Fault-Tolerant Computing

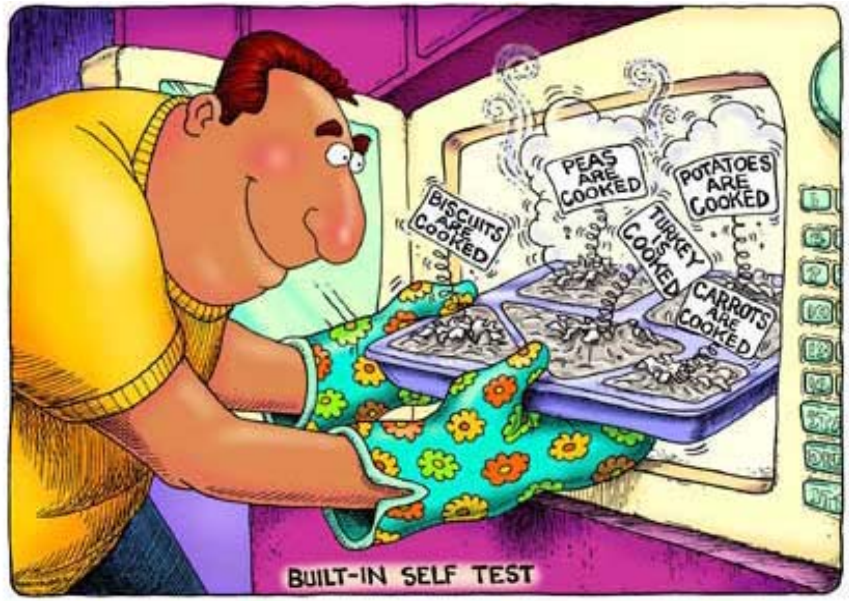
About This Presentation

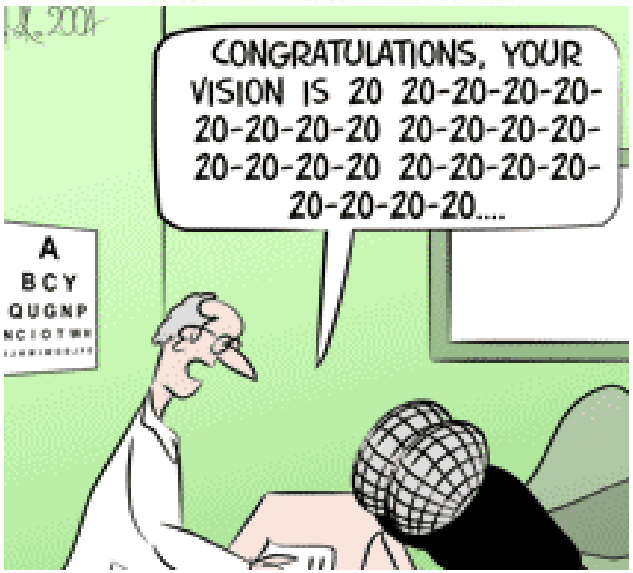
This presentation has been prepared for the graduate course ECE 257A (Fault-Tolerant Computing) by Behrooz Parhami, Professor of Electrical and Computer Engineering at University of California, Santa Barbara. The material contained herein can be used freely in classroom teaching or any other educational setting. Unauthorized uses are prohibited.

Edition	Released	Revised	Revised
First	Oct. 2006		

Fault Testing


45 METRES OF ROPE AND YOU'RE FINDING FAULT WITH THIS LITTLE BIT ?


PEAS ARE COOKED
POTATOES ARE COOKED
BISCUITS ARE COOKED
TURKEY IS COOKED
CARROTS ARE COOKED
BUILT-IN SELF TEST


CONGRATULATIONS, YOUR VISION IS 20 20-20-20-20-20-20-20-20 20-20-20-20-20-20-20-20 20-20-20-20-20-20-20-20....
A
BCY
QUGNP


The good news is that the tests don't show any other problems

Multilevel Model

Component

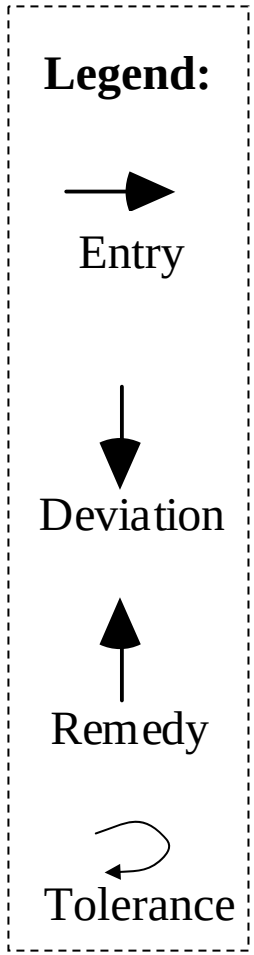
Logic

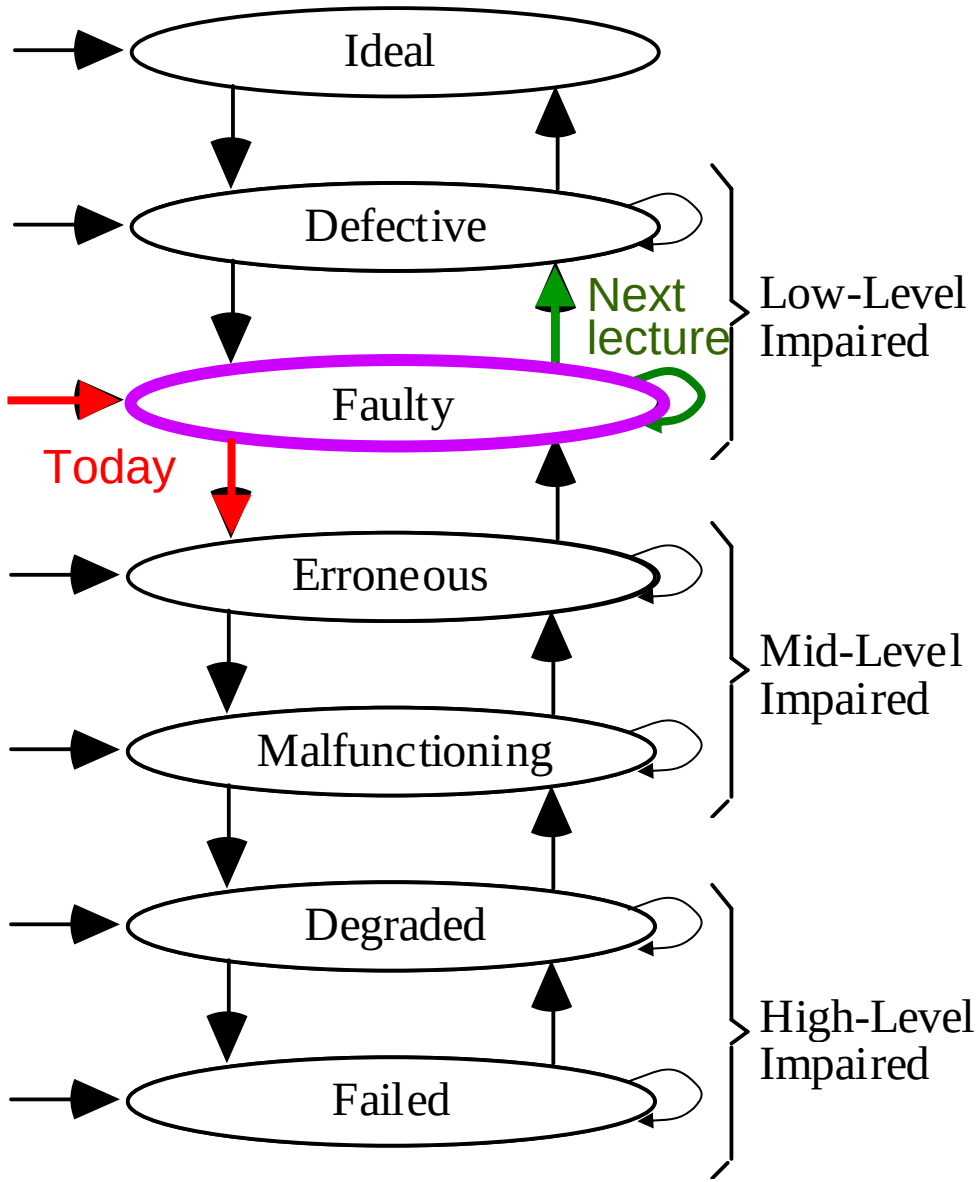
Information

System

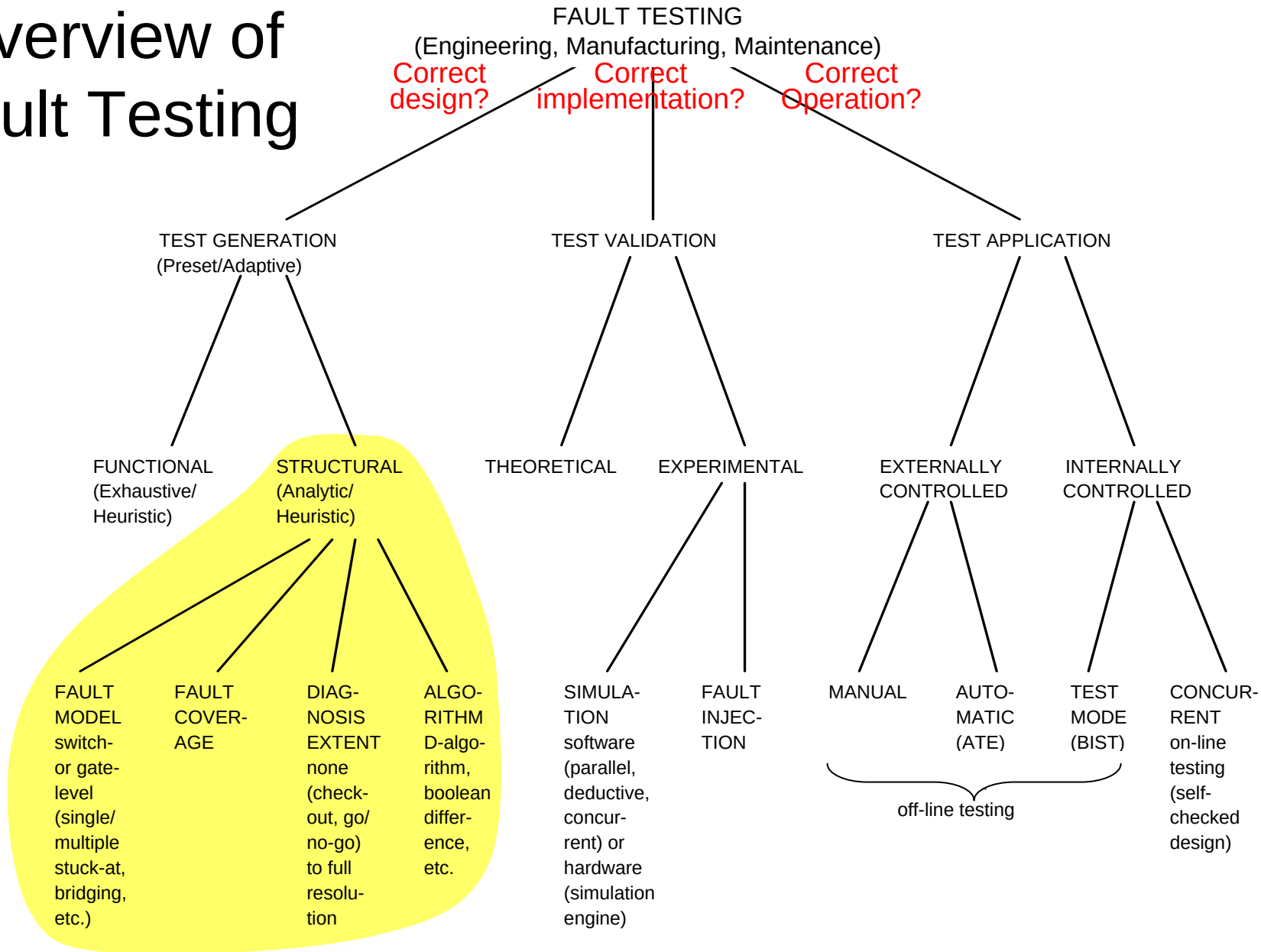
Service

Result

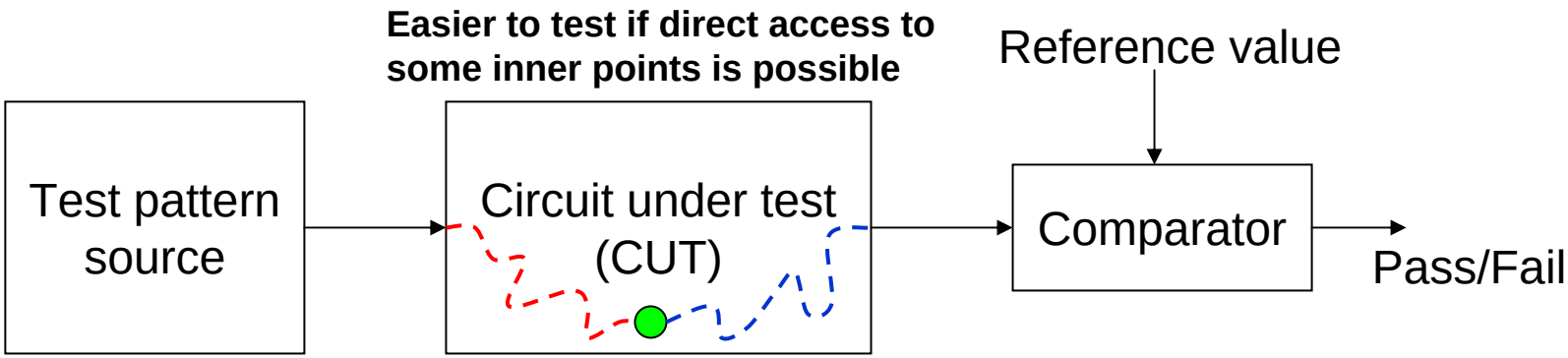




Overview of Fault Testing



Requirements and Setup for Testing



Testability requires **controllability** and **observability**
(redundancy may reduce testability if we are not careful; e.g., TMR)

Reference value can come from a “gold” version or from a table

Test patterns may be randomly generated, come from a preset list, or be selected according to previous test outcomes

Test results may be compressed into a “signature” before comparing

Test application may be off-line or on-line (concurrent)

Importance and Limitations of Testing

Important to detect faults as early as possible
Approximate cost of catching a fault at various levels

Component	$1
Board	$10
System	$100
Field	$1000

Test coverage may be well below 100% (model inaccuracies and impossibility of dealing with all combinations of the modeled faults)

“Trying to improve software quality by increasing the amount of testing is like trying to lose weight by weighing yourself more often.” Steve C. McConnell

“Program testing can be used to show the presence of bugs, but never to show their absence!” Edsger W. Dijkstra

Fault Models at Different Abstraction Levels

Fault model is an abstract specification of the types of deviations in logic values that one expects in the circuit under test

Can be specified at various levels: transistor, **gate**, function, system

Transistor-level faults
- Caused by defects, shorts/opens, electromigration, transients, . . .
- May lead to high current, incorrect output, intermediate voltage, . . .
- Modeled as stuck-on/off, bridging, delay, coupling, crosstalk faults
- Quickly become intractable because of the large model space

Function-level faults
- Selected in an ad hoc manner based on the function of a block (decoder, ALU, memory)

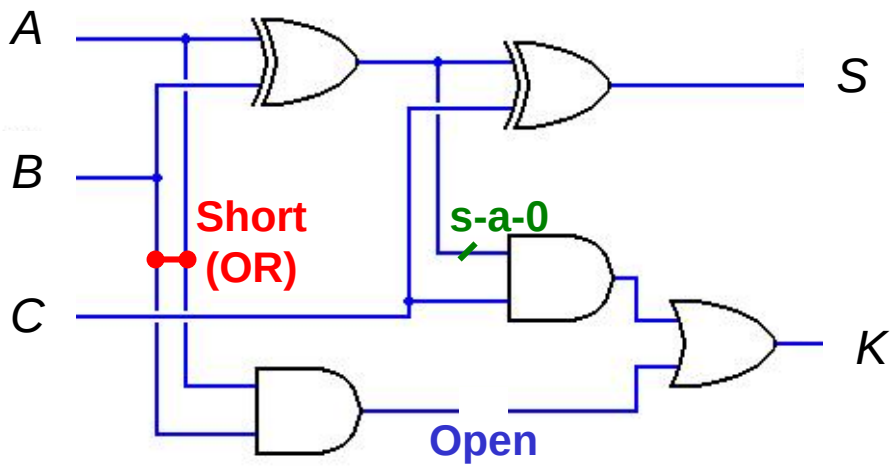
System-level faults (malfunctions, in our terminology)
- Will discuss later in section dealing with mid-level impairments

Gate- or Logic-Level Fault Models

Most popular models (due to their accuracy and relative tractability)

Line stuck faults
- Stuck-at-0 (s-a-0)
- Stuck-at-1 (s-a-1)

Line bridging faults
- Unintended connection (wired OR/AND)



Line open faults
- Often can be modeled as s-a-0 or s-a-1

Delay faults (less tractable than the previous fault types)
- Signals experience unusual delays

Other faults
- Coupling, crosstalk

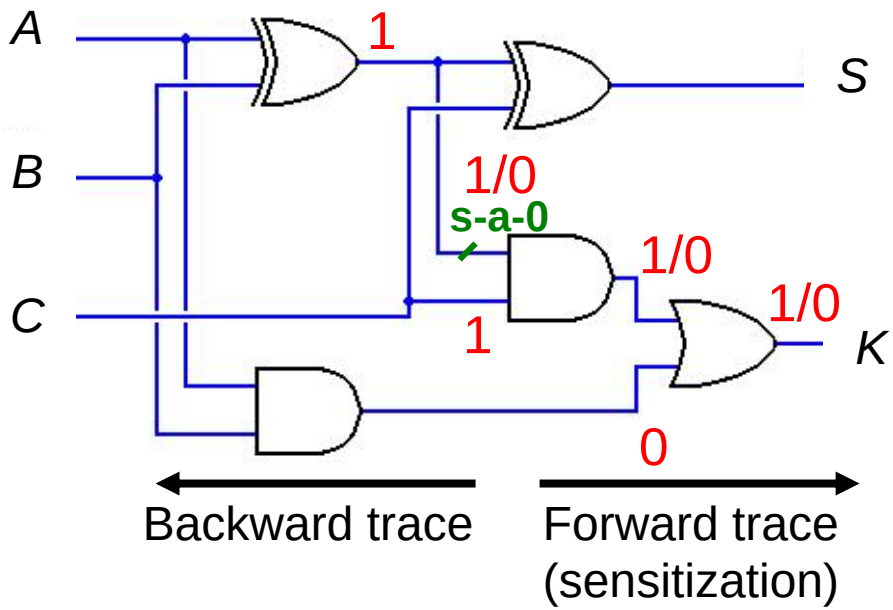
Path Sensitization and D-Algorithm

The main idea behind test design: control the faulty point from inputs and propagate its behavior to some output

Example: s-a-0 fault
Test must force the line to 1

Two possible tests
(*A*, *B*, *C*) = (0 1 1) or (1 0 1)

This method is formalized in the *D*-algorithm



***D*-calculus**

1/0 on the diagram above is represented as *D*

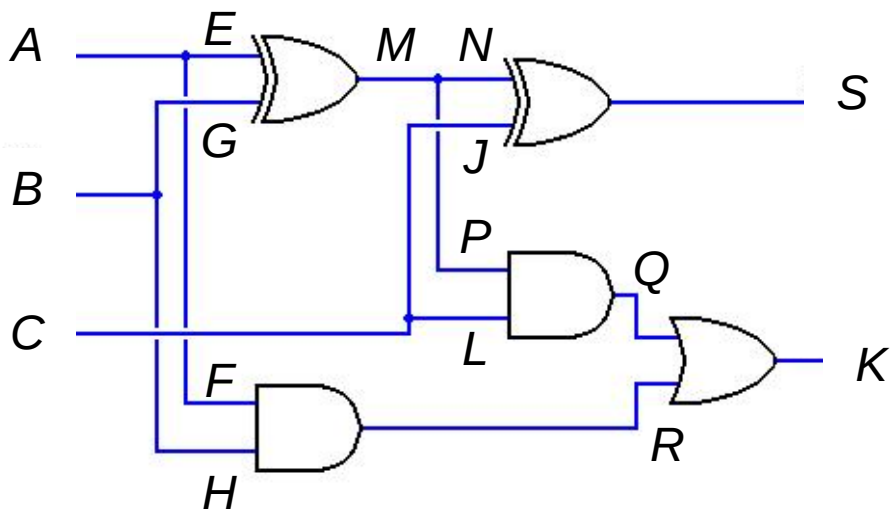
0/1 is represented as $\bar{D}$

Encounters difficulties with XOR gates (PODEM algorithm fixes this)

Selection of a Minimal Test Set

Each input pattern detects a subset of all possible faults of interest (according to our fault model)

A	*B*	*C*	*P*		*Q*	
			s-a-0	s-a-1	s-a-0	s-a-1
0	0	0	-	-	-	x
0	1	1	x	-	x	-
1	0	1	?	?	?	?
1	1	1	?	?	?	?



Choosing a minimal test set is a covering problem

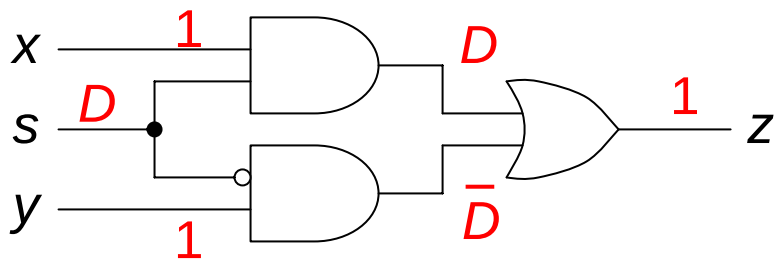
Equivalent faults: e.g., P s-a-0 ≡ L s-a-0 ≡ Q s-a-0

Q s-a-1 ≡ R s-a-1 ≡ K s-a-1

Capabilities and Complexity of D-Algorithm

Reconvergent fan-out
- Consider the *s* input s-a-0

Simple path sensitization does not allow us to propagate the fault to the primary output *z*



PODEM solves the problem by setting *y* to 0

Worst-case complexity of D-algorithm is exponential in circuit size
- Must consider all path combinations
- XOR gates cause the behavior to approach the worst case
- Average case is much better; quadratic

PODEM: Path-oriented decision making
- Developed by Goel in 1981
- Also exponential, but in the number of circuit inputs, not its size

Boolean Difference

$K = f(A, B, C) = AB \vee BC \vee CA$

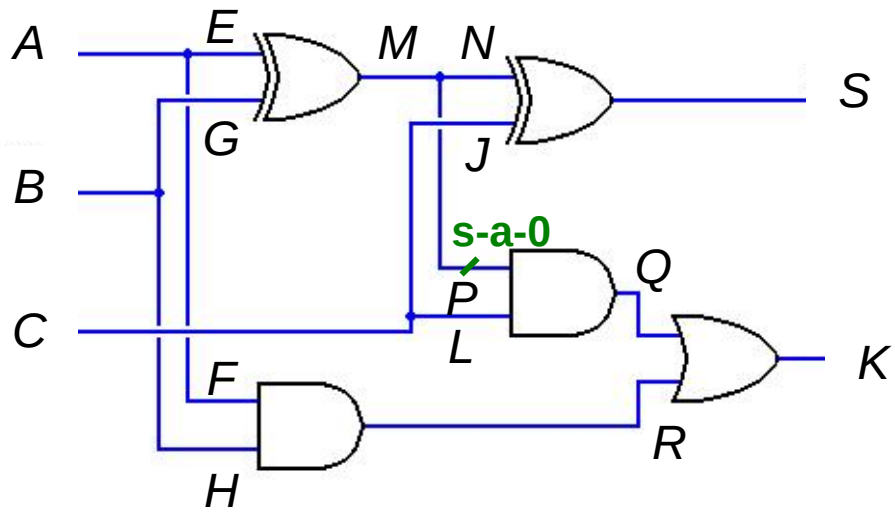
$dK/dB = f(A, 0, C) \oplus f(A, 1, C)$

$= CA \oplus (A \vee C)$

$= A \oplus C$

$K = PC \vee AB$

$dK/dP = AB \oplus (C \vee AB) = C(\overline{AB})$



Tests that detect P s-a-0 are solutions to the equation $P\ dK/dP = 1$

$(A \oplus B)\ C(\overline{AB}) = 1 \quad \Rightarrow \quad C = 1, A \neq B$

Tests that detect P s-a-1 are solutions to the equation $\overline{P}\ dK/dP = 1$

$(\overline{A \oplus B})\ C(\overline{AB}) = 1 \quad \Rightarrow \quad C = 1, A = B = 0$

Complexity of Fault Testing

The satisfiability problem (SAT)
Decision problem: Is a Boolean expression satisfiable?
(i.e., can we assign values to the variables to make the result 1?)

Theorem (Cook, 1971): SAT is NP-complete
In fact, even restricted versions of SAT remain NP-complete

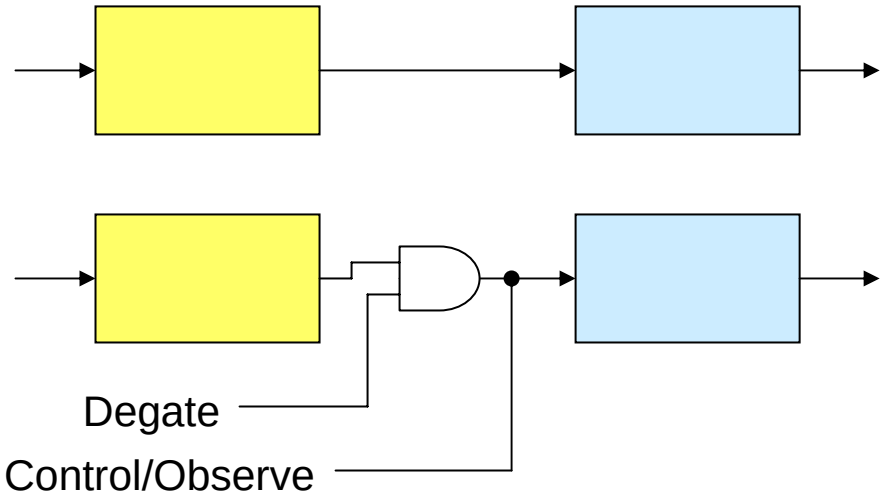
According to the Boolean difference formulation, test generation can be converted to SAT (find the solutions to $P\,dK/dP = 1$)

To prove the NP-completeness of test generation, we need to show that SAT (or some other NP-complete problem) can be converted to test generation

For a simple proof, see [Fuji85], pp. 113-114

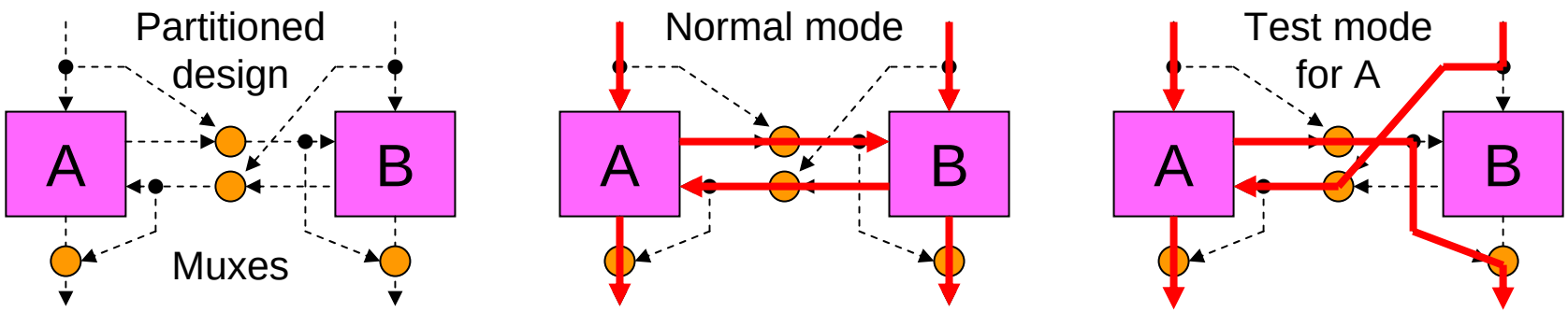
Design for Testability: Combinational

Increase controllability and observability via the insertion of degating mechanisms and control points



Design for dual-mode operation

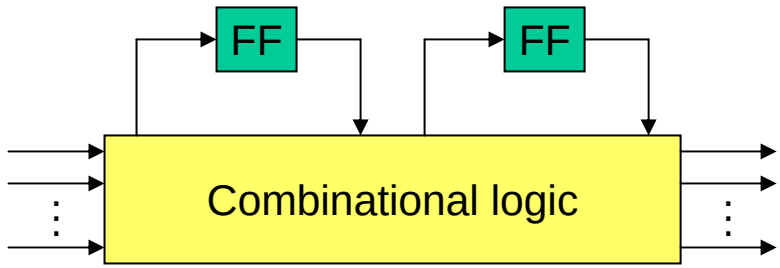
- Normal mode
- Test mode

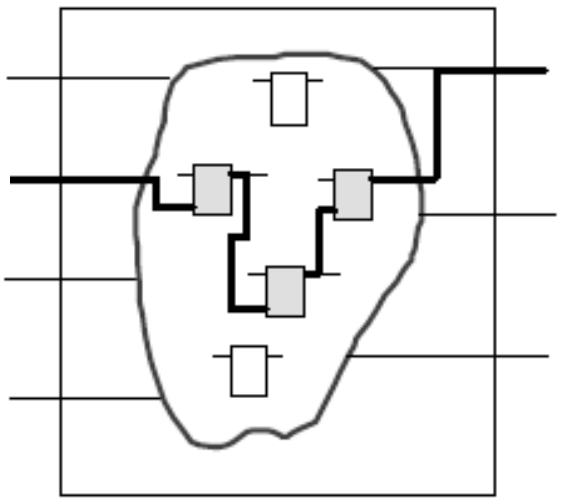


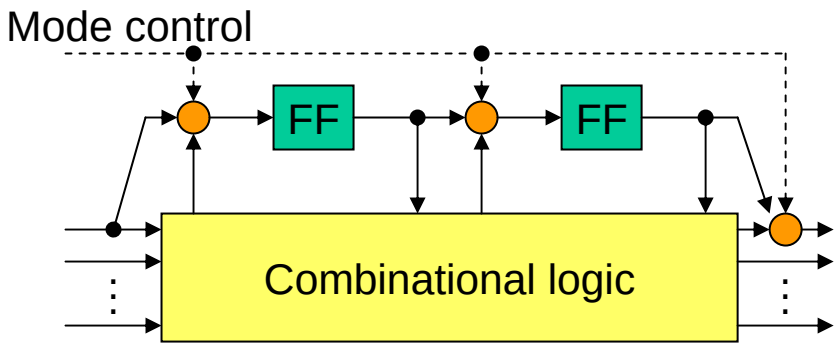
Design for Testability: Sequential

Increase controllability and observability via provision of mechanisms to set and observe internal flip-flops

Scan design
- Shift desired states into FF
- Shift out FF states to observe



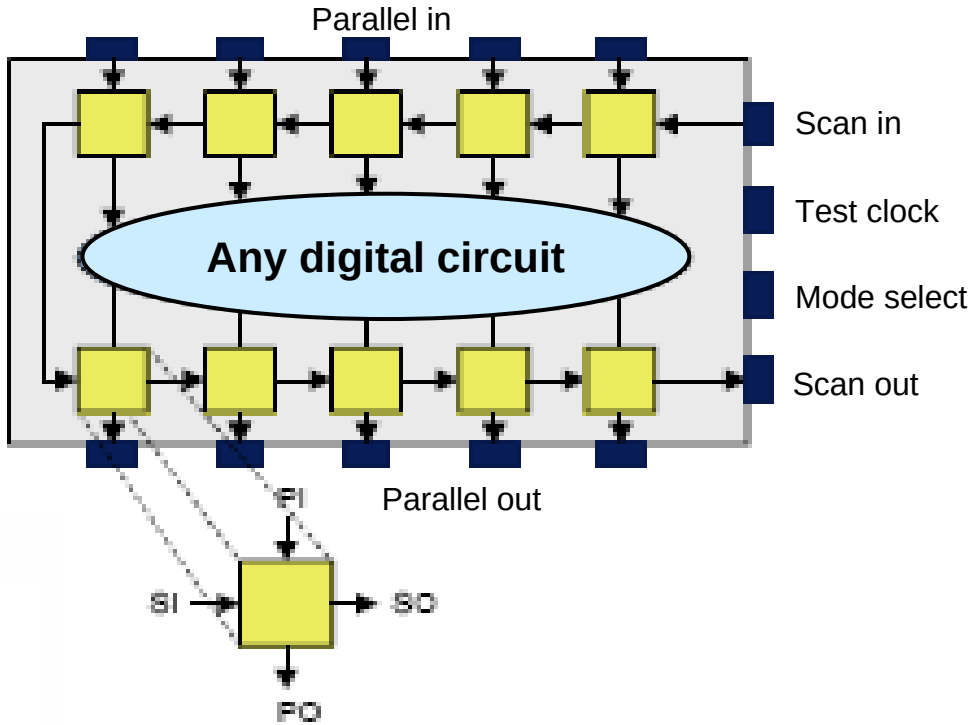


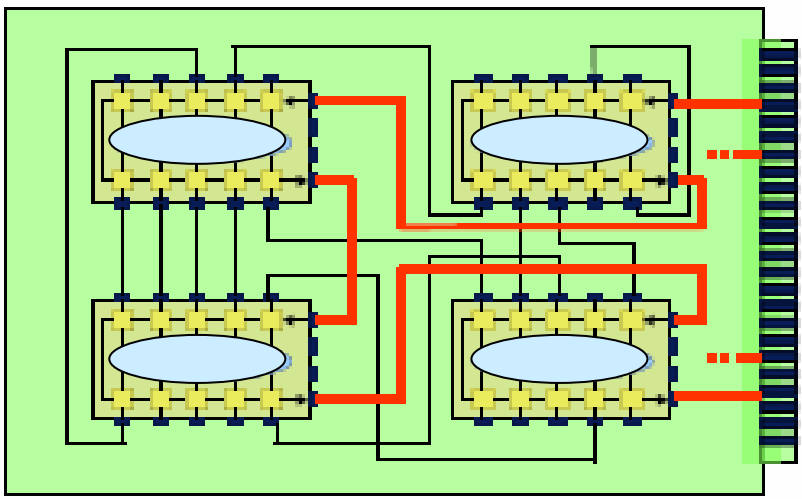


Partial scan design:
Mitigates the excessive overhead of a full scan design

Boundary Scan for Board-Level Testing

Lets us apply arbitrary inputs to circuit parts whose inputs would otherwise not be externally accessible

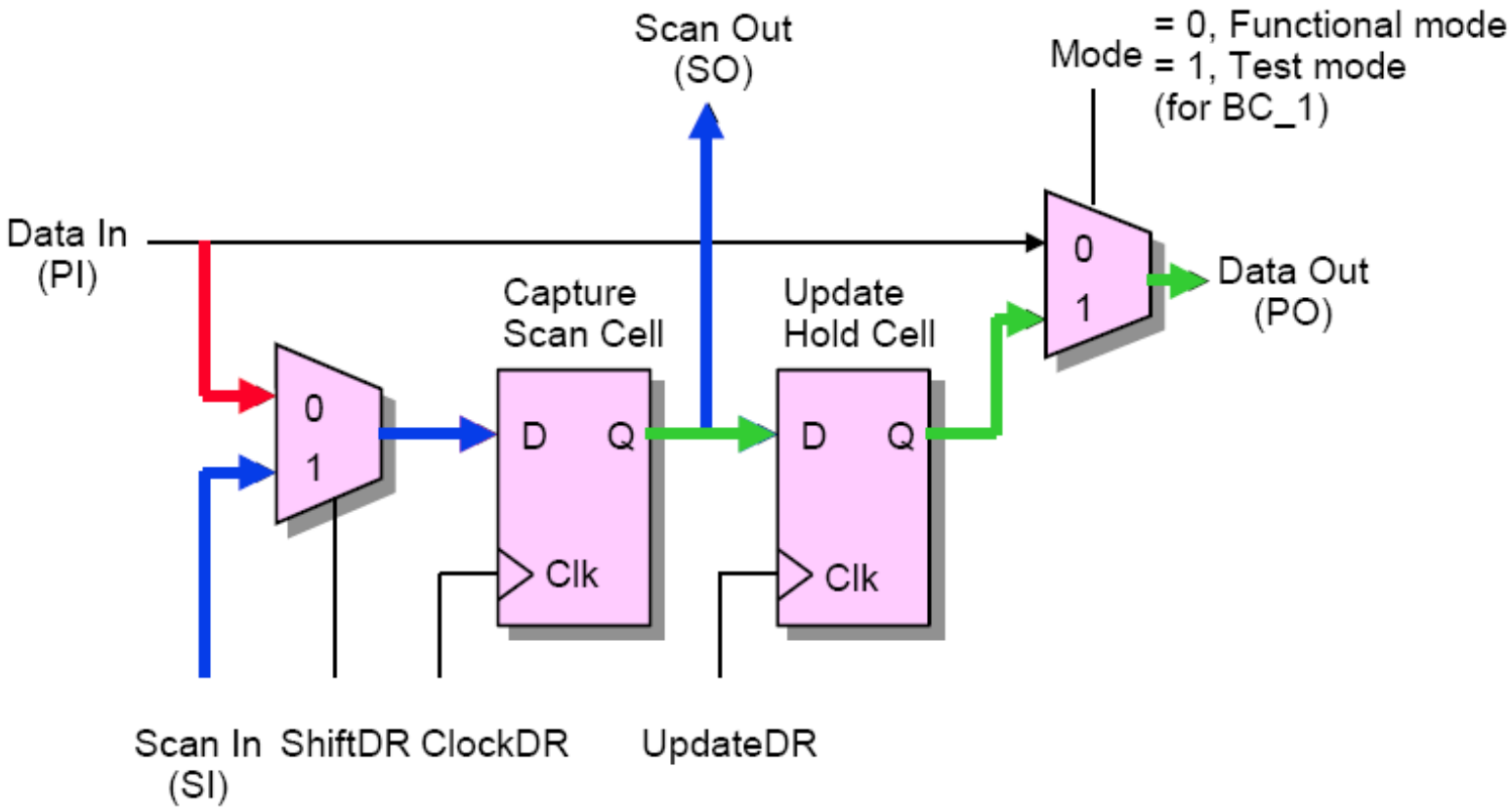




Boundary scan elements of multiple parts are cascaded together into a scan path

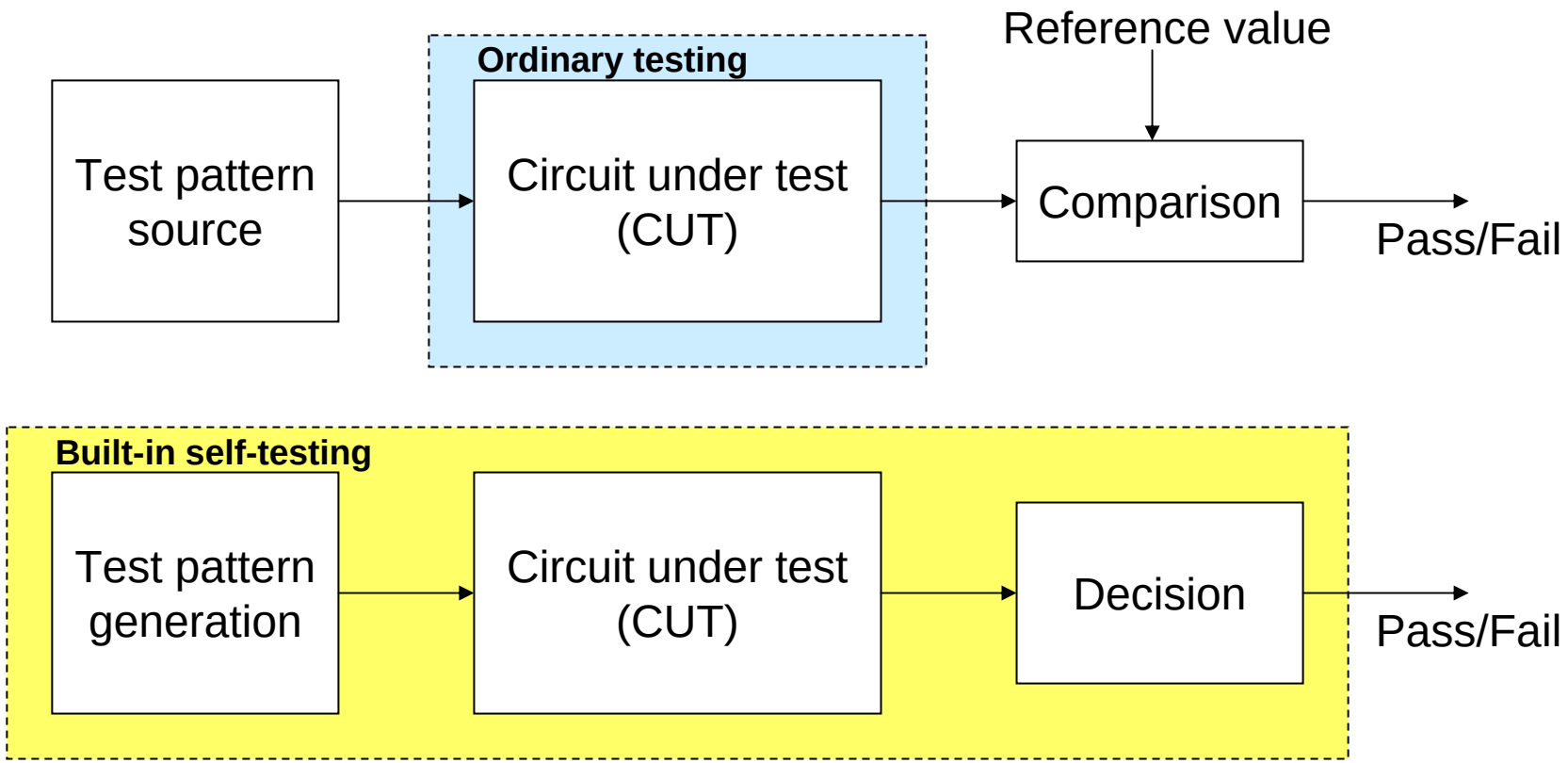
From: http://www.asset-intertech.com/pdfs/boundaryscan_tutorial.pdf

Basic Boundary Scan Cell



From: http://www.asset-intertech.com/pdfs/boundaryscan_tutorial.pdf

Built-in Self-Test (BIST)



Test patterns may be generated (pseudo)randomly – e.g., via LFSRs

Decision may be based on compressed test results

Quantifying Testability: Controllability

Controllability C of a line has a value between 0 and 1

Controllability transfer factor

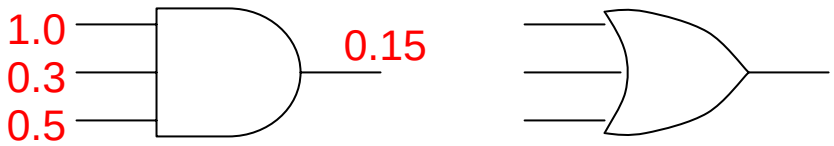
$$CTF = 1 - \left| \frac{N(0) - N(1)}{N(0) + N(1)} \right|$$

$$C_{output} = (\Sigma_i C_{input\ i} / k) \times CTF$$

k-way fan-out

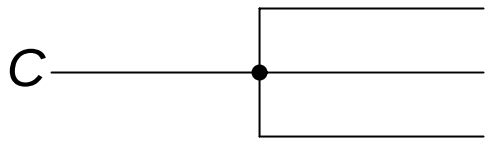
A line with very low controllability is a good place for test point insertion

k-input, 1-output components

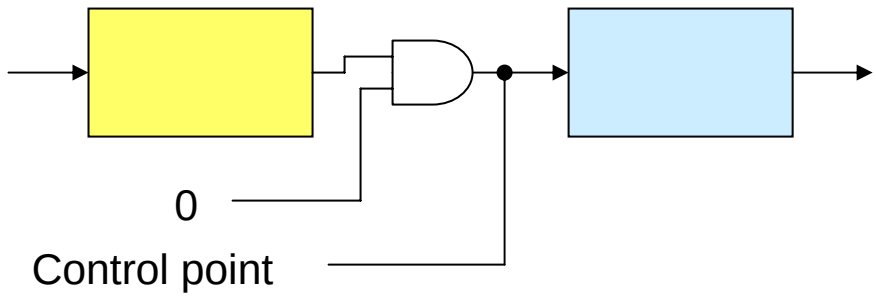


$N(0) = 7$
$N(1) = 1$
$CTF = 0.25$

$N(0) = 1$
$N(1) = 7$
$CTF = 0.25$



$C / (1 + \log k)$ for each of k



Quantifying Testability: Observability

Observability O of a line has a value between 0 and 1

Observability transfer factor

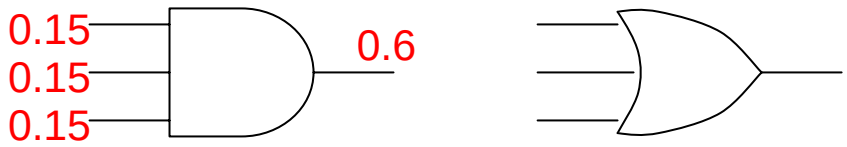
$$OTF = \frac{N(sp)}{N(sp) + N(ip)}$$

$$O_{input} = O_{output} \times OTF$$

k-way fan-out

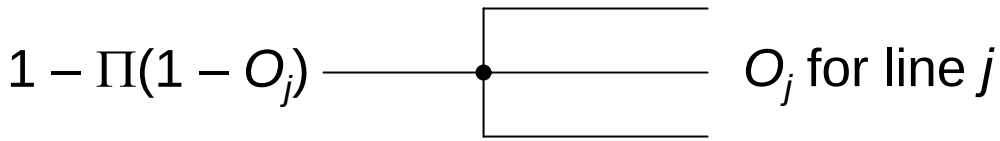
A line with very low observability is a good place for test point insertion

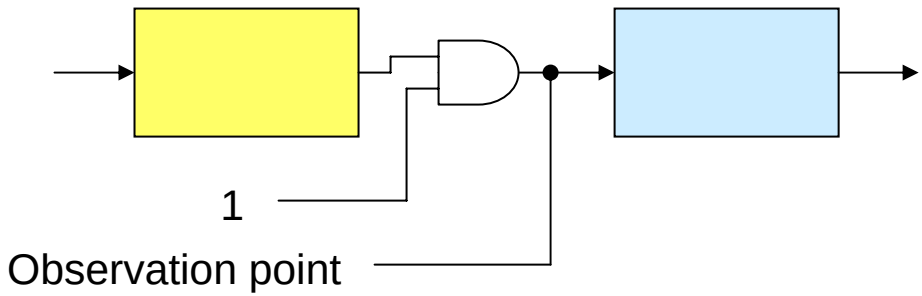
k-input, 1-output components



$N(sp) = 1$
$N(ip) = 3$
$OTF = 0.25$

Sensitizations
Inhibitions

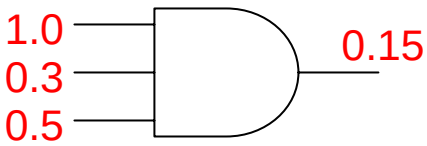




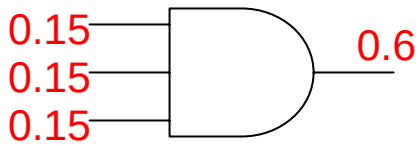
Quantifying Testability: Combined Measure

Testability = Controllability × Observability

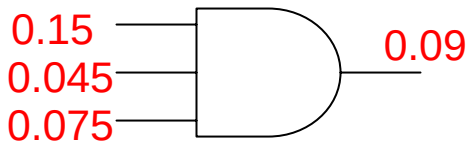
Controllabilities



Observabilities



Testabilities



Overall testability of a circuit = Average of line testabilities