VLSI Testing and Verification (UEC638)

Dr. Bharat Garg,
Assistant Professor, DECE

Department of Electronics and Communication Engineering, Bhadson Road, Patiala, Punjab, INDIA - 1474001

December 30, 2024

Syllabus: VLSI Testing and Verification

- **Introduction:** Role of testing in VLSI design, Issues in test and verification of complex chips, VLSI test process and equipment, Test economics, Yield analysis and product quality.
- **Faults modelling and fault simulation:** Physical faults and their modelling, Stuck-at faults, Bridging faults, Fault collapsing, Fault simulation, Deductive.
- **ATPG for combinational circuits:** D-Algorithm, Boolean Difference, PODEM, Random, Exhaustive and Weighted Test Pattern Generation, Aliasing and its effect on Fault coverage.
- **ATPG for sequential circuits:** ATPG for Single-Clock Synchronous Circuits, Time frame expansion Method, Simulation-Based Sequential Circuit ATPG.
- **Memory testing and BIST:** Permanent, Intermittent and pattern sensitive faults, March test notion, Memory testing using march tests, PLA testing, Ad-Hoc DFT methods, Scan design, Partial scan design.
- **Verification:** Design verification techniques based on simulation, Analytical and formal approaches, Functional verification, Timing verification, Formal verification, Basics of equivalence checking.

Syllabus: VLSI Testing and Verification

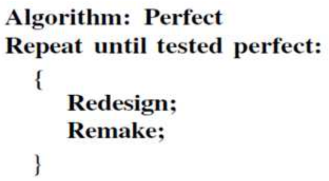
- **Course Learning Outcomes:** The student will able to
  1. Acquire knowledge about fault modelling and collapsing.
  2. Learn about various combinational and sequential automatic test pattern generation techniques.
  3. Analyze different memory faults and its testing methods.
  4. Develop the verification plan for the small to complex VLSI designs.
  5. Develop test bench using HVL for testing and verification of VLSI designs.

- **Text Books:**
  1. M. Bushnell and Vishwani Agrawal, Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits, Springer, ISBN 978-0792379911.
  2. Chris Spear, System Verilog for Verification, Springer, ISBN 978-1-4614-0714-0
- **Reference Books:**
  1. M. Abramovici, M. Breuer, and A. Friedman, Digital System Testing and Testable Design, IEEE Press, 1994 2. Diraj K. Pradhan, “Fault Tolerant Computer System Design”, Prentice Hall.
  2. L. T. Wang, C. W. Wu, and X. Wen, VLSI Test Principles and Architectures, Morgan Kaufmann, 2006, ISBN-13: 978-0-12-370597-6, ISBN-10: 0-12-370597-5.
  3. System-on-a-Chip Verification-Methodology and Techniques, P. Rashinkar, Paterson and L. Singh, Kluwer Academic Publishers, 2001
  4. Janick Bergeron, “Writing test benches functional verification of HDL models” Kluwer Academic Publishers, New York, Boston, Dordrecht, London, Moscow, 2002

Outline

Test Philosophy

- You are quite familiar with word **test** and obviously don't like it, but understand from teacher's point of view.
- The teacher sets a domain of knowledge for testing called course syllabus.
  - May includes contents of books, class notes, lectures, or some combination of all those.
- Testing Method: Teacher asks questions and analyses the response (by matching answers with the correct one from the book)
- The quality of the testing depends on how well the test questions cover the syllabus. Since there no one has infinite time, number of questions must be cleverly devised.
- Teacher makes some assumptions on certain typical errors students most probably commit. Questions are devised to uncover those errors.
- Electronic testing also uses fault modelling and tests are generated for the assumed fault models.
- In successful testing high percentage of the modelled faults is tested.
- Finally, remember that, if you fail, you must repeat the course. This is similar to redesign and remake in our "Algorithm: Perfect."



```
Algorithm: Perfect
Repeat until tested perfect:
  {
      Redesign;
      Remake;
  }
```

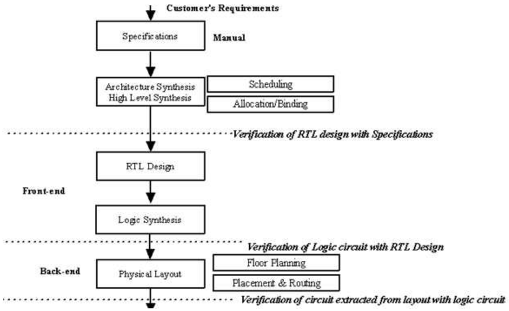
Introduction

- The rapid advances in integration technologies enabling us to fabrication of millions of transistors in a single Integrated Circuit (IC) or chip.
- The functionality of electronics equipments/gadgets has achieved a phenomenal while their physical sizes and weights have come down drastically.
- Integration with a complexity of 10's of transistors is called SSI, with 100's is MSI, with 1000's is LSI, with 10,000 it is Very Large Scale Integration (VLSI).
- Systems of systems can be implemented in a VLSI IC. However, with this rise in functionality of VLSI ICs, design problem has become huge and complex.
- To address complexly issue, after the design specifications are complete almost all the other steps are automated using CAD tools.
    - However, even designs automated using CAD tools may have bugs.
- Due to extremely large size of the design space, it is not possible to verify correctness of the design under all possible situations.
    - So technique is required that can verify correctness of design, without exercising exhaustive input-output combinations, this technique is called formal verification.
- In VLSI designs millions of transistors are packed into a single chip. This leads to manufacturing defects and all the chips need to be physically tested by giving input signals from a pattern generator and comparing responses using a logic analyzer; this process is called Testing.
    - Thus, chip manufacturing exhibits three broad steps: DESIGN-VERIFICATION-TEST.

Introduction

- VLSI ICs can be divided into analog, digital or mixed-signal (both analog and digital on the same chip) based on their functionality.
  - Digital ICs contain logic gates, flip-flops, multiplexers,
  - Work using binary mathematics to process "one" and "zero" signals.
  - Analog ICs, such as current mirrors, voltage followers, filters, etc. work by processing continuous signals.
- When single IC has both analog and digital components it is called mixed signal IC e.g., IC containing Analog to Digital Converter (ADC).

- The automation algorithms and CAD tools are mainly available for digital ICs.
- Verification is done at before moving to the next level.
- Early detection of the bugs reduces the cost of the design.
- In realty, the design and verification process are done in parallel.



Customer's Requirements
Specifications
Manual
Architecture Synthesis High Level Synthesis
Scheduling
Allocation/Binding
Verification of RTL design with Specifications
RTL Design
Front-end
Logic Synthesis
Verification of Logic circuit with RTL Design
Back-end
Physical Layout
Floor Planning
Placement & Routing
Verification of circuit extracted from layout with logic circuit

Introduction to VLSI Testing and Verification

- Testing is a post-silicon (post-fabrication) VLSI design step that ensures that the physical device, manufactured from the synthesized design, has no manufacturing defect.
- Testing determines the presence of fault(s) in a given system.
  - No amount of testing time guarantee that circuit is fault free.
  - Testing increases the confidence (probability) of circuit's proper functionality.
- Verification is predictive analysis to ensure that the synthesized design, when manufactured, will perform the given I/O function.
- Verification is alternative to testing, used to verify the correctness of a design.

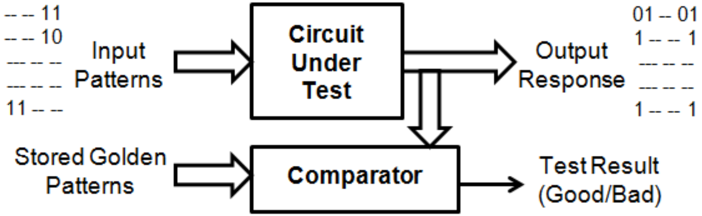


Figure: Principle of Testing

Verification Vs Testing

- Verifies the correctness of the design
- Performed by simulating or formal method.
- Performed once prior to manufacturing.
- Responsible for the quality of the design.

- Verifies the correctness of manufactured h/w.
- Exhibits two process:Test generation and Test application
- Test application performed on each manufactured device.
- Responsible for the quality of the device.

Ideal Vs Practical Testing

- Detects all defects produced in the manufacturing process.
- Ideal tests pass all functionally good devices.
- Ideal tests covers large number and varieties of faults.
- Complex test needs to generated for some complex defect.
- Defect oriented testing has to be done.

- Practical tests detect only defect which can be modeled, but fail detect real faults which may not be modeled.
- Incomplete coverage of modeled faults due to high complexity.
- Some good chips are rejected. The fraction of such chips is called the yield loss.
- Some bad chips pass tests. The fraction (or percentage) of bad chips among all passing chips is called the defect level.

Cost of Testing

- **Design for Testability (DFT):** Refers to hardware design styles or added hardware that reduces test generation complexity since the test generation complexity increases exponentially with the size of the circuit.
    - Chip area overhead and yield reduction
    - Performance overhead
- Software processes of test
    - Test generation and fault simulation
    - Test programming and debugging

  Manufacturing test
    - Automatic Test Equipment (ATE) capital cost
    - Test center operational cost

An 0.5-1.0GHz frequency, analog instruments,1,024 digital pins

- ATE purchase price
  $= \$1.2M + 1{,}024 \times \$3{,}000 = \$4.272M$
- Running cost (five-year linear depreciation)
  $= \text{Depreciation} + \text{Maintenance} + \text{Operation}$
  $= \$0.854M + \$0.085M + \$0.5M$
  $= \$1.439M/\text{year}$
- Test cost (24 hour ATE operation)
  $= \$1.439M/(365 \times 24 \times 3{,}600)$
  $= 4.5\ \text{cents/second}$

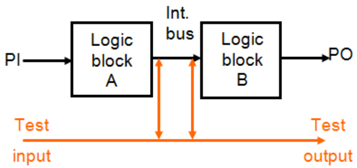


Figure: Test hardware applies tests to blocks A and B and to internal bus; avoids test generation for combined A and B blocks.

Level and Types of Testing

- Testing can be carried out at: Chip level, Board level, and/or at System level.
- Or other ways it can be carried out at Transistor level, Gate level, RTL level, and/or Functional/Behavioral level.
- Rule of 10: Testing cost is 10 times more as we move from top to bottom level of testing. Therefore, early detection of fault/defect is always desirable.
- Types of testing
  - Verification testing, characterization testing, or design debug:
    Verifies correctness of design and test procedure – may require correction of either or both.
  - Manufacturing testing:
    Factory testing of all manufactured chips for parametric and logic faults, and analog specifications.

    Burn-in or stress testing.
  - Acceptance testing (incoming inspection):
    User (customer) tests purchased parts to ensure quality.

Characterization or Verification Testing

- A characterization test determines the exact limits of the device operation values.
  - Ferociously expensive and applied to selected (not all) parts.
  - Functional tests are run and comprehensive AC and DC measurements are made. Used prior to production or manufacturing test.
  - May requires: Scanning Electron Microscope (SEM) tests, Bright-Lite detection of defects, Electron beam testing, Artificial intelligence (expert system) methods and Repeated functional tests.
- Worst-case test
  - Choose test that passes/fails chips, select statistically significant sample of chips.
  - Repeat test for every combination of environmental variables, Diagnose and correct design errors.
- Continue throughout production life of chips to improve design and process to increase yield

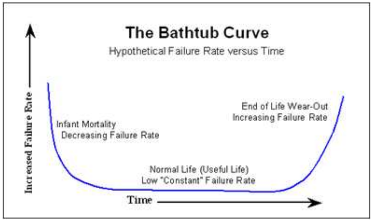


Figure: Bathtub curve

Manufacturing Test

- Determines whether manufactured chip meets specification. Fault diagnosis is not done and every device on chip is tested.
- Must cover high % of modeled faults with minimum test time (to control cost).
- Burn-in or stress test: Chips are subjected to high temperature and over-voltage supply, while running production tests to catch:
  - Infant mortality cases – these are damaged or weak (low reliability) chips that will fail in the first few days of operation – burn-in causes bad devices to fail before they are shipped to customers
  - Freak failures – devices having same failure mechanisms as reliable devices.
- Incoming inspection: can be similar to production testing or more comprehensive than production testing and/or tuned to specific system application.
  - Often done for a random sample size where sample size depends on device quality and system reliability requirement.
  - The actual test selection depends on the manufacturing level (processing, wafer, or package) being tested. For example, wafer sort or packaged device tests.
- Wafer sort or probe test: done before wafer is scribed and cut into chips
  - Includes test site characterization specific test devices are checked with specific patterns to measure: gate threshold, polysilicon field threshold, poly sheet resistance, etc.

Manufacturing Test (Cont...)

- Two types of tests:
  - Parametric: measures electrical properties of pin electronics – delay, voltages, currents, etc. It is fast and cheap.
  - Functional: used to cover very high % of modeled faults, test every transistor and wire in digital circuits. It is long and expensive.
- Two different meaning of functional tests:
  - ATE and Manufacturing World: any vectors applied to cover high % of faults during manufacturing test.
  - Automatic Test-Pattern Generation World: testing with verification vectors, which determine whether hardware matches its specification typically have low fault coverage (<70 %).

Test Specification and Plan

- Test Specifications:
  - Functional Characteristics
  - Type of Device Under Test (DUT)
  - Physical Constraints – package, pin numbers, etc.
  - Environmental Characteristics – power supply, temperature, humidity, etc.
  - Reliability – acceptance quality level (defects/million), failure rate, etc.
- Test plan generated from specifications
  - Type of test equipment to use
  - Types of tests
  - Fault coverage requirement

Test Programming

- Initially test plan and test vectors are developed from the device specifications.
- Test program generator provides sequence of instruction that tester would follow to conduct testing.
- For example sequence of events are:
  - Apply power,
  - Apply clocks and vector in input pins
  - Strobe output pins and
  - Compare output signals with stored expected response.
- Modern testers provide a choice of input signal waveform, mask output signal sense high impedance states and have variety of sophisticated capabilities.

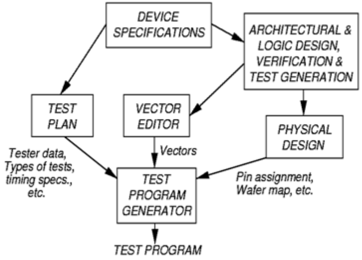

DEVICE SPECIFICATIONS
ARCHITECTURAL & LOGIC DESIGN, VERIFICATION & TEST GENERATION
TEST PLAN
VECTOR EDITOR
PHYSICAL DESIGN
Tester data, Types of tests, timing specs., etc.
Vectors
Pin assignment, Wafer map, etc.
TEST PROGRAM GENERATOR
TEST PROGRAM

Figure: Test Program Generator

Test Data Analysis

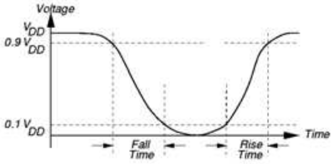
- Uses of ATE test data:
  - Reject bad DUTs
  - Fabrication process information
  - Design weakness information
- Devices that did not fail are good only if tests covered 100% of faults.
- Failure mode analysis (FMA):
  - Diagnose reasons for device failure, and find design and process weaknesses
  - Improve logic and layout design rules

Electrical Parametric Testing

- **Electrical faults** occur on the chip pins modifies the voltages/currents/delays at that pins.
  1. Probe test (wafer sort): Examine die on wafer before broken into chips to weed out grossly defective devices.
  2. Contact electrical test
  3. Functional & layout-related test
  4. DC parametric test
  5. AC parametric test
- Test 1, 2, 4, and 5 are electrical tests while 3 is a functional test.
- The electrical fault observed on the chip affect the device input/output interface.
- There are two kinds of electrical faults.
  - Unacceptable voltage/current/delay at pin
  - Unacceptable device operation limits.

DC and AC Parametric Test

- DC Parametric Test: Measures steady-state electrical characteristics.
  - **Contact test:** Draw out the current from the pin and measure the voltage at the pin to verify that the chip pins have no open or shorts.
  - **Power consumption test:** Finds worst case power consumption in static and dynamic conditions.
  - **Output short current test:** Verifies that the output current drive is sustained at high and low output voltages.
  - **Output drive current test:** For a specific output drive current, this test verifies that the output voltage is maintained.
  - **Threshold test:** This test determines the VIL and VIH.
- AC Parametric Test:
  - We apply AC voltage at some set of frequency to the chip and measure terminal impedance or dynamic resistance.
  - We set a DC bias level for these tests which determines the delays caused by input and output capacitances.
  - Rise/Fall Time Test, Setup, Hold, and Release Times Test, Propagation Delay Test.



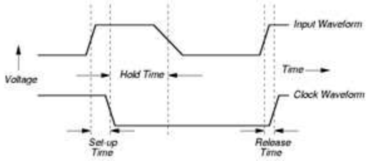


Figure: Delay measurement using AC parametric tests.

Test Economics

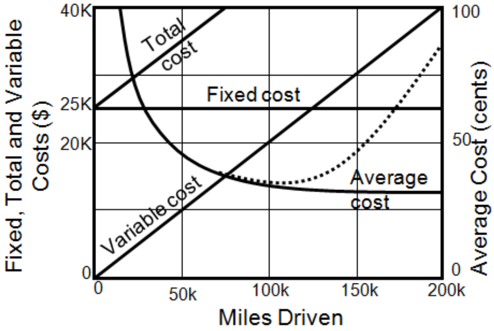
- **Economics** is the study of how men choose to use limited resources (land, labor, machinery, and technical knowledge) to produce various commodities (such as wheat, overcoats, roads, and yachts) and to distribute them to various members of society for their consumption.
- **Engineering Economics** is the study of how engineers choose to optimize their designs and construction methods to produce systems that will optimize their efficiency.
- Various cost includes: Fixed cost, Variable cost, Total cost, and Average cost.
- **Fixed Cost (FC):** is the cost of things that is necessary but do not change with use.
- **Variable Cost (VC):** increases with the production output.
- **Total Cost (TC):** is the sum of the fixed and variable cost, and increases with production output.
- **Average Cost (AC):** is obtained by dividing the total cost by the number of units produced.

Cost Analysis

- **Cost analysis** is done to compare alternatives or minimize cost to optimize the design.

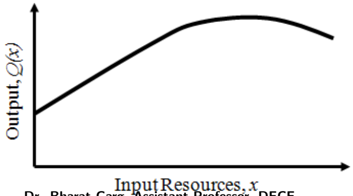
Example: Cost of running a car

Fixed Cost	\$25,000	Purchase price of car
Variable Cost	20 cents/mile	Gasoline, repair
Total Cost	\$25,000 + 0.2x	Travelling x miles
Average Cost	\$25,000/x+0.2	Total-cost/x



Production

- Production is the process of making articles that society needs.
- Short term production: means that some of the inputs are fixed.
- Long term production: is over a period during which the company can change all inputs, including the size of the manufacturing plant.
  - Inputs (X): Labor, land, capital, enterprise, energy (X may include both fixed and variable costs): all are converted into dollars equivalent.
  - Variable costs (X)
  - Production output, $Q = f(X)$
  - Average product, Q/X
  - Marginal product (slope), dQ/dX
  - Total Average cost, X/Q
- Law of Diminishing Return: If one input of production is increased keeping inputs constant, then the output may increase, eventually reaching a point beyond which increasing the inputs will cause progressively less increase in output.

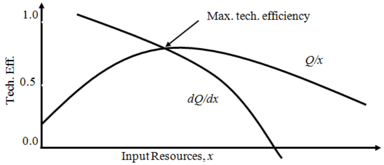


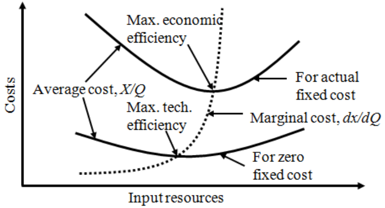
Technological Efficiency

- **Technological efficiency** $= Q/X$, where X is variable cost, Q is production output.
- Maximum economic efficiency minimizes the total average cost X/Q, where X is the total (fixed + variable) cost.
- To maximum technological efficiency:

$$\frac{d}{dx}\frac{Q}{x} = 0; \quad \text{or} \quad \frac{1}{x}\frac{dQ}{dx} - \frac{Q}{x^2} = 0; \quad or \quad \frac{Q}{x} = \frac{dQ}{dx}$$

- Maximum economic efficiency is achieved when total average cost equals the marginal cost, $X/Q = dX/dQ$.
- For average cost = marginal cost
  - Take variable cost to maximize technological efficiency.
  - Take total cost to maximize economic efficiency.





Mass Production

- If production increasing at a faster rate than the increasing of inputs. This is known as increasing returns to scale.
- Some reasons for increasing returns to scale:
  - Technological factors
  - Specialization
  - Only some inputs are increased
- If increase of inputs continues, eventually the law of diminishing returns applies.
- Benefits: Savings in manufacturing costs (capital and operational) and time, reduced wastage, automation, etc.
- Costs: Extra hardware, training of personnel, etc.

$$Benefit/cost(B/C)ratio = \frac{AnnualBenefits}{AnnualCost} > 1$$

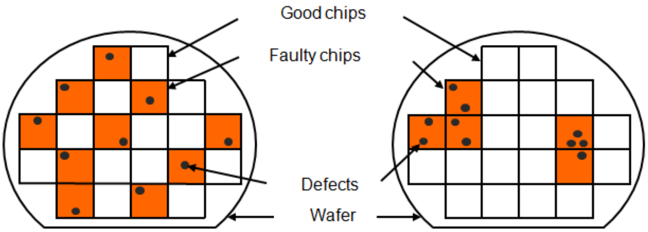
- Thus, Test Economics:
  - teaches us how to make the right trade-offs.
  - It combines common sense, experience and mathematical methods.
  - The overall benefit/cost ratio for design, test and manufacturing should be maximized; one should select the most economic design over the cheapest design.
  - A DFT or test method should be selected to improve the product quality with minimal increase in cost due to area overhead and yield loss.

Yield Analysis and Product Quality

- A manufacturing defect is a finite chip area with electrically malfunctioning circuitry caused by errors in the fabrication process.
- A chip with no manufacturing defect is called a good chip.
- Fraction (or percentage) of good chips produced in a manufacturing process is called the *yield*. Yield is denoted by symbol Y.

$$\textit{Chip-Cost} = \frac{\textit{Cost of fabrication and testing a wafer}}{\textit{Yield X Number of chip sites on wafer}}$$

- Process variations, such as impurities in wafer material and chemicals, dust particles on masks or in the projection system, mask misalignment, incorrect temperature control, etc., can produce defects (physical imperfection) on wafers.



Unclustered defects
Wafer yield = 12/22 = 0.55

Clustered defects (VLSI)
Wafer yield = 17/22 = 0.77

Defect Level or Rejection Ratio

- Defect level (DL) is the ratio of faulty chips among the chips that pass tests.
  - DL is measured as parts per million (ppm).
  - It is a measure of the effectiveness of tests.
  - DL is a quantitative measure of the manufactured product quality. For commercial VLSI chips a DL greater than 500 ppm is considered unacceptable.
- Determination of DL:
  - From field return data: Chips failing in the field are returned to the manufacturer. The number of returned chips normalized to one million chips shipped is the DL.
  - From test data: Fault coverage of tests and chip fallout rate are analyzed. A modified yield model is fitted to the fallout data to estimate the DL.
- Summary:
  - VLSI yield depends on two process parameters, defect density (d) and clustering parameter (α).
  - Yield drops as chip area increases; low yield means high cost.
  - Fault coverage measures the test quality.
  - Defect level (DL) or reject ratio is a measure of chip quality.
  - DL can be determined by an analysis of test data.
  - For high quality: DL $<$ 500 ppm, fault coverage $\approx$ 99%

Faults and Fault Modelling

- In engineering, models bridge the gap between physical reality and mathematical abstraction.
- Incorrectness in electronic circuits can be described as:
  - **Defect** is unintended difference between implemented h/w and its intended design.
  - **Error:** A wrong output signal produced by a defective system is called an error.
  - **Fault:** A representation of a defect at the abstraction function level is called as a fault.
- Example: Consider a digital system consisting of two inputs a and b, one output c, and one two-input AND gate. The system is assembled by connecting a wire between the terminal a and the first input of the AND gate. The output of the gate is connected to c. But the connection between b and the gate is incorrectly made – b is left unconnected and the second input of the gate is grounded. The functional output of this system, as implemented, is $c = 0$ instead of the correct output $c = a.b$
  For this system, we have:
  - Defect: a short to ground
  - Fault: signal b stuck at logic 0.
  - Error:$a = 1$, $b = 1$, output $c = 0$; correct output $c = 1$. Notice that the error is not permanent. As long as at least one input is 0, there is no error in the output.
- Why model faults:
  - I/O function tests inadequate for manufacturing.
  - Real defects (often mechanical) too numerous and often not analyzable.
  - A fault model identifies targets for testing and makes analysis possible.
  - Effectiveness measurable by experiments.

Real Defects in Chips and PCBs

Real Defects in Chips and PCBs

- Processing defects
  - Missing contact windows, Parasitic transistors,
  - Oxide breakdown, ...
- Material defects
  - Bulk defects (cracks, crystal imperfections)
  - Surface impurities (ion migration), ...
- Time-dependent failures
  - Dielectric breakdown, Electromigration, . . .
- Packaging failures
  - Contact degradation, Seal leaks, . . .

- Shorts
- Opens
- Missing components
- Wrong components
- Reversed components
- Bent leads
- Analog specifications
- Digital logic Performance

Common Fault Models

- Single stuck-at faults
- Transistor open and short faults
- Memory faults
- PLA faults (stuck-at, cross-point, bridging)
- Functional faults (processors)
- Delay faults (transition, path)
- Analog faults

Glossary of Fault Models

- Branch Fault: This fault is modelled at the behavioral level where the circuit function is described in a programming language. A branch fault affects a branch statement and causes it to branch to an incorrect destination.
- Bridging Fault: Usually modelled at the gate or transistor level, a bridging fault represents a short between a group of signals.
- Bus Fault: A bus fault specifies the status for each line in a bus as stuck-at-0, stuck-at-1, or fault-free. Thus, for an n-bit bus, there are bus faults. A total bus fault assumes all lines of the bus to be stuck at the same 0 or 1 state.
- Cross-point Fault: These faults are modelled in programmable logic arrays (PLA.) In the layout of a PLA, input and output variable lines are laid out perpendicular to the product-lines. Crossing signal lines either unnecessarily get connections or remain unconnected at cross-points.
    - Missing cross-point
    - Extra cross-point
- Delay Fault: These faults cause the combinational delay of a circuit to exceed the clock period. Specific delay faults are transition faults, gate-delay faults, line-delay faults, segment-delay faults, and path-delay faults.

Glossary of Fault Models (Cont...)

- Stuck-at Fault: This fault is modeled by assigning a fixed (0 or 1) value to a signal line in the circuit. A signal line is an input or an output of a logic gate or a flip-flop. The most popular forms are the single stuck-at faults, i.e., two faults per line, stuck-at-1 (s-a-1 or sal) and stuck-at-0 (s-a-0 or s-a-0).
- Initialization Fault: Circuits with memory elements (e.g., flip-flops) are designed so that they can be initialized by applying suitable input signals. Faults that interfere with such an initialization procedure are called initialization faults.

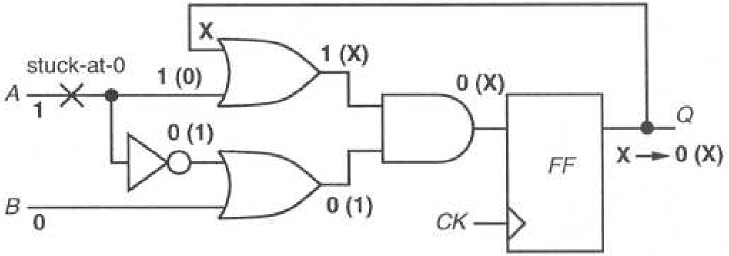


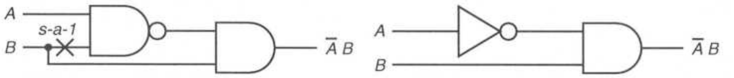
Figure: An initialization fault.

Glossary of Fault Models (Cont...)

- Intermittent Fault: A fault that appears and disappears as a function of time is called an intermittent fault. A fracture in an interconnect may produce an intermittent open for some time before it becomes a permanent fault.
- Logical Faults: These faults affect the state of logic signals. Normally, the state may be modelled as 0, 1, X (unknown), Z (high impedance), and a fault can transform the correct value to any other value.
- Memory Faults: Faults mode led in memory blocks are single cell stuck-at-[0,l] faults, pattern sensitive faults, cell coupling faults, and single stuck-at faults in the address decoder logic.
- Multiple Fault: A multiple fault represents a condition caused by the simultaneous presence of a group of single faults. Frequently considered multiple faults consist of the same type of single faults.
- Parametric Fault: Such a fault changes the values of electrical parameters of active or passive devices from their nominal or expected values.
- Pattern Sensitive Fault: This fault causes an incorrect behaviour in a certain part of the circuit only when a specific state occurs in some other part. Usually modelled in memories, a typical example is a fault condition that prevents writing a 1 in a memory cell when its physical neighbours have 0s stored in them.

Glossary of Fault Models (Cont...)

- Permanent Fault: Any faulty behavior that does not change with time is called a permanent fault. Faults that are not permanent and affect the circuit only at certain times (often at random instants) are called intermittent faults.
- Quiescent Current (I_{DDQ}) Fault: These faults are relevant to the CMOS technology. In the steady state the CMOS logic gate provides no conducting path between the power supply and ground. Thus, the steady state current, also known as the leakage or quiescent current (I_{DDQ}), of a CMOS gate is on the order of only a few μA. Under various fault conditions in the gate, this current can rise by several orders of magnitude thus allowing fault detection via current measurement. Faults detectable by this method are called I_{DDQ} faults.
- Race Fault: Stuck-at faults that cause a race condition in the circuit are called race faults. For a certain initial state and input, the final state of an asynchronous sequential circuit can vary depending on specific delays of its logic gates. Such a condition is known as a race.
- Redundant Fault: Any fault that does not modify the input-output function of the circuit is called a redundant fault.

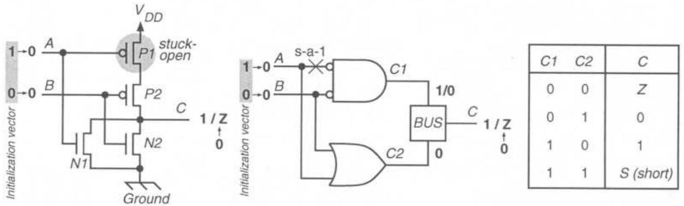


(a) A circuit with a redundant fault. (b) The circuit after removing the redundant fault.

Figure: An redundant fault.

Stuck-Open Fault

- Stuck-Open and Stuck-Short Faults: In purely digital circuits, a MOS transistor works as an ideal switch. A defect in these circuits is modeled as the switch being permanently in either the open or the shorted state. This fault model assumes just one transistor to be stuck-open or stuck-short. Stuck-short is also referred to as stuck-on or stuck-closed.
- In fault free circuit, application of A=0 and B=0 connect output to C to Vdd while isolating from ground. Whereas, application of either A=1 or B=1 connects output to Vgnd while isolation from Vdd.
- In faulty circuit, application of 00 at input makes the output node floating.
- To detect the fault, pre-discharging output node (by applying 10) followed by application of 00 which will produce faulty output 0 over correct logic 1.



C1	C2	C
0	0	Z
0	1	0
1	0	1
1	1	S (short)

(a) A CMOS NOR circuit. (b) Logic gate model for CMOS NOR. (c) Truth table of BUS.

Figure: Stuck-open fault, its modeling and test generation.

Gate level Modeling of Stuck-Open Fault

- Every series interconnection between Vdd/grouhd and output node is replaced by AND gate while, parallel connections are replaced by OR gate.
- The control inputs of pMOS transistors feed these gates via an inversion and those of nMOS transistors feed directly.
- When the two inputs of BUS are different, the output assumes the upper value (C1.)
- For a 00 input the BUS produces a high impedance (Z) state. For a 11 input the BUS produces a logic state S which means a short circuit between the supply nodes.
- Since 00 and 11 inputs of BUS cannot occur in the circuit without a fault because the two logic functions feeding the BUS are complementary.
- A stuck-open fault of a pMOS transistor is modeled as a stuck-at-1 fault at the corresponding input signal and that of an nMOS transistor as a stuck-at-0 fault.

Summary:

- The effect of a stuck-open fault is to produce a floating state at the output of the faulty logic gate. It can be detected by a test that detects a stuck-at fault on that output provided it was appropriately initialized by the previous vector.
- The effect of a stuck-short fault is to produce a power supply to ground conducting path. When the fault is activated, the logic state at the output of the gate containing the faulty transistor depends upon the relative impedances of transistors producing the conducting path and the switching threshold of other devices.

Single Stuck-at Fault

- A stuck-at fault is assumed to affect only the interconnection between gates.
- Each connecting line can have two types of faults: stuck-at-1 and stuck-at-0.
- Line with a stuck-at-1 fault will always have a logic state 1 irrespective of the correct logic output of the gate driving it.
- In general, several stuck-at faults can be simultaneously present in the circuit. A circuit with n lines can have $3^n - 1$ possible stuck line combinations. This is because each line can be in one of the three states: s-a-1, s-a-0, or fault-free.
- Even a moderate value of n will give an enormously large number of multiple stuck-at faults. It is a common practice, therefore, to model only single stuck-at faults.
- An n-line circuit can have at most 2n single stuck-at faults.
- Properties/characteristics of single stuck-at fault:
  1. Only one line is faulty.
  2. The faulty line is permanently set to either 0 or 1.
  3. The fault can be at an input or output of a gate.

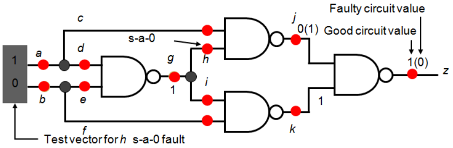
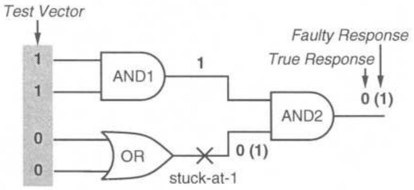
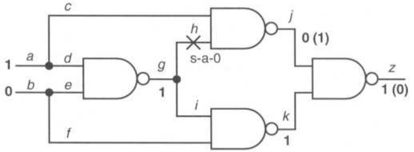


Figure: XOR circuit has 12 fault sites and 24 single stuck-at faults.

Stuck-at Fault: Example

- In example 1, a stuck-at-1 fault as marked at the output of the OR gate. If the normal output of the OR gate is 1, this fault will not affect any signal in the circuit. However, a 00 input to the OR gate will produce a 0 output in the normal circuit. The faulty circuit will have a 1 there.
- Figure shows the normal (faulty) value as 0(1). To propagate this fault to logic 1 must be applied to input of AND2 which in turn requires 11 at input of AND1. Now we have the input vector 1100 as a test for the s-a-1 fault since for this vector the normal output (true response) and the faulty output differ.
- In example 2, the single fault *h* s-a-0 is detectable by a 10 input.
- In a logic circuit, a net contains a stem or source (*g* in this circuit) and fanout branches (*h* and *i*.) The stem is the output of some gate and fanout branches are inputs of some other gates. To consider all possible faults, we model single stuck-at faults on the stem and all fanout branches of the net.





Fault Equivalence

- Let the output function of a single-output combinational circuit with n input variables is $f_0(v)$ where V is an n-bit Boolean vector.
- Let the output function of this circuits in the presence of two faults 1 and 2 are $f_1(v)$ and $f_2(v)$ respectively.
- Any test V for fault 1 must produce different values for $f_0(v)$ and $f_1(v)$. This condition can be expressed as:

$$f_0(v) \oplus f_1(v) = 1$$

- Similarly, a test for fault 2 must satisfy

$$f_0(v) \oplus f_2(v) = 1$$

- When fault 1 and fault 2 have exactly the same tests, i.e., all vectors that satisfy Eq.1 also satisfy Eq.2, and vice-versa, then the Boolean functions on the left hand sides of the two equations are identical. That is

$$[f_0(v) \oplus f_1(v)] \oplus [f_0(v) \oplus f_2(v)] = 0$$

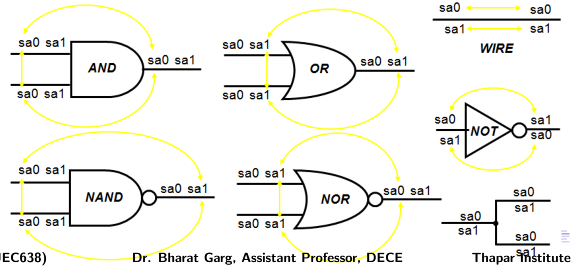
$$[f_1(v) \oplus f_2(v)] = 0$$

- This is known as the indistinguishability condition, shows that the two faulty functions are identical when the faults have the same set of tests.

Fault Equivalence (Cont...)

- Consider a k input AND gate. It has k + 1 s-a-0 faults on its input and output lines. Each s-a-0 fault transforms the AND gate to a constant 0 output function. Thus, all s-a-0 faults are equivalent.
- Two faults of a Boolean circuit are called equivalent iff they transform the circuit such that the two faulty circuits have identical output functions. Or, two faults f1 and f2 are equivalent if all tests that detect f1 also detect f2.
- If faults f1 and f2 are equivalent then the corresponding faulty functions are identical.
- Equivalent faults are also called indistinguishable and have exactly same set of tests.
- In a Boolean gate circuit,

$$\#\textit{Fault-sites} = \#\textit{PI} + \#\textit{Gates} + \#(\textit{Fanout-branches})$$



Equivalent Fault Collapsing

- The set of all faults in a circuit can be partitioned into equivalence sets, such that faults in an equivalent set are equivalent to each other.
- Equivalence sets divide faults into disjoint sets because if a fault exists in two equivalence sets then those sets can be merged together as one equivalence set.
- The process of selecting one fault from each equivalence set is called fault collapsing.
- The set of selected faults is known as equivalence collapsed set. The process of generating the equivalence fault set, known as equivalence fault collapsing.
- The relative size of the equivalence collapsed set with respect to the set of all faults is the collapse ratio:

$$Collapse\text{-}ratio = \frac{|Set\ of\ collapsed\ faults|}{|Set\ of\ all\ faults|}$$

Equivalent Fault Collapsing

- Figure shows two circuits, one without a fanout and the other with fanouts.
- Fault collapsing is performed in a level-by-level pass from inputs to output using local gate fault equivalences.
- The procedure begins at primary inputs and a gate is not processed until all gates feeding its inputs have been processed.
- At a gate, first input faults are examined. Only one among the equivalent faults is retained. Then any input faults that are equivalent to some output fault are deleted.
- The collapse ratio is around 50 or 60% and that more reduction occurs in the absence of fanouts.

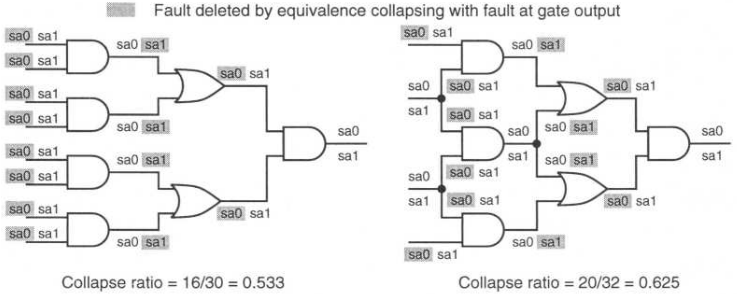

▨ Fault deleted by equivalence collapsing with fault at gate output
sa0 sa1
Collapse ratio = 16/30 = 0.533
Collapse ratio = 20/32 = 0.625

Figure: Example fault collapsing.

Fault Dominance

- Fault dominance: If all tests of fault F1 detect another fault F2, then F2 is said to dominate F1.
- Consider in 3-input AND gate, there are two single s-a-1 faults shown as Fl and F2.
- Suppose, T(F1) is the set of all tests for F1 and T(F2) is the set of all tests for F2.
- T(F1) contains one vector and T(F2) has seven vectors. As shown in Figure, T(F2) is larger and completely contains T(F1). According to the following definition, fault F2 dominates fault F1.
- The two faults are also called **"conditionally" equivalent** with respect to the test set of F1 when two faults F1 and F2 dominate each other, then they are equivalent.

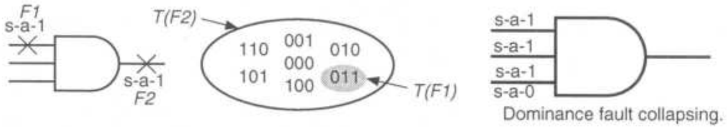
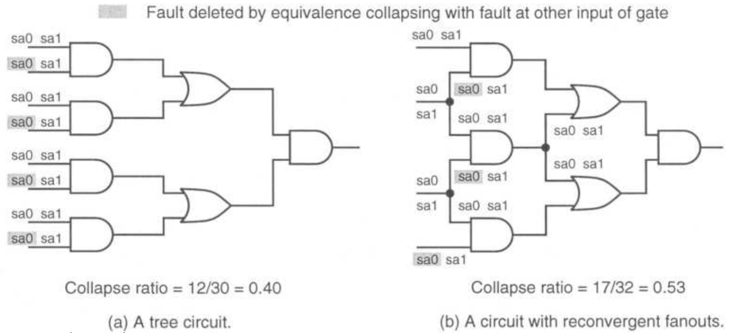


Figure: Example fault collapsing.

Fault Dominance (Cont...)

- Example shown below shows collapsed fault set with using equivalence and dominance.
  - For the fanout-free circuit, the collapsed fault set only contains input faults.
  - For the circuit with fanouts, the collapsed set contains faults located at checkpoints.
- Theorems:
  - Fault detection in fanout-free circuit. A test set that detects all single stuck-at faults on all primary inputs of a fanout-free circuit must detect all single stuck-at faults in that circuit.
  - Checkpoints: Primary inputs and fanout branches of a combinational circuit are called the checkpoints.
  - Checkpoints theorem: A test set that detects all single stuck-at faults of the checkpoints of a combinational circuit detects all single stuck-at faults in that circuit.


▨ Fault deleted by equivalence collapsing with fault at other input of gate
sa0 sa1
sa0 sa1
sa0 sa1
sa0 sa1
sa0 sa1
sa0 sa1
sa0 sa1
sa0 sa1
sa0 sa1
sa0
sa0 sa1
sa1
sa0 sa1
sa0 sa1
sa0 sa1
sa0
sa0 sa1
sa1
sa0 sa1
sa0 sa1
Collapse ratio = 12/30 = 0.40
Collapse ratio = 17/32 = 0.53
(a) A tree circuit.
(b) A circuit with reconvergent fanouts.

Problem on Equivalence and dominance fault collapsing

For the circuit of Figure:

1. What is the number of all potential fault sites?
2. Derive the equivalence collapsed set. What is the collapse ratio?
3. Derive the dominance collapsed set. What is the collapse ratio?

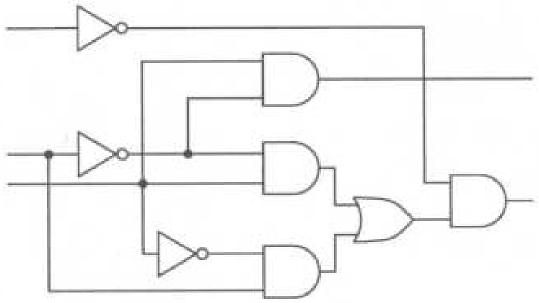
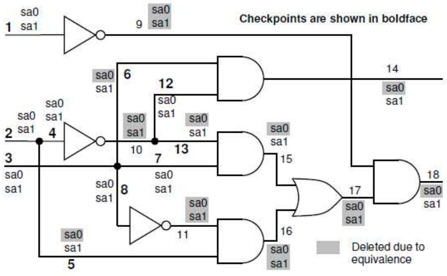


Figure: Problem on equivalence and dominance fault collapsing.

1. The given circuit is shown below with fault sites marked by numbers. The number of potential fault sites is 18. The total number of faults is 36.
2. The figure shows deletion of equivalent faults using an output to input pass. Of the 36 faults, 20 remain, giving a collapse ratio $= 20/36 = 0.56$.
3. Checkpoint lines are shown by boldface numbers. These are three PIs and seven fanout branches. Line 2 fans out to 4 and 5. Line 3 fans out to 6, 7 and 8. Line 10 fans out to 12 and 13. There are ten checkpoints and 20 checkpoint faults. Further, s-a-0 faults on lines 6 and 12 are equivalent and any one of them can be chosen. Similarly, s-a-0 faults on 7 and 13 are equivalent, and so are s-a-0 on 5 and s-a-1 on 8. Thus, the size of the fault set is reduced to 17, giving a collapse ratio $= 17/36 = 0.47$.



Circuit for Problem 4.11: (b) Equivalence collapse ratio = 20/36 = 0.56
(c) Dominance (uncollapsed faults at checkpoints) collapse ratio = 17/36 = 0.47

SIMULATOR

- Simulation is used to predict the result of design without actually building it.
- Simulation in electronic design, is used to verify:
  - The correctness of the design
  - The correctness of the tests.

Simulation for Design Verification

- The design is verified by true-value simulator:
  - A simulator that computes response for the given stimuli without injecting the fault in the design.
- Stimuli are generated from specifications
  - The stimuli correspond to those input and output specifications that are either critical or considered risky by the synthesis procedures
  - A frequently used strategy is to exercise all functions with only critical data patterns.
- Strength and weaknesses:
  - Design can be simulated at abstract level to detailed level (only functionality to critical timing analysis).
  - The simulation based verification depends on engineers expertise/heuristic and the input stimuli.

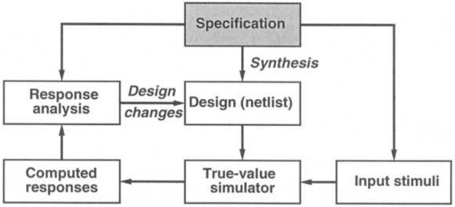


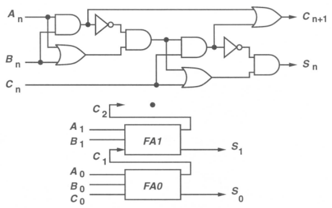
Figure: Simulation for design verification.

Simulation: Example

Figure shows the logic design of an adder circuit. The basic building block in this design is a full-adder that adds two data bits, A_n and B_n and one carry bit, C_n to produce sum S_n and carry outputs C_{n+1}, and respectively. For logic verification, one possible strategy is to select a set of vectors that will apply all possible inputs to each full-adder block.

To reduce the number of test vector while thoroughly checking design:

- Set $A_0 = B_0 = 0$ and apply all four combinations (00, 01, 10, 11) to A_1 and B_1 and then set $A_0 = B_0 = 1$ and again apply the four combinations to A_1 and B_1. These eight vectors include all eight inputs for the FA1 block i.e. simulation of these vectors will thoroughly check FAI.
- Table gives a set of eight vectors for verifying a 4-bit ripple-carry adder. Interestingly, the regular pattern in each vector allows us to expand the width of the vector, without increasing the number of vectors.



Vector no.	Bits: $C_0A_0B_0A_1B_1A_2B_2A_3B_3$	Input $C_nA_nB_n$ to FAn
1	000000000	000 applied to all FAs
2	001010101	001 applied to all FAs
3	010101010	010 applied to all FAs
4	011001100	011 applied to FA0, FA2 & 100 applied to FA1, FA3
5	100110011	100 applied to FA0, FA2 & 011 applied to FA1, FA3
6	101010101	101 applied to all FAs
7	110101010	110 applied to all FAs
8	111111111	111 applied to all FAs

Figure: Design verification of 4-bit RCA.

Simulation for Test Evaluation

- Alternative to true-value simulator (used to verify the design) is the fault simulator.
- Fault simulator has the capability to simulate the circuit response in the presence of faults.
- The fault simulation is typically done after design is verified and verified netlist along with test vectors are available.
- The fault simulator performs two functions:
    - Determines the coverage of a given set of input stimuli for a given fault model or a given fault list.
    - With the help of other programs (a test generator or a vector compacter), it can produce a set of vectors with a given fault coverage for manufacturing test.

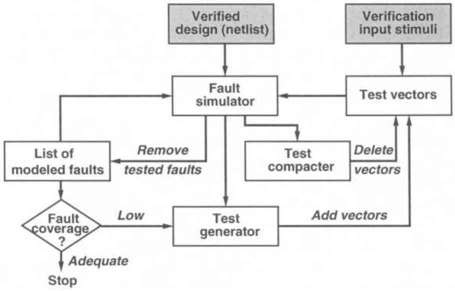


Figure: Fault simulation for test generation.

Fault Simulation

- If the fault list is not given, the simulator generates the one fault from the specified fault model.
- For example, it may make a list of all single stuck-at faults by using the equivalence fault collapsing.

The steps are as follows:

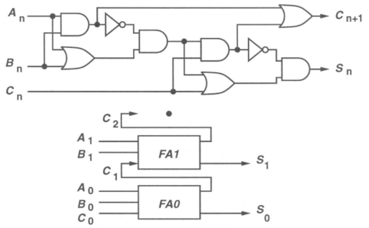
1. The simulator applies the given stimuli (verification vectors) and computes detected faults from each vectors.
2. The detected faults are deleted from the fault list.
3. Fault coverage is (ratio of detected faults to the initial given faults) is computed.
4. If sufficient fault coverage (98-100%) is achieved simulation can be stopped.
5. If there are unsimulated vectors remain in the list, they can be eliminated.
6. Whereas, the undetected faults may be given to the test generator to produce new tests.
7. The simulator can eliminate the test vectors that do not detect new faults while detects already covered fault. This results in test compaction.

Fault Simulation: Example

A four-bit ripple-carry adder circuit contains four full-adders. It has 36 logic gates, 9 primary inputs, and 5 primary outputs. A fault simulator makes a list of 186 single SA faults. The equivalence fault collapsing option in the simulator reduces it to 114 faults.

The result of fault simulation provides following information:

- The last vector does not detect any new fault. We can use first six vectors for testing circuit without any reduction in fault coverage. Even though the last two vectors were necessary for the design verification, they are found to be redundant for testing.
- Therefore, a fault simulator can significantly reduce the size of the vector set, thus reducing the time required for the manufacturing test.
- The collapsed fault set is about 40% smaller than the set of all faults. A 40-50% reduction in the fault list is quite typical.

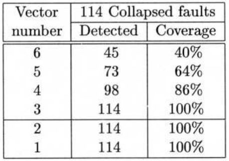


Vector number	186 Uncollapsed faults		114 Collapsed faults	
	Detected	Coverage	Detected	Coverage
1	61	33%	37	32%
2	113	61%	65	57%
3	125	67%	77	68%
4	143	77%	89	78%
5	162	87%	102	89%
6	186	100%	114	100%
7	186	100%	114	100%
8	186	100%	114	100%

Figure: Results of fault simulation.

Fault Simulation: Example (Cont...)

- Although vector 6 is essential as it detects some faults which are not be detected by all 5 previous vectors, it may possible that vector 6 could detect faults covered by some previous vectors making those vectors unnecessary.
- This could not be determined earlier because of fault dropping by the simulator. That is, a fault detected by a vector is immediately dropped from consideration and is not simulated for the following vectors. So, we can simulate vector 6 first.
- Similarly, we can argue that vector 5 could also detect faults covered by some earlier vectors. A simple way to find that out is to simulate the vectors in the reverse order.
- Reverse-order simulation is a simple technique for further compacting the vector set of a combinational circuit.
- We can now use four vectors above the line in this table for a 100% coverage. Unfortunately, the reverse-order simulation cannot be used for sequential circuits.
- Different algorithms are derived to reduce the number of test vectors.



Vector number	114 Collapsed faults	
	Detected	Coverage
6	45	40%
5	73	64%
4	98	86%
3	114	100%
2	114	100%
1	114	100%

Figure: Results of fault simulation.

Algorithm for Simulation

Algorithm for true-value simulation:

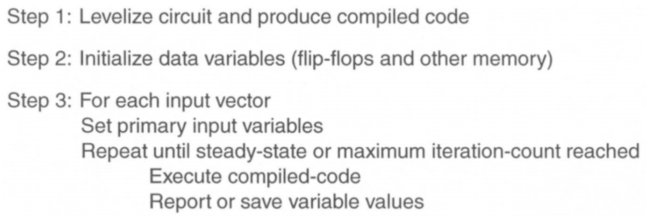
1. Compiled-code simulation
2. Event-driven simulation

Algorithm for Fault simulation:

1. Serial fault simulation
2. Parallel fault simulation
3. Deductive fault simulation
4. Concurrent fault simulation

Compiled-code simulation

- In this method, the circuit is described in language and levelized such that it can be compiled and executed on computer.
- Levelization ascertains that the evaluation of a signal is preceded by the evaluations of its sources.
- The signal are treated as variables in the code and can be typed (Boolean, integer, etc.) suitably.
- For every input vector, the code is repeatedly executed until all variables have attained steady values.



```
Step 1: Levelize circuit and produce compiled code

Step 2: Initialize data variables (flip-flops and other memory)

Step 3: For each input vector
            Set primary input variables
            Repeat until steady-state or maximum iteration-count reached
                    Execute compiled-code
                    Report or save variable values
```

Figure: Algorithm for concurrent fault simulation.

Compiled-code simulation (Cont...)

- Merits:
  - Compiled-code simulator is very effective where two-state (0,1) simulation is sufficient. For example, the combinational logic and sequential logic which is already initialized.
  - It is also very efficient for the high-level design verification where the circuit is described using functional modules and stimuli are generated by testbenches. Because the code execution can be very fast on a computer, such simulators are capable of high speed.
- Demerits:
  - Large gate-level circuits present several problems such as glitches, race conditions, etc. – are not modeled in a compiled-code simulator.
  - The second problem is that of the inefficiency incurred by the evaluation of the entire code when only a few signals may be changing. In digital circuits, generally only 1-10% of signals are found to change at any time.
  - Any detailed timing (such as multiple-delay or min-max-delay) is almost impossible to simulate in the compiled-code.
- Event-driven simulators run much faster at the gate-level.

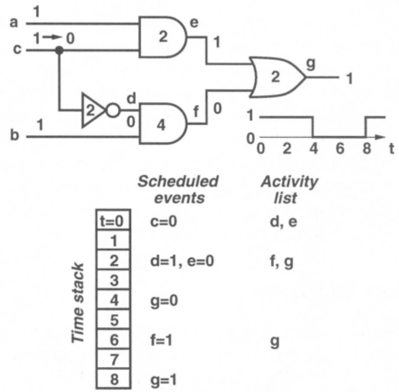
Event-Driven Simulation

- Event-driven simulation is a very effective procedure for discrete-event simulation.
- It is based on the recognition that any signal change (event) must have a cause, which is also an event.
- An event-driven simulator follows the path of events.
    1. First, consider a circuit at the gate-level with all signals are in steady-state.
    2. Change in some inputs, causing events on those input signals.
    3. Gates whose inputs now have events are called active and are placed in an activity list.
    4. Remove a gate from the activity list and evaluates its output (whether has an event or not).
    5. A changing output makes all fanout gates active, which are then added to activity list.
    6. The process of evaluation stops when the activity list becomes empty.
- An event-driven simulator only does the necessary amount of work.
- For logic circuits, in which typically very few signals change at a time, this can result in significant savings of computing effort.
- The biggest advantage of this technique is in its ability to simulate any arbitrary delays using procedure called as event scheduling.

Event-driven simulation: Example

Figure shows a combinational circuit where all gates are assumed to have fixed delays with same rise and fall delays and no interconnects delay. The signal values shown in the figure are the steady-state values for the initial vector. The simulation time is represented by discrete variable t. Attached to each time slot there is an event list of scheduled events and an activity list. Perform the simulation for an input event $1 \rightarrow 0$ on c occurring at $t = 0$.

- At $t = 0$, Add $c = 0$ to the event list and place fanouts d and e in the activity list.
- Remove d from activity list and evaluate it. Since the delay of the NOT gate is 2 units, add the new event $d = 1$ to the event list of time slot $t = 2$.
- Similarly, e is removed from the activity list and its evaluation produces an event $e = 0$ added to the event list of time slot $t = 2$, since the delay of gate e is also 2 units.
- This leaves the activity list empty, so we advance time.
- Repeat the process for each advanced time until no further events.



Time stack	Scheduled events	Activity list
t=0	c=0	d, e
1		
2	d=1, e=0	f, g
3		
4	g=0	
5		
6	f=1	g
7		
8	g=1	

Figure: Event driven simulation: An example

Event-driven simulation: Example

- In event-driven simulation, it can be observed that:
  - For any *current time*, all new events are scheduled only at future times.
  - Also, the difference between the current time and the farthest time at which we can schedule an event is limited by the maximum delay a gate has in the entire circuit.
- We can view the time array as a circular stack, often called the time wheel.
- The number of slots in the time wheel equals max +1, where max units is the largest delay experienced by any event (gate + interconnect) in the circuit.

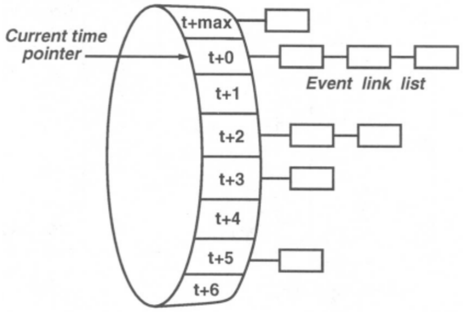


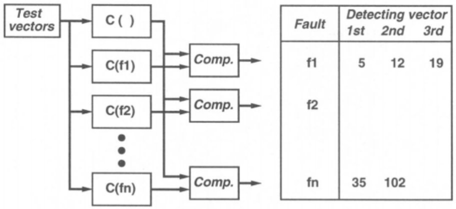
Figure: Time Wheel

True-Value Simulation: Summary

- Compiled-code simulation
  - Applicable to zero-delay combinational logic
  - Also used for cycle-accurate synchronous sequential circuits for logic verification
  - Efficient for highly active circuits, but inefficient for low-activity circuits
  - High-level (e.g., C language) models can be used
- Event-driven simulation
  - Only gates or modules with input events are evaluated (event means a signal change)
  - Delays can be accurately simulated for timing verification
  - Efficient for low-activity circuits
  - Can be extended for fault simulation

Fault Simulator

- A fault simulator classify the given target faults in a circuit as detected or undetected by the given stimuli.
- The block C( ) is the fault-free circuit and blocks C(f1) through C(fn) are copies of the same circuit with faults f1 through fn permanently inserted.
- The same vectors are applied to all blocks and the outputs of the faulty circuits are compared in the comparators.
- If any comparator shows a mismatch, the corresponding fault is noted as detected by the vector being simulated. Thus, the simulator records the numbers of vectors that detect a fault e.g. fault f1 is detected by vectors 5, 12, and 19.
- The simulator may also record the specific output at which a fault is detected.



Fault	Detecting vector 1st	2nd	3rd
f1	5	12	19
f2			
fn	35	102	

(a) Fault simulation. (b) Result.

Figure: Fault Simulator and Results

Fault Simulator

- When fault fn is detected for the first time, the simulation of block C(fn) is suspended beyond that vector, it is known as fault dropping.
- Fault dropping significantly speeds up the fault simulation process.
- The effort of simulating n faults is equivalent to either simulating a circuit that is n times larger, or repeating the original true-value simulation n times with small reduction due to fault dropping.
- The algorithms in the following subsections attempt to reduce this effort.
  - Serial Fault Simulation
  - Parallel Fault Simulation
  - Deductive Fault Simulation
  - Event Driven Simulation

Serial Fault Simulator

- It is the simplest algorithm for simulating faults.
- The step to simulate all faults are as follows:
  1. The circuit is first simulated in the true-value mode for all vectors and primary output values are saved in a file.
  2. Next, faulty circuits are simulated one by one by modifying the circuit description for a target fault and then using the true-value simulator.
  3. As the simulation proceeds, the output values of the faulty circuit are dynamically compared with the saved true responses.
  4. The simulation of a faulty circuit is stopped as soon as the comparison indicates detection of the target fault.
- All faults are simulated serially in this way.

Serial Fault Simulator (Cont...)

Merits:

- The implementation of a serial fault simulator is very simple as it repeatedly uses a true-value simulator.
- It can simulate any fault that can be introduced in the circuit description i.e. stuck-open types, bridging, delay, and analog faults can also be simulated.
- Analog circuit faults are usually simulated by this method.
- Also, a serial fault simulator can easily simulate all types of fault conditions, such as fault-induced races, hazards, loss of initialization, etc., in sequential circuits, which present difficulties to other types of simulators.

Demerits:

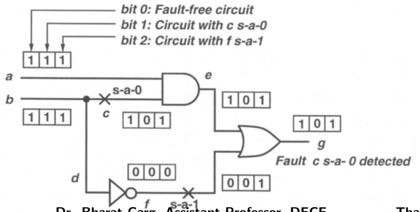
- High CPU Time: The CPU time of a serial simulator can be almost n times that of a true-value simulator for simulating n faults.
- Large Memory Requirement: Its memory requirement basically amounts to that of the true-value simulator.
- When fault-dropping is used, the CPU time can be significantly lower, especially if many faults are detected by a few earlier vectors in the set.
- However, there are more intelligent algorithms that reduce the effort of fault simulation.

Parallel Fault Simulator

- Parallel fault simulation is very effect when:
  - Circuit consists of only logic gates,
  - Signal take only binary, i.e., 0 and 1, values,
  - All gates exhibit the same delay, and
  - Only single stuck-at fault has to be simulated.
- The Parallel fault simulation uses the **bit-parallelism** of logical operations in a digital computer. Since the fault-free and faulty circuits are identical with similar gate delay.
- For a 32-bit machine word, an integer consists of a 32-bit binary vector. A logical AND or OR operation involving two words performs simultaneous AND or OR operations on all respective pairs of bits.
- This allows a simultaneous simulation of 32 circuits with identical connectivity, but with possibly different signal values.
- For a large number of faults, a parallel fault simulator will process w - 1 faults in one pass, where w bits is the machine word size.
- If no fault-dropping is used, a parallel fault simulator will run about w - 1 times faster than a serial fault simulator.
- In parallel fault simulator, all w - 1 faults must be detected before a pass can be terminated. Therefore, the serial fault simulator gains more by fault dropping.

Parallel Fault Simulator (Cont...)

- A circuit is being simulated for two faults: c stuck-at-0 and f stuck-at-1.
- Let, the computer has a three-bit word. To simulate the fault-free and two faulty circuits in parallel, the signal on each line is expressed as one word.
- The state of the left-most bit (bit 0) represents the signal value in the fault-free circuit, the middle bit (bit 1) that in the circuit with c s-a-0, and the right-most bit (bit 2) that in the circuit with f s-a-1. We apply a vector $a = 1,\ b = 1$.
- Since, the fault on c is present only in the first faulty circuit and so it affects the middle bit. Line f is affected by the s-a-1 fault will provide 001 output word.
- Output g is obtained by a bit-by-bit OR of $e = 101$ and $f = 001$. Thus, $g = 101$.
- Since the output of the circuit with c s-a-0 (middle bit) differs from that of the fault-free circuit (left bit.), that fault is detected. While other fault is not detected due to same value of respective bits.


bit 0: Fault-free circuit
bit 1: Circuit with c s-a-0
bit 2: Circuit with f s-a-1
1 1 1
a
b
1 1 1
s-a-0
c
1 0 1
e
1 0 1
1 0 1
g
Fault c s-a- 0 detected
d
0 0 0
0 0 1
f
s-a-1

Parallel Fault Simulator (Cont...)

Merits:

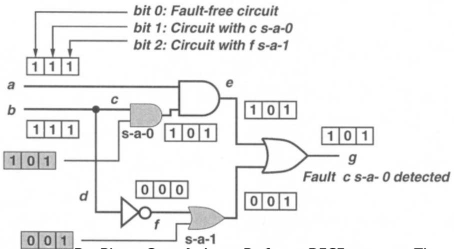
- Parallel fault simulators are implemented both as compiled-code or event-driven simulators.
- The algorithm can be extended for multiple-valued logic simulation. For example, four states (0, 1, X, and Z) can be encoded in two bits.
- A true-value logic simulator can be used as a parallel fault simulator by a simple fault injection method.

Demerits:

- A parallel fault simulator lacks the capability to model accurate rise and fall delays of signals.
- In a parallel simulator, combinational logic is modelled with either zero-delay or unit-delay.
- The simulator must contain special algorithms to deal with race and oscillation faults.

Fault injection for parallel fault simulation

- A stuck-at-0 fault is modelled by inserting a two-input AND gate at the fault site (Shaded AND gate).
- The other input of this gate is permanently set to 1, except for the bit position corresponding to the fault it models. For example, the middle bit (bit 1) is set to 0 to model the c s-a-0 fault.
- Similarly, the faulty circuit with f s-a-1 is modelled by inserting an OR gate in the signal f. The other input of this OR gate is set to 0, except for the right bit position (bit 2), which represents the modelled fault.
- Once these gates are inserted, simulation does not require any further consideration of faults, except the examination of the primary output bits for fault detection.
- This procedure can be extended to model a multiple fault.



Deductive Fault Simulation

- The circuit model used for this simulator is modeled with Boolean gates assuming either zero or unit delay for each gate, and signal states are binary (0,1) variables.
- In the deductive method, only the fault-free circuit is simulated. All signal values in each faulty circuit are deduced from the fault-free circuit values and the circuit structure.
- Since the circuit structure is the same for all faulty circuits, all deductions are carried out simultaneously.
- Deductive fault simulator **processes all faults in a single pass of true-value simulation** augmented with the deductive procedures.
- The deductive simulator provides a tremendous speed, but only when the modeling conditions can be satisfied.
- Variable (separate rise and fall) delays, multiple signal states, and transistor or functional models, though possible, require major changes in the implementation of a deductive fault simulator and slow down the execution.

Deductive Fault Simulation (Cont...)

- The simulation proceeds by simulating a vector in the true-value mode, which can be done by either a compiled-code or an event-driven mechanism.
- Before simulating the next vector, a deductive procedure is applied (fault lists are generated) to all lines in a level-order (for combinational logic) from inputs to outputs.
- The fault list of a signal is derived from the fault lists at the inputs of the gate and any faults associated with that gate.
- In a circuit with feedbacks, the fault lists of a signal may change several times. Only after the fault lists become stable, the simulator proceeds with the next vector.
- The fault list of a signal at any time during simulation contains the names of all faults in the circuit that can change the state of that line.

Deductive Fault Simulation (Cont...)

- An circuit having a fault a_k i.e. stuck-at-k, with k = 0 or 1 is shown in Figure.
- For the AND gate with primary $a = 0$ inputs and $b = 1$ the output is $c = 0$.
- Since a and b are primary inputs, their fault lists simply contain their own faults activated by the present signal values.
- We denote their fault lists as sets, $L_a = [a_1]$ and $L_b = [b_0]$. Since $b = 1$ the path from a to c is sensitized, but the path from b to c is not sensitized.
- Thus, fault list of c contains a_1 with fault s-a-1 on c (c_1) as the current state of c is 0.
- The fanouts d and e simply adopt the fault list from their source stem c and add their own respective faults.

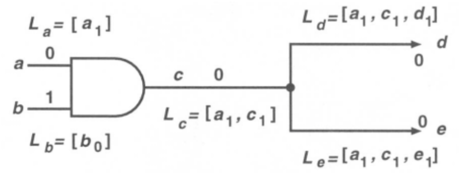
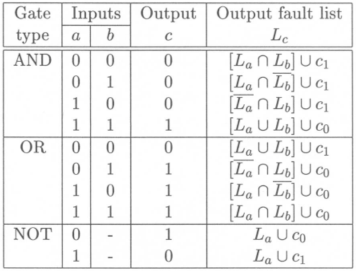


Figure: Fault list in deductive fault simulator.

Deductive Fault Simulation (Cont...)

- The output fault list of a gate is generated by set operations such as union ($\cup$), intersection ($\cap$), and complementation (-), among fault lists.
- For example, when both inputs of a two-input AND gate are 0, to propagate through, the effect of a fault must be present on both inputs. This is achieved by the intersection, $L_a \cap L_b$, of the two input fault lists.
- In addition, the fault c_1 is included.
- Table gives the rules for two-input primitive gates.
- For gates with more inputs and for other gates such as the exclusive-OR, propagation rules can be derived by expanding them in terms of two-input primitive gates.
- This technique has been used for deductive fault simulation of larger functions.



Gate type	Inputs a	b	Output c	Output fault list L_c
AND	0	0	0	$[L_a \cap L_b] \cup c_1$
	0	1	0	$[L_a \cap \overline{L_b}] \cup c_1$
	1	0	0	$[\overline{L_a} \cap L_b] \cup c_1$
	1	1	1	$[L_a \cup L_b] \cup c_0$
OR	0	0	0	$[L_a \cup L_b] \cup c_1$
	0	1	1	$[\overline{L_a} \cap L_b] \cup c_0$
	1	0	1	$[L_a \cap \overline{L_b}] \cup c_0$
	1	1	1	$[L_a \cap L_b] \cup c_0$
NOT	0	-	1	$L_a \cup c_0$
	1	-	0	$L_a \cup c_1$

Figure: Fault list propagation in deductive fault simulator.

Deductive Fault Simulation (Cont...)

- Consider the vector (1,1) applied to the circuit of Figure. We will simulate the s-a-0 and s-a-1 faults on all lines a through g for this vector.
- First, true-value simulation is carried out to determine all signal values that requires a single pass from inputs to the output.
- Next, we conduct a second input to output pass for fault list generation and propagation.
- The lists of primary inputs just contain the respective s-a-0 faults that are active there. Thus: $L_a = [a_0]$ and $L_b = [b_0]$
- The fault list for e is obtained by the propagation rules of Table

$$L_e = [L_a \cup L_c] \cup e_0 = [a_0, b_0, c_0, e_0]$$

- The fault list of d simply propagates through the NOT gate. Thus:

$$L_f = [L_d \cup f_1] = [b_0, d_0, f_1]$$

- Using the propagation rule for OR gate, we get: $L_g = [L_e \cap \bar{L_f}] \cup g_0 = [a_0, c_0, e_0, g_0]$

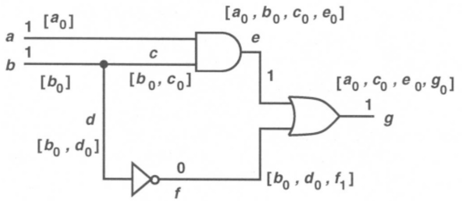


Figure: An example of deductive fault simulation

Deductive Fault Simulation: For Sequential Circuit

- The deductive method of fault simulation is very complex for sequential circuits due to:
  - Fault list propagation rules should be extended to deal with the three-state (0, 1, X) logic.
  - It should work out a method to deal with memory elements.
- A technique often used to simplify the simulation of logic circuits with asynchronous feedback, is to break all feedbacks by inserting ideal delay elements.

Concurrent Fault Simulation

It is the most general method of fault simulation. It can handle various types of circuit models, faults, signal states, and timing models. It is basically the efficient method of event-driven simulation.

A concurrent fault simulator models the problem as follows:

- An event on the fault-free circuit C( ) is called a good-event.
    - Three attributes specify a good-event: Line designation (signal name), type of change (e.g., $0 \rightarrow 1$, or any change among permissible signal states), and the time of change.
    - The same line also exists in faulty circuits, C(f1) through C(fn), and events on it in those circuits, only when they differ from the good-event, are called fault-events.
    - A fault event is specified by the same three attributes required for a good-event and an additional designation of the fault (site and type).
- The circuit C( ) is modelled in the same way as for the true-value simulation. The structure contains the connectivity information (fanouts and fanins) and gate functions. Each gate is called a *good-gate*.
    - A fault-list, usually in the form of a linked-list, is associated with each good-gate. Elements of this list are called *bad-gates*.
    - A bad-gate is not faulty itself but is affected by some fault. At least one of its signals at input or output terminals or internal states differs in value over the good-gate.
- Faults are assumed to be permanent and affect signal values at terminals or internal states (if any) of modules.
    - The fault information (fault site and type) is stored with the good-gate connected to the fault site.
    - **Whenever the signal values of a good-gate make a fault active, a bad-gate is inserted in the fault-list of that good-gate.**

Concurrent Fault Simulation (Cont...)

- The simulation proceeds exactly in the same manner as the event-driven simulation i.e. the circuit activity is simulated by event scheduling and processing. Usually unit-delay model is assumed, though other delays model can also be simulated.
- Whenever the signal value at a fault site differs from the faulty state, the fault becomes active. The good-gate attached to that site instantly inserts a bad-gate with the fault name and differing signal values into its fault-list.
- All good-events and fault-events make good-gates active for evaluation.
- Good-events also make bad-gates active for evaluation.
- All active gates, both good and bad, are evaluated and can generate new events.
- Good-events can be generated only when the activation of a good-gate is caused by a good-event.
- All events either caused by fault-events or caused by good-events activating bad-gates are fault-events.
- Upon evaluation, any bad-gate whose signals become identical to the corresponding good-gate is removed from the faultlist. This bad-gate is said to "converge" to the good-gate and the process is called convergence.
- The process of creating new bad-gates is called divergence.

Concurrent Fault Simulation (Cont...)

- Figure shows a three-gate combinational circuit simulated for an input vector 11.
- All single stuck-at faults are concurrently simulated. Subscript notation is used for faults *e.g.* a_0 represents s-a-0 on a.
- Faults are modeled on all gate inputs, primary inputs, and primary output.
- Good-gates are shown as gates drawn in solid lines. Bad-gates are shown in grey shade with the corresponding fault name written in the center.
- Signal values at inputs and output of each gate are written inside the gate.
- To each good-gate, a number of bad-gates are attached in a linked-list structure *e.g.* four bad-gates (a_0, b_0, c_0, e_0) are attached to good AND gate.
- At the primary output g, any bad-gate whose output differs from that of the good-gate indicates fault detection (*i.e.* faults a_0, c_0, e_0, g_0 are detected).

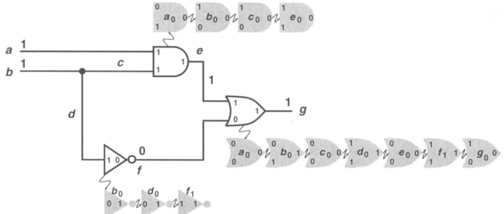


Figure: Fault lists (bad gates) in concurrent fault simulation

Concurrent Fault Simulation (Cont...)

- In Figure, we simulate a 1 to 0 ($1 \rightarrow 0$) good-event at a and examine the changes shown in the AND good-gate and its fault-list (associated bad-gates).
- The top input of all except the a_0 bad-gate change. Only the good-gate output changes, producing a $1 \rightarrow 0$ event on signal e.
- After these evaluations, bad-gates a_0 and e_0 have identical signal values as the good-gate and hence they converge. They are removed from the fault-list.
- At this point, one good-event $1 \rightarrow 0$ on e and no bad-events have been generated.
- The OR gate is evaluated and produces a $1 \rightarrow 0$ good-event on g. Bad-gates are also evaluated but none generates any bad-event.
- After evaluation, bad-gates $a_0, c_0, e_0,$ andg_0 converge to the good OR gate. These are removed from the fault-list. However, the processing of the ($1 \rightarrow 0$) good-event at a is not complete.

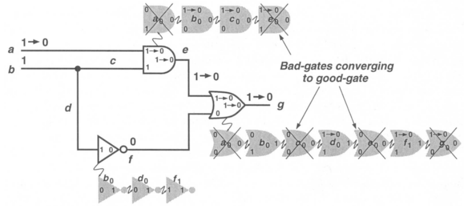


Figure: Fault lists (bad gates) in concurrent fault simulation

Concurrent Fault Simulation (Cont...)

- As Figure shows, changing of signal a activates fault a_1.
- A diverging bad-gate labeled a_1 is inserted in the fault-list of the AND gate. Newly diverging gates are shown with lighter grey shading in Figure.
- Similarly, another bad-gate e_1 is also added.
- These two generate bad-events, which when processed at the OR gate produce further divergence of bad-gates a_1 and e_1 there.
- The change caused by the good-event at the OR gate, discussed above, results in the divergence of another bad-gate e_0.
- All bad-gates at the output g have a different output value than that of the good-gate. Thus, detected faults are b_0, d_0, f_1, a_1, e_1, and g_1.

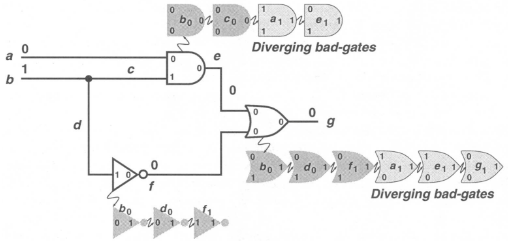


Figure: Bad-gate divergence in concurrent fault simulation.

Complexity Analysis of Fault Simulation

- The complexity of true-value simulation per vector is O(G), where G is the number of gates in the circuit because for each input vector every gate is evaluated no more than once.
- For event-driven simulation, the number of active gates per vector is still proportional to the total number of gates.
- If we assume that the number of vectors needed to verify or test a circuit might increase in proportion to the number of gates, then the total complexity for all vectors will be $O(G^2)$.
- For fault simulation, in the worst case, true-value simulation is repeated for each fault. Therefore, the worst case complexity of fault simulation is $O(G^3)$.
- Fault dropping in serial fault simulation, or the use of clever methods such as parallel, deductive, concurrent, or differential, can save time but cannot do better than the true-value simulation.
- Therefore, the complexity of fault simulation of a G gate VLSI circuit is between $O(G^2)$ and $O(G^3)$.
- Fault sampling is a popular technique used to reduce the effort of fault simulation.
- In this technique, a subset of faults is randomly picked from the set of all faults. This subset, usually a small fraction of the complete fault set, is known as a fault sample or simply a sample.

Problems for Practice

1. Show that when n faults are simulated without fault dropping, a parallel fault simulator on a w-bit word computer will run $(n+1)(w-1)/n$ times faster than a serial fault simulator.
2. Derive the output fault list L_c for an exclusive-OR gate, $c = a \oplus b$ in terms of the input fault lists, L_a and L_b.
3. A VLSI design center is equipped to design digital circuits with embedded memory blocks. The circuits are modeled at the logic level but the simulator must have a four-state $(0, 1, X, Z)$ signal representation. Only single stuck-at faults are to be simulated. In order to select a fault simulator, you must answer the following questions:
    1. Which of the simulators among serial, parallel, deductive, and concurrent, can simulate the circuits?
    2. Which is the best choice among these four simulators?
4. What can you say about the detectability of a multiple stuck-at fault $(f1, f2)$, if concurrent simulation of single stuck-at faults, $f1$ and $f2$, produces the following result?
    1. Both faults are detected.
    2. Only f1 is detected.
    3. None is detected.

Testability Measures

- Testability is the property of a circuit that makes it easy to test.
- The controllability and observability of signals are commonly used metrics to measure the testability of a circuit.
- Controllability for a digital circuit is defined as the difficulty of setting a particular logic signal to a 0 or a 1.
- Observability for a digital circuit is defined as the difficulty of observing the state of a logic signal.
- These measures are important for circuit testing, other methods of observing the internal signals of a circuit are very expensive. For example
    - Electron beam testing, can scan the VLSI chip-under-test and produce a picture of the chip layout.
    - The signals at logic 0 will appear in one color in the image, and those charged to logic 1 appear as another color.
    - However, this testing method is used only for specialized purposes, because of its very high cost.
- Therefore, we must set internal signals by setting primary inputs (PIs), and observe internal signals by propagating their values to primary outputs (POs).

Testability Measures (Cont...)

- The controllability and observability measures are useful because they approximately quantify how hard it is to set and observe internal signals of a circuit.
- Testability analysis usually has two significant attributes:
  - It involves circuit topological analysis, but no test vectors. It is a static type of analysis.
  - It has linear complexity, because otherwise testability analysis is pointless and one might as well use automatic test-pattern generation (ATPG) or fault simulation.
- **Goldstein** developed the SCOAP[1] testability measures, and presented a linear complexity algorithm to compute them.
- Separate 0 and 1 controllabilities for each signal as measures of the effort (or difficulty) of setting the line to a logic 0 or 1 value are defined.
- The SCOAP has significant inaccuracies due to the assumption that signals at reconvergent fanout stems are treated as independent, when they frequently are not.
- It is highly effective in predicting relative coverage levels of the entire circuit fault sets. The numerical values of the SCOAP measures range between zero and infinity.
- Higher values point to testability problems.

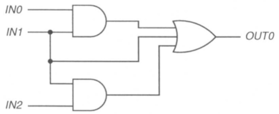


Figure: Reconverging fanouts causing correlation among signals.

Testability Measures (Cont...)

- Another type of testability measure is probability-based.
- These overcome some limitations of SCOAP. Being related to signal probabilities, they are easier to interpret and their values always range between 0 and 1.
- The 1-controllability (C1) is the probability of a signal value on line l being set to 1 by a random vector.
- The 0-controllability (C0) is the probability of a signal value on line l being set to 0 by a random vector.
- Seth and Agrawal developed PREDICT, a numerical technique for this purpose [590]. They break the circuit into partitions, known as supergates, which completely include reconvergent fanouts. However, their algorithm has exponential computation cost in the number of reconvergent fanouts in each supergate.

SCOAP Controllability and Observability

- SCOAP consists of six numerical measures for each signal (l) in the circuit:
  1. Combinational 0-controllability, CC0(l): Signifies the difficulty of setting value to 0.
  2. Combinational 1-controllability, CC1(l)
  3. Combinational observability, CO(l)
  4. Sequential 0-controllability, SC0(l)
  5. Sequential 1-controllability, SC1(l)
  6. Sequential observability, SO (l)
- The controllabilities range between 1 and ∞, and observabilities lie between 0 and ∞.
- The higher the measures for a line, the more difficult it will be to control or observe.

Method of Calculating Controllabilities

The method of calculating controllabilities is as follows:

- First, set the difficulty of controlling each primary input (PI) to 0 (called CC0) to the value 1 and the difficulty of controlling each PI to 1 (called CC1) to the value 1.
- Progress through the circuit in a forward pass, in level order (level of a logic gate is the maximum of the distances (in logic gates) of its various inputs from the PIs).
- **For each traversed logic gate, add 1 to the controllability.** This accounts for the logic depth.
- If a logic gate output is produced by setting only one input to a controlling value, then:

$$\boxed{Output\ controllability = min(Input\ controllabilities) + 1}$$

- If a logic gate output can only be produced by setting all inputs to a non-controlling value, then:

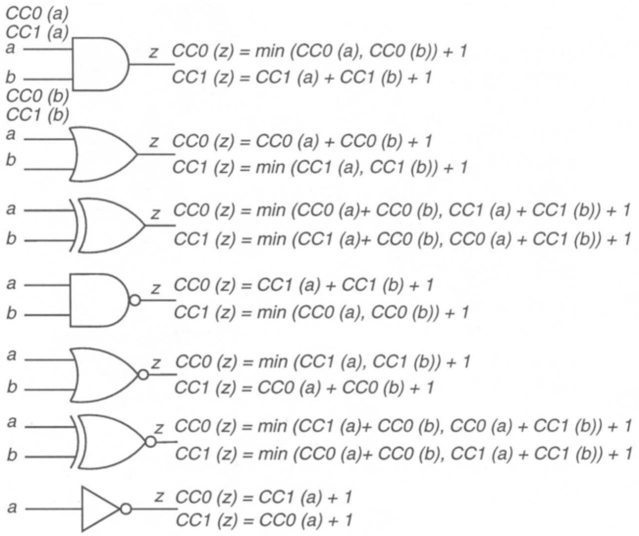
$$\boxed{Output\ controllability = \sum(Input\ controllabilities) + 1}$$

- If an output can be controlled by multiple input sets (e.g., a two-input XOR gate where "01" and "10" input sets will both cause a 1 output), then;

$$\boxed{Output\ controllability = min(controllabilities\ of\ input\ sets) + 1}$$

Method of Calculating Controllabilities

- Figure shows the output controllability calculation for the basic digital logic gates.



CC0 (a)
CC1 (a)
a
b
CC0 (b)
CC1 (b)
z
CC0 (z) = min (CC0 (a), CC0 (b)) + 1
CC1 (z) = CC1 (a) + CC1 (b) + 1
CC0 (z) = CC0 (a) + CC0 (b) + 1
CC1 (z) = min (CC1 (a), CC1 (b)) + 1
CC0 (z) = min (CC0 (a)+ CC0 (b), CC1 (a) + CC1 (b)) + 1
CC1 (z) = min (CC1 (a)+ CC0 (b), CC0 (a) + CC1 (b)) + 1
CC0 (z) = CC1 (a) + CC1 (b) + 1
CC1 (z) = min (CC0 (a), CC0 (b)) + 1
CC0 (z) = min (CC1 (a), CC1 (b)) + 1
CC1 (z) = CC0 (a) + CC0 (b) + 1
CC0 (z) = min (CC1 (a)+ CC0 (b), CC0 (a) + CC1 (b)) + 1
CC1 (z) = min (CC0 (a)+ CC0 (b), CC1 (a) + CC1 (b)) + 1
CC0 (z) = CC1 (a) + 1
CC1 (z) = CC0 (a) + 1

Figure: SCOAP controllability calculation.

Calculating Observabilities

- The observabilities are computed in a reverse pass starting from primary outputs (POs) and moving backwards to the PIs once all controllabilities are evaluated.
- First set the output observability difficulty (called CO) to 0 for both logic 0 and 1.
- **The observability of input node is equal to the observability of the output plus the difficulty of setting all other inputs to non-controlling values, plus 1 to account for the logic depth.**
- For instance, the difficulty of observing the top input a of the AND gate in Figure becomes $CO(Z) + CC1(Z) + 1$.

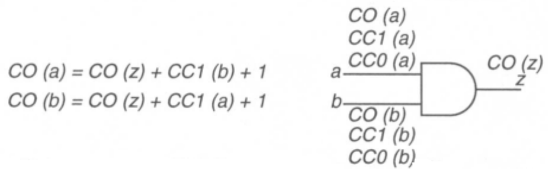


Figure: SCOAP Observability calculation.

Calculating Observabilities (Cont...)

- Figure shows the observability calculation for all of the basic logic gates.

$$CO(a) = CO(z) + CC1(b) + 1$$
$$CO(b) = CO(z) + CC1(a) + 1$$

$$CO(a) = CO(z) + CC0(b) + 1$$
$$CO(b) = CO(z) + CC0(a) + 1$$

$$CO(a) = CO(z) + \min(CC0(b), CC1(b)) + 1$$
$$CO(b) = CO(z) + \min(CC0(a), CC1(a)) + 1$$

$$CO(a) = CO(z) + CC1(b) + 1$$
$$CO(b) = CO(z) + CC1(a) + 1$$

$$CO(a) = CO(z) + CC0(b) + 1$$
$$CO(b) = CO(z) + CC0(a) + 1$$

$$CO(a) = CO(z) + \min(CC0(b), CC1(b)) + 1$$
$$CO(b) = CO(z) + \min(CC0(a), CC1(a)) + 1$$

$$CO(a) = CO(z) + 1$$

$$CO(a) = \min(CO(z1), CO(z2), \ldots, CO(zn))$$

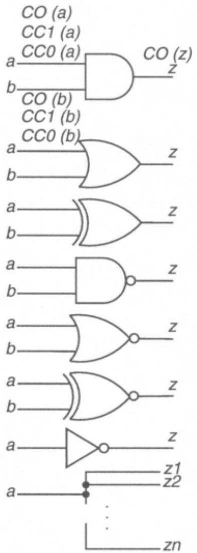


Figure: SCOAP Observability calculation.

Error in Testability Measures

- The error arises in the controllability calculation due to reconvergent fanout where the reconverging signals may correlate, and therefore the controllability becomes inaccurate at the reconvergence point.
- The necessary conditions for controlling, observing, and testing faults on fanout branches frequently differ from the corresponding conditions for the stem.
- One might be inclined to bound this stem observability between min (all fanout branch observabilities) and max (all fanout branch observabilities).
    - A min function is used because we assume that the events of observing a signal through each fanout stem branch are independent, and therefore we observe through the branch with the lowest observability difficulty.
    - A max function is used because we assume that the events of observing a signal through each fanout stem branch are totally dependent, and therefore the branch that is hardest to observe gives the correct observability.
- Goldstein uses the following approximation:

$$\boxed{CO(stem) = min(CO(branches))}$$

Accuracy Problem in Observability of Fanout Branches

- A fanout stem is a signal (a logic gate output or a PI) that branches to many different places, each of which is called a fanout branch.
- The necessary conditions for controlling, observing, and testing faults on fanout branches frequently differ from the corresponding conditions for the stem.
- One can bound the stem observability between min (all fanout branch observabilities) and max (all fanout branch observabilities).
- A min function is used where the events of observing a signal through each fanout stem branch are independent, and therefore we observe through the branch with the lowest observability difficulty.
- A max function is used where the events of observing a signal through each fanout stem branch are totally dependent, and therefore the branch that is hardest to observe gives the correct observability.
- These two scenarios ignore the reality that a signal may need to be simultaneously propagated through some or all fanout branches to make it observable.
- If that were the case, then the stem observability becomes:

$$\sum \text{ or } min(\text{some or all fanout branches observabilities})$$

- Finally, Goldstein uses the following approximation:

$$CO(\text{stem}) = min(CO(\text{branches}))$$

Combinational Circuit Example

- Consider the sequential circuit in Figure. Assume that the FFs in the circuit have special testing hardware attached to them so that it is possible to read out the present state, and also it is possible to set the present state of the FFs.
- Therefore, the FFs can be modeled as a pair of primary input, primary output pairs.
- The D line is referred to as a pseudo-primary output, (PPO), since it can be observed by the flip-flop testing hardware, whereas the Q line is referred to as a pseudo-primary input (PPI), since it can be set by the flip-flop testing hardware.
- Figure (b) shows the circuit with FFs replaced by PPI-PPO pairs.
  - FF 7 is replaced by a PPI for the Q1 line (called PPI7) and a PPO for the D line (called PPO7).
  - Whereas, FF 8 is replaced by PPI8 and PPO8 pairs.

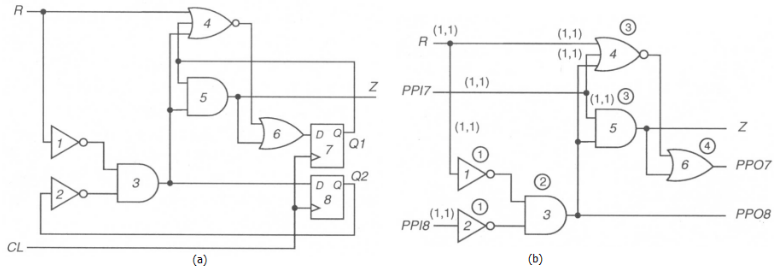


Figure: SCOAP Observability calculation

Levelization Algorithm

Label the gates in level order

- Represent maximum distance (in gates) from a PI. Each gate encountered along any path from a PI to a PO is labeled with a level number giving its maximum distance (in gates) from a PI.

Levelization algorithm: Level number from PIs to POs.

1. Assign level number 0 to all primary inputs.
2. For each PI fanout:
    - Label that circuit line with the level number of the PI, and
    - Queue the logic gate driven by that fanout line.
3. While the queue is not empty:
    - Dequeue the next logic gate in the queue.
    - If all of the gate fanins are labeled with level numbers, then label the logic gate and its fanouts with the maximum of its input level numbers + 1 and queue all fanouts of the logic gate. Otherwise, requeue the logic gate.

Combinational Circuit Example: Computing Controllability

- Each line is labeled with a (CC0, CC1) pair to show its controllabilities.
- First assign (1, 1) to all PIs: R, PPI7, and PPI8.
- Next, these ordered pairs are propagated to all fanout branchs of the PIs, leading to the labeling of the top two inputs of gate 4, the top input of gate 5, and the inputs to INVERTERS 1 and 2 with (1,1).
- Compute controllability for the inverter using:

  $$CC1(OUTPUT) = CCO(INPUT) + 1$$

  and

  $$CC0(OUTPUT) = CC1(INPUT) + 1$$

- This leads to the outputs of INVERTERS 1 and 2 being labeled with (2,2), and both inputs of logic gate 3 being labeled with (2,2).
- Next, process AND gate 3:

  $$CC0(3) = min(CC0(1), CC0(2)) + 1 = min(2, 2) + 1 = 3$$
  $$CC1(3) = CC1(1) + CC1(2) + 1 = 2 + 2 + 1 = 5$$

- For NOR gate 4:

  $$CC0(4) = min(CC1(R), CC1(PPI7), CC1(3)) + 1 = min(1, 1, 5) + 1 = 2$$
  $$CC1(4) = CC0(R) + CC0(PPI7) + CC0(3) + 1 = 1 + 1 + 3 + 1 = 6$$

Combinational Circuit Example: Computing Controllability

- For AND gate 5:

$$CC0(5) = min(CC0(PPI7), CC0(3)) + 1 = min(1,3) + 1 = 2$$
$$CC1(5) = CC1(PPI7) + CC1(3) + 1 = 1 + 5 + 1 = 7$$

- For OR gate 6:

$$CC0(6) = CC0(4) + CC0(5) + 1 = 2 + 2 + 1 = 5$$
$$CC1(6) = min(CC1(4), CC1(5)) + 1 = min(6,7) + 1 = 7$$

- The completely labeled circuit with the final controllabilities is shown below.

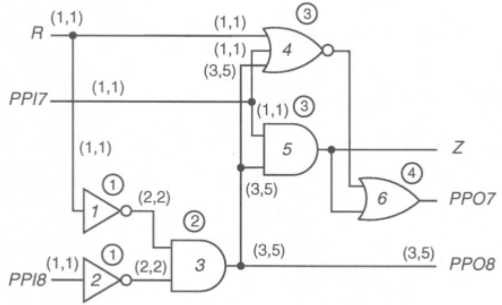


Figure: SCOAP Observability calculation.

Combinational Circuit Example: Computing Observability

- Renumber the circuit level numbers from POs backwards to PIs, so that each gate is labeled with the maximum distance in logic gates of any of its fanouts from a PO.
- Figure shows all gate level numbers in square boxes.
- We process logic gates in level order from POs backwards to PIs.
- Assign 0 to PO Z and PPOs *e.g*. PPO7 and PPO8 (boldface after the (CC0, CC1)). This causes observability 0 to be assigned to gate 6, as well.
- We cannot yet assign observabilities to gates 5 and 3, because they are fanout stems, and not all fanout branch observabilities are known.

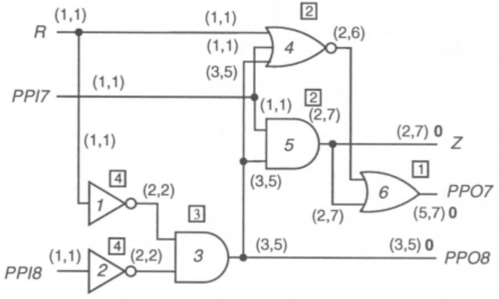


Figure: SCOAP Observability calculation.

Combinational Circuit Example: Computing Observability

- We process OR gate 6 at level 1:

$$CO(4) = CC0(5) + CO(6) + 1 = 2 + 0 + 1 = 3$$
$$CO(5) = CC0(5) + C0(6) + 1 = 2 + 0 + 1 = 3$$

- Next, process NOR gate 4 at level 2:

$$CO(R) = CC0(PPI7) + CC0(3) + CO(4) + 1 = 1 + 3 + 3 + 1 = 8$$
$$CO(R) = CC0(R) + CC0(3) + CO(4) + 1 = 1 + 3 + 3 + 1 = 8$$
$$CO(R) = CC0(R) + CC0(PPI7) + CO(4) + 1 = 1 + 1 + 3 + 1 = 6$$

- Since its stem observability is the minimum of all branch observabilities:

$$CO(5) = min(0, 3) = 0$$

- For a two-input AND gate:

$$CO(PPI7) = CC1(3) + CO(5) + 1 = 5 + 0 + 1 = 6$$
$$CO(3) = CC1(PPI7) + C0(5) + 1 = 1 + 0 + 1 = 2$$

- Process AND gate 3 at level 3:

$$CO(1) = CC1(2) + CO(3) + 1 = 2 + 0 + 1 = 3$$
$$CO(2) = CC1(1) + C0(3) + 1 = 2 + 0 + 1 = 3$$

Combinational Circuit Example: Computing Observability

- Lastly process the two INVERTERS at level 4:

$$CO(R) = CO(1) + 1 = 3 + 1 = 4$$

$$CO(PPI8) = CO(2) + 1 = 3 + 1 = 4$$

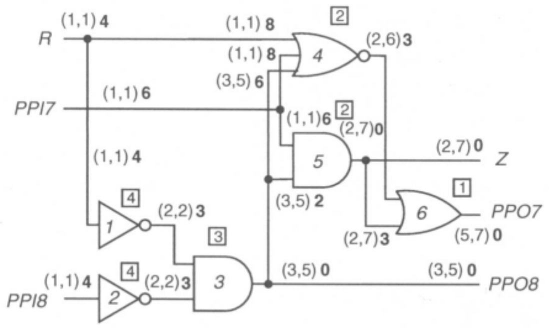


Figure: SCOAP Observability calculation.

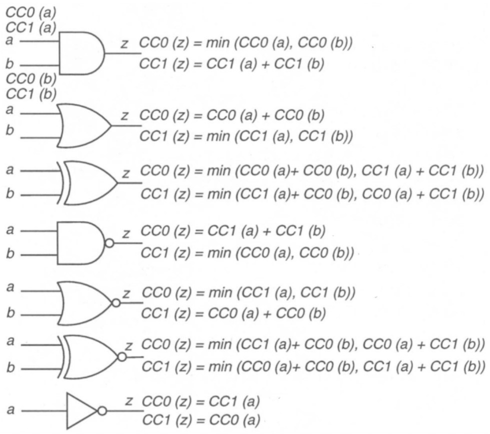
Note: The computational complexity of this algorithm is $O(2n) = O(n)$, where n is the number of gates.

Sequential SCOAP Measures

- There are two main differences in the sequential measures from the combinational controllability and observability measures:
  1. Increment the sequential measure by 1 only when signals propagate from flip-flop inputs to Q or Q outputs, or from flip-flop outputs backwards to D, C (clock), SET, or RESET inputs.
  2. Iterate in calculating controllability numbers in sequential circuits because of feedback loops involving flip-flops.
- Sequential controllabilities SC0 and SC1 roughly measure the number of times various flip-flops must be clocked to control a signal.
  - For example: if a given line l can only be set to 1 by clocking flip-flop a twice and flip-flop b three times, then we would expect $SC(l) = 5$.
- Sequential observability SO measures the number of times various flip-flops must be clocked to observe a signal.
- In a sequential circuit, the CC and CO roughly measure the number of lines that must be set, over all of the required clock periods, to control or observe a signal.
- The sequential CC and CO equations for basic logic gates differ from the equations for combinational gates only in that a 1 is not added as we move from one level of logic to another, but rather a 1 is added when a signal passes through a flip-flop.

Sequential SCOAP Measures

The procedure to measure CC and CO for sequential circuit is given below



CC0 (z) = min (CC0 (a), CC0 (b))
CC1 (z) = CC1 (a) + CC1 (b)

CC0 (z) = CC0 (a) + CC0 (b)
CC1 (z) = min (CC1 (a), CC1 (b))

CC0 (z) = min (CC0 (a)+ CC0 (b), CC1 (a) + CC1 (b))
CC1 (z) = min (CC1 (a)+ CC0 (b), CC0 (a) + CC1 (b))

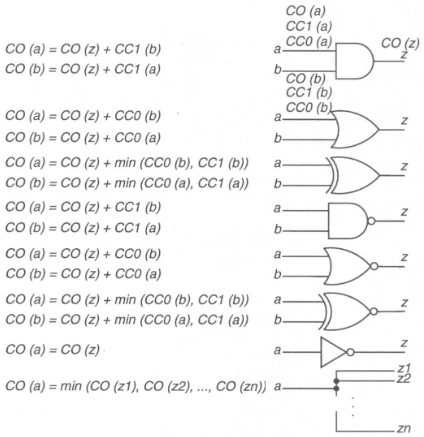
CC0 (z) = CC1 (a) + CC1 (b)
CC1 (z) = min (CC0 (a), CC0 (b))

CC0 (z) = min (CC1 (a), CC1 (b))
CC1 (z) = CC0 (a) + CC0 (b)

CC0 (z) = min (CC1 (a)+ CC0 (b), CC0 (a) + CC1 (b))
CC1 (z) = min (CC0 (a)+ CC0 (b), CC1 (a) + CC1 (b))

CC0 (z) = CC1 (a)
CC1 (z) = CC0 (a)

Figure: SCOAP Controllability calculation.



CO (a) = CO (z) + CC1 (b)
CO (b) = CO (z) + CC1 (a)

CO (a) = CO (z) + CC0 (b)
CO (b) = CO (z) + CC0 (a)

CO (a) = CO (z) + min (CC0 (b), CC1 (b))
CO (b) = CO (z) + min (CC0 (a), CC1 (a))

CO (a) = CO (z) + CC1 (b)
CO (b) = CO (z) + CC1 (a)

CO (a) = CO (z) + CC0 (b)
CO (b) = CO (z) + CC0 (a)

CO (a) = CO (z) + min (CC0 (b), CC1 (b))
CO (b) = CO (z) + min (CC0 (a), CC1 (a))

CO (a) = CO (z)

CO (a) = min (CO (z1), CO (z2), ..., CO (zn))

Figure: SCOAP Observability calculation.

Sequential SCOAP Measures

- To control the Q line to 1: Set D to 1, provide a falling clock (C) edge (first a 1 and then a 0), and control the RESET line to 0.
  - The combinational and sequential difficulties of controlling Q to a 1 are:

$$CC1(Q) = CC1(D) + CC1(C) + CC0(C) + CC0(RESET)$$

$$SC1(Q) = SC1(D) + SC1(C) + SC0(C) + SC0(RESET) + 1$$

  - CC1 measures how many lines in the circuit must be set to make Q as 1, whereas SC1 measures how many flip-flops in the circuit must be clocked to set Q to 1.
- Q can be set to 0 can either use the RESET line and apply a falling edge to clock C, or by clocking a 0 into Q through the D line.

$$CC0(Q) = min(CC1(RESET) + CC1(C) + CC0(C), CC0(D) + CC1(C) + CC0(C))$$

$$SC0(Q) = min(SC1(RESET) + SC1(C) + SC0(C), SC0(D) + SC1(C) + SC0(C)) + 1$$

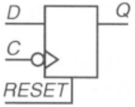


Figure: Synchronously resettable -ve edge triggered flip-flop.

Sequential SCOAP Measures

- The D line can be observed at Q by holding RESET low and generating a falling edge on the clock line C:

$$CO(D) = CO(Q) + CC1(C) + CC0(C) + CC0(RESET)$$
$$SO(D) = SO(Q) + SC1(C) + SC0(C) + SC0(RESET) + 1$$

- RESET can be observed by setting Q to a 1 and using RESET:

$$CO(RESET) = CO(Q) + CC1(Q) + CC1(C) + CC0(C) + CC1(RESET)$$
$$SO(RESET) = SO(Q) + SC1(Q) + SC1(C) + SC0(C) + SC1(RESET) + 1$$

- Three ways to indirectly observe the clock line C:
  - Set Q to 1 and clock in a 0 from D,
  - Set Q to 1 and synchronously apply RESET, or
  - Set Q to 0 and clock in a 1 from D while holding RESET to 0.

$$\begin{aligned} CO(C) = min[&CO(Q) + CC1(Q) + CC0(D) + CC1(C) + CC0(C), \\ &CO(Q) + CC1(Q) + CC1(RESET) + CC1(C) + CC0(C), \\ &CO(Q) + CC0(Q) + CC0(RESET) + CC1(D) + CC1(C) + CC0(C)] \end{aligned} \tag{1}$$

$$\begin{aligned} SO(C) = min[&SO(Q) + SC1(Q) + SC0(D) + SC1(C) + SC0(C), \\ &SO(Q) + SC1(Q) + SC1(RESET) + SC1(C) + SC0(C), \\ &SO(Q) + SC0(Q) + SC0(RESET) + SC1(D) + SC1(C) + SC0(C)] + 1 \end{aligned} \tag{2}$$

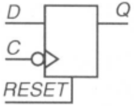


Figure: Synchronously resettable -ve edge triggered flip-flop.

Algorithm to Set both Combinational and Sequential Measures

1. For all PIs I, set $CC0(I) = CC1(I) = 1$ while $SC0(I) = SC1(I) = 0$.
2. For all other nodes N, set $CC0(N) = CC1(N) = \infty$ and $SC0(N) = SC1(N) = \infty$.
3. Working from PIs to POs, use the CC0, CC1, SC0, and SC1 equations to map logic gate and flip-flop input controllabilities into output controllabilities. Iterate until the controllability numbers stabilize in feedback loops.
4. For all POs U, set $CO(U) = SO(U) = 0$
5. For all other nodes N set $CO(N) = SO(N) = \infty$
6. Working from POs to PIs, use the CO and SO equations and the pre-computed controllabilities to map output node observabilities of gates and flip-flops into input observabilities.
7. For fanout stems Z with branches $Z1, Z2, \cdots ZN$,
   $SO(Z) = min(SO(Z1), \cdots SO(ZN))$ and $CO(Z) = min(CO(Z1), \cdots CO(ZN))$.
8. If any node remains with:
   - $CC0(SC0) = \infty$, then that node is 0-uncontrollable.
   - $CC1(SC1) = \infty$ then that node is 1-uncontrollable.
   - $CO = \infty$ or $SO = \infty$ then that node is unobservable.

Sequential Testability Measures: Example

- In this example, CC0, CC1, and CO are shown as (CC0,CC1)CO and SC0, SC1, and
- SO are shown as [SC0, SC1]SO below the combinational measures.
- The observabilities are always shown in **bold** faces.
- Forward level numbers for logic gates and FFs are shown circled over the gates.
- The primary inputs and primary outputs and internal nodes are initialized to appropriate value.
  - Signals R and CL, and all of their fanouts, are set to $(CC0, CC1) = (1, 1)$ and $[SC0, SC1] = [0, 0]$.
  - All other nodes are set to $(CC0, CC1) = [SC0, SC1] = [\infty, \infty]$.

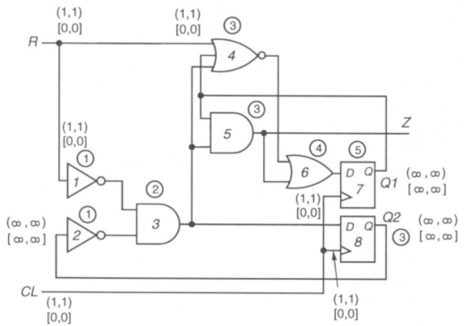
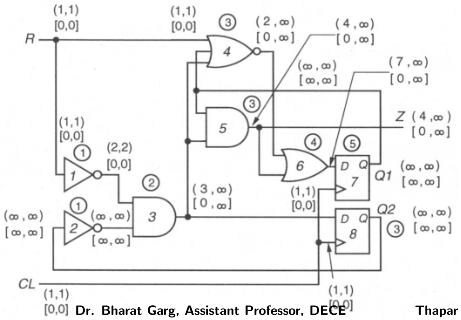


Figure: Example for computing Testability measures

Sequential Testability Measures: Example

- Start computation of controllabilities from input to output. The controllabilities of output of INVERTER 1 is: $(CC0, CC1) = (2, 2)$ and $[SC0, SC1] = [0, 0]$
- INVERTER 2 provides infinite value due feedback loop: $(CC0, CC1) = (\infty, \infty)$ and $[SC0, SC1] = [\infty, \infty]$
- For AND gate 3: $(CC0, CC1) = (3, \infty)$ and $[SC0, SC1] = [0, \infty]$
- For NOR gate 4:
  $CC0(4) = min(CC1(R) + CC1(Q1) + CC1(3)) + 1 = min(1, \infty, \infty) + 1 = 2$
  $(CC0, CC1) = (2, \infty)$ and $[SC0, SC1] = [0, \infty]$
- Similarly, the controllability for gate 5 and gate 6 are calculated as shown.



Sequential Testability Measures: Example

- For flip-flop 7: $CC0(7) = CC0(6) + CC1(CL) + CC0(CL) = 7 + 1 + 1 = 9$
  $(CC0, CC1) = (9, \infty)$ and $[SC0, SC1] = [1, \infty]$
- For flip-flop 8: $(CC0, CC1) = (5, \infty)$ and $[SC0, SC1] = [1, \infty]$
- The process it repeated and controllability is measured.
- The figure shows the controllabilities after two iterations.

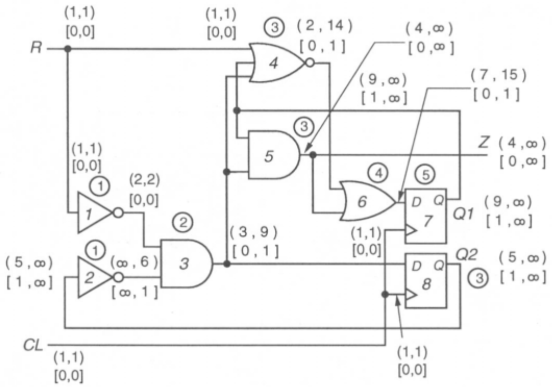


Figure: Example for computing Testability measures.

Sequential Testability Measures: Example

- Figure shows the situation after three iterations.
- The circuit requires three iteration to stabilize the values.

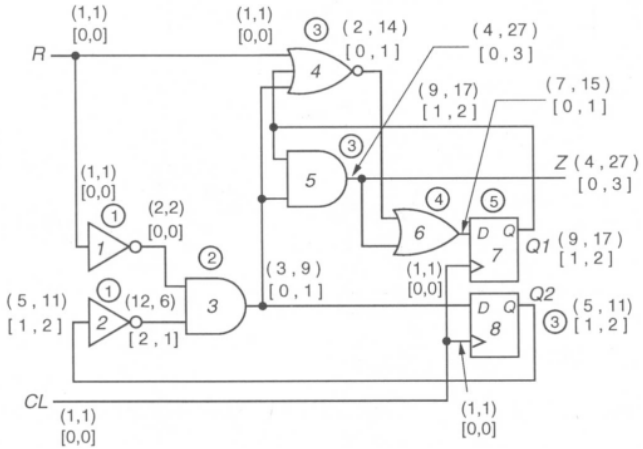


Figure: Sequential circuit measures after stabilization.

Sequential Testability Measures: Example

- The only difficult step is computing CO from line CL to 7 which is

$$CO = CO(Q) + CC1(CL) + CC0(CL) + CC0(D) + CC1(Q) = 10 + 1 + 1 + 7 + 17 = 36$$
$$SO = SO(Q) + SC1(CL) + SC0(CL) + SC0(D) + SC1(Q) + 1 = 1 + 0 + 0 + 0 + 2 + 1 = 4$$

- Similarly, for line 8:

$$CO = CO(Q) + CC1(CL) + CC0(CL) + CC0(D) + CC1(Q) = 22 + 1 + 1 + 1 + 13 = 38$$
$$SO = SO(Q) + SC1(CL) + SC0(CL) + SC0(D) + SC1(Q) + 1 = 2 + 0 + 0 + 0 + 2 + 1 = 5$$

- Input line CL: $CO(CL) = min(36, 38) = 36$ while $SO(CL) = min(4, 5) = 4$

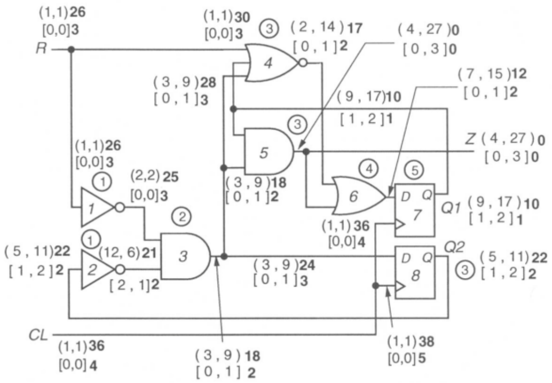
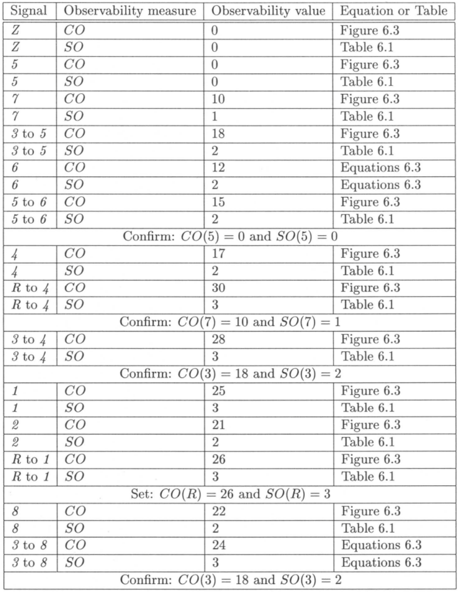


Figure: Sequential circuit measures after stabilization.

Observability Metrics

- Table shows the observability of all the gates.



Signal	Observability measure	Observability value	Equation or Table
Z	*CO*	0	Figure 6.3
Z	*SO*	0	Table 6.1
5	*CO*	0	Figure 6.3
5	*SO*	0	Table 6.1
7	*CO*	10	Figure 6.3
7	*SO*	1	Table 6.1
3 to *5*	*CO*	18	Figure 6.3
3 to *5*	*SO*	2	Table 6.1
6	*CO*	12	Equations 6.3
6	*SO*	2	Equations 6.3
5 to *6*	*CO*	15	Figure 6.3
5 to *6*	*SO*	2	Table 6.1
Confirm: $CO(5) = 0$ and $SO(5) = 0$			
4	*CO*	17	Figure 6.3
4	*SO*	2	Table 6.1
R to *4*	*CO*	30	Figure 6.3
R to *4*	*SO*	3	Table 6.1
Confirm: $CO(7) = 10$ and $SO(7) = 1$			
3 to *4*	*CO*	28	Figure 6.3
3 to *4*	*SO*	3	Table 6.1
Confirm: $CO(3) = 18$ and $SO(3) = 2$			
1	*CO*	25	Figure 6.3
1	*SO*	3	Table 6.1
2	*CO*	21	Figure 6.3
2	*SO*	2	Table 6.1
R to *1*	*CO*	26	Figure 6.3
R to *1*	*SO*	3	Table 6.1
Set: $CO(R) = 26$ and $SO(R) = 3$			
8	*CO*	22	Figure 6.3
8	*SO*	2	Table 6.1
3 to *8*	*CO*	24	Equations 6.3
3 to *8*	*SO*	3	Equations 6.3
Confirm: $CO(3) = 18$ and $SO(3) = 2$			

Figure: Sequential circuit measures after stabilization

Testability of a Stuck-at Fault

- SCOAP may also be used to predict the length of the test vector set for a circuit.
- In order to detect a fault at x, one must set x to the opposite value from the fault and observe x at a PO.
- The testabilities of the stuck-at faults at node x are defined as:
  - T(Stuck-at-0)= $CC1(x) + CO(x)$
  - T(Stuck-at-1)= $CC0(x) + CO(x)$
  - Testability index= $log \sum_{all\ f_i} T(f_i)$

Problem on Testability Measures

For the circuit of Figure, compute the combinational and sequential SCOAP testability measures (both controllability and observability) (including those for CK and the synchronous $\overline{RESET}$.

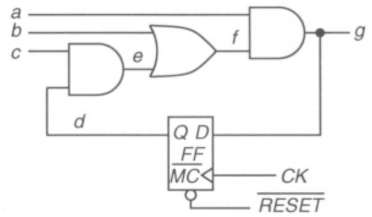


Figure: Sequential circuit.

Solution: Problem on Testability Measures

The initial step of calculation for SCOAP testability measures is shown in the Figure. Combinational measures are shown as (CC0;CC1)CO and sequential measures as [SC0; SC1]SO.

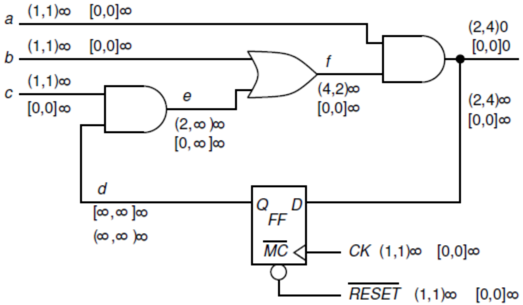


Figure: Sequential circuit.

SOLUTION: PROBLEM ON TESTABILITY MEASURES

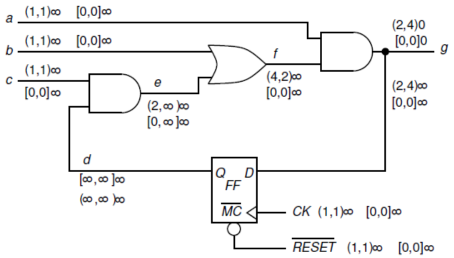


Figure: conversed testability measures.

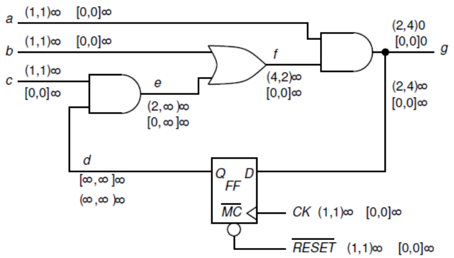


Figure: All controllability and observability measures.

Querries Are Wellcome