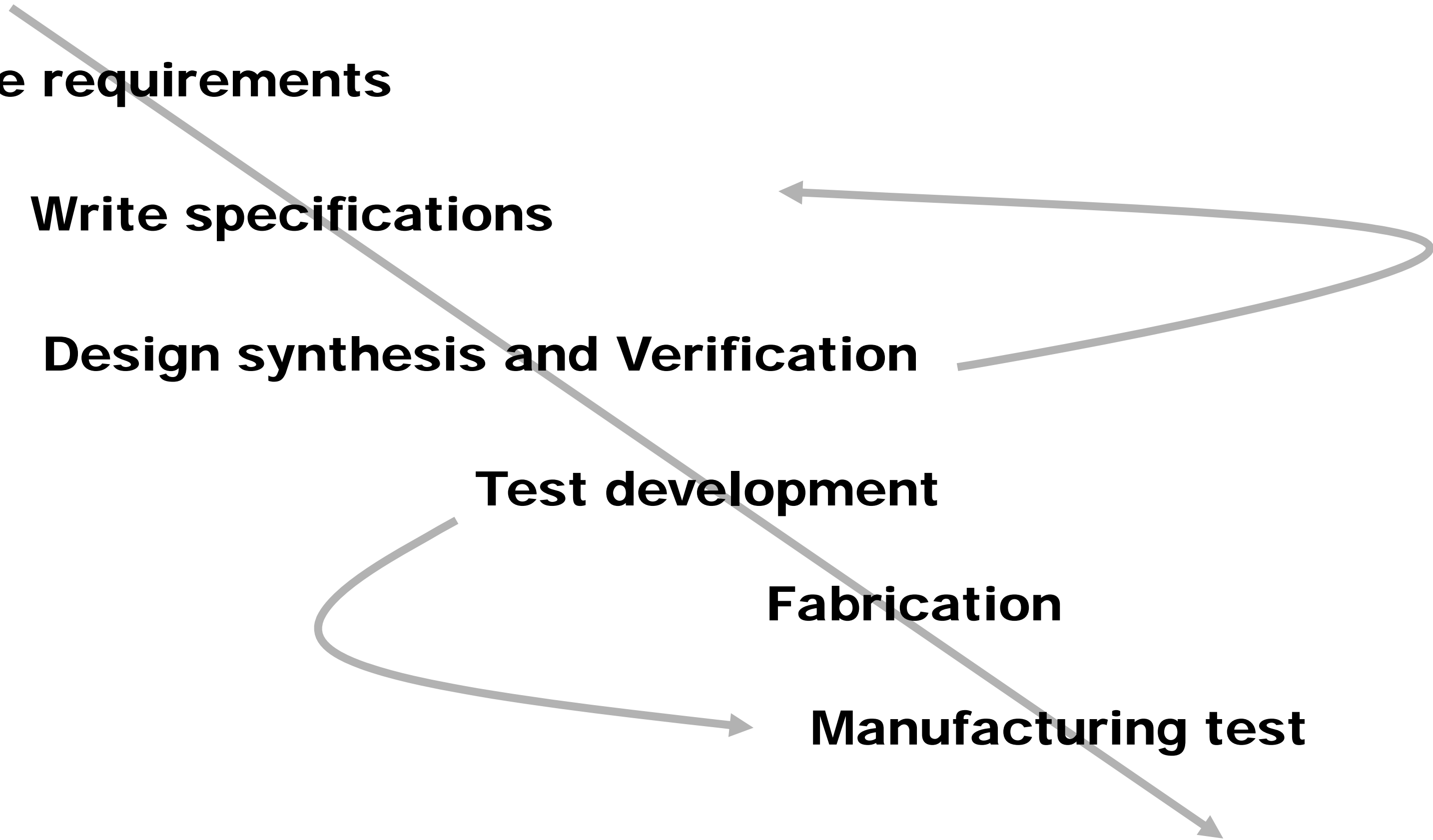
Computer Aided Digital Systems Design - EE 4743/6743

Sherif Abdelwahed

Design for Test

Department of Electrical and Computer Engineering
Mississippi State University

Digital Design Realization Process



Definitions

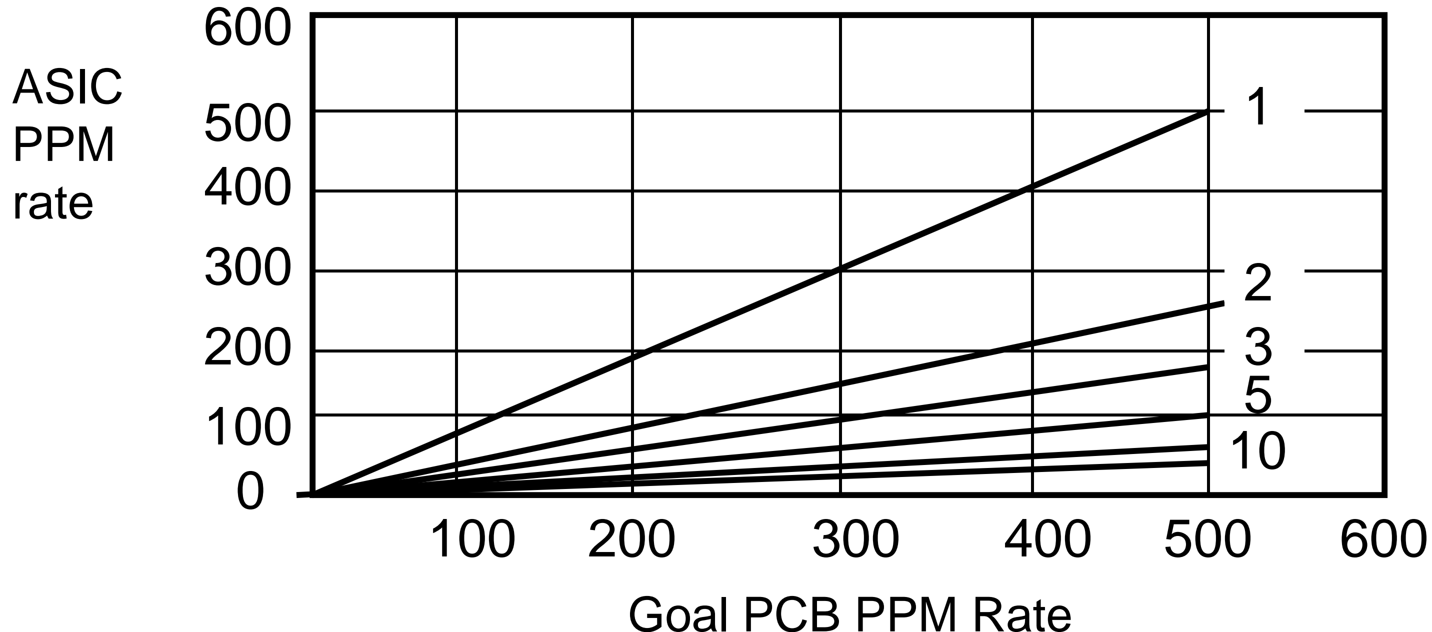
- **Design synthesis**: Given an I/O function, develop a procedure to manufacture a digital device using known design infrastructure.
- **Verification**: Predictive analysis to ensure that the synthesized design, when manufactured, will perform the given I/O function.
- **Test**: A manufacturing step that ensures that the physical device, manufactured from the synthesized design, has no manufacturing defect.

Practical Aspects of Digital Design Tests

- Based on analyzable fault models, which may not map on real defects.
- Incomplete coverage of modeled faults due to high complexity.
- Some good chips are rejected. The fraction (or percentage) 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.***

The Effects of Single Chip Defects

Consider the effects of having multiple chips on a board



Source: TI's IEEE 1149.1 Testability Primer

If you have 5 chips on a board, you need to have around 60 PPM (parts per million) defective chips to achieve 300 PPM defective boards.

Some Real Defects in Chips

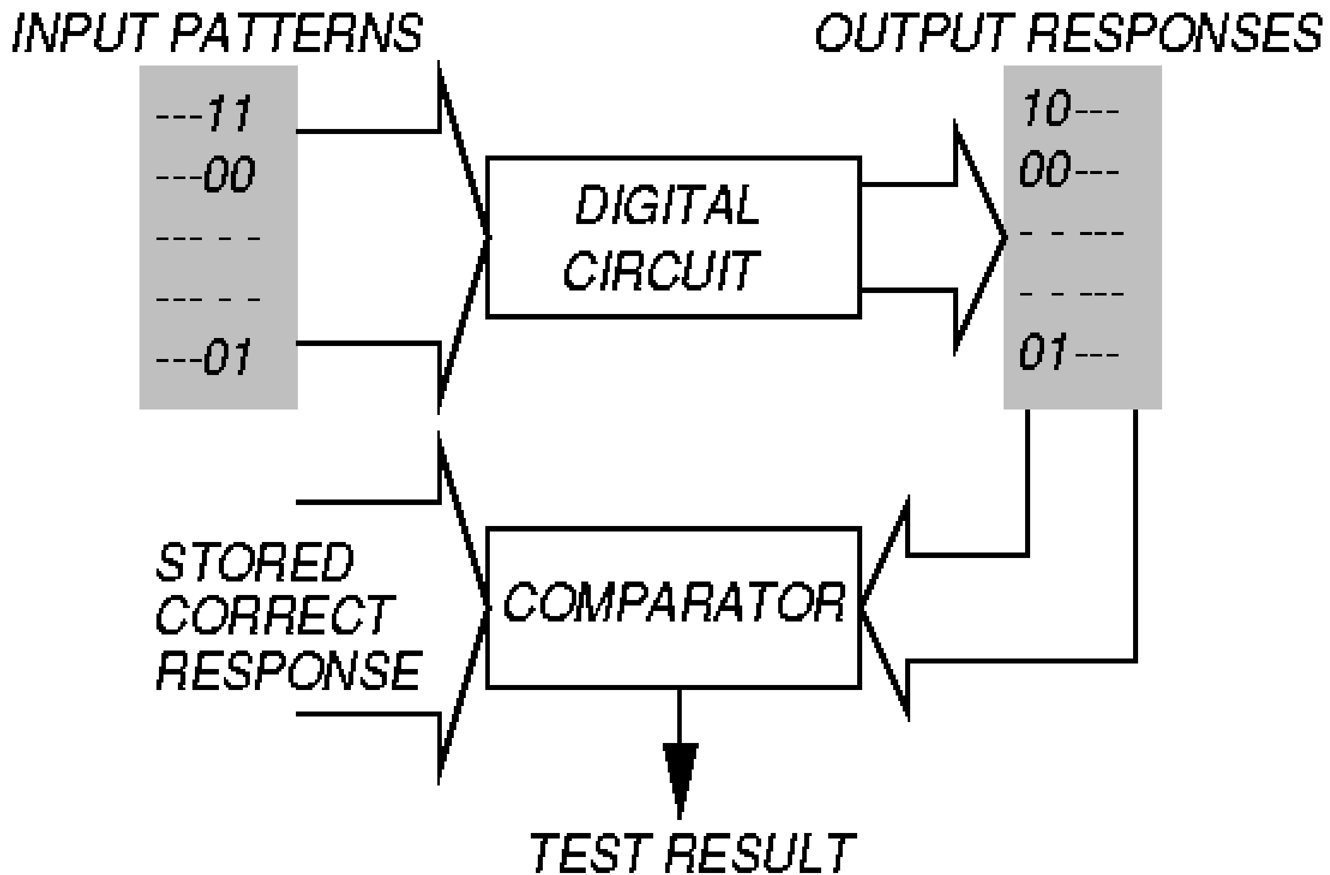
- **Processing defects**
  - Missing contact windows
  - Parasitic transistors
  - Oxide breakdown
- **Material defects**
  - Bulk defects (cracks, crystal imperfections)
  - Surface impurities (ion migration)
- **Time-dependent failures**
  - Dielectric breakdown
  - Electromigration
- **Packaging failures**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

Ref.: M. J. Howes and D. V. Morgan, *Reliability and Degradation - Semiconductor Devices and Circuits,* Wiley, 1981.

Costs of Testing

- *Design for testability* (DFT)
  - 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

Testing Principle



Fault Modeling

- Logical vs. Physical Faults
  - Logical faults represent the behavioral effects of Physical faults
- Why Logical Fault Modeling?
  - Reduced complexity: real defects (often mechanical) too numerous and often not analyzable
  - A fault model identifies targets for testing
  - A fault model makes analysis possible
  - Effectiveness measurable by experiments
  - Technology-independent
- Easily modeled for diagnosis

Logical Fault Models

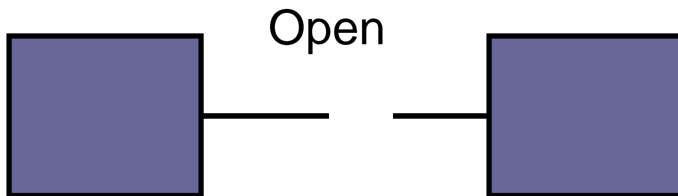
- Explicit vs. Implicit Models
  - **Explicit:** Every fault is individually identified and all faults are explicitly enumerated
  - **Implicit:** Collectively identifies faults of interest and defines their characteristic properties
- Structural vs. Functional Faults
  - **Structural:** Faults related to a structural model, i.e. opens, shorts
  - **Functional:** Faults related to their effect on the circuit function (higher level of abstraction)
    - 1 + 2 = 4?
- Permanent vs. Intermittent (transient) faults
  - We'll only consider permanent faults

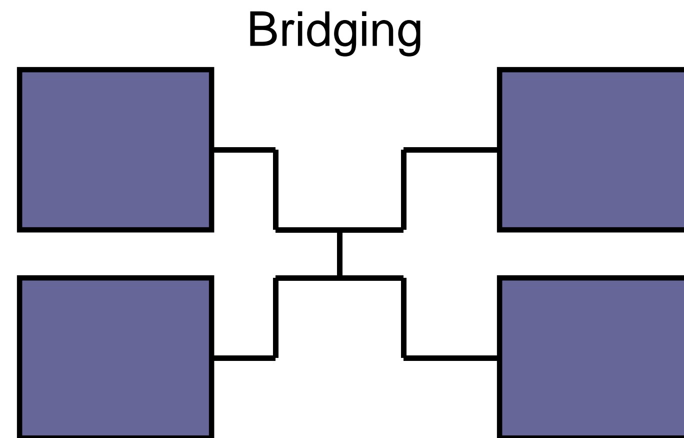
Common Faults in Digital Systems

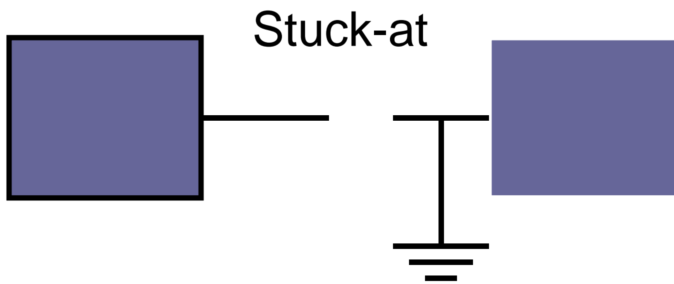
- Single stuck-at faults
- Transistor open and short faults
- Memory faults
- PLA faults (stuck-at, cross-point, bridging)
- Functional faults (processors)
- Delay faults (transition, path)
- Analog faults

Common Faults in Digital Components

- Shorts, opens, stuck-at, bridging faults

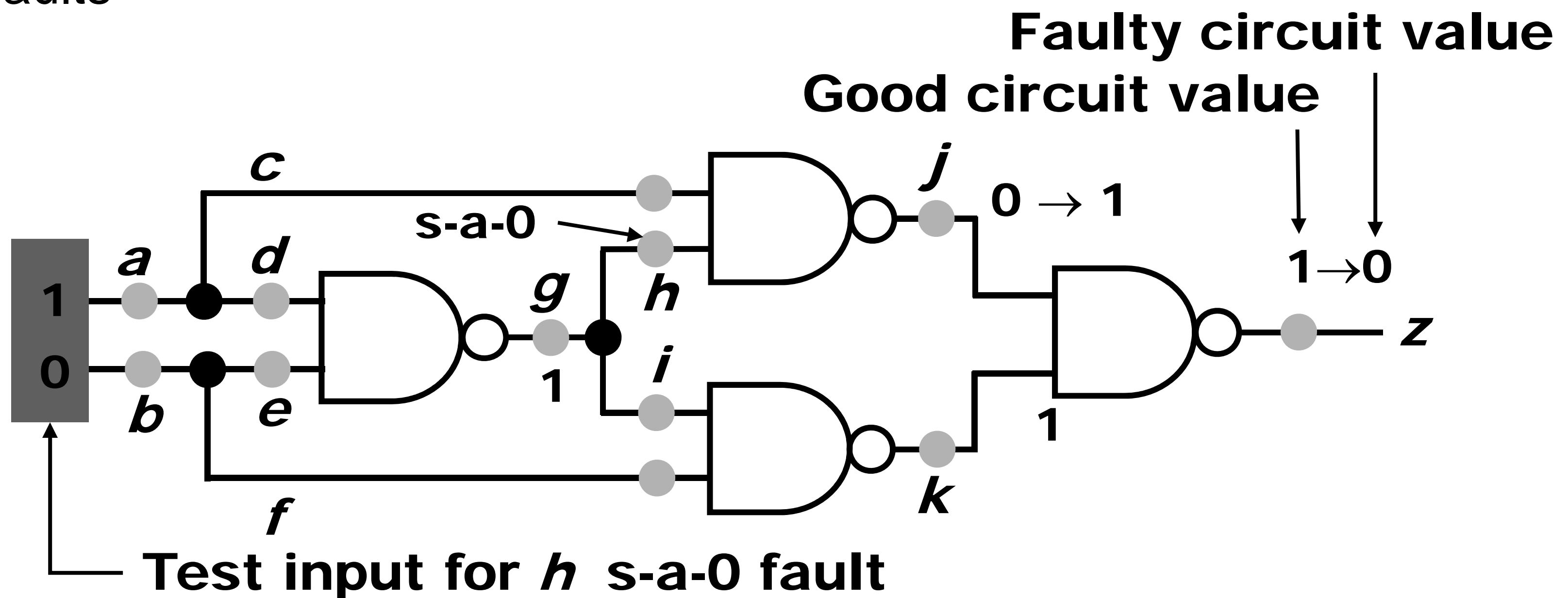






Single Stuck-at Fault

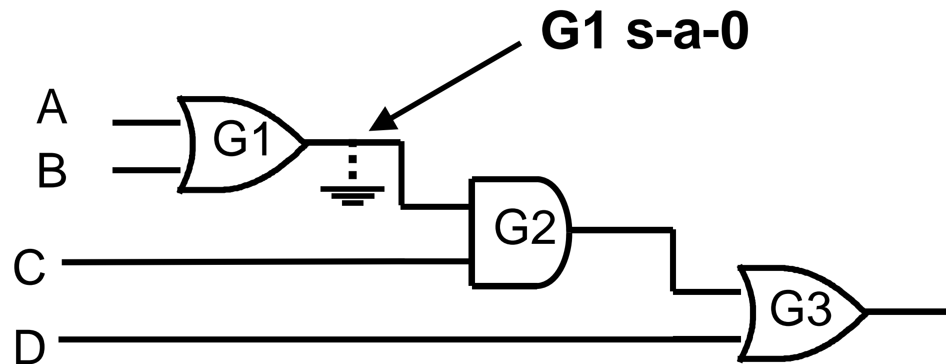
- Three properties define a single stuck-at-value (s-a-v) fault
  - Only one line is faulty
  - The faulty line is permanently set to 0 or 1
  - The fault can be at an input or output of a gate
- Example: XOR circuit has 12 fault sites (●) and 24 single stuck-at faults



Fault Detectability

- Let $Z(x)$ be the logic function of a circuit N, where x represents an arbitrary input vector and $Z(x)$ denotes the mapping realized by N.
- Faulty circuit: The presence of a fault f transforms N into a new circuit N_f with a new function $Z_f(x)$.
- Test: Denote by t a specific input vector (test vector), and by $Z(t)$ the response of N.
- The circuit is tested by applying a **test vector** t and by comparing the output response with the expected output response of N.
- A test t **detects** a fault *f* iff $Z(t) \neq Z_f(t)$.

Detectability Example



$$Z(t) = ((A+B).C)+D$$

- The test vector t = 1010 (ABCD) detects the fault f = G1 s-a-0

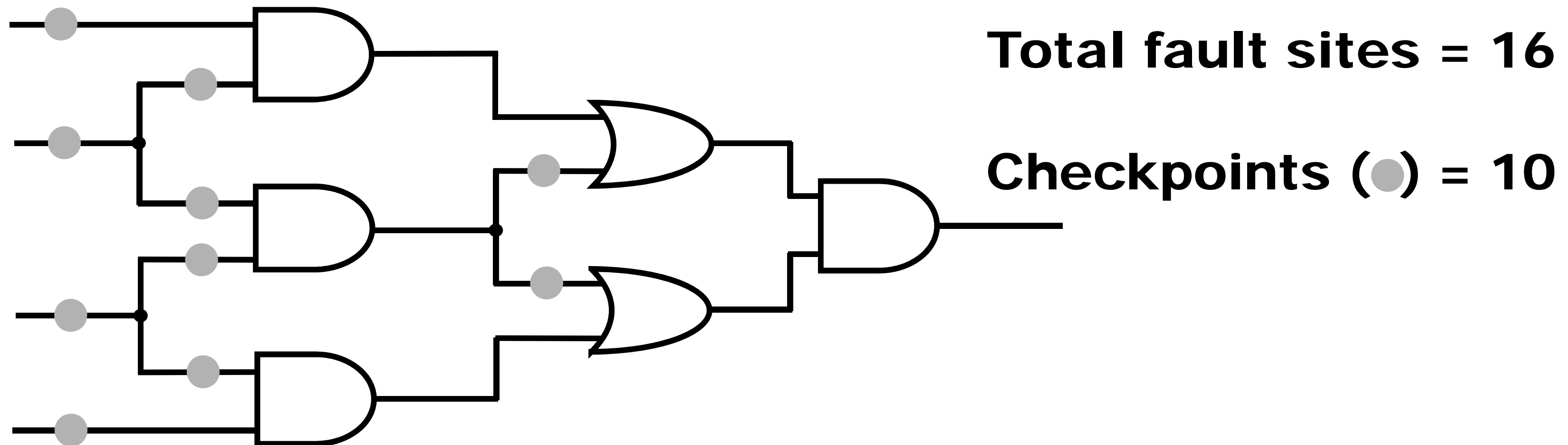
$$Z(t) = 0 \text{ but } Z_f(t) = 1$$

Redundancy

- A combinational circuit with undetectable stuck fault is said to be **redundant:** the circuit can be simplified by removing at least a gate or gate input:
  - Ex. a s-a-1 fault on an input of an AND gate is undetectable.
  - Since the gate function does not change in the presence of the fault, we can permanently place a 1 value on that input.
  - But an n-input AND with a constant 1 value on one input is logically equivalent to the (n-1)-input AND.
  - What is the case for an AND input s-a-0 that is undetectable?
- Redundancy may be introduced on purpose in the following cases:
  - in designing fault-tolerant or self-testing circuits,
  - in designing circuits to avoid hazards

Checkpoints

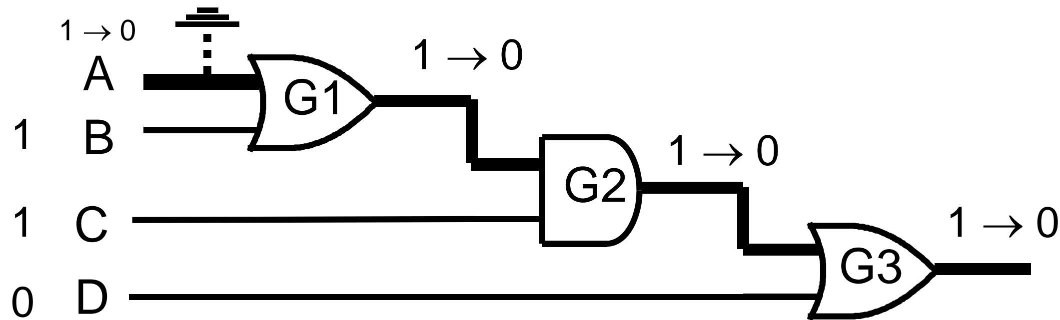
- Primary inputs and fanout branches of a combinational circuit are called checkpoints.
- **Checkpoint theorem:** A test set that detects all single (multiple) stuck-at faults on all checkpoints of a combinational circuit, also detects all single (multiple) stuck-at faults in that circuit.



Fault Sensitization

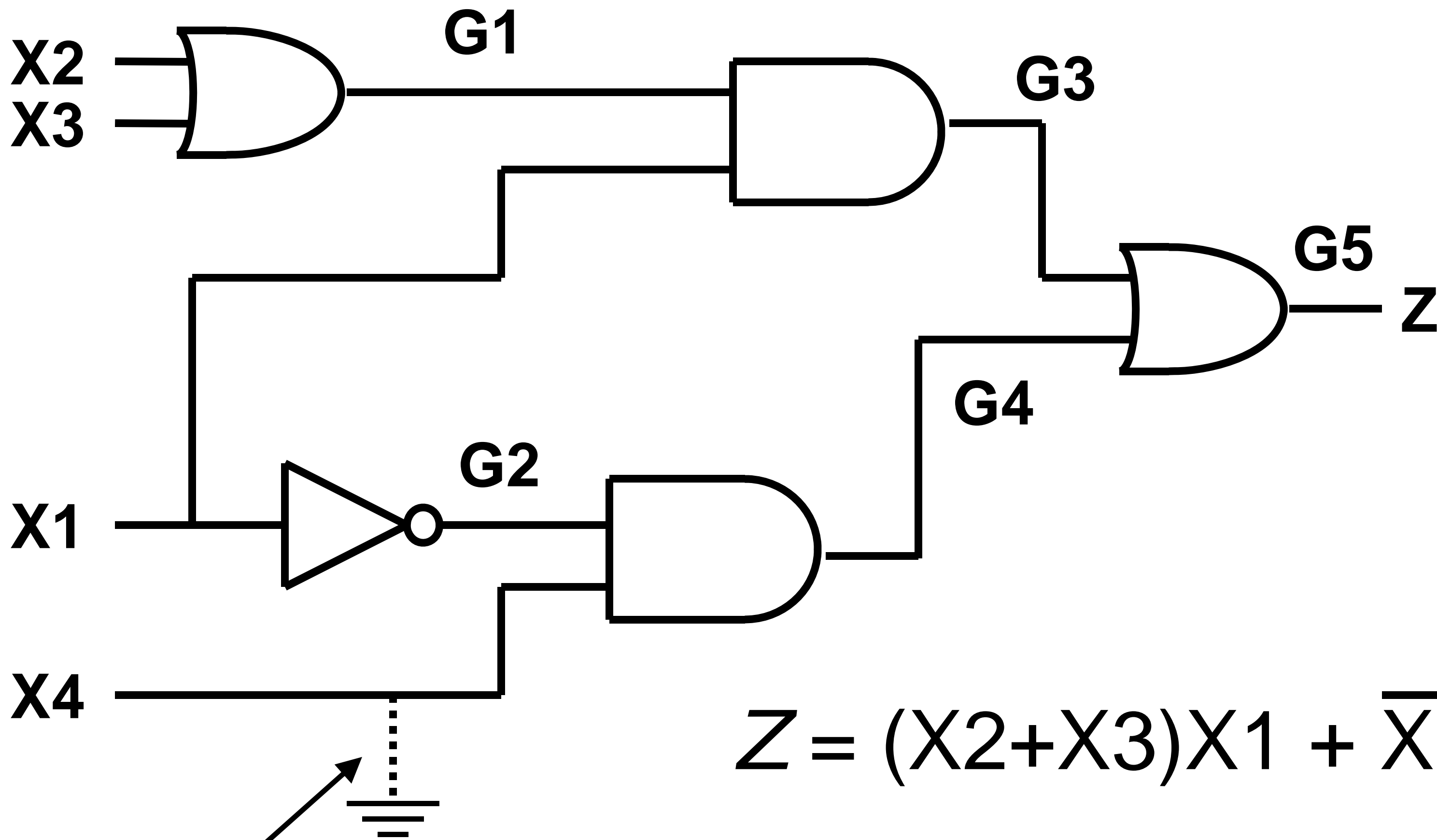
- A line in a circuit whose value in the test *t* changes in the presence of the fault f is said to be **sensitized** to the fault *f* by the test *t*.
- We say also: the test *t* activates a fault *f*.
- A path composed of sensitized lines is called a **sensitized path**.
- We say also: the test t propagates a fault (fault effect) along sensitized path.

Fault Sensitization



- The fault A s-a-0 is sensitized by the value 1 on a line A
- A test *t* = 1110 is simulated, both without and with the fault A s-a-0.
- The results of the simulation are different in the two cases, shown in a form v → v' where v and v' are corresponding signal values in the fault-free and in the faulty circuit.
- A path from the faulty line ***A*** is sensitized (bold lines) to the primary output of the circuit.

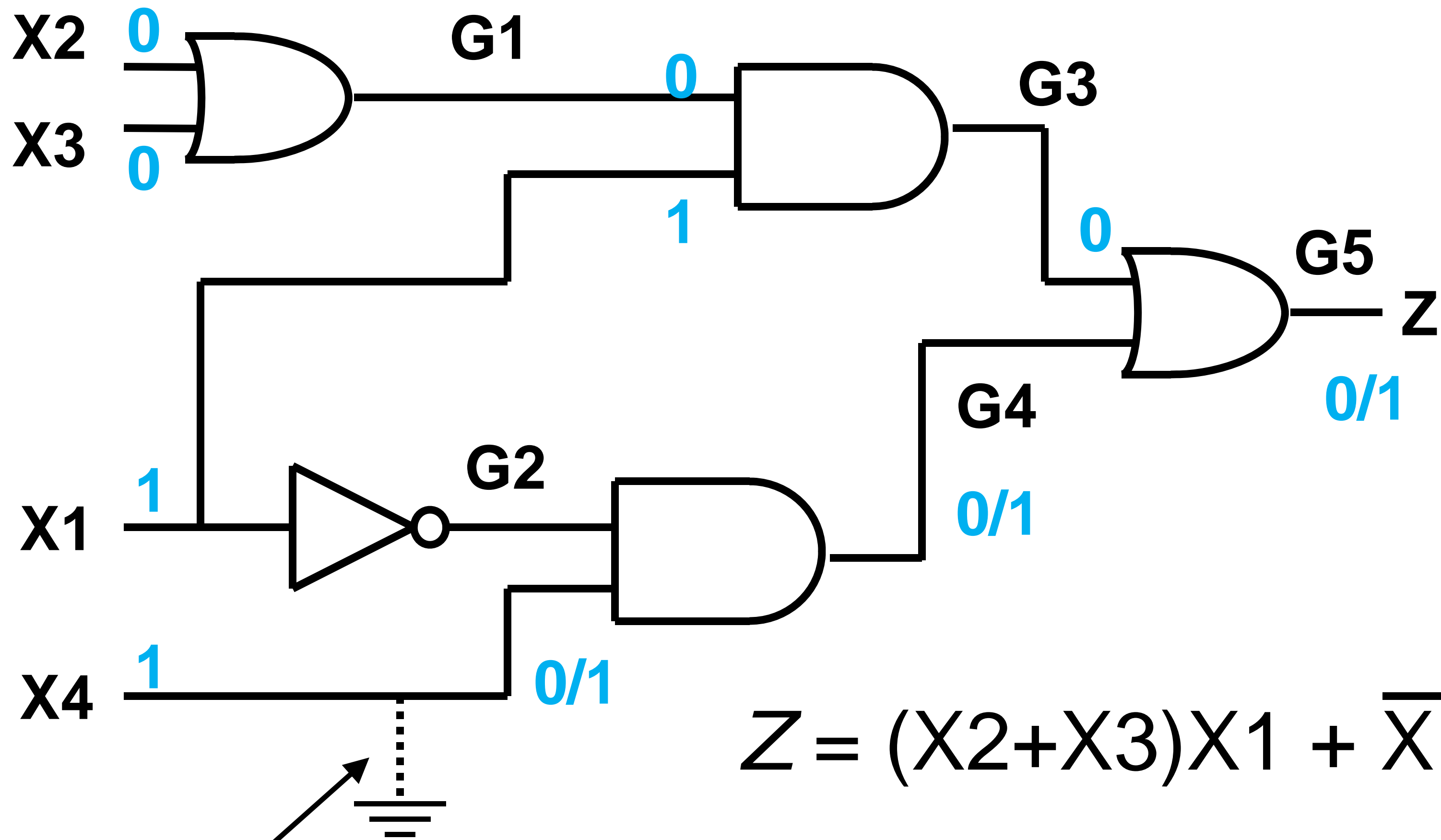
Another Example



$$Z = (X2+X3)X1 + \overline{X1}X4$$

$$Z_f = (X2+X3)X1$$

Sensitization and Propagation



$$Z = (X2+X3)X1 + \overline{X1}X4$$

$$Z_f = (X2+X3)X1$$

Extending the Concept

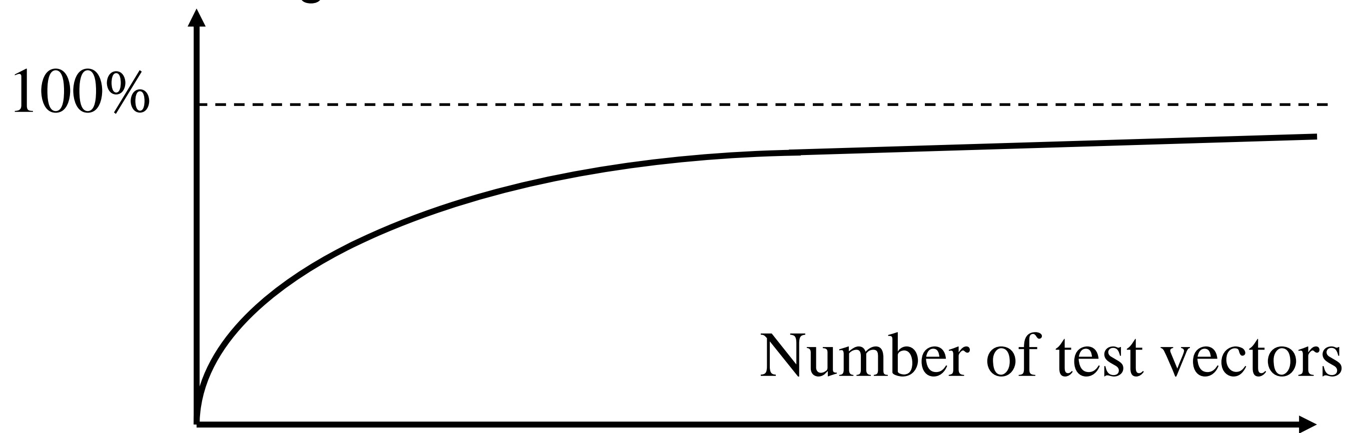
- A set of inputs and outputs designed to test a specific gate is called a ***test vector***
  - A test vector can check multiple faults at once
- A collection test vectors is a ***test*** or a ***test vector suite***
- Also need to develop a way to get those inputs to the block
  - And directly observe the block's outputs

Fault Testing Problems

- Four additional test vectors are required for every additional 2-input gate.
  - The number of test vectors increases as the number of gates increases
- The number of gates is increasing much faster than the number of inputs/outputs.
  - This is making test detetability more difficult
- Sequential circuits can be notoriously difficult to test.
  - For example, a 16-bit counter has to cycled through its entire count for complete testing.
- Not all faults can be tested
  - Fault coverage problem

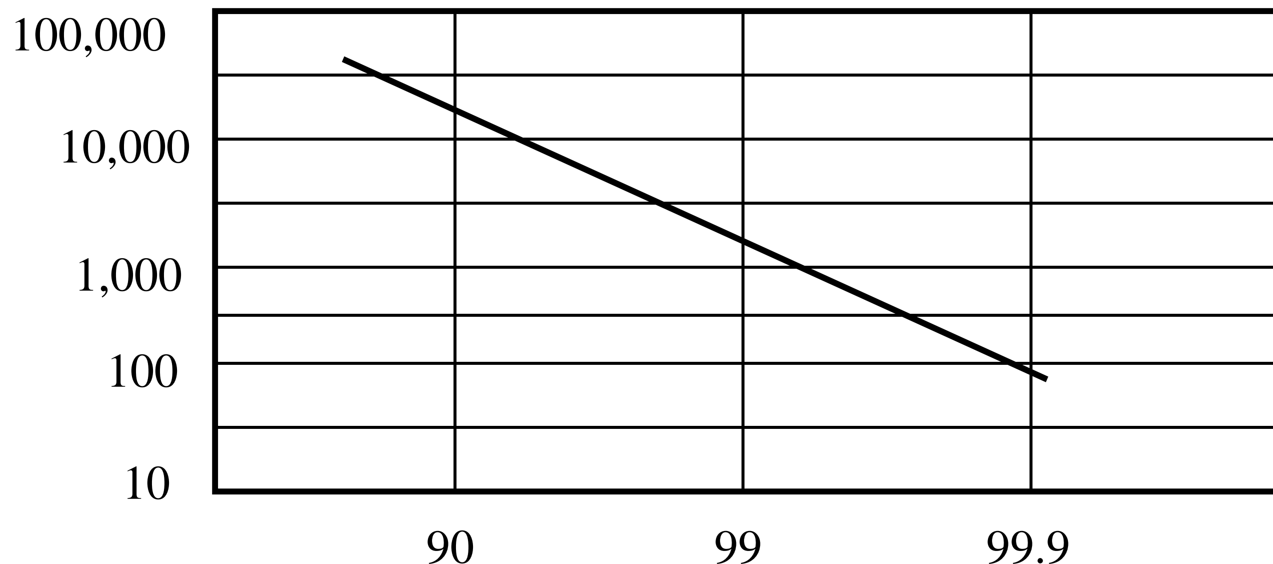
Fault Coverage

- Cannot apply all test vectors
- One test vector can test many faults
- Only use the test vectors that test the most faults
- Decreasing return for every extra test vector
  - At some point, it becomes inefficient to test more
- **Fault Coverage**:
  - Percentage of gates have been completely tested by the test vectors applied to the device
- Fault Coverage curves look like:



Is Fault Coverage Sufficient?

Study results show defect level vs. fault coverage:



Source: 1980 study by Delco and Motorola, as cited in TI's IEEE 1149.1 Testability Primer

To get a single chip defect rate of 100 defective parts per million parts, you need 99.9% fault coverage! Do you really need a defect rate of less than 100 ppm?

What does all this mean?

- You need fault coverage in the 99% to 99.9% range
- You need automated test vector generation and fault grading tools
  - Fault grading tells you % fault coverage for a test vector suite
- You need extra hardware to improve testability.

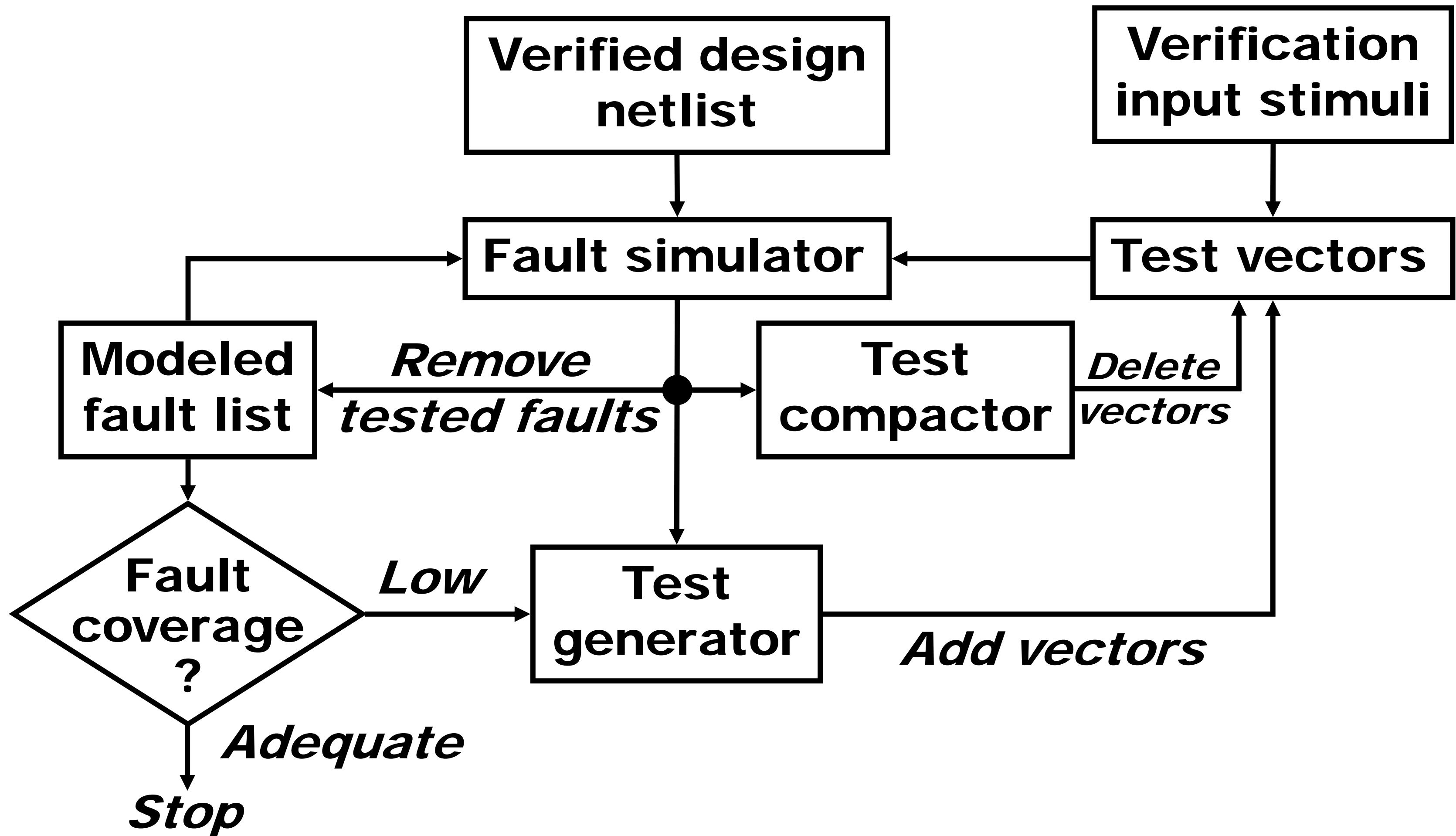
Fault Simulation: Problem and Motivation

- Fault simulation Problem
  - Given
    - A circuit
    - A sequence of test vectors (a test)
    - A fault model (describing type and location of faults)
  - Determine
    - Fault coverage - fraction (or percentage) of modeled faults detected by test vectors
    - Set of undetected faults
- Motivation
  - Determine test quality and in turn product quality
  - Find undetected fault targets to improve tests

Modeling for Simulation

- Modules, blocks or components described by
  - Input/output (I/O) function
  - Delays associated with I/O signals
  - Examples: binary adder, Boolean gates, resistors, capacitors …
- Interconnects represent
  - ideal signal carriers, or
  - ideal electrical conductors
- **Netlist:** a format (or language) that describes a design as an interconnection of modules.
  - A part or device used in a netlist is called an **instance**.
  - Each instance has a **master**, or **definition**.
  - These definitions will usually list the connections that can be made to that kind of device, and some basic properties of that device.

Fault simulator in Digital Design Process



Fault Simulation Settings

- Circuit model: mixed-level
  - Mostly logic with some switch-level for high-impedance (Z) and bidirectional signals
  - High-level models (memory, etc.) with pin faults
- Signal states: logic
  - Two (0, 1) or three (0, 1, X) states for purely Boolean logic circuits
  - Four states (0, 1, X, Z) for sequential MOS circuits
- Timing:
  - Zero-delay for combinational and synchronous circuits
  - Mostly unit-delay for circuits with feedback

Fault Simulation Scenario

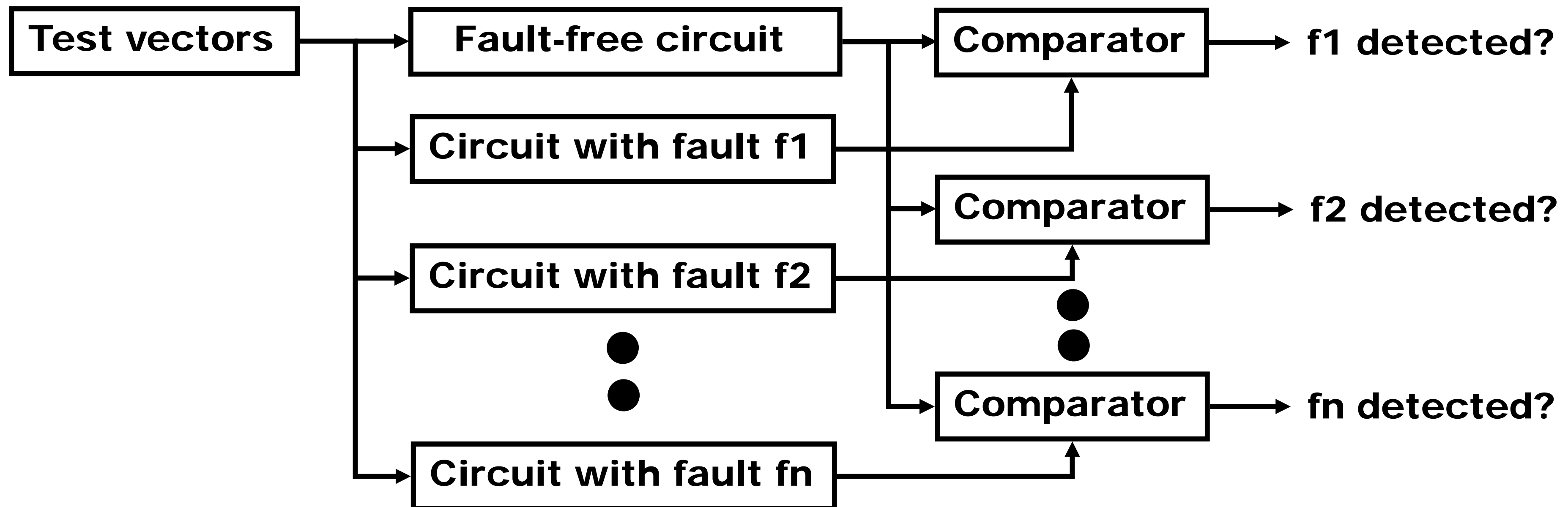
- Mostly single stuck-at faults
- Sometimes stuck-open, transition, and path-delay faults; analog circuit fault simulators are not yet in common use
- Fault-dropping -- a fault once detected is dropped from consideration as more vectors are simulated; fault-dropping may be suppressed for diagnosis
- Fault sampling -- a random sample of faults is simulated when the circuit is large

Serial Algorithm

- Algorithm: Simulate fault-free circuit and save responses. Repeat following steps for each fault in the fault list:
  - Modify netlist by injecting one fault
  - Simulate modified netlist, vector by vector, comparing responses with saved responses
  - If response differs, report fault detection and suspend simulation of remaining vectors
- Advantages:
  - Easy to implement; needs only a true-value simulator, less memory
  - Most faults, including analog faults, can be simulated

Serial Algorithm

- Disadvantage: Much repeated computation; CPU time prohibitive for large circuits
- Alternative: Simulate many faults together

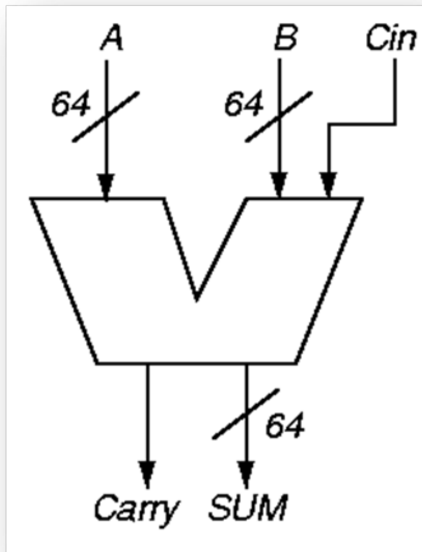


Automated Test Vector Generation

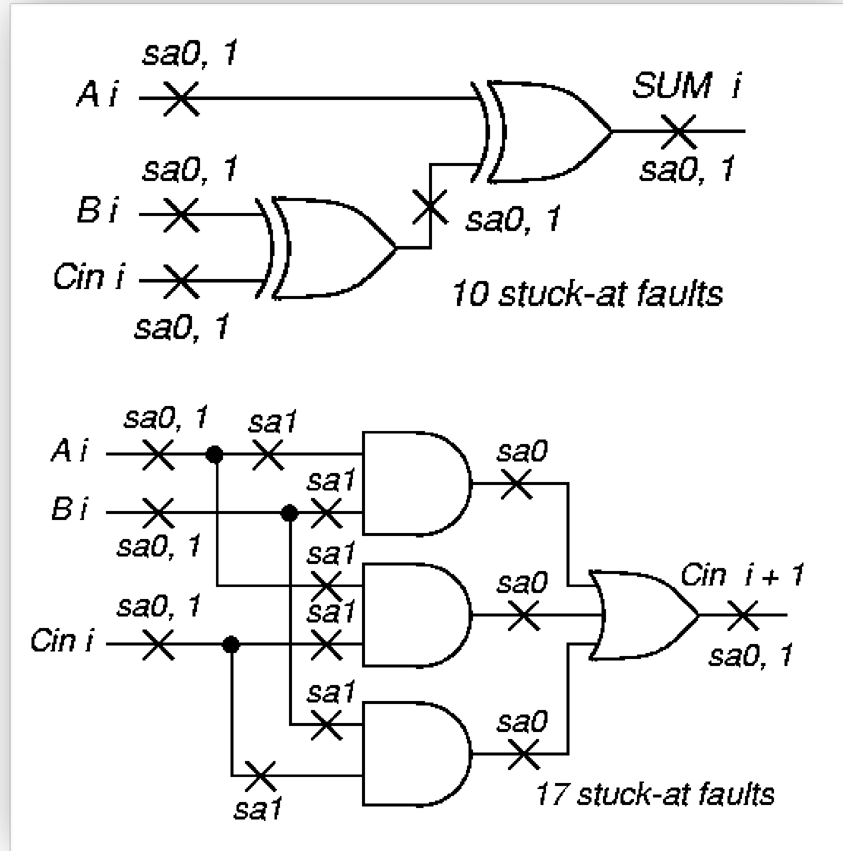
- Usually use software to generate all test vectors based on your HDL
- Automatic Test Pattern Generation
  - Inject fault into circuit modeled in computer
  - Use various ways to activate and propagate fault effect through hardware to circuit output
  - Output flips from expected to faulty signal

Functional vs. Structural ATPG

A 64-bit Adder circuit



Functional

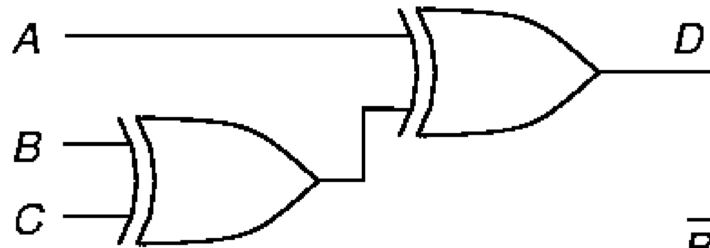


Structural – 1 bit module

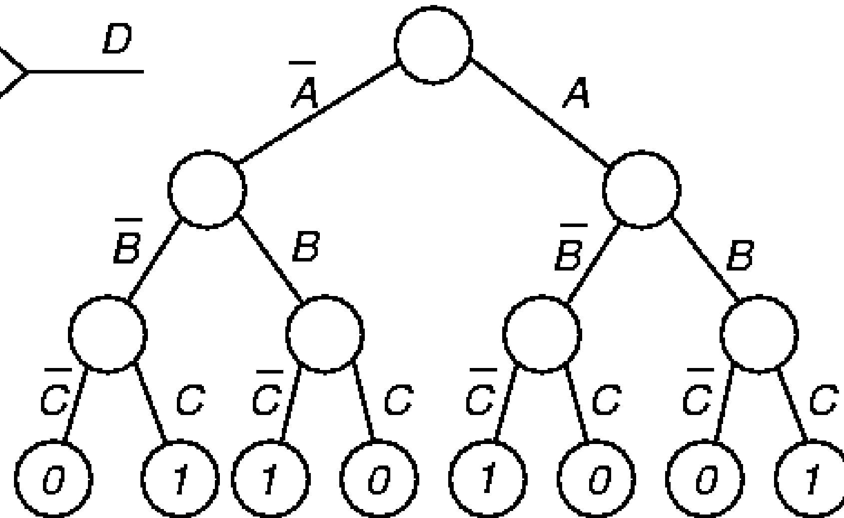
Functional vs. Structural ATPG

- Functional ATPG – generate complete set of tests for circuit input-output combinations
  - 129 inputs, 65 outputs:
  - 2^{129} = 680,564,733,841,876,926,926,749, 214,863,536,422,912 patterns
  - Using 1 GHz PC, would take 2.15×10^{22} years
- Structural test:
  - No redundant adder hardware, 64 bit slices
  - Each with 27 faults (using fault equivalence)
  - At most 64 x 27 = 1728 faults (tests)
  - Takes 0.000001728 sec. on 1 GHz PC
- Designer gives small set of functional tests – augment with structural tests to boost coverage to 98^{+} %

Circuits and Binary Decision Tress



(a) Circuit.

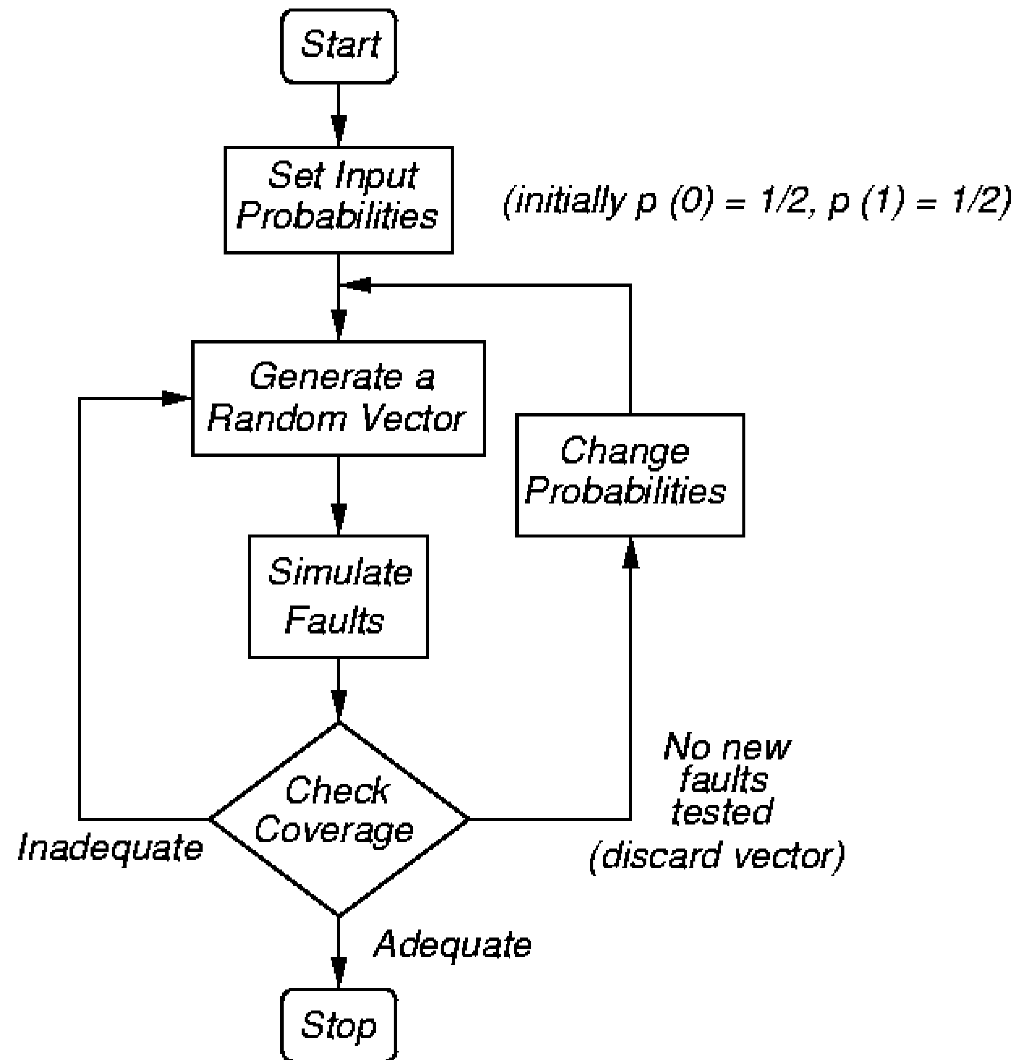


(b) Binary decision tree.

- Definition: Algorithm is **complete** if it ultimately can search entire binary decision tree, as needed, to generate a test
- *Untestable fault* – no test for it even after entire tree searched
- Combinational circuits only – untestable faults are **redundant**, showing the presence of unnecessary hardware

Random Pattern Generation

- Flow chart for method
- Use to get tests for 60-80% of faults, then switch to other ATPG algorithm for rest



Design for testability

- Design for testability (DFT) refers to those design techniques that make test generation and test application cost-effective.
- DFT methods for digital circuits:
  - Ad-hoc methods
  - Structured methods:
    - Scan
    - Partial Scan
    - Built-in self-test (BIST)
    - Boundary scan
- DFT method for mixed-signal circuits:
    - Analog test bus

Ad-Hoc DFT Methods

- Good design practices learnt through experience are used as guidelines:
  - Avoid asynchronous (unclocked) feedback.
  - Make flip-flops initializable.
  - Avoid redundant gates. Avoid large fanin gates.
  - Provide test control for difficult-to-control signals.
  - Avoid gated clocks.
  - Consider ATE requirements (tristates, etc.)
- Design reviews conducted by experts or design auditing tools.
- Disadvantages of ad-hoc DFT methods:
  - Experts and tools not always available.
  - Test generation is often manual. No guarantee of high fault coverage.
  - Design iterations may be necessary.

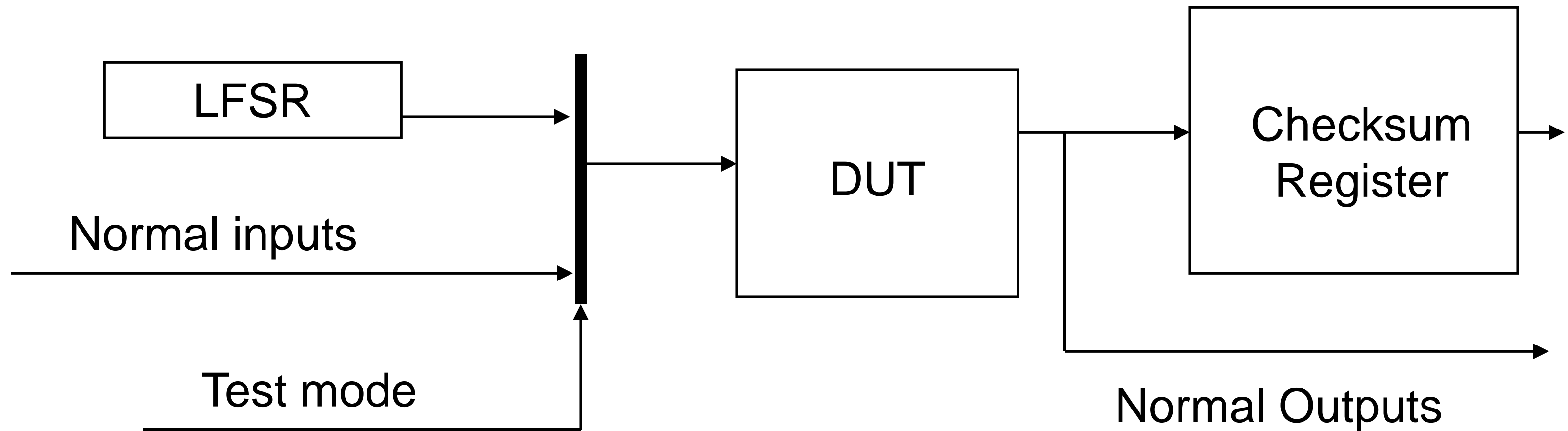
Built-In Self Test (BIST)

- Without extra hardware added explicitly to help with testing, fault coverage is typically in the 90% to 95% range.
- On-chip test mechanisms can increase this to above 99%
- One way to add reliability to a component or part of a chip is with Built-In Self Test (BIST)
- An example is the Motorola 68030 MCU
  - Motorola has added a small set of instructions which they certify tests a significant fraction of the 68030's gates.
  - Can be incorporated into the low level startup process to detect and processor faults.

Built-In Self Test (BIST) Approach

- Add circuitry so that on powerup a small ROM or Sequence generator applies a series of test vectors
- The outputs are tested for a correct answer, usually by accumulating the outputs into a checksum value (either by XOR or Cyclic Redundancy Check - CRC)
- The Psuedo-random pattern generation and checksum accumulation does not need many gates
- The probability of passing self-test with an undiagnosed fault can be made small (depends on number of patterns).

Built in Self Test (BIST)



DUT = device under test

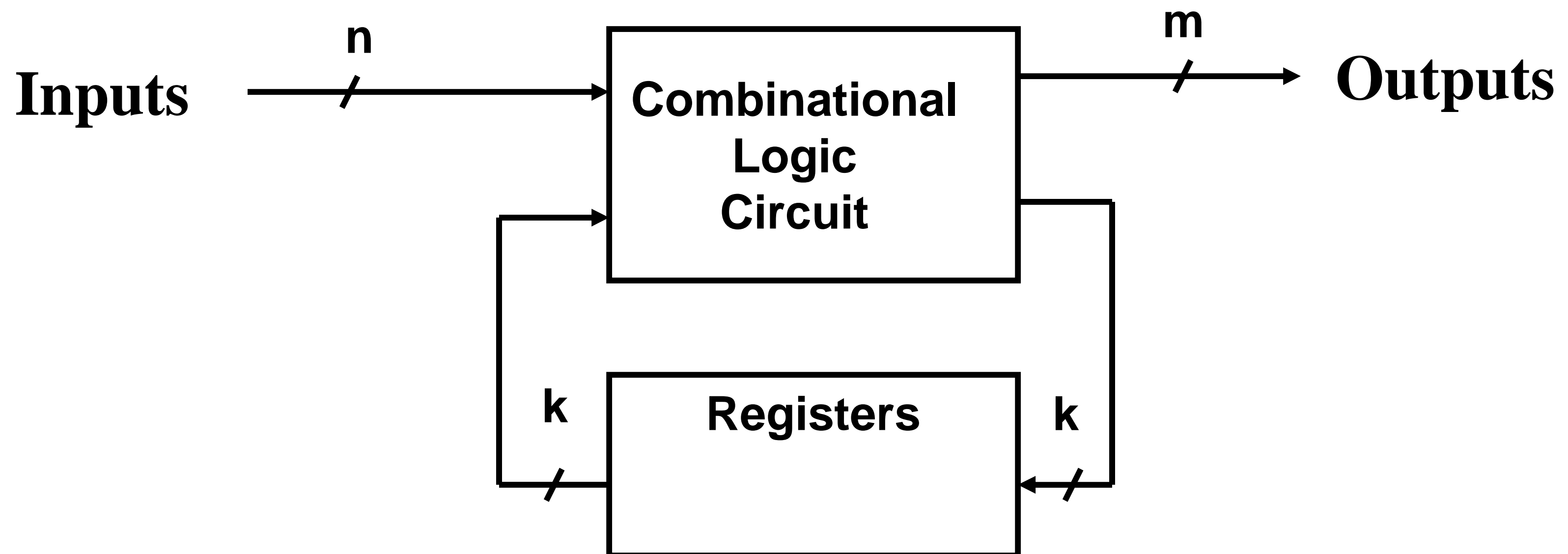
When in self test, inputs to Device Under Test (DUT) is taken from LFSR and checksum register accumulates the output. After some X number of test vectors have been applied, compare check sum output against a golden value – if the same, then system has passed self test.

Scan Design

- Circuit is designed using pre-specified design rules.
- Test structure (hardware) is added to the verified design:
  - Add a test control (TC) primary input.
  - Replace flip-flops by scan flip-flops (SFF) and connect to form one or more shift registers in the test mode.
  - Make input/output of each scan shift register controllable/observable from PI/PO.
- Use combinational ATPG to obtain tests for all testable faults in the combinational logic.
- Add shift register tests and convert ATPG tests into scan sequences for use in manufacturing test.

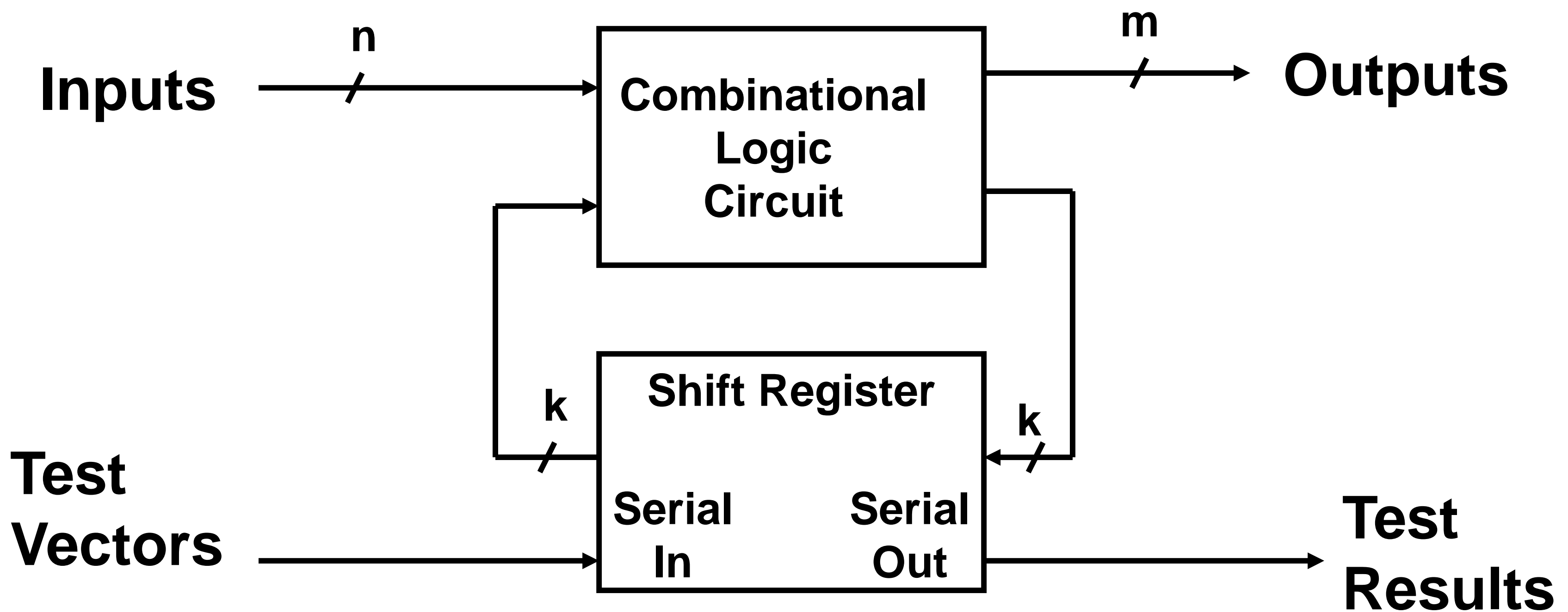
Scan Path Design

- Use a **Scan Path**
  - A Scan Path ties all FFs in your design into a shift register.
  - Requires Changing DFFs

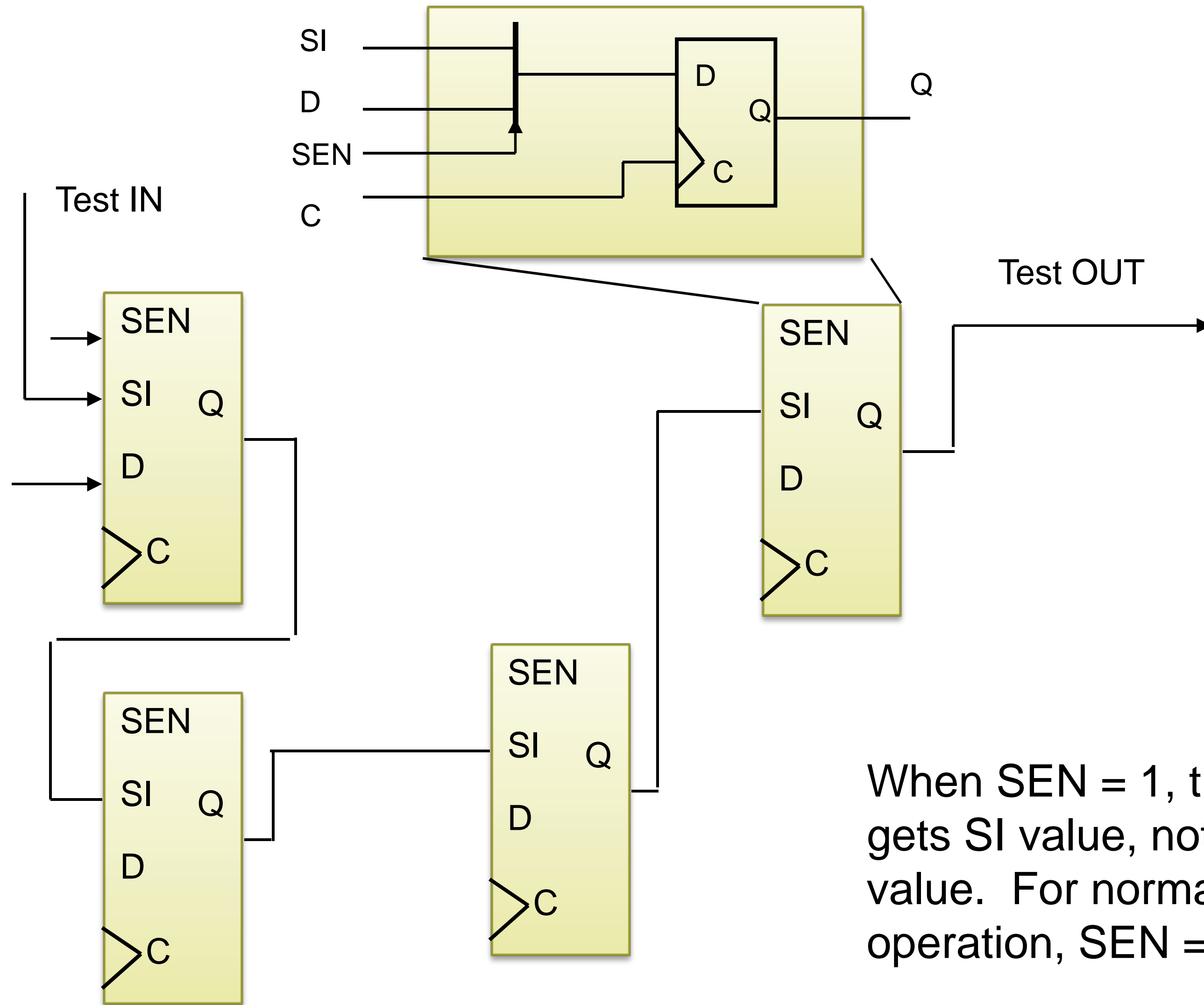


Scan Path Design

- Serially shift a test vector into the registers
- Apply entire vector in parallel
- Capture the result of the test back into the registers
- Serially shift results out of registers



Scan Path Design

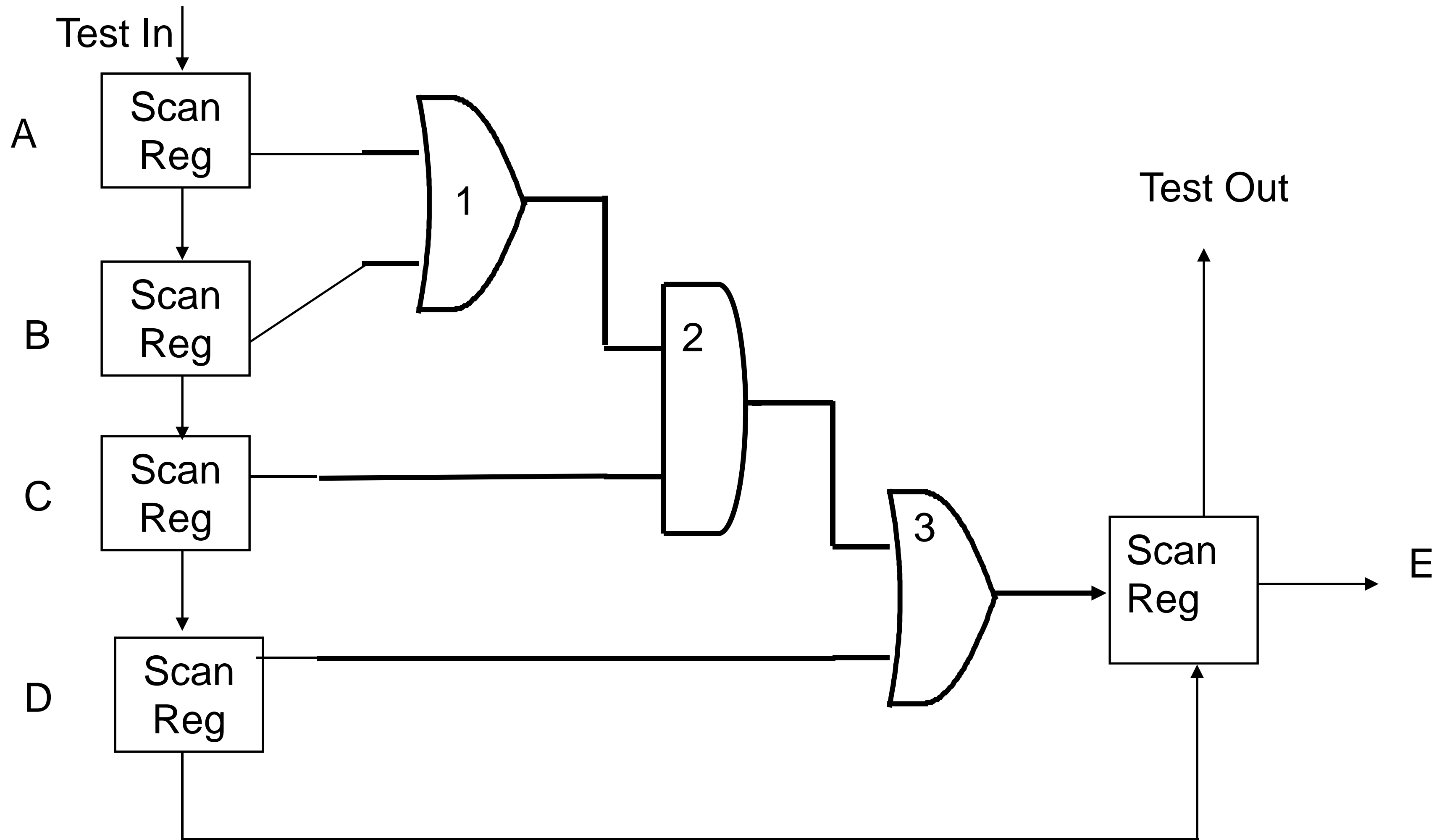


When SEN = 1, then DFF gets SI value, not the D value. For normal operation, SEN = 0

Scan Path Operation

- Assume ALL FFs in design are in tied into a Scan Path and let there be N FFs.
- Apply SEN = '1', and clock test vector serially into scan path. Will take N clocks.
- Apply, SEN = '0', and clock ONCE. This will test the design using the test vector!
- Apply SEN = '1', and clock in next test vector. At same time, you clocking OUT the result of the last test vector!!!
- After first test vector, each test vector takes N+1 clocks!!

Three-Gate example



A,B,C,D,E do not have to externally accessible. Could be buried deep within the design

Scan Design Summary

- Scan is the most popular DFT technique:
  - Rule-based design
  - Automated DFT hardware insertion
  - Combinational ATPG
- Advantages:
  - Design automation
  - High fault coverage; helpful in diagnosis
  - Hierarchical – scan-testable modules are easily combined into large scan-testable systems
  - Moderate area (~10%) and speed (~5%) overheads
- Disadvantages:
  - Large test data volume and long test time
  - Basically a slow speed (DC) test

Introducing IEEE 1149.1 (JTAG)

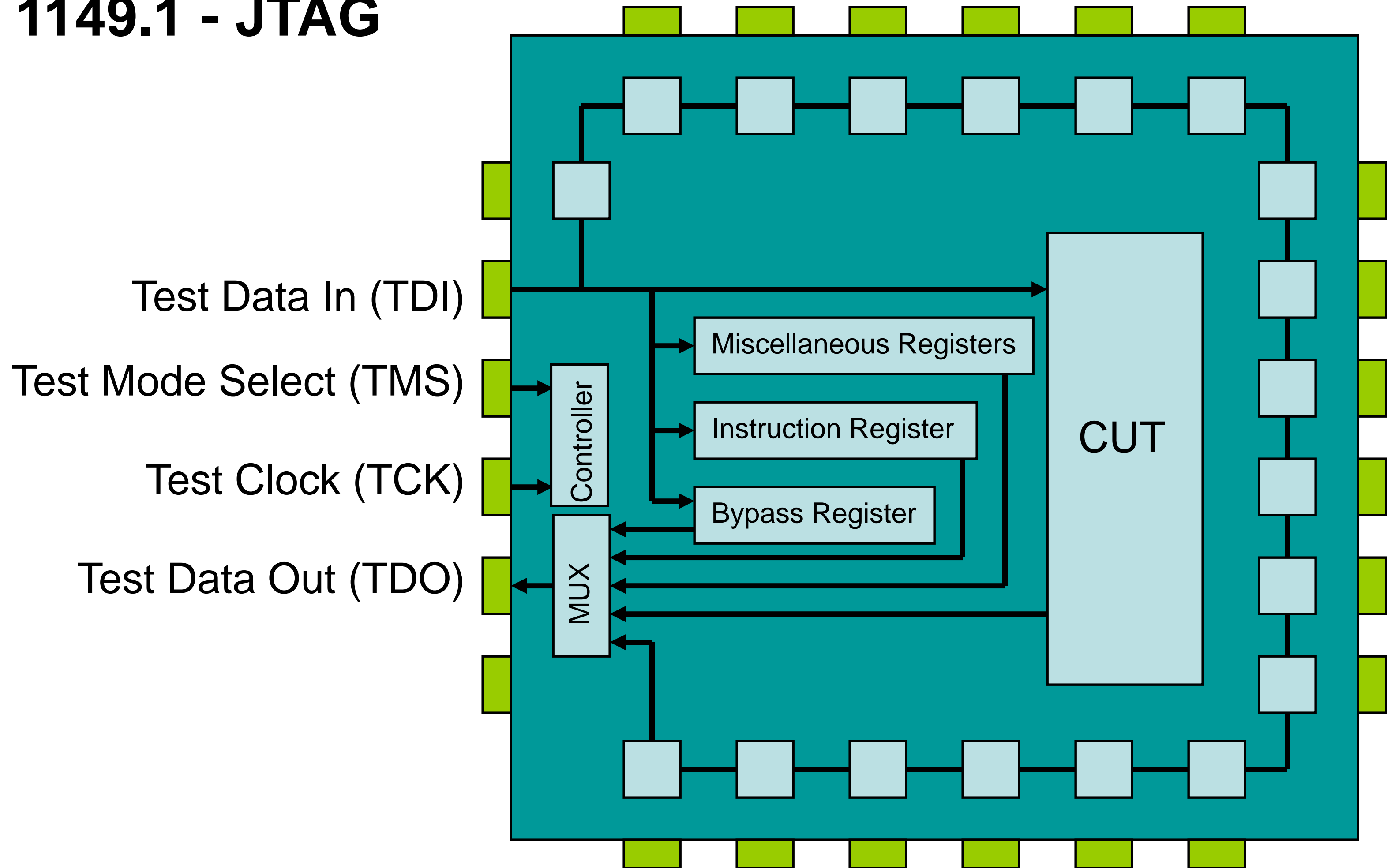
- For standardizing scan path design and support
  - The Joint Test Action Group
  - Developed an industry-wide standard for scan path support
- It has been adopted as IEEE Standard 1149.1
- Also adds registers around I/O - called Boundary Scan
- Every JTAG compatible port can be chained together to form a serial device
- Can be used to support fault detection at the single chip level, detection of board level faults, and even some PLDs/FPGAs can be programmed with it.

Basics of JTAG

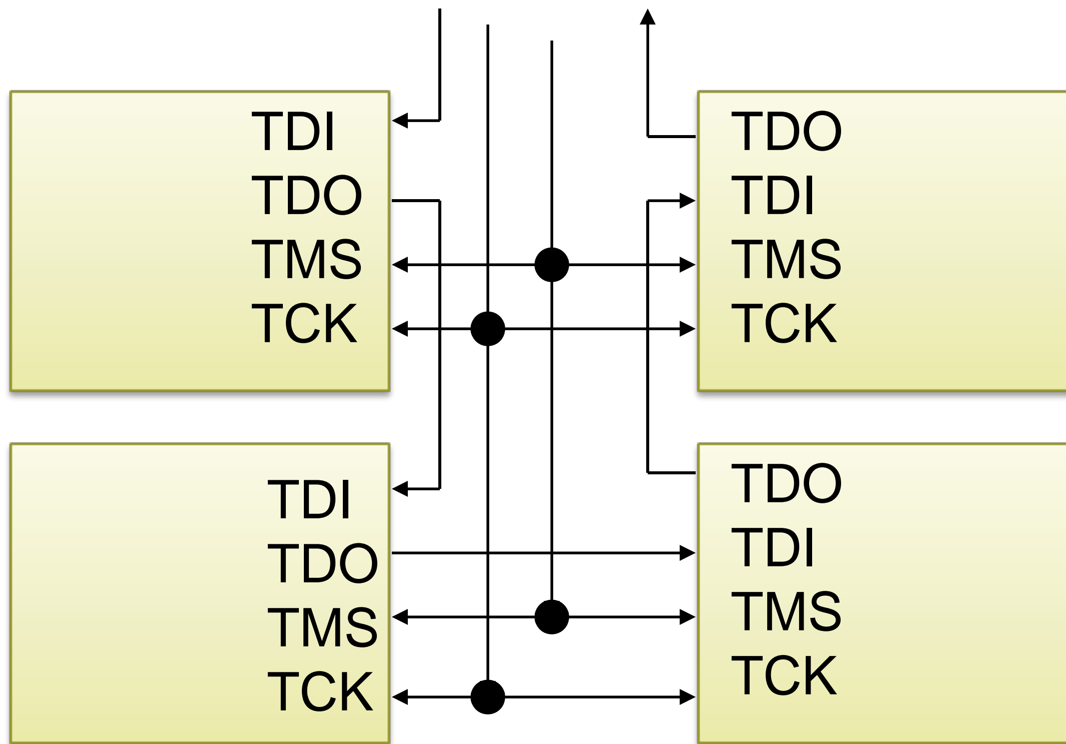
- The JTAG standard adds four signals to every compatible device
  - TCK: a clock signal for the test port
  - TMS: Test Mode Select - determines whether or not the scan path is active and controls the mode it is in.
  - TDI: Test Data In - serial data input
  - TDO: Test Data Out - serial data output
- The standard also specifies some standard operating modes
  - The serial shift register can be set to bypass the chip
  - Makes access to other chips faster
- Vendor specific commands be added (like programming FPGAs)

Serial-Scan DFT Chip Architecture

IEEE 1149.1 - JTAG



JTAG for Board Level Fault Diagnosis



Only four external pins needed.

The JTAG port can drive outputs and capture inputs, board connections between pins can be tested!

Boundary Scan can be used to either stimulate ASICs or simply test board connections between chips.

Some Keys to an Effective Test Strategy

- Decide early, not late, on test strategy
  - Other test approaches beside BIST, JTAG
- Identify software tools for test vector generation, fault grading
- Have to plan test strategy through development, prototyping, manufacturing, and maintenance
- Adding a JTAG bus to a board (and JTAG compatible components) can be a cost effective way of adding a large amount of testability to your design