Design Verification

Lecture 09 - Incremental Multi-Level Logic Verification

1. Motivation
   - directly checking for miter output equivalence can be time consuming
   - complexity can be reduced by exploring structural similarity between the spec and impl circuits
   - first prove equivalence of internal nodes, then use this knowledge to help prove equivalence of miter output
2. Definitions
   - *signal pair*: (l_1, l_2) is a signal pair if l_1 and l_2 are signals in different circuits (e.g., l_1 from C_1 and l_2 from C_2)
   - *equivalent pair*: (l_1, l_2) is an equivalent (signal) pair if responses of l_1 and l_2 to any input vector are identical
   - *permissible pair*: (l_1, l_2) is a permissible (signal) pair if l_1 is a permissible function of l_2. In other words, replacing signal l_1 by l_2 in the miter does not change the miter output function
   - NOTE: equivalent pair $\rightarrow$ permissible pair, but not vice versa
   - Two circuits are *similar* if large percentage of signals belong to equivalent or permissible pairs
3. General incremental verification algorithm
   - step 1: identify candidate permissible pairs
   - step 2: incrementally prune the miter circuit by checking if candidate permissible pairs are truly permissible, and replace signals that are permissible
   - step 3: check equivalence of miter output
4. Step 1: Identify candidate permissible pairs
   - simulate k random vectors and form pairs that respond the same way
   - pair up signals of same signal name
   - designer-specified or provided

- worst case: $O(n_1 \times n_2)$ pairs, where n_1 = number of signals in C_1, and so on. In reality, much smaller

5. Step 2: Incremental pruning of miter

   - most important step of the incremental verification process
   - can check for equivalence or permissibility of candidate pairs
   - iterate this step until no more candidate pairs to be checked

   Example 1

   Example 2

6. Permissible pair may not be equivalent pair!

7. Inverse permissible pair: l_1 may be replaced by $\overline{l_2}$

8. Step 3: Check equivalence of miter output
   - check equivalence of output by SAT, ATPG, BDD, etc., as the circuit size is now reduced, and hopefully checking for equivalence is also simplified

9. Incremental BDD based verification
   - benefit: if 2 circuits do not have much similarity
   - key: build local BDDs based on proven equivalent signals, which are internal signals (not necessarily primary inputs)
   - define *cutset*: a set of values, λ, in input cones of a signal a such that a can be represented as a function of λ
     $\longrightarrow$ a cutset is also called a *support*
   - idea: a signal pair (l_1, l_2) is an equivalent pair if no value combinations on the cutset of l_1 and l_2 can distinguish the pair. We want to formulate the cutset to include the equivalent pairs found so far

Example 3

10. False negative problem

    - false negative: declare (l_1, l_2) to be inequivalent falsely
    - this happens when some signals in the cutset are correlated
      $\rightarrow$ some value combinations on the cut set may be impossible

    Example 4

11. Overcome the false negative problem

    - idea: use dynamic cutset/support
    - extend cutset back towards primary inputs until candidate pair proven equivalent

    Example 5

12. Incremental transformation-based verification

    - instead of incrementally reducing the circuit, augment/enlarge it to enhance similarity between 2 circuits under verification
    - idea: step 1: identify dissimilar regions, step 2: add logic to reduce the dissimilarity, and step3: check equiv. of miter output after transformation
    - dissimilar regions identified based on indirect implications

13. Identifying dissimilar regions

    - compute implications for the miter circuit
    - dissimilar regions contain the nodes that do not imply anything (thus, no other nodes imply them either by contra-positive)

14. Increase similarity among dissimilar regions

    - Take a pair of nodes (l_1, l_2) where l_1 is in C_1 and l_2 is in C_2, and that both belong to dissimilar regions
    - add a logic $\delta(l_1, l_2)$ such that the function of miter network remains the same
    - for example, if $\delta()$ = OR gate, under what circumstances may I insert the OR gate into the circuit without changing the miter function?
    - Furthermore, the addition of this OR gate increases the number of implications possible in the miter circuit (i.e., increase similarity)

Example 6

Example 6 continued

15. Boolean difference

Example 7

Example 8

Example 9

16. Transformation can involve other gate types as well

17. Use ATPG to identify which transformation to use
    - ATPG can use the help of implications to reduce search space