# VLSI Logic Test, Validation and Verification
# Lecture 7
# Properties & Applications of Binary Decision Diagrams

Instructor: Priyank Kalla
Department of Electrical and Computer Engineering
University of Utah, Salt Lake City, UT 84112
Email: kalla@ece.utah.edu

In this lecture, we will first analyze the effect of variable ordering on BDDs, and then look at some particular applications of BDDs to problems in the VLSI-CAD domain.

## I. Effect of Variable Ordering on BDDs

Just as a truth-table is a canonical form, subject to the constraint that the variables have a total ordering (also called linear ordering) imposed on them, a Binary Decision Diagram (BDD) is also canonical form w.r.t. a variable order. The characteristics of Reduced Ordered BDDs, also called ROBDDs, depends critically on the variable order. This is not the case for a truth table, since a truth-table is an explicit representation of each and every point in the domain of the function. The reduction operation on BDDs, on the other hand, performs some type of *implicit minimization* of the function. Hence, ROBDDs represent the minterms of the function in some simplified form as cubes. From now on, we will refer to ROBDDs as simply BDDs without diluting their meaning. The variable order defines how and which minterms are simplified as cubes. Since each path from a root node to a terminal vertex in a BDD corresponds to a cube of the function, reduction in the number of paths would imply reduction in the number of cubes. How do we bring about a reduction in the number of paths in a BDD? By reducing the number of vertices in a BDD, of course. And how do we reduce the number of vertices in a BDD? We can do that by computing an *optimal* order on the variables of the BDD. See the example below that shows the effect of variable ordering on the size of the BDDs.

### A. BDD Reduction and Logic Minimization

A word of caution - Finding an optimum order on the variables of a BDD so that the size of the ROBDD is an absolute minimum DOES NOT MEAN that we have performed exact logic minimization of the function (Karnaugh-map type). This is actually a rather tricky issue. Going from a non-reduced, full-blown BDD, to a reduced ordered BDD (ROBDD) does entail some logic simplification. However, the simplification is not complete in the sense that the function is not exactly minimized. The reduction operation on the BDD does provide some logic minimization but the extent of minimization really depends upon the character of the function.

### B. Finding an Optimal Variable Order

Given an ROBDD, how do we recompute an ordering of its variables so that the size of the BDD is minimized? Welcome to yet another intractable problem! Since it is impossible to precompute the effect of any variable order on BDD size, unless we try all possible orderings, it is not possible to know which ordering is the best. However, there are some observations that help in computing a good variable order:

- The variables upon which a large number of minterms depend, should be closer to the root. Why? This should be intuitive.
- Variables that feed deep cones of logic should come first in the order, because they are important decision makers.
- It has been shown that for multipliers, it doesn't matter what the order on the variables is, there will be at least one particular output of the multiplier that will have an exponential size of the BDD. Note that exponential size of the ROBDD means that in the original, full blown, non-reduced BDD, there was no possibility of reduction. For multipliers, would there be any benefit of even trying to find a BDD order?
- When we use the APPLY (ITE) operator to compute a new BDD, we might require a new order on the variables to minimize the size of all the BDDs in the manager.
- Suppose we are given two BDDs $f, g$, and we wish to compute some arbitrary Boolean operation $f \odot g$. It might so happen that even though the final result ROBDD for $f \odot g$ is small in size (i.e. does not have exponential complexity), some intermediate result of the computation may be prohibitively large. In such a case, even if we could store the final result in computer memory, the intermediate computation cannot be stored. This renders the entire computational process a fiasco. In practise, this is usually where BDDs tend to show their bad behaviour.

For the above reasons, *dynamic variable ordering* is a technique that has been devised to recompute an order on the variables every time a new ITE/APPLY computation is performed - hence dynamic. Reference [1] describes a dynamic variable ordering algorithm that uses sifting of nodes to compute a good order. Notice that this is a trial and error technique: it swaps nodes in adjacent levels and recomputes the size of the BDD. If swapping leads to fewer nodes, it accepts the order and continues. Even though such a process is intractable in the worst-case, the technique in [1] makes use of intelligent heuristics to compute a good variable order.

## II. Applications of BDDs

In this section let us look at some applications of BDDs. First, we will look at BDDs as an *implicit set representation*, then exploit their canonicity property to prove *equivalence* of Boolean functions, then show how to solve the

*SAT and Tautology* problems. Later on, during the progress of this course, we will analyze some other problems that may be solved using BDDs. Perhaps, some of you might wish to work on a project that would encompass solving some decision problem using BDDs.

### *A. BDDs as an Implicit Set Representation*

Believe it or not, you already have solved this problem - it is that of Boolean function representation. Consider the Boolean function as a set of minterms. By using BDDs as an implicit set representation, you are indeed representing the minterms or cubes of the Boolean function implicitly. Why implicitly and not explicitly? An explicit representation would, as the word suggest, explicitly represent each and every point in the Boolean space. Because the reduction operation on BDDs performs some implicit minimization, ROBDD representation of a Boolean function represents cubes (simplified minterms) of the function (implicitly).

Another interesting application of BDDs as an implicit set representation is the use of *characteristic functions*. Given a set $S$, the elements of $S$ can be represented by a characteristic function of $S$, $\chi_S(s)$, which satisfies the property: $s \in S \iff \chi_S(s) = \mathbf{1}$. In simple English, the formula means that if an element $s$ belongs to $S$ ($s \in S$), then $s$ evaluates the characteristic function $\chi_S(s)$ to **1** and *vice-versa*. This seemingly obvious result can come in very handy when using set operations using BDDs. For example, let $r_1, r_2$ be two flip-flops of a sequential circuit. The total number of possible states of the circuit is $2^2 = 4$. Assume that the underlying state machine of the circuit only has 3 states, out of which, say $r_1 = 0, r_2 = 0$ be the illegal state. So the characteristic function of the legal (also called reachable) states has the following elements: $r_1 = 0, r_2 = 1; r_1 = 1, r_2 = 0$; and $r_1 = 1, r_2 = 1$. In other words, $\chi_S(r_1, r_2) = \overline{r_1} \cdot r_2 + r_1 \cdot \overline{r_2} + r_1 \cdot r_2$ is the characteristic function of the set of legal states. What would the BDD for this characteristic function look like? Draw it and see if the representation is explicit or implicit?

### *B. Equivalence Verification*

This is simple! In the last lecture we have already seen how we can prove or disprove the equivalence of two combinational circuits using BDDs I think this need no further mention.

### C. SAT using BDDs

Let us revisit the circuit satisfiability problem that was introduced in the previous chapters. The circuit is reproduced in Fig. 1. Let us suppose that it is required to compute input assignments to the circuit in order to satisfy the output requirements: $\{u = 1, v = 1, w = 1\}$. Computing the conjunction of the individual constraints $(u, v, w)$ provides all satisfying assignments. Let $f = u \cdot v \cdot w$ represent the solutions, then $f = ac + bc + a\overline{b}$.

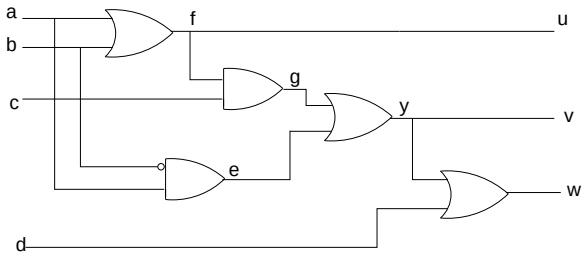


Fig. 1. An example circuit.

Each of the individual constraints, in the above case $u, v, w$, can be represented using an ROBDD. The conjunction of all the constraints can be performed by computing the Boolean product of $u, v, w$. The resulting ROBDD would contain all satisfying solutions. The ROBDD for $f = u \cdot v \cdot w = ac + bc + a\overline{b}$ is shown in Fig. 2. Selecting any of the cubes as a SAT solution amounts to traversing any path from the root node to the terminal vertex **1**. For example, as shown in the Fig. 2, traversing the path $\{a = 1, b = 1, c = 1\}$ from the root leads to the terminal vertex **1**; thus the cube $a \cdot b \cdot c$ is a SAT solution to the problem.

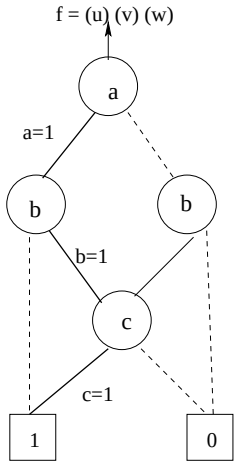


Fig. 2. Satisfi ability using path traversal on a BDD.

Since an ROBDD is a DAG, the shortest path problem can be solved using the root node as the *source* and the terminal **1** as the *sink*. Note that traversing the shortest path amounts to exercising the *largest cube* in the BDD [2] [3]. Note that in the above ROBDD, $a \cdot \overline{b}$ is the shortest path (and hence the largest cube) from the root to terminal **1**.

#### C.1 Tautology using BDDs

The dictionary meaning of tautology which I gathered from www.dictionary.com, is:

> *An empty or vacuous statement composed of simpler statements in a fashion that makes it logically true whether the simpler statements are factually true or false; for example, the statement "Either it will rain tomorrow or it will not rain tomorrow" is a tautology.*

In other words, if a Boolean function is *always* TRUE, then the function is tautology; $e.g., f = a + a'$ is an obvious tautology. What would a BDD for a tautology look like? Draw the BDD for $f = a + a'$ and see for yourself.

All the elements of the set, i.e. all paths of the BDD should go to terminal vertex **1**. A reduction operation would eleminate ALL nodes from the BDD except the terminal node one. (Do you know why do all the BDDs representing tautology have only the terminal vertex **1**? Because of canonicity!) Tautology checking is much like the SAT problem. The only difference is that: SAT problem checks for the *existence* of the truth assignment, whereas the tautology problem checks for the *universality* of the truth assignment.

The non-tautology problem questions the existence of an assignment that makes the expression FALSE. How would you solve the non-tautology problem on a BDD?

Can you think of any other decision/optimization/representation problems that you can solve using BDDs?

## REFERENCES

[1] K. S. Brace, R. Rudell, and R. E. Bryant, "Effi cient Implementation of the BDD Package", *in DAC*, pp. 40–45, 1990.

[2] B. Lin and F. Somenzi, "Minimization of Symbolic Relations", *in ICCAD*, 90.

[3] S. Jeong and F. Somenzi, "A New Algorithm for the Binate Covering Problem and its Application to the Minimization of Boolean Relations", *in ICCAD*, 92.