

TEST GENERATION FOR SYNCHRONOUS SEQUENTIAL CIRCUITS

More difficult than combinational circuits

Usually requires test sequence.
(rather than single input vector)

Response depends on initial state
(Initialization desirable).

Propagation of errors through
memory elements (flip-flops)

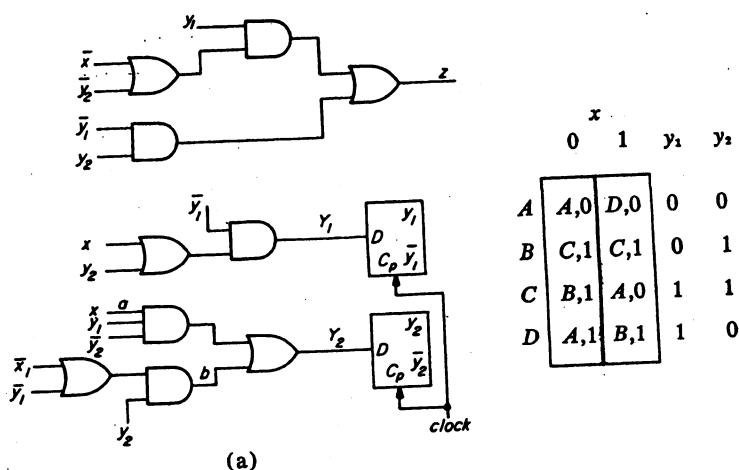
Let R_0 be set of possible responses
to test sequence T of normal circuit,
 R_α be set of possible responses
to T of circuit with fault α

T strongly detects α if and only if
(iff)

$$\{R_\alpha\} \cap \{R_0\} = \emptyset$$

(no sequence in R_0 is also in R_α)

T strongly distinguishes α and β
if $\{R_\alpha\} \cap \{R_\beta\} = \emptyset$

Example

Initial State	Output sequence			δ
	Normal	α (a s-a-1)	β (b s-a-0)	
A	01011	01100	01011	01101
B	11100	11100	11011	11101
C	00011	00011	01010	01010
D	11001	11001	11010	11010

$T = 10111$

Figure 3.1 Output sequences as a function of initial state and fault.

T does not strongly detect α
(detects if initial state is A but not if B, C, D)

T does not strongly detect β
(if initial state is A)

T does strongly detect δ
(However specification of error symptom is very complex)

T does ~~not~~ strongly distinguish α and β
(no identical output sequences)

Less general definition of test

For a sequential circuit C and an input sequence T and a fault α

T detects α if the output sequence of the normal and faulty circuits at time t are always different for some fixed $t \leq n$ where n is the length of T ($|T| = n$).

We will develop algorithms which can be used for synchronous sequential circuits and stuck at faults α to derive tests which detect α

Sequential Circuit $\rightarrow$ Logic Array Correspondence

sc 4

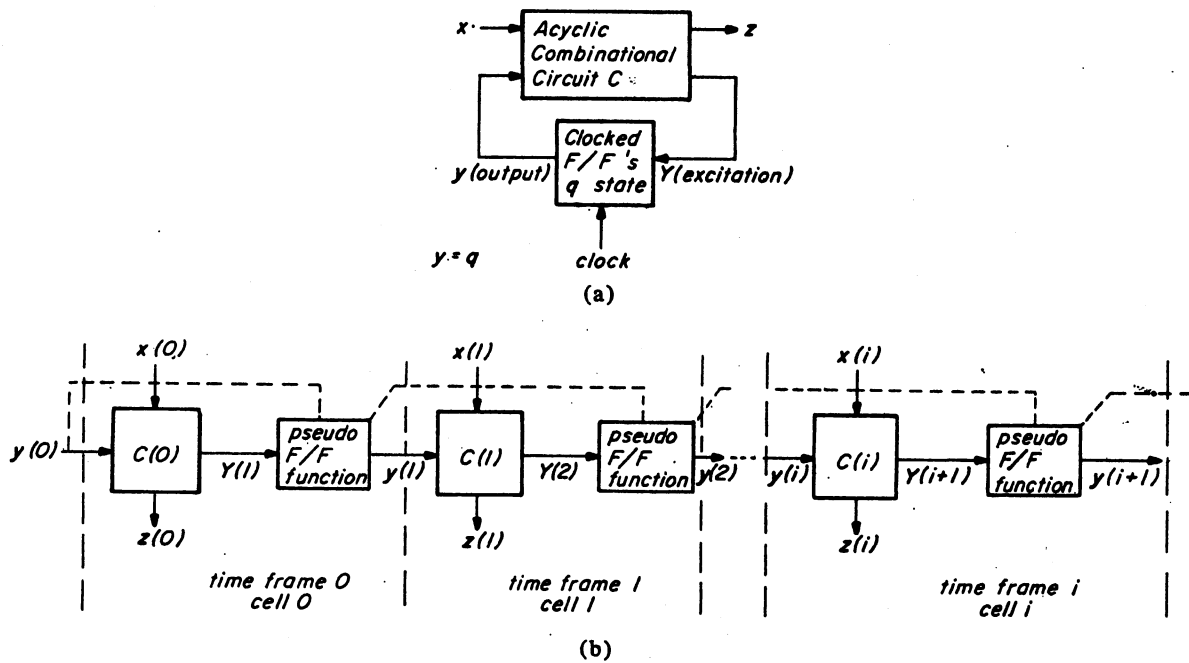


Figure 3.2 (a) General form of a synchronous sequential circuit S (b) Equivalent combinational iterative array

Every (time) input-output sequence in the sequential circuit corresponds to a (space) input-output sequence in the corresponding iterative array.

A stuck fault (single) in ^{C of} the sequential circuit corresponds to a (multiple) stuck fault in each $C(i)$ of the corresponding iterative array.

Array is combinational circuit.

q	J	K	q^+	y	$\bar{y}$
0	0	0	0	0	1
0	0	1	0	0	1
0	1	0	1	1	0
0	1	1	1	1	0
1	0	0	1	1	0
1	0	1	0	0	1
1	1	0	1	1	0
1	1	1	0	0	1

(a)

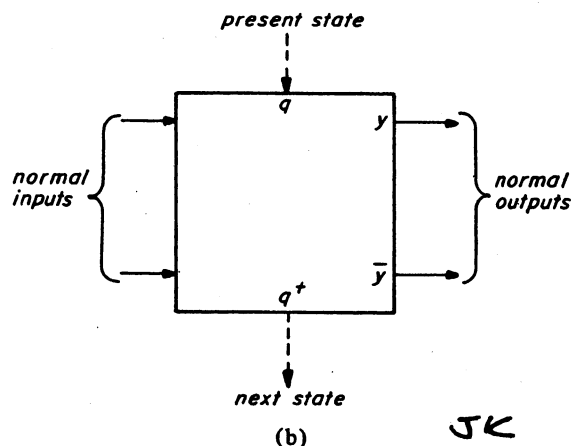


Figure 3.3 Representation for a pseudo flip-flop

$$Y = q^+ = J\bar{y} + \bar{K}y \quad (\text{Characteristic Equation for JK Flip-Flop!})$$

Using 4 valued ($0, \bar{0}, 0, 1$) Boolean Algebra
can determine propagation through
FF

$$(y=0, \bar{y}=1, J=D, K=0) \quad Y = J\bar{y} + \bar{K}y \\ = D \cdot 1 + 1 \cdot 0 = D$$

$$(y=0, \bar{y}=1, J=D, K=1) \quad Y = J\bar{y} + \bar{K}y \\ = D \cdot 1 + 1 \cdot 1 = D + 1 = 1$$

$$(y=D, \bar{y}=\bar{D}, J=D, K=D) \quad Y = J\bar{y} + \bar{K}y \\ = D \cdot \bar{D} + \bar{D} \cdot D = 0 + 0 = 0$$

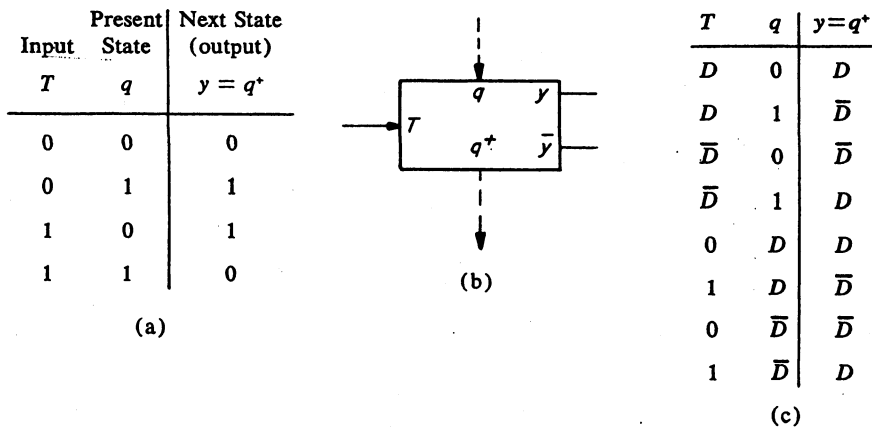


Figure 3.4 Pseudo T flip-flop (a) truth table (b) representation (c) propagation D -cubes

*Note that the propagation of D 's through a T , D , or JK flip-flop can be determined from the characteristic equations of the flip-flops.

$$y = T \oplus y = \bar{T} \cdot y + T \cdot \bar{y}$$

Procedure 3.1 (*Test generation algorithm for synchronous circuits*):

1. Construct the iterative array model of Figure 3.2(b). Set $y(0) \leftarrow y_1$, and set $k \leftarrow 0$.
2. Ignoring all output lines except for those in $z(k)$, apply the combinational circuit D algorithm (Procedure 2.2)*. Restrict each state vector $y(j)$, $j = 1, 2, \dots, k$ to be unique. If no test is found, increment k by one and repeat (2). Terminate when $k > 4^n$, where n is the number of state variables (flip-flops). If no test is found the circuit is redundant. $\square$

Note that each component of the state vector $y(i)$ will have one of five values, namely $(0, 1, \times, D, \bar{D})$. If a test exists, the $\times$'s can be replaced by 0's and 1's, hence only four values need be considered. It is clear that in testing a circuit it is never necessary to enter the same state twice, hence each $y(i)$ can be restricted to be unique, and there are only 4^n such unique states, where n is the number of state variables.

We will now present the extension of the D algorithm for generating a test to detect a fault α in a synchronous circuit, assuming a composite initial state vector y_1 .

In general, when choices exist in this procedure, it has been found advantageous to first select those alternatives which drive D 's ($\bar{D}$'s) into the memory elements, i.e., it is easier to generate an incorrect output ($z = D$ or $\bar{D}$) from an incorrect state than from the correct state.

In propagating a fault in a sequential circuit, a signal which has the value D (or $\bar{D}$) may propagate onto a wire which is itself faulty. This is the same situation which occurred in generating tests for multiple faults in combinational circuits. As we have previously observed a single fault in a sequential circuit is logically equivalent to a multiple fault in the corresponding iterative array.

Example 3.2: Consider the sequential circuit of Figure 3.5(a) and the gate input fault α (a s - a -1) on G_4 . We will derive a test to detect this fault, assuming the initial state is $y_1 = y_2 = 0$.

Time frame 0: Since $y_2 = 0$ and a is s - a -1 we have a $\bar{D}$ at the location of the fault. Setting $x(0) = 1$ allows this $\bar{D}$ to propagate through G_4 and since $G_3 = 0$, we produce the next state vector $(0, \bar{D})$.

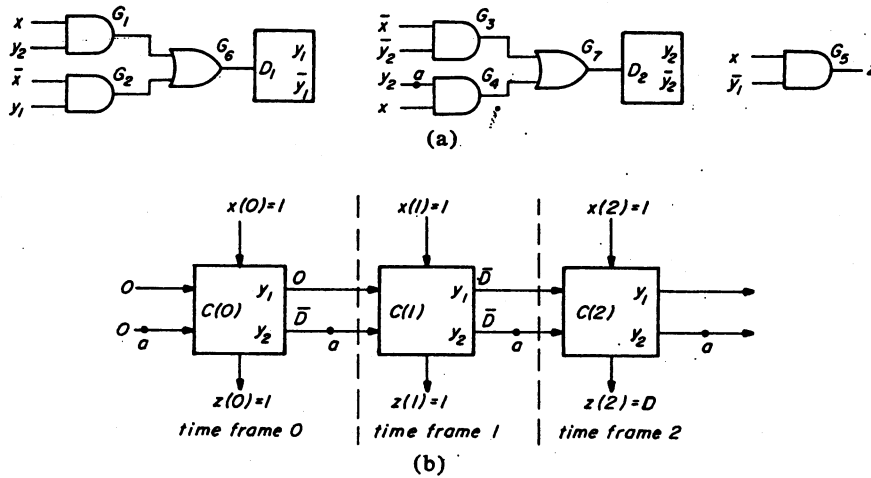


Figure 3.5

Time Frame 1: We cannot set $z = D$ or $\bar{D}$, hence we set $x(1) = 1$ and enter the state $(\bar{D}, \bar{D})$.

Time Frame 2: Since $\bar{y}_1 = D$, we set $x(2) = 1$ and obtain $z = D$ and hence a test sequence exists which detects this fault, namely $X = 111$. $\square$

Example 6.18: Consider the sequential circuit in Figure 6.75(a) and the fault a s-a-1. We will derive a test sequence to detect this fault, assuming the initial state is $q_1=q_2=0$. Figure 6.75(b) shows the model of one time frame.

Time frame 1: We apply $q_1=q_2=0$, which creates a $\bar{D}$ at the location of the fault. The vector $I(1)=1$ propagates the error to q_2^+ . Because the fault effect does not reach Z , we save the sequence $S_1 = (1)$ and the state $q^+(1) = (0, \bar{D})$ in the set SOL_1 .

Time frame 2: We apply $(0, \bar{D})$ to the q lines of frame 2. The D -frontier is $\{G_1, G_3, G_4\}$. Selecting G_1 or G_4 for error propagation results in $I(2)=1$ and the next state $q^+(2)=(\bar{D}, \bar{D})$. Trying to propagate the error through G_3 results in $I(2)=0$ and the next state $q^+(2)=(0, D)$.

Time Frame 3: Now the set SOL_2 contains two partial solutions: (1) the sequence $(1,1)$ leading to the state $(\bar{D}, \bar{D})$ and (2) the sequence $(1,0)$ leading to the state $(0, D)$. Trying the first solution, we apply $(\bar{D}, \bar{D})$ to the q lines of frame 3, which results in the D -frontier $\{Z, G_1, G_2, G_3, G_4\}$. Selecting Z for error propagation we obtain $I(3)=1$ and $Z=D$. The resulting test sequence is $I=(1,1,1)$. $\square$

With the given initial state, it may not always be possible to activate the fault in the first time frame; then a sequence is needed for activation.

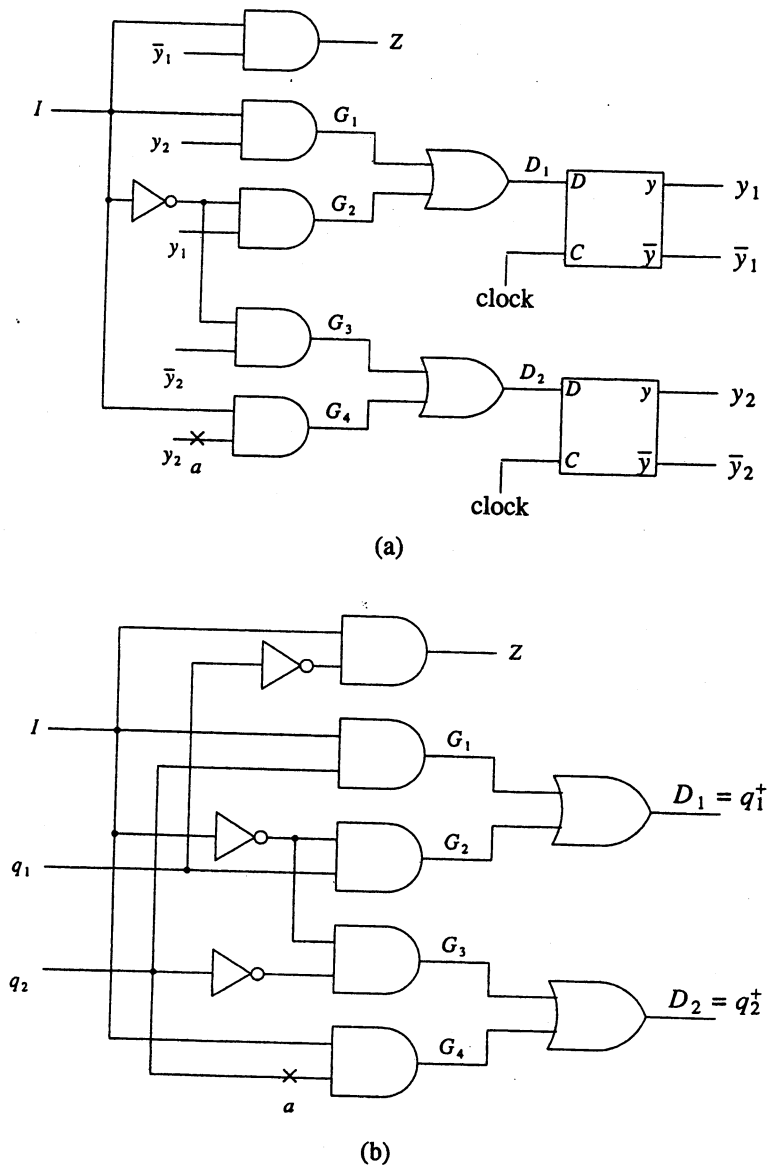


Figure 6.75

INITIALIZATION

SC 10.

In general, the problem of circuit initialization or synchronization deals with finding an input sequence which will drive a circuit, starting in an unknown state, to a fixed state. In test generation it is usually desirable to initialize not only the fault free circuit, but also the faulty ones. Of course, these circuits need not be driven to the same state.

We will first describe a common heuristic procedure used for solving this problem, which employs a new logic value denoted by u to represent the unknown state of a line.

3-VALUED ALGEBRA $\rightarrow$ 5 VALUED

$\{0, 1, \underline{u}\}$
unknown

$\{0, 1, u, D, \bar{D}\}$
 $\begin{matrix} \swarrow \text{Normal} \\ 0 \end{matrix}$
 $\begin{matrix} \swarrow \text{Faulty} \\ 1 \end{matrix}$

$\cdot$	0	1	u
0	0	0	0
1	0	1	u
u	0	u	u

$+$	0	1	u
0	0	1	u
1	1	1	1
u	u	1	u

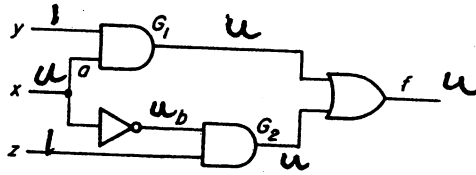


Figure 3.11

Example which fails

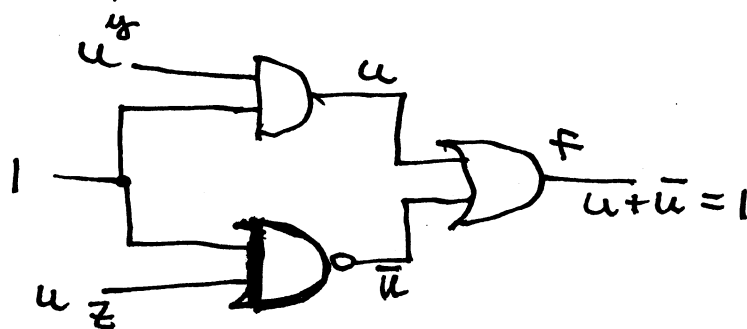
Predicts $f = u$

However if $x = 0 \rightarrow f = 1$
and if $x = 1 \rightarrow f = 1$

Reason $G_1 = \bar{b}$

Use 4 valued algebra $(0, 1, u, \bar{u})$

Works for above circuit.
But not for following one



If $y = 0, z = 1 \rightarrow f = 0$

Must have different u ($u_1, u_2, \dots$)
etc. for each independent
variable whose value is unknown

Impractical.

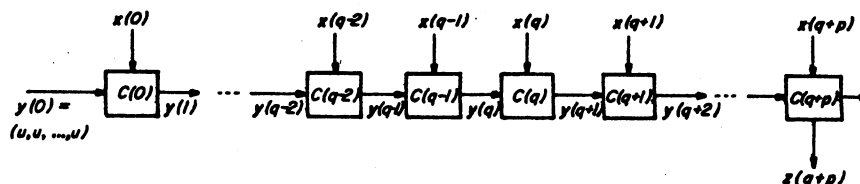


Figure 3.9 General form of computational model for test generation for synchronous circuits requiring self initialization sequences

procedure 3.2 (Construction of a self initializing test sequence):

- (1) Determine the maximum values P and Q for p and q . Usually P and Q are user specified, where $P + Q \leq n^s$ and n is the number of state variables.
- (2) Select initial values for p and q (usually $p = q = 0$).
- (3) Construct the corresponding iterative array model corresponding to p and q , and attempt to generate a test for the fault α insuring that $y(0)$ retains the value $y_i = (u, u, \dots, u)$. If a test is found exit; otherwise continue.
- (4) Select the next set of values for p and q , if they exist, and return to step 3, otherwise exit with no test. $\square$

Example 3.3: Consider the circuit shown in Figure 3.10(a) and the fault α (a s-a-0). We first attempt a solution with $p = 0$ (the fault is on an output line) and $q = 1$.

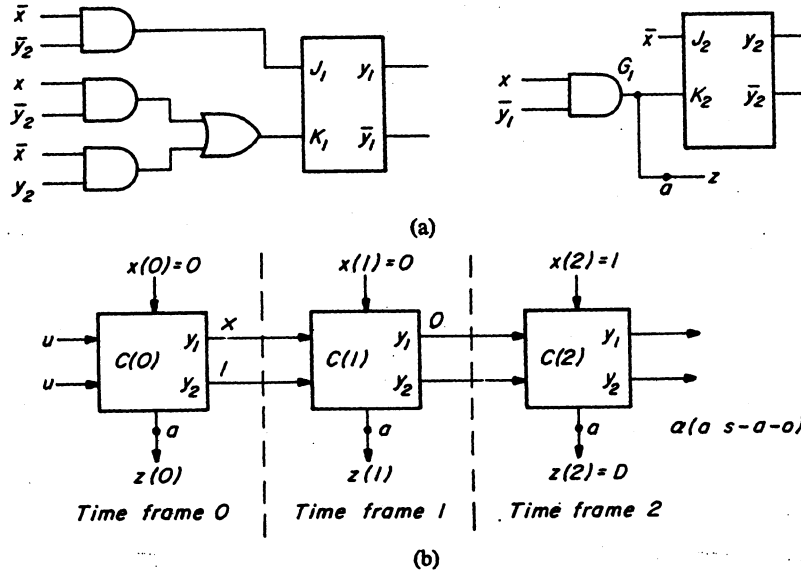


Figure 3.10

Time frame 1: To detect α we require $G_1 = 1$, hence $x(1) = 1$ and $\bar{y}_1(1) = 1$. This produces $z = D$ and a D has been propagated to the output using $p = 0$. (Note that if $q = 0$, since $y_1(0) = y_2(0) = u$ the condition $\bar{y}_1(0) = 1$ cannot be satisfied.)

Time frame 0: For this time frame we have that the present (initial) state of y_1 is u , and its next state must be 0. To achieve this condition we can attempt to justify the line settings $J_1(0) = 0$ and $K_1(0) = 1$. This can be done by setting either

- (a) $x(0) = 1$ and $\bar{y}_2(0) = 1$ or
- (b) $x(0) = 0$ and $y_2(0) = 1$.

However, since $y_2(0) = \bar{y}_2(0) = u$, no test exists.

We must therefore attempt a solution using $q = 2$. The iterative array for this case is shown in Figure 3.10(b).

Time frame 1: We can now justify the condition $y_1(2) = 0$ by either $J_1(1) = 0$, $K_1(1) = 1$ or by assuming $y_1(1) = 0$. We first try the former condition. To satisfy this condition we will first try condition (a) stated previously, i.e., we attempt to justify $\bar{y}_2(1) = 1$.

Time frame 0: To justify $\bar{y}_2(1) = 1$ we require $J_2(0) = 0$ (hence $x(0) = 1$) and $K_2(0) = 1$. But the output of G_1 cannot be set to 1 since $\bar{y}_1(0) = u$. We therefore must backtrack to an earlier choice.

Time frame 1: We next try to justify the setting $J_1(1) = 0$ and $K_1(1) = 1$ by setting $y_2(1) = 1$, as shown in Figure 3.10(b).

Time frame 0: Now by setting $x(0) = 0$ we obtain $J_2(0) = 1$ and $K_2(0) = 0$ and hence $y_2(1) = 1$ independent of the initial state of $y_2(0)$.

The generated self initializing test sequence is $X = 001$, and requires $q = 2$ and $p = 0$. $\square$

Example 6.19: Let us derive a self-initializing test sequence for the fault $Z s-a-0$ in the circuit of Figure 6.77(a). Figure 6.77(b) shows the structure of one time frame, using a module to realize the function $q^+ = J\bar{q} + q\bar{K}$. The primitive cubes of this module are given in Figure 6.69(b) (substitute J, q, K, q^+ for a, b, c, Z).

Time frame 1: The only test that detects $Z s-a-0$ is $(I, q_1)=10$. Since Z has value D , an error is propagated to the PO using $r=1$ frames. Since $q_1(1) \neq x$, we need $p \geq 1$.

Time frame 0: To justify $q_1^+(0)=0$, we first try $J_1=0$ and $K_1=1$ (see the decision tree in Figure 6.78(a)). Both solutions justifying $J_1=0$ involve assignments to q_2 . The other two solutions justifying q_1^+ involve assignments to q_1 . Hence there is no solution with

$p=1$. Next we try $p=2$ and we return to the first solution for $J_1=0$, namely $I=0$ and $q_2=1$.

Time frame -1: To justify $q_2^+=1$, we first try $J_2=1$ and $K_2=0$, which are both satisfied by setting $I=0$. Since all lines are justified while q_1 and q_2 of this time frame are both

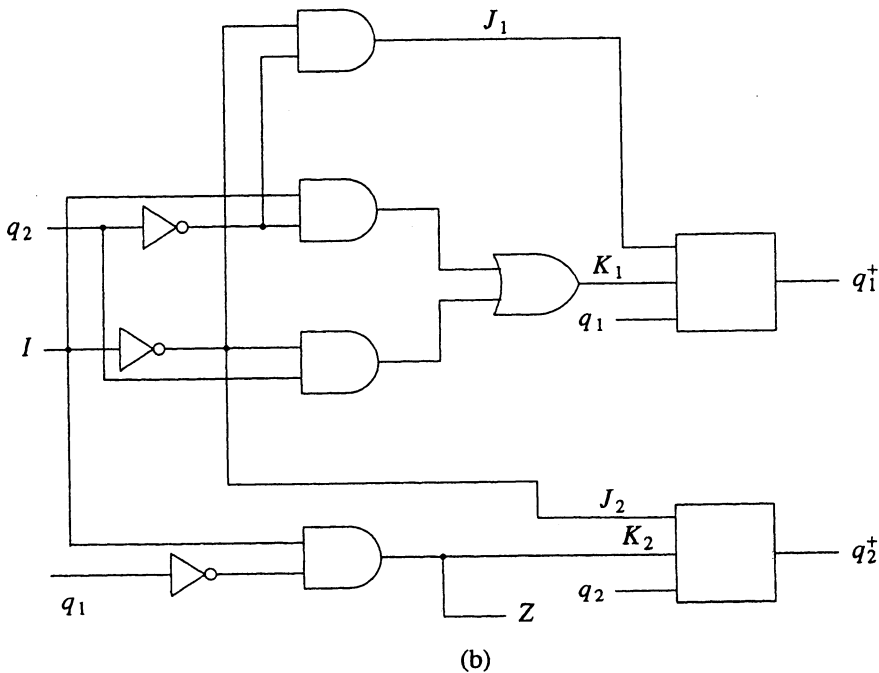
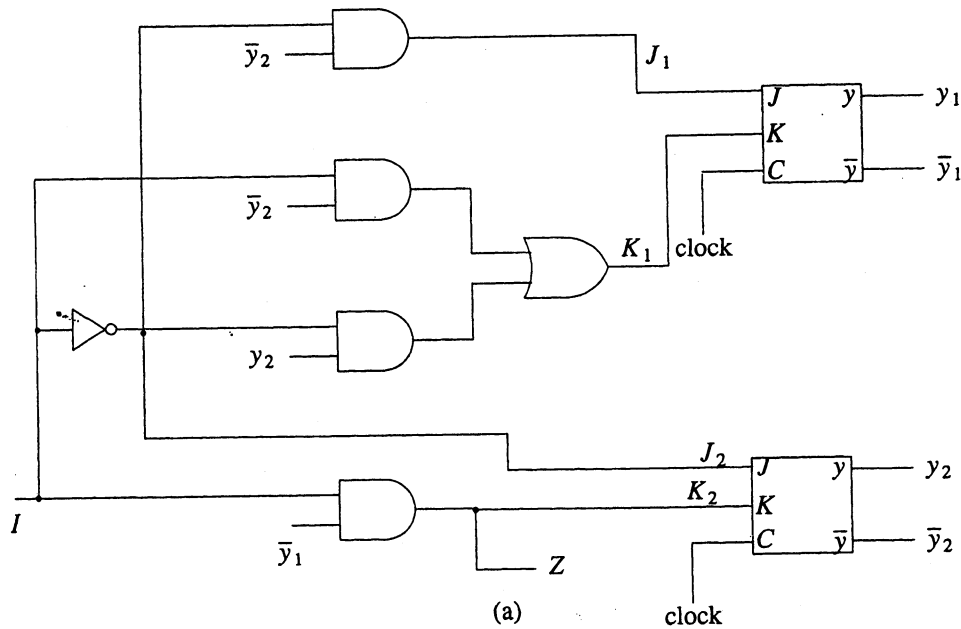


Figure 6.77

x , we have obtained the self-initializing test sequence $I=(0,0,1)$ with $p=2$ and $r=1$. Figure 6.78(b) shows the corresponding iterative array. $\square$

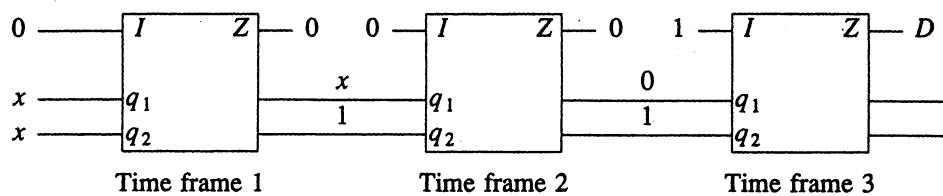
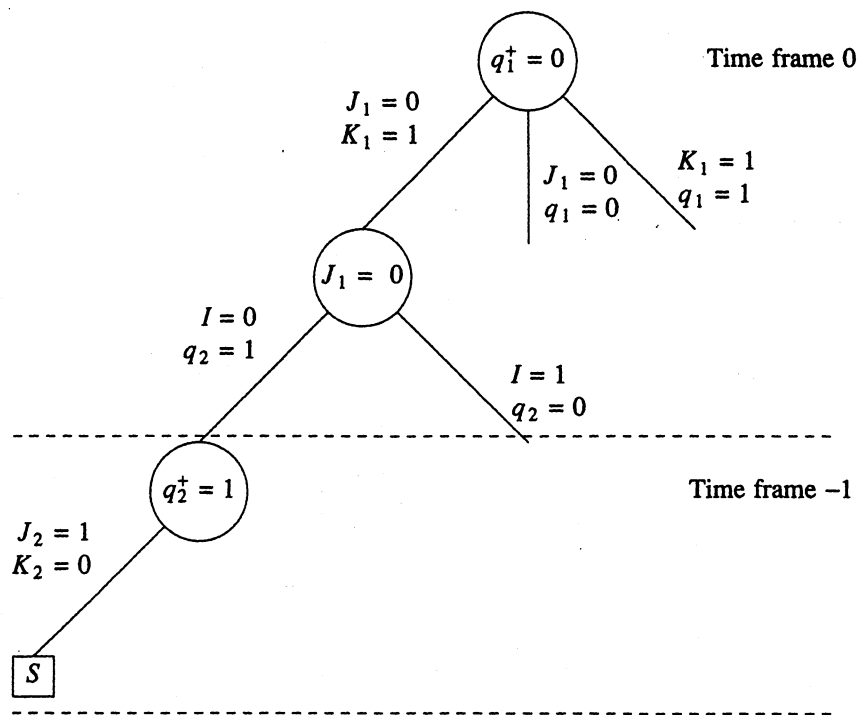


Figure 6.78 Example 6.19 (a) Decision tree (b) Iterative array

Fault Collapsing in Sequential Circuits

Simple results may not work

Consider the portion of the circuit shown in Figure 3.34. Assume that initially $y = 1$ and consider the test sequence shown in the figure. For the normal circuit the output sequence generated is $Z = 0000$. For the fault $\alpha(x_2 s-a-1)$ the first input does not reset y and the fourth input propagates this error to z . Thus the generated output sequence is 0001 and $x_2 s-a-1$ is detected. Now consider the same test sequence and the fault $\beta(G_1, s-a-0)$, which dominates α in a combinational circuit sense. The first input again does not reset y . However, the fourth input generates an erroneous 0 at G_2 and the two effects of this single fault cancel each other ($z = D \cdot \bar{D} = 0$) and the fault is not detected. Thus fault collapsing techniques based on fault dominance on gates (such as Theorems 2.2 and 2.3) are not applicable to sequential circuits. Of course collapsing techniques based on fault equivalence on gates can still be used.

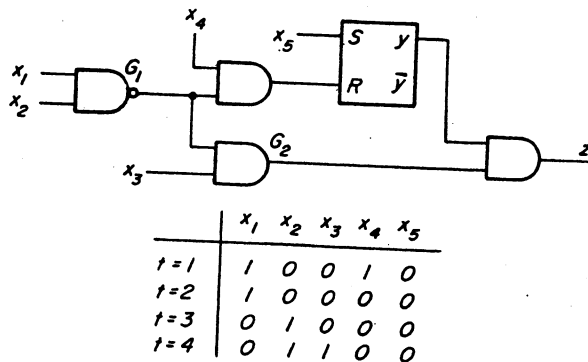


Figure 3.34

Functional Testing

F1

3.3.1 Functional Testing

Functional testing refers to the selection (usually manual), of tests which will validate the functional operation of a device. For example, a functional test for a flip-flop may consist of the following:

- (1) Validate that the flip-flop can be set (0 to 1 transition), and reset (1 to 0 transition).
- (2) Validate that the flip-flop can hold its state.

- 2.18** A decoder is a combinational circuit containing n inputs and 2^n outputs, labeled respectively $(x_0, x_1, \dots, x_{n-1})$, and $(z_0, z_1, \dots, z_{2^n-1})$. The circuit functions as follows: If $(a_0, a_1, \dots, a_{n-1})$ is a binary input vector, representing the integer k , then $z_k = 1$ and $z_j = 0$ for all $j \neq k$. Based upon the functional operation of this device, determine a minimal set of tests which will detect any single fault on an input or output line of this device.

2.19 A multiplexer is a combinational circuit C having the structure shown in Figure 2.60.

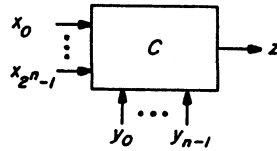


Figure 2.60

If $(a_0, a_1, \dots, a_{n-1})$ is an input vector, representing the integer k , applied to $(y_0, y_1, \dots, y_{n-1})$, then $z = x_k$, where $0 \leq k \leq 2^n - 1$. That is, input line x_k is connected to the output z . Based upon the functional operation of this device, determine a minimal set of tests which will detect any single stuck fault on an input or output line of this device.

3.3.2 Checking Sequences

A more formal approach to functional testing which has application to sequential circuit testing and the related problem of circuit verification is the derivation of an input/output sequence which distinguishes a given n -state sequential machine M from all other machines with the same inputs, outputs and *at most* n states. The existence of such sequences is based on the following theorem [22].

Theorem 3.2: For any reduced strongly connected n -state sequential machine M , there exists an input-output sequence pair $T = (I, O)$ which can be generated by M but cannot be generated by any other sequential machine M' with n or fewer states. $\square$

Such an input-output sequence is called a *checking sequence*. Note that the checking sequence only depends on the state table of M and is independent of the particular circuit realization and fault model assumed, with the restriction that no fault increases the number of states. Such a sequence

can be used for testing for classical and non-classical faults, as well as for circuit verification, and constitutes a *functional test* in the strictest sense of the term. However, since it requires the complete state table to be generated, its usefulness is limited to very small circuits. Since checking sequences are primarily of theoretical interest we will only consider them briefly.

We will show how they work
(Not how they are derived).

Example 3.12: Consider the following input-output sequence generated by a 3-state machine having one input x and one output z . Denote the states of this machine as q_1, q_2, q_3 .

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Input Sequence:	0	0	0	0	0	1	0	0	1	0	0	0	1	0	0
Output Sequence:	0	1	0	0	1	1	1	0	0	1	0	0	0	0	1

Let us denote the state when the i th input was applied as $S(i)$. We arbitrarily label the initial state $S(1)$ as q_1 . Since $S(2)$ responds to a 0 input with a 1 output while $S(1)$ responds to the same input with a different output, $S(2) \neq S(1)$ and we arbitrarily label $S(2)$ as q_2 . Therefore, $N(q_1, 0) = q_2$, $Z(q_1, 0) = 0$, and $Z(q_2, 0) = 1$. Similarly, $S(3) \neq S(2)$ and since $S(3)$ responds to the input sequence 00 with an output sequence 00 while $S(1)$ responds to the same input sequence with the output sequence 01, $S(3) \neq S(1)$, and hence $S(3)$ must be q_3 (since the circuit has at most 3 states). Therefore, $N(q_2, 0) = q_3$ and $Z(q_3, 0) = 0$. In a similar manner $S(4) \neq S(2)$ and $S(4) \neq S(3)$. Since the machine has only three states, $S(4) = S(1) = q_1$ and hence $N(q_3, 0) = q_1$. At this stage the partial state table of Figure 3.33(a) has been derived. From this table we can deter-

	x	
	0	1
q_1	$q_2, 0$	
q_2	$q_3, 1$	
q_3	$q_1, 0$	
	N, Z	

(a)

	x	
	0	1
q_1	$q_2, 0$	$q_3, 0$
q_2	$q_3, 1$	$q_1, 0$
q_3	$q_1, 0$	$q_2, 1$
	N, Z	

(b)

Figure 3.33 State table corresponding to given input-output sequence

mine that $S(5) = q_2$, $S(6) = q_3$ and $S(7) = q_2$ (since the 0 input applied to $S(7)$ produces a 1 output). Therefore, $N(q_3, 1) = q_2$ and $Z(q_3, 1) = 1$. Similarly, $S(8) = q_3$, $S(9) = q_1$, and $S(10) = q_2$. Therefore $N(q_1, 1) = q_2$ and $Z(q_1, 1) = 0$. Similarly $S(11) = q_3$, $S(12) = q_1$, $S(13) = q_2$, and $S(14) = q_1$. Therefore, $N(q_2, 1) = q_1$ and $Z(q_2, 1) = 0$ and the sequential machine of Figure 3.33(b) has been derived as the only three-state machine which can generate this input-output sequence. $\square$

Testing of Random Access Memories ^{F6}

In some technologies memory must be periodically "refreshed"

Memory Structure

Figure 3.37 shows the typical functional circuit configuration for a RAM. Assume the RAM has an n bit capacity, i.e., we can read and/or write a bit of information into n different locations, called cells, each randomly addressable. This requires s address bits where $s = \lceil \log_2 n \rceil$. Typically the s -bit address is partitioned into an r -bit field and a q -bit field, where $r + q = s$. The memory array is then constructed using 2^r columns of 2^q cells each, (or vice-versa). The r -bit-field is used to address a column in the memory, and the q -bit-field addresses a row. The cell addressed is that in the column and row addressed.

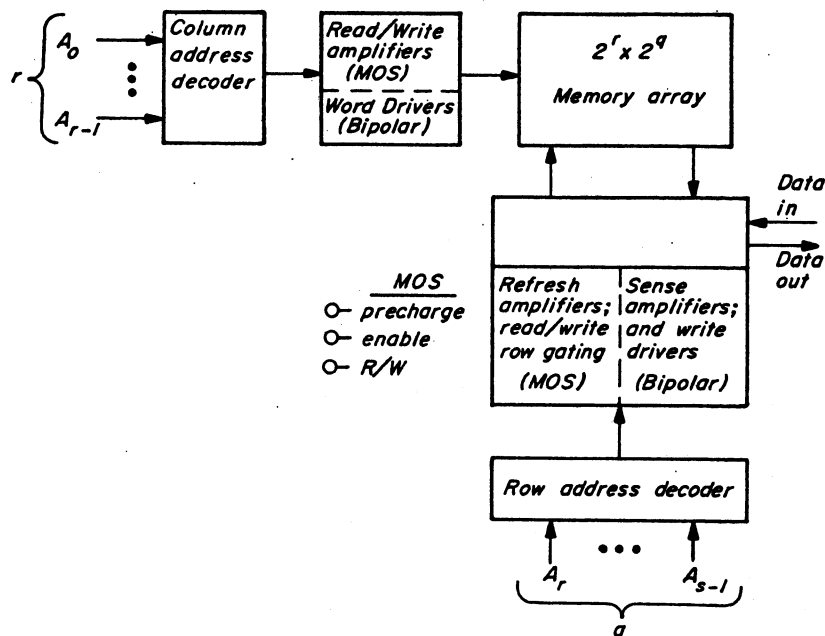


Figure 3.37 Organization of a dynamic MOS and bipolar semiconductor RAM

Hence the following parameters have a significant effect on how the memory is to be tested.

1. Total capacity
2. Array configuration
3. Addressing layout
4. Refresh parameters

Categories of Memory Faults

Faults can occur in the memory matrix, decoders, input buffers, read/write circuitry, data input circuitry or sense amplifiers. These internal faults can lead to functional failures such as the inability to read or write, erroneous data storage, to d.c. parametric failures such as unacceptable output levels, or to dynamic failures such as slow access times. More specifically, these semiconductor memories exhibit malfunctions such as the following:

- (1) *Opens and shorts.*
- (2) *Open decoders* — the total memory cannot be truly addressed.
- (3) *Multiple writes* — in the act of writing in one cell the chip actually writes in more than one cell.
- (4) *Pattern sensitivity* — the contents of a cell become complemented due to read and write operations in "electronically adjacent" cells. Such an error may be a function of (a) the information being read and written; (b) the cells being addressed, and (c) the sequence in which these cells are addressed.
- (5) *Write recovery* — memory may not produce information at the specified access time when each read cycle is preceded by a write cycle.
- (6) *Sense amplifier sensitivity* — memory may not respond with the proper information after reading a long series of similar data bits followed by a single transition of the opposite data value.
- (7) *Sleeping Sickness* — memory loses information in less than the stated hold time.

The problem of pattern sensitivity arises primarily due to the high component density which exists in RAM's and the related effect of unwanted interacting signals. Because of this problem the following situation may occur: (1) store (write) a value in cell i ; (2) read this value several times to verify that it is indeed stored in cell i ; (3) read and write in cells other than cell i ; (4) read cell i and find the value is now wrong. This effect can occur even though each cell is capable of being correctly addressed and individually can store a 0 and a 1.

To prove that a read/write RAM is completely functional, the following aspects must be verified.

1. Every cell of the memory must be capable of storing a 0 and a 1.
2. The cell addressing circuits, or decoders, must correctly address every cell.
3. The sense amplifiers must operate correctly.
4. There must be no interaction between cells.
5. For dynamic MOS memories, the cells must be capable of storing data for a specified time without being refreshed.

Testing Patterns

Numerous classes of canonical test pattern sequences have been devised for testing RAM's. These tests can be classified under a number of categories, such as complexity and function. Function refers to what fault mode the test is primarily designed to detect. Complexity refers to the number of operations (READ or WRITE) required to carry out the test, as a function of the memory size n . Most current tests are either of order $\mathcal{O}(n)$, $\mathcal{O}(n^{3/2})$, or $\mathcal{O}(n^2)$. * Note that the number of possible patterns (states) in a memory is 2^n , hence, in practice, exhaustive testing is infeasible.

Another factor to be considered in designing a test is the difficulty in computing the sequence of addresses, of input data bits, and READ and WRITE control inputs. These factors influence the complexity of the test generation algorithm and the cost of the tester.

In the discussion of test procedure, the following notation will be used:
The statement

WRITE: $c_i \leftarrow \delta$

implies that cell i (i is the address) should be set to the value δ , $\delta \in \{0,1\}$.
The statement

READ: $c_i (= \delta)$

implies that cell i is read, and the value read should be δ .

* $\mathcal{O}(n^k)$ means "of the order n^k ". For example a test of order n^2 contains Cn^2 steps for some constant C .

COMMONLY USED FUNCTIONAL RAM TEST PATTERNS

MSCAN Test Procedure

For $i = 0, 1, 2, \dots, n - 1$ (addresses) in turn, do the following.

WRITE: $c_i \leftarrow 0$

READ: $c_i (=0)$

WRITE: $c_i \leftarrow 1$

READ: $c_i (=1)$.

This test sets (WRITE) each cell to a 0 and 1, and verifies via a READ that the cell contains the desired piece of information. Analysis indicates that there is very little information gained from this test procedure. All that is known at the end of this procedure is that there is at least one cell in the RAM which can be set to a 0 and 1, and in which this data can be read. This is because a fault could have occurred in the address decoder modifying the logic of the RAM so that the same cell is always referenced. Since this test requires four operations per bit, its complexity is $4n$.

Test:

Column Bars (similar to checkerboard test)

Purpose: Tests for possible shorts between adjacent columns. For RAM which are refreshed an entire row at a time, this pattern, or a checkerboard pattern*, may be used for worst case refresh tests.

Test Procedure:

Step 1: WRITE 1's in all even columns and 0's in all odd columns

Step 2: READ each column and row

Step 3: Repeat steps 1 and 2 for complementary patterns (i.e., interchange 0's and 1's)

Complexity: $4n$

*A checkerboard pattern consists of alternate 0's and 1's in each row and column (analogous to the coloring of squares on a checkerboard).

Test:**Volatility Test Pattern**

Purpose: Checks for hold time in dynamic memories.

Test Procedure:

Step 1: Load memory with a test pattern (various different simple patterns are typically used)

Step 2: Pause T units of time* (inhibit all clocks)

Step 3: READ entire memory

Step 4: Repeat for complementary patterns

Complexity: $4n$

Test:**Marching 1's and 0's**

Purpose: Minimal functional testing; detection of many decoder errors; minimal check on cell interactions.

Test Procedure:

Step 1: WRITE: $c_i \leftarrow 0$ for $i = 0, 1, \dots, n-1$

Step 2: For $i = 0, 1, \dots, n-1$ do

 READ: $c_i (=0)$

 WRITE: $c_i \leftarrow 1$

 READ: $c_i (=1)$

Step 3: For $i = n-1, n-2, \dots, 0$ do

 READ: $c_i (=1)$

 WRITE: $c_i \leftarrow 0$

 READ: $c_i (=0)$

Step 4: Repeat steps 1-3 interchanging 0's and 1's. That is repeat for the complementary pattern.

Comment: As the memory is scanned in ascending order, any effect that writing in a cell has on a higher numbered cell is detected when the second cell is read. Similarly, when processing in descending order, effects on lower numbered cells are detected. By leaving a trail of 1's behind each iteration in step 2, it is known that we are addressing a new cell at each step. Therefore, this process guarantees that each cell exists and that every cell is capable of storing a 1 and a 0. Most of the decoder is tested.

Complexity: $12n$

* T is a function of the refresh and hold times which in turn are a function of temperature.

Test:**Shifted Diagonal Test**

Purpose: This test is designed to identify slow recovery in sense amplifiers. That is, after reading a long sequence of the same bit value, switching to a new output value may be delayed.

Test Procedure: The test consists of placing a diagonal of 1's in the memory, all other cells storing 0's. This diagonal is then shifted one position at a time so that it sweeps through the memory.

Step 1: Load memory with 0's in every cell except those on a diagonal (assuming a square array), where 1's are stored.

Step 2: Repeat the following two steps $K = 2^r$ times (once for each column).

2a: READ column k for $k = 1, 2, \dots, K$

2b: Shift the diagonal right one position. (The last shift cannot be deleted.)

Step 3: Repeat steps 1 and 2 using a diagonal of 0's rather than 1's.

Comment: For a 64×64 array, the original pattern is set up by first applying a 1 after every 63 0's. We see that this test applies a long sequence of 0's (1's) followed by a single 1 (0) to the sense amplifiers during the READ operation.

Complexity: To load and read memory requires $2n$ operations. This procedure is done K times for the diagonals of 0's and 1's, hence $C = 4K$ test steps are required. For a square array (K maximum), $K = n^{1/2}$, hence $C = 4n^{3/2}$. The complexity for this test varies from $64n$ (for $K = 16$) to $4n^{3/2}$ (for $K = n^{1/2}$), and is often considered to be of order $n \log_2 n$.

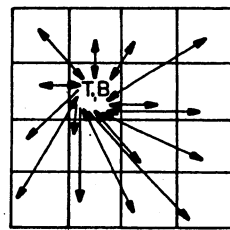
So far we have not considered testing for faults involving cell interaction i.e., pattern sensitive faults. To do this, a sub-procedure, called ping-pong has been devised. We will first discuss how this sub-procedure works, then present several tests which incorporate this ping-pong concept. Due to the complexity of the ping-pong process, all of these tests will be of complexity $O(n^{3/2})$ or greater.

Sub-test:

Ping-Pong Procedure

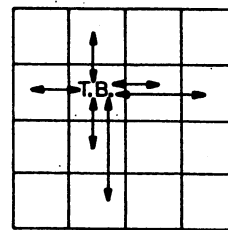
Ping-pong is a sub-test procedure used in several functional test patterns. Ping-ponging is used to test for the interaction between pairs of cells in memory. In a full ping-pong (Figure 3.43(a)), one cell c_i is designated

the test bit. Now for all $j \neq i$ some sequence of READ and WRITE operations are done on cells c_i and c_j . If this procedure is repeated for each i , then a test of order n^2 results. Since tests using a full ping-pong procedure usually require a large amount of tester time, some simplifications are often employed. One common procedure is to assume that interference between cells occurs if they are either in the same "electrical" column or row. Note that these cells share common addressing and refresh circuitry. If we only ping-pong between cells in the same row and column in which the test bit resides, as shown in Figure 3.43(b), the test complexity is reduced to about $n^{3/2}$. The actual complexity of this procedure is a function of the array configuration.



Full Ping-Pong

(a)



Row/Column Ping-Pong

(b)

Figure 3.43 Two versions of ping-pong

The following two test procedures employ ping-ponging.

Test:

Walking 1's and 0's

Purpose: Verifies that each cell can be set to both 0 and 1; that any cell can be set to either state without causing any other cell to change its state; and that decoder addressing is correct. Slow amplifier recovery is also detected by this test.

Test Procedure:

Step 1: WRITE: $c_i \leftarrow 0$ for $i = 0, 1, \dots, n - 1$

Step 2: For $i = 0, 1, \dots, n - 1$ carry out the following full ping-pong procedure on test bit c_i .

2a: WRITE: $c_i \leftarrow 1$

2b: READ: $c_j (=0)$ for all $j \neq i$ (tests that no cell is disturbed).

2c: READ: $c_i (=1)$ (tests that test bit is still correct).
rect."

2d: WRITE: $c_i \leftarrow 0$ (restores test bit to original value).

Complexity: $2n^2 + 6n$.

Test:

Gallop 1's and 0's

Purpose: To test all possible address transitions with all possible data transitions when reading.

Test Procedure (GALPAT I): Same as for "Walking 0's and 1's" except that in Step 2b each READ: $c_j (=0)$ operation is followed by a READ: $c_i (=1)$ operation. A modified version of this test procedure, known as GALPAT II, also exists.

Complexity: $2n^2 + 8n$

Test	Complexity	Test Time (seconds) for a 500 ns cycle 4K RAM	Purpose
1 Column bars	$4n$	.008	Shorts between columns; refresh
2 Checkerboard	$4n$	.008	Refresh
3 Volatility	$4n$	.008	Refresh
4 Marching 1's/0's	$12n$	.024	Minimal functional test, simple multiple select
5 Shifted Diagonal	$4n^{3/2}$	.5	Test for slow recovery in sense amplifiers
6 Ping-pong*	n^2 (full) $n^{3/2}$ (row/column)	8 .128	Procedure to pair off a test bit with other bits in the array
7 Walking 1's/0's	$2n^2 + 6n$	16	Verify that each cell can be set to both 0 and 1 without causing inter- ference
8 Galloping 1's & 0's (GALPAT I)	$2n^2 + 8n$	16	Test all possible read/ write combinations (multiple selects)
9 Galloping 1's & 0's (GALPAT II)	$4n(2n - 1)$	64	Test all possible read/ write combinations (multiple selects)

*Ping-pong is not a test but a procedure used in tests 7, 8, and 9.

Figure 3.38 Summary of RAM Test Characteristics

FUNCTIONAL TESTING OF LOGIC ARRAYS IJA1

A one-dimensional array of combinational cells, where the signal flow is left to right, and notation to be used is shown in Figure 1.

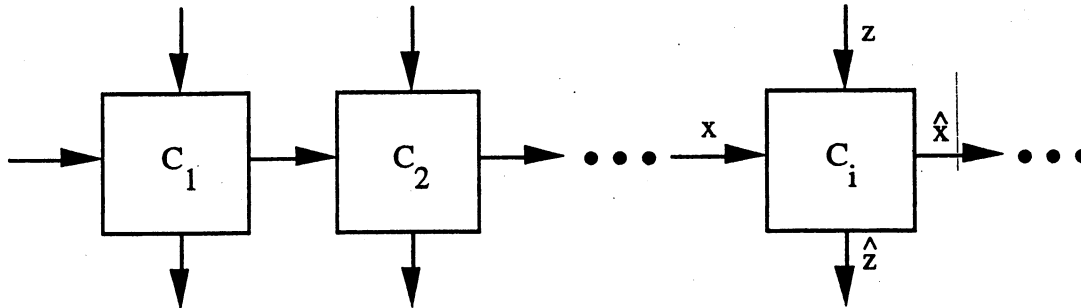


Figure 1. A typical Cell in a one-dimensional array.

A typical cell receives the input x from its left hand neighbor and an external input z . It generates an external output $\hat{z}$ and transmits an output $\hat{x}$ to its right hand neighbor. The controllable inputs consist of the x input to the first (leftmost) cell and the z inputs to all cells in the array. All $\hat{z}$ outputs and the $\hat{x}$ output of the last cell are observable. The z input to cell c_i is assumed independent of the z input to cell c_j for $i \neq j$. Subscripts are used to refer to specific inputs or outputs where necessary. The behavior of a cell can be described by a state table, which has a row for each x input (also referred to as state) and a column for each z input. The entries in the table consist of pairs of cell outputs $(\hat{x}, \hat{z})$ for each combination (x, z) of cell inputs. The entries in row x_i , column z_j will be represented functionally as $\hat{x}(x_i, z_j)$, $\hat{z}(x_i, z_j)$. Note that the variables are not necessarily binary and may represent combinations of binary variables.

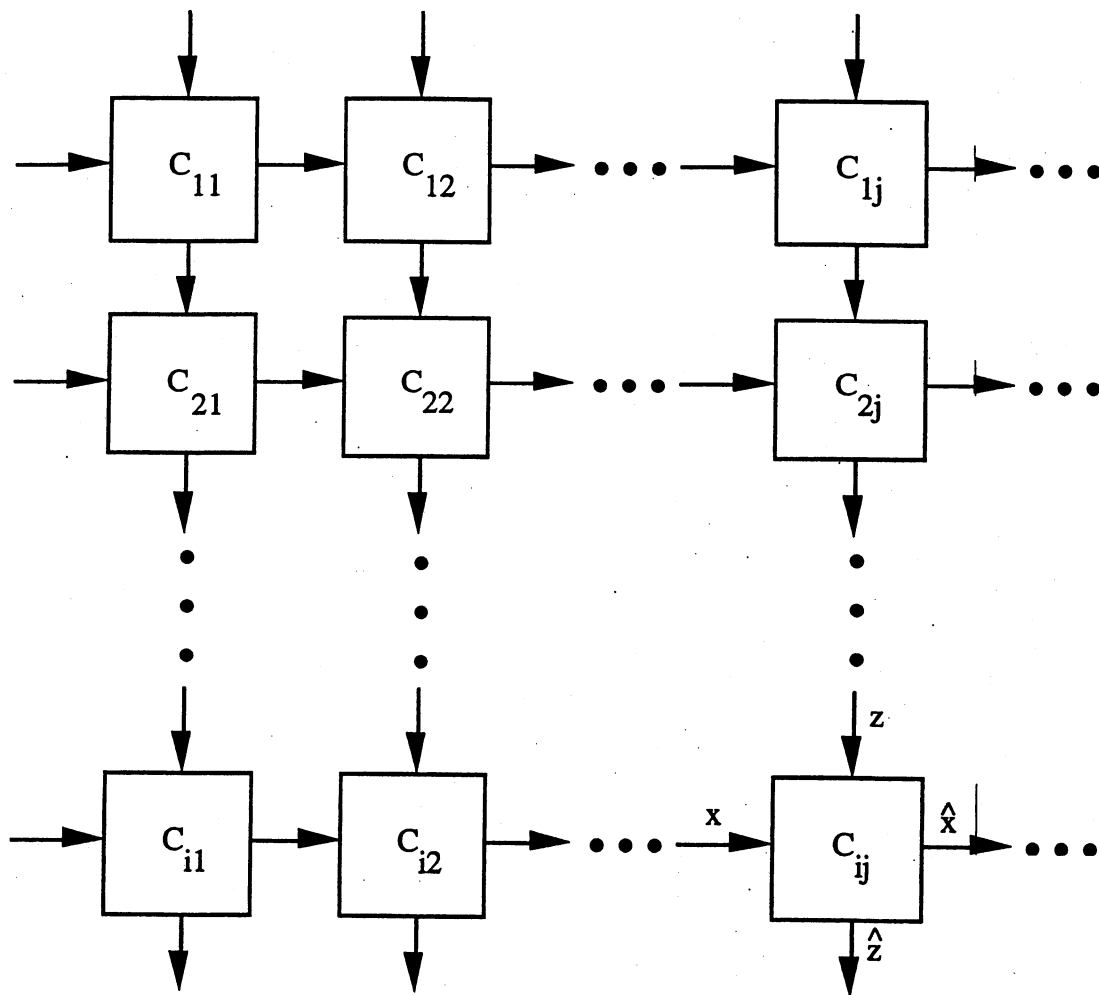


Figure 2. A typical cell in a two-dimensional array.

The problem of testing iterative arrays is made difficult by the facts that only external inputs are directly controllable and only external outputs and border cells are directly observable. Thus in a k -dimensional array most of the cell inputs are not directly controllable and most of the cell outputs are not directly observable.

The problem of testing ILA's was first considered by Kautz [7]. Arrays are tested by applying test inputs to their accessible (external) input terminals and observing the outputs on accessible output terminals. The accessible inputs and outputs are usually associated with the boundaries of the array. An array is *testable* if it is possible to detect any faulty cells in the array. Of course any array may be completely tested for all detectable faults by applying all possible combinations of inputs. This is impractical due to the large number of tests that may be necessary.

It is assumed that all possible cell inputs must be applied to each cell in order to test it completely and that the fault in a cell may affect the cell outputs in any arbitrary way. We shall refer to these as the *general-fault assumptions*. At most one cell is assumed to be faulty.

Kautz showed that the following conditions are necessary and sufficient for an ILA to be testable [7]:

1. It is possible to apply to the input terminals of any cell, a complete set of tests for detecting all faults in the cell, independent of the position of the cell in the array.
2. For each such test, it is possible to propagate the effect of the fault to an observable output.

These conditions correspond to the generation of the desired signal value at the location of a fault in a combinational circuit and the sensitization of a path to an observable output.

State Tables with M columns N rows

MN tests necessary to test each cell individually

For 1D Array of p cells

$MN \cdot (N-1)$ necessary to test each cell

→ $< pMN^2$ tests total

If no column contains two or more identical entries

any error in $\hat{x}$ or $\hat{z}$ output of any cell automatically propagates to boundary.

We will now prove some results related to arrays which are easily testable. We define two such classes of arrays [2].

Definition 1

An ILA defined by a basic cell with N states and M input columns is O (optimal) testable if it can be tested with $M \cdot N$ tests independent of the size of the array.

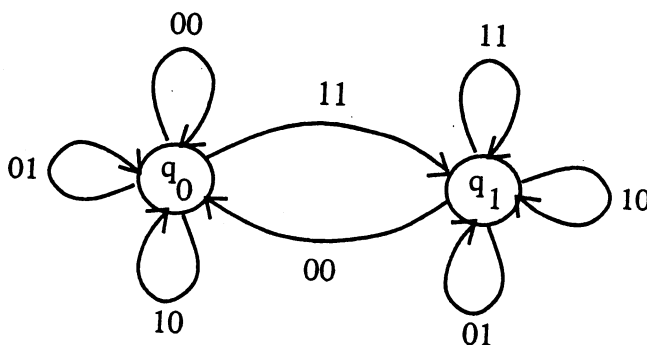
Definition 2

An ILA defined by a basic cell with N states and M input columns is C (Constant) testable if it can be tested with C (a constant) $\geq M \cdot N$ tests independent of the size of the array.

EXAMPLE

State Diagram

(1a) Adder



Cell State Table

		XY			
		00	01	11	10
initial state →	q_0	$q_{0,0}$	$q_{0,1}$	$q_{1,0}$	$q_{0,1}$
	q_1	$q_{0,1}$	$q_{1,0}$	$q_{1,1}$	$q_{1,0}$

Adder Cell

Theorem 2 Defining Circuits

Satisfies O-testability

(a) $(q_0, 00)$ defines Test 1

(b) $(q_0, 01)$ defines Test 2

(c) $(q_0, 10)$ defines Test 3

(d) $(q_1, 01)$ defines Test 4

(e) $(q_1, 10)$ defines Test 5

(f) $(q_1, 11)$ defines Test 6

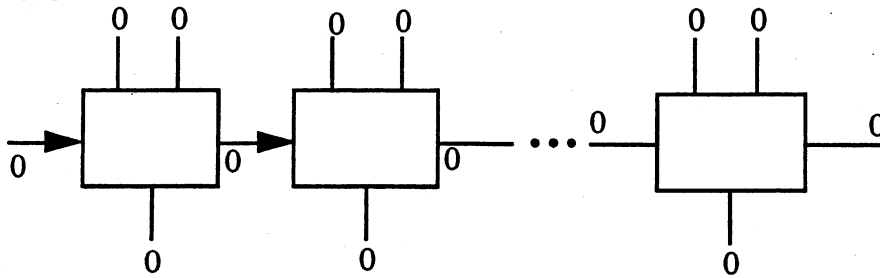
(g) $(q_0, 11) \rightarrow (q_1, 00) \rightarrow (q_0, 11)$

defines Tests 7 and 8

$$Y + X + d_o \quad Y + \sum_{i=1}^p X + \sum_{i=1}^p d_{oi}$$

Tests corresponding to circuits Test 1

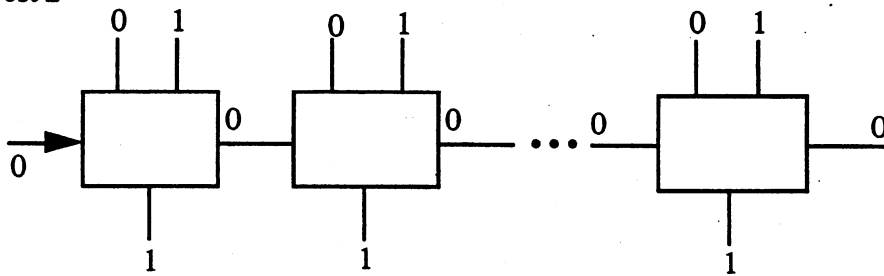
Test 1



Test 1 for Adder Cell

$[q_0, (00)(00) \dots (00)]$
 Applies $(q_0, 00)$ to all
 cells. $\hat{Z} = [00 \dots 0]$

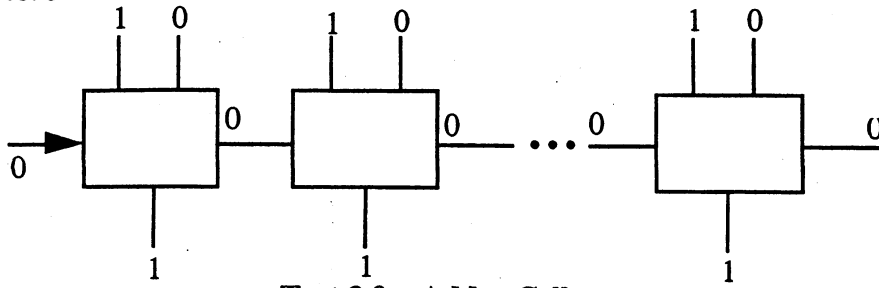
Test 2



Test 2 for Adder Cell

$[q_0, (01)(01) \dots (01)]$
 Applies $(q_0, 01)$ to all
 cells. $\hat{Z} = [11 \dots 1]$

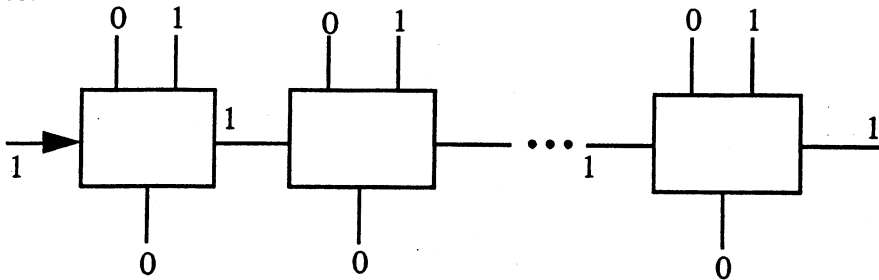
Test 3



Test 3 for Adder Cell

$[q_0, (10)(10) \dots (10)]$
 Applies $(q_0, 10)$ to all
 cells. $\hat{Z} = [11 \dots 1]$

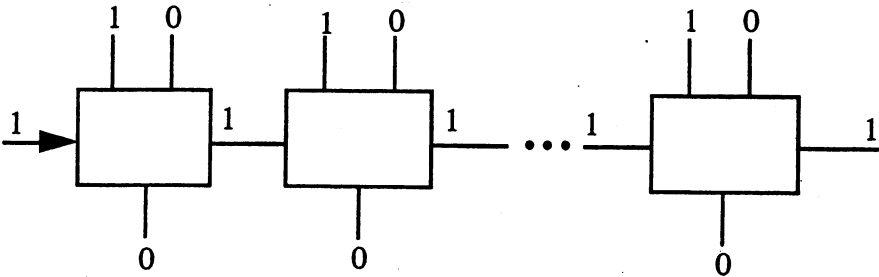
Test 4



Test 4 for Adder Cell

$[q_1, (01)(01) \dots (01)]$
 Applies $(q_1, 01)$ to all
 cells. $\hat{Z} = [00 \dots 0]$

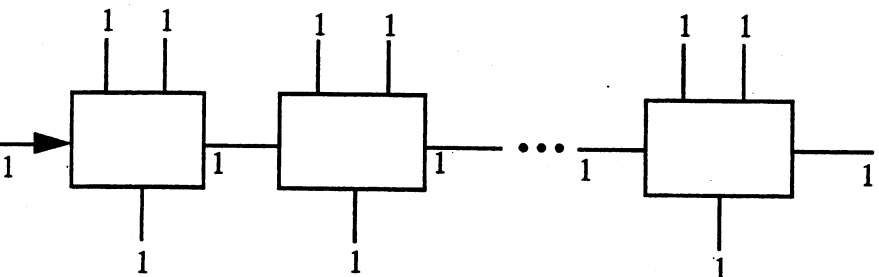
Test 5



Test 5 for Adder Cell

$[q_1, (10)(10) \dots (10)]$
 Applies $(q_1, 10)$ to all
 cells. $\hat{Z} = [00 \dots 0]$

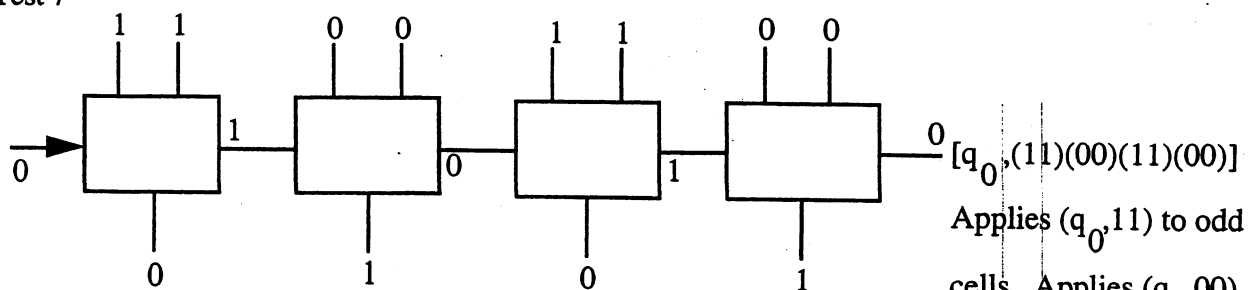
Test 6



Test 6 for Adder Cell

$[q_1, (11)(11) \dots (11)]$
 Applies $(q_1, 11)$ to all
 cells. $\hat{Z} = [11 \dots 1]$

Test 7

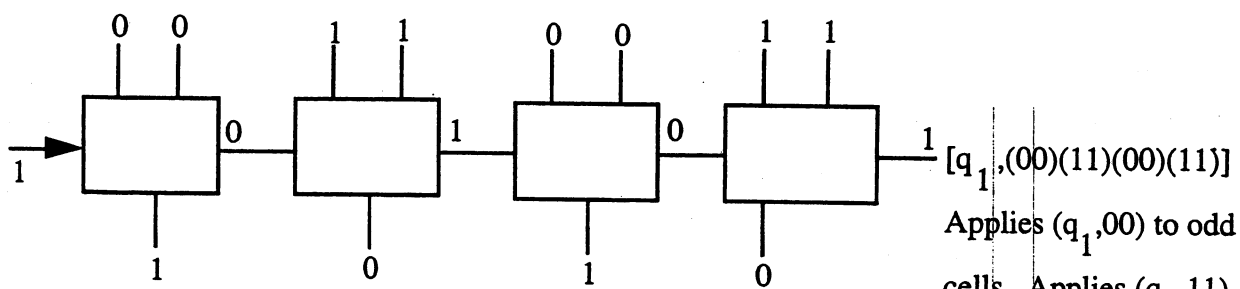


Test 7 for Adder Cell

$[q_0, (11)(00)(11)(00)]$
 Applies $(q_0, 11)$ to odd
 cells. Applies $(q_1, 00)$
 to even cells.

$$\hat{Z} = [0101 \dots]$$

Test 8



Test 8 for Adder Cell

$[q_1, (00)(11)(00)(11)]$
 Applies $(q_1, 00)$ to odd
 cells. Applies $(q_0, 11)$
 to even cells.

$$\hat{Z} = [1010 \dots]$$

Theorem 2

For a one-dimensional ILA defined by a basic cell C in which all single cell functional faults are propagated by all inputs.

(a) The ILA is O-testable (for fault detection of a single cell functional faults) if the state diagram of C can be partitioned into disjoint subsets of branches each subset defining a closed cycle.

(b) The ILA is C-testable (for fault detection of a single cell functional fault) if the state diagram of C can be divided into ^(non disjoint) ~~disjoint~~ subsets of branches each subset defining a closed cycle.

ProofSufficiency

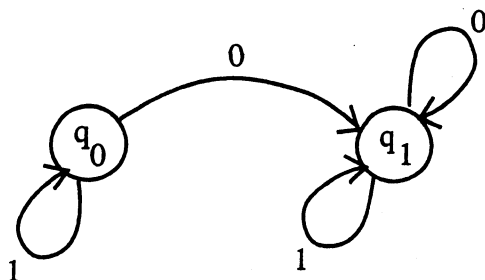
Each such closed cycle containing k branches defines a set of k tests which apply each of the k (state-input) combinations defined by the branches to all cells in the array as demonstrated in Example 1. *for Adder cell*

Necessity

If the array is C or O-testable, for each state-input combination there must exist a test which applies this combination to every r^{th} cell (for some finite r). The repetitive subsequence of length r must define a closed circuit.

Example 3

We consider one dimensional arrays defined by basic cells corresponding to each of the functions of Example 1.

State Diagram(2a) Incrementor

Does not satisfy conditions
for either O-testability or
C-testability since branch
($q_0, 0$) is not contained in
any circuit.

Figure 5. State Diagram for Incrementor Cell

The 1-dimensional array requires $q+3$ tests (where there are q cells). The first q tests apply the inputs ($q_0, 1$), ($q_1, 0$), ($q_1, 1$) to all cells. The q other tests are required to apply the input ($q_0, 1$) to each of the q cells.

	<u>Input</u>	<u>Output</u>	<u>Applies</u>
Test 1	$[q_0, 11 \dots 1]$	$[000 \dots 0]$	$(q_0, 1)$ to all cells
Test 2	$[q_1, 00 \dots 0]$	$[000 \dots 0]$	$(q_1, 0)$ to all cells
Test 3	$[q_1, 11 \dots 1]$	$[111 \dots 1]$	$(q_1, 1)$ to all cells
Test 4	$[q_0, 00 \dots 0]$	$[100 \dots 0]$	$(q_0, 0)$ to cell 1
Test 5	$[q_0, 100 \dots 0]$	$[0100 \dots 0]$	$(q_0, 0)$ to cell 2
:			
Test $i+4$	$[q_0, 11 \dots 100 \dots 0]$	$[00 \dots 010 \dots 0]$	$(q_0, 0)$ to cell i
:	$\uparrow$ i	$\uparrow$ i	
Test $q+3$	$[q_0, 111 \dots 10]$	$[00 \dots 01]$	$(q_0, 0)$ to cell q

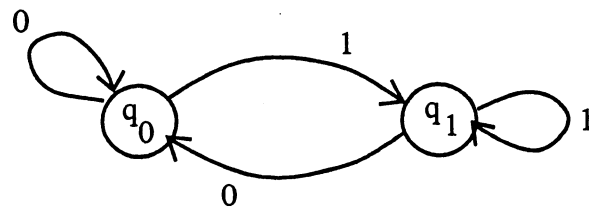


Figure 10. State Diagram for Constant (2x) Multiplier Cell

The state diagram can be decomposed into 3 disjoint circuits and is therefore O-testable.

- $(q_0, 0)$ defines Test 1
- $(q_1, 1)$ defines Test 2
- $(q_0, 1) \rightarrow (q_1, 0) \rightarrow (q_0, 1)$ defines Tests 3 and 4

<u>Input</u>	<u>Applies</u>	<u>$\hat{Z}$ Output</u>
1. $[q_0, 00 \dots 0]$	$(q_0, 0)$ to all cells	$[00 \dots 0]$
2. $[q_1, 11 \dots 1]$	$(q_1, 1)$ to all cells	$[11 \dots 1]$
3. $[q_0, 1010 \dots 10]$	$(q_0, 1)$ to odd cells $(q_1, 1)$ to even cells	$[0101 \dots 01]$
4. $[q_1, 0101 \dots 01]$	$(q_1, 0)$ to odd cells $(q_0, 1)$ to even cells	$[1010 \dots 10]$

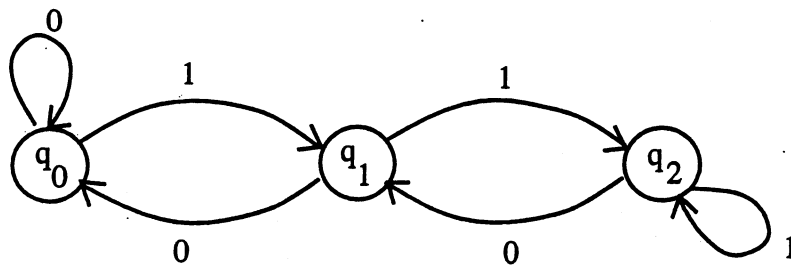


Figure 11. State Diagram for Constant (3x) Multiplier Cell

The state diagram can be decomposed into 4 disjoint circuits and is therefore O-testable.

$(q_0, 0)$ defines Test 1

$(q_2, 1)$ defines Test 2

$(q_0, 1) \rightarrow (q_1, 0) \rightarrow (q_0, 1)$ defines Tests 3 and 4

$(q_1, 1) \rightarrow (q_2, 0) \rightarrow (q_1, 1)$ defines Tests 5 and 6

(6) None of the functions considered previously were C-testable but not O-testable. A 1-dimensional array defined by the following state table satisfies Theorem 1 for fault propagation. The state diagram can be divided into non-disjoint circuits.

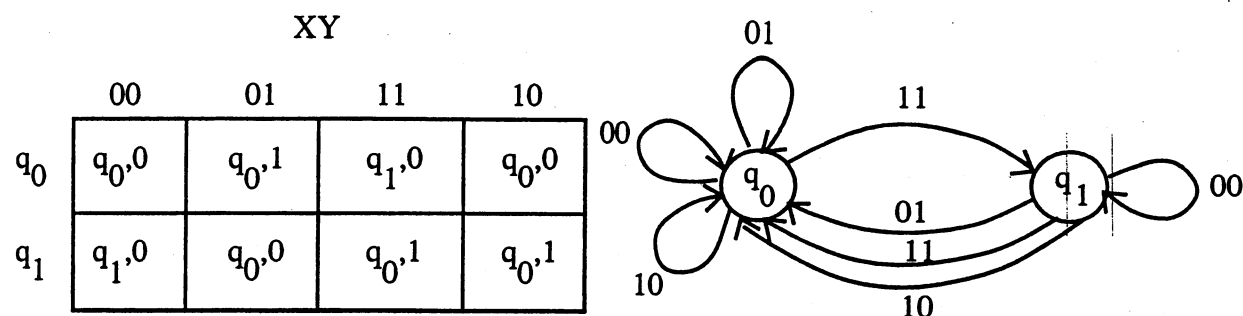


Figure 12. State Table and State Diagram for Cell which is C-testable but not O-testable

$(q_0, 00)$ defines Test 1

$(q_0, 01)$ defines Test 2

$(q_0, 10)$ defines Test 3

$(q_1, 00)$ defines Test 4

$(q_0, 11) \rightarrow (q_1, 01) \rightarrow (q_0, 11)$ defines Tests 5 and 6

$(q_0, 11) \rightarrow (q_1, 10) \rightarrow (q_0, 11)$ defines Tests 7 and 8

$(q_0, 11) \rightarrow (q_1, 11) \rightarrow (q_0, 11)$ defines Tests 9 and 10

Input	Applies	$\hat{Z}$ Output
1. $[q_0, (00)(00) \dots (00)]$	$(q_0, 00)$ to all cells	$[00 \dots 0]$
2. $[q_0, (01)(01) \dots (01)]$	$(q_0, 01)$ to all cells	$[11 \dots 1]$
3. $[q_0, (10)(10) \dots (10)]$	$(q_0, 10)$ to all cells	$[00 \dots 0]$
4. $[q_1, (00)(00) \dots (0)]$	$(q_1, 00)$ to all cells	$[00 \dots 0]$
5. $[q_0, (11)(01)(11)(01) \dots]$	$(q_0, 11)$ to odd cells $(q_1, 01)$ to even cells	$[00 \dots 0]$
6. $[q_1, (01)(11)(01)(11) \dots]$	$(q_1, 01)$ to odd cells	$[00 \dots 0]$
7. $[q_0, (11)(10)(11)(10) \dots]$	~~$(q_0, 11)$ to even cells~~ $(q_1, 10)$ to even cells	
8. $[q_1, (10)(11)(10)(11) \dots]$	$(q_1, 10)$ to odd cells	
9. $[q_0, (11)(11)(11) \dots]$	$(q_1, 11)$ to even cells	
10. $[q_0, (11)(11)(11) \dots]$	$(q_1, 11)$ to odd cells	

RANDOM TESTING

R1

3.6 RANDOM TEST GENERATION AND TESTING

3.6.1 Practice

Testers

There are two major categories of automatic test equipment (ATE) called *stored program* ATE and *random* or *comparison* type ATE. Stored program testers usually contain a minicomputer and back-up storage such as disc, and the test sequence is stored vector by vector, or as a high level program interpreted by the computer. Stored program ATE typically also store the expected response and (sometimes) a fault dictionary, which are usually generated by simulation. The actual test sequence can be obtained using an algorithmic procedure (e.g., the *D*-algorithm), manually (e.g., by a test engineer), or randomly (e.g., by a pseudo random pattern generator).

Because of a number of factors, such as the effectiveness of stored programs, the need for simulation processing, and the slow rate of test application, stored program test sequences are typically not of great length, varying from a few vectors to several thousand per UUT.

Random or comparison type ATE employ pseudo random patterns as test vectors. These patterns can be derived in many ways and are sometimes not totally random. These vectors are not stored in a minicomputer but rather are generated via a hard-wired digital pattern generator. Hence vectors can be generated at very high rates, such as from 1-40 MHz. Testing a board at these high rates for a few seconds allows several million

vectors to be applied. Errors on the output of the UUT can be detected using a comparison scheme as shown in Figure 3.40.

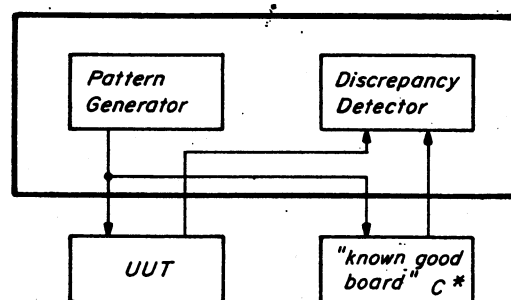


Figure 3.40 Comparison type ATE system

Here two circuit cards, the UUT and a known good copy of the circuit under test, denoted by *C**, are inserted into the ATE. The pattern generator then applies several million pseudo random patterns to both the UUT and *C**. The outputs of the boards are then compared by the discrepancy detector. A mismatch indicates a fault in the UUT.