

## Unit V LOGIC CIRCUIT TESTING

### 1. Faults in logic circuits

A failure is said to have occurred in a logic circuit or system if it deviates from its specified behavior. A fault, on the other hand, refers to a physical defect in a circuit. For example, a short between two signal lines in the circuit or a break in a signal line is a physical defect. An error is usually the manifestation of a fault in the circuit; thus a fault may change the value of a signal in a circuit from 0 (correct) to 1 (erroneous) or vice versa. However, a fault does not always cause an error; in that case, the fault is considered to be latent.

A fault is characterized by its *nature*, *value*, *extent*, and *duration* [2]. The nature of a fault can be classified as *logical* or *nonlogical*. A logical fault causes the logic value at a point in a circuit to become opposite to the specified value. Nonlogical faults include the rest of the faults such as the malfunction of the clock signal, power failure, etc. The value of a logical fault at a point in the circuit indicates whether the fault creates fixed or varying erroneous logical values. The extent of a fault specifies whether the effect of the fault is localized or distributed. A local fault affects only a single variable, whereas a distributed fault affects more than one. A logical fault, for example, is a local fault, whereas the malfunction of the clock is a distributed fault. The duration of a fault refers to whether the fault is *permanent* or *temporary*.

#### 1. Stuck-At Fault

#### 2. Bridging Faults

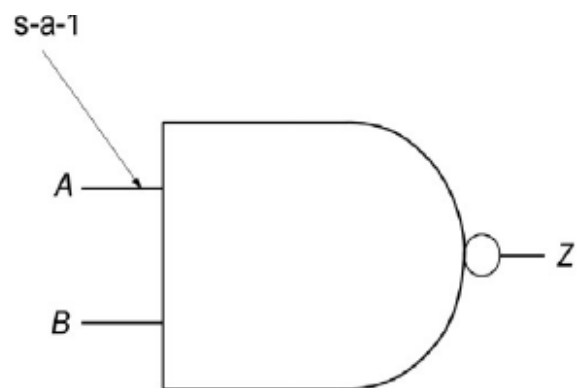
#### 3. Delay Faults

### Stuck-At Fault

The most common model used for logical faults is the *single stuck-at fault*. It assumes that a fault in a logic gate results in one of its inputs or the output is fixed at either a logic 0 (*stuck-at-0*) or at logic 1 (*stuck-at-1*). Stuck-at-0 and stuck-at-1 faults are often abbreviated to *s-a-0* and *s-a-1*, respectively.

Let us assume that in Figure 1.1 the A input of the NAND gate is s-a-1. The NAND gate perceives the A input as a logic 1 irrespective of the logic value placed on the input. For example, the output of the NAND gate is 0 for the input pattern  $A=0$  and  $B=1$ , when input A is s-a-1. In the absence of the fault, the output will be 1. Thus,  $AB=01$  can be considered as the *test* for the A inputs-a-1, since there is a difference between the output of the fault-free and faulty gate.

The single stuck-at fault model is often referred to as the *classical fault model* and offers a good representation for the most common types of defects [e.g., shorts and opens in complementary



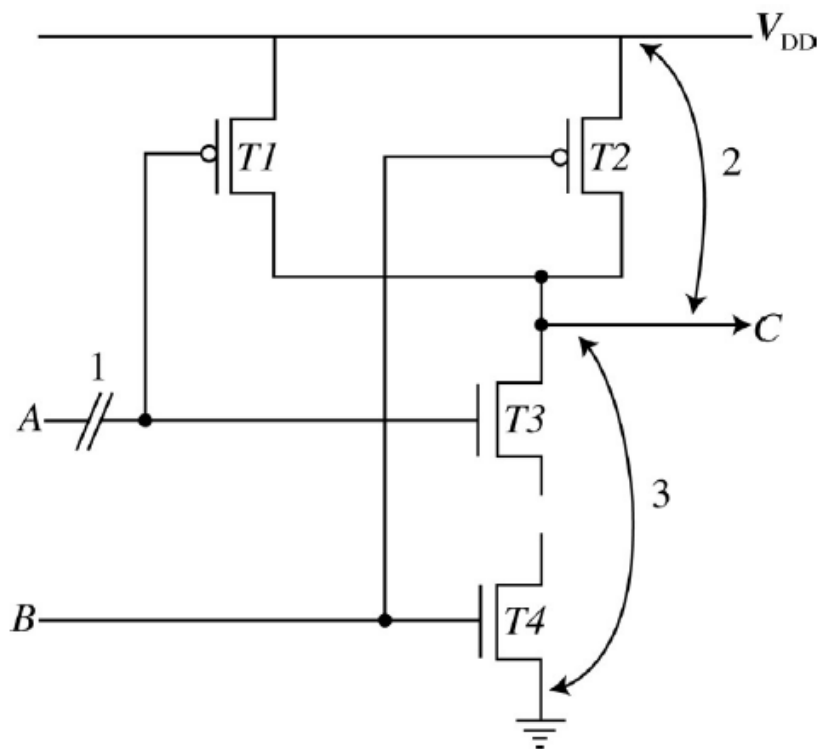
**FIGURE 1.1:** Two-input NAND gate.

metal oxide semiconductor (CMOS) technology]. Figure 1.2 illustrates the CMOS realization of the two-input NAND:

The number 1 in the figure indicates an open, whereas the numbers 2 and 3 identify the short between the output node and the ground and the short between the output node and the VDD, respectively. A short in a CMOS results if not enough metal is removed by the photolithography, whereas over-removal of metal results in an open circuit [3]. Fault 1 in Figure 1.2 will disconnect

input  $A$  from the gate of transistors  $T1$  and  $T3$ . It has been shown that in such a situation one transistor may conduct and the other remain nonconducting [4]. Thus, the fault can be represented by a stuck at value of  $A$ ; if  $A$  is s-a-0,  $T1$  will be ON and  $T3$  OFF, and if  $A$  is s-a-1,  $T1$  will be OFF and  $T3$  ON. Fault 2 forces the output node to be shorted to  $V_{DD}$ , that is, the fault can be considered as an s-a-1 fault. Similarly, fault 3 forces the output node to be s-a-0.

The stuck-at model is also used to represent multiple faults in circuits. In a multiple stuck-at fault, it is assumed that more than one signal line in the circuit are stuck at logic 1 or logic 0; in other

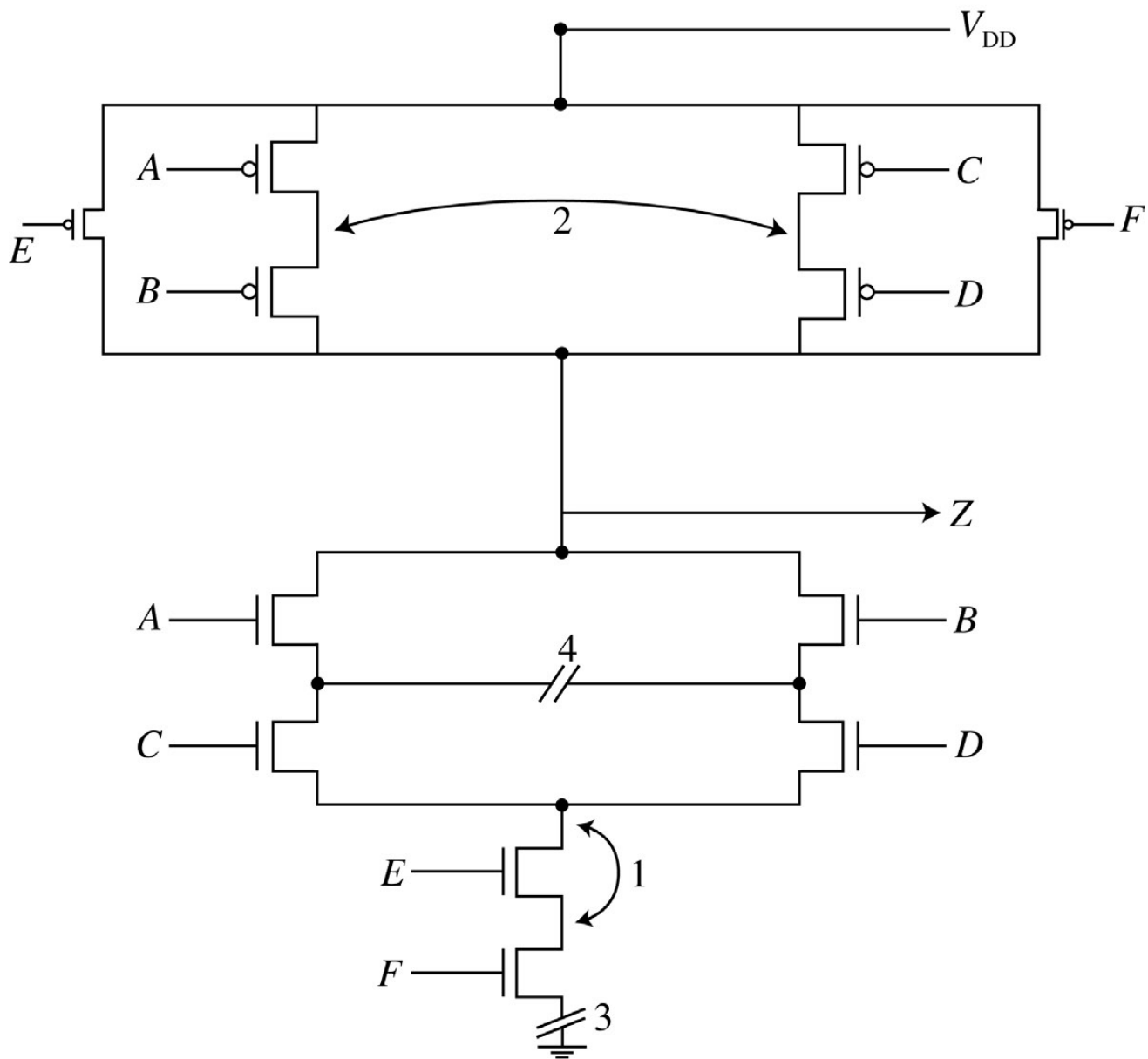


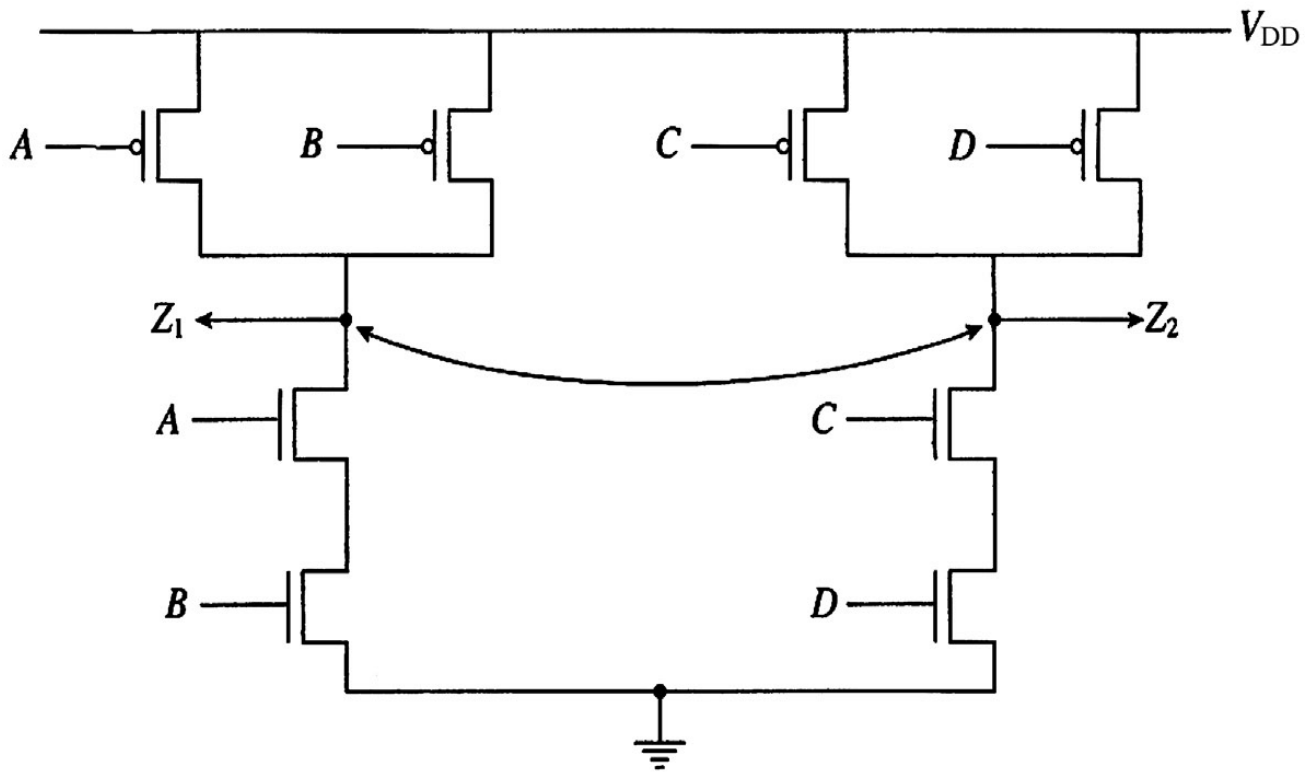
**FIGURE 1.2:** Two-input NAND gate in CMOS gate.

on \_words, a group of stuck-at faults exist in the circuit at the same time. A variation of the multiple fault is the *unidirectional fault*. A multiple fault is unidirectional if all of its constituent faults are either s-a-0 or s-a-1 but not both simultaneously. The stuck-at model has gained wide acceptance in the past mainly because of its relative success with small scale integration. However, it is not very effective in accounting for all faults in present day very large scale integrated (VLSI), circuits which mainly uses CMOS technology. Faults in CMOS circuits do not necessarily produce logical faults that can be described as stuck-at faults [5, 6, 7]. For example, in Figure 1.2, faults 3 and 4 create stuck-on transistors faults.

$$Z = \overline{(A + B)(C + D)} \cdot EF.$$

Two possible shorts numbered 1 and 2 and two possible opens numbered 3 and 4 are indicated in the diagram. Short number 1 can be modeled by s-a-1 of input  $E$ ; open number 3 can be modeled by s-a-0 of input  $E$ , input  $F$ , or both. On the other hand, short number 2 and open number





**FIGURE 1.4:** CMOS implementation of  $Z_1 = \overline{AB}$  and  $Z_2 = \overline{CD}$ .

### Bridging Faults

Bridging faults form an important class of permanent faults that cannot be modeled as stuck-at faults. A bridging fault is said to have occurred when two or more signal lines in a circuit are accidentally connected together. Earlier study of bridging faults concentrated only on the shorting of signal lines in gate-level circuits. It was shown that the shorting of lines resulted in *wired logic* at the connection.

Bridging faults at the gate level has been classified into two types: *input bridging* and *feedback bridging*. An input bridging fault corresponds to the shorting of a certain number of primary input lines.

A feedback bridging fault results if there is a short between an output and input line. A feedback bridging fault may cause a circuit to oscillate, or it may convert it into a sequential circuit.

Bridging faults in a transistor-level circuit may occur between the terminals of a transistor or between two or more signal lines. Figure 1.5 shows the CMOS logic realization of the Boolean function:

$$Z_1 = Z_2 = \overline{AB} + \overline{CD}$$

A short between two lines, as indicated by the dotted line in the diagram will change the function of the circuit.

The effect of bridging among the terminals of transistors is technology-dependent. For example, in CMOS circuits, such faults manifest as either stuck-at or stuck-open faults, depending on the physical location and the value of the bridging resistance.

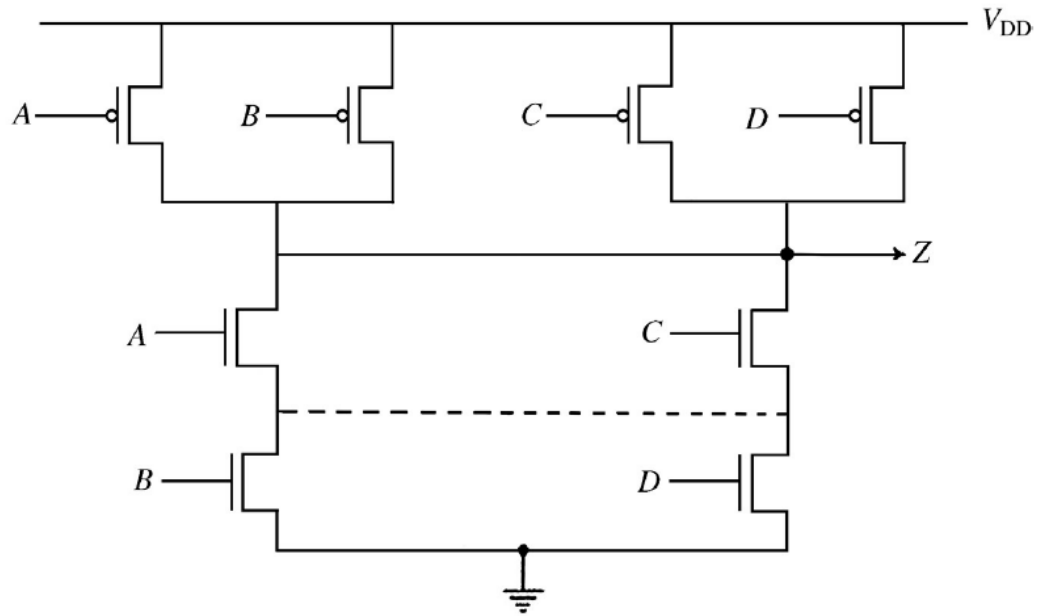


FIGURE 1.5: CMOS implementation of  $Z(A, B, C, D) = \overline{AB} + \overline{CD}$

### Delay Faults

As mentioned previously, not all manufacturing defects in VLSI circuits can be represented by the stuck-at fault model. The size of a defect determines whether the defect will affect the logic function of a circuit. Smaller defects, which are likely to cause partial open or short in a circuit, have a higher probability of occurrence due to the statistical variations in the manufacturing process [8]. These defects result in the failure of a circuit to meet its timing specifications without any alteration of the logic function of the circuit. A small defect may delay the transition of a signal on a line either from 0 to 1, or vice versa. This type of malfunction is modeled by a *delay fault*.

Two types of delay faults have been proposed in literature: *gate delay fault* and *path delay fault*. Gate delay faults have been used to model defects that cause the actual propagation delay of a faulty gate to exceed its specified worst case value. For example, if the specified worst case propagation delay of a gate is  $x$  units and the actual delay is  $x + \Delta x$  units, then the gate is said to have a delay fault of size  $\Delta x$ . The main deficiency of the gate delay fault model is that it can only be used to model isolated defects, not distributed defects, for example, several small delay defects. The path delay fault model can be used to model isolated as well as distributed defects. In this model, a fault is assumed to have occurred if the propagation delay along a path in the circuit under test exceeds the specified limit.

## 2. BASIC CONCEPTS OF FAULT DETECTION

Fault detection in a logic circuit is carried out by applying a sequence of tests and observing the resulting outputs. A *test* is an input combination that specifies the expected response that a fault-free circuit should produce. If the observed response is different from the expected response, a fault is

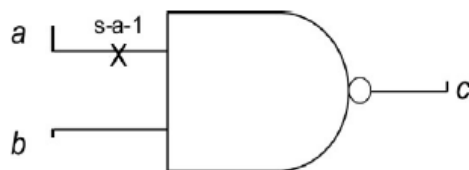


FIGURE 1.8: A NAND gate with a stuck-at-1 fault.

INPUT		OUTPUT	
$a$	$b$	$c$ (FAULT-FREE)	$c$ (FAULT-PRESENT)
0	0	1	1
0	1	1	0
1	0	1	1
1	1	0	0

present in the circuit. The aim of testing at the gate level is to verify that each logic gate in the circuit is functioning properly and the interconnections are good. Henceforth, we will deal with stuck-at faults only unless mentioned otherwise. If only a single stuck-at fault is assumed to be present in the circuit under test, then the problem is to construct a test set that will detect the fault by utilizing only the inputs and the outputs of the circuit.

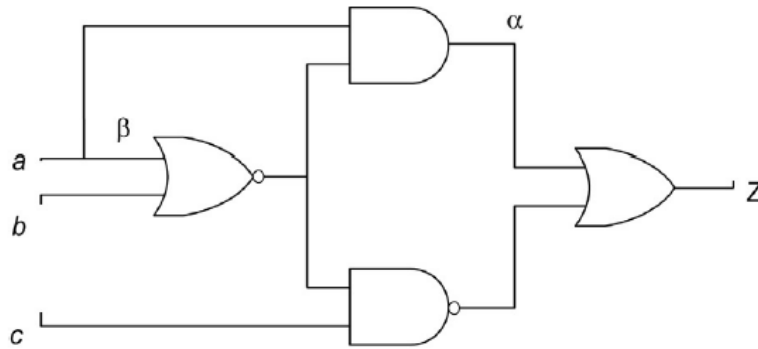
### Controllability and Observability

As indicated above, in order to generate a test  $a$  for a stuck-at fault on a signal line, it must first be forced to a value that is opposite to the stuck-at value on the line. This ability to apply input patterns to the primary inputs of a circuit to set up appropriate logic value at desired locations of a circuit is known as *controllability*. For example, in the presence of a stuck-at-0 fault, the location of the fault must be set to logic 1 via the primary inputs; this is known as *1-controllability*. Similarly, for a stuck-at-1 fault, the location of the fault must be set to logic 0 to excite the fault; this is known as *0-controllability*.

The sensitization part of the test generation process requires application of appropriate input values at the primary inputs so the effect of the fault is observable at the primary outputs. For example, in Figure 1.9, the effect of the stuck-at-1 fault can be observed at output  $Z$  if input  $d$  is set at 1; if  $d$  is set to 0, the output will be 0 and the effect of the fault will be masked (i.e., the fault will not be detected). The ability to observe the response of a fault on an internal node via the primary outputs of a circuit is denoted as *observability*.

### Undetectable Faults

A fault is considered to be undetectable if it is not possible to activate the fault or to sensitize its effect to primary outputs. In other words, a test for detecting the fault does not exist. To illustrate, let us consider the  $\alpha$  s-a-0 fault shown in Figure 1.10. It is not possible to set the node  $\alpha$  to logic 1. Therefore, the fault cannot be excited and thus undetectable. The fault  $\beta$  s-a-0 can be excited by making  $ab=10$ , but no sensitized path is available for propagating the effect of the fault to the



**FIGURE 1.10:** Circuit with a stuck-at-0 faults.

output; hence, the fault is undetectable. A combinational circuit is denoted as *redundant* if it has an undetectable fault.

A test set for a circuit is derived based on the assumption that only a single fault is present in the circuit when the tests are applied. Thus, the simultaneous presence of an undetectable fault and a detectable fault violates this assumption. Furthermore, the presence of an undetectable fault may prevent the detection of a detectable fault.

### Equivalent Faults

A test, in general, can detect more than one fault in a circuit, and many tests in a set detect the same faults. In other words, the subsets of faults detected by each test from a test set are not disjoint. Thus, a major objective in test generation is to reduce the total number of faults to be considered by grouping equivalent faults in subsets. It is then sufficient only to test one fault from each equivalent set to cover all faults in the set, thus avoiding redundancy in the test generation process.

In an  $m$ -input gate, there can be  $2(m+1)$  stuck-at faults. Thus, the total number of single stuck-at faults in a two-input NOR gate shown in Figure 1.11a is 6 ( $=2 \times 3$ ), e.g.,  $a$  s-a-0,  $b$  s-a-0,  $a$  s-a-1,  $b$  s-a-1,  $c$  s-a-0 and  $c$  s-a-1. However, a stuck-at fault on an input may be indistinguishable from a stuck-at fault at the output. For example, in a NOR gate (Figure 1.11a), any input s-a-1 fault is indistinguishable from the output s-a-0; similarly, in a NAND gate (Figure 1.11b), an input s-a-0 fault is indistinguishable from the output s-a-1.

### Temporary Faults

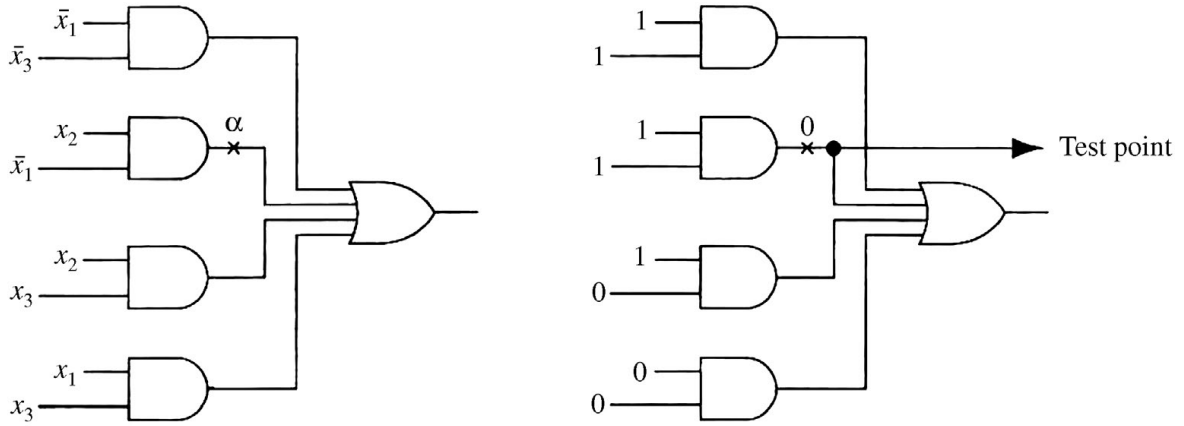
As stated earlier, an error is a manifestation of a fault. A temporary fault can result in an *intermittent* or a *transient* error. Transient errors are the major source of failures in VLSI chips. They are nonrecurring and are not repairable because there is no physical damage to the hardware. Very deep submicron technology has enabled the packing of millions of transistors on a VLSI chip by reducing the transistor dimensions. However, the reduction of transistor sizes also reduces their noise margins. As a result, they become more vulnerable to noise, cross-talk, etc., which in turn result in transient errors. In addition, small transistors are affected by terrestrial radiation and suffer temporary malfunction, thereby increasing the rate of transient errors.

Intermittent faults are recurring faults that reappear on a regular basis. Such faults can occur due to loose connections, partially defective components, or poor designs. Intermittent faults occurring due to deteriorating or aging components may eventually become permanent. Some intermittent faults also occur due to environmental conditions such as temperature, humidity, vibration, etc. The likelihood of such intermittent faults depends on how well the system is protected from its physical environment through shielding, filtering, cooling, etc. An intermittent fault in a circuit causes a malfunction of the circuit only if it is active; if it is inactive, the circuit operates correctly. A circuit is said to be in a fault active state if a fault present in the circuit is active, and it is said to be in the fault-not-active state if a fault is present but inactive.

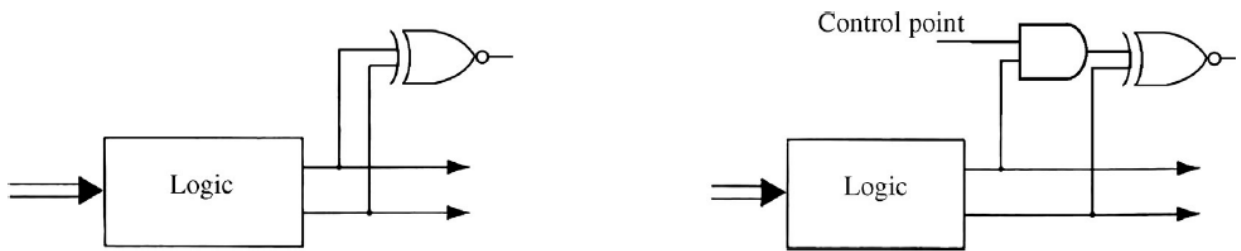
## 3. AD HOC TECHNIQUES

One of the simplest ways of improving the testability of a circuit is to provide more tests and control points. Test points are, in general, used to observe the response at a node inside the circuit, whereas control points are utilized to control the value of an internal node to any desired value, 0 or 1. For

example, in the circuit shown in Figure 3.1a, the fault  $\alpha$  s-a-0 is undetectable at the circuit output.



(a) Circuit with undetectable fault  $\alpha$  s-a-0 (b) Fault detected when  $x_1x_2x_3 = 010$  is applied

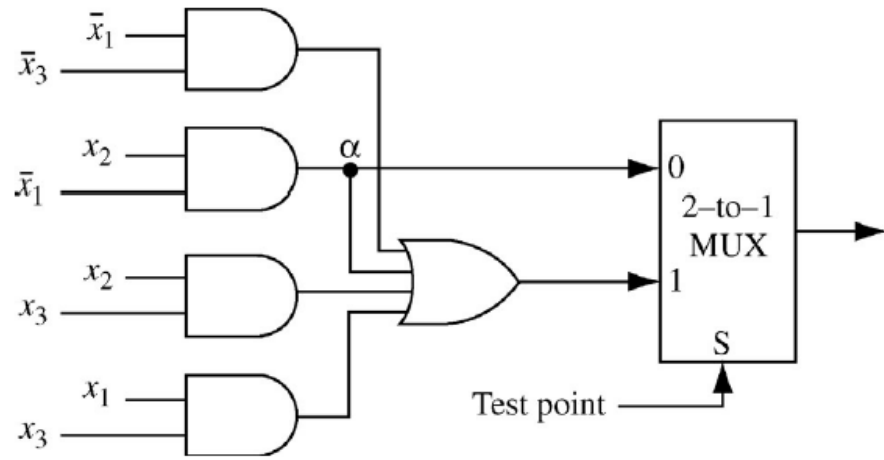


**FIGURE 3.2:** (a) EX-NOR gate not testable. (b) EX-NOR gate easily testable.

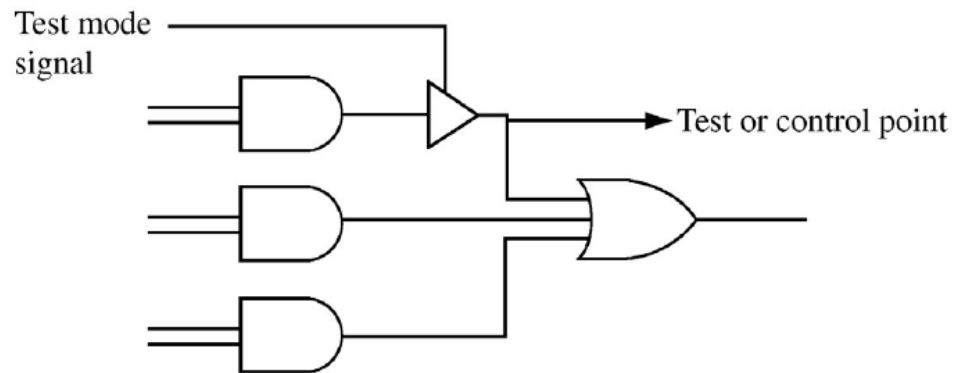
By incorporating a test point at node  $\alpha$  as shown in Figure 3.1b, the input combination 010 or 011 can be applied to detect the fault.

The usefulness of adding a control point can be appreciated from the circuit shown in Figure 3.2a. If the output of the EX-NOR gate in the circuit is always 1, indicating that both the outputs of the logic block are the same, it is not possible to say whether the EX-NOR gate is operating correctly or not. If a control point is added to the circuit, as shown in Figure 3.2b, the input of the EX-NOR gate and hence the operation of the circuit can be controlled via the added point. During the normal operation of the circuit, the control point is set at logic 1. To test for an s-a-1 fault at the output of the EX-NOR gate, the control point is set at logic 0 and an input combination that produces logic 1 at the outputs has to be applied.

Another way of improving the testability of a particular circuit is to insert multiplexers in order to increase the number of internal nodes that can be controlled or observed from the external points. For example, the fault  $\alpha$  s-a-0 in Figure 3.1a can also be detected by incorporating a 2-to-1 multiplexer as shown in Figure 3.3. When the test input (i.e., the select input of the multiplexer) is



**FIGURE 3.3:** Use of multiplexer to enhance testability.



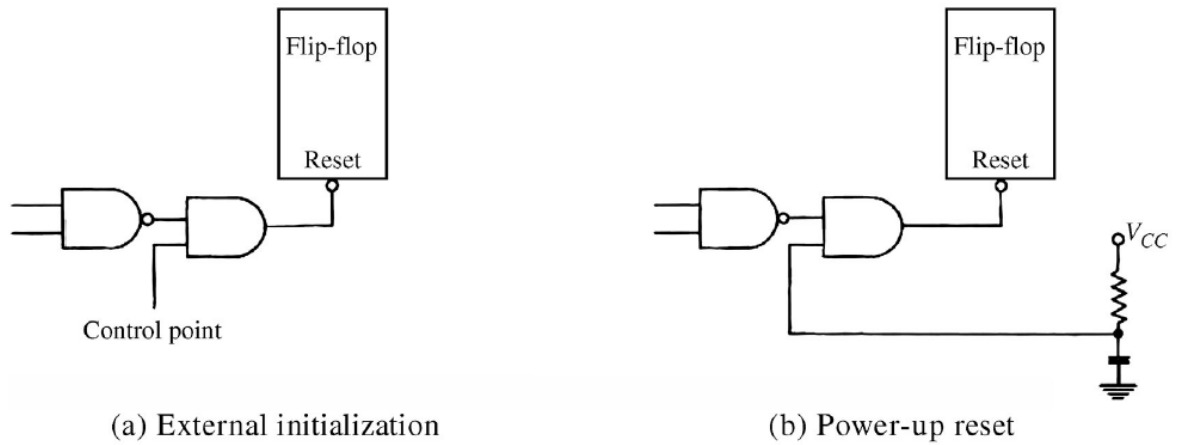
**FIGURE 3.4:** Use of tristate drivers to improve testability.

at logic 1, the output of the circuit is transferred to the output of the multiplexer. On the other hand, if the control input is at logic 0 and the input combination 010 or 011 is applied to the circuit, the state of node  $\alpha$  can be observed at the multiplexer output.

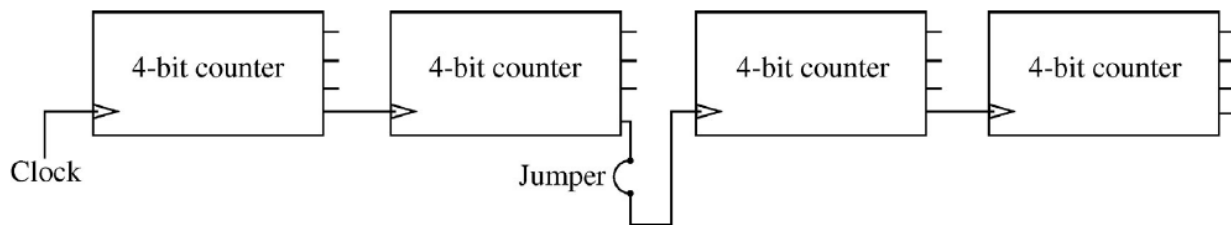
A different way of accessing internal nodes is to use tristate drivers as shown in Figure 3.4.

A test mode signal could be used to put the driver into the high-impedance state. In this mode, the input of the OR gate can be set to logic 0 or logic 1 from an external point. When the driver is enabled, the same external point becomes a test point.

The test mode signals required by the added components, such as multiplexers, tristate drivers, etc., cannot always be applied via external points, because it is often not practicable to have many such points. To reduce the number of external points, a *test state register* may be included in the



**FIGURE 3.5:** (a) External initialization. (b) Power-up reset.



**FIGURE 3.6:** Breaking up of a counter chain.

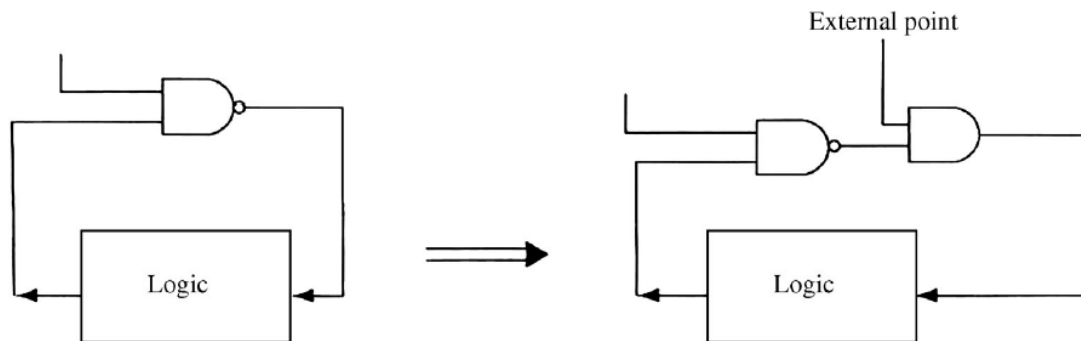
circuit. This could in fact be a shift register that is loaded and controlled by just a few pins. The testability hardware in the circuit can then be controlled by the parallel outputs of the shift register.

Frequently, flip-flops, counters, shift registers, and other memory elements assume unpredictable states when power is applied, and they must be set to known states before testing can begin.

Ideally, all memory elements should be reset from external points (Figure 3.5a). Alternatively, a power-up reset may be added to provide internal initialization (Figure 3.5b).

A long counter chain presents another test problem. For example, the counter chain in Figure 3.6 requires thousands of clock pulses to go through all the states. One way to avoid this problem is to break up the long chains into smaller chains by filling jumpers to them; the jumpers can be removed during testing. A tristate driver can function as a jumper in this case. The input of the tristate driver is connected to the clock, and the output to the clock input of the second counter chain. When the control input of the tristate driver is disabled, the clock is disconnected from the second counter chain; thus, this chain can be tested separately from the first chain.

A feedback loop is also difficult to test because it hides the source of the fault. The source can be located by breaking the loop and bringing both lines to external points that are shown during



**FIGURE 3.7:** Breaking a feedback loop by using an extra gate.

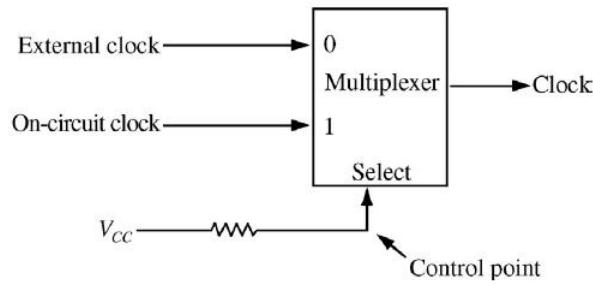


FIGURE 3.8: Replacement of on-circuit clock.

normal operation. When not shorted, the separate lines provide a control point and a test point. An alternative way of breaking a feedback loop is to add to the feedback path a gate that can be interrupted by a signal from a control point (Figure 3.7).

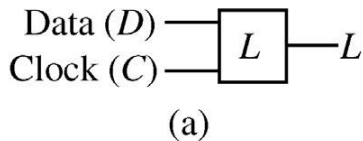
On-circuit clock oscillators should be disconnected during test and replaced with an external clock. The external clock can be single-stepped to check the logic values at various nodes in the circuit during the fault diagnosis phase. Figure 3.8 shows how the onboard clock can be replaced by an external one.

#### 4. LEVEL-SENSITIVE SCAN DESIGN

One of the best known and the most widely practiced methods for synthesizing testable sequential circuits is IBM's level-sensitive scan design (LSSD) [2–5]. The *level-sensitive* aspect of the method means that a sequential circuit is designed so that the steady-state response to any input state change is independent of the component and wire delays within the circuit. Also, if an input state change involves the changing of more than one input signal, the response must be independent of the order in which they change. These conditions are ensured by the enforcement of certain design rules, particularly pertaining to the clocks that evoke state changes in the circuit. *Scan* refers to the ability to shift into or out of any state of the circuit.

##### Clocked Hazard-Free Latches

In LSSD, all internal storage is implemented in hazard-free polarity-hold latches. The polarity-hold latch has two-input signals as shown in Figure 3.18a. The latch cannot change state if  $C=0$ . If  $C$  is set to 1, the internal state of the latch takes the value of the excitation input  $D$ . A flow table for this sequential circuit, along with an excitation table and a logic implementation, is shown in Figure 3.18b, 3.18c, and 3.18d, respectively.

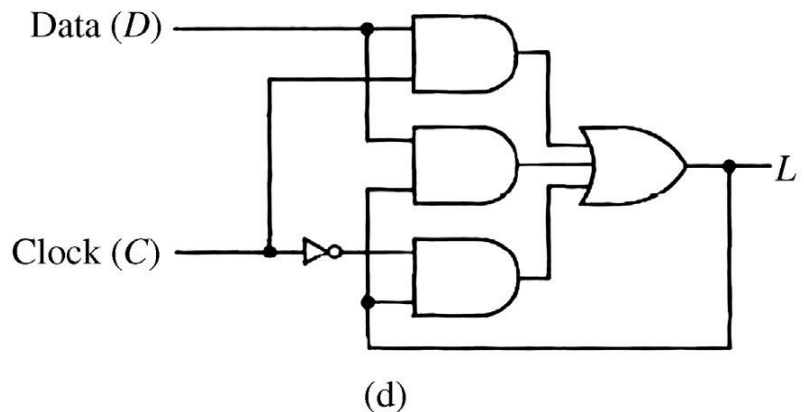


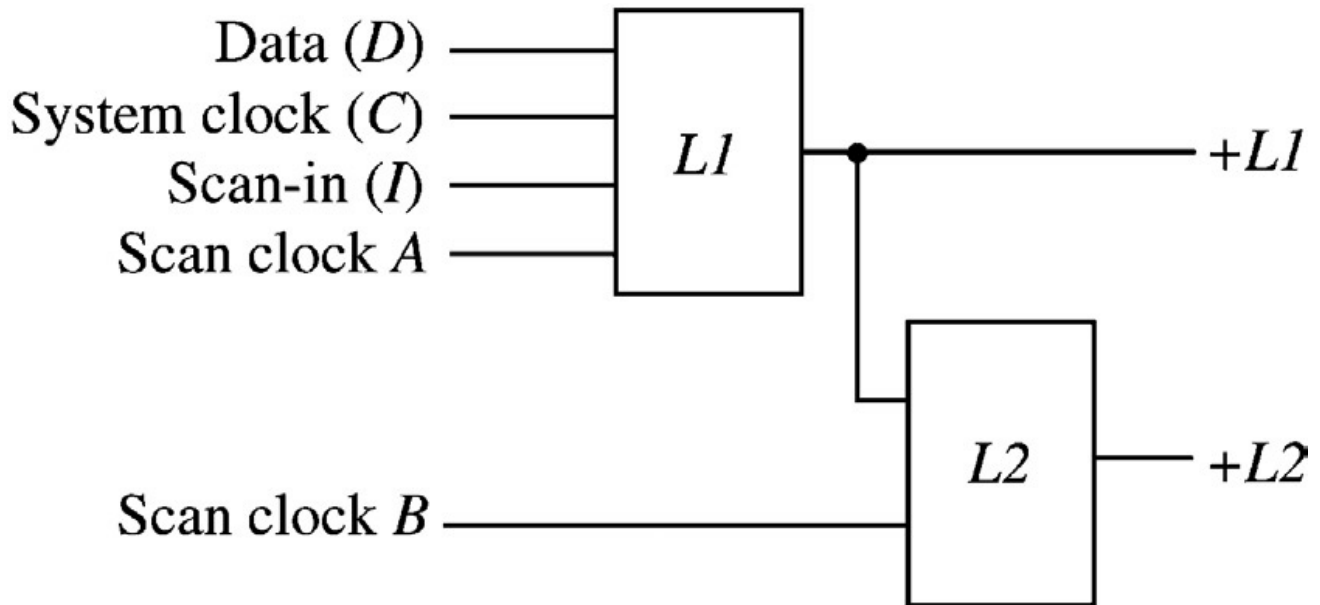
		CD				
		00	01	11	10	L
State	A	A	A	B	A	0
	B	B	B	B	A	1

(b)

		CD				
		00	01	11	10	L
y	0	0	0	1	0	0
	1	1	1	1	0	1

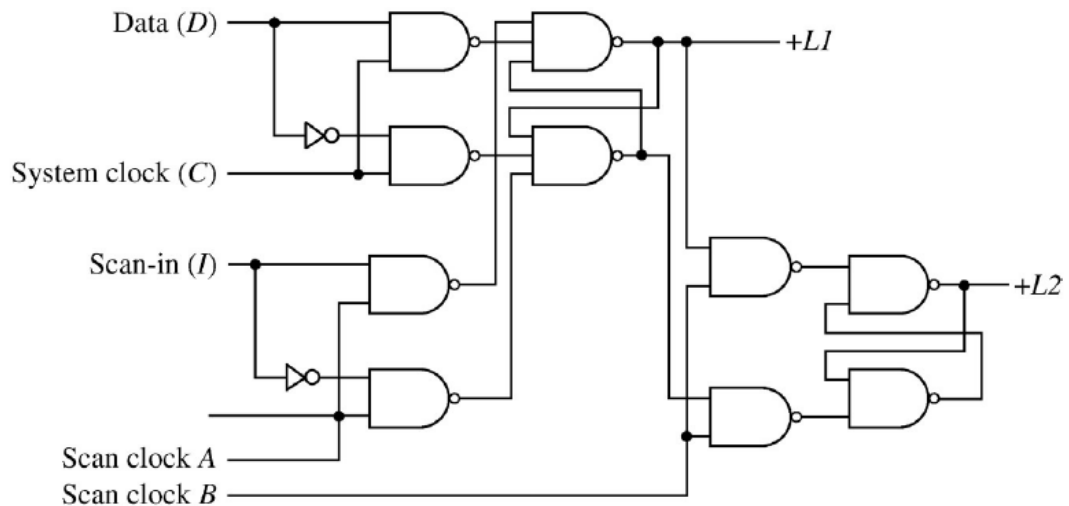
(c)



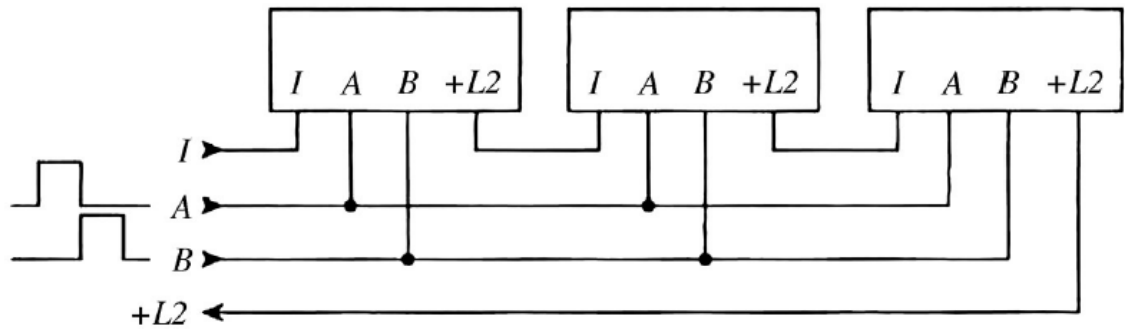


The clock signal  $C$  will normally occur (change from 0 to 1) after the data signal  $D$  has become stable at either 1 or 0. The output of the latch is set to the new value of the data signal at the time the clock signal occurs. The correct changing of the latch does not depend on the rise or fall time of the clock signal, but only on the clock signal being 1 for a period equal to a greater than the time required for the data signal to propagate through the latch and stabilize.

A shift register latch (SRL) can be formed by adding a clocked input to the polarity-hold latch  $L1$  and including a second latch  $L2$  to act as intermediate storage during shifting (Figure 3.13). As long as the clock signals  $A$  and  $B$  are both 0, the  $L1$  latch operates exactly like a polarityhold latch. Terminal  $I$  is the scan-in input for the SRL and  $+L2$  is the output. The logic implementation of the SRL is shown in Figure 3.14. When the latch is operating as a shift register data from the preceding stage are gated into the polarity-hold switch via  $I$ , through a change of the clock  $A$  from 0 to 1. After  $A$  has changed back to 0, clock  $B$  gates the data in the latch  $L1$  into the output latch  $L2$ . Clearly,  $A$  and  $B$  can never both be 1 at the same time if the SRL is to operate properly



**FIGURE 3.14:** Logic for shift-register latch (Reprinted from Ref. [2], © 1978).



**FIGURE 3.15:** Linkage of three SRLs (Reprinted from Ref. [2], © 1978).

The SRLs can be interconnected to form a shift register as shown in Figure 3.15. The input  $I$  and the output  $+L2$  are strung together in a loop, and the clocks  $A$  and  $B$  are connected in parallel. A specific set of design rules has been defined to provide level-sensitive logic subsystems with a scannable design that would aid testing:

*Rule 1.* Use only hazard-free polarity-hold latches as memory elements.

*Rule 2.* The latches must be controlled by nonoverlapping clocks.

*Rule 3.* Clock signals must be applied via primary inputs.

*Rule 4.* Clocks may not feed the data inputs to memory elements either directly or through combinational logic.

*Rule 5.* Test sequences must be applied via a primary input.

### Double-Latch and Single-Latch LSSD

A sequential logic circuit that is level-sensitive and also has the scan capability is called a Level Sensitive Scan Design (LSSD). Figure 3.16 depicts a general structure for an LSSD system, known as a *double-latch design* in which all system outputs are taken from the  $L2$  latch. In this configuration, each SRL operates in a master-slave mode. Data transfer occurs under system clock and scan clock  $B$  during normal operation and under scan clock  $A$  and scan clock  $B$  during scan-path operation. Both latches are therefore required during system operation.

In the single-latch configuration, the combinational logic is partitioned into two disjoint sets,  $Comb1$  and  $Comb2$  (Figure 3.17). The system clocks used for SRLs in  $Comb1$  and  $Comb2$  are denoted by Clock 1 and Clock 2, respectively; they are nonoverlapping. The outputs of the SRLs in  $Comb1$  are fed back as secondary variable inputs to  $Comb2$ , and vice versa. This configuration uses the output of latch  $L1$  as the system output; the  $L2$  latch is used only for shifting. In other words, the  $L2$  latches are redundant and represent the overhead for testability.

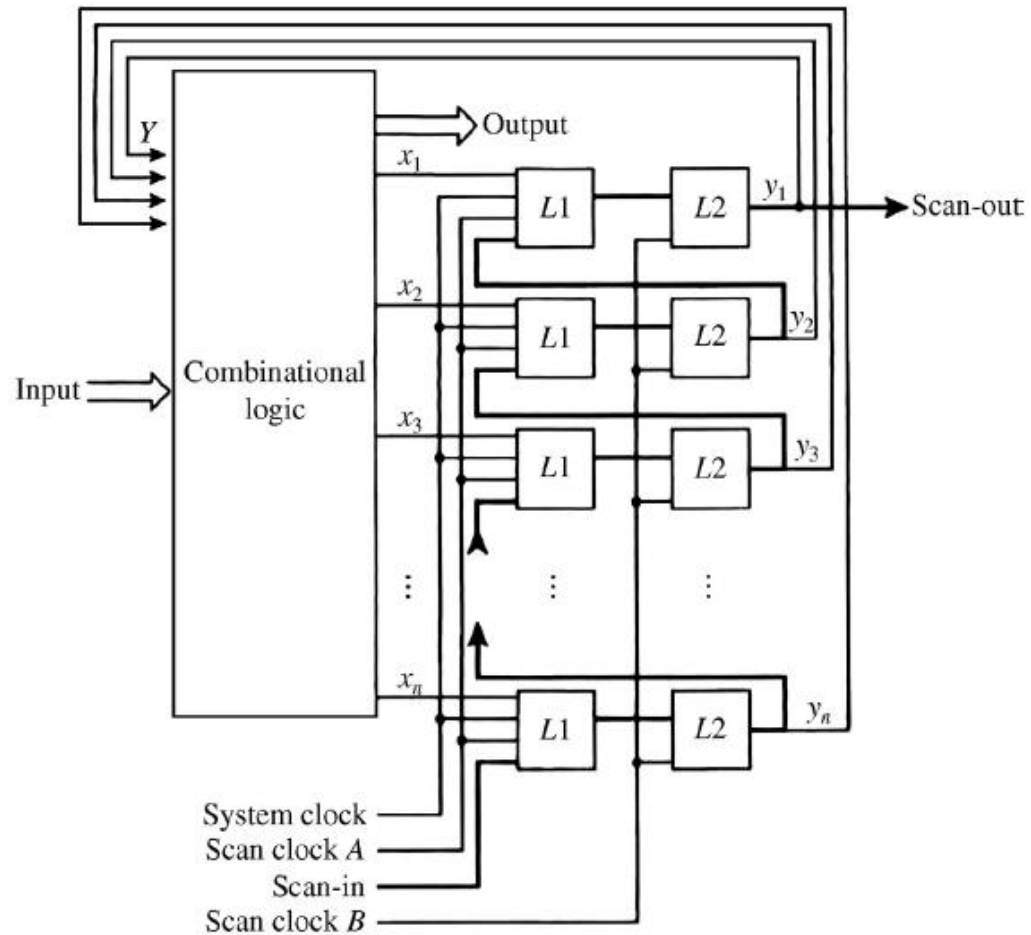
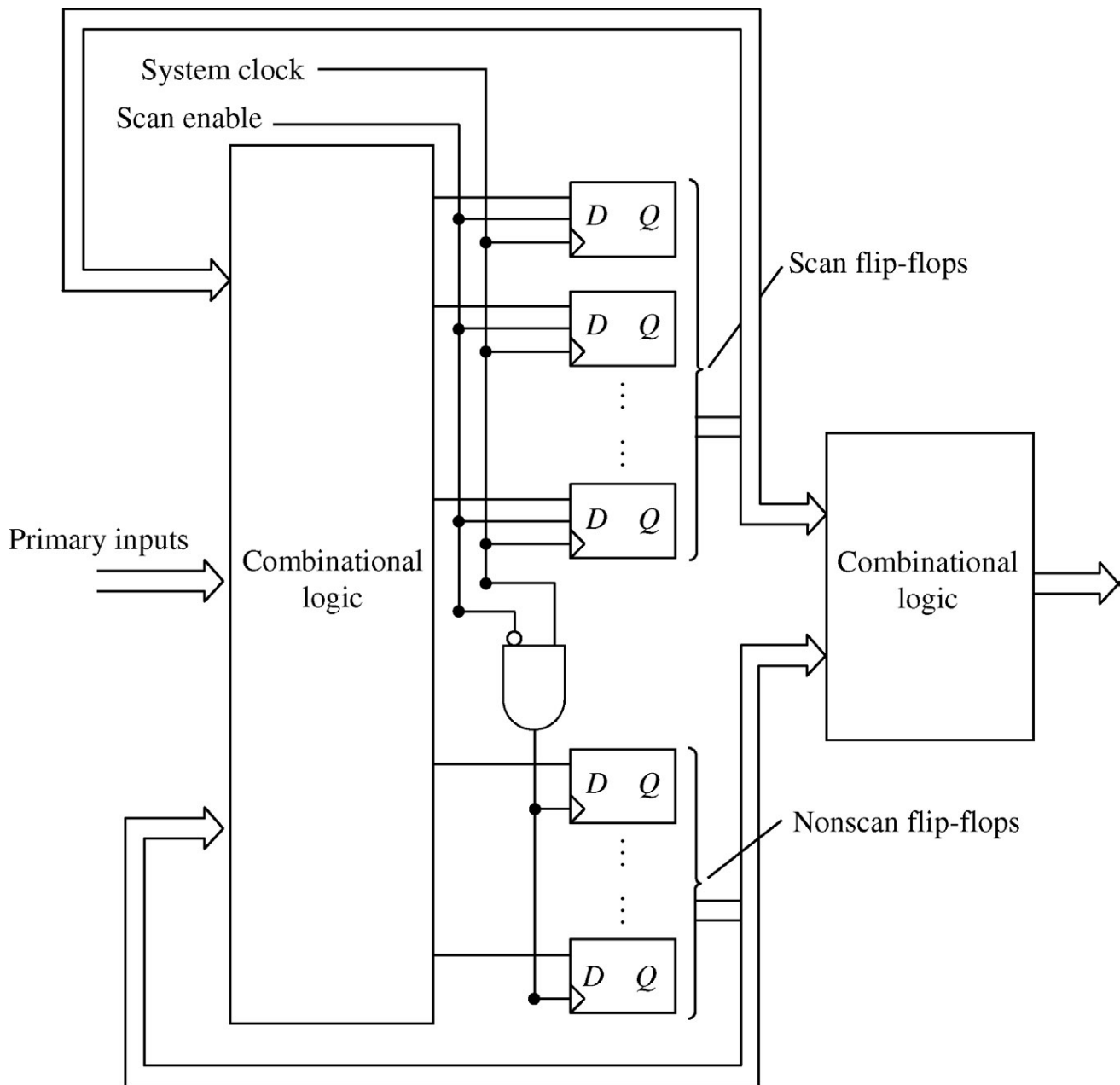


FIGURE 3.16: Double-latch LSSD (Adapted from Ref. [6]).

## 5. PARTIAL SCAN

In full scan, all flip-flops in a circuit are connected into one or more shift registers; thus, the states of a circuit can be controlled and observed via the primary input and outputs, respectively. In partial scan, only a subset of the circuit flip-flops is included in the scan chain in order to reduce the overhead associated with full scan design [6]. Figure 3.21 shows a structure of partial scan design. This has two separate clocks: a system clock and a scan clock. The scan clock controls only the scan flip-flops. Note that the scan clock is derived by gating the system clock with the scan-enable signal; no external clock is necessary. During the normal mode of operation, i.e., when the scan-enable signal is at logic 0, both scan and nonscan flip-flops update their states when the system clock is applied. In the scan mode operation, only the state of the shift register (constructed from the scan flip-flops) is shifted one bit with the application of the scan flip-flop; the nonscan flip-flops do not change their states.

The disadvantage of two-clock partial scan is that the routing of two separate locks with small skews is very difficult to achieve. Also, the use of a separate scan clock does not allow the testing of the circuit at its normal operating speed.



**Fig. partial scan using two clocks**

A partial scan scheme that uses the system clock as the scan clock is shown in Figure 3.22. [8]. Both scan and nonscan flip-flops move to their next states when the system clock is applied. A test sequence is derived by shifting data into the scan flip-flops. This data together with contents of nonscan flip-flops constitute the starting state of the test sequence. The other patterns in the sequence are obtained by single-bit shifting of the contents of scan flip-flops, which form part of the required circuit states. The remaining bits of the states, i.e., the contents of the scan flip-flops are determined by the functional logic. Note this form of partial scan scheme allows only a limited number of valid next states to be reached from the starting state of the test sequence. This may limit the fault coverage obtained by using the technique.

The selection flip-flops to be included in the partial scan is done by heuristic methods. It has been shown that the fault coverage in a circuit can be significantly increased by including 13–23%

of the flip-flops in the partial scan [9].

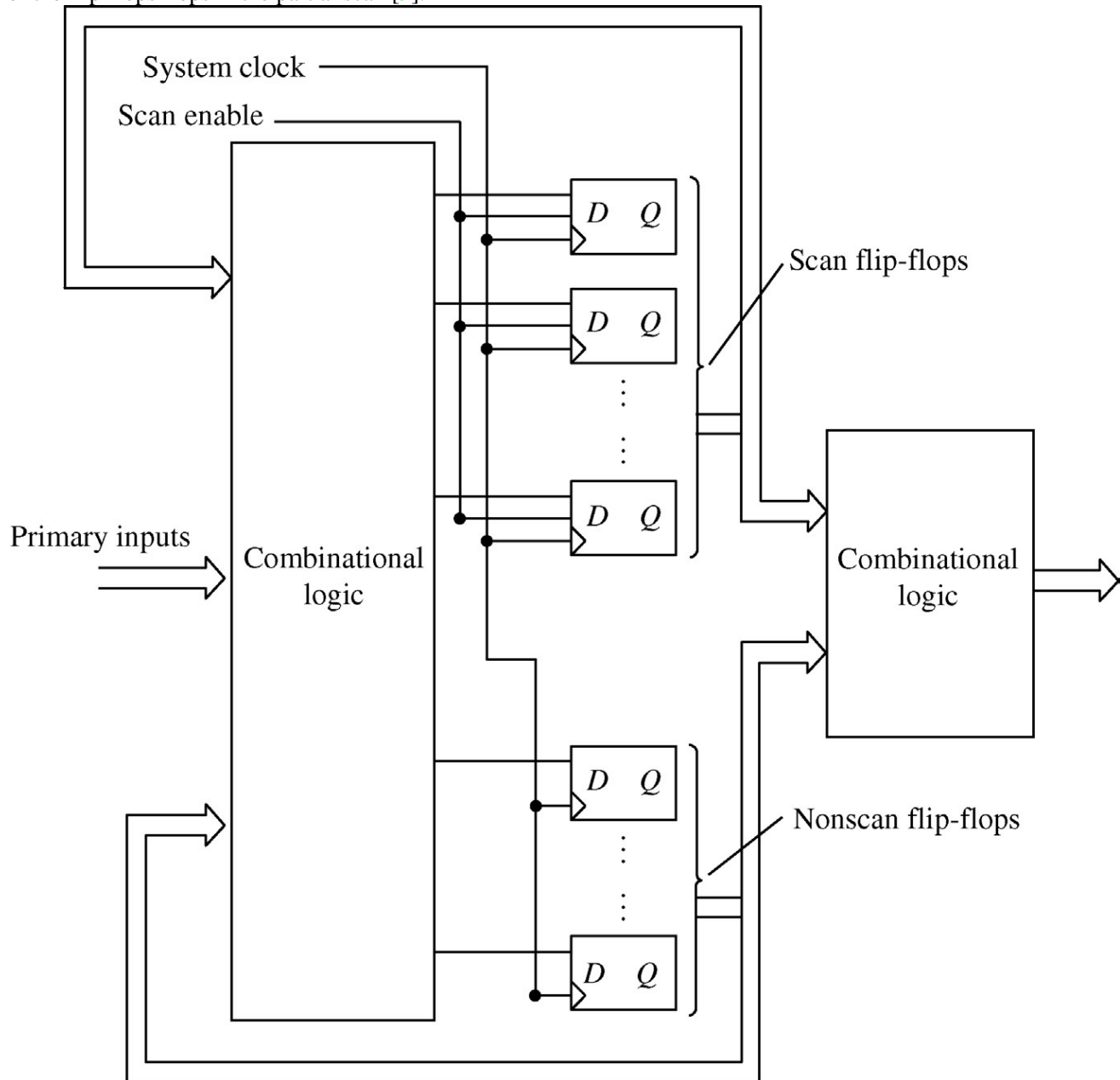


fig. Partial scan using system clock

## 5. TEST PATTERN GENERATION FOR BIST

Test pattern generation approaches for BIST schemes can be divided into four categories:

- 1.Exhaustive testing;
- 2.Pseudoexhaustive testing;
- 3.Pseudorandom testing;
- 4.Deterministic testing.

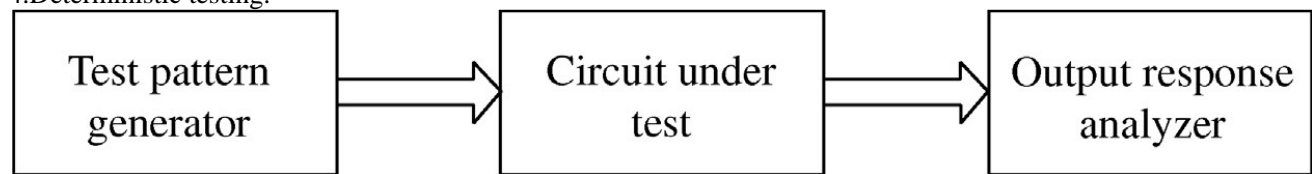


Fig.BIST configuration

### 4.1.1 Exhaustive Testing

In the exhaustive testing approach, all possible input patterns are applied to the circuit under test. Thus, for an  $n$ -input combinational circuit, all possible  $2^n$  patterns need to be applied. The advantage of this approach is that all nonredundant faults can be detected; however, any fault that converts

the combinational circuit into a sequential circuit, for example, a bridging fault, cannot be detected. The drawback of this approach is that when  $n$  is large, the test application time becomes prohibitive, even with high clock speeds. Thus, exhaustive testing is practical only for circuits with a limited number of inputs.

A modified form of exhaustive testing is *pseudoexhaustive testing* [1]. It retains the advantages of exhaustive testing while significantly reducing the number of test patterns to be applied. The basic idea is to partition the circuit under test into several sub-circuits such that each sub-circuit has few enough inputs for exhaustive testing to be feasible for it. This concept has been used in *autonomous design verification* technique proposed in Ref. [2]. To illustrate the use of the technique, let us consider the circuit shown in Figure 4.2a. The circuit is partitioned into two sub-circuits  $C_1$  and  $C_2$  as shown in Figure 4.2b. The functions of the two added control inputs MC1 and MC2 are as follows:

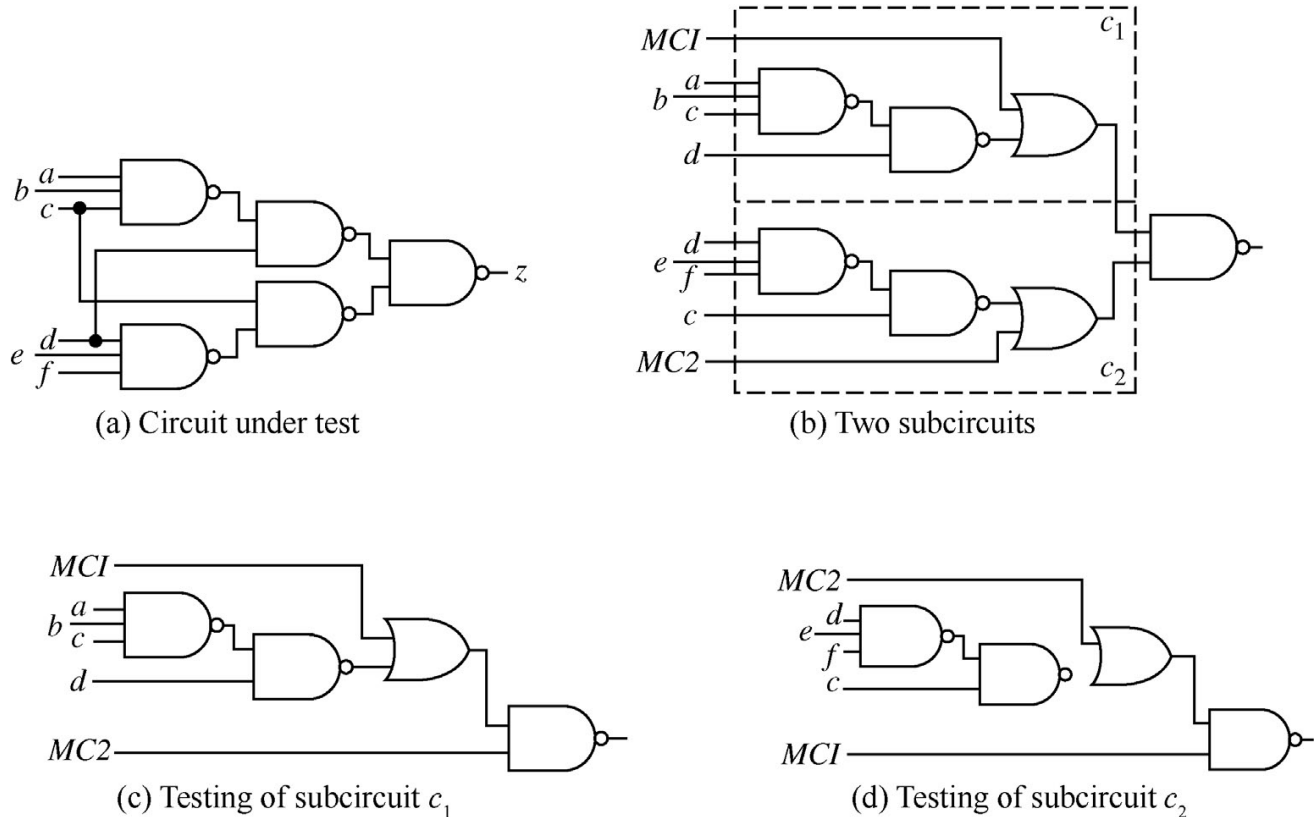


Fig. Partitioning of circuit for a autonomous testing

When  $MC1=0$  and  $MC2=1$ , sub-circuit  $C_2$  is disabled; sub-circuit  $C_1$ , shown in Figure 4.2c,

can be tested by applying all possible input combinations at  $a, b, c,$  and  $d$ . Similarly, when  $MC1=1$

and  $MC2=0$ , sub-circuit  $C_1$  is disabled. In this mode, the circuit structure is as shown in Figure 4.2d and sub-circuit  $C_2$  can be tested by applying all input combinations at  $c, d, e,$  and  $f$ .

When  $MC1=MC2=0$ , the circuit functions as the unmodified circuit except for the added gate delay. The advantage of the design method is that any fault in the circuit itself and the testing circuit is detectable.

### Pseudoexhaustive Pattern Generation

A combinational circuit with  $n$  inputs can be pseudoexhaustively tested with  $2^n$  or fewer binary patterns if none of the outputs of the circuit is a function of more than  $w$  out of  $n$  inputs. For example, the following six test patterns can be used to pseudoexhaustively test a six-input circuit provided no output depends on more than two-input variables:

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

These test patterns can be generated by a non-primitive polynomial, for example,  $x^6+x^4+x^3+x^2+x+1$ . However, if an output of a multi-output circuit depends on more than two input variables, the derivation of minimal test patterns using a non-primitive polynomial may not be feasible.

In general, the pseudoexhaustive patterns needed to test an  $n$ -input and  $m$ -output combinational circuit are derived by using one of the following methods.

- Syndrome driver counter;
- Constant weight counter;
- Linear feedback shift register /SR (LFSR/SR); and
- LFSR/EX-OR gates (LFSR/EX-OR).

The *syndrome driver counter* method checks if  $p (<n)$  inputs of the circuit under test can share the same test values with the remaining  $(n-p)$  inputs [4]. Then the circuit can be exhaustively tested with  $2^p$  inputs.

### Pseudorandom Pattern Generator

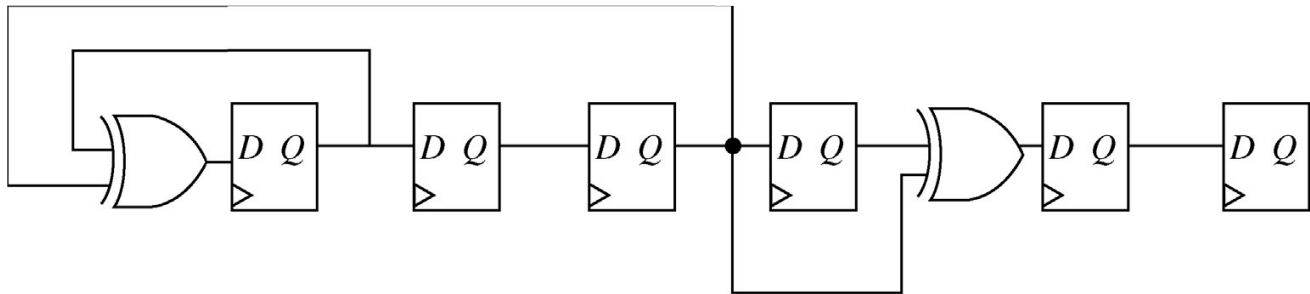


Fig.Convolved LFSR/SR

- Determination of the number of test patterns;
- Evaluation of the fault coverage; and
- Detection of random pattern-resistant faults.

The fault coverage can be evaluated by using exhaustive fault simulation. However, pseudorandom patterns needed to test a circuit are typically large; thus, fault simulation can be expensive. A relationship between a pseudorandom test sequence of length  $L$  and the expected fault coverage  $E(c)$  is given in [10]:

$$E(c) = 1 - \sum_{k=1}^{2^n-1} \left(1 - \frac{L}{2^n}\right)^k \frac{b_k}{M}$$

### Deterministic Testing

Traditional test generation techniques may also be used to generate test patterns that can be applied to the circuit under test when it is in BIST mode. The test patterns and the corresponding output responses are normally stored in a read only memory (ROM). If the output responses of the circuit under test do not match the expected responses when the stored test patterns are applied, the presence of a fault(s) is assumed. Although, in principle, this is a satisfactory approach for fault detection, it is rarely used because of the high overhead associated with storing test patterns and their responses.

## 6. BIST ARCHITECTURES

Over the years, several BIST architectures have been proposed by researchers in industry and universities. We discuss some of these in this section.

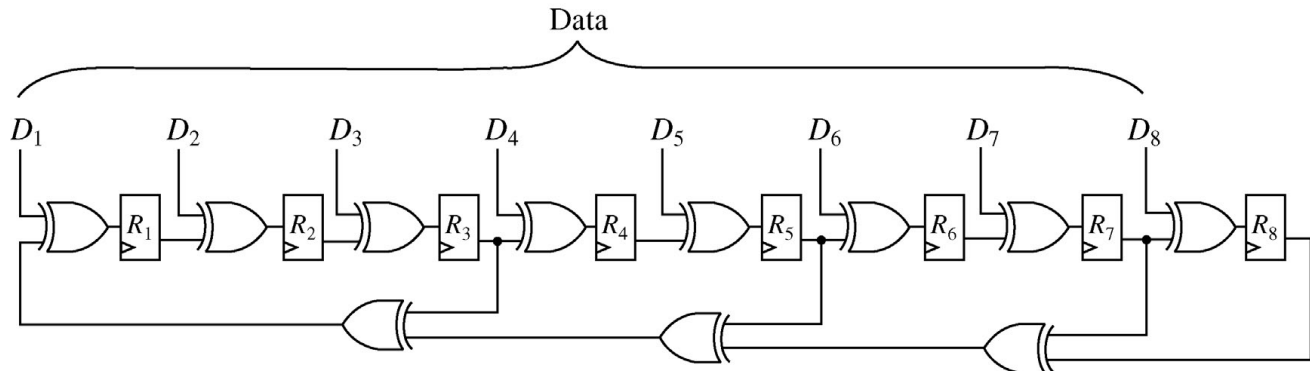


Fig.MISR

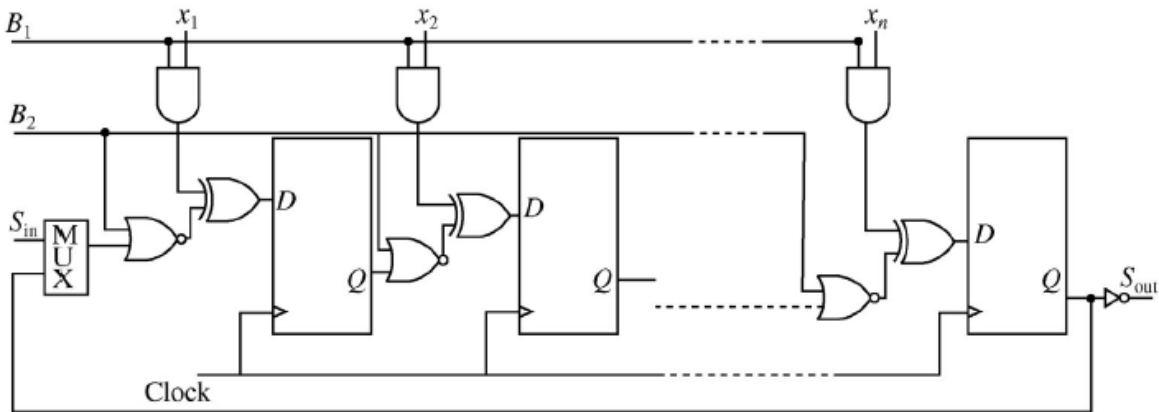
built-in logic block observer (BILBO) structure, both scan-path and signature analysis techniques are employed [18]. It uses a multipurpose module, called a BILBO, that can be configured to function as an input test pattern generator or an output signature analyzer. This is composed of a row of flip-flops and some additional gates for shift and feedback operations. Figure 4.23a shows the logic diagram of a BILBO. The two control inputs  $B_1$  and  $B_2$  are used to select one of the four function modes:

*Mode 1.*  $B_1=0, B_2=1$ . All flip-flops are reset.

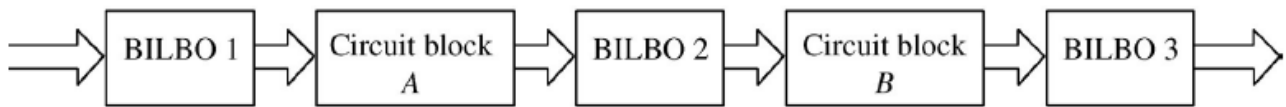
*Mode 2.*  $B_1=1, B_2=1$ . The BILBO behaves as a latch. The input data  $x_1, \dots, x_n$  can be simultaneously clocked into the flip-flops and can be read from the  $Q$  and  $\bar{Q}$  outputs.

*Mode 3.*  $B_1=0, B_2=0$ . The BILBO acts as a serial SR. Data are serially clocked into the register through  $S_{in}$ , while the register contents can be simultaneously read at the parallel  $Q$  and  $\bar{Q}$  outputs or clocked out through the serial output  $S_{out}$ .

*Mode 4.*  $B_1=1, B_2=0$ . The BILBO is converted into an MISR. In this mode, it may be used for performing parallel signature analysis or for generating pseudorandom sequences. The latter application is achieved by keeping  $x_1, \dots, x_n$  at fixed values.

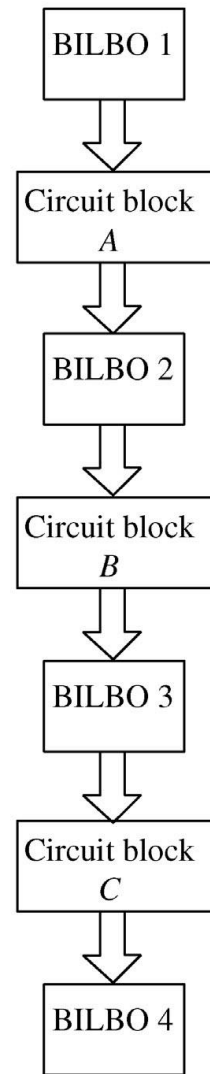
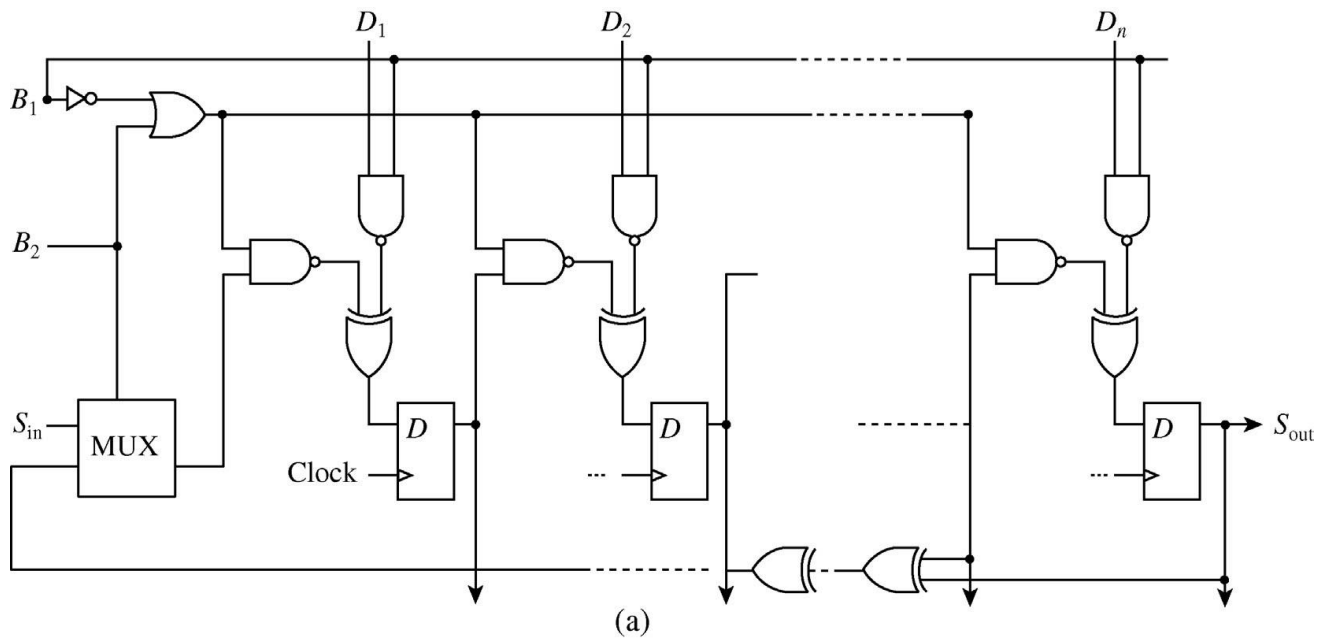


(a)



(b)

FIGURE 4.23: (a) Logic diagram of a BILBO. (b) BILBO-based BIST architecture.



$B_1$	$B_2$	Mode
0	0	Serial Scan
0	1	LFSR
1	0	Normal
1	1	MISR

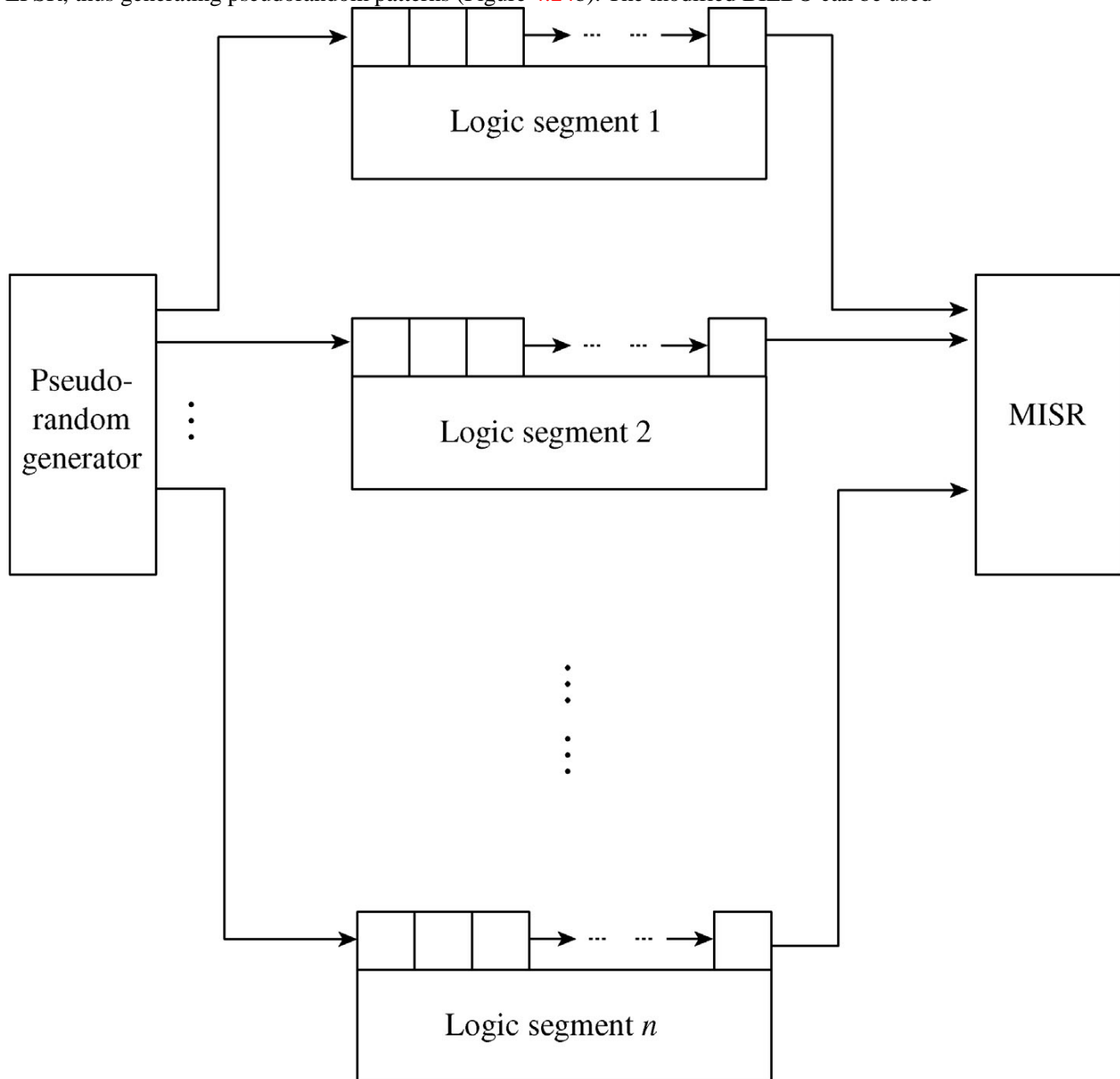
(b)

**Figure 4.24:** (a) Logic diagram of a modified BILBO. (b) Operating mode. (c) Simultaneous testing of pipeline structures.

Figure 4.23b shows the BILBO-based BIST architecture for two cascaded circuit blocks A and B. BILBO 1 in this structure is configured as a pseudorandom pattern generator, the outputs of which are applied as test inputs to circuit block A. BILBO 2 is configured as a parallel signature

register and receives its inputs from circuit block A. Similarly, BILBOs 2 and 3 should be configured to act as a pseudorandom pattern generator and a signature register, respectively, to test circuit block B. It should be clear that circuit blocks A and B cannot be tested in parallel, because BILBO 2 has to be modified to change its role during the testing of these blocks.

A modified version of the conventional BILBO structure is shown in Figure 4.24a [19]. In addition to normal, serial scan, and MISR function, the modified BILBO can also function as an LFSR, thus generating pseudorandom patterns (Figure 4.24b). The modified BILBO can be used

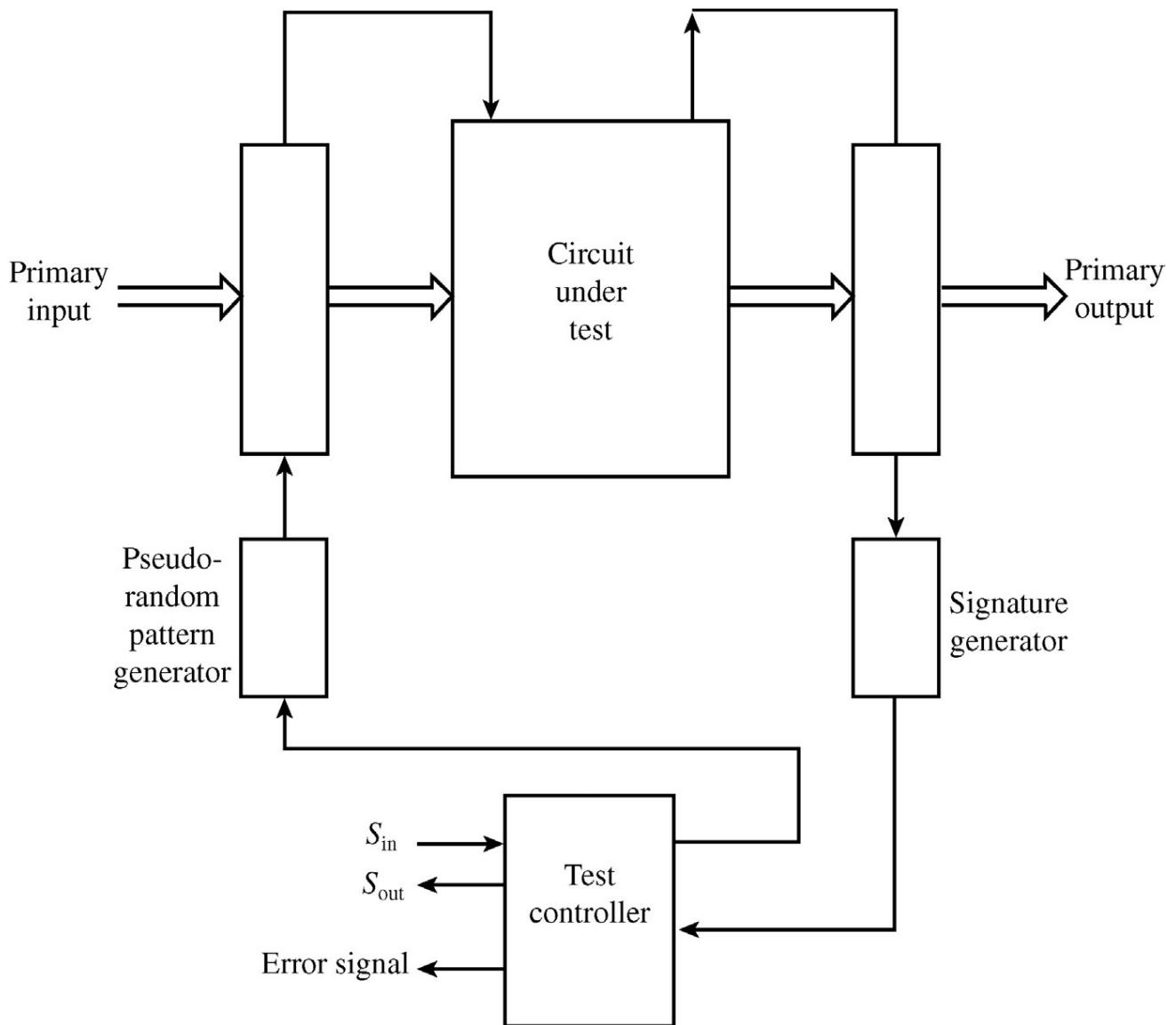


**Figure 4.25:** STUMPS configuration.

for simultaneous testing of pipeline structure. For example, in Figure 4.24c circuit, blocks A and C can be simultaneously tested by operating BILBOs 1 and 3 in the LFSR mode and BILBOs 2 and 4 in the MISR mode. Circuit block B can be tested individually by making BILBOs 2 and 3 operate in the LFSR and MISR modes, respectively.

### Self-Testing Using an MISR and Parallel Shift Register Sequence Generator

Self-testing using an MISR and parallel shift register sequence generator (STUMPS) uses multiple serial scan paths that are fed by a pseudorandom number generator as shown in Figure 4.25 [20]. Each scan path corresponds to a segment of the circuit under test and is fed by a pseudorandom number generator. Because the scan paths may not be of same length, the pseudorandom generator is run till the largest scan path is loaded. Once data has been loaded into the scan paths, the system clock is activated. The test results are loaded into the scan paths and then shifted into the MISR.



**Figure 4.26:** The LOCST configuration.

### LSSD on-Chip Self-Test

LSSD on-chip self-test (LOCST) combines pseudorandom testing with LSSD-based circuit structure [20]. The inputs are applied via boundary scan cells; also, the outputs are obtained via boundary scan cells. The input and the output boundary scan cells together with the memory elements in the circuit under test form a scan path. Some of the memory elements at the beginning of the scan path are configured into an LFSR for generating pseudorandom numbers. Also, some memory elements at the end of the scan path are configured into another LFSR, which functions as a signature generator. Figure 4.26 shows the LOCST configuration.

The test process starts by serially loading the scan path consisting of input boundary scan cells, with pseudorandom patterns generated by the LFSR. These patterns are applied to the combinational part of the circuit under test, and the resulting output patterns are loaded in parallel into the scan path consisting of output boundary scan cells. These output bits are then shifted into the signature register. The resulting signature is then compared with the reference signature for verification purposes.

### 7. OUTPUT RESPONSE ANALYSIS

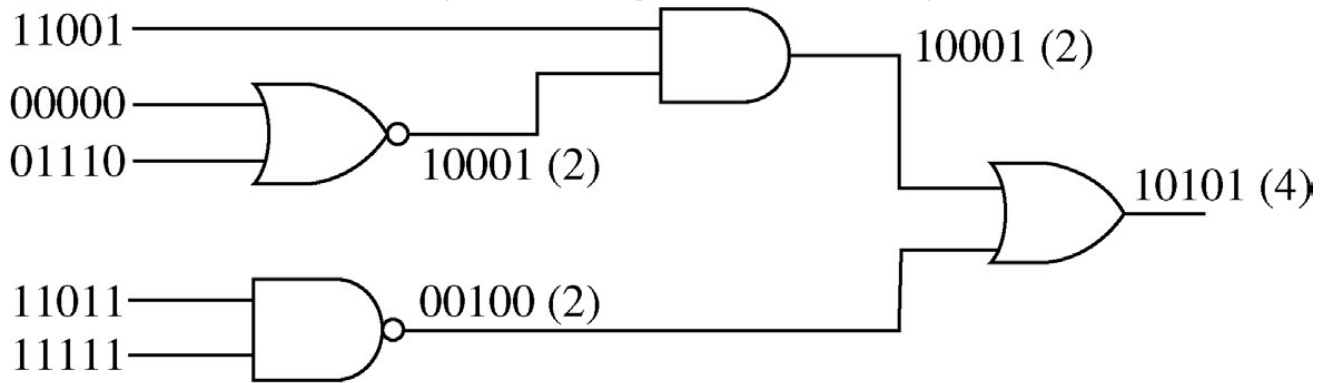
As stated earlier, BIST techniques usually combine a built-in binary pattern generator with circuitry for compressing the corresponding response data produced by the circuit under test. The compressed form of the response data is compared with a known fault-free response. Several compression techniques that can be used in a BIST environment have been proposed over the years; these include:

- Transition count;
- Syndrome checking; and
- Signature analysis.

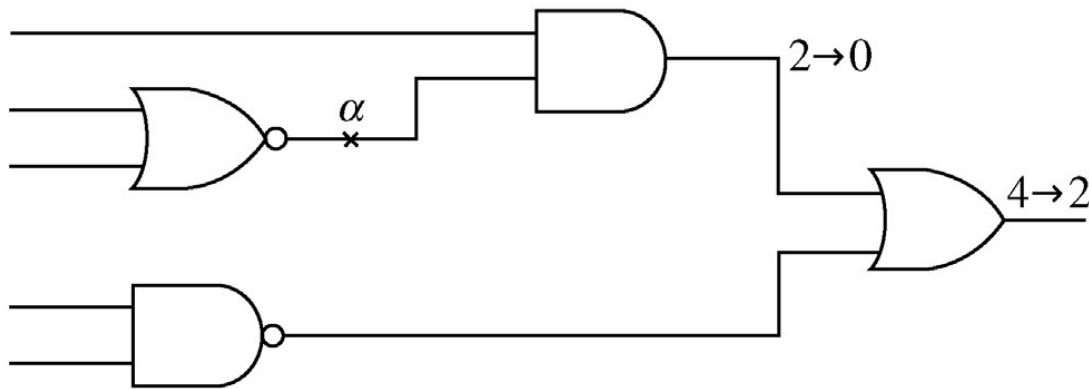
## Transition Count

The transition count is defined as the total number of transitions of  $1 \rightarrow 0$  and  $0 \rightarrow 1$  in an output response sequence corresponding to a given input test sequence. For example, if an output response sequence  $Z=10011010$ , then the transition count  $c(Z)=4$ . Thus, instead of recording the entire output response sequence, only the transition count is recorded. The transition count is then compared with the expected one, and if they differ, the circuit under test is declared faulty [15].

Figure 4.16a shows the response sequences and the corresponding transition counts at various nodes of a circuit resulting from the application of a test sequence of length 4. Let us suppose there is a fault  $\alpha$  s-a-0 in the circuit (Figure 4.16b). The presence of the fault changes the transition



(a)



(b)

**Figure 4.16:** (a) Response to test sequence of length 4. (b) Changes in transition counts.

counts at certain nodes in the circuit (shown by *arrows*). As can be seen in the diagram, the transition count at the output node changes from 4 to 2, resulting in the detection of the fault  $\alpha$  s-a-0.

The main advantage of transition counting is that it is not necessary to store the correct response sequence or the actual response sequence at any test point; only the transition counts are needed. Clearly, this results in the reduction of data storage requirements. However, this data compression may give rise to the *fault-masking errors*. This is because most transition counts correspond to more than one sequence; for example, the transition count 2 is generated by each of the following 6-bit sequences: 01110, 01100, 01000, 00110, 11011, and 10001. Hence, there is a possibility that a faulty sequence will produce the same transition count as the good sequence and therefore go undetected. However, as the sequence length increases, the hazard of fault masking diminishes.

## Syndrome Checking

The syndrome of a Boolean function is defined as  $S=K/2^n$ , where  $K$  is the number of min-terms realized by the function and  $n$  is the number of input lines [16]. For example, the syndrome of a three-input AND gate is  $1/8$  and that of a two-input OR gate is  $3/4$ . Because the syndrome is a functional property, various realizations of the same function have the same syndrome.

The input-output syndrome relation of a circuit having various interconnected blocks depends on whether the inputs to the blocks are disjoint or conjoint, as well as on the gate in which

the blocks terminate. For a circuit having two blocks with unshared inputs, if  $S_1$  and  $S_2$  denote the syndromes of the functions realized by the blocks 1 and 2, respectively, the input–output syndrome relation  $S$  for the circuit is:

TERMINATING GATE	SYNDROME RELATION $S$
OR	$S_1 + S_2 - S_1S_2$
AND	$S_1S_2$
EX-OR	$S_1 + S_2 - 2S_1S_2$
NAND	$1 - S_1S_2$
NOR	$1 - (S_1 + S_2 - S_1S_2)$

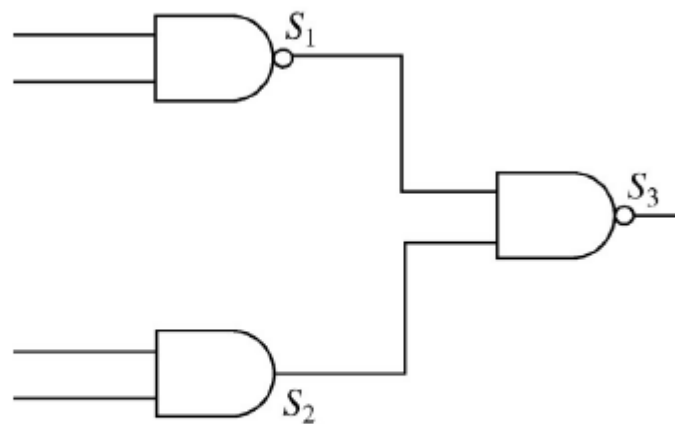
If blocks 1 and 2 have shared inputs and realize the functions  $F$  and  $G$ , respectively, then the following relations hold:

$$S(F + G) = S(F) + S(G) - S(FG),$$

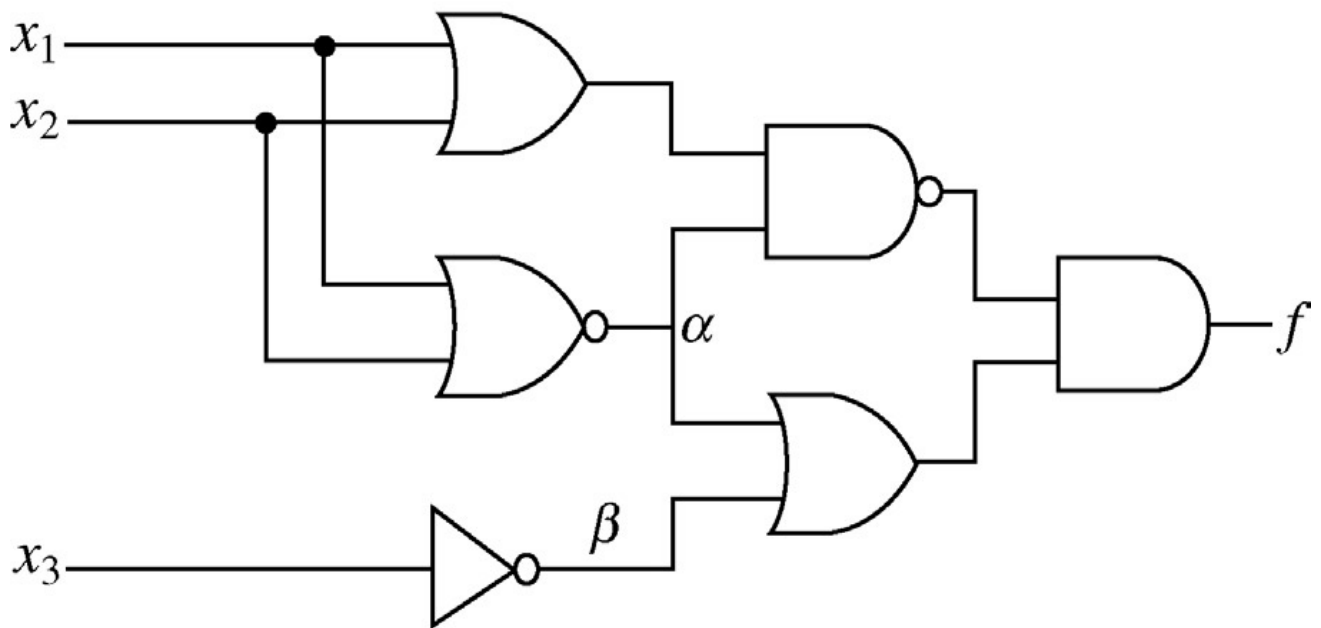
$$S(FG) = S(F) + S(G) - S(\overline{FG}) - 1,$$

$$S(F \oplus G) = S(\overline{FG}) + S(F\overline{G}).$$

As an example, let us find the syndrome and the number of min-terms realized by the fan-out-free circuit of Figure 4.17. We have  $S_1=3/4$  and  $S_2=1/4$ . Hence,  $S_3=1 - S_1S_2=13/16$ , and  $K=S_32^n=13$ . Table 4.2 lists the syndrome of the fault-free circuit of Figure 4.18, and the syndromes in the presence of fault  $\alpha$  s-a-0 and the fault  $\beta$  s-a-1.



**FIGURE 4.17:** A fan-out-free circuit.

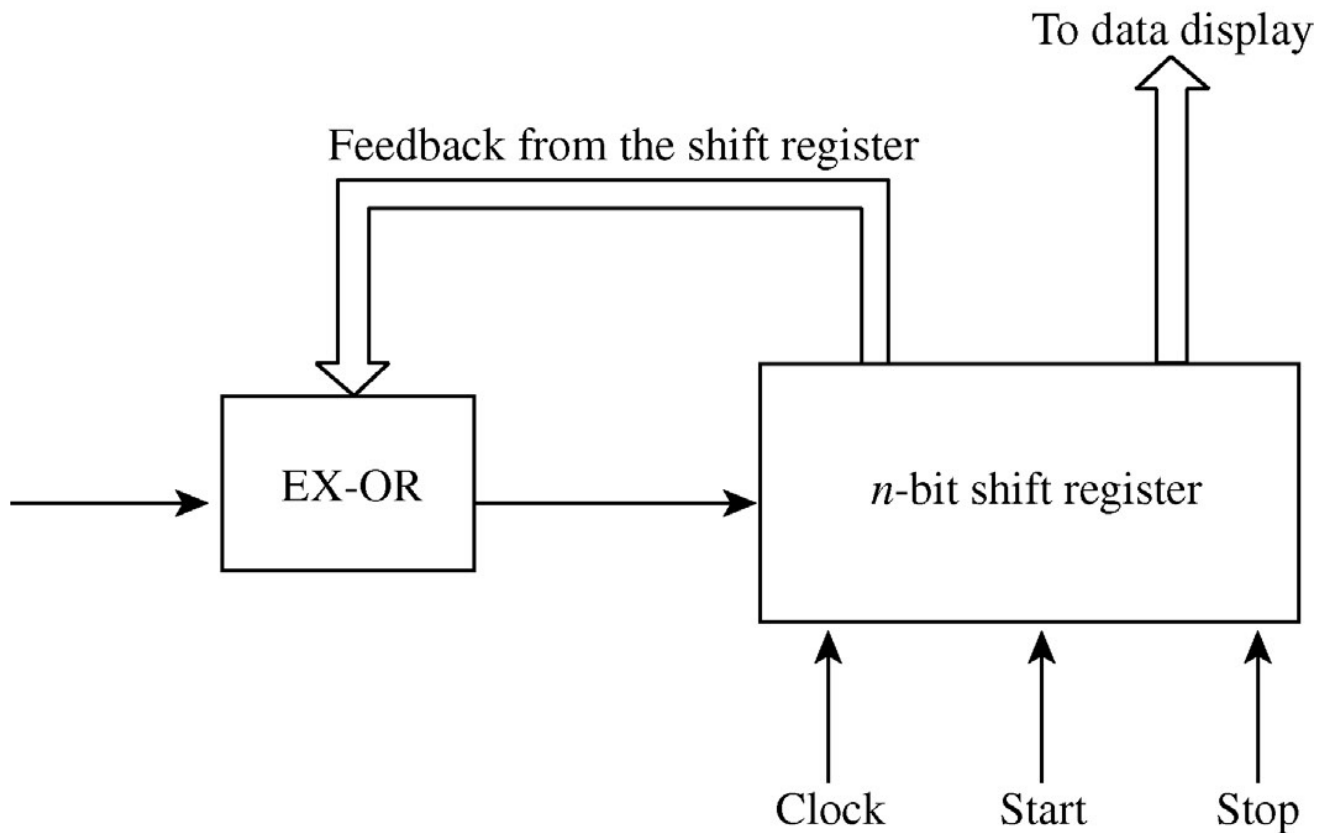


**Figure 4.18:** Circuit under test.

### Signature Analysis

Signature analysis technique is pioneered by Hewlett-Packard Ltd. that detects errors in data streams caused by hardware faults [17]. It uses a data compaction technique to reduce long data streams into a unique code called the *signature*. Signatures can be created from the data streams by feeding the data into an  $n$ -bit LFSR. The feedback mechanism consists of EX-ORing selected taps

TABLE 4.2: Fault-free and faulty syndromes			
$x_1x_2x_3$	OUTPUT RESPONSE		
	<i>FAULT-FREE</i>	$\alpha$ s-a-1	$\beta$ s-a-0
000	1	1	1
001	1	1	1
010	1	0	0
011	0	0	0
100	1	0	0
101	0	0	0
110	1	0	0
111	0	0	0
Syndrome	5/8	2/8	2/8



**Figure 4.19:** Signature analyzer circuit.

of the SR with the input serial data as shown in Figure 4.19. After the data stream has been clocked through, a residue of the serial data is left in the SR. This residue is unique to the data stream and represents its signature. Another data stream may differ by only 1 bit from the previous data stream, and yet its signature is radically different from the previous one. To form the signature of a data stream, the SR is first initialized to a known state and then shifted using the data stream; normally, the all-0 state is chosen as the initial state.

Figure 4.20a shows a simplified 4-bit signature generator. Assuming the content of the register is all 0, if a 1 is applied to the circuit, the EX-OR gate will have output 1. The next clock pulse will shift the gate output into the first stage of the register and 0s from the preceding stages into the second, third, and fourth stages, which leaves the register containing 1000, i.e., in state 8. From the state diagram of Figure 4.20b, the register contents or signatures can be identified for any data stream.