Design Verification

Lecture 06 - Multi-Level Logic Verification: BDD

1. Technique #6: Binary Decision Diagrams (BDD’s)
   - represent function with BDD’s, proposed in 1956 (Lee), popularized in 1978 (Akers)
   - BDD’s are directed acyclic graphs (DAG’s)
   - OBDD’s (Ordered BDD’s) are canonical → their form of representation is unique [Bryant 1985]

2. BDD terminology
   - Make-up of BDD
     - Each vertex is either terminal or non-terminal
     - non-terminal: variable with 2 successors, low() and high()
     - terminal: no successor nodes, and has value of 1 or 0
   - BDD Categories
     - complete BDD: each variable is visited exactly once from root to any terminal
     - free BDD: each variable is visited at most once from root to any terminal
     - ordered BDD: a free BDD where variables are visited in the same order from root to any terminal via any path
   - Evaluation of BDD: given an $n$-input assignment, traverse the BDD in $O(n)$

Example 1
Example 2
3. Verification using BDD’s: Verify equivalence of the BDD’s
   - can’t simulate one BDD on another
   - how to store and manipulate BDD’s crucial
   - comparing Free BDD’s (2 BDD’s of different variable order)
     - NP-complete
     - Need a specific variable ordering - OBDDs are canonical.
     - Build a BDD that represents the miter circuit
       Given a miter BDD, Tautology check can be done in constant time for any terminal 0’s. Tautology means NO terminal 0’s.

4. How to build BDD’s: recursive use of co-factoring (Boole’s Expansion Theorem)
   Example 3

5. If circuit is given as a netlist:
   - build BDD by constructing a node at each gate-output
     starting with primary inputs and compute functions at each gate
   - build BDD by iteratively computing for each gate via ITE’s
   Example 4
6. Boolean synthesis from two ROBDD’s

- in the worst case, synthesizing an ROBDD from two given ROBDD’s may require exponential time
- Rules for synthesizing two ROBDD’s

Example 5
7. Cofactoring using ROBDD’s

\( f_{x_i} \rightarrow \) traverse ROBDD of \( f \) top-down, starting at the root, and eliminate all nodes which are marked with \( x_i \) and adjust the edges accordingly.

Example 6

8. Composing 2 ROBDD’s

Compose \((f_1, v, f_2)\) means to substitute \( f_2 \) for variable \( v \) in \( f_1 \).

\[ f_1|_{v=f_2} = (f_2 \land f_1|_v) \lor ((\neg f_2) \land f_1|_\bar{v}) \]
9. Backward construction from a netlist: repeated composition of BDDs

Example 7

10. Reducing BDD’s:

- Step 1: identification of isomorphic subgraphs
- Step 2: removal of redundant nodes

Example 8
11. BDD Reduction Algorithm:

Given an OBDD:

- Set ROBDD = 0; levelize OBDD; set id(v) = 1 for terminal 0’s and id(v) = 2 for terminal 1’s
- Gradually add nodes to ROBDD in a bottom-up fashion
- Define: key(v) = (a, b), where a = True_child(v), b = False_child(v)

Example 9
12. Complexity of reduction: $O(n \times lg(n))$, where $n = \#$ vertices (but we could have exponential $\#$ vertices)

13. Complement edges
   - allow BDD for $f$ and $\bar{f}$ to be shared
   - benefit 1: save space
   - benefit 2: complementation is now constant time

14. Wouldn’t it be nice if the first BDD built is *already* reduced?
   - Build BDD from netlist
   - ITE() function (ITE = If-Then-Else)

15. $ITE(f, g, h) = fg + \bar{f}h$; $f$, $g$, and $h$ all have the same variable ordering
    let $z = ite(f, g, h)$
    $= fg + \bar{f}h$
    $= x(fg + \bar{f}h)_x + \bar{x}(fg + \bar{f}h)_{\bar{x}}$
    $= x(f_xg_x + \bar{f}_xh_x) + \bar{x}(f_xg_{\bar{x}} + \bar{f}_xh_{\bar{x}})$
    $= xG + \bar{x}H$
    $= ITE(x, ITE(f_x, g_x, h_x), ITE(f_x, g_x, h_x))$
16. ITE can be used to compute many boolean expressions!

\[
\begin{align*}
    f \cdot g &= \text{ite}(f, g, 0) & \quad f + g &= \text{ite}(f, 1, g) \\
    f \cdot \bar{g} &= \text{ite}(f, \bar{g}, 0) & \quad f + \bar{g} &= \text{ite}(f, 1, \bar{g}) \\
    \bar{f} \cdot g &= \text{ite}(f, 0, g) & \quad \bar{f} + g &= \text{ite}(f, g, 1) \\
    f \oplus g &= \text{ite}(f, \bar{g}, g) & \quad \bar{f} \oplus g &= ??? \\
    \bar{f} &= \text{ite}(f, 0, 1) & \quad ... \\
    \text{ite}(1, f, g) &= \text{ite}(0, g, f) = \text{ite}(f, 1, 0) = \text{ite}(g, f, f)
\end{align*}
\]

17. ITE’s are excellent for constructing ROBDD’s!

Example 10
Example 11

18. Data structures for ITEs

- Unique Table - contains a key for each vertex of an ROBDD
  → does there exist a node with vertex $v$ and children $g$ and $h$?
  → UT is indexed with $f = \text{ite}(v, g, h)$

- Computed Table - contains recently computed functions (ROBDDs)
  → did we recently compute $f_1$ AND $f_2$ (eg. $\text{ite}(f_1, f_2, 0)$)?
  → CT is indexed with $f_1, f_2, 0$, if it exists, it returns a pointer to UT

- both UT and CT are implemented as hash tables
- each node of the BDD has an entry in the UT
CT is not absolutely necessary, but helpful in avoiding recomputing previously computed BDD nodes

19. ITE algorithm

ITE(f, g, h) /* all functions denoted by their pointer */
if (terminalCase(f, g, h))
    return corresponding terminal result;
found = CT.entry(f, g, h);
if (found) /* found in CT */
    return (corresponding link/entry in UT);
v = top variable of (f, g, h);
T = ITE(f_v, g_v, h_v);
E = ITE(f_v, g_v, h_v);
if (T == E)
    return (T);
R = findOrAdd_UT(v, T, E);
insertCT(f, g, h, R);
return(R);

terminalCase(f, g, h)
if (f == 1) then return(g);
else if (f == 0) then return(h);
else if (g == h) then return(g);
else if ((g == 1) AND (h == 0)) then return(f);

20. Verification:
First, build two BDDs (eg. using ITE) for circuits F and G
F \Rightarrow G \Leftrightarrow F + G = \text{ite}(F, G, 1)
G \Rightarrow F \Leftrightarrow \bar{G} + F = \text{ite}(G, F, 1)
So, if F \equiv G, then F \Rightarrow G, and G \Rightarrow F.
Given a BDD, Tautology check can be done in constant time for any terminal 0's. Tautology means NO terminal 0's.