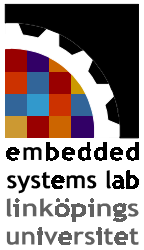
Built-In Self-Test (BIST)

Abdil Rashid Mohamed, abdmo@ida.liu.se
Embedded Systems Laboratory (ESLAB)
Linköping University, Sweden



Introduction

- BIST --> Built-In Self Test
- BIST - part of the circuit (chip, board or system) is used to test the circuit itself
  - *V. Agrawal defines BIST as a DFT technique in which testing (test generation and test application) is accomplished through built-in hardware features*
- BIST = BIT (Built in test) + self test

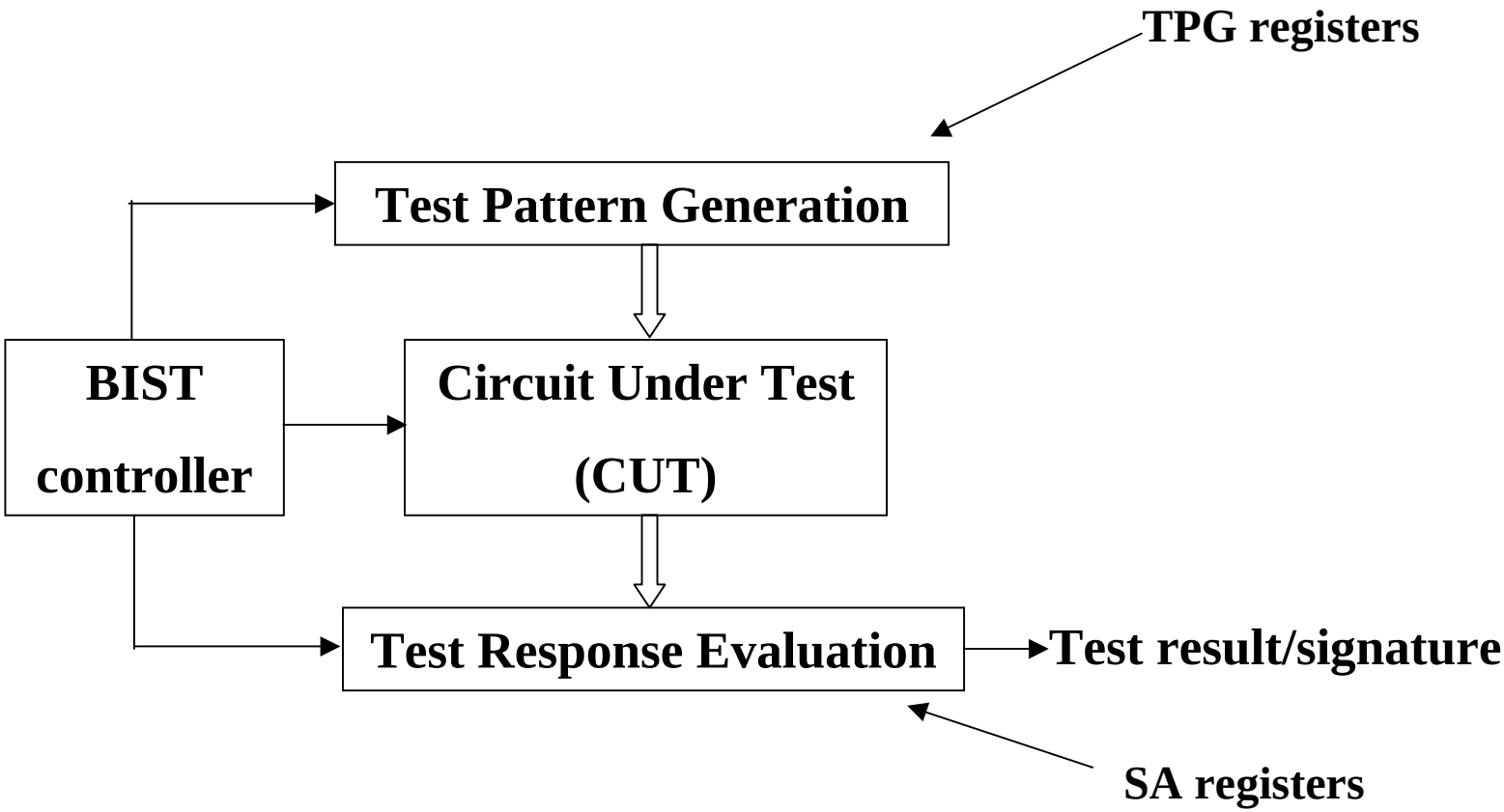
Advantages of BIST

- ***fast, effcicient and hierarchical*** - same hardware is capable of testing chips, boards and systems.
  - At system level BIST is a cheap test solution
  - alternative test solutions are *chip-wise and system-foolish*
- No need of expensive ATE (cost >= $10 milion)
- testing during operation and maintenance
- uniform technique for production, system and maintenance tests
- dynamic properties of the circuit can be tested at speed
- support concurrent testing
- can be used for ***delay testing*** as it can be used in real time

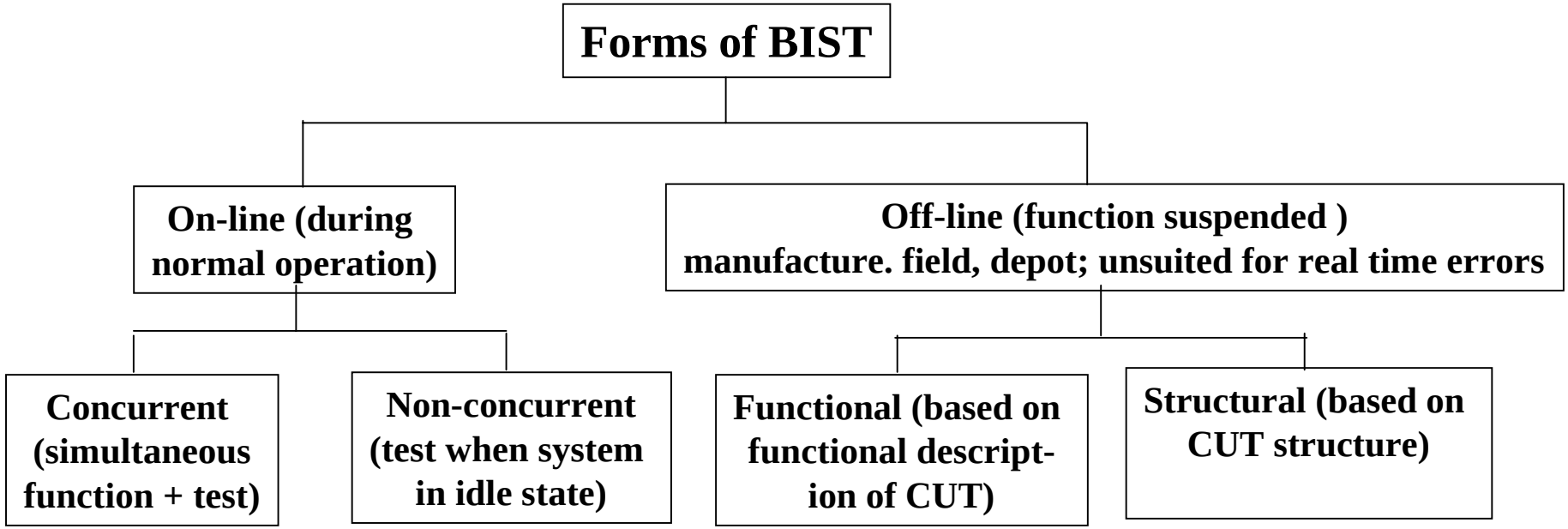
Problems with BIST

- Additional BIST hardware overhead
- Performance degradation, timing problems
- additional delay and elongated critical path

Basic BIST Principle



BIST Classification



Glossary of BIST Test Structures

- TPG – test pattern generator generates test vectors on-chip
  - PRPG – pseudorandom PG generates random, but biased vectors
  - LFSR – Linear feedback shift register commonly used PRPG
  - SRSG – shift register sequence generator single output PRPG
- ORA – (generic) output response analyzer
  - SISR – single input signature register
  - MISR – multiple input signature register

- BILBO – Built in Logic Block Observer non-simultaneous PG & SA
- CBILBO – Concurrent BILBO simultaneous PG & SA
  - too big !

BIST Hardcore

- ***Hardcore -***
  - Part of the circuit that must be operational to perform self test
  - Power supply, ground, clock circuitry, etc.
- Self test failure
  - CUT fails, or
  - Hardcore is faulty
- Hardcore is tested by:
  - external ATE or
  - made self testable by ***redundancy*** such as ***duplication*** or ***self checking logic***

Levels of Self Test

- Production testing of newly manufactured components
  - Levels: chip board system
  - with ***boundary scan*** BIST can be used at all levels of system
- Field testing
  - BIST can diagnose faults down to field-replaceable units – no ATE
  - Improves **maintainability and life cycle cost** of hardware
  - **2-level maintenance -**
    - system performs a self test, and
    - automatic diagnosis of faults to field replaceable units, such as boards.

Test Pattern Generation for BIST

- 1. Exhaustive testing
  - exhaustive test-pattern generators
- 2. Pseudorandom testing
  - weighted and adaptive test generator
- 3. Pseudoexhaustive testing
  - Counters: syndrome driver & constant weight
  - combined LFSR and shift register
  - combined LFSR and XOR gates
  - Others: condensed LFSR, cyclic LFSR, etc.

1. Exhaustive Testing

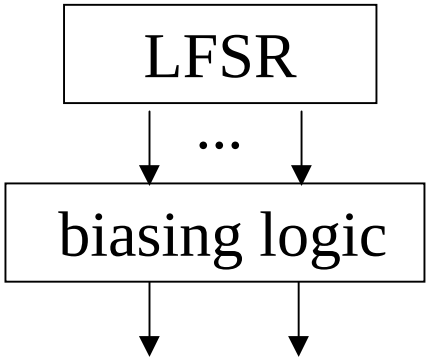
- Assume CUT: ***n-input, m-output combinational***
- Applies all 2^n input test vectors
  - complete testing for all static faults
  - detects all detectable faults
- Suitable TPGs
  - binary counters
  - max. length autonomous LFSR (*complete LFSR*) modified to include an all-0s state
- Drawbacks:
  - unfeasible if ***n > 22*** □ too many test vetors
  - not applicable to sequential circuits

2. Pseudorandom Testing

- Test patterns: $< 2^n$
  - random, but deterministic and repeatable
  - autonomous LFSR is used
- Generation
  - with replacement(pattern generated more than once) or
  - without replacement (each pattern is unique)
- FC can be determined by *fault simulation*
- An acceptable level of FC is obtained by selecting suitable Test length
- Random resistant faults
  - long test lengths to insure high FC
- LFSR produces test patterns with equal no. of 0's and 1's on each input line
- Biased distribution of 0's and 1's captures more faults, ↑FC, ↓test vectors

Weighted Test Generation

- TPG with ***non-uniform*** distribution of 0's and 1's on the output lines
  - change prob. Of a 0 or 1 to improve FC
  - For example: LFSR Prob(0) = Prob(1) = 0.5
  - Weighted LFSR Prob(0) = 0.25; Prob(1) = 1- Prob(0)
- Pre-process procedure is used to determine the weights and design weighting circuitry
- Constructed using an autonomous LFSR and combinational circuit



Adaptive Test Generation

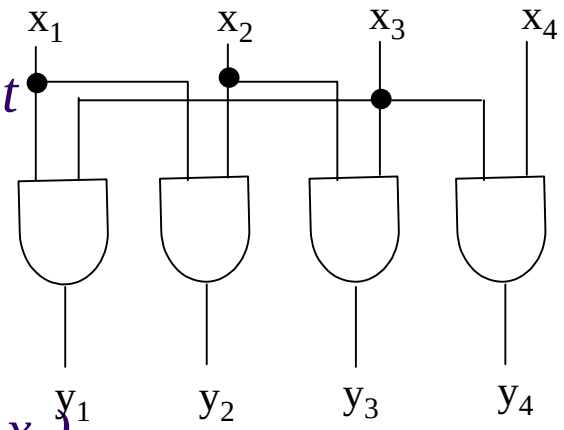
- Employs a weighted TPG
- Use fault simulation results to modify the weights
  - one or more probability distributions for the test patterns
- a TPG is designed based on the probability distributions above
- Adaptive TPG
  - efficient in terms of test length, but
  - hardware can be complex

3. Pseudoexhaustive Testing

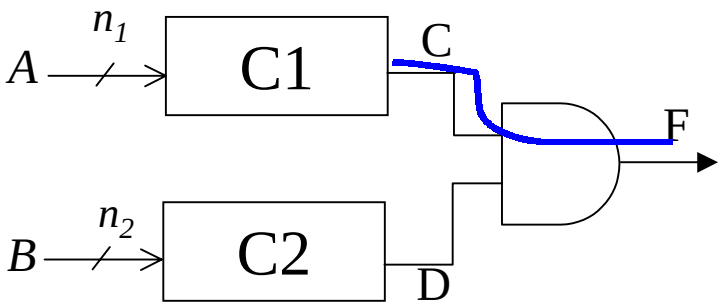
- All benefits of exhaustive testing, but ***fewer test patterns***
- Idea: Segment ***circuit*** and ***exhaustively test*** each segment
  - logical segmentation
    - cone segmentation (verification testing)
    - sensitized path segmentation
  - physical segmentation
- To test n-input circuit
  - reconfigure input lines to generate tests on m lines (m < n)
  - M lines –*test signals*- fan out and drive the n lines to CUT

Cone segmentation

- Cone segmentation (verification testing)
  - Segment *m* output circuit into *m cones*,
  - cone consist of logic associated with *one output*
  - test each cone exhaustively
  - all cones are tested concurrently
- An example of an *(n,w)-CUT = (4,2)-CUT*
  - $y_1=f_1(x_1,x_3)$; $y_2=f_2(x_1,x_2)$; $y_3=f_3(x_2,x_3)$; $y_4=f_4(x_3,x_4)$
  - *4 cones* y_1, y_2, y_3 & y_4 *tested concurrently*
  - *4 test vectors to each cone*



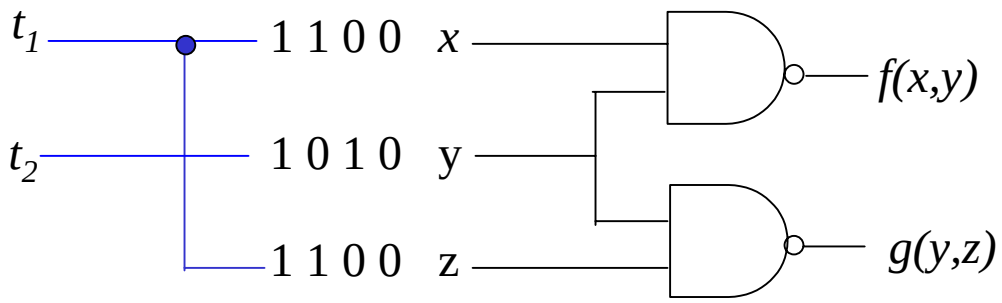
Sensitized-path segmentation



- Segmentation based on ***path sensitization***
- Exhaustive test of C1 –
  - establish sensitized path from C to F by:
  - Apply 2^{n1} patterns to A
  - Set B such that D = 1
  - AND gate is also tested
- Similar testing for C2
- Effective testing using only $2^{n1}+ 2^{n2} +1$ test vectors instead of 2^{n1+n2}
- If A, B --> PRPG, F --> MISR,
  - BIST hardware sharing is achieved

Identification of Test Signal Inputs

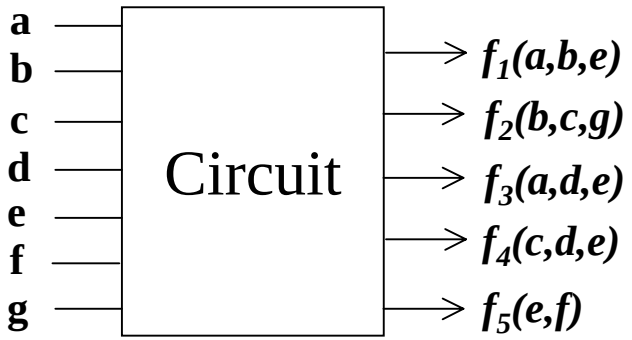
- ***P-test signals*** drive all *N-inputs* of circuit, *p<n.*
- Some of *p lines* fan-out to one/more of the normal input lines
- Exhaustive test of multiple output function *(f,g)* needs 8 test vectors
- 4 test vectors test the two functions, *f and g*, exhaustively and concurrently
  - No output is a function of both *x and z*
  - *Same test data can be applied to both lines x & z one test signal*
  - *Circuit testable using only two test signals*
  - *Verification test inputs x=z*



Procedure - Identify Minimal Set of Test Signals

- ***Step 1:*** Partition the circuit into disjoint subcircuits
- ***Step 2:*** For each disjoint subcircuit
  - Generate a dependency matrix, D
  - Partition D into groups of inputs so that two/more inputs in a group do not affect same output
  - Collapse each group to form an equivalent input, called a test signal input.
- For n-input, m-output circuit, dependency matrix D =[d_{ij}]; m rows, n columns
  - Dij = 1 if output i depends on input j; otherwise dij =0
- ***Step 3:*** Extract from Dc: number of partitions (width, p) and maximum number of 1s in the row (weight, w).
  - P represents max. no. Of input signals to test a disjoint subcircuit
  - W represents max. no. Of signals on which any output depends
  - Thus, pseudoexhaustive test length, L is given by: $2^w <= L <= 2^p$
- ***Step 4: construct test patterns***

Example



$\Longrightarrow$ D =

a	b	c	d	e	f	g	
1	1	0	0	1	0	0	f_1
0	1	1	0	0	0	1	f_2
1	0	0	1	1	0	0	f_3
0	0	1	1	1	0	0	f_4
0	0	0	0	1	1	0	f_5

$\Longrightarrow$ D_g =

I		II		III	IV		
a	c	b	d	e	f	g	
1	0	1	0	1	0	0	f_1
0	1	1	0	0	0	1	f_2
1	0	0	1	1	0	0	f_3
0	1	0	1	1	0	0	f_4
0	0	0	0	1	1	0	f_5

- Less than two 1s in each row and no. Of groups should be minimal
  - No output is driven by more than one input from each group
- Collapsed equivalent matrix, D_c, OR-ing each row in a group to form a single column

$$\Longrightarrow \quad D_c = \begin{array}{c|cccc|c} & \text{I} & \text{II} & \text{III} & \text{IV} & \\ & t_1 & t_2 & t_3 & t_4 & \\ \hline & 1 & 1 & 1 & 0 & f_1 \\ & 1 & 1 & 0 & 1 & f_2 \\ & 1 & 1 & 1 & 0 & f_3 \\ & 1 & 1 & 1 & 0 & f_4 \\ & 0 & 0 & 1 & 1 & f_5 \end{array}$$

- Number of partitions in Dc (width), p=4
  - p - max. no. of input signals to test disjoint subcircuit
- max. no. of 1's in any row (weight), w=3
  - max. np. Of signals on which any output depends
- Thus, pseudoexhaustive test length, L is given by: $2^w <= L <= 2^p$

Coffee Break

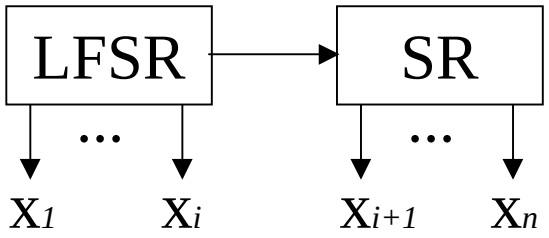
TPG for Pseudoexhaustive Tests

- Counters
  - Syndrome drive counter
  - constant weight counter
- Linear Feedback Shift Registers (LFSRs)
  - combined LFSR/SR
  - combined LFSR/XOR gates

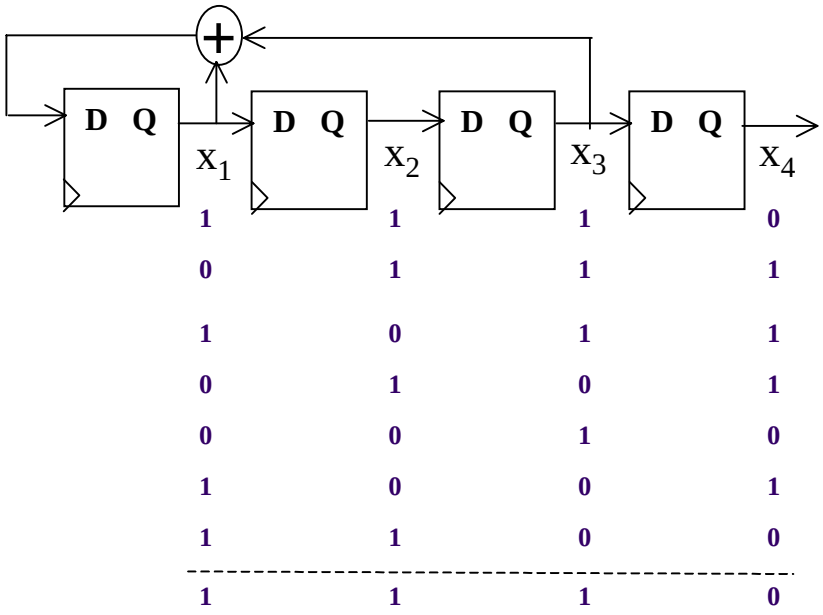
Constant Weight Counter

- Constant weight counter
  - ***N-out-of-M*** code
  - A set of codewords of M-bits, where each codeword has *N* 1s.
- E.g. 2-out-of-4 CWC is: ***1100, 1010, 1001, 0110, 0101, 0011***
- Any *(n,w)* circuit can be pseudoexhaustively tested by a constant-weight counter implementing a ***w-out-of-K*** code, for an appropriate value of K.
- For large values of N and M,
  - constant weight counter is too complex high.

Combined LFSR/SR

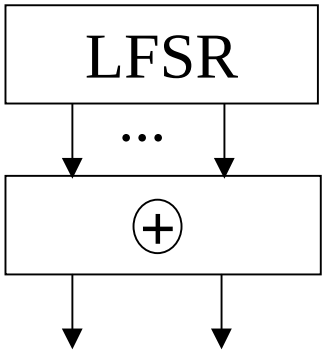


An LFSR/SR verification test generator

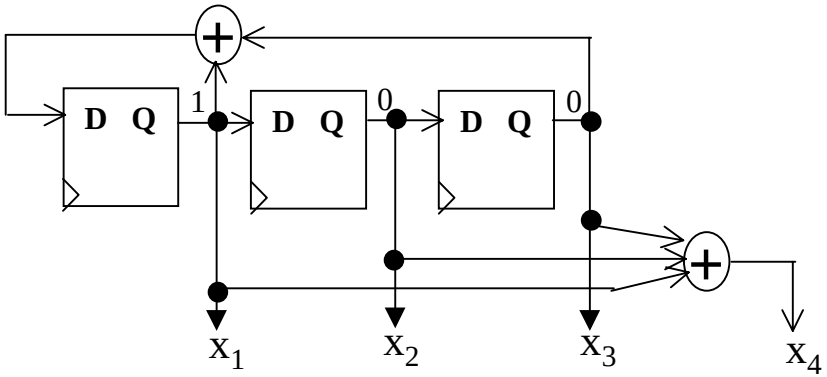


- Combination of LFSR and SR
- Advantages and drawbacks
  - cheaper than a constant weight counter,
  - number of vectors is near minimal when w << n, e.g w < n/2
  - generates more test vectors and LFSR needs at least 2 seed values
- On the left - 4-stage combined LFSR/SR for testing a (4,2)-CUT

Combined LFSR/XOR Gates



An LFSR/XOR verification test generator



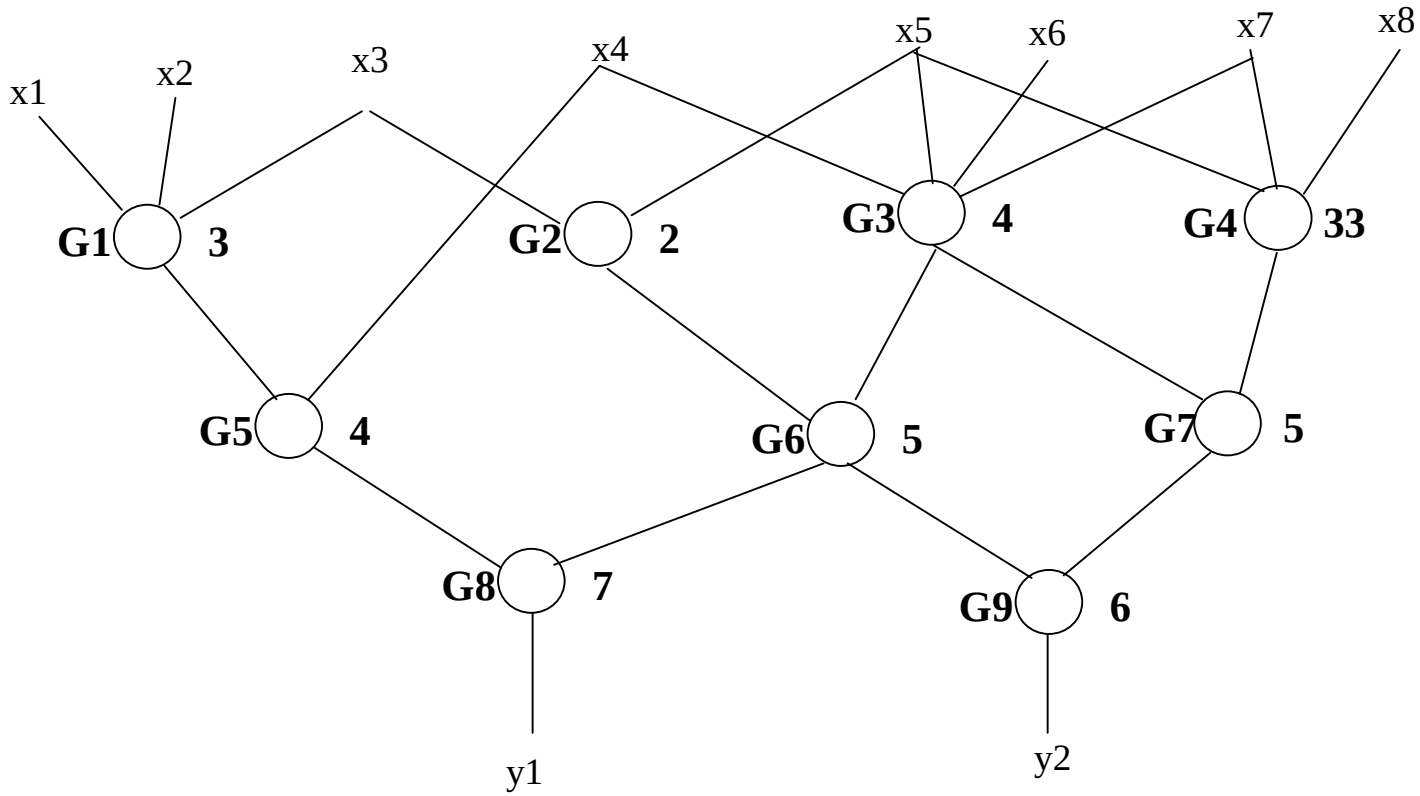
1	0	0	1
1	1	0	0
1	1	1	1
0	1	1	0
1	0	1	0
0	1	0	1
0	0	1	1
1	0	0	1

- combination of LFSR and XOR gates (linear) network
  - based on *linear sums or linear codes*
  - Requires at most two seeds
  - no.of patterns approaches that required for LFSR/SR
- example design for testing a (4,2) CUT

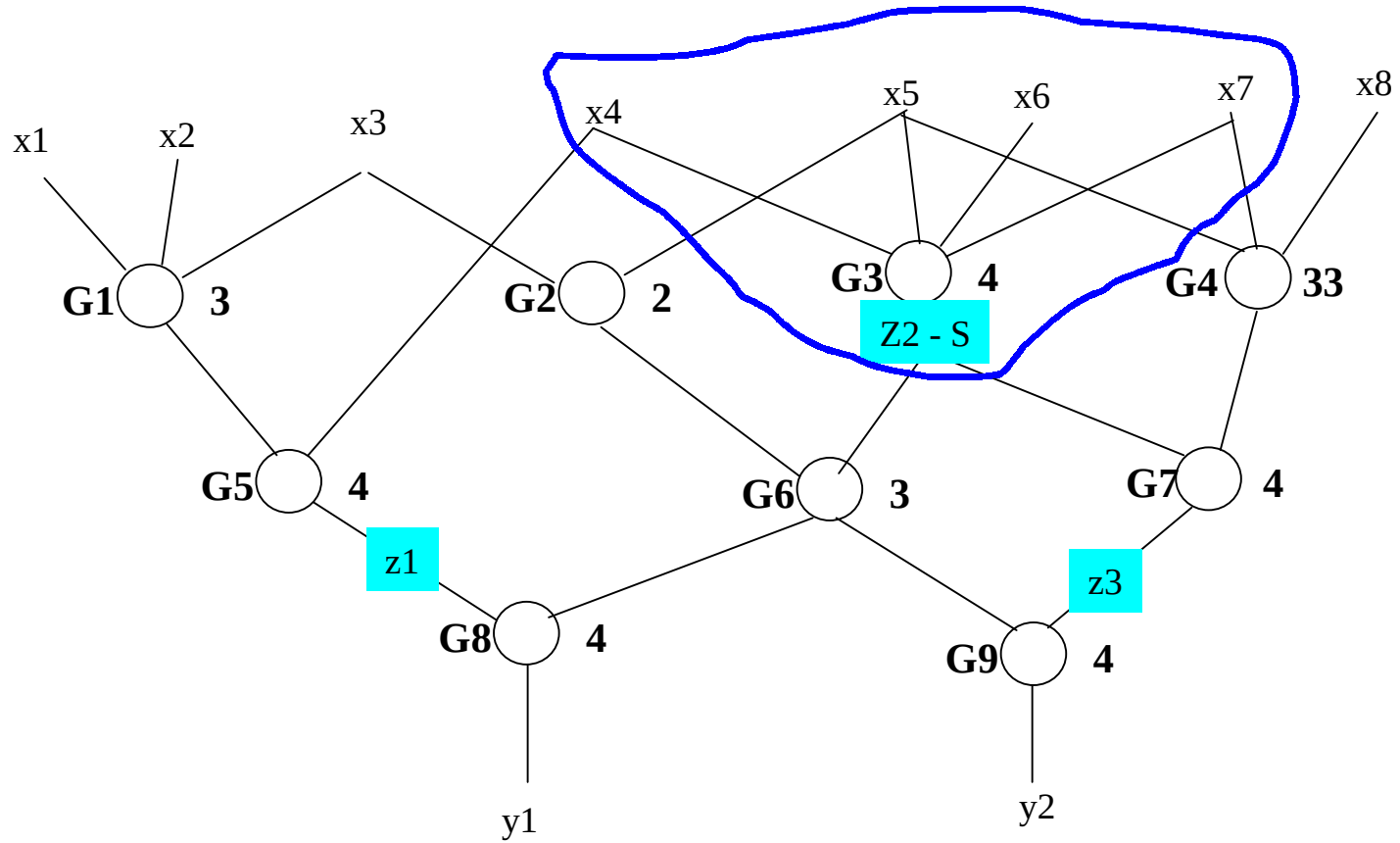
Physical Segmentation

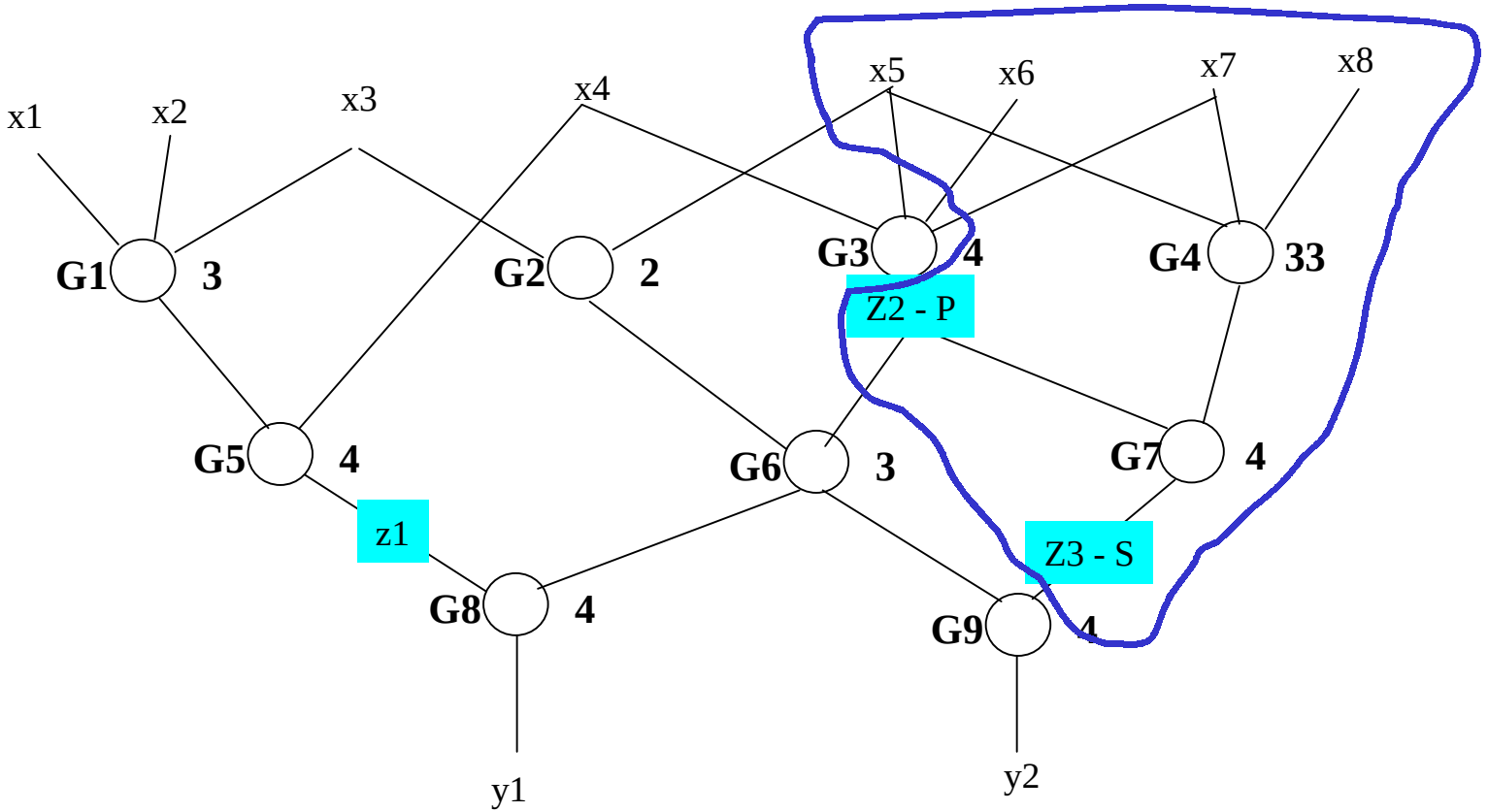
- Pseudoexhaustve methods large test sets for large circuits
- Employ physical segmentation to achieve pseudoexhaustive test sets
- Circuit is partitioned using hardware segmentation techniques
  - Chapter 9 partitioning circuit to reduce test generation cost
  - Insert **bypass storage** cells in some signal lines
    - bypass storage cell acts as a wire in normal mode and as part of LFSR in test mode
- **Bypass cell on line x **
  - If associated LFSR is PRPG it generate test patterns on x
  - If associated LFSR is MISR it detect errors on x

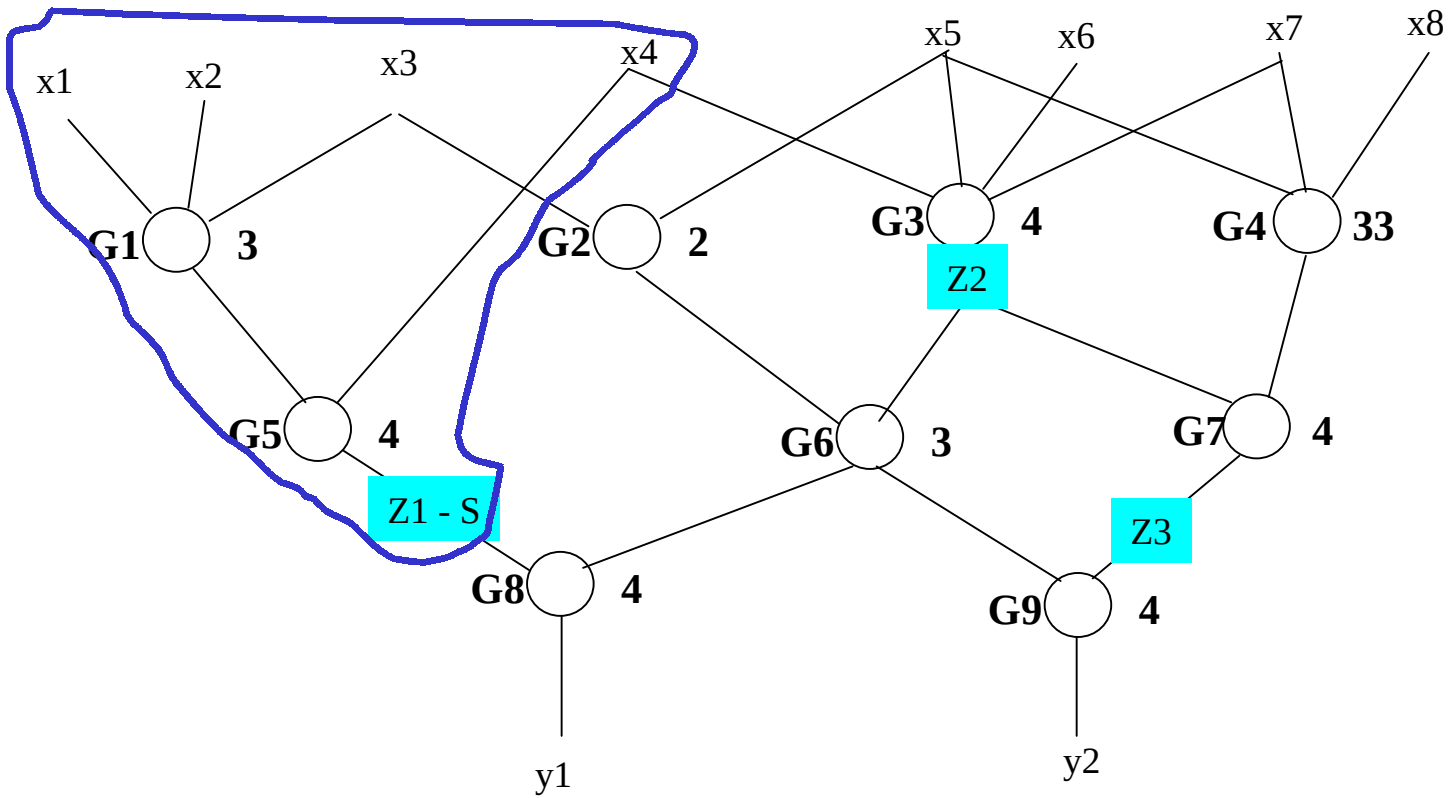
Adding bypass storage cells

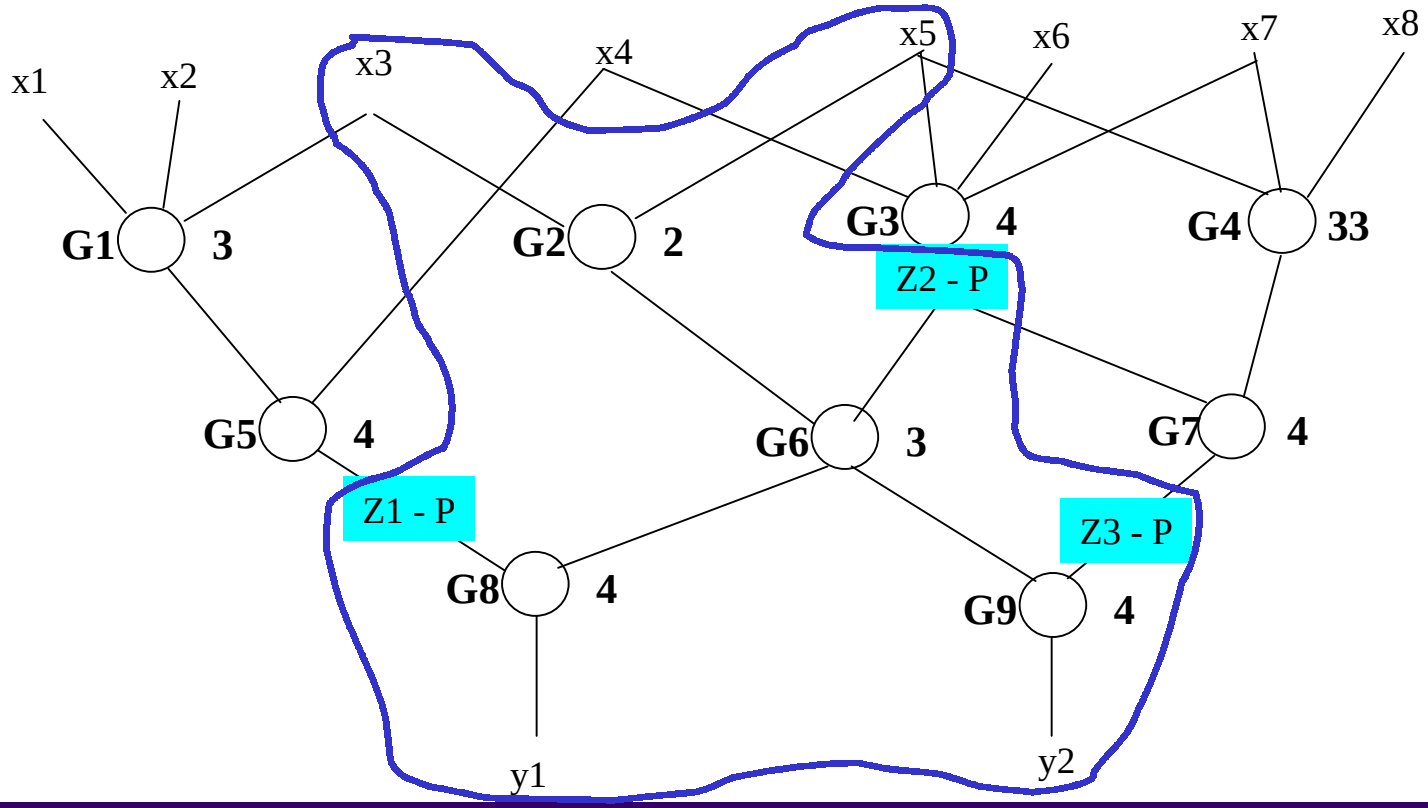


- Segment circuit such that each signal is a function of not more than 4 variables




x1
x2
x3
x4
x5
x6
x7
x8
G1
3
G2
2
G3
4
G4
33
Z2 - P
G5
4
G6
3
G7
4
z1
Z3 - S
G8
4
G9
4
y1
y2


x1
x2
x3
x4
x5
x6
x7
x8
G1
3
G2
2
G3
4
Z2
G4
33
G5
4
G6
3
G7
4
Z1 - S
Z3
G8
4
G9
4
y1
y2

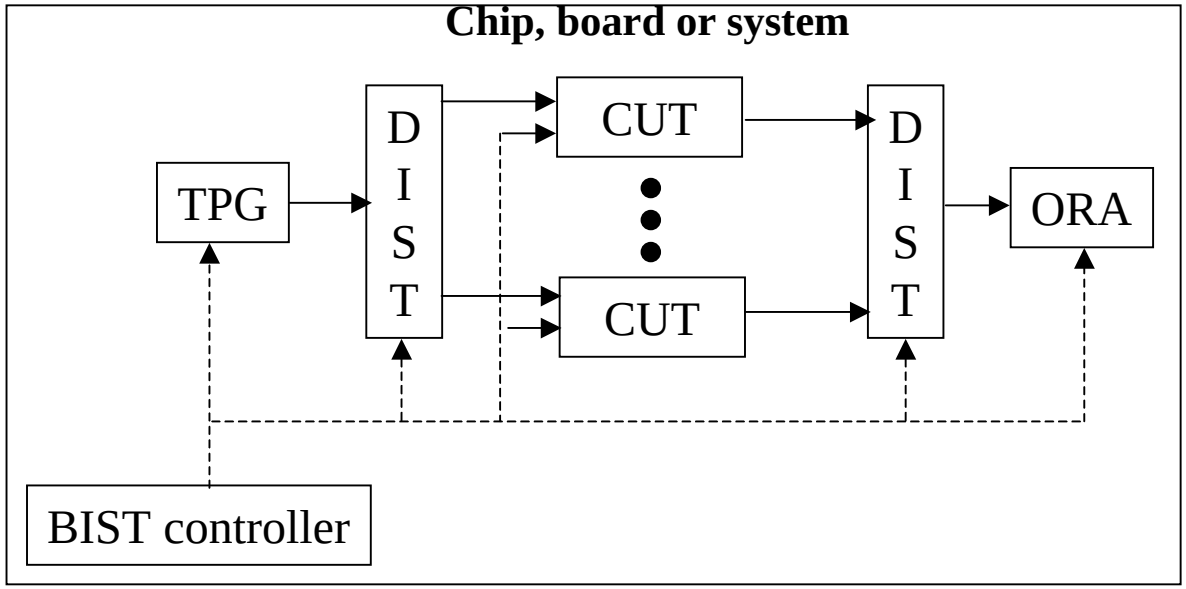

x1
x2
x3
x4
x5
x6
x7
x8
G1
3
G2
2
G3
4
G4
33
Z2 - P
G5
4
G6
3
G7
4
Z1 - P
Z3 - P
G8
4
G9
4
y1
y2

Generic-Off Line BIST Architectures

- BIST architecture
  - BIST structures for testing chips and boards consisting of blocks of combinational logic interconnected by storage cells
- BIST architecture components
  - TPGs, ORAs, CUT
  - Distribution system (DIST) for data: TPGs to CUT and CUT to ORAs
  - Interconnections(wires), busses, multiplexers and scan paths
  - BIST controller for controlling the BIST circuitry and CUT during BIST mode
- Type 1: **Centralized or distributed BIST** architecture
- Type 2: **Embedded or separate BIST** architecture

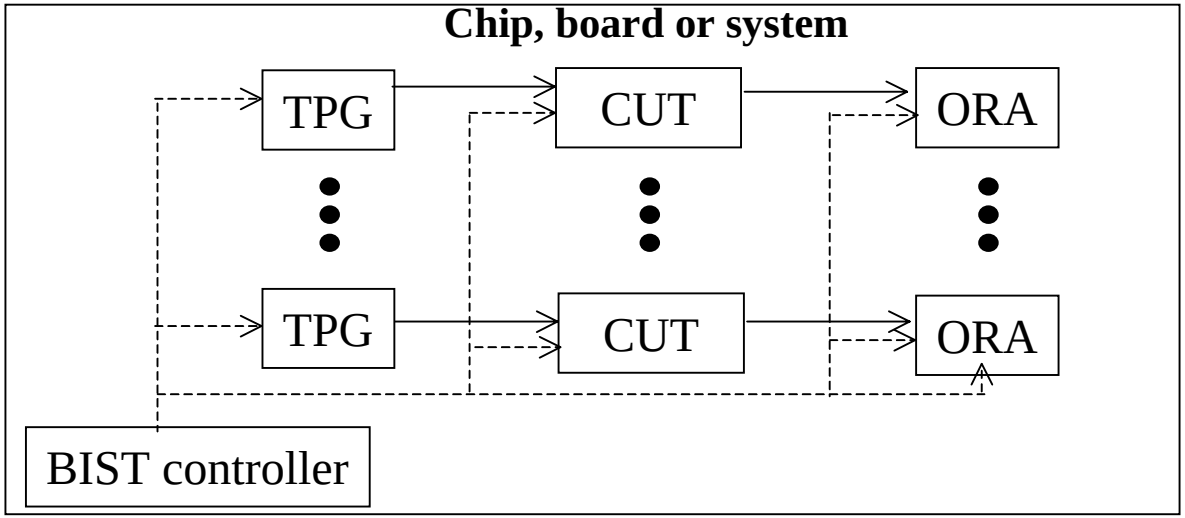
Centralized and Separate BIST architecture

- Several CUTs share TPGs and ORAs
  - Reduced overhead
  - Increased testing time
  - separate architecture - BIST circuitry is external to CUT and not part of functional circuitry



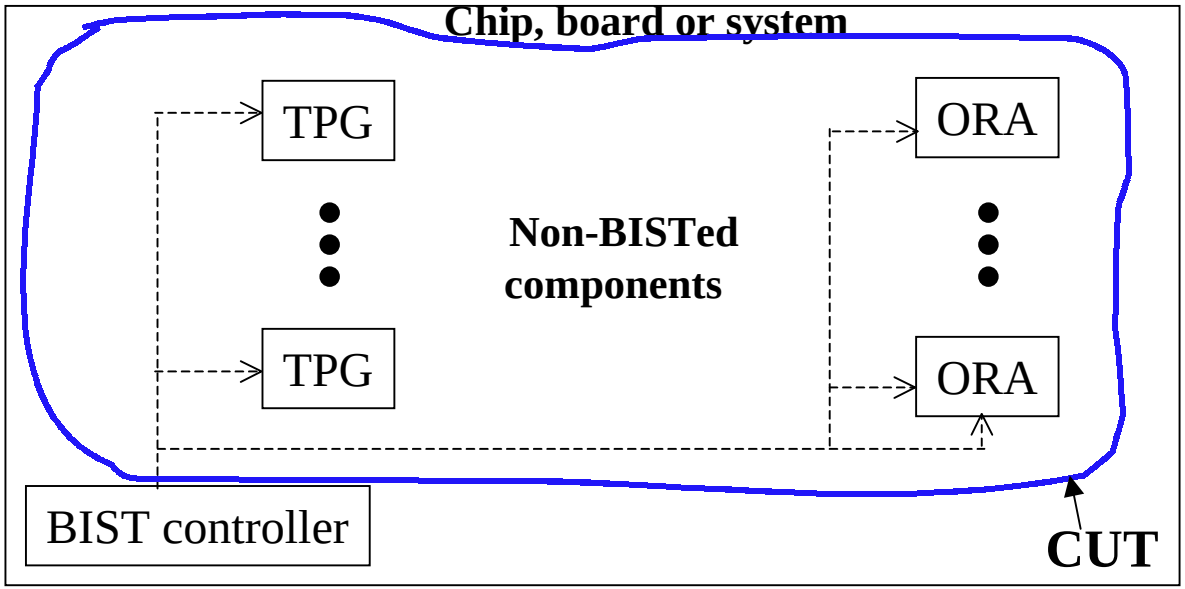
Distributed and Separate BIST architecture

- Each CUT has its own TPGs and ORAs circuitry
  - more overhead
  - reduced test time and accurate diagnosis
  - separate architecture - BIST circuitry is external to CUT and not part of functional circuitry



Distributed and Embedded BIST Architecture

- TPG and ORA are configured from among functional elements in the CUT
  - complex BIST controller
  - less hardware than distributed and separate architectures



Centralized and Embedded Architecture

- Test for attentiveness !
- Draw ”centralized and embedded architecture” and show CUT

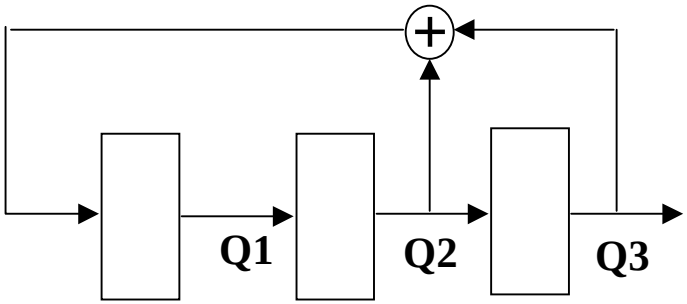
BIST Controller

- Single step CUT through some test sequences
- **Inhibit system** clocks and **control test** clocks
- Communicate with other test controllers using test buses
- Control self test operations
  - seeding of registers.
  - Keep track of the number of shift commands required in a scan operation
  - keep track of the number of test patterns processed

Choosing BIST Architecture

- Hints
  - low test time & degree of test parallelism--> distributed BIST
  - high FC -->distributed, customized TPG/ORA to each CUT
  - level of packaging --> at higher levels use centralized BIST
  - physical constraints (size, weight, power, cooling) --> embedded & separate BIST requires more hardware, high performance degradation
  - complexity of replaceable units --> self testable replaceable units must contain TPG & ORA.
    - If a system is the lowest level of replaceable units, then its constituent boards need not have TPG/ORA and more centralized BIST arch. Can be used.
  - Factory and field test-and-repair strategy --> BIST affects type and usage of ATE
  - performance degradation --> BIST hardware in critical paths reduces the system clock rate

Linear Feedback Shift Register (LFSR) - Example



S0	0	1	1
S1	0	0	1
S2	1	0	0
S3	0	1	0
S4	1	0	1
S5	1	1	0
S6	1	1	1
S7	0	1	1

Problems of Testing SoC

- Heteregoneous components - processors, memories, random logic.
- Core based design - reduced accessibility oif internal structures
- increasung complexity - large amount of test data
- number of access ports - remains the same
- high speed/frequency - high demand on tester’s driver/sensor mechanism
- deep submicron technolkogy - compliucated failure mechanism

- Solution --> BIST

Main Issues to be Considered in BIST for SoC

- Exploit existing circuits for BIST purpose to reduce hardware overhead
- optimize the BIST design with th rest of the circuit to avoid performance degradation
- share the same BIST components for different modules
- testing the BIST logic itself

The end

– ”BIST & DFT is fun and fine”