Counselling Code : VBIT

VIGNANA BHARATHI Institute of Technology

(A UGC Autonomous Institution, Approved by AICTE, Accredited by NBA & NAAC-A Grade, Affiliated to JNTUH)

UNIT - I

Minimization and Transformation of Sequential Machines

P.VIDYA SAGAR (ASSOCIATE PROFESSOR)

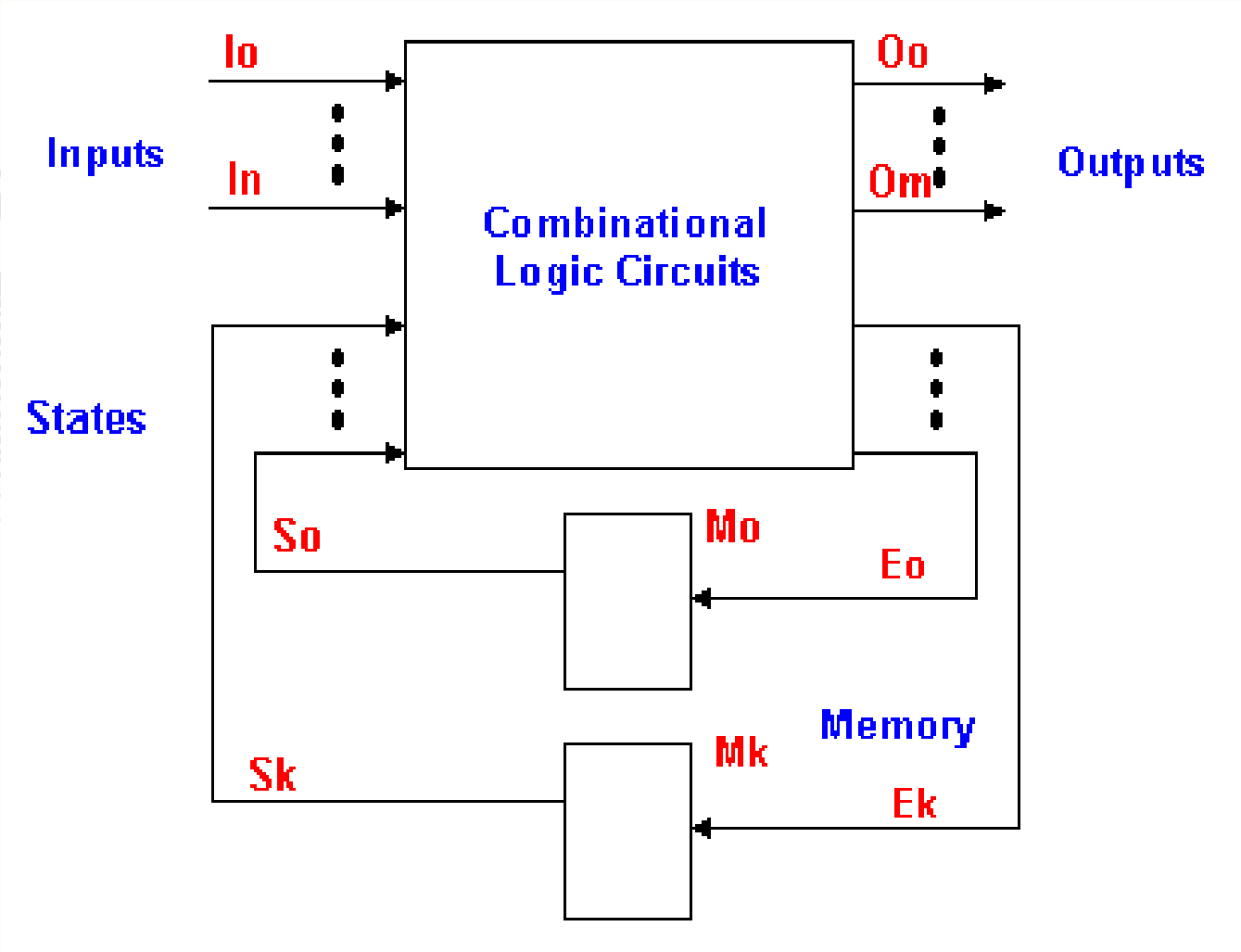
https://potharajuvidyasagar.wordpress.com

Sequential logic circuits

- The main characteristic of combinational logic circuits is that their output values depend on their present input values.
- Sequential logic circuits differ from combinational logic circuits because they contain memory elements so that their output values depend on both present and past input values
- Sequential circuits can be Asynchronous or synchronous.
  - Asynchronous sequential circuits change their states and output values whenever a change in input values occurs.
  - Synchronous sequential circuits change their states and output values at fixed points of time, i.e. clock signals.

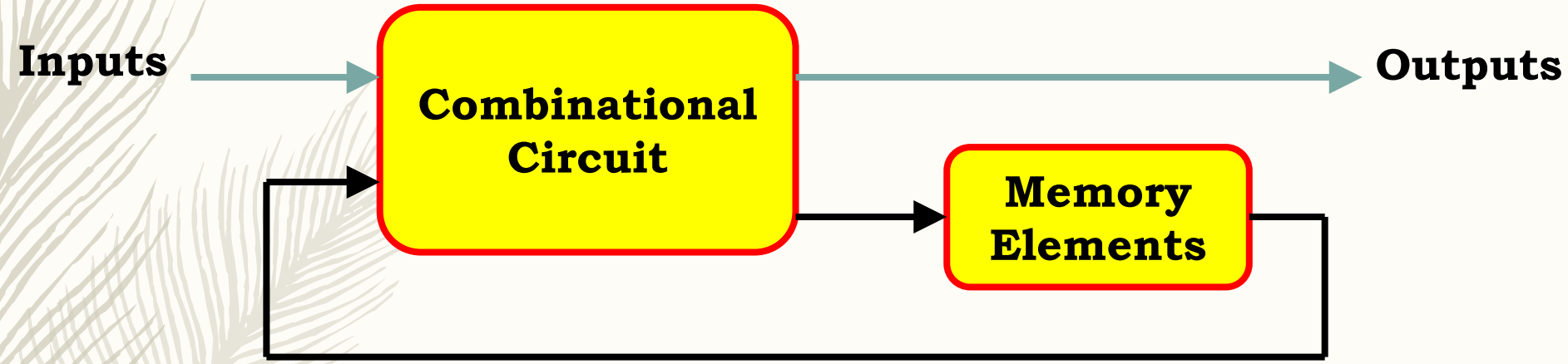
Sequential Circuit Models

Universal model

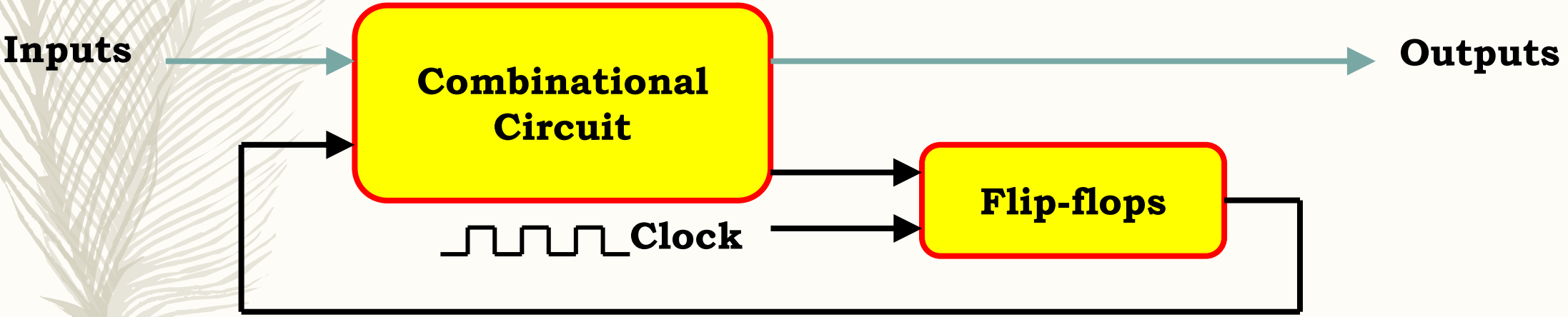


Sequential Circuits

- **Asynchronous**



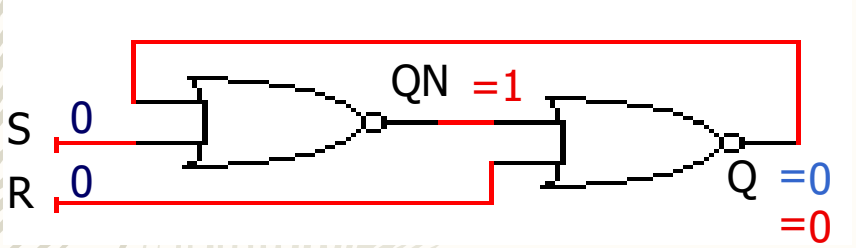
- **Synchronous**

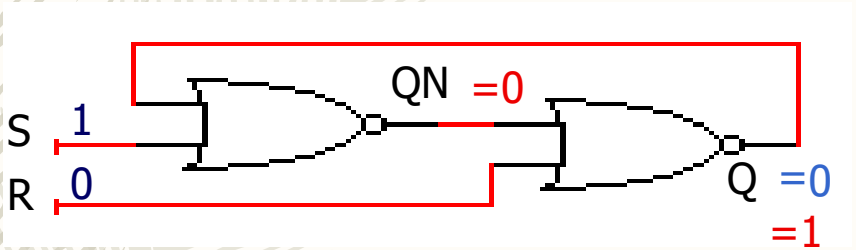


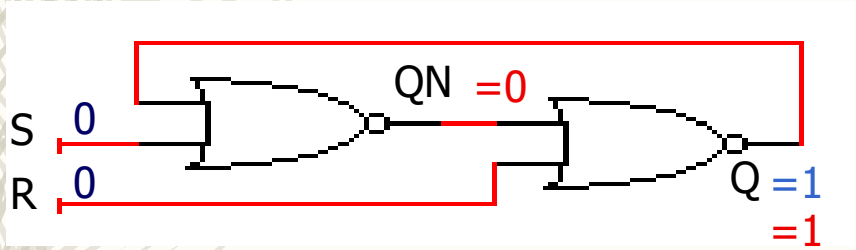
Synchronous Sequential Circuit

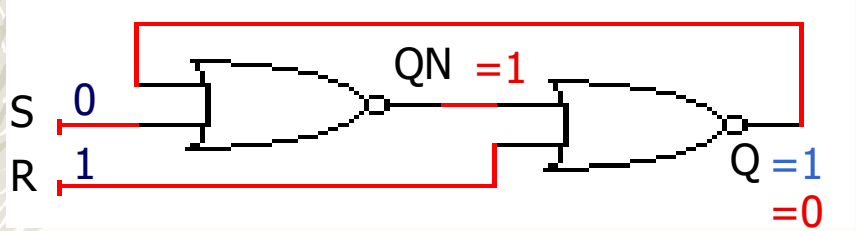
- The change of internal state occurs in response to the synchronized clock pulses.
- Input changes occur between clock pulses
- Data are read during the clock pulse
- It is supposed to wait long enough after the external input changes for all flip-flop inputs to reach a steady value before applying the new clock pulse
- Unsuitable Situations:
  - Inputs change at any time and cannot be synchronized with a clock
  - Circuit is large, Clock skew can not be avoided
  - High performance design

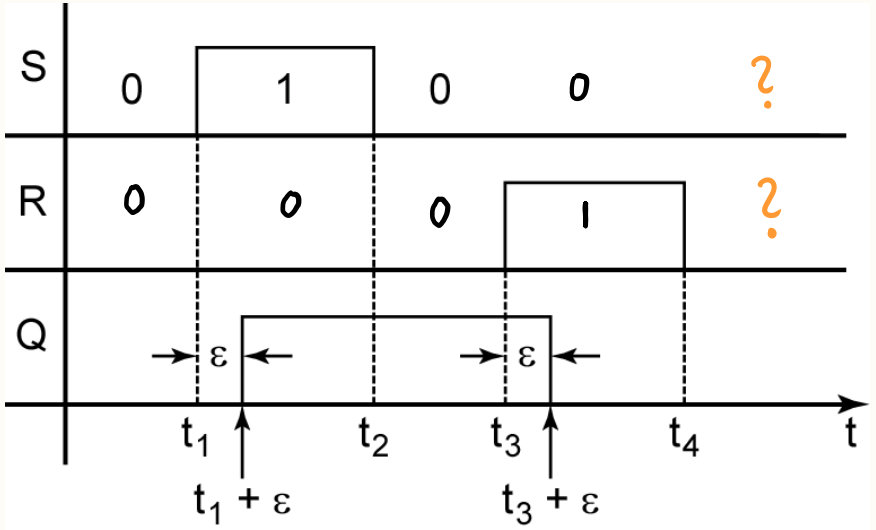
S-R latch operation









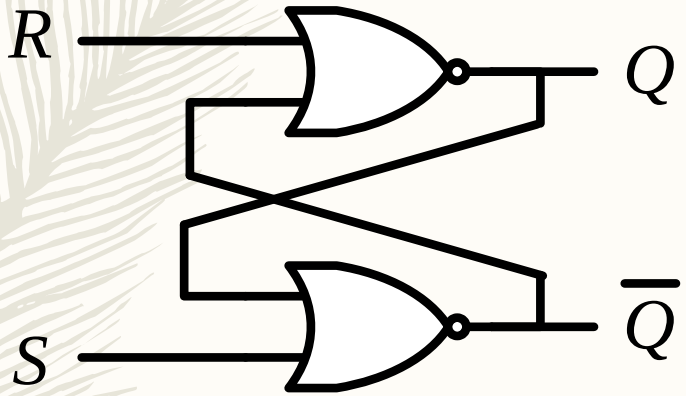


S(t)	R(t)	Q(t)	Q(t + ϵ)	QN (t + ϵ)
0	0	0	0	1
0	0	1	1	0
0	1	0	0	1
0	1	1	0	1
1	0	0	1	0
1	0	1	1	0
1	1	0		
1	1	1		

Still to analyze

Latches

- *SR* Latch



Present value		Next value	
S	R	Q	Q^+
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	not
1	1	1	allowed

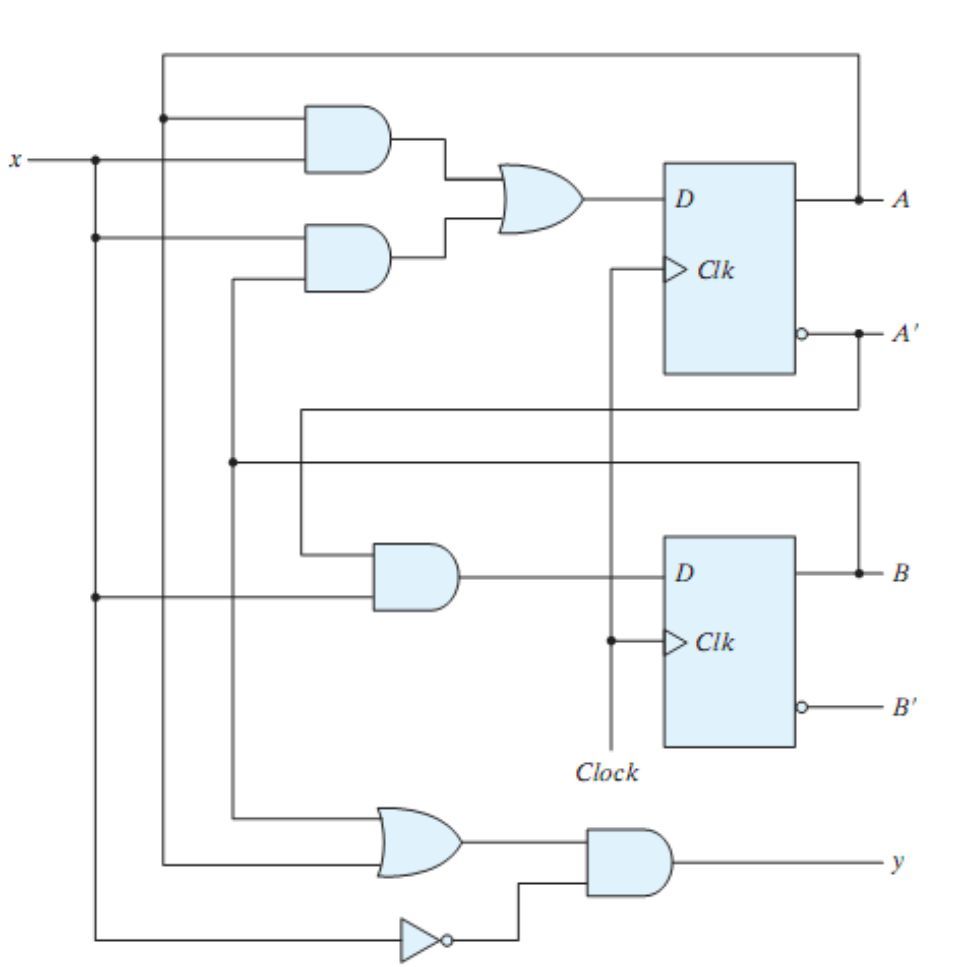
- If S = 1 (Set), $Q_+ = 1$
- If R = 1 (Reset), $Q_+ = 0$
- If S = R = 0, $Q_+ = Q$ (no change)
- S = R = 1 is not allowed.

State Equations

–A **state equation** is a Boolean expression which specifies the next state and output as a function of the present state and inputs.

–**Example:**

- The shown circuit has two D-FFs (A,B), an input x and output y.
- The D input of a FF determines the next state
  - A(t+1) = A(t)x+B(t)x = Ax+Bx
  - B(t+1) = A'(t)x = A'x
- Output:
  - y = (A+B)x'



State Table

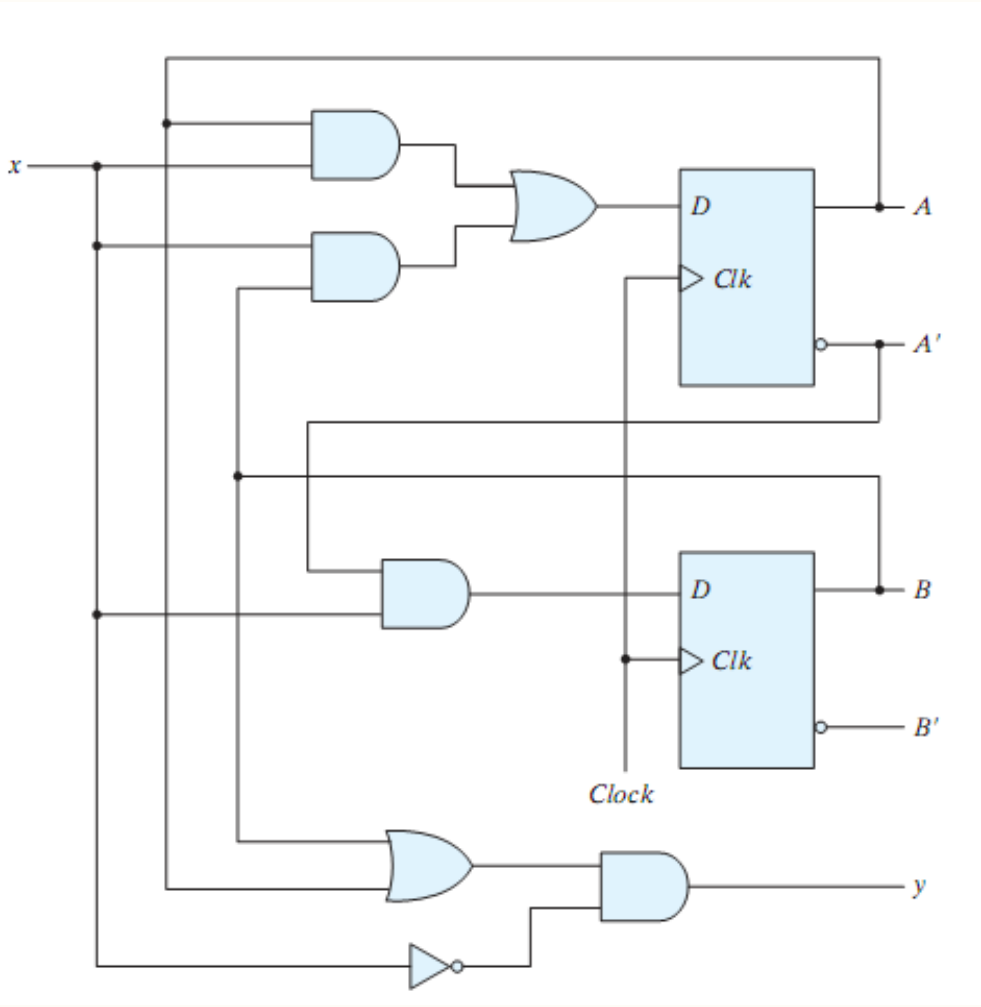
–A **state table** is a table enumerating all present states, inputs, next states and outputs.

•Present state, inputs: list all combinations

•Next states, outputs: derived from state equations

4 sections

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	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0



State Table

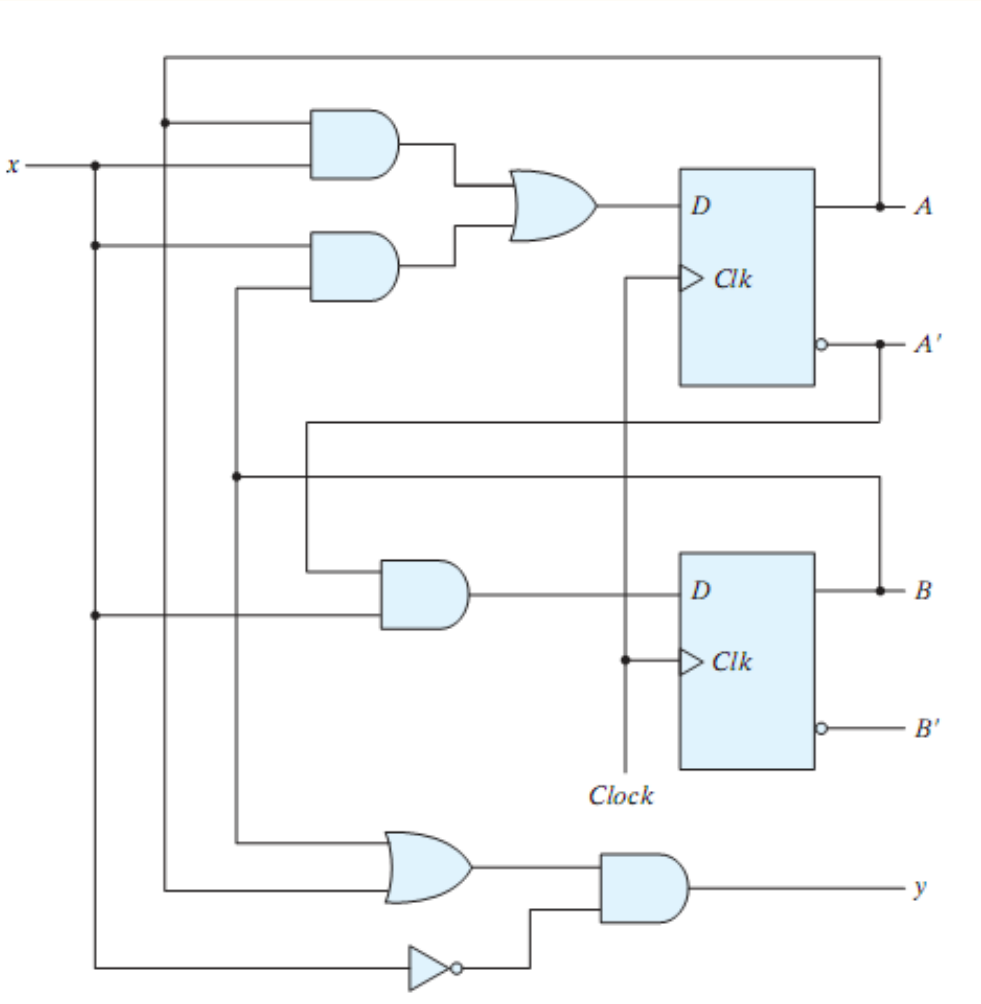
–A **state table** is a table enumerating all present states, inputs, next states and outputs.

•Present state, inputs: list all combinations

•Next states, outputs: derived from state equations

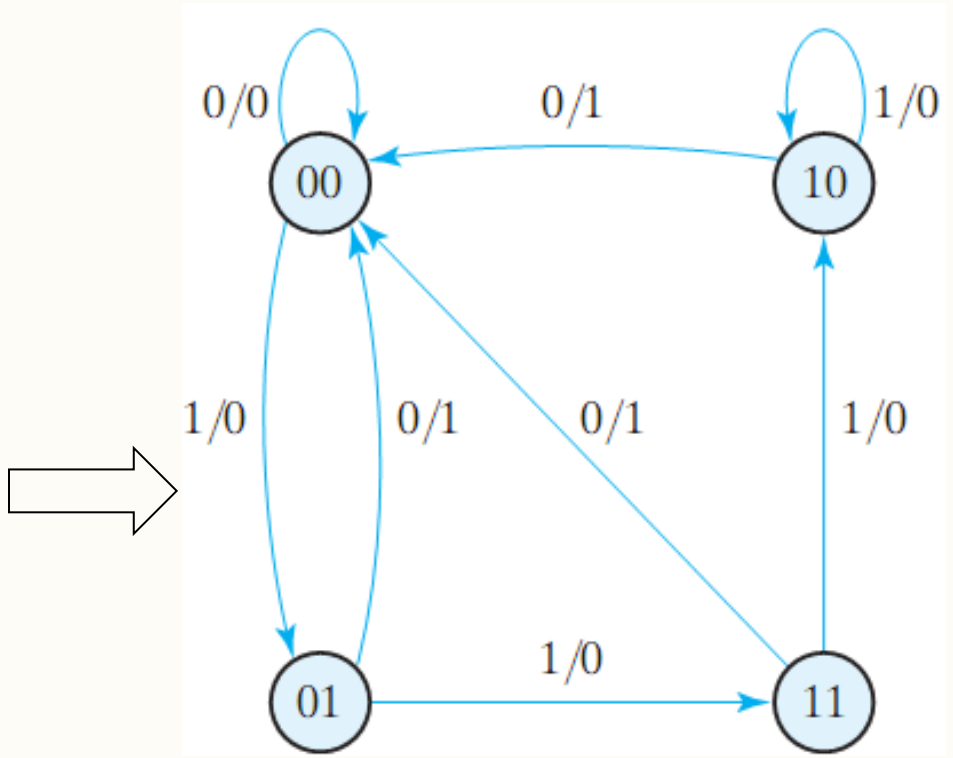
2-D Form

Present State		Next State				Output	
		$x = 0$		$x = 1$		$x = 0$	$x = 1$
A	B	A	B	A	B	y	y
0	0	0	0	0	1	0	0
0	1	0	0	1	1	1	0
1	0	0	0	1	0	1	0
1	1	0	0	1	0	1	0



State Diagram

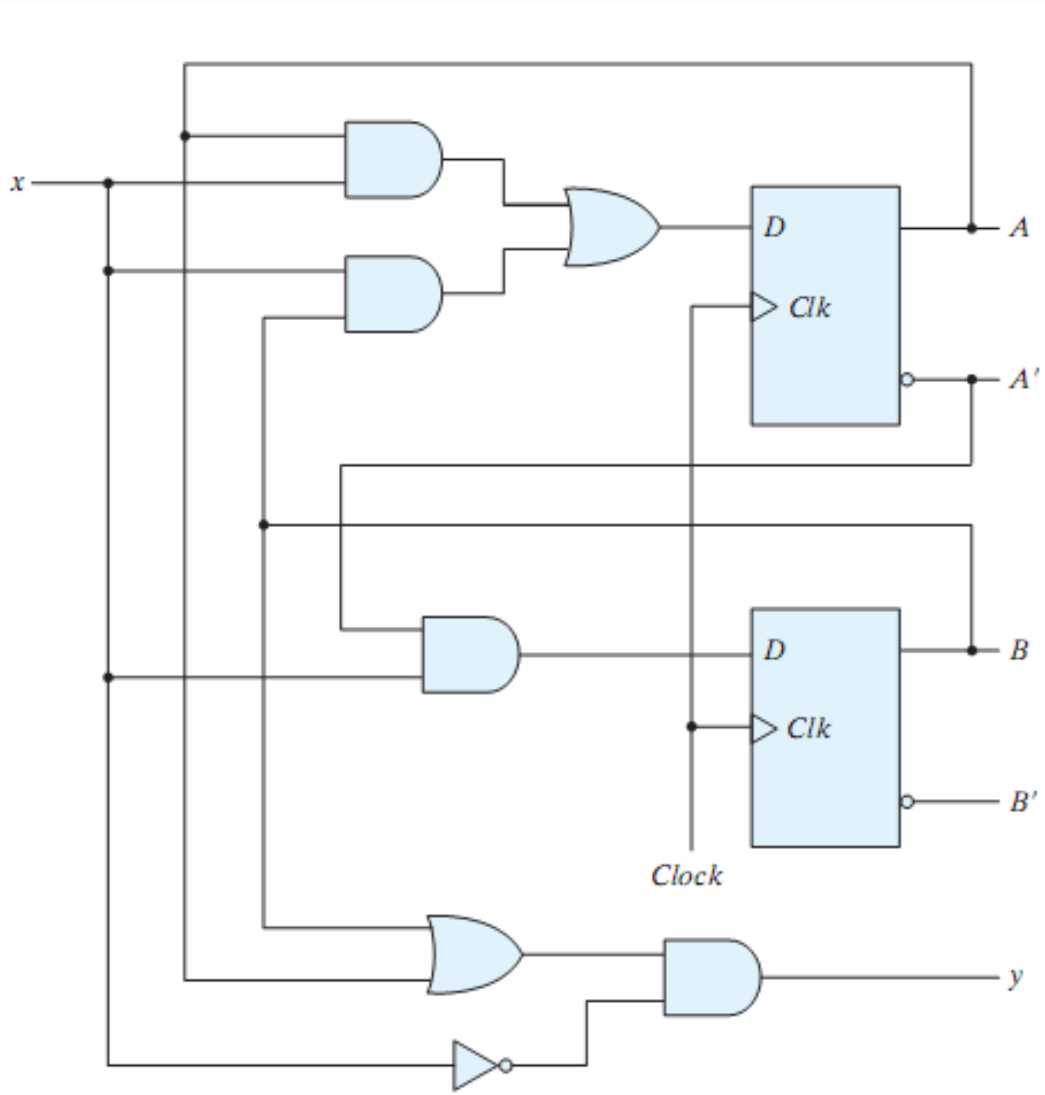
Present State		Input	Next State		Output
A	B	X	A	B	Y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0



- The **state diagram** is a graphical representation of a state table (provides same information)
- Circles are states (FFs), Arrows are transitions between states
- Labels of arrows represent inputs and outputs

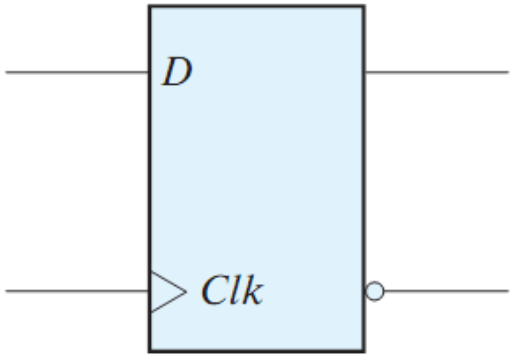
Example 1

- Analyze this circuit?
- Is this a sequential circuit? Why?
- How many inputs?
- How many outputs?
- How many states?
- What type of memory?

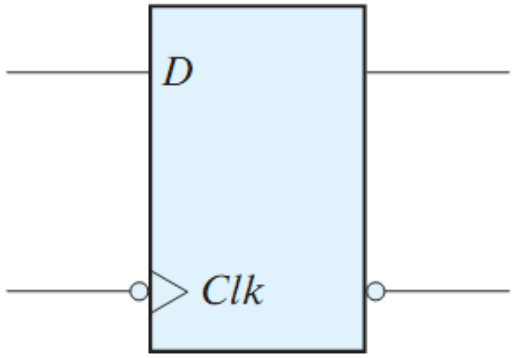


Example 1 (cont.)

D Flip Flop (review)



(a) Positive-edge



(a) Negative-edge

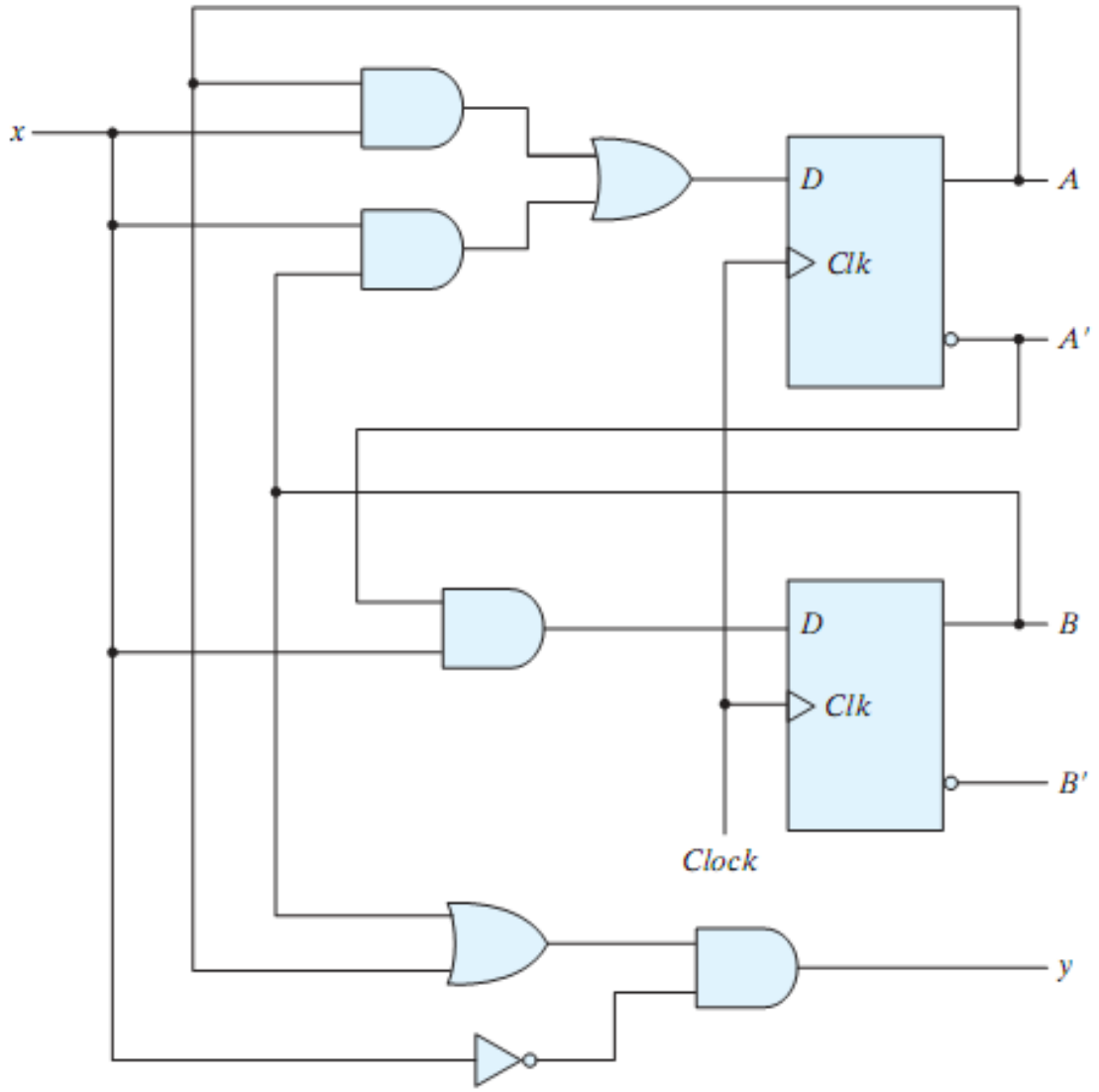
Characteristic Tables and Equations

Q(t)	D	Q(t+1)
0	0	0
0	1	1
1	0	0
1	1	1

D	Q(t+1)
0	0
1	1

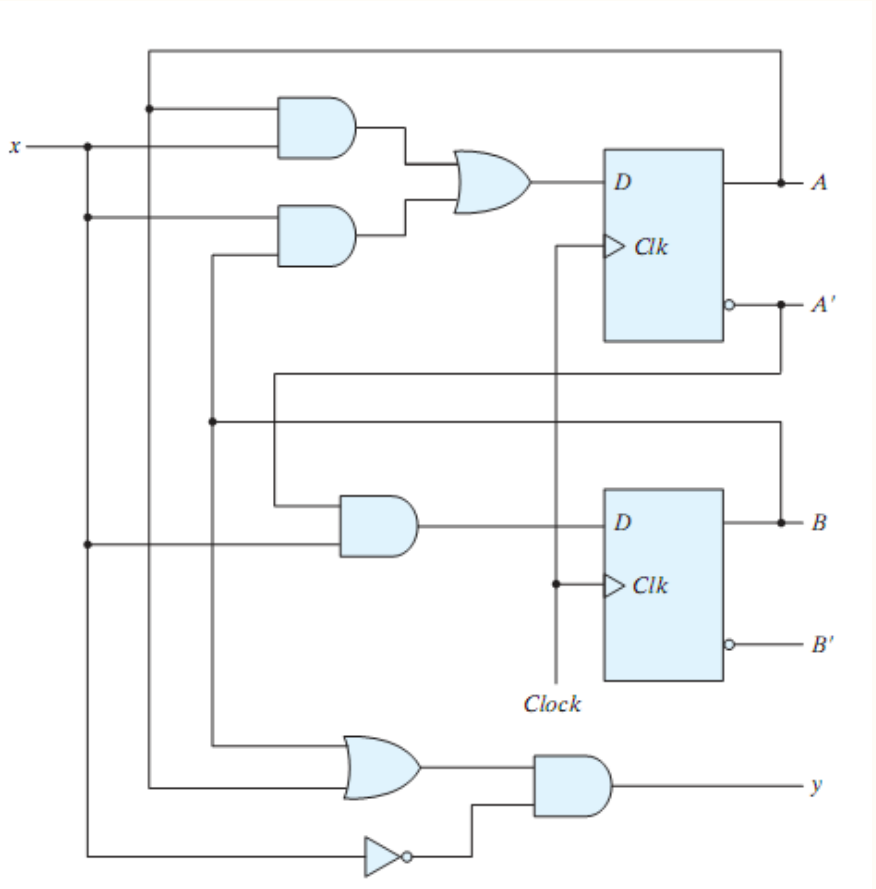
Q(t+1) = D

Example 1 (cont.)



Example 1 (cont.)

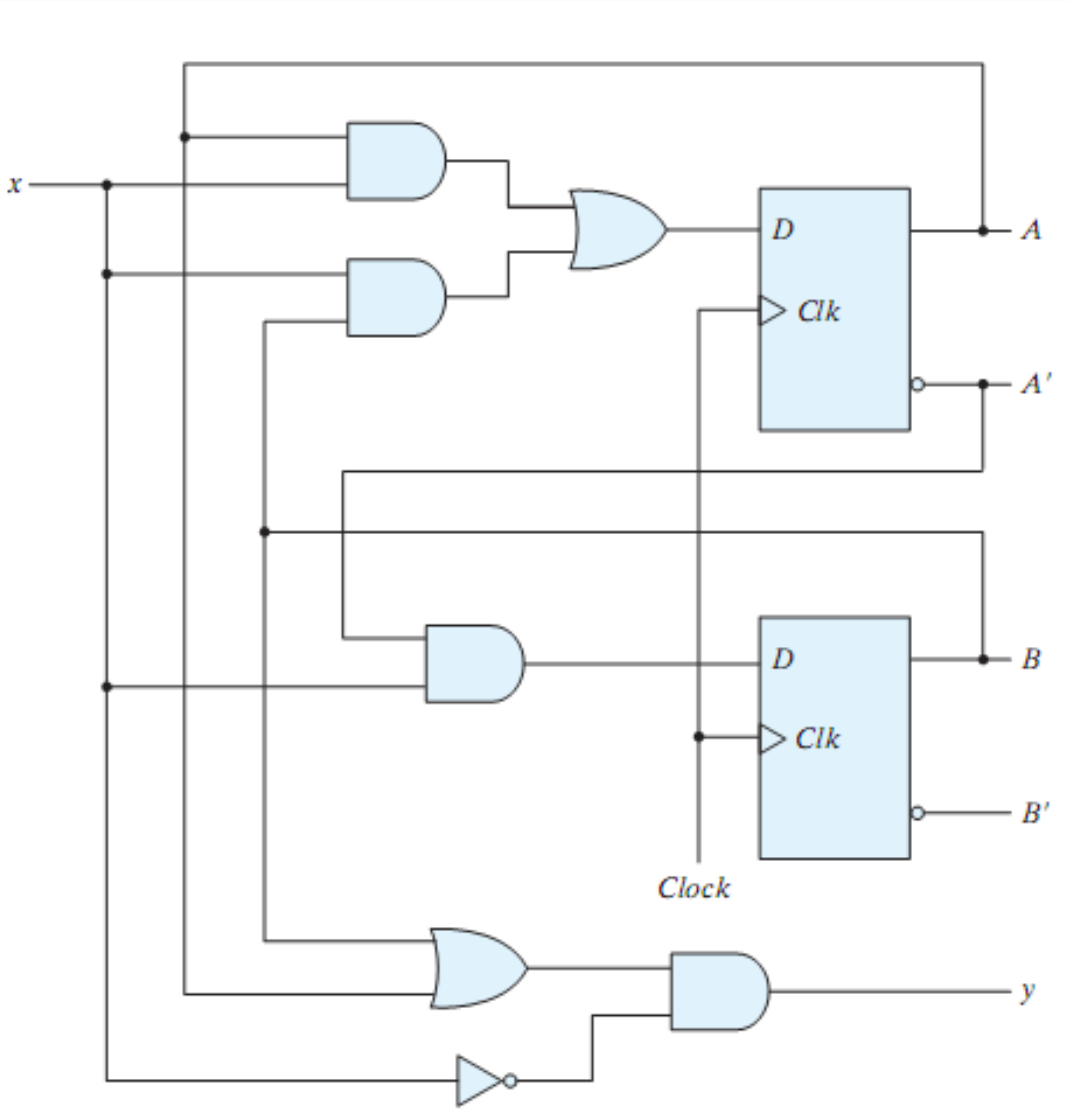
- **State equations:**
- $D_A = AX + BX$
- $D_B = A'X$
- $Y = (A + B)X'$



Example 1 (cont.)

- **State equations:**
- $D_A = AX + BX$
- $D_B = A'X$
- $Y = (A + B)X'$
- **State table:**

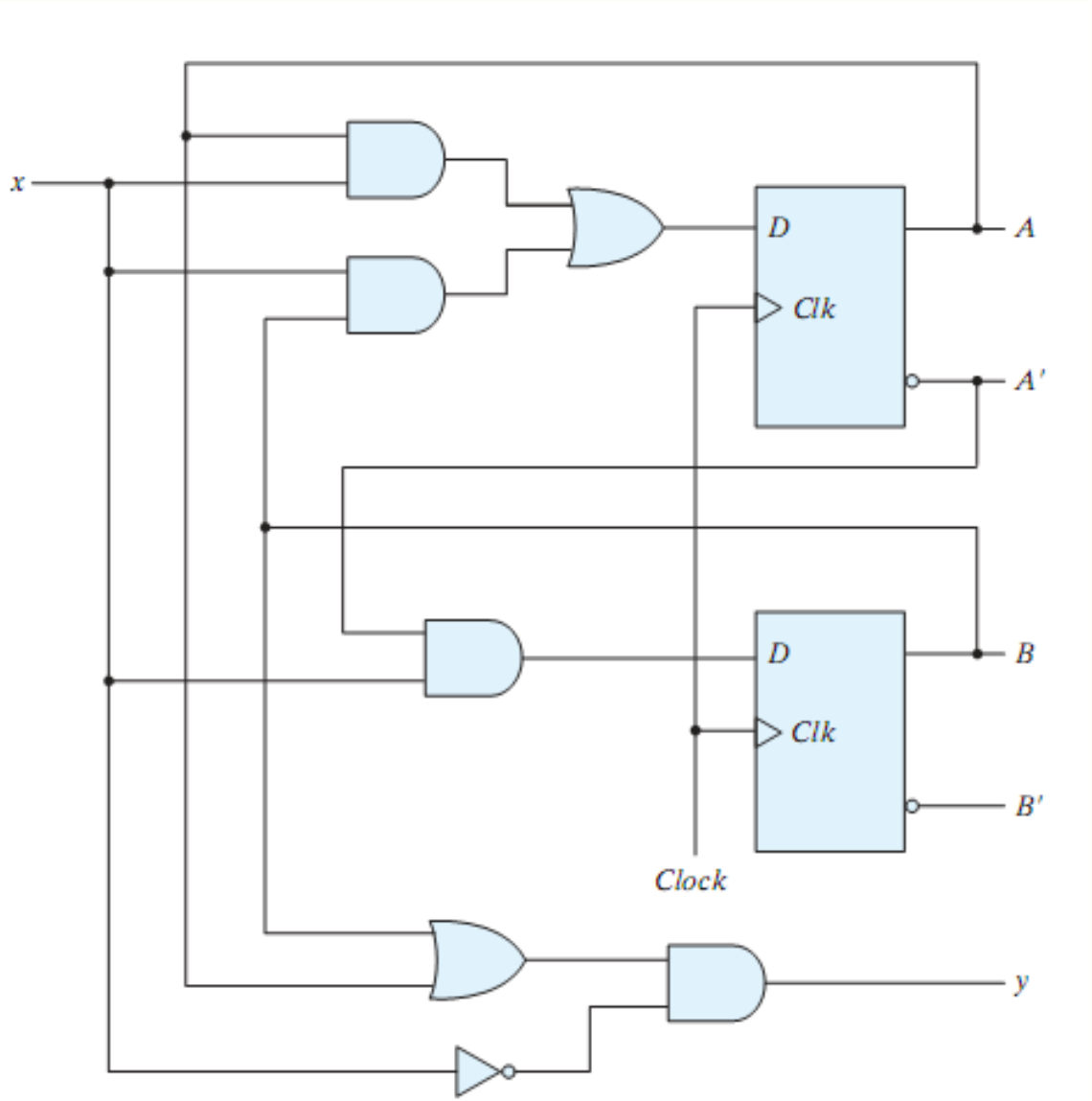
Present State		Input	Next State		Output
A	B	x	A	B	y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0



Example 1 (cont.)

- **State equations:**
- $D_A = AX + BX$
- $D_B = A'X$
- $Y = (A + B)X'$
- **State table (2D):**

Present State		Next State				Output	
		$x = 0$		$x = 1$		$x = 0$	$x = 1$
A	B	A	B	A	B	y	y
0	0	0	0	0	1	0	0
0	1	0	0	1	1	1	0
1	0	0	0	1	0	1	0
1	1	0	0	1	0	1	0

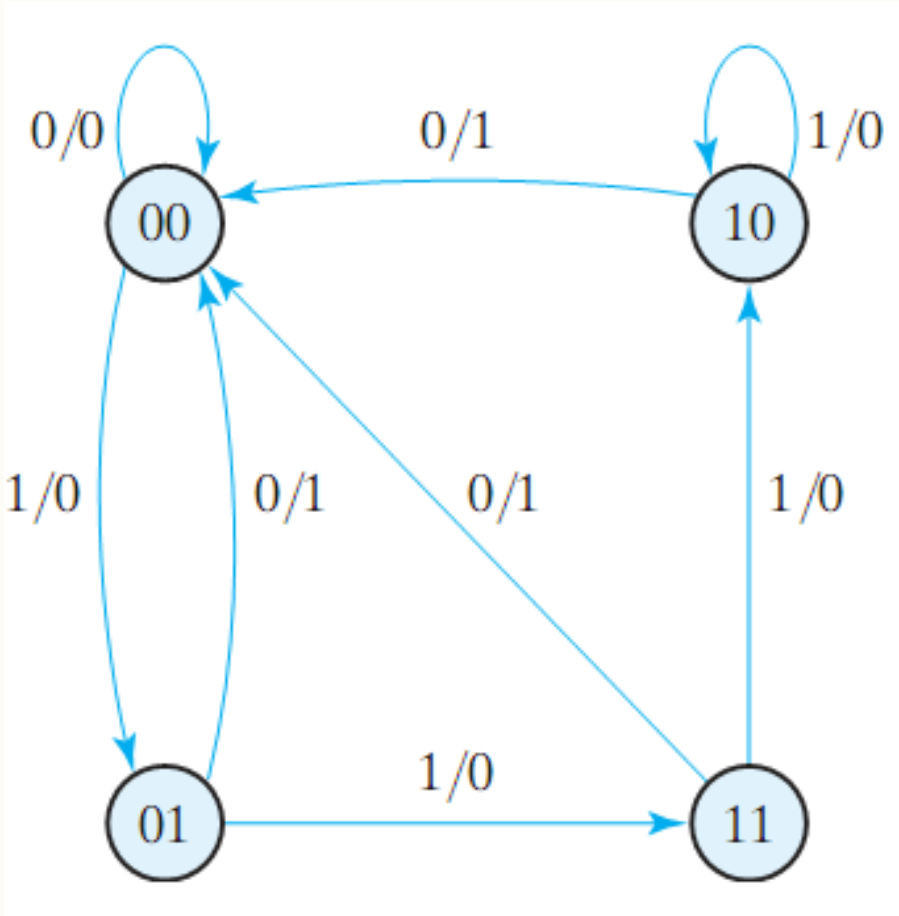


Example 1 (cont.)

- **State equations:**
- $D_A = AX + BX$
- $D_B = A'X$
- $Y = (A + B)X'$
- **State table:**

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	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0

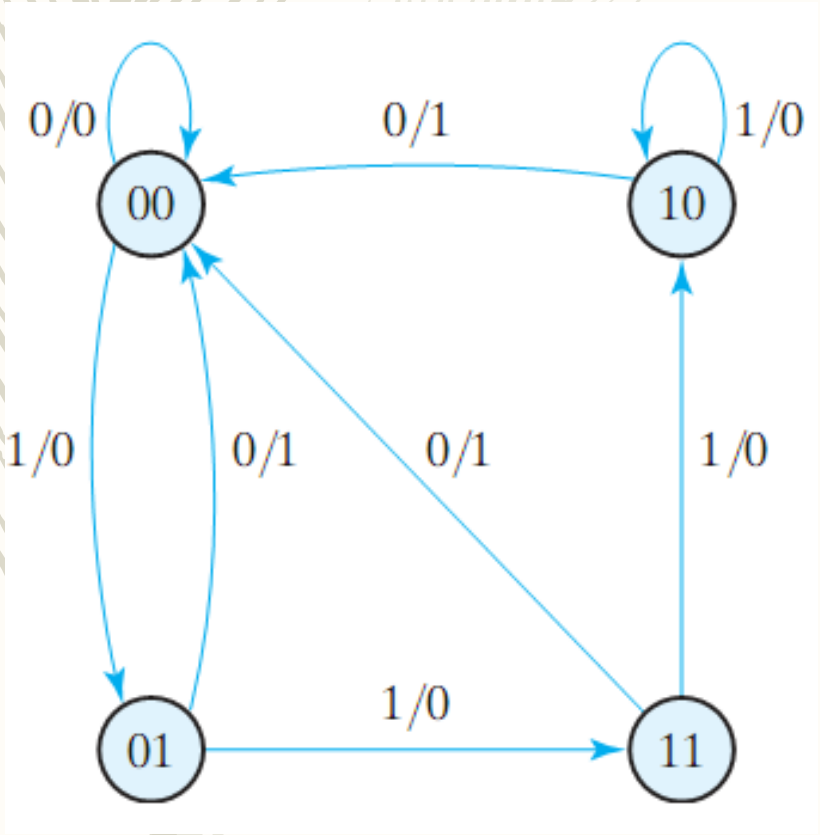
State diagram:



Mealy vs Moore Finite State Machine (FSM)

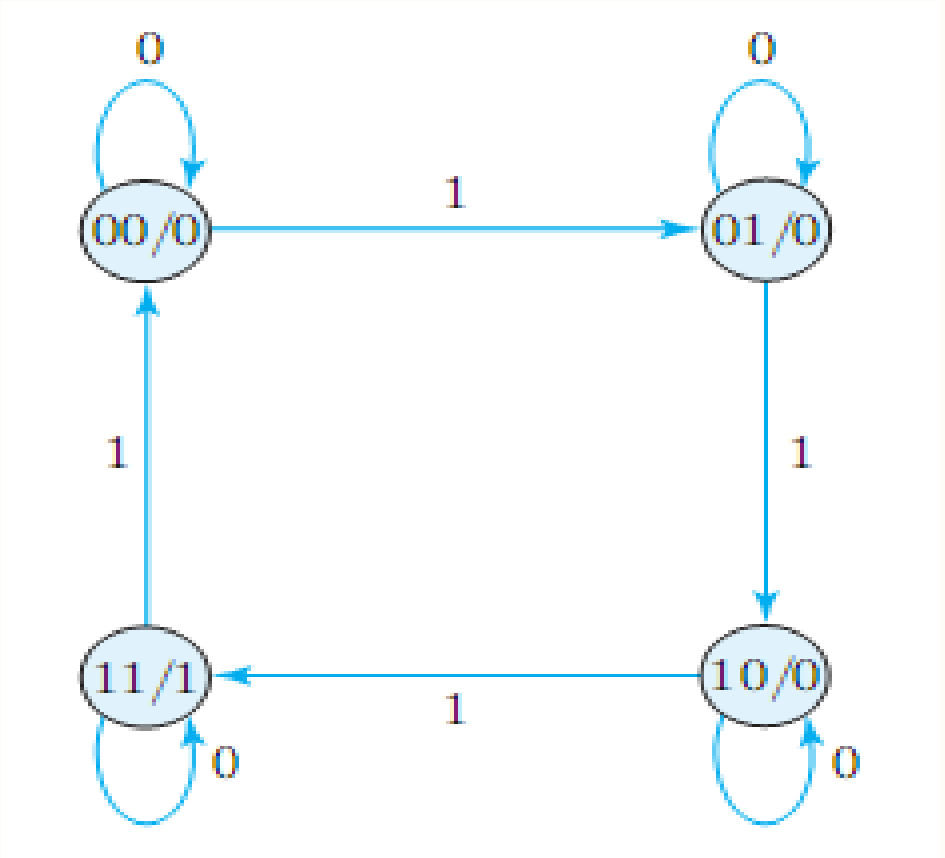
–Mealy FSM:

- Output depends on current state and input
- Output is not synchronized with the clock



–Moore FSM:

- Output depends on current state only



Sequential network as FSM (finite state machines)

- State minimization

  - *DEFINITION OF 1-SUCCESSOR : If the machine moves from state Si to state Sv when input w = 1, then we say that Sv is a 1-successor of Si*
  - *DEFINITION OF 0-SUCCESSOR : If the machine moves from state Sj to state Su when input w = 0, then we say that Su is a 0-successor of Si*
  - *DEFINITION: Two states Si and Sj are said to be equivalent if and only if for every input sequence , the same output sequence will be produced regardless of whether Si or Sj are the initial states.*

  - *IF STATES Si AND Sj ARE EQUIVALENT, THEN THEIR*
  - *CORRESPONDING K-SUCCESSORS (FOR ALL K) ARE THE SAME OR ARE ALSO EQUIVALENT.*

State equivalence and machine minimization

- In constructing the state diagram (or table) for a finite-state machine, it often happens that the diagram contains redundant states, i.e., states whose functions can be accomplished by other states. We note that the number of memory elements required for the realization of a machine is directly related to the number of states. (Recall that, for an n-state machine, $k = \log_2 n$ state variables are needed for an assignment.) Consequently, the minimization of the number of states does reduce the complexity and cost of the realization in many cases.
- It is, therefore, desirable to develop techniques for transforming a given machine into another machine that has no redundant states, such that both have the same terminal behavior.

Equivalent State & distinguishable Definitions

- Two states are equivalent if their response for each possible input sequence is an identical output sequence.
- Alternatively, two states are equivalent if their outputs produced for each input symbol is identical and their next states for each input symbol are the same or equivalent.
- Two states that are not equivalent are distinguishable
- Two states, Si and Sj , of a machine M are distinguishable if and only if there exists at least one finite input sequence that, when applied to M, causes different output sequences depending on whether Si or Sj is the initial state.
- The sequence that distinguishes these states is called a distinguishing sequence for the pair (Si, Sj).

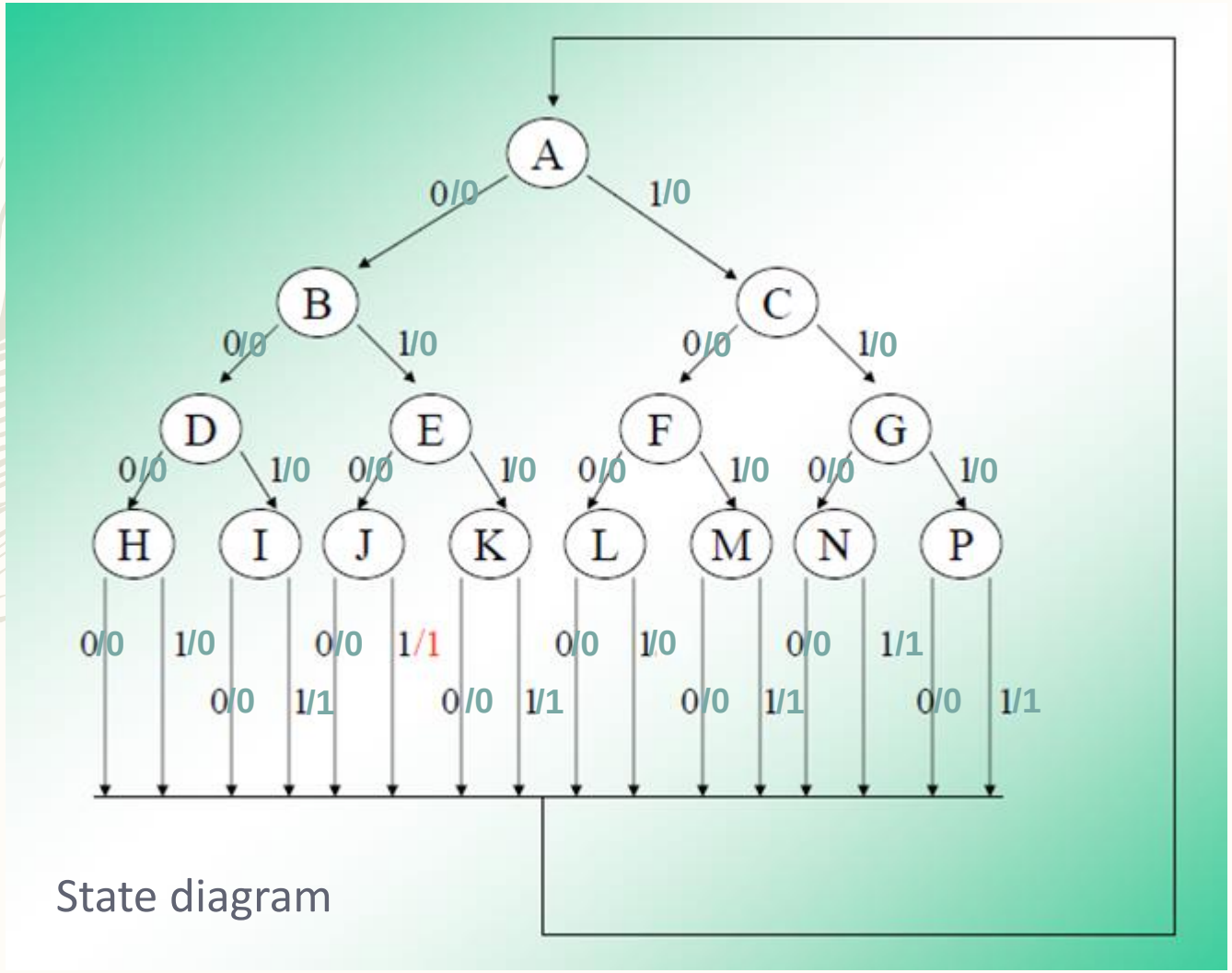
(machine minimization)State Reduction:

- The three main methods of state reduction include:
- 1. row matching,
- 2. implication charts, and
- 3. Successive partitioning.

1.Row matching, which is the easiest of the three, works well for state transition tables which have an obvious next state and output equivalences for each of the present states. This method will generally not give the most simplified state machine available, but its ease of use and consistently fair results is a good reason to pursue the method. The implication chart uses a graphical grid to help find any implications or equivalences and is a great systematic approach to reducing state machines. Successive partitioning is almost a cross between row matching and implication chart where both a graphical table and equivalent matching is used.

- Each of these methods will, in most cases, reduce a state machine into a smaller number of states. Keep in mind that one method may result in a simpler state machine than another.

Elimination of Redundant States



State diagram

Elimination of Redundant States

- When first setting up the state table, we will not be overly concerned with inclusion of extra states, and when the table is complete, we will eliminate any redundant states.

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	1
010	*J*	*A*	*A*	0	1
011	*K*	*A*	*A*	0	1
100	*L*	*A*	*A*	0	0
101	*M*	*A*	*A*	0	1
110	*N*	*A*	*A*	0	1
111	*P*	*A*	*A*	0	1

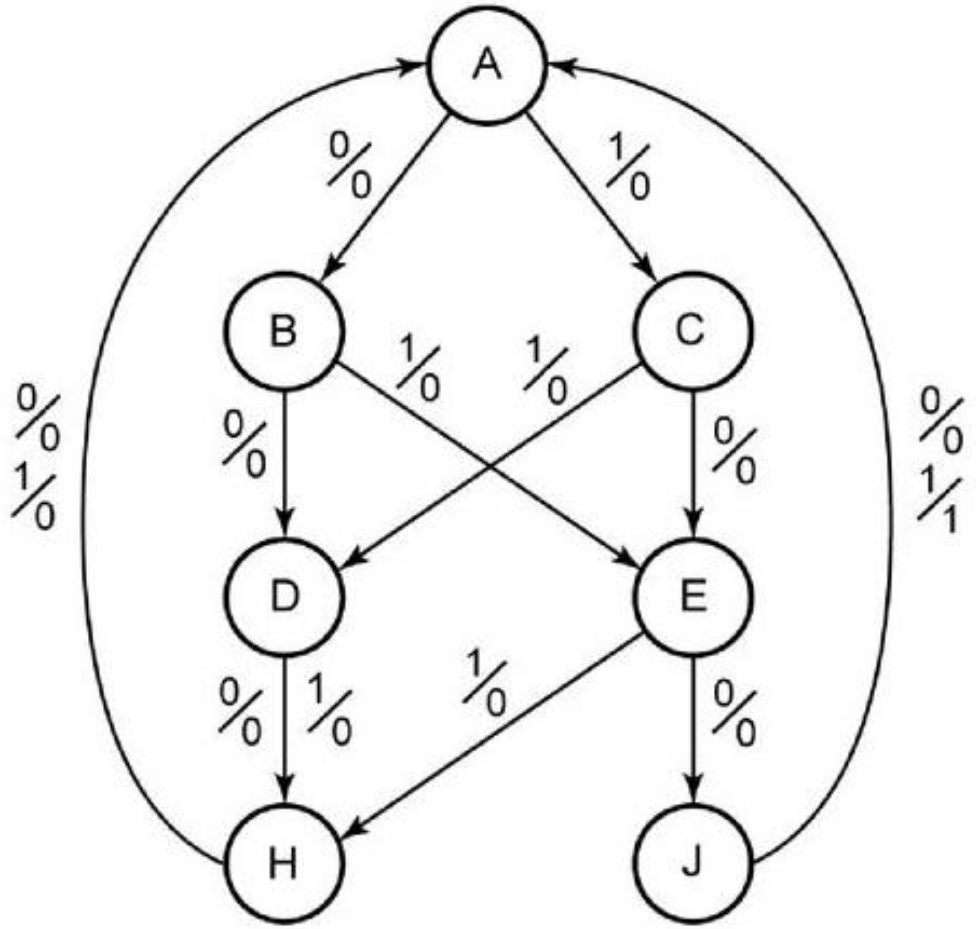
Since states *H* and *I* have the same next states and the same outputs, there is no way of telling states *H* and *I* apart.

We can replace *I* with *H*.

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~~

Present State	Next State $X=0$	Next State $X=1$	Output $X=0$	Output $X=1$
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

Reduced State Table



Reduced State Graph

Implication Table

- Two states are equivalent if their response for each possible input sequence is an identical output sequence.
- Alternatively, two states are equivalent if their outputs produced for each input symbol is identical and their next states for each input symbol are the same or equivalent.
- Two states that are not equivalent are distinguishable

- The checking of each pair of states for possible equivalence in a table with a large number of states can be done systematically on an Implication Table.
- The Implication Table is a chart that consists of squares, one for every possible pair of states.

Equivalent States

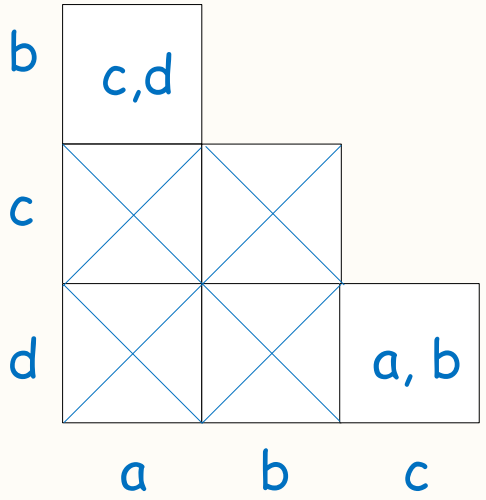
- completely specified state table

If c and d are equivalent, then
a and b are equivalent (a,b)⇨(c,d)

Present State	Next State		Output	
	X=0	X=1	X=0	X=1
a	*c*	*b*	0	1
b	*d*	*a*	0	1
c	*a*	*d*	1	0
d	*b*	*d*	1	0

(c,d)⇨(a,b) ⟹ (c,d) && (a,b)

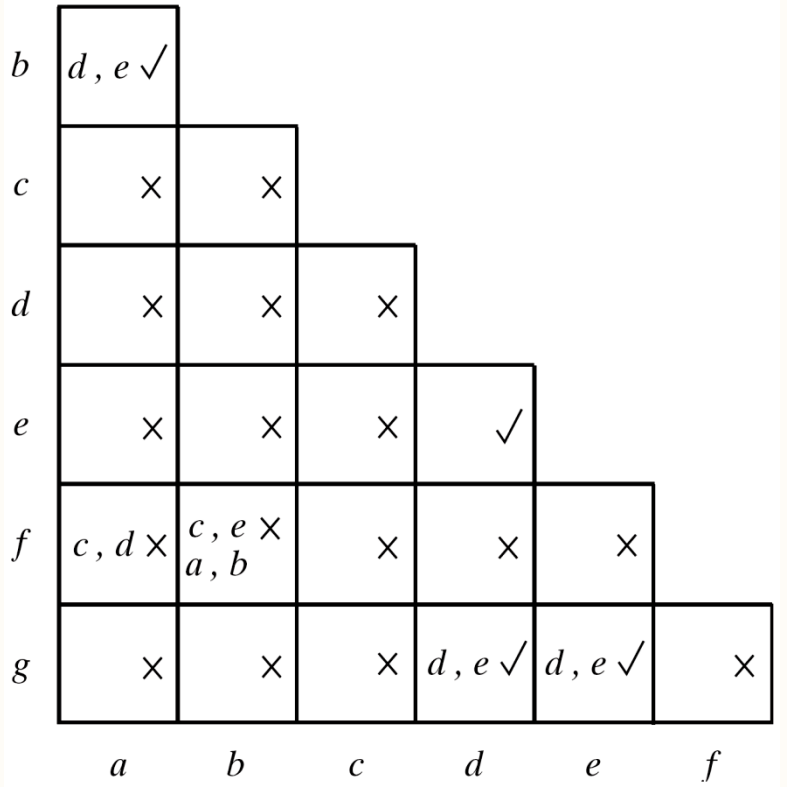
both pairs are equivalent



Implication Table (Example):

- Place a cross in any square corresponding to a pair whose outputs are not equal
- Enter in the remaining squares the pairs of states that are implied by the pair of states representing the squares. (Start form the top square in the left column and going down and then proceeding with the next column to the right).
- Make successive passes through the table to determine whether any additional squares should be marked with a 'x'.
- Finally, all the squares that have no crosses are
-

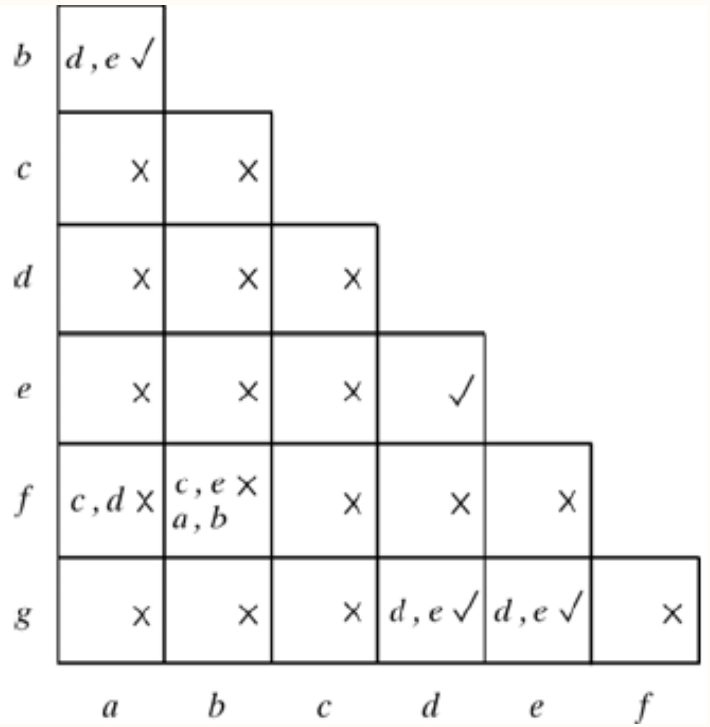
Present State	Next State		Output	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
a	d	b	0	0
b	e	a	0	0
c	g	f	0	1
d	a	d	1	0
e	a	d	1	0
f	c	b	0	0
g	a	e	1	0



Implication Table (Example):

- Its clear that (e,d) are equivalent. And this leads (a,b) and (e,g) to be equivalent too.
- Finally we have [(a,b) , c , (e,d,g) , f] so four states.
- So the original flow table can be reduced to:

Present State	Next State		Output	
	X=0	X=1	X=0	X=1
a	*d*	*a*	0	0
c	*d*	*f*	0	1
d	*a*	*d*	1	0
f	*c*	*a*	0	0



Partitioning Minimization Procedure
Successive partitioning

- PROCEDURE:
  - *1) all states belong to the initial partition p1*
  - *2) p1 is partitioned in blocks such that the states in each block generate the same output.*
  - *3) continue to perform new partitions by testing whether the k-successors of the states in each block are contained in one block. Those states whose k-successors are in different blocks cannot be in one block.*
  - *4) prcedure ends when a new partition is the same as the previous partition*

Finding Equivalent States by Partitioning

	Partition blocks				Action
Partition P_0	$(ABCDE)$				
Output for $x = 0$	11100				Separate (ABC) and (DE)
Output for $x = 1$	00011				Separate (ABC) and (DE)
Partition P_1	(ABC)		(DE)		
Next state for $x = 0$	CCB		DE		
Next state for $x = 1$	BEE		BA		Separate (A) and (BC)
Partition P_2	(A)	(BC)	(DE)		
Next state for $x = 0$	C	CB	DE		
Next state for $x = 1$	B	EE	BA		Separate (D) and (E)
Partition P_3	(A)	(BC)	(D)	(E)	
Next state for $x = 0$	C	CB	D	E	
Next state for $x = 1$	B	EE	B	A	
Partition $P_4 = P_3$	(A)	(BC)	(D)	(E)	

States B and C are equivalent.

Partitioning examples

$$P_1 = (AD)(BE)(CF)(GH)$$
$$P_2 = (AD)(BE)(CF)(G)(H)$$
$$P_3 = P_2$$

$$P_1 = (ACG)(BDEH)(F)$$
$$P_2 = (A)(CG)(BH)(DE)(F), \quad \text{from column } x = 0$$
$$= (A)(C)(G)(BH)(DE)(F), \quad \text{from column } x = 1$$
$$P_3 = P_2$$

	x = 0	x = 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)

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

(a)

Yet another partitioning example

	x_1x_2 = 00	01	11	10
A	D/0	D/0	F/0	A/0
B	C/1	D/0	E/1	F/0
C	C/1	D/0	E/1	A/0
D	D/0	B/0	A/0	F/0
E	C/1	F/0	E/1	A/0
F	D/0	D/0	A/0	F/0
G	G/0	G/0	A/0	A/0
H	B/1	D/0	E/1	A/0

(a)

$$P_1 = (ADFG)(BCEH)$$
$$P_2 = (AFG)(D)(BCEH)$$
$$P_3 = (AF)(G)(D)(BCH)(E)$$
$$P_4 = P_3$$

Incompletely specified circuits: partition method

- The partitioning minimization procedure which was applied to completely specified state tables can also be applied to incompletely specified state tables.
- To perform the partitioning process, we can assume that the unspecified outputs have a specific value.
- The partitioning method is equally applicable to Mealy type FSMs in the same way as for Moore-type FSMs.

Partition minimization method

Example of incompletely specified FSM.

Present state	Next state		Output z	
	$w = 0$	$w = 1$	$w = 0$	$w = 1$
A	B	C	0	0
B	D	–	0	–
C	F	E	0	1
D	B	G	0	0
E	F	C	0	1
F	E	D	0	1
G	F	–	0	–

Let consider both unspecified outputs = 0

P_1=(*ABCDEFG*)

P_2=(*ABDG*)(*CEF*)

P_3=(*AB*)(*D*)(*G*)(*CEF*)

P_4=(*A*)(*B*)(*D*)(*G*)(*CE*)(*F*)

P_5= P_4

Let consider both unspecified outputs = 1

P_1=(*ABCDEFG*)

P_2=(*AD*)(*BCEFG*)

P_3=(*AD*)(*B*)(*CEG*)(*F*)

P_4=(*AD*)(*B*)(*CEG*)(*F*)

Merger Chart Methods: Merger graphs

The merger graph is a state reducing tool used to reduce states in the incompletely specified machine. The merger graph is defined as follows.

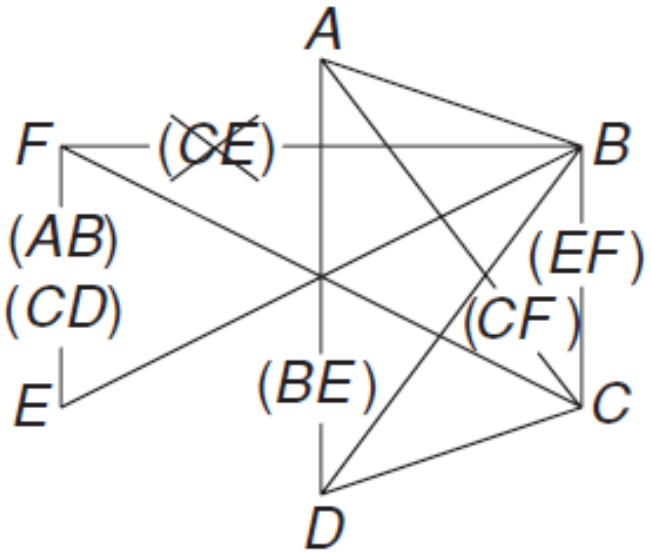
1. Each state in the state table is represented by a vertex in the merger graph. So it contains the same number of vertices as the state table contains states.
2. Each compatible state pair is indicated by an unbroken line draw between the two state vertices
3. Every potentially compatible state pair with non-conflicting outputs but with different next states is connected by a broken line. The implied states are written in theline break between the two potentially compatible states.
4. If two states are incompatible no connecting line is drawn.

Consider a state table of an incompletely specified machine shown in fig. the corresponding merger graph shown in fig.**State table:**

For the machine *M*, the merger graph reveals the existence of nine compatible pairs:
(*AB*), (*AC*), (*AD*), (*BC*), (*BD*), (*BE*), (*CD*), (*CF*), (*EF*)

Moreover, since (*AB*), (*AC*), and (*BC*) are compatibles then (*ABC*) is also a compatible

	NS, z			
PS	I_1	I_2	I_3	I_4
A	—	*C*, 1	*E*, 1	*B*, 1
B	*E*, 0	—	—	—
C	*F*, 0	*F*, 1	—	—
D	—	—	*B*, 1	—
E	—	*F*, 0	*A*, 0	*D*, 1
F	*C*, 0	—	*B*, 0	*C*, 1



In Fig.the set of highest-order polygons are the tetragon (*ABCD*) and the arcs (*CF*), (*BE*), and (*EF*). Generally, after a complete polygon of order *n* has been found, all polygons of order *n* − 1 contained in it can be ignored. Consequently, the triangles (*ABC*), (*ACD*), etc., are not considered.

In order to find a minimal set of compatibles, which covers the original machine and can be used as a basis for the construction of a minimal machine, it is often useful to find the set of maximal compatibles. Recall that a compatible is maximal if it is not contained in any other compatible. In terms of the merger graph, we are looking for complete polygons that are not contained within any higher-order complete polygons.

The closed sets of compatibles

- Consider the set of compatibles {(*ABCD*), (*EF*)} of machine *M*. Since this is the minimal number of compatibles covering all the states of *M*, it defines a *lower bound* on the number of states in the minimal machine that covers *M*. However, if we select the maximal compatible (*ABCD*) to be a state in the reduced machine, its *I*2- and *I*3-successors, (*CF*) and (*BE*), respectively, must also be selected. Since none of these compatible pairs is contained in the above set the lower bound cannot be achieved, and the set of maximal compatibles {(*ABCD*), (*EF*)} cannot be used to define the states of a minimal machine that covers *M*. {(*ABCD*), (*BE*), (*CF*), (*EF*)}.

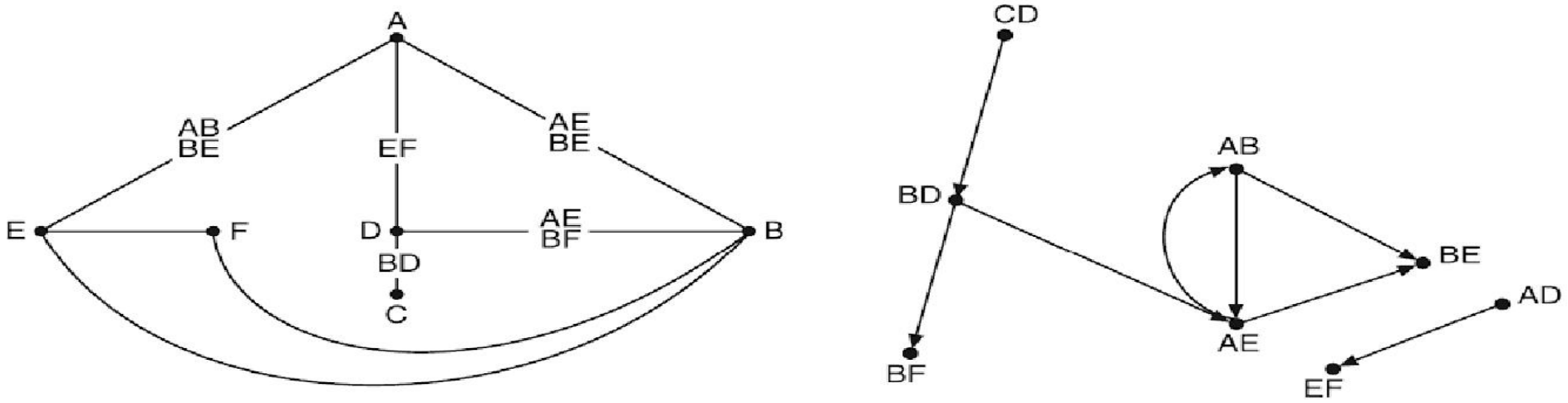
closed covering

- set of compatibles (for a machine *M*) is said to be *closed* if, for every compatible contained in the set, all its implied compatibles are also contained in the set. A closed set of compatibles that contains all the states of *M* is called a *closed covering*.
- **Example** For *M*6, the set {(*AD*), (*BE*), (*CD*)} is closed. The set {(*AB*),(*CD*), (*EF*)} is a closed covering.

compatibility graph

Step 1. *Identify and draw vertices:* Refer to the reduced merger graph shown in Figure . Mark the vertices corresponding to all compatible pairs. In the merger graph, each compatible state pair is indicated by a line drawn between the two state vertices. For example, in the merger graph shown in Figure ′ a, there are lines between A and B, A and D, A and E, B and D, B and E, B and F, C and D, and E and F. This gives us eight vertices—AB, AD, AE, BD, BE, BF, CD, and EF in the compatibility graph shown in Figure b.

Step 2. *Draw arcs:* Draw the arcs lead from vertex (S_i, S_j) to vertex (S_p, S_q) if and only if the compatible pair (S_i, S_j) implies the pair (S_p, S_q). The implied pairs are shown in the merger graph of Figure a. The compatible pair AB implies (AE) and (BE), AD implies (EF), AE implies (AB) and (BE), BD implies (AE) and (BF), CD implies (BD), and so on.



(a) Reduced merger graph

(b) Compatibility graph for merger graph

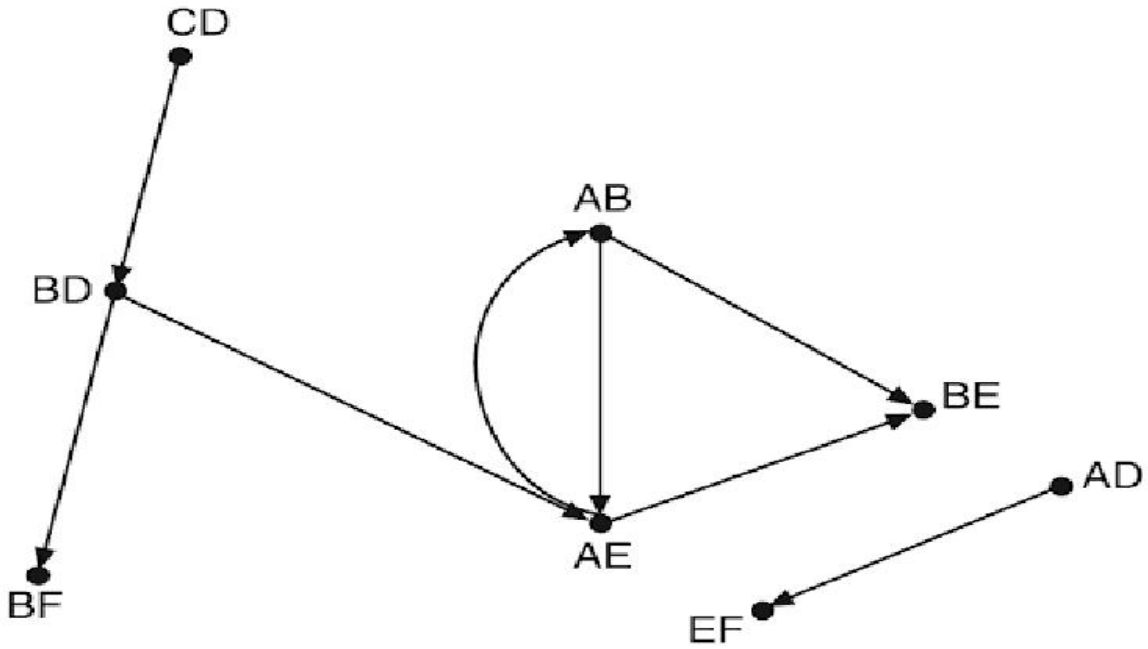
Figure ′ Reduced merger graph and compatibility graph.

Steps to construct compatibility graph from merger table: For the given state table of Figure , first draw the merger table shown in Figure a.

Step 1. *Identify and draw vertices:* Refer to the merger table of the sequential machine shown in Figure a. Mark the vertices corresponding to all compatible pairs in the merger table. In the merger table, each cell of the table except those marked (×) corresponds to the compatible pair defined by the intersection of the row and column headings . If we observe the merger table from right to left and top to bottom the compatible pairs are EF, CD, BD, BE, BF, AB, AD, and AE. This gives us eight vertices in the compatibility graph as shown in Figure b.

	A	B	C	D	E
B	AE BE				
C	×	×			
D	EF	AE BF	BD		
E	AB BE	✓	×	×	
F	~~BC~~	✓	×	×	✓

(a) Merger table

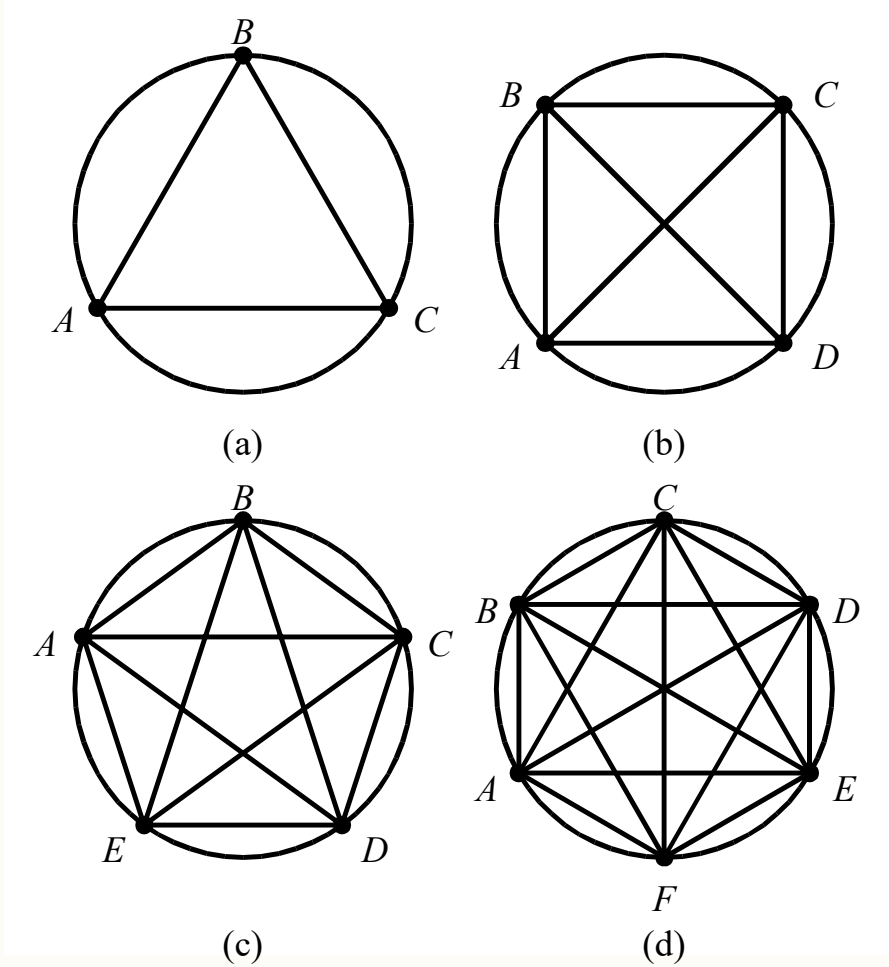


(b) Compatibility graph for merger table

Figure Merger table and compatibility graph from merger table.

Merger diagrams

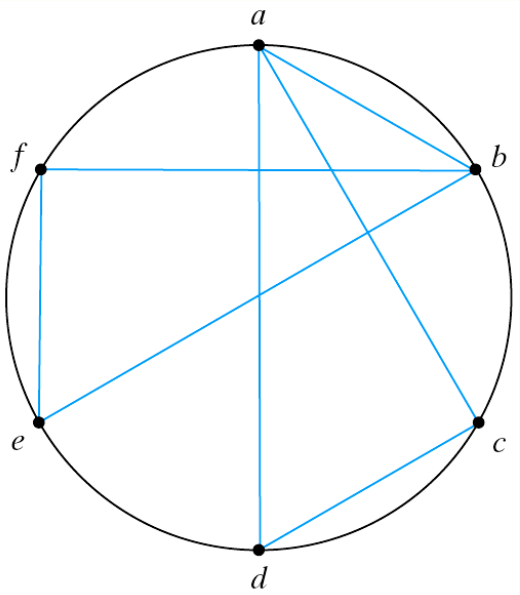
- States are represented as dot in a circle
- Lines connect states couples compatible
- Maximal sets can be identified as those sets in which every states is connected to every other state by a line segment
- The rules for extracting maximal sets from a merger diagram are:
  - Make each maximal set as large as possible
  - Each state in a maximal set must be connected to every other state in the set
  - Each related pair of states must appear in at least one maximal set
- The set of maximal compatible is always complete and consistent. However the set may not be minimal



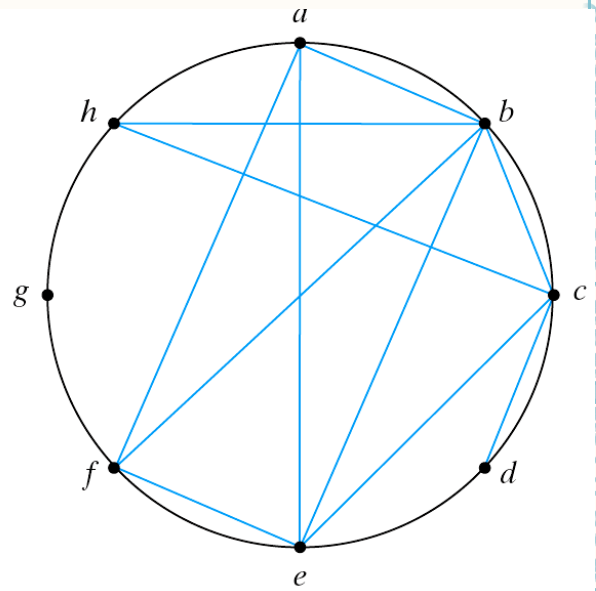
Maximal Compatibles

- A group of compatibles that contains all the possible combinations of compatible states.
  - Obtained from a merger diagram.
  - A line in the diagram represents that two states are compatible.
- n-state compatible → n-sided fully connected polygon.
  - All its diagonals connected.
  - Not all maximal compatibles are necessary

- an isolated dot: a state that is not compatible to any other state
- a line: a compatible pair
- a triangle: a compatible with three states
- an n-state compatible: an n-sided polygon with all its diagonals connected



(a) Maximal compatible:
(a, b) (a, c, d) (b, e, f)



(b) Maximal compatible:
(a, b, e, f) (b, c, h) (c, d) (g)

Closed Covering Condition

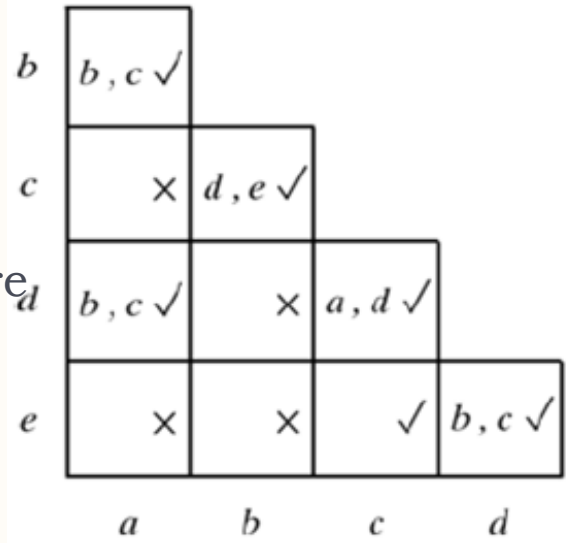
- The condition that must be satisfied for row merging is that the set of chosen compatibles must:
  - Cover all states.
  - Be closed: (the closure condition is satisfied if there are no implied states or if the implied states are included within a set)
- In the first example, the maximal compatibles are (a, b) (a, c, d), (b , e , f)
- If we remove (a, b), we get a set of two compatibles: (a, c, d), (b, e , f):
  - All the six states are included in this set.
  - There are no implied states for (a,c); (a,d);(c,d);(b,e);(b,f) and (e,f) [you can check the implication table] . The closer condition is satisfied
- The original primitive flow table can be merged into two rows, one for each of the compatibles.

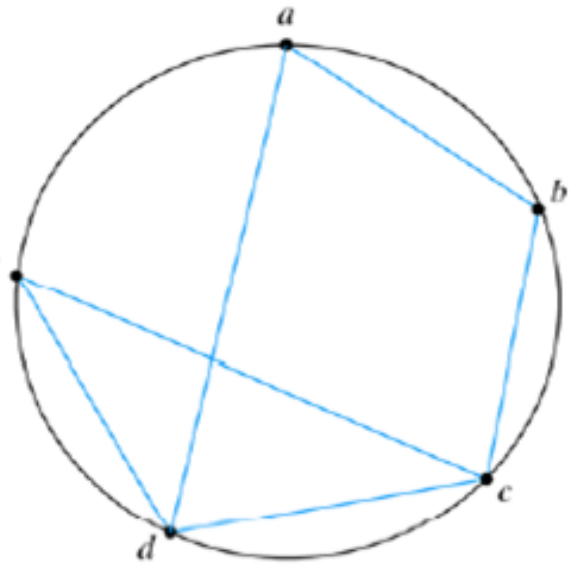
Closed Covering Condition (Example)

- • From the aside implication table, we have the following compatible: pairs: (a, b) (a , d) (b, c) (c , d) (c , e) (d , e)
- • From the merger diagram, we determine the maximal compatibles: (a , b) (a , d) (b , c) (c , d , e)
- • If we choose the two compatibles: (a , b) (c, d, e)
- All the 5 states are included in this set.
- The implied states for (a, b) are (b, c). But (b, c) are not include in the chosen set. This set is not closed.
- A set of compatibles that will satisfy the closed covering condition is (a, d) (b, c) (c, d, e)
- Note that: the same state can be repeated more than once

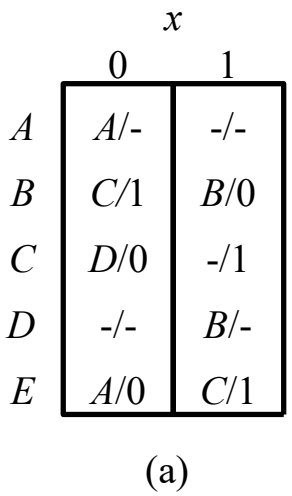
Compatibles	(a, b)	(a, d)	(b, c)	(c, d, e)
Implied states	(b, c)	(b, c)	(d, e)	(a, d) (b, c)

Closure table



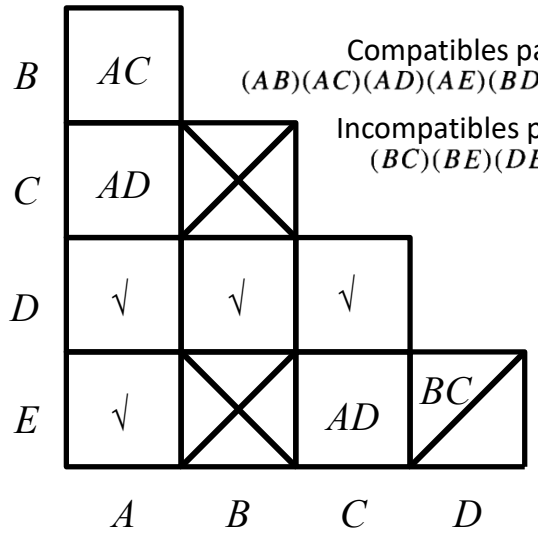


Example -- State reduction problem



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

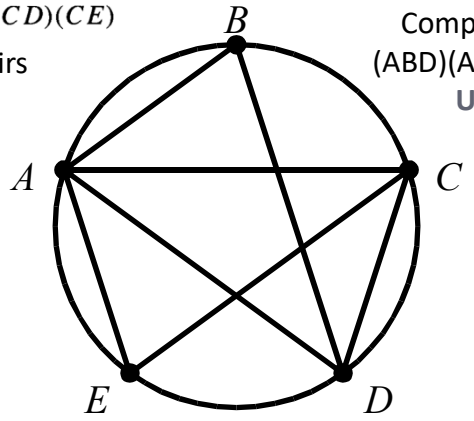
(a)



Compatibles pairs
$(AB)(AC)(AD)(AE)(BD)(CD)(CE)$

Incompatibles pairs
$(BC)(BE)(DE)$

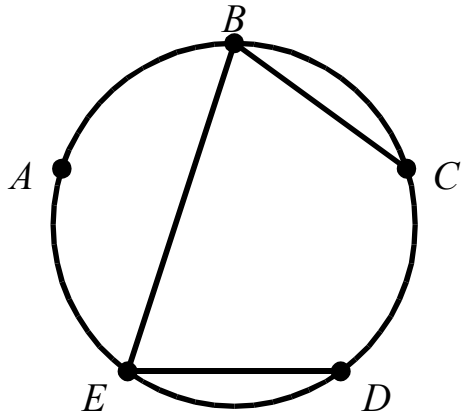
(b)



Maximum Compatibles
(ABD)(ACD)(ACE)
U=3

(c)

Maximum Incompatibles
(BE)(BC)(ED)
L=2



(d)

	x = 0	x = 1
(ABD)	AC	B
(ACD)	AD	B
~~(ACE)~~	AD	C

(e)

Closure table

	x = 0	x = 1
A′	B′/1	A′/0
B′	A′/0	B′/1

(f)

Reduced state table

$A' = (ABD)$

$B' = (ACE)$

Asynchronous Sequential Circuits

Asynchronous sequential circuits basics

✓No clock signal is required

✓Internal states can change at any instant of time when there is a change in the input variables

✓Have better performance but hard to design due to timing problems

Why Asynchronous Circuits?

✓Accelerate the speed of the machine (no need to wait for the next clock pulse).

✓Simplify the circuit in the small independent gates.

✓Necessary when having multi circuits each having its own clock.

Analysis Procedure

✓The analysis consists of obtaining a table or a diagram that describes the sequence of internal states and outputs as a function of changes in the input variables.

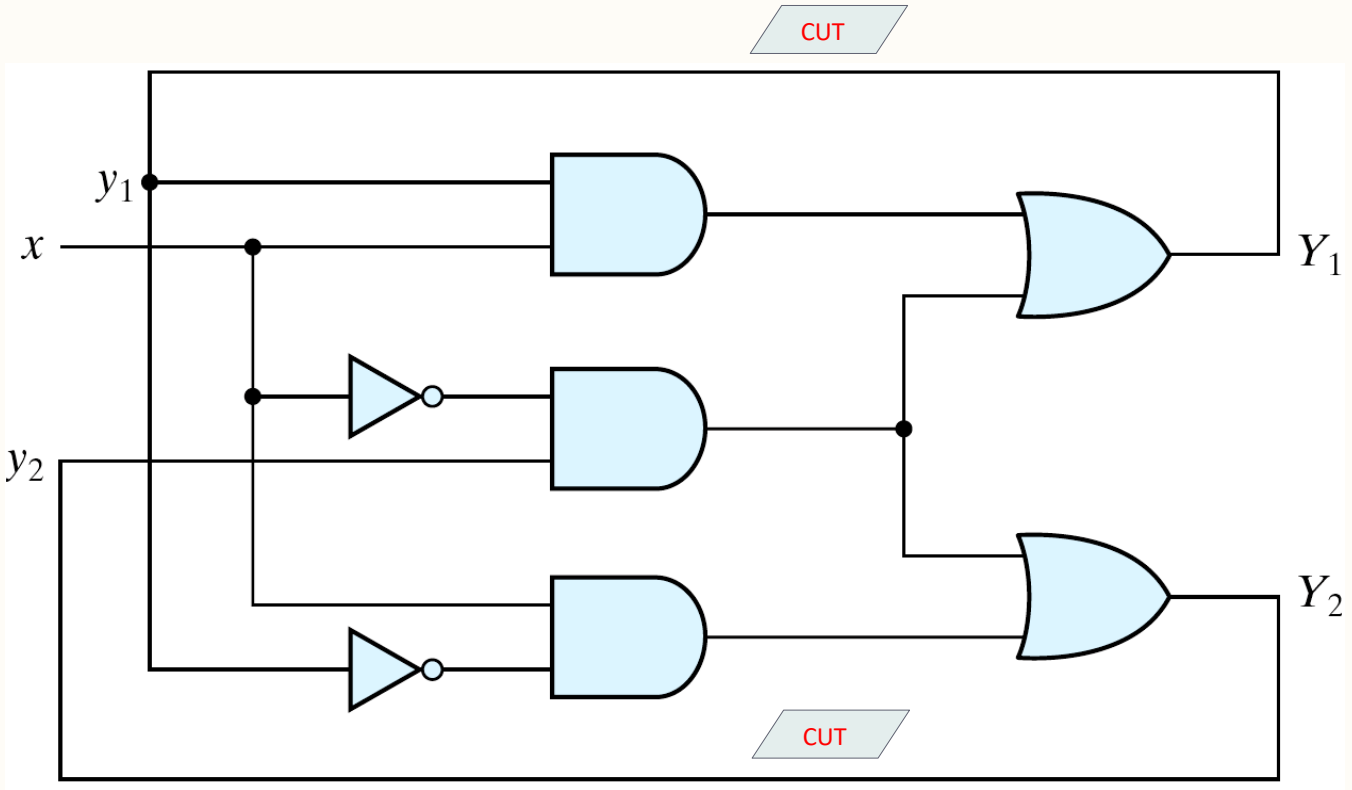
Transition Table

- Transition table is useful to analyze an asynchronous circuit from the circuit diagram. Procedure to obtain transition table:

  - 1. Determine all feedback loops in the circuits
  - 2. Mark the input (yi) and output (Yi) of each feedback loop
  - 3. Derive the Boolean functions of all Y's
  - 4. Plot each Y function in a map and combine all maps into one table (flow table)
  - 5. Circle those values of Y in each square that are equal to the value of y in the same row

Asynchronous Sequential Circuit

- The excitation variables: Y_1 and Y_2
  - $Y_1 = xy_1 + \bar{x}y_2$
  - $Y_2 = x\bar{y}_1 + \bar{x}y_2$



Transition Table

- Combine the internal state with input variables
  - Stable total states:

    y_1y_2x = 000, 011, 110 and 101

y_1y_2 \ x	0	1
00	0	0
01	1	0
11	1	1
10	0	1

(a) Map for $Y_1 = xy_1 + x'y_2$

y_1y_2 \ x	0	1
00	0	1
01	1	1
11	1	0
10	0	0

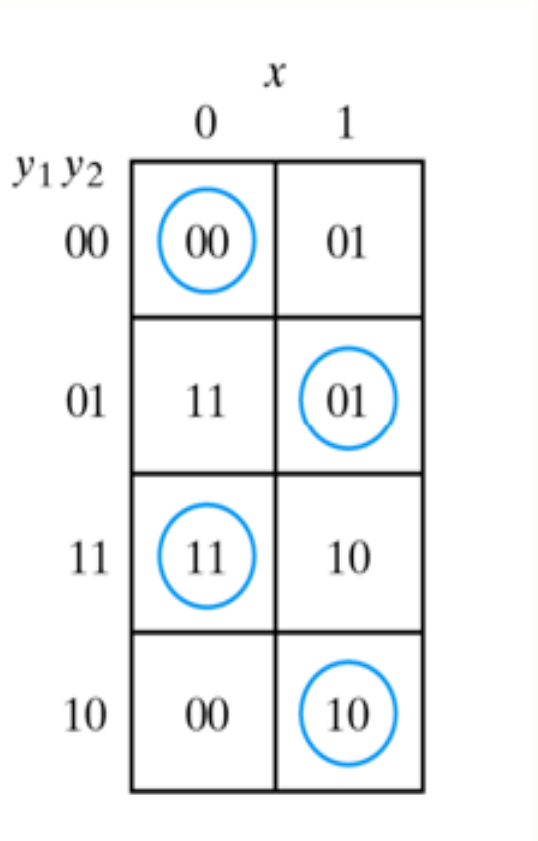
(b) Map for $Y_2 = xy'_1 + x'y_2$

y_1y_2 \ x	0	1
00	(00)	01
01	11	(01)
11	(11)	10
10	00	(10)

(c) Transition table

Transition Table

- In an asynchronous sequential circuit, the internal state can change immediately after a change in the input.
- It is sometimes convenient to combine the internal state with input value together and call it the **Total State of the circuit.** (Total state = Internal state + Inputs)
- In the example , the circuit has
  - 4 **stable total states: (y_1y_2x= 000, 011, 110, and 101)**
  - 4 **unstable total states: (y_1y_2x= 001, 010, 111, and 100)**

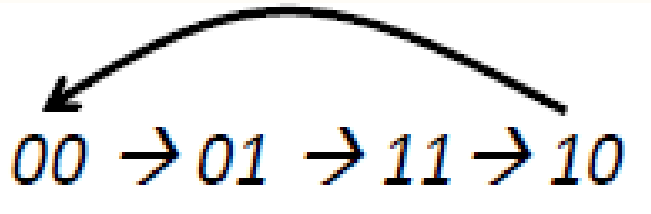


y_1y_2 \ x	0	1
00	(00)	01
01	11	(01)
11	(11)	10
10	00	(10)

Transition Table

- If y=00 and x=0 Y=00 (Stable state)
- If x changes from 0 to 1 while y=00, the circuit changes Y to 01 which is temporary unstable condition (Y≠y)
- As soon as the signal propagates to make Y=01, the feedback path causes a change in y to 01. (transition form the first row to the second row)
- If the input alternates between 0 and 1, the circuit will repeat the sequence of states

$y_1 y_2$	x = 0	x = 1
00	(00)	01
01	11	(01)
11	(11)	10
10	00	(10)



Flow Table

- A flow table is similar to a transition table except that the internal state are symbolized with letters rather than binary numbers.
- It also includes the output values of the circuit for each stable state.

y \ x	0	1
a	(a)	b
b	c	(b)
c	(c)	d
d	a	(d)

(a) Four states with one input

x_1x_2	00	01	11	10
a	(a), 0	(a), 0	(a), 0	b , 0
b	a , 0	a , 0	(b), 1	(b), 0

(b) Two states with two inputs and one output

Flow Table: Example 2

- Two states, two inputs, one output.

	x_1x_2 = 00	01	11	10
a	(a ,0)	(a ,0)	(a ,0)	b ,0
b	a ,0	a ,0	(b ,1)	(b ,0)

- Each row has more than one stable state.
- If $x_1 = 0$, state is a.
- If $x_1x_2 = 00 \rightarrow x_1x_2 = 10$, then state becomes b.
- For $x_1x_2 = 11$, state is either a or b.
  - If previously in $x_1x_2 = 01$, keeps state a,
  - If previously in $x_1x_2 = 10$, keeps state b.
  - Reminder: cannot go from 00 to 11.

Flow Table

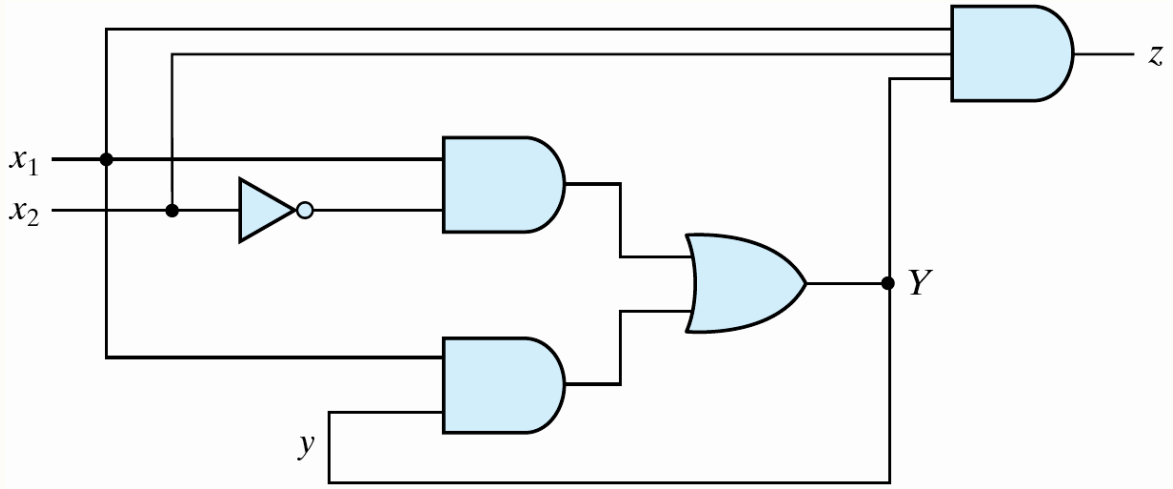
- In order to obtain the circuit described by a flow table, it is necessary to convert the flow table into a transition table from which we can derive the logic diagram.
- This can be done through the assignment of a distinct binary value to each state.

y \ x_1x_2	00	01	11	10
0	0	0	0	1
1	0	0	1	1

(a) Transition table
$Y = x_1x'_2 + x_1y$

y \ x_1x_2	00	01	11	10
0	0	0	0	0
1	0	0	1	0

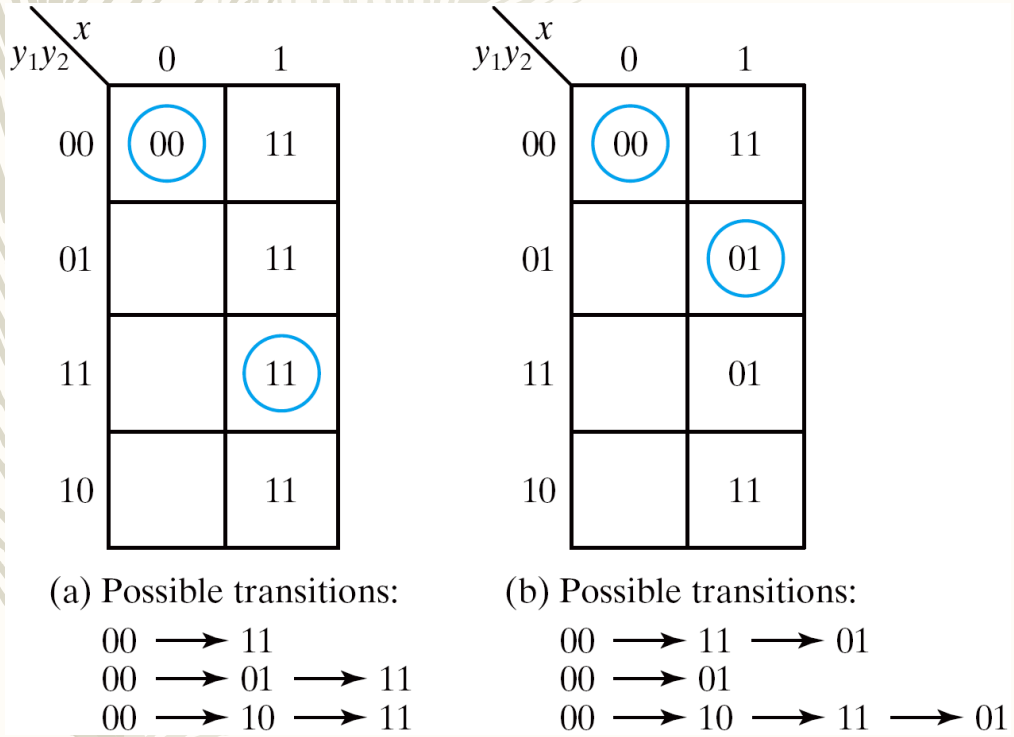
(b) Map for output
$z = x_1x_2\,y$

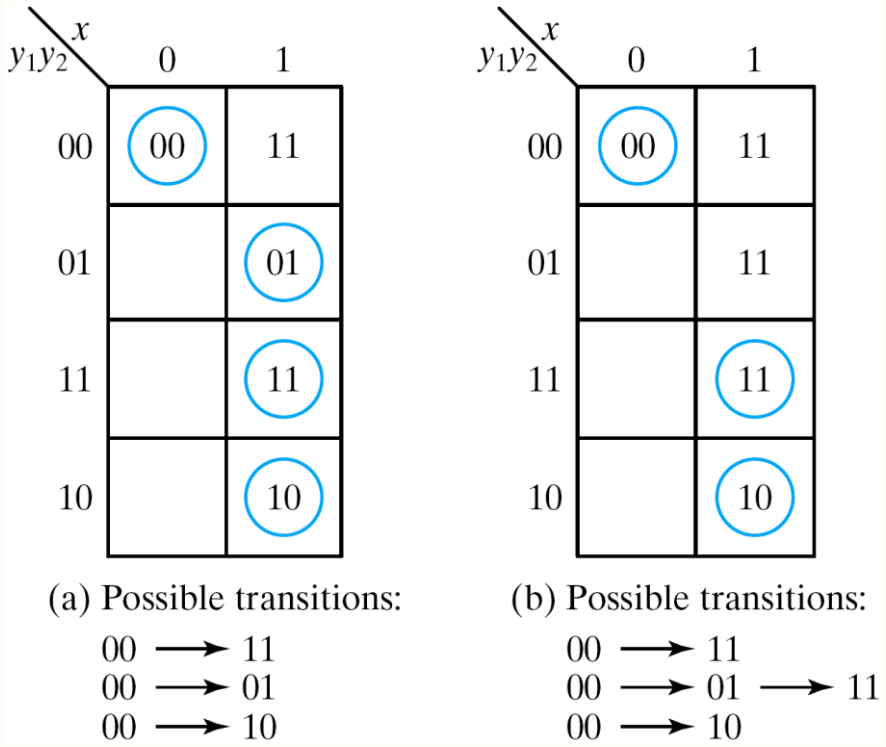


(c) Logic diagram

Race condition

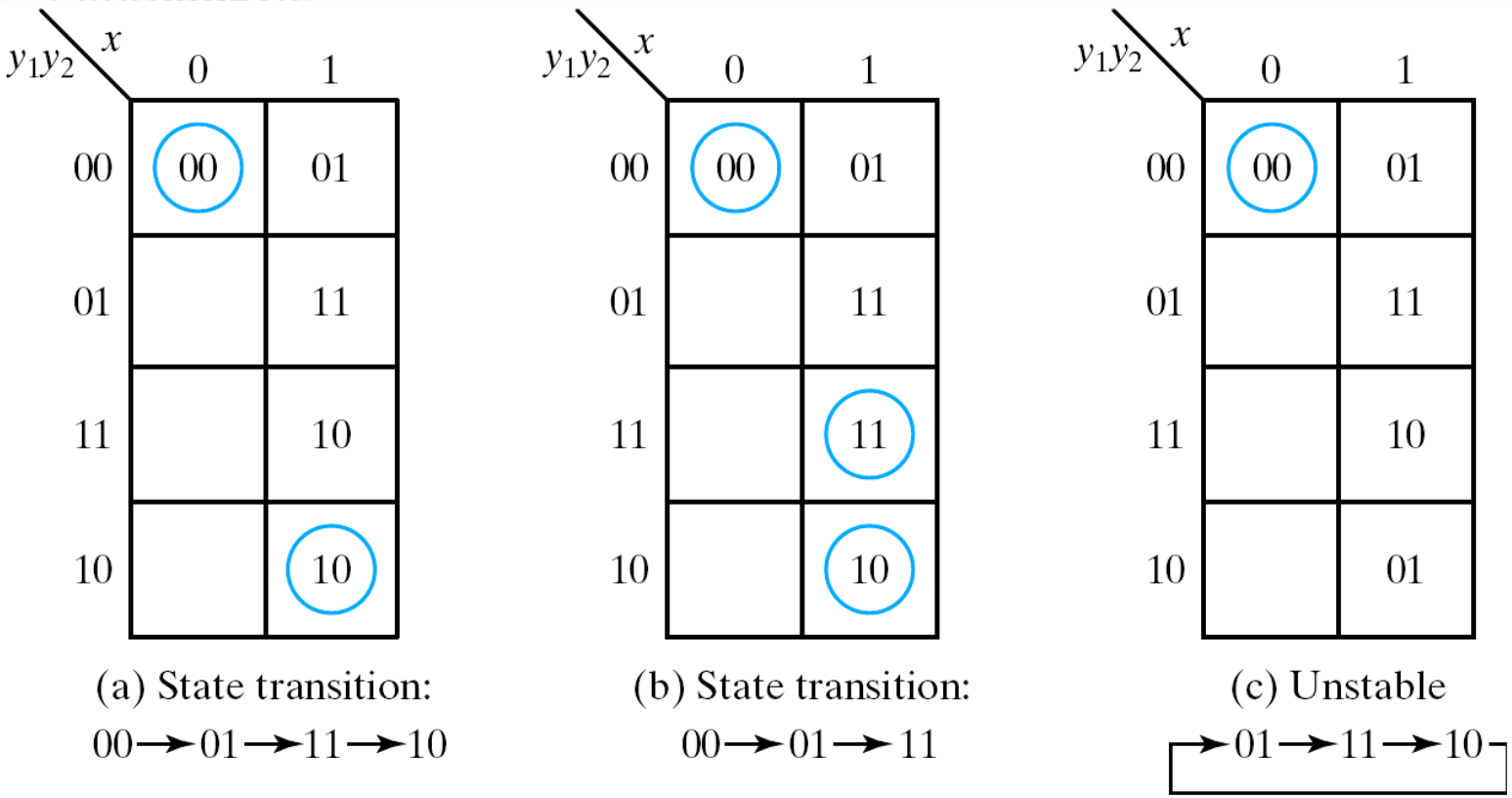
- Two or more binary state variables will change value when one input variable changes.
- Cannot predict state sequence if unequal delay is encountered.
- **Non-critical race**: The final stable state does not depend on the change order of state variables
- **Critical race**: The change order of state variables will result in different stable states. Must be avoided !!





Race Solution

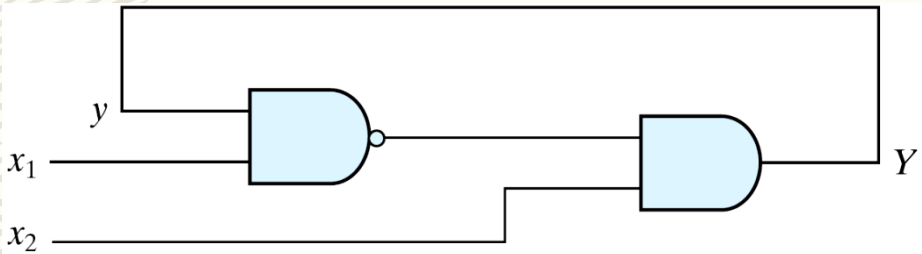
- It can be solved by making a proper binary assignment to the state variables.
- The state variables must be assigned binary numbers in such a way that only one state variable can change at any one time when a state transition occurs in the flow table.



Stability Check

- Asynchronous sequential circuits may oscillate between unstable states due to the feedback
  - Must check for stability to ensure proper operations
- Can be easily checked from the transition table
  - Any column has no stable states ⟶ unstable
  - For $x_1x_2 = 11$, there is no steady state.
    - → Oscillation.
  - For $x_1x_2 = 11$, Y = y' → unstable.

Y=x2(x1y)'=x'1x2+x2y'



(a) Logic diagram

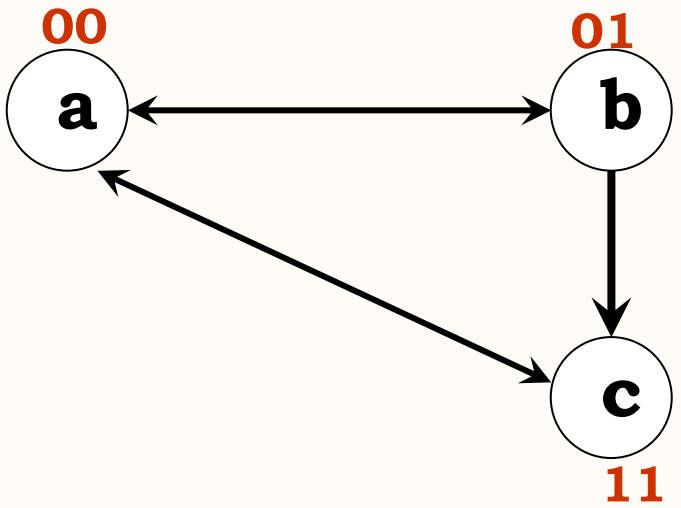
y \ x_1x_2	00	01	11	10
0	0	1	1	0
1	0	1	0	0

(b) Transition table

No-Race State Assignment

- Must assign binary values to states such that:
  - *one change in an input may not cause two changes in state variables.*
    - because, due to delays, one of the variable change sooner and may stay in an unwanted stable state.
  - *From a, if x1x2 = 10 → 11, must go to c and stay there.*
  - *But by the following assignment, it may go to b and stay there.*

	$x_1 x_2$ = 00	01	11	10
a	a	b	c	a
b	a	b	b	c
c	a	c	c	c

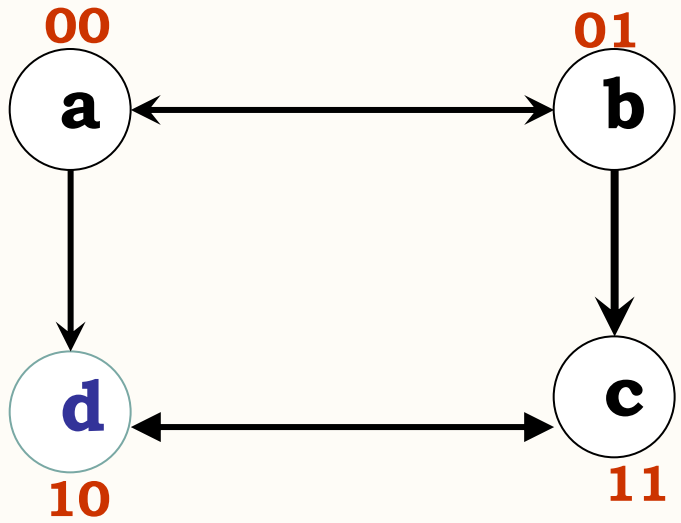


- a and b must be different in one bit,
- a and c must be different in one bit.

No-Race State Assignment

- Impossible → add one more row.

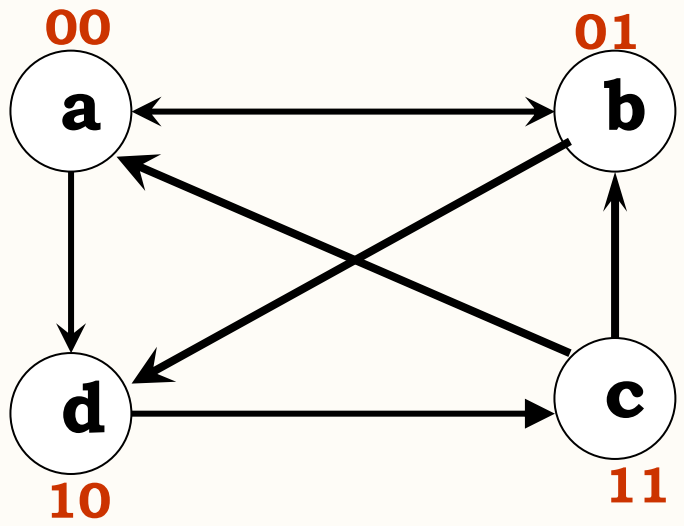
	$x_1 x_2$ = 00	01	11	10
a	a	b	**d**	a
b	a	b	b	c
c	**d**	c	c	c
d	a	-	c	-



- d is an intermediate (unstable) state.
- - means any value can be assigned (Except d=10).

Example 2

	$x_1 x_2$ = 00	01	11	10
a	b	a	d	a
b	b	d	b	a
c	c	a	b	c
d	c	d	d	c



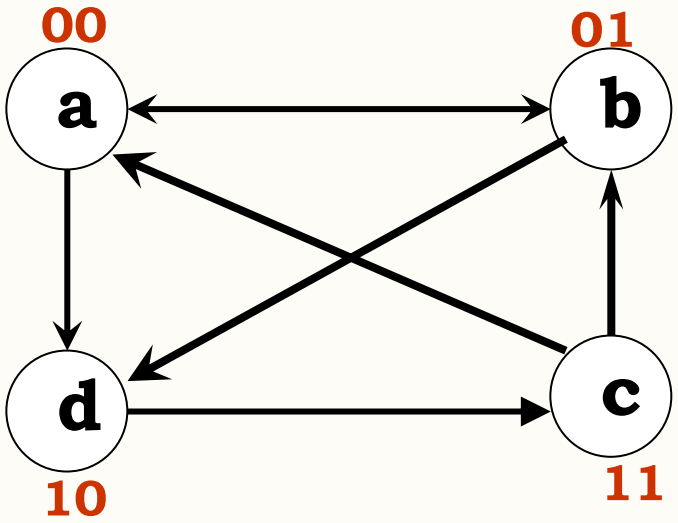
- If there were no diagonal transition, it would be possible
- Impossible → add some more rows.

Example 2

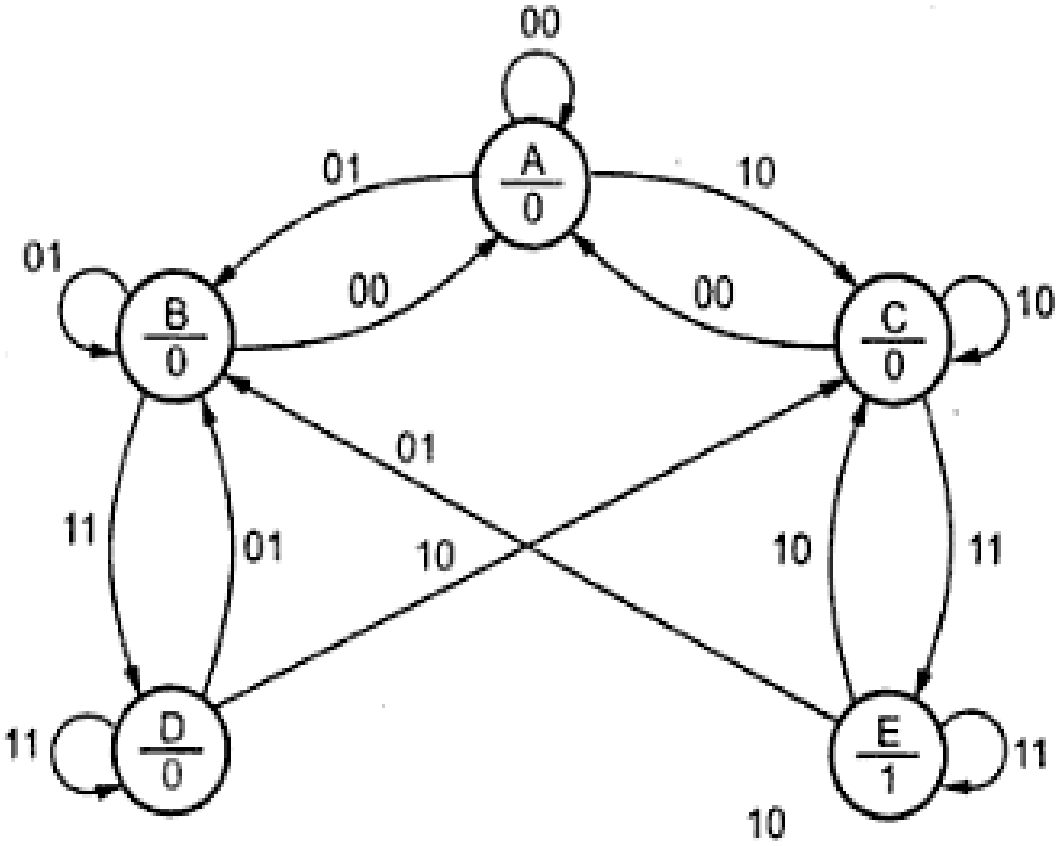
	$x_1 x_2$ = 00	01	11	10
a = 000	b	a	e	a
b = 001	b	d	b	a
c = 011	c	g	b	c
g = 010	-	a	-	-
110	-	-	-	-
f = 111	c	-	-	c
d = 101	f	d	d	f
e = 100	-	-	d	-

y_3	$y_1 y_2$ = 00	01	11	10
0	a	b	c	g
1	e	d	f	0

- b is adjacent to a, c, d
- c → a through g
- a → d through e
- d → c through f



Prob 2: Design a circuit with inputs A and B to give an output Z=1 when AB=11 but only if A=1 before B, by drawing total state diagram, primitive flow table and output map in which transient state is included



Present State	Next State , Output Z for AB Inputs			
	00	01	11	10
A	(A) , 0	B , –	– , –	C , –
B	A , –	(B) , 0	D , –	– , –
C	A , –	– , –	E , –	(C) , 0
D	– , –	B , –	(D) , 0	C , –
E	– , –	B , –	(E) , 1	C , –

Fig. 9.57 Primitive flow table for given problem

$(A, B, D) \rightarrow S_0$

$(C, E) \rightarrow S_1$

Present State	Next State, Output Z for AB inputs			
	00	01	11	10
S_0	(S_0), 0	(S_0), 0	(S_0), 0	S_1, –
S_1	S_0, –	S_0, –	(S_1), 1	(S_1), 0

Fig. 9.58 Reduced primitive flow table

Present State	Next state, Output Z for AB inputs			
F	00	01	11	10
0	0, 0	0, 0	0, 0	1, –
1	0, –	0, 1	1, 1	1, 0

Table 9.3 Transition table

For F^+

F \ AB	00	01	11	10
0	0	0	0	1
1	0	0	1	1

$F^+ = F + A\bar{B}$

For Z

F \ AB	00	01	11	10
0	0	0	0	X
1	X	1	1	0

$Z = FB$

Merging of the Flow Table

- The state table may be incompletely specified: combinations of inputs or input sequences may never occur (Some next states and outputs are don't care).
- Primitive flow tables are always incompletely specified
  - Several synchronous circuits also have this property
- Incompletely specified states are not "equivalent" as in completely specified circuits. Instead, we are going to find "compatible" states
- Two states of a incompletely specified circuit are compatible if they have the same output and compatible next states whenever specified.
- Three procedural steps:
  - Determine all compatible pairs by using the implication table
  - Find the maximal compatibles by using a merger diagram
  - Find a minimal closed collection of compatible that covers all states and is closed

State Assignment

- Primary Objective of Synchronous Networks
  - Simplification of Logic
  - Improvement of Performance
  - Improvement of Testability
  - Minimization of Power Consumption.
- Primary Objective of Asynchronous Networks
  - Prevention of Critical Races
  - Simplification of Logic

Race-Free State Assignment

- Objective: choose a proper binary state assignment to prevent critical races
- Only one variable can change at any given time when a state transition occurs
- States between which transitions occur will be given adjacent assignments
  - Two binary values are said to be adjacent if they differ in only one variable
- To ensure that a transition table has no critical races, every possible state transition should be checked
  - A tedious work when the flow table is large
  - Only 3-row and 4-row examples are demonstrated

3-Row Flow-Table Example

- Three states require two binary variables (in the flow table outputs are omitted for simplicity)
- Representation by a transition diagram
- a and c are not adjacent in such an assignment!
  - Impossible to make all states adjacent if only 3 states are used

	$x_1 x_2$ = 00	01	11	10
a	(*a*)	*b*	*c*	(*a*)
b	*a*	(*b*)	(*b*)	*c*
c	*a*	(*c*)	(*c*)	(*c*)

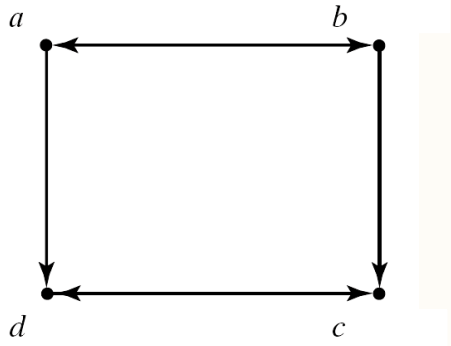
(a) Flow table

3-Row Flow-Table Example

- A race-free assignment can be obtained if we add an extra row to the flow table
- Only provide a race-free transition between the stable states
- The transition from a to c must now go through d
- 00 ⇨ 10 ⇨ 11 (no race condition)
- Note that no stable state can be introduced in row d

	x_1x_2 = 00	01	11	10
a	(a)	b	d	(a)
b	a	(b)	(b)	c
c	d	(c)	(c)	(c)
d	a	–	c	–

(a) Flow table



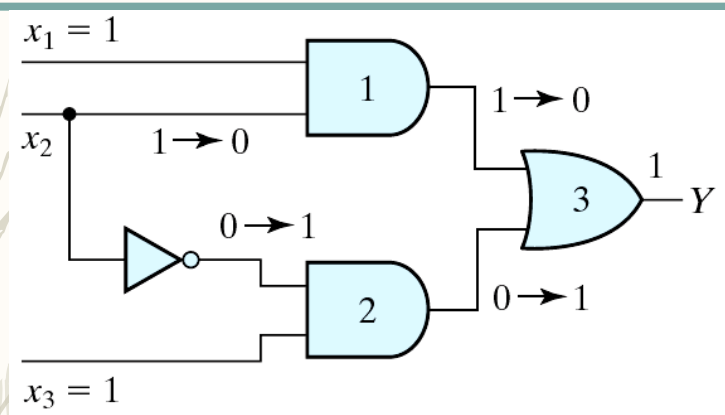
	x_1x_2 = 00	01	11	10
a = 00	(00)	01	10	(00)
b = 01	00	(01)	(01)	11
c = 11	10	(11)	(11)	(11)
d = 10	00	–	11	–

don't care but cannot be 10 (cannot stable)

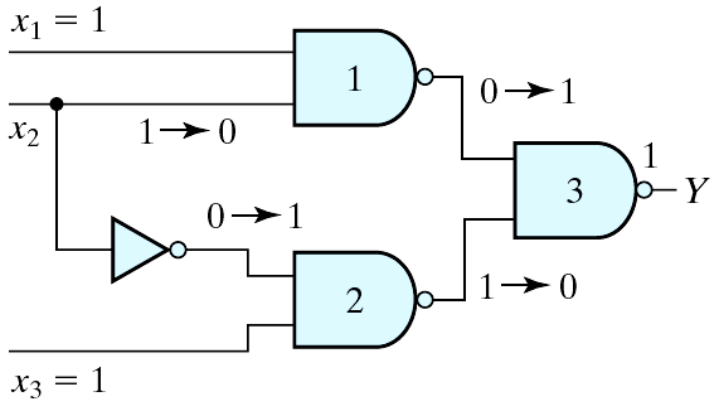
Hazards

- A timing problem arises due to gate and wiring delays
- Hazards: Unwanted switching transients at the network output, caused by input changes and due to different paths through the network from input to output that may have different propagation delays
- Hazards occur in combinational and asynchronous circuits:
  - In combination circuits, they may cause a temporarily false output value.
  - In asynchronous circuits, they may result in a transition to a wrong stable state.
- Asynchronous Sequential Circuits:
  - Hyphothesis:
    - *Operated in fundamental mode with only one input changing at any time*
  - Objectives:
    - *Free of critical races*
    - *Free of hazards*

Hazards in combinational circuits



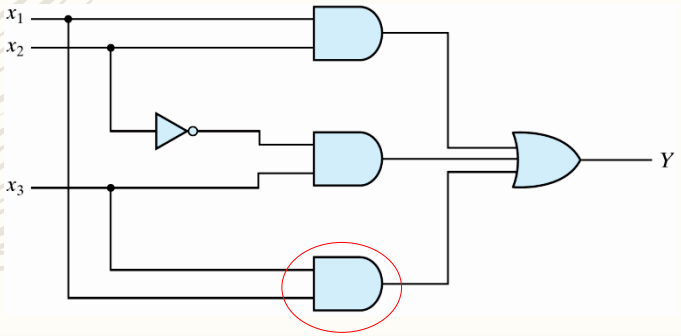
(a) AND–OR circuit

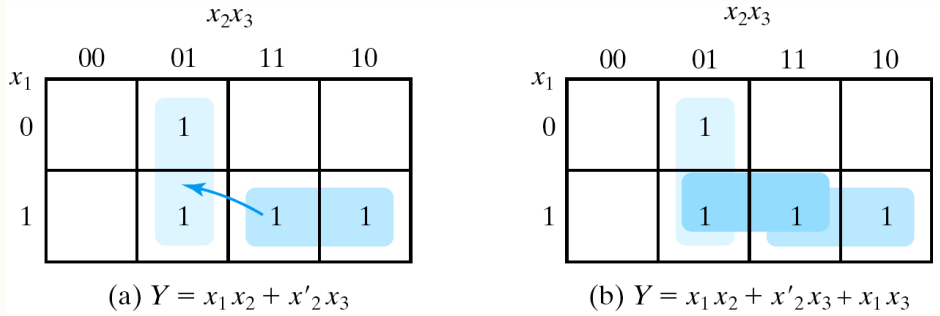


(b) NAND circuit

- ✓ static 1-hazard (sum of products)
  - the removal of static 1-hazard guarantees that no static 0-hazards or dynamic hazards
  - The remedy
    the circuit moves from one product term to another
    additional redundant gate

(a) Static 1-hazard (b) Static 0-hazard (c) Dynamic hazard

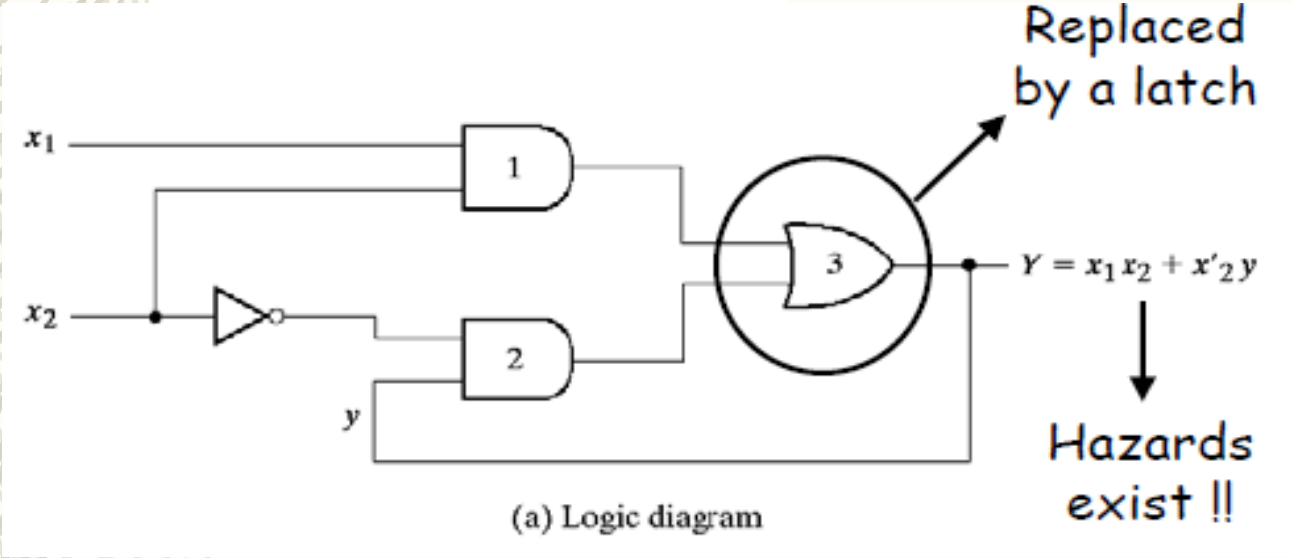




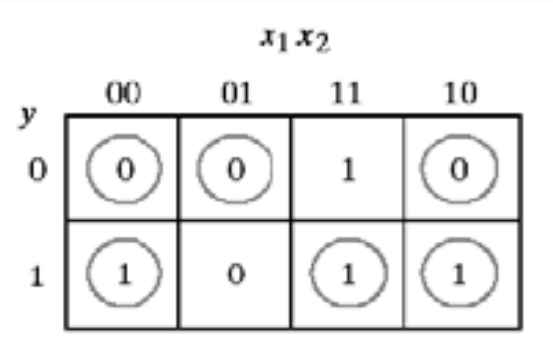
(a) $Y = x_1x_2 + x'_2x_3$ (b) $Y = x_1x_2 + x'_2x_3 + x_1x_3$

Remove Hazards with Latches

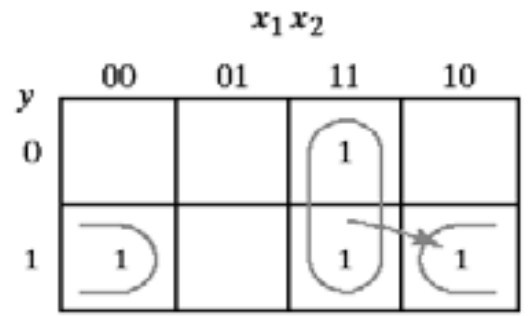
- Implement the asynchronous circuit with SR latches can also remove static hazards
  - A momentary 0 has no effects to the S and R inputs of a NOR latch
  - A momentary 1 has no effects to the S and R inputs of a NAND latch



(a) Logic diagram



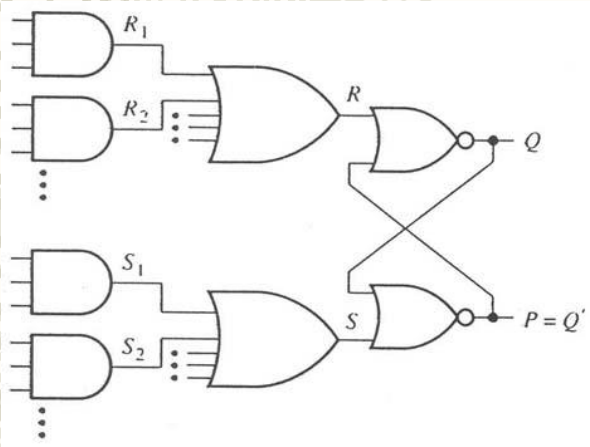
(b) Transition table



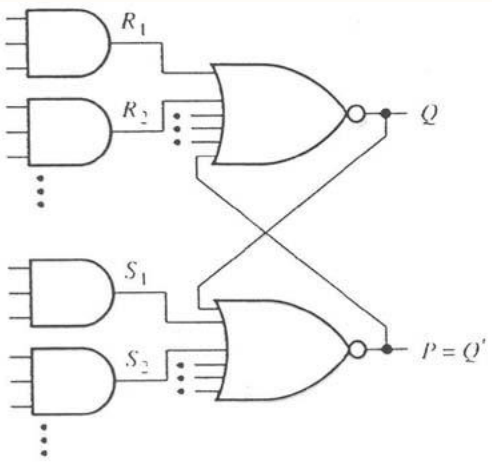
(c) Map for Y

Hazard-Free Realization

- S-R Latch (NOR type)
  - The network realizing S and R must be free of 0-hazards.
- Minimum sum-of-product must be free of 0-hazard
  - Apply simple transformations to avoid 0-hazard



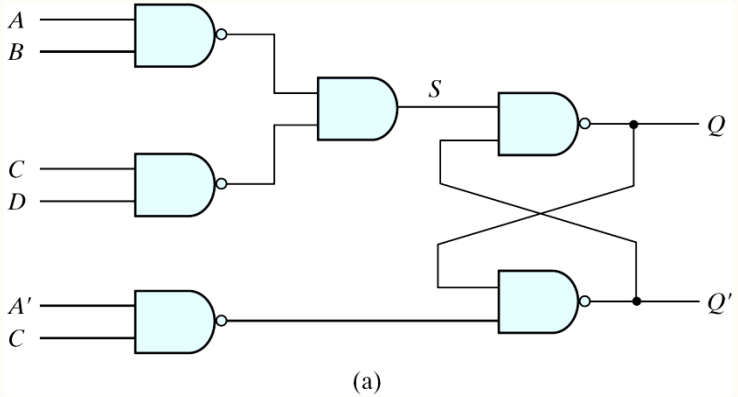
S-R Flip-Flop Driven by 2-level AND-OR Networks



Equivalent Network Structure (in general faster)

Example

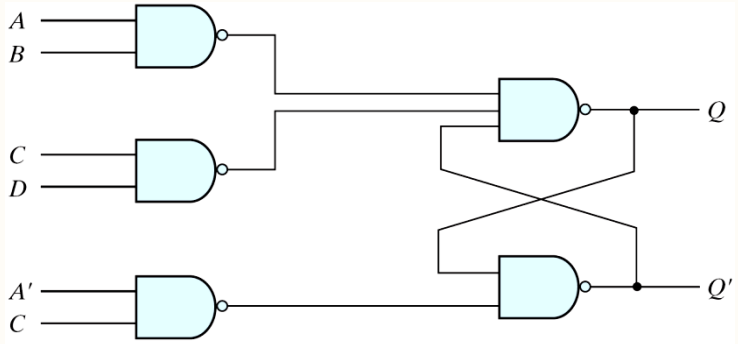
- Consider a NAND SR-latch with the following Boolean functions for S and R
- S = AB + CD
- R = A'C



(a)

S = (AB + CD)' =(AB)'(CD)'
R = (A'C)'

- Since this is a NAND latch we must use the complement value for S and R
- The Boolean function for output is
- Q = (Q'S)' = [Q' (AB)'(CD)']'



(b)

- The output is generated with two levels of NAND gates:

If output Q is equal to 1, then Q' is equal to 0. If two of the three inputs go momentarily to 1, the NAND gate associated with output Q will remain at 1 because Q' is maintained at 0.

Essential Hazards

- Besides static and dynamic hazards, another type of hazard in asynchronous circuits is called: Essential Hazard
- It is caused by unequal delays along two or more paths that originate from the same input
- Cannot be corrected by adding redundant gates
- Can only be corrected by adjusting the amount of delay in the affected path
  - Each feedback path should be examined carefully !!