Lecture 14
Reduction of State Tables

- Elimination of redundant states

- Problem: input X and output Z. If input forms 0101 or 1001, then Z = 1. The network resets after every four inputs.

X = 0101 0010 1001 0100

Z = 0001 0000 0001 0000

reset reset reset

Elimination of redundant states

- Designate each next state as a bit is received. We may have redundant states. 0101 or 1001

Table 15–1
State Table for Sequence Detector

Input Sequence	Present State	Next State $X=0$	Next State $X=1$	Present Output $X=0$	Present Output $X=1$
reset	A	B	C	0	0
0	B	D	E	0	0
1	C	F	G	0	0
00	D	H	I	0	0
01	E	J	K	0	0
10	F	L	M	0	0
11	G	N	P	0	0
000	H	A	A	0	0
001	I	A	A	0	0
010	J	A	A	0	1
011	K	A	A	0	0
100	L	A	A	0	1
101	M	A	A	0	0
110	N	A	A	0	0
111	P	A	A	0	0

Elimination of redundant states

- Find the equivalent states and eliminate those that have the same next state and outputs.
  - (I, K, M, N, P => H, keep H)

Table 15–2
State Table for Sequence Detector

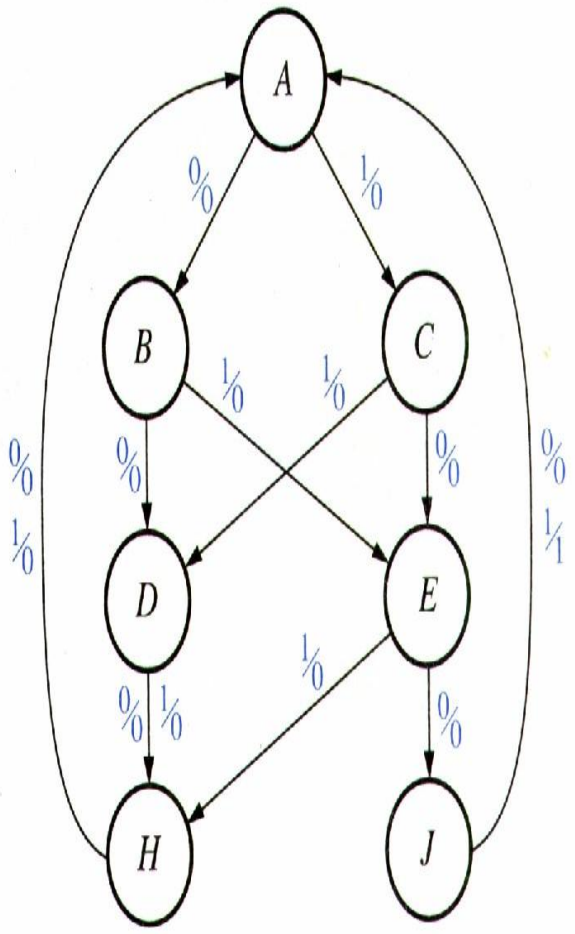
Present State	Next State $X=0$	Next State $X=1$	Present Output $X=0$	Present Output $X=1$
A	B	C	0	0
B	D	E	0	0
C	~~F~~ E	~~G~~ D	0	0
D	H	~~I~~ H	0	0
E	J	~~K~~ H	0	0
~~F~~	~~L~~ J	~~M~~ H	~~0~~	~~0~~
~~G~~	~~N~~ H	~~P~~ H	~~0~~	~~0~~
H	A	A	0	0
~~I~~	~~A~~	~~A~~	~~0~~	~~0~~
J	A	A	0	1
~~K~~	~~A~~	~~A~~	~~0~~	~~0~~
~~L~~	~~A~~	~~A~~	~~0~~	~~1~~
~~M~~	~~A~~	~~A~~	~~0~~	~~0~~
~~N~~	~~A~~	~~A~~	~~0~~	~~0~~
~~P~~	~~A~~	~~A~~	~~0~~	~~0~~

Elimination of redundant states (cont.)

- Row matching: sufficient only to network reset to the starting state after receiving a fixed number of inputs.

Present State	Next State		Output	
	$X = 0$	$X = 1$	$X = 0$	$X = 1$
A	B	C	0	0
B	D	E	0	0
C	E	D	0	0
D	H	H	0	0
E	J	H	0	0
H	A	A	0	0
J	A	A	0	1

(a)



(b)

Equivalent States ≡

- Equivalent: two states are equivalent if there is no way of telling them apart from observation of network inputs and outputs.
  - N_1: started in state p
  - N_2: started in state q.
  - For every possible input sequence X, the output sequences are the same (Z_1 and Z_2). Then we say that p is equivalent to q.

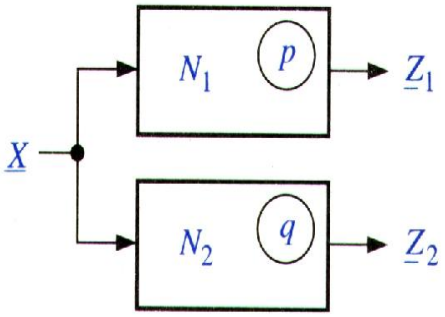


Figure 15–2

Equivalent States

- State p in N_1.
- State q in N_2.
  - Output sequence is a function of the initial state and input sequence. Then,
    - We have $\underline{Z_1} = \lambda_1(p,\underline{X})$ and $\underline{Z_2} = \lambda_2(q, \underline{X})$.
    - State p in N_1 is *equivalent* to state q in N_2 iff $\underline{Z_1} = \underline{Z_2}$ for every possible input sequence X.
    - $\underline{X}$ = X1, X2, X3 ….
    - $\underline{Z_1}$ = Z1, Z2, Z3…..

Theorem

- Two states p and q of a sequential network are equivalent iff for every single input X, the outputs are the same and the next states are equivalent, that is,
- $\lambda(p,X) = \lambda(q,X)$ (output) and
- $\delta(p, X) \equiv \delta(q, X)$ (next state)

Where $\lambda(p,X)$ is the output given the present state p and input X and $\delta(p, X)$ is the next state given the present state p and input X.

Application of the Theorem

- Are S_0 and S_2 equivalent?
  - Present output is the same for S_0 and S_2.
  - $S_0 \equiv S_2$ iff $S_3 \equiv S_3$, $S_2 \equiv S_0$, $S_1 \equiv S_1$ and $S_0 \equiv S_1$. But $S_0 \equiv S_1$ (due to outputs.)
  - The answer is No.

TABLE 13-4
A State Table with Multiple Inputs and Outputs

Present State	Next State X_1X_2 = 00	01	10	11	Present Output (Z_1Z_2) X_1X_2 = 00	01	10	11
S_0	S_3	S_2	S_1	S_0	00	10	11	01
S_1	S_0	S_1	S_2	S_3	10	10	11	11
S_2	S_3	S_0	S_1	S_1	00	10	11	01
S_3	S_2	S_2	S_1	S_0	00	00	01	01

Implication Table for State Equivalence

- Use an implication table (pair chart) to check each pair of states for possible equivalence.

Table 15–3

Present State	Next State $X = 0$	1	Present Output
a	d	c	0
b	f	h	0
c	e	d	1
d	a	e	0
e	c	a	1
f	f	b	1
g	b	h	0
h	c	g	1

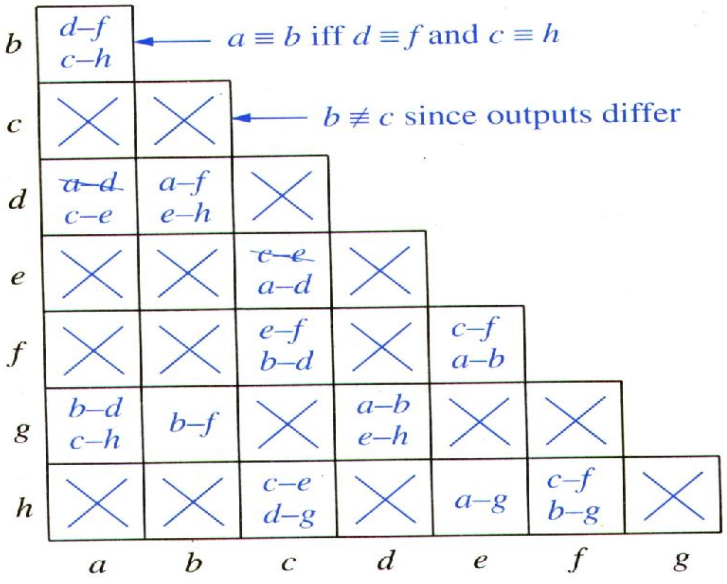


Figure 15–3
Implication Chart for Table 15–3

Implication Table for State Equivalence (cont.)

- $a \equiv b$ iff $d \equiv f$ and $c \equiv h$. This "implied pair" is placed in a-b square. Self-implied pairs are redundant. So eliminate them.

Table 15–3

Present State	Next State $X = 0$	1	Present Output
a	d	c	0
b	f	h	0
c	e	d	1
d	a	e	0
e	c	a	1
f	f	b	1
g	b	h	0
h	c	g	1

a ≡ d iff
a ≡ d and
c ≡ e.

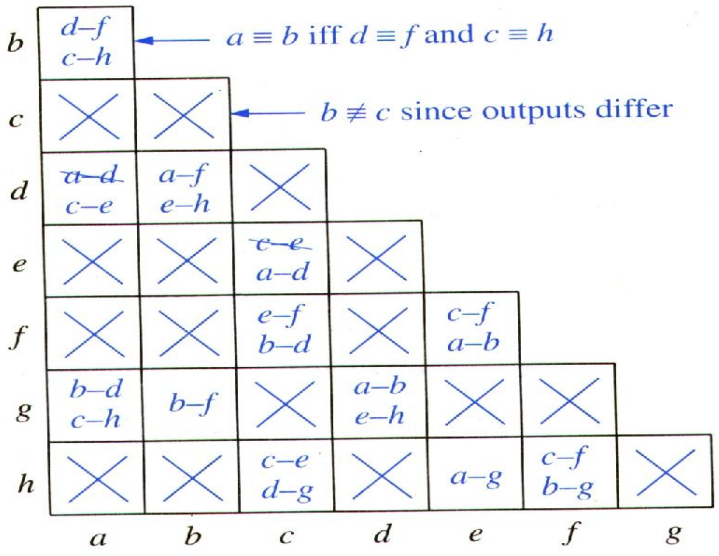
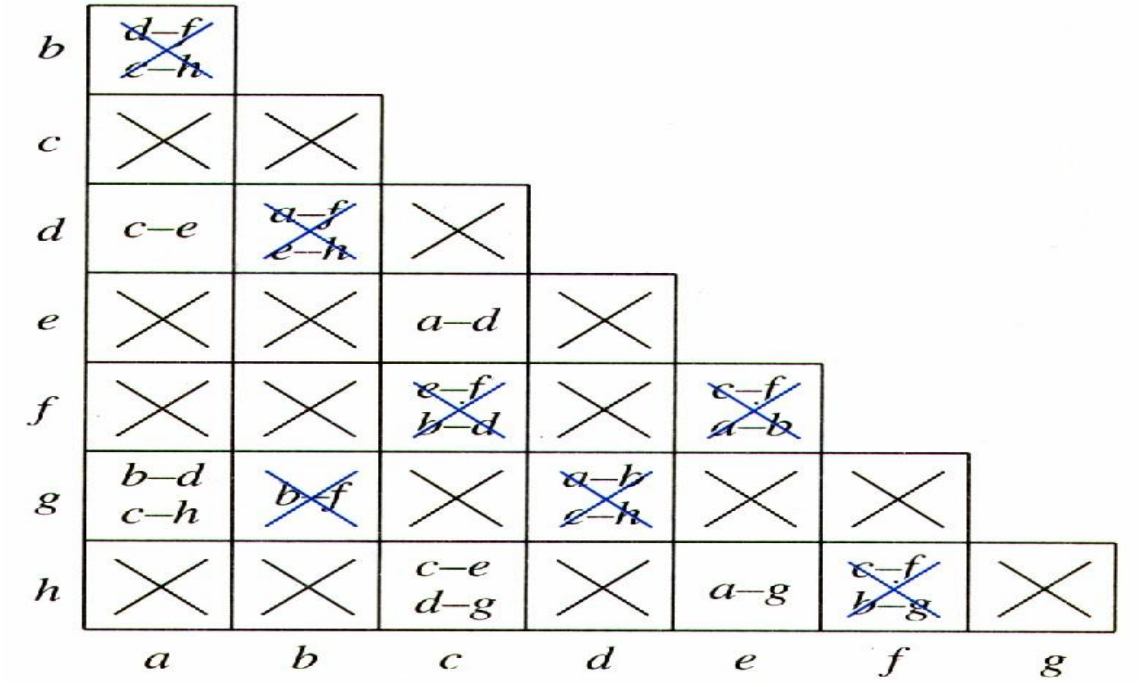


Figure 15–3
Implication Chart for Table 15–3

Implication Table for State Equivalence (cont.)

- For *a*-*b* square, we need *d*≡*f* and *c*≡*h* for *a*≡*b*. But in the *d*-*f* square we see a X which means that *d* is not equivalent to *f*. So *a* is not equivalent to *b*. We place a X in the *a*-*b* square.
- For *a*-*d* square, square *c*-*e* does not contain a X. So at this point, we can not determine whether *a* ≡*d*.
- Do the first pass column by column.



Implication Table for State Equivalence (cont.)

- After the first pass, we do the second pass from *a* column again. Then, we do the third pass and find no new X's are added. Then the process terminates.
- The result is $a \equiv d$ and $c \equiv e$.
- So we replace *d* with *a* and *e* with *c* and eliminate row *d* and *e*.

Table 15–4

Present State	Next State X = 0	Next State 1	Output
a	*a*	*c*	0
b	*f*	*h*	0
c	*c*	*a*	1
f	*f*	*b*	1
g	*b*	*h*	0
h	*c*	*g*	1

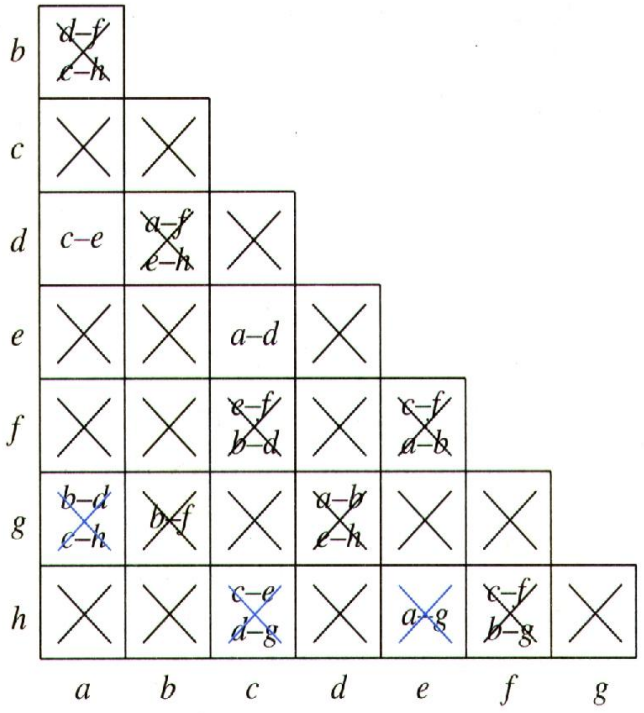


Figure 15–5
Implication Chart After Second Pass

Equivalent Sequential Networks

- Sequential network N_1 is equivalent to sequential network N_2 if for each state p in N_1 there is a state q in N_2 such that $p \equiv q$, and conversely, for each state s in N_2 there is a state t in N_1 such that $s \equiv t$.

- That is, for N_1 and N_2, the output sequences are the same for the same input sequences.

Equivalent Sequential Networks (cont.)

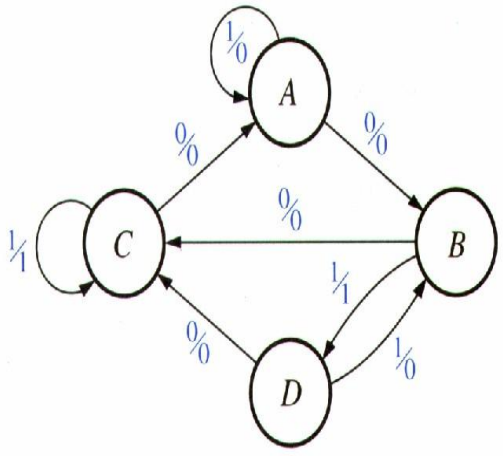
- $A \equiv S_2$? (output is the same). IF $A \equiv S_2$, then $B \equiv S_0$. This further implies $D \equiv S_1$ and $C \equiv S_3$. This is true from the table. (Next states are equivalent and outputs are the same.)

N_1

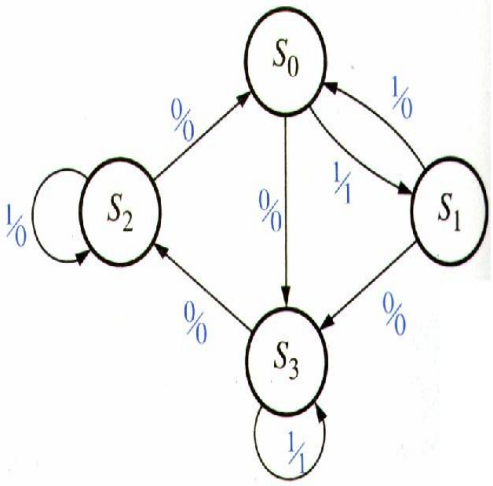
	$X=0$	1	$X=0$	1
A	B	A	0	0
B	C	D	0	1
C	A	C	0	1
D	C	B	0	0

N_2

	$X=0$	1	$X=0$	1
S_0	S_3	S_1	0	1
S_1	S_3	S_0	0	0
S_2	S_0	S_2	0	0
S_3	S_2	S_3	0	1



(a)



(b)

Equivalent Sequential Networks (cont.)

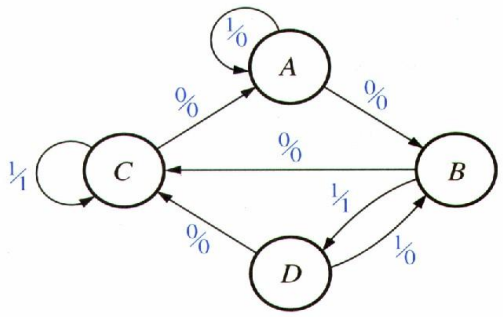
- Using implication table to determine network equivalence, place X in the square for different outputs for the compared pair.

N_1

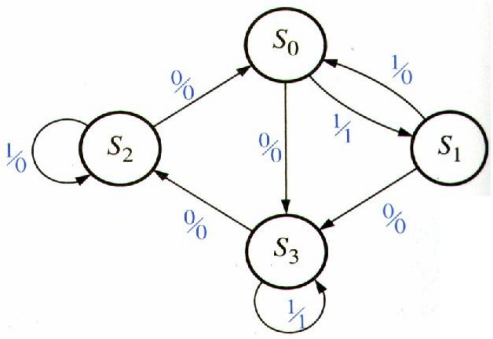
	$X=0$	1	$X=0$	1
A	B	A	0	0
B	C	D	0	1
C	A	C	0	1
D	C	B	0	0

N_2

	$X=0$	1	$X=0$	1
S_0	S_3	S_1	0	1
S_1	S_3	S_0	0	0
S_2	S_0	S_2	0	0
S_3	S_2	S_3	0	1



(a)



(b)

$A \equiv S_2$, $B \equiv S_0$, etc

	A	B	C	D
S_0	X	$C-S_3$ $D-S_1$	$A-S_3$ $C-S_1$	X
S_1	$B-S_3$ $A-S_0$	X	X	$C-S_3$ $B-S_0$
S_2	$B-S_0$ $A-S_2$	X	X	$C-S_0$ $B-S_2$
S_3	X	$C-S_2$ $D-S_3$	$A-S_2$ $C-S_3$	X

(a)

	A	B	C	D
S_0	X	$C-S_3$ $D-S_1$	~~$A-S_3$~~ ~~$C-S_1$~~	X
S_1	~~$B-S_3$~~ ~~$A-S_0$~~	X	X	$C-S_3$ $B-S_0$
S_2	$B-S_0$ $A-S_2$	X	X	~~$C-S_0$~~ ~~$B-S_2$~~
S_3	X	~~$C-S_2$~~ ~~$D-S_3$~~	$A-S_2$ $C-S_3$	X

(b)

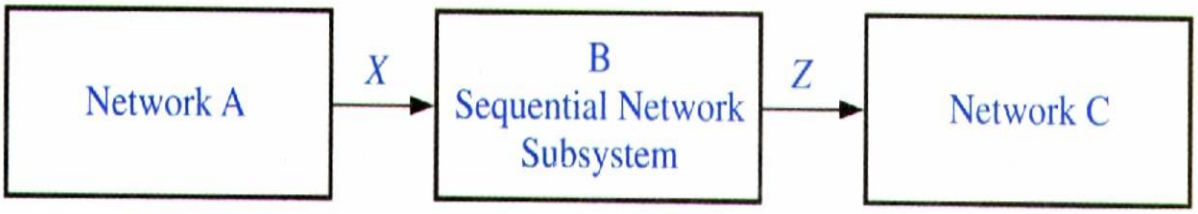
Figure 15–7
Implication Tables for Determining Network Equivalence

Incompletely Specified State Table

- Problem statement
  - X only has X = 100 and X = 110 sequence
  - Z for X = 100 is 0.
  - Z for X = 110 is 1.
  - Come up each possible state.

t_0 t_1 t_2	t_0 t_1 t_2
$X = 1\ 0\ 0$	$Z = -\ -\ 0$
$1\ 1\ 0$	$-\ -\ 1$

(– is a don't care output)



Incompletely Specified State Table(cont.)

- Unspecified states or outputs.
  - For the next state of S_0, X = 0 does not occur. (S_0 : reset)
  - Entering t_1, we have S_2 or S_3 two states depending on X.
  - Fill in don't cares for row matching.

t_0 t_1 t_2	t_0 t_1 t_2
$X = 1\ 0\ 0$	$Z = -\ -\ 0$
$1\ 1\ 0$	$-\ -\ 1$

(– is a don't care output)

Table 15–5

Incompletely Specified State Table

	$X=0$	1	0	1
S_0	–	S_1	–	–
S_1	S_2	S_3	–	–
S_2	S_0	–	0	–
S_3	S_0	–	1	–

(a)

	$X=0$	1	0	1
S_0	(S_0)	S_1	(0)	–
S_1	S_2 S_0	S_3	(1)	–
S_2	S_0	(S_1)	0	–
S_3	S_0	(S_3)	1	–

(b)

$S_0 \equiv S_2, S_1 \equiv S_3$

	$X=0$	1	0	1
S_0	S_0	S_1	0	–
S_1	S_0	S_1	1	–

(c)

Derivation of Flip-Flop Input Equations

- 3 FFs for 7 states

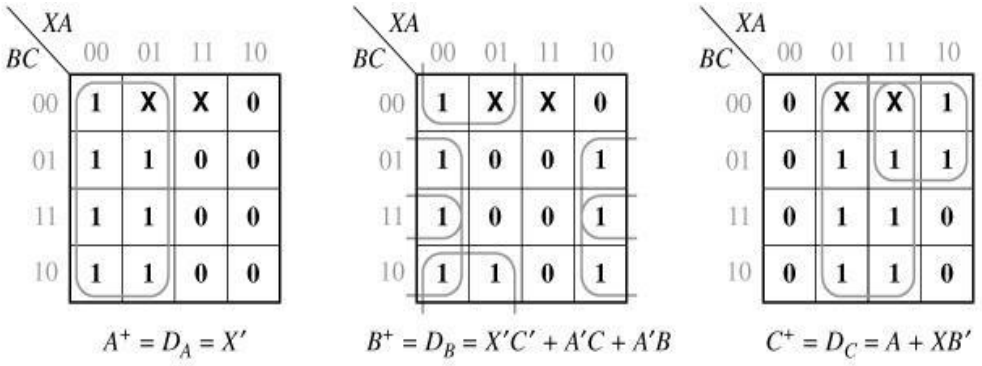
TABLE 15-6

(a) State table

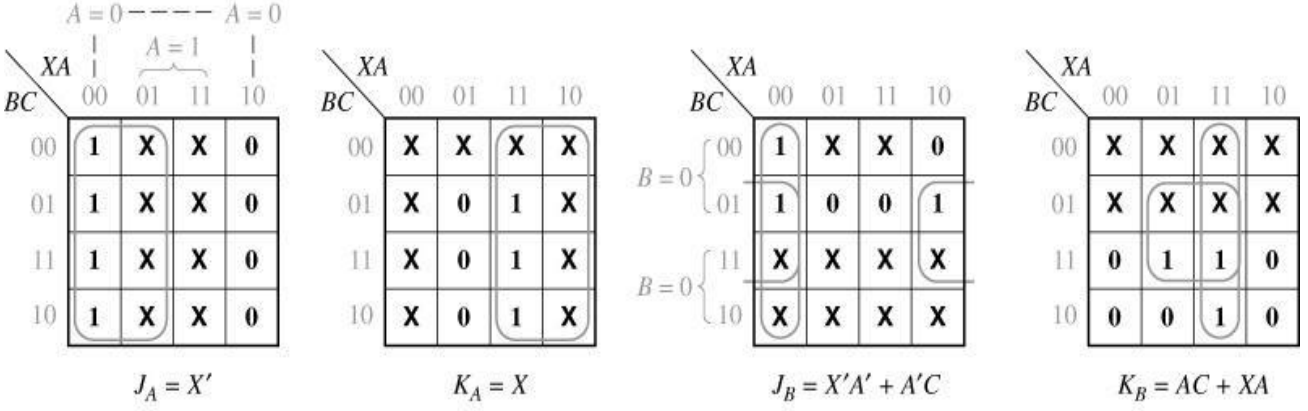
	$X = 0$	1	0	1
S_0	S_1	S_2	0	0
S_1	S_3	S_2	0	0
S_2	S_1	S_4	0	0
S_3	S_5	S_2	0	0
S_4	S_1	S_6	0	0
S_5	S_5	S_2	1	0
S_6	S_1	S_6	0	1

(b) Transition table

	$A^+B^+C^+$		Z	
ABC	$X = 0$	1	0	1
000	110	001	0	0
110	111	001	0	0
001	110	011	0	0
111	101	001	0	0
011	110	010	0	0
101	101	001	1	0
010	110	010	0	1



(a) Derivation of D flip-flop input equations



(b) Derivation of J-K flip-flop input equations

Equivalent State Assignment

- State assignment for 3 states S_0, S_1, and S_2. (Table 14-, 101 detector)
  - We need 2 FFs. S_0 can have 00, 01, 10, or 11. In this way, there are 4x3x2x1 = 24 possible combinations to evaluate.
  - Assignment 1 and assignment 3 has the same cost. Since only the labeling of FF is changed. (change column = same cost)
  - Assignment 4 and 6 has row change. They will have different cost.

Table 15–8
State Assignments for 3-Row Tables

Different cost

AB	1	2	3	4	5	6	7	⋯	19	20	21	22	23	24
S_0	00	00	00	00	00	00	01	⋯	11	11	11	11	11	11
S_1	01	01	10	10	11	11	00		00	00	01	01	10	10
S_2	10	11	01	11	01	10	10		01	10	00	10	00	01

Same cost

Interchanging or Complementing State Assignment

- Assignment b: interchanging the column of a
- Assignment c: complementing the columns of a.
  - **For J-K FF, the cost is the same for all three assignments for any kind of logic gates.**
  - **For D FF, if AND and OR gates are available, then the cost is the same.**

Table 15–9

Assignments "a"	"b"	"c"	Present State	Next State X = 0	1	Output 0	1
00	00	11	S_1	S_1	S_3	0	0
01	10	10	S_2	S_2	S_1	0	1
10	01	01	S_3	S_2	S_3	1	0

Assignment "a"	*Assignment "b"*	*Assignment "c"*
$J_1 = XQ_2'$	$J_2 = XQ_1'$	$K_1 = XQ_2$
$K_1 = X'$	$K_2 = X'$	$J_1 = X'$
$J_2 = X'Q_1$	$J_1 = X'Q_2$	$K_2 = X'Q_1'$
$K_2 = X$	$K_1 = X$	$J_2 = X$
$Z = X'Q_1 + XQ_2$	$Z = X'Q_2 + XQ_1$	$Z = X'Q_1' + XQ_2'$
$D_1 = XQ_2'$	$D_2 = XQ_1'$	$D_1 = X' + Q_2'$
$D_2 = X'(Q_1 + Q_2)$	$D_1 = X'(Q_2 + Q_1)$	$D_2 = X + Q_1Q_2$

Minimum Cost Realization of State Assignment

- Nonequivalent assignment: by eliminating states that can be obtained by permuting or complementing columns.
  - We can complement one or more columns. So any state assignment can be converted to one in which the first state is assigned all 0's.
  - For symmetrical FFs, need only to try three assignments for minimum cost.
  - The number of distinct states increases rapidly as the number of states increases.

Table 15–10

Nonequivalent Assignments for 3 and 4 States

	3-State Assignments			4-State Assignments		
States	1	2	3	1	2	3
a	00	00	00	00	00	00
b	01	01	11	01	01	11
c	10	11	01	10	11	01
d	--	--	--	11	10	10

Guidelines for State Assignment

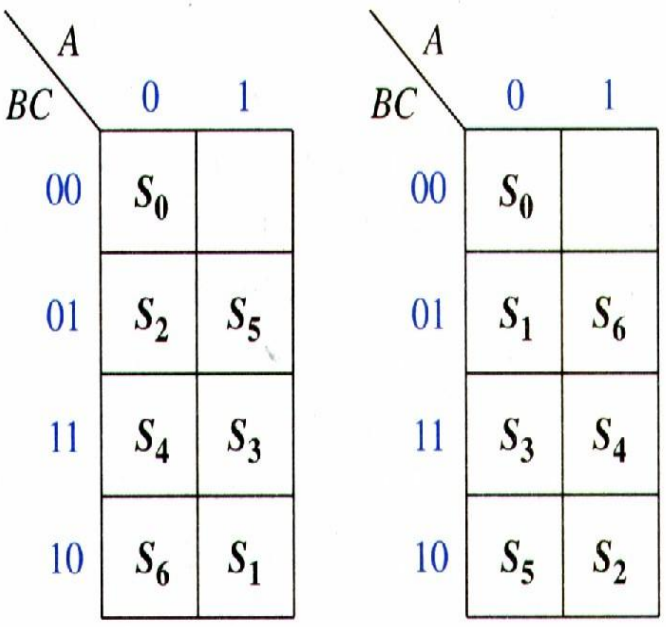
- Try to places 1's on the FF input maps in adjacent squares.
  - 1: States which have the same next state for a given input should be given adjacent assignment.
  - 2: States which are the next states of the same state should be given adjacent assignment.
  - 3: States which have the same output for a given input should be given adjacent assignment (for output simplification.)

Guidelines for State Assignment: Example

- G_1: (S_0, S_1,S_3, S_5) (S_3, S_5) (S_4,S_6) (S_0,S_2,S_4,S_6)→ S_1 as the next state.
- G_2: (S_1, S_2) (S_2, S_3) (S_1,S_4) (S_2,S_5)2x (S_1,S_6)2x
- Try to fulfill as many of these adjacency conditions as possible.
- G_1 preference > G_2 preference.

ABC		*X* = 0	1	0	1
000	S_0	S_1	S_2	0	0
110	S_1	S_3	S_2	0	0
001	S_2	S_1	S_4	0	0
111	S_3	S_5	S_2	0	0
011	S_4	S_1	S_6	0	0
101	S_5	S_5	S_2	1	0
010	S_6	S_1	S_6	0	1

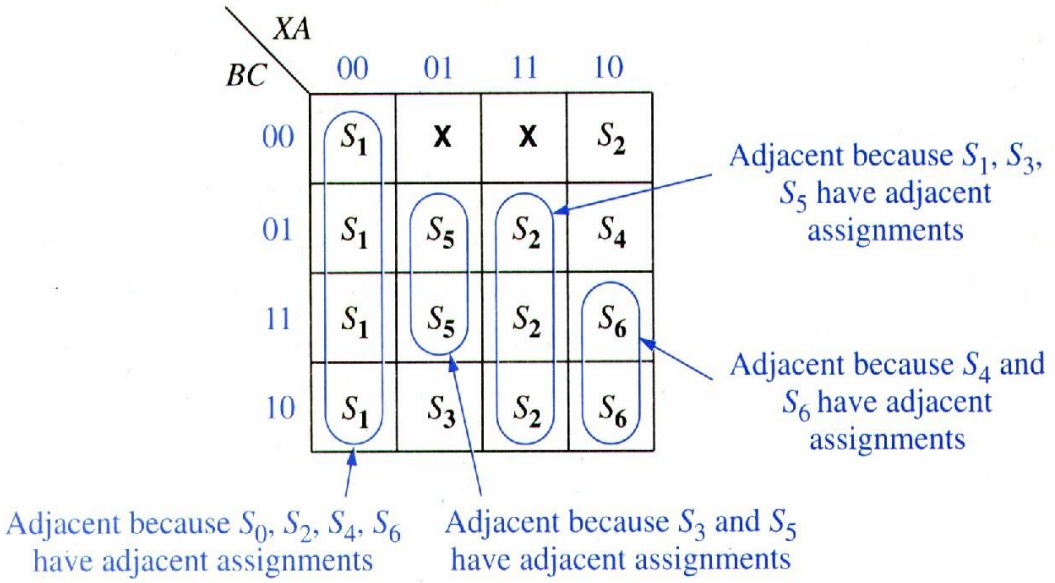
(a) state table



(b) assignment maps

Why is it a better assignment?

- Next state map shows that S_1 appears in four adjacent squares, and etc. Example: S_1 = 110

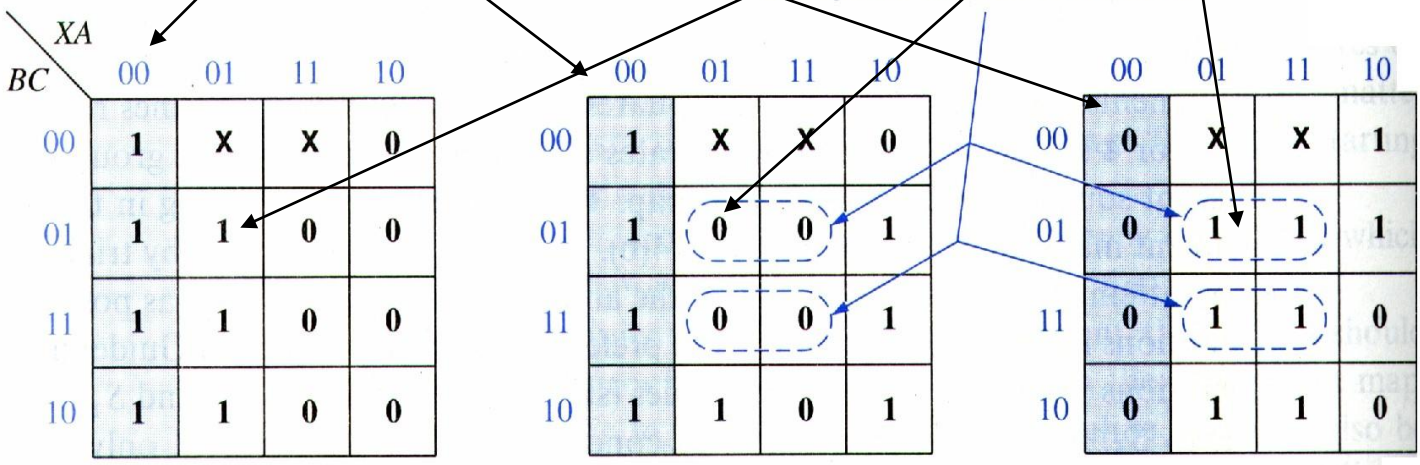


(a) Next state maps for Fig. 15-14

S_1 = 110

S_5 =101

These pairs are adjacent because S_2 and S_5 have adjacent assignments



$A^+ = D_A = X'$

$B^+ = D_B = X'C' + A'C + A'B$

$C^+ = D_C = A + XB'$

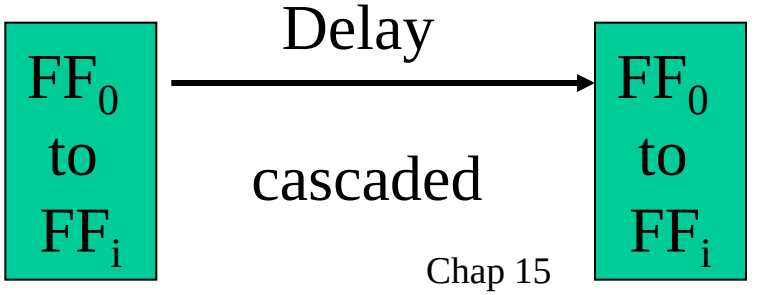
(b) Next state maps for Fig. 15-14 (cont.)

On State Assignment

- In some cases, the assignment which satisfies the most guidelines is not necessarily the best assignment. Therefore, it is a good idea to try several assignments which satisfy most of the guidelines and choose the one which gives the lowest cost function.
- In general, the best assignment for one type of flip-flop is not the best for another type.

State Assignment for CPLDs

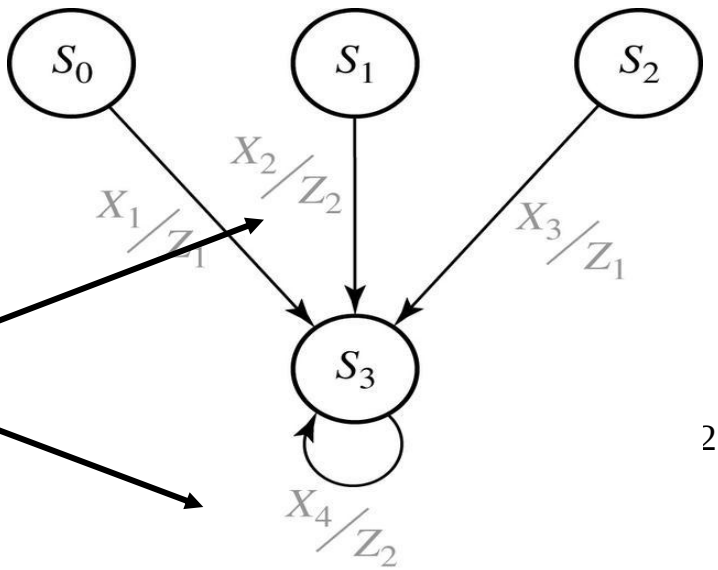
- In CPLDs, FPGAs, a logic cell has one or more FFs.
- FFs are there whether used or not.
  - May not be important to minimize the # of FFs used in the design.
- Need to reduce the logic cells used and the interconnection between cells for shorter delay. LCs are cascaded to realize a function. So min # of LCs !!
- In order design fast logic, minimize the # of cells used.



A logic cell

One-Hot State Assignment

- Use one FF for each state.
- For a 4-state machine, use 4 FFs.
  - $S_0 = Q_0Q_1Q_2Q_3 = 1000$, S_1 :0100, S_2: 0010, S_3:0001.
  - $Q_3^+ = X_1(Q_0Q_1'Q_2'Q_3') + X_2(Q_0'Q_1Q_2'Q_3') + X_3(Q_0'Q_1'Q_2Q_3') + X_4(Q_0'Q_1'Q_2'Q_3)$: this is AND
  - $Q_3^+ = X_1Q_0 + X_2Q_1 + X_3Q_2 + X_4Q_3$ reduced Q^+
  - **Each term contains only one state variable (fewer variables).**
  - **More next-state equations are required (FFs are there.)**
  - One FF is reset to 1 instead of resetting all FFs to 0 when resetting the system.
  - **But next state and output equations may contain fewer variables, meaning that fewer logic cells are required.**



$Z_2 = X_2Q_1 + X_4\,Q_3$

IN CPLD and FPGA

- Try both for state assignment
  - Minimum number of states
  - One-hot assignment
  - And see which leads to the use of the smallest number of logic cells.
    - Less delays, higher speed!