Design Verification

Lecture 07 - Multi-Level Logic Verification: BDD 2

1. Exact Minimization Algorithm:
   - construct BDD for all $n!$ variable orders
   - worst case: $O(2^n)$ for a given BDD
   - exact algorithm complexity: $O(2^n \times n!)$

2. Non-exact minimization method - BDD variable ordering
   - (1) similar to exact: exhaust only $m$ variable order permutations, where $m < n$
   - (2) static technique a - partition inputs into the following:
     - input lines with fanout
     - input lines without fanout
     - alg: inputs with fanout are added to the ordering directly, with those without fanout deferred. may use technique (b) to rank each group
   - (3) static technique b - based on depth-first search from PO’s, and give higher weight to variables that are used more than once. (var’s that fanout to multiple POs, etc.)
     $\longrightarrow$ implications on fanout-free circuits:
   - (4) static technique c - primary output labeled with weight of 1. Inputs of a gate are labeled with weight = weight(output) / # fanin. Weight of fanout stem = sum of weights of fanout branches. Order the variables by their weights, starting with the max weight.

Example 1
• (5) Dynamic technique - based on locally swapping variables
  Rules for exchanging adjacent variables

Two dynamic techniques exist: Window permutation and Sifting
• Window Permutation Example
• Sifting Example
  – Not restricted to only a few variables in window
  – Sifting all variables may lead to optimal solution

3. Other dynamic variable ordering techniques

• exploit symmetry
  → regular sifting unable to recognize that variables of a symmetric group
  have a strong attraction to each other and should be sifted together
  → a variable of a symmetric group sifted by regular sifting likely to return
  to its initial position due to attraction of other variables

• evolutionary algorithms
  → evolve a good variable order

4. Variable symmetry

• definition: a function $f$ is symmetric in a pair of variables $(x_i, x_j)$ iff
  $f(a_1, ..., a_i, ..., a_j, ..., a_n) = f(a_1, ..., a_j, ..., a_i, ..., a_n)$ for any value combina-
  tions of $a_1...a_n$. In other words, swapping of variables $a_i$ and $a_j$ do not
  change the function.

• definition: restrictions of $f$: $F_{x_i} = \{ f_{x_1=a_1,\ldots,x_{i-1}=a_{i-1}}, \forall a_1\ldots a_{i-1} \in \{0,1\} \}$. $F_{x_i}$ denotes the set of BDD nodes with variable $x_i$ with respect to all variables
  that precede $x_i$. Note $F_{x_1} = f$ (if $x_1$ is the first variable)
Lemma: $f$ is symmetric w.r.t. $\{x_i, x_j\}$ iff each function $g \in F_{x_i}$ is symmetric w.r.t $\{x_i, x_j\}$.

Thus, we simply need to check symmetry in the set of $g$'s.

Theorem 1: If $f$ is symmetric w.r.t $\{x_i, x_j\}$, then each $g \in F_{x_i}$ depends on both $x_i$ and $x_j$ or depends neither on $x_i$ nor $x_j$.

Theorem 2: If $f$ is symmetric w.r.t $\{x_i, x_j\}$, then for all $g \in F_{x_i}$, we have $g_{x_i x_j} \equiv g_{\overline{x_i} \overline{x_j}}$.

This means that if $\exists$ one function $g \in F_{x_i}$ that is not symmetric w.r.t $\{x_i, x_j\}$, then $f$ must not be symmetric in $\{x_i, x_j\}$.

Observation: exchange of symmetric variables does not change the size of BDD.

Observation: size of a totally symmetric function is $O(n^2)$

Example 2: symmetric function

5. Detection of symmetries - naive method

- for each pair $(x_i, x_j)$
- $\rightarrow$ compute BDDs for $f_{x_i x_j}$ and $f_{\overline{x_i} \overline{x_j}}$
- $\rightarrow$ check equivalence of these two BDDs
- Problem: creation of these BDDs expensive

6. Detection of symmetries - alternative method

- Suppose $x_i$ is higher in order than $x_j$
- Observation: $f$ is asymmetric in $\{x_i, x_j\}$ if any node in the BDD with variable $x_i$ does not have any successor with label $x_j$. Similarly, $f$ is asymmetric in $\{x_i, x_j\}$ if any path from root of the BDD to variable $x_j$ does not contain any node with label $x_i$. 
• Algorithm: for each node of the BDD, compute the set of predecessor variables. This can be computed by depth-first-search. Then, for all nodes with $x_j$, we can check if $x_i$ is in each predecessor set.

• Goal: instead of finding symmetric pairs, we find those that CANNOT be symmetric

7. Symmetries in incompletely-specified functions

Example 3

8. Evolutionary algorithms

• population of individuals
• each individual represents an ordering
• fitness of individual measures the quality of the var ordering
  $\rightarrow$ may need to build partial/full BDD - abort early when bad order is chosen
• selection, cross-over and mutation to generate successive generation
Example 4: initialization of population

Example 5: selection

Example 6: cross-over

Example 7: mutation
9. Better Evolutionary alg

- initialize population with some statically computed var orders
- one extra genetic operator: invert. Similar to mutation, but inverts some var order within a range

10. ATPG-based BDD variable ordering

- call PODEM to backtrace to find the variable order

```c
cover() {
    if PO == 1 or 0 then return;
    (PI, v) = backtrace(PO, 1);
    logicSim(PI, v);
    cover(); /* recursive call */
    logicSim(PI, \bar{v}); /* traverse other subtree */
    cover();
    logicSim(PI, X);
}
```

backtrace() is the same used in ATPG

**Example 8**
11. Extensions of BDD

- Algebraic Decision Diagrams (ADD) \( f = (1 - x)f_x + xf_x \)

- Binary Moment Diagrams (BMD)

- Multiplicative Binary Moment Diagrams (*BMD)

- useful for verifying arithmetic circuits