VLSI Design Verification and Testing

ATPG Systems and Testability Measures

Mohammad Tehranipoor
Electrical and Computer Engineering
University of Connecticut

Motivation

- ATPG Systems
  - Increase fault coverage
  - Reduce overall effort (CPU time)
  - Fewer test vectors (test application time)
- Testability Measures
  - A powerful heuristic used during test generation (more on its uses in later slides)

ATPG Systems

- **Reduce cost of fault simulation**
  - Fault list reduction
  - Efficient and diverse fault simulation methods suited for specific applications and environment
  - Fault sampling method

ATPG Systems

- **Reduce cost of test generation**
  - Two phase approach to test generation
    - **Phase 1:** low-cost methods initially
      - Many faults can be detected with little effort. For example use random vectors.
      - Initially the coverage rises rather fast.
    - **Phase 2:** use methods that target specific faults till desired fault coverage is reached

ATPG Systems

- **Phase 1 issues:**
  - When to stop phase 1 and switch to phase 2
    - Continue as long as many new faults are detected
    - Do not keep a test that does not detect many new faults
    - When many consecutive vectors have been discarded
    - Etc.

ATPG Systems

- **Phase 2 issues (during deterministic test generation):**
  - Efficient heuristics for backtrace
  - What fault to pick next
  - Choice of backtrack limit
  - Switch heuristics
  - Interleave test generation and fault simulation
  - Fill in x's (test compaction: static and dynamic)
  - Identify untestable faults by other methods

ATPG Systems

- **Efficient heuristics for backtrace**
  - Easy/hard heuristic
    - If many choices to meet an objective, and satisfaction of any one of the choices will satisfy the objective – choose the easiest one first
    - If all conditions must be satisfied to meet the desired objective, choose the hardest one first
  - Easy/hard can be determined
    - Distance from Pis and Pos
    - Testability measures

7

ATPG Systems

- **Which fault to pick next**
  - Target to generate tests for easy faults first
    - Hard faults "may get" detected with no extra effort
  - Target to generate tests for hard faults first
    - Easy faults "will be" detected anyway, why waste time
  - Target faults near PIs
  - Target faults near POs
  - etc.

Anytime a fault is detected, there is a chance that the test vector detects a lot more faults.

Any pattern generated by ATPG contains a large number of X's providing opportunity for detecting more faults.

8

ATPG Systems

- **Choice of backtrack limit (High backtrack ⇒ more time)**
  - It has been observed that as the number of backtrack increases the success rate goes down.Thus we may wish to keep low backtrack limit.
    - Known as "Abort limit" in commercial tools
  - Some faults may not get detected due to lack of time spent on them
    - Untdetected Faults
      - Such definitions are different from one tool to another.
  - Could start with low limit and increase it when necessary or in second round (often used heuristic)

9

ATPG Systems

- **Switch heuristics**
- Interleave test generation and fault simulation
  - Drop detected faults after generation of each test
    - This has higher switching cost but generally works well
    - This strategy may not be usable with certain fault simulators
    - Sequential tests may not have other options and this may be the only practical option in some cases

10

ATPG Systems

- **Fill in x's (test compaction)**
  - Test generator generates vectors with some inputs unspecified
    - Can fill these values with random (0, 1) values (often termed as *dynamic compaction*). It may detect some more faults.
    - *Static compaction*
    - More on compaction on next three slides

For large circuits, ATPG patterns may contain up to 99% X's.

11

Static and Dynamic Compaction of Sequences

- **Static Compaction**
  - ATPG should leave unassigned inputs as X
  - Two patterns *compatible* – if no conflicting values for any PI
  - Combine two tests t_a and t_b into one test $t_{ab} = t_a \cap t_b$ using D-intersection
  - Detects union of faults detected by t_a & t_b
- **Dynamic Compaction**
  - Process every partially-done ATPG vector immediately
  - Assign 0 or 1 (e.g. randomly) to PIs to test additional faults

12

Compaction Example

- t_1 = 0 1 X t_2 = 0 X 1
  t_3 = 0 X 0 t_4 = X 0 1
- **Combine t_1 and t_3, then t_2 and t_4**
- **Obtain:**
  - t_{13} = 0 1 0 t_{24} = 0 0 1
- ***Test Length* shortened from 4 to 2**

Test Compaction

- **Fault simulate test patterns in reverse order of generation**
  - ATPG patterns go first
    - Patterns detecting Hard faults can detect easy ones
  - Randomly-generated patterns go last (because they may have less coverage)
  - When coverage reaches 100%, drop remaining patterns (which are the useless random ones)
  - Significantly shortens test sequence – economic cost reduction

ATPG Systems

- **Identify untestable faults by other methods**
  - If the goal is to identify only untestable faults as opposed to find a test, some other methods may do a better job – example of such techniques are:
    - Recursive learning
    - Controllability evaluations
    - Probability analysis
    - etc.

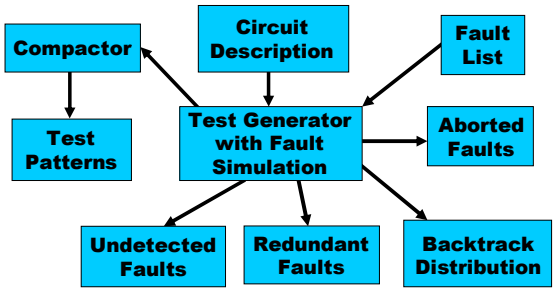
Fault Coverage and Efficiency

$$\textit{Fault Coverage} = \frac{\text{\# of detected faults}}{\text{Total \# faults}}$$

$$\textit{Fault Efficiency} = \frac{\text{\# of detected faults}}{\text{Total \# faults -- \# untestable faults}}$$

Also known as Test Coverage

ATPG Systems



Testability Measure

Testability Analysis - Purpose

- **Need approximate measure of:**
  - Difficulty of setting internal circuit lines to 0 or 1 by setting primary circuit inputs
    - Used in Fault Activation
  - Difficulty of observing internal circuit lines by observing primary outputs
    - Used in Propagation
- **Uses:**
  - Analysis of difficulty of testing internal circuit parts – redesign or add special test hardware (test point)
  - Guidance for algorithms computing test patterns – avoid using hard-to-control lines
  - Estimation of fault coverage
    - Obtaining # of untestable faults
  - Estimation of test vector length

19

Origins

- Control theory
- Rutman 1972 -- First definition of controllability
- Goldstein 1979 -- SCOAP
  - First definition of observability
  - First elegant formulation
  - First efficient algorithm to compute controllability and observability
- Parker & McCluskey 1975
  - Definition of Probabilistic Controllability
- Brglez 1984 -- COP
  - 1st probabilistic measures
- Seth, Pan & Agrawal 1985 – PREDICT
  - 1st exact probabilistic measures

20

Testability Analysis - Constraints

- Involves Circuit Topological analysis, but no test vectors and no search algorithm
  - Static analysis
- Linear computational complexity
  - Otherwise, is pointless – might as well use automatic test-pattern generation and calculate:
    - Exact fault coverage
    - Exact test vectors

21

Types of Measures

- SCOAP – Sandia Controllability and Observability Analysis Program
- Combinational measures:
  - *CC0* – Difficulty of setting circuit line to logic 0
  - *CC1* – Difficulty of setting circuit line to logic 1
  - *CO* – Difficulty of observing a circuit line
- Sequential measures – analogous:
  - *SC0*
  - *SC1*
  - *SO*

22

Range of SCOAP Measures

- Controllabilities – 1 (easiest) to infinity (hardest)
- Observabilities – 0 (easiest) to infinity (hardest)

- Combinational measures:
  - Roughly proportional to # circuit lines that must be set to control or observe given line
- Sequential measures:
  - Roughly proportional to # times a flip-flop must be clocked to control or observe given line

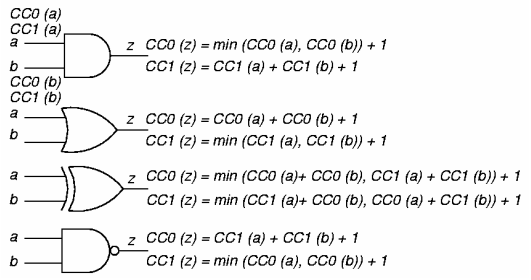
23

Goldstein's SCOAP Measures

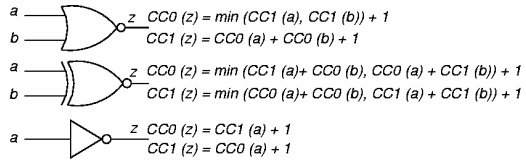
- **AND gate O/P 0 controllability:**
  output_controllability = min (input_controllabilities) + 1
- **AND gate O/P 1 controllability:**
  output_controllability = Σ (input_controllabilities) + 1
- **XOR gate O/P controllability**
  output_controllability = min (controllabilities of each input set) + 1
- **Fanout Stem observability:**
  Σ or min (some or all fanout branch observabilities)

24

Controllability Examples

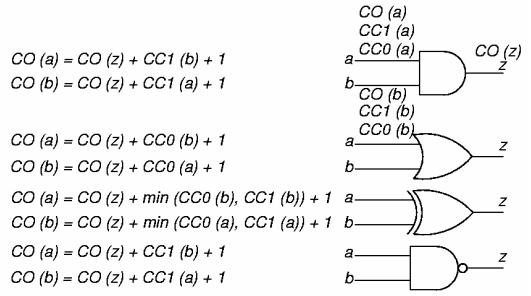


More Controllability: Examples



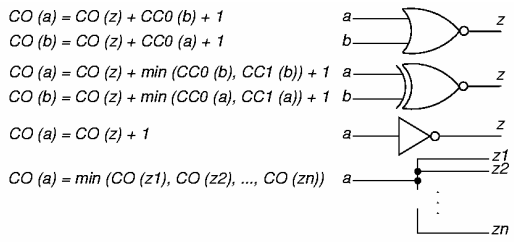
Observability Examples

To observe a gate input: Observe output and make other input values non-controlling



More Observability Examples

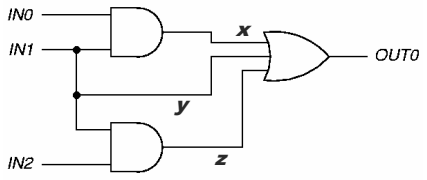
To observe a fanout stem: Observe it through branch with best observability



Error Due to Stems & Reconverging Fanouts

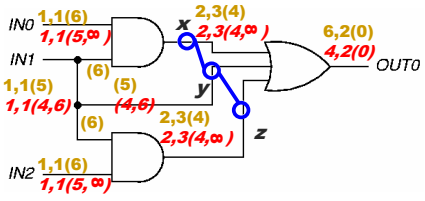
SCOAP measures wrongly assume that controlling or observing *x, y, z* are independent events

- ***CC0 (x), CC0 (y), CC0 (z)*** **correlate**
- ***CC1 (x), CC1 (y), CC1 (z)*** **correlate**
- ***CO (x), CO (y), CO (z)*** **correlate**



Correlation Error Example

- **Exact computation of measures is NP-Complete and impractical**
- **Italicized (red) measures show correct values – SCOAP measures are not italicized CC0,CC1 (CO)**

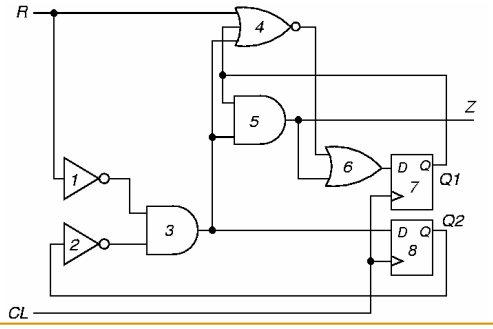


Levelization Algorithm 6.1

- **Label each gate with max # of logic levels from primary inputs or with max # of logic levels from primary output**
- **Assign level # 0 to all *primary inputs* (PIs)**
- **For each PI fanout:**
  - **Label that line with the PI level number, &**
  - **Queue logic gate driven by that fanout**
- **While queue is not empty:**
  - **Dequeue next logic gate**
  - **If all gate inputs have level #'s, label the gate with the maximum of them + 1;**
  - **Else, requeue the gate**

31

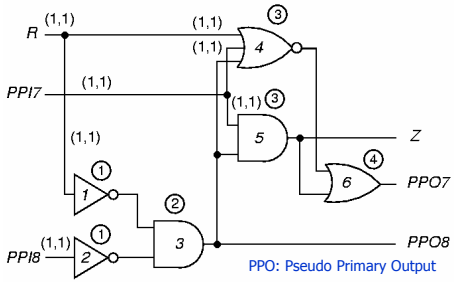
Sequential Example



32

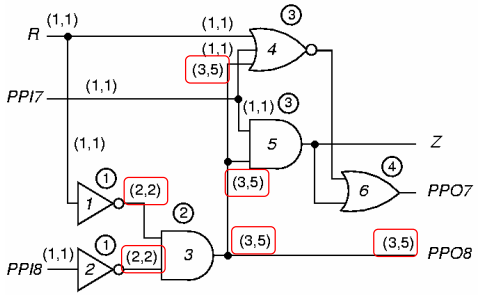
Controllability Through Level 0

Circled numbers give level number. (CC0, CC1) denote 0 and 1 controllability on each net.



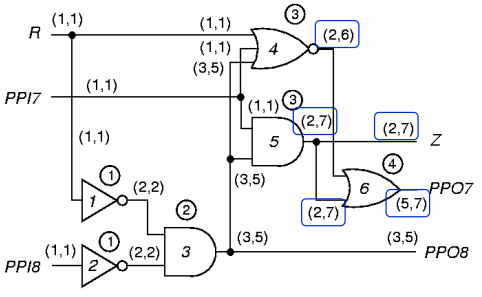
33

Controllability Through Level 2



34

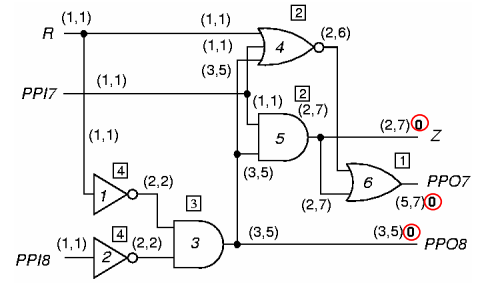
Final Combinational Controllability



35

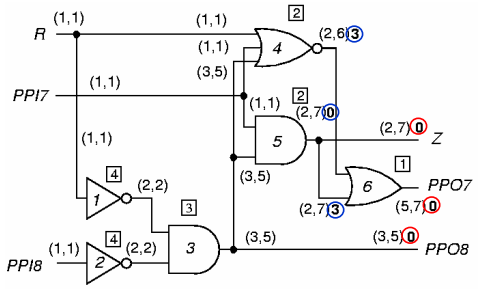
Combinational Observability for Level 1

Number in square box is level from *primary outputs* (POs). (CC0, CC1) CO



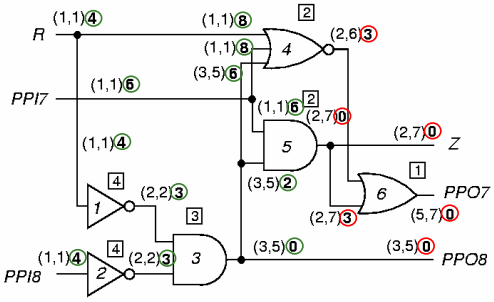
36

Combinational Observabilities for Level 2



37

Final Combinational Observabilities



38

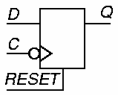
Sequential Measure Differences

- **Combinational**
  - **Increment *CC0, CC1, CO* whenever you pass through a gate, either forwards or backwards**
- **Sequential**
  - **Increment *SC0, SC1, SO* only when you pass through a flip-flop, either forwards or backwards, to *Q*, $\overline{Q}$, *D*, *C*, *SET*, or *RESET***
- **Both**
  - **Must iterate on feedback loops until controllabilities stabilize**

39

D Flip-Flop Equations

- **Assume a synchronous RESET line.**
- ***CC1* (*Q*) = *CC1* (*D*) + *CC1* (*C*) + *CC0* (*C*) + *CC0* (*RESET*)**
- ***SC1* (*Q*) = *SC1* (*D*) + *SC1* (*C*) + *SC0* (*C*) + *SC0* (*RESET*) + 1**
- ***CC0* (*Q*) = min [*CC1* (*RESET*) + *CC1* (C), CC1(RESET) + *CC0* (*C*), *CC0* (*D*) + *CC1* (*C*) + *CC0* (*C*) + CC0(RESET)]**
- ***SC0* (*Q*) is analogous**
- ***CO* (*D*) = *CO* (*Q*) + *CC1* (*C*) + *CC0* (*C*) + *CC0* (*RESET*)**
- ***SO* (*D*) is analogous**



40

D Flip-Flop Clock and Reset

- ***CO* (*RESET*) = *CO* (*Q*) + *CC1* (*Q*) + *CC1* (*RESET*) + *CC1* (*C*) + *CC0* (*C*)**
- ***SO* (*RESET*) is analogous**
- **Three ways to observe the clock line:**
  1. **Set *Q* to 1 and clock in a 0 from *D***
  2. **Set the flip-flop and then reset it**
  3. **Reset the flip-flop and clock in a 1 from *D***
- ***CO* (*C*) = min [*CO* (*Q*) + *CC1* (*Q*) + *CC0* (*D*) + *CC1* (*C*) + *CC0* (*C*), *CO* (*Q*) + *CC1* (*Q*) + *CC1* (*RESET*) + *CC1* (*C*) + *CC0* (*C*), *CO* (*Q*) + *CC0* (*Q*) + *CC0* (*RESET*) + *CC1* (*D*) + *CC1* (*C*) + *CC0* (*C*)]**
- ***SO* (*C*) is analogous**

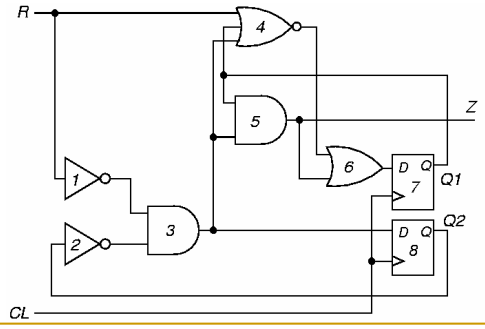
41

Algorithm 6.2: Testability Computation

1. **For all PIs, *CC0* = *CC1* = 1 and *SC0* = *SC1* = 0**
2. **For all other nodes, *CC0* = *CC1* = *SC0* = *SC1* = ∞**
3. **Go from PIs to POS, using *CC* and *SC* equations to get controllabilities -- Iterate on loops until *SC* stabilizes -- convergence guaranteed**
4. **For all POs, set *CO* = *SO* = ∞**
5. **Work from POs to PIs, Use *CO*, *SO*, and controllabilities to get observabilities**
6. **Fanout stem (*CO*, SO) = min branch (CO, SO)**
7. **If a *CC* or *SC* (*CO* or *SO*) is∞ , that node is uncontrollable (unobservable)**

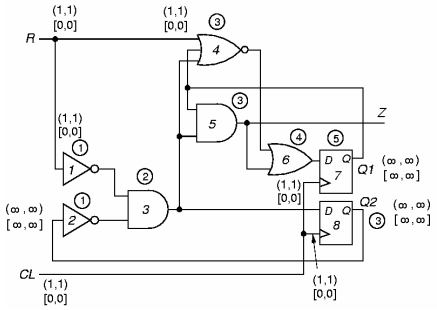
42

Sequential Example



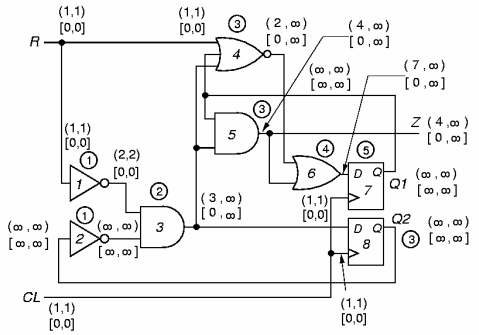
43

Sequential Example Initialization



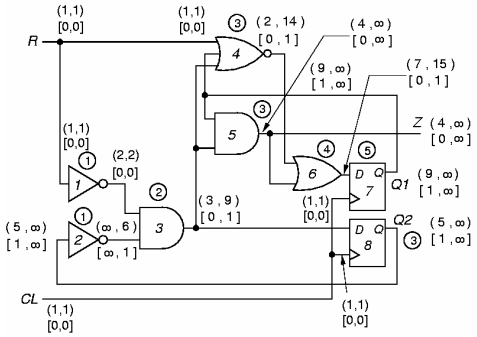
44

After 1 Iteration



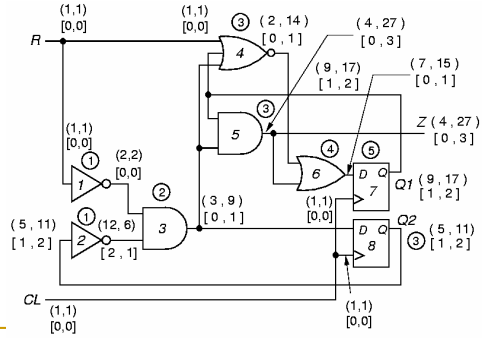
45

After 2 Iterations



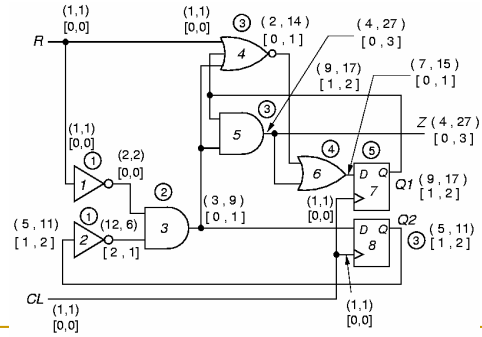
46

After 3 Iterations



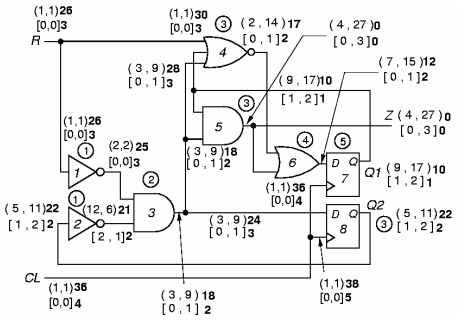
47

Stable Sequential Measures



48

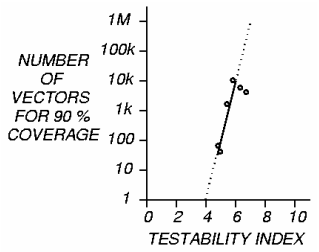
Final Sequential Observabilities



Test Vector Length Prediction

- **First compute *testabilities* for stuck-at faults**
  - $T(x \text{ sa0}) = CC1(x) + CO(x)$
  - $T(x \text{ sa1}) = CC0(x) + CO(x)$
  - $\textit{Testability index} = \log_{f_i} \Sigma T(f_i)$

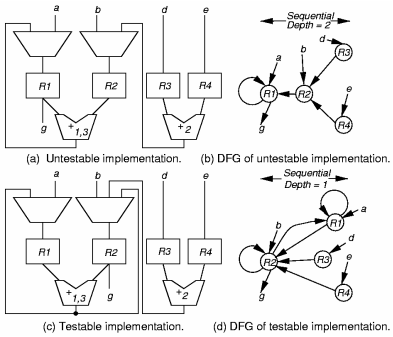
Number Test Vectors vs. Testability Index



High Level Testability

- **Build *data path control graph* (DPCG) for circuit**
- **Compute sequential depth -- # arcs along path between PIs, registers, and POs**
- **Improve Register Transfer Level Testability with redesign**

Improved RTL Design



(a) Untestable implementation. (b) DFG of untestable implementation.

(c) Testable implementation. (d) DFG of testable implementation.

Summary

- **ATPG systems**
  - **Methods to reduce test generation effort while generating efficient test vectors**
- **Testability approximately measures:**
  - **Difficulty of setting circuit lines to 0 or 1**
  - **Difficulty of observing internal circuit lines**
  - **Examples for computing these values**
- **Uses:**
  - **Analysis of difficulty of testing internal circuit parts**
    - **Redesign circuit hardware or add special test hardware where measures show bad controllability or observability**
  - **Guidance for algorithms computing test patterns**
  - **Estimation of fault coverage – 3-5 % error (see text)**
  - **Estimation of test vector length**