Designing for Test

Testing of Digital Systems

Boundary Scan

Board-Level Testing

- Each chip used in a PCB is pretested by the chip's vendor and declared fault-free
- Testing at board level focuses on
  - —Inter-chip interconnect faults
    - – Opens
      - + Caused when chip pins do not bond properly to board
      - + That occur in PCB traces due to defects during PCB manufacturing
    - – Shorts caused when extra solder flows between pins or PCB traces
  - —Faults internal to chips
    - – Faults induced due to improper handling, e.g., excessive heat or shock during PCB assembly
    - – Faulty behaviors that become apparent only when a chip is integrated into a PCB, e.g., the ability to drive a large load
- Since repair possible, diagnosis also important

Board-Level Testing (cont.)

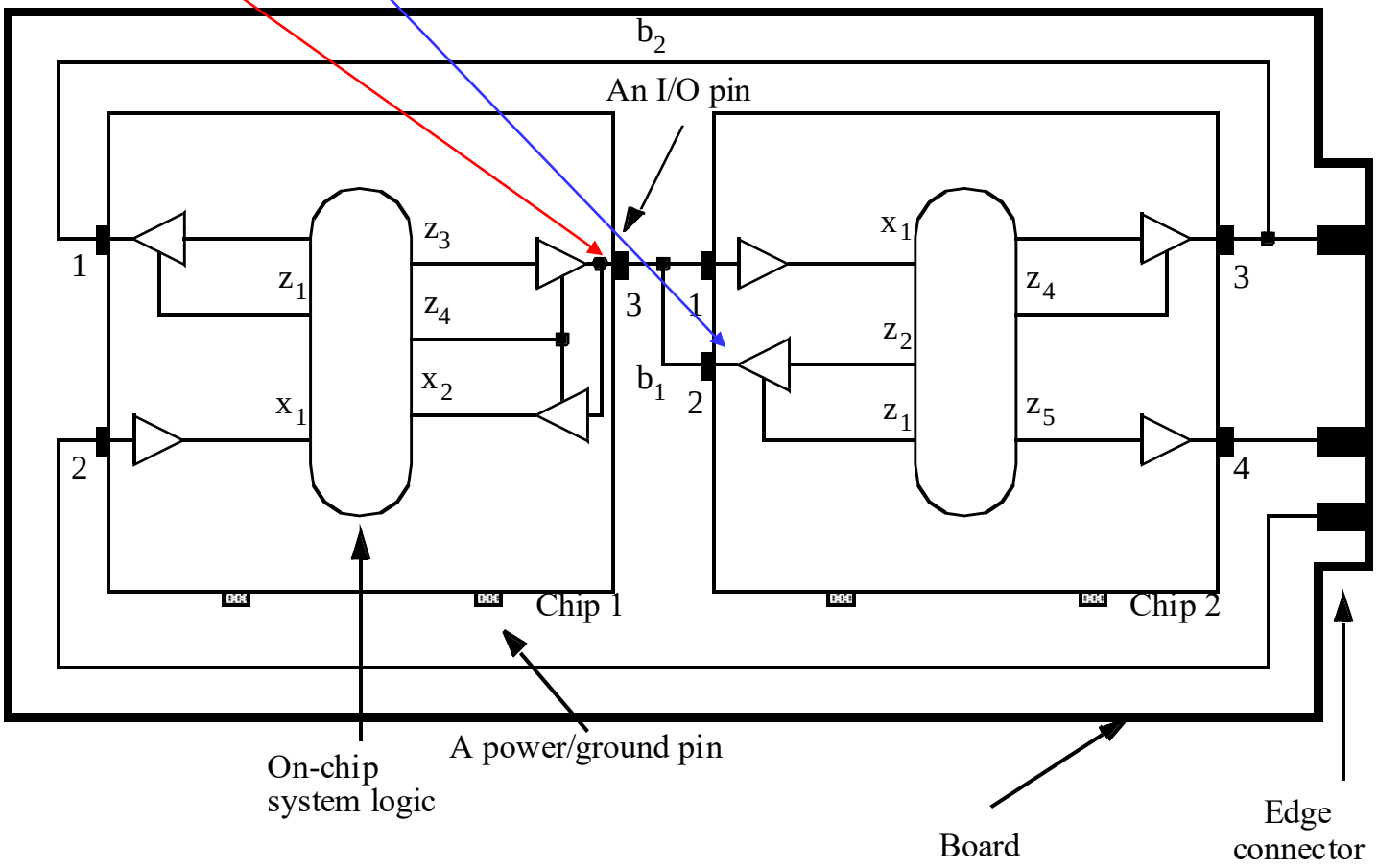
- Possible approaches for board testing
  - —Consider the entire board as one circuit and generate tests: Not possible because
    - The circuit too large for ATPG tools
    - Net-list of most chips unavailable
  - —In-circuit testing
    - Probes were used to access input/output pins of chips
    - Care had to be taken to ensure probes driving values did not damage output pins
    - No longer used because
      - Pins are too small and too close to be reliably probed
      - Pins at bottoms of chips cannot be probed in multi-layered PCBs
  - —Boundary scan

Boundary Scan Structure

- Boundary scan incorporates DFT circuitry that allows direct access to chip input and output pins via scan chains
- A board contains chips from multiple manufacturers
- Boundary scan circuits must interoperate to achieve above objectives
- Hence, a standard, namely IEEE Std 1149.1, defined

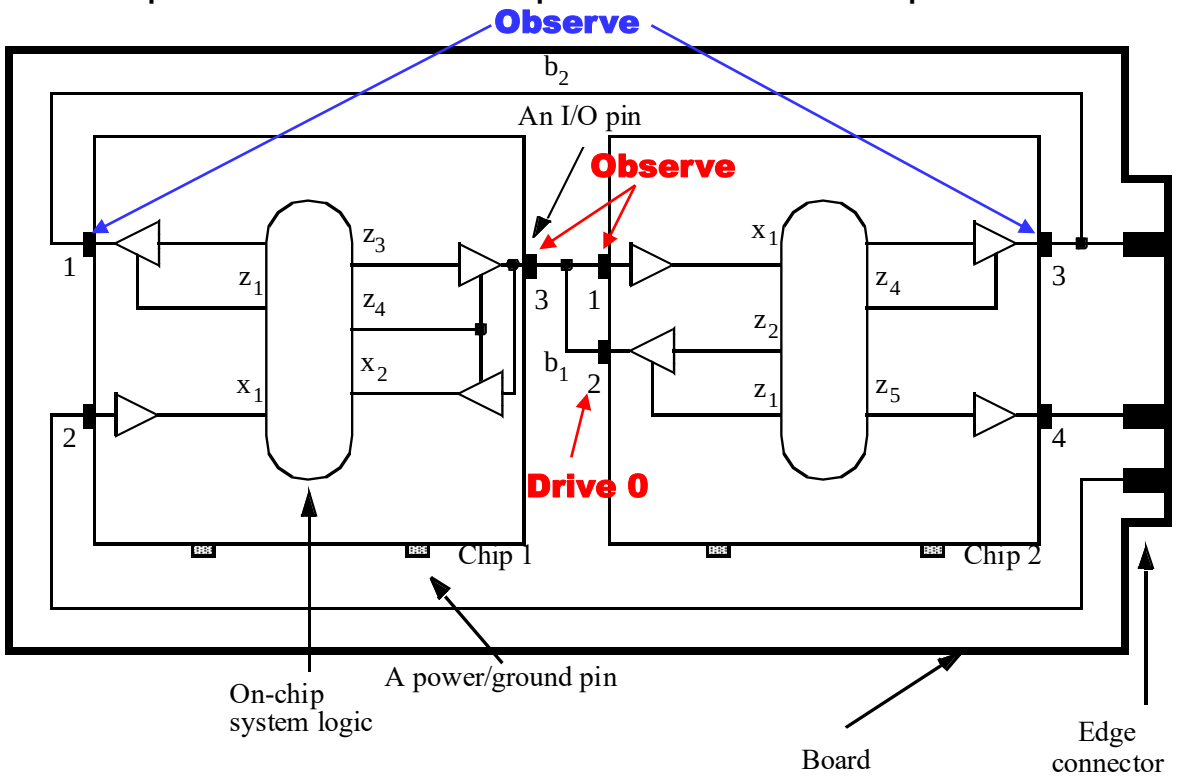
Example: A Board Without Boundary Scan

- Consider an example board. Especially note
  - Bi-directional driver/receiver – Pin 3 of Chip 1
  - Tri-state driver – Pin 2 of Chip 2



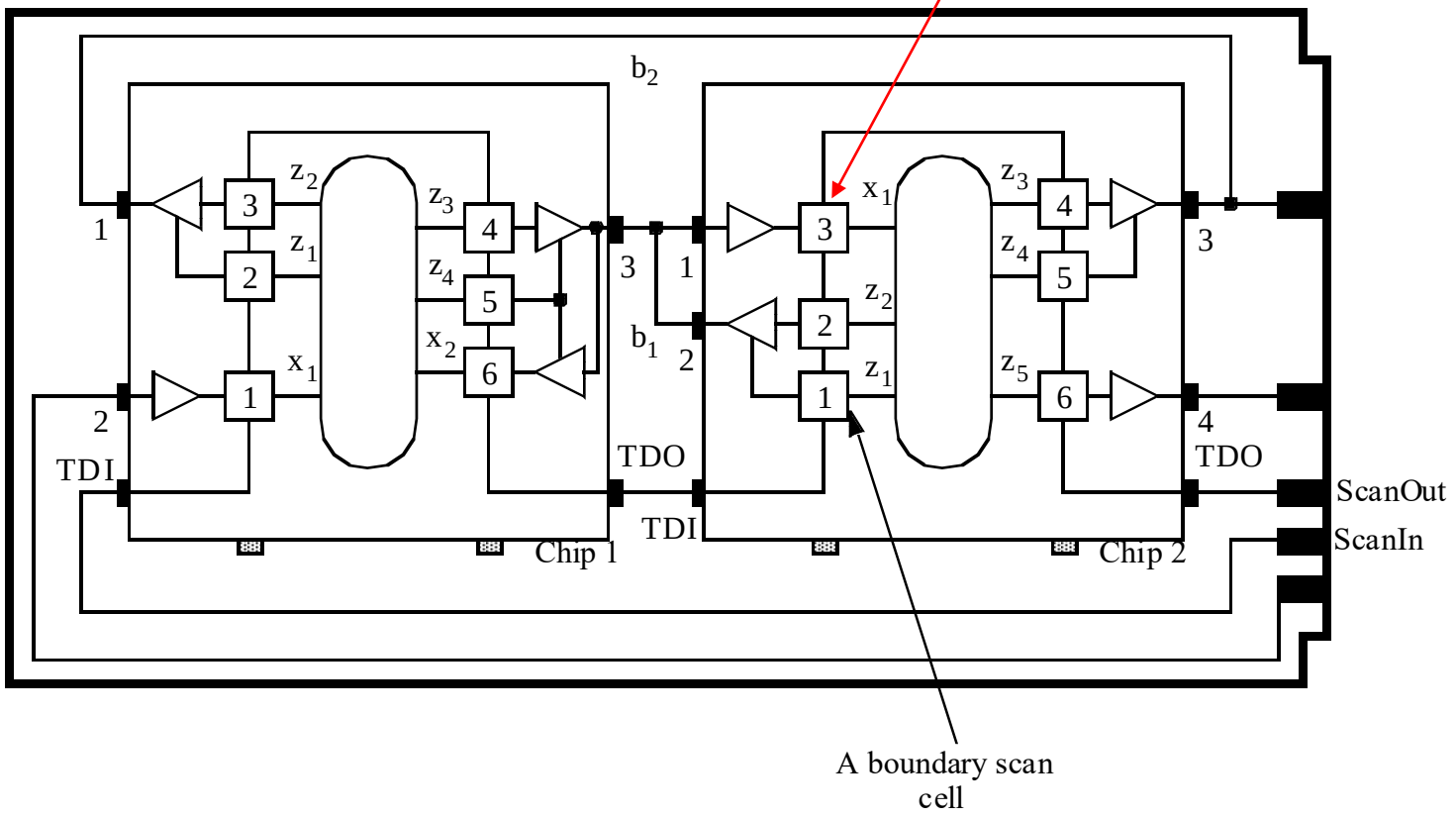
Example: Testing Without Boundary Scan (cont.)

- Assume that
  - A receiver disconnected from its drivers due to opens interprets the input as a value 1
  - A control value 1 enables a tri-state/bi-directional driver
- To test for an open at net b1
  - Use a probe to apply 0 at Pin 2 of Chip 2
  - Use another probe to observe the value at Pin 3 of Chip 1 and Pin 1 of Chip 2
- To test for an short between b1 and b2
  - Probe above pins and Pin 1 of Chip 1 and Pin 3 of Chip 2



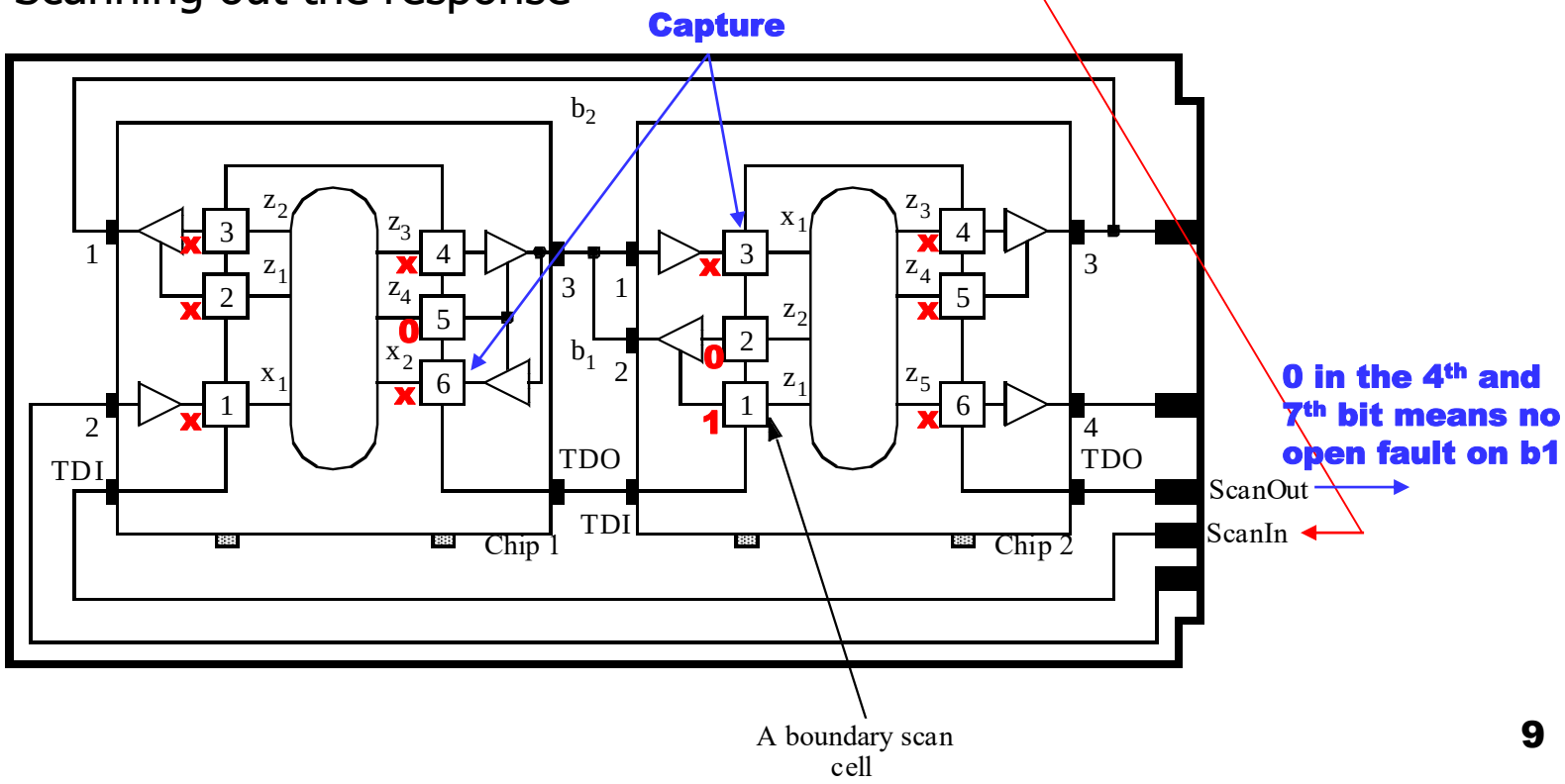
Example: A Board With Boundary Scan

- Boundary scan register (BSR)
  - Formed using boundary scan cells (BSCs)
    - At each input and output pin
    - As well as at control cells of tri-state and bi-directional pins

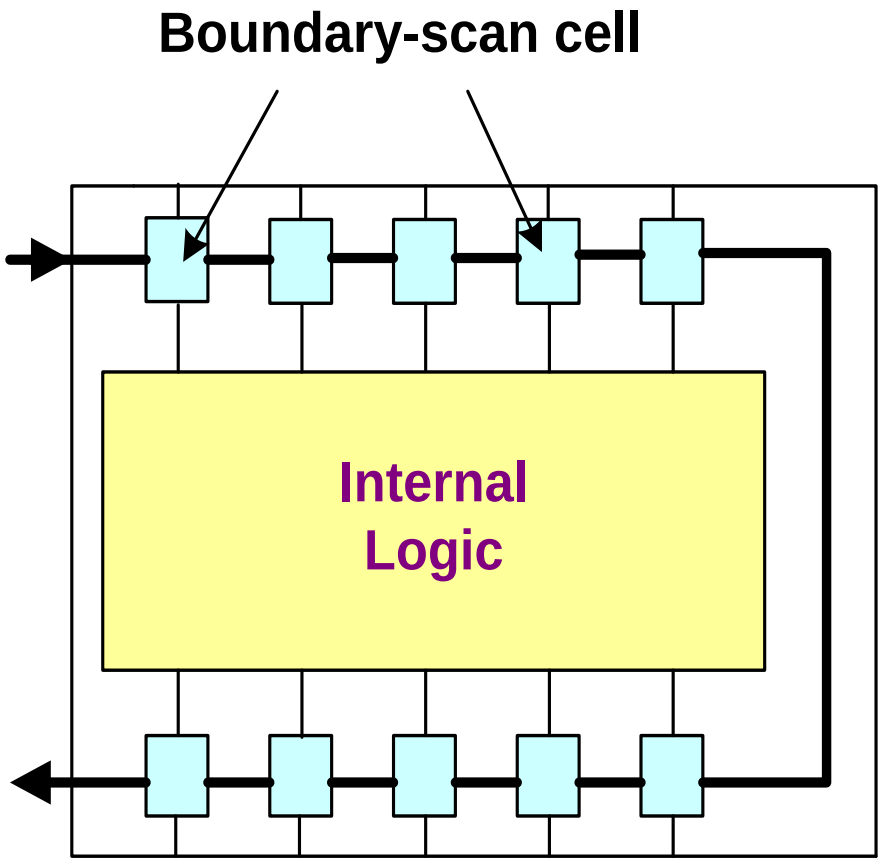


Example: Testing With Boundary Scan (cont.)

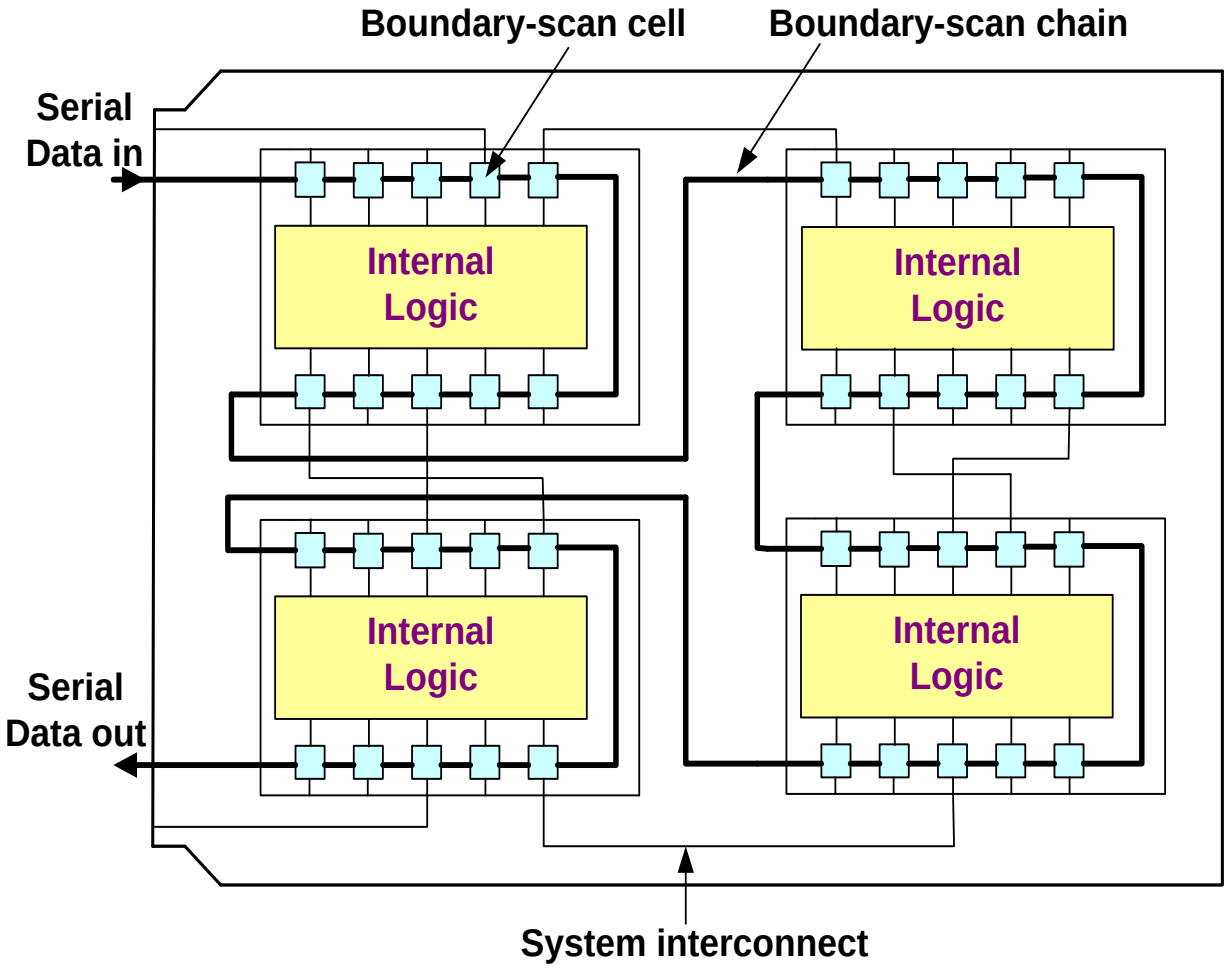
- Probing no longer necessary
  - E.g., an open at the simple receiver of net b1 tested by
    - Scanning in (x, x, x, x, 0, 1, x, 0, x, x, x, x) into BSCs
    - Applying the values scanned in BSCs to corresponding pins
    - Capturing the response from the input pins into corresponding BSCs
    - Scanning out the response



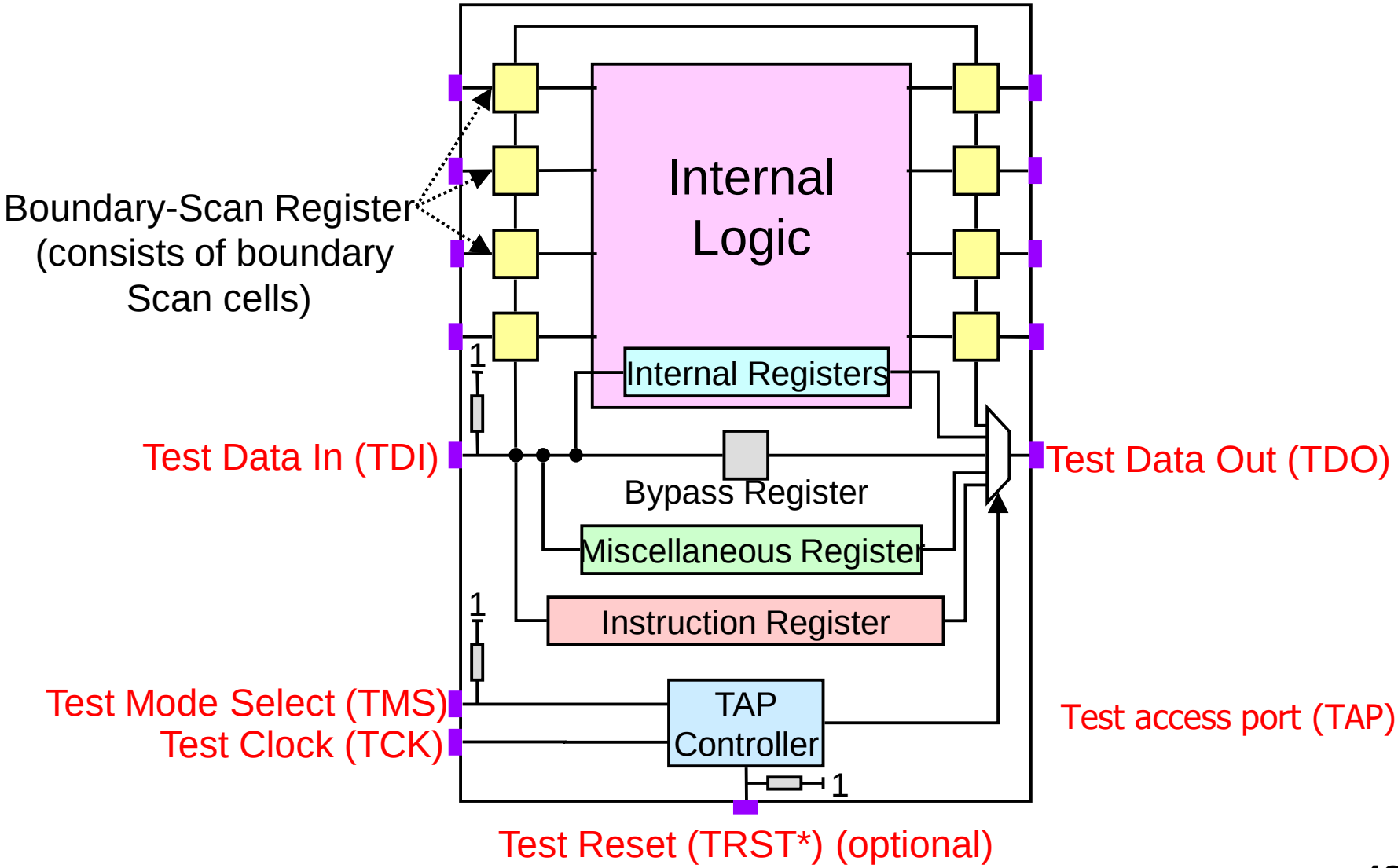
Basic Idea of Boundary Scan



A Board Containing 4 IC's with Boundary Scan



JTAG IEEE 1149.1 Boundary-Scan Architecture



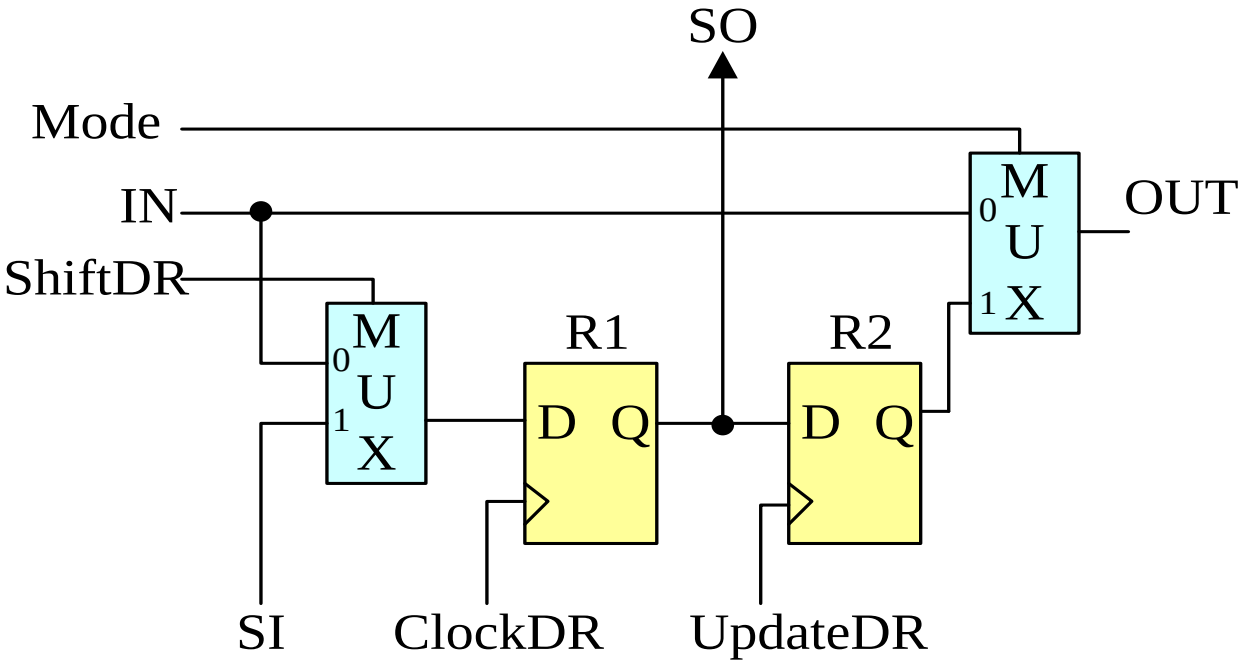
Hardware Components of 1149.1 (JTAG)

- Joint Test Action Group (1990)
- A test access port (TAP) consisting of :
  - —4 mandatory pins: Test data in (TDI), Test data out (TDO), Test mode select (TMS), Test clock (TCK), and
  - —1 optional pin: Test reset (TRST*)
- A test access port controller (TAPC)
- An instruction register (IR)
- Several test data registers
  - —A boundary scan register (BSR) consisting of boundary scan cells (BSCs)
  - —A bypass register (BR)
  - —Some optional registers (Device-ID register, design-specified registers such as scan registers, LFSRs for BIST, etc.)

Basic Operations

1. Instruction sent (serially) through TDI into instruction register.
2. Selected test circuitry configured to respond to the instruction.
3. Test pattern shifted into selected data register and applied to logic to be tested
4. Test response captured into some data register
5. Captured response shifted out; new test pattern shifted in simultaneously
6. Steps 3-5 repeated until all test patterns are applied.

A Typical Boundary-Scan Cell (BSC)

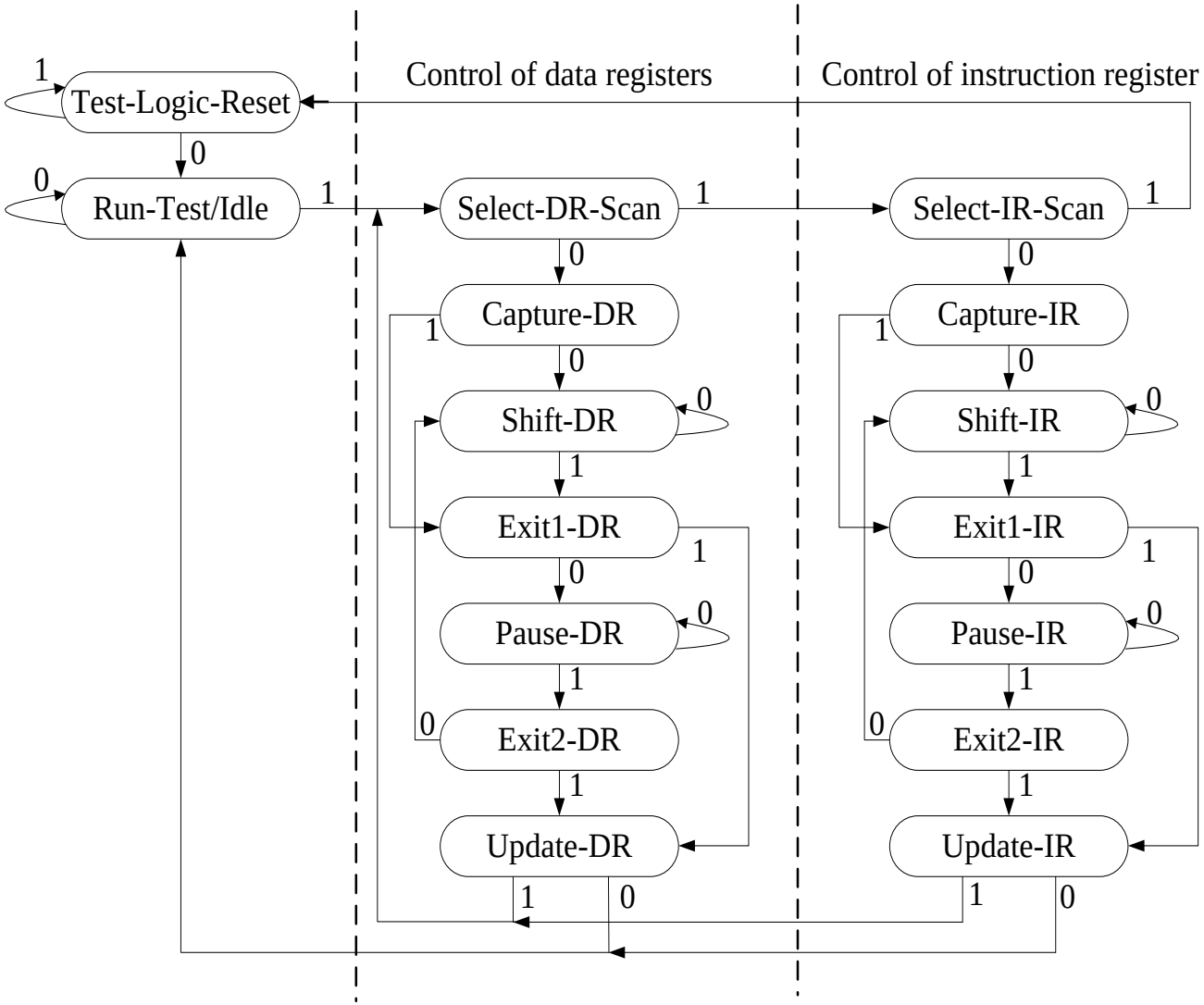


- Four main operation modes
  1. Normal: IN → OUT (Mode = 0)
  2. Shift: SI → ... → IN → OUT → ... → SO (ShiftDR = 1, ClockDR)
  3. Capture: IN → R1, OUT driven by IN or R2 (ShiftDR = 0, ClockDR)
  4. Update: R1 → OUT (Mode = 1, UpdateDR)

TAP Controller

- Interprets the control inputs applied serially at TMS input (sampled at each rising edge of TCK)
- A finite state machine with 16 states
- Input: TCK, TMS
- Output: 9 or 10 signals, i.e. ClockDR, UpdateDR, ShiftDR, ClockIR, UpdateIR, ShiftIR, Select, Enable, TCK and TRST* (optional).

State Diagram of TAP Controller



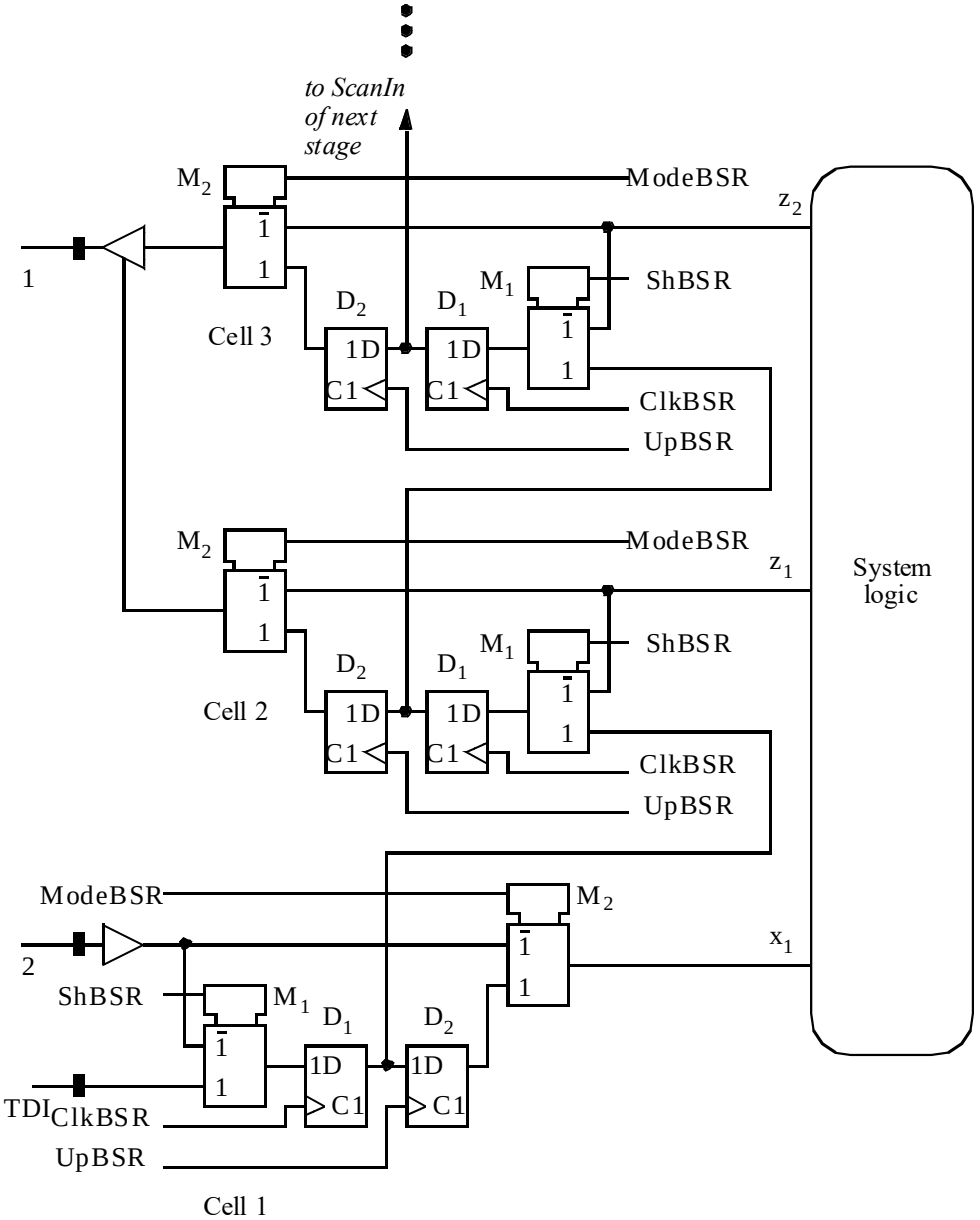
Main Functions of TAP Controller

- Providing control signals to
  - —Reset BS circuitry
  - —Load instructions into instruction register
  - —Perform test capture operation
  - —Perform test update operation
  - —Shift test data in and out

Examples of Mode of Operations

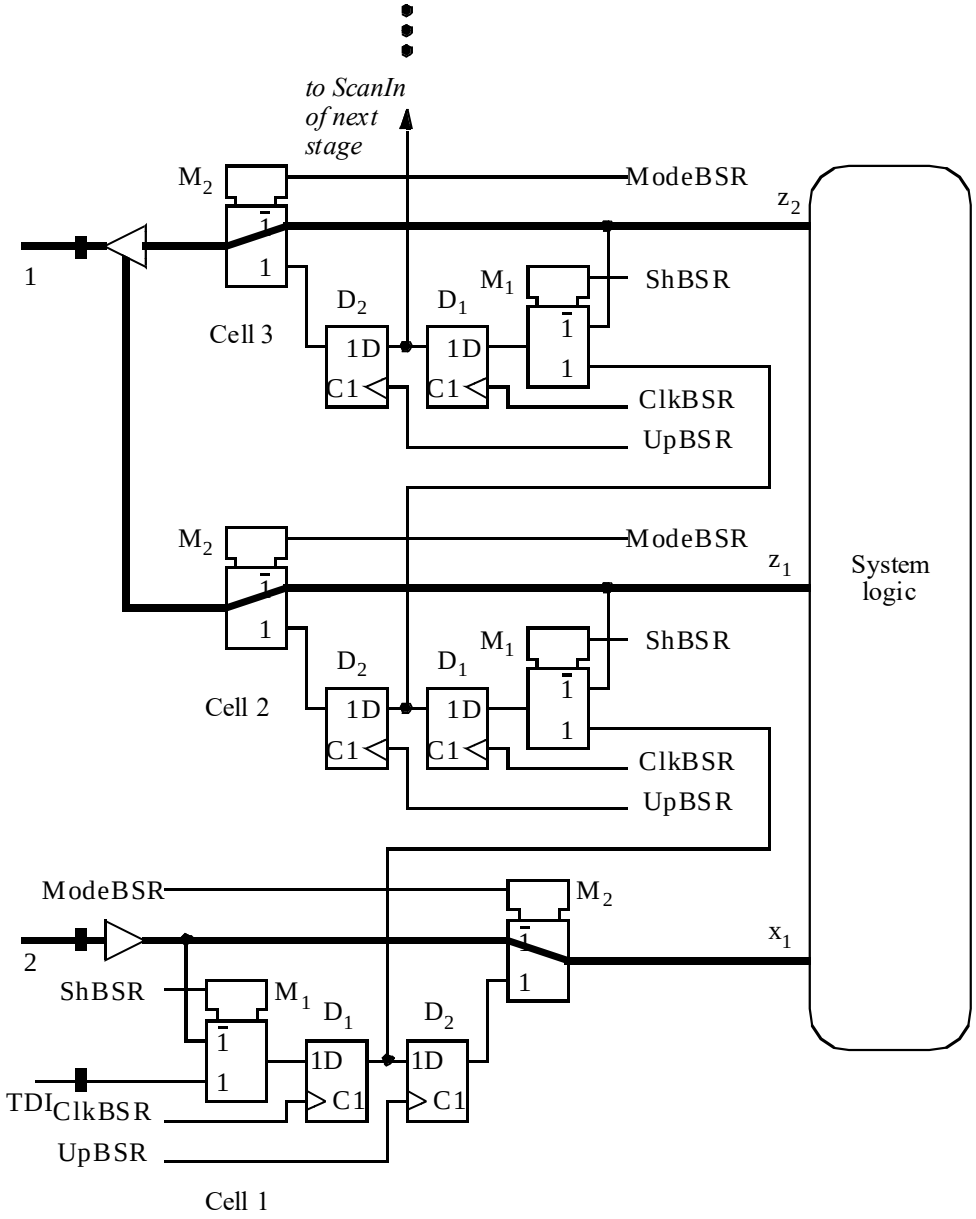
Example Configuration

- Input: x1
- Output: z1 and z2
- Number of boundary scan cells: 3



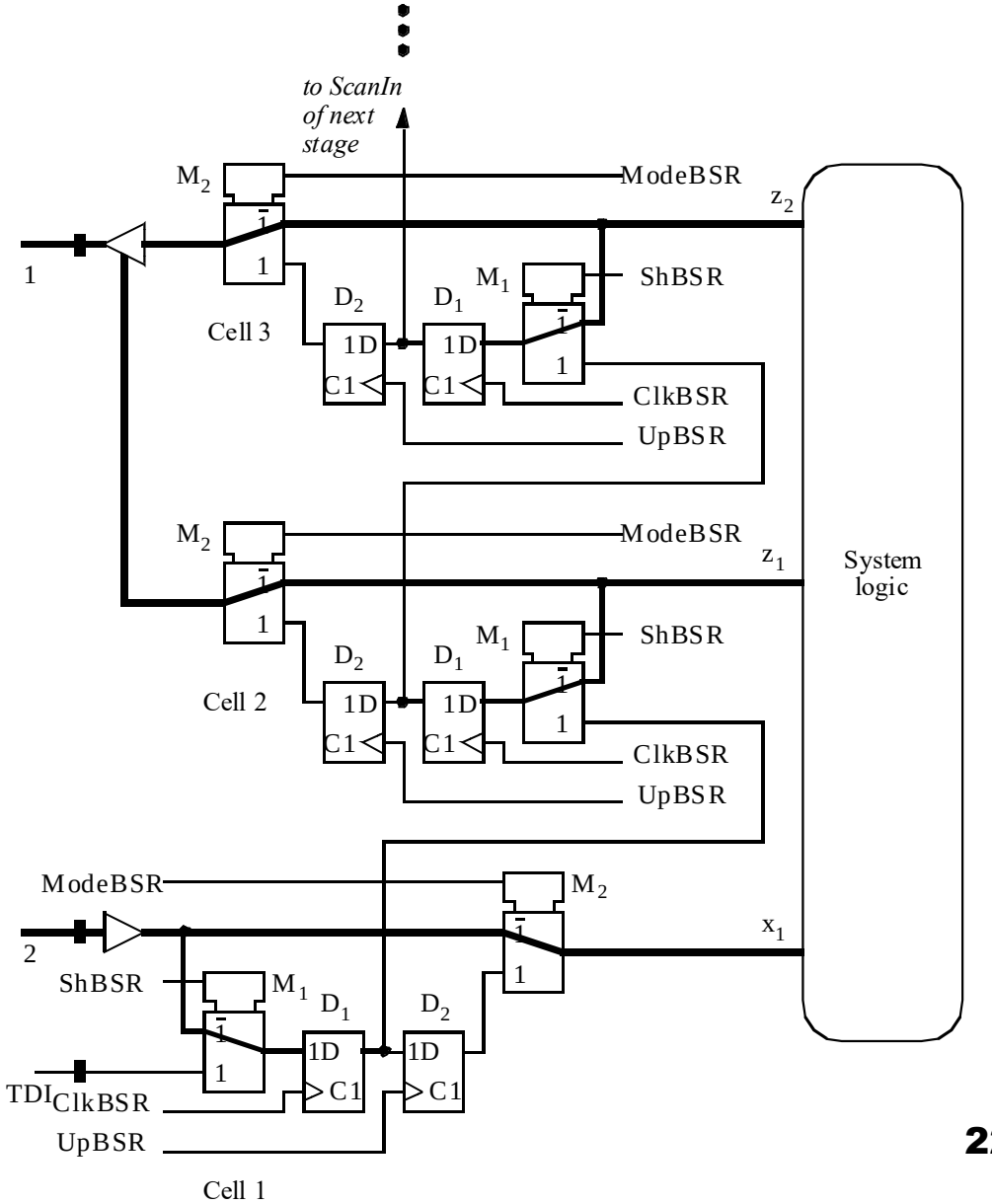
Normal Mode

- 0 applied at *ModeBSR*
  - The value at *NormalIn* appears at *NormalOut*
  - That is, the system logic operates in normal mode
- Gated clocks *ClkBSR* and *UpBSR* disabled
- The value at *ShBSR* does not affect operation



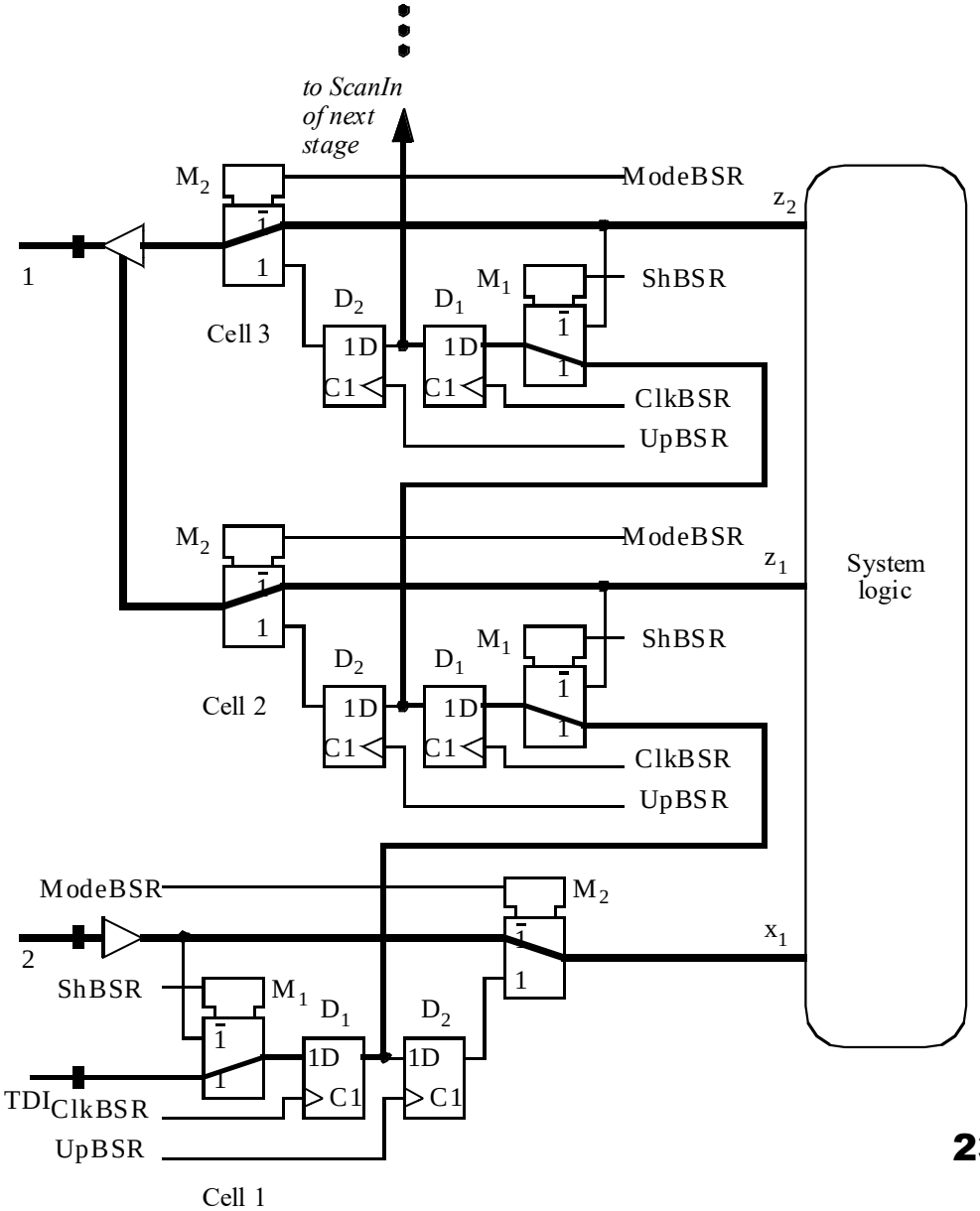
Normal-Sample Mode

- 0 applied at *ModeBSR*
  - The value at *NormalIn* appears at *NormalOut*
  - That is, the system logic operates in normal mode
- Gated clock *UpBSR* disabled
- 0 applied at *ShBSR*
- One clock pulse applied to *ClkBSR* to capture the values at *NormalIn* into the corresponding D_1 FFs
  - This clock synchronized with system clock that controls data at *NormalIn* inputs
  - This occurs only when TAP controller in *Capture-DR* state



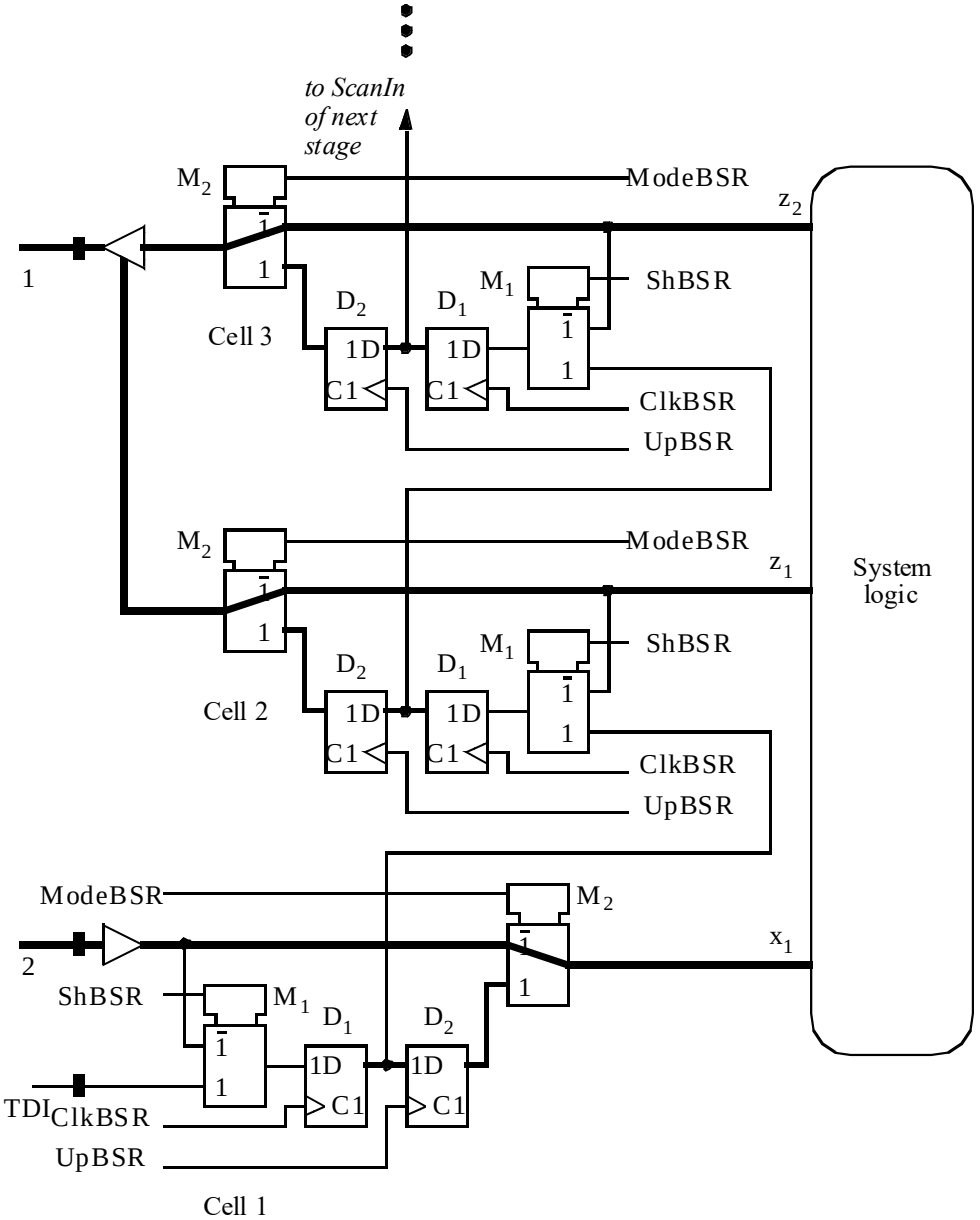
Normal-Scan Mode

- 0 applied at *ModeBSR*
  - The value at *NormalIn* appears at *NormalOut*
  - That is, the system logic operates in normal mode
- Gated clock *UpBSR* disabled
- 1 applied at *ShBSR*
- A sequence of clock pulses applied to *ClkBSR* to
  - Scan out any response previously captured in D_1 FFs
  - Scan in appropriate values into D_1 FFs from TDI
  - This occurs only when TAP controller in *Shift-DR* state



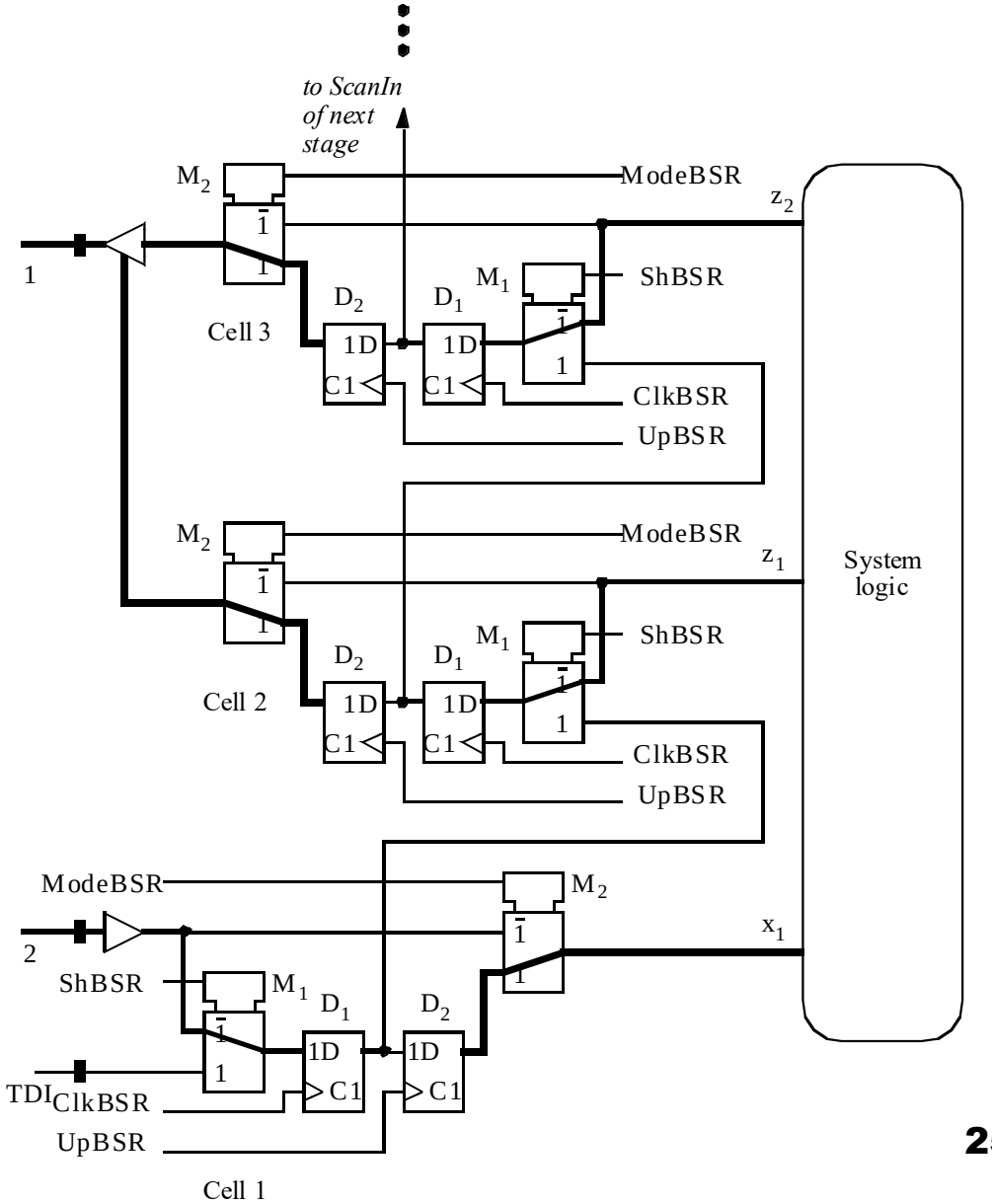
Normal-Preload Mode

- 0 applied at *ModeBSR*
  - The value at *NormalIn* appears at *NormalOut*
  - That is, the system logic operates in normal mode
- *ClkBSR* held inactive, *ShBSR* unimportant
- One clock pulse applied at *UpBSR*
  - To load the data currently held in D_1 FFs into corresponding D_2 FFs
  - This occurs only when TAP controller in *Update-DR* state



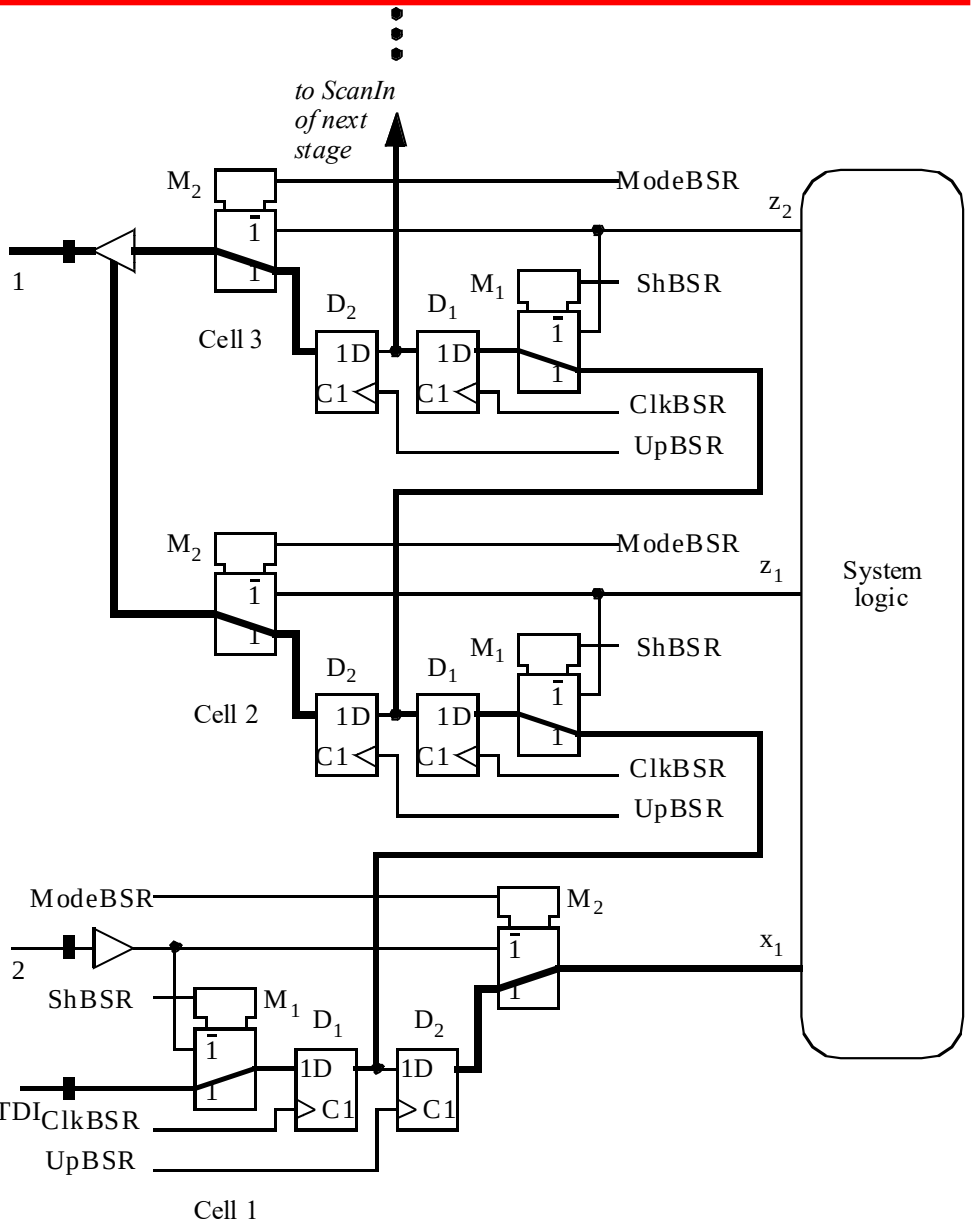
Test-Capture Mode

- 1 applied at *ModeBSR*
  - Values at D_2 FFs appear at *NormalOut*
  - That is, the system logic operates in test mode
- Gated clock *UpBSR* disabled
- 0 applied at *ShBSR*
- One clock pulse applied to *ClkBSR* to capture the values at *NormalIn* into the corresponding D_1 FFs
  - This occurs only when TAP controller in *Capture-DR* state



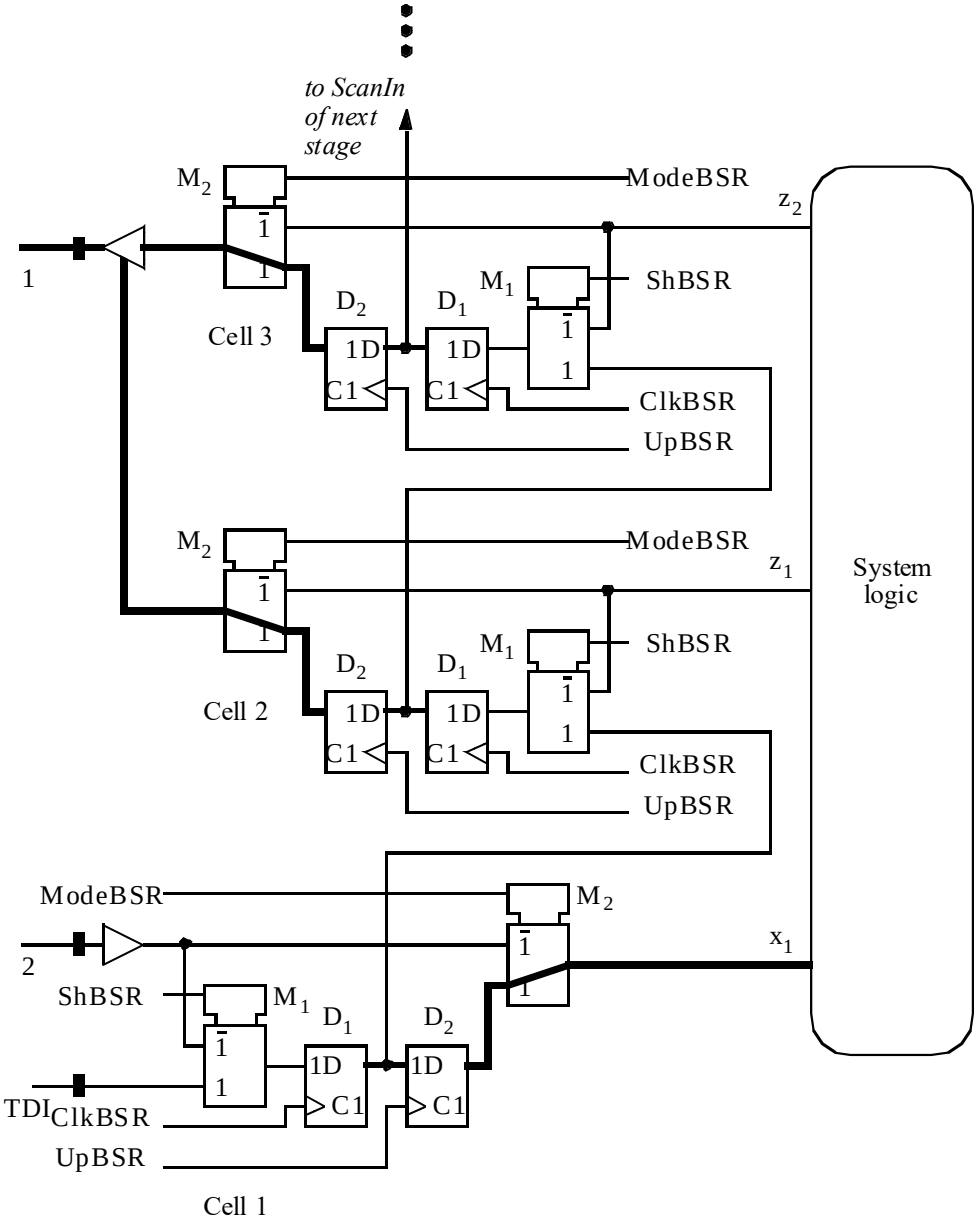
Test-Scan Mode

- 1 applied at *ModeBSR*
  - Values at D_2 FFs appear at *NormalOut*
  - That is, the system logic operates in test mode
- *UpBSR* held inactive
- 1 applied at *ShBSR*
- A sequence of clock pulses applied to *ClkBSR* to
  - Scan out any response previously captured in D_1 FFs
  - Scan in appropriate values into D_1 FFs from TDI
  - This occurs only when TAP controller in *Shift-DR* state



Test-Update Mode

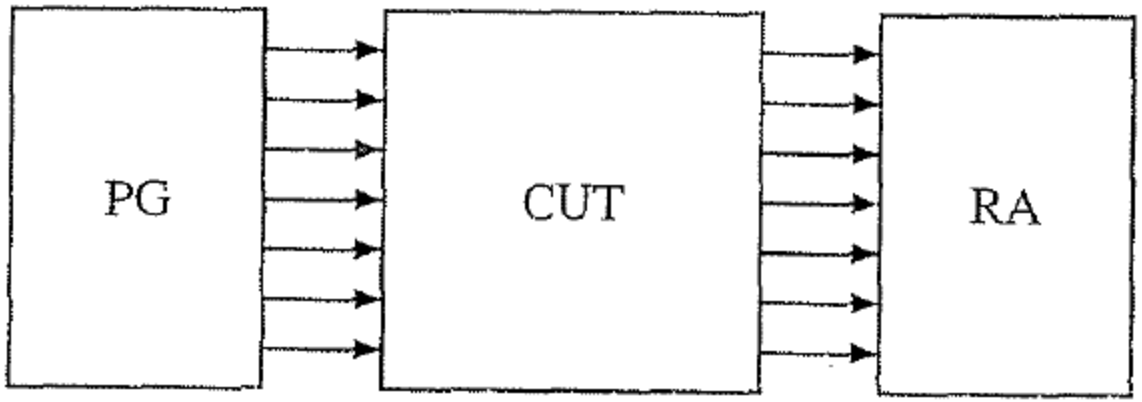
- 1 applied at *ModeBSR*
  - Values at D_2 FFs appear at *NormalOut*
  - That is, the system logic operates in test mode
- *ClkBSR* held inactive, *ShBSR* unimportant
- One clock pulse applied at *UpBSR*
  - To load the data currently held in D_1 FFs into corresponding D_2 FFs
  - This occurs only when TAP controller in *Update-DR* state



Built-In Self-Test (BIST)

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).

Definitions

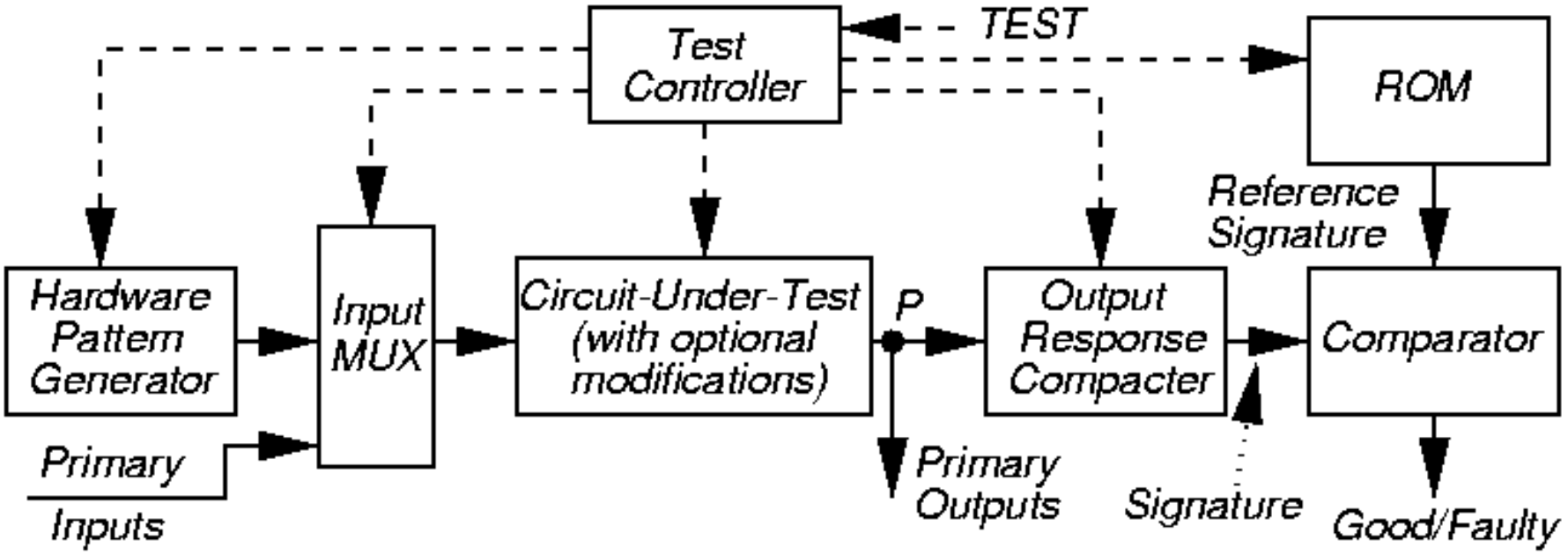
- **LFSR** – *Linear feedback shift register,* hardware that generates pseudo-random pattern sequence
- **BILBO** – *Built-in logic block observer*, extra hardware added to flip-flops so they can be reconfigured as an LFSR pattern generator or response compacter, a scan chain, or as flip-flops
- ***Concurrent* 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

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-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 Architecture

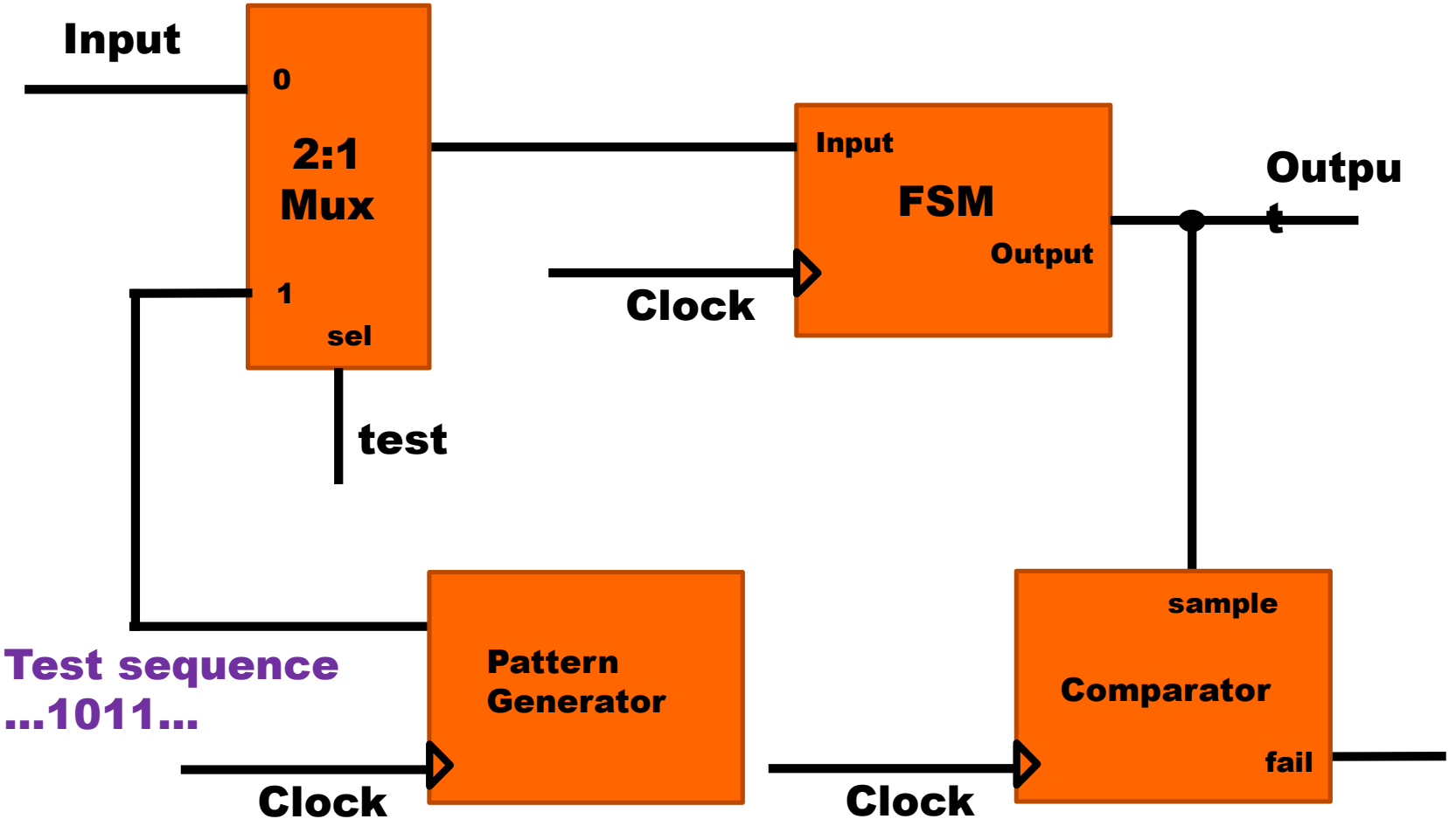
- BIST cannot test all wires and transistors
  - —From PI pins to Input MUX
  - —From POs to output pins



Built-in Self-test

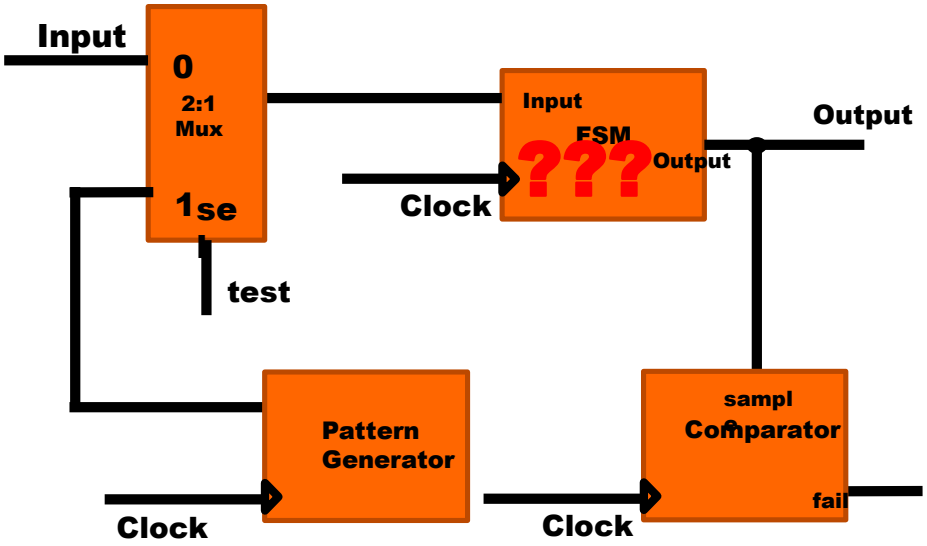
Normal Use of Circuit

Built-in Self-test



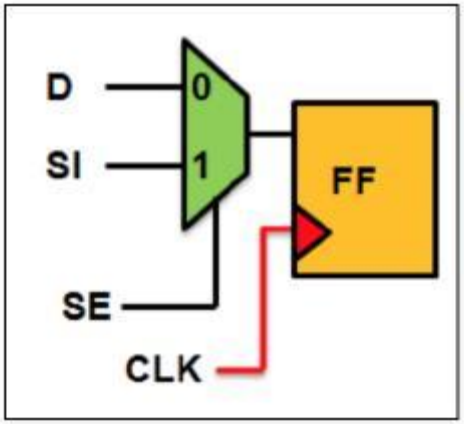
State Sequence Testing

Testing a sequence in a FSM is difficult
How to initialize the internal state?



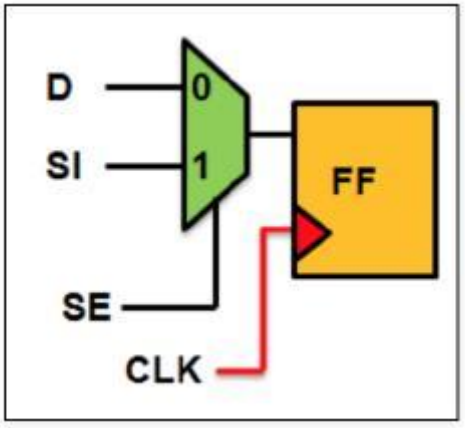
Scan Flip-Flops

Augment Flip-Flop with a 2:1 Mux



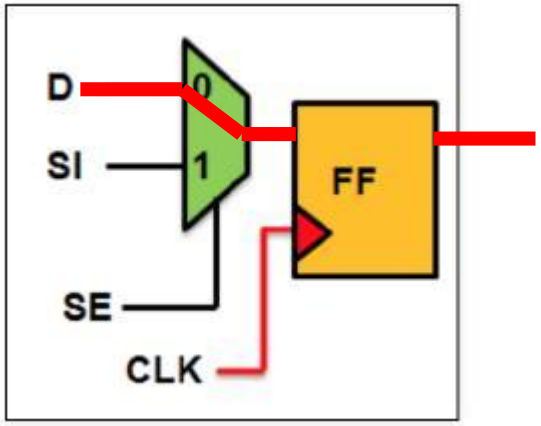
Scan Flip-Flops

Two modes: Normal and Test



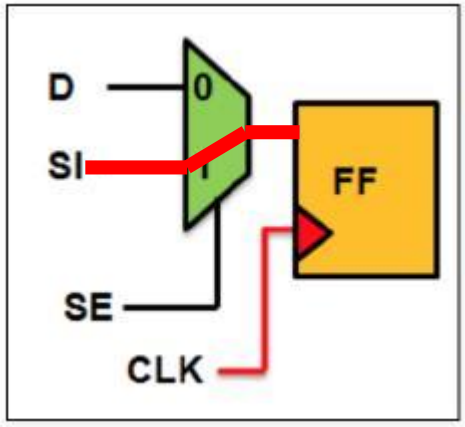
Scan Flip-Flops

Normal Mode
Select Mux (SE) set to zero



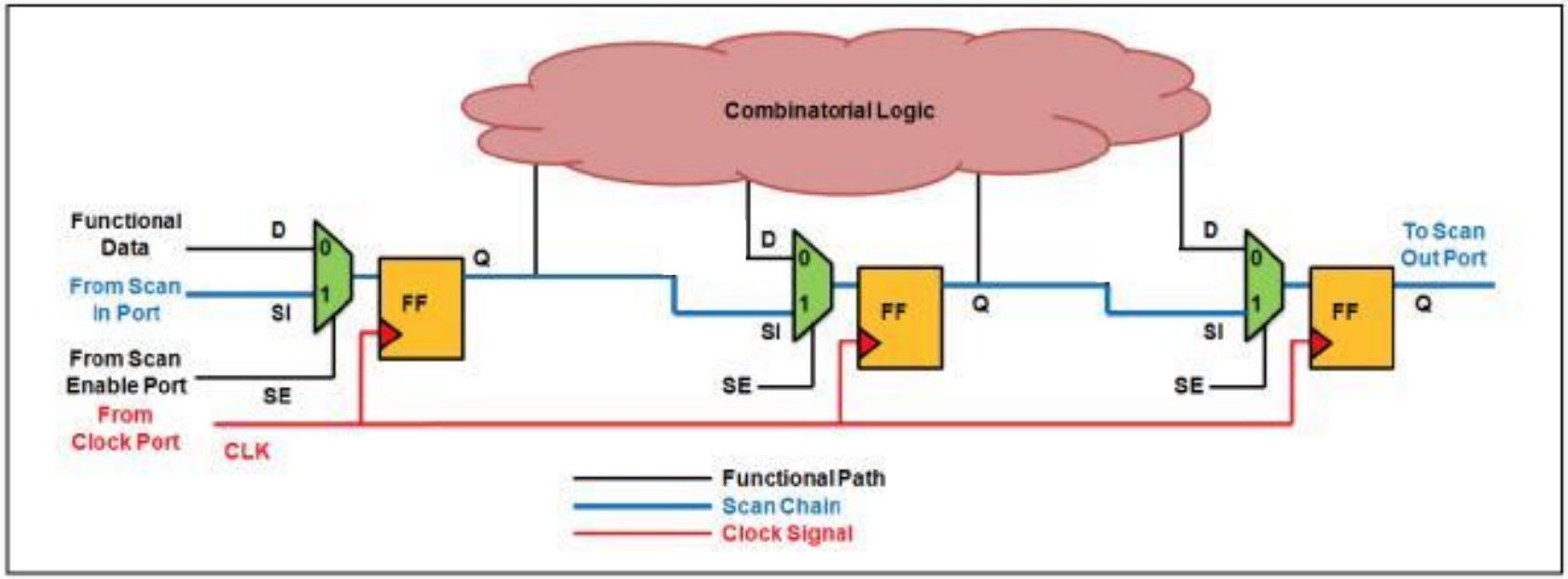
Scan Flip-Flops

Test or Scan In Mode
Select Mux (SE) set to one
Scan in path is enabled to load state of FF

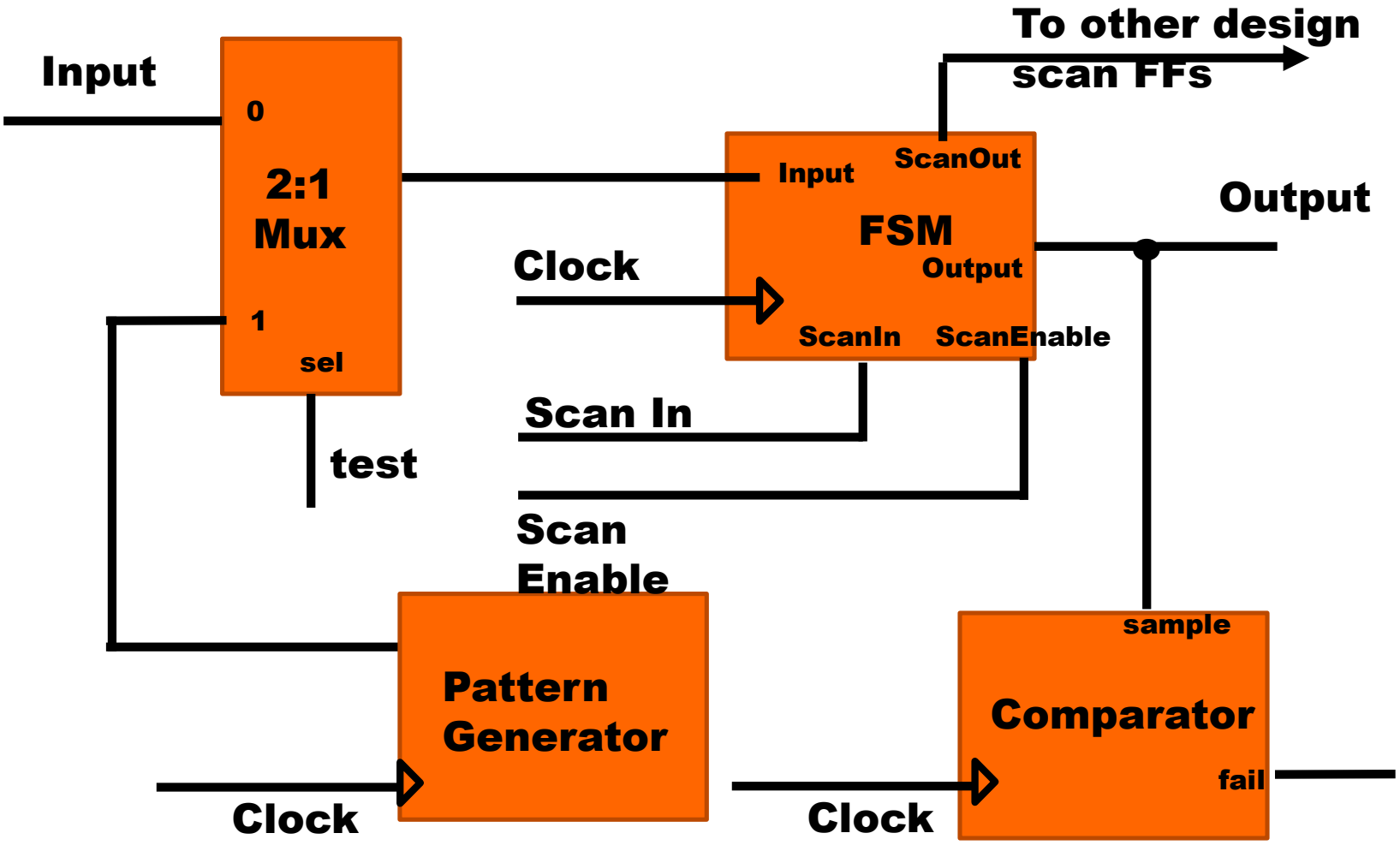


Scan Chain to Load States

Form chain by connecting all FF_{Q_i} to $FF_{SI_{i+1}}$
Select Mux (SE) set to one
N Flip-flops may be initialized in N clock cycles



Built-in Self Test with scan chain



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

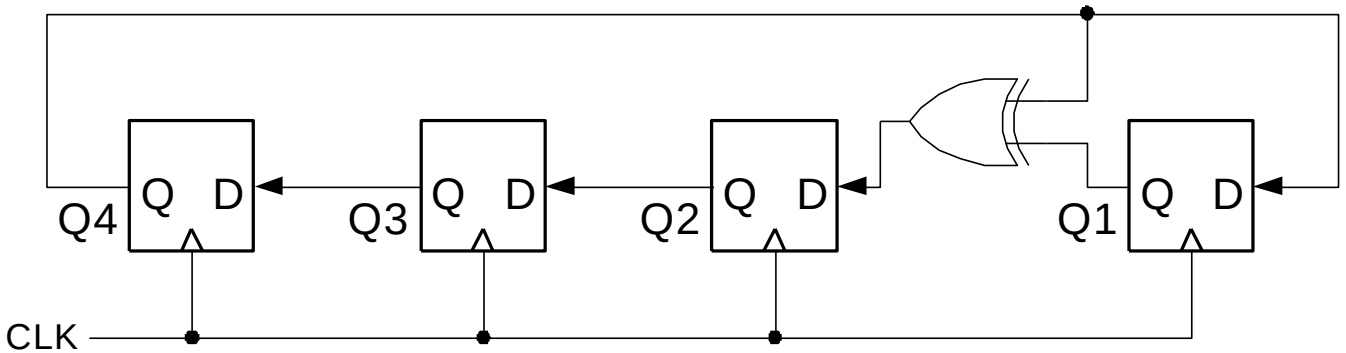
Linear Feedback Shift Register (LFSR)

- 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 (standard LFSR)
  2. External XOR (modular LFSR)
- Every LFSR is described by its feedback (characteristic) polynomial represented as: $\varphi(x) = \varphi_n x^n + \ldots + \varphi_1 x + \varphi_0$

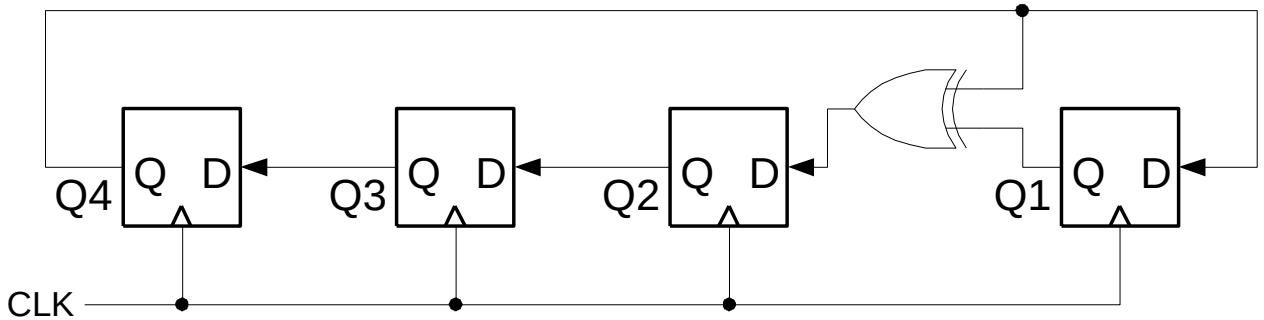
Linear Feedback Shift Register Background

Linear Feedback Shift Registers (LFSRs)

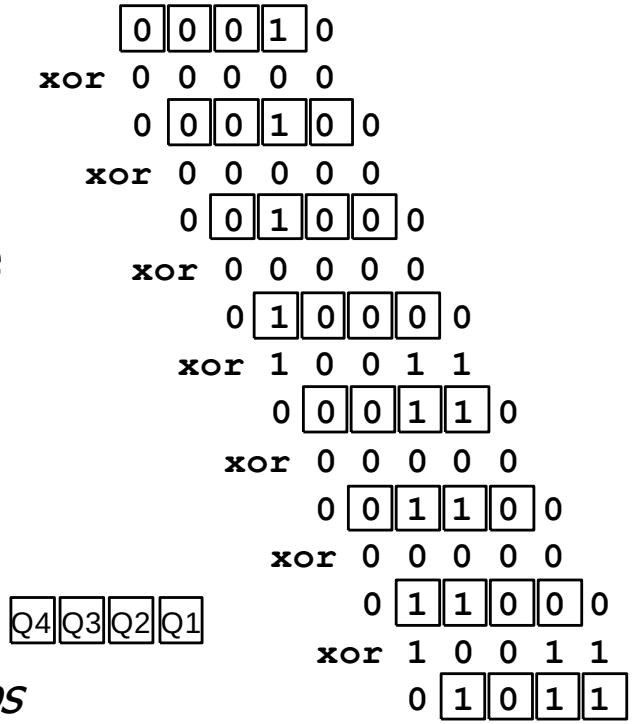
- These are n-bit counters exhibiting *pseudo-random* behavior.
- Built from simple shift-registers with a small number of xor gates.
- Used for:
  - random number generation
  - counters
  - error checking and correction
- Advantages:
  - very little hardware
  - high speed operation
- Example 4-bit LFSR:



4-bit LFSR



- Circuit counts through 2^4-1 different non-zero bit patterns.
- Left most bit determines shiftl or more complex operation
- Can build a similar circuit with any number of FFs, may need more xor gates.
- In general, with n flip-flops, 2^n-1 different non-zero bit patterns.
- (Intuitively, this is a counter that *wraps around* many times and in a strange way.)



0001
0010
0100
1000
0011
0110
1100
1011
0101
1010
0111
1110
1111
1101
1001
0001

Galois Fields - the theory behind LFSRs

- LFSR circuits performs multiplication on a *field*.
- A field is defined as a *set* with the following:
  - two operations defined on it:
    - "addition" and "multiplication"
  - closed under these operations
  - associative and distributive laws hold
  - additive and multiplicative identity elements
  - additive inverse for every element
  - multiplicative inverse for every non-zero element
- Example fields:
  - set of rational numbers
  - set of real numbers
  - set of integers is *not* a field (why?)
- Finite fields are called *Galois* fields.
- Example:
  - Binary numbers 0,1 with XOR as "addition" and AND as "multiplication".
  - Called GF(2).
  - 0+1 = 1
  - 1+1 = 0
  - 0-1 = ?
  - 1-1 = ?

Galois Fields - The theory behind LFSRs

- Consider *polynomials* whose coefficients come from GF(2).
- Each term of the form x^n is either present or absent.
- Examples: 0, 1, x, x^2, and $x^7 + x^6 + 1$

$$= 1 \cdot x^7 + 1 \cdot x^6 + 0 \cdot x^5 + 0 \cdot x^4 + 0 \cdot x^3 + 0 \cdot x^2 + 0 \cdot x^1 + 1 \cdot x^0$$

- With addition and multiplication these form a field:
- "Add": XOR each element individually with no carry:

$$\begin{array}{r} x^4 + x^3 + \quad + x + 1 \\ + \quad x^4 + \quad + x^2 + x \\ \hline x^3 + x^2 \quad + 1 \end{array}$$

- "Multiply": multiplying by x^n is like shifting to the left.

$$\begin{array}{r} x^2 + x + 1 \\ \times \quad x + 1 \\ \hline x^2 + x + 1 \\ x^3 + x^2 + x \\ \hline x^3 \quad + 1 \end{array}$$

So what about division (mod)

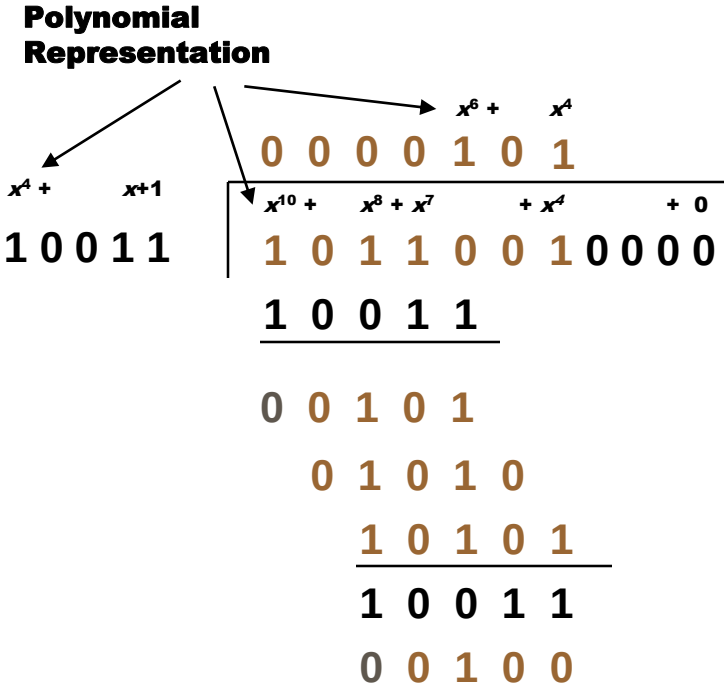
$$\frac{x^4 + x^2}{x} = x^3 + x \text{ with remainder } 0$$

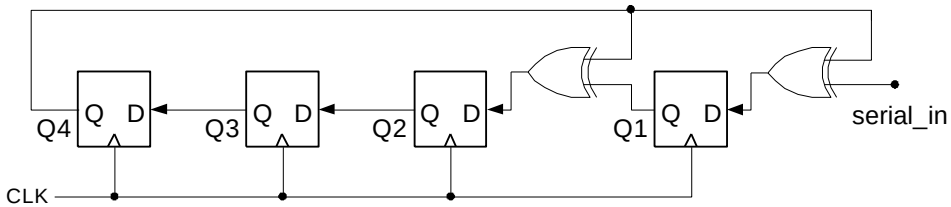
$$\frac{x^4 + x^2 + 1}{X + 1} = x^3 + x^2 \text{ with remainder } 1$$

$$
\begin{array}{rl}
 & x^3 + x^2 + 0x + 0 \\
X + 1 \;\big| & x^4 + 0x^3 + x^2 + 0x + 1 \\
 & x^4 + x^3 \\
\hline
 & \quad x^3 + x^2 \\
 & \quad x^3 + x^2 \\
\hline
 & \quad\quad 0x^2 + 0x \\
 & \quad\quad\quad 0x + 1
\end{array}
$$

Remainder 1

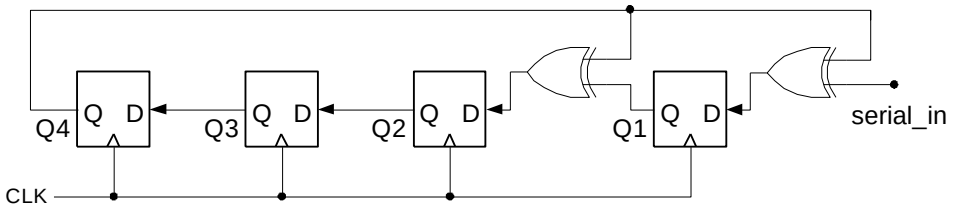
Polynomial division





- When MSB is zero, just shift left, bringing in next bit
- When MSB is 1, XOR with divisor and shiftl

CRC encoding



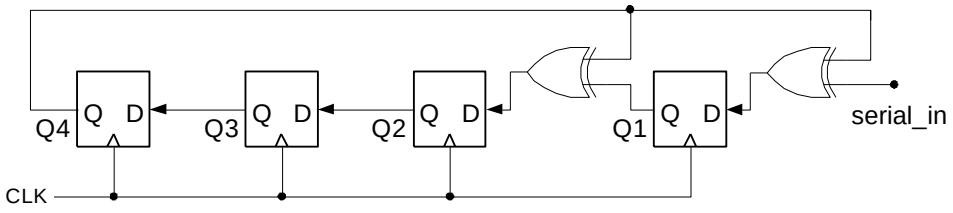
1 0 1 1 0 0 1 0 0 0 0

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

Message sent:

1 0 1 1 0 0 1 1 0 1 0

CRC decoding



1 0 1 1 0 0 1 1 0 1 0

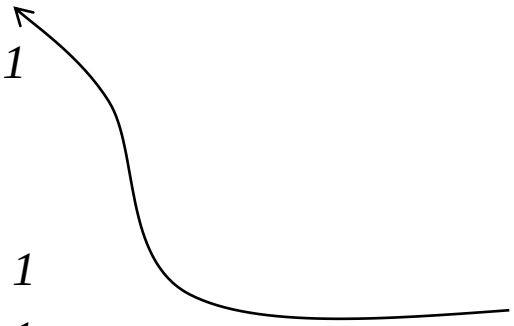
0	0	0	0	
0	0	0	1	0 1 1 0 0 1 1 0 1 0
0	0	1	0	1 1 0 0 1 1 0 1 0
0	1	0	1	1 0 0 1 1 0 1 0
1	0	1	1	0 0 1 1 0 1 0
0	1	0	1	0 1 1 0 1 0
1	0	1	0	1 1 0 1 0
0	1	1	0	1 0 1 0
1	1	0	1	0 1 0
1	0	0	1	1 0
0	0	0	0	0
0	0	0	0	

Galois Fields - The theory behind LFSRs

- These polynomials form a *Galois (finite)* field if we take the results of this multiplication modulo a prime polynomial $p(x)$.
  - A prime polynomial is one that cannot be written as the product of two non-trivial polynomials $q(x)r(x)$
  - Perform modulo operation by subtracting a (polynomial) multiple of $p(x)$ from the result. If the multiple is 1, this corresponds to XOR-ing the result with $p(x)$.
- For any degree, there exists at least one prime polynomial.
- With it we can form $GF(2^n)$
- Additionally, …
- Every Galois field has a primitive element, α, such that all non-zero elements of the field can be expressed as a power of α. By raising α to powers (modulo $p(x)$), all non-zero field elements can be formed.
- Certain choices of $p(x)$ make the simple polynomial x the primitive element. These polynomials are called *primitive*, and one exists for every degree.
- For example, $x^4 + x + 1$ is primitive. So $\alpha = x$ is a primitive element and successive powers of α will generate all non-zero elements of GF(16). *Example on next slide.*

Galois Fields – Primitives

$$\alpha^0 = 1$$
$$\alpha^1 = x$$
$$\alpha^2 = x^2$$
$$\alpha^3 = x^3$$
$$\alpha^4 = x + 1$$
$$\alpha^5 = x^2 + x$$
$$\alpha^6 = x^3 + x^2$$
$$\alpha^7 = x^3 + x + 1$$
$$\alpha^8 = x^2 + 1$$
$$\alpha^9 = x^3 + x$$
$$\alpha^{10} = x^2 + x + 1$$
$$\alpha^{11} = x^3 + x^2 + x$$
$$\alpha^{12} = x^3 + x^2 + x + 1$$
$$\alpha^{13} = x^3 + x^2 + 1$$
$$\alpha^{14} = x^3 + 1$$
$$\alpha^{15} = 1$$



- Note this pattern of coefficients matches the bits from our 4-bit LFSR example.

$$\alpha^4 = x^4 \bmod x^4 + x + 1$$
$$= x^4 \text{ xor } x^4 + x + 1$$
$$= x + 1$$

- In general finding primitive polynomials is difficult. Most people just look them up in a table, such as:

Primitive Polynomials

$x^2 + x + 1$
$x^3 + x + 1$
$x^4 + x + 1$
$x^5 + x^2 + 1$
$x^6 + x + 1$
$x^7 + x^3 + 1$
$x^8 + x^4 + x^3 + x^2 + 1$
$x^9 + x^4 + 1$
$x^{10} + x^3 + 1$
$x^{11} + x^2 + 1$

$x^{12} + x^6 + x^4 + x + 1$
$x^{13} + x^4 + x^3 + x + 1$
$x^{14} + x^{10} + x^6 + x + 1$
$x^{15} + x + 1$
$x^{16} + x^{12} + x^3 + x + 1$
$x^{17} + x^3 + 1$
$x^{18} + x^7 + 1$
$x^{19} + x^5 + x^2 + x + 1$
$x^{20} + x^3 + 1$
$x^{21} + x^2 + 1$

$x^{22} + x + 1$
$x^{23} + x^5 + 1$
$x^{24} + x^7 + x^2 + x + 1$
$x^{25} + x^3 + 1$
$x^{26} + x^6 + x^2 + x + 1$
$x^{27} + x^5 + x^2 + x + 1$
$x^{28} + x^3 + 1$
$x^{29} + x + 1$
$x^{30} + x^6 + x^4 + x + 1$
$x^{31} + x^3 + 1$
$x^{32} + x^7 + x^6 + x^2 + 1$

Galois Field — Hardware

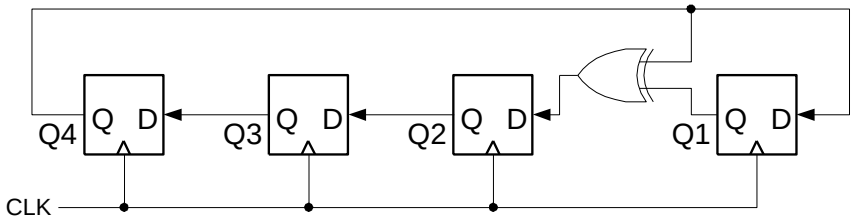
Multiplication by x $\Leftrightarrow$ shift left

Taking the result mod $p(x)$ $\Leftrightarrow$ XOR-ing with the coefficients of $p(x)$ when the most significant coefficient is 1.

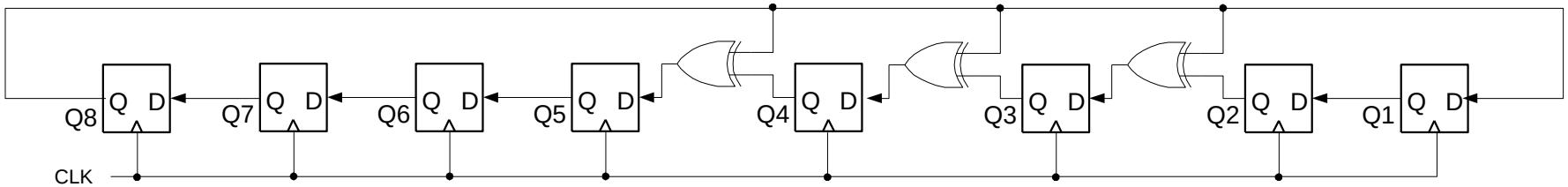
Obtaining all 2^n-1 non-zero elements by evaluating x^k for $k = 1, \ldots, 2^n-1$ $\Leftrightarrow$ Shifting and XOR-ing 2^n-1 times.

Building an LFSR from a Primitive Poly

- For *k-bit* LFSR number the flip-flops with FF1 on the right.
- The feedback path comes from the Q output of the leftmost FF.
- Find the primitive polynomial of the form $x^k + \ldots + 1$.
- The $x^0 = 1$ term corresponds to connecting the feedback directly to the D input of FF 1.
- Each term of the form x^n corresponds to connecting an xor between FF n and $n+1$.
- 4-bit example, uses $x^4 + x + 1$
  - $x^4 \Leftrightarrow$ FF4's Q output
  - $x \Leftrightarrow$ xor between FF1 and FF2
  - $1 \Leftrightarrow$ FF1's D input



- To build an 8-bit LFSR, use the primitive polynomial $x^8 + x^4 + x^3 + x^2 + 1$ and connect xors between FF2 and FF3, FF3 and FF4, and FF4 and FF5.



Generating Polynomials

- CRC-16: $G(x) = x^{16} + x^{15} + x^2 + 1$
  - detects single and double bit errors
  - All errors with an odd number of bits
  - Burst errors of length 16 or less
  - Most errors for longer bursts
- CRC-32: $G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$
  - Used in ethernet
  - Also 32 bits of 1 added on front of the message
    - Initialize the LFSR to all 1s

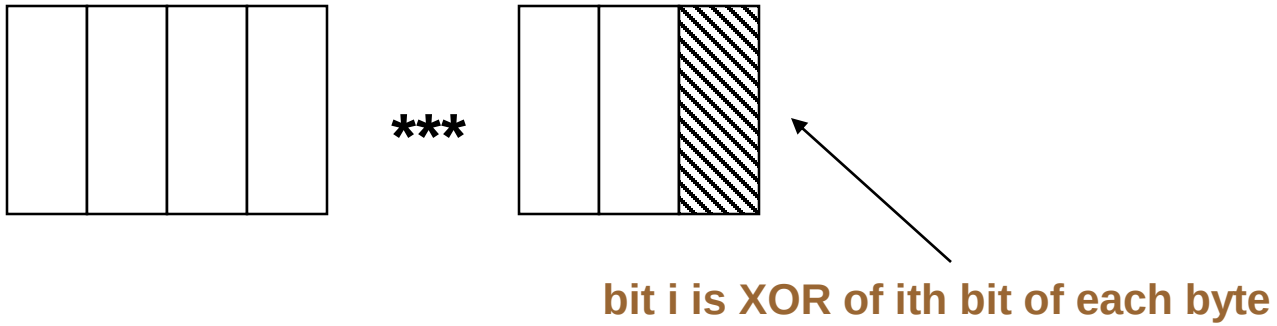
Applications of LFSRs

- Performance:
  - In general, xors are only ever 2-input and never connect in series.
  - Therefore the minimum clock period for these circuits is:

    $T > T_{2\text{-input-xor}}$ + clock overhead
  - Very little latency, and independent of n!
- This can be used as a fast counter, if the particular sequence of count values is not important.
  - Example: micro-code micro-pc
- Can be used as a random number generator.
  - Sequence is a pseudo-random sequence:
    - numbers appear in a random sequence
    - repeats every 2^n-1 patterns
  - Random numbers useful in:
    - computer graphics
    - cryptography
    - automatic testing
- Used for error detection and correction
    - CRC (cyclic redundancy codes)
    - ethernet uses them

Concept: Redundant Check

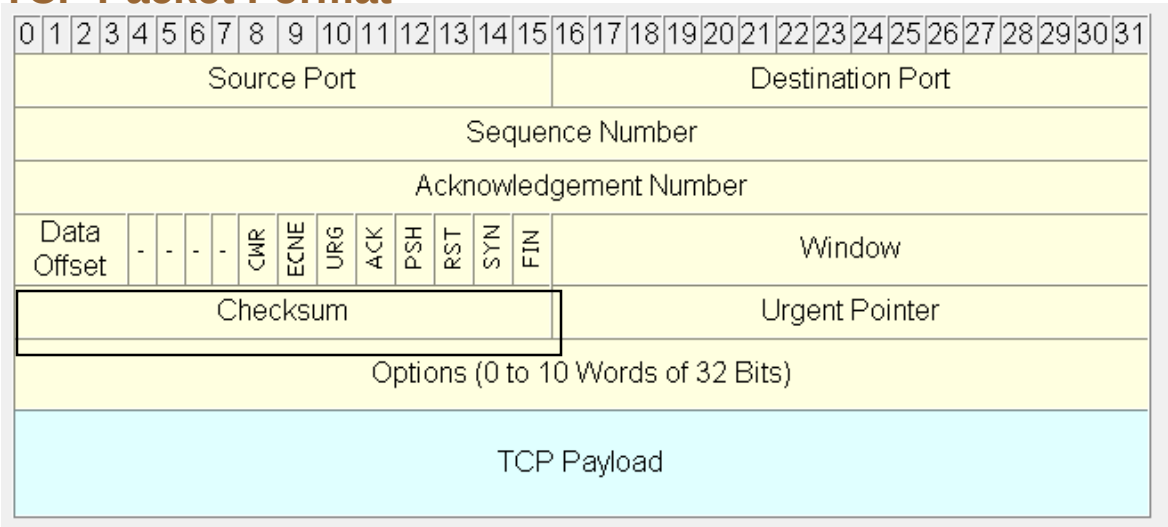
- Send a message M and a "check" word C
- Simple function on <M,C> to determine if both received correctly (with high probability)
- Example: XOR all the bytes in M and append the "checksum" byte, C, at the end
  - Receiver XORs <M,C>
  - What should result be?
  - What errors are caught?



Example: TCP Checksum

7	Application (HTTP,FTP, DNS)
4	Transport (TCP, UDP)
3	Network (IP)
2	Data Link (Ethernet, 802.11b)
1	Physical

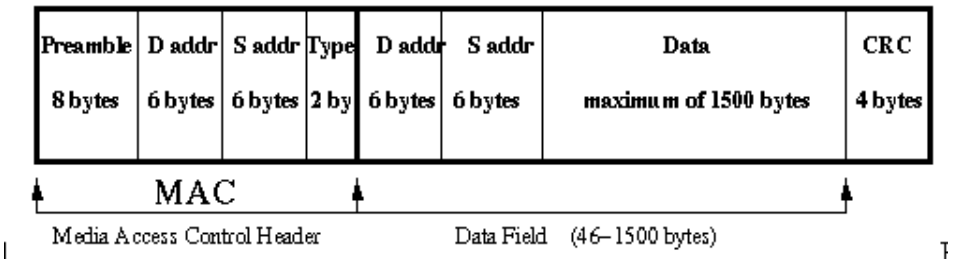
TCP Packet Format

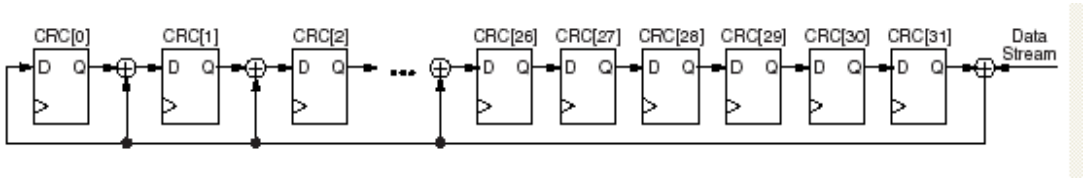


- **TCP Checksum a 16-bit checksum, consisting of the one's complement of the one's complement sum of the contents of the TCP segment header and data, is computed by a sender, and included in a segment transmission. (note end-around carry)**
- **Summing all the words, including the checksum word, should yield zero**

Example: Ethernet CRC-32

7	Application (HTTP,FTP, DNS)
4	Transport (TCP, UDP)
3	Network (IP)
2	Data Link (Ethernet, 802.11b)
1	Physical





CRC concept

- I have a msg polynomial M(x) of degree m
- We both have a generator poly G(x) of degree m
- Let r(x) = remainder of M(x) x^n / G(x)
  - M(x) x^n = G(x)p(x) + r(x)
  - r(x) is of degree n
- What is (M(x) x^n – r(x)) / G(x) ?

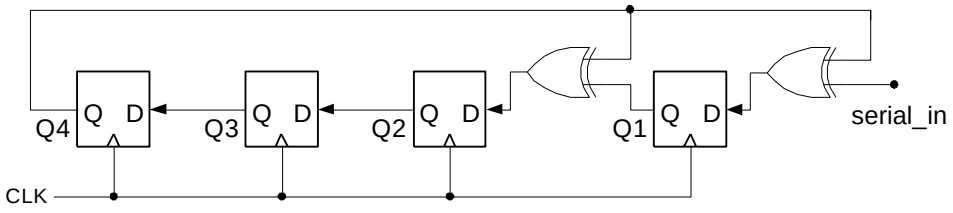
n bits of zero at the end

- So I send you M(x) x^n – r(x)
  - m+n degree polynomial
  - You divide by G(x) to check
  - M(x) is just the m most signficant coefficients, r(x) the lower m

tack on n bits of remainder
Instead of the zeros

- n-bit Message is viewed as coefficients of n-degree polynomial over binary numbers

CRC encoding



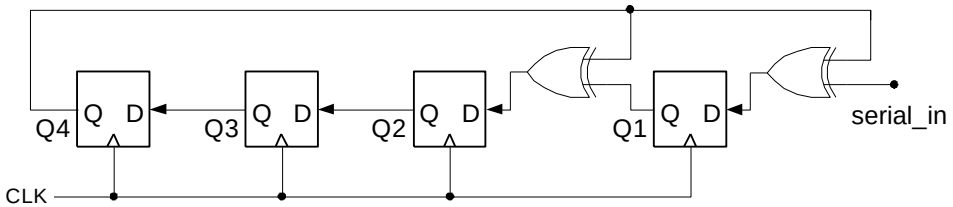
1 0 1 1 0 0 1 0 0 0 0

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

Message sent:

1 0 1 1 0 0 1 1 0 1 0

CRC decoding



1 0 1 1 0 0 1 1 0 1 0

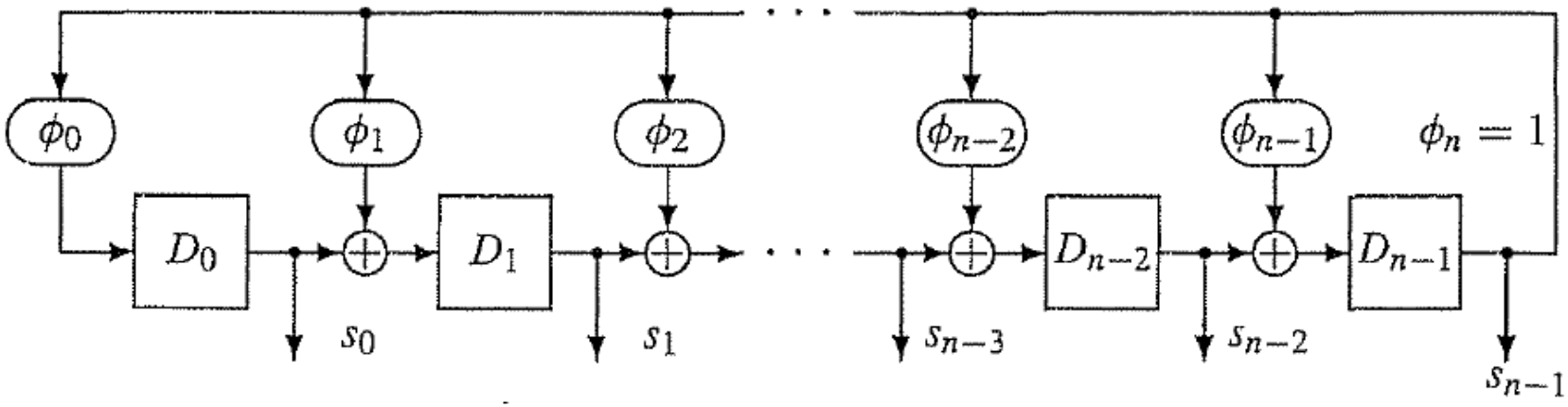
0	0	0	0	
0	0	0	1	0 1 1 0 0 1 1 0 1 0
0	0	1	0	1 1 0 0 1 1 0 1 0
0	1	0	1	1 0 0 1 1 0 1 0
1	0	1	1	0 0 1 1 0 1 0
0	1	0	1	0 1 1 0 1 0
1	0	1	0	1 1 0 1 0
0	1	1	0	1 0 1 0
1	1	0	1	0 1 0
1	0	0	1	1 0
0	0	0	0	0
0	0	0	0	

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:

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

Note the order of φ_i's

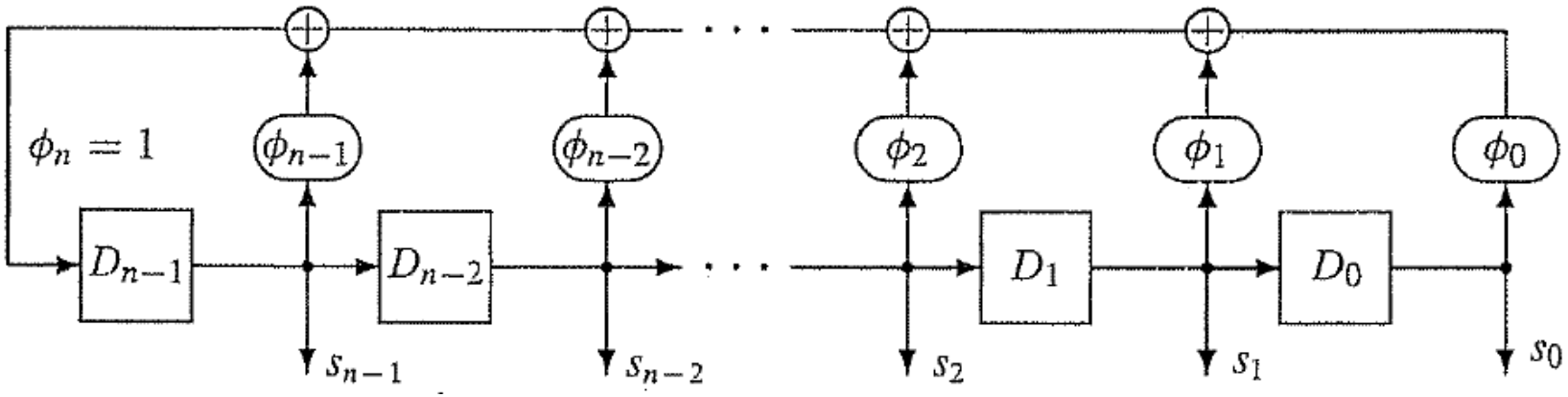


LFSR with External XOR

- 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:

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

Note the order of φ_i's



LFSR Example – Internal XOR

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

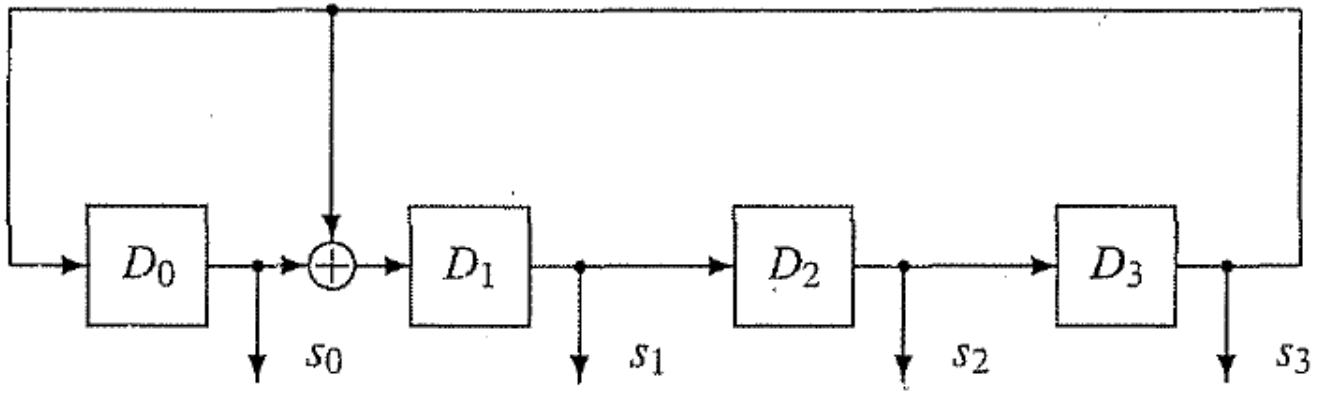
$$\begin{bmatrix} s_0(t+1) \\ s_1(t+1) \\ s_2(t+1) \\ s_3(t+1) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} s_0(t) \\ s_1(t) \\ s_2(t) \\ s_3(t) \end{bmatrix}$$

- The sequence:

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

Initial Value

Cyclic Repetition



LFSR Example – External XOR

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

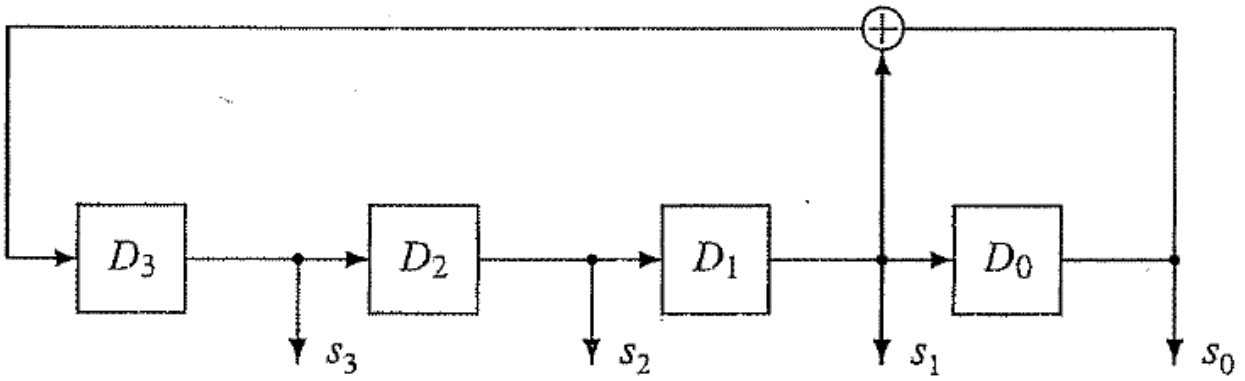
$$\begin{bmatrix} s_0(t+1) \\ s_1(t+1) \\ s_2(t+1) \\ s_3(t+1) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} s_0(t) \\ s_1(t) \\ s_2(t) \\ s_3(t) \end{bmatrix}$$

- The sequence:

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

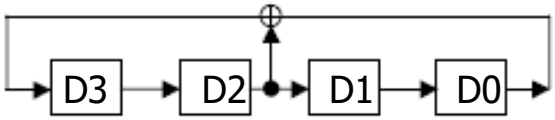
Initial Value

Cyclic Repetition

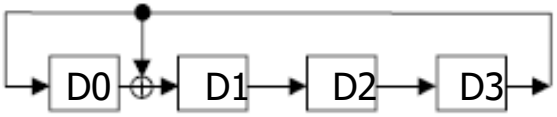


Primitive vs. Non-Primitive Polynomial

- (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
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

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

Response Compaction

Motivation for Test Data Compaction

- Huge amount of data in CUT response to LFSR patterns – example:
  - Generate 5 million random patterns
  - CUT has 200 outputs
  - Leads to: 5 million x 200 = 1 billion bits response
- Uneconomical to store and check all of these responses on chip
- Responses must be compacted

Definitions

- ***Aliasing*** – Due to information loss, signatures of good and some bad machines match
- ***Compaction*** – Drastically reduce # bits in original circuit response – lose information
- ***Compression*** – Reduce # bits in original circuit response – no information loss – fully invertible (can get back original response)
- ***Signature analysis*** – Compact good machine response into *good machine signature*. Actual signature generated during testing, and compared with good machine signature
- ***Transition Count Response Compaction*** – Count # transitions from 0→1 and 1→0 as a signature

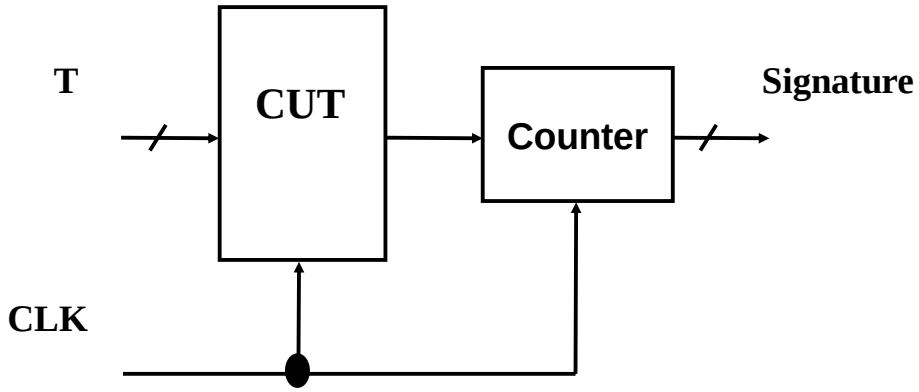
Response Compaction

Response Analysis Mechanisms

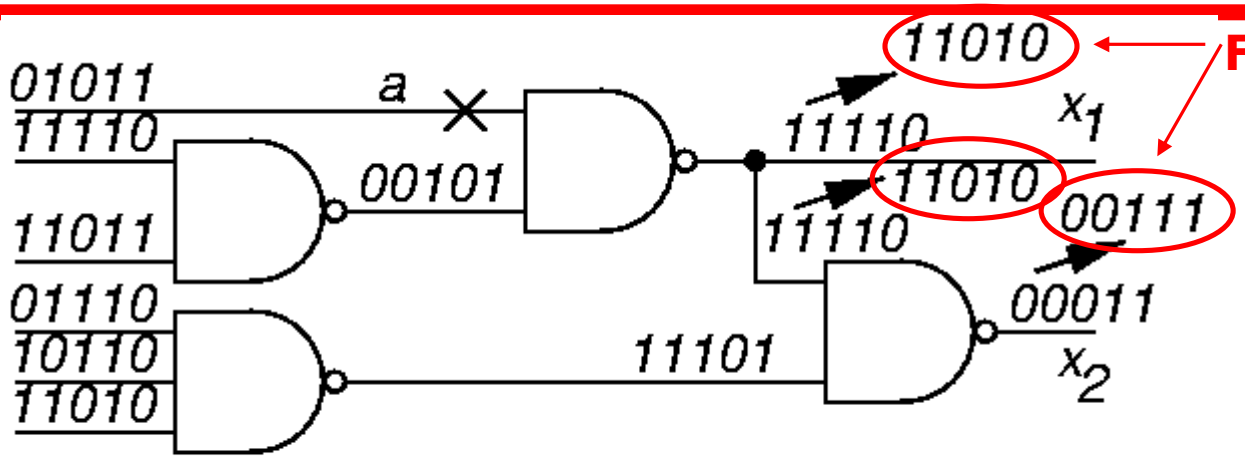
- Ones count testing
- Transition count testing
- Signature analysis
  - —Serial
  - —parallel

Ones Counting

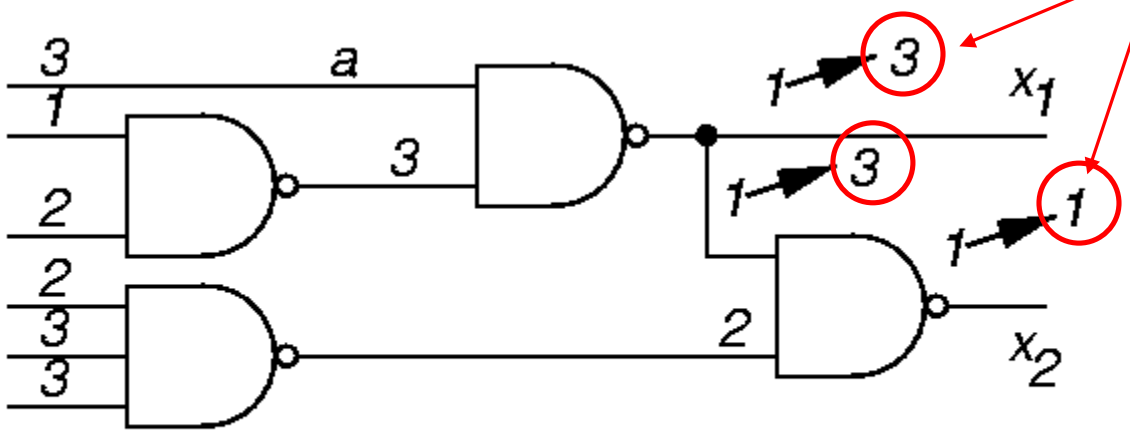
- Ones count testing will need a counter to count the number of 1s in the bit stream.



Transition Counting



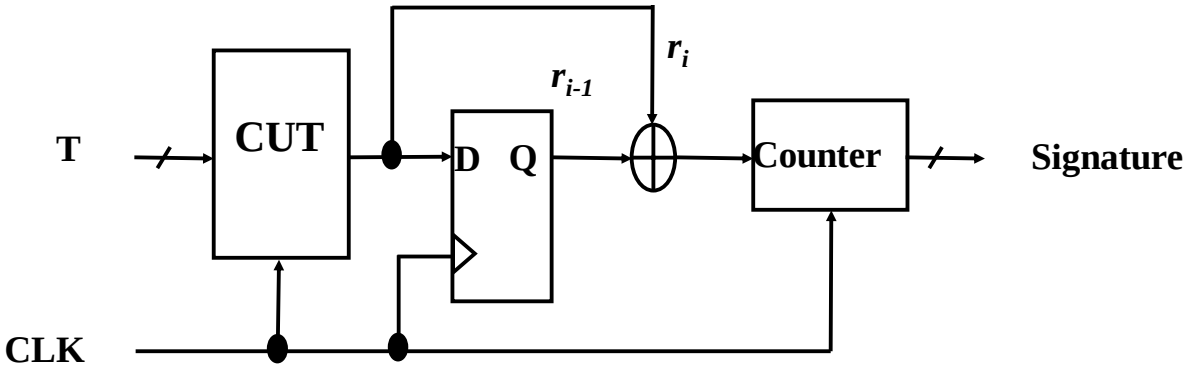
(a) Logic simulation of good machine and fault a stuck-at-1.



(b) Transition counts of good and failing machines.

Transition Counting Details

- Transition count when L bit test sequence is applied to CUT: $C(R) = \Sigma_{i=1 \text{ to } L-1} (r_i \oplus r_{i-1})$
- To maximize fault coverage:
  - Make $C(R_{fault\text{-}free})$ – good machine transition count – as large or as small as possible



Problems with OC and TC Methods

- The aliasing probability in both OC and TC methods depends on the fault-free response
- The aliasing probability is minimum (or maximum) when the fault-free response is in its minimum (or maximum) value of its respective parameter

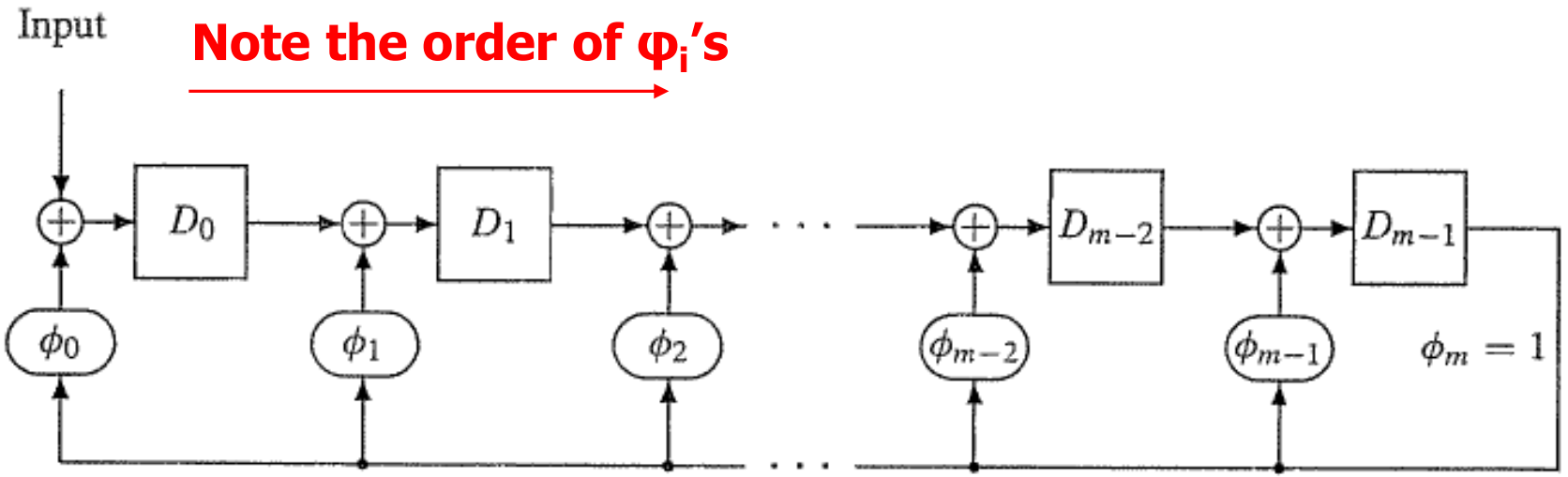
LFSR for Response Compaction

- Use *cyclic redundancy check code* (CRCC) generator (LFSR) for response compacter
- Treat data bits from circuit POs to be compacted as a decreasing order coefficient polynomial
- CRCC divides the PO polynomial by its characteristic polynomial
  - Leaves remainder of division in LFSR
  - Must initialize LFSR to *seed value* (usually 0) before testing
- After testing – compare signature in LFSR to known good machine signature
- Critical: Must compute good machine signature

Serial LFSR Compaction

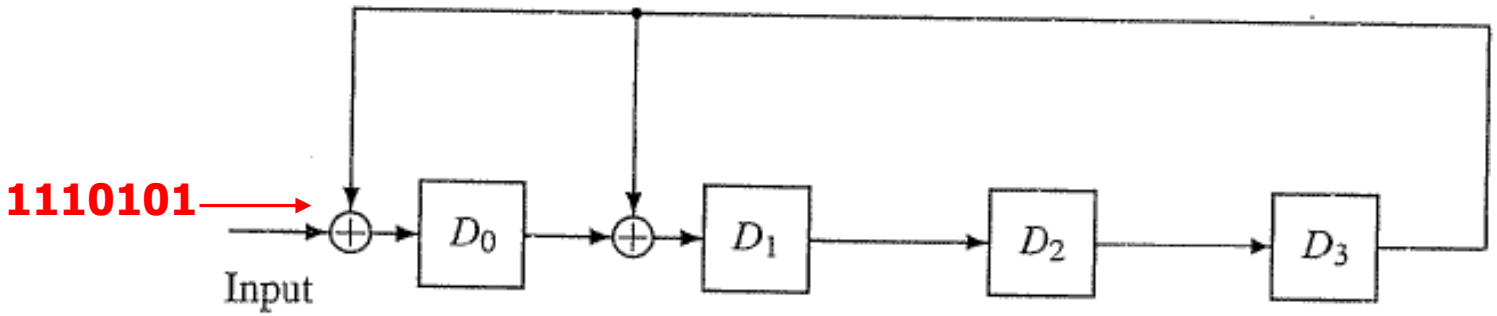
Serial LFSR Compactor

- The polynomial $\varphi(x) = \varphi_m x^m + \ldots + \varphi_1 x + \varphi_0$
- The polynomial relationship
  - L-bit sequence coming to compactor: $m_0 m_1 m_2 \ldots m_{L-1}$ (m_{L-1} is the first bit arriving at compactor)
  - The input polynomial: $M(x)=m_0+m_1x+m_2x^2+\ldots+ m_{L-1}x^{L-1}$
  - $M(x)=q(x)\,\varphi(x)+r(x)$ where $r(x)$ the polynomial remainder of $M(x)/\varphi(x)$ will be the final compactor's response



LFSR Compactor Example I

- The polynomial: $\varphi(x) = x^4 + x + 1$
- The sequence: 1110101 that is $M(x)=1+x+x^2+x^4+x^6$
- Initial state: 0000

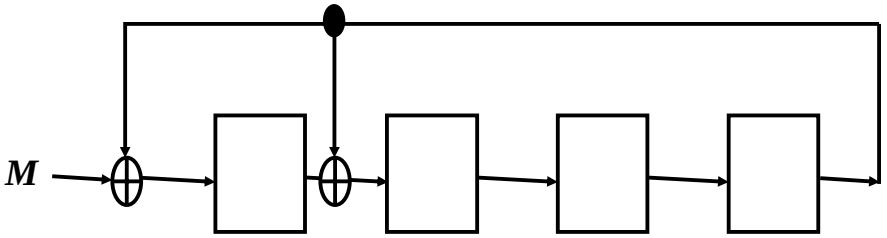


$$
\begin{array}{r|l}
x^4+\ x+\ 1 & x^2+\ 1 \quad \leftarrow \text{Quotient} \\
\hline
 & x^6+\ x^4+\ x^2+\ x+\ 1 \\
 & x^6+\ x^3+\ x^2 \\
\hline
 & x^4+\ x^3+\ x+\ 1 \\
 & x^4+\ x+\ 1 \\
\hline
\text{Signature} \rightarrow & x^3
\end{array}
$$

Time	Input	LFSR state				Output
–	–	0	0	0	0	–
0	$r_1 = 1$	1	0	0	0	0
1	$r_2 = 0$	0	1	0	0	0
2	$r_3 = 1$	1	0	1	0	0
3	$r_4 = 0$	0	1	0	1	0
4	$r_5 = 1$	0	1	1	0	1
5	$r_6 = 1$	1	0	1	1	0
6	$r_7 = 1$	0	0	0	1	1

Final signature (remainder)

LFSR Compactor Example I (cont.)



M	r_0	r_1	r_2	r_3
1	0	0	0	0
1	1	0	0	0
0	1	1	0	0
1	0	1	1	0
1	1	0	1	1
0	0	0	0	1
0	1	1	0	0
1	0	1	1	0
R	1	0	1	1

(a) Fault-free signature

M'	r_0	r_1	r_2	r_3
1	0	0	0	0
1	1	0	0	0
0	1	1	0	0
1	0	1	1	0
0	1	0	1	1
0	1	0	0	1
1	1	0	0	0
1	1	1	0	0
R'	1	1	1	0

Will be detected

(b) Signature for fault f_1

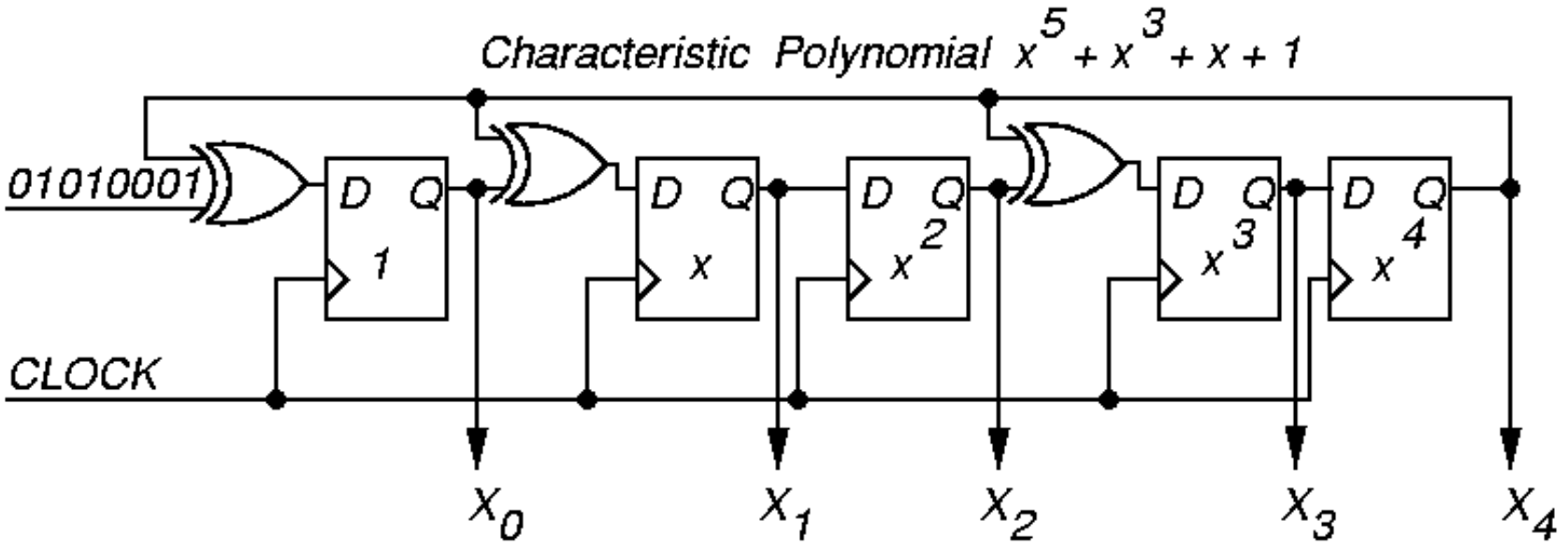
M''	r_0	r_1	r_2	r_3
1	0	0	0	0
0	1	0	0	0
1	0	1	0	0
1	1	0	1	0
0	1	1	0	1
0	1	0	1	0
1	0	1	0	1
1	0	1	1	0
R''	1	0	1	1

Cannot be detected

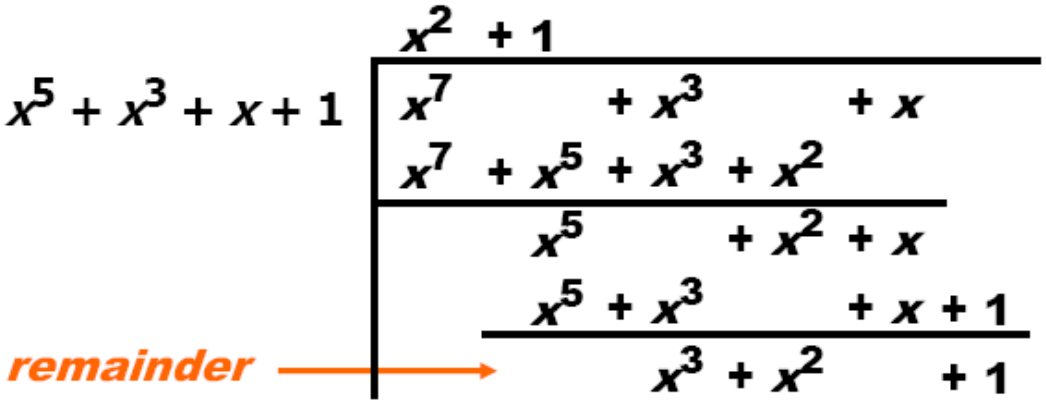
(c) Signature for fault f_2

LFSR Compactor Example II

- LFSR seed value is "00000"
- Input sequence: 0 1 0 1 0 0 0 1
- Symbolic polynomial: $0x^0+1x^1+0x^2+1x^3+0x^4+0x^5+0x^6+1x^7$
- Logic simulation: *Remainder* = $1 + x^2 + x^3$ (i.e. 10110)



LFSR Compactor Example II (cont.)



Logic Simulation:

Inputs	X^0	X^1	X^2	X^3	X^4
Initial State	0	0	0	0	0
1	1	0	0	0	0
0	0	1	0	0	0
0	0	0	1	0	0
0	0	0	0	1	0
1	1	0	0	0	1
0	1	0	0	1	0
1	1	1	0	0	1
0	1	0	1	1	0

Logic simulation: *Remainder* = $1 + x^2 + x^3$

Parallel LFSR Compaction (MISR)

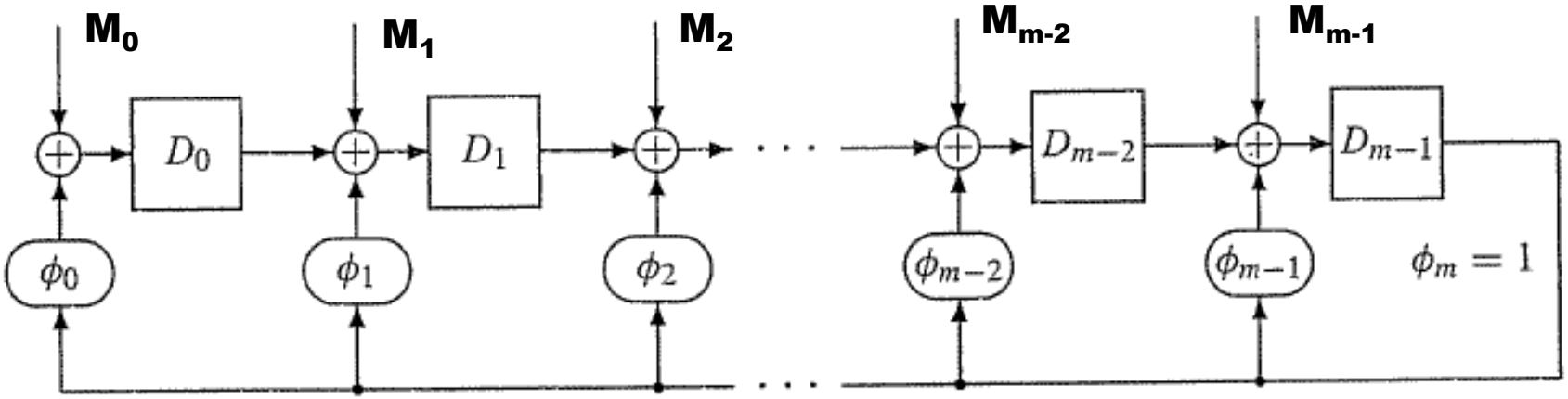
Multiple-Input Signature Register (MISR)

- Problem with ordinary LFSR response compacter:
  - Too much hardware if one of these is put on each *primary output* (PO)
- Solution: MISR – compacts all outputs into one LFSR
  - Works because LFSR is linear – obeys *superposition principle*
  - Superimpose all responses in one LFSR – final remainder is XOR sum of remainders of polynomial divisions of each PO by the characteristic polynomial

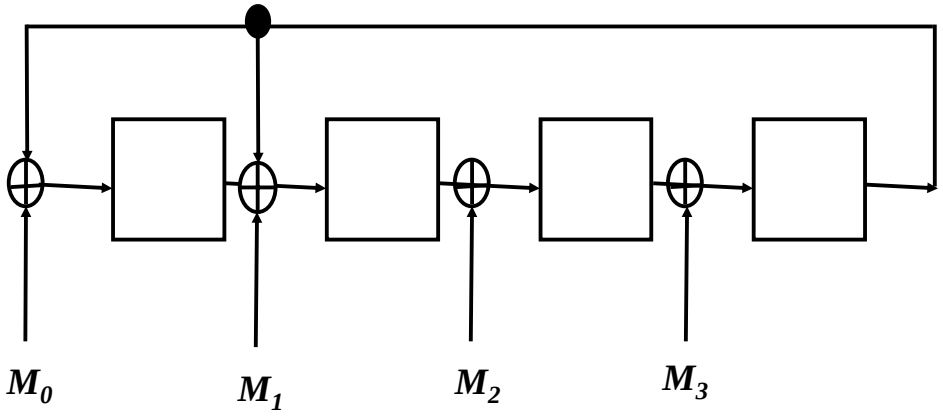
MISR Structure

- The polynomial $\varphi(x) = \varphi_m x^m + \ldots + \varphi_1 x + \varphi_0$
- The polynomial relationship
  - Up to m number of L-bit sequences coming to each input of MISR
  - The effective input polynomial will be c combination of all input polynomials: $M(x)=M_0(x)+x\,M_1(x)+x^2\,M_2(x)+\ldots+x^m\,M_m(x)$
  - $M(x)=q(x)*\varphi(x)+r(x)$ where r(x) the polynomial remainder of $M(x)/\varphi(x)$ will be the final compactor's response

Note the order of φ_i's



A 4-Stage MISR Example



First word coming in

M_0	1 0 0 1 0
M_1	0 1 0 1 0
M_2	1 1 0 0 0
M_3	1 0 0 1 1
M	1 0 0 1 1 0 1 1

- The equivalent sequence will be:
  $1x^0+0x^1+0x^2+1x^3+1x^4+0x^5+1x^6+1x^7$

MISR Matrix Equation

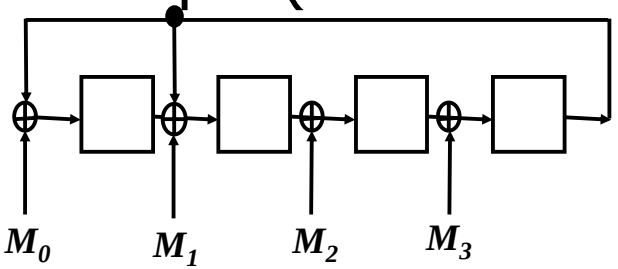
- The polynomial: $\varphi(x) = x^4 + x + 1$
- $M_i(t)$: Input streams of MISR at time t
- The transition matrix/relation:

Internal XOR Transition Matrix (A)

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

$$\begin{bmatrix} s_0(t+1) \\ s_1(t+1) \\ s_2(t+1) \\ s_3(t+1) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} s_0(t) \\ s_1(t) \\ s_2(t) \\ s_3(t) \end{bmatrix} + \begin{bmatrix} M_0(t) \\ M_1(t) \\ M_2(t) \\ M_3(t) \end{bmatrix}$$

- Example (note that * is AND and + is XOR)



Data Stream

M_0	M_1	M_2	M_3
0	0	0	1
1	1	0	1
0	0	0	0
0	1	1	0
1	0	1	1

$$A \cdot \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \qquad A \cdot \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$A \cdot \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \qquad A \cdot \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$A \cdot \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$