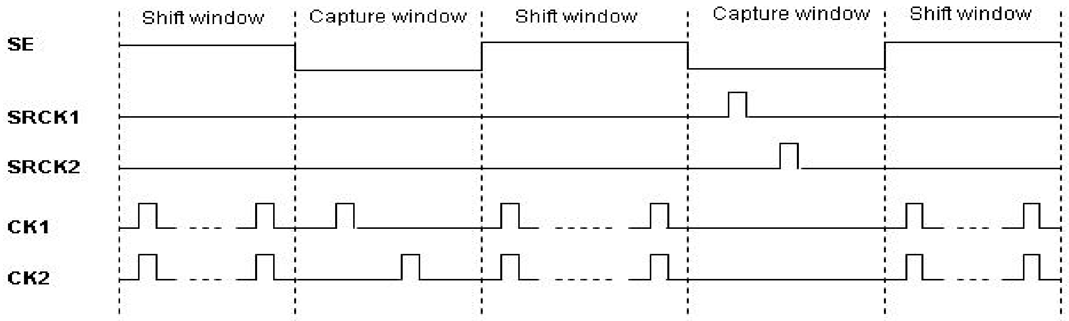
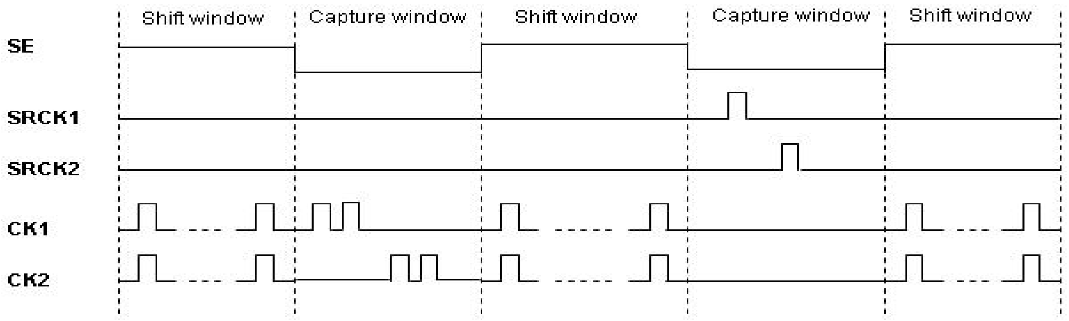
Chapter 5 Exercise Solutions

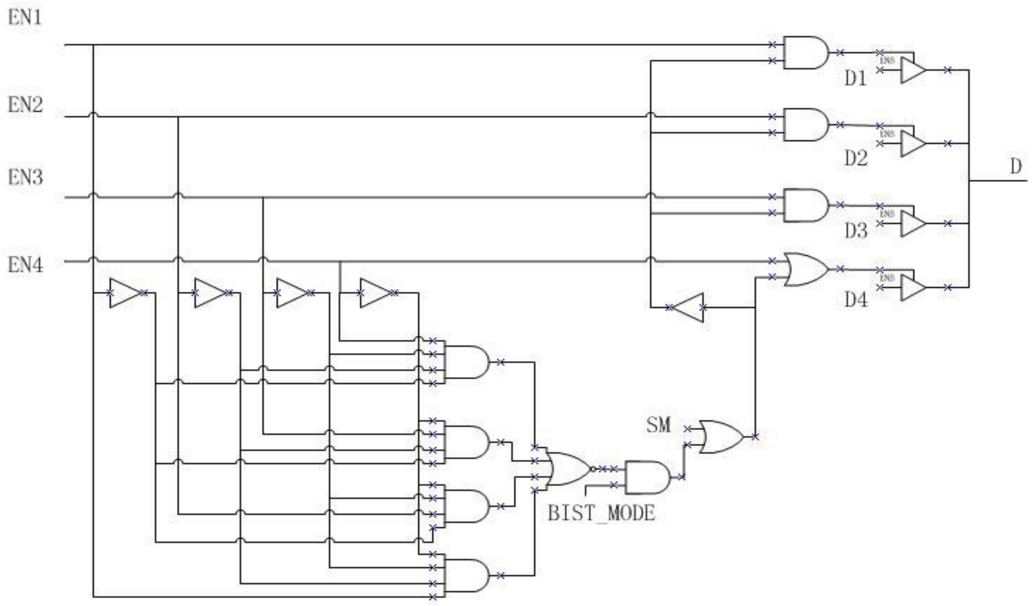
5.1



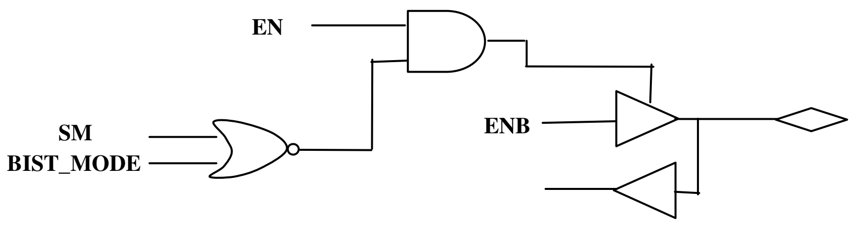


The second timing diagram shows that more delay faults in functional logic can be tested with a double-capture approach.

5.2



5.3 Reconfigure a bidirectional bus to input mode



5.4

To insert a zero state in each hybrid LFSR, an XOR gate is inserted into the last stage of the hybrid LFSR, and a NOR gate (with *n*-1 inputs from the first *n*-1 stages of LFSR) is used as a zero detector. For minimization in a standard CFSR, it can be done by Boolean minimization. For a modular CFSR, we can replace the XOR gate at the last stage of the CFSR with an OR gate and reconnect the tap to the OR-gate output. The period is 32.

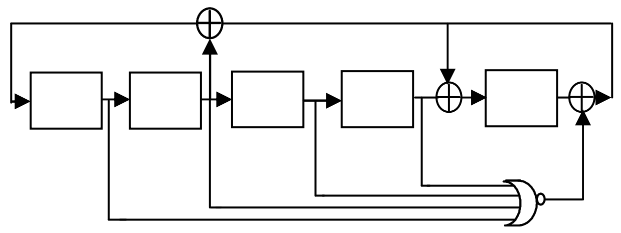


Figure 4a: Adding zero state into 5.13(a)

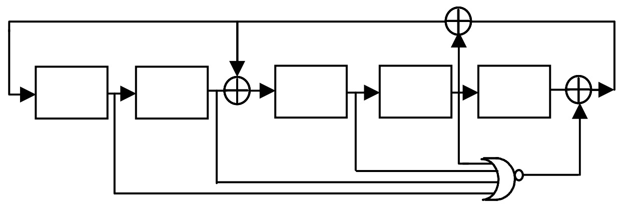
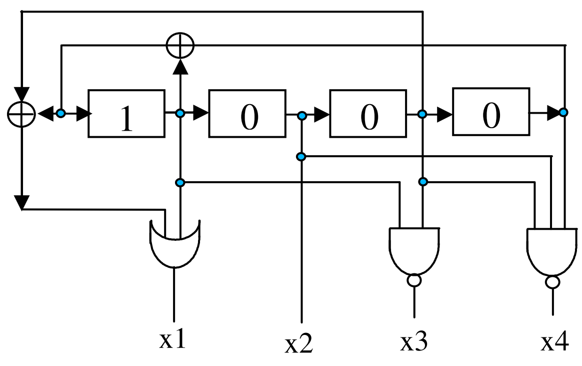


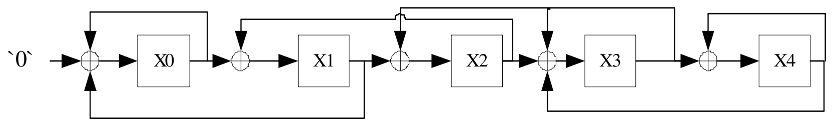
Figure 4b: Adding zero state into 5.13(b)

5.5



5.6

1) A 5-stage CA



The test sequence generated is:

00001	01001	00010	11110
00011	10111	00111	
00100	10110	01110	
01010	10101	11001	
10011	10001	01111	
11100	11011	11010	
00110	01000	01011	
01101	10100	10000	
11101	10010	11000	
00101	11111	01100	

From the above sequence, we can find the 5-stage CA generates a maximum-length of 31 distinct states.

2)
One of the construction rules for cellular automata of length 54 is 404,121,000,146,677,141.
One of the construction rules for cellular automata of length 55 is 1,204,121,000,146,676,412.

5.7

For (n,w) = (8,3) compute the smallest integer k such that $w < [k/(n-k+1)]+[k/(n-k+1)]$ that is $3 < [k/(9-k)]+[k/(9-k)]$, then $k = 6$. The primitive polynomial of degree 6 is $1+x+x^6$ using $f(x) = g(x)p(x) = (1+x+x^2)(1+x+x^6) = 1+x^3+x^6+x^7+x^8$.

Comparison with the (8,5) cyclic LFSR given in Figure 6.15:

	Number of XOR Gates	***Test Length***
(8,3) Condensed LFSR	3	63
(8,5) Cyclic LFSR	3	31

5.8

Given the (15,5) cyclic LFSR with $g(x) = (1+x)(1+x+x^4) = 1+x^2+x^4+x^5$, $h(x) = (1+x^{15})/g(x) = 1+x^2+x^5+x^6+x^8+x^9+x^{10}$, $p(x) = 1+x^2+x^3+x^4+x^5$, and $f(x) = h(x)p(x) = 1+x^3+x^5+x^8+x^9+x^{11}+x^{12}+x^{13}+x^{15}$. Figure 1 shows the (15,5) LFSR with $S_0(x) = h(x) =$ < 101001101110000 >. It uses seven XOR gates.

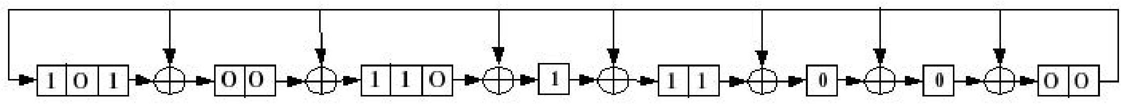


Figure 8: A (15,5) LFSR

In order to get (8,5) cyclic LFSR from (15,5) cyclic LFSR, let's pick the first 6 stages and the last 2 stages of the (15,5) cyclic LFSR that uses the least number of XOR gates. Then try to find the inputs necessary to produce the present state, $x_i(t)$, or the next state, $x_i(t+1)$, of the shortened (8, 5) LFSR with the same initial state.

1. Find the next state of the (8,5) LFSR from the (15,5) LFSR. From $f(x)$ and Figure 1, we know that $x_{13}(t+1) = \quad x_{12}(t) + x_{14}(t)$.
2. Find the present state of the (8,5) LFSR from the (15,5) LFSR.
   Matrix H below is a generator matrix of the cyclic code generated by $h(x)$. The H matrix is formed by using $h(x) = 1+x^2+x^5+x^6+x^8+x^9+x^{10}$.

$$H=\begin{bmatrix} x0 & x1 & x2 & x3 & x4 & x5 & x6 & x7 & x8 & x9 & x10 & x11 & x12 & x13 & x14 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$$

It has a rank k = 5. Let x_i (i = 1, 2, ..., 14) denote the ith column of matrix H. From matrix H, we know that $x_{13}(t+1) = x_{12}(t) + x_{14}(t) = x_5(t) + x_2(t)$. So, the (8, 5) cyclic LFSR is obtained as shown in Figure 2.

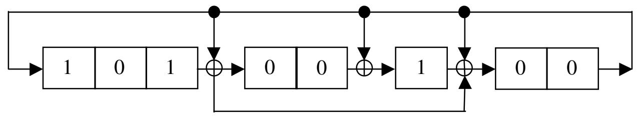


Figure 8: The (8,5) cyclic LFSR shortened from (15,5) cyclic LFSR

5.9

According to the question the cyclic LFSR (15,5) can be built by $g(x) = (1+x)(1+x+x^4) = 1+x^2+x^4+x^5, h(x) = (1+x^{15})/g(x) = 1+x^2+x^5+x^6+x^8+x^9+x^{10}$. Then, a 5-by-15 matrix H can be generated by $h(x)$:

$$H=\begin{bmatrix} x0 & x1 & x2 & x3 & x4 & x5 & x6 & x7 & x8 & x9 & x10 & x11 & x12 & x13 & x14 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$$

and matrix H' can be generated by applying elementary row manipulation on H such that the first k columns of the resulting matrix H' form a 5-by-5 identity matrix:

$$H' = \begin{bmatrix} x0 & x1 & x2 & x3 & x4 & x5 & x6 & x7 & x8 & x9 & x10 & x11 & x12 & x13 & x14 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$$

In matrix *H'*, there are 8 zeros in the first row <x1, x2, x3, x4, x6, x8, x9, x12>. It means that they are linear independent with x0. Then, a 4-by-14 matrix *H''* can be formed by simply removing the first row and first column in matrix *H'*:

$$H'' = \begin{bmatrix} x1 & x2 & x3 & x4 & x5 & x6 & x7 & x8 & x9 & x10 & x11 & x12 & x13 & x14 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$$

In order to avoid the loss of linear independence, matrix *M* can be formed by selecting *<x1, x2, x3, x4, x6, x8, x9, x12>* :

$$M = \begin{bmatrix} x1 & x2 & x3 & x4 & x6 & x8 & x9 & x12 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \end{bmatrix}$$

Build a (14,4) LFSR by using $f(x)=h(x)p(x)$, where $p(x)$ is the primitive polynomial of degree 4. Then, by using $f(x)=h(x)p(x) = (1+x^2+x^5+x^6+x^8+x^9+x^{10})(1+x+x4) = 1+x^2+x^3+x^4+x^5+x^6+x^7+x^8+x^{10}+x^{11}+x^{12}+x^{13}+x^{14}$, an (8,4) LFSR can be obtained by choosing the 8 stages of *<x1,x2,x3,x4,x6,x8,x9,x12>* of the (14,4) LFSR. Let $V_i(t)$ and $V_i(t+1)$ denote the present state and next states of stage V_i of the (14,4) LFSR respectively. Only those discontinuities in the eight columns must be replicated in order to preserve the shortened cyclic code properly. These discontinuities are the present state of V_{14}, and the next state of V_6, V_8 and V_{12}. They can be obtained from the (14,4) LFSR with $V_{14}(t) = V_8(t)+V_{12}(t)$; $V_6(t+1) = V_4(t)+V_8(t)$; $V_8(t+1) = V_2(t)+V_6(t)$; $V_{12}(t+1) = V_2(t)+V_9(t)$. Fig. 9(a) shows a (14,4) LFSR with $S_0(x) = h(x) =$ <10100110111000>, and Fig. 9(b) shows an (8,4) LFSR obtained from the (14,4) LFSR.

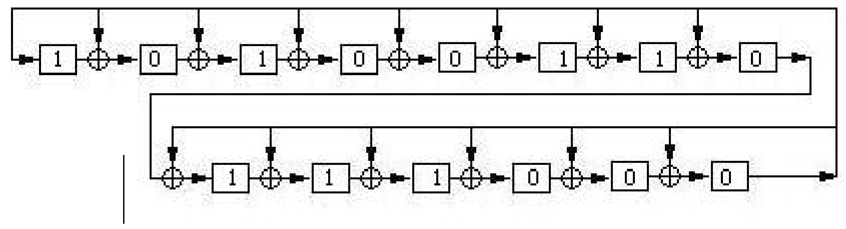


Figure 9(a): A (14,4) LFSR

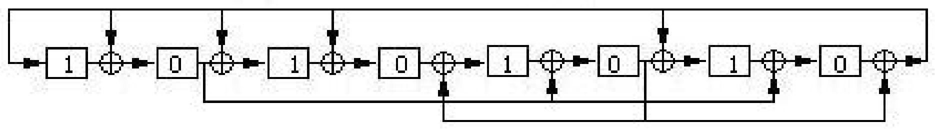
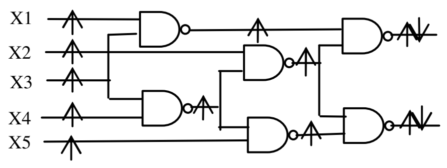


Figure 9(b): An (8,4) LFSR

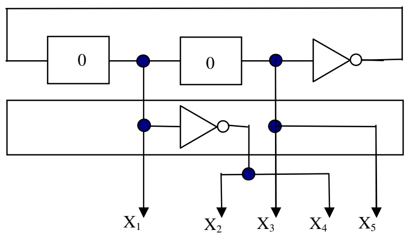
5.10

All collapsed single stuck-at faults in Figure 5.25(a) have been shown in the following figure:

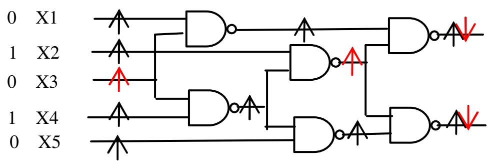


(a) An (n, w) = (5, 4) CUT

Pattern Generate in the structure of 5.25(b)

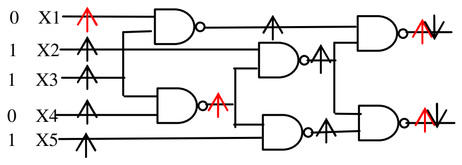


Faults detected by test pattern $(X_1,X_3) = \{00\}$ given below are marked by red color:



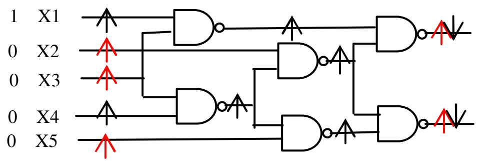
(b) An (n, w) = (5, 4) CUT

Faults detected by test pattern $(X_1,X_3) = \{01\}$ given below are marked by red color:



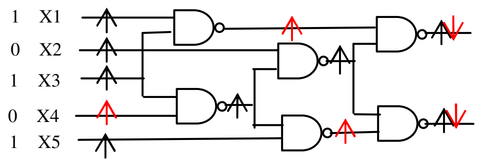
(c) An (n, w) = (5, 4) CUT

Faults detected by test pattern $(X_1,X_3) = \{10\}$ given below are marked by red color:



(d) An (n, w) = (5, 4) CUT

Faults detected by test pattern $(X_1,X_3) = \{11\}$ given below are marked by red color:



(e) An (n, w) = (5, 4) CUT

5.11

Given $R_0 = \{01101111\}$ and $R_1 = \{00110001\}$:

When using ones count testing, we obtain $OC(R_0) = 6$ and $OC(R_1) = 3$. Because $OC(R_0) \neq OC(R_1)$, this fault can be detected. The aliasing probability is:

$$P_{OC}(6) = (C(8, 6) - 1) / (2^8 - 1) = 27 / 255 = 0.11$$

When using transition count testing, we obtain $TC(R_0) = 3$ and $TC(R_1) = 3$. Because $TC(R_0) = TC(R_1)$, this fault cannot be detected. The aliasing probability is:

$$P_{TC}(3) = (2C(7, 3) - 1) / (2^8 - 1) = 69 / 255 = 0.27$$

5.12

Given $f(x) = 1 + x + x^4$ and $M = \{10011011\}$, we obtain the fault-free signature $R = \{1011\}$. For the faulty sequence $M' = \{11111111\}$ or $M'(x) = 1 + x + x^2 + x^3 + x^4 + x^5 + x^6 + x^7$, by polynomial division $M'(x) = q'(x)f(x) + r'(x)$, we can obtain $q'(x) = x + x^2 + x^3$, and $r'(x) = 1 + x^2 + x^3$. Thus, the faulty signature R' = {1011}. Since R' = R, the fault is undetected.

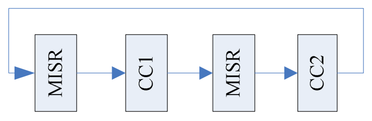
Another solution can be deduced by using error sequence E where $E = M + M' = \{01100100\}$ or $E(x) = x + x^2 + x^5$. By polynomial division, we obtain $E(x) = xf(x)$. Since $f(x)$ divides $E(x)$, this fault is undetected.

5.13

Given $M_0' = \{00010\}$, $M_1' = \{00010\}$, $M_2' = \{11100\}$, and $M_3' = \{10000\}$. By $M'(x) = M_0'(x) + xM_1'(x) + x^2M_2'(x) + x^3M_3'(x)$, we have $M' = \{00110000\}$ or $M'(x) = x^2 + x^3$. The faulty signature $R' = \{0011\}$. For $M = \{10011011\}$ and fault-free signature $R = \{1011\}$, because $R' \neq R$, this fault is detected.

5.14

1. To test a pipelined-oriented circuit using BILBOs



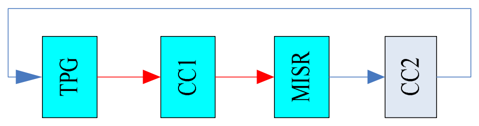
BILBO1 BILBO2

As one can see, using BILBOs, the signature data from the previous module must be used as test patterns for the next module, since the test generation and signature analysis modes cannot be separated. If n vectors are used to test the circuit, the test time required would be n clock cycles. In this case, a detailed fault simulation is required to achieve 100% single-stuck fault coverage.

2. To test a pipelined-oriented circuit using MBILBOs

Because test generation and signature analysis are separated from each other in time but not in space, *CC1* and *CC2* can be tested alternatively.

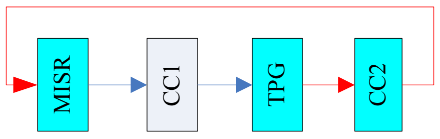
Step 1:



MBILBO1 MBILBO2

Now, suppose *CC1* is to be tested. MBILBO1 will be reconfigured as a PRPG by setting B1 = 1, B2 = 0, and B3 = 1. Meanwhile, MBILBO2 will be reconfigured as a MISR by setting B1 = 1, B2 = 0, and B3 = 0. During this time, *CC2* is not tested.

Step 2:

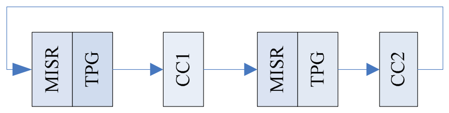


MBILBO1 MBILBO2

Now, suppose *CC2* is to be tested. By reconfiguring MBILBO1 and MBILBO2 as MISR and TPG, respectively, *CC2* can be tested exhaustively or pseudo-exhaustively. During this time, *CC1* is not tested. According to the above analysis, we can conclude that if *n* vectors can be used to test each module exhaustively or pseudo-exhaustively, then total test time may become 2*n* clock cycles. In this case, a detailed fault simulation is not required.

3. To test a pipelined-oriented circuit using CBILBOs

Because test generation and signature analysis are separated from each other both in time and space with using two storage elements in each CBILBO, it is possible to test both *CC1* and *CC2* simultaneously. In addition, no fault simulation will be required to achieve 100% single-stuck fault coverage if exhaustive or pseudo-exhaustive patterns are used. Therefore, *n* clock cycles are only needed.



CBILBO1 CBILBO2

The following table shows the advantages and disadvantages of using the BILBO, MBILBO, and CBILBO approaches:

	Hard cost	Test time	Fault coverage
BILBO	Low	Short	Low
MBILBO	Medium	Long	High
CBILBO	High	Short	High

5.15

(1) When both shift and capture at 200MHz:

STUMPS:
Test time = [100000*1000+100000]*(1/200M) = 0.5005s

BILBO:
Test time = 100000*(1/200M)+2*1000*100*(1/200M) = 1.5ms

(2) When shift at 20MHz and capture at 200MHz

STUMPS:
Test time = 100000*1000*(1/20M)+100000*(1/200M) = 5.005s

(3) The main reason why the STUMPS-based architecture is gaining more popularity than the BILBO-based architecture is because STUMPS can be implemented with the conventional scan architecture, without adding additional logic to the scan chains. Also, the hardware cost is low.

5.16

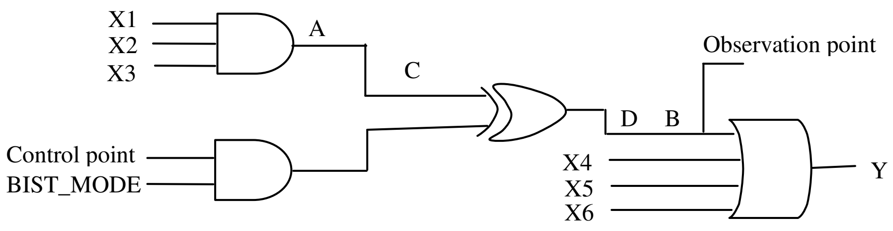
(1) Before inserting test point:
For the stuck-at-0 fault present at X_3, all inputs of the AND gate, X_1, X_2, X_3, X_4, X_5 and X_6, need to be 1. We need a 1 at X_3 to activate the stuck-at-0 fault and a 1 for each other input to propagate the faults. So the detection probability of a stuck-at-0 fault at X_3 is 1/64 (= 1/2 * 1/2 * 1/2 * 1/2 * 1/2 * 1/2).

For the stuck-at-1 fault present at X_6, input X_6 needs to be 0 to activate the fault, and all other inputs of the AND gate, X_1, X_2, X_3, X_4, and X_5, must be set to 1 to propagate the fault. So the detection probability of stuck-at-one fault at X_6 is 1/64 (= 1/2 * 1/2 * 1/2 * 1/2 * 1/2 * 1/2).

(2) After inserting test point:
For the stuck-at-0 fault present at X_3, input X_3 needs to be 1 to activate the fault. Inputs X_1 and X_2 need to be 1 and the control point needs to be 0 to propagate the fault. So the detection probability of the stuck-at-0 fault at X_3 is 1/16 (= 1/2 * 1/2 * 1/2 * 1/2).

For the stuck-at-1 fault present at X_6, input X_6 needs to be 0 to activate the fault. Assume that another input of the OR gate (with control point) is A, and the OR gate output is B. To propagate the stuck-at-1 at X_6, Inputs X_4, X_5, and B must to set to 1. Thus, either A or the control point need to be 1. The probability of 1 at B is 9/16 (=1- 7/8*1/2). So the detection probability of the stuck-at-one fault at X_6 is 9/128 (= 9/16 * 1/2 * 1/2 * 1/2).

5.17



The test point added is presented in figure above. The 0/1 probability of nodes is presented below:

	X_1	X_2	X_3	A	C	Control point	D	B	X_4	X_5	X_6	Y
P1	1/2	1/2	1/2	1/8	1/8	1/2	1/2	1/2	1/2	1/2	1/2	15/16
P0	1/2	1/2	1/2	7/8	7/8	1/2	1/2	1/2	1/2	1/2	1/2	1/16

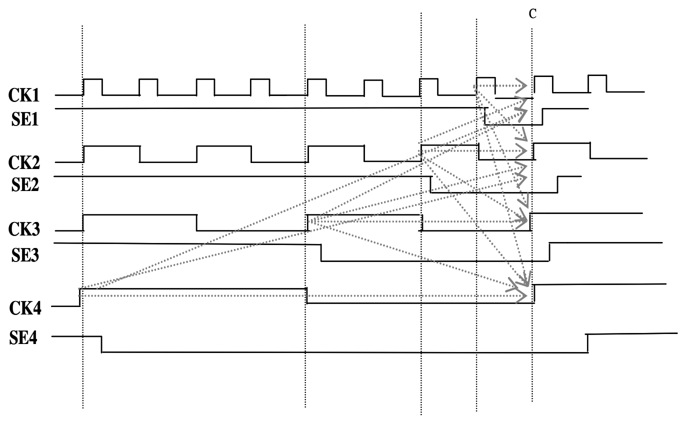
The detection probability of each fault is:
Stuck-at-1 at X_1, X_2 and X_3 is 1/16 (=1/8 *1/2)
Stuck-at-0 at X_1, X_2 and X_3 is 1/16 (=1/8*1/2)
Stuck-at-1 at A and C is 7/16 (=7/8*1/2)

Stuck-at-0 at A and C is 1/16 (=1/8*1/2)
Stuck-at-1 at D is 1/2
Stuck-at-0 at D is 1/2
Stuck-at-1 at B is 1/16 (=1/2*1/2*1/2*1/2)
Stuck-at-0 at B is 1/16 (=1/2*1/2*1/2*1/2)
Stuck-at-1 at X_4, X_5 and X_6 is 1/16 (=1/2*1/2*1/2*1/2)
Stuck-at-0 at X_4, X_5 and X_6 is 1/16 (=1/2*1/2*1/2*1/2)
Stuck-at-1 at Y is 1/16
Stuck-at-0 at Y is 15/16

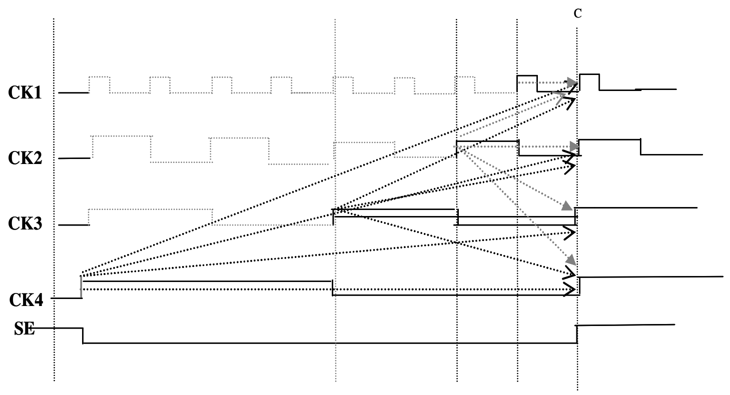
The detection probabilities are all greater than or equal to 1/16.

5.18

Aligned skewed-load in capture:

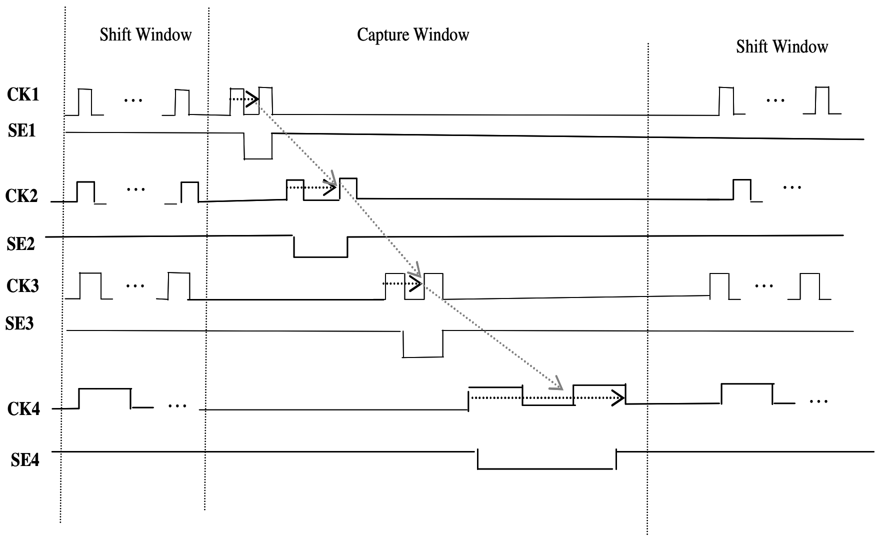


Aligned double-capture - I:

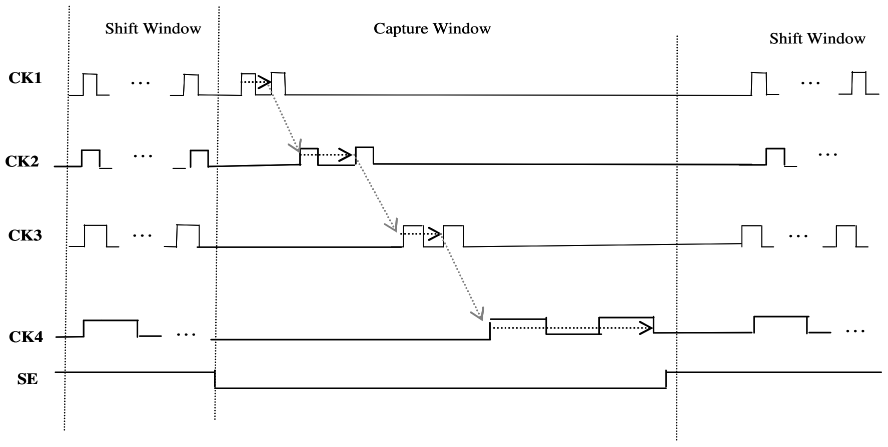


5.19

Staggered skewed-load:

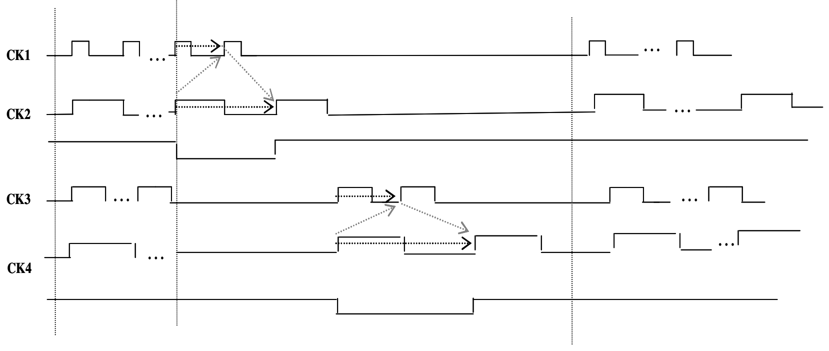


Staggered double-capture:



5.20

Hybrid double-capture:


CK1
CK2
CK3
CK4