CMPE 212
Principles of Digital Design

Lecture 26

Optimization of Sequential Circuits Design

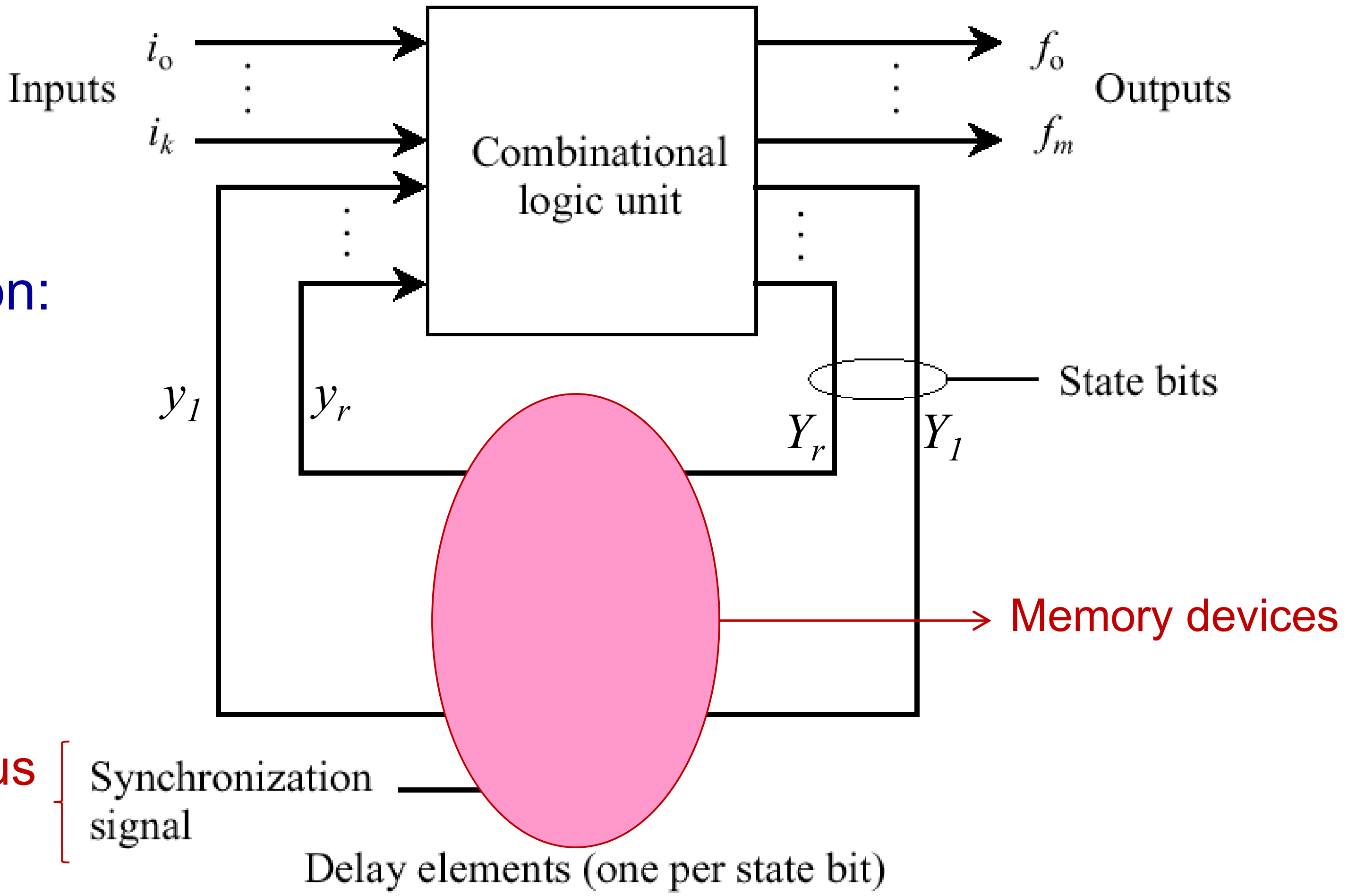
April 27, 2016

www.csee.umbc.edu/~younis/CMPE212/CMPE212.htm

Lecture's Overview

- ***Previous Lecture***
  - ➔ What does synthesis of sequential circuit mean?
  - ➔ Completely and incompletely specified sequential circuits
  - ➔ Procedure for synthesizing a sequential circuit
    1. From a word description of the problem, derive a state table
    2. Use state reduction techniques to minimize the state count
    3. Choose a state assignment and generate the state transition table
    4. Determine the type of flip-fops (memory devices) to be used
    5. Produce the switching logic equations using the excitation maps
    6. Draw a schematic of the sequential circuit
- ***This Lecture***
  - ➔ Simplification of sequential circuits

Sequential Circuit Model



Possible realization:

1. Mealy model
2. Moore model

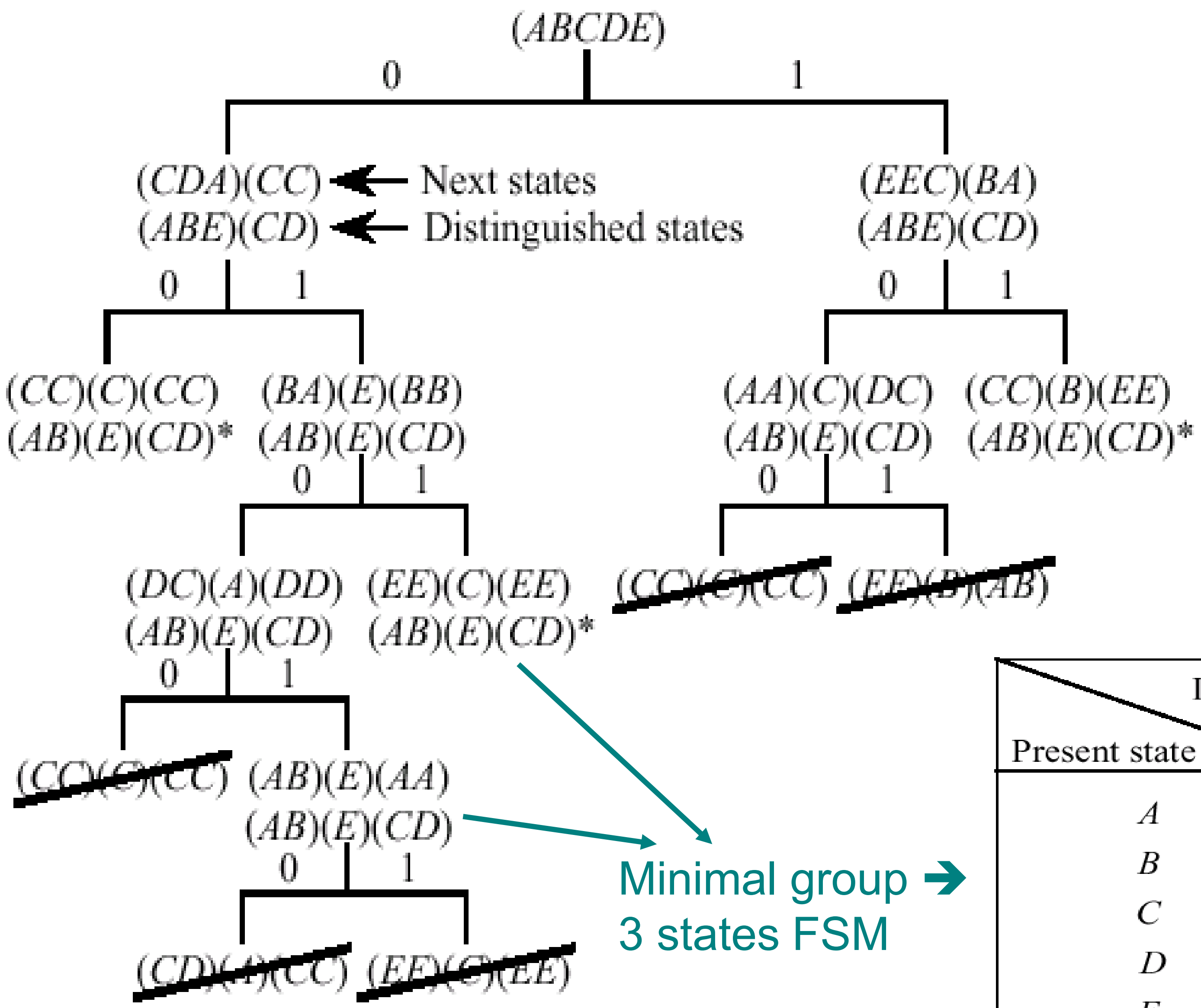
Can be synchronous or asynchronous

- ❑ Composed of a combinational logic unit and delay elements in a feedback path, which maintains state information
- ❑ Defined by output relation to input and circuit state (values in flip-flops)

State Reduction

- After designing an FSM, the following question usually arises:
  Is there a functionally equivalent machine with fewer states?
- Machine equivalence:
  Two FSM are equivalent if they produce similar outputs for the same input sequence for all possible input sequences
- State equivalence:
  Two states are equivalent if they are indistinguishable from each other for any set of inputs by just watching the FSM output
- The objective of state reduction techniques is to find equivalent states within the FSM so that they can be eliminated
- Since state equivalence is based on monitoring the machine output, state reduction techniques classifies states based on outputs for a string of inputs
- Indistinguishable states are usually grouped in parenthesis

Distinguishing Tree



- Initially an FSM can be in any state
- After applying an input, states are regrouped based on the output
- Repeated groups are not pursued further (marked by *)
- Final groups represent minimal number of states

Present state \ Input	X = 0	X = 1
A	C/0	E/1
B	D/0	E/1
C	C/1	B/0
D	C/1	A/0
E	A/0	C/1

Simpler Approach: Inspection

	x = 0	x = 1
A	🟥	🟥
B	🟪	🟪
C	G/0	D/0
D	E/1	F/0
E	🟪	🟪
F	🟥	🟥
G	G/0	F/0

	x = 0	x = 1
A	B/0	A/0
B	C/0	A/0
C	G/0	D/0
D	B/1	A/0
G	G/0	A/0

- Replace F with A and eliminate F
- Replace E with B and eliminate E

Two states are equivalent if the next state rows are identical or when the next-state rows are identical except for "self-loop-back" entries

	x = 0	x = 1
A	*B/0*	*C/1*
B	*B/0*	*A/1*
C	*D/1*	*B/0*
D	*D/0*	*A/1*

	x = 0	x = 1
A	*B/0*	*C/1*
B	*D/0*	*A/1*
C	*D/1*	*B/0*
D	*B/0*	*A/1*

	x = 0	x = 1
A	*B/0*	*C/1*
B	*B/0*	*A/1*
C	*B/1*	*B/0*

B and D are equivalent states

Good first step and may not reveal all cases (requires experience)

State Partitioning Technique

- Constructing distinguishing trees is laborious because of the large set of possibilities and repetitions
- Simplification can be made by considering the groups of the first level of partitioning and checking the states whether their next states are in one group or different groups
- If two states S_1 and S_2, within one group have their next states in separate groups, then S_1 and S_2 are distinguishable (after applying one more input)

Present state \ Input	X = 0	X = 1
	0	1
A	C/0	E/1
B	D/0	E/1
C	C/1	B/0
D	C/1	A/0
E	A/0	C/1

Reduced to ⬇ 3 states FSM

Current state \ Input	X = 0	X = 1
	0	1
AB: A′	*B′/0*	*C′/1*
CD: B′	*B′/1*	*A′/0*
E: C′	*A′/0*	*B′/1*

P_1 = (ABCDE)

A and B generate same output and similarly for C and D while E is unique

P_2 = (ABE)(CD)

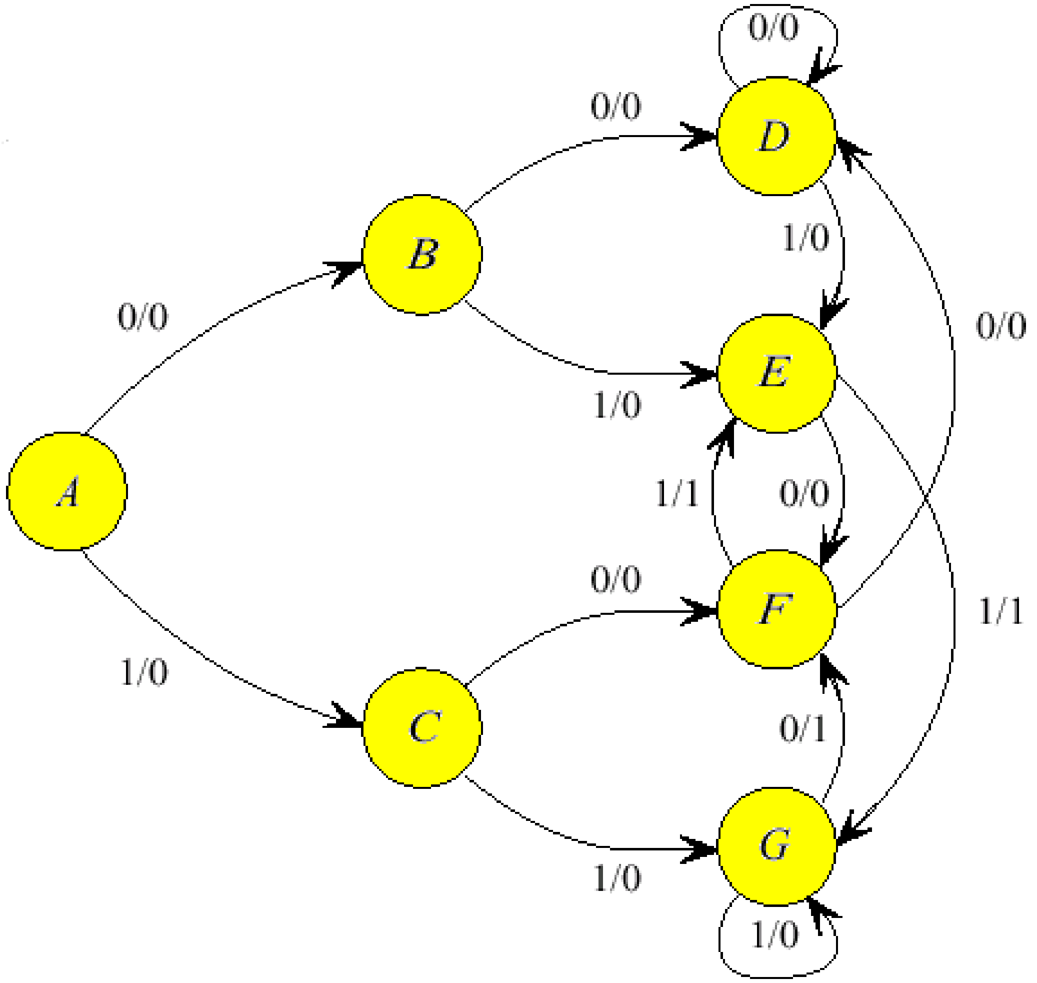
On 0: (AB) → (CD), (CD) →(C)

On 1: (AB) → (E), (CD) →(AB)

P_3 = (AB)(CD)(E) ✓

Sequence Detector

- Design a machine that outputs a 1 when exactly two of the last three inputs are 1, e.g. input sequence of 011011100 produces an output sequence of 001111010.



Present state \ Input	X = 0	X = 1
A	*B*/0	*C*/0
B	*D*/0	*E*/0
C	*F*/0	*G*/0
D	*D*/0	*E*/0
E	*F*/0	*G*/1
F	*D*/0	*E*/1
G	*F*/1	*G*/0

```
Input:   0 1 1 0 1 1 1 0 0
Output:  0 0 1 1 1 1 0 1 0
Time:    0 1 2 3 4 5 6 7 8
```

* Slide is courtesy of M. Murdocca and V. Heuring

Sequence Detector

Present state \ Input	X = 0	X = 1
A	$B/0$	$C/0$
B	$D/0$	$E/0$
C	$F/0$	$G/0$
D	$D/0$	$E/0$
E	$F/0$	$G/1$
F	$D/0$	$E/1$
G	$F/1$	$G/0$

Reduced to 6 states

Present state \ Input	X = 0	X = 1
$A: A'$	$B'/0$	$C'/0$
$BD: B'$	$B'/0$	$D'/0$
$C: C'$	$E'/0$	$F'/0$
$E: D'$	$E'/0$	$F'/1$
$F: E'$	$B'/0$	$D'/1$
$G: F'$	$E'/1$	$F'/0$

P_1 = (ABCDEFG)

A, B, C and DB generate same output and similarly for E and F while G is unique

P_2 = (ABCD)(EF)(G)

On 0: (ABCD) → (BD)(F),
(EF) →(F)(D)

On 1: (ABCD) → (C)(E)(G),
(EF) →(E)(G)

P_3 = (A)(BD)(C)(E)(F)(G)

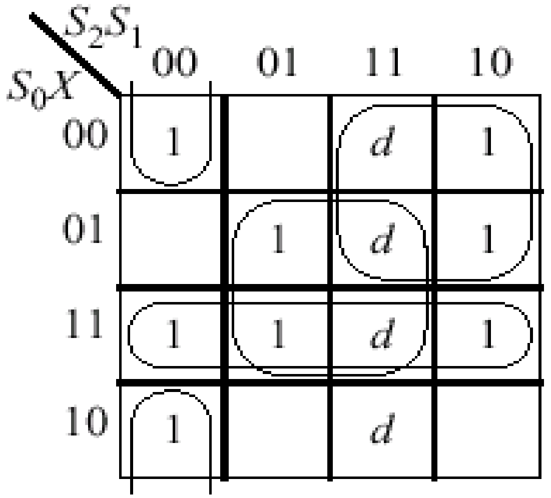
P_4 = (A)(BD)(C)(E)(F)(G) ✓

Sequence Detector Logic Design

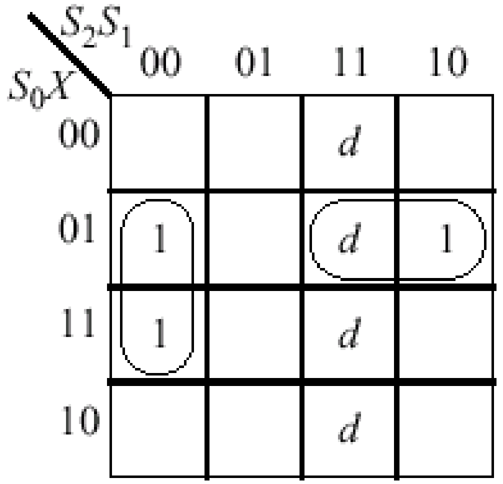
Present state \ Input	X = 0	X = 1
$A: A'$	$B'/0$	$C'/0$
$BD: B'$	$B'/0$	$D'/0$
$C: C'$	$E'/0$	$F'/0$
$E: D'$	$E'/0$	$F'/1$
$F: E'$	$B'/0$	$D'/1$
$G: F'$	$E'/1$	$F'/0$

Assign States

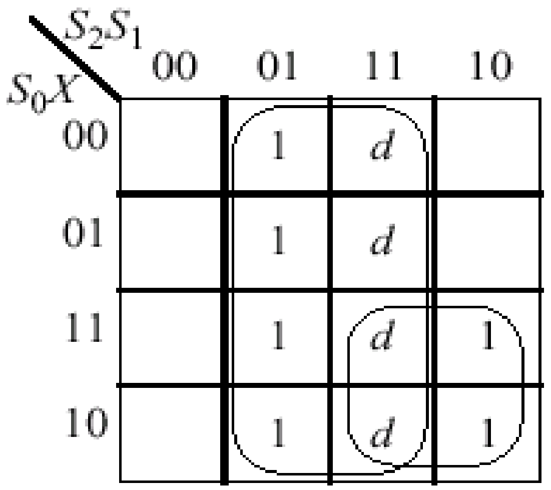
Present state \ Input	X = 0	X = 1
$S_2S_1S_0$	$S_2S_1S_0Z$	$S_2S_1S_0Z$
A': 000	001/0	010/0
B': 001	001/0	011/0
C': 010	100/0	101/0
D': 011	100/0	101/1
E': 100	001/0	011/1
F': 101	100/1	101/0



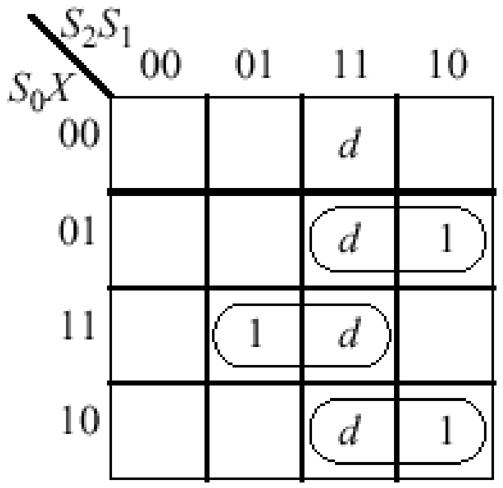
$$S_0 = \overline{S_2}\,\overline{S_1}\,\overline{X} + S_0X + S_2\overline{S_0} + S_1X$$



$$S_1 = \overline{S_2}\,\overline{S_1}X + S_2\overline{S_0}X$$



$$S_2 = S_2S_0 + S_1$$



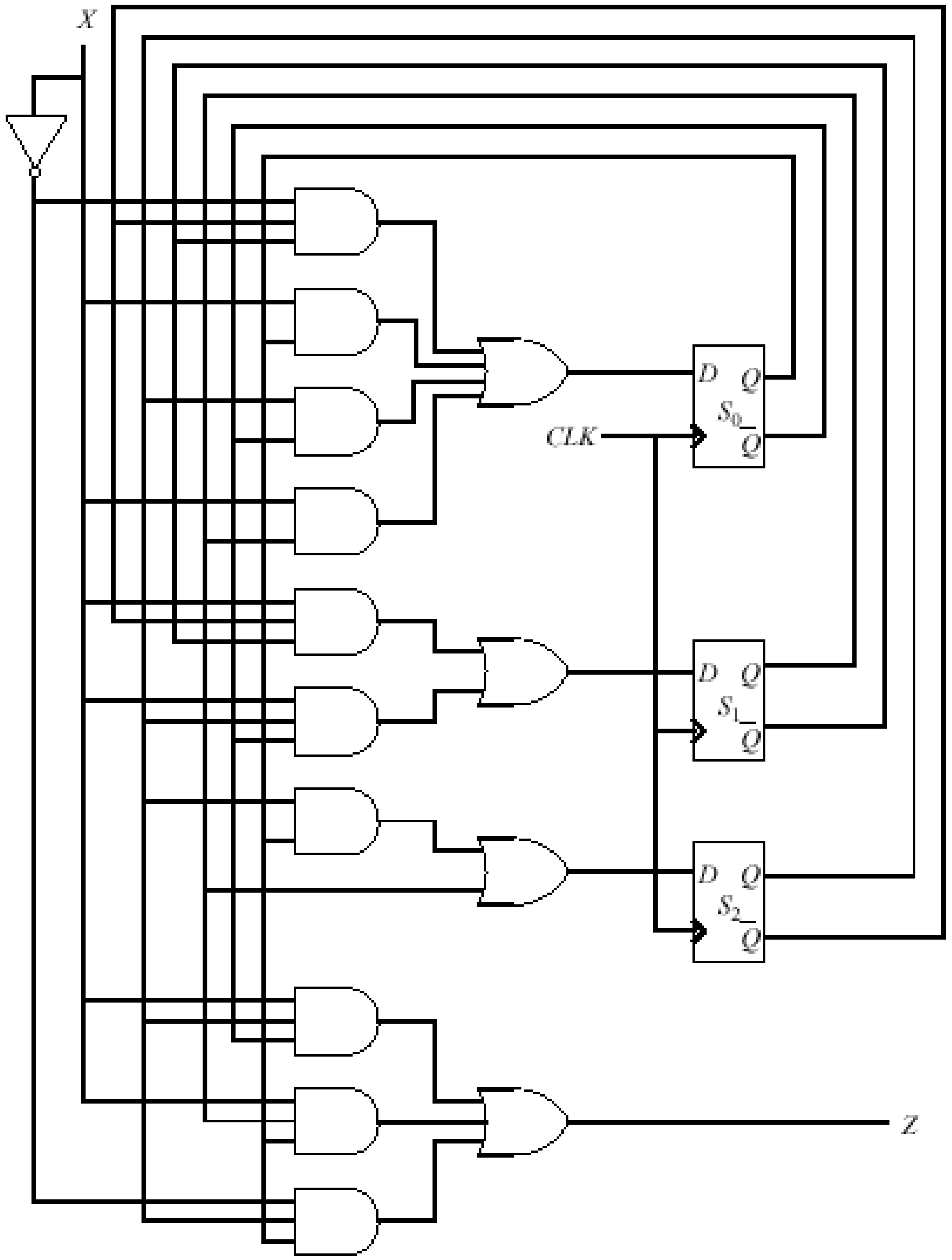
$$Z = S_2\overline{S_0}X + S_1S_0X + S_2S_0\overline{X}$$

Sequence Detector Circuit

D flip-flop

Q_t	Q_{t+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

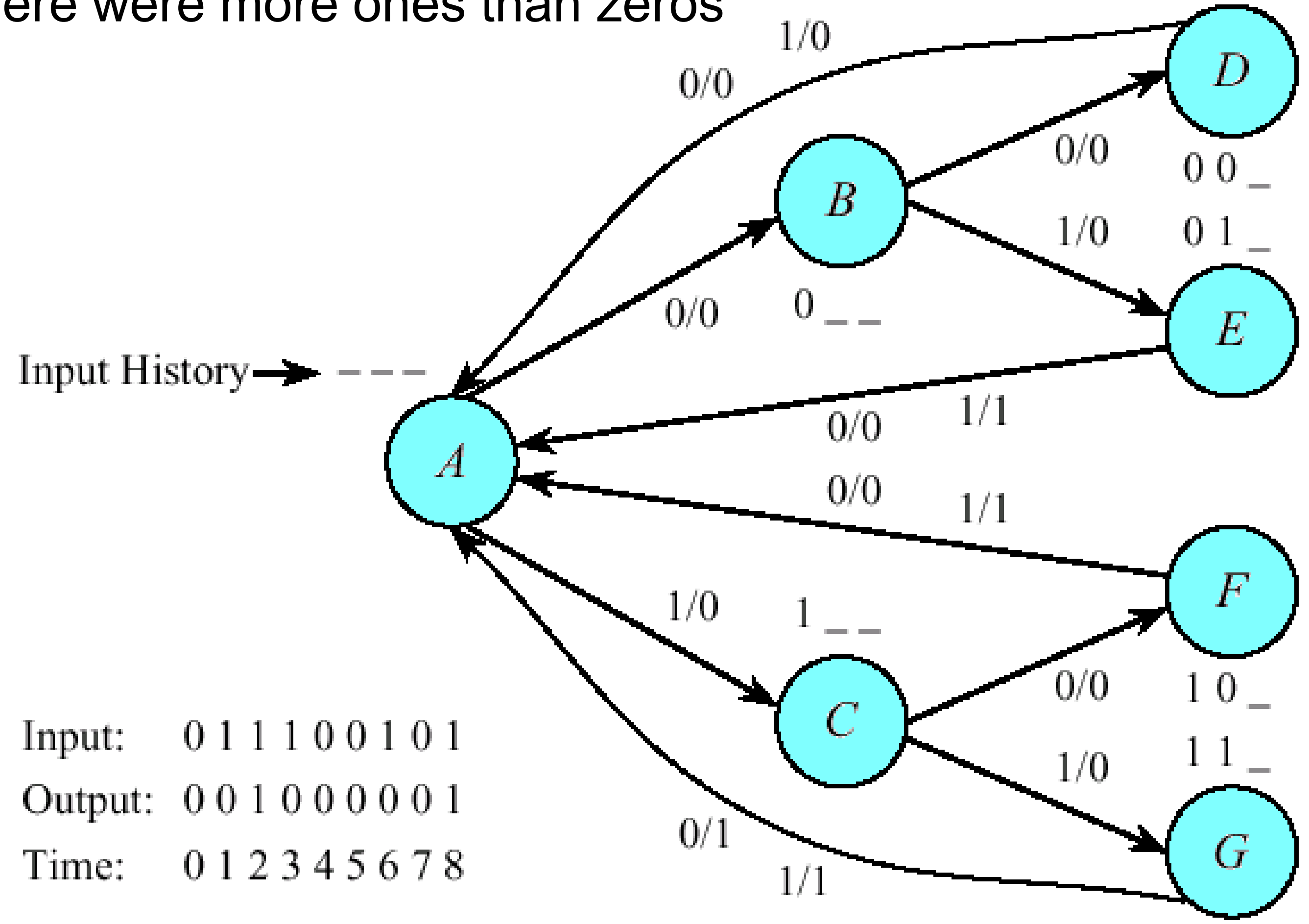
For a D flip-flop the new state simply matches the D input



* Slide is courtesy of M. Murdocca and V. Heuring

Majority Finite State Machine

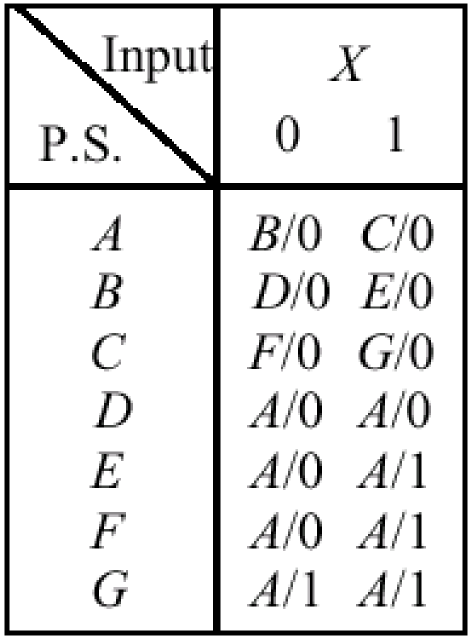
- Design a circuit using T flip-flops and 8-to-1 MUX that computes the majority function of batches of 3 inputs (no overlapping)
- The machine should output zeros until encountering 3 inputs and then output a 1 if there were more ones than zeros



* Figure is courtesy of M. Murdocca and V. Heuring

Majority FSM State Table

State table for majority FSM

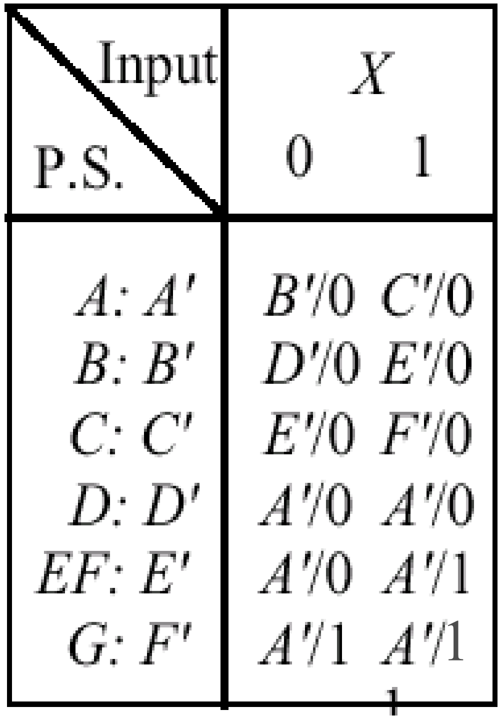


Input / P.S.	X = 0	X = 1
A	$B/0$	$C/0$
B	$D/0$	$E/0$
C	$F/0$	$G/0$
D	$A/0$	$A/0$
E	$A/0$	$A/1$
F	$A/0$	$A/1$
G	$A/1$	$A/1$

Partitioning

$$P_0 = (ABCDEFG)$$
$$P_1 = (ABCD)(EF)(G)$$
$$P_2 = (AD)(B)(C)(EF)(G)$$
$$P_3 = (A)(B)(C)(D)(EF)(G)$$
$$P_4 = (A)(B)(C)(D)(EF)(G) \surd$$

Reduced state table



Input / P.S.	X = 0	X = 1
$A: A'$	$B'/0$	$C'/0$
$B: B'$	$D'/0$	$E'/0$
$C: C'$	$E'/0$	$F'/0$
$D: D'$	$A'/0$	$A'/0$
$EF: E'$	$A'/0$	$A'/1$
$G: F'$	$A'/1$	$A'/1$

* Slide is courtesy of M. Murdocca and V. Heuring

Majority FSM State Assignment

State assignment for reduced majority FSM:

Using D flip-flops

Input / P.S.	X = 0	X = 1
$S_2S_1S_0$	$S_2S_1S_0Z$	$S_2S_1S_0Z$
A': 000	001/0	010/0
B': 001	011/0	100/0
C': 010	100/0	101/0
D': 011	000/0	000/0
E': 100	000/0	000/1
F': 101	000/1	000/1

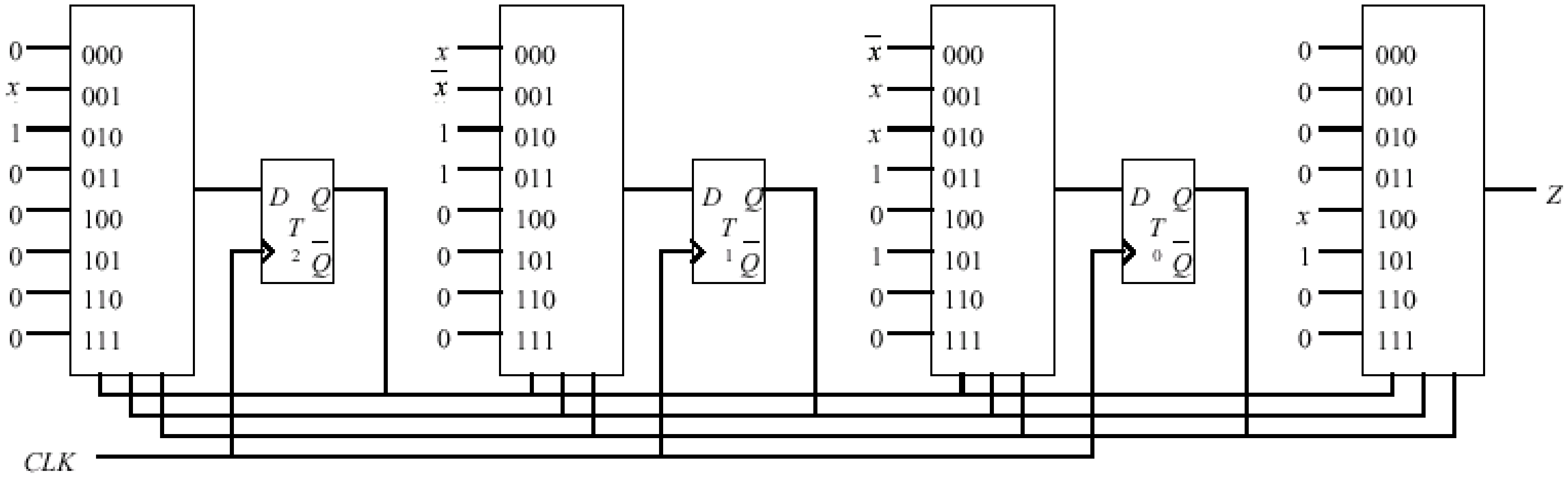
Using T flip-flops

Input / P.S.	X = 0	X = 1
$S_2S_1S_0$	$T_2T_1T_0Z$	$T_2T_1T_0Z$
A': 000	001/0	010/0
B': 001	010/0	101/0
C': 010	110/0	111/0
D': 011	011/0	011/0
E': 100	100/0	100/1
F': 101	101/1	101/1

T values are driven from D counterparts and referencing the excitation table for T flip-flops

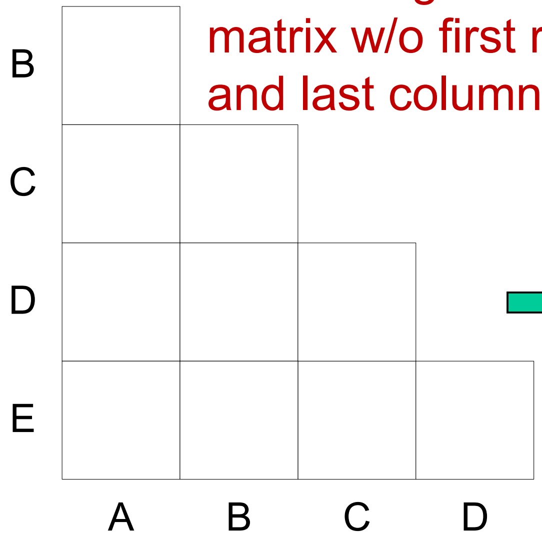
* Slide is courtesy of M. Murdocca and V. Heuring

Majority FSM Circuit

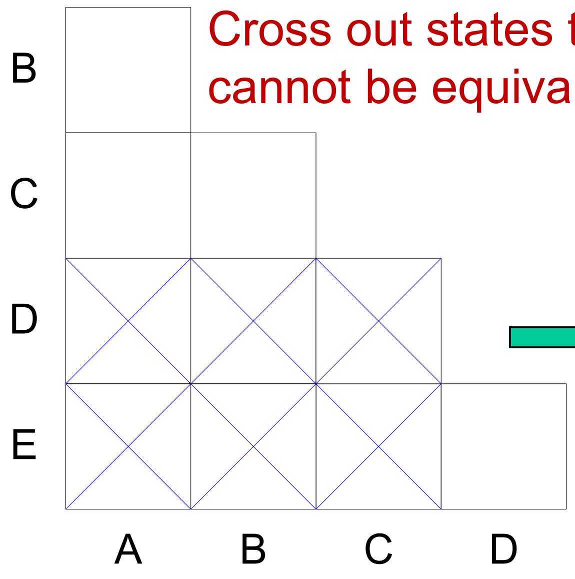


* Slide is courtesy of M. Murdocca and V. Heuring

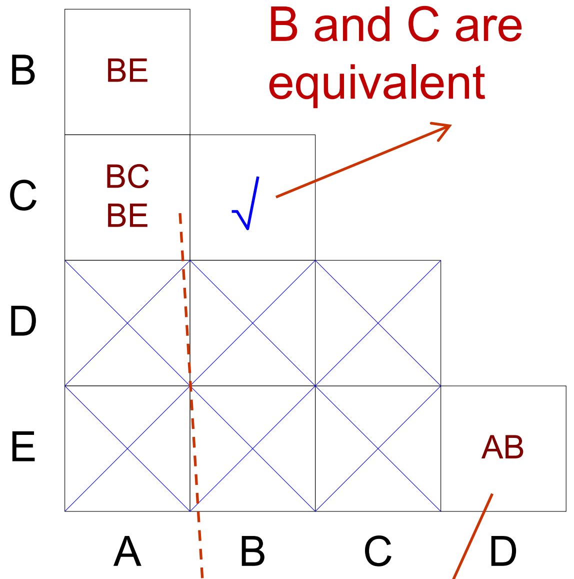
Implication Table



Like a diagonal matrix w/o first row and last column



Cross out states that cannot be equivalent



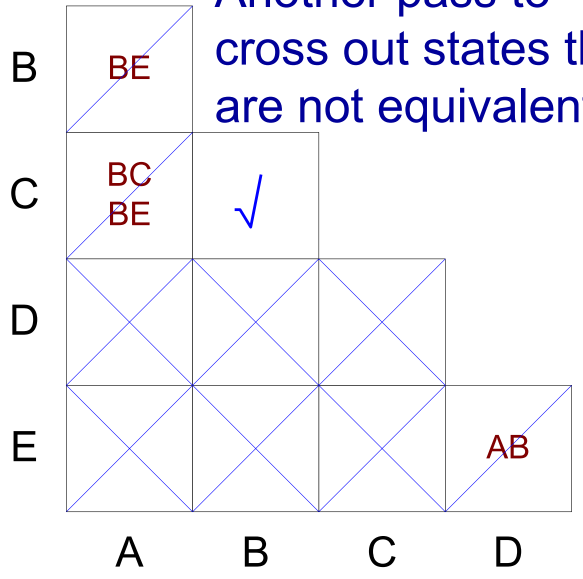
By inspection, B and C are equivalent

E and D are equivalent only if A and B are equivalent

	x = 0	1
A	*C/1*	*B/0*
B	*C/1*	*E/0*
C	*B/1*	*E/0*
D	*D/0*	*B/1*
E	*E/0*	*A/1*

Reduced to

	x = 0	1
A	*B/1*	*B/0*
B	*B/1*	*E/0*
D	*D/0*	*B/1*
E	*E/0*	*A/1*

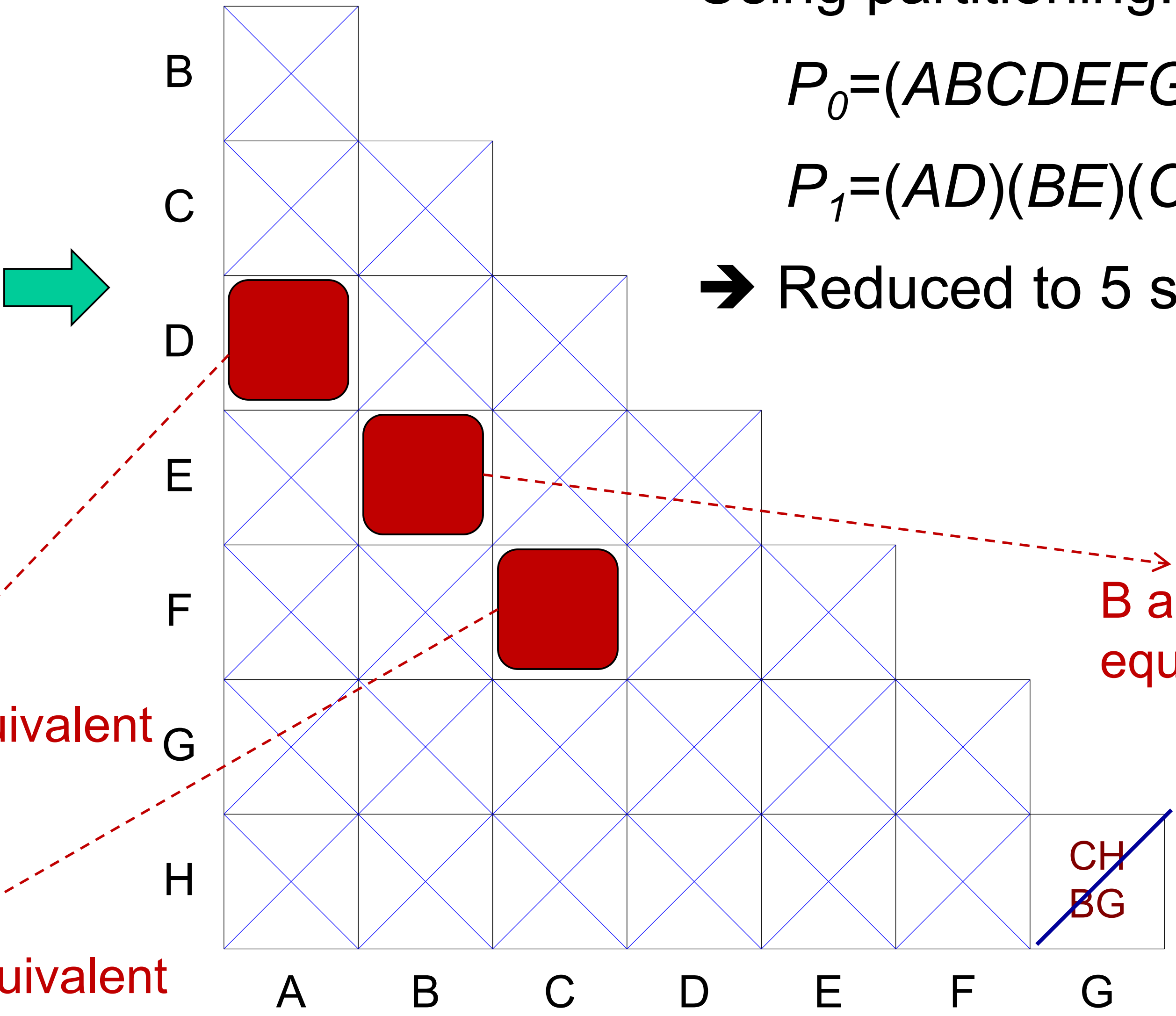


Another pass to cross out states that are not equivalent

List all possible conditions

Example: Implication Table

	x = 0	1
A	*E/0*	*D/0*
B	*A/1*	*F/0*
C	*C/0*	*A/1*
D	*B/0*	*A/0*
E	*D/1*	*C/0*
F	*C/0*	*D/1*
G	*H/1*	*G/1*
H	*C/1*	*B/1*



A and D are equivalent

B and E are equivalent

C and F are equivalent

Using partitioning:

$P_0=(ABCDEFGH)$

$P_1=(AD)(BE)(CF)(G)(H)$

➔ Reduced to 5 states

The State Assignment Problem

- State assignment can have an impact on the complexity of the implementation at the gate-level
- Number of possible assignments = $2^{N_{ff}}$, where $N_{ff} = \lceil \log(\#states) \rceil$
- A MUX based implementation makes all possible state assignment of the same complexity

Input / P.S.	X = 0	X = 1
A	$B/1$	$A/1$
B	$C/0$	$D/1$
C	$C/0$	$D/0$
D	$B/1$	$A/0$

Machine M_2

Input / S_0S_1	X = 0	X = 1
A: 00	01/1	00/1
B: 01	10/0	11/1
C: 10	10/0	11/0
D: 11	01/1	00/0

State assignment SA_0

Input / S_0S_1	X = 0	X = 1
A: 00	01/1	00/1
B: 01	11/0	10/1
C: 11	11/0	10/0
D: 10	01/1	00/0

State assignment SA_1

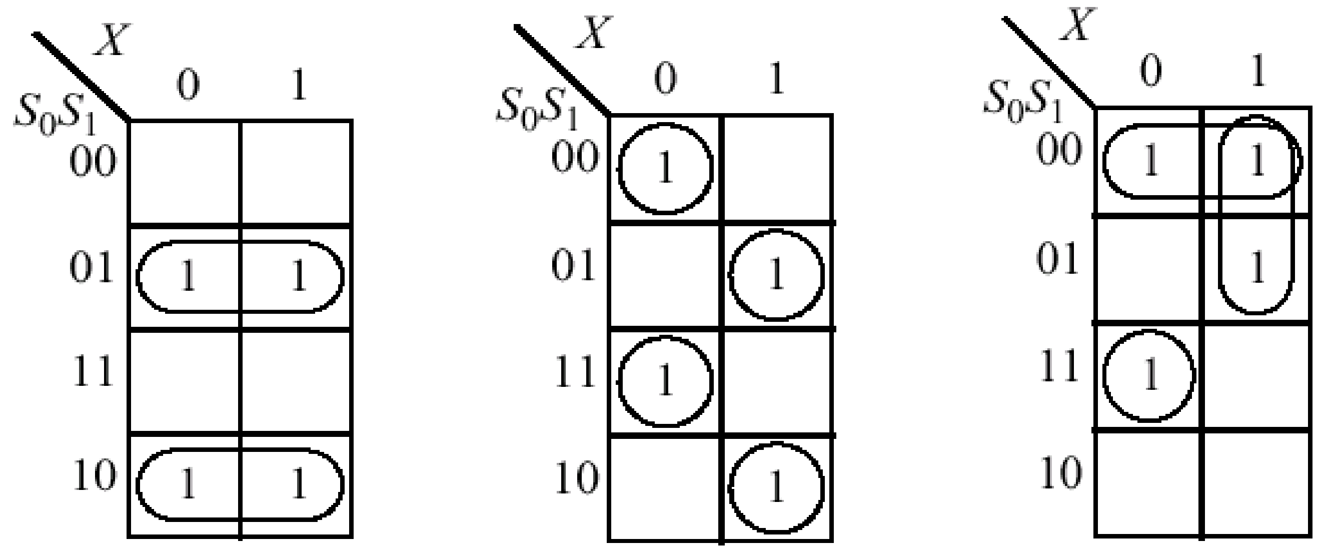
Example: Two state assignments for machine M_2, Which one is better?

State Assignment SA_0

Input S_0S_1	X = 0	X = 1
A: 00	01/1	00/1
B: 01	10/0	11/1
C: 10	10/0	11/0
D: 11	01/1	00/0

State assignment SA_0

- Boolean equations for machine M_2 using state assignment SA_0
- Input count = 29



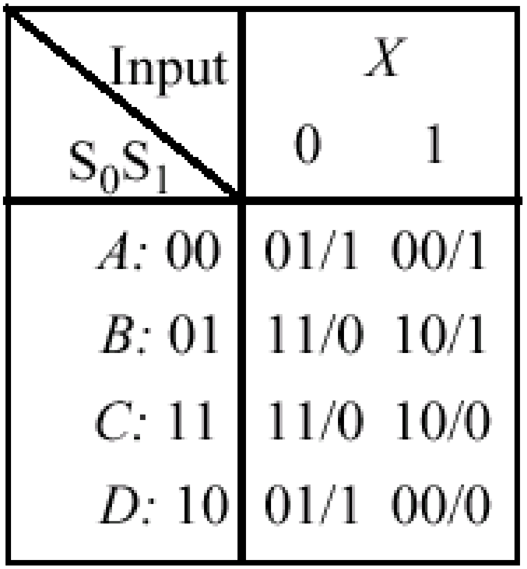
$$S_0 = \overline{S_0}S_1 + S_0\overline{S_1}$$

$$S_1 = \overline{S_0}\,\overline{S_1}\,\overline{X} + \overline{S_0}S_1X + S_0S_1\overline{X} + S_0\overline{S_1}X$$

$$Z = \overline{S_0}\,\overline{S_1} + \overline{S_0}X + S_0S_1\overline{X}$$

* Slide is courtesy of M. Murdocca and V. Heuring

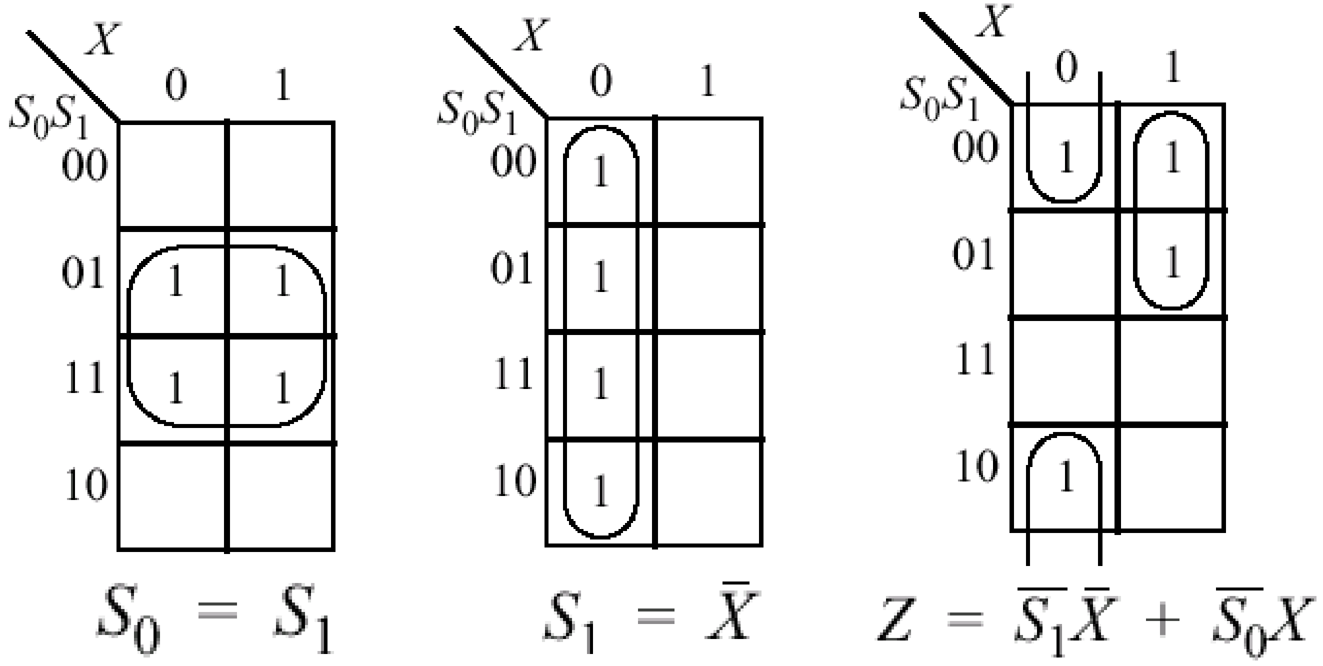
State Assignment SA_1



Input \ S_0S_1	X = 0	1
A: 00	01/1	00/1
B: 01	11/0	10/1
C: 11	11/0	10/0
D: 10	01/1	00/0

State assignment SA_1

- Boolean equations for machine M_2 using state assignment SA_1
- Input count = 6 ➔ SA_1 is better than SA_0
- How to know the best assignment upfront?



$$S_0 = S_1 \qquad S_1 = \bar{X} \qquad Z = \overline{S_1}\bar{X} + \overline{S_0}X$$

* Slide is courtesy of M. Murdocca and V. Heuring

Optimal State Assignment

Popular objectives

- Minimal gate count and/or fan-in per gate
- Minimal cost circuit
- Circuit with reduced dependency

Guidelines

- Rule 1: States that have the same next states for a given input should be given logically adjacent assignments
- Rule 2: States that are the next states of a single present state under logically adjacent inputs should be given logically adjacent assignments

Algorithms

(1) Implication graph (2) State partitioning

- Focus on reducing the gate input count
- Detect relationship between two states that may lead to logic reduction

Guidelines: Illustrative Example

	x = 0	1
A	$C/0$	$D/0$
B	$C/0$	$A/0$
C	$B/0$	$D/0$
D	$A/1$	$B/1$

y_2y_1 \ x	0	1
00	**11/0**	**10/0**
01	**11/0**	**00/0**
11	**01/0**	**10/0**
10	**00/1**	**01/1**

$Y_2\,Y_1/z$

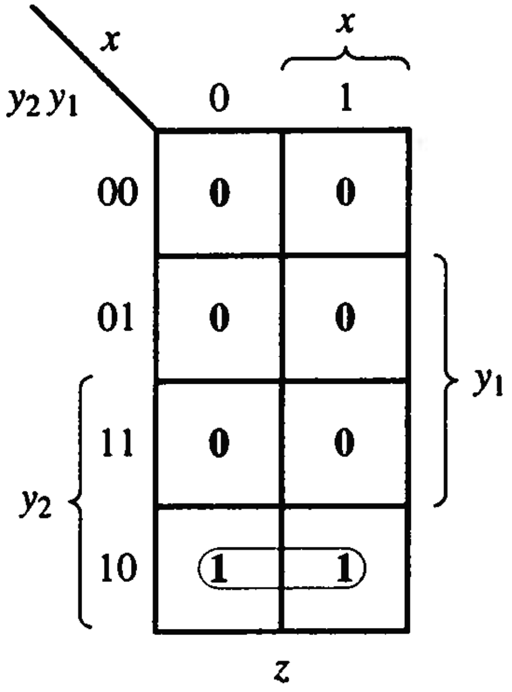
Assignment #1

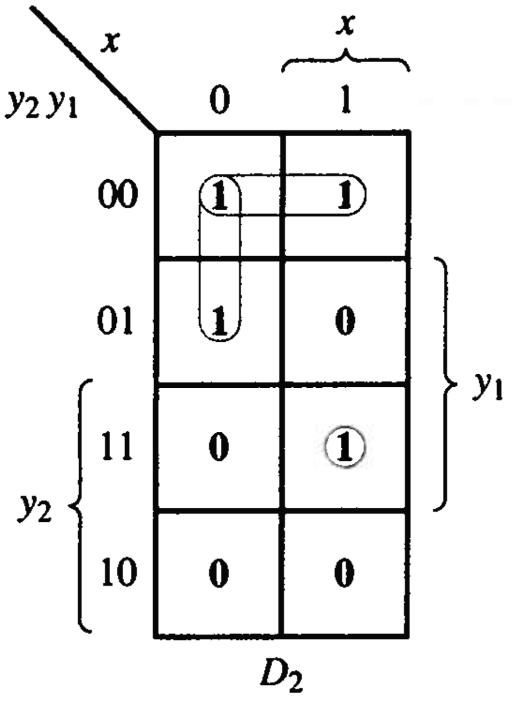
$$D_2 = \overline{y_1}\,\overline{y_2} + \bar{x}\,\overline{y_2} + x y_1 y_2$$

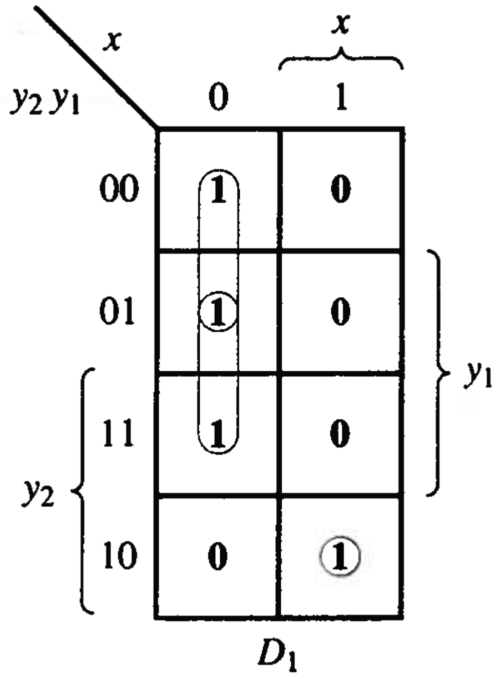
$$D_1 = \bar{x}\,\overline{y_1} + \bar{x}\,\overline{y_2} + x\overline{y_1} y_2$$

$$Z = \overline{y_1} y_2$$

Total 20 gate inputs







Illustrative Example (Cont.)

	$x=0$	$x=1$
A	*C*/0	*D*/0
C	*B*/0	*D*/0
B	*C*/0	*A*/0
D	*A*/1	*B*/1

y_2y_1 \ x	0	1
00	01/0	10/0
01	11/0	10/0
11	01/0	00/0
10	00/1	11/1

$Y_2\,Y_1/z$

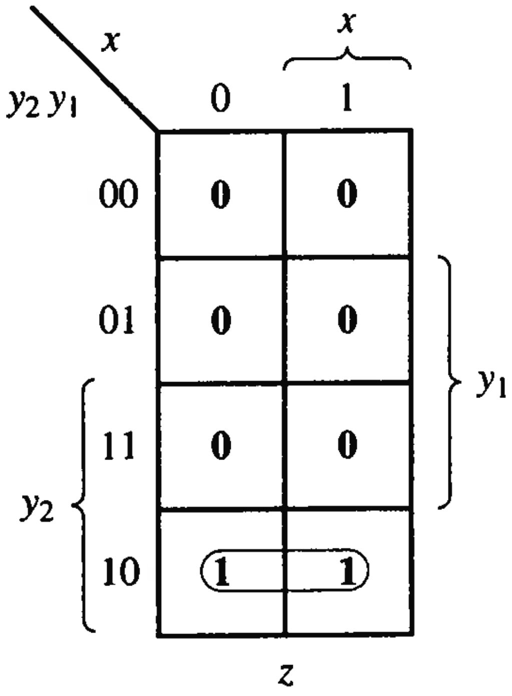
Assignment #2

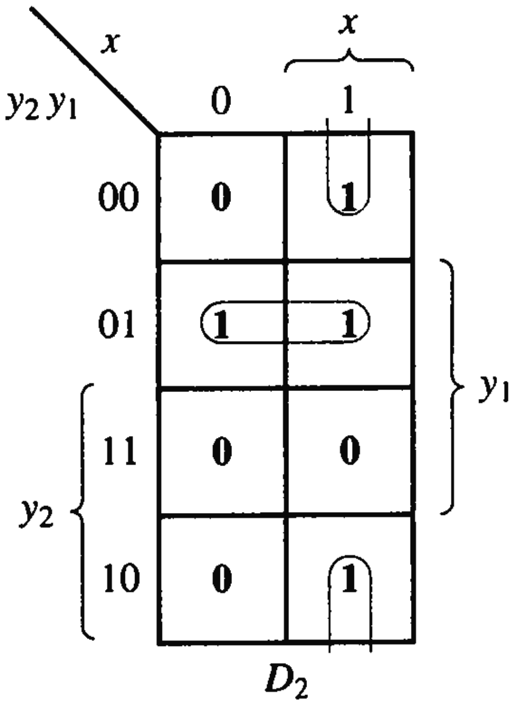
$$D_2 = x\overline{y_1} + y_1\overline{y_2}$$

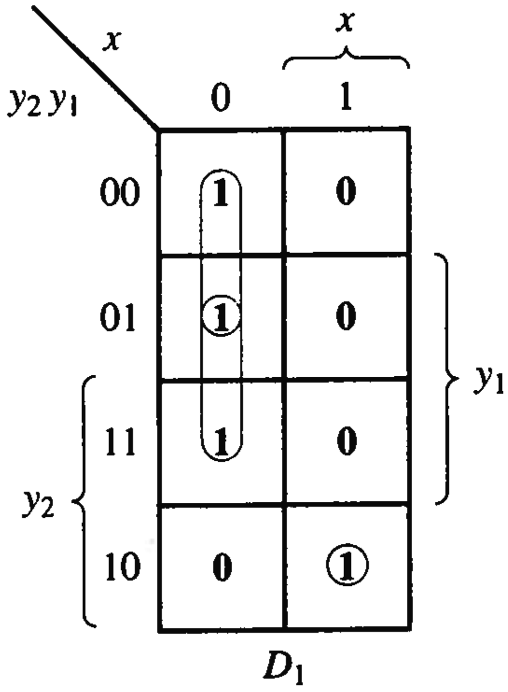
$$D_1 = \bar{x}\,\overline{y_2} + \bar{x}y_1 + x\overline{y_1}y_2$$

$$Z = \overline{y_1}y_2$$

Total 18 gate inputs







Illustrative Example (Cont.)

	$x = 0$	$x = 1$
A	*C/0*	*D/0*
C	*B/0*	*D/0*
D	*A/1*	*B/1*
B	*C/0*	*A/0*

y_2y_1 \ x	0	1
00	**01/0**	**11/0**
01	**10/0**	**11/0**
11	**00/1**	**10/1**
10	**01/0**	**00/0**

$Y_2\,Y_1/z$

Assignment #3

$$D_2 = x\overline{y_2} + xy_1 + y_1\overline{y_2}$$

$$D_1 = x\overline{y_2} + \bar{x}\,\overline{y_1}$$

✓

$$Z = y_1y_2$$

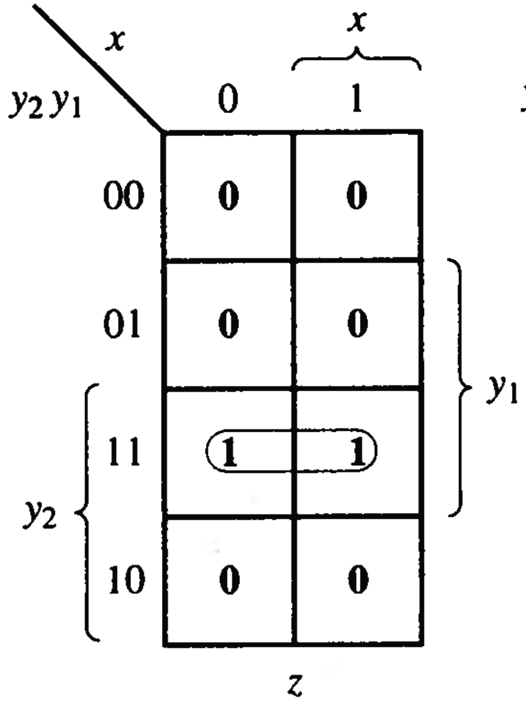
Total 15 gate inputs

Rule 1:

A and B go to same state ➔ A adjacent to B

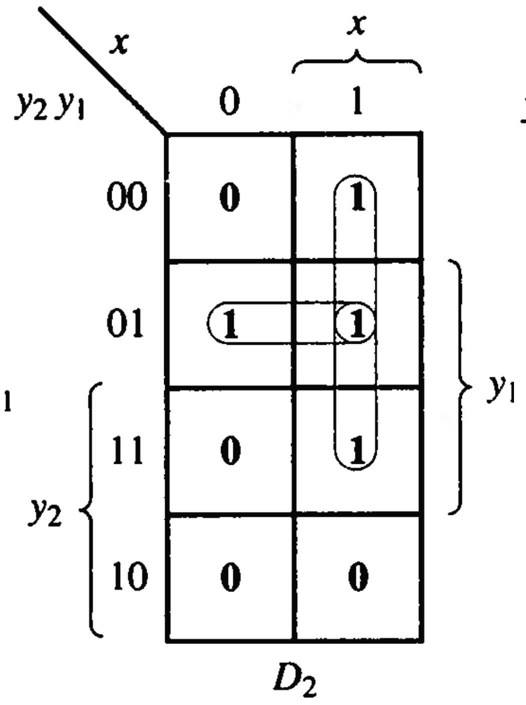
Rule 2:

A adj B,
A adj C,
B adj D, and
C adj D



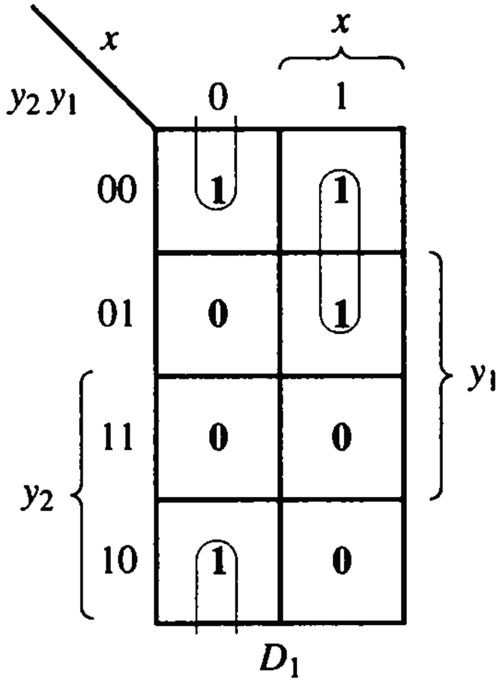
y_2y_1 \ x	0	1
00	0	0
01	0	0
11	1	1
10	0	0

z



y_2y_1 \ x	0	1
00	0	1
01	1	1
11	0	1
10	0	0

D_2



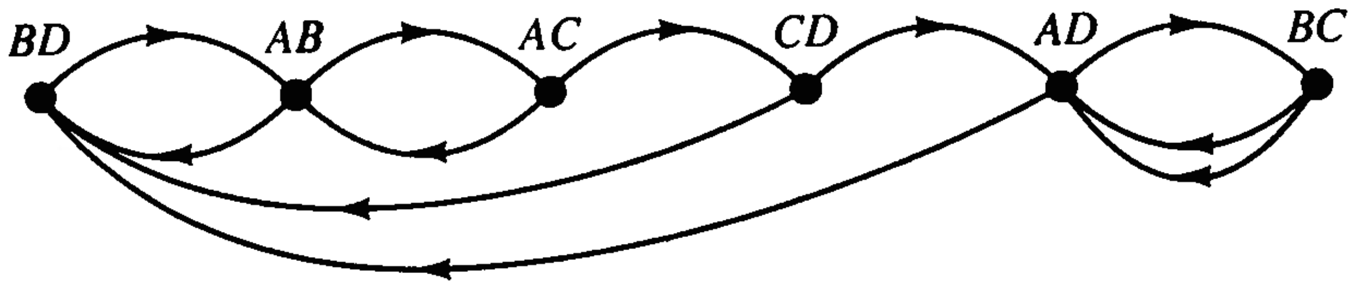
y_2y_1 \ x	0	1
00	1	1
01	0	1
11	0	0
10	1	0

D_1

Implication Graph

- Is a flow graph whose nodes represent pairs of states; nodes are connected if there is a state transition between two pairs of states for a given input

	x = 0	x = 1
A	B/0	C/0
B	D/0	A/1
C	A/1	D/0
D	D/1	B/1

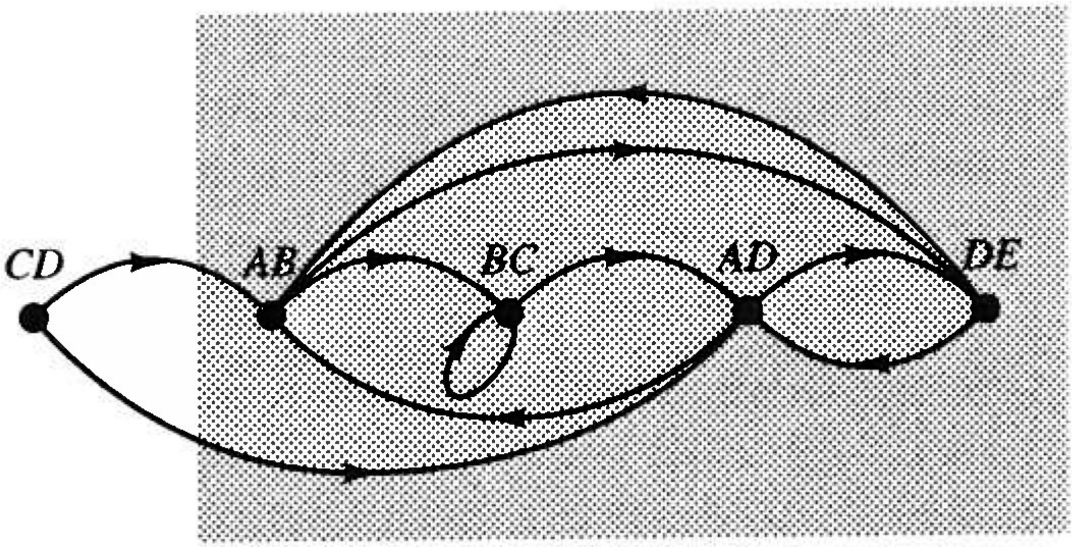


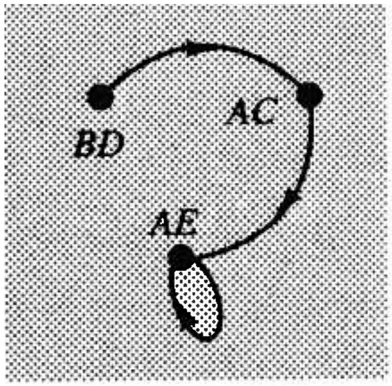
- Start with a pair, e.g. AB and identify the pair that they transition to, i.e., BD.
- Using BD find the implies state pairs for one input, which AB (x=1), and so on

- An implication graph is said to be *complete* if it contains all possible pairs of states (often the implication is not complete)
- A sub-graph is defined as the a part of a complete graph
- A sub-graph is called *closed*, if all outgoing arcs for each node in the sub-graph terminate on nodes completely contained within the sub-graph and every state in the circuit is represented by at least one node in the sub-graph

Closed Sub-graphs based Adjacency

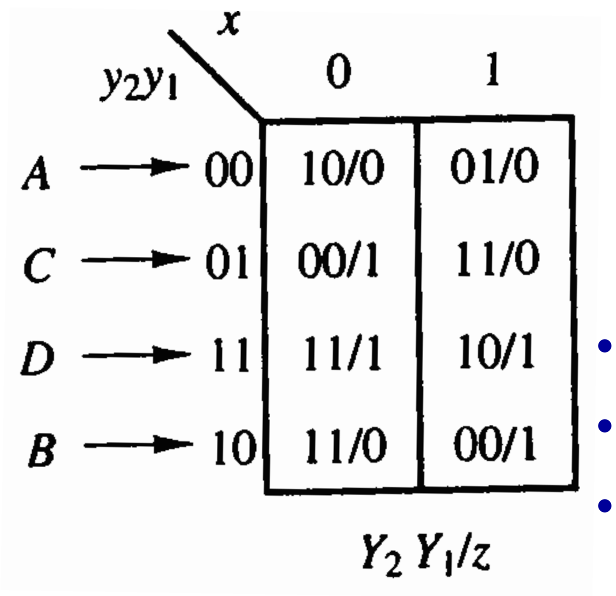
	x = 0	x = 1
A	B/0	E/0
B	C/1	D/1
C	B/0	A/0
D	A/0	D/0
E	B/1	A/1





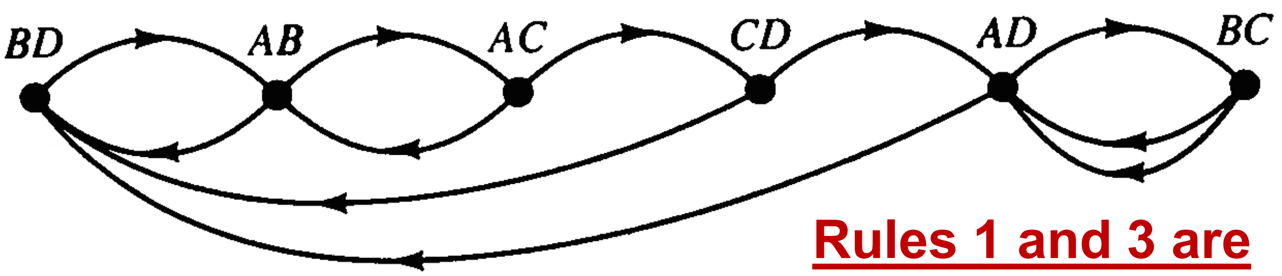
- Closed Sub-graphs: Arcs may enter a closed sub-graph from the exterior, but none may originate within a closed sub-graph and exit
- After applying the two rules to exploit logical adjacency through careful assignment, the implication graph can be used for two logically adjacent nodes to make their next-state-pairs also physically adjacent in a closed sub-graph of implication graph

Example



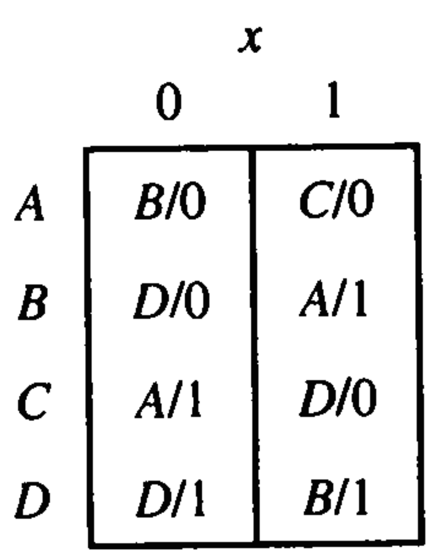
	y_2y_1 \ x	0	1
A	00	10/0	01/0
C	01	00/1	11/0
D	11	11/1	10/1
B	10	11/0	00/1

$Y_2\,Y_1/z$

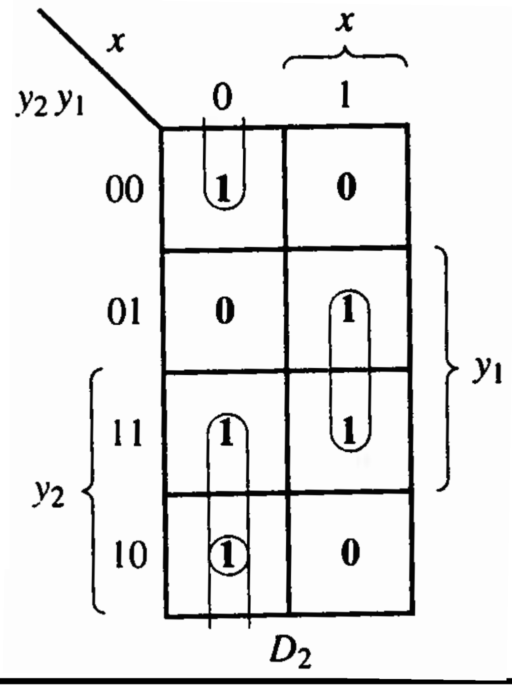


- Rule 1: B adj D
- Rule 2: D adj B, B adj C, and A adj D
- Rule 3: BD → AB, AC, and CD (could not support AD and BC as part of a closed sub-graph)

Rules 1 and 3 are most important

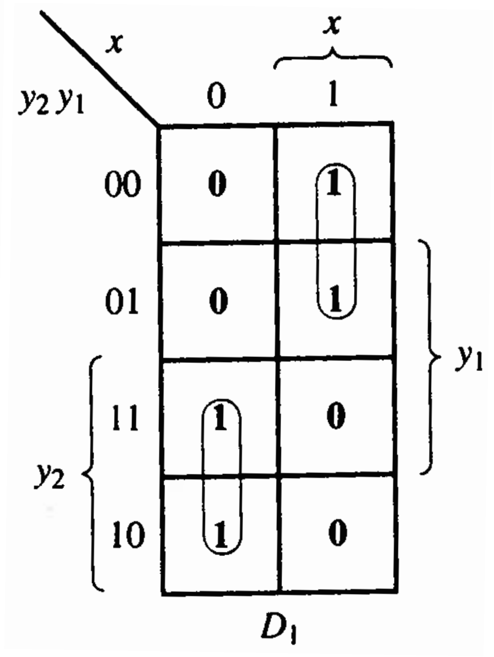


	x = 0	x = 1
A	B/0	C/0
B	D/0	A/1
C	A/1	D/0
D	D/1	B/1



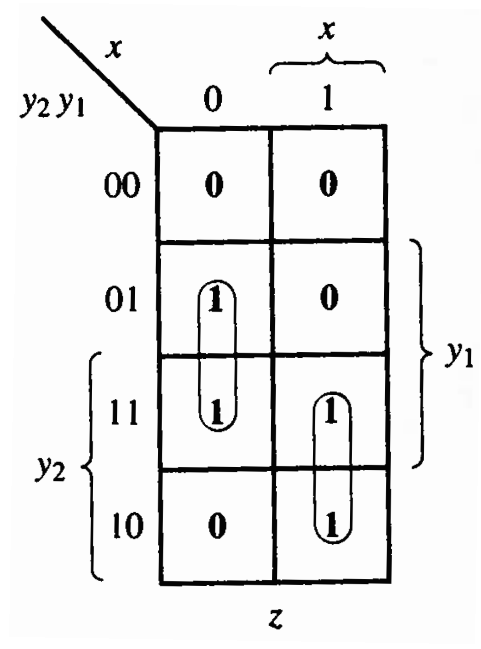
y_2y_1 \ x	0	1
00	1	0
01	0	1
11	1	1
10	1	0

D_2



y_2y_1 \ x	0	1
00	0	1
01	0	1
11	1	0
10	1	0

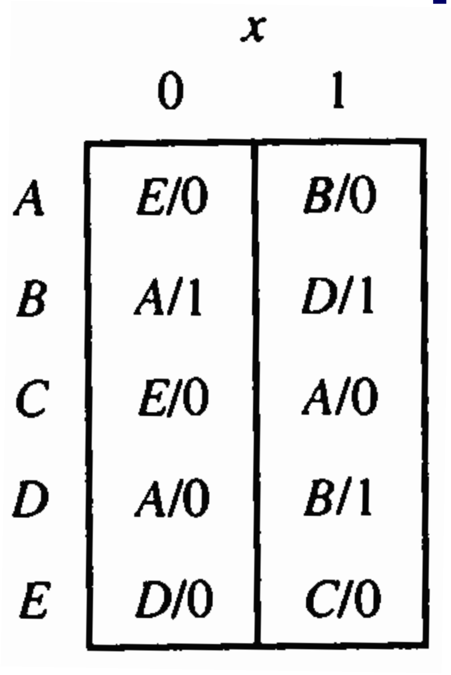
D_1



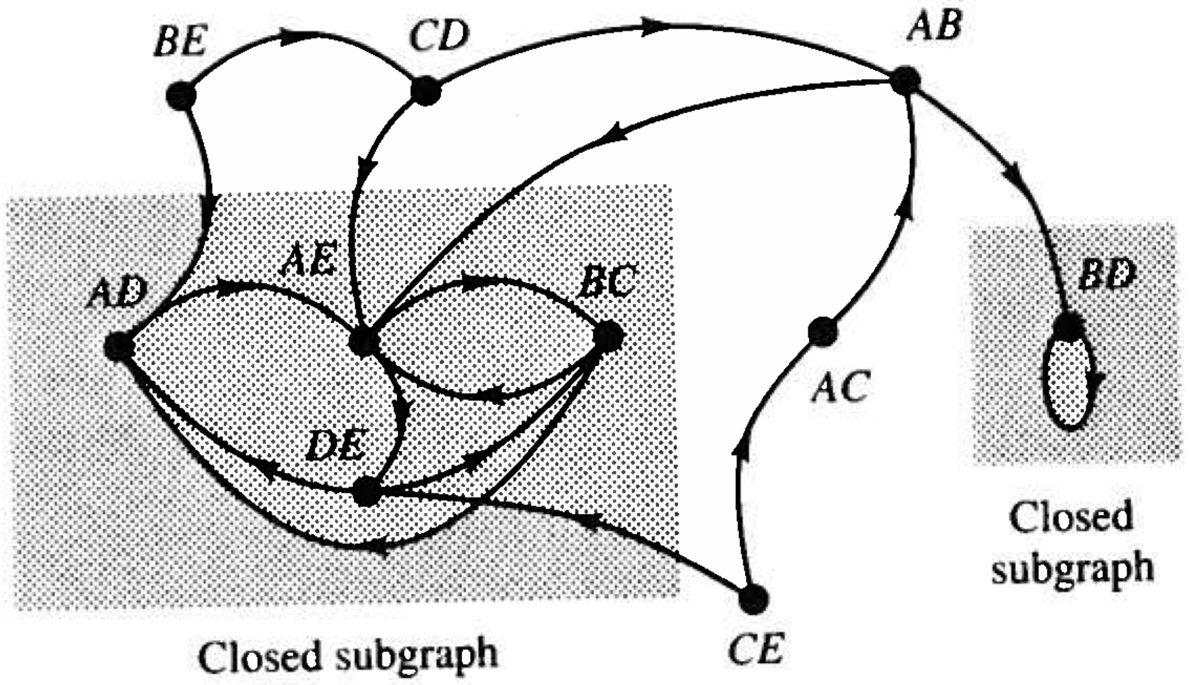
y_2y_1 \ x	0	1
00	0	0
01	1	0
11	1	1
10	0	1

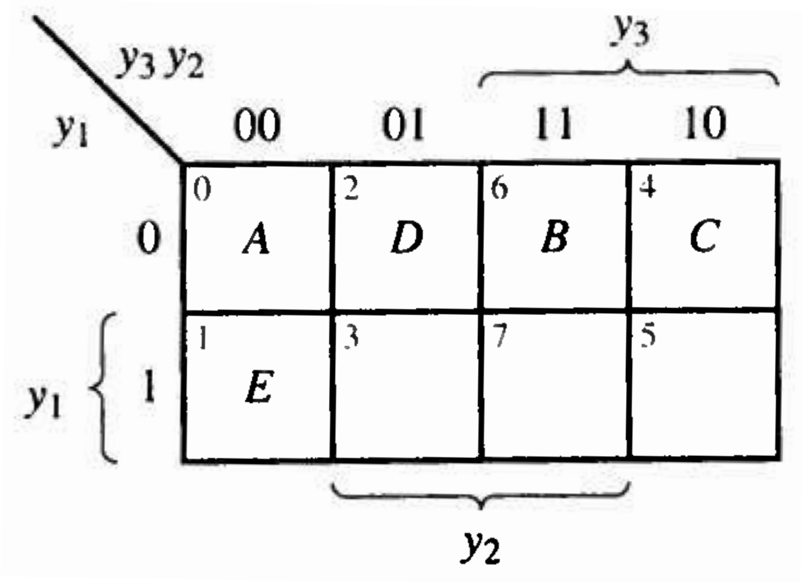
z

Example



	x = 0	x = 1
A	E/0	B/0
B	A/1	D/1
C	E/0	A/0
D	A/0	B/1
E	D/0	C/0





$y_1 \backslash y_3 y_2$	00	01	11	10
0	0: A	2: D	6: B	4: C
1	1: E	3	7	5

Rule 1: (A, C), (B, D), (A, D)

Rule 2: (A, D), (B, E), (A, B), (C, D), (A, E)

Rule 3: (A, D), (A, E), (B, C), (D, E), (B, D)

Rule 4: Assign all zeros combination to the most transferred to state (maximize the zeros in k-maps)

➢ **When cannot handle all, rules 1 and 3 are more important than 2**

Conclusion

- ***Summary***
  - ➔ Machine equivalence
    (state equivalence, distinguishable states)
  - ➔ State reduction
    (Distinguishing trees, state partitioning, implication table)
  - ➔ Examples
    (Sequence detector, Majority machine)
  - ➔ Optimal state code assignment
    (problem complexity, objectives, guidelines)
  - ➔ Code assignment heuristics
    (Implication graph and assignment rules)
- ***Next Lecture***
  - ➔ Review

Reading assignment: Sections 9.1, 9.2, 9.4.1-9.4.2 in the textbook