Design Verification and Test of Digital VLSI Circuits NPTEL Video Course

Module-XI

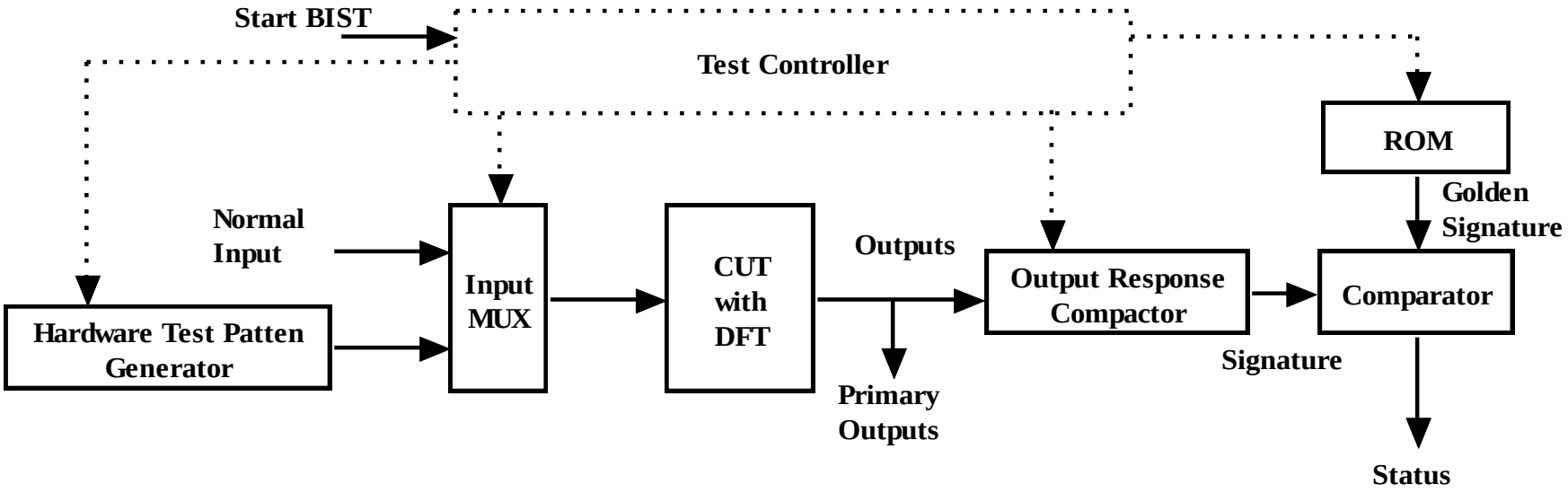
Lecture-I

Built in Self Test

Introduction

- VLSI testing, only from the context where the circuit needs to be put to a “test mode” for validating that it is free of faults.
  - Circuits tested OK are shipped to the customers with the assumption that they would not fail within their expected life time; this is called off-line testing
  - However, this assumption does not hold for modern day ICs, based on deep sub-micron technology, because they may develop failures even during operation within expected life time.
- To cater to this problem sometimes redundant circuitry are kept on-chip which replace the faulty parts.
- Testing a circuit every time before they startup, is called Built-In-Self-Test (BIST).

Basic architecture of BIST



Basic architecture of BIST

Hardware Test Pattern Generator:

•This module generates the test patterns required to sensitize the faults and propagate the effect to the outputs

• As the test pattern generator is a circuit (not equipment) its area is limited.

  •So storing and then generating test patterns obtained by ATPG algorithms on the CUT (discussed in Module XI) using the hardware test pattern generator is not feasible.

  •Instead, the test pattern generator is basically a type of register which generates random patterns which act as test patterns. The main emphasis of the register design is to have low area yet generate as many different patterns (from 0 to 2^n-1, if there are n flip-flops in the register) as possible.

Basic architecture of BIST

Input Mux: This multiplexer is to allow normal inputs to the circuit when it is operational and test inputs from the pattern generator when BIST is executed. The control input of the multiplexer is fed by a central test controller.

Output response compactor: Output response compacter performs lossy compression of the outputs of the CUT. The output of the CUT is to be compared with the expected response (called golden signature

Similar to the situation for test pattern generator, expected output responses cannot be stored explicitly in a memory and compared with the responses of the CUT. So CUT response needs to be compacted such that comparisons with expected responses (golden signatures) become simpler in terms of area of the memory that stores the golden signatures.

Basic architecture of BIST

ROM: Stores golden signature that needs to be compared with the compacted CUT response.

Comparator: Hardware to compare compacted CUT response and golden signature (from ROM).

Test Controller: Circuit to control the BIST. Whenever an IC is powered up (signal start BIST is made active) the test controller starts the BIST procedure. Once the test is over, the status line is made high if fault is found. Following that, the controller connects normal inputs to the CUT via the multiplexer, thus making it ready for operation.

Hardware pattern generator

There are two main targets for the hardware pattern generator—

(i) low area and

(ii) pseudo-exhaustive pattern generation (i.e., generate as many different patterns from 0 to $2^n - 1$ as possible, if there are n flip-flops in the register).

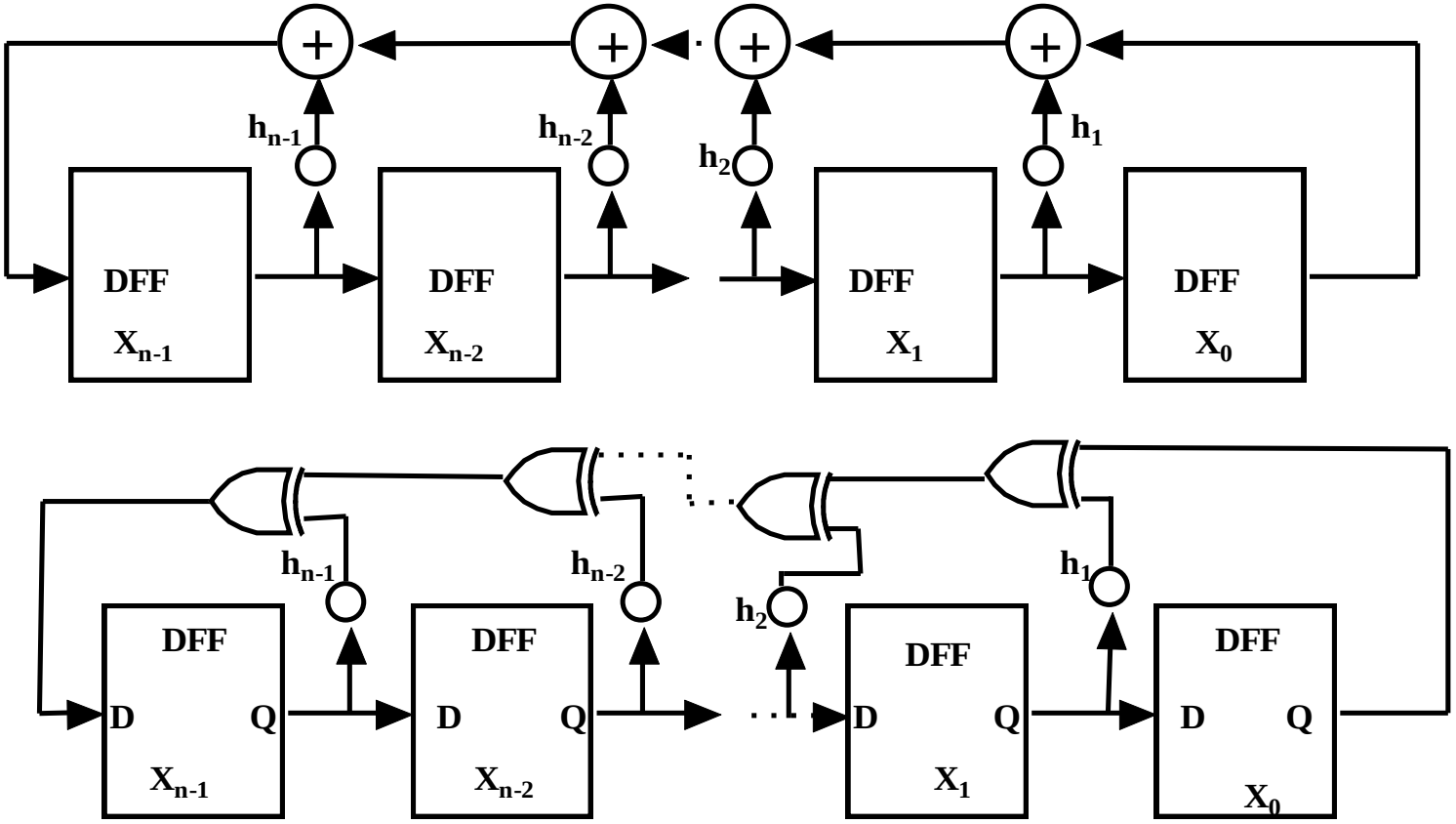
Linear feedback shift register (LFSR) pattern generator is most commonly used for test pattern generation in BIST because it satisfies the above two conditions.

There are basically two types of LFSRs,

(i) standard LFSR

(ii) modular LFSR.

Standard LFSRs



A properly-designed LFSR can generate as a near-exhaustive set of patters, as it can cycle through distinct $2^n - 1$ states (except 0s is all flip-flops). Such a properly designed LFSR is known as a maximal length LFSR.

Standard LFSRs

This LFSR in terms of the matrix can be written as $X(t+1) = T_S X(t)$.

$$\begin{bmatrix} X_0(t+1) \\ X_1(t+1) \\ \cdots\cdots\cdots\cdots \\ X_{n-3}(t+1) \\ X_{n-2}(t+1) \\ X_{n-1}(t+1) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0.....0 & 0 \\ 0 & 0 & 1.....0 & 0 \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ 0 & 0 & 0.....1 & 0 \\ 0 & 0 & 0.....0 & 1 \\ 1 & h_1 & h_2 \quad h_{n-2} & h_{n-1} \end{bmatrix} \begin{bmatrix} X_0(t) \\ X_1(t) \\ \cdots\cdots\cdots\cdots \\ X_{n-3}(t) \\ X_{n-2}(t) \\ X_{n-1}(t) \end{bmatrix}$$

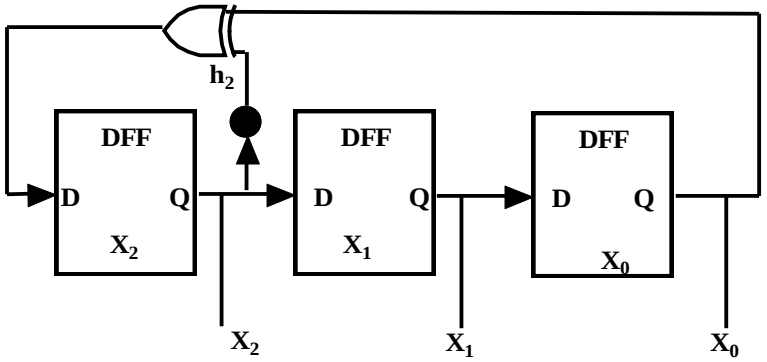
Standard LFSRs

Leaving behind the first column and the last row T_S is an identity matrix; this indicates that X_0 gets input from X_1, X_1 gets input from X_2 and so on. Finally, the first element in the last row is 1 to indicate that X_{n-1} gets input from X_0. Other elements of the last row are the tap points $h_1, h_2, \ldots h_{n-2}, h_{n-1}$. The value of $h_i = 1,\ (1 \le i \le n-1)$, indicates that output of flip-flop X_i provides feedback to the linear XOR function. Similarly, the value of $h_i = 0,\ (1 \le i \le n-1)$, indicates that output of flip-flop X_i does not provide feedback to the linear XOR function.

This LFSR can also be described by the characteristic polynomial:

$$f(x) = 1 + h_1 x + h_2 x^2 + \ldots h_{n-2} x^{n-2} + h_{n-1} x^{n-1} + x^n$$

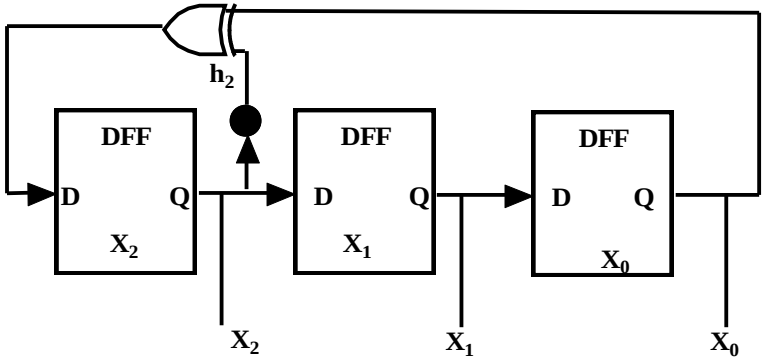
Standard LFSR: Example



$$\begin{bmatrix} X_0(t+1) \\ X_1(t+1) \\ X_2(t+1) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_0(t) \\ X_1(t) \\ X_2(t) \end{bmatrix}$$

It may be noted that output of flip-flop X_2 provides feedback to the XOR network, while flip-flop X_1 does not; so $h_1 = 0$ and $h_2 = 1$. The characteristic polynomial of the LFSR is $f(x) = 1 + x^2 + x^3$.

Standard LFSR: Example



$$\begin{bmatrix} X_0(t+1) \\ X_1(t+1) \\ X_2(t+1) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_0(t) \\ X_1(t) \\ X_2(t) \end{bmatrix}$$

$$\begin{bmatrix} X_0 \\ X_1 \\ X_2 \end{bmatrix} ::: \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

So the LFSR generates 7 patterns (excluding all 0s) after which a pattern is repeated. It may be noted that this LFSR generates all patters (except all 0s) which are generated by a 3 bit counter, however, the area of the LFSR is much lower compared to a counter. In a real life scenario, the number of inputs of a CUT is of the order of hundreds. So LFSR has minimal area compared to counters (of order of hundreds).

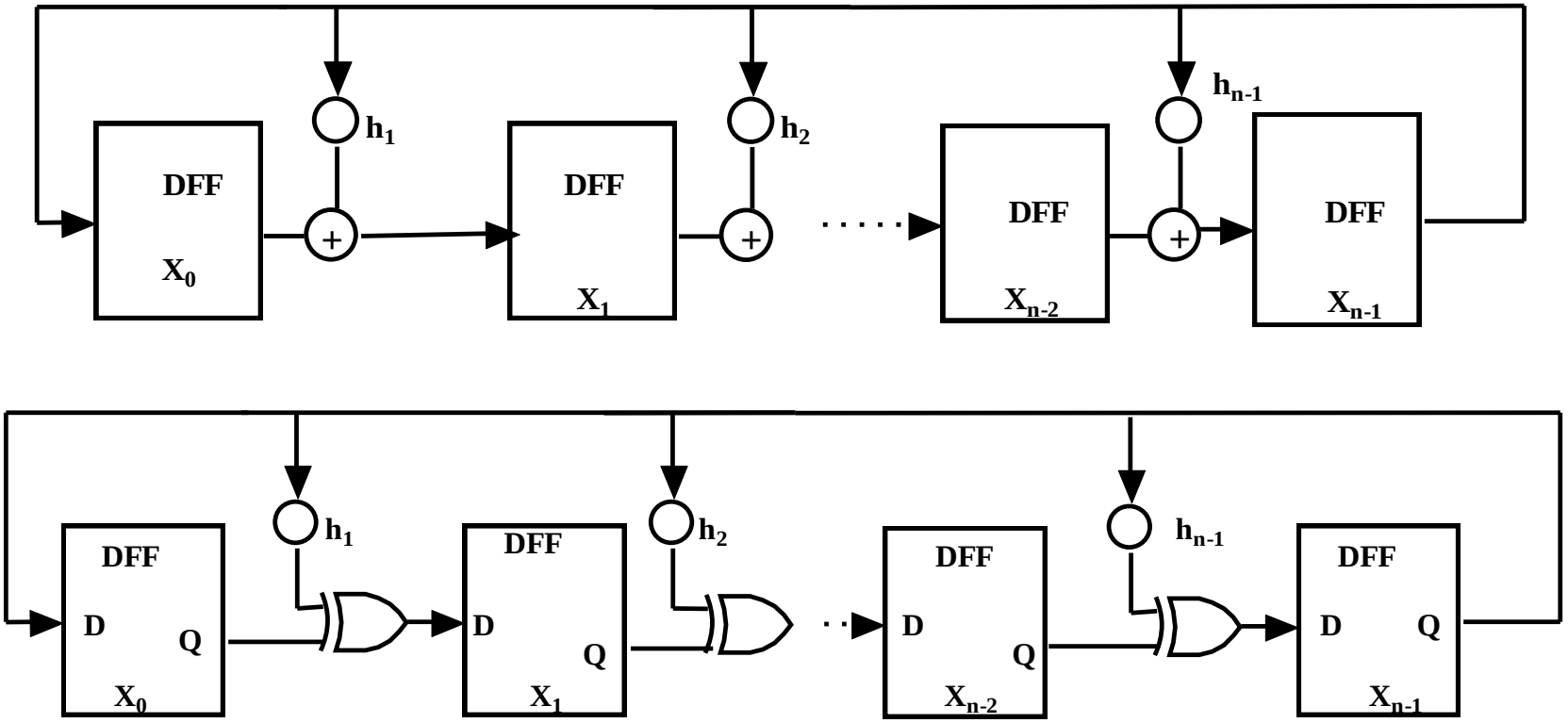
Design Verification and Test of Digital VLSI Circuits NPTEL Video Course

Module-XI

Lecture-II

Built in Self Test

Modular LFSRs



.The difference in modular LFSR compared to standard LFSR is due to the positions of the XOR gates in the feedback function; in modular LFSR XOR gates are in between adjacent flip-flops. Modular LFSR works faster than standard LFSR, because it has at most one XOR gate between adjacent flip-flops, while there can be several levels of XOR gates in the feedback of standard LFSR.

Modular LFSRs

In modular LSFR the output of any flip-flop may or may not participate in the XOR function; if output of any flip-flop X_i say, provides input to the XOR gate which feeds the input of flip-flop X_{i+1} then corresponding tap point h_i is 1. In the circuit representation of h_i=1, then there is an XOR gate from output of flip-flop X_i to input of flip-flop X_{i+1}; else output of flip-flop X_i is directly fed to input of flip-flop X_{i+1}.

$$\begin{bmatrix} X_0(t+1) \\ X_1(t+1) \\ X_2(t+1) \\ \cdots\cdots\cdots\cdots \\ X_{n-2}(t+1) \\ X_{n-1}(t+1) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0.....0 & 1 \\ 1 & 0 & 0.....0 & h_1 \\ 0 & 1 & 0.....0 & h_2 \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ 0 & 0 & 0.....0 & h_{n-2} \\ 0 & 0 & 0.....1 & h_{n-1} \end{bmatrix} \begin{bmatrix} X_0(t) \\ X_1(t) \\ X_2(t) \\ \cdots\cdots\cdots\cdots \\ X_{n-2}(t) \\ X_{n-1}(t) \end{bmatrix}$$

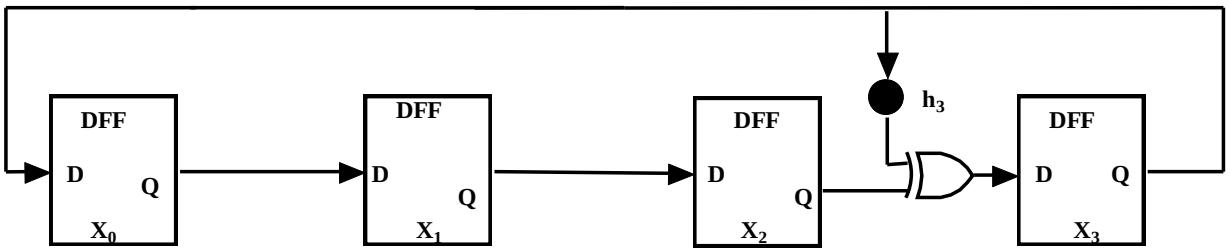
Modular LFSRs

This LFSR given the matrix can be written as $X(t+1) = T_S X(t)$. In this case $X_0(t+1) = X_{n-1}(t)$, which implies that X_{n-1} directly feedbacks X_0. $X_1(t+1) = X_0(t) + h_1 X_{n-1}(t)$, which implies that depending on h_1 =0 (or 1), input to X_1 is X_0 (or X_0 XORed with output of X_{n-1}). Similar logic holds for inputs to all flip-flops from X_1 to X_{n-1}.

This LFSR can also be described by the characteristic polynomial:

$$f(x) = 1 + h_1 x + h_2 x^2 + \ldots h_{n-2} x^{n-2} + h_{n-1} x^{n-1} + x^n$$

Modular LFSRs: Example



$$\begin{bmatrix} X_0(t+1) \\ X_1(t+1) \\ X_2(t+1) \\ X_3(t+1) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} X_0(t) \\ X_1(t) \\ X_2(t) \\ X_3(t) \end{bmatrix}$$

$$\begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{bmatrix} \begin{matrix} \\ \\ :: \\ \\ \end{matrix} \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

The characteristic polynomial of the LFSR is $f(x) = 1 + x^3 + x^4$.

If the initial values of the flip-flops are $X_0 = 1, X_1 = 0, X_2 = 0, X_3 = 0$

Modular LFSRs: Example

Now, the question arises, whether any LFSR would generate all $2^n - 1$ patters? The answer is no. Only for a few characteristic polynomials the LFSR is maximal length; such polynomials are called primitive polynomials (List of such polynomials can be found in Bardell et al.

P. H. Bardell, W. H. McAnney, and J. Savir, Built-In Test for VLSI: Pseudorandom Techniques. New York: John Wiley and Sons, Inc., 1987.

Hardware response compactor

•Expected output (i.e., golden response) of the CUT cannot be sorted explicitly in a memory and compared with response obtained from the CUT.

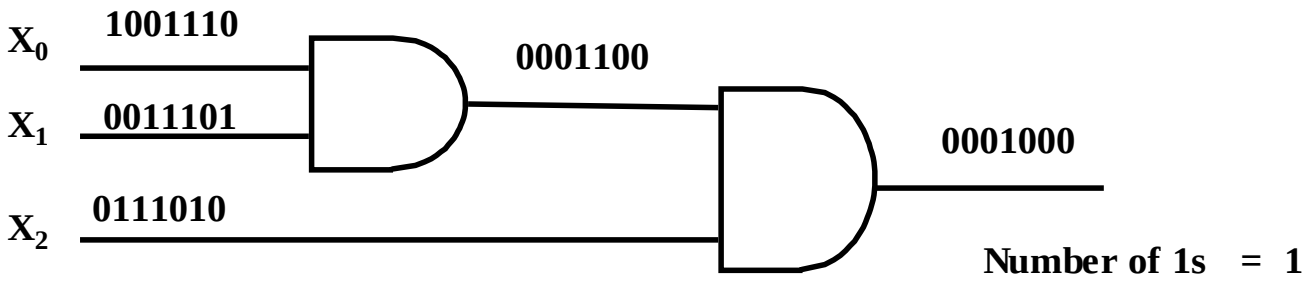
In other words, in BIST, it is necessary to compress the large number of CUT responses to a manageable size that can be stored in a memory and compared. In response compaction, sometimes it may happen that the compacted response of the CUT under normal and failure conditions are same. This is called aliasing during compaction.

•Simple techniques to compress CUT responses namely
(i) number of 1s in the output and
(ii) transition count at the output.

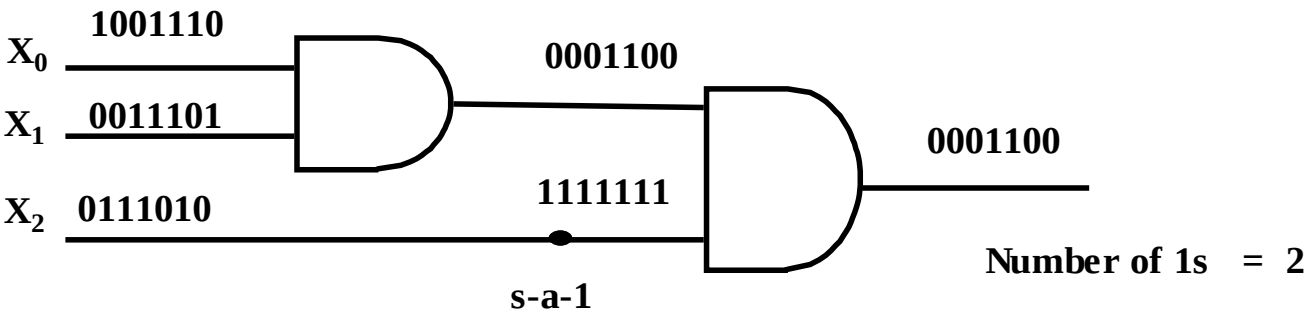
•Other complex techniques like LFSR based compaction, multiple input signature register based compaction, built-in logic observer based compaction etc.

Number of 1s compaction

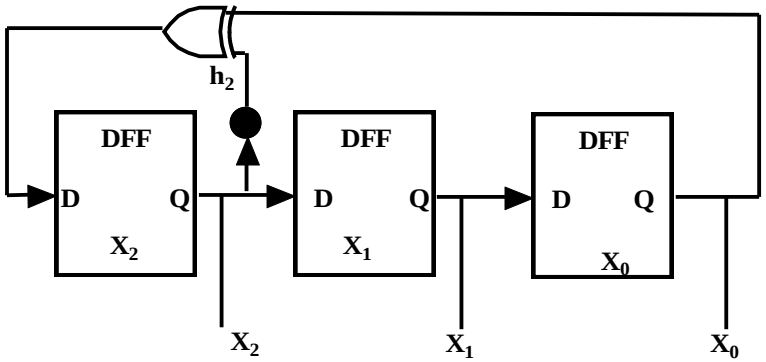
•Number of 1s compaction, is a very simple technique where we count the number of ones in the output responses from the CUT.



(a) Normal Circuit



(b) Circuit with s-a-1 fault



$$\begin{bmatrix} X_0 \\ X_1 \\ X_2 \end{bmatrix} ::: \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

Number of 1s compaction

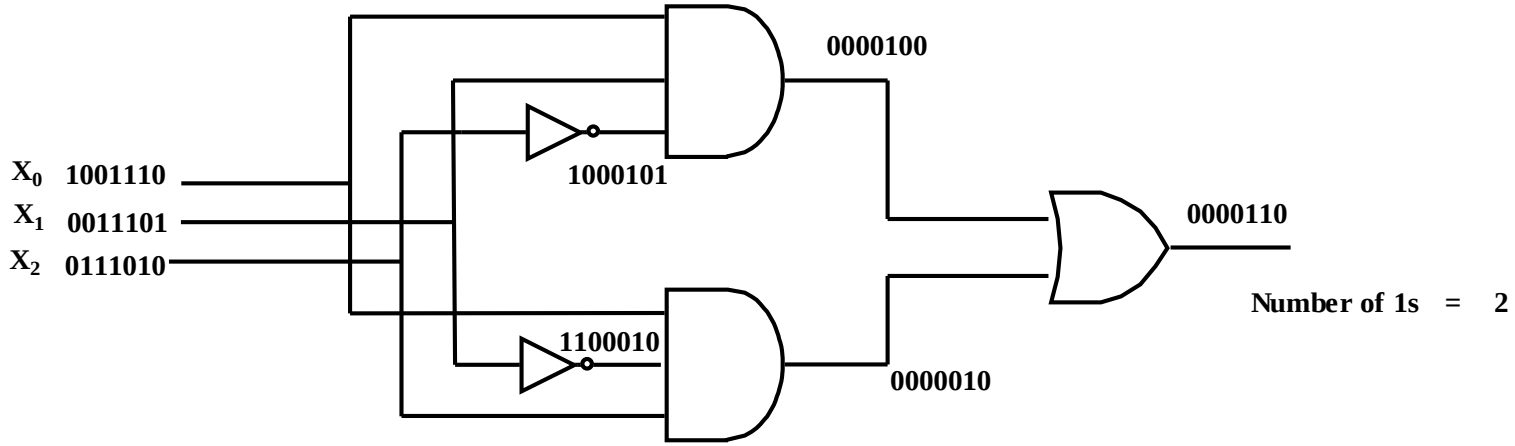
•It may be noted that 7 patterns (non-similar) were generated by the LFSR, which when given as input to the CUT generates output as 0001000, making number of 1s as 1.

•The same circuit with s-a-1 fault--When the same inputs are given to the CUT the output is 0001100, making number of 1s as 2.

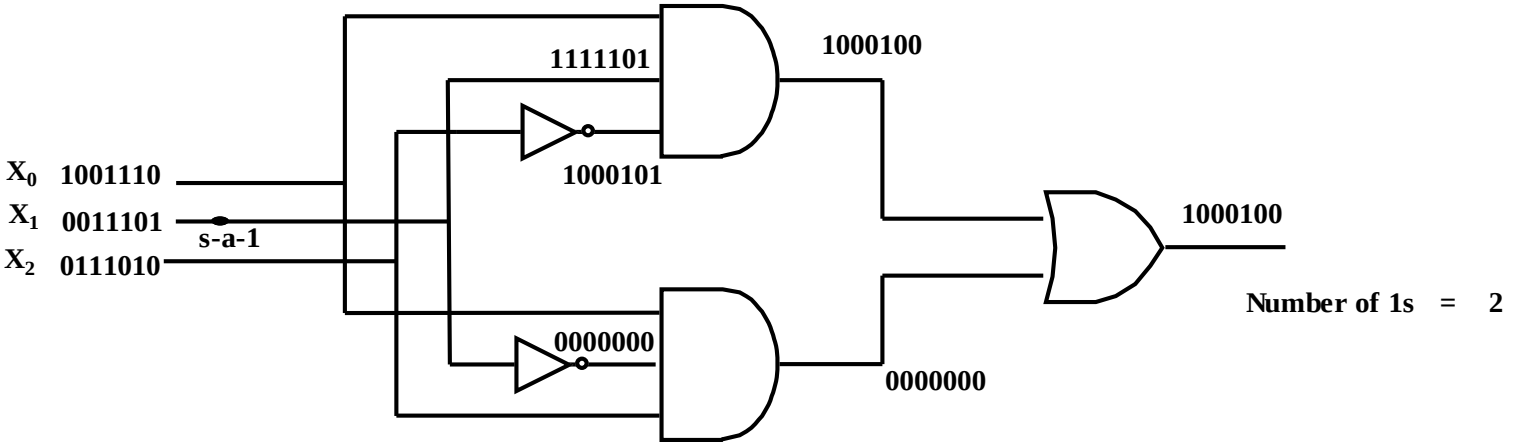
•So fault can be detected by the compaction as there is difference in number of 1s at the output of the CUT for the given input patterns.

•In other words, for the input patterns (from the LFSR), “number of 1s” based compaction is not aliasing. It may be noted that corresponding to the input patters, value of 1 is stored (as golden signature) in the memory which is compared with compacted response of the CUT.

Number of 1s compaction



(a) Normal Circuit



(b) Circuit with s-a-fault

Number of 1s compaction

The CUT generates output as 0000110, making number of 1s as 2.

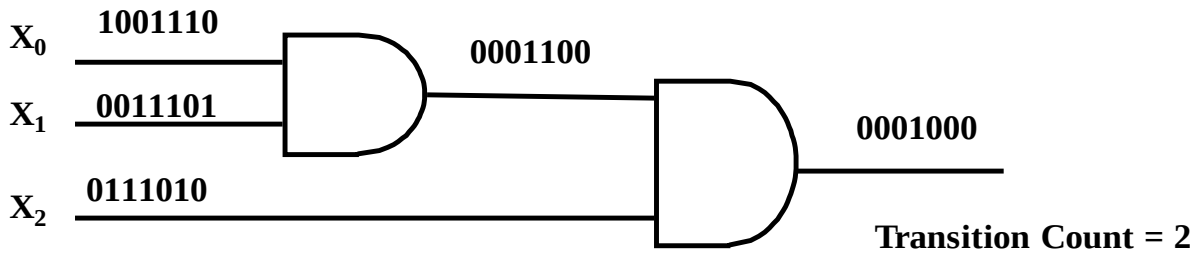
The same circuit with s-a-1 fault--When the same inputs are given to the CUT the output is 1000100, making number of 1s as 2.

So fault cannot be detected by the compaction; the number of 1s at the output of the CUT for the given input patterns is same under normal and s-a-1 conditions. In other words, for the input patterns (from the LFSR), “number of 1s” based compaction is aliasing.

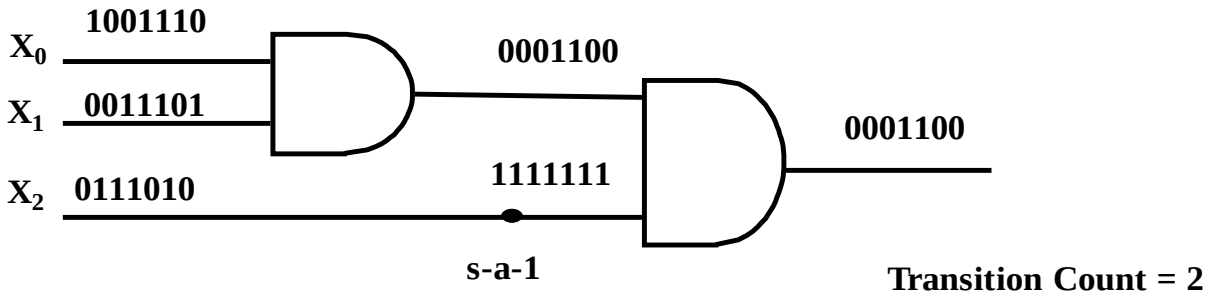
Transition count response compaction

•In this method of response compaction the number of transitions from 0 to 1 and 1 to 0 at outputs of the CUT are counted.

•The CUT generates output as 0001000, making transition count as 2; in the output sequence there is a transition from 0 to 1 and then from 1 to 0.
•The same circuit with s-a-1 fault. When the same inputs are given to the CUT the output is 0001100, making transition count as 2.
•So fault cannot be detected by the compaction.

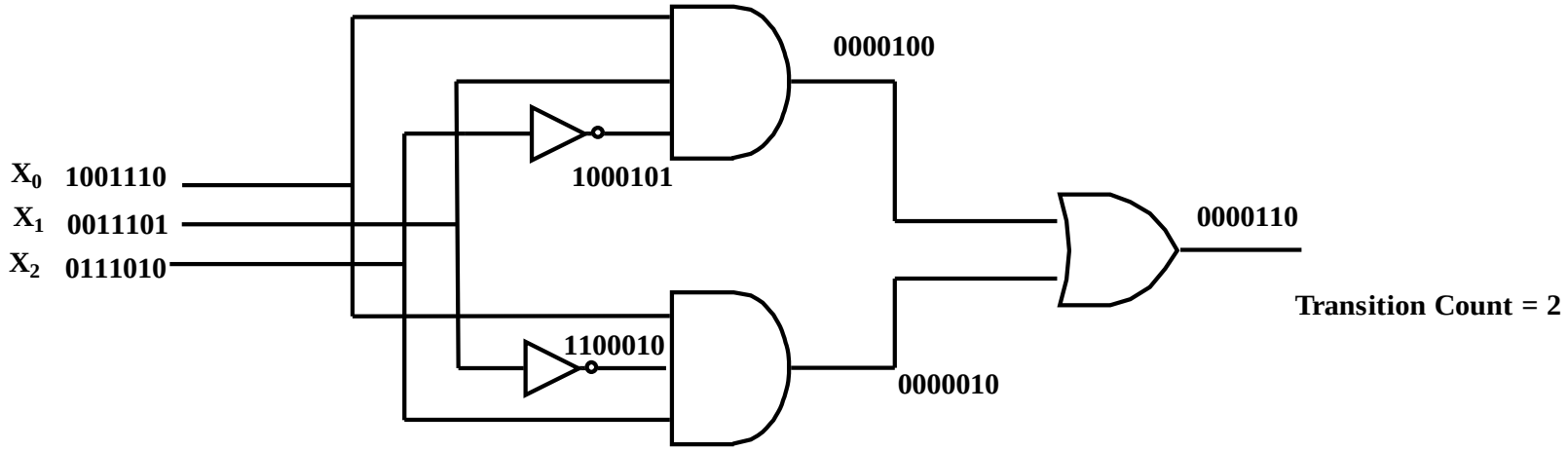


(a) Normal Circuit

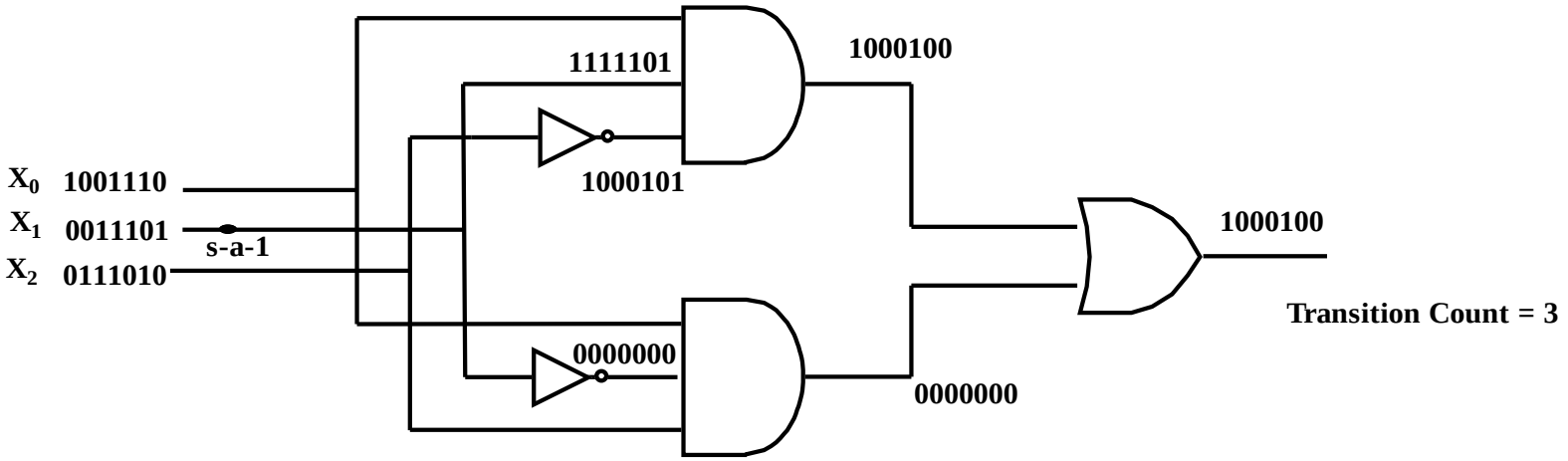


(a) Circuit with s-a-1 fault

Transition count response compaction



(a) Normal Circuit



(b) Circuit with s-a-fault

Questions and Answers

Question: In BIST how do we know which LFSR is to be used pattern generation?

Answer: Firstly, (obviously) the width of the LFSR is same as that of the number of inputs of the circuit.

Secondly, the characteristic polynomial of the LFSR should be primitive polynomial. This ensures that the LFSR generates all possible patterns (except all 0s).

Thirdly, the seed will decide the sequence of patterns. By fault simulation the test patterns can be decided.

Following that a proper compaction technique with minimal aliasing property for the circuit needs to be chosen. Then the seed is to be chosen in a manner so that the required test patterns among all the patterns generated by the LFSR should be generated initially. In other words, the seed should be chosen such that patterns of LFSR which do not test any faults (or any new faults compared to previous patterns) should be generated at latter phases. When all the required patterns have been generated and all faults have been tested the LFSR can be stopped (without requirement to generate the full cycle of patterns).

Questions and Answers

Question: Why LFSR cannot have all 0 state?

Answer:
If the seed is all 0 state, then the LFSR will be stuck at all 0 state as the feedback logic is XOR gates.

Thank You

Design Verification and Test of Digital VLSI Circuits NPTEL Video Course

Module-XI

Lecture-III

Memory Testing

Introduction

•VLSI testing, only from the context where the circuit is composed of logic gates and flip-flops.

• However, memory blocks form a very important part of digital circuits but are not composed of logic gates and flip-flops. This necessitates different fault models and test techniques for memory blocks.

•In memory technology, the capacity quadruples roughly every 3 years, which leads to decrease in memory price per bit (being stored).

•High storage capacity is obtained by raise in density, which implies decrease in the size of circuit (capacitor) used to store a bit. Experiments with new materials having high dielectric constant like barium strontium titanate are being done that facilitate greater capacitance to be maintained in the same physical space.

•Further, for faster access of the memory, various methods are being developed which includes fast page mode (FP), extended data output (EDO), synchronous DRAM (SDRAM), double data rate etc.

Introduction

•Unlike general circuits we generally do not discard faulty memory chips.

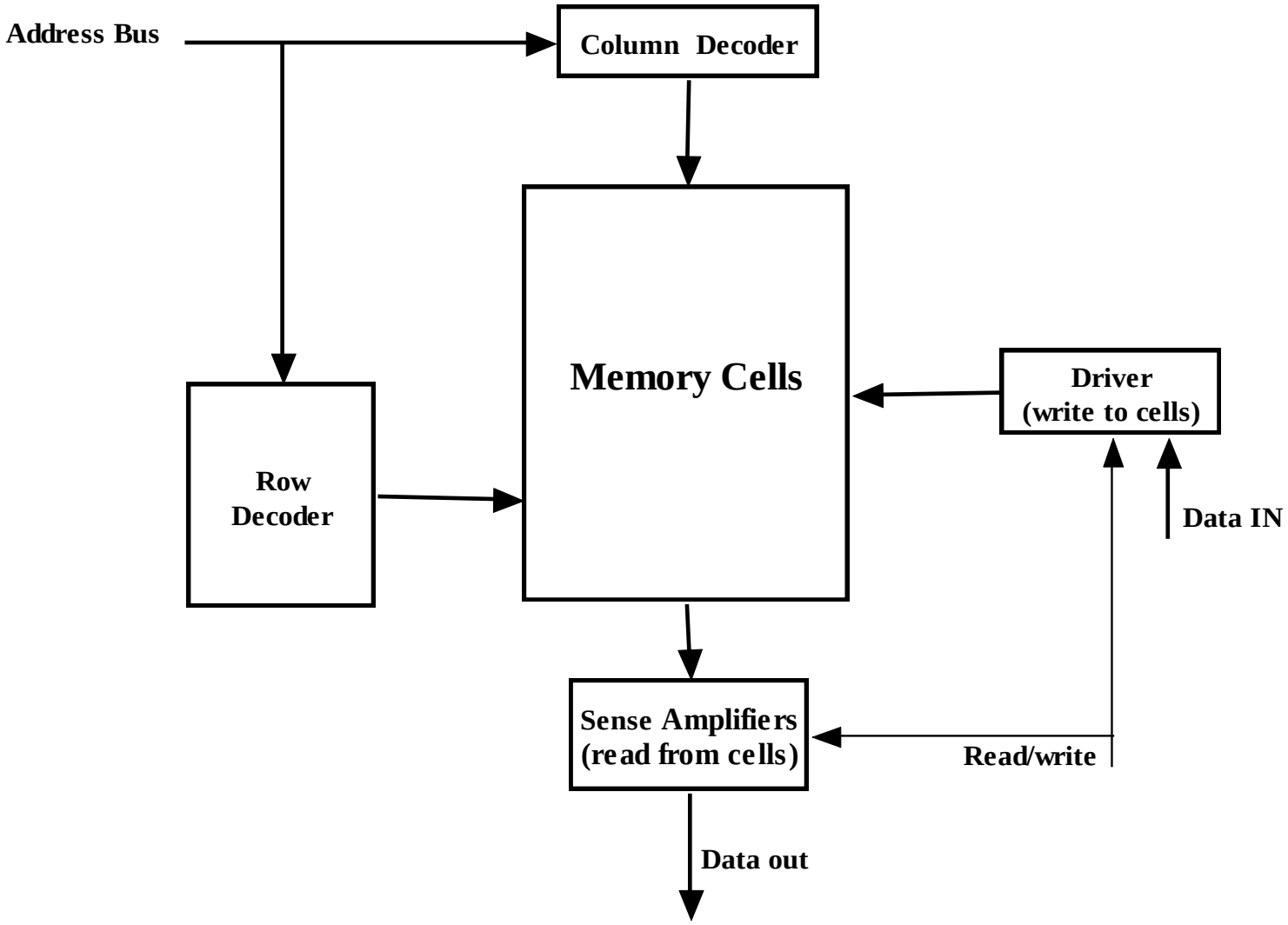
•Multiple faults will be present in any memory chip. The yield of memory chips would be nearly 0%, since every chip has defects. During manufacturing test, the faults are not only to be detected but also their locations (in terms of cell number) are to be diagnosed.

•As almost all memories will have faults in some cells, there are redundant (extra) cells in the memory. One a fault is diagnosed, the corresponding cell is disconnected and a new fault free cell is connected in the appropriate position. This replacement is achieved by blowing fuses (using laser) to reroute defective cells to normal spare cells.

•The sole functionality of a cell is to sore a bit information which is implemented using a capacitor; when the capacitor is charged it represents 1 and when there is no charge it represents 0. No logic gates are involved in a memory. Use of logic gates (in flip-flops) instead of capacitors to store bit information would lead to a very large area.

•The above two points basically differentiate testing of logic gate circuits from memory.
•New fault models and test procedures are required for testing memories. In this lecture we will study the most widely used fault models and test techniques for such fault models in memories.

Memory fault models



Memory fault models

- When data is to be read from the memory, first the row and column decoders determine the location (i.e., the cell) from the address (sent in the address bus) that needs to be accessed.

- Based on the address in the row and column decoders the cell of the appropriate row and column gets connected to the sense amplifier, which sends the data out.

- Similar situation (for accessing the required cells) holds when data is to be written in the memory, however, in case of writing, special driver circuitry writes the values in the cells from the data bus.

It may be noted that from the testing perspective we would only check if

- Required value (0/1) can be written to a cell
- The stored value can be read from a cell
- The proper cell is accessed, i.e., the row and column decoder do not have faults.

Memory fault models

The row and column decoders are digital circuits implemented using logic gates (which are different from memory cell implementation).

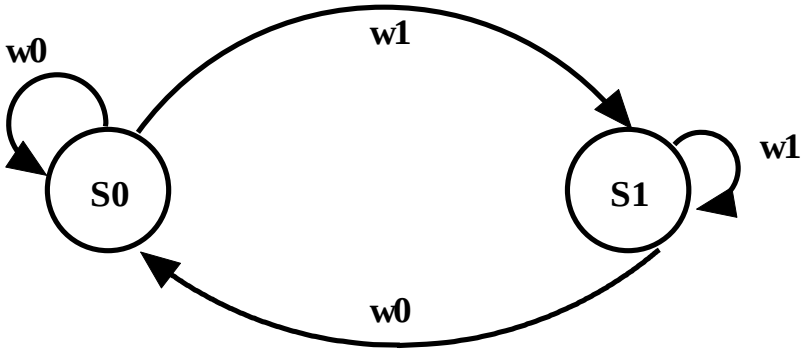
The sense amplifier and driver are analog circuits.

In testing of memory, we do not consider the decoders as gate level digital circuits nor the sense amplifier and driver as analog circuits. For the decoders, we test the functionality whether they can access the desired cells based on the address in the address bus. For the amplifier and driver we check if they can pass the values to and from the cells correctly.

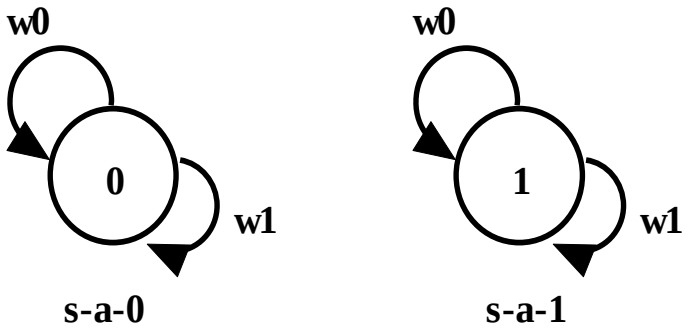
The following faults called “reduced functional faults” are sufficient for functional memory testing

- Stuck-at fault
- Transition fault
- Coupling fault
- Neighborhood pattern sensitive fault
- Address decoder faults

Stuck-at fault



State diagram for a good memory cell

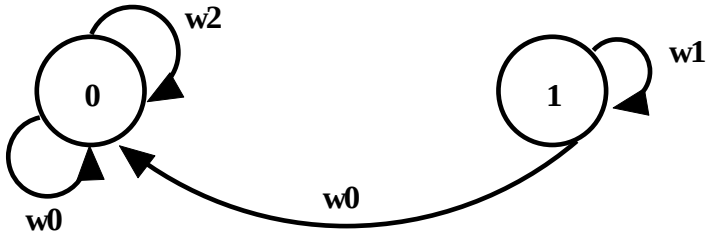


State diagram for a s-a1 memory cell and s-a-0 memory cell

Stuck-at-fault in memory is the one in which the logic value of a cell (or line in the sense amplifier or driver) is always 0 or always 1.

S0 is the state where the cell contains 0, while is S1 is the state where it contains 1. w1 (w0) indicates value of (0) 1 being written.

Transition fault



State diagram for up transition fault in a memory cell

In transition fault a cell fails to make a (0 to 1) transition or a (1 to 0) transition when it is written; up transition fault is denoted as $\langle \uparrow | 0 \rangle$ and a down transition fault is denoted as $\langle \downarrow | 1 \rangle$. A cell having up transition fault is same a s-a-0 fault, however, the cell can take and retain value 1 if a 0 has not yet been written to the cell. The dual happens for down transition fault.

Coupling Faults:

Coupling fault, as the name suggests, implies deviation from normal behavior of a cell because of coupling with others.

As there can be exponential number of combinations of coupling of a cell with others cells, we assume that in coupling faults a faulty cell can get coupled with another faulty cell.

In other words, in the widely used coupling fault model it is assumed that any “two” cells can couple and normal behavior changes in these two cells; it is called 2-coupling fault model.

So if there are n cells in a memory then there can be ${}^{n}C_{2}$ number of 2-coupling faults.

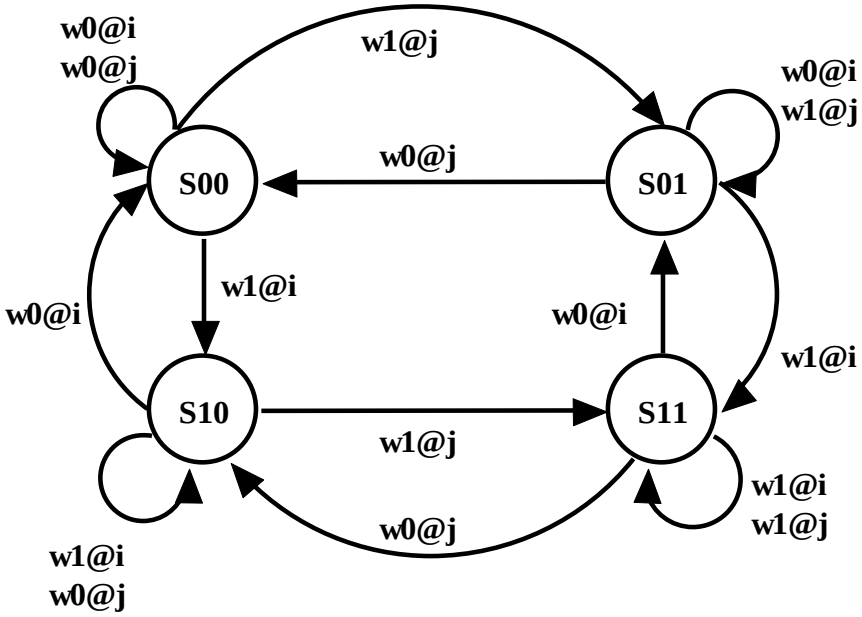
To reduce the number of 2-coupling faults further from ${}^{n}C_{2}$, we assume that only neighboring cells (decided on threshold distance) can be involved in the fault. We consider two types of coupling faults namely, (i) inversion coupling faults and (ii) idempotent coupling faults.

Inversion coupling faults

In a 2-inversion coupling fault $cfinv_{i,j}$ say, involving cells i and j, a transition (0 to1 or 1 to 0) in memory cell j causes an unwanted change in memory cell i. Memory cell i is the *coupled* cell (where fault occurs) and memory cell j is the *coupling* cell. The two possible 2-inversion coupling faults involving cells i, j (denoted as $cfinv_{i,j}$) are

- Rising: $\langle \uparrow | \updownarrow \rangle$ (implying 0 to1 change in cell j complements the content of cell i)
- Falling: $\langle \downarrow | \updownarrow \rangle$ (implying 1 to 0 change in cell j complements the content of cell i)

Inversion coupling faults



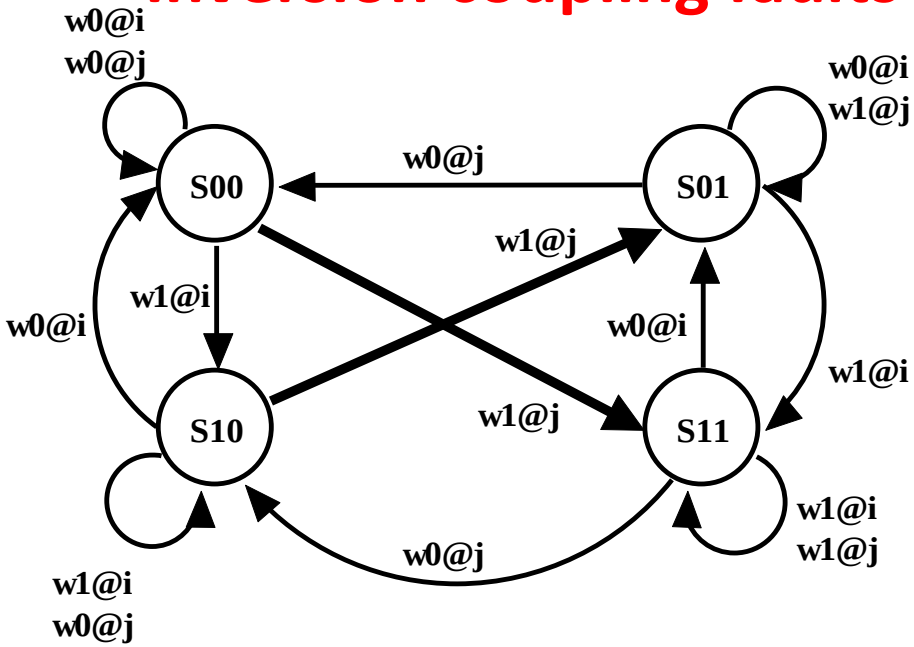
The state diagram for two cells i and j under normal condition.

State $S00$ implies that both the cells have 0 values

The self loop at state $S00$, marked w0@i implies that if 0 is written to cell i then the same state is retained; another transition w0@j is associated with the same self loop which implies that if 0 is written to cell j then $S00$ retained.

If we write 1 to cell j (from state $S00$) i.e., w1@j, we go to state $S01$; this is indicated by the transition from $S00$ to $S01$ marked w1@j.

Inversion coupling faults



State machine for two cells i and j under rising inversion coupling fault $cfinv_{i,j}$.

Under normal condition if we write 1 to cell j (from state $S00$) we go to state $S01$, however, under rising $cfinv_{i,j}$ we go to state $S11$.

When we write 1 to cell j in $S00$ it makes a 0 to 1 (rising) transition. As j is the coupling cell it complements the value of the coupled cell i (from 0 to 1).

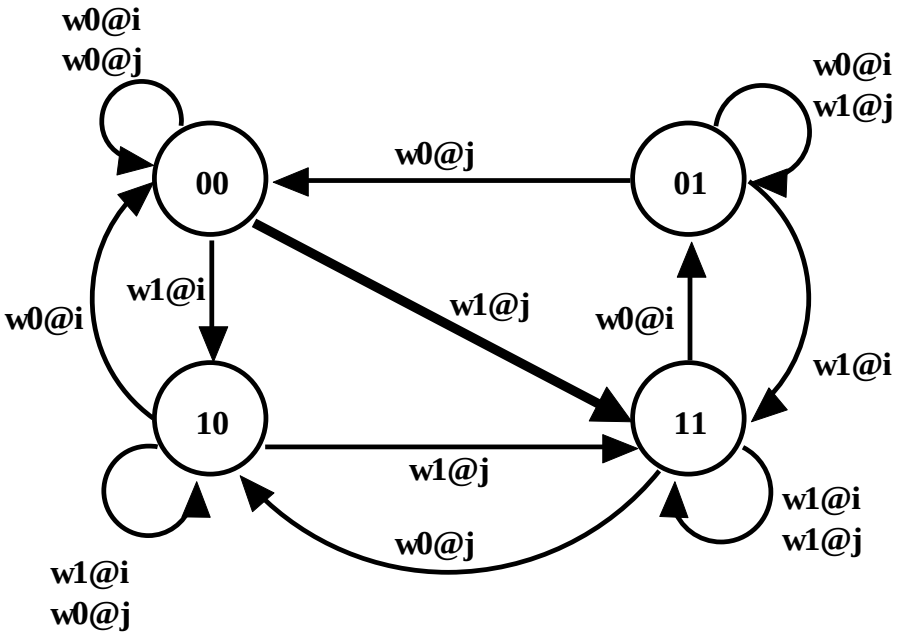
The faulty transitions are indicated by thick arrows; $S00$ to $S11$ and $S10$ to $S01$ are two such transitions.

Idempotent coupling faults

In a 2-indempotent coupling fault $cfid_{i,j}$ say, involving cells i and j, a transition (0 to1 or 1 to 0) in memory cell j sets the value in memory cell i to be 0 or 1. The four possible 2- idempotent coupling faults involving cells i, j (denoted as $cfid_{i,j}$) are

- Rising-0: $\langle \uparrow | 0 \rangle$ (0 to1 change in cell j sets the content of cell i to be 0)
- Rising-1: $\langle \uparrow | 1 \rangle$ (0 to1 change in cell j sets the content of cell i to be 1)
- Falling-0: $\langle \downarrow | 0 \rangle$ (1 to 0 change in cell j sets the content of cell i to be 0)
- Falling-1: $\langle \downarrow | 1 \rangle$ (1 to 0 change in cell j sets the content of cell i to be 1)

Idempotent coupling faults



The state machine for two cells i and j under rising-1 idempotent coupling fault $cfid_{i,j}$.

We note that under normal condition if we write 1 to cell j (from state $S00$) we go to state $S01$, however, under rising-1 $cfid_{i,j}$ we go to state $S11$. This situation is similar to ↑ inverse coupling fault

However, unlike ↑ inverse coupling fault, in rising-1 idempotent coupling fault we do not have a faulty transition from $S10$ to $S01$.

Bridging fault

A bridging fault is a short circuit between two or more cells. As in the case of coupling faults, to keep the number of faults within a practical number, it is assumed that only two cells can be involved in a bridging fault. There are two types of bridging faults

- AND bridging fault $ANDbf_{i,j}$ (involving cells i and j) which results in values in cells i and j to be logic AND of the values in these cells under normal condition. AND bridging fault is represented by $\langle v_i, v_j \mid v_i ANDv_j, v_i ANDv_j \rangle$ where the first two places represent the values in cells i and j under normal condition and the two values following "|"represent the values in cells i and j under AND bridging fault. $\langle 0,0|0,0\rangle, \langle 0,1|0,0\rangle, \langle 1,0|0,0\rangle, \langle 1,1|1,1\rangle$ are the four types of AND bridging faults possible.

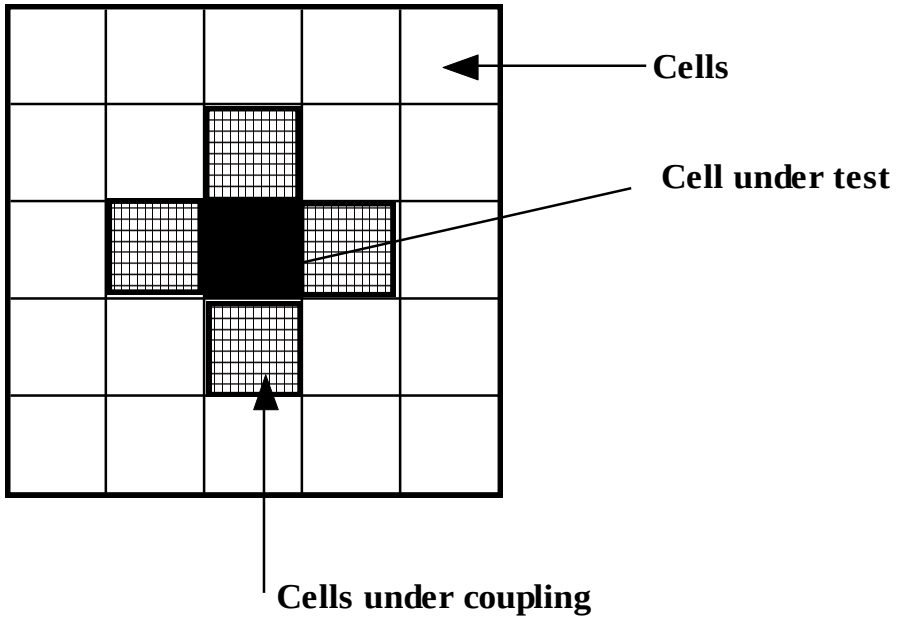
Bridging fault

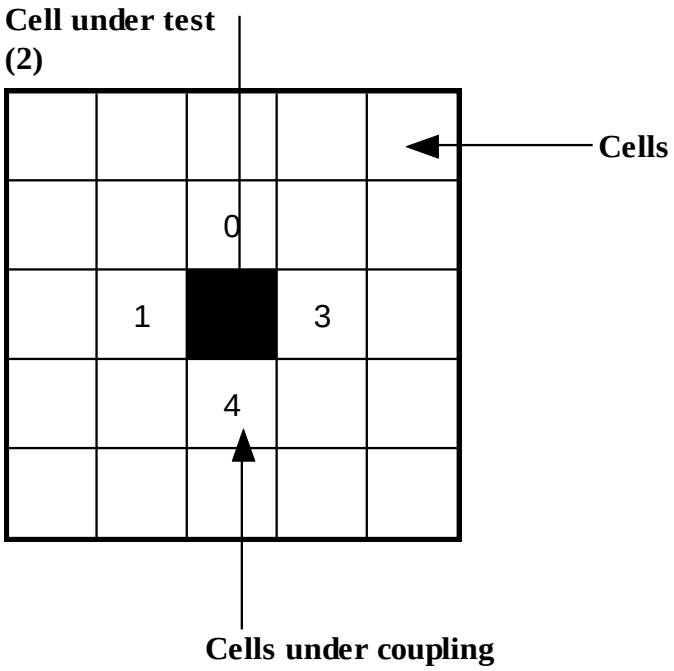
- OR bridging fault $ORbf_{i,j}$ (involving cells i and j) which results in values in cells i and j to be logic OR of the values in these cells under normal condition. $\langle 0,0|0,0\rangle, \langle 0,1|1,1\rangle, \langle 1,0|1,1\rangle, \langle 1,1|1,1\rangle$ are the four types of OR bridging faults possible.

Neighborhood pattern sensitive coupling faults

One of the most important and different kind of fault in memory compared logic gate circuits is neighborhood pattern sensitive faults (NPSFs). As memory cells are very close to each other, the cells behave normally except for certain patterns in the neighborhood cells. For example, if a cell i has 0 and all the neighboring cells have 1, then the value of cell i may be pulled up to 1. It is obvious that given a cell there can be infinite number of neighborhood combinations. However for all practical cases there are two types of neighborhoods used in fault modeling for the cell under test.

Neighborhood pattern sensitive coupling faults

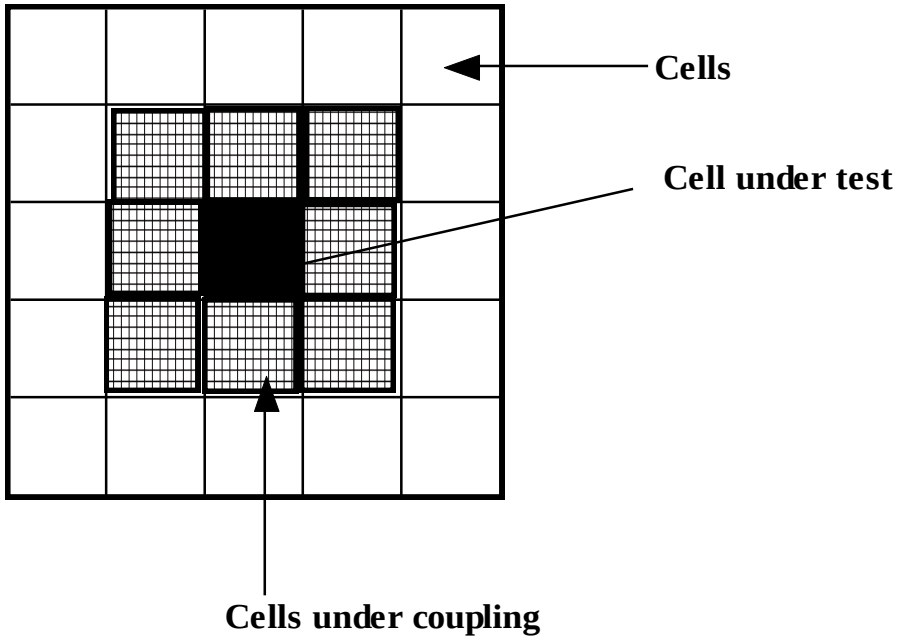


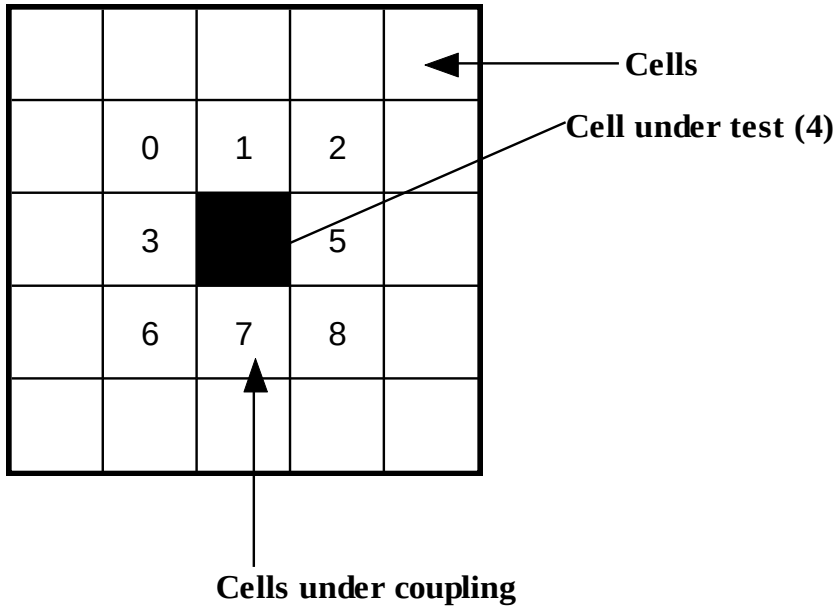


Type-1 neighborhood

The black colored cell is the one under test and the four cells around it (filled by small check boxes) are called neighborhood cells. Patterns in the neighborhood cells cause faults in the cell under test.

Neighborhood pattern sensitive coupling faults





Type-2 neighborhood

Complex than Type-1 neighborhood

Neighborhood pattern sensitive coupling faults

- Active NPSF (ANPSF)

The value in the cell under test changes due to a change in ONE cell of the neighborhood (type-1 or type-2 depending on the one being used); all other cells of the neighborhood make a pattern. An ANPSF is represented as "$vcut\ \langle v_0, v_1, v_3, v_4 \mid fe \rangle$", where $vcut$ is the value in the cell under test, v_0, v_1, v_3, v_4 represent the values in the neighboring cells (at cell no. 0,1,3,4 respectively) including the one which changes and fe represents fault effect in the cell under test. For example, $1\langle 0, \downarrow, 0, 0 \mid 0 \rangle$ represents the ANPSF were the cell under test initially has value of 1, the pattern made by neighboring cells is 0000 (values at cell no. 0,1,3,4 respectively) and fault effect at cell under test is 0 when a 1 to 0 transition in made in cell 1.

Neighborhood pattern sensitive coupling faults

- Passive NPSF (PNPSF)

  PNPSF implies that a certain neighborhood pattern prevents the cell under test from changing its value. An PNPSF is represented as $vcut\ \langle v_0, v_1, v_3, v_4 \mid fe \rangle$, where $vcut$ is the value in the cell under test, v_0, v_1, v_3, v_4 represent the values in the neighboring cells and fe represents fault effect in the cell under test. There can be three types of fe PNPSF:

  - $\uparrow \mid 0$: cell under test cannot be changed from 0 to 1 (initial value of cell under test is 0)
  - $\downarrow \mid 1$: cell under test cannot be changed from 1 to 0 (initial value of cell under test is 1)
  - $\updownarrow \mid x$: cell under test cannot be changed regardless of content.

Address decoder faults

From the context of memory testing four types of faults are considered in address decoder (for both reading and writing)

- No cell is accessed for a certain address
- No address can access a certain cell
- With a particular address, multiple cells are simultaneously accessed
- A particular cell can be accessed with multiple addresses.

Testing of memory faults

“March Test” which is used widely for memory testing.
March testing basically involves applying (writing and reading) patterns to each cell in memory before proceeding to the next cell and if a specific pattern is applied to one cell, then it must be applied to all cells. This is either done in increasing memory address order or decreasing order.

Match test basically involves the following steps:

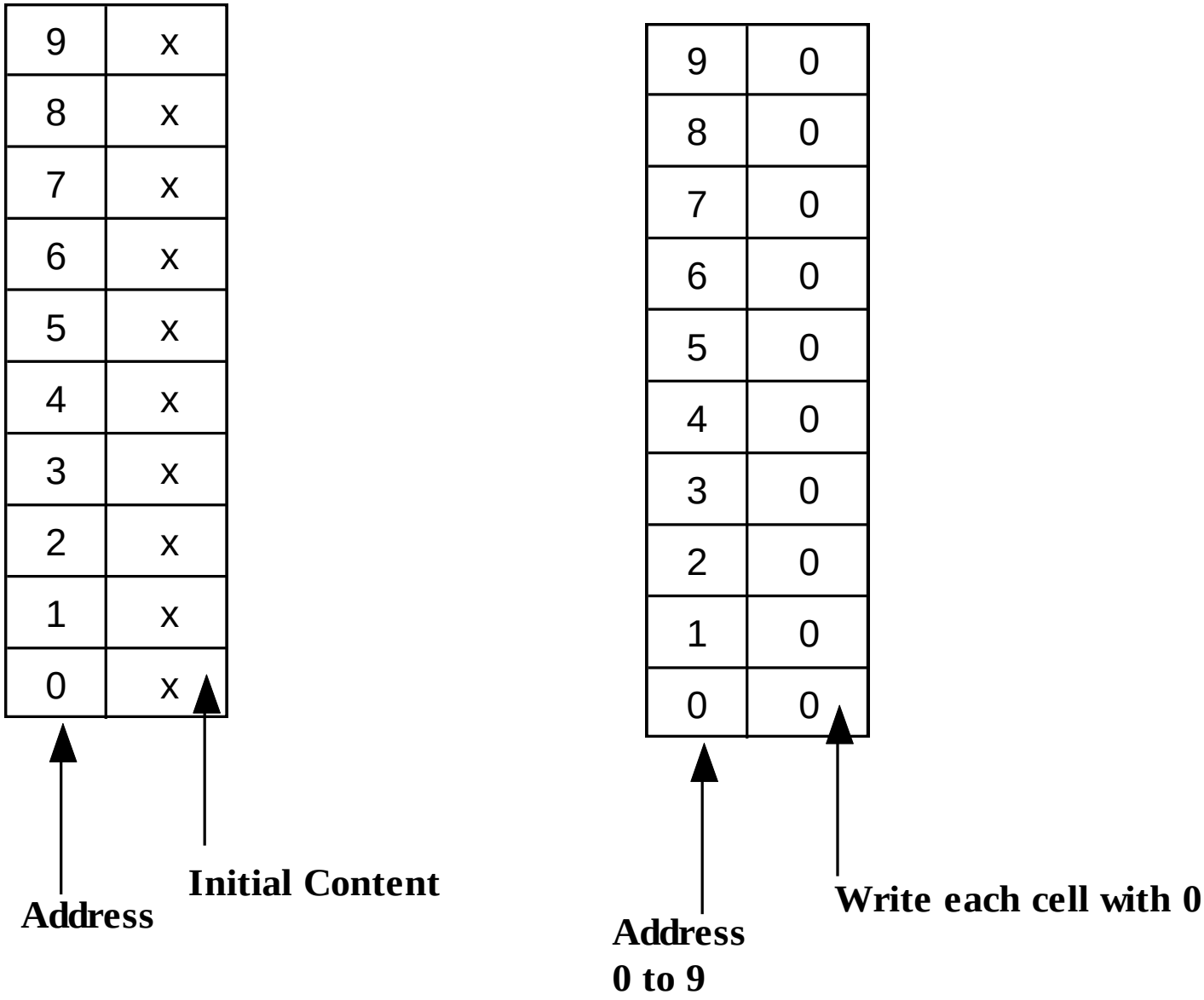
1. In increasing order of address of the memory cells, write 0s to the cells;

1. In decreasing order of address of the memory cells, read the cells (expected value 0) and write 1 to the cells;

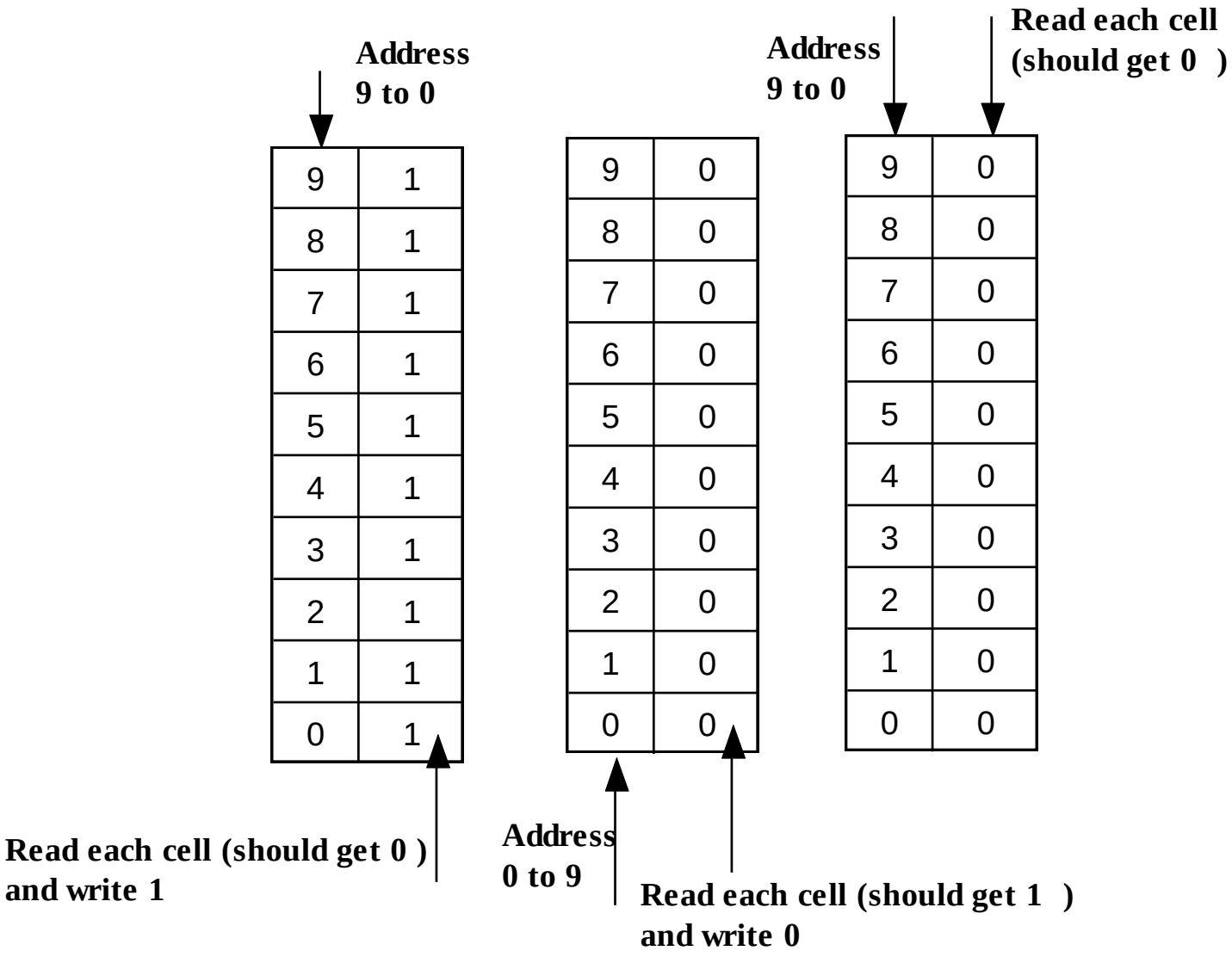
2. In increasing order of address of the memory cells, read the cells (expected value 1) and write 0 to the cells;

3. In decreasing order of address of the memory cells, read the cells (expected value 0);

Testing of memory faults



Testing of memory faults



March Test: Stuck at fault model

March test obviously tests s-a-0 and s-a-1 faults in the cells because 0 and 1 in each cell is written and read back.

March Test: Transition fault

In March test during Step 1 all cells are written with 0 and in Step 2 all cells are written with 1s, thereby making a 0 to 1 transition in the cells. In Step 2 it is verified if cells have 0 in them and in Step 3 it is verified if cells have 1, thereby verifying 0 to 1 transition in the cells. So, Step 1 through Step 3 tests absence of $\langle \uparrow | 0 \rangle$ fault. In a similar manner, Step 3 through Step 5 tests absence of $\langle \downarrow | 0 \rangle$ fault.

Thank you

Design Verification and Test of Digital VLSI Circuits NPTEL Video Course

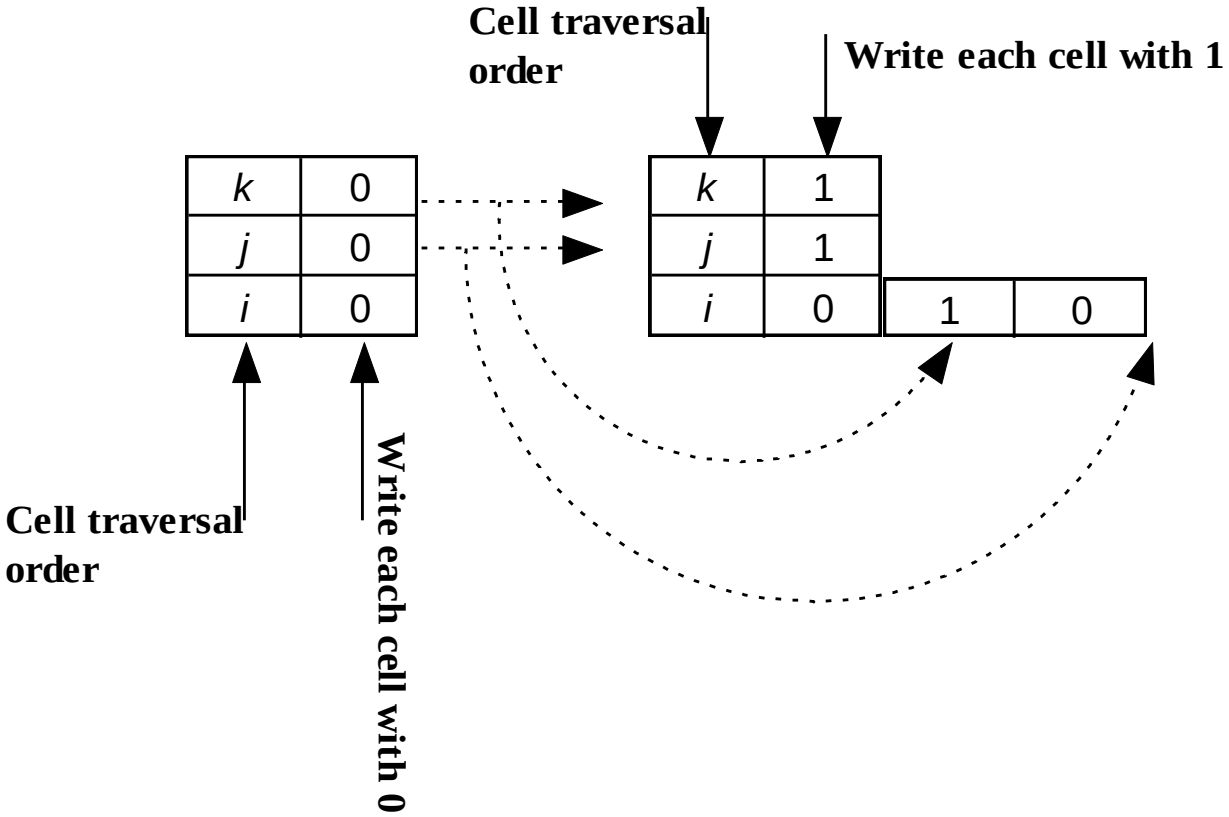
Module-XI

Lecture-IV

Memory Testing

March Test: Coupling Faults

March tests cannot detect all coupling faults. Let is consider three cells i, j, k such that address of $i <$ address of j and address of $j <$ address of k. Cell i is coupled with cell j and cell k by fault $\langle \uparrow | \updownarrow \rangle$; j and k are the coupling cell. In march test as we go either in ascending order or descending order of memory address of cells, both i and j are either visited before or after cell k.



March Test: Coupling Faults

As Step-1 of March test all the cells i, j, k are written with 0. Following that in Step 2, all the cells (in order of) k, j, i are written with 1 (after successful reading of 0 from the cells). It may be noted that first cell k is written with 1; as cell i is coupled with cell k having fault $\langle \uparrow | \updownarrow \rangle$, the 0 to 1 transition in cell k inverts the content of cell i. Following that, cell j is written with 1; as cell i is also coupled with cell j having fault $\langle \uparrow | \updownarrow \rangle$, the 0 to 1 transition in cell j inverts the content of cell i again. Now when cell i is read, the value determined is 0 which means absence of two coupling faults (i) rising $cfinv_{i,j}$ and (ii) rising $cfinv_{i,k}$. In other words, "rising $cfinv_{i,k}$" masks "rising $cfinv_{i,j}$".

March Test: Coupling Faults

Inverting rising coupling fault $\langle\uparrow|\updownarrow\rangle$ between cell i (*coupled cell*) and j (*coupling cell*): (i) Cell j is to be written with a 0 and read back, (ii) value at cell i is to be read and remembered, (iii) cell j is to be written with a 1 and read back, and (iv) value at cell i is to be read and checked that it is same as the one remembered (i.e., no inversion has happened).

Inverting falling coupling fault $\langle\downarrow|\updownarrow\rangle$ between cell i and j : (i) Cell j is to be written with a 1 and read back, (ii) value at cell i is to be read and remembered, (iii) cell j is to be written with a 0 and read back, and (iv) value at cell i is to be read and checked that it is same as the one remembered (i.e., no inversion has happened).

March Test: Coupling Faults

Idempotent Rising-0 coupling fault $\langle \uparrow | 0 \rangle$ between cell i and j : (i) Cell j is to be written with a 0 and read back, (ii) cell i is to be written with 1 and read back, (iii) cell j is to be written with a 1 and read back, and (iv) value at cell i is to be read and checked to be 1.

Idempotent Rising-1 coupling fault $\langle \uparrow | 1 \rangle$ between cell i and j : (i) Cell j is to be written with a 0 and read back, (ii) cell i is to be written with 0 and read back, (iii) cell j is to be written with a 1 and read back, and (iv) value at cell i is to be read and checked to be 0.

March Test: Coupling Faults

Idempotent Falling-0 coupling fault $\langle\downarrow|0\rangle$ between cell i and j: (i) Cell j is to be written with a 1 and read back, (ii) cell i is to be written with 1 and read back, (iii) cell j is to be written with a 0 and read back, and (iv) value at cell i is to be read and checked to be 1.

Idempotent Falling-1 coupling fault $\langle\downarrow|1\rangle$ between cell i and j: (i) Cell j is to be written with a 1 and read back, (ii) cell i is to be written with 0 and read back, (iii) cell j is to be written with a 0 and read back, and (iv) value at cell i is to be read and checked to be 0.

March Test: Bridging faults

Like coupling faults March tests cannot detect all bridging faults.

$\langle 0,0|0,0\rangle, \langle 0,1|0,0\rangle, \langle 1,0|0,0\rangle, \langle 1,1|1,1\rangle$ are the four types of AND bridging faults possible.

This implies that cells i, j which are involved in bridging faults must have the four combinations of inputs 00,01,10 and 11.

No cell pairs have all the four combinations 00,01,10 and 11. So to test bridging faults the following test pattern sequences are required.

March Test: Bridging faults

AND bridging fault $ANDbf_{i,j}$ (involving cells i and j):

(i) write 0 in cell i and 0 in cell j and read back the values (which must remain same),

(ii) write 0 in cell i and 1 in cell j and read back the values,

(iii) write 1 in cell i and 0 in cell j and read back the values, and

(iv) write 1 in cell i and 1 in cell j and read back the values.

It may be noted that the above four test pattern sequence are enough to test OR bridging fault also because we write all possible combinations in the two cells (involved in fault) and read back to check if they retain their values.

March Test: Address decoder faults

A little variation of March test can test all four address decoder faults. The test sequence (of modified March test) and that tests all four address decoder faults are as follows

- In increasing order of address of the memory cells, read the value of the memory cells and write complement value in the cell. If 1 is read at cell 0, value of 1 is written to cell 0; following that same procedure is followed for cell 2 and so on for entire memory.

- In decreasing order of address of the memory cells, read the cells (match with expected value) and write complement value in the cell.

The basic principle is that as the memory writing and examination operation moves through memory, any address decoder fault that causes unexpected accesses of memory locations will cause those locations to be written to an unexpected value. As the test proceeds, it will discover those locations and report a fault.

Basics of memory BIST

- For March test an address generator (increasing and decreasing order) and a data reader cum writer is required. So BIST for March test will be simply an LFSR and a data reader cum writer. As in the case of logic BIST, the LFSR should have primitive polynomial (so that it generates all numbers from 1 to 2^n), and along with this the LFSR for memory BIST should the following features

  - Be able to generate all the 0 pattern to access the last memory location
  - Be able to generate forward and reverse order patterns i.e., if 1-0-2-3 be the sequence of the LFSR (when initialized with 1) then there should be a mode to generate the sequence 1-3-2-0.

Basics of memory BIST

- March test can be modified by replacing “sequential read/write” with “arbitrary order read/write, but covering all cells” without loss in test capability.

- We illustrated cell traversal from 0 to 9 and then from 9 to 0.

- However, the test capability will not change if sequence of cell traversal is any other sequence, for example, 1-0-2-5-7-3-4-6-9-8 while moving in ascending order and 1-8-9-6-4-3-7-5-2-0 in reverse order.

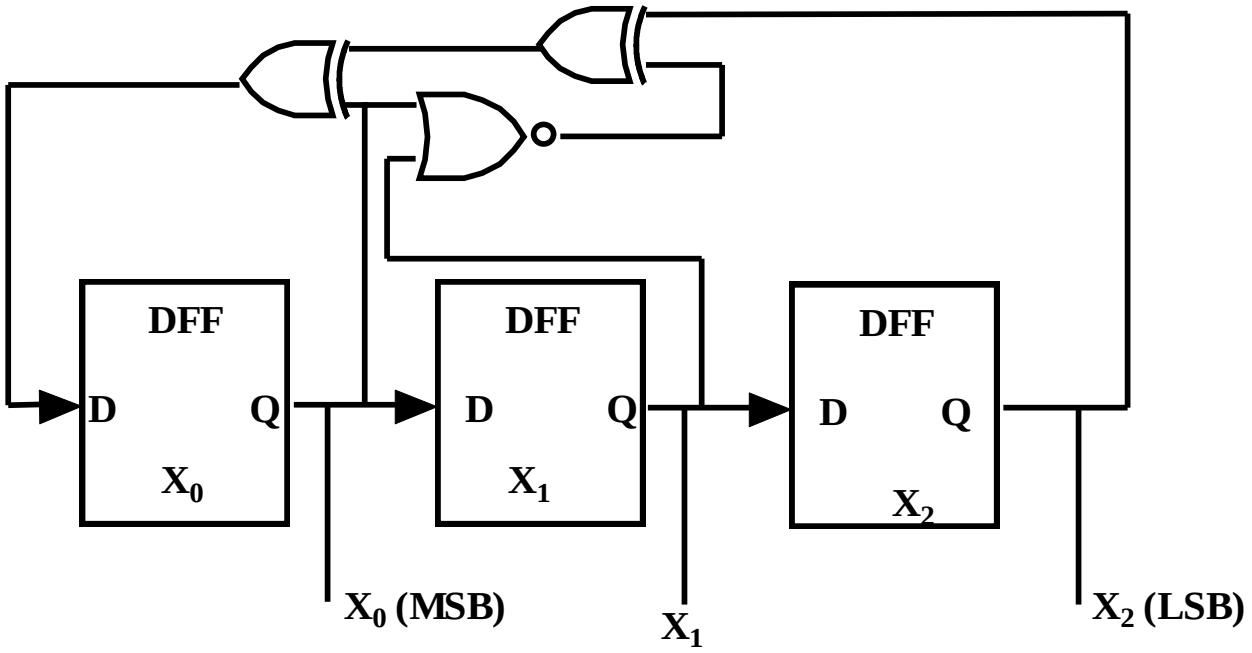
Basics of memory BIST

- March test can be modified by replacing “sequential read/write” with “arbitrary order read/write, but covering all cells” without loss in test capability.

- We illustrated cell traversal from 0 to 9 and then from 9 to 0.

- However, the test capability will not change if sequence of cell traversal is any other sequence, for example, 1-0-2-5-7-3-4-6-9-8 while moving in ascending order and 1-8-9-6-4-3-7-5-2-0 in reverse order.

Basics of memory BIST



$$\begin{bmatrix} X_0(MSB) \\ X_1 \\ X_2(LSB) \end{bmatrix} ::: \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{bmatrix}$$

Thank You