EEDG/CE 6303: Testing and Testable Design

Mehrdad Nourani

Dept. of ECE
Univ. of Texas at Dallas

Session 10

Built-In Self-Test (BIST)

Basic Concept

Key Issues

- Motivation and economics
- Definitions for components and procedures
- *Built-in self-testing* (BIST) process
- BIST *pattern generation* (PG)
- BIST *response compaction* (RC)
- *Aliasing probability*

Motivation

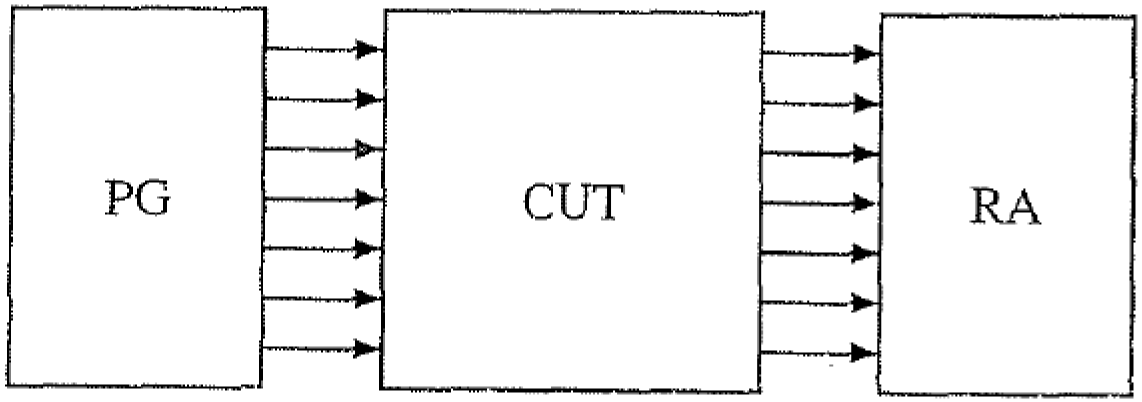
- The problems in today's semiconductor testing
  - Traditional test techniques become quite expensive
  - No longer provide sufficiently high fault coverage
- Why do we need built-in self-test (BIST)?
  - For mission-critical applications
  - Detect un-modeled faults
  - Provide remote diagnosis
- Useful for field test and diagnosis (less expensive than a ATE)
- Software tests for field test and diagnosis:
  - Low hardware fault coverage
  - Low diagnostic resolution
  - Slow to operate
- Hardware BIST benefits:
  - Lower system test effort
  - Improved system maintenance and repair
  - Improved component repair
  - Better diagnosis

Motivation (cont.)

- Costly test problems can be alleviated by BIST
  - —Increasing chip logic-to-pin ratio – harder observability
  - —Increasingly dense devices and faster clocks
  - —Increasing test generation and application times
  - —Increasing size of test vectors stored in ATE
  - —Expensive ATE needed for above 1 GHz clocking chips
  - —Hard testability insertion – designers unfamiliar with gate-level logic, since they design at behavioral level
  - —*In-circuit testing* no longer technically feasible
  - —Shortage of test engineers
  - —Circuit testing cannot be easily partitioned

BIST Concept

- Built-in self-test refers to techniques and circuit configurations that enable a chip to test itself
- In this methodology, test patterns are generated and test responses are analyzed on-chip.
- The simplest of BIST designs has a pattern generator (PG) and a response analyzer (RA)

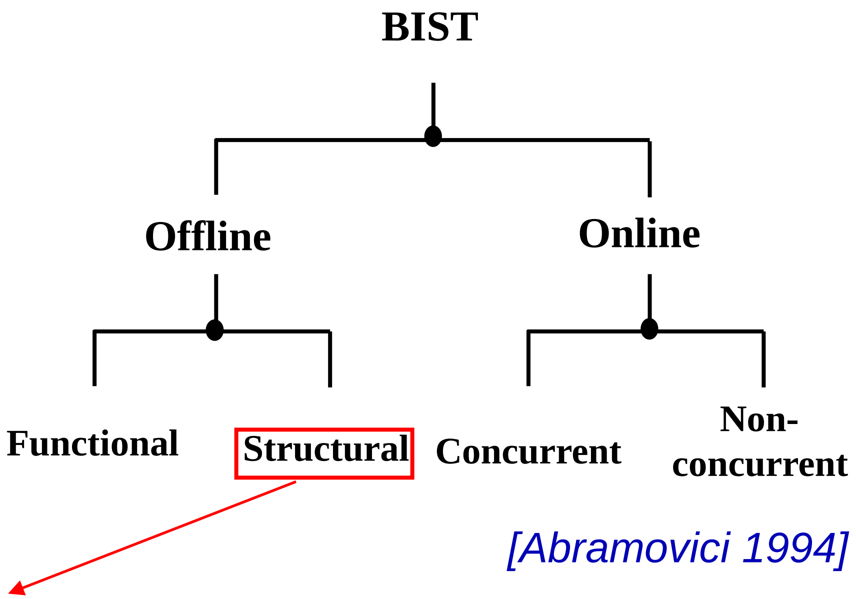


BIST Advantages

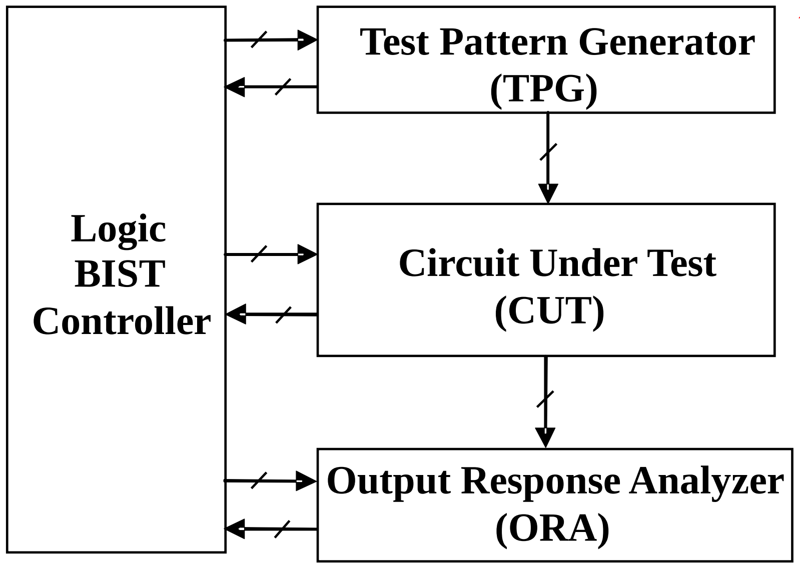
- BIST offers several advantages over testing using automatic test equipment (ATE)
  1. In BIST the test circuitry is incorporated on-chip and no external tester is required (especially attractive for high-speed circuits).
  2. Self-test can be performed at the circuit's normal clock rate.
  3. Self-testable chip has the ability to perform self-test even after it is incorporated into a system (either for periodic testing or to diagnose system failures).

Main BIST Techniques

- Logic BIST Techniques
  1. Online BIST
     - Concurrent online BIST
     - Non Concurrent online BIST
  2. Offline BIST
     - Functional offline BIST
     - Structural offline BIST
- A typical logic BIST system



[Abramovici 1994]



Economics – BIST Costs

- Chip area overhead for:
  - Test controller
  - Hardware pattern generator
  - Hardware response compacter
  - Testing of BIST hardware
- Pin overhead -- At least 1 pin needed to activate BIST operation
- Performance overhead – extra path delays due to BIST
- *Yield loss* – due to increased chip area or more chips in system because of BIST
- Reliability reduction – due to increased area
- Increased BIST hardware complexity – happens when BIST hardware is made testable

Benefits

- Faults tested:
  - Single combinational / sequential stuck-at faults
  - Delay faults
  - Single stuck-at faults in BIST hardware
- BIST benefits
  - Reduced testing and maintenance cost
  - Lower test generation cost
  - Reduced storage / maintenance of test patterns
  - Simpler and less expensive ATE
  - Can test many units in parallel
  - Shorter test application times
  - Can test at functional system speed

Definitions

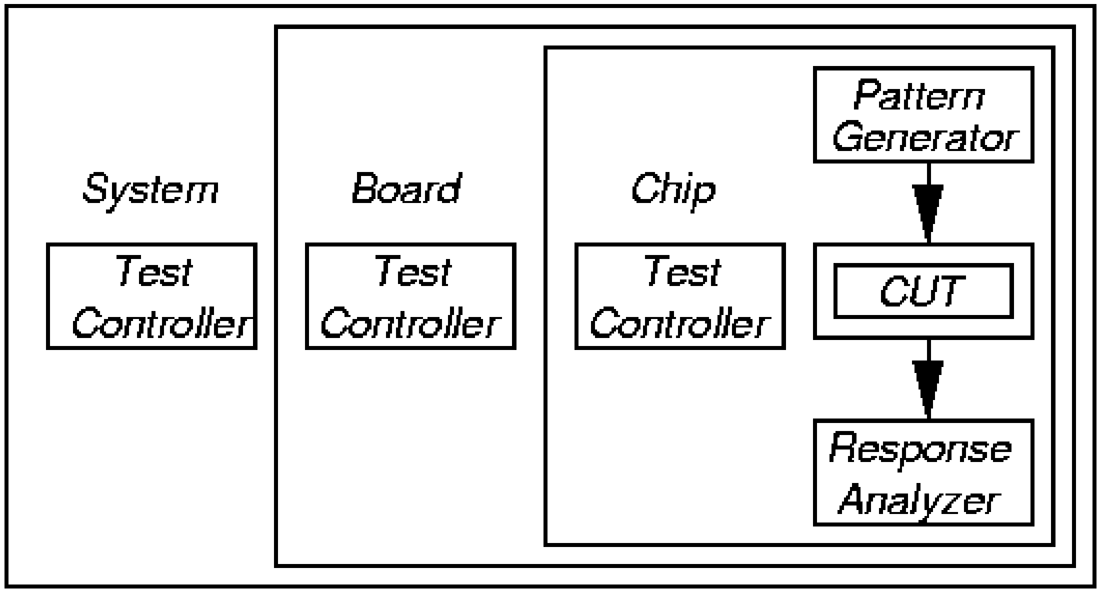
- **BILBO** – *Built-in logic block observer*, extra hardware added to flip-flops so they can be reconfigured as an LFSR pattern generator or response compacter, a scan chain, or as flip-flops
- ***Concurrent* testing** – Testing process that detects faults during normal system operation
- **CUT** – *Circuit-under-test*
- ***Exhaustive testing*** – Apply all possible 2^n patterns to a circuit with n inputs
- ***Irreducible polynomial*** – Boolean polynomial that cannot be factored
- **LFSR** – *Linear feedback shift register,* hardware that generates pseudo-random pattern sequence

Definitions (cont.)

- ***Primitive polynomial*** – Boolean polynomial *$p(x)$* that can be used to compute increasing powers *n* of *x^n modulo $p(x)$* to obtain all possible non-zero polynomials of degree less than *$p(x)$*
- ***Pseudo-exhaustive testing*** – Break circuit into small, overlapping blocks and test each exhaustively
- ***Pseudo-random testing*** – Algorithmic pattern generator that produces a subset of all possible tests with most of the properties of randomly-generated patterns
- ***Signature*** – Any statistical circuit property distinguishing between bad and good circuits
- **TPG** – Hardware *test pattern generator*

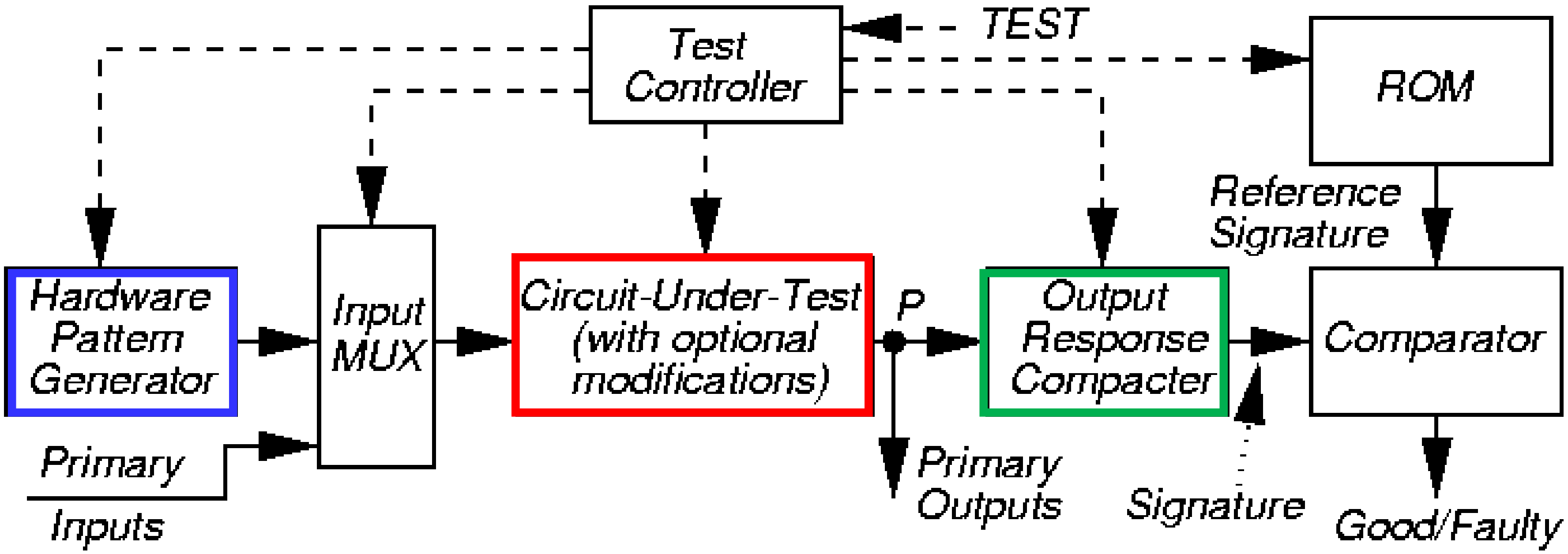
BIST Structure & Process

- *Test controller* – Hardware that activates self-test simultaneously on all PCBs
- Each board controller activates parallel chip BIST Diagnosis effective only if very high fault coverage



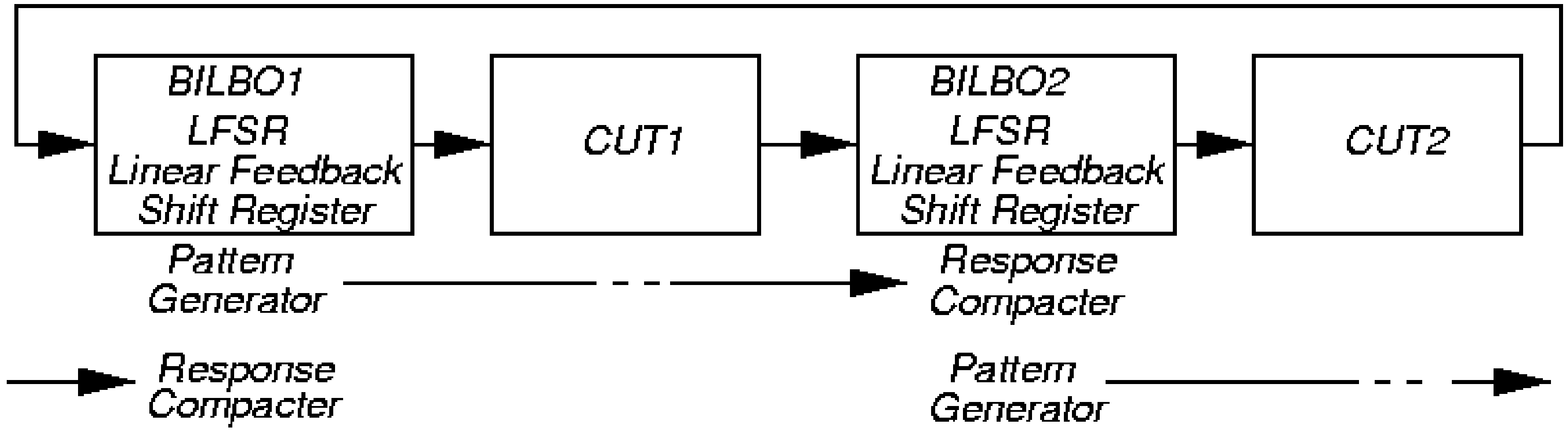
BIST Architecture

- BIST cannot test wires and transistors
  - —From PI pins to Input MUX
  - —From POs to output pins



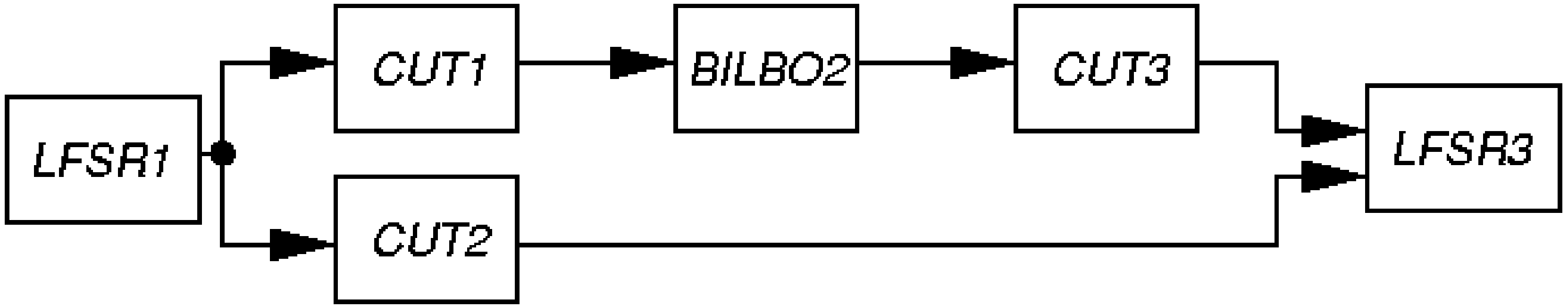
Built In Logic Block Observer (BILBO)

- Can work as a PG or RC. It has 4 modes:
  1. Flip-flop
  2. LFSR pattern generator
  3. LFSR response compacter
  4. Scan chain for flip-flops



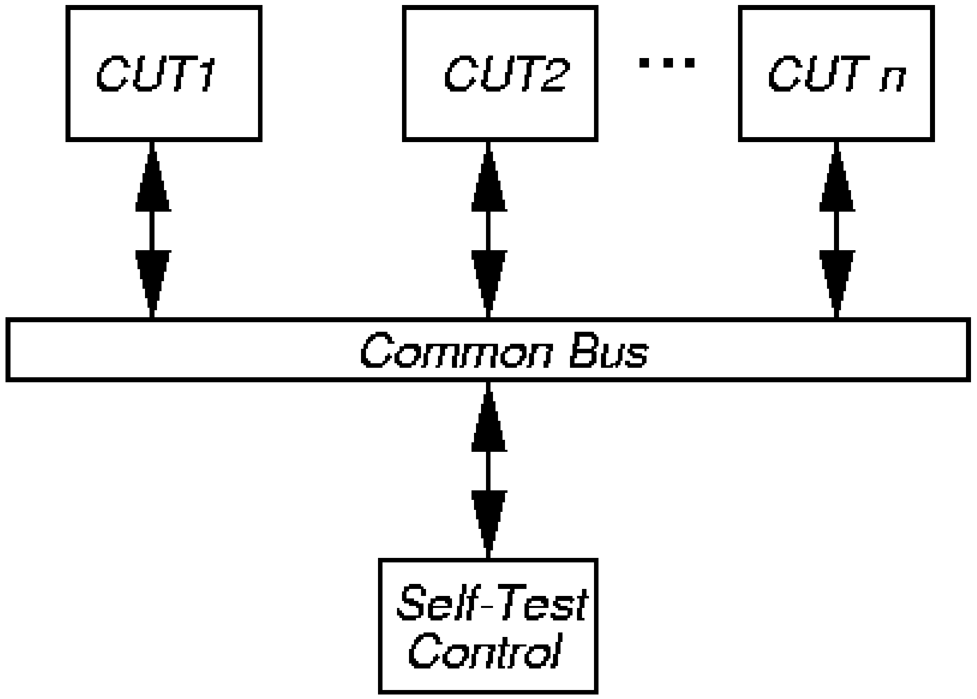
Complex BIST Architecture

- Testing session I
  - LFSR1 generates tests for CUT1 and CUT2
  - BILBO2 (LFSR3) compacts CUT1 (CUT2)
- Testing session II
  - BILBO2 generates test patterns for CUT3
  - LFSR3 compacts CUT3 response



Bus-Based BIST Architecture

- *Self-test control* broadcasts patterns to each CUT over bus – parallel pattern generation
- Awaits bus transactions showing CUT's responses to the patterns: serialized compaction



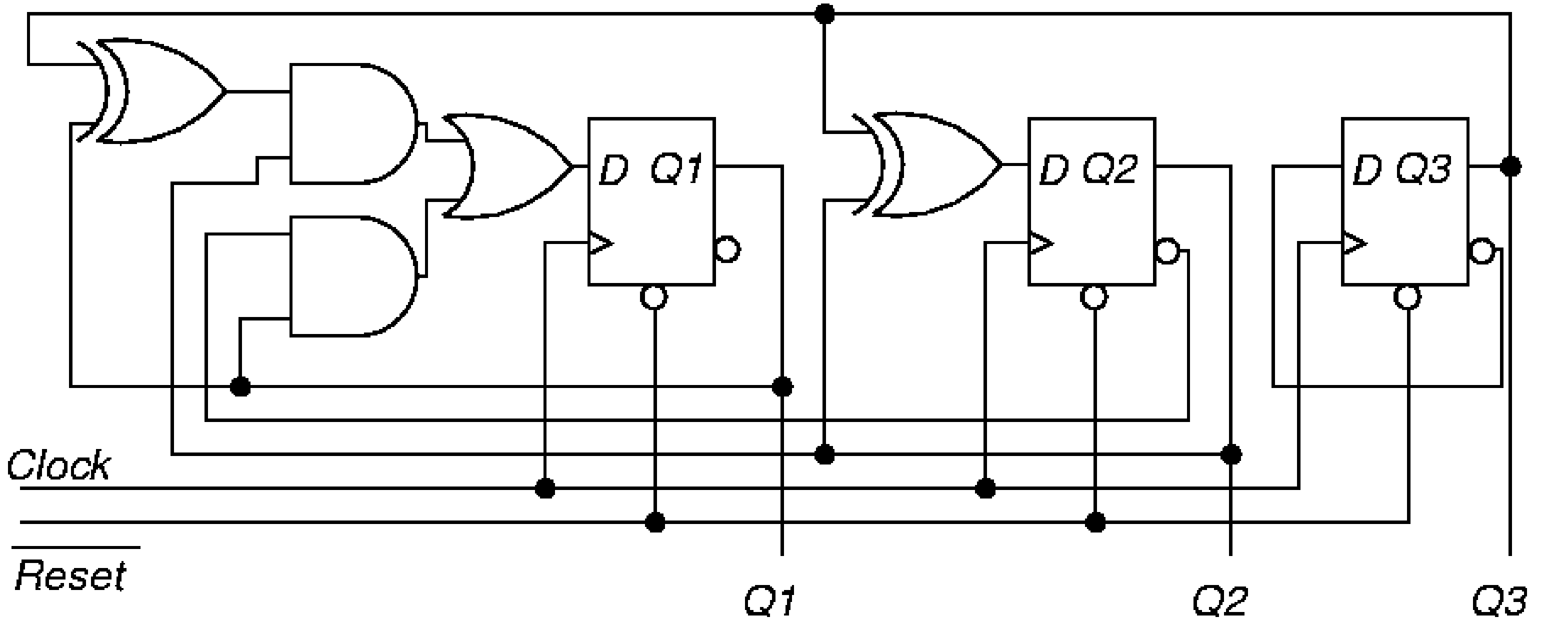
Pattern Generation in BIST

Mechanisms for Pattern Generation

- Store in ROM – too expensive
- *Exhaustive*
- *Pseudo-exhaustive*
- *Pseudo-random* (LFSR) – Preferred method
- Binary counters – use more hardware than LFSR
- Modified counters
- Test pattern *augmentation*
  - LFSR combined with a few patterns in ROM
  - *Hardware diffracter* – generates pattern cluster in neighborhood of pattern stored in ROM

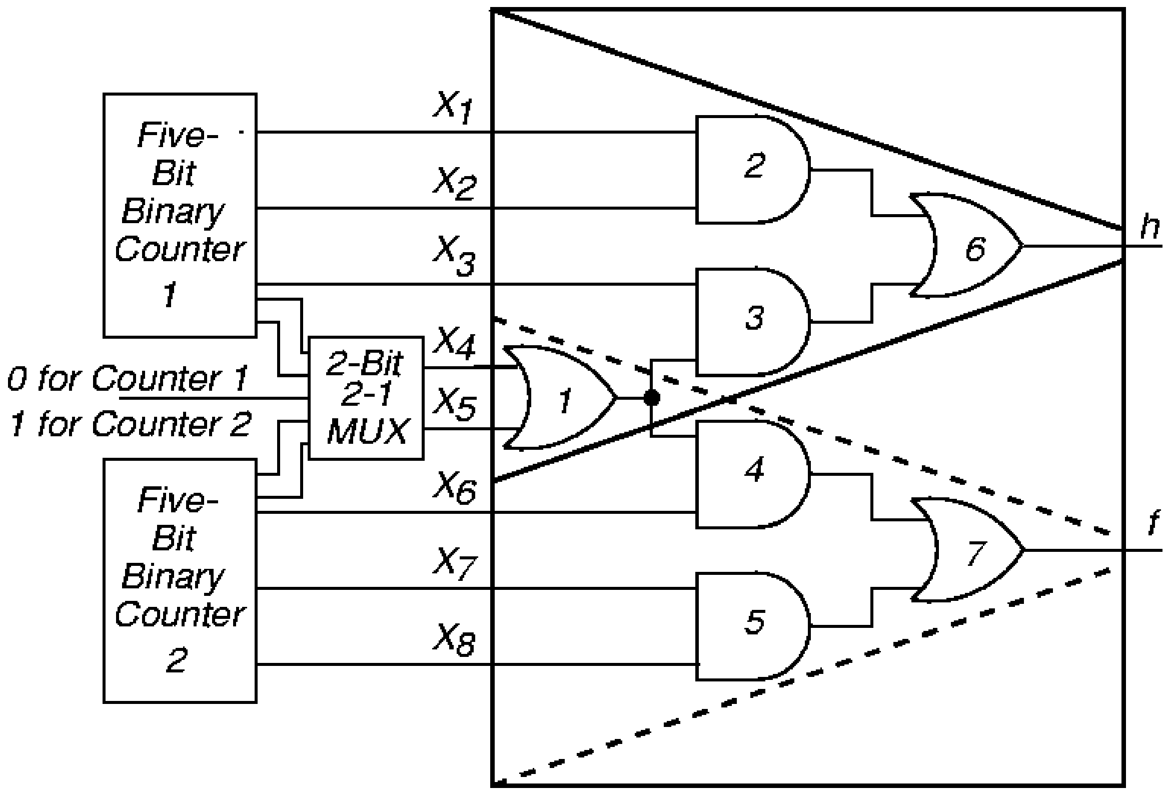
Exhaustive Pattern Generation

- Shows that every state and transition works
- For *n*-input circuits, requires all 2^n vectors
- An n-bit counter can do the job
- Impractical (in terms of running time) for $n > 20$



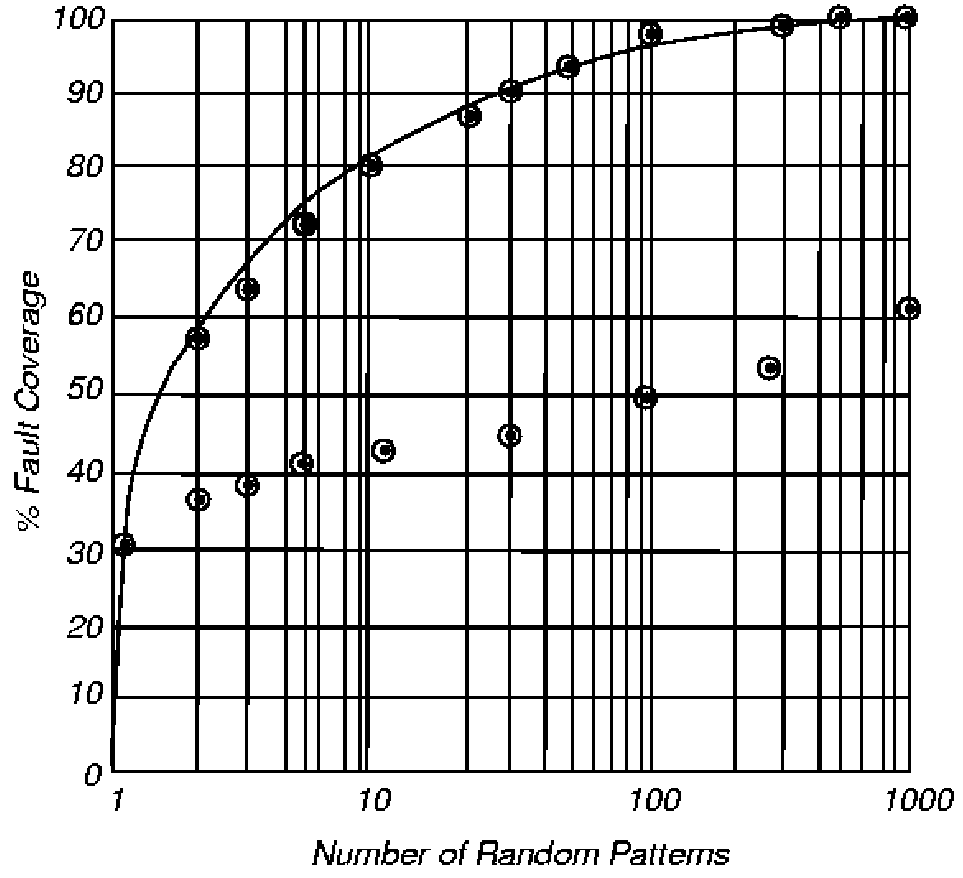
Pseudo-Exhaustive Method

- Partition large circuit into *fanin cones*
  - Backtrace from each PO to PIs influencing it
  - Test fanin cones in parallel
- Reduced # of tests from $2^8 = 256$ to $2^5 \times 2 = 64$
  - Incomplete fault coverage



Quality of Random Pattern Testing

- Some circuits may have random-pattern resistant faults. Thus, the quality of test may be low.



(a) Top curve -- random pattern testing with acceptable fault coverage.
(b) Bottom curve -- unacceptable random pattern testing.

Linear Feedback Shift Register (LFSR)

- Used to generate pseudo-random patterns
- Produces patterns algorithmically – repeatable
  - Has most of desirable random # properties
  - Near-exhaustive (maximal length) LFSR
  - Cycles through $2^n - 1$ states (excluding all-0)
- Need not cover all 2^n input combinations
- Long sequences needed for good fault coverage
- Two types of LFSR
  1. Internal XOR (modular LFSR)
  2. External XOR (standard LFSR)
- Every LFSR is described by its feedback (characteristic) polynomial represented as: $\varphi(x) = \varphi_n x^n + \ldots + \varphi_1 x + \varphi_0$

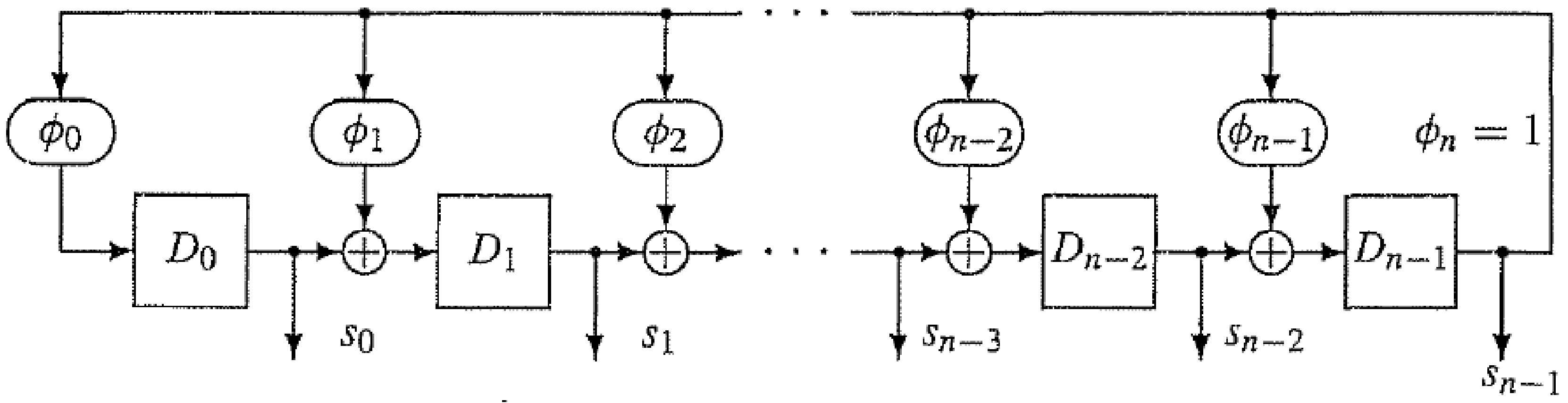
LFSR with Internal XOR

- The polynomial $\varphi(x) = \varphi_n x^n + \ldots + \varphi_1 x + \varphi_0$
- The state transition relationship: s(t+1)=A*s(t), where * is matrix multiplication using modulo-2 arithmetic and A is an nxn matrix:

$$\begin{pmatrix} 0 & 0 & 0 & \cdots & 0 & \phi_0 \\ 1 & 0 & 0 & \cdots & 0 & \phi_1 \\ 0 & 1 & 0 & \cdots & 0 & \phi_2 \\ 0 & 0 & 1 & \cdots & 0 & \phi_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & \phi_{n-1} \end{pmatrix}$$

(n-1) X (n-1) Identity Matrix

Note the order of φ_i's



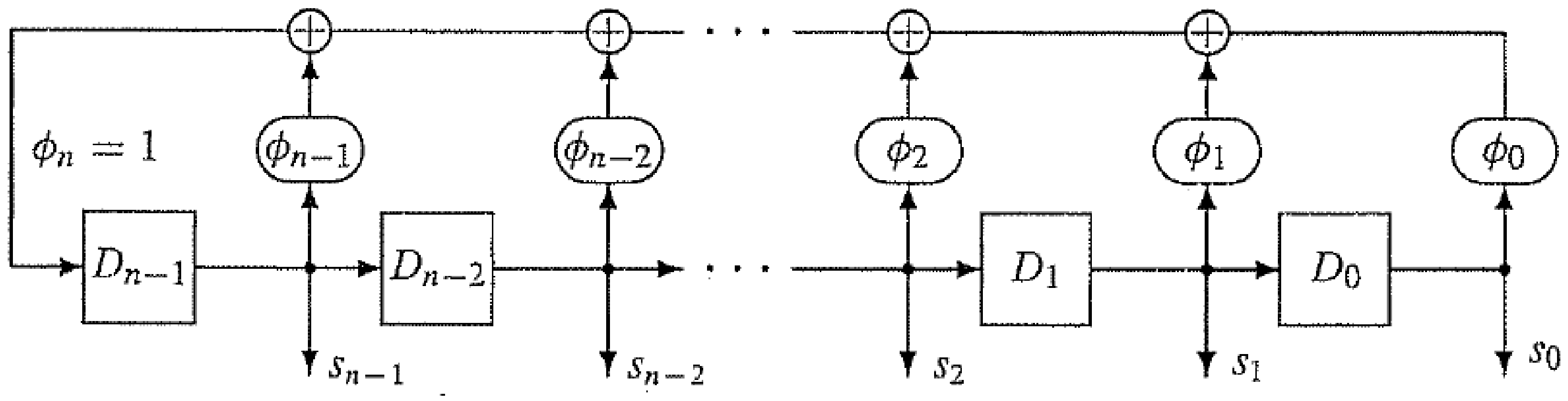
LFSR with External XOR

- The polynomial $\varphi(x) = \varphi_n x^n + \ldots + \varphi_1 x + \varphi_0$
- The state transition relationship: s(t+1)=A*s(t), where * is matrix multiplication using modulo-2 arithmetic and A is an nxn matrix:

(n-1) X (n-1) Identity Matrix

$$\begin{pmatrix} 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 1 \\ \phi_0 & \phi_1 & \phi_2 & \cdots & \phi_{n-2} & \phi_{n-1} \end{pmatrix}$$

Note the order of φ_i's



LFSR Example – Internal XOR

- The polynomial: $\varphi(x) = x^4 + x + 1$
- The transition matrix/relation:

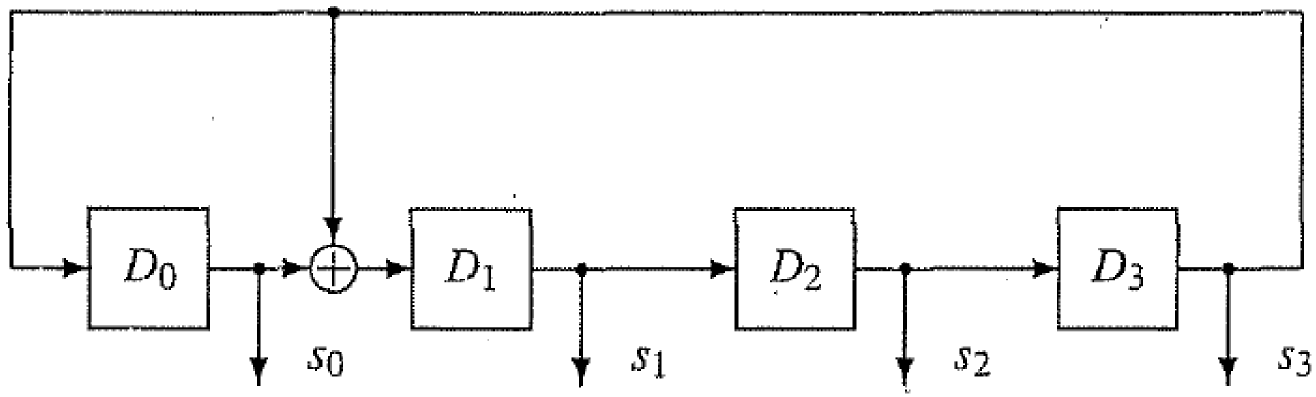
$$\begin{bmatrix} s_0(t+1) \\ s_1(t+1) \\ s_2(t+1) \\ s_3(t+1) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} s_0(t) \\ s_1(t) \\ s_2(t) \\ s_3(t) \end{bmatrix}$$

- The sequence:

$$\begin{array}{|c|cccccccccccccc|c|} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \end{array}$$

Initial Value

Cyclic Repetition



LFSR Example – External XOR

- The polynomial: φ(x) = $x^4 + x + 1$
- The transition matrix:

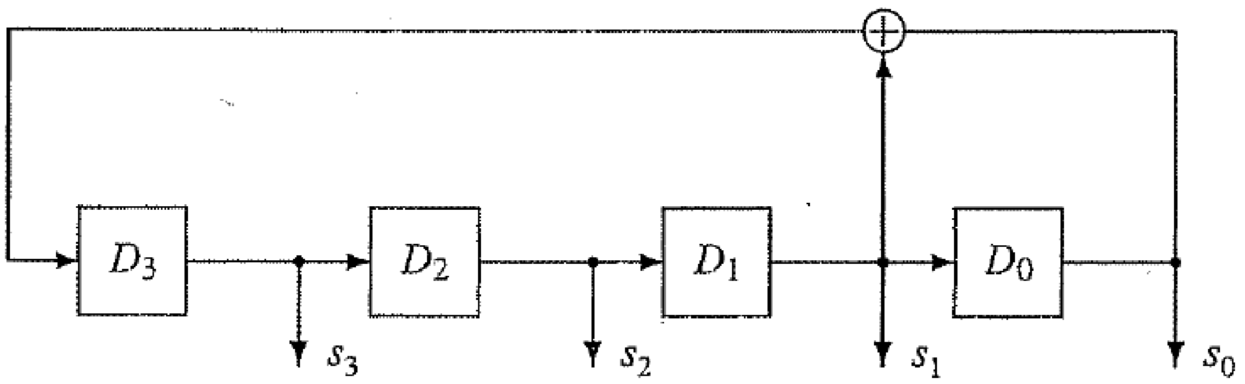
$$\begin{bmatrix} s_0(t+1) \\ s_1(t+1) \\ s_2(t+1) \\ s_3(t+1) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} s_0(t) \\ s_1(t) \\ s_2(t) \\ s_3(t) \end{bmatrix}$$

- The sequence:

$$\begin{array}{|c|cccccccccccccc|c|} \hline 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ \hline \end{array}$$

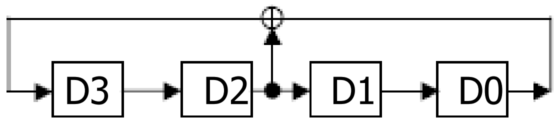
Initial Value

Cyclic Repetition

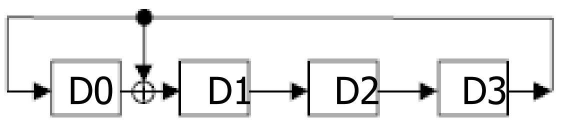


Primitive vs. Non-Primitive Polynomial

- (a) uses a non-primitive polynomial $\varphi(x) = x^4+x^2+1$
- (b) uses a primitive polynomial $\varphi(x) = x^4+x+1$



(a) A 4-stage standard LFSR



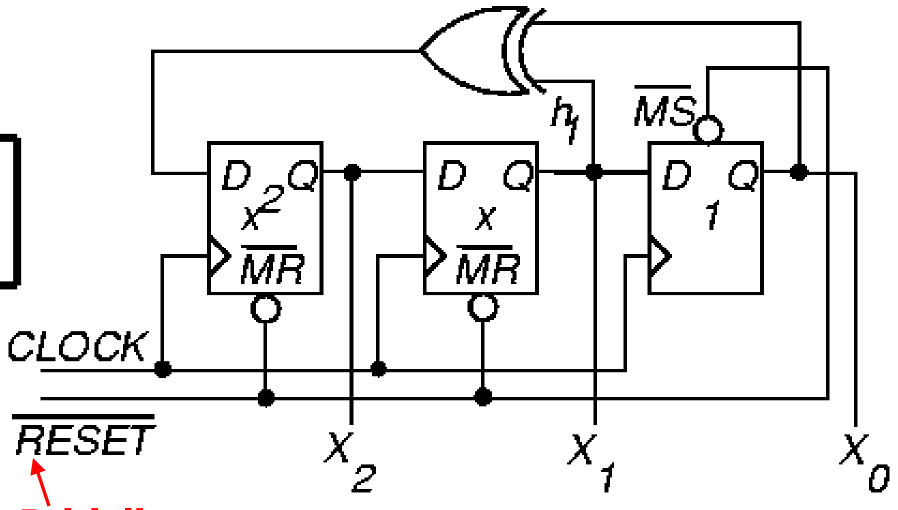
(b) A 4-stage modular LFSR

0 0 0 1	**0 0 0 1**
1 0 0 0	1 1 0 0
0 1 0 0	0 1 1 0
1 0 1 0	0 0 1 1
0 1 0 1	1 1 0 1
0 0 1 0	1 0 1 0
0 0 0 1	0 1 0 1
1 0 0 0	1 1 1 0
0 1 0 0	0 1 1 1
1 0 1 0	1 1 1 1
0 1 0 1	1 0 1 1
0 0 1 0	1 0 0 1
0 0 0 1	1 0 0 0
1 0 0 0	0 1 0 0
0 1 0 0	0 0 1 0
1 0 1 0	**0 0 0 1**

Other Examples

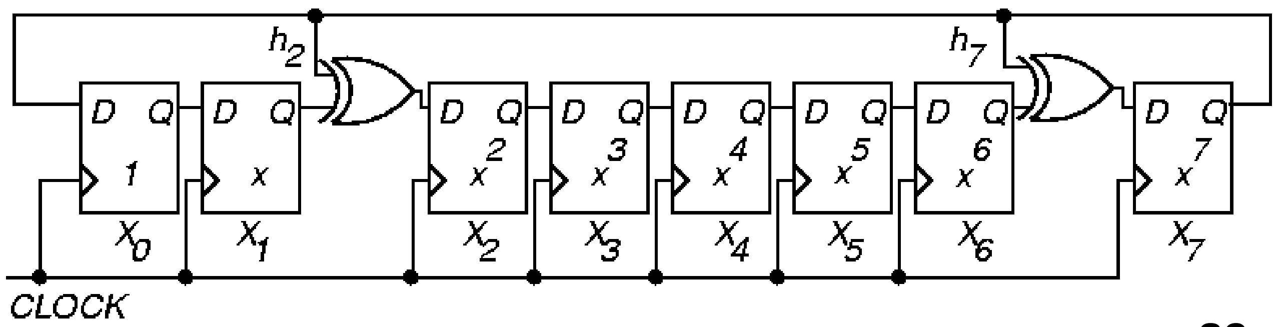
- **External XOR with polynomial φ (*x*) = x^3+x+1**
  (read taps from right to left)

$$\begin{bmatrix} s_0(t+1) \\ s_1(t+1) \\ s_2(t+1) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} s_0(t) \\ s_1(t) \\ s_2(t) \end{bmatrix}$$



Initializes to $x_2x_1x_0$=001

- **φ(*x*) = $x^8+x^7+x^2+1$**
  (read taps from left to right)



LFSR Theory

LFSR and Field Theory

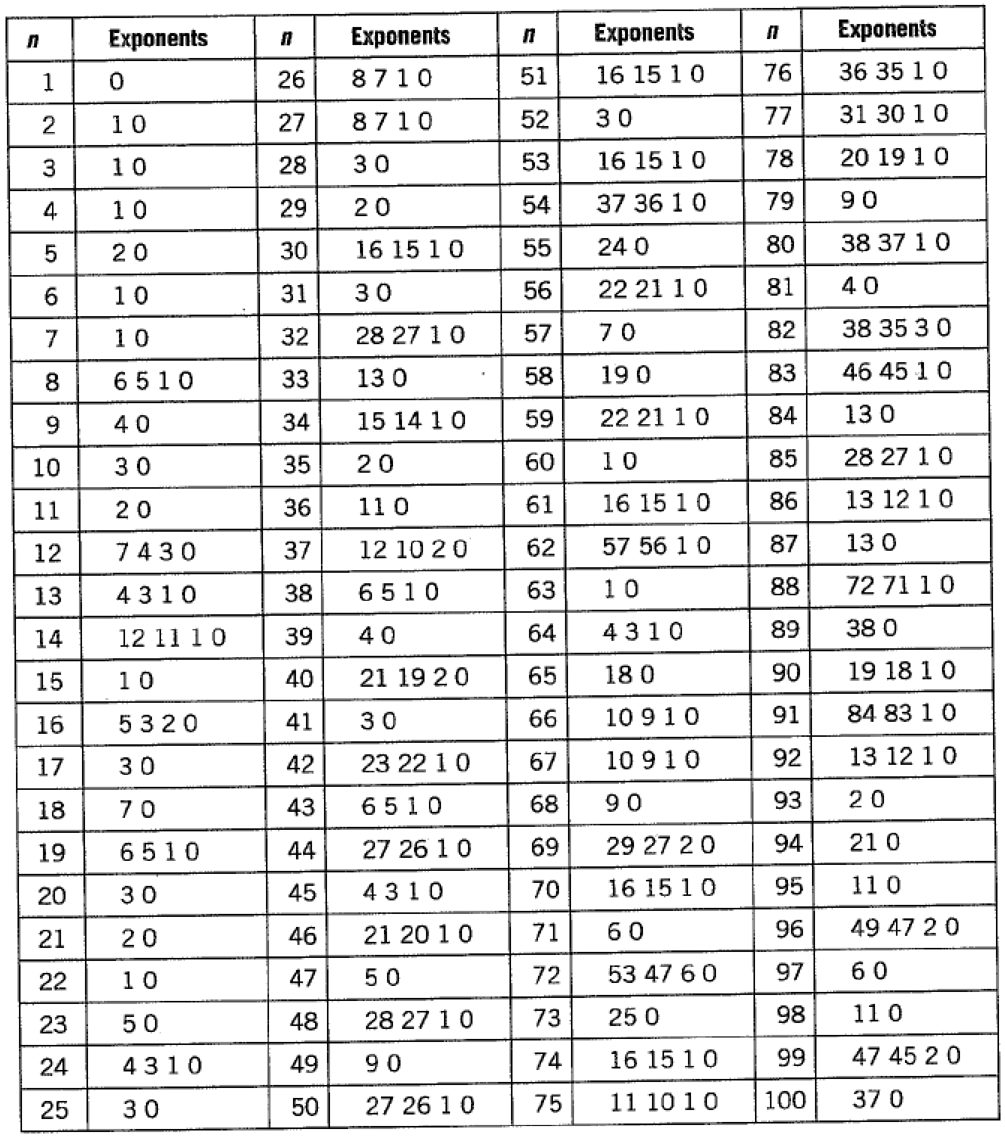
- The foundation of LFSR is a mature field theory used in many fields such as coding, encryption and testing.
- *Galois field* (mathematical system):
  - —Multiplication by *x* same as right shift of LFSR
  - —Addition operator is XOR
- Due to its linearity
  - —LFSR with all-zero state will remain there forever
  - —starting from a non-zero state cycles through at most 2^n -1 states

Primitive (Irreducible) Polynomials

- The maximum length sequence (2^n-1) can be generated only if the LFSR uses a primitive polynomial.
- For a given length n, the primitive polynomial is not always unique.

Degree	Primitive polynomial
2	$x^2 + x + 1$
3	$x^3 + x + 1$
3	$x^3 + x^2 + 1$
4	$x^4 + x + 1$
4	$x^4 + x^3 + 1$
5	$x^5 + x^2 + 1$
6	$x^6 + x + 1$
7	$x^7 + x + 1$
8	$x^8 + x^4 + x^3 + x^2 + 1$
9	$x^9 + x^4 + 1$
10	$x^{10} + x^3 + 1$

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 $p(x) = x^{24} + x^4 + x^3 + x^1 + x^0$

Galois Fields

- finite fields play a key role in cryptography
- can show number of elements in a finite field **must** be a power of a prime, i.e. p^n where `n` is a positive integer
- known as Galois fields and denoted by $GF(p^n)$
- in particular often use these fields:
  - $n=1 \rightarrow GF(p)$
  - $p=2 \rightarrow GF(2^n)$

Polynomial Arithmetic with Modulo Coefficients

- when computing value of each coefficient do calculation modulo some value
- could be modulo any prime
- but we are most interested in `mod 2`
  - i.e. all coefficients are 0 or 1
  - Addition `mod 2` is the same as XOR
  - Multiplication `mod 2` is the same as AND
  - e.g. let $f(x) = x^3 + x^2$ and $g(x) = x^2 + x + 1$
    - $f(x) + g(x) = x^3 + x + 1$ (e.g. $2x^2$ does not exist because $x^2+x^2 \rightarrow 1+1 \pmod 2) = 1 \oplus 1 = 0$)
    - $f(x) \times g(x) = x^5 + x^2$

Polynomial Arithmetic Over GF(2)

$$
\begin{array}{lllll}
x^7 & + x^5 + x^4 + x^3 & & + \; x \; + \; 1 & \\
 & + (x^3 & & + \; x \; + \; 1 \,) & \\
\hline
x^7 & + x^5 + x^4 & & &
\end{array}
$$

(a) Addition

$$
\begin{array}{lllll}
x^7 & + x^5 + x^4 + x^3 & & + \; x \; + \; 1 & \\
 & - (x^3 & & + \; x \; + \; 1 \,) & \\
\hline
x^7 & + x^5 + x^4 & & &
\end{array}
$$

(b) Subtraction

$$
\begin{array}{llllll}
 & & x^7 & + x^5 + x^4 + x^3 & & + \; x \; + \; 1 \\
 & & & \times (x^3 & & + \; x \; + \; 1 \,) \\
\cline{3-6}
 & & x^7 & + x^5 + x^4 + x^3 & & + \; x \; + \; 1 \\
 & x^8 & + x^6 + x^5 + x^4 & & + x^2 + x & \\
x^{10} & + x^8 + x^7 + x^6 & & + x^4 + x^3 & & \\
\hline
x^{10} & & & + x^4 & + x^2 & + \; 1
\end{array}
$$

(c) Multiplication

$$
\begin{array}{r|llll}
\multicolumn{1}{r}{} & x^4 + 1 & & & \\
\cline{2-5}
x^3 + x + 1 & x^7 & + x^5 + x^4 + x^3 & & + \; x \; + \; 1 \\
 & x^7 & + x^5 + x^4 & & \\
\cline{2-3}
 & & & x^3 & + \; x \; + \; 1 \\
 & & & x^3 & + \; x \; + \; 1 \\
\cline{4-5}
\end{array}
$$

(d) Division

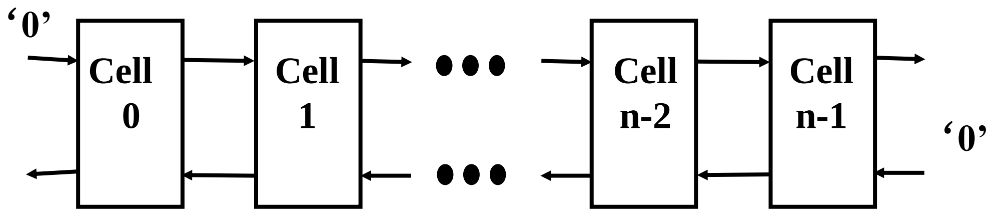
Modular Polynomial Arithmetic

- can write any polynomial in the form:
  - $f(x) = q(x)\ g(x) + r(x)$
  - can interpret $r(x)$ as being a remainder
  - $r(x) = f(x) \bmod g(x)$
- if have no remainder say $g(x)$ divides $f(x)$ written as $g(x)|f(x)$
- if $g(x)$ has no divisors other than itself & 1 say it is **irreducible** (called also **prime** or **primitive**) polynomial
- arithmetic modulo an irreducible polynomial forms a field

Using Cellular Automata

Cellular Automata (CA)

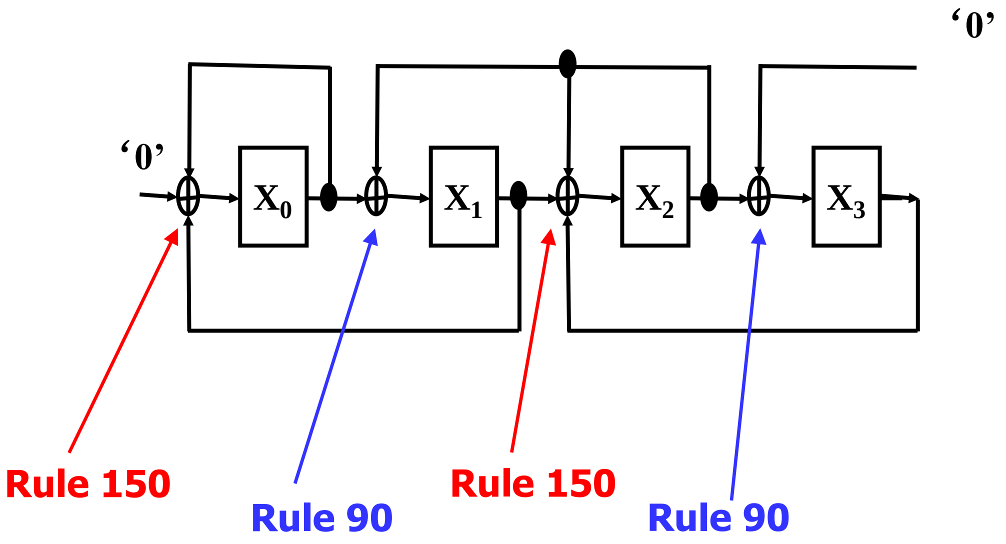
- Provide more random test patterns
- Provide high fault coverage in a random-pattern resistant (RP-resistant) circuit
- Implementation advantage
- A general structure of an n-stage CA is based on some rules
  - Rule 90: $x_i(t+1) = x_{i-1}(t) + x_{i+1}(t)$
  - Rule 150: $x_i(t+1) = x_{i-1}(t) + x_i(t) + x_{i+1}(t)$
  - Each rule determines the next state of a cell based on the state of the cell and its neighbors



Example of Cellular Automaton

A 4-stage CA

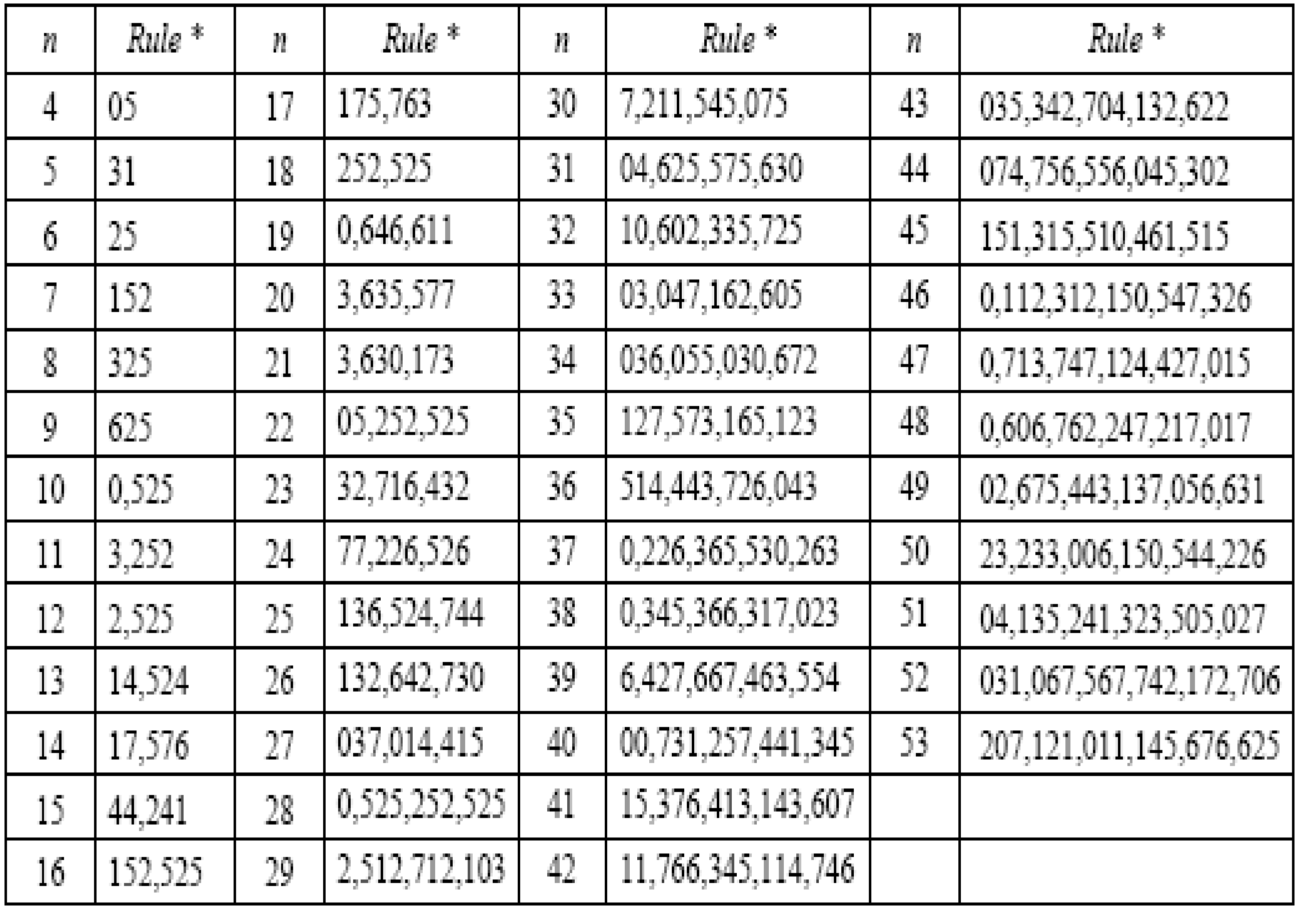
Test sequence



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

CA Rule Notation: 05 = 0101

Construction Rules for CAs of Length n up to 53



n	*Rule* *	*n*	*Rule* *	*n*	*Rule* *	*n*	*Rule* *
4	05	17	175,763	30	7,211,545,075	43	035,342,704,132,622
5	31	18	252,525	31	04,625,575,630	44	074,756,556,045,302
6	25	19	0,646,611	32	10,602,335,725	45	151,315,510,461,515
7	152	20	3,635,577	33	03,047,162,605	46	0,112,312,150,547,326
8	325	21	3,630,173	34	036,055,030,672	47	0,713,747,124,427,015
9	625	22	05,252,525	35	127,573,165,123	48	0,606,762,247,217,017
10	0,525	23	32,716,432	36	514,443,726,043	49	02,675,443,137,056,631
11	3,252	24	77,226,526	37	0,226,365,530,263	50	23,233,006,150,544,226
12	2,525	25	136,524,744	38	0,345,366,317,023	51	04,135,241,323,505,027
13	14,524	26	132,642,730	39	6,427,667,463,554	52	031,067,567,742,172,706
14	17,576	27	037,014,415	40	00,731,257,441,345	53	207,121,011,145,676,625
15	44,241	28	0,525,252,525	41	15,376,413,143,607		
16	152,525	29	2,512,712,103	42	11,766,345,114,746		

[Hortensius 1989]

**For n=7, Rule=152=001,101,010=1,101,010, where "0" denotes a rules 90 and "1" denotes a rule 150 cell, or vice versa*