VLSI / SOC Testing

Lecture 14

1. Sequential Circuit Test Set Compaction
   - Motivation: reduce test application time and test data volume circuit
   - Problem: unlike combinational test set compaction, removing a vector in $T = \{t_0, t_1, ..., t_n\}$, the sequence is disrupted
2. Terminology: S_k/S_k^f = the fault-free/faulty state at time frame k for fault f
3. Compaction by insertion
   - If $S_k/S_k^f \equiv S_m/S_m^f$, where $m > k$, then what would inserting the subsequence starting from m at k cause?

4. Compaction by omission
   - Associate each vector with a flag omitted

     initialize omitted[i] = 0 $\forall$ i
     for i = 0 to n
         set omitted[i] = 1;
         faultsim vectors in T whose omitted flag = 0;
         if FC the same or higher, vector i is omitted;
         else reset omitted[i] = 0;
   - This loop can be repeated, and with a different order (i.e., n downto 0)

5. Compaction by selection
   - for each detected fault f, compute the subsequence that can detect f
   - then, compute the compact set using covering algorithm (note, subsequences can overlap)

6. The 3 previous techniques computationally expensive

7. Compaction by subsequence removal
   - Motivation: test sets generally traverse through a small set of states, thus many states are repeated
     $\longmapsto$ Can we remove subsequences that start and end on the same state?

8. Inert subsequence removal: an inert subsequence is one where start and end states are the same, and there are no faults detected within the subsequence

   Example 1:

9. Compaction by state-recurrence subsequence removal

   - a state-recurrence subsequence is one that start and end on the same state, but some faults may be detected within
   - Algorithm:

     step 1a: compute state recurrence subsequences for test set T
     step 1b: compute excitation and detection points for each fault
     step 1c: identify all faults, F_{sr} detected within such subsequences
     step 2: for all faults in F_{sr}, perform faultsim without fault-dropping
     step 3: analysis phase: if in a state-rec subsequence, all faults detected within are also detected elsewhere, and no other faults excited within and propagated beyond the subsequence, then this subsequence may be removed

10. Combining inert and state-recurrence subseq removal

    - First remove inert subsequences
    - Next, perform rec. subseq removal

11. Relaxed subseq. removal

    - Motivation: not every state variable needed to reach the next state

12. State relaxation for test set T

Example 2:

13. Reducing cost of compaction (can be applied to any compaction algorithm)

    - Fault partitioning: compact w.r.t. only hard-faults, since easy faults may be detected by vectors and sequences that detect hard faults
      $\longmapsto$ only 10% faults need to be targeted

14. Compaction by vector restoration

- step 1: compute detection time for each fault (with fault dropping)
  $\longmapsto$ each fault has one unique detection point
- step 2: for each fault (starting with the last detected fault), restore it
- procedure for restoring a fault involves repetitive fault simulation

  i = detTime for fault f
  $T_f = [t_i]$
  repeat
  if (faultsim(T_f) detects f) $\rightarrow$ done
  else $T_f = [t_{i-1}, T_f]$
  until f detected

- worst case: no vector is omitted (generally this is not the case)

Example 3:

15. Using compaction for helping ATPG

    - Motivation: compaction eliminates unnecessary vectors, thereby leaving only good ones
      $\longmapsto$ inherent information embedded within compacted test set

16. Simple approach: extract weights from compacted test set

17. Observations for the simple approach

    - Most weights between 0.4 and 0.6

18. Approach #2: vector copying and holding

    - compacted vectors are there for a reason. Copy them exploits spatial locality
    - holding vectors exploits temporal locality. Observation tells us that holding useful vectors several clocks helps traverse state space deeper

19. Approach #3: correlation-based

    - instead of computing weights, compute spatial and temporal correlation within the compacted test set
    - generate additional vectors using these correlation

20. Approach #4: Spectrum-based

    - Analyze spectrum of compacted test set
    - Since compaction removes unwanted/unneeded vectors, it *filters* the noise out
    - What frequency components does each PI bit-stream have in the compacted test set?
    - Issue: how to analyze spectrum?