Design Verification

Lecture 13 - More Sequential Logic Verification

1. State traversal a key to verification:

   - Can we use partitioned-ROBDD (not partitioned FSM)?
     $\longmapsto$ potentially eliminate memory explosion problem
   - Recall POBDDs have the following characteristics:
     - divide boolean space into k partitions
     - a separate BDD for each paritition
     - sum of all partitioned BDDs exponentially smaller than monolithic BDD
     - partitioned BDDs also canonical
   - formal definition of POBDD:
     - Given a function f, POBDD $= \{(w_1, f^1), ..., (w_k, f^k)\}$, over k partitions
     - where w_i is a Boolean function (called "window function") representing the partition and
     - $w_1 + w_2 + ... + w_k = 1$ (i.e., the partitions cover the entire Boolean space)
     - $f^i = w_i f$
     - both w_i and f^i can be represented as ROBDDs, with variable ordering π_i
     - we can also restrict that all partitions are orthogonal: $w_i \wedge w_j = 0, \forall i \neq j$

2. Apply POBDD to FSM traversal

   Conventional_Image_based_traversal$(I(ps), T(ps, ns, i))$
   $R(ps) = F(ps) = I(ps)$; /* R, F, I all BDDs, I = initial set of states */
   while $(F(ps) \neq 0)$ /* iterate if newly found states not empty */
   $N(ns) =$ apply $T(ps, ns, i)$ on $F(ps)$;
   $F(ps) = N(ns \leftarrow ps) \wedge \overline{R(ps)}$; /* F = the newly found states */
   $R(ps) = R(ps) + F(ps)$;

   - now, R, F, and I represented as POBDDs, R_i for partition i, etc.
   - need to compute transition for each partition using the complete T
   - Problem: resulting BDDs not disjoint, i.e., they cross partition boundaries

- Solution: partition T as well with respect to ps and ns: $w_i(ps)$ and $w_i(ns)$:
  $R_j = \exists_x \exists_{PS}\ w_j(ps) w_j(ns) R(ps) \cdot T(ps, x, ns)$
  $= w_j(ns) \exists_x \exists_{PS}\ w_j(ps) R(ps) \cdot T(ps, x, ns)$
- N_j guaranteed to lie within partition j
- Need to compute Img() for all $i \times j$ possible partitions from R_i to N_j

3. Non-traversal-based approaches:
   - Can use ATPG for the miter (product machine) output fault
   - Problem: sequential ATPG needed, and the target fault may be hard
   - Enhancements by finding structural similarities as in combinational circuits
4. Structural similarity in sequential circuits
   - Recall in combinational circuits, two nodes a and b, if found to be an equivalent or permissible pair, one may substitute another explicitly
   - In sequential circuits, we can't replace a by b or vice versa because (a, b) may be any of the following five combinations: $(1,1)$, $(0,0)$, (x,x), $(x,1)$, $(x,0)$. If we replace one for the other, $(x,1)$ and $(x,0)$ now can never occur, and one signal will be *over-specified*
   - Thus, we must check for them *implicitly*, that a and b must not have conflicting values (0,1) or (1,0) in any of the preceeding time frames - a constrained search

Example 1

5. Identifying equivalent flip-flop pairs

   - Similar to finding equivalent internal node pairs, but can play some tricks here
   - Naive approach: construct a miter for each candidate flip-flop pair, and test for stuck fault
     $\longmapsto$ Problem: fault may be hard to detect
   - Better method (assume-then-verify): Assume all candidate flip-flop pairs to be equivalent in all time frames, and test for the target fault
   - If a pair of FFs is found to be non-equivalent (XOR output stuck-at-0 detected without violating any constraints), then this pair is guaranteed to be non-equivalent

   Example 2

6. Equivalent Flip-Flop identification algorithm

   random simulation to identify candidate FF pairs
   while ((not done) and (iteration less than K))
   assume all candidate pairs to be equivalent in *previous* time-frames
   for each candidate pair (FF_i, FF_j)
   connect them via an XOR gate
   test for XOR output stuck-at-0 without violating any
   if fault detectable, (FF_i, FF_j) definitely not equivalent
   if no pair found to be inequivalent in this iteration, then done=true
   else iteration++;

7. After equivalent FFs identified...
   - target the PO miter fault with the reduced circuit for reset equivalence
   - can't merge circuit if targeting 3-value equivalence

   Example 3 (Reduced Circuit)

8. Equivalent internal nodes can help identify equivalent FF pairs and also help identify equivalence on primary outputs
   - Once equivalent FF's are identified, the circuit is simpler
   - Can start with the combinational portion of the circuit: assumption checker circuit, with the candidate internal nodes tied to an XOR gate
   - For all possible starting states, S, that can detect XOR output stuck-at-0, compute the pre-image of S, until fixed-point. If reset state is included, then candidate internal pair not equivalent
     $\longmapsto$ Use symbolic backtward traversal to help, if $|S|$ is large.

   Example 4 (Assumption Check Circuit)

Example 5

9. Incremental verification in assumption checker
    - similar to combinational circuits, the equivalent internal nodes can form a cut-set, with which future internal pairs can be checked against
    - watch out of false-negatives, in which case must back up the cut-set to PIs and FFs
10. Review of Incremental Equivalence Checking Process
    - step 1: build miter
    - step 2: random simulation to identify equiv FFs and internal nodes
    - step 3: identify true equivalent FF pairs
    - step 4: use symbolic equiv checking to identify equivalent internal node pairs
    - step 5: on the merged and reduced circuit, check equivalence on PO miter