Week 5: Course Material

Lecture 22

CONTEST [1989]

- Two-stage process
- 1st stage: aim to detect as many faults as possible
  - Fitness = a x #detected + b x #excited
- 2nd stage: aim to detect remaining hard faults individually
  - Fitness depends on if the target fault has been excited, and how many fault effects are in the circuit

GATEST [1994]

- GA-based ATPG for seq ckts
- Tournament selection, uniform crossover
- 1st phase: initialize the seq ckts
- 2nd phase: detect & excite as many faults as possible
- 3rd phase: similar to phase 2, but to monitor fault-free and faulty ckt events
- 4th phase: individuals now become sequence of vectors, aim to detect & excite as many faults as possible

Seeding the Initial Population

- Place non-random individuals in the initial population
- This can reduce the number of generations needed for the GA to obtain a good solution
- Aggressively used in STRATEGATE [1997]
  - Target individual faults rather than groups of faults
  - Seeding of propagation sequences
  - Seeding of justification sequences

Delay Testing

- Delay defects: class of defects that affects the functionality only when the circuit is running at a high speed
- Stuck-at fault model insufficient to model all delay-related defects
- Delay fault models
  - Path delay fault
  - Transition fault
  - Segment delay fault

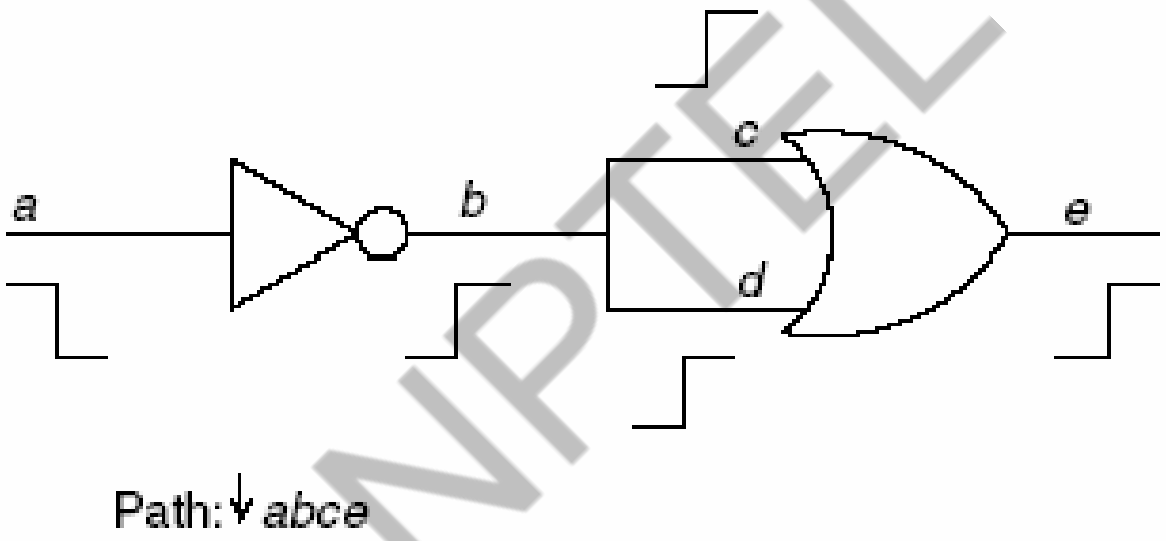
Path Delay Fault

- Models a combinational path in the circuit
  - Considers the cumulative effect of the delay along the path
  - On-inputs of a path
  - Off-inputs of a path
- A transition is launched at the start of the path, and a test must propagate the transition to the end of the path
  - Two faults associated with every path: rising and falling transition at the start of the path
- Number of paths can be exponential to the number of gates in the circuit
- Two vectors needed
  - V1: initialization vector
  - V2: launch and capture vector

Classification of Path Delay Faults

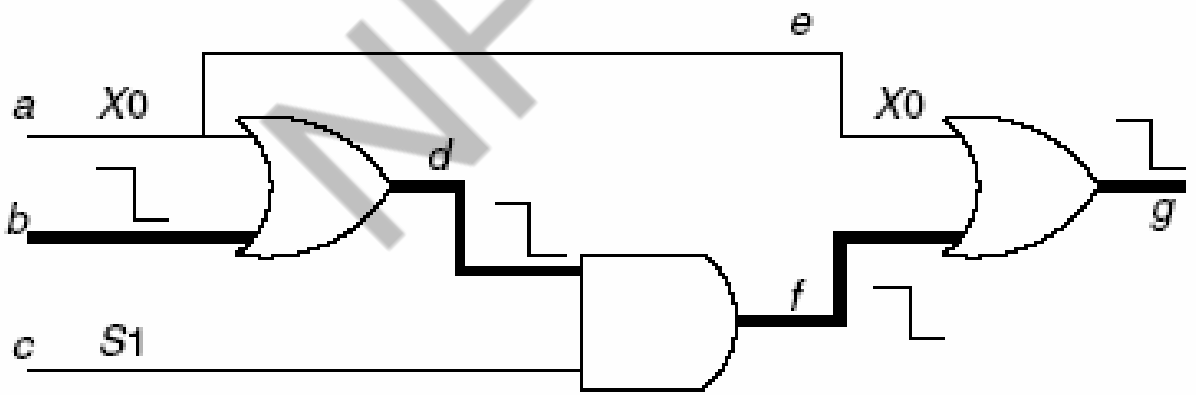
- Statically sensitizable: all off-inputs of a path P can be assigned to non-controlling values by some vector
- Single-path sensitizable: all off-inputs of a path can be set to non-controlling values for both vectors of a test
- False path: a transition cannot propagate from the start to the end of path
  - Not all necessary off-input values can be set to non-controlling values simultaneously

Statically Unsensitzable Path



Robustly Testable Paths

- Single-path sensitization is too stringent
- May not need to set all off-inputs to non-controlling values in V1 in order to propagate a transition
  - Highlighted path is robustly testable

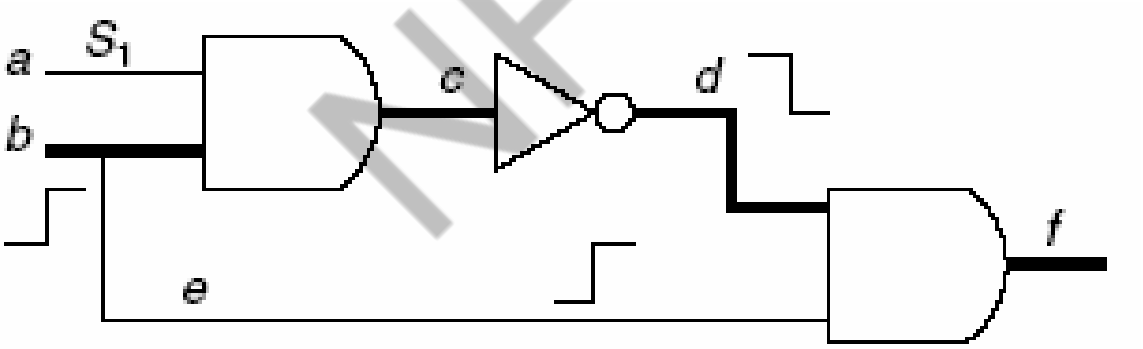


Robustly Testable (Cont.)

- D If a path is robustly testable, then the corresponding test can verify the correctness of the path irrespective of other delays in the circuit
- D Value criteria for robust testable path:
  - When the corresponding on-input of P has a controlling to non-controlling transition, the value in the first vector for the off-input can be X with the value for the off-input as a non-controlling value in the second vector.
  - When the corresponding on-input of P has a non- controlling to controlling transition, the values for
  - the off-input must be a steady non-controlling value for both vectors.

Non-robustly Testable Path

- Not all paths are robustly testable
- Further relax requirements for V1
- Test is valid if circuit has no other delay faults
  - Highlighted path is non-robustly testable



Non-robust Path (cont.)

- Non-robust test only valid if no other delay fault is present in the circuit
- Value criteria for non-robust testing:
  - Irrespective of the transition on the on-input, the value in the first vector for the off-input can be X, with the value for the off-input as a non-controlling value in the second vector.

ATPG for Path-Delay Faults

- Can use new value algebra to consider both vectors simultaneously during ATPG
  - *S*0—Initial and final values are both logic 0.
  - *S*1—Initial and final values are both logic 1.
  - *U*0—Initial logic can be either 0 or 1, but final value is logic 0.
  - *U*1—Initial logic can be either 0 or 1, but final value is logic 1.
  - *XX*—Both initial and final values are "don't cares."

Boolean Operations

AND	S0	U0	S1	U1	XX
S0	S0	S0	S0	S0	S0
U0	S0	U0	U0	U0	U0
S1	S0	U0	S1	U1	XX
U1	S0	U0	U1	U1	XX
XX	S0	U0	XX	XX	XX

NOT	
S0	S1
U0	U1
S1	S0
U1	U0
XX	XX

OR	S0	U0	S1	U1	XX
S0	S0	U0	S1	U1	XX
U0	U0	U0	S1	U1	XX
S1	S1	S1	S1	S1	S1
U1	U1	U1	S1	U1	U1
XX	XX	XX	S1	U1	XX

RESIST [1994]

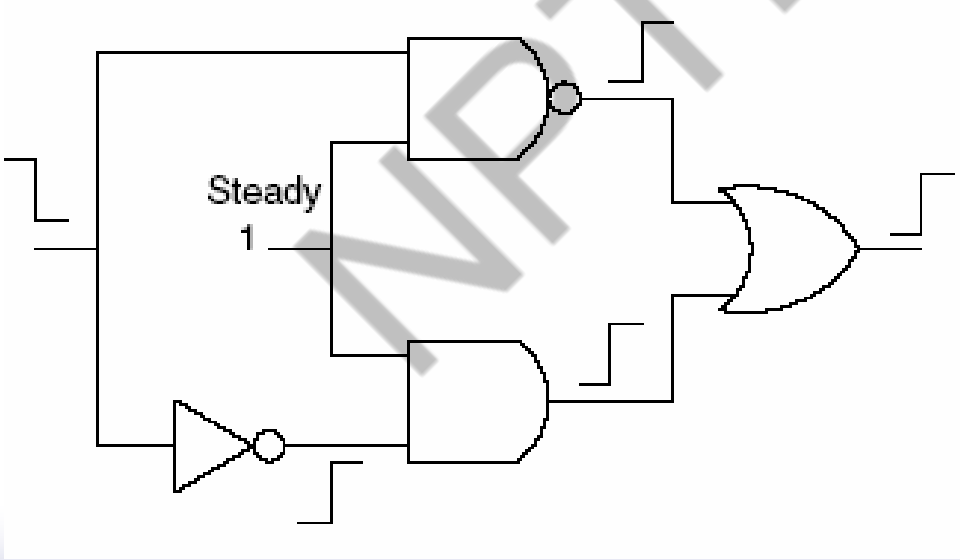
- Recursion-based path-delay-faultATPG
  - Starts at a PI
  - Depth-first-search through the circuit along each path
  - Generate a test for each path
  - Takes advantage of many paths that share common path-segments

Transition Fault Model

- Assumes a large/gross delay is present at a circuit node
- Irrespective of which path the effect is propagated, the gross delay defect will be late arriving at an observable point
- Most commonly used in industry
  - Simple and number of faults linear to circuit size
  - Also needs 2 vectors to test
- Node x slow-to-rise (x-STR) can be modeled simply as two stuck-at faults
  - First time-frame: x/1 needs to be excited
  - Second time-frame: x/0 needs to be excited and propagated

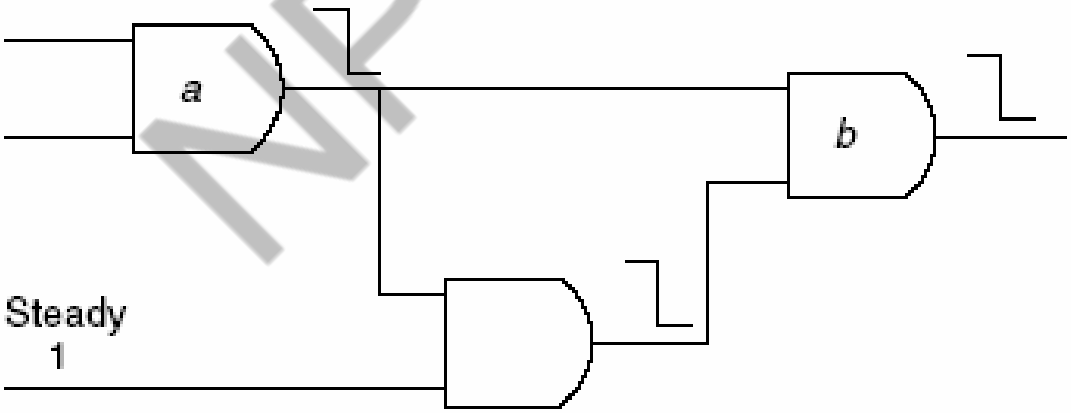
Transition Fault Properties

- Lemma: a transition fault may be launched robustly, non-robustly, or neither
- Example: STR at output of OR gate



Transition Fault Properties (cont.)

- Lemma: a transition fault may be propagated robustly, non-robustly, or neither
- Example: STF at output of gate ‘a’



Transition Fault Testing with Stuck-At *ATPG*

- Simply treat each transition fault as two stuck-at faults
- Can test it with broadside, skewed-load, or enhanced scan

Properties of Chaining Stuck-at *vectors*

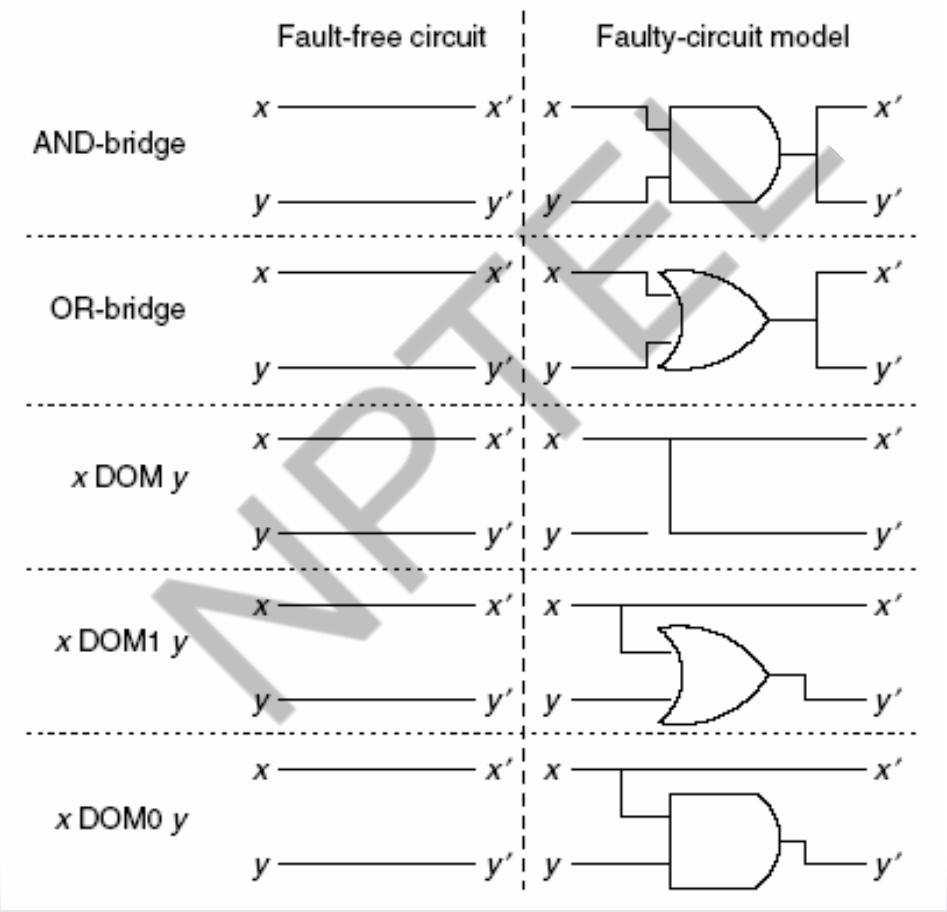
- Consider a sequence of 3 vectors: (vi, vj, vk) forming two vector-pairs (vi, vj) and (vj, vk)
- Theorem: Transition faults detected by (vi,vj) and pattern (vj,vk) are mutually exclusive.
- Why?

Lecture 23

Bridging Fault

- Models shorts between two circuit nodes
- The bridge fault is not excited unless the two circuit nodes have opposing logic values
- Faulty value depends on the bridge-fault type:
  - AND bridge: faulty value is the AND of the two involved nodes' values
  - OR bridge: faulty value is the OR of the two involved nodes' values
  - X Dom y: value of x dominates
  - X Dom1 y: x dominates y if x=1
  - X Dom0 y: x dominates y if x=0

Illustration of the Bridge



Bridging Fault ATPG

- Modeled as a constrained stuck-at ATPG
- Consider AND-bridge(x,y), we can do either:
  - Detect x/0 with setting y=0
  - Detect y/0 with setting x=0
- Conventional stuck-at ATPG can be modified to handle bridge faults

Combinational Test Set Compaction

- Want to reduce the test set size to reduce test data storage and test application time
- Idea: find a minimal set of vectors that can detect every fault
- First build a detection dictionary

Test Set Compaction

	f_1	f_2	f_3	f_4	f_5	f_6
v_1	X		X		X	
v_2					X	X
v_3	X			X		X
v_4		X	X	X	X	

- Essential vector: a vector that detects some faults that no other vector can detect
  - V4 is essential
- A set covering algorithm is applied to find a min test set such that every fault is covered

Test Set Compaction (cont.)

- If vectors are incompletely specified
  - Some vectors may be compatible: 1X0X and X100 are compatible. Just one vector 1100 is sufficient
- Reverse-order simulation
  - Simulate the test set in reverse order, some vectors may no longer be needed

N-Detect ATPG

- Idea: detect every fault at least N times
  - N vectors that detect a fault must be different
- Although the same fault coverage, can significantly enhance the defect coverage
  - If x/0 is detected 2 times, one with y=1, and the other with y=0, then the AND-bridge fault of (x,y) would have been detected by the second test
- ATPG can be modified to N-Detect ATPG

Concluding Remarks

- Covered a number of topics
  - Theoretical Foundations
  - Combinational & sequential ATPG
  - Untestable fault identification
  - Simulation-based & hybrid ATPG
  - Delay testing
  - Bridging fault testing
  - Compaction, N-Detect, FSM testing
- Challenges Ahead
  - Fast untestable fault identification essential to remove large numbers of stuck-at, bridge, delay faults
  - Sequential ATPG remains an open research area

Lecture 24

Logic Built-In Self-Test

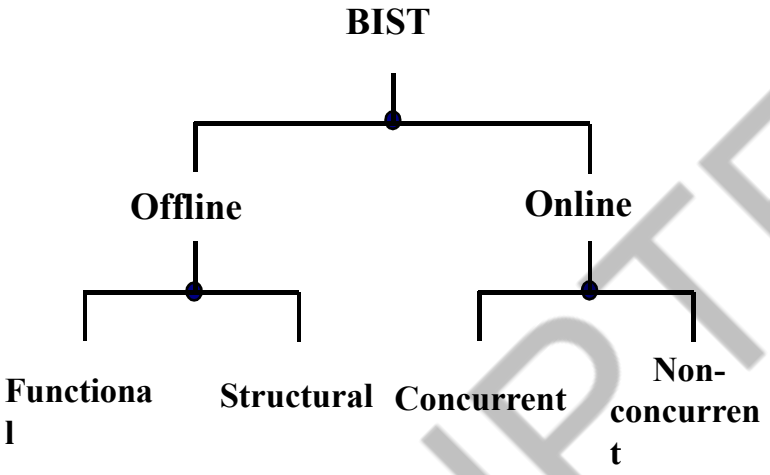
Introduction

- What are the problems in today's semiconductor testing?
  - Traditional test techniques become quite expensive
  - No longer provide sufficiently high fault coverage
- Why do we need built-in self-test (BIST)?
  - For mission-critical applications
  - Detect un-modeled faults
  - Provide remote diagnosis

BIST Techniques Categories

- Online BIST
  - Concurrent online BIST
  - Non Concurrent online BIST
- Offline BIST
  - Functional offline BIST
  - Structural offline BIST

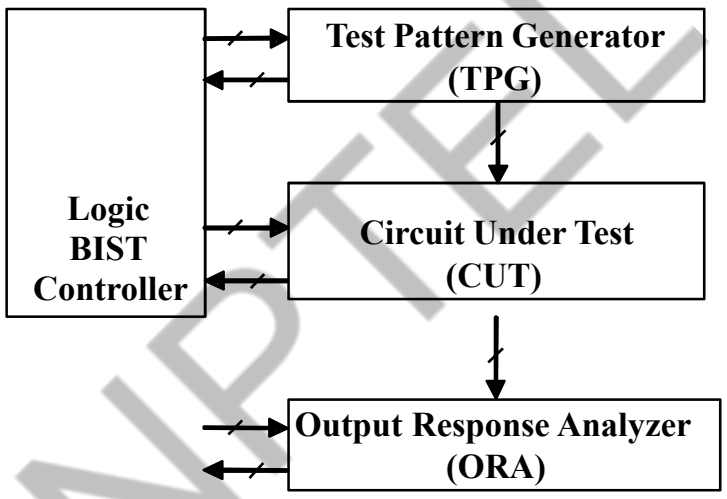
A General Form of Logic BIST



[Abramovici 1994]

Logic BIST Techniques

A Typical Logic BIST System

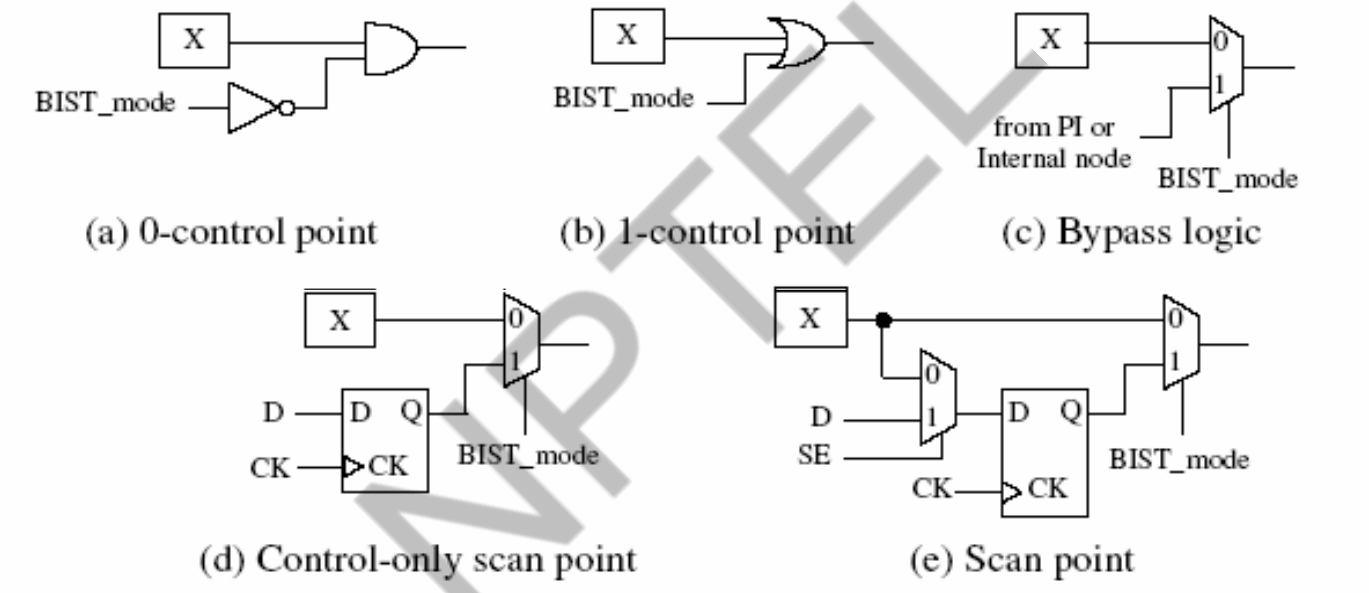


Structural off-line BIST

BIST Design Rules

Logic BIST requires much more stringent design restrictions when compared to conventional scan. Therefore, when designing a logic BIST system, it is essential that the circuit under test meet all *scan design rules* and *BIST specific design rules*, called BIST design rules.

Typical X-bounding Methods



Methods for blocking an unknown (X) source

X-bounding Methods

Depending on the nature of each unknown (X) source, several X-bounding methods can be appropriate for use.

Common problems:

(1) Increase the area of the design.

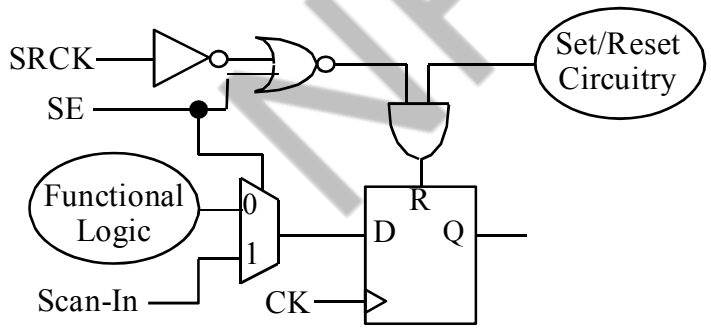
(2) Impact timing.

Typical Unknown Sources

- **Analog Blocks**
  - Adding bypass logic.
  - Adding control-only scan point
- **Memories and Non-Scan Storage Elements**
  - Bypass logic
  - Initialization
- **Combinational Feedback Loops**
  - Scan points

Typical Unknown Sources (cont'd)

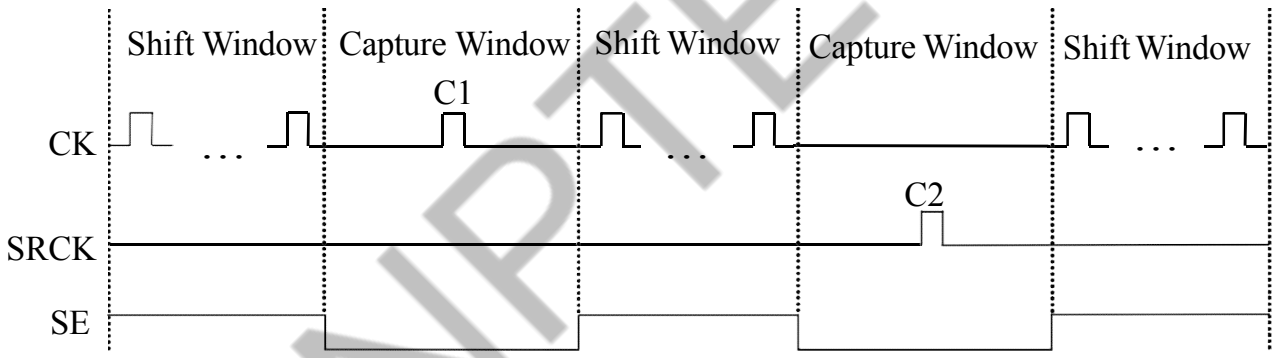
- **Asynchronous Set/Reset Signals**
  - using the existing scan enable (SE) signal to protect each shift operation and adding a **set/reset clock point** (SRCK) on each set/reset signal to test the set/reset circuitry.



[Abdel-Hafez 2004]

Typical Unknown Sources (cont'd)

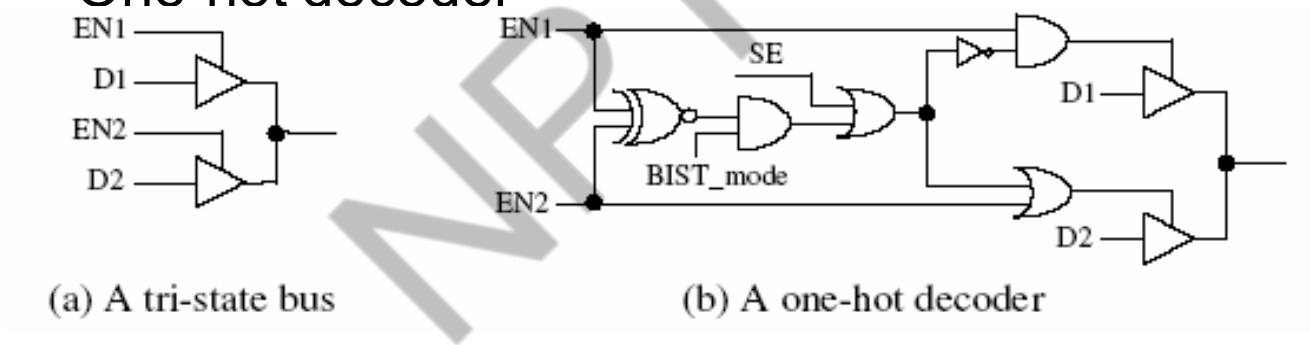
- **Asynchronous Set/Reset Signals**



Timing control diagram for testing data and set/reset faults

Typical Unknown Sources (cont'd)

- **Tri-State Buses**
  - Re-synthesize each bus with multiplexers.
  - One-hot decoder

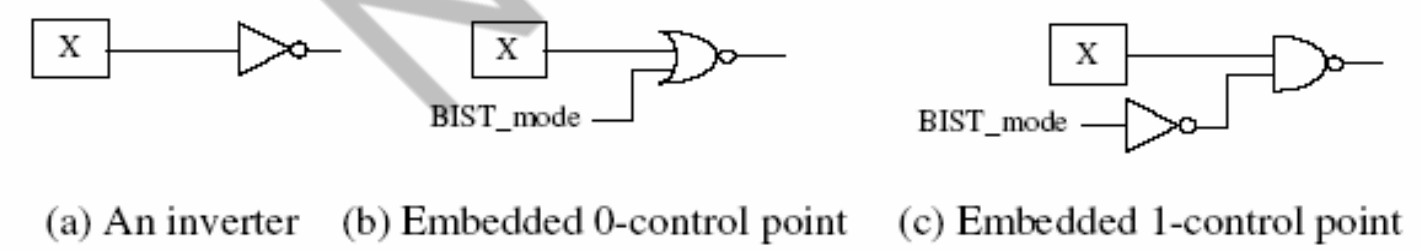


A one-hot decoder for testing a tri-state bus with 2 drivers

Lecture 25

Typical Unknown Sources (cont'd)

- **False Paths**
  - 0-control point
  - 1-control point
- **Critical Paths**
  - Adding an extra input pin to a selected combinational gate on the critical



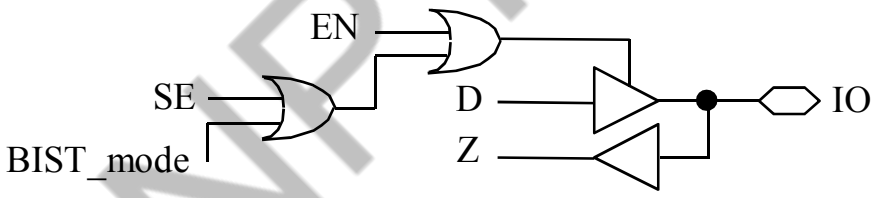
(a) An inverter (b) Embedded 0-control point (c) Embedded 1-control point

Typical Unknown Sources (cont'd)

- **Multiple-Cycle Paths**
  - 0-control point
  - 1-control point
  - Holding certain scan cell output states
- **Floating Ports**
  - PI or PO must have a proper connection to Power (Vcc) or Ground (Vss).
  - Floating inputs to any internal modules must be avoided.

Typical Unknown Sources (cont'd)

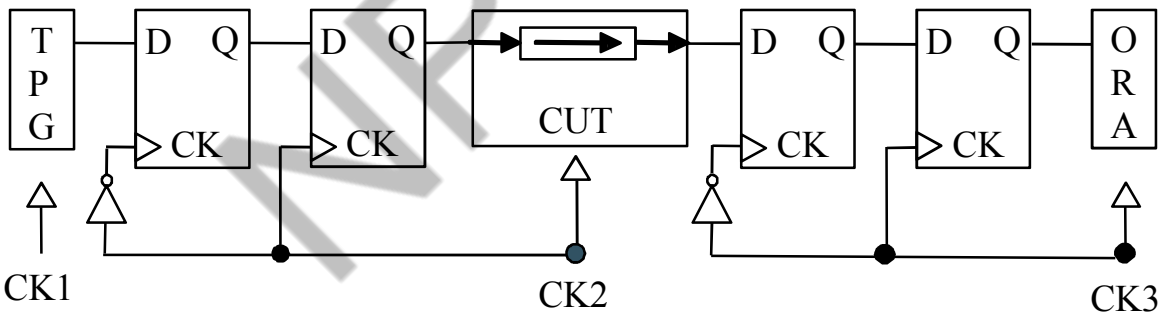
- **Bi-directional I/O Ports**
  - Fix the direction of each bi-directional I/O port to either input or output mode.



Forcing a bi-directional port to output mode

Re-Timing

Races and hazards caused by clock skews may occur between the TPG and the (scan chain) inputs of the CUT as well as between the (scan chain outputs of the CUT and the ORA. To avoid these potential problems and ease physical implementation, we recommend adding *re-timing* logic between the TPG and the CUT and between the CUT and the ORA.



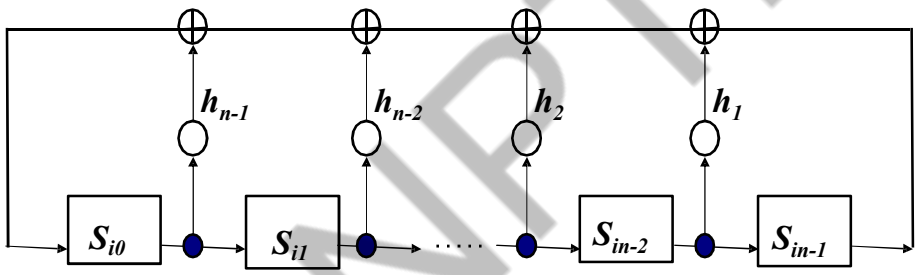
Re-timing logic among the TPG, CUT, and ORA

Test Pattern Generation

- Test pattern generators (TPGs) constructed from linear feedback shift registers (LFSRs)
- TPG
  - Exhaustive testing
  - Pseudo-random testing
  - Pseudo-exhaustive testing

Standard LFSR

- Consists of *n* D flip-flops and a selected number of exclusive-OR (XOR) gates

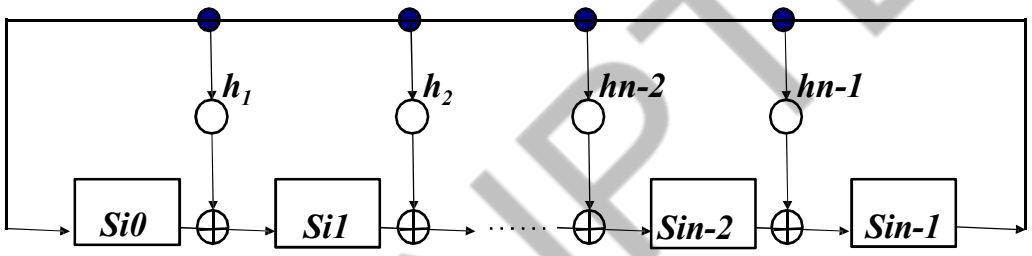


[Golomb 1982]

An n-stage (external-XOR) standard LFSR

Modular LFSR

- Each XOR gate placed between two adjacent D flip-flops



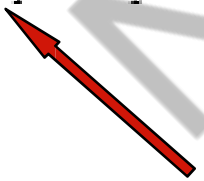
[Golomb 1982]

An n-stage (internal-XOR) modular LFSR

LFSR Properties

- The internal structure of the *n*-stage LFSR can be described by a characteristic polynomial of degree *n*, *f(x)*.

$$f(x) = 1 + h_1x + h_2x^2 + \ldots + h_{n-1}x^{n-1} + x^n.$$



h_i is either 1 or 0,depending on the feedback path

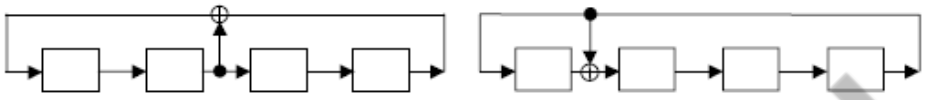
LFSR Properties

- Let S_i represent the contents of the n-stage LFSR after i^{th} shifts of the initial contents, S_0, of the LFSR, and $S_i(x)$ be the polynomial representation of S_i

$$S_i(x) = S_{i0} + S_{i1}x + S_{i2}x^2 + \ldots + S_{in-2}x^{n-2} + S_{in-1}x^{n-1}.$$

If T is the smallest positive integer such that f(x) divides $1 + x^T$ *, then the integer T is called the period of the LFSR.*

4-stage standard and modular



(a) A 4-stage standard LFSR

(b) A 4-stage modular LFSR

0 0 0 1
1 0 0 0
0 1 0 0
1 0 1 0
0 1 0 1
0 0 1 0
0 0 0 1
1 0 0 0
0 1 0 0
1 0 1 0
0 1 0 1
0 0 1 0
0 0 0 1
1 0 0 0
0 1 0 0
1 0 1 0

(c) Test sequence generated by (a)

0 0 0 1
1 1 0 0
0 1 1 0
0 0 1 1
1 1 0 1
1 0 1 0
0 1 0 1
1 1 1 0
0 1 1 1
1 1 1 1
1 0 1 1
1 0 0 1
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1

(d) Test sequence generated by (b)

- 4-stage Standard LFSR

$$f(x) = 1 + x^2 + x^4$$

- 4-stage Modular LFSR

$$f(x) = 1 + x + x^4$$

$$s_0 = x^3$$

Primitive polynomials list

Primitive polynomials of degree n up to 100

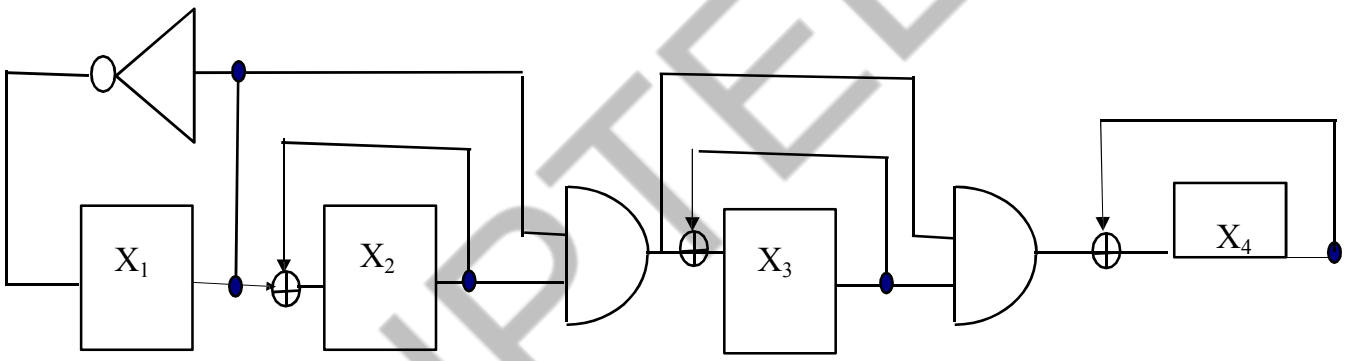
n	*Exponents*	*n*	*Exponents*	*n*	*Exponents*	*n*	*Exponents*
1	0	26	8 7 1 0	51	16 15 1 0	76	36 35 1 0
2	1 0	27	8 7 1 0	52	3 0	77	31 30 1 0
3	1 0	28	3 0	53	16 15 1 0	78	20 19 1 0
4	1 0	29	2 0	54	37 36 1 0	79	9 0
5	2 0	30	16 15 1 0	55	24 0	80	38 37 1 0
6	1 0	31	3 0	56	22 21 1 0	81	4 0
7	1 0	32	28 27 1 0	57	7 0	82	38 35 3 0
8	6 5 1 0	33	13 0	58	19 0	83	46 45 1 0
9	4 0	34	15 14 1 0	59	22 21 1 0	84	13 0
10	3 0	35	2 0	60	1 0	85	28 27 1 0
11	2 0	36	11 0	61	16 15 1 0	86	13 12 1 0
12	7 4 3 0	37	12 10 2 0	62	57 56 1 0	87	13 0
13	4 3 1 0	38	6 5 1 0	63	1 0	88	72 71 1 0
14	12 11 1 0	39	4 0	64	4 3 1 0	89	38 0
15	1 0	40	21 19 2 0	65	18 0	90	19 18 1 0
16	5 3 2 0	41	3 0	66	10 9 1 0	91	84 83 1 0
17	3 0	42	23 22 1 0	67	10 9 1 0	92	13 12 1 0
18	7 0	43	6 5 1 0	68	9 0	93	2 0
19	6 5 1 0	44	27 26 1 0	69	29 27 2 0	94	21 0
20	3 0	45	4 3 1 0	70	16 15 1 0	95	11 0
21	2 0	46	21 20 1 0	71	6 0	96	49 47 2 0
22	1 0	47	5 0	72	53 47 6 0	97	6 0
23	5 0	48	28 27 1 0	73	25 0	98	11 0
24	4 3 1 0	49	9 0	74	16 15 1 0	99	47 45 2 0
25	3 0	50	27 26 1 0	75	11 10 1 0	100	37 0

Note: "24 4 3 1 0" means $p(x) = x^{24} + x^4 + x^3 + x^1 + x^0$

Exhaustive Testing

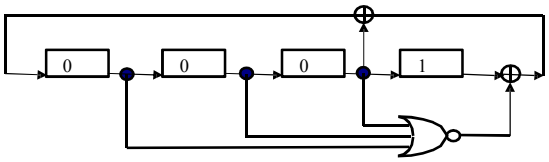
- Exhaustive Testing
  - Applying 2^n exhaustive patterns to an *n*-input combinational *circuit under test* (CUT)
- Exhaustive pattern generator
  - Binary counter
  - Complete LFSR => generates all zero also

Binary counter

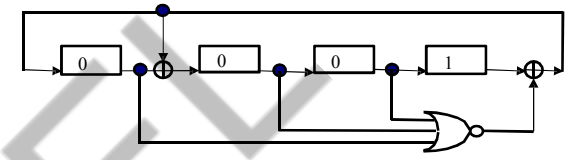


Example binary counter as EPG

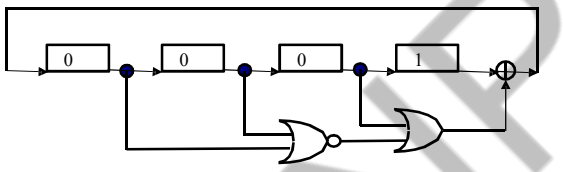
Complete LFSR



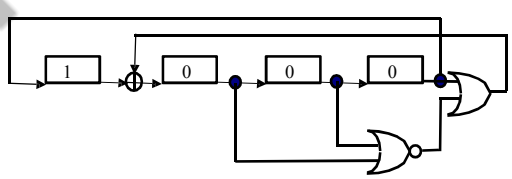
(a) 4-stage standard CFSR



(b) 4-stage modular CFSR



(c) A minimized version of (a)



(d) A minimized version of (b)

Example complete LFSRs as EPG

Lecture 26

Exhaustive Testing performance

- D Exhaustive Testing guarantees all detectable, combinational faults will be detected.
- D Test time maybe be prohibitively longif input number is large than 20.

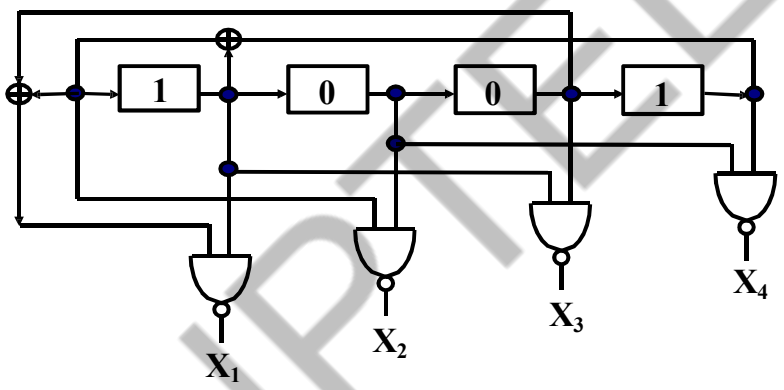
Pseudo-Random Testing

- Pseudo-random pattern generator
- Reduce test length but sacrifice the fault coverage
- Difficult to determine the required test length and fault coverage

Pseudo-Random Testing

- Maximum-length LFSR
  - RP-resistant problem
- Weighted LFSR
- Cellular Automata

Weighted LFSR



Example weighted LFSR as PRPG

Pseudo-Exhaustive Testing

- Reduce test time while retaining many advantages of exhaustive testing
- Guarantee 100% single-stuck fault coverage

  - Verification test technique
  - Segmentation test technique

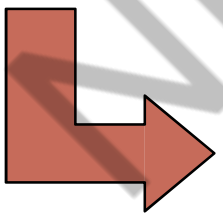
Verification Testing

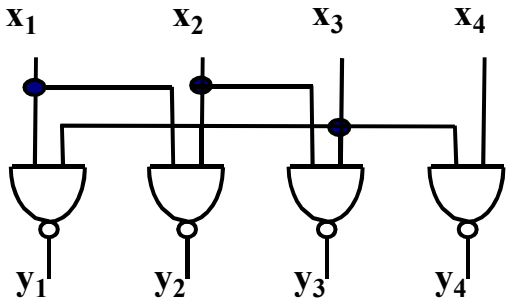
Divide the CUT into m cones, backtracing from each output to determine the inputs that drive the output. Each cone will receive exhaustive test patterns and are tested concurrently.

[McCluskey 1984]

Pseudo-exhaustive pattern generators

PEPGs



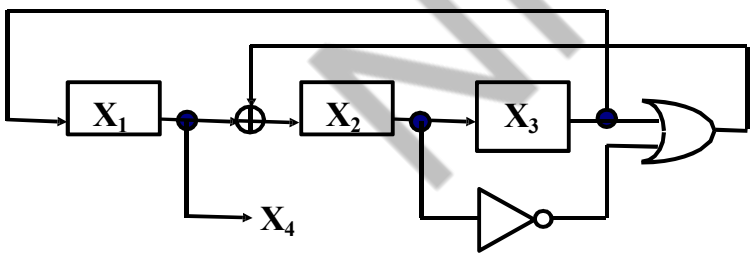


An (n, w)=(4, 2) CUT

Syndrome Driver Counter

Use SDC to generate test patterns. Check whether some inputs can share the same test signal. If *n-p* Inputs can share test inputs with other *p* inputs, then the circuit can be tested exhaustively with these *p* inputs.

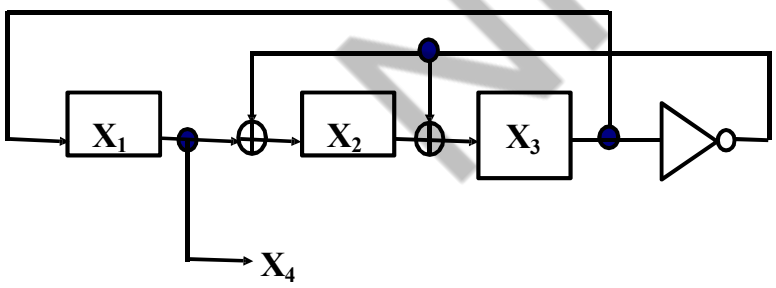
[Savir 1980]



A 3-stage syndrome driver counter

Constant-Weight Counter

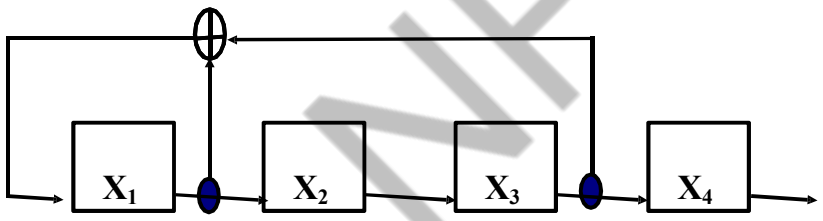
Use CWCs to generate test patterns. Constant-Weight counters are constructed using *constant-weight code* or *M-out-of-N code.* The constant-weight test set is a minimum-length test set for many circuits.



A 3-stage constant-weight counter

Combined LFSR/SR

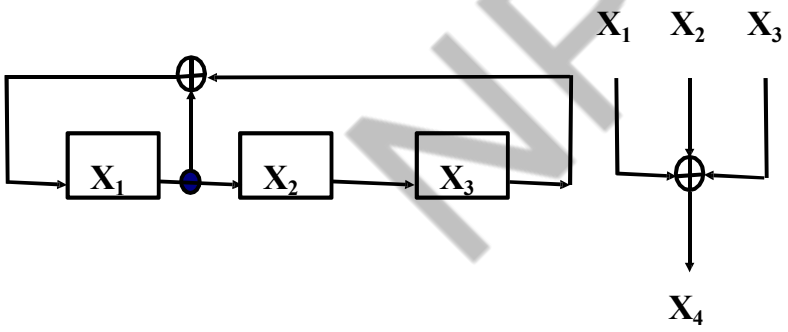
Use a combination of an LFSR and a shift register (SR) for pattern generation. The method is most effective when *w* is much less than *n*. In general, this technique requires much more tests than other schemes when *w* is greater than *n/2*.



A 4-stage combined LFSR/SR

Combined LFSR/PS

A combined LFSR/PS approach using a combination of an LFSR and a linear phase shifter which includes a network of XOR gates to generate test pattern. Similar to combined LFSR/SR, this technique requires more tests than other schemes when *w* is greater than *n/2*.



A 3-stage combined LFSR/PS

Segmentation Testing

- Used when
  - Test length using previous techniques is too long or
  - Output depends on all inputs.
- Divide the circuit into segments
  - Hardware partitioning
  - Sensitized partitioning

Output Response Analysis

- Ones count testing
- Transition count testing
- Signature analysis

Ones Count Testing

Assume the CUT has one output and the output contains a stream of *L* bits. Let the fault-free output response be

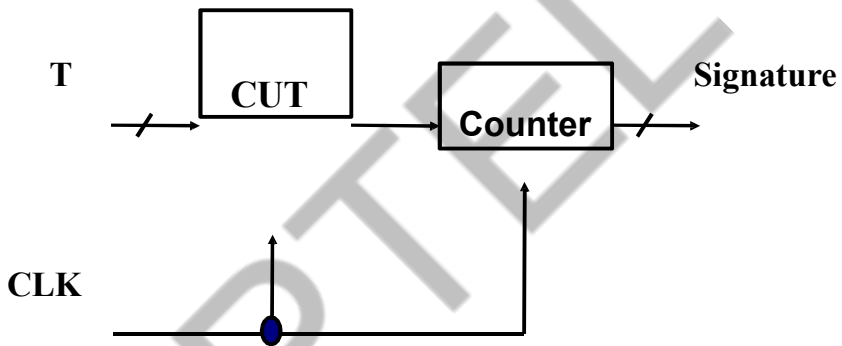
$$\{r_0, r_1, r_2 \ldots r_{L-1}\}$$

Ones count testing will need a counter to count the number of 1s in the bit stream.

Aliasing probability

$$P_{OC}(m) = (C(L,m) - 1)/(2^L - 1)$$

One Count Testing



One counter as ORA

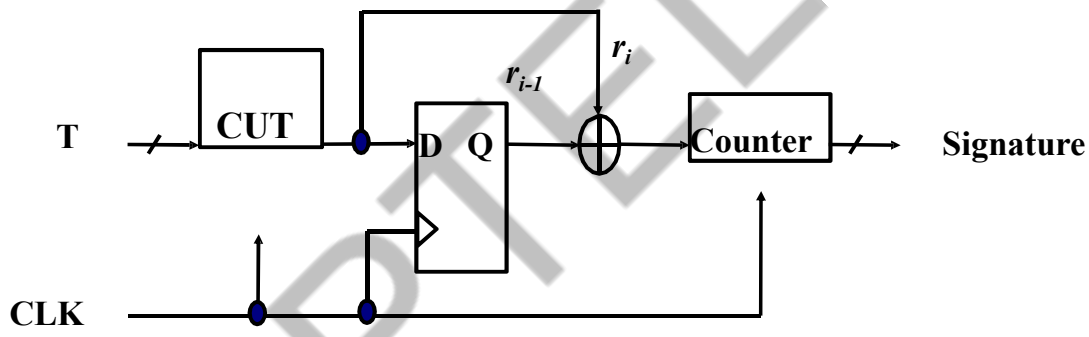
Transition Count Testing

Transition count testing is similar to that for ones count testing, except the signature is defined as the number of *1-to-0* and *0-to-1 transitions.*

Aliasing probability

$$P_{TC}(m) = (2C(L-1,m)-1)/(2^{L}-1)$$

Transition Count Testing



Transition counter as ORA

Signature Analysis

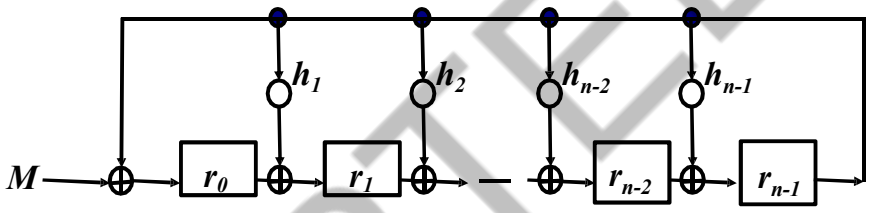
Signature analysis is the most popular compaction technique used today, based on cyclic redundancy checking.

Two signature analysis schemes

- Serial signature analysis
- Parallel signature analysis

Serial Signature Analysis

An n-stage single-input signature register



Define *L-bit* output sequence *M*

$$M(x) = m_0 + m_1 x + m_2 x + \ldots + m_{L-1} x^{L-1}$$

Let the polynomial of the modular be *f(x)*

IF $M(x) = q(x) f(x) + r(x)$ → Signature is the polynomial remainder, *r(x)*