State Reduction (1)

Goal: reduce the number of states while keeping the external input-output requirements unchanged.

State reduction example:
a: input 0 → output 0, circuit stays in same state
a: input 1 → output 0, circuit goes to state *b*

b: input 0 → output 0, circuit goes to state *c*

c: input 1 → output 0, circuit goes to state *d*

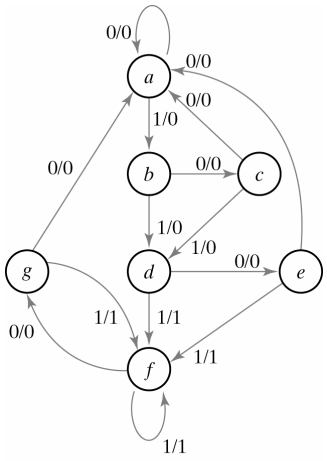

Fig. 5-22 State Diagram

1

State Reduction (2)

1. Construct the state table from the state diagram:

Present State	Next State		Output	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
a	*a*	*b*	0	0
b	*c*	*d*	0	0
c	*a*	*d*	0	0
d	*e*	*f*	0	1
e	*a*	*f*	0	1
f	*g*	*f*	0	1
g	*a*	*f*	0	1

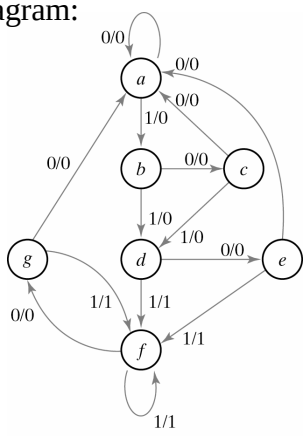

Fig. 5-22 State Diagram

2

State Reduction (3)

State Reduction Algorithm: Two states are equivalent if, for each member of the set inputs, they give the same output and send the circuit to the same state or equivalent state.

Present State	Next State		Output	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
a	*a*	*b*	0	0
b	*c*	*d*	0	0
c	*a*	*d*	0	0
d	*e*	*f*	0	1
e	*a*	*f*	0	1
f	*e* → *g*	*f*	0	1
~~*g*~~	~~*a*~~	~~*f*~~	~~0~~	~~1~~

equivalent states

Row with present state *g* is removed, and state *g* is replaced by state *e* each time it occurs.

3

State Reduction (4)

State Reduction Algorithm: Two states are equivalent if, for each member of the set inputs, they give the same output and send the circuit to the same state or equivalent state.

Present State	Next State		Output	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
a	*a*	*b*	0	0
b	*c*	*d*	0	0
c	*a*	*d*	0	0
d	*e*	*f* (*d*)	0	1
e	*a*	*f* (*d*)	0	1
~~*f*~~	~~*e*~~	~~*f*~~	~~0~~	~~1~~

equivalent states

Row with present state *f* is removed, and state *f* is replaced by state *d* each time it occurs.

4

State Reduction (final)

Present State	Next State		Output	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
a	*a*	*b*	0	0
b	*c*	*d*	0	0
c	*a*	*d*	0	0
d	*e*	*d*	0	1
e	*a*	*d*	0	1

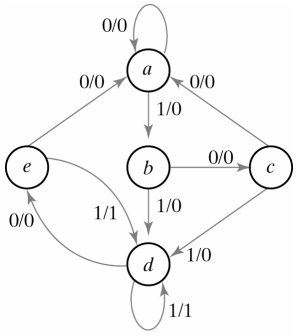

Fig. 5-23 Reduced State Diagram

5

State Coded Binary Assignment

Present State	Next State		Output	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
a 000	*a* 000	*b* 001	0	0
b 001	*c* 010	*d* 011	0	0
c 010	*a* 000	*d* 011	0	0
d 011	*e* 100	*d* 011	0	1
e 100	*a* 000	*d* 011	0	1

Reduced State Table with Binary Assignment

6

Sequential Circuits: Design Procedure

Recommended Design Steps

• Derive the state diagram from the word description and the specifications of the desired operation.

• Reduce the number of states if necessary.

• Assign binary values to the states.

• Obtain the binary coded-state table.

• Choose the type of flip-flops to be used.

• Derive the simplified flip-flop input and output equations.

• Draw the logic diagram.

7

Sequence Detector Sequential Circuit (1)

Design a circuit that detects three or more consecutive 1's in a string of bits using *D* Flip-Flops.

• Start with state S_0
• If the input is 0 → circuit stays in the same state
• If the next input is 1 → circuit goes to S_1 to indicate that 1 was detected
• If the next input is 1 → circuit goes to S_2 to indicate that the arrival of two consecutive 1's.
• But if the input were 0 → circuit goes back to S_0.
• The third consecutive 1 sends the circuit to S_3.
• If more 1 are detected → circuit stays in S_3.

8

Sequence Detector Sequential Circuit (2)

- State table is derived directly from the state diagram.
- We choose 2 *D* Flip-Flops (outputs *A*, *B*)
- There is one input *x* and one output *y*

Present State		Input	Next State		Output
A	*B*	*x*	*A*	*B*	*y*
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	1	1	1	1

D flip-flop state Equations:

$A(t+1) = D_A(A,B,x) = \Sigma(3,5,7)$

$A(t+1) = D_B(A,B,x) = \Sigma(1,5,7)$

$y(A,B,x) = \Sigma(6,7)$

This state table is the result of Moore implementation:
→output depends on the present state only.
We can, however, implement a Mealy machine:
→output depends on the present state and the input.

9

Sequence Detector Sequential Circuit (final)

- Obtain the simplified functions from the K-Maps:

$A(t+1) = D_A(A,B,x) = \Sigma(3,5,7)$

$A(t+1) = D_B(A,B,x) = \Sigma(1,5,7)$

$y(A,B,x) = \Sigma(6,7)$

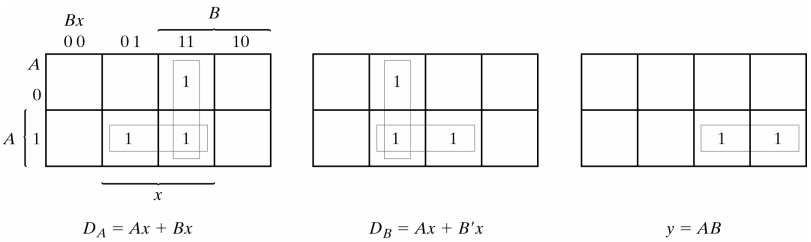

Fig. 5-25 Maps for Sequence Detector

10

Design using *D* flip flops

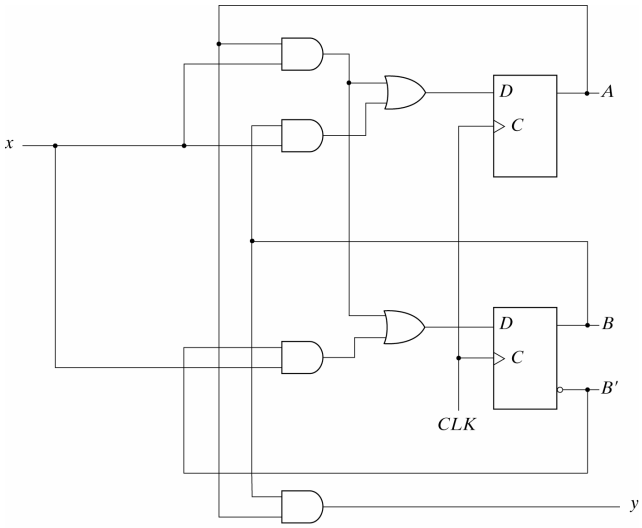

Fig. 5-26 Logic Diagram of Sequence Detector

11

Design using *JK* flip-flops (1)

In order to determine the input equations for the *JK* flip-flops, it is necessary to derive a functional relationship between the state table and the input equations.

Present State		**Input**	**Next State**		**Flip-Flop inputs**			
A	*B*	*x*	*A*	*B*	J_A	K_A	J_B	K_B
0	0	0	0	0	0	X	0	X
0	0	1	0	1	0	X	1	X
0	1	0	1	0	1	X	X	1
0	1	1	0	1	0	X	X	0
1	0	0	1	0	X	0	0	X
1	0	1	1	1	X	0	1	X
1	1	0	1	1	X	0	X	0
1	1	1	0	0	X	1	X	1

Flip-Flop Excitation table:

$Q(t)$	$Q(t+1)$	*J*	*K*
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

$$Q(t+1) = JQ' + K'Q$$

12

Design using *JK* flip-flops (2)

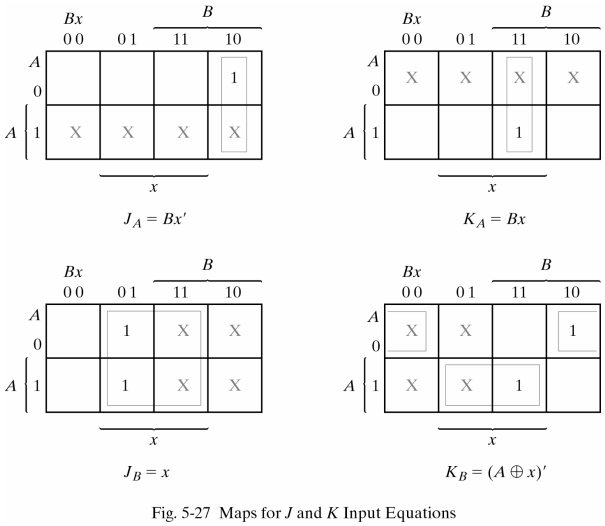

Fig. 5-27 Maps for *J* and *K* Input Equations

13

Design using *JK* flip-flops (final)

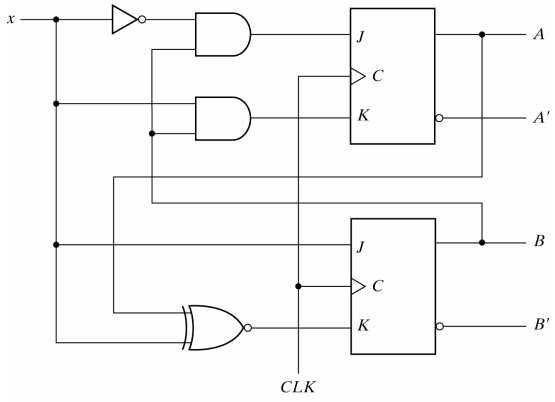

Fig. 5-28 Logic Diagram for Sequential Circuit with *JK* Flip-Flops

14