Figure 10-1 Testing AND and OR Gates for Stuck-at Faults

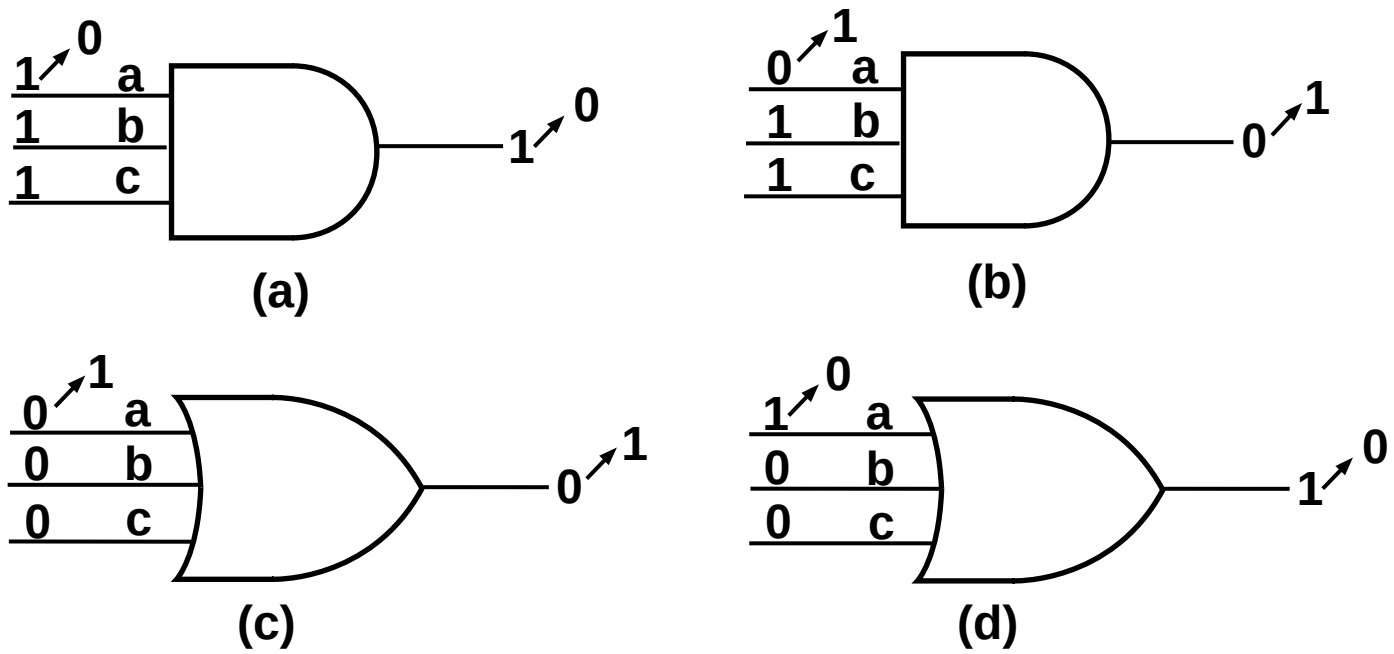
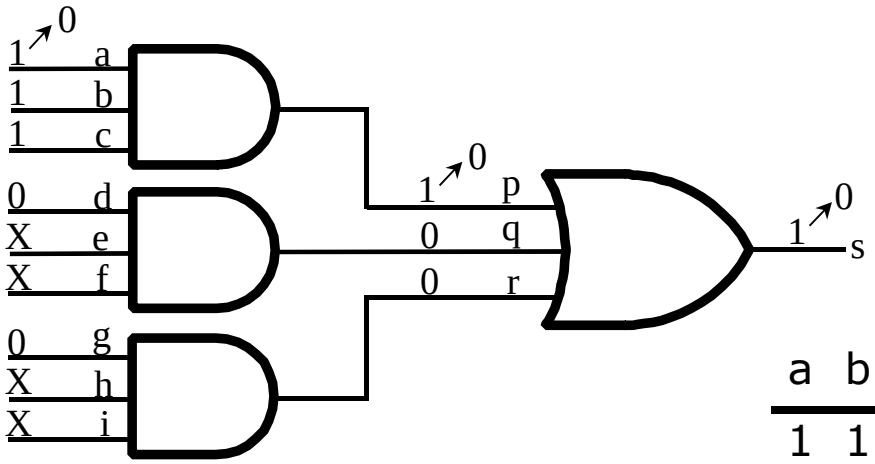
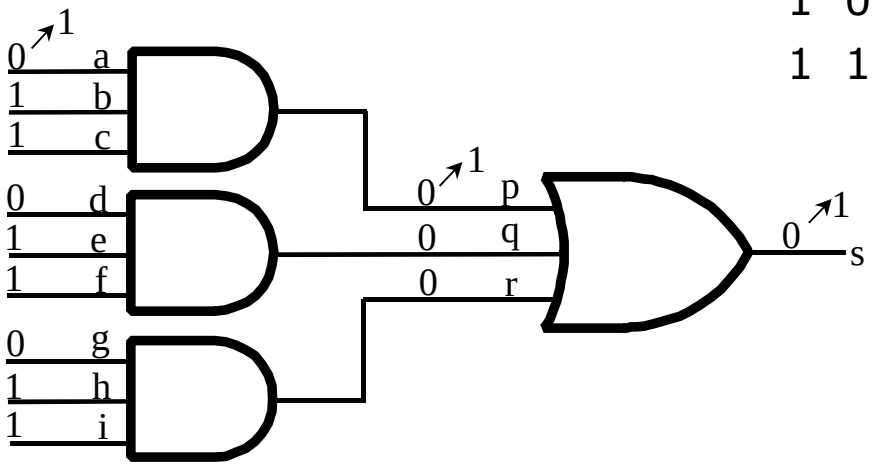


Fig. 10-2 Testing an AND-OR Network / Table 10-1 Test Vectors for Fig. 10-2



(a) stuck-at-0 test



(b) stuck-at-1 test

a	b	c	d	e	f	g	h	i	Faults Tested
1	1	1	0	X	X	0	X	X	a0, b0, c0, p0
0	X	X	1	1	1	0	X	X	d0, e0, f0, q0
0	X	X	0	X	X	1	1	1	g0, h0, i0, r0
0	1	1	0	1	1	0	1	1	a1, d1, g1, p1, q1, r1
1	0	1	1	0	1	1	0	1	b1, e1, h1, p1, q1, r1
1	1	0	1	1	0	1	1	0	c1, f1, i1, p1, q1, r1

Figure 10-3 Fault Detection Using Path Sensitization

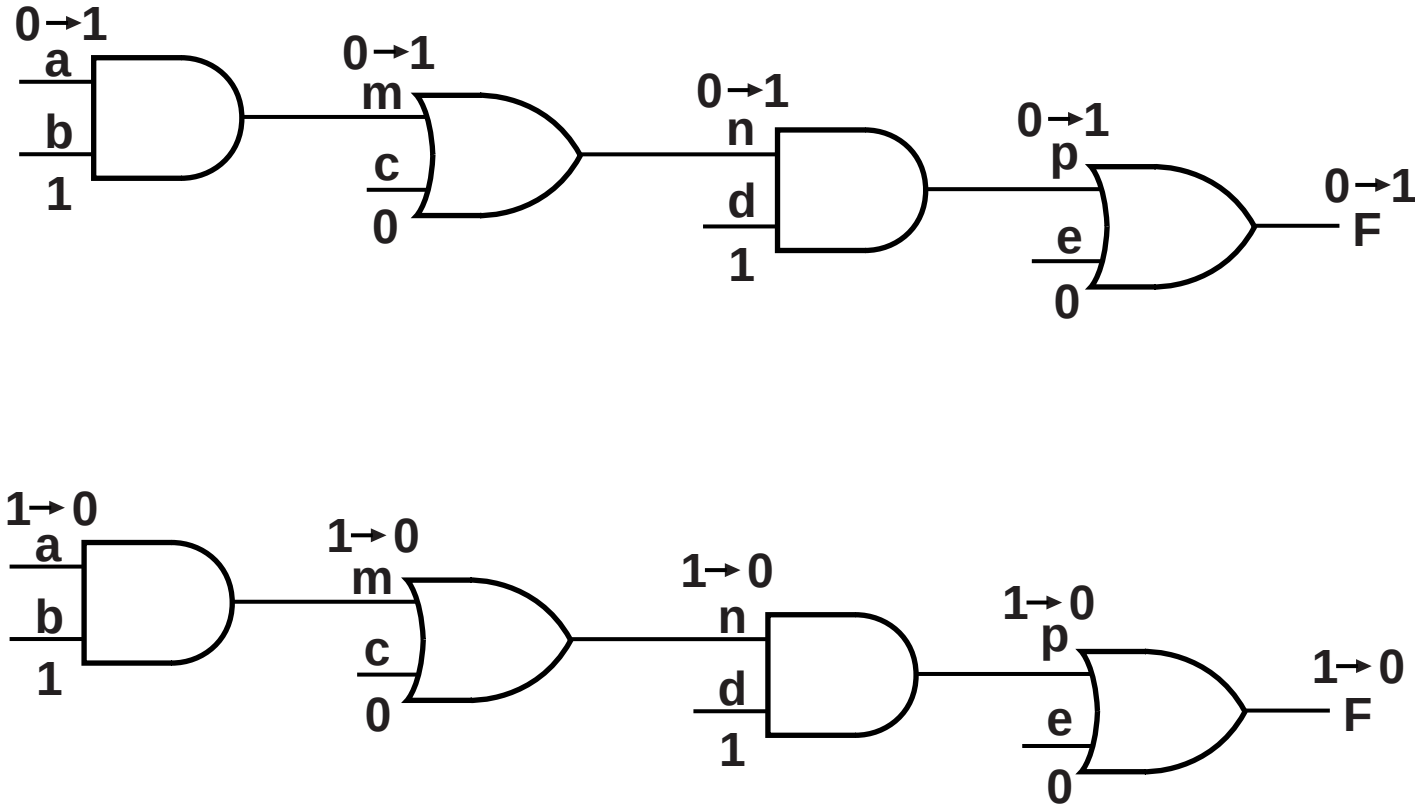


Figure 10-4 Example Network for Stuck-at Fault Testing

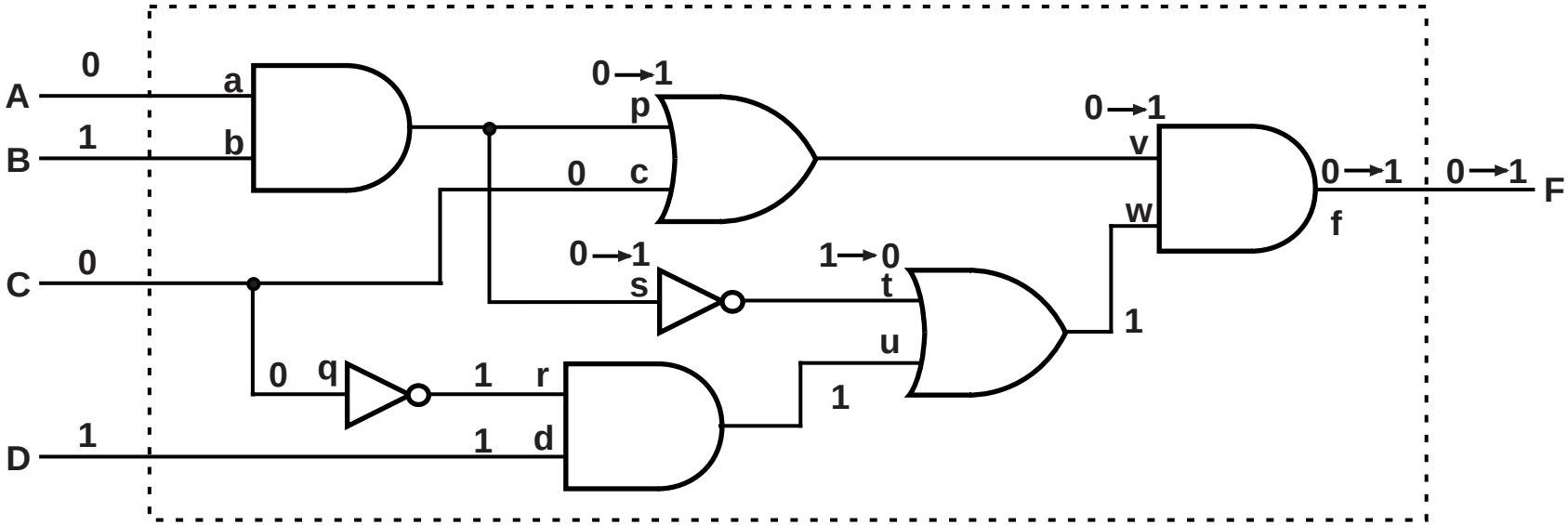


Table 10-2 Tests for Stuck-at Faults in Figure 10-4

				Normal Gate Inputs											
A	B	C	D	a	b	p	c	q	r	d	s	t	u	v	w	Faults Tested
0	1	0	1	0	1	0	0	0	1	1	0	1	1	0	1	a1 p1 c1 v1 f1
1	1	0	1	1	1	1	0	0	1	1	1	0	1	1	1	a0 b0 p0 q1 r0 d0 u0 v0 w0 f0
1	0	1	1	1	0	0	1	1	0	1	0	1	0	1	1	b1 c0 s1 t0 v0 w0 f0
1	1	0	0	1	1	1	0	0	1	0	1	0	0	1	0	a0 b0 d1 s0 t1 u1 w1 f1
1	1	1	1	1	1	1	1	1	0	1	1	0	0	1	0	a0 b0 q0 r1 s0 t1 u1 w1 f1

Figure 10-5 Sequential and Iterative Networks

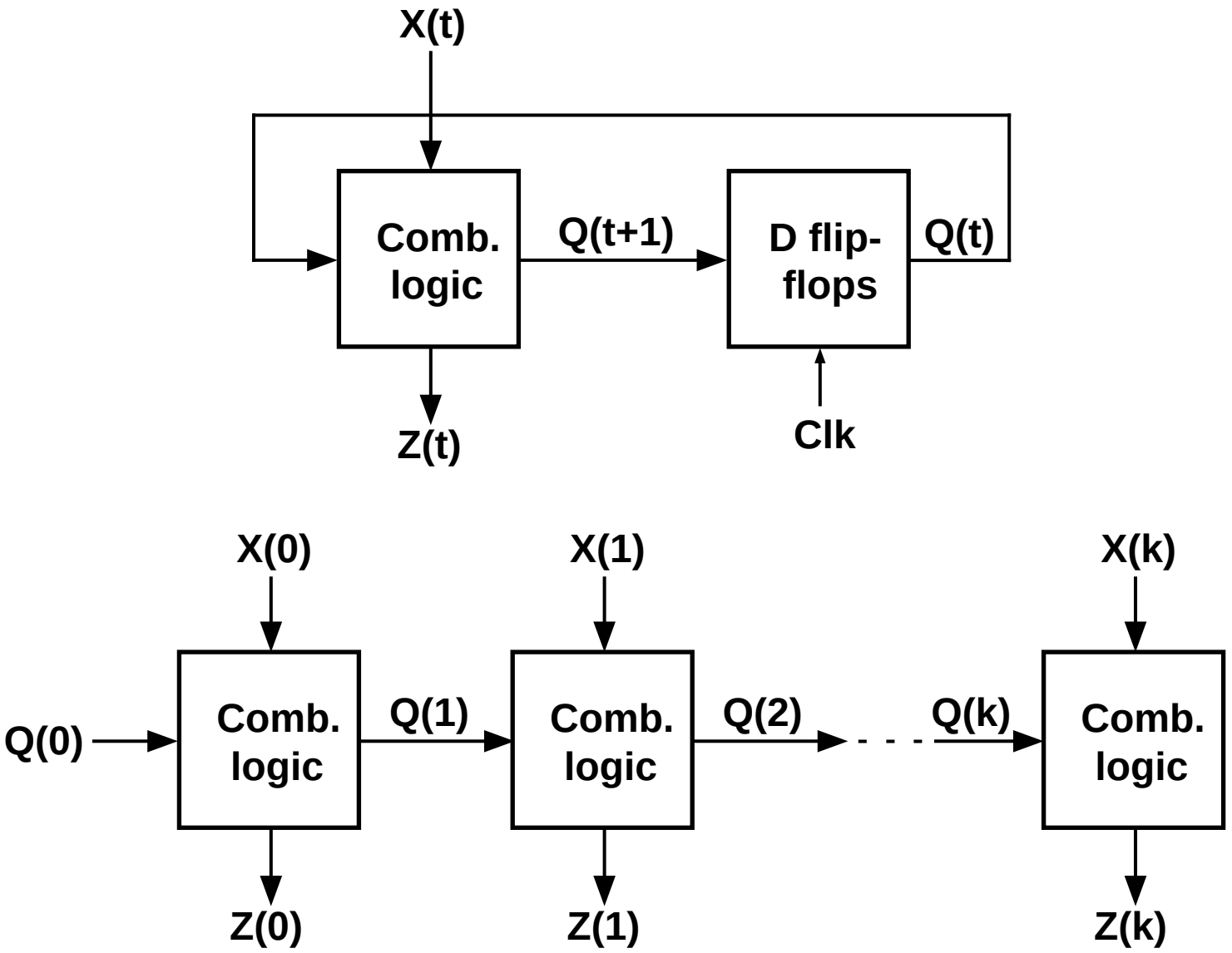


Figure 10-6 State Graph for Test Example

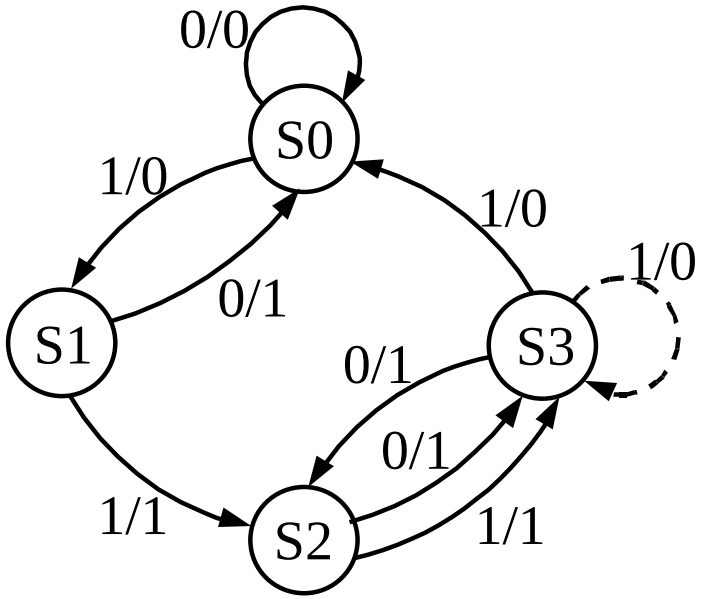


Table 10-3 State Table for Figure 10-6

Q1Q2	State	Next State		Output	
		X=0	1	X=0	1
00	S0	S0	S1	0	0
10	S1	S0	S2	1	1
01	S2	S3	S3	1	1
11	S3	S2	S0	1	0

Input	Output	Transition Verified
R 0 1 1	0 0 1	(S0 to S0)
R 1 1 1	0 1 1	(S0 to S1)
R 1 0 1 1	0 1 0 1	(S1to S0)
R 1 1 1 1	0 1 1 0	(S1 to S2)
R 1 1 0 1 1	0 1 1 0 0	(S2 to S3)
R 1 1 1 1 1	0 1 1 0 0	(S2 to S3)
R 1 1 0 0 1 1	0 1 1 1 1 0	(S3 to S2)
R 1 1 0 1 1 1	0 1 1 0 1 0	(S3 to S0)

Figure 10-7 Realization of Figure 10-6

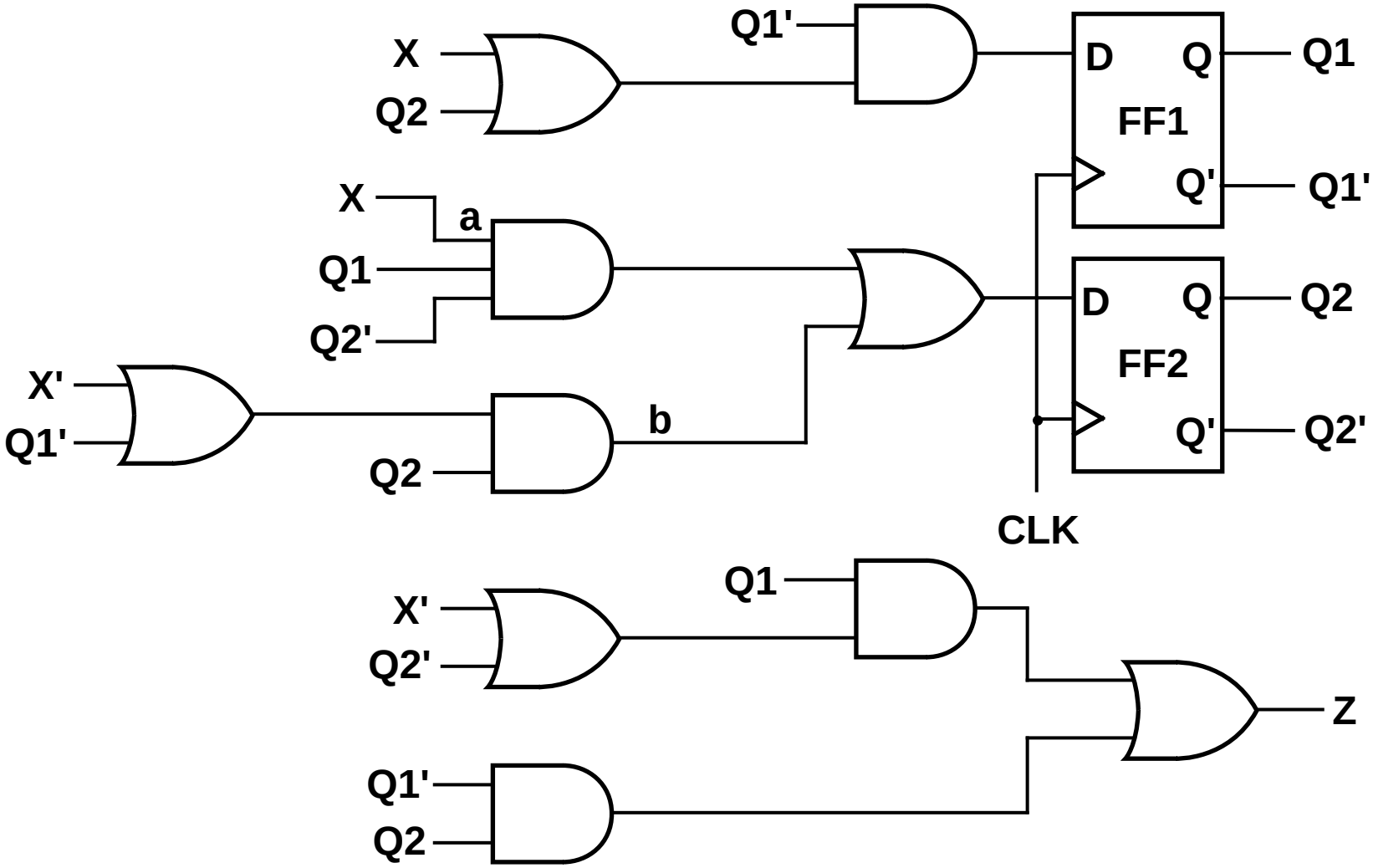
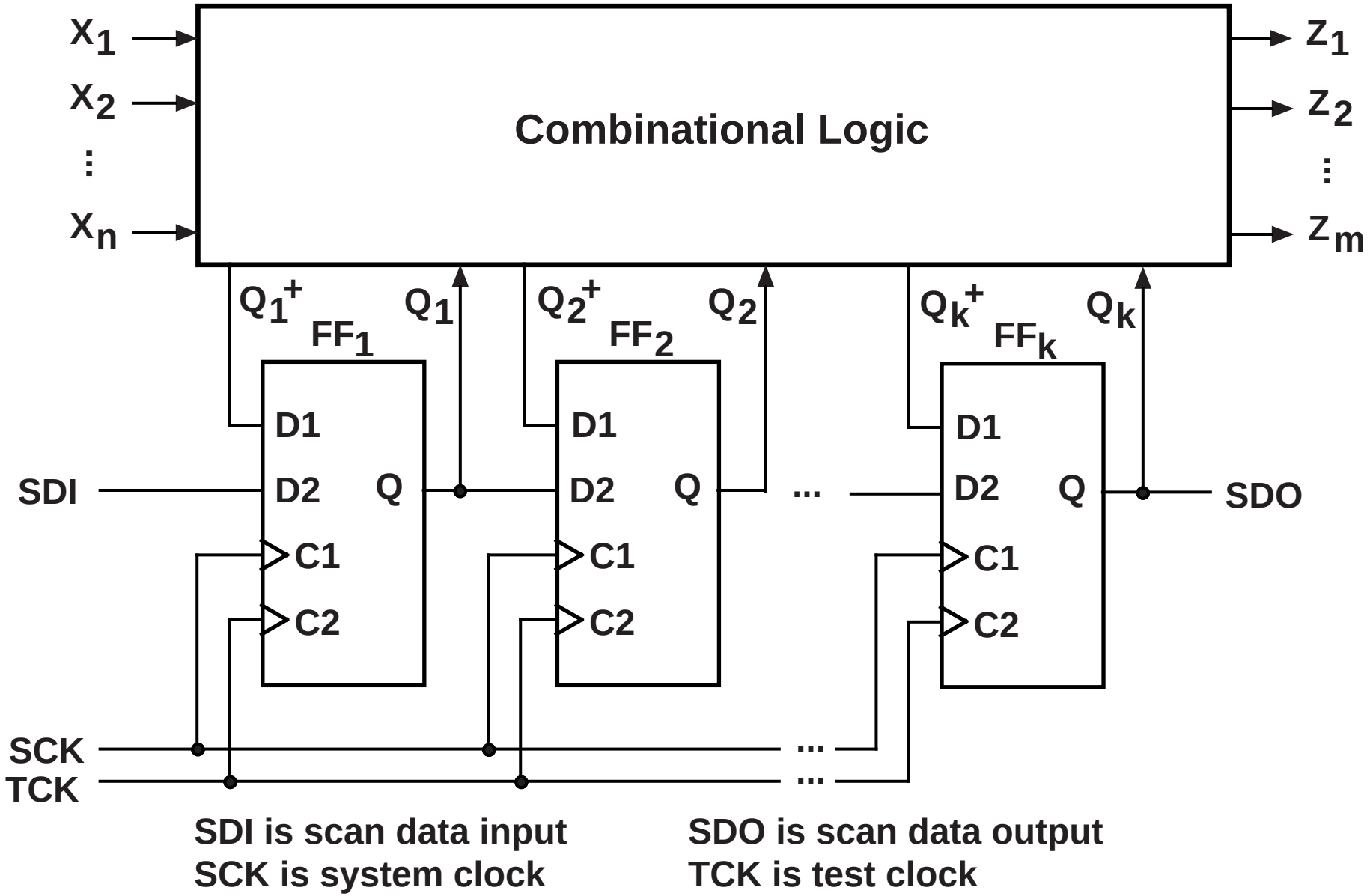


Figure 10-8 Scan Path Test Circuit Using Two-port Flip-flops



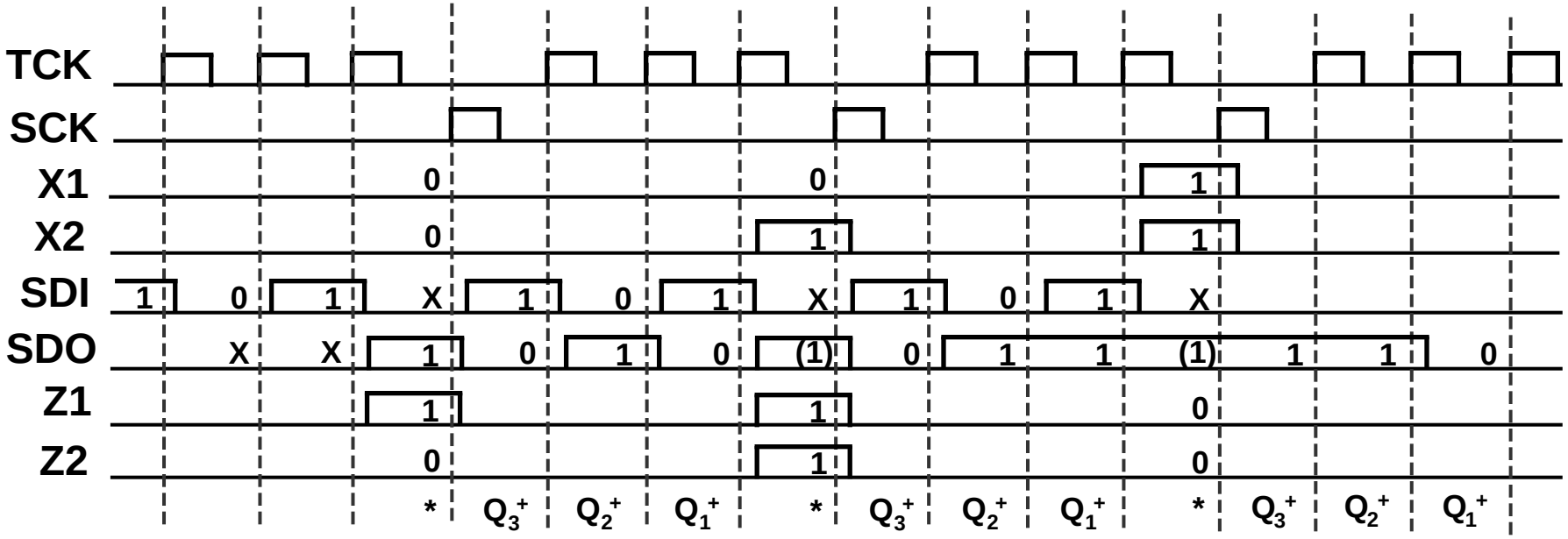
From Page 348 – Procedure for Scan Testing

1. Scan in the test vector Q_i values via *SDI* using the test clock *TCK*.
2. Apply the corresponding test values to the X_i inputs.
3. After sufficient time for the signals to propagate through the combinational network, verify the output Z_i values.
4. Apply one clock pulse to the system clock *SCK* to store the new values of Q_i^+ into the corresponding flip-flops.
5. Scan out and verify the Q_i values by pulsing the test clock *TCK*.
6. Repeat steps 1 though 5 for each test vector.

One row of the state transition table:

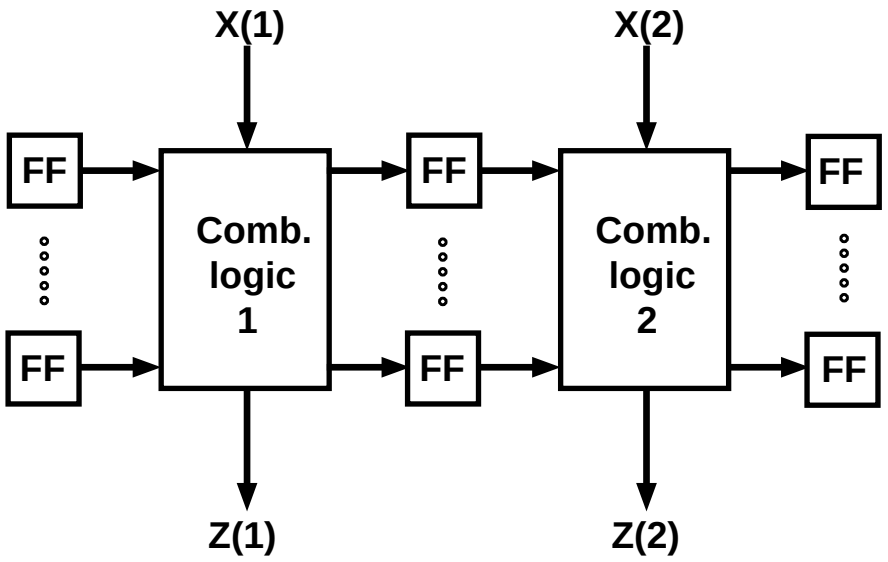
$Q_1Q_2Q_3$	$Q_1^+Q_2^+Q_3^+$				Z_1Z_2			
X_1X_2 =	00	01	11	10	00	01	11	10
101	010	110	011	111	10	11	00	01

Figure 10-9 Timing Chart for Scan Test

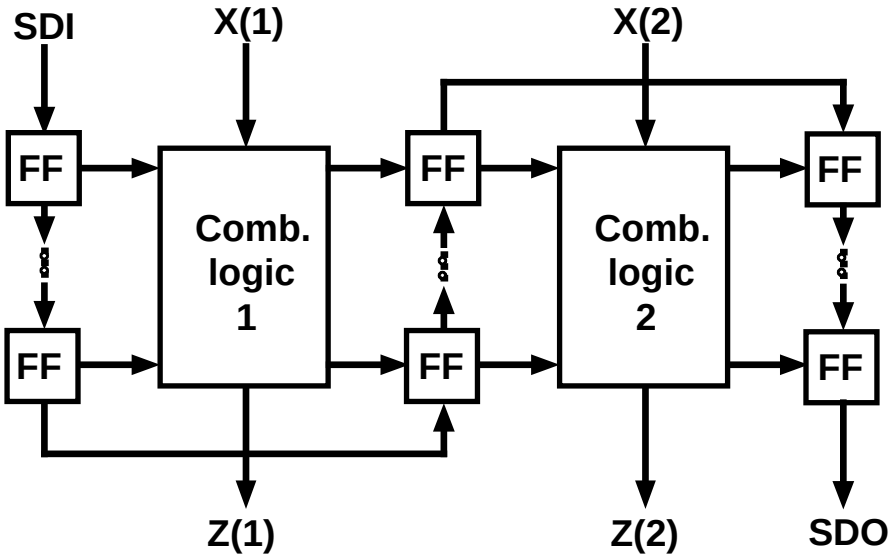


* Read output (output at other times not shown)

Figure 10-10 System with Flip-flop Registers and Combinational Logic Blocks



(a) Without scan chain



(b) With scan chain added

Figure 10-11 Scan Test Configuration with Multiple ICs

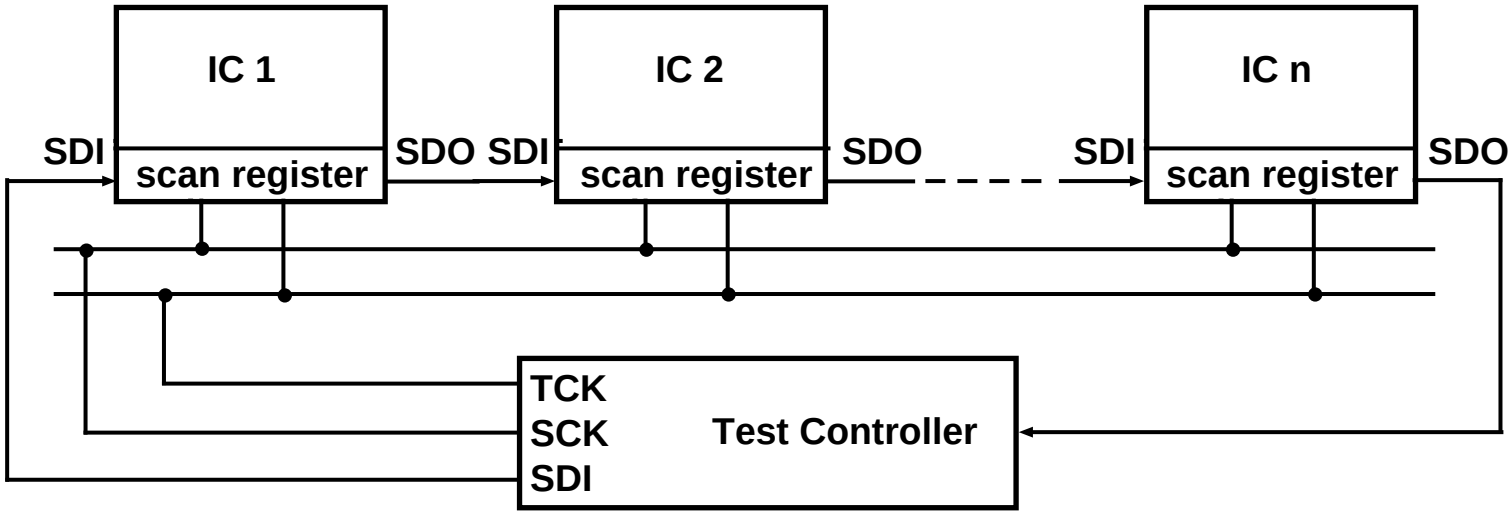
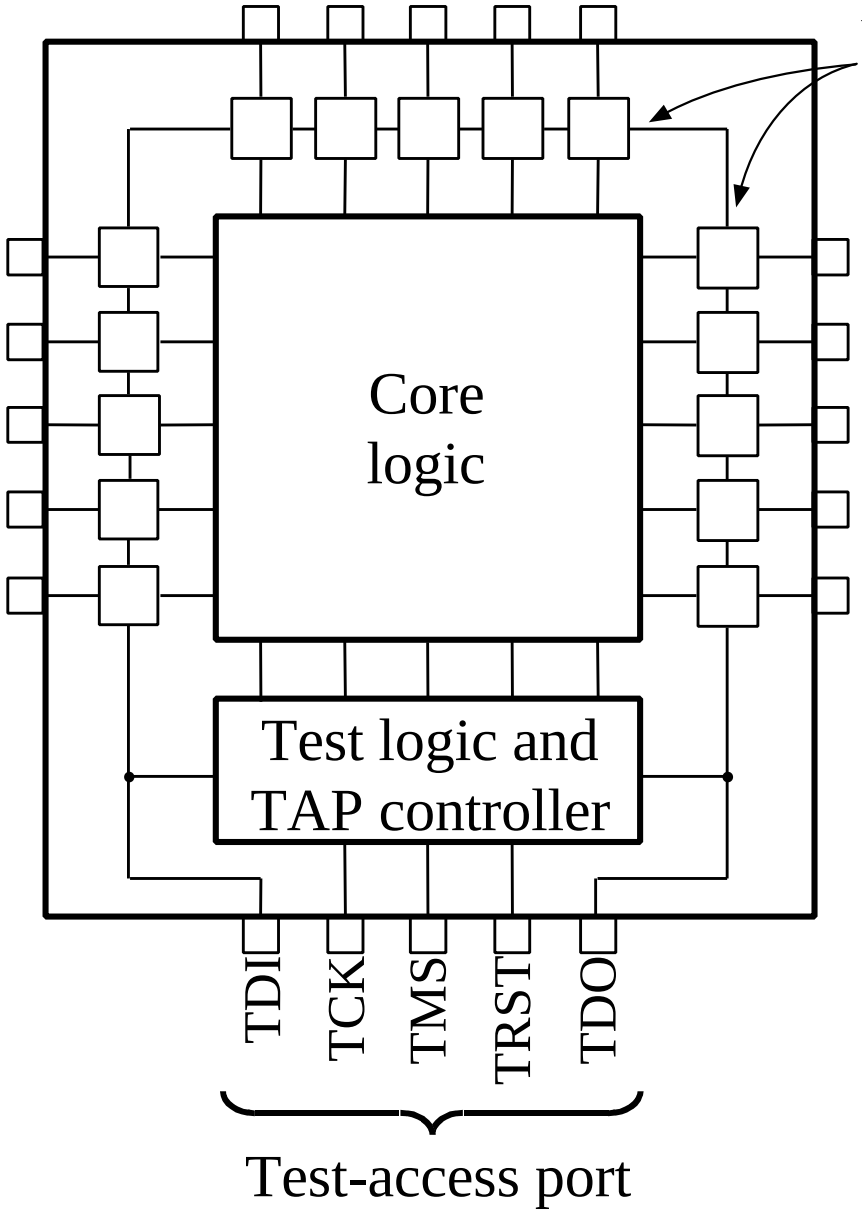


Figure 10-12 IC with Boundary Scan Register and Test-access Port



TDI Test data input (this data is shifted serially into the BSR)

TCK Test clock

TMS Test mode select

TDO Test data output (serial output from the BSR)

TRST Test reset (resets the TAP controller and test logic; optional pin)

Figure 10-13 PC Board with Boundary Scan ICs

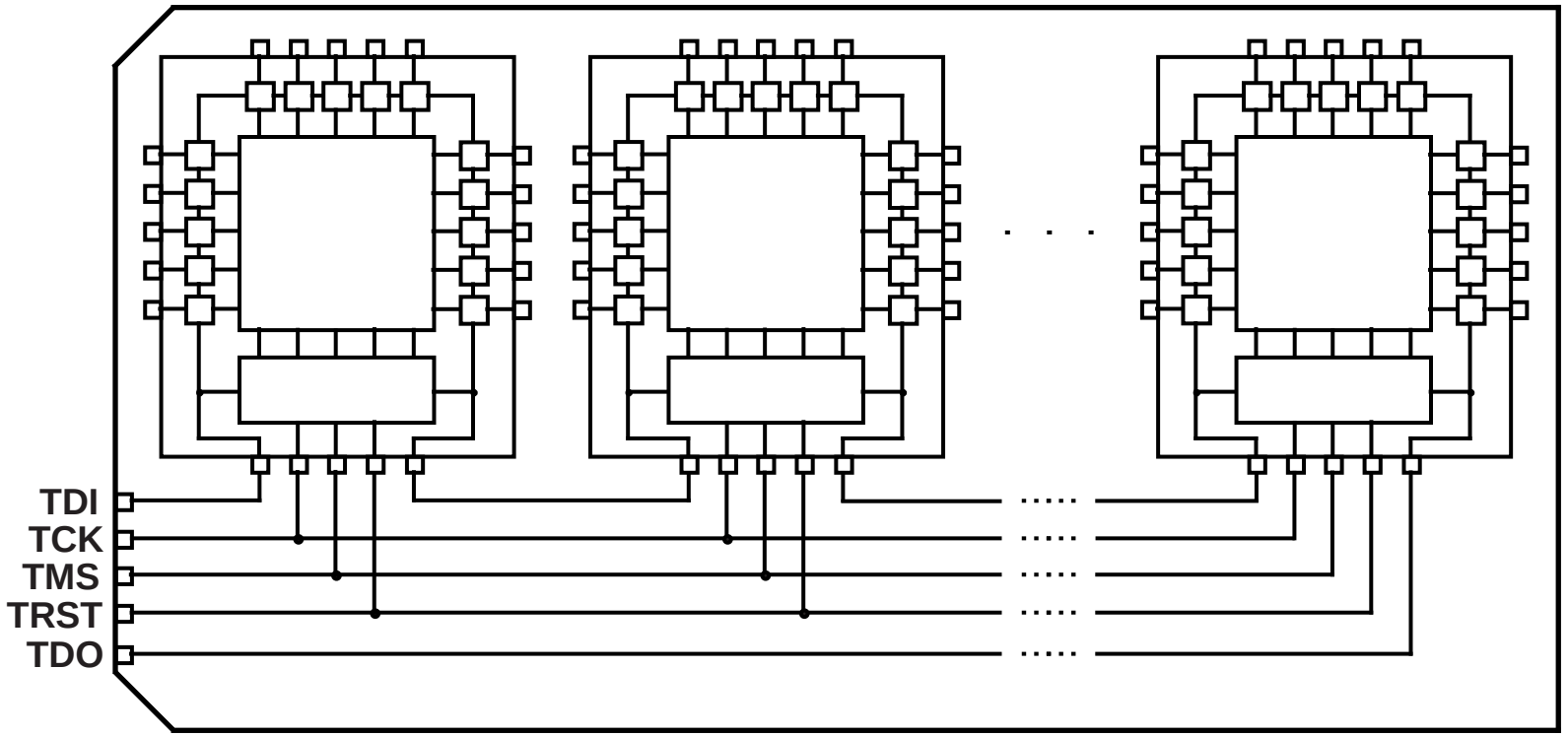


Figure 10-14 Typical Boundary Scan Cell

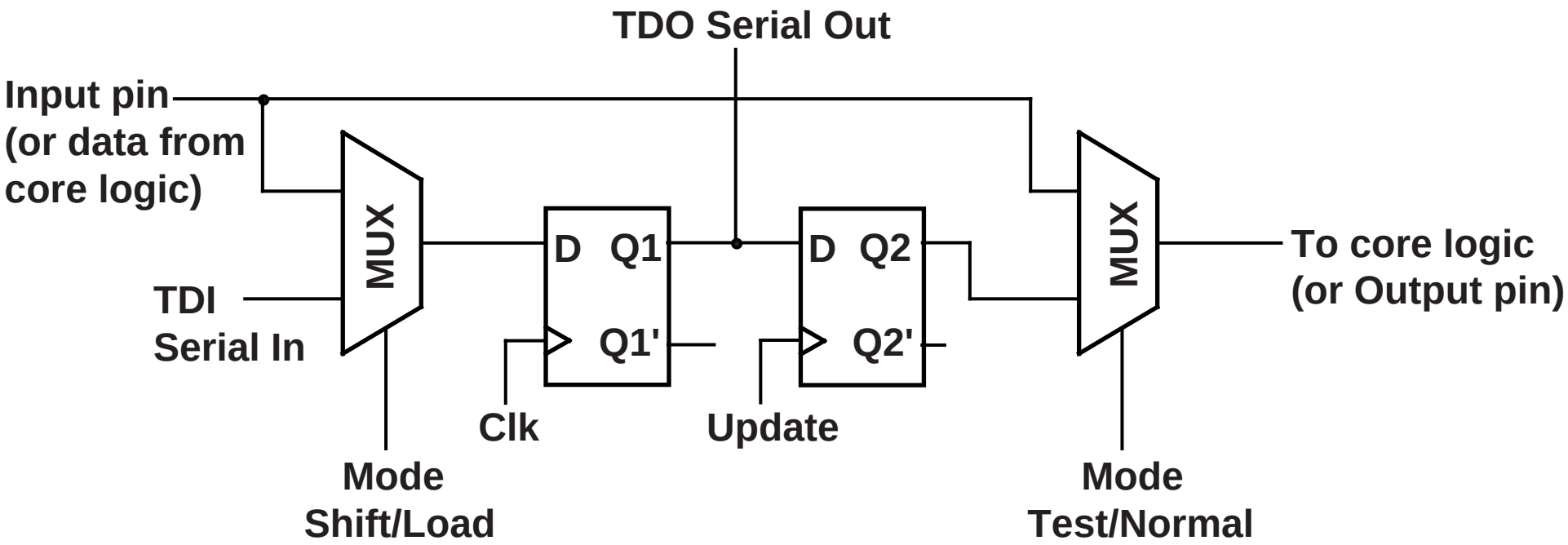
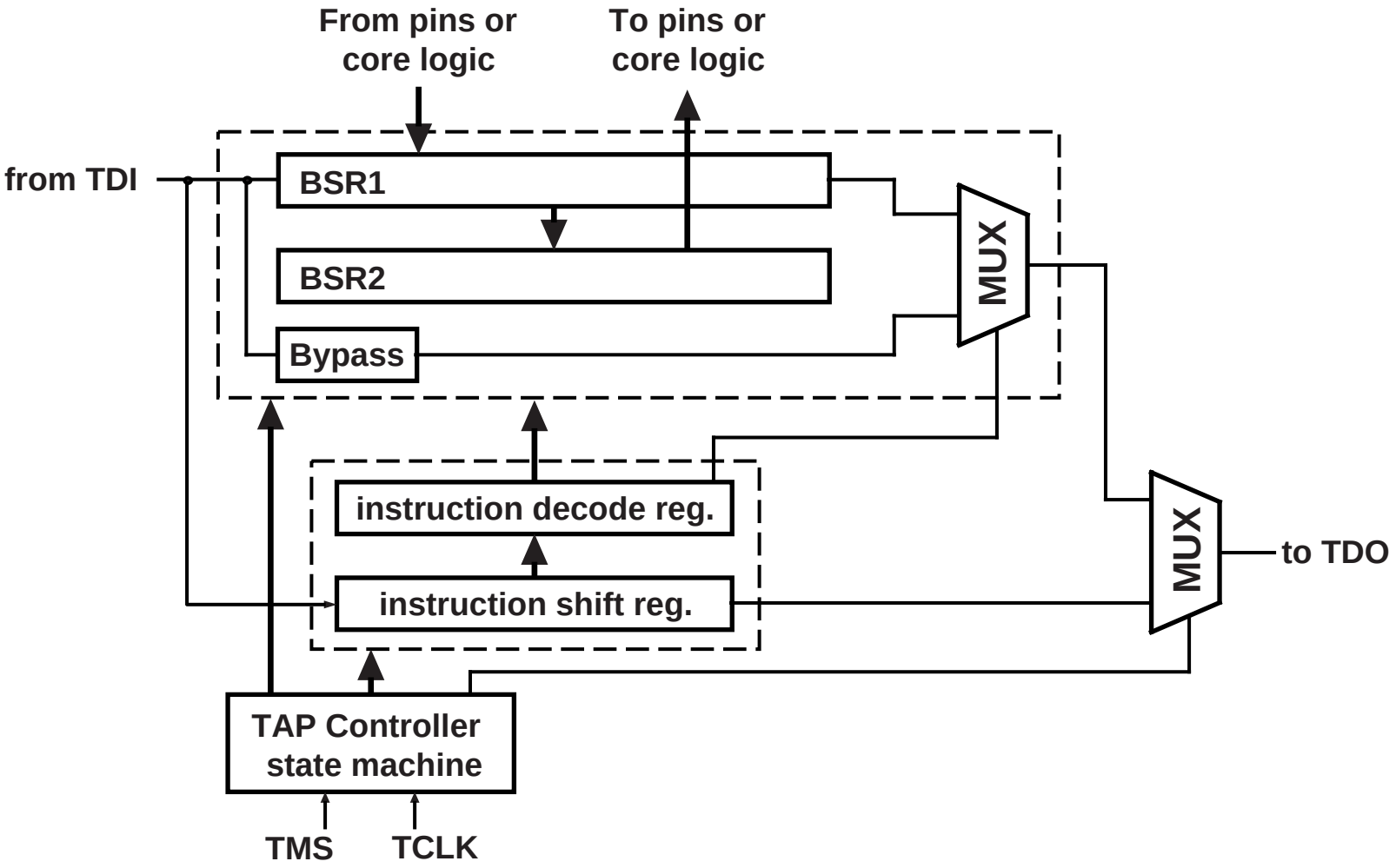


Figure 10-15 Basic Boundary Scan Architecture



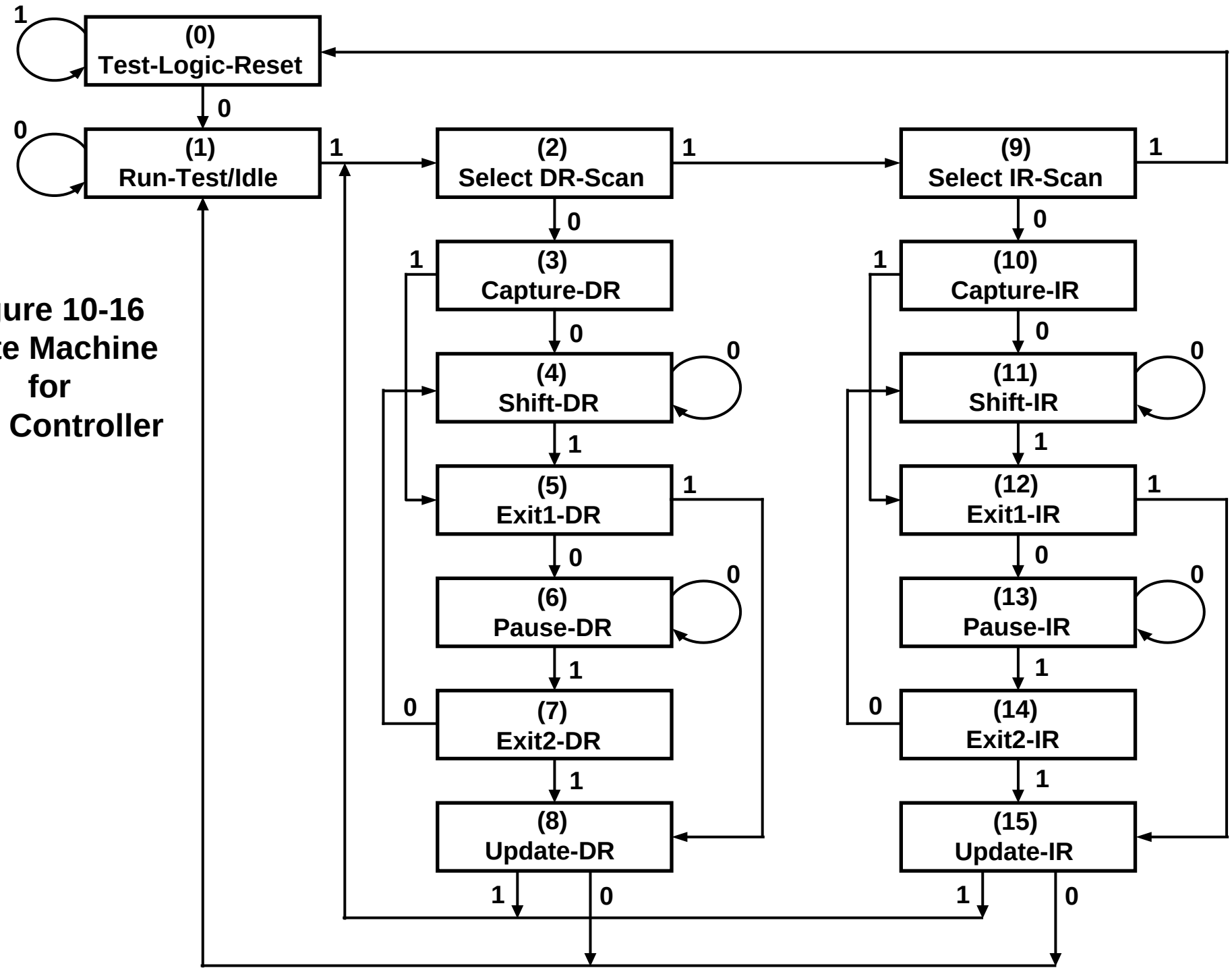


Figure 10-16
State Machine
for
TAP Controller

From Page 354-355

Instructions Defined in IEEE Standard

BYPASS -- allows TDI serial data to go through a 1-bit bypass register on the IC instead of through the boundary scan register.

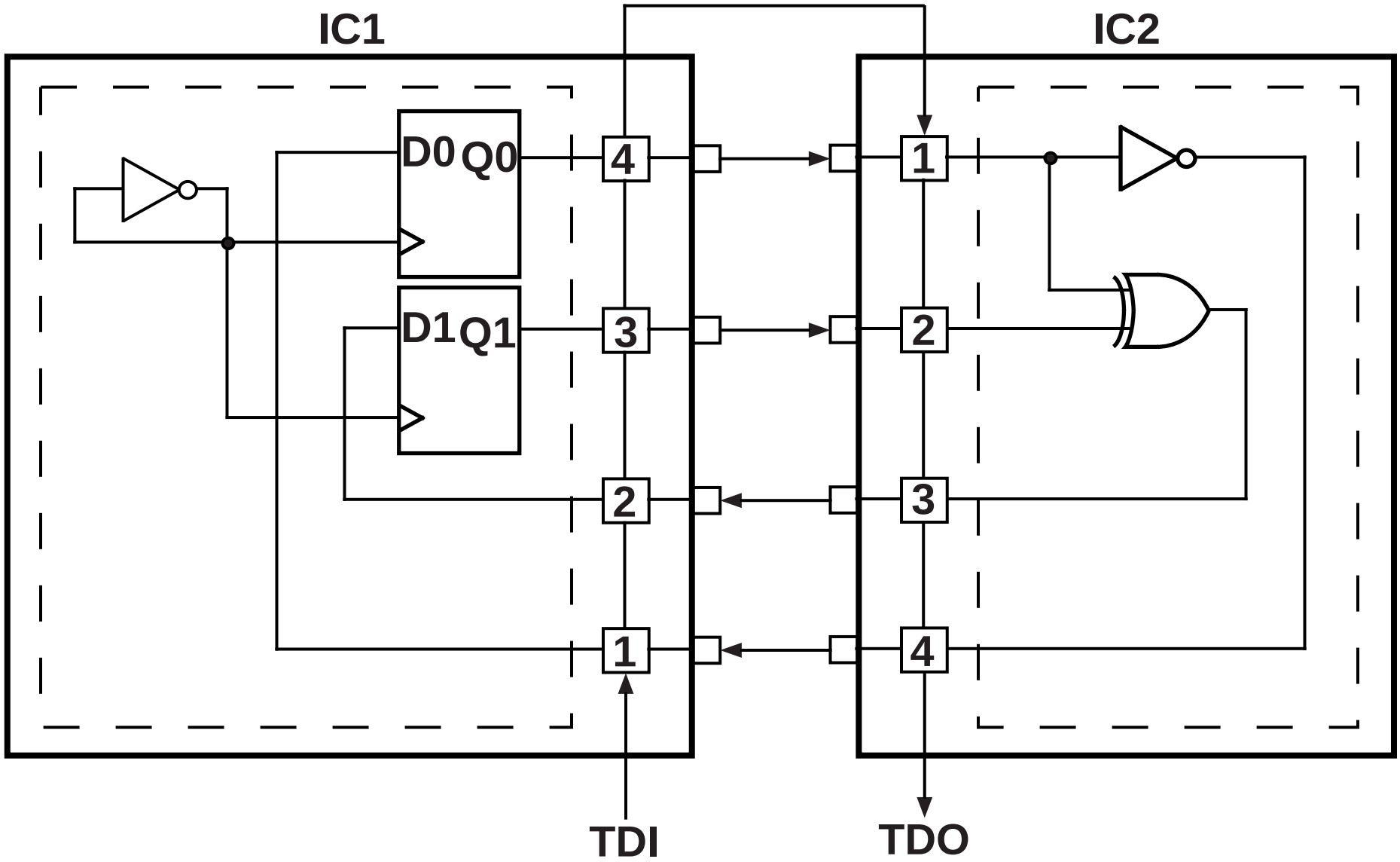
SAMPLE/PRELOAD -- used to scan the boundary-scan register without interfering with the normal core logic operation. Data is transferred to or from the core logic from or to the IC pins without interference. Samples of this data can be taken and scanned out through the boundary scan register. Test data can be shifted into the BSR.

EXTEST -- allows board-level interconnect testing and testing of clusters of components which do not incorporate the boundary scan test features. Test data is shifted into the BSR and then it goes to the output pins. Data from the input pins is captured by the BSR.

INTEST (optional) -- this instruction allows testing of the core logic by shifting test data into the boundary-scan register. Data shifted into the BSR takes the place of data from the input pins, and output data from the core logic is loaded into the BSR.

RUNBIST (optional) -- this instruction causes special built-in self-test (BIST) logic within the IC to execute.

Figure 10-17 Interconnection Testing Boundary Scan



From Page 355-357: Interconnection Test Sequence

1. Reset the TAP state machine to the Test-Logic-Reset state by inputting a sequence of five 1's on TMS. The TAP controller is designed so that a sequence of five 1's will always reset it regardless of the present state. Alternatively, TRST could be asserted if it is available.

2. Scan in the SAMPLE/PRELOAD instruction to both ICs using the sequences for TMS and TDI given below. The state numbers refer to Figure 10-16.

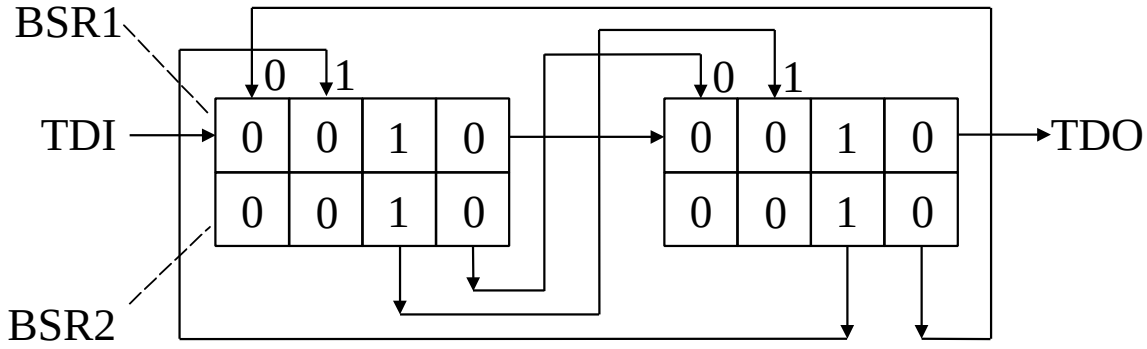
State:	0	1	2	9	10	11	11	11	11	11	11	12	15	2
TMS:	0	1	1	0	0	0	0	0	0	0	1	1	1	
TDI:	–	–	–	–	–	1	0	0	1	0	0	–	–	

The TMS sequence 01100 takes the TAP controller to the Shift-IR state. In this state, copies of the SAMPLE/PRELOAD instruction (code 001) are shifted into the instruction registers on both ICs. In the Update-IR state, the instructions are loaded into the instruction decode registers. Then the TAP controller goes back to the Select DR-scan state.

3. Preload the first set of test data into the ICs using the sequences for TMS and TDI given below.

State:	2	3	4	4	4	4	4	4	4	4	5	8	2
TMS:	0	0	0	0	0	0	0	0	0	1	1	1	
TDI:	–	–	0	1	0	0	0	1	0	0	–	–	

Data is shifted into BSR1 in the Shift-DR state, and it is transferred to BSR2 in the Update-DR state. The result is as follows:



4. Scan in the EXTEST instruction to both ICs using the following sequences:

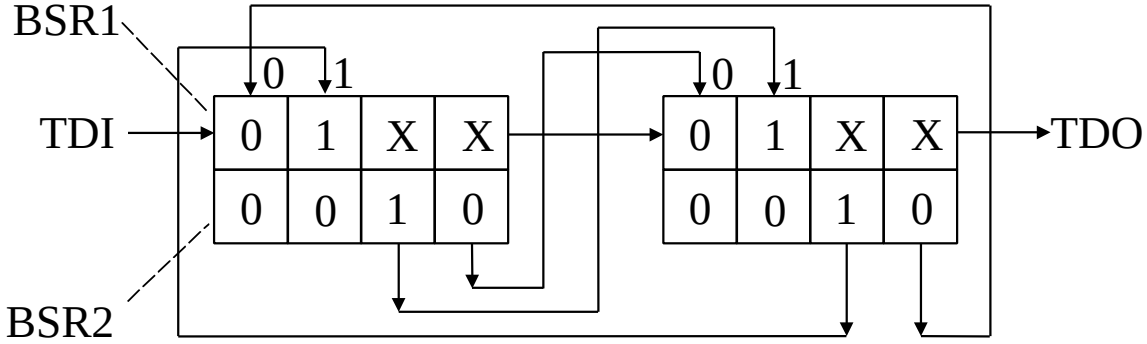
State:	2	9	10	11	11	11	11	11	11	12	15	2
TMS:	1	0	0	0	0	0	0	0	1	1	1	
TDI:	–	–	–	0	0	0	0	0	0	–	–	

The EXTEST instruction (000) is scanned into the instruction register in state Shift-IR and loaded into the instruction decode register in state Update-IR. At this point, the preloaded test data goes to the output pins, and it is transmitted to the adjacent IC input pins via the printed circuit board traces.

5. Capture the test results from the IC inputs. Scan this data out to TDO and scan the second set of test data in using the following sequences:

State:	2	3	4	4	4	4	4	4	4	4	5	8	2
TMS:	0	0	0	0	0	0	0	0	0	1	1	1	
TDI:	–	–	1	0	0	0	1	0	0	0	–	–	
TDO:	–	–	x	x	1	0	x	x	1	0	–	–	

The data from the input pins is loaded into BSR1 in state Capture-DR. At this time, if no faults have been detected, the BSRs should be configured as shown below, where the X's indicate captured data which is not relevant to the test.



The test results are then shifted out of BSR1 in state Shift-DR as the new test data is shifted in. The new data is loaded into BSR2 in the Update-IR state.

6. Capture the test results from the IC inputs. Scan this data out to TDO and scan all 0's in using the following sequences:

State:	2	3	4	4	4	4	4	4	4	4	5	8	2	9	0
TMS:	0	0	0	0	0	0	0	0	0	1	1	1	1	1	
TDI:	–	–	0	0	0	0	0	0	0	0	–	–	–	–	
TDO:	–	–	x	x	0	1	x	x	0	1	–	–	–	–	

The data from the input pins is loaded into BSR1 in state Capture-DR. Then it is shifted out in state Shift-DR as all 0's are shifted in. The 0's are loaded into BSR2 in the Update-IR state. The controller then returns to the Test-Logic-Reset state and normal operation of the ICs can then occur. The interconnection test passes if the observed TDO sequences match the ones given above.

Figure 10-18(a) VHDL Code for Basic Boundary Scan Architecture

```
entity BS_arch is
      generic (NCELLS: natural range 2 to 120 := 2);  -- number of boundary scan cells

      port (TCK, TMS, TDI: in bit;
            TDO: out bit;
            BSRin: in bit_vector(1 to NCELLS);
            BSRout: inout bit_vector(1 to NCELLS);
            CellType: bit_vector(1 to NCELLS));
                  -- '0' for input cell, '1' for output cell
end BS_arch;

architecture behavior of BS_arch is
      signal IR,IDR: bit_vector(1 to 3);                    -- instruction registers
      signal BSR1,BSR2: bit_vector(1 to NCELLS);            -- boundary scan cells
      signal BYPASS: bit;                                   -- bypass bit
      type TAPstate is (TestLogicReset, RunTest_Idle,
            SelectDRScan, CaptureDR, ShiftDR, Exit1DR, PauseDR, Exit2DR, UpdateDR,
            SelectIRScan, CaptureIR, ShiftIR, Exit1IR, PauseIR, Exit2IR, UpdateIR);
      signal St: TAPstate;                                  -- TAP Controller State
begin
      process (TCK)
      begin
            if (TCK='1') then
            -- TAP Controller State Machine
                  case St is
                  when TestLogicReset =>
                        if TMS='0' then St<=RunTest_Idle; else St<=TestLogicReset;  end if;
```

Figure 10-18(b) VHDL Code for Basic Boundary Scan Architecture

```
when RunTest_Idle =>
      if TMS='0' then St<=RunTest_Idle; else St<=SelectDRScan;  end if;
when SelectDRScan =>
      if TMS='0' then St<=CaptureDR;   else St<=SelectIRScan; end if;
when CaptureDR =>
      if IDR = "111" then BYPASS <= '0';
      elsif IDR = "000" then  -- EXTEST (input cells capture pin data)
            BSR1 <= (not CellType and BSRin) or (CellType and BSR1);
      elsif IDR = "001" then  -- SAMPLE/PRELOAD
            BSR1 <= BSRin;  end if;       -- all cells capture cell input data
      if TMS='0' then St<=ShiftDR;       else St<=Exit1DR;        end if;
when ShiftDR =>
      if IDR = "111" then BYPASS <= TDI;  -- shift data though bypass reg.
            else BSR1 <= TDI & BSR1(1 to NCELLS -1); end if;
                  -- shift data into BSR
      if TMS='0' then St<=ShiftDR;       else St<=Exit1DR;        end if;
when Exit1DR =>
      if TMS='0' then St<=PauseDR;      else St<=UpdateDR;      end if;
when PauseDR =>
      if TMS='0' then St<=PauseDR;      else St<=Exit2DR;        end if;
when Exit2DR =>
      if TMS='0' then St<=ShiftDR;       else St<=UpdateDR;      end if;
when UpdateDR =>
      if IDR = "000" then -- EXTEST (update output reg. for output cells)
            BSR2 <= (CellType and BSR1) or (not CellType and BSR2);
      elsif IDR = "001" then   -- SAMPLE/PRELOAD
            BSR2 <= BSR1; end if;         -- update output reg. in all cells
```

Figure 10-18(c) VHDL Code for Basic Boundary Scan Architecture

```
                    if TMS='0' then St<=RunTest_Idle;  else St<=SelectDRScan;     end if;
                when SelectIRScan =>
                    if TMS='0' then St<=CaptureIR;    else St<=TestLogicReset;     end if;
                when CaptureIR =>
                    IR <= "001";  -- load 2 LSBs of IR with 01 as required by the standard
                    if TMS='0' then St<=ShiftIR;      else St<=Exit1IR;            end if;
                when ShiftIR =>
                    IR <= TDI & IR(1 to 2);  -- shift in instruction code
                    if TMS='0' then St<=ShiftIR;      else St<=Exit1IR;            end if;
                when Exit1IR =>
                    if TMS='0' then St<=PauseIR;      else St<=UpdateIR;           end if;
                when PauseIR =>
                    if TMS='0' then St<=PauseIR;      else St<=Exit2IR;            end if;
                when Exit2IR =>
                    if TMS='0' then St<=ShiftIR;      else St<=UpdateIR;           end if;
                when UpdateIR =>
                    IDR <= IR;   -- update instruction decode register
                    if TMS='0' then St<=RunTest_Idle;  else St<=SelectDRScan;     end if;
                end case;
            end if;
        end process;

        TDO <= BYPASS when St = ShiftDR and IDR = "111"  -- BYPASS
            else BSR1(NCELLS) when St=ShiftDR            -- EXTEST or SAMPLE/PRELOAD
            else IR(3) when St=ShiftIR;
        BSRout <= BSRin when (St=TestLogicReset or not (IDR="000"))
            else BSR2;                                   -- define cell outputs
end behavior;
```

Figure 10-19(a) VHDL Code for Interconnection Test Example

```
-- Boundary Scan Tester
entity system is
end system;
architecture IC_test of system is
      component BS_arch is
            generic (NCELLS:natural range 2 to 120:= 4); --number of boundary scan cells
            port (TCK, TMS, TDI: in bit;
                  TDO: out bit;   BSRin: in bit_vector(1 to NCELLS);
                  BSRout: inout bit_vector(1 to NCELLS);
                  CellType: in bit_vector(1 to NCELLS));
                        -- '0' for input cell, '1' for output cell
      end component;
      signal TCK,TMS,TDI,TDO,TDO1: bit;  signal Q0, Q1, CLK1: bit;
      signal BSR1in, BSR1out, BSR2in, BSR2out: bit_vector(1 to 4);
      signal count: integer := 0;
      constant TMSpattern: bit_vector(0 to 62) :=
            "01100000001110000000001111000000011100000000011100000000111111";
      constant TDIpattern: bit_vector(0 to 62) :=
            "00000100100000001000100000000000000001000100000000000000000000";
begin
      BS1: BS_arch port map(TCK, TMS, TDI, TDO1, BSR1in, BSR1out, "0011");
      BS2: BS_arch port map(TCK, TMS, TDO1, TDO, BSR2in, BSR2out, "0011");
      -- BSR each has two input cells and two output cells

      BSR1in(1) <= BSR2out(4);                        -- IC1 external connections
      BSR1in(2) <= BSR2out(3);
      BSR1in(3) <= Q1;                                -- IC1 internal logic
      BSR1in(4) <= Q0;
```

Figure 10-19(b) VHDL Code for Interconnection Test Example

```
    CLK1 <= not CLK1 after 7 ns;                          -- internal clock
    process(CLK1)
    begin
        if (CLK1='1') then  Q0 <= BSR1out(1); Q1 <= BSR1out(2);  end if; -- D flip-flops
    end process;

    BSR2in(1) <= BSR1out(4);                              -- IC2 external connections
    BSR2in(2) <= BSR1out(3);
    BSR2in(3) <= BSR2out(1) xor BSR2out(2);               -- IC2 internal logic
    BSR2in(4) <= not BSR2out(1);
    TCK <= not TCK after 5 ns;                            -- test clock
    process
    begin
        TMS <= '1';
        wait for 70 ns;                                   -- run internal logic
        wait until TCK='1';
        for i in TMSpattern'range loop                    -- run scan test
            TMS <= TMSpattern(i);
            TDI <= TDIpattern(i);
            wait for 0 ns;
            count <= i+1;                                 -- count triggers listing output
            wait until TCK='1';
        end loop;
        wait for 70 ns;                                   -- run internal logic
        wait;                                             -- stop
    end process;
end IC_test;
```

Figure 10-20 Generic BIST Scheme

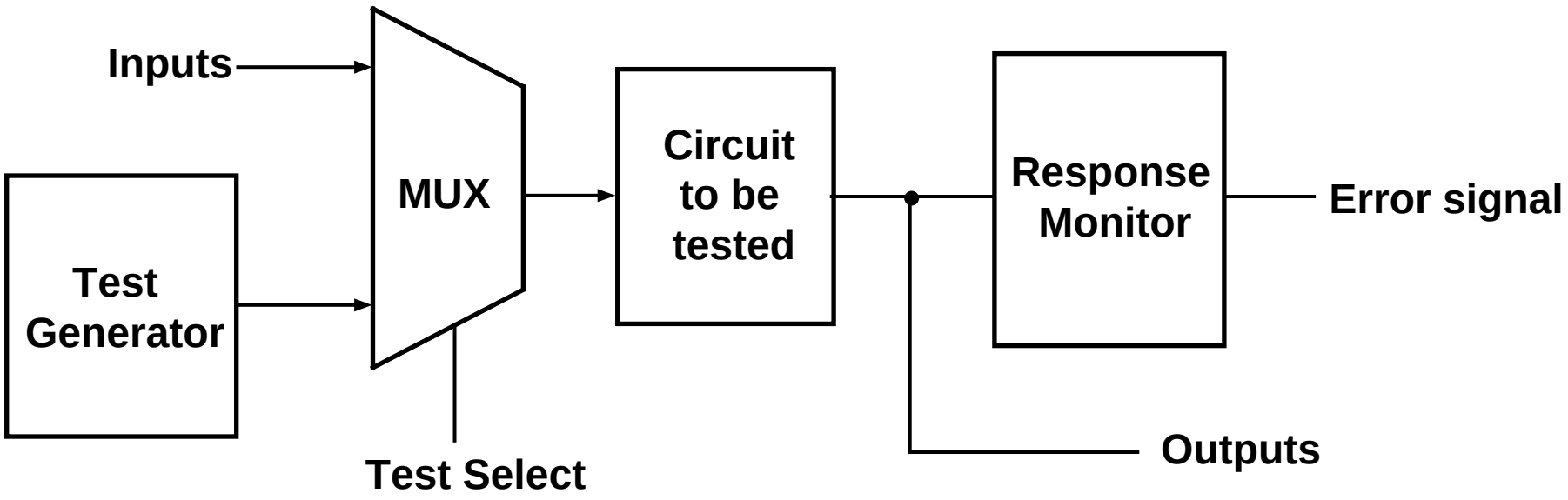


Figure 10-21 Self-test Circuit for RAM

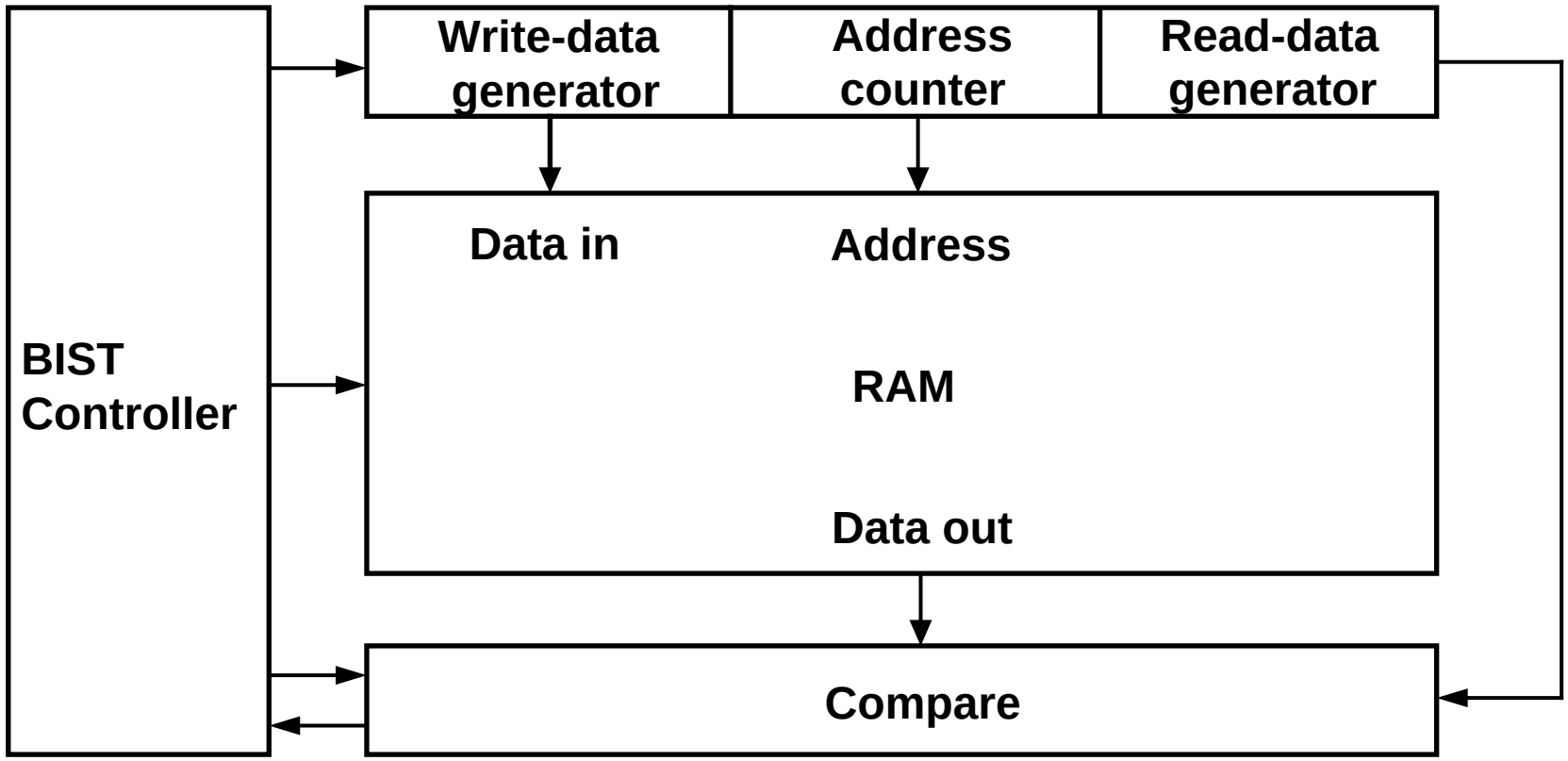


Figure 10-22 Self-test Circuit for RAM with Signature Register

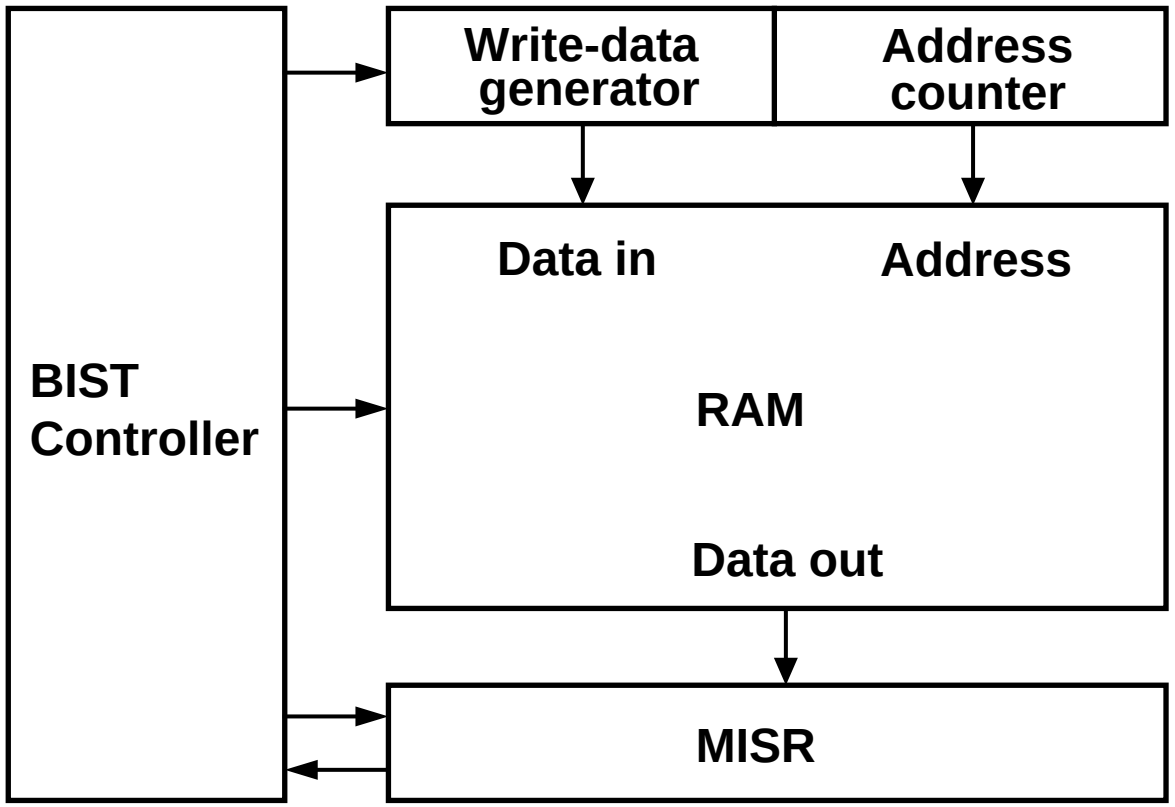


Figure 10-23 4-bit Linear Feedback Shift Register (LFSR)

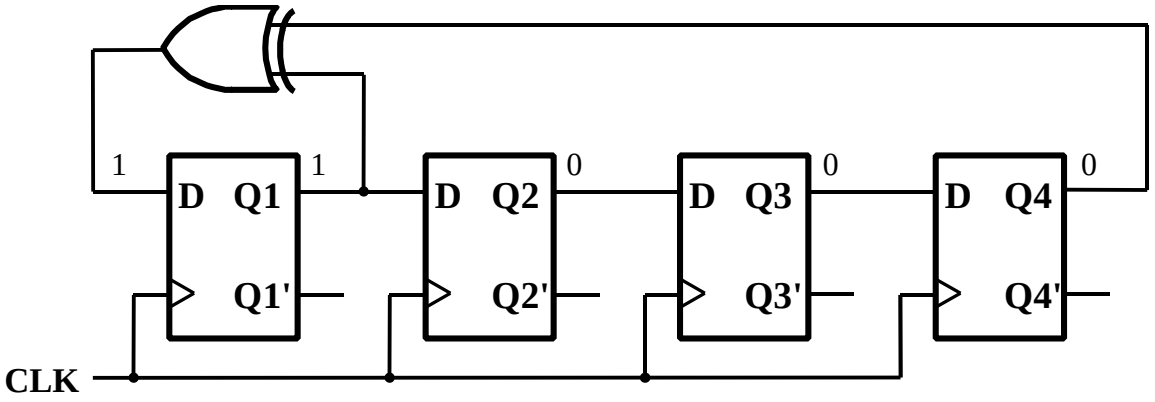


Table 10-4 Feedback for Maximum-length LFSR Sequence

n	Feedback
4, 6, 7	$Q_1 \oplus Q_n$
5	$Q_2 \oplus Q_5$
8	$Q_2 \oplus Q_3 \oplus Q_4 \oplus Q_8$
12	$Q_1 \oplus Q_4 \oplus Q_6 \oplus Q_{12}$
14, 16	$Q_3 \oplus Q_4 \oplus Q_5 \oplus Q_n$
24	$Q_1 \oplus Q_2 \oplus Q_7 \oplus Q_{24}$
32	$Q_1 \oplus Q_2 \oplus Q_{22} \oplus Q_{32}$

Patterns generated are:

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

Figure 10-24 Modified LFSR with 0000 State

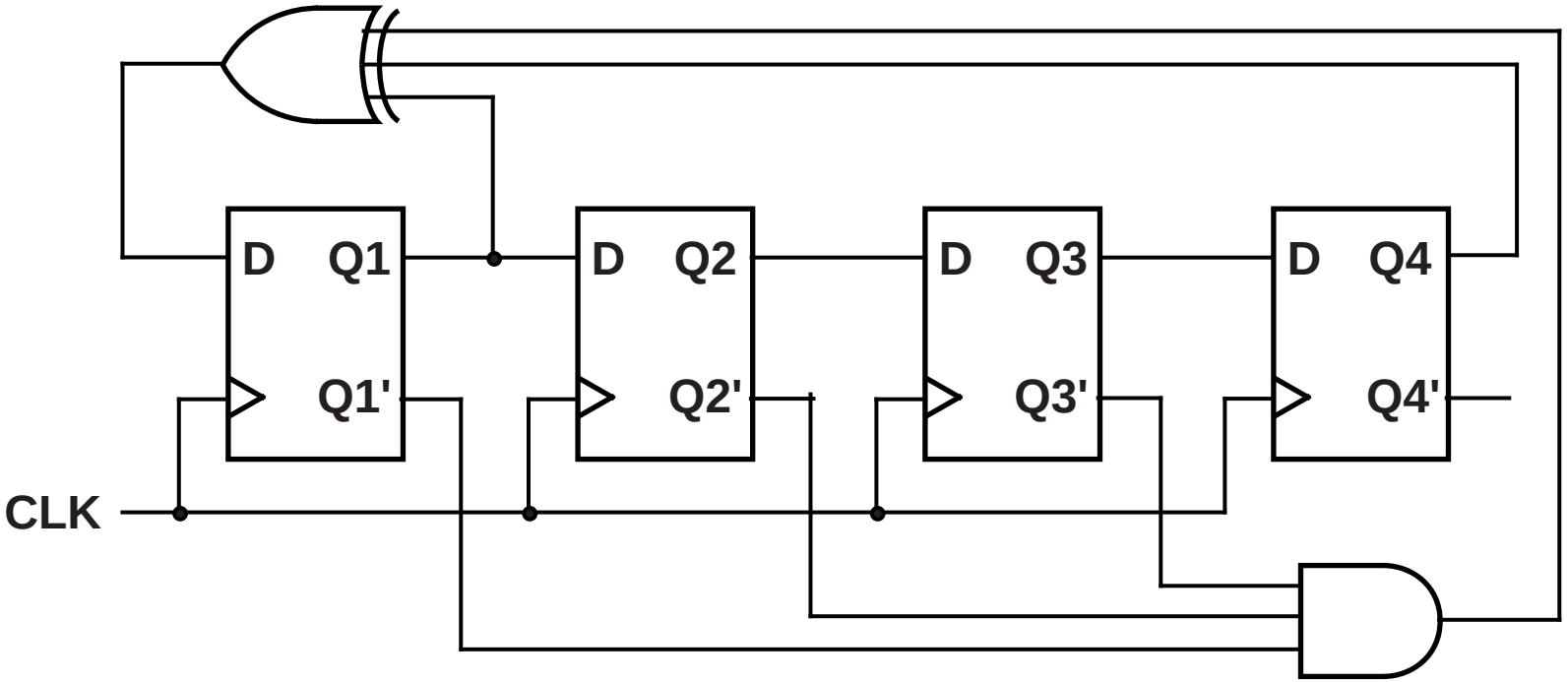


Figure 10-25 Multiple-Input Signature Register (MISR)

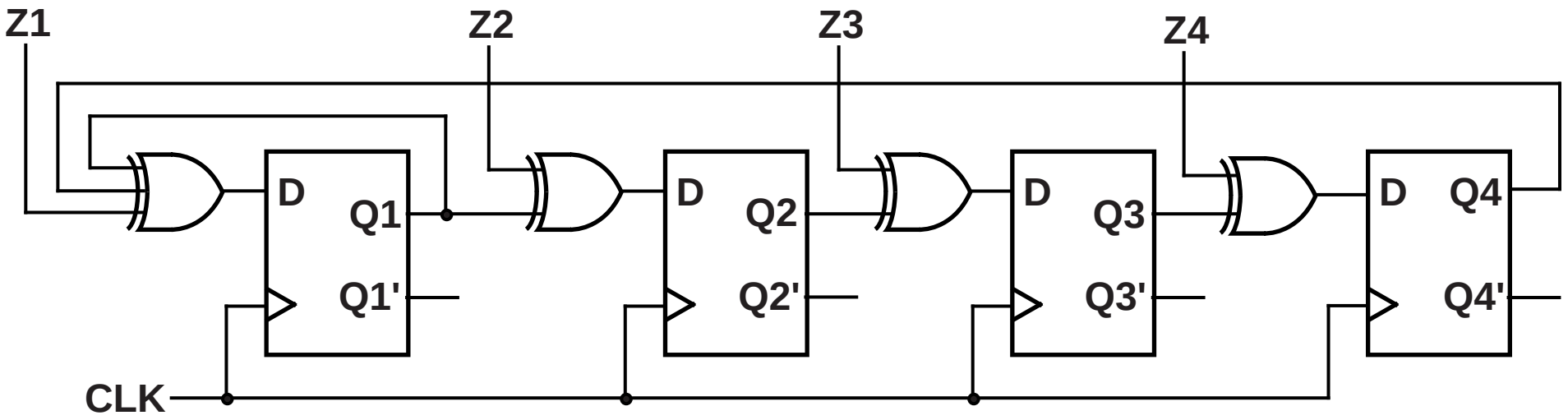
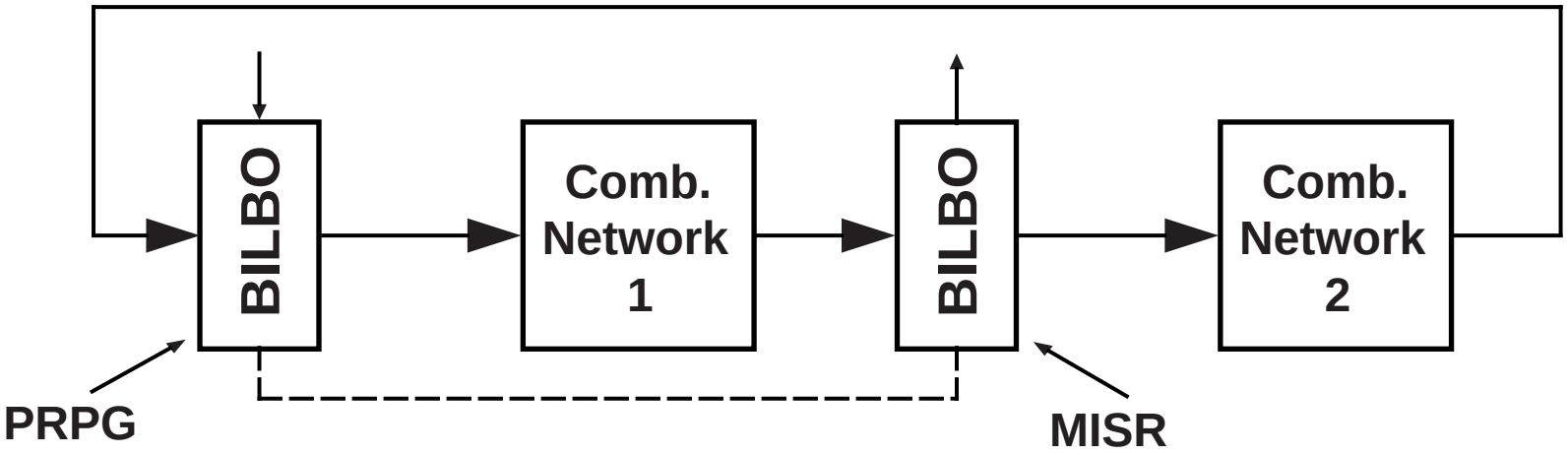
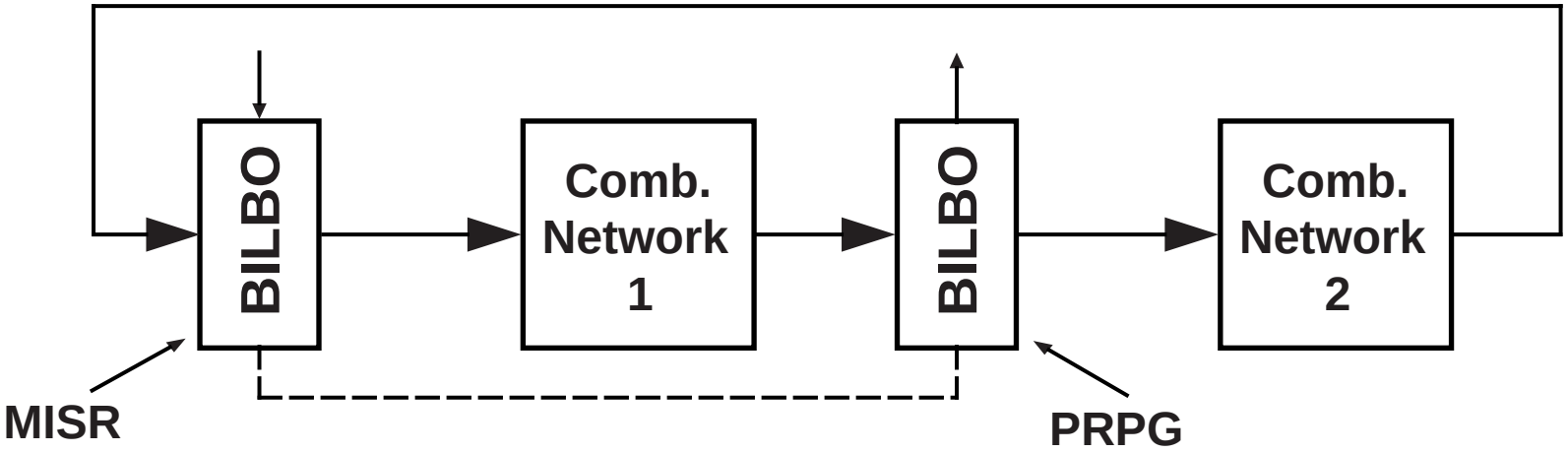


Figure 10-26 BIST Using BILBO Registers



(a) Testing combinational network 1



(b) Testing combinational network 2

Figure 10-27 4-bit BILBO Register

Equations for BILBO register:

$D_1 = Z_1 B_1 \oplus (Si B_2' + FB B_2)(B_1' + B_2)$

$D_i = Z_i B_1 \oplus Q_{i-1}(B_1' + B_2) \quad (i > 1)$

When $B_1 = B_2 = 0$ (shift register mode):

$D_1 = Si$ and $D_i = Q_{i-1} \quad (i > 1)$

When $B_1 = 0$ and $B_2 = 1$ (PRPG mode):

$D_1 = FB$, $D_i = Q_{i-1}$

When $B_1 = 1$ and $B_2 = 0$ (normal mode):

$D_1 = Z_1$, $D_i = Z_i$

When $B_1 = B_2 = 1$:

$D_1 = Z_1 \oplus FB$, $D_i = Z_i \oplus Q_{i-1}$

B1B2	operating mode
00	shift register
01	PRPG
10	normal
11	MISR

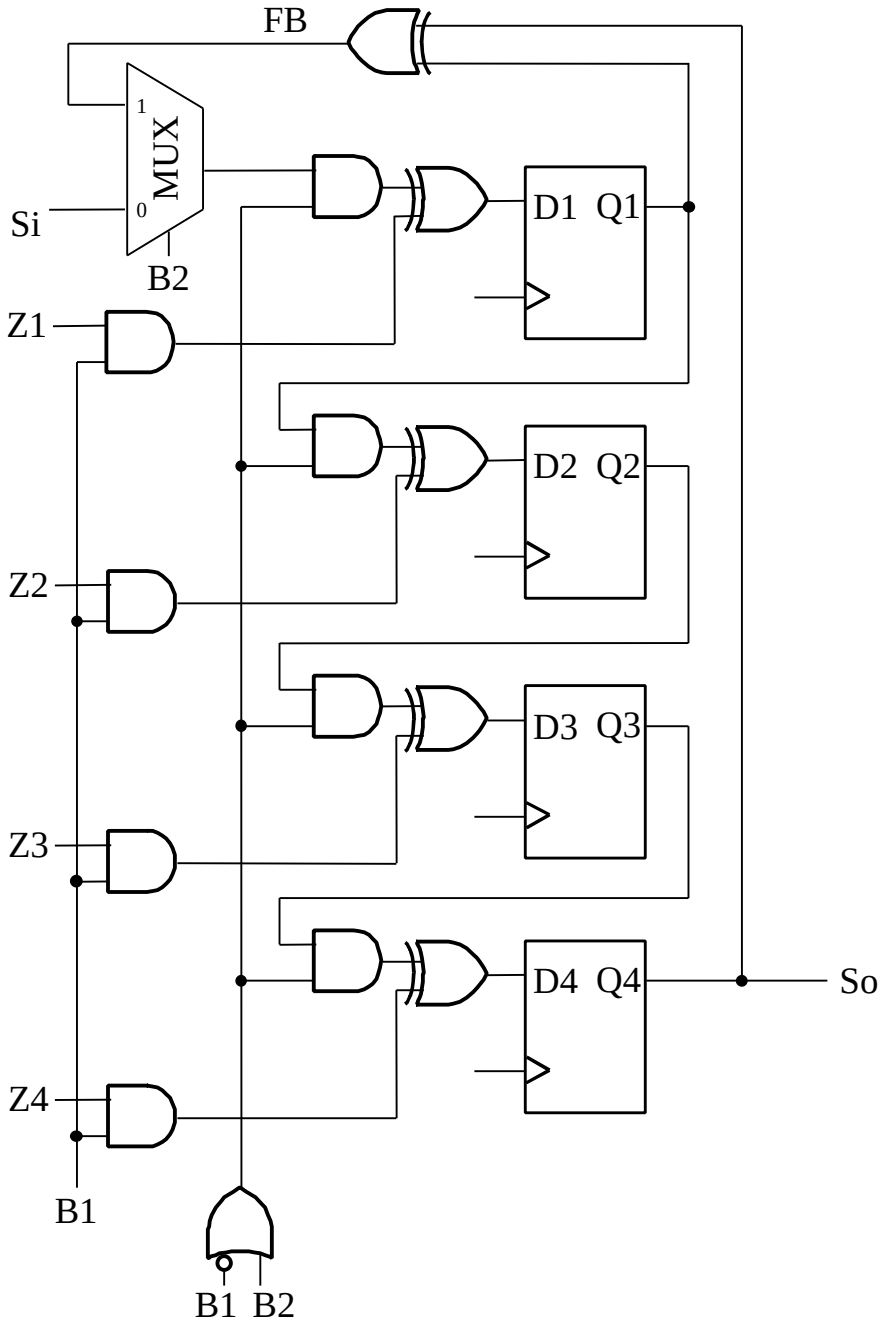


Figure 10-28 VHDL Code for BILBO Register of Figure 10-27

```
entity BILBO is                                          -- BILBO Register
      generic (NBITS: natural range 4 to 8 := 4);
      port (Clk, CE, B1, B2, Si: in bit;
            So: out bit;     Z: in bit_vector(1 to NBITS);  Q: inout bit_vector(1 to NBITS));
end BILBO;
architecture behavior of BILBO is  signal FB: bit;
begin
      FB <=  Q(2) xor Q(3) xor Q(4) xor Q(NBITS) when (NBITS=8)
            else Q(2) xor Q(NBITS) when (NBITS=5)  else Q(1) xor Q(NBITS);
      process(Clk)
            variable mode: bit_vector(1 downto 0);
      begin
            if (Clk = '1' and CE = '1') then
                  mode := B1 & B2;
                  case mode is
                        when "00" =>                  -- Shift register mode
                              Q <= Si & Q(1 to NBITS-1);
                        when "01" =>                  -- Pseudo Random Pattern Generator mode
                              Q <= FB & Q(1 to NBITS-1);
                        when "10" =>                  -- Normal Operating mode
                              Q <= Z;
                        when "11" =>                  -- Multiple Input Signature Register mode
                              Q <= Z(1 to NBITS) xor (FB & Q(1 to NBITS-1));
                  end case;
            end if;
      end process;
      So <= Q(NBITS);
end;
```

Figure 10-29 System with BILBO Registers and Tester

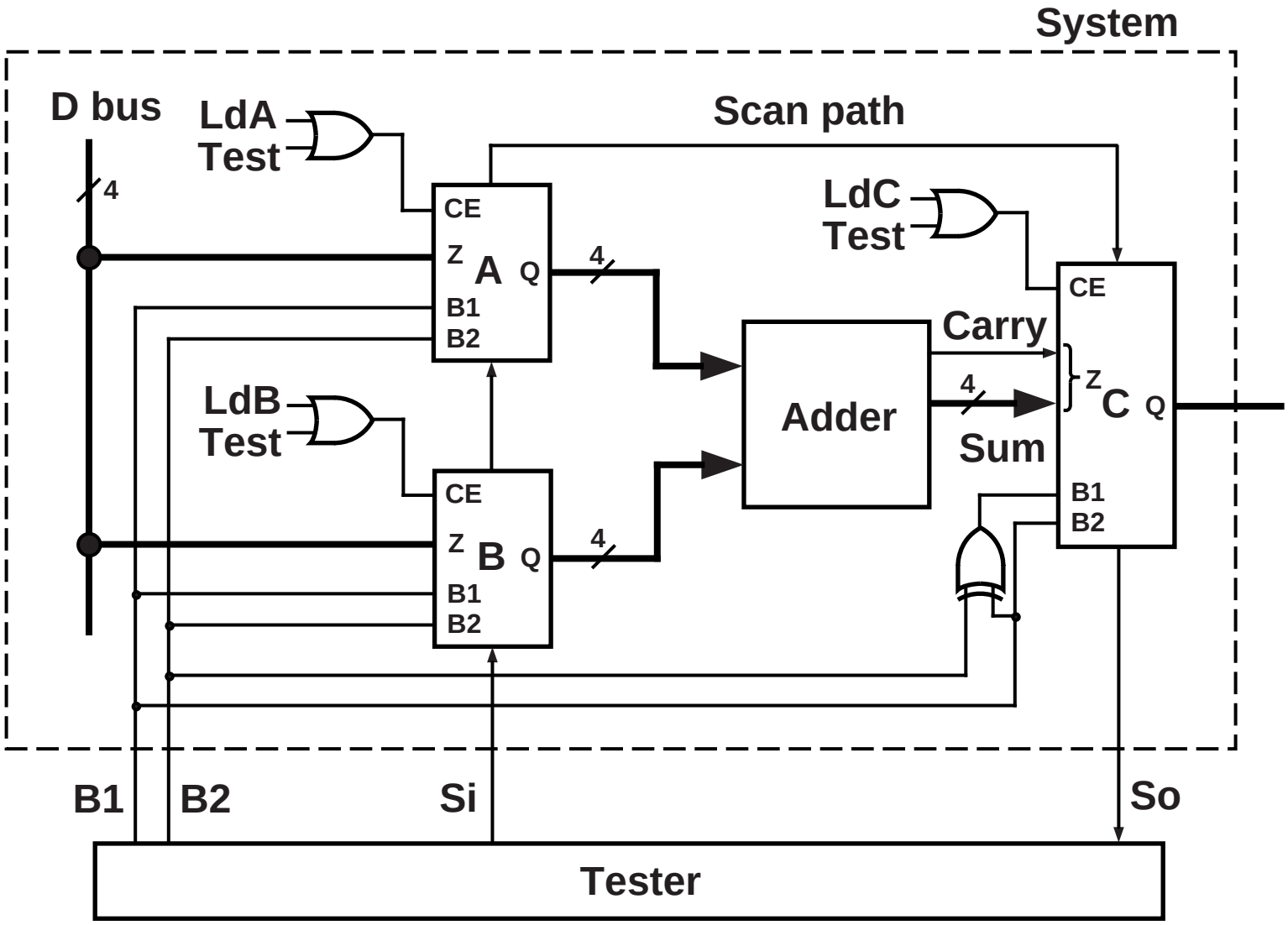


Figure 10-30 VHDL Code for System with BILBO Registers and Tester

```
entity BILBO_System is
      port (Clk, LdA, LdB, LdC, B1, B2, Si: in bit;
            So: out bit;  DBus: in bit_vector(3 downto 0);
            Output: inout bit_vector(4 downto 0));
end BILBO_System;
architecture BSys1 of BILBO_System is
      component Adder4 is
      port (A, B: in bit_vector(3 downto 0); Ci: in bit;
            S: out bit_vector(3 downto 0); Co:out bit);
      end component;
      component BILBO is
      generic (NBITS: natural range 4 to 8 := 4);
      port (Clk, CE, B1, B2, Si : in bit;
            So: out bit;  Z: in bit_vector(1 to NBITS);  Q: inout bit_vector(1 to NBITS));
      end component;
      signal Aout, Bout: bit_vector(3 downto 0);  signal Cin: bit_vector(4 downto 0);
      alias Carry: bit is Cin(4);  alias Sum: bit_vector(3 downto 0) is Cin(3 downto 0);
      signal ACE, BCE, CCE, CB1, Test, S1, S2: bit;
begin
      Test <= not B1 or B2;  ACE <= Test or LdA;  BCE <= Test or LdB;
      CCE <= Test or LdC;  CB1 <= B1 xor B2;
      RegA: BILBO generic map (4) port map(Clk, ACE, B1, B2, S1, S2, DBus, Aout);
      RegB: BILBO generic map (4) port map(Clk, BCE, B1, B2, Si, S1, DBus, Bout);
      RegC: BILBO generic map (5) port map(Clk, CCE, CB1, B2, S2, So, Cin, Output);
      Adder: Adder4 port map(Aout, Bout, '0', Sum, Carry);
end BSys1;
```

Figure 10-31(a) Test Bench for BILBO System

```
-- System with BILBO test bench

entity BILBO_test is
end BILBO_test;

architecture Btest of BILBO_test is
	component BILBO_System is
	port (Clk, LdA, LdB, LdC, B1, B2, Si: in bit;
		So: out bit;
		DBus: in bit_vector(3 downto 0);
		Output: inout bit_vector(4 downto 0));
	end component;

	signal Clk: bit := '0';
	signal LdA, LdB, LdC, B1, B2, Si, So: bit := '0';
	signal DBus: bit_vector(3 downto 0);
	signal Output: bit_vector(4 downto 0);
	signal Sig: bit_vector(4 downto 0);

	constant test_vector: bit_vector(12 downto 0) := "1000110000000";
	constant test_result: bit_vector(4 downto 0) := "01011";
```

Figure 10-31(b) Test Bench for BILBO System

```
begin
      clk <= not clk after 10 ns;
      Sys: BILBO_System port map(Clk,Lda,LdB,LdC,B1,B2,Si,So,DBus,Output);
      process
      begin
      B1 <= '0'; B2 <= '0';          -- shift in test vector
      for i in test_vector'right to test_vector'left loop
            Si <= test_vector(i);
            wait until (clk = '1');
      end loop;
      B1 <= '0'; B2 <= '1';          -- Use PRPG and MISR
      for i in 1 to 15 loop
            wait until (clk = '1');
      end loop;
      B1 <= '0'; B2 <= '0';          -- Shift Signature out
      for i in 0 to 5 loop
            Sig <= So & Sig(4 downto 1);
            wait until (clk = '1');
      end loop;
      if (Sig = test_result) then     -- Compare signature
            report "System passed test.";
      else
            report "System did not pass test!";
      end if;
      wait;
      end process;
end Btest
```