, but for any two distinct assignments $l \cdot r$ and $l' \cdot r'$ in $\mathcal{A}_{\mathrm{VLSI}}(L, R)$, either $f(l \cdot r') \neq z$ or $f(l' \cdot r) \neq z$.

Lemma 1 (Lipton and Sedgewick) *If for every balanced partition (L, R), function f has a fooling set $\mathcal{A}_{\mathrm{VLSI}}(L, R)$ containing at least $2^{\alpha n}$ elements, for some $\alpha > 0$, then any VLSI implementation of f must have area-time complexity $AT^2 = \Omega(n^2)$.*

Proof: See [7]. In brief, the argument involves showing that any chip computing f must transfer at least αn bits of information across the chip. In particular, imagine a line drawn across the chip partitioning the inputs into those supplied to the left hand side of the chip L and those supplied to the right hand side of the chip R. Suppose furthermore that this partitioning is balanced. Consider the sequences of 0's and 1's appearing on the wires crossing this line over the course of the computation (called the *crossing sequence*). This crossing sequence is the only information provided to the right hand side of the chip about the inputs on the left hand side, and vice versa. If the output is on the right hand side of the chip, then its value is fully determined by the right input assignment and the crossing sequence. Similarly, if the output is on the left hand side of the chip, then its value is fully determined by the left input assignment and the crossing sequence.

Suppose that input assignments $l{\cdot}r$ and $l'{\cdot}r'$ in $\mathcal{A}_{\mathrm{VLSI}}(L, R)$ cause the same crossing sequence. Then input assignments $l'{\cdot}r$ and $l{\cdot}r'$ would also cause this crossing sequence. If the output appears on the right hand side of the chip, then input $l'{\cdot}r$ must cause the same output value as input $l{\cdot}r$, implying that $f(l'{\cdot}r) = f(l{\cdot}r) = z$. Similarly, if the output appears on the left hand side of the chip, then input $l'{\cdot}r$ must cause the same output value as $l'{\cdot}r'$, implying that $f(l'{\cdot}r) = f(l'{\cdot}r') = z$. A similar argument, however, shows that $f(l{\cdot}r') = z$. These conditions cannot all hold, and hence input assignments $l{\cdot}r$ and $l'{\cdot}r'$ must cause different crossing sequences.

Each input assignment in $\mathcal{A}_{\mathrm{VLSI}}(L, R)$ must cause a unique crossing sequence, and this requires communicating $\log|\mathcal{A}_{\mathrm{VLSI}}(L, R)| \geq \alpha n$ bits across the partition. For a chip w wires wide, at most wT bits can be transferred in T time units. Thus, $AT^2 \geq (wT)^2 \geq (\alpha n)^2$.

□

For OBDD's, a fooling set consists of left input assignments. That is, for partition (L, R), a set of left input assignments $\mathcal{A}_{\mathrm{OBDD}}(L, R)$ is a fooling set if it satisfies the following property:

> For any two distinct l and l' in $\mathcal{A}_{\mathrm{OBDD}}$, there exists a right input assignment r such that $f(l'{\cdot}r) \neq f(l{\cdot}r)$.

Such a right assignment is said to *distinguish* between l and l'.

Lemma 2 *If for every balanced partition (L, R), function f has a fooling set $\mathcal{A}_{\mathrm{OBDD}}(L, R)$ containing at least c^n elements, for some $c > 1$, then any OBDD for f must have $\Omega(c^n)$ vertices.*

Proof: For variable ordering $\pi: X \to \{1, \ldots, |X|\}$, let k be a value $1 \leq k \leq |X|$ such that fraction ω of the elements of Y are among the first k variables, i.e., the partition defined as

$$L \;=\; \{\pi^{-1}(i) \mid 1 \leq i \leq k\}$$

$$R \;=\; \{\pi^{-1}(i) \mid k+1 \leq i \leq |X|\}$$

is balanced. Consider any OBDD representing function f for ordering π. A left input assignment defines a path in the OBDD from the root to either a terminal label or to a vertex labeled by a variable in R. Suppose that fewer than $|\mathcal{A}_{\rm OBDD}(L,R)|$ such destinations are unique. Then there must be two distinct left input assignments l and l' in $\mathcal{A}_{\rm OBDD}(L,R)$ leading either to a single terminal label or to a single vertex. In the former case, the function value is independent of the values assigned to the variables in R, and hence $f(l \cdot r) = f(l' \cdot r)$ for any right input assignment r. In the latter case, from any vertex v labeled by a variable in R, a right input assignment r defines a path from v to a terminal label T or F. Thus, for any right input assignment r, input assignments $l \cdot r$ and $l' \cdot r$ each define paths in the OBDD from the root to the same terminal label, implying that $f(l \cdot r) = f(l' \cdot r)$. Both of these cases violate the property of the fooling set. Hence the OBDD must contain at least $|\mathcal{A}_{\rm OBDD}(L,R)| - 2 = \Omega(c^n)$ distinct vertices.

□

The Boolean function $\mathrm{a1} \cdot \mathrm{b1} + \mathrm{a2} \cdot \mathrm{b2} + \mathrm{a3} \cdot \mathrm{b3}$ illustrates the concept of an OBDD fooling set. For the partition $L = \{\mathrm{a1}, \mathrm{a2}, \mathrm{a3}\}$ and $R = \{\mathrm{b1}, \mathrm{b2}, \mathrm{b3}\}$, the set of all 8 possible left input assignments is a fooling set. To see this, suppose two left input assignments differ in the value assigned to input a_i. Then the right input assignment that assigns 1 to b_i and 0 otherwise distinguishes between the two left input assignments. Hence, the graph in Figure 2 contains 8 arcs leading from vertices labeled by the variable a3. One leads to terminal label F, but the rest lead to distinct vertices labeled by variables in R. As Figure 1 illustrates, however, not all partitions have such large fooling sets. For the partition $L = \{\mathrm{a1}, \mathrm{b1}, \mathrm{a2}\}$ and $R = \{\mathrm{b2}, \mathrm{a3}, \mathrm{b3}\}$, no fooling set has more than 3 elements.

4. Relation between the Models

The two proof techniques described in the previous section are clearly very similar. In both cases, we show that for any balanced partitioning of the inputs (L, R), a large amount of information about the inputs in the two partitions must be combined to compute f. In the case of VLSI, this information is encoded as different bit combinations sent along wires, giving a quadratic area-time lower bound. In an OBDD, this information is encoded as distinct arc destinations, giving an exponential lower bound. The difference between the two is due to the acyclic nature of BDD's. On a chip, information can flow in both directions across the partition. In a BDD, there must be enough distinct destinations to encode all information about the left hand input values that is needed to compute the function given the right hand input values. As will be shown, this distinction allows some functions to have efficient VLSI implementations, but no efficient OBDD representation.

The following theorem shows that when Lipton and Sedgewick's method gives a quadratic area-time lower bound for a function, then any OBDD representation of this function must have exponential size, where $c = 2^{\alpha}$.

Theorem 1 *For any VLSI fooling set $\mathcal{A}_{\rm VLSI}$, there is an OBDD fooling set $\mathcal{A}_{\rm OBDD}$ of the* *same size.*

Proof: Given $\mathcal{A}_{\mathrm{VLSI}}$, a fooling set $\mathcal{A}_{\mathrm{OBDD}}$ can be constructed by including each left input assignment l such that there is some input assignment $l \cdot r$ in $\mathcal{A}_{\mathrm{VLSI}}$. Observe that $\mathcal{A}_{\mathrm{VLSI}}$ cannot contain two distinct elements $l' \cdot r$ and $l \cdot r'$ for which $l = l'$, or else we would have $f(l{\cdot}r') = f(l'{\cdot}r') = f(l{\cdot}r) = f(l'{\cdot}r)$. Hence $|\mathcal{A}_{\mathrm{OBDD}}| = |\mathcal{A}_{\mathrm{VLSI}}|$. Furthermore, for any 2 distinct elements $l \cdot r$ and $l' \cdot r'$ in $\mathcal{A}_{\mathrm{VLSI}}$, either $f(l \cdot r') \neq z = f(l' \cdot r')$ implying that r' distinguishes l from l', or $f(l' \cdot r) \neq z = f(l \cdot r)$ implying that r distinguishes l from l'.

□

We can prove by counterexample, however, that the converse does not hold. The *hidden weighted bit* function is an example of one that requires an OBDD of exponential size, but has a VLSI implementation with low area-time complexity. This function has n inputs: $X = \{a_n, \ldots, a_1\}$. For input assignment x, define its "weight" to be the number of inputs set to 1. That is

$$wt(x) \quad = \quad \Big|\{a_i \mid 1 \leq i \leq n, \text{ and } x(a_i) = 1\}\Big|$$

The hidden weighted bit function selects the ith input, where $i = wt(x)$:

$$HWB(x) \quad = \quad \begin{cases} 0, & wt(x) = 0 \\ x\Big(a_{wt(x)}\Big), & wt(x) > 0 \end{cases}$$

Theorem 2 *Any OBDD representation of HWB requires $\Omega(1.14^n)$ vertices.*

Proof: Assume initially that n is a multiple of 10. Consider the set of balanced partitions for $Y = X$, and $\omega = 0.6$. Let X_H and X_L denote two sets of $0.4n$ inputs each in the center of the input range:

$$\begin{aligned} X_H &= \{a_{.9n}, \ldots a_{.5n+1}\} \\ X_L &= \{a_{.5n}, \ldots a_{.1n+1}\} \end{aligned}$$

Since the right partition R contains only $0.4n$ elements, and $|X_H| + |X_L| = 0.8n$, either $|X_H \cap L| \geq 0.2n$ or $|X_L \cap L| \geq 0.2n$. Therefore, we can choose a set W of size $0.2n$ where either $W \subseteq X_H \cap L$ or $W \subseteq X_L \cap L$.

For the case where $W \subset X_H$, define the fooling set $\mathcal{A}_{\mathrm{OBDD}}$ as those left input assignments l such that exactly half the inputs in W, as well as all the inputs in $L - W$ are set to 1. Observe that any of these left input assignments has $0.5n$ inputs set to 1. For the case where $W \subset X_L$, define the fooling set $\mathcal{A}_{\mathrm{OBDD}}$ as those left input assignments l such that exactly half the inputs in W, as well as all the inputs in $L - W$ are set to 0. Observe that any of these left input assignments has $0.1n$ inputs set to 1. In either case:

$$|\mathcal{A}_{\mathrm{OBDD}}| \quad = \quad \binom{n/5}{n/10} \quad = \quad \Omega\left(\frac{2^{n/5}}{\sqrt{n}}\right) \quad = \quad \Omega\Big(2^{(1/5-\epsilon)n}\Big) \quad = \quad \Omega(1.14^n)$$

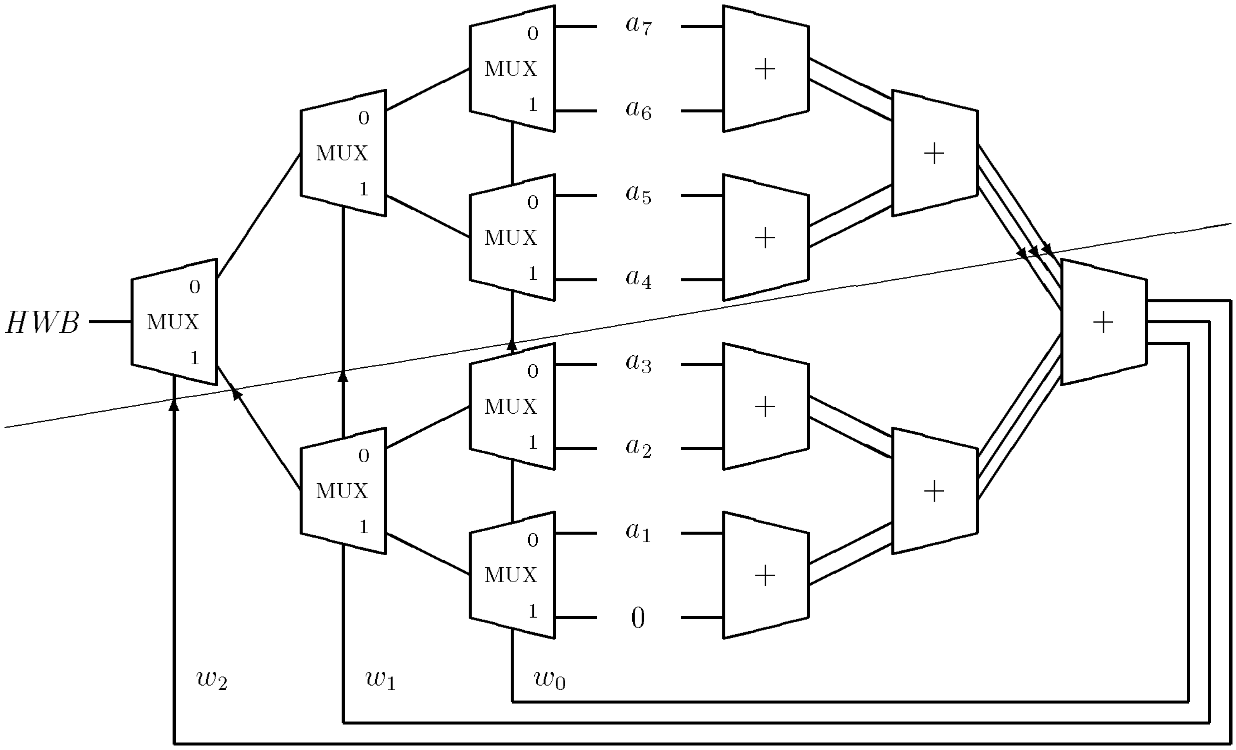

Figure 3: **Circuit Design for the 7-Input Hidden Weighted Bit Function.** Arrows indicate information flow across the partition represented by the diagonal line.

The above can be derived using Stirling's approximation [6, Problem 1.2.6.46].

To show that $\mathcal{A}_{\text{OBDD}}$ is a fooling set, consider any elements l and l' such that $l(a_i) \neq l'(a_i)$ for some i. In the case where $W \subset X_H$, we must have $0.5n < i \leq 0.9n$. Therefore, if we let r be any right input assignment such that exactly $i - 0.5n$ inputs in R are set to 1, then $wt(l{\cdot}r) = wt(l'{\cdot}r) = i$, and

$$HWB(l{\cdot}r) \;=\; l(a_i) \;\neq\; l'(a_i) \;=\; HWB(l'{\cdot}r) \tag{2}$$

Similarly, in the case where $W \subset X_L$, we must have $0.1n < i \leq 0.5n$. Therefore, if we let r be any right input assignment such that exactly $i - 0.1n$ inputs in R are set to 1, then $wt(l{\cdot}r) = wt(l'{\cdot}r) = i$, and Equation 2 holds.

For the case where n is not a multiple of 10, observe that for any $n' < n$, the hidden weighted bit function over n' inputs can be embedded in the hidden weighted bit function over n inputs simply by setting inputs $a_{n'+1}$ through a_n to 0. Hence, any OBDD fooling set for the hidden weighted bit function over $n' = 10\lfloor n/10 \rfloor$ inputs can be transformed into an OBDD fooling set for the function over n inputs. This will affect our lower bound by only a small constant factor.

□

Theorem 3 *There is a VLSI implementation of HWB with area-time complexity* $AT^2 =$

$O(n^{1+\epsilon})$ for any $\epsilon > 0$.

Proof: Consider the following circuit design, where $m = \lceil \log(n+1) \rceil$. Figure 3 illustrates the design for $n = 7$. The circuit inputs are fed as data into both a balanced tree of adders and a balanced tree of 2-input multiplexors. The adder tree computes a binary representation of $wt(x)$. The m wires at the root of the tree, $w_{m-1}, \ldots w_0$, serve as control signals for the multiplexor tree, causing the value on $a_{wt(x)}$ to appear at the multiplexor tree output.

A straightforward implementation of this circuit would use m-bit adders with area and delay $O(\log n)$. The adder tree, having delay $O(\log^2 n)$, width $O(n \log n)$, and height $O(\log n)$, would dominate both the layout area and the circuit delay. Thus, the chip would have area-time complexity $AT^2 = O(n \log^6 n) = O(n^{1+\epsilon})$.

□

On a practical note, the area-time complexity could be reduced considerably by using carry-save adders and an H-tree layout [12]. Furthermore, as Figure 3 illustrates, the word sizes of the adders in the adder tree could be scaled up from 1 at the inputs to m at the root. This would yield a circuit with area $O(n)$ and delay $O(\log n)$.

It is interesting to observe how this VLSI implementation exploits two-way communication. If we consider an imaginary line across the chip that divides the inputs into two equal halves, we see around $\log n$ bits of information crossing in both directions. For example, if we consider the diagonal line in Figure 3, a set of control signals encoding the weight of the last $n/2$ inputs crosses from top to bottom. This weight is added to the weight of the remaining inputs, and the resulting value crosses from the bottom to the top as multiplexor control signals. The only data signal crossing the partition is the output of the bottom half of the multiplexor tree, in the event that the selected bit is among the first inputs. Suppose, on the other hand, that information were restricted to flow upward across the partition. Then there would be no way to communicate the weight of the last $n/2$ inputs to the bottom part of the chip. We would need to transfer $\Omega(n)$ data bits, since there would be no way to accurately determine which bit is required. An OBDD representation of *HWB* encounters exactly this problem. The set of vertices labeled by variable a_i in the graph must encode both the weight and the values of all variables a_j with $\pi(a_j) \leq \pi(a_i)$.

5. Integer Multiplication

Consider an n-bit multiplier having inputs $X = A \cup B$, where $A = \{a_{n-1}, \ldots a_0\}$, and $B = \{b_{n-1}, \ldots b_0\}$. Input sets A and B represent the two n-bit arguments to the multiplier, where a_{n-1} and b_{n-1} are the most significant bits, and a_0 and b_0 are the least significant bits. Suppose the multiplier outputs are numbered, starting with the least significant bit, from 0 to $2n-1$. Let $Mult_i^n$ denote the Boolean function representing output i of an n-bit multiplier. We will first focus on the function $Mult_{n-1}^n$.

Theorem 4 *For the function $Mult_{n-1}^n$, any VLSI implementation has area-time complexity $\Omega(n^2)$, and any OBDD representation has $\Omega(1.09^n)$ vertices.*

Consider the set of balanced partitions of the inputs when $Y = A$, and $\omega = 0.5$. For convenience, assume n is even. The case where n is odd is handled by an embedding argument similar to the one in the proof of Theorem 2. The proof of the theorem follows directly from the following lemma, and the fact that $2^{1/8} > 1.09$:

Lemma 3 *For any balanced partitioning (L, R), the Boolean function $Mult^n_{n-1}$ has a VLSI fooling set such that $|\mathcal{A}_{\mathrm{VLSI}}(L, R)| \geq 2^{n/8}$.*

Proof: The general idea of the proof is as follows. Given any balanced partition, two words U and V can be embedded in the upper and lower halves of input A, such that for s values of i where $s \geq n/8$, inputs u_i and v_i are split between partitions L and R. Two bits of input B are set to 1, while all others are set to 0, such that output $n-1$ of the multiplier equals the high order output bit in the sum of U and V. The fooling set consists of 2^s different input assignments x for which $x(u_i) = \neg x(v_i)$ for every bit position i, giving $Mult^n_{n-1}(x) = 1$. For any two distinct input assignments $l \cdot r$ and $l' \cdot r'$ in the fooling set, we can show that at least one of $Mult^n_{n-1}(l' \cdot r)$ and $Mult^n_{n-1}(l \cdot r')$ equals 0.

More formally, divide input A into its low and high order halves, i.e., $A_U = \{a_{n-1}, \ldots, a_{n/2}\}$ and $A_D = \{a_{n/2-1}, \ldots, a_0\}$. Define a partitioning of A into 4 sets, according to its low and high order halves, as well as its partitioning between L and R as follows:

$$\begin{aligned} A_{UL} &= A_U \cap L \\ A_{DL} &= A_D \cap L \\ A_{UR} &= A_U \cap R \\ A_{DR} &= A_D \cap R \end{aligned}$$

Observe that if $|A_{UL}| = k$, then $|A_{DL}| = |A_{UR}| = n/2 - k$, and $|A_{DR}| = k$.

The integer p, where $-n/2 < p < n/2$, is introduced to denote an alignment of the lower half of A relative to the upper half. That is, $p = 0$ represents the case where the two halves align exactly. A value $p > 0$ represents an alignment where the lower half is shifted right p positions with respect to the upper half. A value $p < 0$ represents an alignment where the lower half is shifted left $-p$ positions with respect to the upper half. For a given alignment, the set $Args_p$ denotes the set of input pairs formed. For $p \geq 0$, this set is defined as:

$$Args_p = \{\langle a_{i+n/2}, a_{i+p}\rangle \mid 0 \leq i < n/2 - p\}$$

For $p < 0$, this set is defined as:

$$Args_p = \{\langle a_{i+n/2-p}, a_i\rangle \mid 0 \leq i < n/2 + p\}$$

Define the set $Split_p$ to be those input pairs that are split between L and R:

$$Split_p = Args_p \cap \Big[(A_{UL} \times A_{DR}) \cup (A_{UR} \times A_{DL})\Big]$$

We claim that for some value p, the set $Split_p$ contains at least $n/8$ elements. To see this, observe that each input $a_i \in A_{UL}$ combined with each input $a_j \in A_{DR}$ contributes exactly one element to $Split_p$ for exactly one value of p. Namely, it contributes an element to $Split_{j-i+n/2}$. Similarly, each input $a_i \in A_{UR}$ combined with each input $a_j \in A_{DL}$ contributes exactly one element to $Split_p$ for exactly one value of p. Therefore, the sum of the number of elements in all of these sets equals the total numbering of pairs in $A_{UL} \times A_{DR} \cup A_{UR} \times A_{DL}$. That is, if $|A_{UL}| = k$, then:

$$\begin{aligned}\sum_{-n/2<p<n/2} |Split_p| &= |A_{UL} \times A_{DR}| + |A_{UR} \times A_{DL}| \\ &= k^2 + (n/2-k)^2 \\ &\geq n^2/8\end{aligned}$$

The latter inequality can be shown by minimizing $k^2 + (n/2-k)^2$ with respect to k. The minimum occurs when $k = n/4$. Now suppose that $|Split_p| < n/8$ for all values of p. This would imply that

$$\begin{aligned}\sum_{-n/2<p<n/2} |Split_p| &< \sum_{-n/2<p<n/2} n/8 \\ &= (n-1)n/8 \\ &< n^2/8,\end{aligned}$$

a contradiction. Hence, $Split_p$ contains at least $n/8$ elements for some value p.

The value of p from the previous paragraph determines our embedding of two m-bit input words U and V into the upper and lower halves of A, where $m = n/2 - |p|$. Define the set U as $U = \{u_{m-1}, u_{m-2}, \ldots u_0\}$, where

$$u_i = \begin{cases} a_{i+n/2-p}, & p < 0 \\ a_{i+n/2}, & p \geq 0 \end{cases}$$

Define the set V as $V = \{v_{m-1}, v_{m-2}, \ldots v_0\}$, where

$$v_i = \begin{cases} a_i, & p < 0 \\ a_{i+p}, & p \geq 0 \end{cases}$$

Observe that by this definition the set $Args_p$ consists of all pairs $\langle u_i, v_i \rangle$, for $0 \leq i < m$, and that the set $Split_p$ consists of a subset of these pairs.

The input assignments in the fooling set $\mathcal{A}_{\mathrm{VLSI}}(L, R)$ differ only for those inputs u_i and v_i such that $\langle u_i, v_i \rangle \in Split_p$. Let us restrict our attention to a class of input assignments χ, such that each $x \in \chi$ obeys the following.

1. Among the inputs in A, only those appearing in the words U and V can be nonzero. That is, $x(a_i) = 0$ for each $a_i \in A - (U \cup V)$.
2. Exactly 2 of the inputs in B are set to 1. That is, $x(b_p) = x(b_{n/2}) = 1$ when $p \geq 0$, and $x(b_0) = x(b_{n/2-p}) = 1$ when $p < 0$. $x(b_i) = 0$ for all other i such that $0 \leq i < n$.

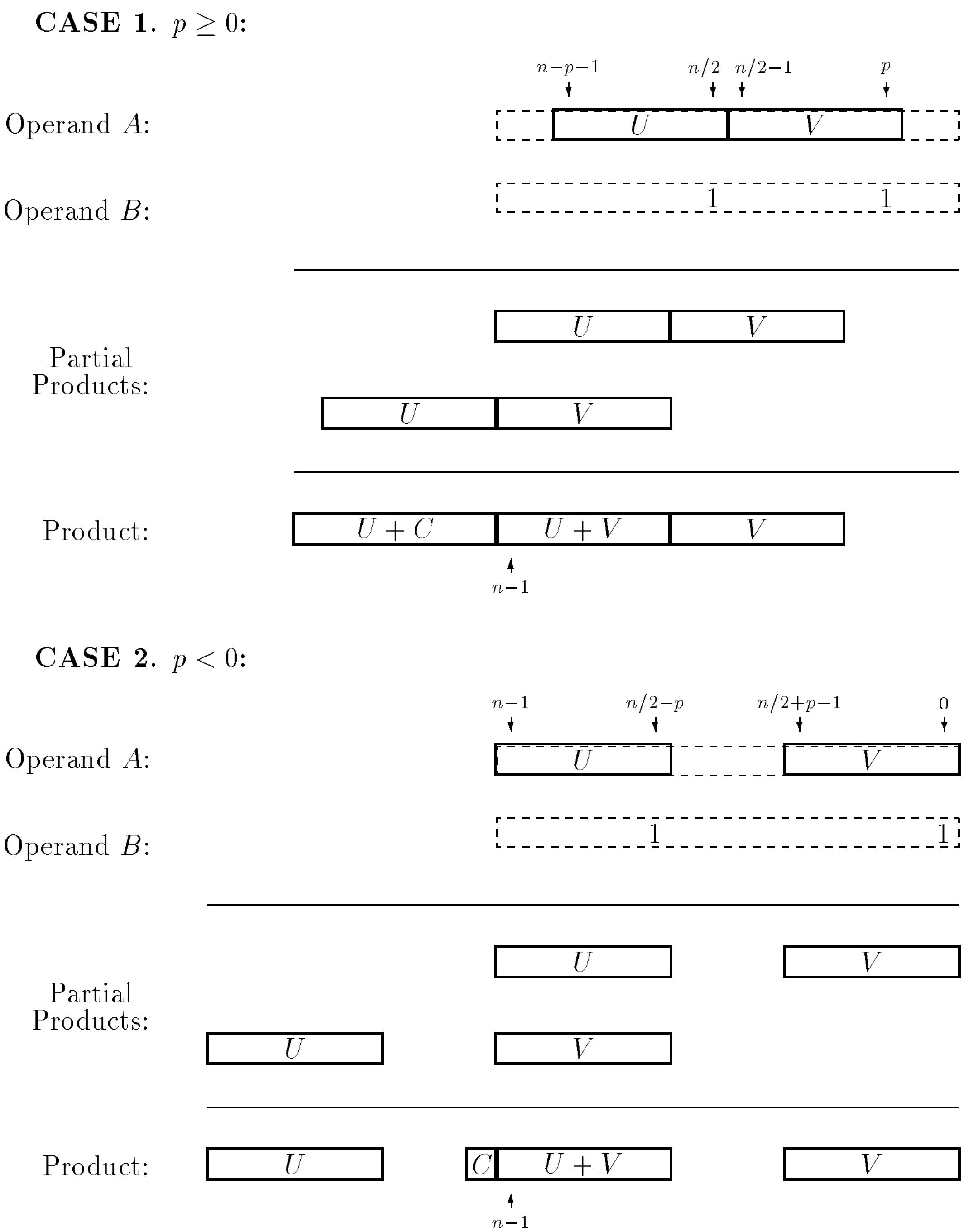

Figure 4: **Embedding of arguments U and V.** Output $n-1$ is the high order bit in the sum of U and V.

3. Those argument pairs that are not in $Split_p$ are set to fixed values. That is, $x(u_i) = 1$, and $x(v_i) = 0$ for each $\langle u_i, v_i \rangle \in Args_p - Split_p$.

The first two restrictions guarantee that multiplier output $n-1$ equals the high order bit in the sum of integers represented by the words U and V. This is illustrated in Figure 4. As is shown in the figure, the two inputs in B that are set to 1 cause two partial products: one in which the high order bit of U is at position $n-1$, and one in which the high order bit of V is at position $n-1$. Furthermore, none of the low order terms in the partial products causes a carry into the sum of U and V. On the other hand, the sum of U and V can cause a carry value (shown as C) into the high order terms, but this will not affect the value of output $n-1$. The third convention is chosen to cause a carry propagation across the positions in the addition where the inputs are not split between L and R.

Observe that the restriction imposed on elements in χ are all of the form that certain inputs are fixed at 0 or 1. Thus, for any two assignments $l \cdot r$ and $l' \cdot r'$ in χ, the assignments $l' \cdot r$ and $l \cdot r'$ must also be in χ. This makes it possible to demonstrate that some subset of χ is a fooling set by studying the high order bit in the sum of input words U and V.

Now consider the set $\mathcal{A}$ consisting of those elements of χ for which each u_i and v_i are set to complementary values:

$$\mathcal{A} \quad = \quad \{x \in \chi \mid x(u_i) = \neg x(v_i), \text{ for all } 0 \leq i < m\}$$

Observe that if $Split_p$ contains $s \geq n/8$ elements, then $\mathcal{A}$ contains $2^s \geq 2^{n/8}$ elements.

As has already been noted, for any input assignment in χ, output $n-1$ of the multiplier is the high order bit in the addition of the two m-bit words U and V. This value can be expressed simply as:

$$Mult^n_{n-1}(x) = \begin{cases} x(u_{m-1}) \oplus x(v_{m-1}), & x(u_i) = \neg x(v_i) \text{ for all } 0 \leq i < m-1 \\ x(u_{m-1}) \oplus x(v_{m-1}) \oplus x(u_k), & k = \max\{i \mid 0 \leq i < m-1, \text{ and } x(u_i) = x(v_i)\} \end{cases} \tag{3}$$

where $\oplus$ denotes EXCLUSIVE-OR. Equation 3 is best understood by considering the operation of a carry-ripple adder. Suppose that k is the highest order bit position less than $m-1$ for which the two adder inputs are equal. Then this stage will produce a carry output equal to its two input values. This carry value will propagate through any intervening stages and be EXCLUSIVE-OR'ed with the two inputs at position $m-1$ to produce the output. If, on the other hand, no bit position less than $m-1$ has both inputs equal, then the final stage will have carry input 0.

Each input assignment $x \in \mathcal{A}$ has the property that $x(u_i) = \neg x(v_i)$ for each bit position i. Substituting into the first case of Equation 3 gives

$$Mult^n_{n-1}(x) \quad = \quad \neg x(v_{m-1}) \oplus x(v_{m-1}) \quad = \quad 1.$$

Thus, every input assignment in $\mathcal{A}$ produces output 1.

For any two distinct input assignments $l \cdot r$ and $l' \cdot r'$ in $\mathcal{A}$, consider the relation between input assignments $l' \cdot r$ and $l \cdot r'$. As illustrated in Table 1, for each i, considering each of

Location		$l{\cdot}r$		$l'{\cdot}r'$		$l'{\cdot}r$		$l{\cdot}r'$	
u_i	v_i	u_i	v_i	u_i	v_i	u_i	v_i	u_i	v_i
L	L	1	0	1	0	1	0	1	0
R	R	1	0	1	0	1	0	1	0
L	R	a	$\neg a$	b	$\neg b$	b	$\neg a$	a	$\neg b$
R	L	a	$\neg a$	b	$\neg b$	a	$\neg b$	b	$\neg a$
						*	†	†	*

Table 1: **Input values caused by fooling set assignments $l{\cdot}r$ and $l'{\cdot}r'$.** Regardless of the splitting among L and R, complementary values appear in the two columns marked by *, and in the two columns marked by †.

the four possible allocations of inputs u_i and v_i to partitions L and R, it can be seen that $l{\cdot}r'(u_i) = \neg l'{\cdot}r(v_i)$, and $l{\cdot}r'(v_i) = \neg l'{\cdot}r(u_i)$. In particular, the two input assignments can be related as follows:

$$\begin{array}{lllllllll} l'{\cdot}r(u_i) & = & \neg l'{\cdot}r(v_i) & = & l{\cdot}r'(u_i) & = & \neg l{\cdot}r'(v_i), & \text{when } l{\cdot}r(u_i) & = \ l'{\cdot}r'(u_i) \\ l'{\cdot}r(u_i) & = & l'{\cdot}r(v_i) & = & \neg l{\cdot}r'(u_i) & = & \neg l{\cdot}r'(v_i), & \text{when } l{\cdot}r(u_i) & \neq \ l'{\cdot}r'(u_i) \end{array} \tag{4}$$

The first condition in this equation can be seen in the top two rows of Table 1, and in the bottom two rows for $a = b$. The second condition can be seen in the bottom two rows for $a = \neg b$.

To determine the multiplier outputs for input assignments $l'{\cdot}r$ and $l{\cdot}r'$, consider first the case where $l{\cdot}r(u_i) = l'{\cdot}r'(u_i)$ for all $0 \leq i < m-1$, but $l{\cdot}r(u_{m-1}) \neq l'{\cdot}r'(u_{m-1})$. As Equation 4 indicates, both input assignments $l'{\cdot}r$ and $l{\cdot}r'$ produce equal values only at bit position $m-1$. Hence

$$Mult^n_{n-1}(l'{\cdot}r) \ = \ l'{\cdot}r(u_{m-1}) \oplus l'{\cdot}r(v_{m-1}) \ = \ l'{\cdot}r(u_{m-1}) \oplus l'{\cdot}r(u_{m-1}) \ = \ 0$$

and

$$Mult^n_{n-1}(l{\cdot}r') \ = \ l{\cdot}r'(u_{m-1}) \oplus l{\cdot}r'(v_{m-1}) \ = \ \neg l'{\cdot}r(u_{m-1}) \oplus \neg l'{\cdot}r(u_{m-1}) \ = \ 0$$

Thus, the multiplier would produce output 0 for both input assignments $l'{\cdot}r$ and $l{\cdot}r'$.

Next, consider the case where there is some maximum value k less than $m-1$ for which $l{\cdot}r(u_k) = \neg l'{\cdot}r'(u_k)$. Equations 3 and 4 then give:

$$\begin{array}{rcl} Mult^n_{n-1}(l'{\cdot}r) & = & l'{\cdot}r(u_{m-1}) \oplus l'{\cdot}r(v_{m-1}) \oplus l'{\cdot}r(u_k) \\ & = & \neg l{\cdot}r'(v_{m-1}) \oplus \neg l{\cdot}r'(u_{m-1}) \oplus \neg l{\cdot}r'(u_k) \\ & = & \neg[l{\cdot}r'(u_{m-1}) \oplus l{\cdot}r'(v_{m-1}) \oplus l{\cdot}r'(u_k)] \\ & = & \neg Mult^n_{n-1}(l{\cdot}r') \end{array}$$

Thus, in either of the two possible cases, the multiplier would produce output 0 for one of the input assignments $l'{\cdot}r$ and $l{\cdot}r'$.

We have therefore shown that the set $\mathcal{A}$ is a VLSI fooling set for the partition (L, R).

□

As a straightforward extension of Theorem 4, we can obtain lower bounds on the other outputs of a multiplier by observing that an n-bit multiplier can perform i-bit multiplication for any $i \leq n$

Corollary 1 *For the functions $Mult^n_{i-1}$ and $Mult^n_{2n-i-1}$, where $1 \leq i \leq n$, any VLSI implementation has area-time complexity $AT^2 = \Omega(i^2)$, and any ordered OBDD representation has $\Omega(1.09^i)$ vertices.*

Proof: Consider an n-bit multiplier with input words A and B. If two i-bit words A' and B' are embedded in the low order i bits of A and B, while the remaining inputs are set to 0, then multiplier output $i-1$ will equal bit $i-1$ in the product of A' and B'. Similarly, if these words are embedded in the high order i bits of A and B, while the remaining inputs are set to 0, then multiplier output $2n-i-1$ will equal bit $i-1$ in the product of A' and B'. Thus, a VLSI fooling set for the function $Mult^i_{i-1}$ can be transformed into VLSI fooling sets for the functions $Mult^n_{i-1}$ and $Mult^n_{2n-i-1}$.

□

6. Discussion

It is interesting to see how a lower bound proof technique devised for one model of Boolean functions can be adapted for a seemingly much different model. In one case a proof involves bounding the number of different bit patterns that must be appear on a set of wires, while in the other case it involves bounding the number of distinct vertices in a graph. These aspects can be abstracted away such that the proofs differ only because of the power of two-way versus one-way communication. For example, suppose we were to hypothesize a one-way VLSI model, in which all signals travel either rightward or downward along the wires. Then a single proof would show that a function has quadratic area-time complexity in this model as well as exponential complexity in the OBDD model.

Finding functions that do not have efficient representations as OBDD's lends some insight into the weaknesses of this model. For example, in a circuit where some inputs act as "control" and others act as "data", it is generally wise to pick a variable ordering that places the control input variables before the data input variables. The hidden weighted bit function is an example of one where the same inputs serve as both control (in determining the weight) and data (in determining the final output value). This difficulty also arises in circuits containing shifters where the shift amount depends on the data, such as in a floating point alignment unit. Multiplication is a particularly difficult function, because it can express many different shifting and adding combinations.

7. Acknowledgements

Don Coppersmith of IBM provided a key insight leading to the lower bound proof for integer multiplication. Discussions with Carl-Johann Seger were also helpful in refining and extending this work.

References

References

[1] H. Abelson, and P. Andreae, "Information Transfer and Area-Time Trade-offs for VLSI Multiplication", *Comm. ACM*, Vol. 23, No. 1 (Jan. 1980), pp. 20–23.

[2] S. B. Akers, "Binary Decision Diagrams", *IEEE Transactions on Computers*, Vol. C-27, No. 6 (August, 1978), pp. 509–516.

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

[4] R. E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation", *IEEE Transactions on Computers*, Vol. C-35, No. 8 (August, 1986), pp. 677–691.

[5] M. Fujita, H. Fujisawa, and N. Kawato, "Evaluations and Improvements of a Boolean Comparison Program Based on Binary Decision Diagrams", *International Conference on Computer-Aided Design*, 1988. pp. 2–5.

[6] D. E. Knuth, *The Art of Computer Programming, Vol. 1*, 2nd Edition, Addison-Wesley, 1973.

[7] R. J. Lipton, and R. Sedgewick, "Lower Bounds for VLSI", *Proc. Thirteenth Annual ACM Symposium on the Theory of Computing*, 1981, pp. 300–307.

[8] S. Malik, A. Wang, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Logic Verification Using Binary Decision Diagrams in a Logic Synthesis Environment", *International Conference on Computer-Aided Design*, 1988. pp. 6–9.

[9] J. E. Savage, "Planar Circuit Complexity and the Performance of VLSI Algorithms", in *VLSI Systems and Computation*, Kung, Sproull, and Steele, *Eds.*, Computer Science Press, 1981, pp. 61–68.

[10] C. E. Shannon, "The Synthesis of Two-Terminal Switching Circuits", *Bell System Technical Journal*, Vol. 28 (January, 1949), pp. 59–98.

[11] C. D. Thompson, "Area-Time Complexity for VLSI", *Proc. Eleventh Annual ACM Symposium on the Theory of Computing*, 1979, pp. 81–88.

[12] J. D. Ullman, *Computational Aspects of VLSI*, Computer Science Press, 1983.

[13] I. Wegener, "On the Complexity of Branching Programs and Decision Trees for Clique Functions", *J. ACM*, Vol. 35, No. 2 (April, 1988), pp. 461–471.

[14] A. C. Yao, "Some Complexity Questions Related to Distributive Computing", *Proc. Eleventh Annual ACM Symposium on the Theory of Computing*, 1979, pp. 209–213.