

---

# UNIT-III

# Asynchronous Circuits

---

- ✓ Not synchronized by a common clock
- ✓ States change immediately after input changes
- ✓ The circuit reaches a **steady-state condition** when  $y_i = Y_i$  for all  $i$ .
- ✓ For a given value of input variables, **the system is stable if the circuit reaches a steady state condition.**
- ✓ A transition from one stable state to another occurs only in response to a change in an input variable
- ✓ **Fundamental-mode operation**
  - The input signals change only when the circuit is in a **stable condition**
  - The input signals change **one at a time**
- ✓ The time between two input changes must be longer than the time it takes the circuit to reach a stable state.
- ✓ Timing is a Major Problem because of
  - **Unequal delays** through various paths in the circuit

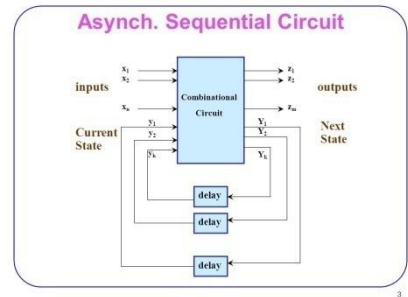
# Asynchronous Sequential Circuits

## 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 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.

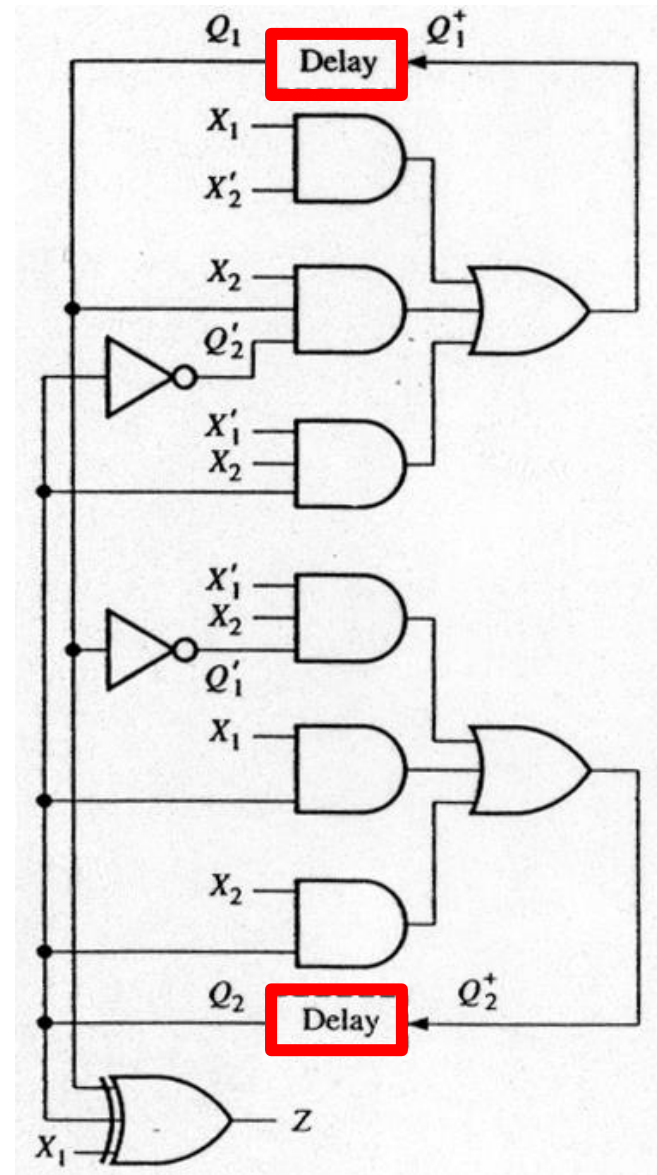
# Example Circuit

- ✓ Construction of Asynchronous Circuits:
  - using only gates
  - with feedback paths
- ✓ Analysis:
  - Lump all of the delay associated with each feedback path into a "delay" box
  - Associate a state variable with each delay output
  - Construct the flow table
- ✓ Network equations

$$Q_1^+ = X_1X_2' + X_1'X_2Q_2 + X_2Q_1Q_2'$$

$$Q_2^+ = X_1'X_2Q_1' + X_1Q_2 + X_2Q_2$$

$$Z = X_1 \oplus Q_1 \oplus Q_2$$



# Example Circuit: Output Table

- ✓ 1. Starting in total state  
 $X_1X_2Q_1Q_2 = 0000$
- ✓ 2. Input changes to 01
  - Internal state changes to 01 and then to 11.
- ✓ 3. Input changes to 11.
  - Go to unstable total state 1111 and then to 1101.
- ✓ 4. Input changes to 10.
  - Go to unstable total state 1001 and then to 1011.
- ✓ The output sequence:
  - 0 (0) (1) 0 (1) 0 (0) 1
  - Condensed to the form  
 $0 (1) 0 (1) 0 1$ .
  - Two transient 1 outputs can be eliminated by proper design.

$Q_1Q_2 \backslash X_1X_2$		00	01	11	10
		00	01	00	10
00	00	01	00	10	
01	00	11	01	11	
11	00	11	01	11	
10	00	10	10	10	

$Q_1Q_2 \backslash X_1X_2$		00	01	11	10
		0	0	1	1
00	0	0	1	1	
01	1	1	0	0	
11	0	0	1	1	
10	1	1	0	0	

Z

# Transition Table

---

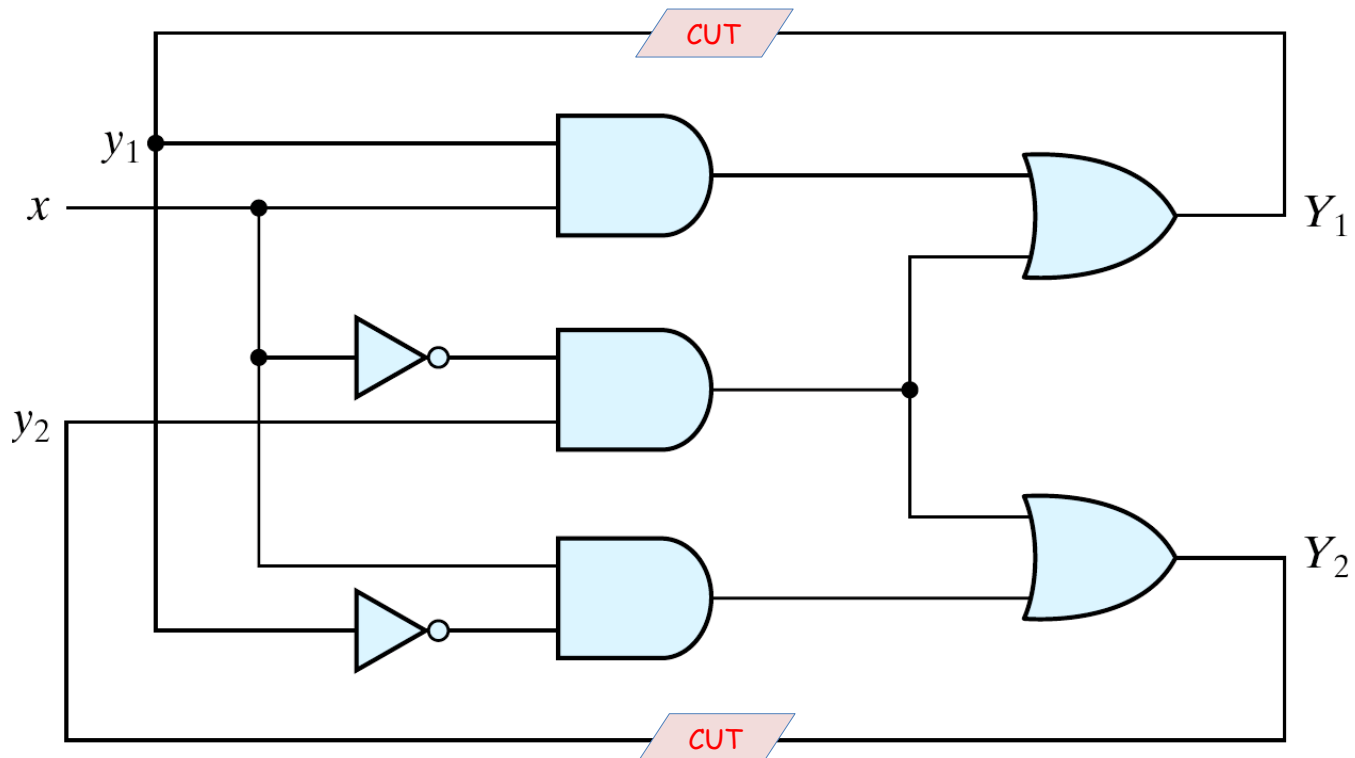
- ✓ Transition table is useful to analyze an asynchronous circuit from the circuit diagram. Procedure to obtain transition table:
  1. Determine all feedback loops in the circuits
  2. Mark the input ( $y_i$ ) and output ( $Y_i$ ) of each feedback loop
  3. Derive the Boolean functions of all  $Y$ 's
  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:

$$y_1 y_2 x = 000, 011, 110 \text{ and } 101$$

	$x$	
	0	1
$y_1 y_2$		
00	0	0
01	1	0
11	1	1
10	0	1

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

	$x$	
	0	1
$y_1 y_2$		
00	0	1
01	1	1
11	1	0
10	0	0

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

	$x$	
	0	1
$y_1 y_2$		
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: ( $y_1y_2x = 000, 011, 110, \text{ and } 101$ )
  - 4 unstable total states: ( $y_1y_2x = 001, 010, 111, \text{ and } 100$ )

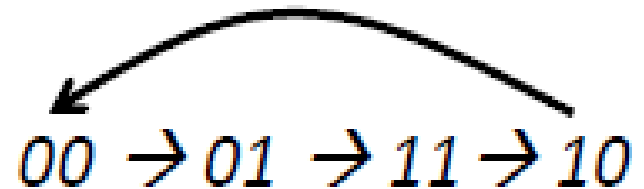
		$x$	
		0	1
$y_1y_2$	00	00	01
	01	11	01
	11	11	10
	10	00	10

# Transition Table

---

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

		$x$	
		0	1
$y_1 y_2$	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.

$y \backslash x$	0	1
$a$	$\textcircled{a}$	$b$
$b$	$c$	$\textcircled{b}$
$c$	$\textcircled{c}$	$d$
$d$	$a$	$\textcircled{d}$

(a) Four states with one input

$x_1 x_2$	00	01	11	10
$a$	$\textcircled{a}, 0$	$\textcircled{a}, 0$	$\textcircled{a}, 0$	$b, 0$
$b$	$a, 0$	$a, 0$	$\textcircled{b}, 1$	$\textcircled{b}, 0$

(b) Two states with two inputs and one output

# 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.

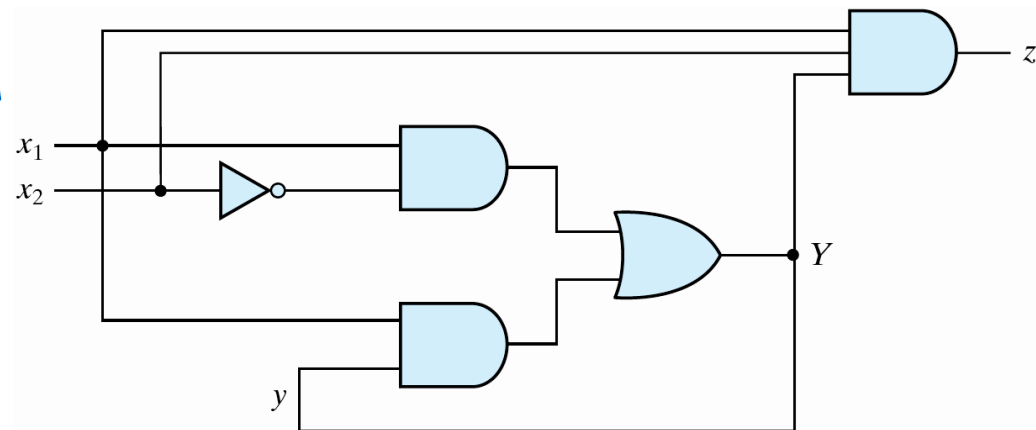
$y \backslash x_1x_2$		00	01	11	10
0	0	0	0	0	1
1	0	0	0	1	1

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

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

(b) Map for output  
 $z = x_1x_2y$

- ✓ This can be done through the assignment of a distinct binary value to each state.



(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 !!**

		x	
		0	1
y <sub>1</sub> y <sub>2</sub>	00	00	11
	01		11
	11		11
	10		11

(a) Possible transitions:

00 → 11  
 00 → 01 → 11  
 00 → 10 → 11

		x	
		0	1
y <sub>1</sub> y <sub>2</sub>	00	00	11
	01		01
	11		01
	10		11

(b) Possible transitions:

00 → 11 → 01  
 00 → 01  
 00 → 10 → 11 → 01

		x	
		0	1
y <sub>2</sub>	00	00	11
	01		01
	11		11
	10		10

(a) Possible transitions:

00 → 11  
 00 → 01  
 00 → 10

		x	
		0	1
y <sub>1</sub> y <sub>2</sub>	00	00	11
	01		11
	11		11
	10		10

(b) Possible transitions:

00 → 11  
 00 → 01 → 11  
 00 → 10

# 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.

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

(a) State transition:  
00 → 01 → 11 → 10

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

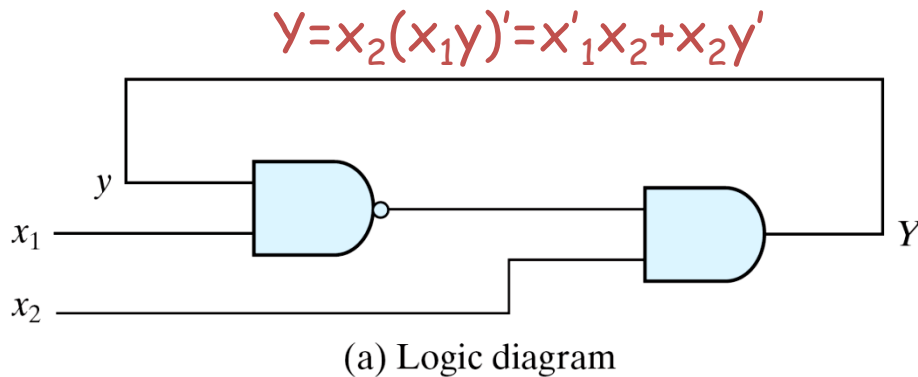
(b) State transition:  
00 → 01 → 11

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

(c) Unstable  
→ 01 → 11 → 10 →

# 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  $\longrightarrow$  unstable  
Ex: when  $x_1x_2=11$  in (b),  $Y$  and  $y$  are never the same



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

(b) Transition table

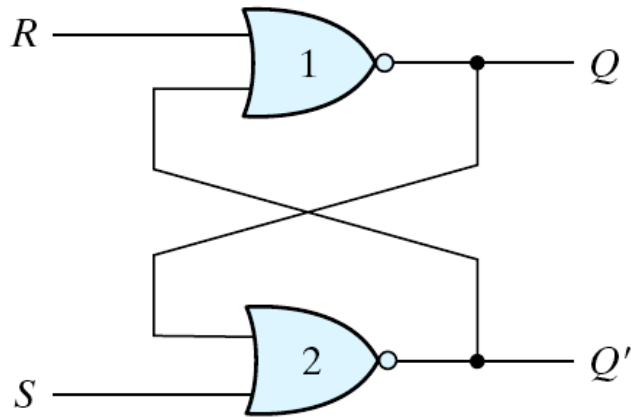
# Latches in Asynchronous Circuits

---

- ✓ The traditional configuration of asynchronous circuits is using one or more feedback loops
  - No real delay elements.
- ✓ It is more convenient to employ the SR latch as a memory element in asynchronous circuits
  - Produce an orderly pattern in the logic diagram with the memory elements clearly visible.
- ✓ SR latch is an asynchronous circuit
  - So will be analyzed first using the method for asynchronous circuits.



# SR Latch with NOR Gates



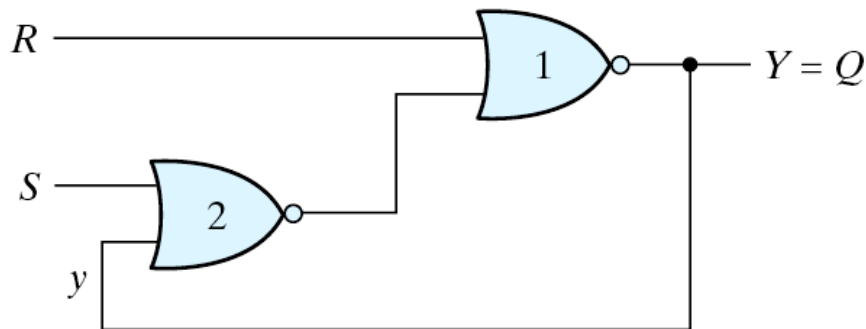
(a) Cross-coupled circuit

$S$	$R$	$Q$	$Q'$
1	0	1	0
0	0	1	0
0	1	0	1
0	0	0	1
1	1	0	0

(After  $SR = 10$ )

(After  $SR = 01$ )

(b) Truth table



(c) Circuit showing feedback

$SR$		00	01	11	10
$y$					
0		0	0	0	1
1		1	0	0	1

$$Y = SR' + R'y$$

(d) Transition table

## Constraints on Inputs

---

The condition to be avoided is that both S and R inputs must not be 1 simultaneously. This condition is avoided when  $SR = 0$  (i.e., ANDing of S and R must always result in 0).

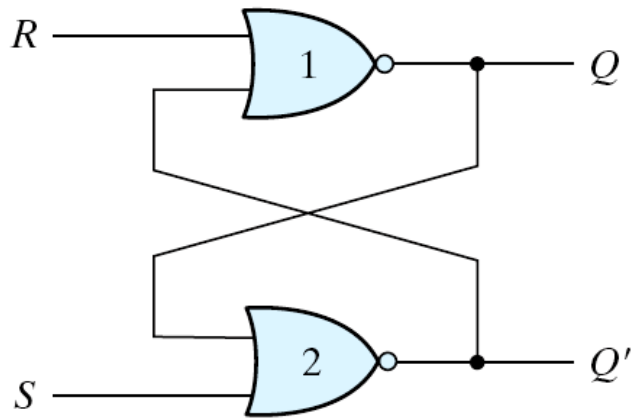
When  $SR = 0$  holds at all times, the excitation function derived previously:

$$Y = SR' + R'y$$

can be expressed as:

$$\boxed{Y = S + R'y}$$

# SR Latch with NOR Gates



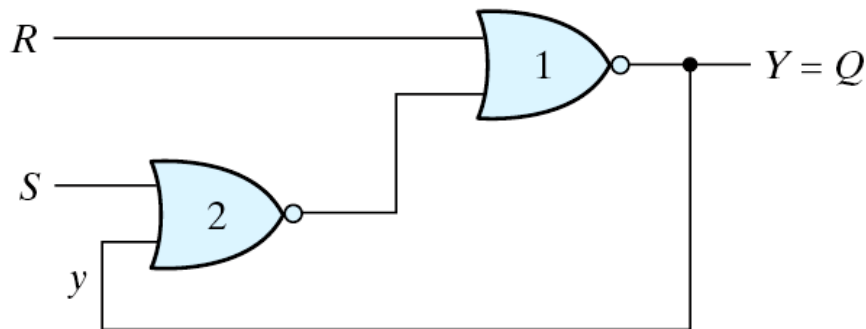
(a) Cross-coupled circuit

$S$	$R$	$Q$	$Q'$
1	0	1	0
0	0	1	0
0	1	0	1
0	0	0	1
1	1	0	0

(After  $SR = 10$ )

(After  $SR = 01$ )

(b) Truth table



(c) Circuit showing feedback

$SR$		00	01	11	10
$y$	0	0	0	0	1
	1	1	0	0	1

$S=1, R=1$  ( $SR = 1$ )  
should not be used  
 $\Rightarrow SR = 0$  is  
normal mode

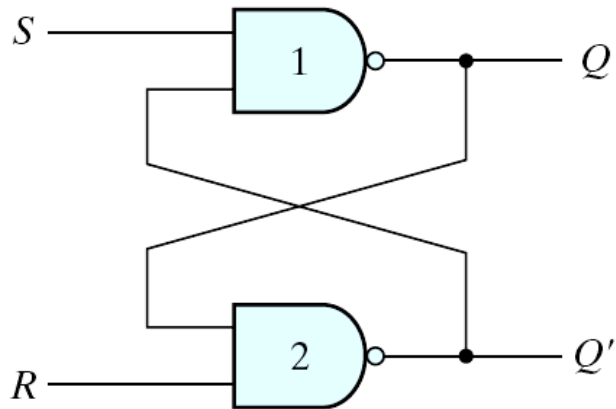
$$Y = SR' + R'y$$

$$Y = S + R'y \text{ when } SR = 0 \Rightarrow$$

**should be carefully  
checked first**

(d) Transition table

# SR Latch with NAND Gates



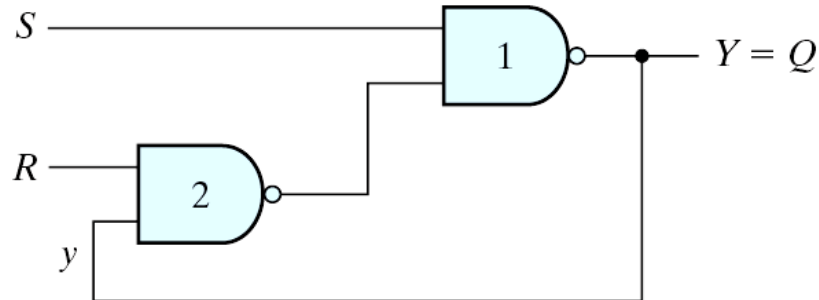
(a) Cross-coupled circuit

$S$	$R$	$Q$	$Q'$
1	0	0	1
1	1	0	1
0	1	1	0
1	1	1	0
0	0	1	1

(After  $SR = 10$ )

(After  $SR = 01$ )

(b) Truth table



(c) Circuit showing feedback

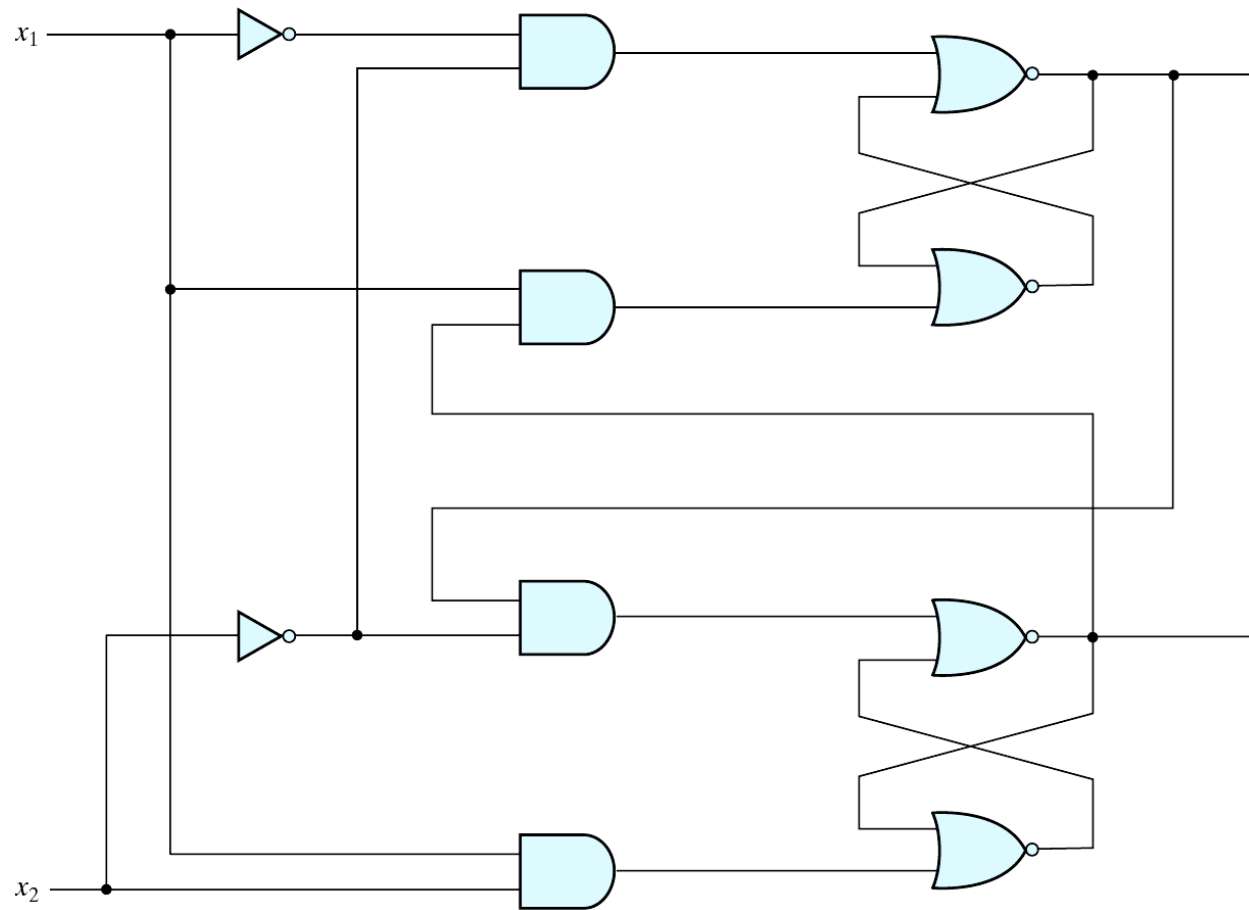
$SR$		00	01	11	10
$y$	0	1	1	0	0
	1	1	1	1	0

(d) Transition table

$S=0, R=0$  ( $S+R=0$ )  
 should not be used  
 $\Rightarrow S+R=1$  is  
 normal mode  
 (eq.  $S'R'=0$ )  
**should be carefully**  
 checked first, so it is  
 obtained  
 $Y = S' + Ry$

# Analysis Example

---



# Analysis Example

---

- ✓ The procedure for analyzing an asynchronous sequential circuit with SR latches can be summarized as follows:
  - Label each latch output with  $Y_i$  and its external feedback path with  $y_i$  for  $i=1,2,\dots,k$
  - Derive the Boolean functions for the  $S_i$  and  $R_i$  inputs in each latch.

$$S_1 = x_1 y_2$$

$$R_1 = x_1' x_2'$$

$$S_2 = x_1 x_2$$

$$R_2 = x_2' y_1$$

# Analysis Example

---

- Check whether  $SR = 0$  for each NOR latch or whether  $S'R' = 0$  for each NAND latch. (if either of these two conditions is not satisfied, there is a possibility that the circuit may not operate properly)

$$S_1 R_1 = x_1 y_2 x_1' x_2' = 0$$

$$S_2 R_2 = x_1 x_2 x_2' y_1 = 0$$

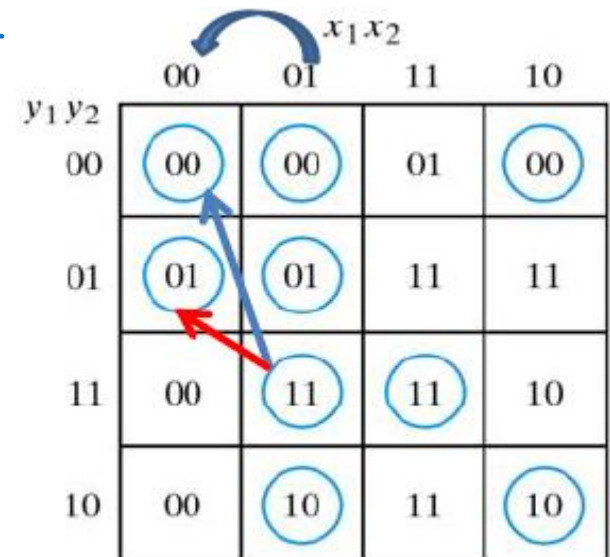
- Evaluate  $Y = S + R'y$  for each NOR latch or  $Y = S' + Ry$  for each NAND latch.

$$Y_1 = S_1 + R_1' y_1 = x_1 y_2 + x_1 y_1 + x_2 y_1$$

$$Y_2 = S_2 + R_2' y_2 = x_1 x_2 + x_2 y_2 + y_1' y_2$$

# Analysis Example

- Construct a map, with the  $y$ 's representing the rows and the  $x$  inputs representing the columns.
- Plot the value of  $Y=Y_1Y_2\dots Y_k$  in the map.
- Circle all stable states such that  $Y=y$ . The result is then the transition table.
- The transition table shows that the circuit is **stable**
- Race Conditions: there is a **critical race** condition when the circuit is initially in total state  $y_1y_2x_1x_2 = \underline{1101}$  and  $x_2$  changes from 1 to 0.
- The circuit should go to the total state 0000.
- If  $Y_1$  changes to 0 before  $Y_2$ , the circuit goes to total state 0100 instead of 0000.



		$x_1x_2$			
		00	01	11	10
$y_1y_2$	00	00	00	01	00
	01	01	01	11	11
	11	00	11	11	10
	10	00	10	11	10

Transition Table



# Implementation Procedure

---

- ✓ Procedure to implement an asynchronous sequential circuits with SR latches:
  - Given a transition table that specifies the excitation function  $Y = Y_1 Y_2 \dots Y_k$ , derive a pair of maps for each  $S_i$  and  $R_i$  using the latch excitation table
  - Derive the Boolean functions for each  $S_i$  and  $R_i$  (do not to make  $S_i$  and  $R_i$  equal to 1 in the same minterm square)
  - Draw the logic diagram using  $k$  latches together with the gates required to generate the  $S$  and  $R$  (for NAND latch, use the complemented values in step 2)

# Implementation Example

- ✓ Given a transition table that specifies the excitation function  $Y = Y_1 Y_2 \dots Y_k$ , then the general procedure for implementing a circuit with SR latches can be summarized as follows:

- Given a transition table

$y \backslash x_1 x_2$	00	01	11	10
0	0	0	0	1
1	0	0	1	1

(a) Transition table

$$Y = x_1 x'_2 + x_1 y$$

- Determine the Boolean functions for the S and R inputs of each latch (This is done by using the latch excitation table)

$y \backslash x_1 x_2$	00	01	11	10
0	0	0	0	1
1	0	0	X	X

(c) Map for  $S = x_1 x'_2$

$y \backslash x_1 x_2$	00	01	11	10
0	X	X	X	0
1	1	1	0	0

(d) Map for  $R = x'_1$

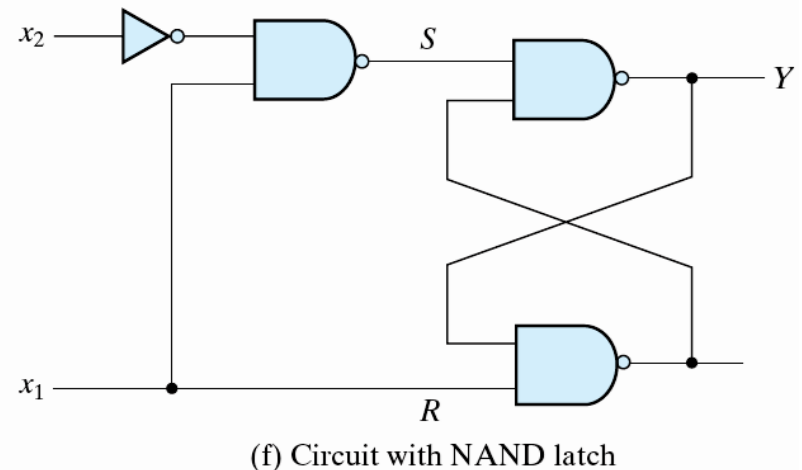
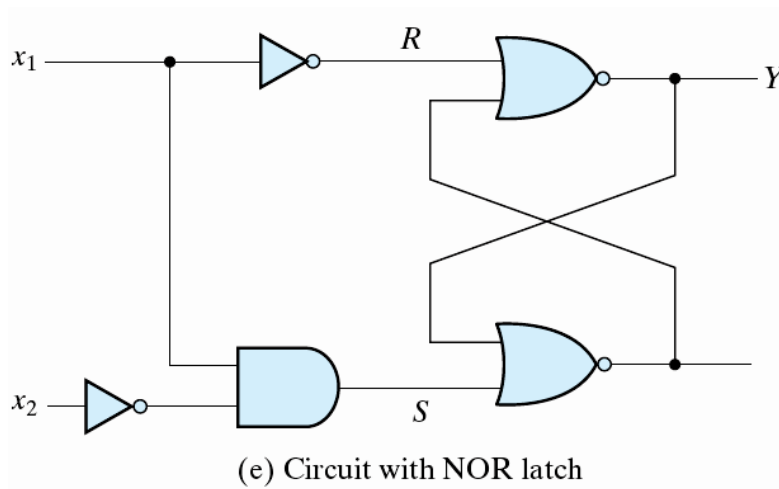
# Implementation Example

- From maps: the simplified Boolean functions are

$$S = x_1 x'_2 \quad \text{and} \quad R = x'_1 \quad \Rightarrow \quad \text{NOR latch}$$

$$S = (x_1 x'_2)' \quad \text{and} \quad R = x_1 \quad \Rightarrow \quad \text{NAND latch}$$

- Check whether  $SR=0$  for each NOR latch or whether  $S'R'=0$  for each NAND latch:  
 $SR = x_1 x'_2 x'_1 = 0$      $S'R' = (x_1 x'_2)' x_1 = 0$
- Draw the logic diagram, using k latches together with the gates required to generate the S and R Boolean functions obtained in step 1 (for NAND latches, use the complemented values)



# Primitive Flow Table

---

- ✓ Primitive flow table - has exactly one stable total state (internal state + input) per row
  - Can be further reduced
- ✓ To avoid the timing problems:
  - Only one input variable changes at a time
  - Networks reach a stable total state between input changes (Fundamental Mode)
- ✓ Every change in input changes the state

# Design procedure

---

1. Obtain a primitive table from specifications
2. Reduce flow table by merging rows in the primitive flow table
3. Assign binary state variables to each row of reduced table
4. Assign output values to dashes associated with unstable states to obtain the output map
5. Simplify Boolean functions for excitation and output variables;
6. Draw the logic diagram

# Design Example 1:

---

## ✓ Problem Statement:

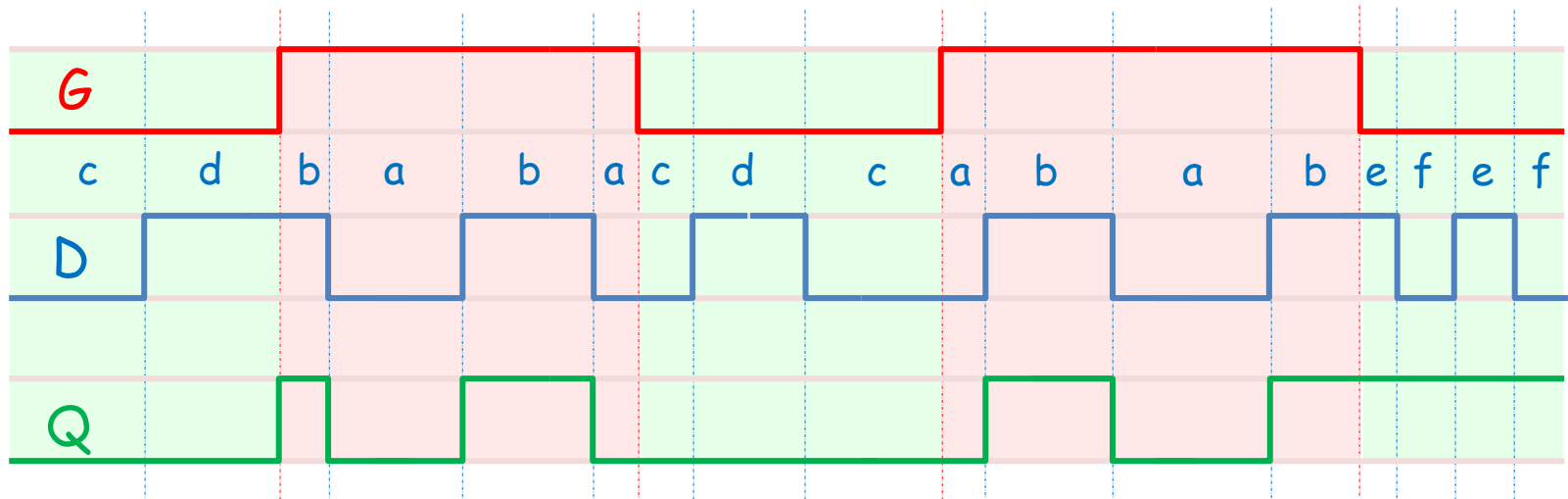
- Design a gated latch circuit (memory element) with two inputs,  $G$ (gate) and  $D$ (Data) and one output  $Q$ .
- The  $Q$  output will follow the  $D$  input as long as  $G=1$ . When  $G$  goes to 0, the information that was present at the  $D$  input at the time of transition is retained at the  $Q$  output.
  - $Q = D$  when  $G = 1$
  - $Q$  retains its value when  $G$  goes to 0

# Design Example 1:

## 1-Primitive Flow Table

- ✓ A primitive flow table is a flow table with only one stable total state (internal state + input) in each row.
- ✓ In order to form the primitive flow table, we first form a table with all possible total states, combinations of the inputs and internal states, simultaneous transitions of two input variables are not allowed

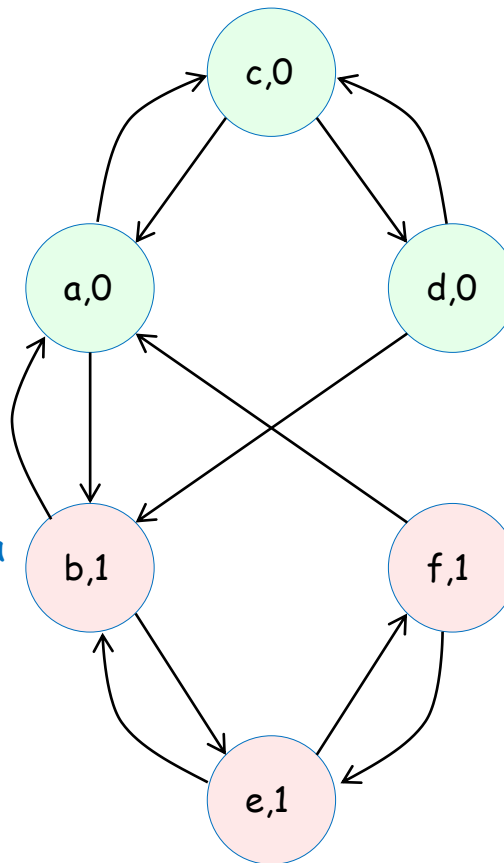
State	Inputs		Output	Comments
	<i>D</i>	<i>G</i>	<i>Q</i>	
<i>a</i>	0	1	0	$D = Q$ because $G = 1$
<i>b</i>	1	1	1	$D = Q$ because $G = 1$
<i>c</i>	0	0	0	After state <i>a</i> or <i>d</i>
<i>d</i>	1	0	0	After state <i>c</i>
<i>e</i>	1	0	1	After state <i>b</i> or <i>f</i>
<i>f</i>	0	0	1	After state <i>e</i>



# Design Example 1

## 1-Primitive Flow Table

- First, we fill in one square in each row belonging to the stable state in that row.
- Next we note that both inputs are not allowed to change at the same time, we enter dash marks in each row that differs in two or more variables from the input variables associated with the stable state.
- All outputs associated with unstable states are marked with a dash to indicate don't care conditions for the next state and output.
- Next it is necessary to find values for two more squares in each row. The comments listed in the previous table may help in deriving the necessary information.



	DG			
	00	01	11	10
a	c, -	<b>a</b> , 0	b, -	-, -
b	-, -	a, -	<b>b</b> , 1	e, -
c	<b>c</b> , 0	a, -	-, -	d, -
d	c, -	-, -	b, -	<b>d</b> , 0
e	f, -	-, -	b, -	<b>e</b> , 1
f	<b>f</b> , 1	a, -	-, -	e, -



# Design Example 1

## 2-Reduction of the Primitive Flow Table

- Two or more rows can be merged into one row if there are non-conflicting states and outputs in every columns.
- After merged into one row:
  - Don't care entries are overwritten
  - Stable states and output values are included
  - A common symbol is given to the merged row

		DG			
		00	01	11	10
States	a	c, -	a, 0	b, -	-, -
	c	c, 0	a, -	-, -	d, -
	d	c, -	-, -	b, -	d, 0

		DG			
		00	01	11	10
States	b	-, -	a, -	b, 1	e, -
	e	f, -	-, -	b, -	e, 1
	f	f, 1	a, -	-, -	e, -

(a) States that are candidates for merging

		DG			
		00	01	11	10
States	a, c, d	c, 0	a, 0	b, -	d, 0
	b, e, f	f, 1	a, -	b, 1	e, 1

		DG			
		00	01	11	10
States	a	a, 0	a, 0	b, -	a, 0
	b	b, 1	a, -	b, 1	b, 1

(b) Reduced table (two alternatives)

# Design Example 1

## 3-Transition Table and Logic Diagram

- ✓ In order to obtain the circuit described by the reduced flow table, it is necessary to assign a distinct binary value to each state.
- ✓ This converts the flow table to a transition table.
- ✓ A binary state assignment must be made to ensure that the circuit will be free of critical race.

**a=0, b=1 in this example**

Transition table and output map for gated Latch

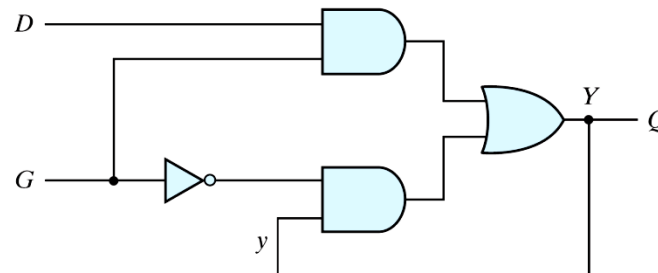
$DG$ $y$		00	01	11	10
		0	0	1	0
1		1	0	1	1

(a)  $Y = DG + G'y$

$DG$ $y$		00	01	11	10
		0	0	0	0
1		1	1	1	1

(b)  $Q = Y$

Gated-latch logic diagram



# Design Example 1

## 4. Implementation with SR Latch

	<i>DG</i>			
	00	01	11	10
<i>y</i>				
0	0	0	1	0
1	X	0	X	X

(a)  $S = DG$

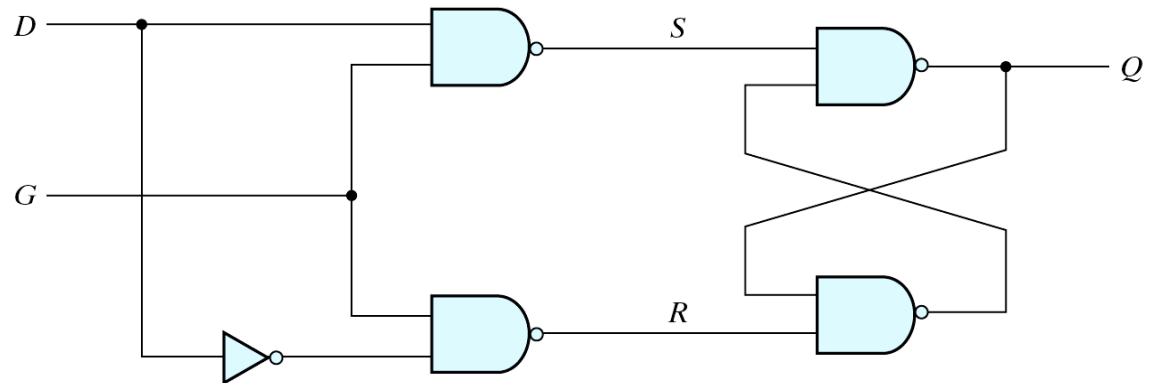
	<i>DG</i>			
	00	01	11	10
<i>y</i>				
0	X	X	0	X
1	0	1	0	0

$R = D'G$

(a) Maps for  $S$  and  $R$

- ✓ Listed according to the transition table and the excitation table of SR latch

Circuit with SR latch



(b) Logic diagram

# Design Example:

---

## 5- Assigning Outputs to Unstable States

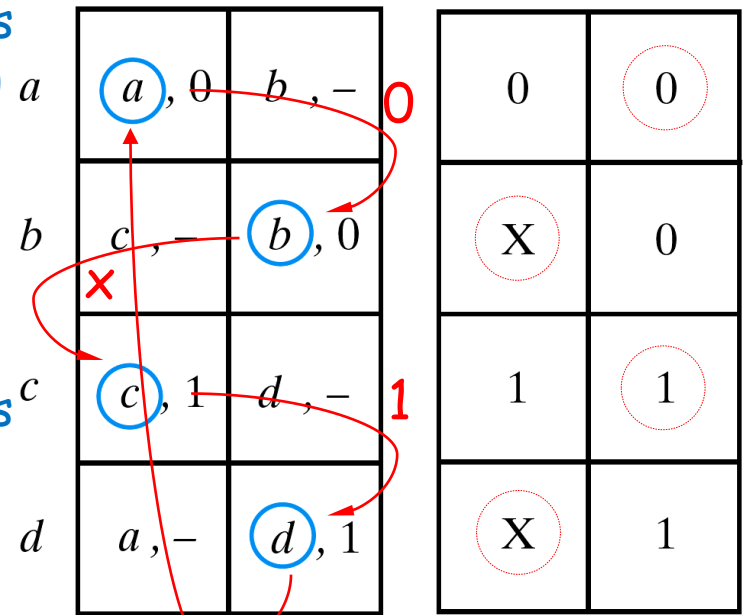
- ✓ While the stable states in a flow table have specific output values associated with them, the unstable states have unspecified output entries designated by a dash.
  - ✓ These unspecified output values must be chosen so that no momentary false outputs occur when the circuit switches between stable states.
- 
- *If the two stable states have the same output value, then an unstable state that is a transient state between them must have the same output.*
  - *If an output variable is to change as a result of a state change, then this variable is assigned a don't care condition.*

# Design Example 1

## 5- Assigning Outputs to Unstable States

Ex:

- If a changes to b, the two stable states have the same output value = 0 ( $0 \Rightarrow 0: 0$ ) the transient unstable state b in the first row must have the same output value = 0  
if c changes to d same for  $1 \Rightarrow 1: 1$
- If b changes to c, the two stable states have different output values  $0 \Rightarrow 1: x$  the transient unstable state c in the second row is assigned a don't care condition  
if d changes to a same for  $1 \Rightarrow 0: x$



(a) Flow table   b) Output assignment

# Equivalent State 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**
- ✓ 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.

# 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 from 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 recorded with check marks.

Present State	Next State		Output	
	X=0	X=1	X=0	X=1
<i>a</i>	<i>d</i>	<i>b</i>	0	0
<i>b</i>	<i>e</i>	<i>a</i>	0	0
<i>c</i>	<i>g</i>	<i>f</i>	0	1
<i>d</i>	<i>a</i>	<i>d</i>	1	0
<i>e</i>	<i>a</i>	<i>d</i>	1	0
<i>f</i>	<i>c</i>	<i>b</i>	0	0
<i>g</i>	<i>a</i>	<i>e</i>	1	0

<i>b</i>	<i>d, e</i> ✓					
<i>c</i>	×	×				
<i>d</i>	×	×	×			
<i>e</i>	×	×	×	✓		
<i>f</i>	<i>c, d</i> ×	<i>c, e</i> × <i>a, b</i>	×	×	×	
<i>g</i>	×	×	×	<i>d, e</i> ✓	<i>d, e</i> ✓	×
	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>

# 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:

b	d, e ✓					
c	x	x				
d	x	x	x			
e	x	x	x	✓		
f	c, d x	c, e x a, b	x	x	x	
g	x	x	x	d, e ✓	d, e ✓	x
	a	b	c	d	e	f

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



# Implication Table

---

- the equivalent states
  - (a,b), (d,e), (d,g), (e,g)
- the reduced states
  - (a,b), (c), (d,e,g), (f)
- the state table:

<b>Present State</b>	<b>Next State</b>		<b>Output</b>	
	<b><math>x = 0</math></b>	<b><math>x = 1</math></b>	<b><math>x = 0</math></b>	<b><math>x = 1</math></b>
<i>a</i>	<i>d</i>	<i>a</i>	0	0
<i>c</i>	<i>d</i>	<i>f</i>	0	1
<i>d</i>	<i>a</i>	<i>d</i>	1	0
<i>f</i>	<i>c</i>	<i>a</i>	0	0

# Implication Table (Implementation):

- ✓ 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 from 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 recorded with check marks.

	<i>x</i>	
	0	1
<i>A</i>	<i>C</i> /1	<i>B</i> /0
<i>B</i>	<i>C</i> /1	<i>E</i> /0
<i>C</i>	<i>B</i> /1	<i>E</i> /0
<i>D</i>	<i>D</i> /0	<i>B</i> /1
<i>E</i>	<i>E</i> /0	<i>A</i> /1

(a)

# 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

# Incompletely Specified Circuits

---

- ✓ Next states and/or outputs are not specified for all states
- ✓ *Applicable input sequences*: an input sequence is applicable to state,  $S_i$ , of an incompletely specified circuit if and only if when the circuit is in state  $S_i$  and the input sequence is applied, all next states are specified except for possibly the last input of the sequence.
- ✓ *Compatible states*: two states  $S_i$  and  $S_j$  are compatible if and only if for each input sequence applicable to both states the same output sequence will be produced when the outputs are specified.
- ✓ *Compatible states*: two states  $S_i$  and  $S_j$  are compatible if and only if the following conditions are satisfied for any possible input  $I_p$ 
  - The outputs produced by  $S_i$  and  $S_j$  are the same, when both are specified
  - The next states  $S_k$  and  $S_l$  are compatible, when both are specified.
- ✓ *Incompatible states*: two states are said to be incompatible if they are not compatible.

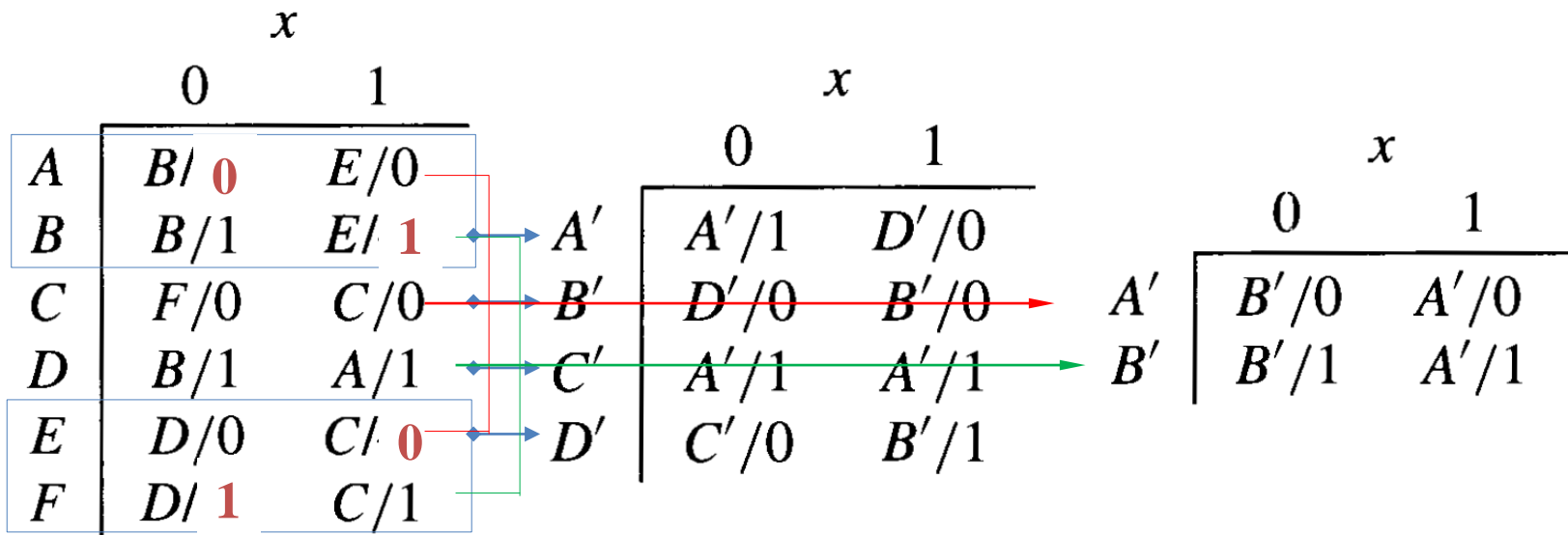
# Compatibility Relations

---

- ✓ *Compatibility relation:* let  $R$  be a relation on a set  $S$ .  $R$  is a compatibility relation on  $S$  if and only if it is reflexive and symmetric. A compatibility relation on a set partitions the set into compatibility classes. They are typically not disjoint.
- ✓ Compatible pairs may be found using implication tables
- ✓ Maximal compatibles may be found using merger diagrams

# Incompletely specified circuits

- ✓ Finding compatible states by inspection



# 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*

$$P_1 = (ABCDEFGG)$$

$$P_2 = (ABDG)(CEF)$$

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

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

$$P_5 = P_4$$

*Let consider both unspecified outputs = 1*

$$P_1 = (ABCDEFGG)$$

$$P_2 = (AD)(BCEFG)$$

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

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



# Equivalent and Compatible States

- ✓ **completely** specified state table  $\Rightarrow$  **equivalence**

If a and b are equivalent, and b and c are equivalent then also a and c are equivalent:  $(a,b), (b,c) \Rightarrow (a,c)$

- ✓ **uncompletely** specified state table  $\Rightarrow$  **compatibility**

If a and b are compatible, and b and c are compatible not necessarily a and c are compatible:  $(a,b), (b,c) \not\Rightarrow (a,c)$

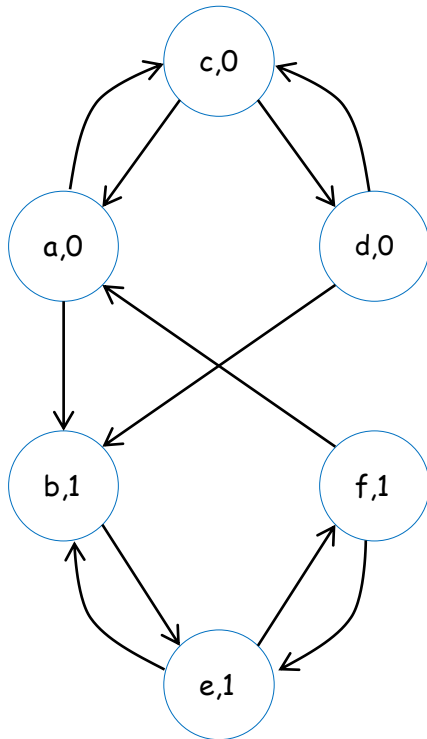
y \	0	1	
a	(A)0	C,1	b
b	(B)-	C,-	c
c	(C)1	(C)1	

✓	
X	✓
a	b

# Compatible Pairs

- ✓ Implication tables are used to find compatible states.
  - We can adjust the dashes to fit any desired condition.
  - Must have no conflict in the output values to be merged.



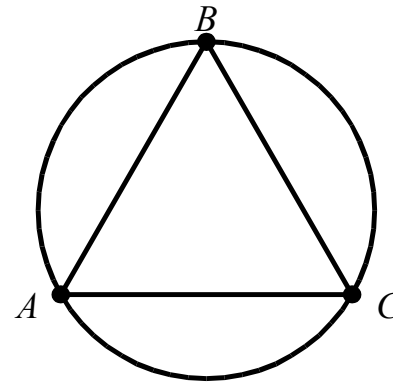
	00	01	11	10
a	c, -	<b>a</b> , 0	b, -	-, -
b	-, -	a, -	<b>b</b> , 1	e, -
c	<b>c</b> , 0	a, -	-, -	d, -
d	c, -	-, -	b, -	<b>d</b> , 0
e	f, -	-, -	b, -	<b>e</b> , 1
f	<b>f</b> , 1	a, -	-, -	e, -

(a) Primitive flow table

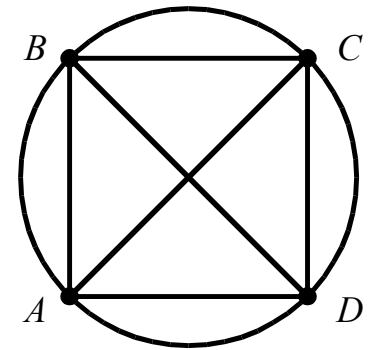
# Merger diagrams

---

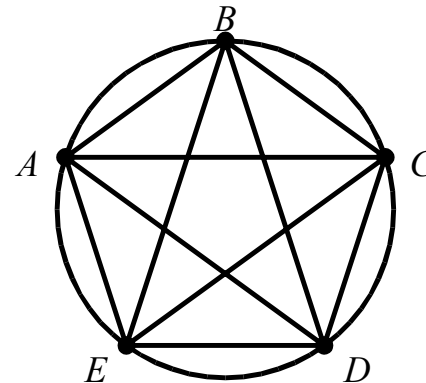
- ✓ States are represented as dot in a circle
- ✓ Lines connect states couples compatible
- ✓ **Maximal sets** can be identified as those sets in which every states is connected to every other state by a line segment
- ✓ The rules for extracting maximal sets from a merger diagram are:
  - Make each maximal 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



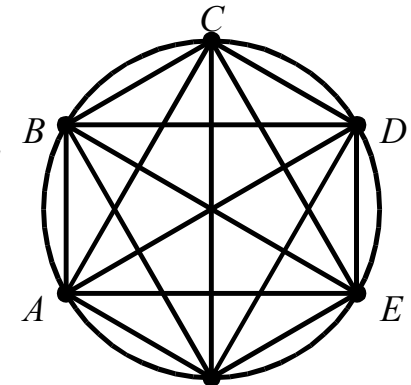
(a)



(b)



(c)

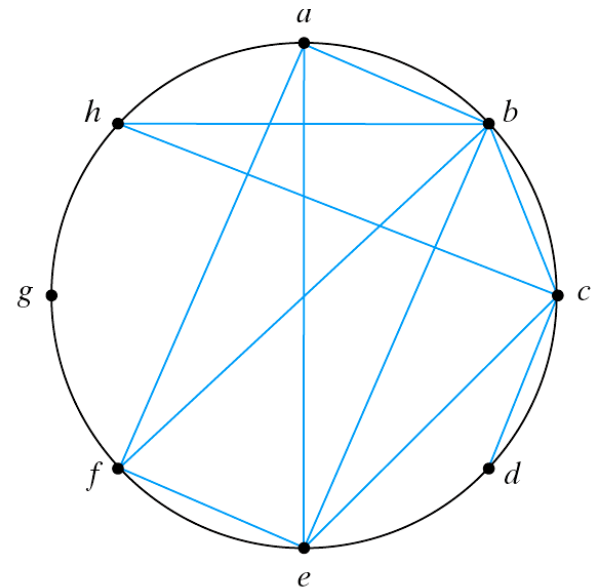
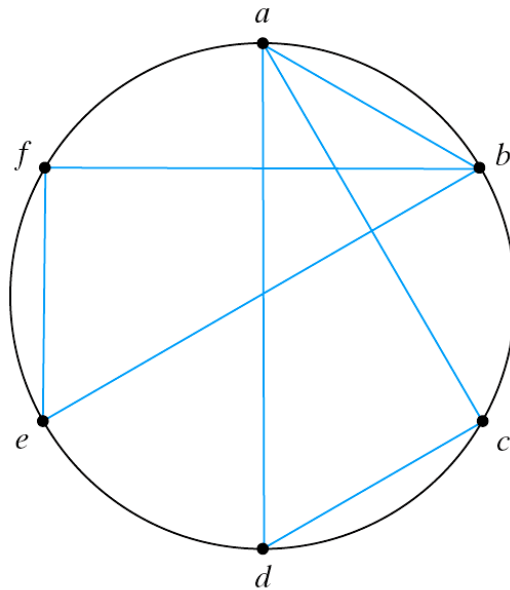


(d)

# 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  $\rightarrow$  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



# 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)

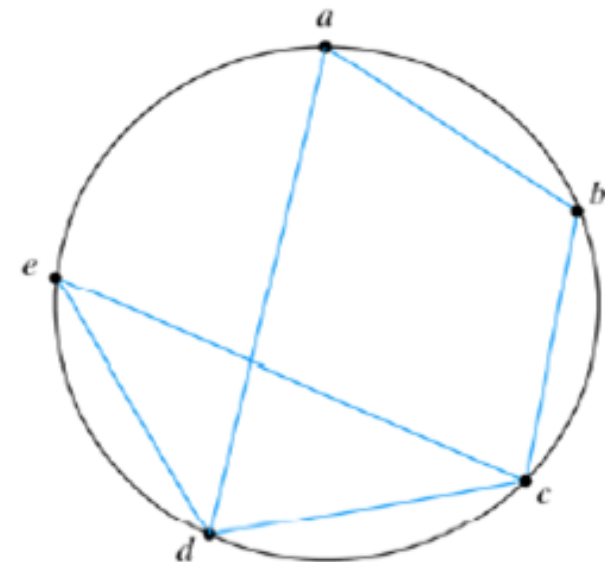
- From the aside implication table, we have the following compatible: pairs:  $(a, b)$   $(a, d)$   $(b, c)$   $(c, d)$   $(c, e)$   $(d, e)$
- From the merger diagram, we determine the maximal compatibles:  $(a, b)$   $(a, d)$   $(b, c)$   $(c, d, e)$
- If we choose the two compatibles:  $(a, b)$   $(c, d, e)$

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

Closure table

- 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

	a	b	c	d
b	$b, c \checkmark$			
c	$\times$	$d, e \checkmark$		
d	$b, c \checkmark$	$\times$	$a, d \checkmark$	
e	$\times$	$\times$	$\checkmark$	$b, c \checkmark$



# Minimization of compatible classes

---

Select a set of compatibility classes so that the following conditions are satisfied:

- ✓ *Completeness*: all states of the original machine must be covered
- ✓ *Consistency*: the chosen set of compatibility classes must be closed
- ✓ *Minimality*: the smallest number of compatibility classes is used

# Bounding the number of states

---

- ✓ Unfortunately the process of selecting the set of compatibility classes that meets the three conditions **must be done by trial and error**.
- ✓ Let  $U$  be the upper bound on the number of states needed in the minimized circuit.

Then  $U = \text{minimum}(\text{NSMC}, \text{NSOC})$

- where NSMC = the number of sets of maximal compatibles
- and NSOC = the number of states in the original circuit
- ✓ Let  $L$  be the lower bound on the number of states needed in the minimized circuit

Then  $L = \text{maximum}(\text{NSMI}_1, \text{NSMI}_2, \dots, \text{NSMI}_i)$

- where  $\text{NSMI}_i$  = the number of states in the  $i$ th group of the set of maximal incompatibles of the original circuit.



# State Reduction Algorithm

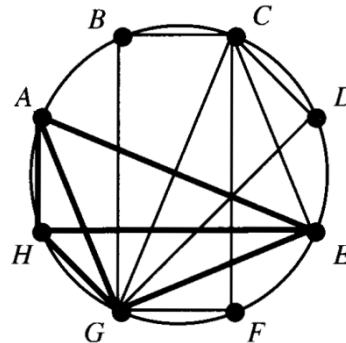
---

- ✓ Step 1 -- find the maximal compatibles
- ✓ Step 2 -- find the maximal incompatibles
- ✓ Step 3 -- Find the upper and lower bounds on the number of states needed
- ✓ Step 4 -- Find a set of compatibility classes that is complete, consistent, and minimal
- ✓ Step 5 -- Produce the minimum state table

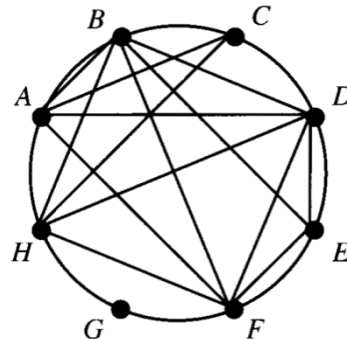
# Generating Maximal Compatibles and Incompatibles

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

B	AC						
C		BG AE					
D	AC	BC AC	CG CE				
E	√	AB AC	AG	AC			
F	AD AC	BD	DG AE	CD AC	AD AC		
G	CG	AG	EG	CG	AG CG	DG AG	
H	CD	AD	GH DE	CH CD	AH CD	DH AD	DG
	A	B	C	D	E	F	G



Maximum Compatibles  
 (AEGH)(BCG)(CDG)  
 (CEG)(CFG)  
 U=5



Maximum Incompatibles  
 (ABDF)(AC)(BDEF)  
 (CH)(BDFH)  
 L=4

# Reduced state table

---

- ✓ Closure table: treat maximum compatibles as states and find their sets of next states

	x			x	
	0	1		0	1
(AEGH)	AGH	CDG	A'	A'/1	C'/1
(BCG)	BG	AEG	B'	B'/-	A'/0
(CDG)	CG	CEG	C'	B', C', D', E'/1	D'/0
(CEG)	AG	CEG	D'	A'/1	D'/0
(CFG)	DG	AEG	E'	C'/-	A'/0

Closure table

Reduced state table

- ✓ All maximal compatibles are used as states of the reduced machine. Hence, the final five states are:

$$A' = (AEGH), \quad D' = (CEG)$$

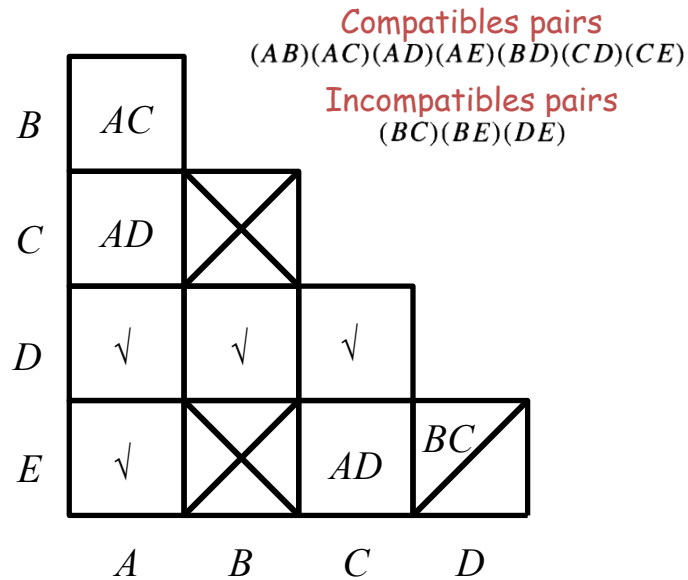
$$B' = (BCG), \quad E' = (CFG)$$

$$C' = (CDG)$$

# Example -- State reduction problem

	<i>x</i>	
	0	1
<i>A</i>	<i>A</i> /-	-/-
<i>B</i>	<i>C</i> /1	<i>B</i> /0
<i>C</i>	<i>D</i> /0	-/1
<i>D</i>	-/-	<i>B</i> /-
<i>E</i>	<i>A</i> /0	<i>C</i> /1

(a)



(b)

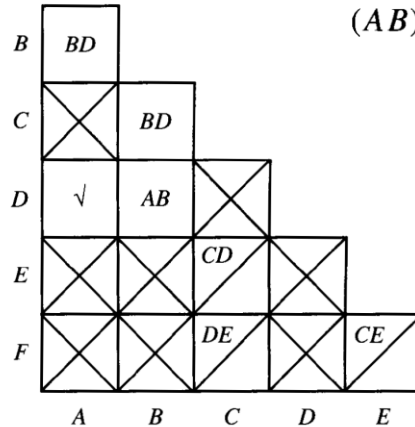
Maximum  
Incompatibles  
(*BE*)(*BC*)(*ED*)  
**L=2**

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

# Another state table reduction problem

	<i>x</i>	
	0	1
<i>A</i>	<i>B</i> /1	<i>D</i> /0
<i>B</i>	—/—	<i>B</i> /0
<i>C</i>	<i>E</i> /0	<i>D</i> /—
<i>D</i>	<i>B</i> /1	<i>A</i> /0
<i>E</i>	—/—	<i>C</i> /1
<i>F</i>	—/0	<i>E</i> /1

(a)



(b)

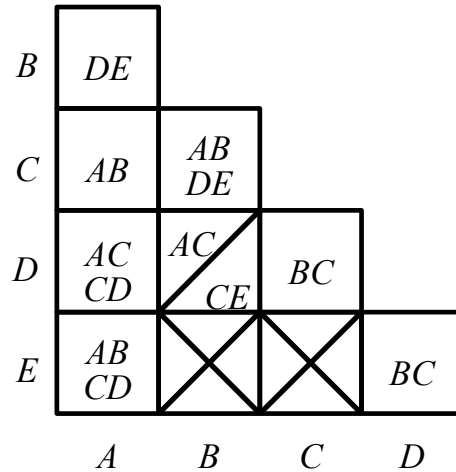
Compatible pairs  
 $(AB)(AD)(BC)(BD)$

# Yet another state reduction problem

---

	<i>x</i>	
	0	1
<i>A</i>	<i>D</i> /-	<i>A</i> /-
<i>B</i>	<i>E</i> /0	<i>A</i> /-
<i>C</i>	<i>D</i> /0	<i>B</i> /-
<i>D</i>	<i>C</i> /-	<i>C</i> /-
<i>E</i>	<i>C</i> /1	<i>B</i> /-

(a)



(b)

Maximum  
Compatibles  
(*ABC*)(*ACD*)(*ADE*)  
**U=3**

Maximum  
Incompatibles  
(*BD*)(*BE*)(*CE*)  
**L=2**

# 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
$a$	$a$	$b$	$c$	$a$
$b$	$a$	$b$	$b$	$c$
$c$	$a$	$c$	$c$	$c$

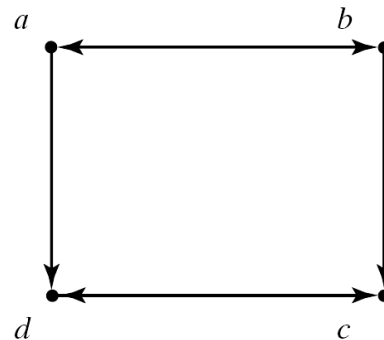
(a) Flow table

# 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  
 $00 \Rightarrow 10 \Rightarrow 11$  (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)



# Shared-Row Method

---

- ✓ The shared row is not assigned to any specific stable state
- ✓ Used to convert a critical race into a cycle that goes through adjacent transitions between two stable states
- ✓ May require more extra rows

# Unique State Assignments for Four States

NUMBER OF STATE ASSIGNMENTS

$N_S$	$N_{FF}$	$N_{SA}$	$N_{UA}$
1	0	—	—
2	1	2	1
3	2	24	3
4	2	24	3
5	3	6,720	140
6	3	20,160	420
7	3	40,320	840
8	3	40,320	840
9	4	$4.15 \times 10^9$	10,810,800
10	4	$2.91 \times 10^{10}$	75,675,600

$$N_{UA} = \frac{(2^{N_{FF}} - 1)!}{(2^{N_{FF}} - N_S)! N_{FF}!}$$

Present state	$x$	
	0	1
A	A/0	B/0
B	A/0	C/0
C	C/0	D/0
D	C/1	A/0

(a)

States	Assignments		
	1 $y_1 y_2$	2 $y_1 y_2$	3 $y_1 y_2$
A	00	10	00
B	01	11	10
C	11	01	11
D	10	00	01

(b)

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

$$D_2 = y_1 \bar{x} + \bar{y}_1 x$$

$$D_1 = y_1 \bar{x} + \bar{y}_2 x$$

$$D_2 = \bar{y}_1 \bar{x} + y_1 x$$

$$D_1 = y_2 \bar{x} + \bar{y}_2 x$$

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

# State adjacencies for assignments

---

## ✓ State adjacencies for four-state assignments

**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.

## 4-Row Flow-Table Example

- ✓ A flow table with 4 states requires an assignment of two state variables.
- ✓ If there were no transitions in the diagonal direction (from a to c or from b to d), it would be possible to find adjacent assignment for the remaining 4 transitions.

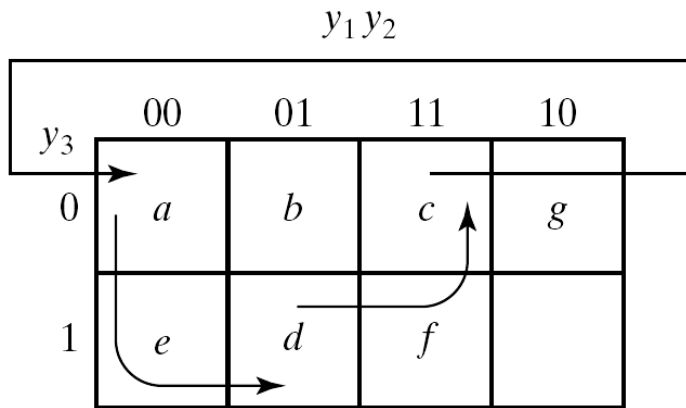
	00	01	11	10
a	b	<b>a</b>	d	<b>a</b>
b	<b>b</b>	d	<b>b</b>	a
c	<b>c</b>	a	b	<b>c</b>
d	c	<b>d</b>	<b>d</b>	c

(a) Flow table

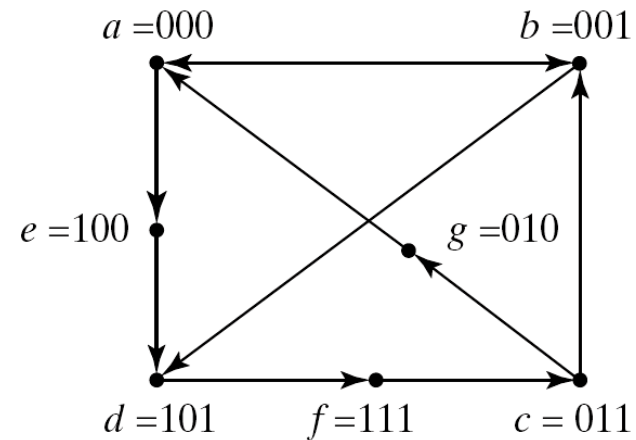
- ✓ In general in order to satisfy the adjacency requirement, at least 3 binary variables are needed.

# 4-Row Flow-Table Example

- The following state assignment map is suitable for any table.
  - a, b, c, and d are the original states.
  - e, f, and g are extra states.
  - States placed in adjacent squares in the map will have adjacent assignments



(a) Binary assignment



(b) Transition diagram

# 4-Row Flow-Table Example

✓ To produce cycles:

- The transition from a to d must be directed through the extra state e
- The transition from c to a must be directed through the extra state g
- The transition from d to c must be directed through the extra state f

	00	01	11	10
a	b	a	d	a
b	b	d	b	a
c	c	a	b	c
d	c	d	d	c

(a) Flow table

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

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

Although the flow table has 7 rows, there are only 4 stable states.

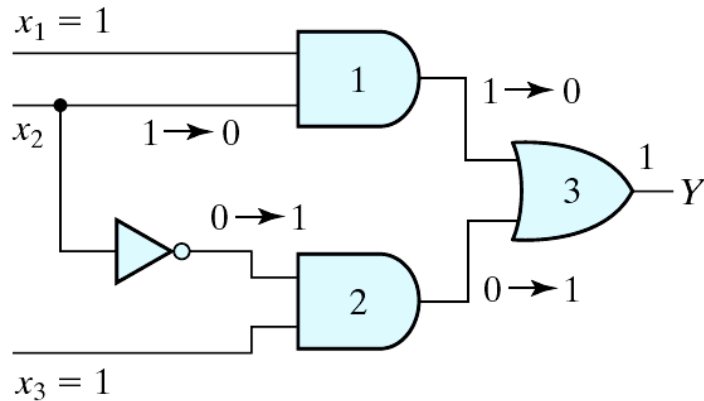


# Hazards

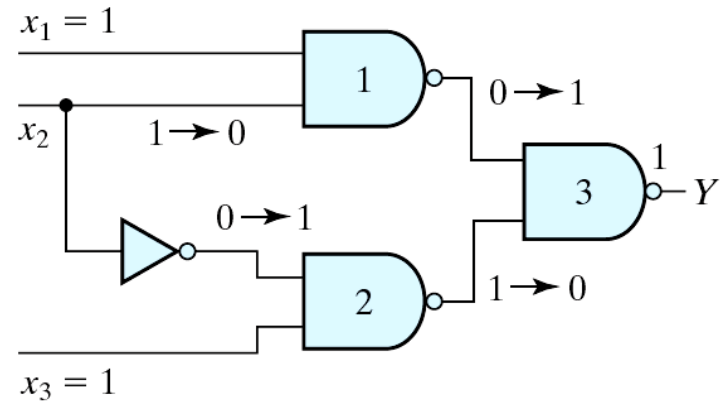
---

- ✓ 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:
  - Hypothesis:
    - Operated in fundamental mode with only one input changing at any time
  - Objectives:
    - Free of critical races
    - Free of hazards

# Hazards in combinational circuits



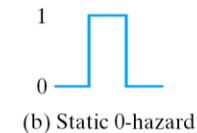
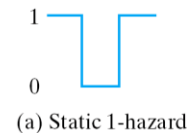
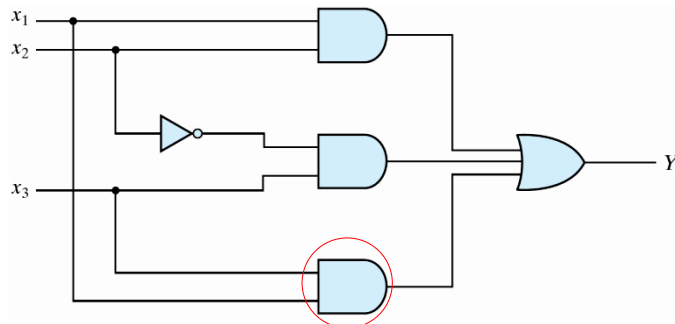
(a) AND-OR circuit



(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)  $Y = x_1 x_2 + x'_2 x_3$

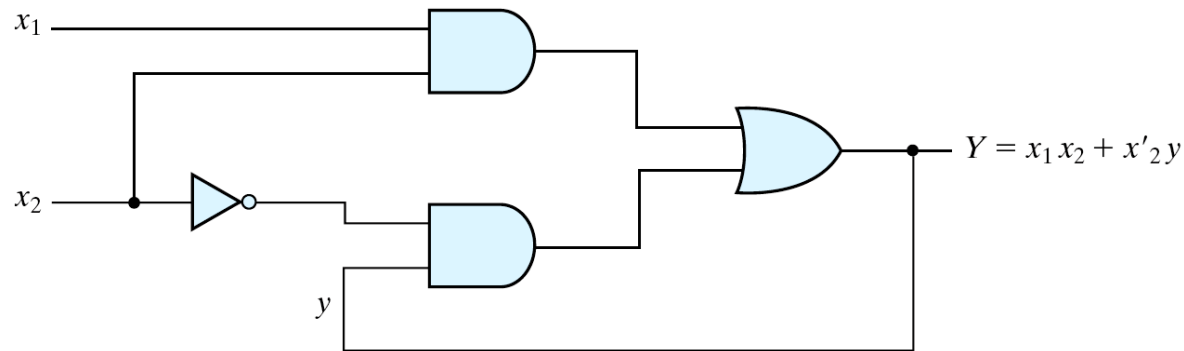
	$x_2 x_3$			
	00	01	11	10
$x_1$ 0		1		
$x_1$ 1		1	1	1

(b)  $Y = x_1 x_2 + x'_2 x_3 + x_1 x_3$

	$x_2 x_3$			
	00	01	11	10
$x_1$ 0		1		
$x_1$ 1		1	1	1

# Hazards in sequential circuits

✓ An asynchronous example:



(a) Logic diagram

	$x_1 x_2$			
	00	01	11	10
$y$				
0	0	0	1	0
1	1	0	1	1

(b) Transition table

$111 \rightarrow 110$   
 $111 \rightarrow 010$

	$x_1 x_2$			
	00	01	11	10
$y$				
0			1	
1	1		1	1

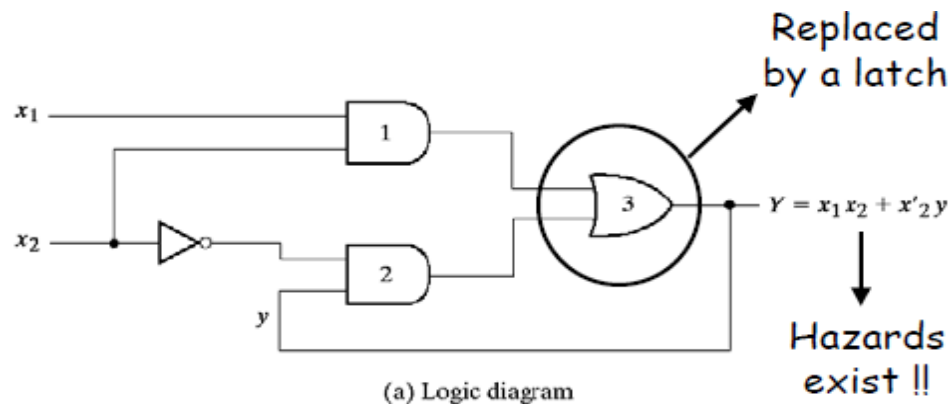
(c) Map for  $Y$

✓ Implementation with SR latches

- a momentary 0 signal at the S or R inputs of NOR latch has no effect
- a momentary 1 signal at the S or R inputs of NAND latch has no effect

# 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



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

(b) Transition table

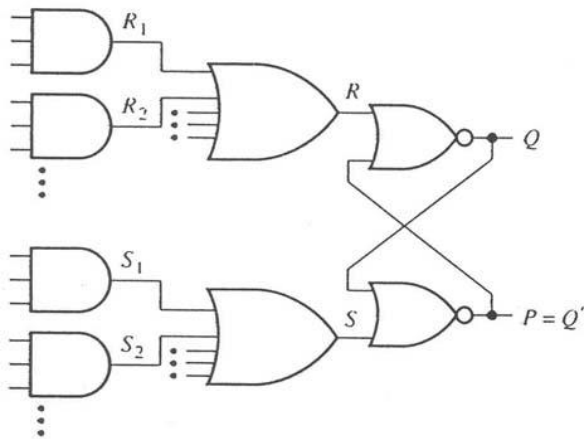
	$x_1x_2$			
	00	01	11	10
y				
0			1	
1	1		1	1

(c) Map for Y

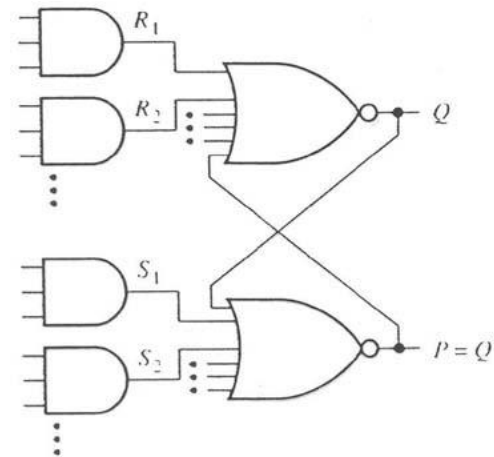
# Hazard-Free Realization

- ✓ 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

$$S = AB + CD$$

$$R = A'C$$

- ✓ Since this is a NAND latch we must use the complement value for S and R

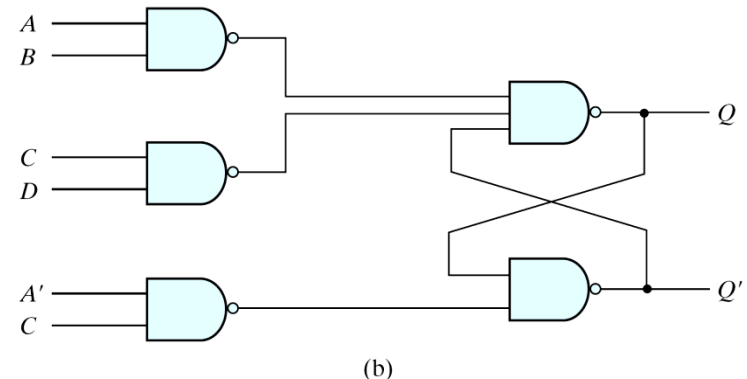
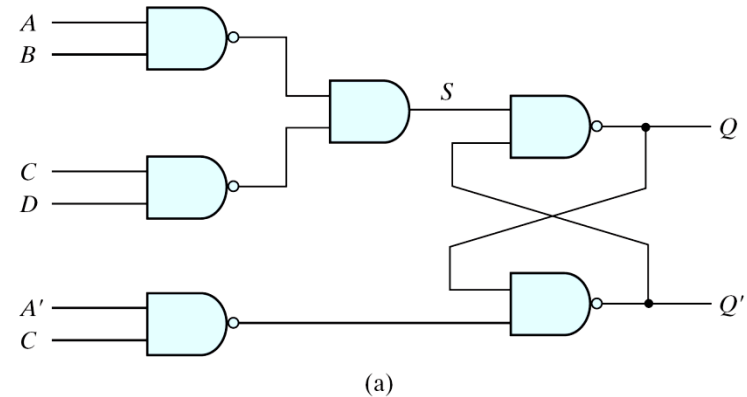
$$S = (AB + CD)' = (AB)'(CD)'$$

$$R = (A'C)'$$

- ✓ The Boolean function for output is

$$Q = (Q'S)' = [Q' (AB)'(CD)']'$$

- ✓ 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 !!

# Design Example

---

## Recommended Design Procedure:

1. State the design specifications.
2. Derive a Primitive Flow Table.
3. Reduce the Flow Table by merging rows.
4. Make a race-free binary state assignment.
5. Obtain the transition table and output map.
6. Obtain the logic diagram using SR latches.



# Design Example

---

## 1) Design Specifications:

It is necessary to design a negative-edge-triggered T flip-flop. The circuit has two inputs T (toggle) and C (clock) and one output Q. The output state is complemented if  $T=1$  and the clock changes from 1 to 0 (negative-edge-triggering). Otherwise, under all input condition, the output remains unchanged.

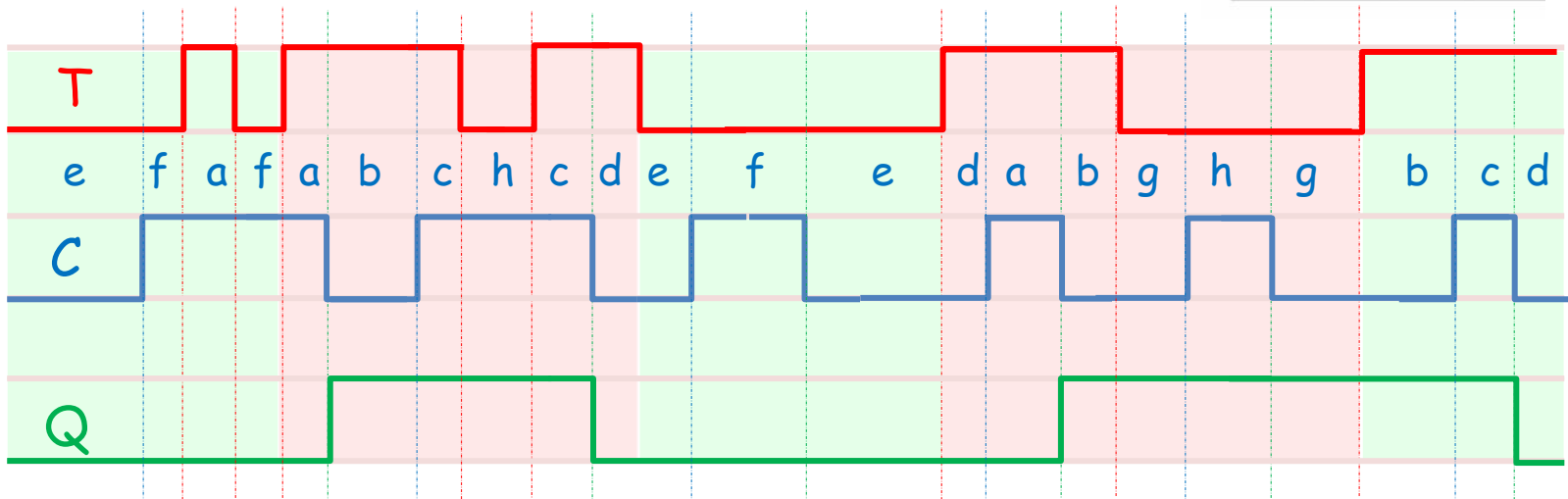
- A Negative-Edge-Triggered T FF
- Two inputs : T, C
- Flip-Flop changes state when  $T = 1$  and C changes from 1 to 0
- Q remains constant under all other conditions
- T and C do not change simultaneously

# Design Example

## 2) Primitive Flow Table

State	Inputs		Output	Comments
	$T$	$C$	$Q$	
$a$	1	1	0	Initial output is 0
$b$	1	0	1	After state $a$
$c$	1	1	1	Initial output is 1
$d$	1	0	0	After state $c$
$e$	0	0	0	After states $d$ or $f$
$f$	0	1	0	After states $e$ or $a$
$g$	0	0	1	After states $b$ or $h$
$h$	0	1	1	After states $g$ or $c$

	$TC$			
	00	01	11	10
$a$	-, -	$f$ , -	$a$ , 0	$b$ , -
$b$	$g$ , -	-, -	$c$ , -	$b$ , 1
$c$	-, -	$h$ , -	$c$ , 1	$d$ , -
$d$	$e$ , -	-, -	$a$ , -	$d$ , 0
$e$	$e$ , 0	$f$ , -	-, -	$d$ , -
$f$	$e$ , -	$f$ , 0	$a$ , -	-, -
$g$	$g$ , 1	$h$ , -	-, -	$b$ , -
$h$	$g$ , -	$h$ , 1	$c$ , -	-, -



# Design Example

## 3) Merging of the Flow Table

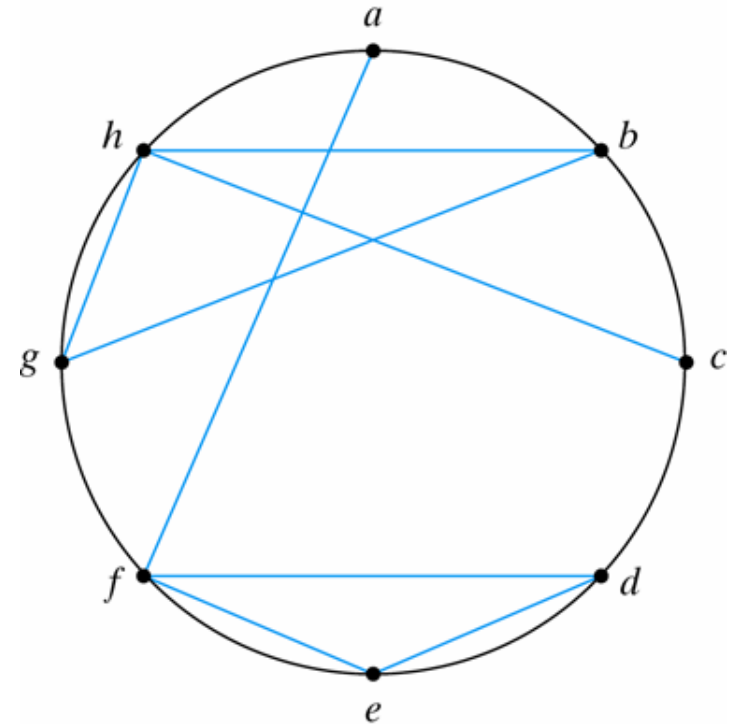
Implication Table

b	a, c X						
c	X	b, d X					
d	b, d X		X	a, c X			
e	b, d X	e, g X b, d X	f, h X	✓			
f	✓	e, g X a, c X	f, h X a, c X	✓	✓		
g	f, h X	✓	b, d X	e, g X b, d X	X	e, g X f, h X	
h	f, h X a, c X	✓	✓	d, e X c, f X	e, g X f, h X	X	✓
	a	b	c	d	e	f	g

Compatible pairs:

$(a, f) (b, g) (b, h) (c, h) (d, e) (d, f) (e, f) (g, h)$

Merger Diagram



The maximal compatibles pairs are:

$(a, f) (b, g, h) (c, h) (d, e, f)$

# Design Example

- ✓ In this particular example, the minimal collection of compatibles is also the maximal compatibles set that satisfy also the closed condition:

(a, f) (b, g, h) (c, h) (d, e, f)

	00	01	11	10
TC				
a, f	e, -	(f), 0	(a), 0	b, -
b, g, h	(g), 1	(h), 1	c, -	(b), 1
c, h	g, 1	(h), 1	(c), 1	d, -
d, e, f	(e), 0	(f), 0	a, -	(d), 0

(a)

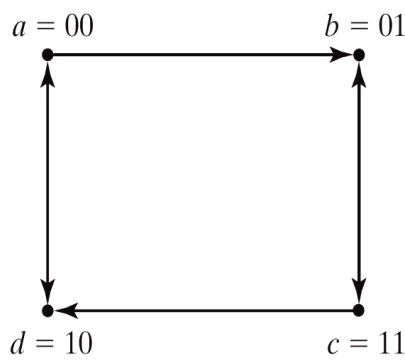
	00	01	11	10
TC				
a	d, -	(a), 0	(a), 0	(b), -
b	(b), 1	(b), 1	c, -	(b), 1
c	b, -	(c), 1	(c), 1	d, -
d	(d), 0	(d), 0	a, -	(d), 0

(b)

# Design Example

## 4) State Assignment and Transition Table

- ✓ No diagonal lines in the transition diagram: No need to add extra states (race-free binary state assignment!)



Transition diagram showing four states:  $a = 00$ ,  $b = 01$ ,  $c = 11$ ,  $d = 10$ . Transitions:  $a \rightarrow b$ ,  $b \rightarrow c$ ,  $c \rightarrow d$ ,  $d \rightarrow a$ .

		TC			
		00	01	11	10
$y_1 y_2$	00	10	00	00	01
	01	01	01	11	01
	11	01	11	11	10
	10	10	10	00	10

(a) Transition table

		TC			
		00	01	11	10
$y_1 y_2$	00	0	0	0	X
	01	1	1	1	1
	11	1	1	1	X
	10	0	0	0	0

(b) Output map  $Q = y_2$

# Design Example

## 5) Logic Diagram

	<i>TC</i>			
	00	01	11	10
$y_1y_2$				
00	1	0	0	0
01	0	0	1	0
11	0	X	X	X
10	X	X	0	X

(a)  $S_1 = y_2 TC + y'_2 TC'$

	<i>TC</i>			
	00	01	11	10
$y_1y_2$				
00	0	X	X	X
01	X	X	0	X
11	1	0	0	0
10	0	0	1	0

(b)  $R_1 = y_2 TC' + y'_2 TC$

	00	01	11	10
$y_1y_2$				
00	0	0	0	1
01	X	X	X	X
11	X	X	X	0
10	0	0	0	0

(c)  $S_2 = y'_1 TC'$

	00	01	11	10
$y_1y_2$				
00	X	X	X	0
01	0	0	0	0
11	0	0	0	1
10	X	X	X	X

(d)  $R_2 = y_1 TC'$

