VLSI / SOC Testing

Lecture 19

1. Design for Testability
    - Goal: change circuit structure to achieve
        - easy to generate test vectors (manually or automatic)
        - small test set - shorter test application time and data volume
        - easy to compute fault-free response
        - easier for diagnosis and debug
    - Important: the modified circuit must retain original circuit functionality
    - general aims:
        - increase the controllability/observability of some signals - make them easier to control/observe
        - make justification of states easier
    - penalties:
        - area overhead: extra gates/pins/routing added
        - performance degradation: might slow down the circuit speed

    Example 1: Ad-hoc circuit partitioning

2. Where to partition circuit
   - around existing muxes
   - along sensitized paths (make long paths shorter)
   - along buses (separate around buses)
3. Testability Point Insertion
   - Increase control/observability of nodes in circuit
   - First identify nodes that are hard to control/observe
   - Addition of muxes (extra area)
   - Avoid placing on critical paths

   Example 2:

4. Scan Design for sequential circuits
   - Make FFs fully controllable/observable at a cost
   - scanned FFs connected in a chain
     - becomes a shift-register
     - can now force specific values into FFs
     - can also shift out values to be observed

5. Scan Cell Design
    - LSSD (Level-Sensitive Scan Design)
    - Mux-based scan design
    - Cost: area overhead in each scanned FF
    - Cost: performance overhead
    - Avoid scanning FFs on critical path

    Example 3:

6. Full-Scan Design: scan every FF
    - Converts sequential circuit into a combinational circuit, only combinational ATPG needed
    - Large test set application time and test data volume
        - every vector requires $n+1$ cycles, where $n = \#$ FFs
        - every pattern has $n+m$ bits, where $m = \#$ PIs
        - expected responses (FFs + POs) for each pattern also need to be stored
7. Multiple Scan Chains Design
    - test application time now $\frac{n}{k}$, $k = \#$ chains
    - need more test pins for k chains
    - test data volume the same
8. Partial Scan Design: scan only a subset of FFs
    - scan enough FFs to achieve FC similar to full-scan
    - test application reduced

- test data volume reduced?
- but, circuit still sequential, need sequential ATPG
- validation of vectors more complicated
- Key issue: which subset of FFs to scan?

9. Connecting Scan FFs
    - order of connection critical to area

   Example 4:

10. Partial-Reset
    - add a separate partial-reset pin to a subset of FFs
        - makes state *jump* to a different state
        - helps to re-orient traversal

11. Direct Loading of FFs
    - instead of reseting or shifting in the value into a FF, directly load the desired value → test application time reduced
    - area overhead
    - at-speed testing possible → can capture delay defects

   Example 5:

12. Boundary scan
    - scan PIs and POs on a PC board
13. How to select FFs for scan/load/etc.?
    - testability-based
    - cycle-cutting based (structure-based)
    - ATPG-based
    - hybrid
14. Testability-Based
    - compute SCOAP measures for original circuit
    - select the FF with highest C0/C1/O
    - scan it, its C0=C1=O = 0
    - recompute SCOAP for circuit and repeat
    - stop when all FFs C0/C1/O are below a threshold or a maximum number of FFs have been scanned
    - Problems with this approach: (1) SCOAP only a metric, (2) FF selection a greedy approach, thus not optimal

    Example 6:

15. Structure-Based
    - Construct the S-graph for circuit (nodes = FFs, directed edge = combinational path in one time-frame between the 2 FFs)
    - Cycles within S-graph means that FF values may depend on one another
      $\longmapsto$ hard to control some FF with only PIs
      $\longmapsto$ hard to propagate fault effect from some FF to a PO
    - If break the cycle, sequential depth no longer ∞
      $\longmapsto$ recall testing of acyclic circuits

- Goal: break all cycles (excluding self-loops)
- Algorithm:
  - identify all cycles in S-graph excluding self-loops
  - select min # FFs to scan such that all cycles are broken
- Problems with this approach: self-loops ignored, they can still cause problems in testing

Example 7:

16. ATPG-Based

- Quick run of ATPG
- for all faults that were aborted
  ⟼ examine why (which state was hard to justify, etc.)
- From the set of aborted states
  ⟼ compute the minimal subset of FFs to scan
- Problems with this approach: quick run of ATPG can still be expensive

17. Random Access Scan (RAS)

    - arrange scan flip-flops in a two-dimensional array
    - need additional pins to address the specific scan cell
    - do not need to load every scan FF for each pattern
      $\longmapsto$ only need to load those FFs whose value change from the obtained output of the previous test pattern

18. Optimizations to RAS

    - instead of indexing both row and column of RAS, only address the column. The rows advance progressively. (Progressive RAS)
    - one can also re-arrange test vectors to minimize the number of loads necessary, thus reducing test application time, as well as test power
    - can also use intelligent ATPG to generate vectors that target RAS or PRAS.