ECE 3060
VLSI and Advanced Digital Design

Testing

Outline

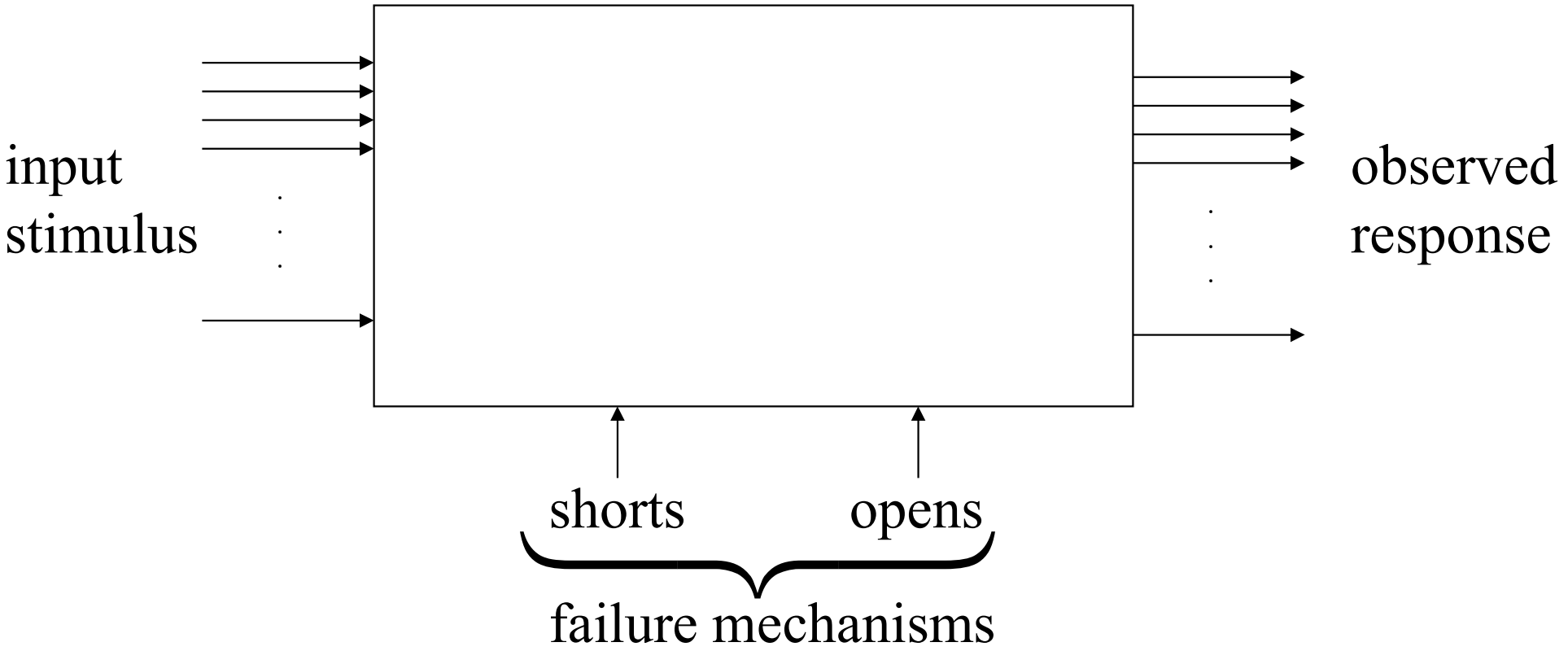
- Definitions
  - Faults and Errors
  - Fault models and definitions
- Fault Detection
- Undetectable Faults
  - can be used in synthesis
- Fault Simulation
- Observability of Faults
- Design for Testability
  - test point insertion
  - scan registers
- Built-in Self Test
- Boundary Scan

Test Cost

- A large percentage (5-40%) of the manufacturing cost of a VLSI chip is due to test
- Cost is larger for larger circuits
  - more functionality in a "System-on-a-Chip" means more potential faults
  - brute-force "stimulate all possible input vectors" can take centuries or more
- Mixed-technology test
  - analog, digital on same chip

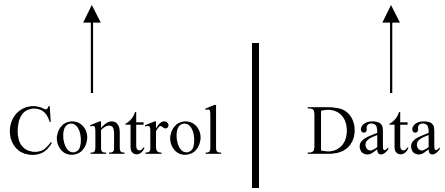
Testing: Definition

- Testing = experiment in which a system is stimulated and its response analyzed in order to determine if the system will behave correctly under arbitrary input stimulus and prescribed failure mechanisms



Failure mechanisms

- Failure mechanisms can be analyzed by looking at how they affect system behaviour at different levels of abstraction
  - messages => system level
  - programs/data structures => processor level
  - instructions/words => instruction set level
  - logic values/words => register level
  - logic values => logic level

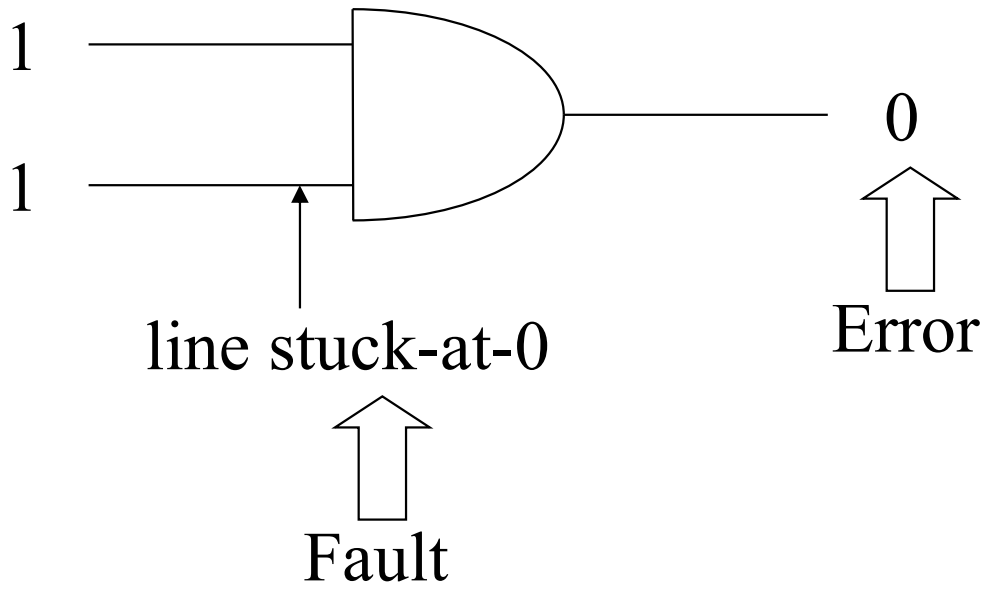


How to Test

- (1) Stimulate the Integrated Circuit (IC) and observe the response using Automatic Test Equipment (ATE)
- (2) Run test/diagnostic program on the processor IC
- (3) Have one or more subsystems test each other by sending and receiving messages

Definitions

- Fault = Defect = Physical failure
  - e.g., shorts, opens, near-open, near-short
- Error = effect of fault, i.e., incorrect logic value due to fault



Source of Errors

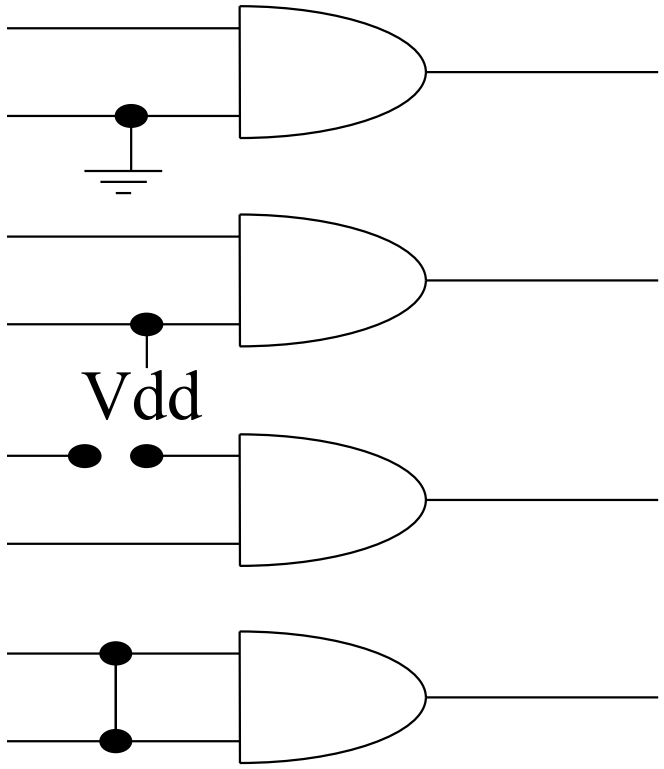
- Note that errors may be due to design errors or fabrication errors
- Design errors:
  - incomplete or inconsistent specification
  - incorrect mappings between different levels of design
  - violations of design rules
- Fabrication errors:
  - greater than λ error in placement of metal/poly/via/etc.
  - wrong components
  - incorrect wiring
  - shorts caused by improper soldering

Nature of Faults

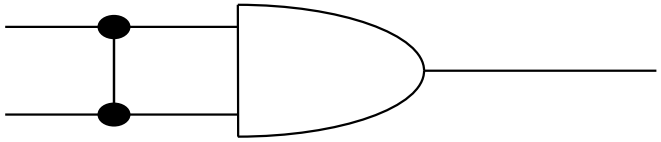
- Permanent
  - always present
- Intermittent
  - occurs at regular intervals
- Transient
  - one time and gone

Fault Model

- Model describes nature of the fault
- stuck at 0
- stuck at 1
- stuck open
- stuck closed



Stuck Closed: Wired-AND or Wired-OR



- Wired-AND

a	b	c	d	output
0	0	0	0	0
0	1	0	0	0
1	0	0	0	0
1	1	1	1	1

- Wired-OR

a	b	c	d	output
0	0	0	0	0
0	1	1	1	1
1	0	1	1	1
1	1	1	1	1

Test Evaluation

- Fault coverage
  - IC with *n* nodes and *z* faults
  - coverage = # of faults detected / *z*
- Obtained via fault simulation

Summary of Definitions

- Testing: definition
  - experiment in which a system is stimulated and its response analyzed in order to determine if the system will behave correctly under arbitrary input stimulus and prescribed failure mechanisms
- Faults and Errors
  - Fault = Defect = Physical failure
  - Error = effect of fault, i.e., incorrect logic value
- Fault Models
  - stuck at 0, stuck at 1, stuck open, stuck closed
- Fault coverage

Criterion	Attribute of testing method	Terminology
When is testing performed?	• Concurrently with the normal system operation	On-line testing Concurrent testing
	• As a separate activity	Off-line testing
Where is the source of the stimuli?	• Within the system itself	Self-testing
	• Applied by an external device (tester)	External testing
What do we test for?	• Design errors	Design verification testing
	• Fabrication errors	Acceptance testing
	• Fabrication defects	Burn-in
	• Infancy physical failures	Quality-assurance testing
	• Physical failures	Field testing Maintenance testing
What is the physical object being tested?	• IC	Component-level testing
	• Board	Board-level testing
	• System	System-level testing
How are the stimuli and/or the expected response produced?	• Retrieved from storage	Stored-pattern testing
	• Generated during testing	Algorithmic testing Comparison testing
How are the stimuli applied?	• In a fixed (predetermined) order	
	• Depending on the results obtained so far	Adaptive testing

Figure 1.2 Types of testing

Testing by *diagnostic programs* is performed off-line, at-speed, and at the system level. The stimuli originate within the system itself, which works in a self-testing mode. In systems whose control logic is microprogrammed, the diagnostic programs can also be microprograms (microdiagnostics). Some parts of the system, referred to as *hardcore*, should be fault-free to allow the program to run. The stimuli are generated by software or

Criterion	Attribute of testing method	Terminology
How fast are the stimuli applied?	• Much slower than the normal operation speed	DC (static) testing
	• At the normal operation speed	AC testing At-speed testing
What are the observed results?	• The entire output patterns	
	• Some function of the output patterns	Compact testing
What lines are accessible for testing?	• Only the I/O lines	Edge-pin testing
	• I/O and internal lines	Guided-probe testing Bed-of-nails testing Electron-beam testing In-circuit testing In-circuit emulation
Who checks the results?	• The system itself	Self-testing Self-checking
	• An external device (tester)	External testing

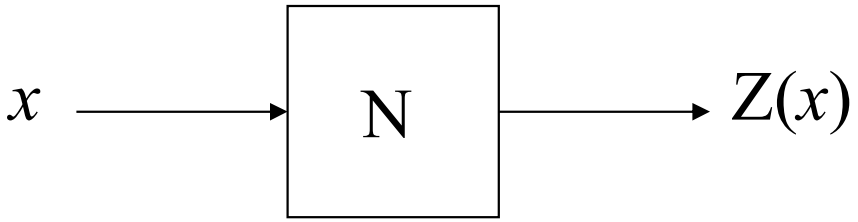
Figure 1.2 (Continued)

firmware and can be adaptively applied. Diagnostic programs are usually run for field or maintenance testing.

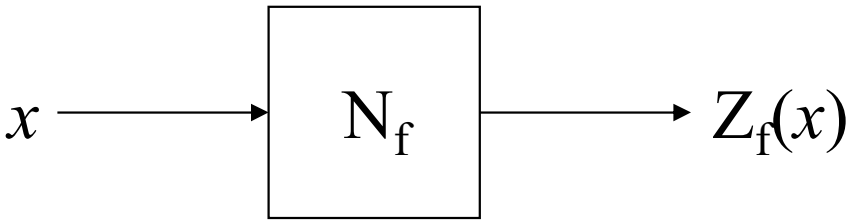
In-circuit emulation is a testing method that eliminates the need for hardcore in running diagnostic programs. This method is used in testing microprocessor (μP)-based boards and systems, and it is based on removing the μP on the board during testing and accessing the μP connections with the rest of the UUT from an external tester. The tester can emulate the function of the removed μP (usually by using a μP of the same type). This configuration allows running of diagnostic programs using the tester's μP and memory.

In *on-line testing*, the stimuli and the response of the system are not known in advance, because the stimuli are provided by the patterns received during the normal mode of operation. The object of interest in on-line testing consists not of the response itself, but of some properties of the response, properties that should remain invariant throughout the fault-free operation. For example, only one output of a fault-free decoder should have logic value 1. The operation code (opcode) of an instruction word in an instruction set processor is restricted to a set of "legal" opcodes. In general, however, such easily definable properties do not exist or are difficult to check. The general approach to on-line testing is based on *reliable design techniques* that create invariant properties that are easy to check during the system's operation. A typical example is the use of an additional parity bit for every byte of memory. The parity bit is set to create an easy-to-check

Fault Detection

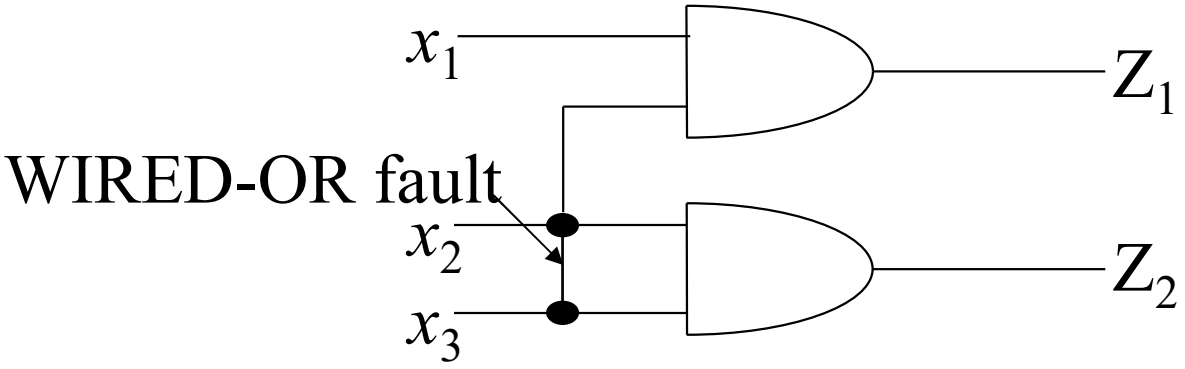


x = input Z(t) = response



- Test set = $\{\mathbf{t_1}, \mathbf{t_2}, \ldots, \mathbf{t_m}\}$
  - $Z(\mathbf{t_1}), Z(\mathbf{t_2}), \ldots, Z(\mathbf{t_m})$
  - $Z_f(\mathbf{t_1}), Z_f(\mathbf{t_2}), \ldots, Z_f(\mathbf{t_m})$
- Definition:
  - **t** detects f iff $Z_f(\mathbf{t}) \mathrel{!}= Z(\mathbf{t})$

Test Set for Fault f



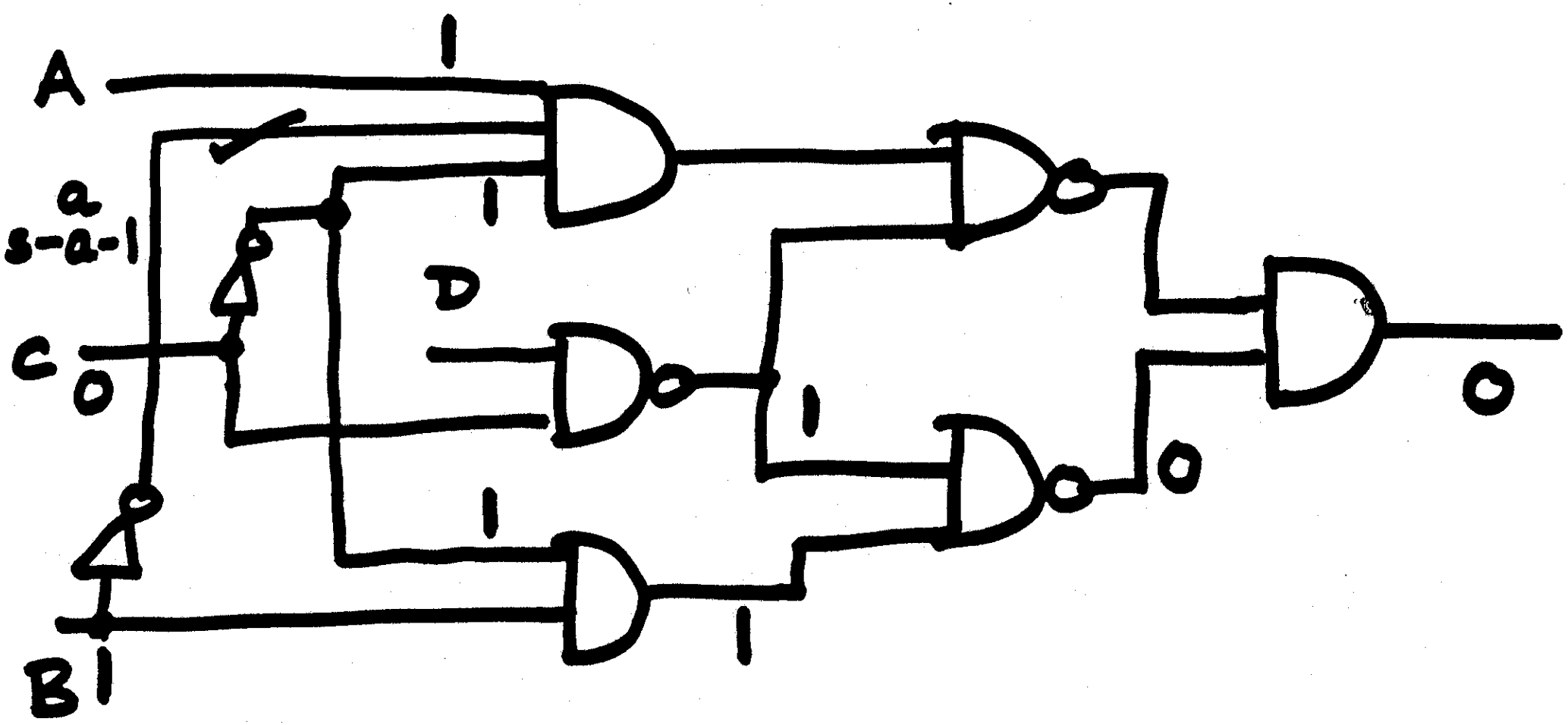
- $Z_1 = x_1 \cdot x_2$, $Z_2 = x_2 \cdot x_3$
- $Z_{1f} = x_1 \cdot (x_2 + x_3)$, $Z_{2f} = x_2 + x_3$
- $\mathbf{t} = \{0,0,1\}$ detects f
- Set of all tests that detect f is given by
  - $\mathbf{Z}(\boldsymbol{x}) \oplus \mathbf{Z}_f(\boldsymbol{x}) = 1$

Example

Example (continued)

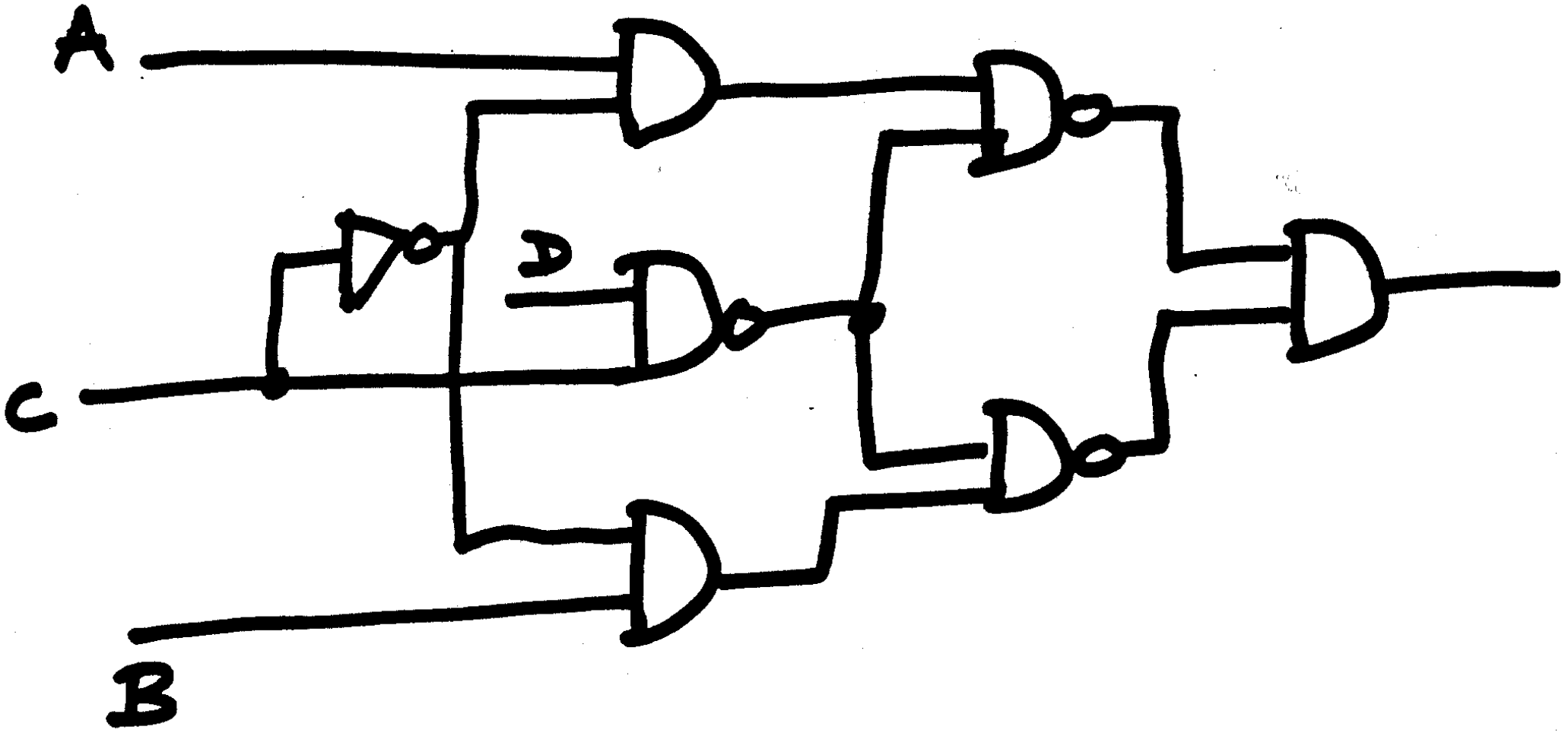
- Note in the example
  - error propogation
  - sensitized path
- A gate whose output is sensitized to a fault f must have at least one of its inputs sensitized to the fault as well

Undetectable Fault



– a stuck at 1 fault on input a is undetectable

Simplification of Previous Circuit

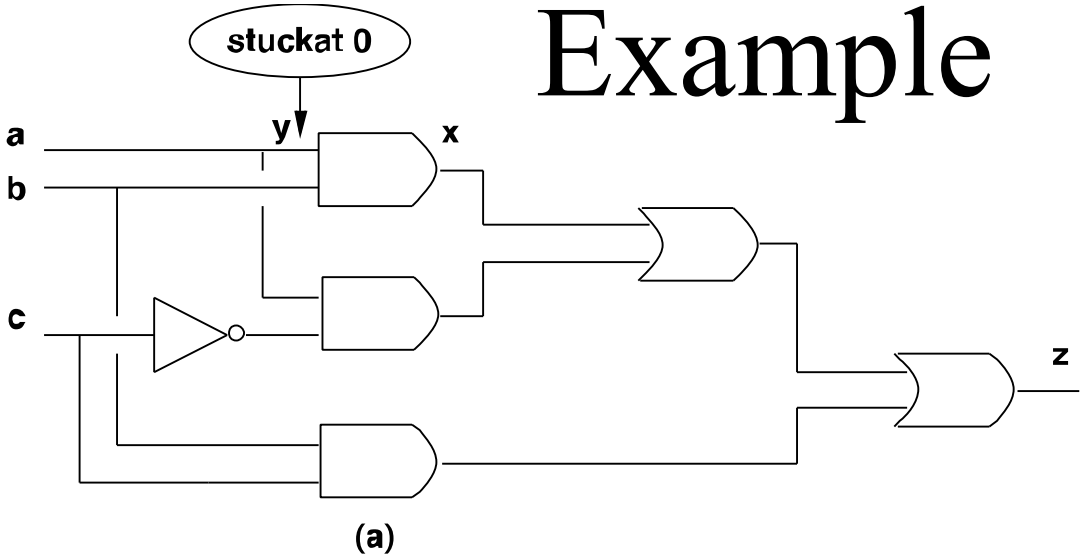


– can use discovery of an undetectable fault to trigger logic minimizer on circuit with undetectable fault

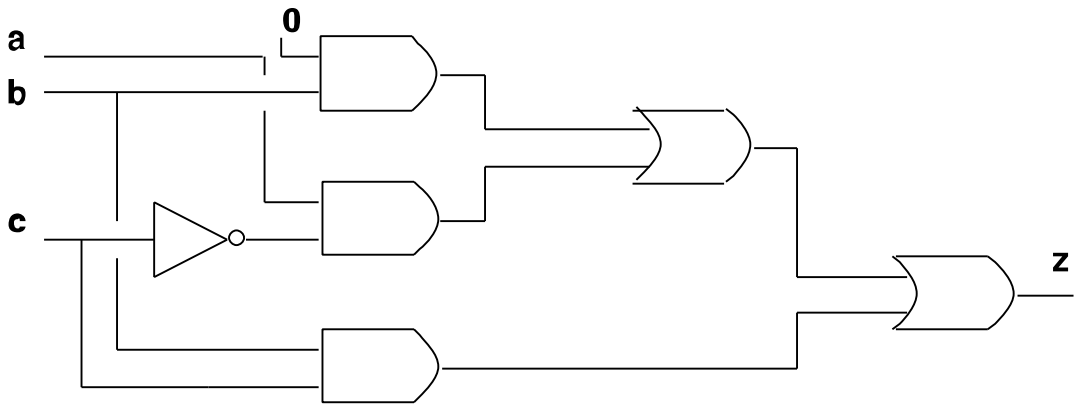
Synthesis and Testability

- Assumptions
  - combinational circuit
  - single or multiple *stuck-at* faults
- Redundancy removal
  - use tests to search for untestable faults and reduce circuit
- Algorithm:
  - if stuck-at-0 on net y is untestable
    - set y = 0
    - propagate constant, reducing logic
  - if stuck-at-1 on net y is untestable
    - set y = 1
    - propagate constant, reducing logic

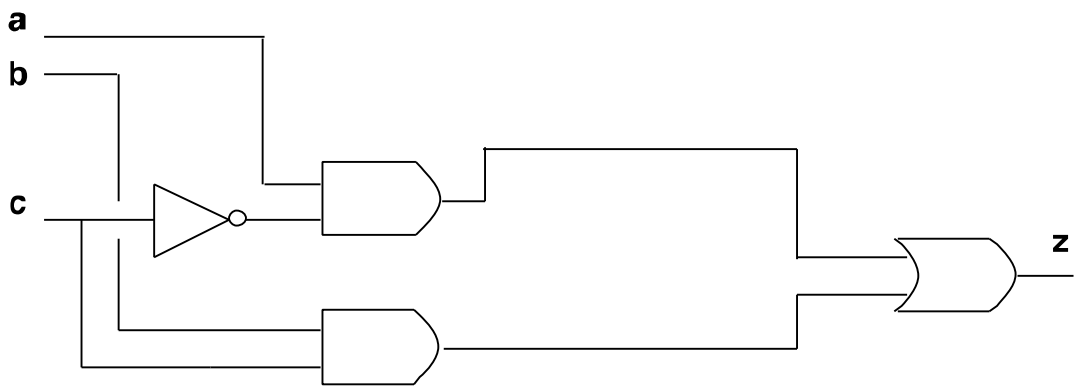
Example



(a)



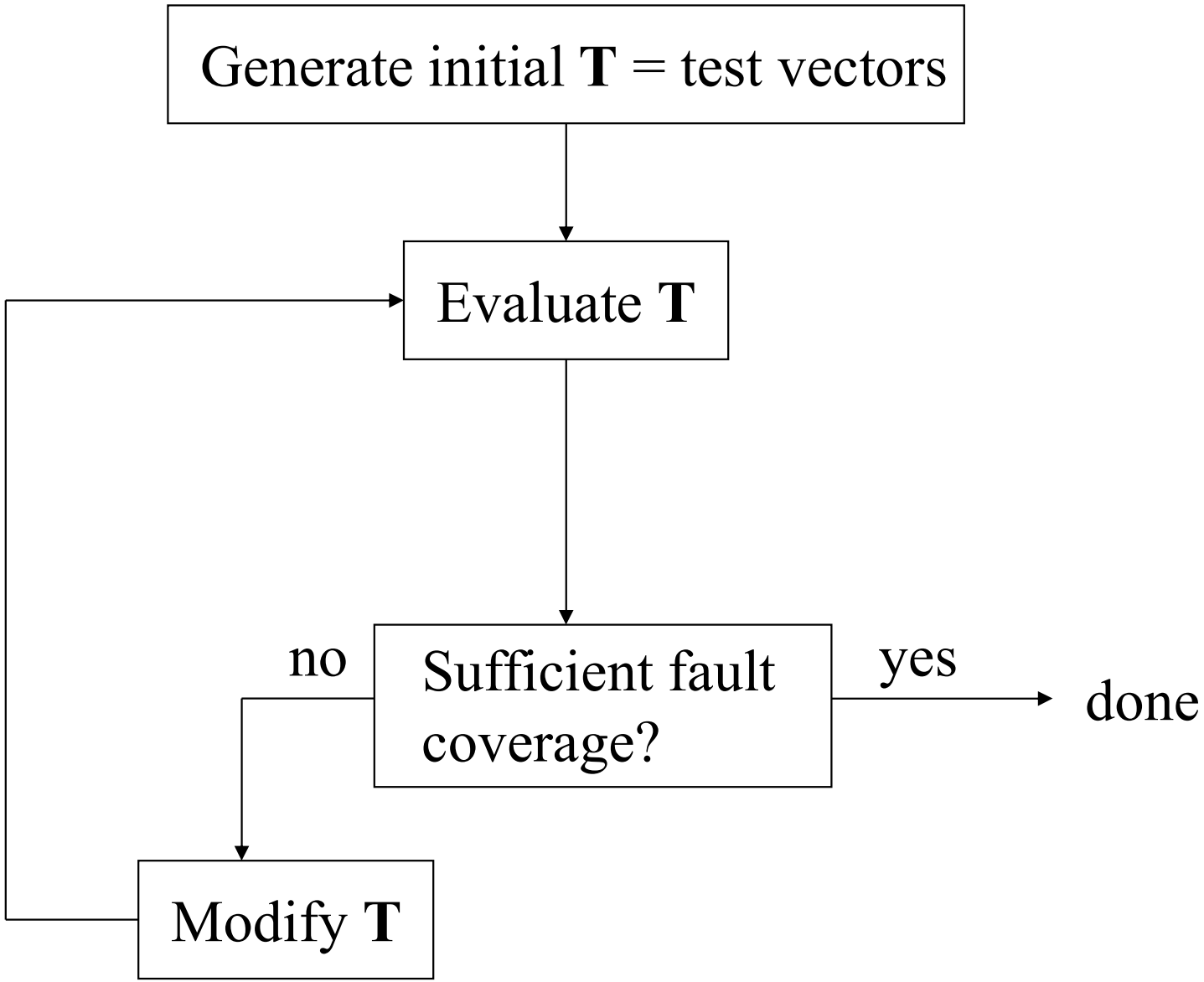
(b)

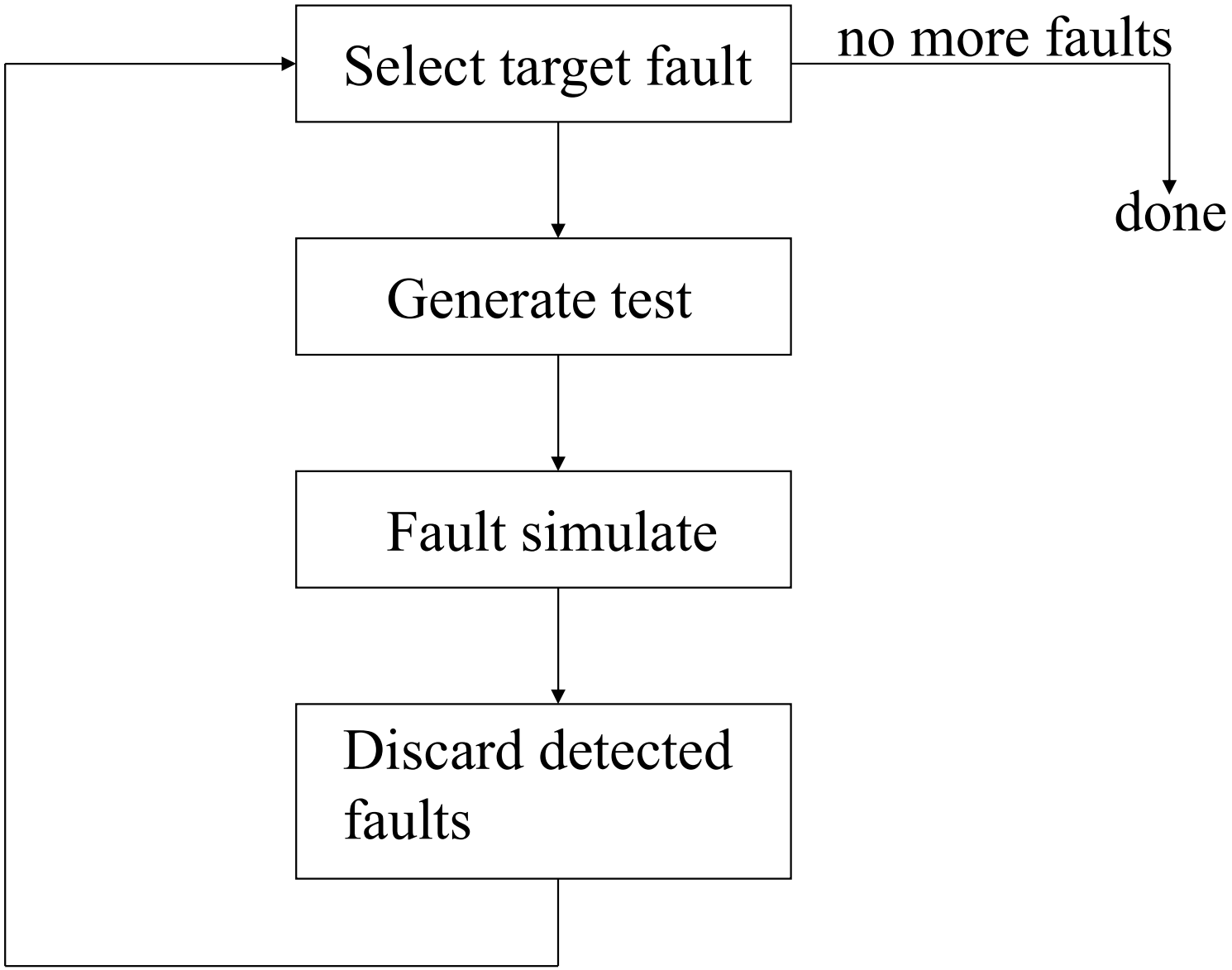


(c)

Fault Simulation

- Fault simulation = simulation of the circuit in the presence of faults
- To evaluate (grade) a test, we use
  - Y = manufacturing yield = probability that a manufactured circuit is defect-free
  - DL = defect level = probability of shipping a defective product
  - d = defect coverage of test
  - $DL = 1 - Y^{1-d}$
- if Y = 0.5 and we desire DL = 0.01
  => d = 99% fault coverage



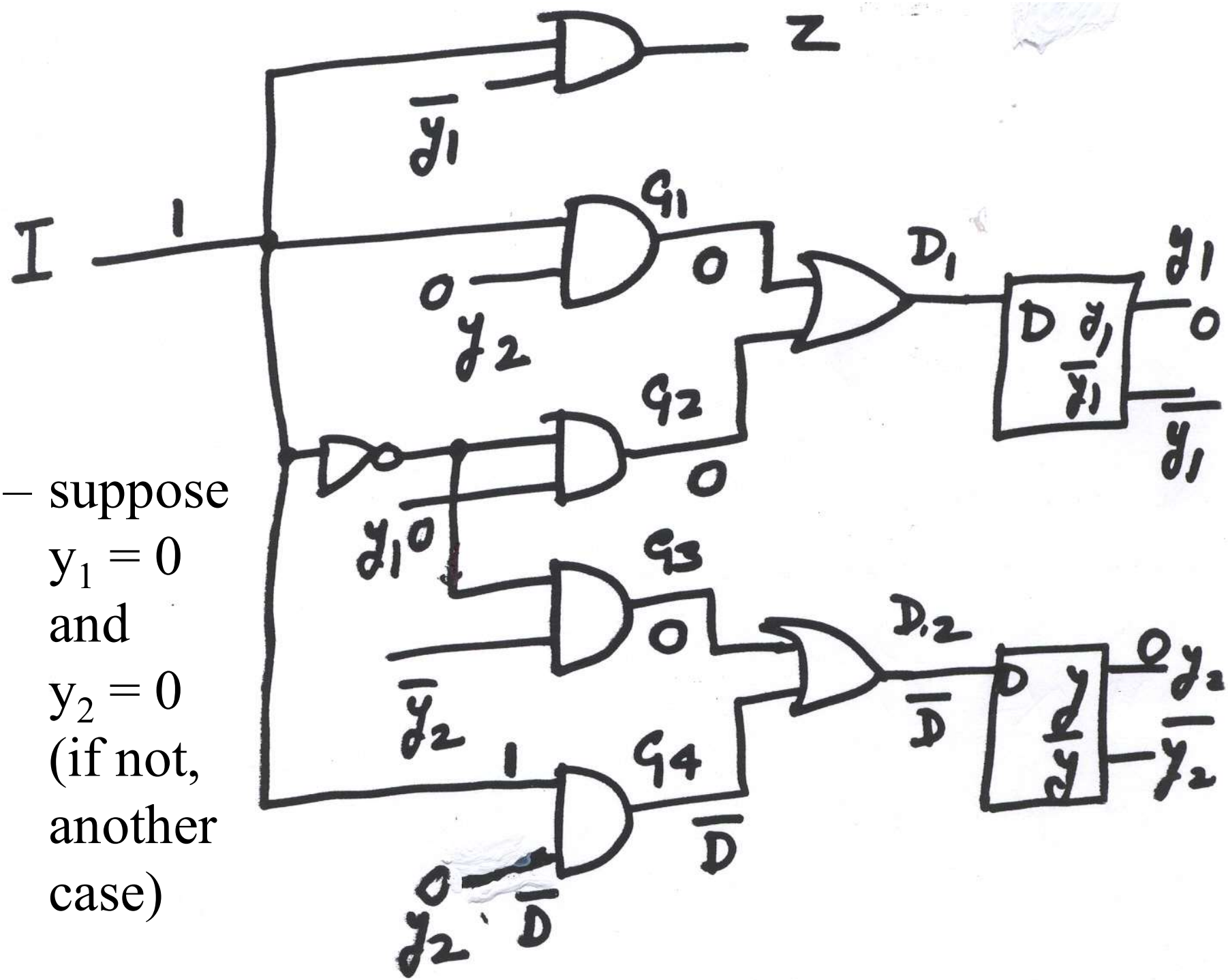


- Fault simulation
  - target faults during test generation
  - used to construct fault dictionaries

Observability Problem

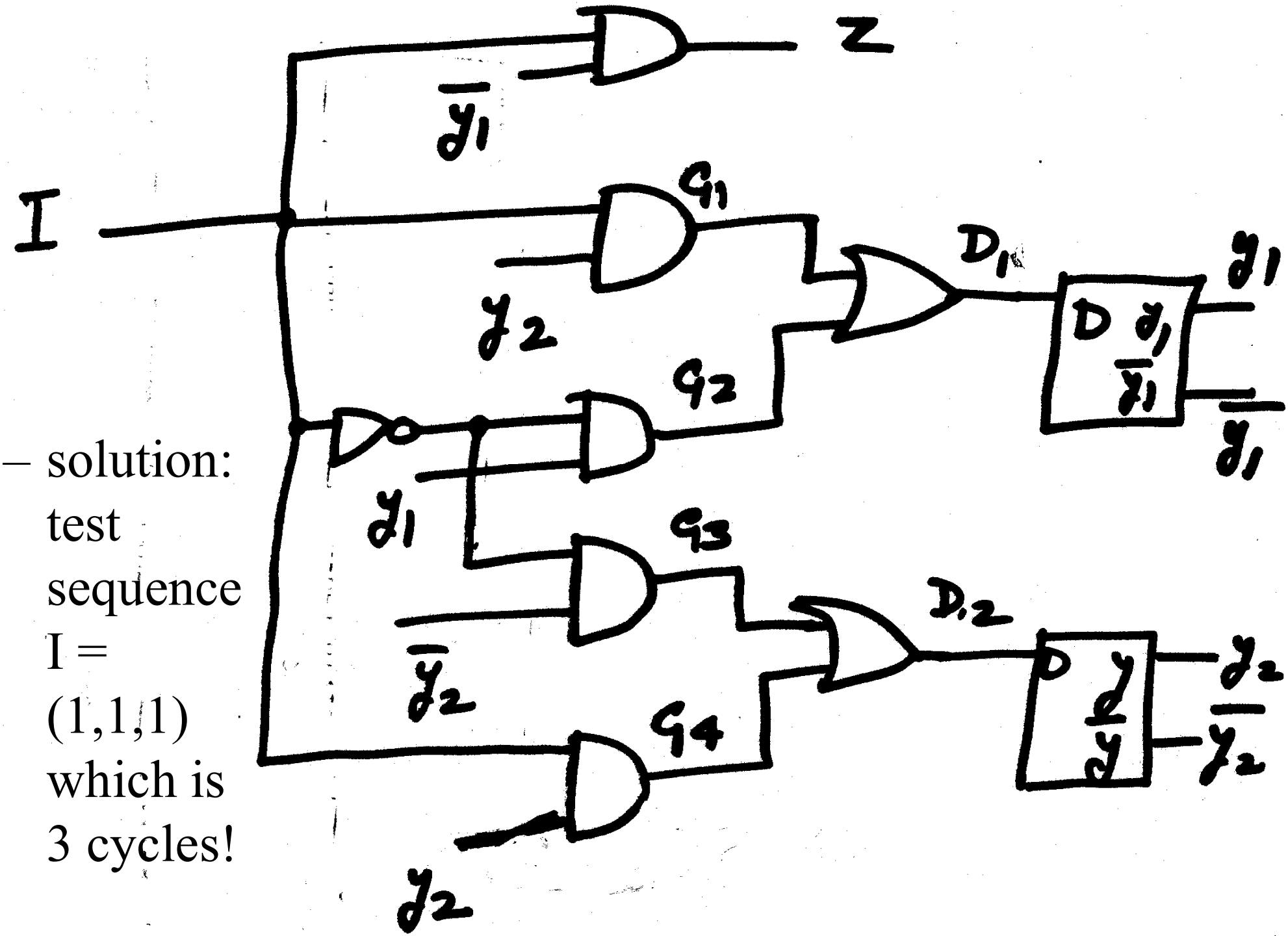
- Not all faults can be observed right away
  - need to propagate faults to output
- For n FFs
  - 2^n states
  - $2^n \cdot 2^n$ state sequences
- While searching for a test sequence, save every sequence that propagates an error to an internal state variable

Example



– suppose $y_1 = 0$ and $y_2 = 0$ (if not, another case)

Example (continued)

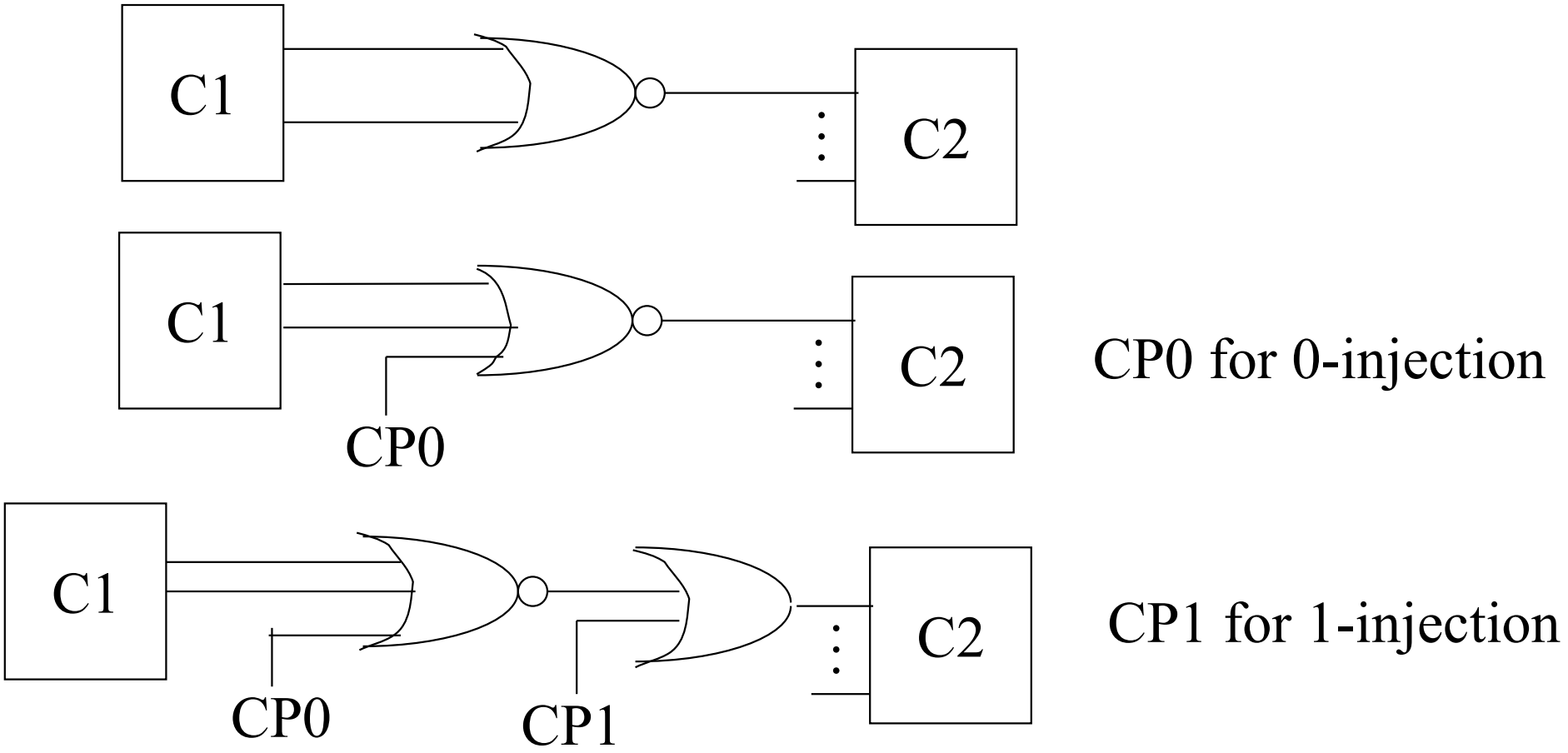


– solution: test sequence I = (1,1,1) which is 3 cycles!

Design for Testability

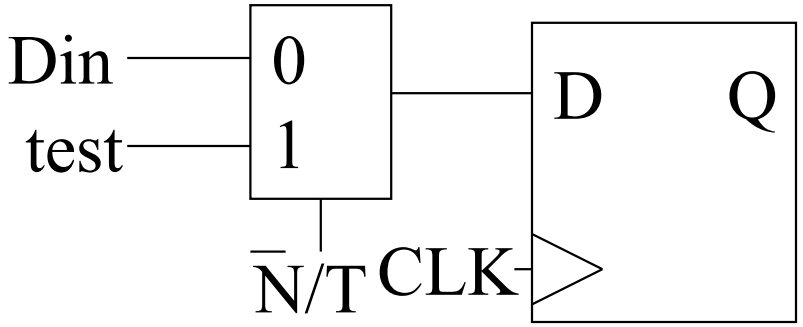
- Status (normal, inoperable, degraded) of a device to be determined
- DFT allows cost-effective development of tests to determine status
- Controllability
  - setting net *y* to a 1 (1-controllability)
  - setting net *y* to a 0 (0-controllability)
- Observability
  - ability to drive an error from net *y* to a primary output

Test Point Insertion

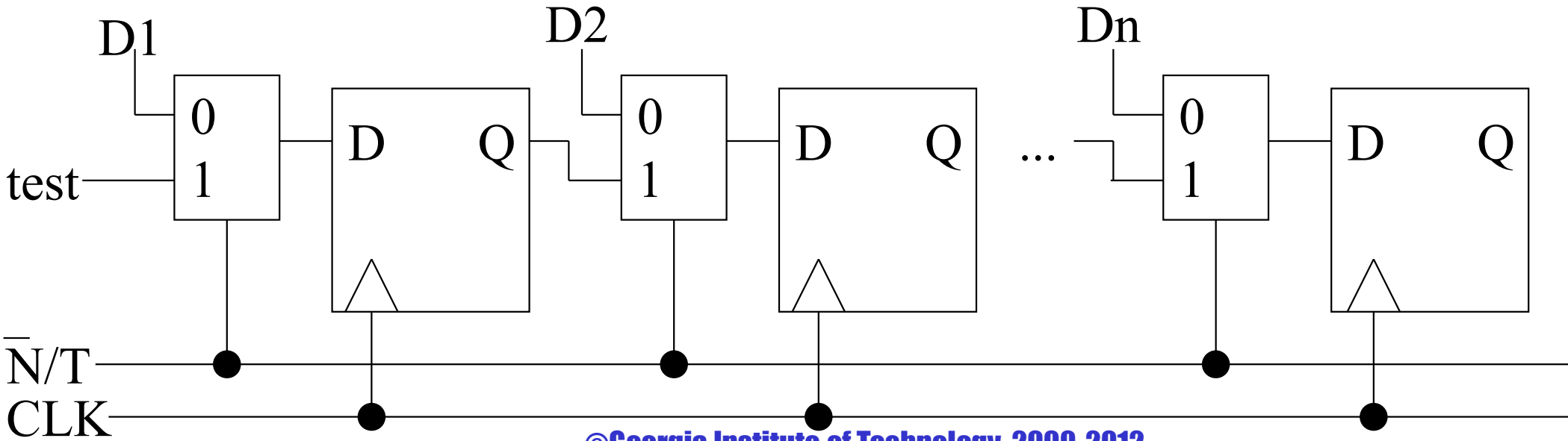


- Design circuits to be initializable
- Partition large circuits into smaller subcircuits in order to reduce test cost

Scan Registers



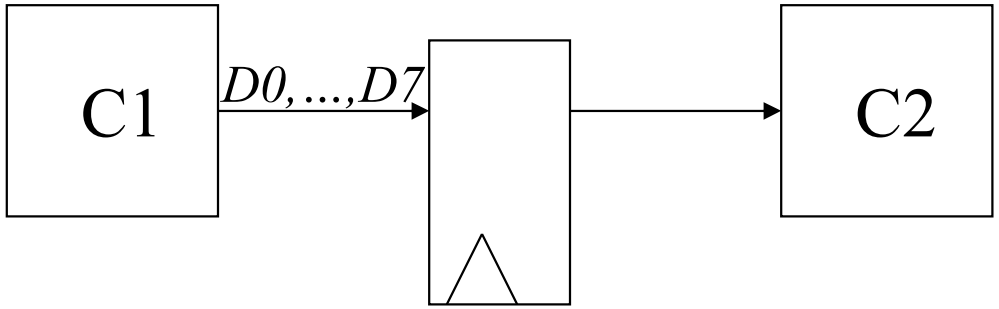
- $\overline{N}/T = 0$ => data loaded from data line Din
  - normal mode
- $\overline{N}/T = 1$ => data loaded from test input
  - test mode



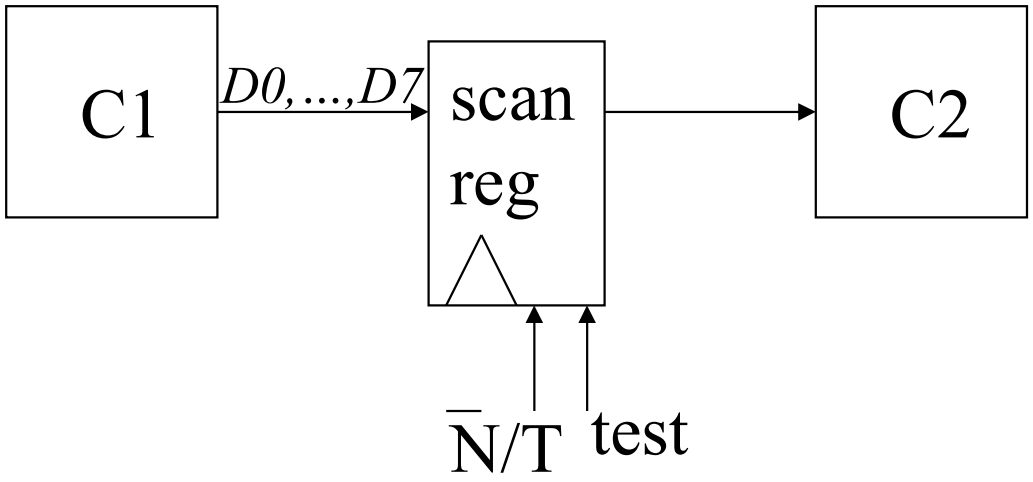
Scan Register Usage

- BIST
  - Built-In-Self-Test
- Boundary Scan

Scan Register Usage

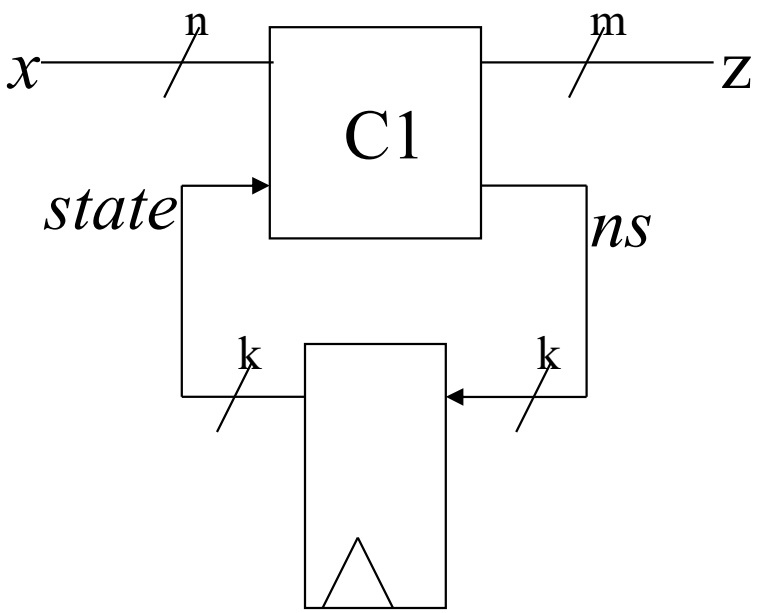


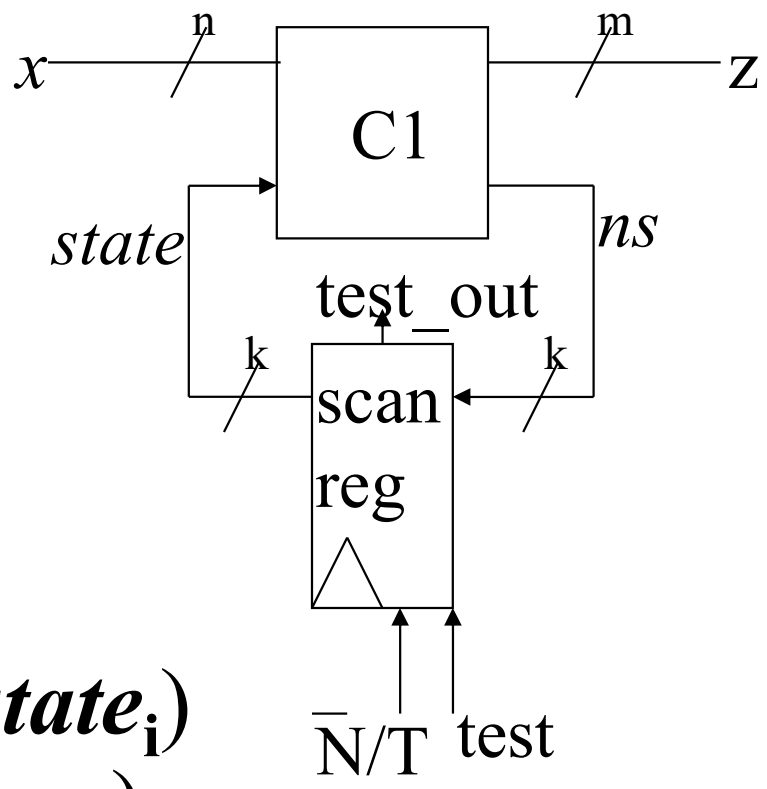
replace with



- $\overline{N}/T = 0$ => normal mode
- $\overline{N}/T = 1$ => data loaded serially from test input

Fully Integrated Serial Scan





- test seq: $(\boldsymbol{x}_1,\boldsymbol{state}_1),\ldots,(\boldsymbol{x}_i,\boldsymbol{state}_i)$
- responses: $(\boldsymbol{z}_1,\boldsymbol{ns}_1)$, …, $(\boldsymbol{z}_i,\boldsymbol{ns}_i)$
- set $\overline{\text{N/T}}$=1, scan $\boldsymbol{state}_1$ into scan reg.
- Apply $\boldsymbol{x}_1$ to input, $\overline{\text{N/T}} = 0$
- next clock edge latches $\boldsymbol{ns}_1$
- set $\overline{\text{N/T}} = 1$, scan out $\boldsymbol{ns}_1$ while scanning in $\boldsymbol{x}_2$
- repeat i times

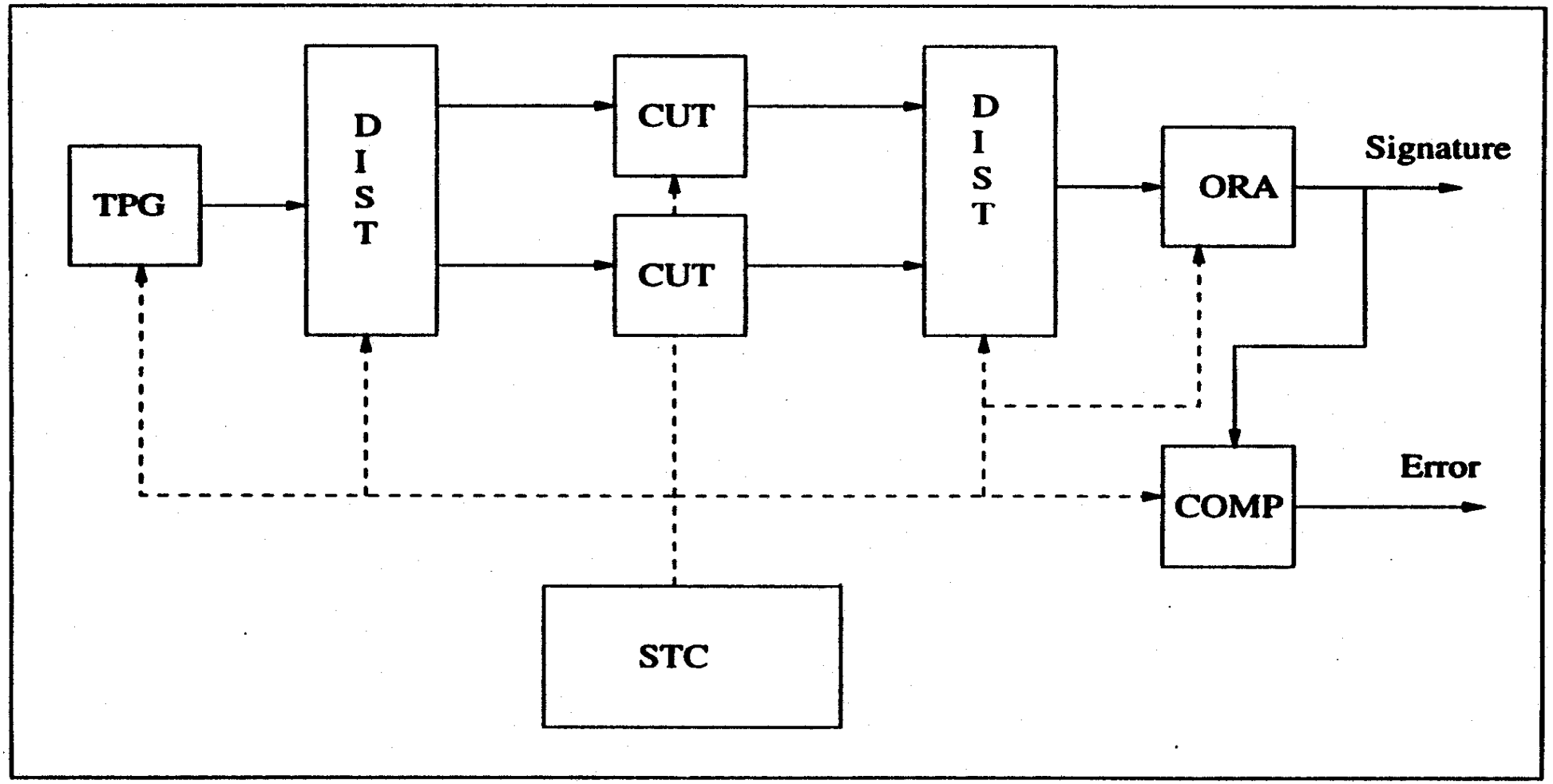
Structured Testability Approaches

Internal Scan

Built-In Self Test

Boundary Scan

BIST ARCHITECTURE



Testing MCMs using IEEE 1149.1

Dies in an MCM are BS standard compatible.

BS registers in the dies are connected to form a serial register chain.

Enables interconnect test of dies in an MCM.

Provides application of BIST of ICs.

EXTEST, SAMPLE, INTEST and RUNBIST modes of test.

Boundary Scan Standard Compatible Chip