

VLSI Design : 2021-22

Lecture 19

CMOS Testing

By Dr. Sanjay Vidhyadharan

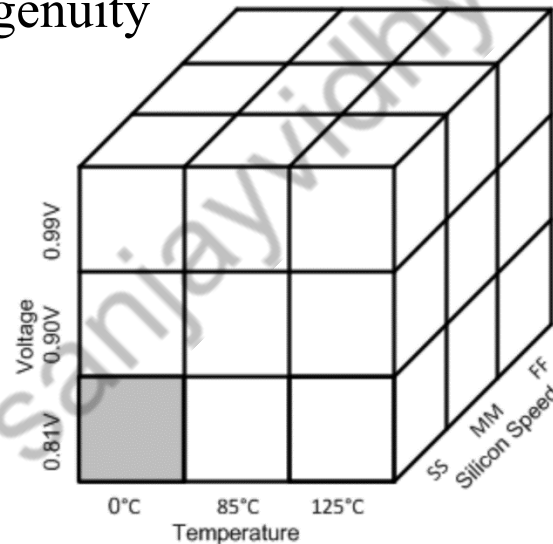
Testing

- Testing is one of the most expensive parts of chips
 - Logic verification accounts for $> 50\%$ of design effort for many chips
 - Debug time after fabrication has enormous cost
 - Shipping defective parts can sink a company
- Example: Intel FDIIV bug
 - Logic error not caught until $> 1\text{M}$ units shipped
 - Recall cost \$450M (!!!)

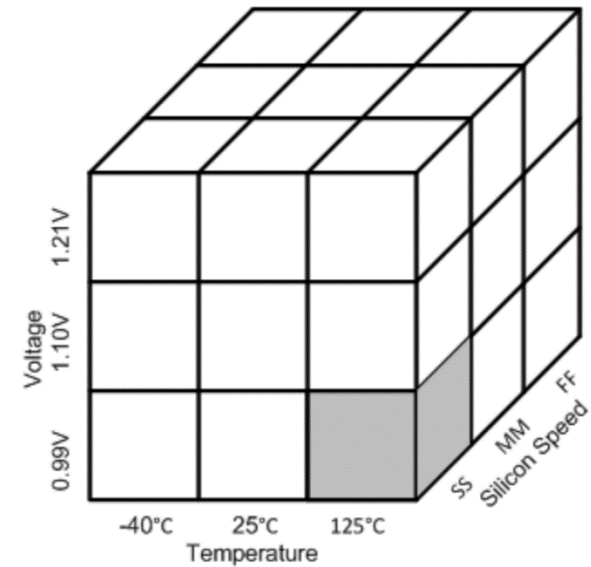
Pre-fabrication Testing / Logic Verification

Does the chip simulate correctly?

- Usually done at HDL level
- Verification engineers write test bench for HDL
- Can't test all cases
- Look for corner cases
- Ex: 32-bit adder
- Test all combinations of corner cases as inputs:
- 0, 1, 2, $2^{31}-1$, -1 , -2^{31} , a few random numbers
- Good tests require ingenuity



Worst Case Corner in 28nm HPM Process



Worst Case Corner in 40nm LP Process

Pre-fabrication Testing / Logic Verification

The **Corner analysis** simulates your design with the minimum and maximum value of each parameter. But it does not reproduce the mismatching between devices!

•Corner Analysis

- CMOS thickness:** wp, ws, wo, wz.
- Resistor value:** wp, ws.
- Capacitor value:** wp, ws.
- Temperatures:** (typ.)-20 to 85°C
- Voltage supply:** depend on your supply source, etc.

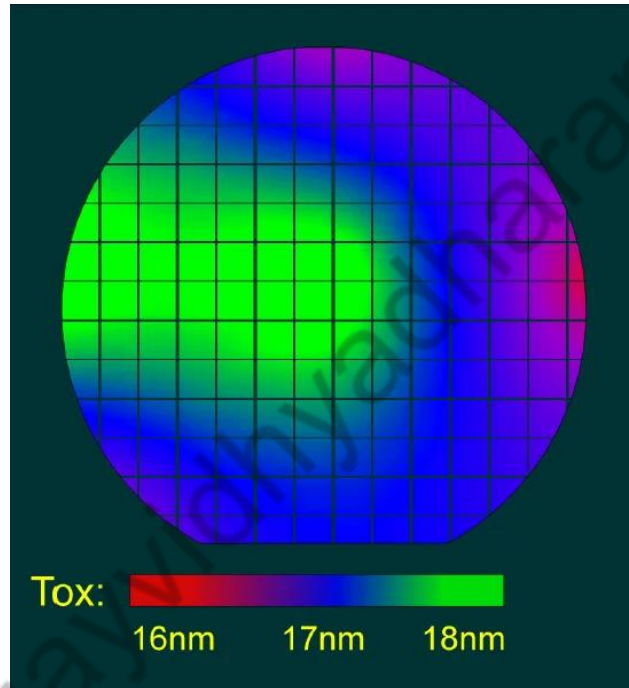
- ws = *worst speed*
- wp = *worst power*
- wo = *worst one (Fast NMOS & Slow PMOS)*
- wz = *worst zero (Slow NMOS & Fast PMOS)*

Corners	<input checked="" type="checkbox"/> Nominal	<input checked="" type="checkbox"/> Corner-vdd
Temperature		-20 85
Design Variables		
vdd		1 1.3
Click to add		
Parameters		
Click to add		
Model Files		
config.scs	<input checked="" type="checkbox"/>	default
param.scs	<input checked="" type="checkbox"/>	3s
bip.scs	<input checked="" type="checkbox"/>	tm
cap.scs	<input checked="" type="checkbox"/>	wp ws
dio.scs	<input checked="" type="checkbox"/>	tm
mos.scs	<input checked="" type="checkbox"/>	ws wp wo wz
photo.scs	<input checked="" type="checkbox"/>	tm
res.scs	<input checked="" type="checkbox"/>	wp ws
xh018.scs	<input type="checkbox"/>	<section>
Click to add		
Model Group(s)		<modelgroup>
Click to add		
Tests		
<input checked="" type="checkbox"/> ...invThy_vco7:1	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Number of Corners	1	64

Typical configuration for a corner simulation

Pre-fabrication Testing / Logic Verification

Monte Carlo analysis is a statistical way to analyze a circuit in VLSI.



On **each simulation run**, it **calculates every parameter randomly** according to a statistical distribution model. The **drawback** of Monte Carlo is the large **number of simulations** required to have acceptable results. It should be **at least 250** to have a significant sample

Design for Testability

Observability & Controllability

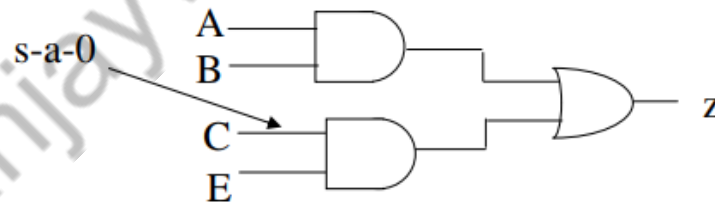
- *Observability*: ease of observing a node by watching external output pins of the chip
- *Controllability*: ease of forcing a node to 0 or 1 by driving input pins of the chip
- Combinational logic is usually easy to observe and control
- Finite state machines can be very difficult, requiring many cycles to enter desired state
 - Especially if state transition diagram is not known to the test engineer

Stuck-At Faults

How does a chip fail?

- Usually failures are shorts between two conductors or opens in a conductor
- This can cause very complicated behavior
- A simpler model: *Stuck-At*
 - Assume all failures cause nodes to be “stuck-at” 0 or 1, i.e. shorted to GND or V_{DD}
 - Not quite true, but works well in practice

SSL Fault Detection



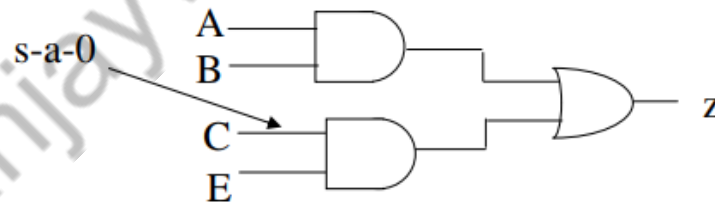
$ABCE = 0011$ is a test pattern for C s-a-0

Stuck-At Faults

How does a chip fail?

- Usually failures are shorts between two conductors or opens in a conductor
- This can cause very complicated behavior
- A simpler model: *Stuck-At*
 - Assume all failures cause nodes to be “stuck-at” 0 or 1, i.e. shorted to GND or V_{DD}
 - Not quite true, but works well in practice

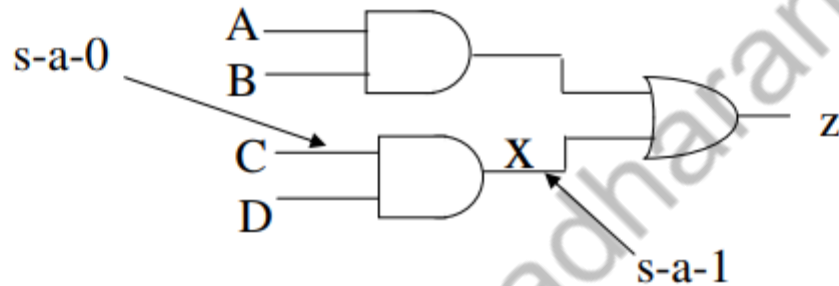
SSL Fault Detection



$ABCE = 0011$ is a test pattern for C s-a-0

Multiple Stuck-Line (MSF) Faults

- More than one line may be stuck at a logic value



Fault: {C s-a-0, x s-a-1}

How many MSL fault can there be in a circuit with n nodes?

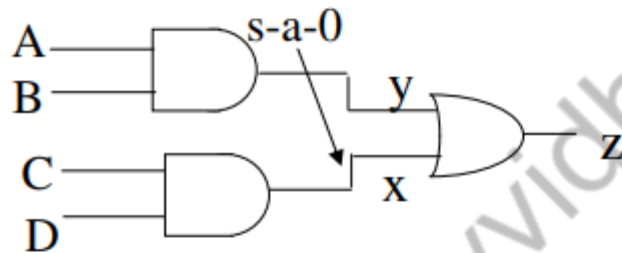
How to get test patterns for MSL faults?

Fault universe is too large, MSL fault model seldom used, especially since tests for SSL faults cover many MSL faults

Test Pattern Generation

- Exhaustive testing: Apply 2^n pattern to n -input circuit
- Not practical for large n
- Advantage: Fault-model independent

Fault-Oriented Test Generation Algorithm:



- 1) Set x to 1: activate fault
- 2) Justify D on x, propagate D

to z

Set C and D to 1

Set y to 0

Set either A or B to 0

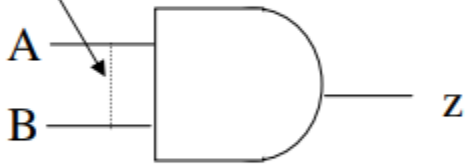
Example test pattern: ABCD = 0011

- Backtracking may be necessary
- Test generation is NP-complete

Bridging Faults

- Models short circuits, pairs of nodes considered
- Number of bridging faults?
- Feedback vs non-feedback bridging faults

bridge

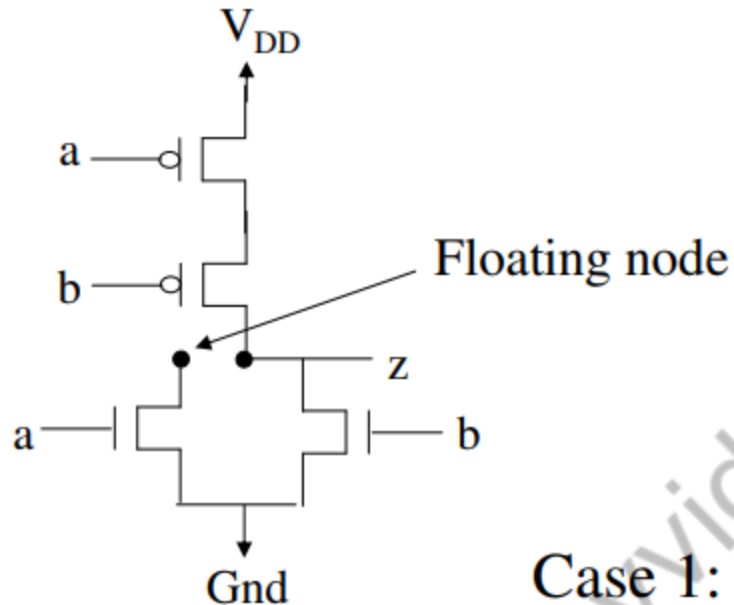


A	B	z	z^f	Wired-AND	Wired-OR
0	0	0	0	0	0
0	1	0	?	0	1
1	0	1	?	0	1
1	1	1	1	1	1

$z^f = ?$

What are the test patterns in this example?

Stuck-Open Faults



Fault-free circuit: $z = \overline{a+b}$

Faulty circuit: $z^f = \overline{a+b} + \overline{ab}\tilde{z}$

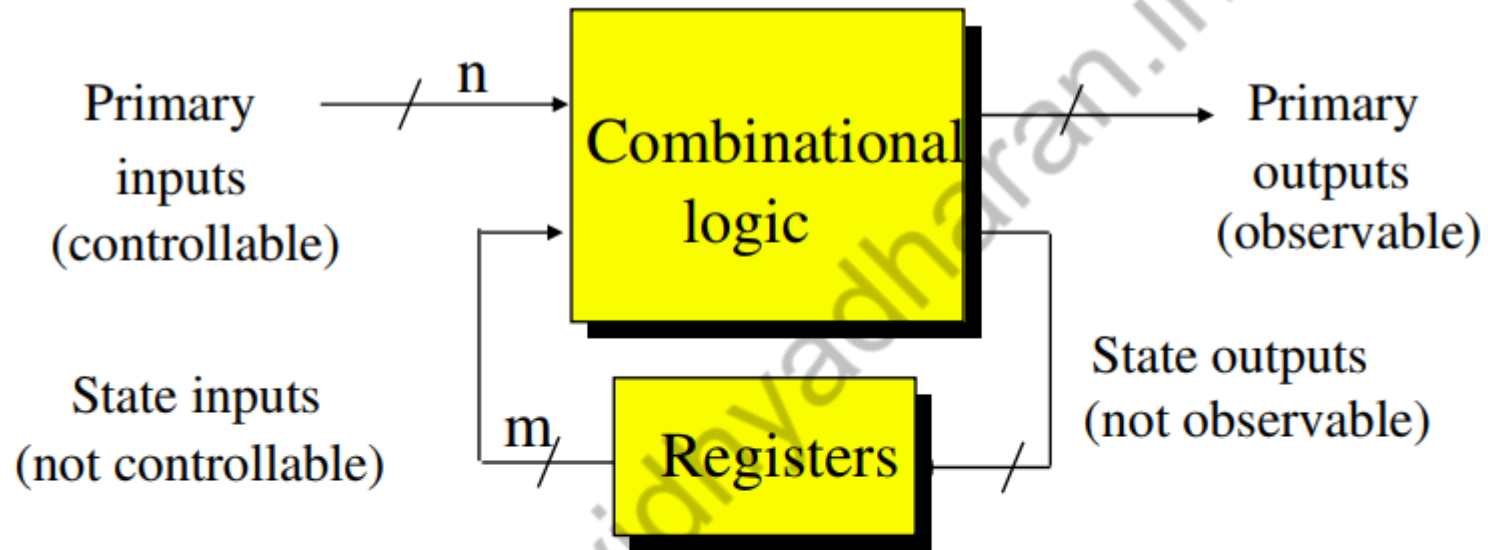
\tilde{z} : Previous value of z

Case 1: $a = b = 1$, z pulled down to 0

Case 2: $a = 1$, $b = 0$, z retains previous state

A test for a stuck-open fault requires two patterns
 $\{ab = 00, ab = 10\}$

Sequential Circuit Test Generation



- Difficult problem!
- Exhaustive testing requires 2^{m+n} patterns (2^m states and 2^n transitions from each state)
- Every fault requires a sequence of patterns

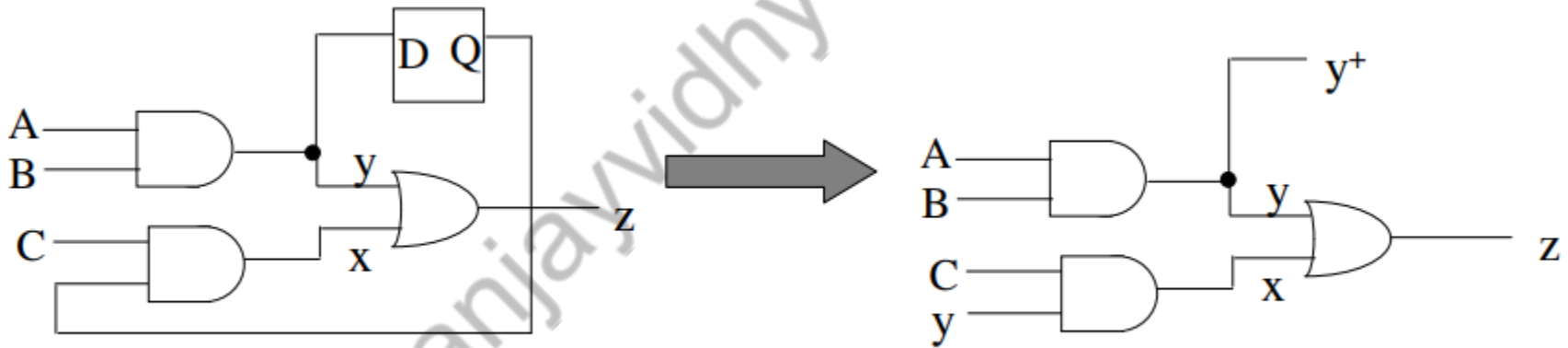
Initializing sequence: drive to known state

Test activation

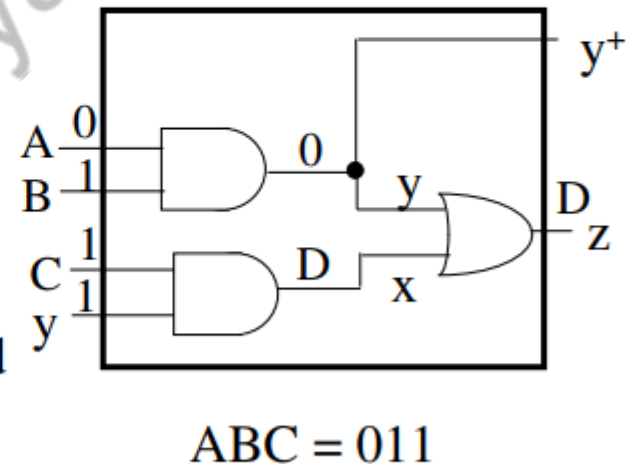
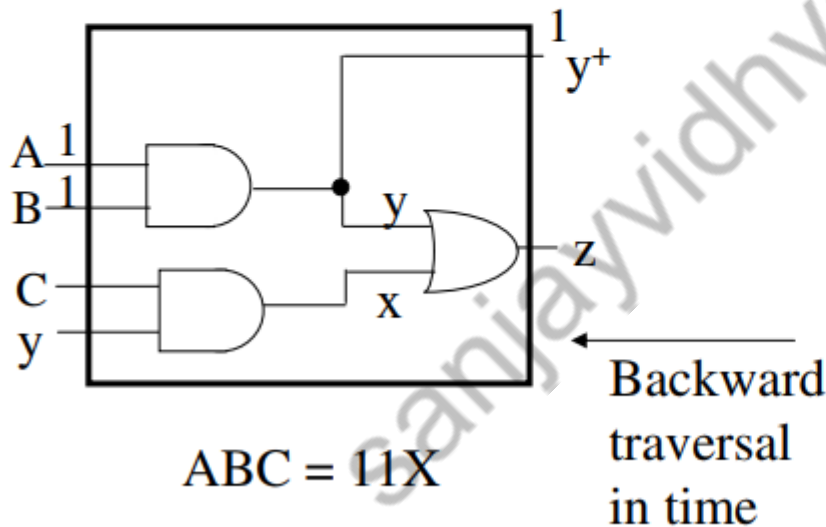
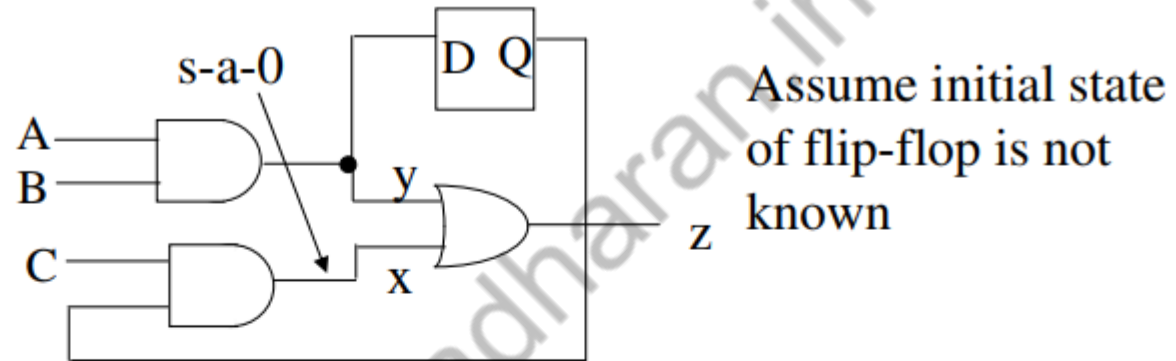
Propagation sequence: propagate discrepancy to observable output

Sequential Circuit Test Generation

- Iterative-array model (pseudo-combinational circuit)



Sequential Circuit Test Generation



Test pattern sequence: { 11X, 011 }

Current time frame

Design for Testability

➤ Ad Hoc Design for Testability Techniques

- Method of test points
- Multiplexing and demultiplexing of test points
- Time sharing of I/O for normal working and testing modes
- Partitioning of registers and large combinational circuits

➤ Scan-Path Design

- Scan-path design concept
- Controllability and observability by means of scan-path
- Full and partial serial scan-paths
- Non-serial scan design
- Classical scan designs

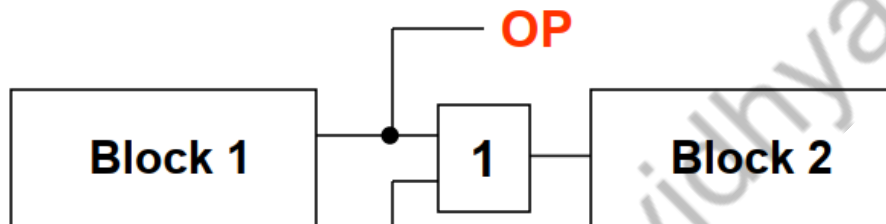
Ad Hoc Design for Testability

Method of Test Points:



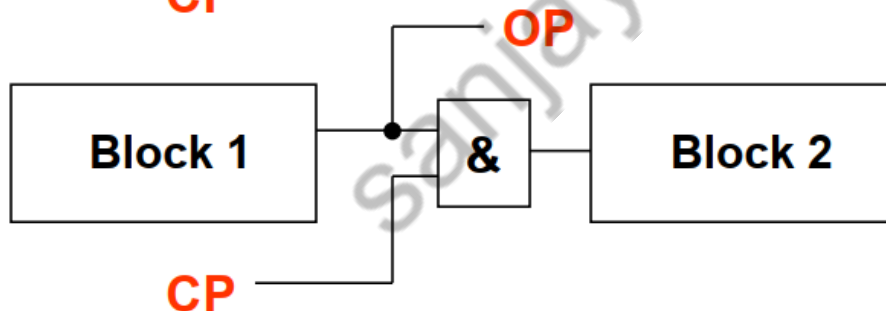
Block 1 is not observable,
Block 2 is not controllable

Improving controllability and observability:



1- controllability:

CP = 0 - normal working mode
CP = 1 - controlling Block 2
with signal 1

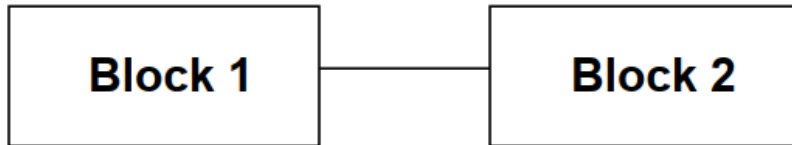


0- controllability:

CP = 1 - normal working mode
CP = 0 - controlling Block 2
with signal 0

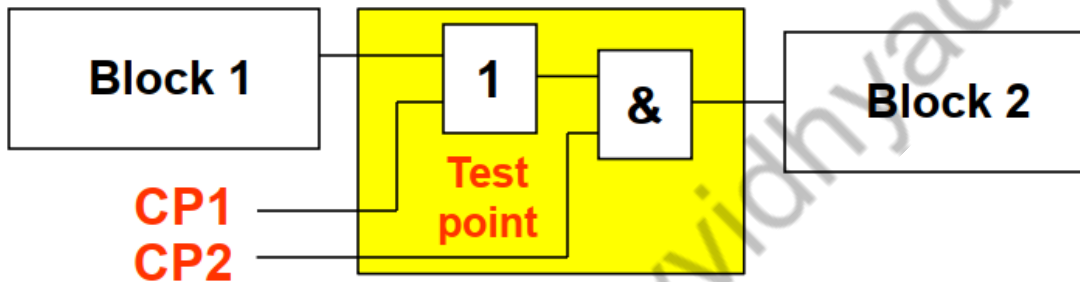
Ad Hoc Design for Testability

Method of Test Points:



Block 1 is not observable,
Block 2 is not controllable

Improving controllability:



Normal working mode:

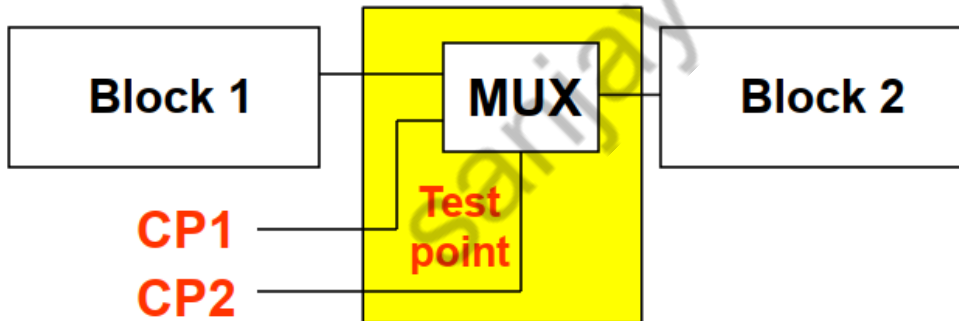
CP1 = 0, CP2 = 1

Controlling Block 2 with 1:

CP1 = 1, CP2 = 1

Controlling Block 2 with 0:

CP2 = 0



Normal working mode:

CP2 = 0

Controlling Block 2 with 1:

CP1 = 1, CP2 = 1

Controlling Block 2 with 0:

CP1 = 0, CP2 = 1

Ad Hoc Design for Testability

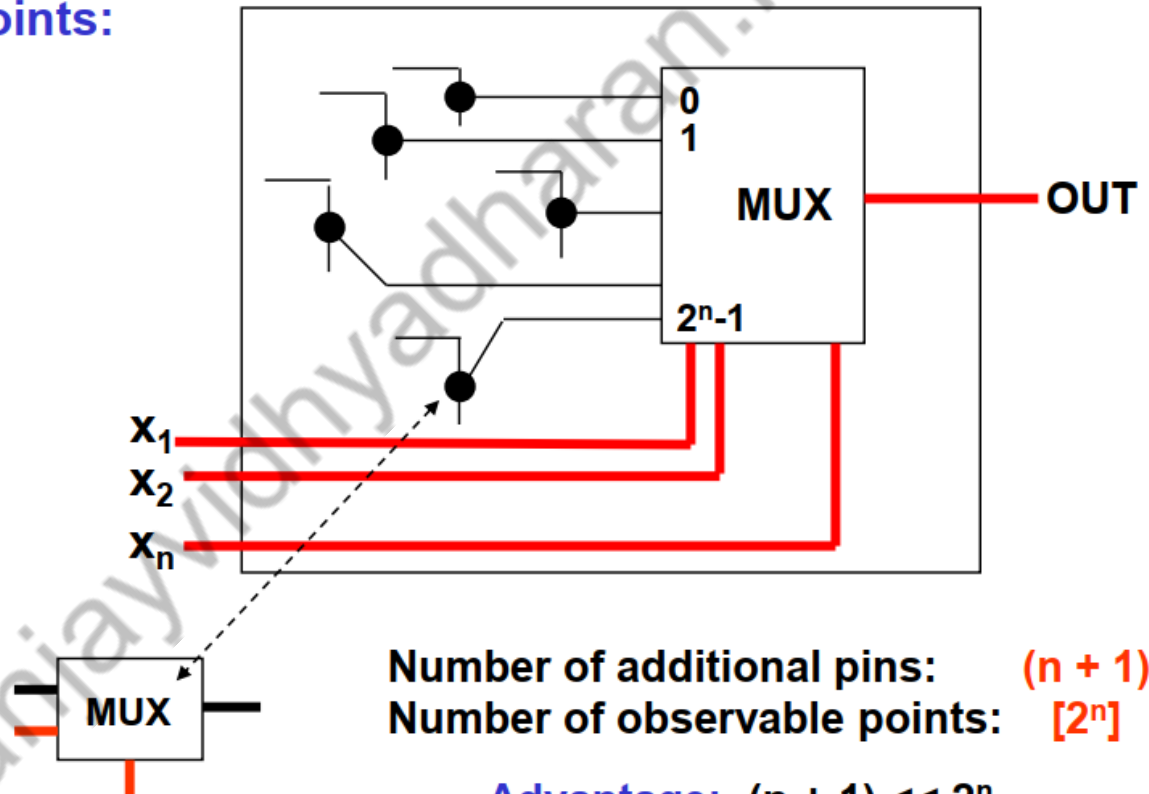
Multiplexing monitor points:

To reduce the number of output pins for observing monitor points, multiplexer can be used:

2^n observation points are replaced by a **single output** and **n inputs** to address a selected observation point

Disadvantage:

Only one observation point can be observed at a time



Number of additional pins: $(n + 1)$

Number of observable points: $[2^n]$

Advantage: $(n + 1) \ll 2^n$

Ad Hoc Design for Testability

Multiplexing monitor points:

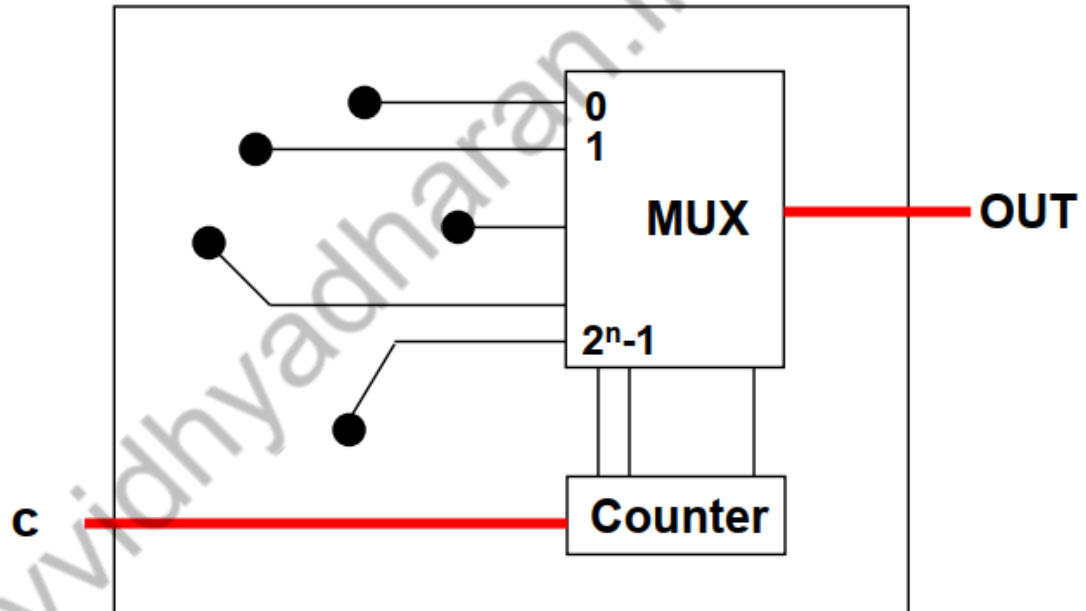
To reduce the number of output pins for observing monitor points, multiplexer can be used:

To reduce the number of inputs, a **counter** (or a shift register) can be used to drive the address lines of the multiplexer

Disadvantage:

Only one observation point can be observed at a time

Reset for counter?

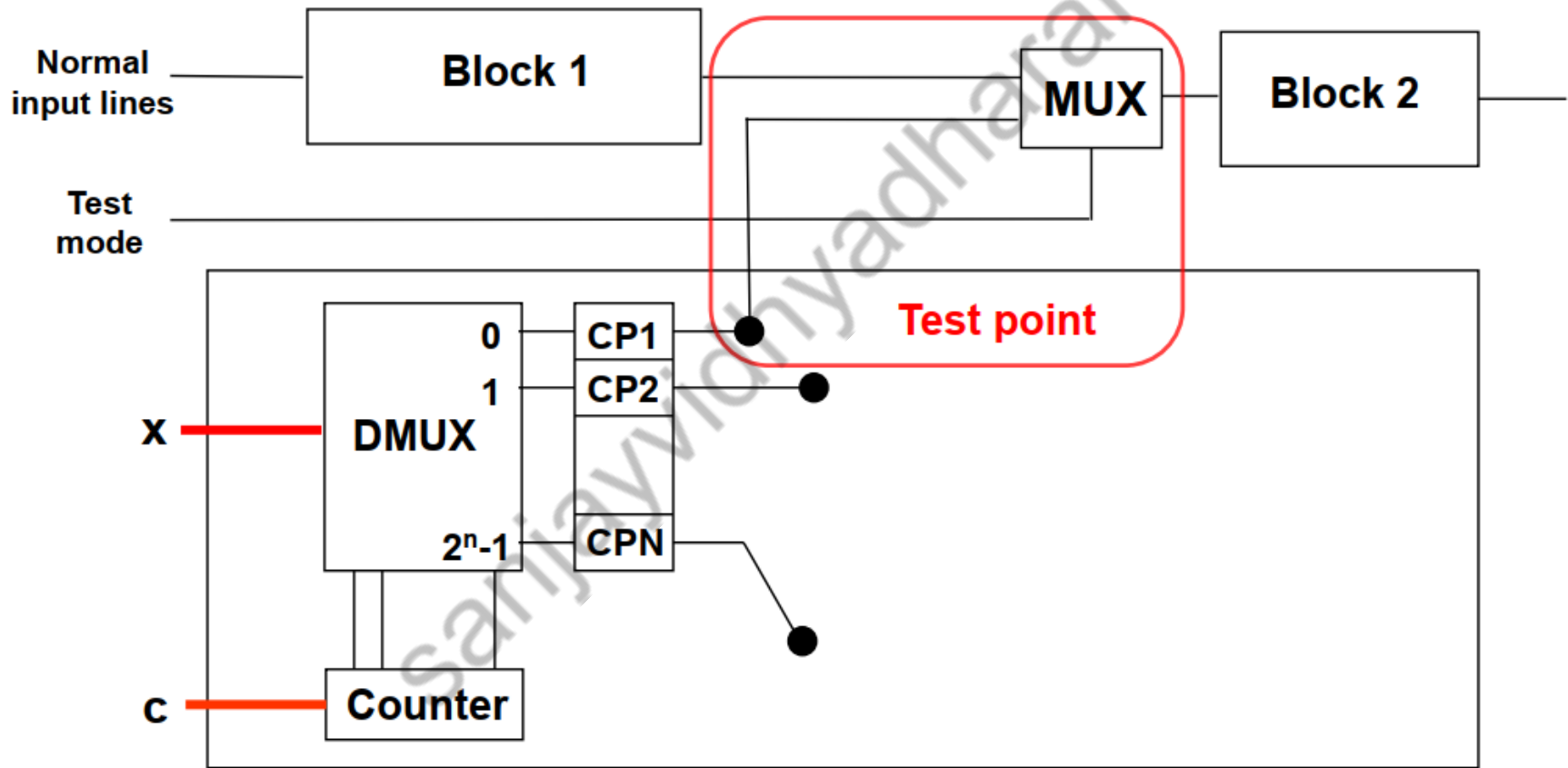


Number of additional pins: **2**
Number of observable points: **$[2^n]$**

Advantage: $2 < n \ll 2^n$

Ad Hoc Design for Testability

Demultiplexer for implementing control points:



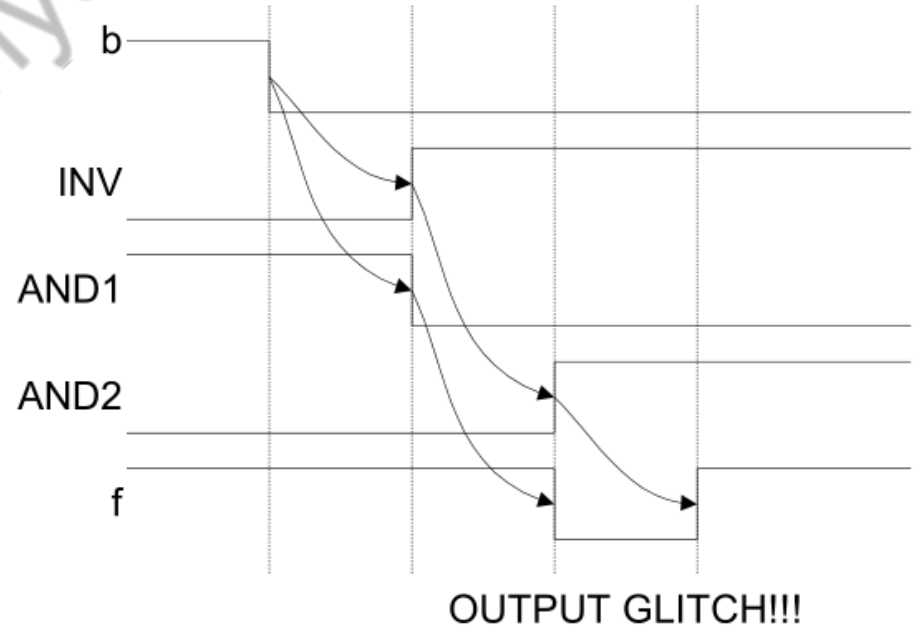
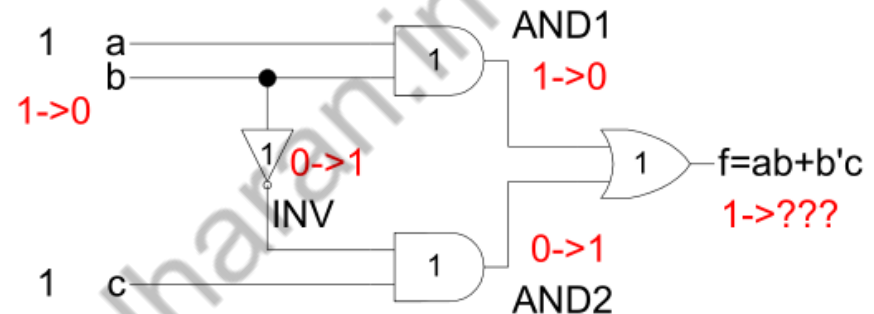
Static Glitch Example

Consider the following circuit with delays where only one input (input b) changes...

Draw a timing diagram to see what happens at output with delays.

From the logic expression, we see that b changing should result in the output remaining at logic level 1...

Due to delay, the output goes 1->0->1 and this is an output glitch; **we see a static-1 hazard.**



Static Glitch Elimination

When circuits are implemented as **2-level SOP (2-level POS)**, we can detect and remove hazards by inspecting the K-Map and **adding redundant product (sum) terms**.

		bc			
		00	01	11	10
a	0	0	1	0	0
	1	0	1	1	1

$f = ab + b'c$

Observe that when input b changes from 1→0 (as in the previous timing diagram), that we “jump” from one product term to another product term.

- ***If adjacent minterms are not covered by the same product term, then a HAZARD EXISTS!!!***

Static Glitch Elimination

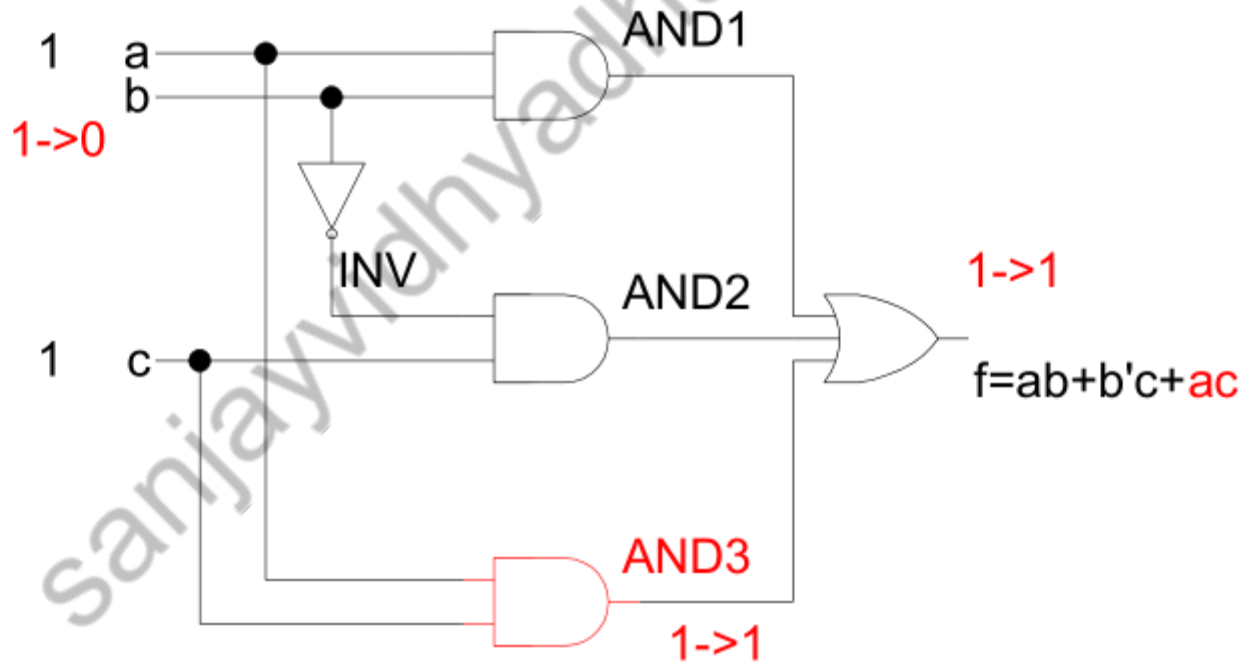
		bc			
		00	01	11	10
a	0	0	1	0	0
	1	0	1	1	1

$$f = ab + b'c + ac$$

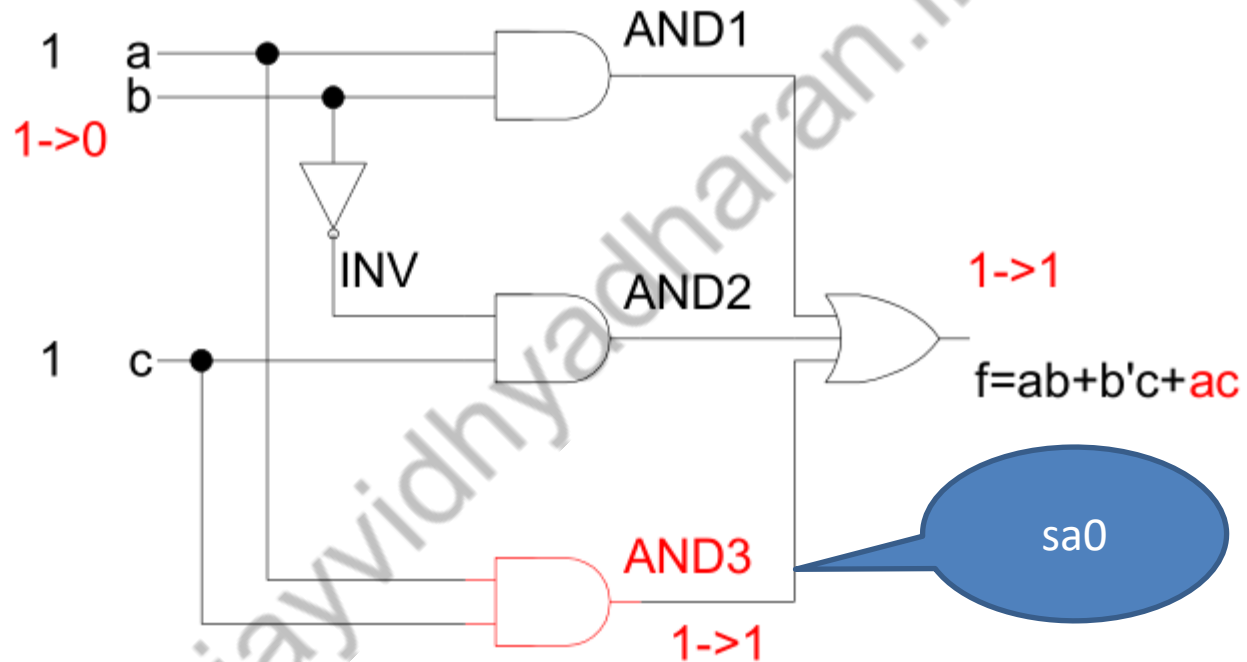
The extra product term does not include the changing input variable, and therefore serves to prevent possible momentary output glitches due to this variable.

Static Glitch Elimination

The redundant product term is not influenced by the changing input.



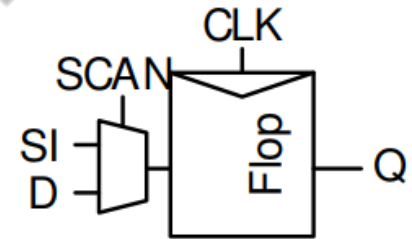
Ad Hoc Design for Testability



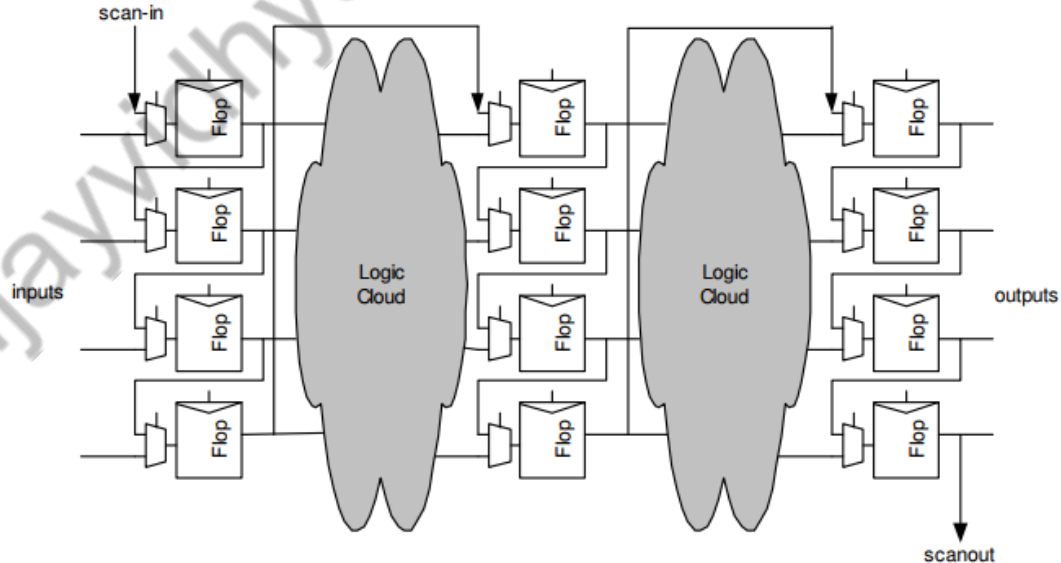
Much test generation time can be spent in trying to generate a test for a redundant fault

Scan Design

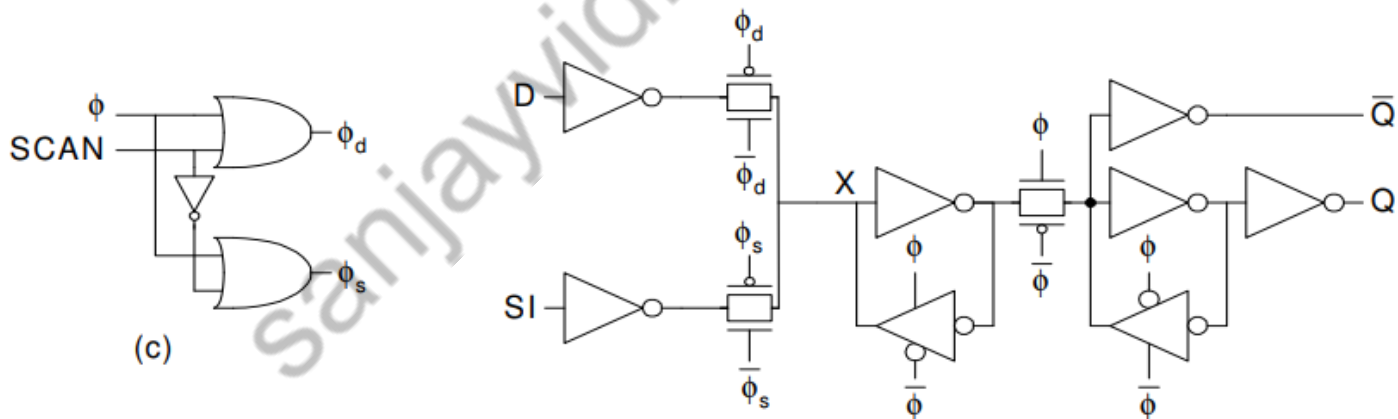
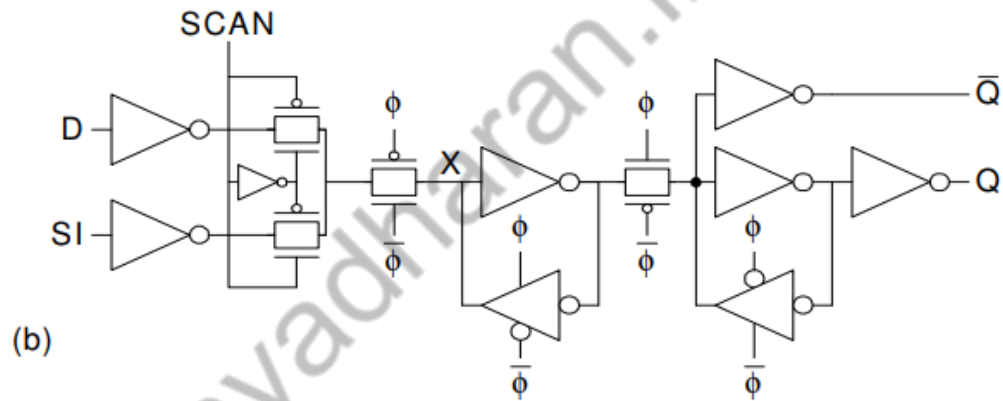
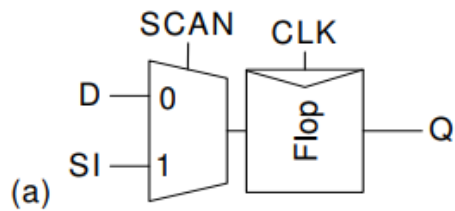
- Convert each flip-flop to a scan register
 - Only costs one extra multiplexer
- Normal mode: flip-flops behave as usual
- Scan mode: flip-flops behave as shift register



- Contents of flops can be scanned out and new values scanned in



Scannable Flip-flops

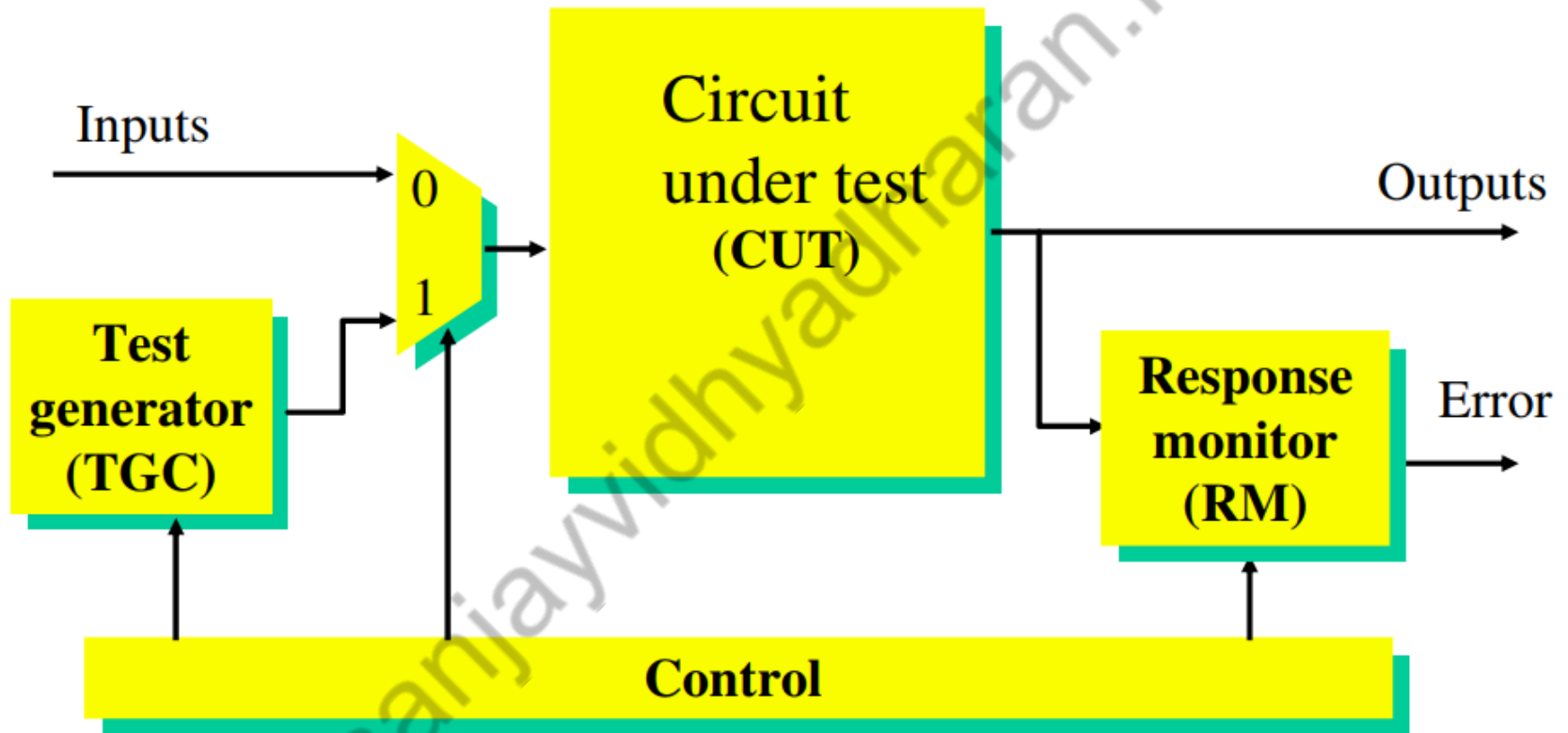


Built-in Self-test (BIST)

Built-in self-test lets blocks test themselves

- Generate pseudo-random inputs to comb. logic
- Combine outputs into a *syndrome*
- With high probability, block is fault-free if it produces the expected syndrome

Built-in Self-test (BIST)



On-chip test generator and response monitor

Built-in Self-test (BIST)

Advantages

- Lower cost due to elimination of external tester
- In-system, at-system, high-quality testing
- Faster fault detection, ease of diagnosis
- Overcomes pin limitations and related interfacing problems
- Reduces maintenance and repair costs at system level

Issues

Test strategy (random, exhaustive, deterministic)

Circuit partitioning

Test pattern generation

Exhaustive: counters

Random: Linear-feedback shift registers (LFSRs)

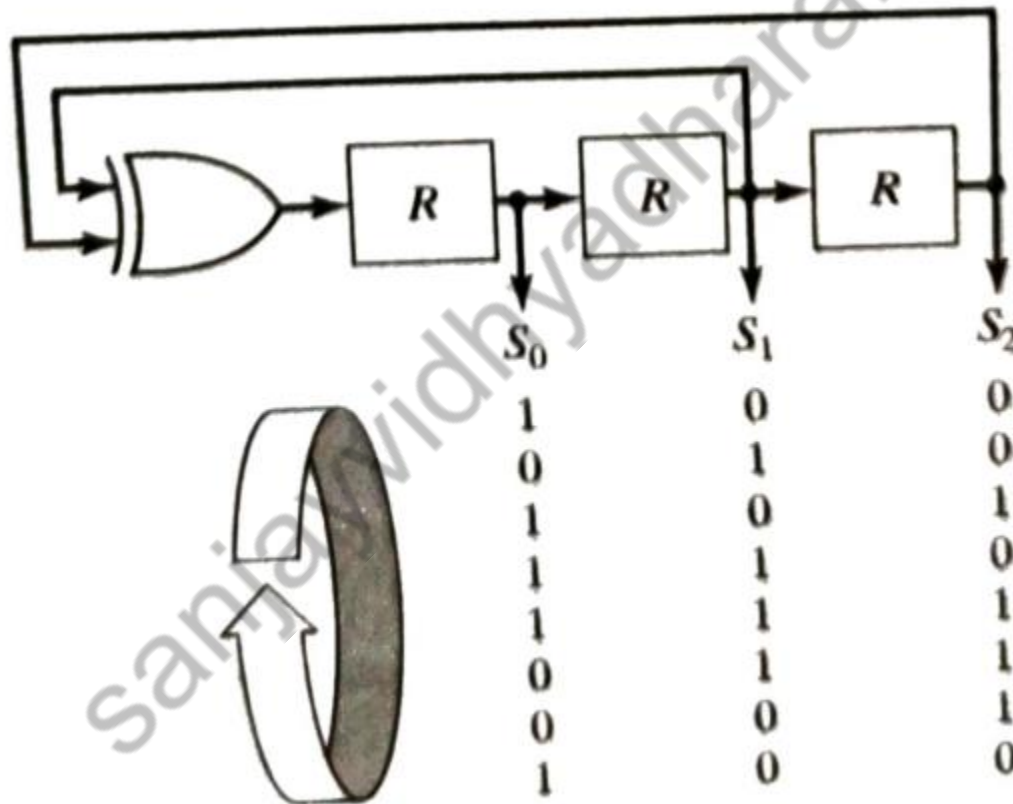
Deterministic: ROM, other methods?

Response analysis

Test control and scheduling

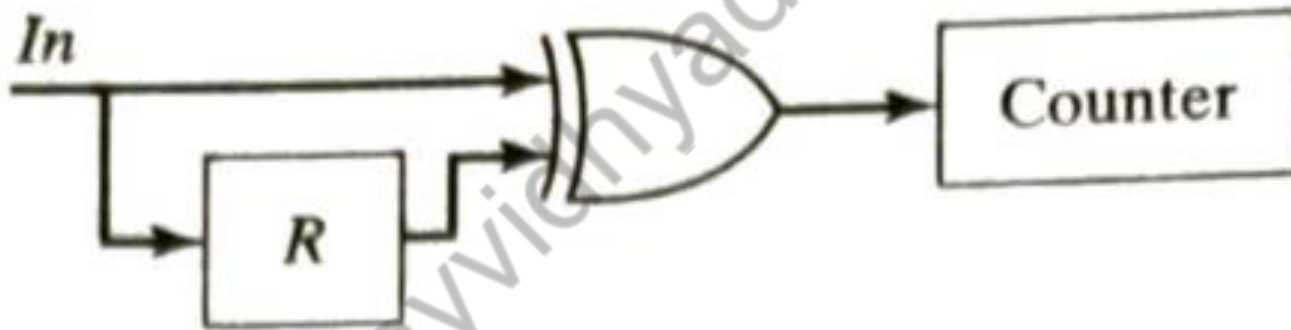
BIST Logic Circuits

Linear-feedback shift-register (LFSR)



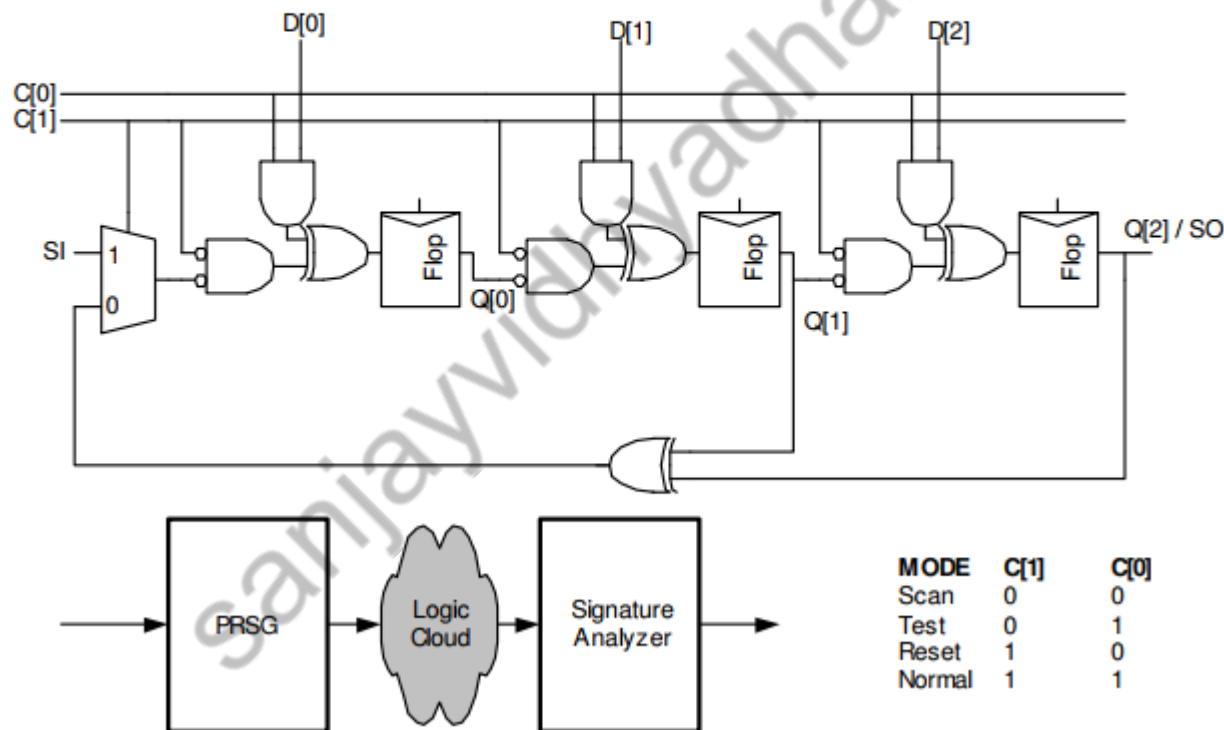
BIST Logic Circuits

Single Bit signature register (MISR)



BILBO

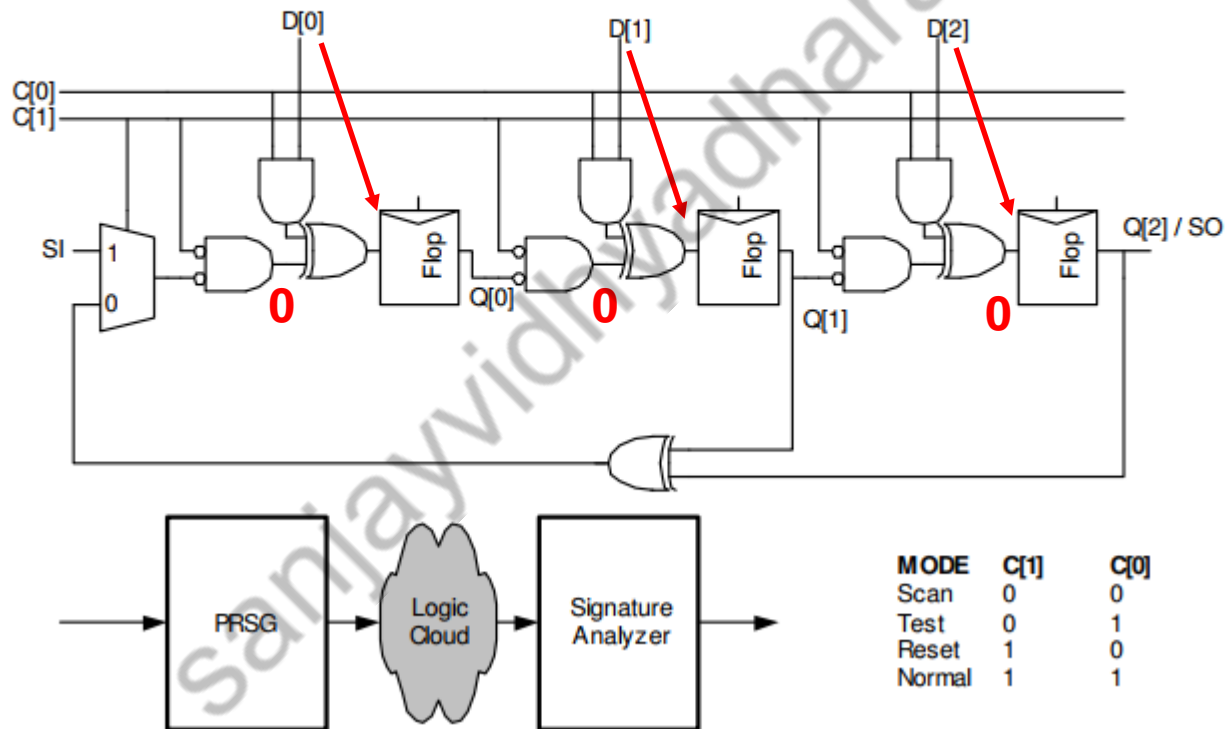
- Built-in Logic Block Observer
 - Combine scan with PRSG & signature analysis



BILBO

- Built-in Logic Block Observer
 - Combine scan with PRSG & signature analysis

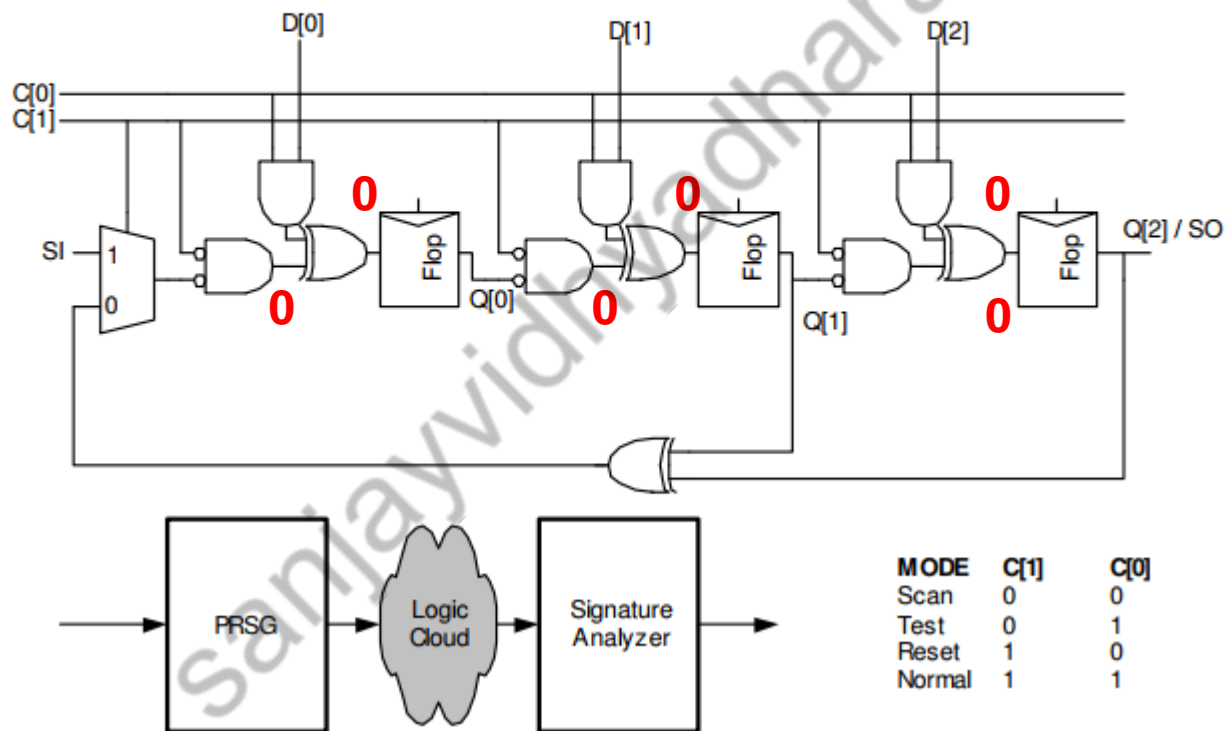
11



BILBO

- Built-in Logic Block Observer
 - Combine scan with PRSG & signature analysis

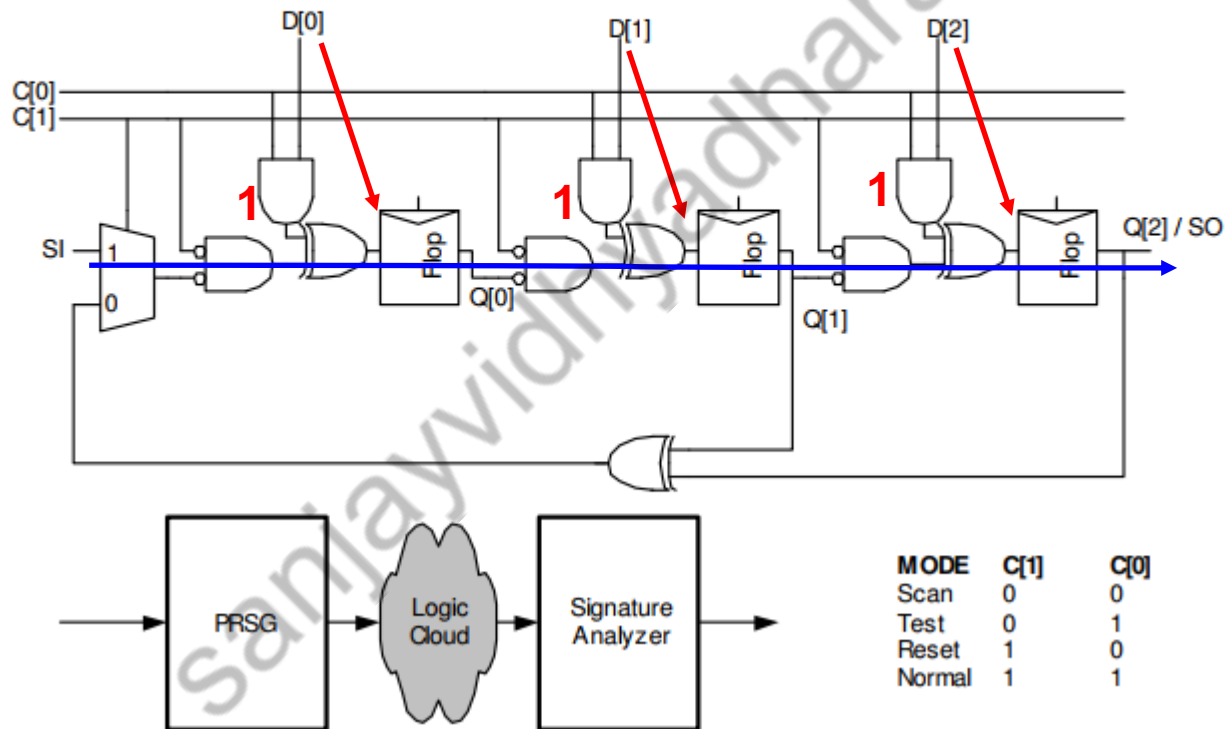
10



BILBO

- Built-in Logic Block Observer
 - Combine scan with PRSG & signature analysis

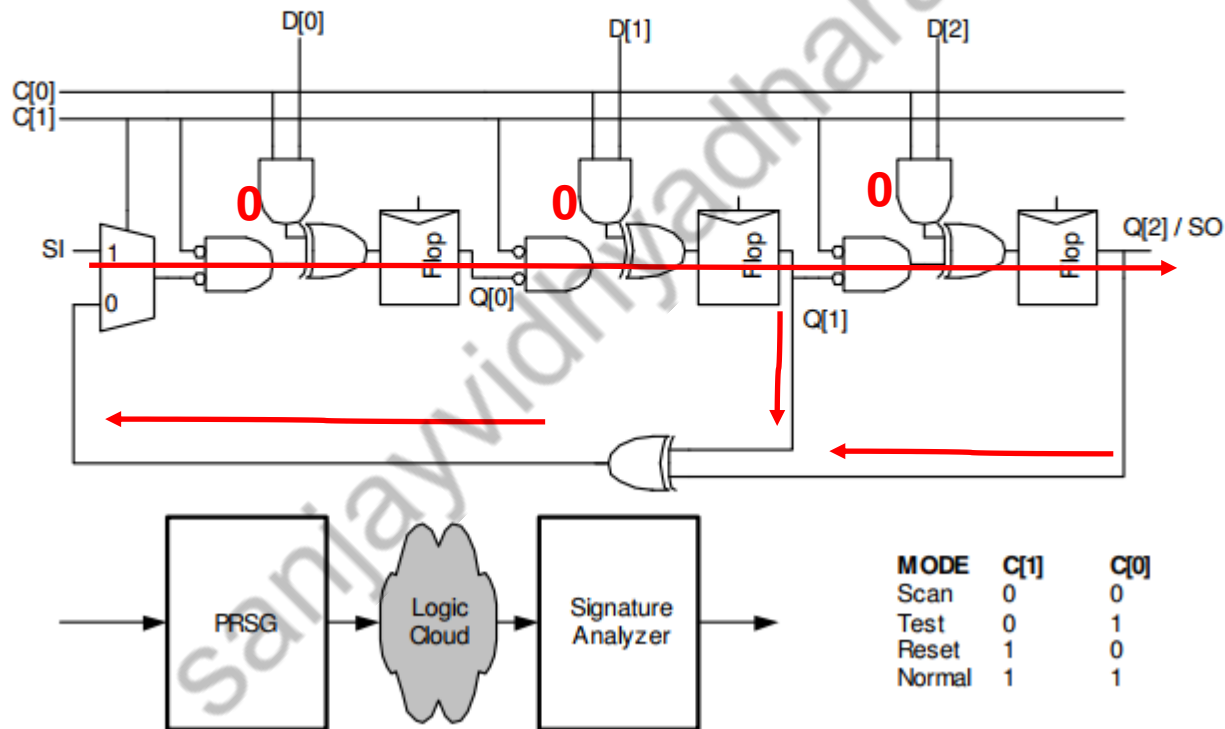
01



BILBO

- Built-in Logic Block Observer
 - Combine scan with PRSG & signature analysis

00



Thank you