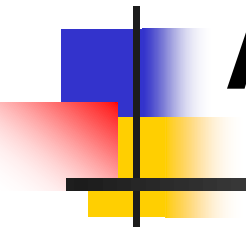


Asynchronous Sequential Logic

Chapter 9

9.1 Introduction

- **Synchronous sequential circuits**
  - Flip-flops share a single clock

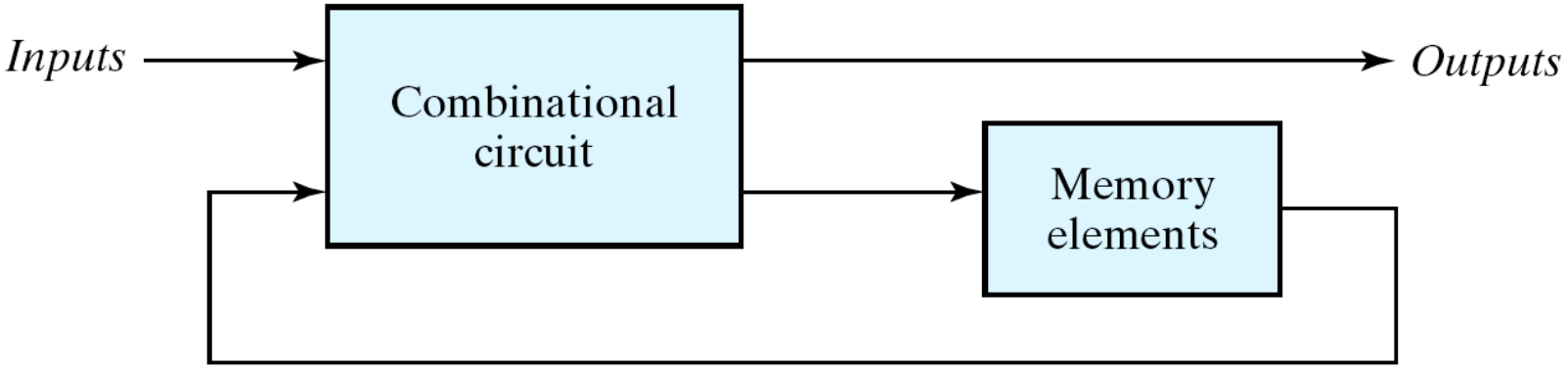


Fig. 5-1 Block Diagram of Sequential Circuit

- **Asynchronous sequential circuits**

Block Diagram of an Asynchronous Sequential Circuit

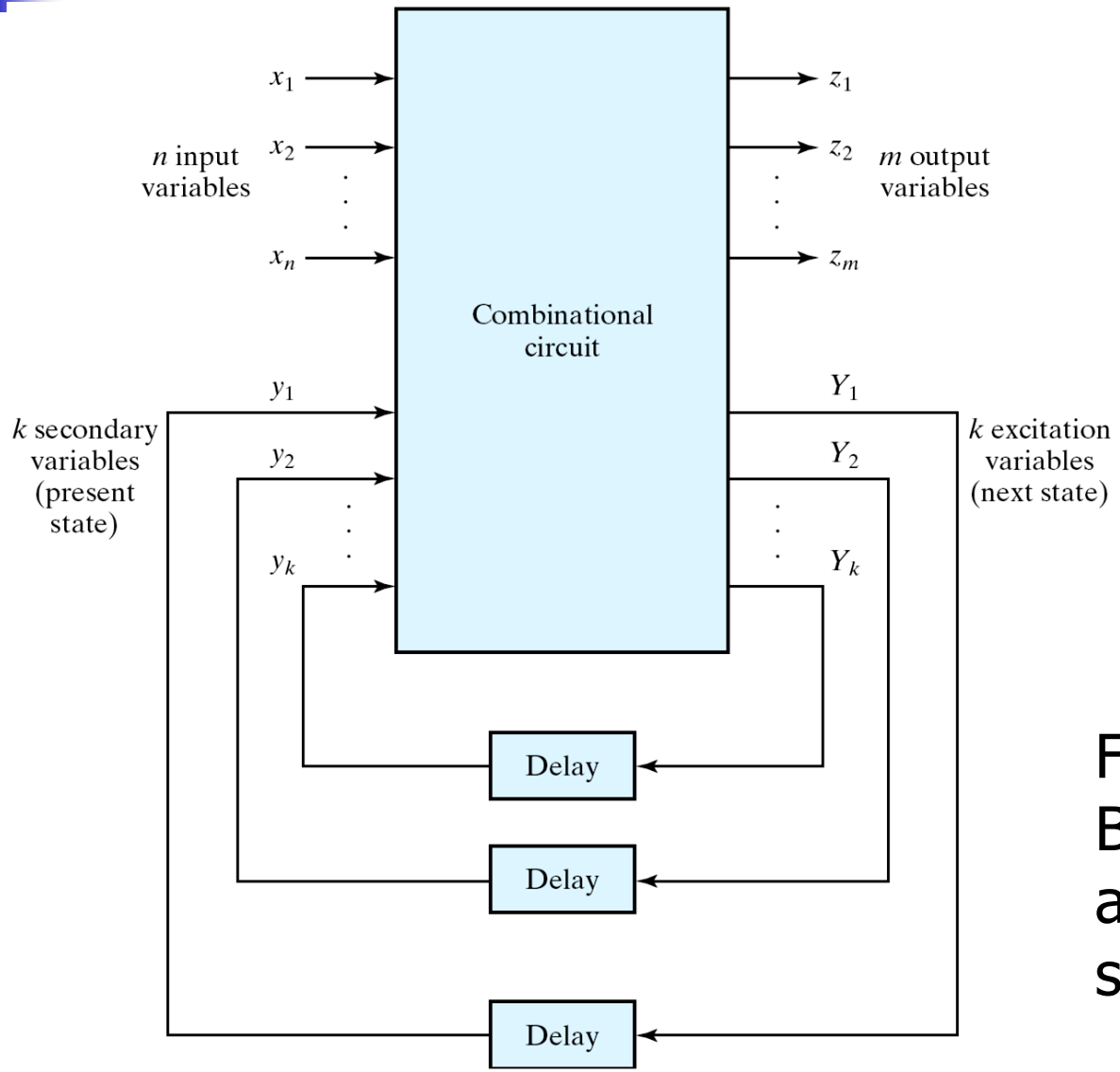


Fig. 9.1
Block diagram of an asynchronous sequential circuit

Asynchronous Sequential Circuit

- No clock pulse
- Memory elements in asynchronous circuits are either **latches** or **time delay elements**
- In a gate-type circuit, the propagation delay that exists in the combinational circuit path from input to output provides sufficient delay along the feedback loop so that no specific delay elements are actually inserted in the feedback path
- Difficult to design: Timing problems involved in the feedback path

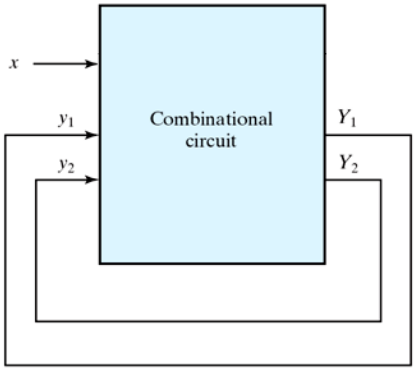
Asynchronous Sequential Circuit

- Must attain a stable state before the input is changed to a new value
- Because of delays in the wires and the gates, it is impossible to have two or more input variables change at exactly the same instant of time without an uncertainty as to which one changes first.
- Therefore, simultaneous changes of two or more variables are usually prohibited.
- This restrictions means that only **one input variable can change at any one time** and **the time between two input changes must be longer than the time it takes the circuit to reach a stable state.**

9-2 Analysis Procedure (No Latches)

- **The procedure:**
  - Determine all feedback loops
  - Assign Y_i's (excitation variables), y_i's (the secondary variables)
  - Derive the Boolean functions of all Y_i's
  - Plot each Y function in a map
  - Construct the **state table**
  - Circle the **stable states**

Example (No Latches)



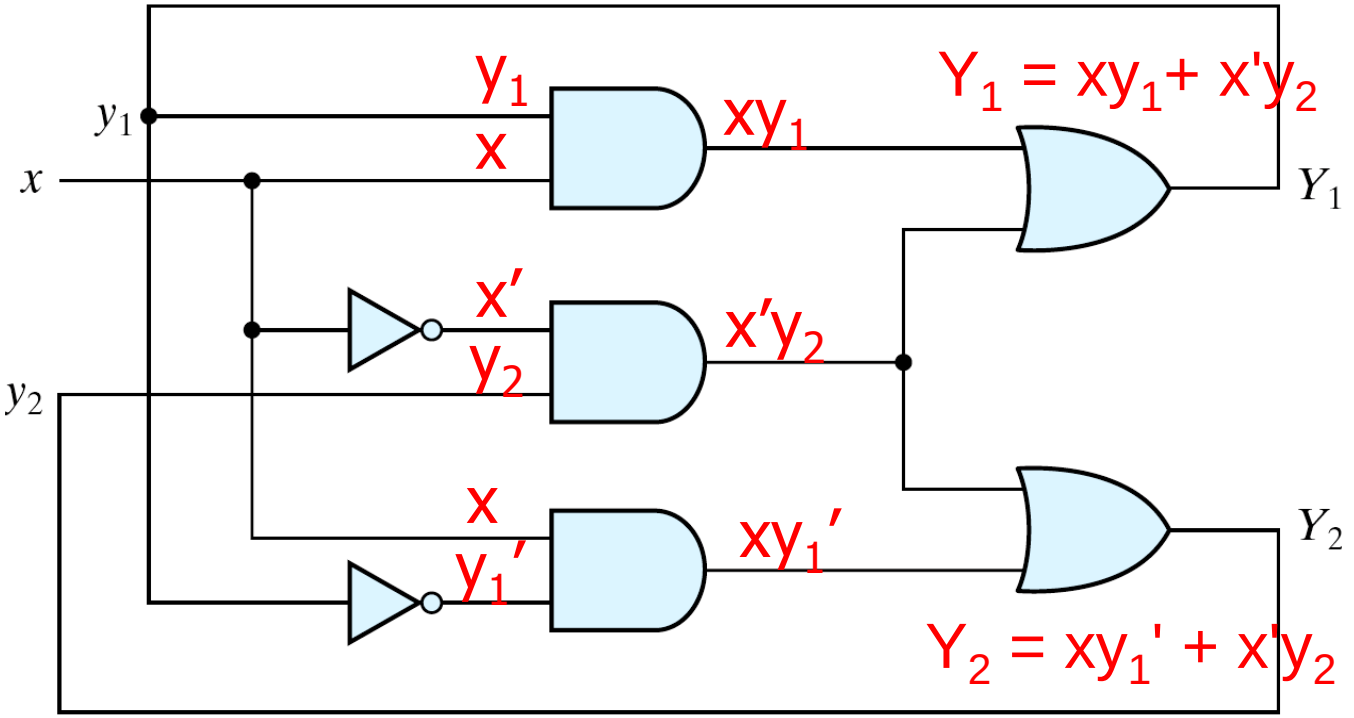
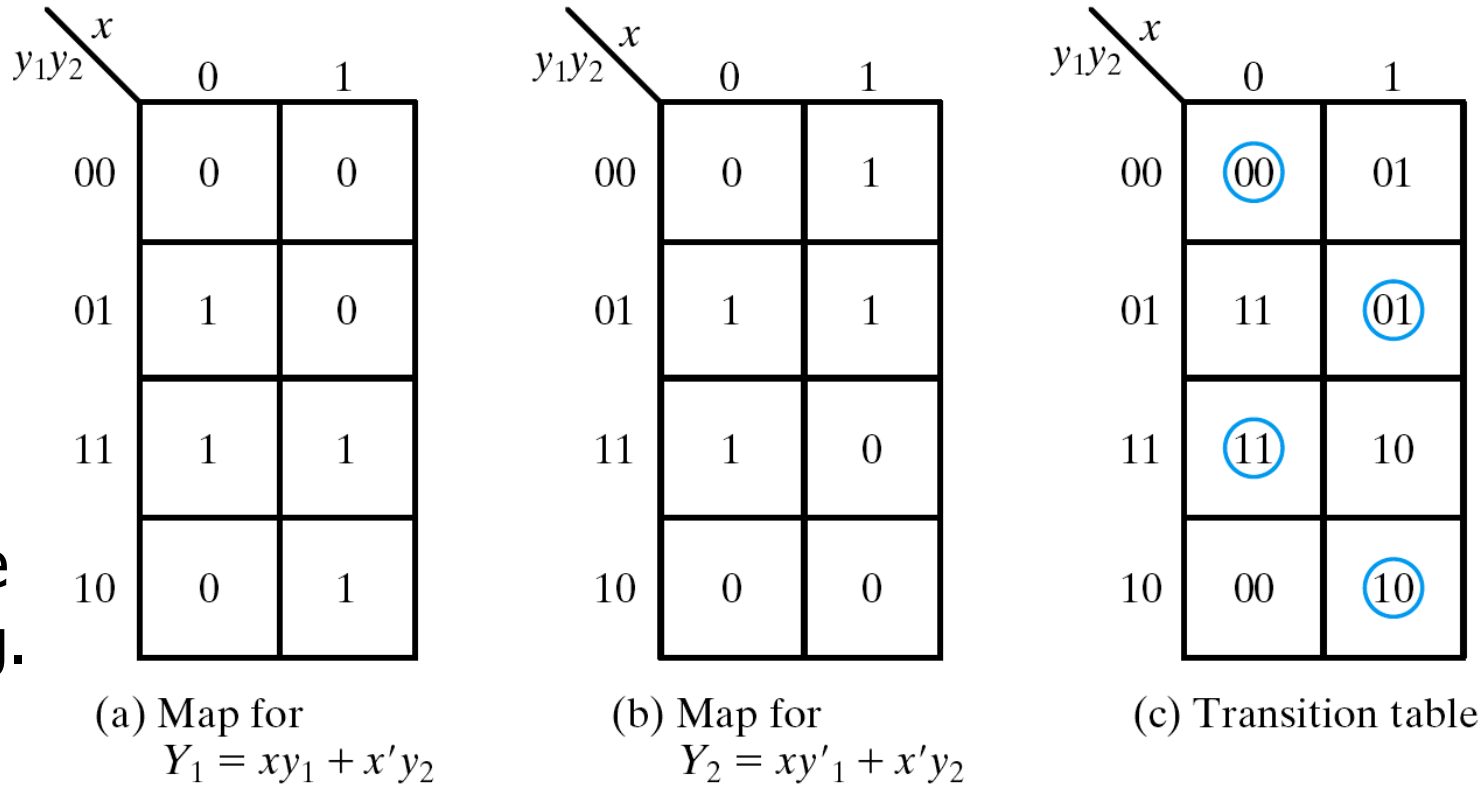


Fig. 9.2
Example of an asynchronous sequential circuit

- the excitation variables: Y_1 and Y_2
- $Y_1 = xy_1 + x'y_2$
- $Y_2 = xy_1' + x'y_2$

Maps and Transition Table



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

Fig. 9.3 Maps and transition table for the circuit of Fig. 9.2

- the y variables for the rows
- the external variable for the columns

- Circle the stable states
  - Y = y

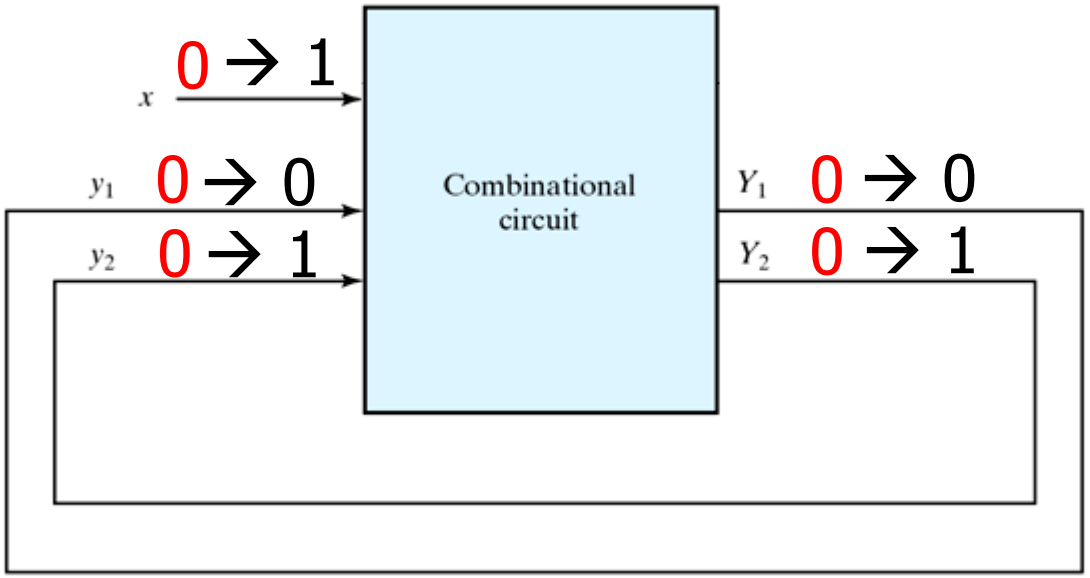
Transition Table

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

Y_1Y_2

y_1y_2

Transition table



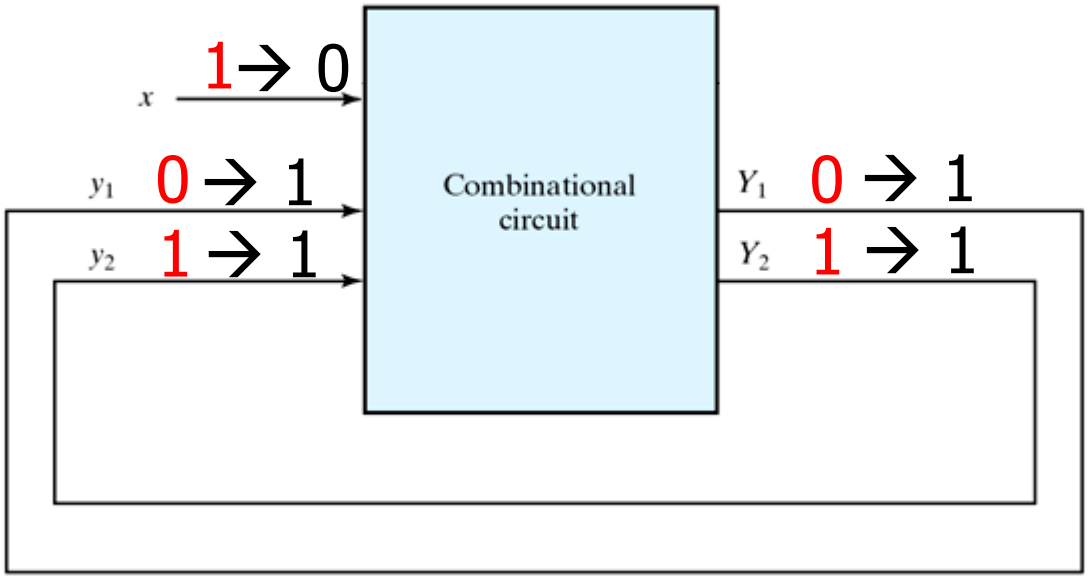
Transition Table

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

Y_1Y_2

y_1y_2

Transition table



State Transition Table

Table 9.1
State Table for the Circuit of Fig. 9.2

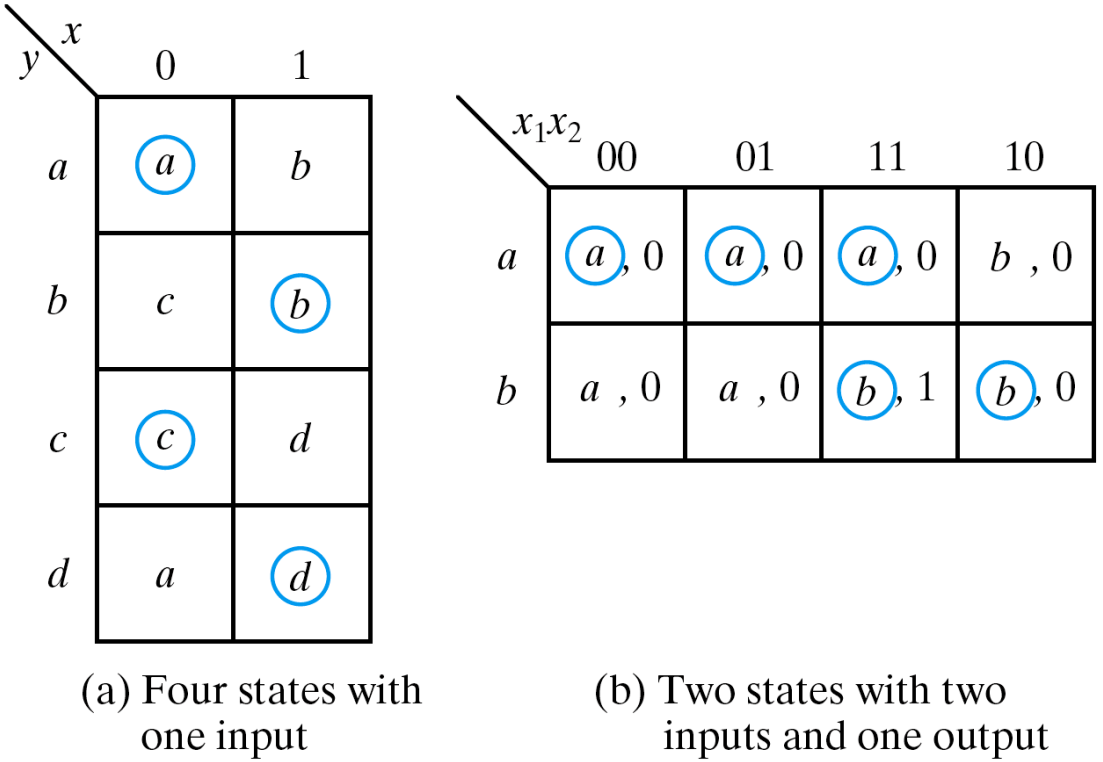
Present State		Next State			
		$x = 0$		$x = 1$	
0	0	0	0	0	1
0	1	1	1	0	1
1	0	0	0	1	0
1	1	1	1	1	0

- **The difference:**
  - synchronous design: state transition happens only when the triggering edge of the clock
  - asynchronous design: the internal state can change immediately after a change in the input

- The **total state** of the asynchronous circuit
  - Combine internal state with the input value
  - y: the present state
  - Y: the next state

Flow Table

- a state transition table with its internal state being symbolized with letters



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

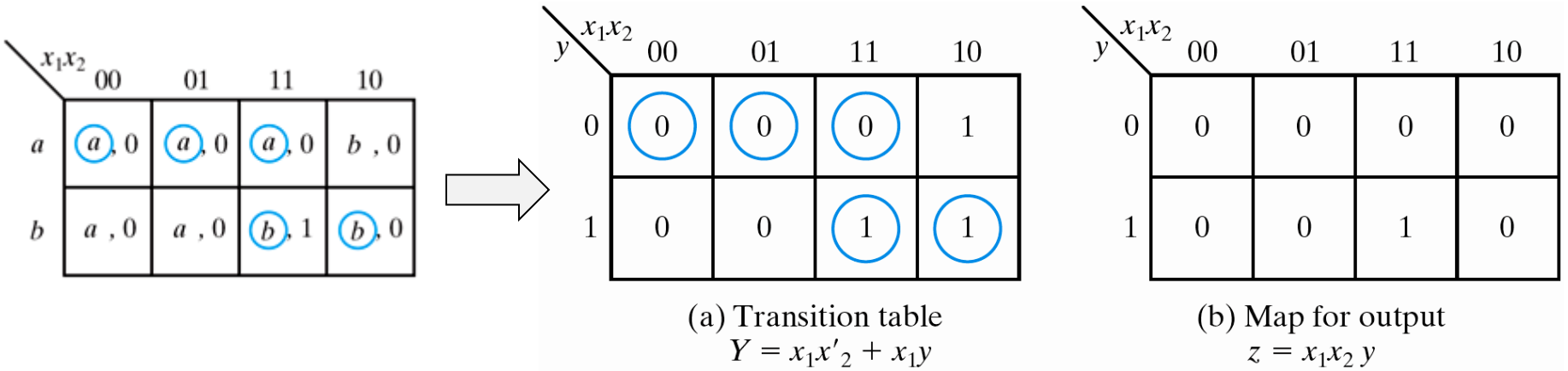
(b) Two states with two inputs and one output

Fig. 9.4
Example of flow tables

- Fig. 9-4(a) is called a primitive flow table because it has only one stable state in each row

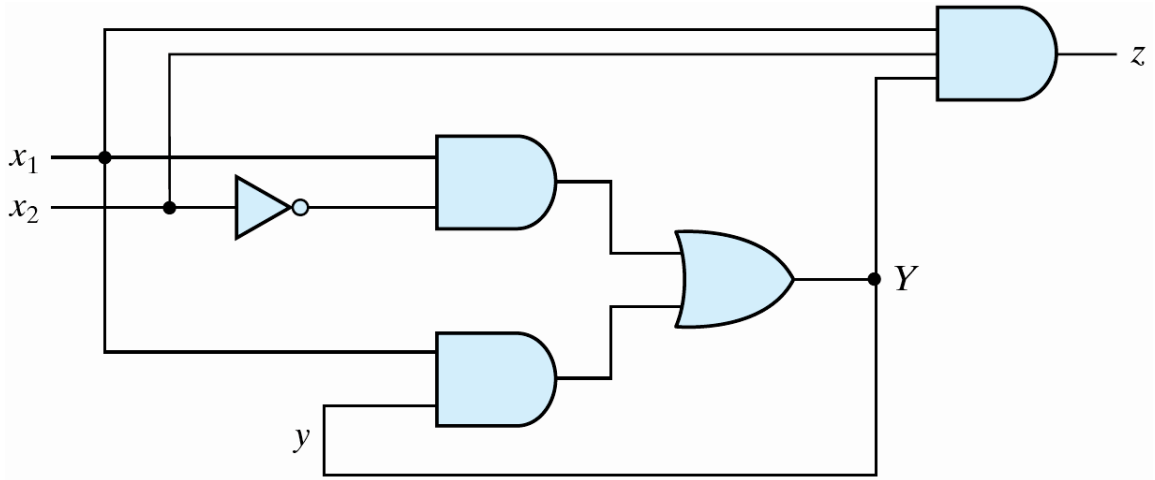
Implementation (No Latches):

- state assignment ⇒derive the logic diagram



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

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



(c) Logic diagram

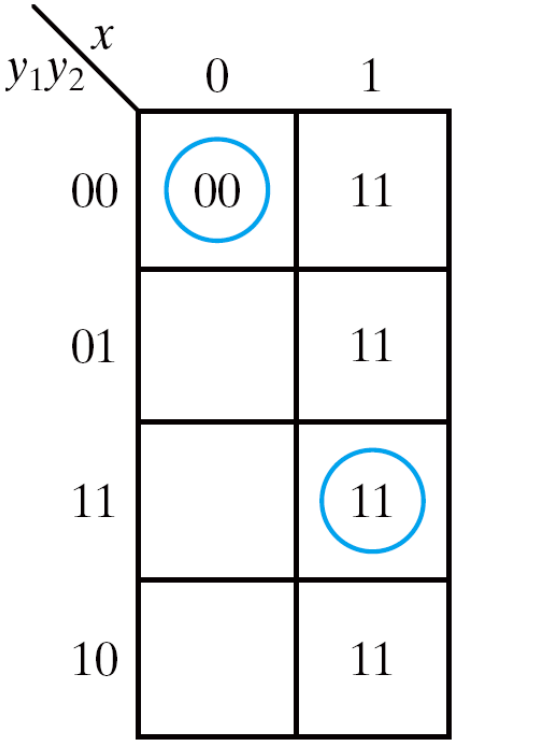
Fig. 9.5
Derivation of a circuit specified by the flow table of Fig. 9.4(b)

Race Conditions

- When two or more binary state variables change value
  - $00 \rightarrow 11$
  - $00 \rightarrow 10 \rightarrow 11$ or $00 \rightarrow 01 \rightarrow 11$

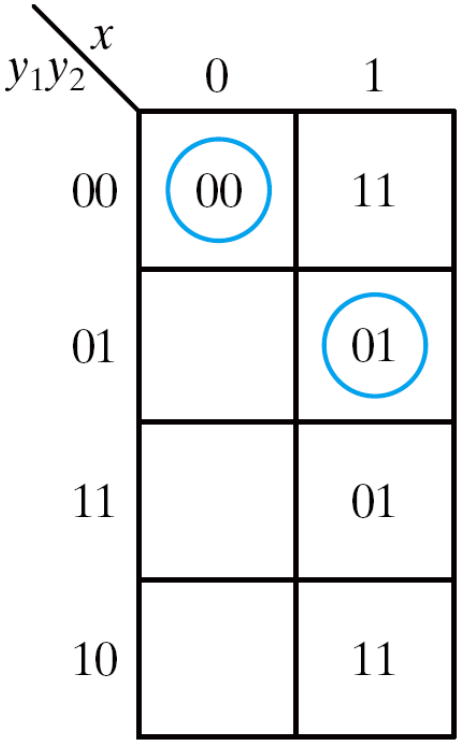
- A **noncritical race**
  - if they reach the same final state
  - otherwise, **a critical state**

Noncritical Race



(a) Possible transitions:

00 → 11
00 → 01 → 11
00 → 10 → 11

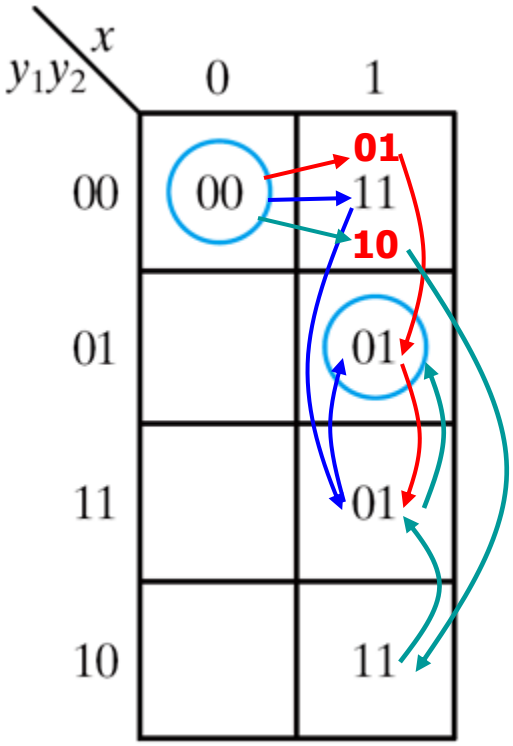


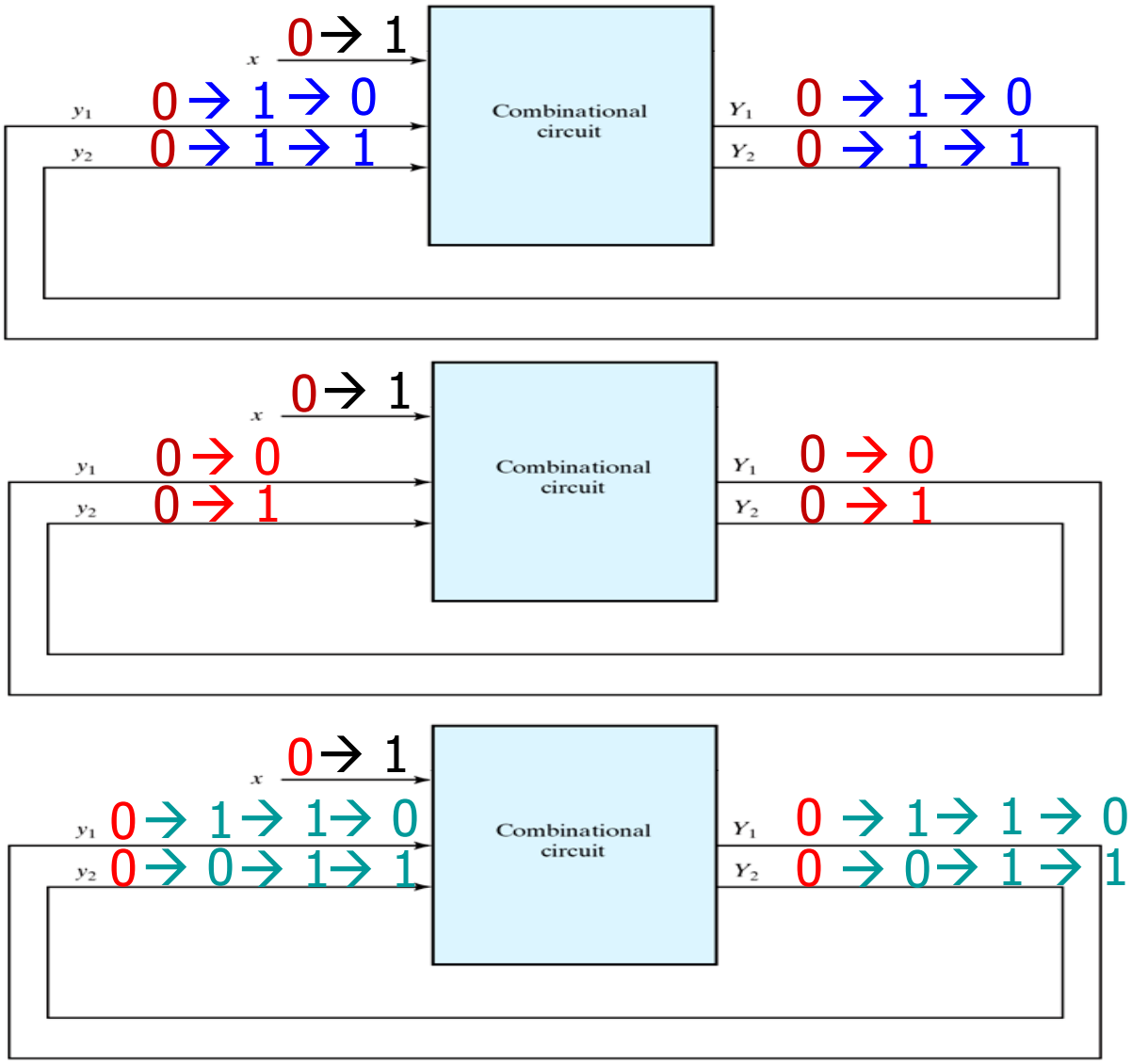
(b) Possible transitions:

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

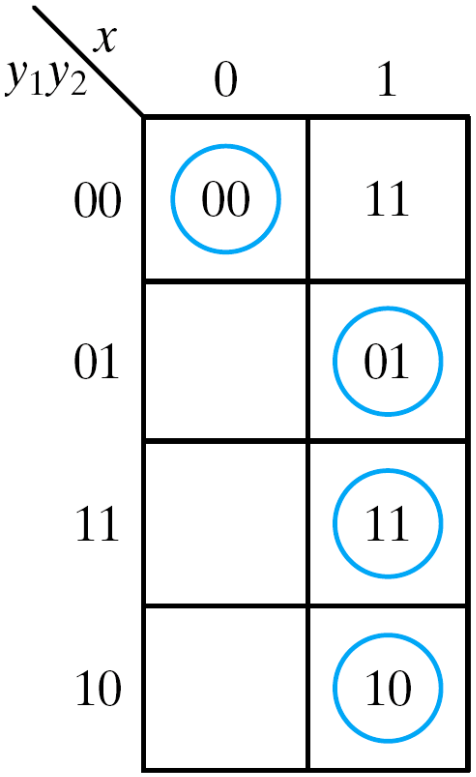
Fig. 9.6
Examples of non-critical races

Noncritical Race



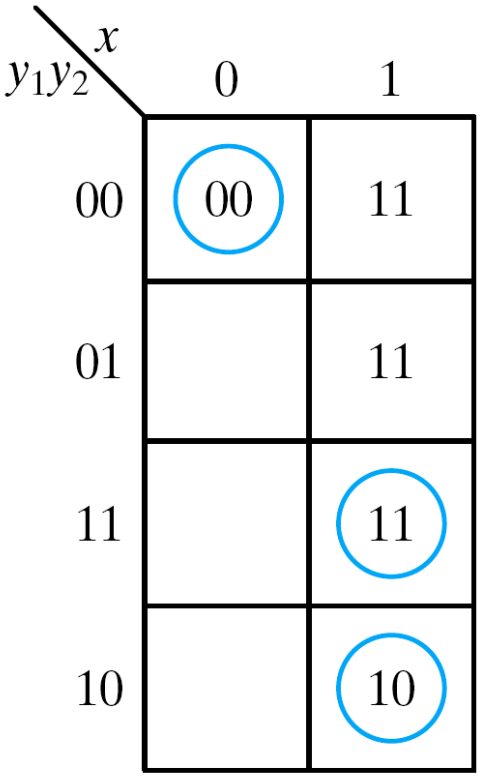


Critical Race



(a) Possible transitions:

00 → 11
00 → 01
00 → 10

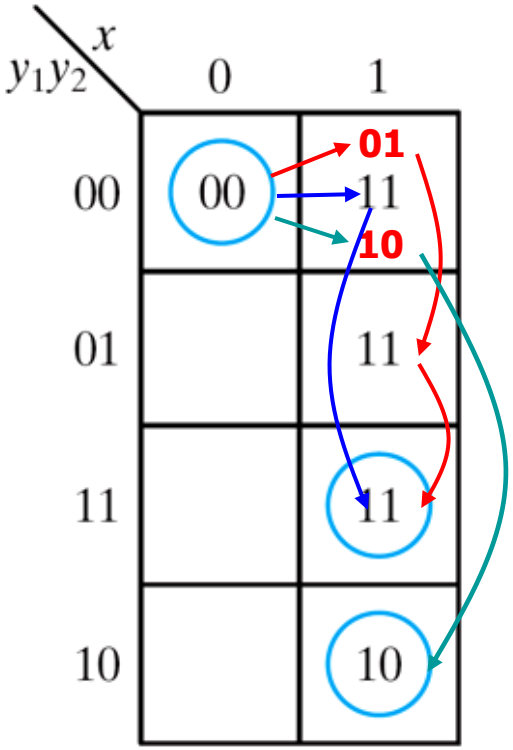


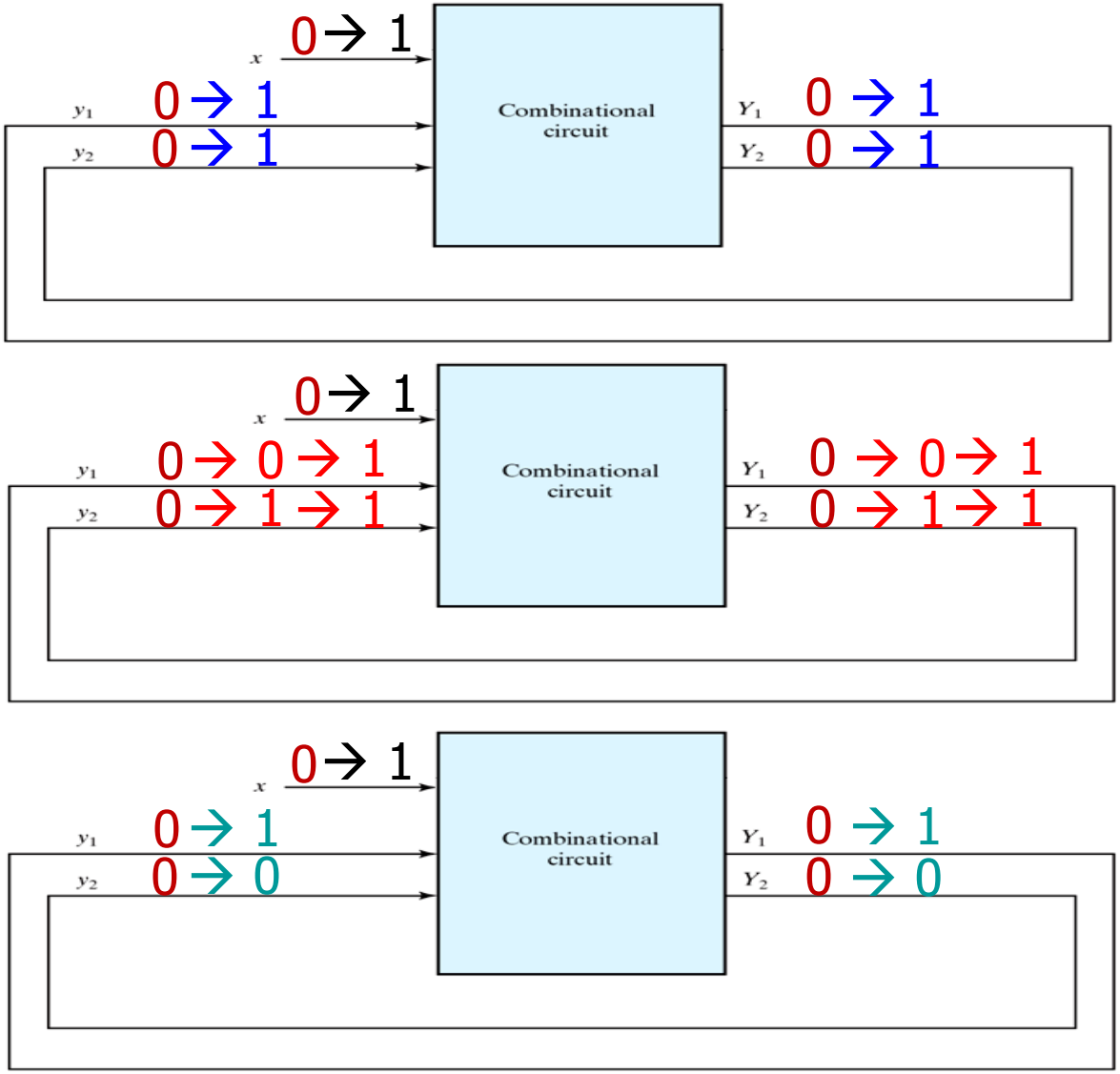
(b) Possible transitions:

00 → 11
00 → 01 → 11
00 → 10

Fig. 9.7
Examples of critical races

Critical Race





Race-Free

- Races may be avoided
  - Race-Free Assignment: Section (9-6)
    - Proper binary assignment to the state variables.
    - The state variable must be assigned binary numbers such that only one state can change at any one time
  - Insert intermediate unstable states with a unique state-variable change. It is said to have a **cycle**.

Cycle

- a unique sequence of unstable states

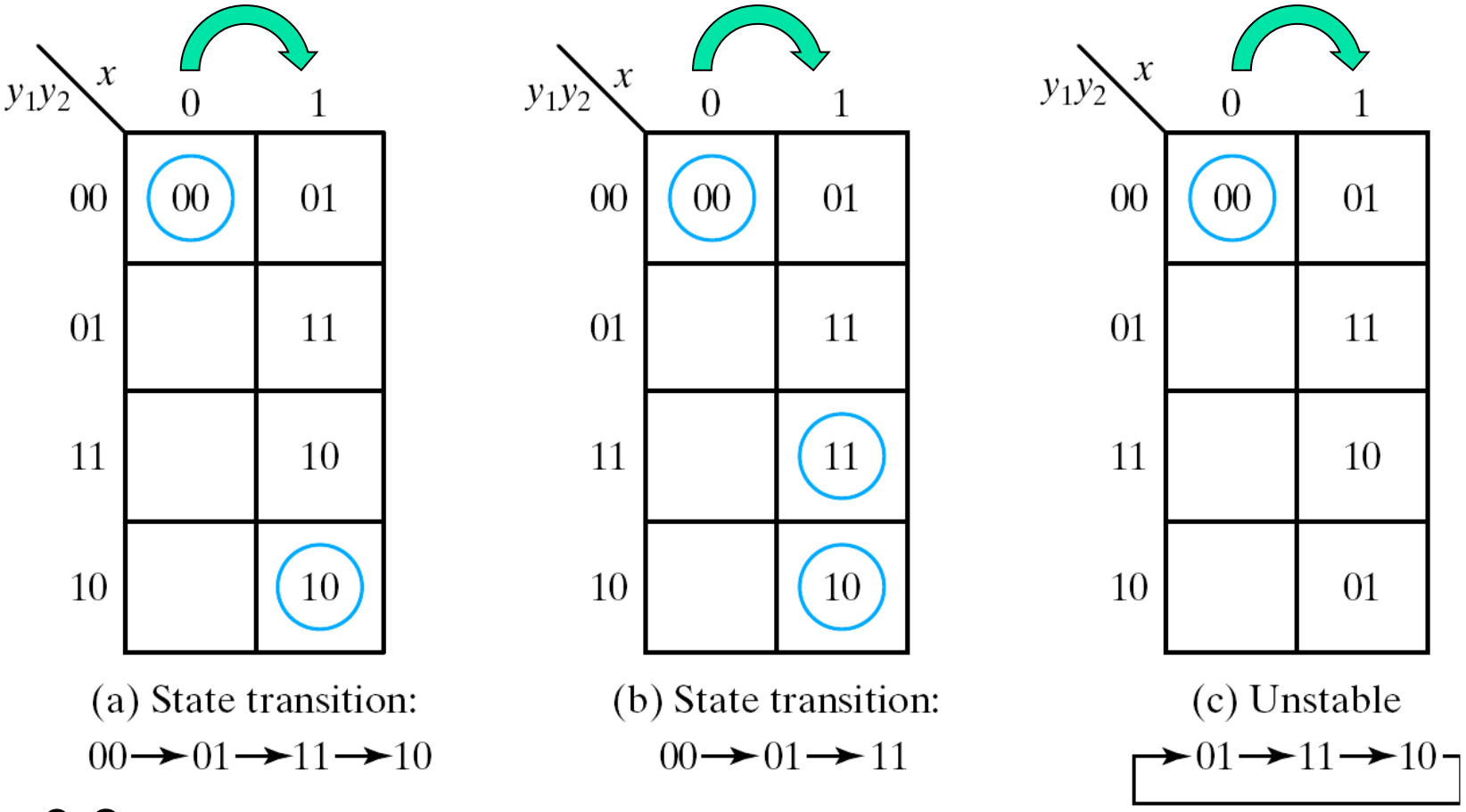
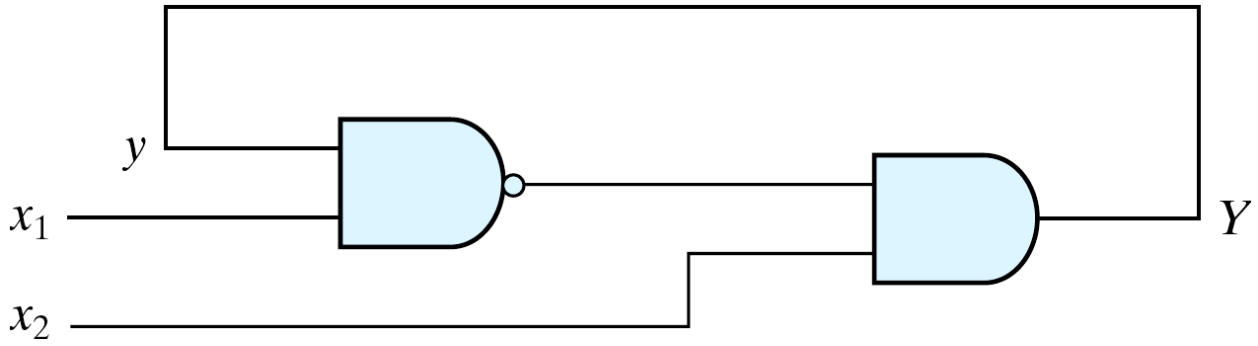


Fig. 9.8
Examples of cycles

Stability Considerations

- A square waveform generator?



(a) Logic diagram

$Y = (x_1y)'x_2$

$Y = x_1'x_2 + y'x_2$

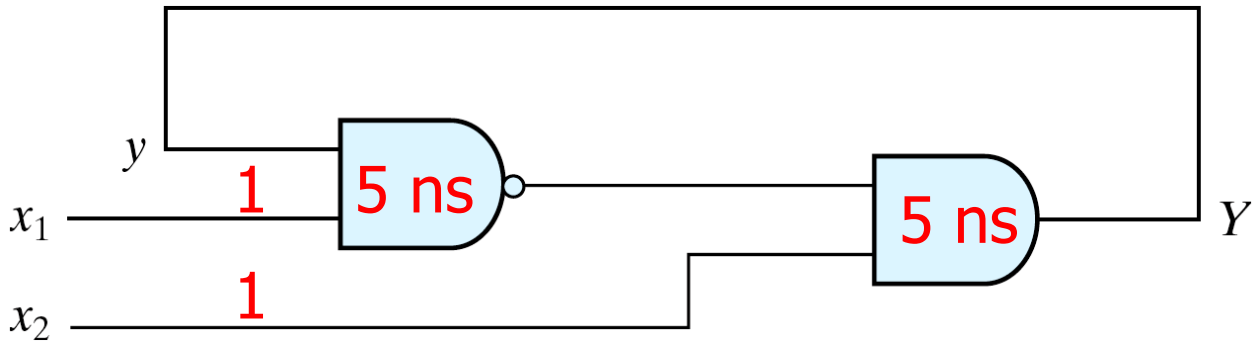
y \ x_1x_2	00	01	11	10
0	0	1	1	0
1	0	1	0	0

(b) Transition table

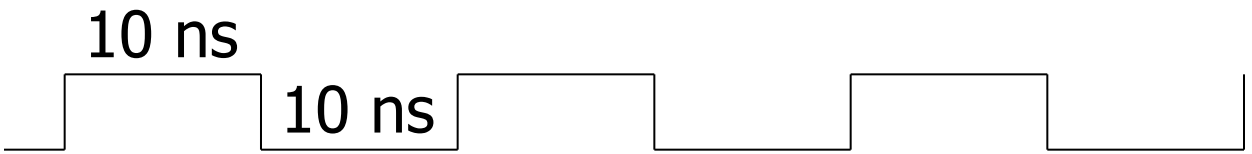
Fig. 9.9
Example of an unstable circuit

Stability Considerations

- Note that column 11 has no stable states.
- This mean that with inputs x_1x_2 fixed at 11, the value of Y and y are never the same.



(a) Logic diagram

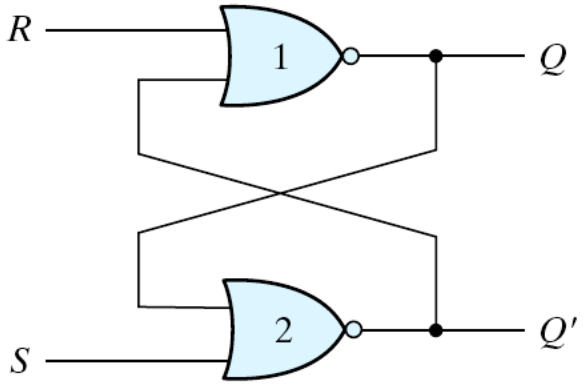


Frequency = 1 / 20ns = 50 MHz

9-3 Circuits with Latches

- Asynchronous sequential circuits
  - were known and used before synchronous design
  - the use of SR latches in asynchronous circuits produces a more orderly pattern
  - reduce the circuit complexity

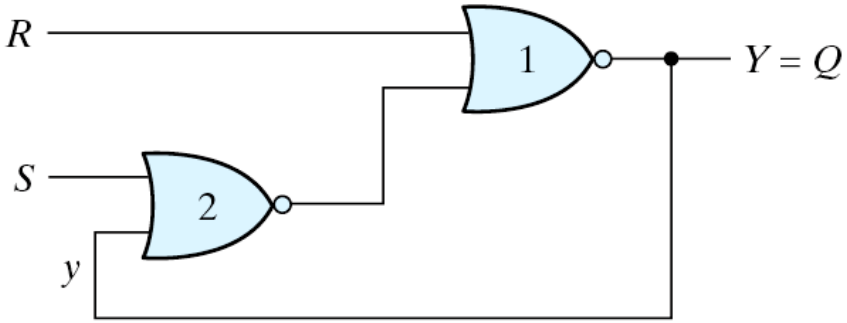
SR Latch - Two Cross-Coupled NOR Gates



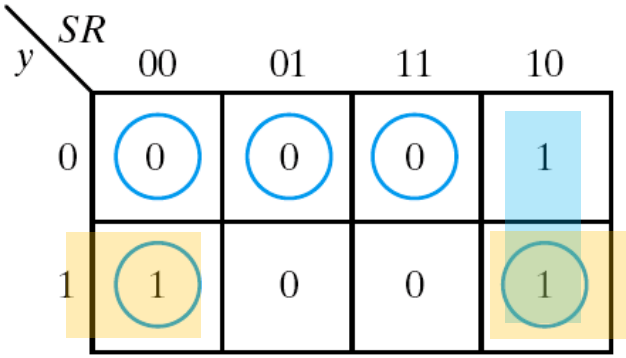
(a) Cross-coupled circuit

S	R	Q	Q′	
1	0	1	0	
0	0	1	0	(After $SR = 10$)
0	1	0	1	
0	0	0	1	(After $SR = 01$)
1	1	0	0	

(b) Truth table



(c) Circuit showing feedback



$Y = SR' + R'y$

$Y = S + R'y$ when $SR = 0$

(d) Transition table

Fig. 9.10

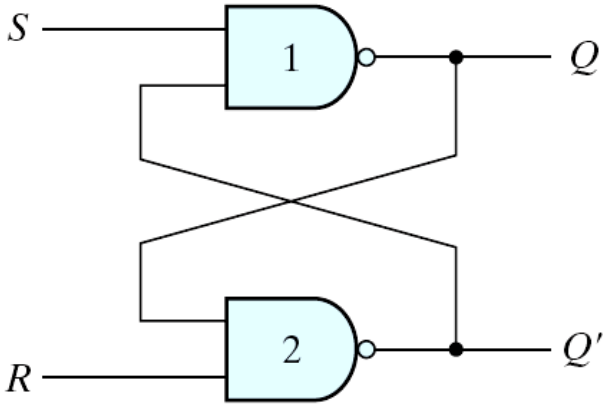
SR latch with NOR gates

SR Latch

- Y = ((S+y)'+R)' = (S+y)R' = SR'+R'y
- the state transition table
- an unpredictable result when SR: 11 $\rightarrow$ 00
- SR = 0 in operation
- SR' + SR = S(R'+R) = S
- $\Rightarrow$Y = S + R'y when SR = 0

- two cross-coupled NAND gate
  - S'R' = 0
  - Y = (S(Ry)')' = S'+ Ry when S'R' = 0
  - S'R' latch

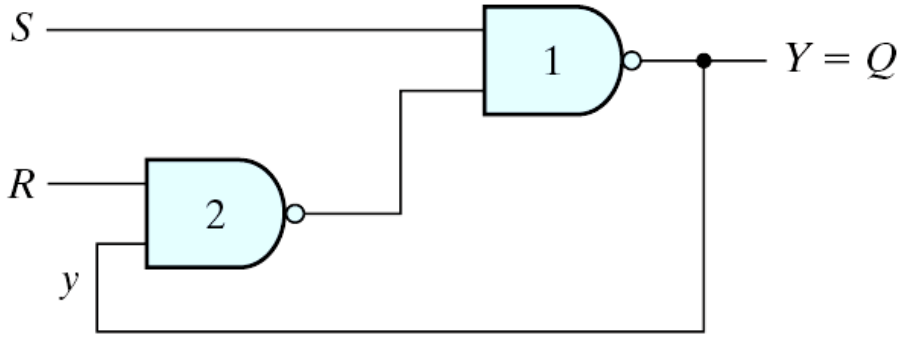
SR Latch - Two Cross-Coupled NAND Gates



(a) Cross-coupled circuit

S	R	Q	Q′	
1	0	0	1	
1	1	0	1	(After $SR = 10$)
0	1	1	0	
1	1	1	0	(After $SR = 01$)
0	0	1	1	

(b) Truth table



(c) Circuit showing feedback

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

(d) Transition table

Fig. 9.11
SR latch with NAND gates

Analysis Example (With Latches):

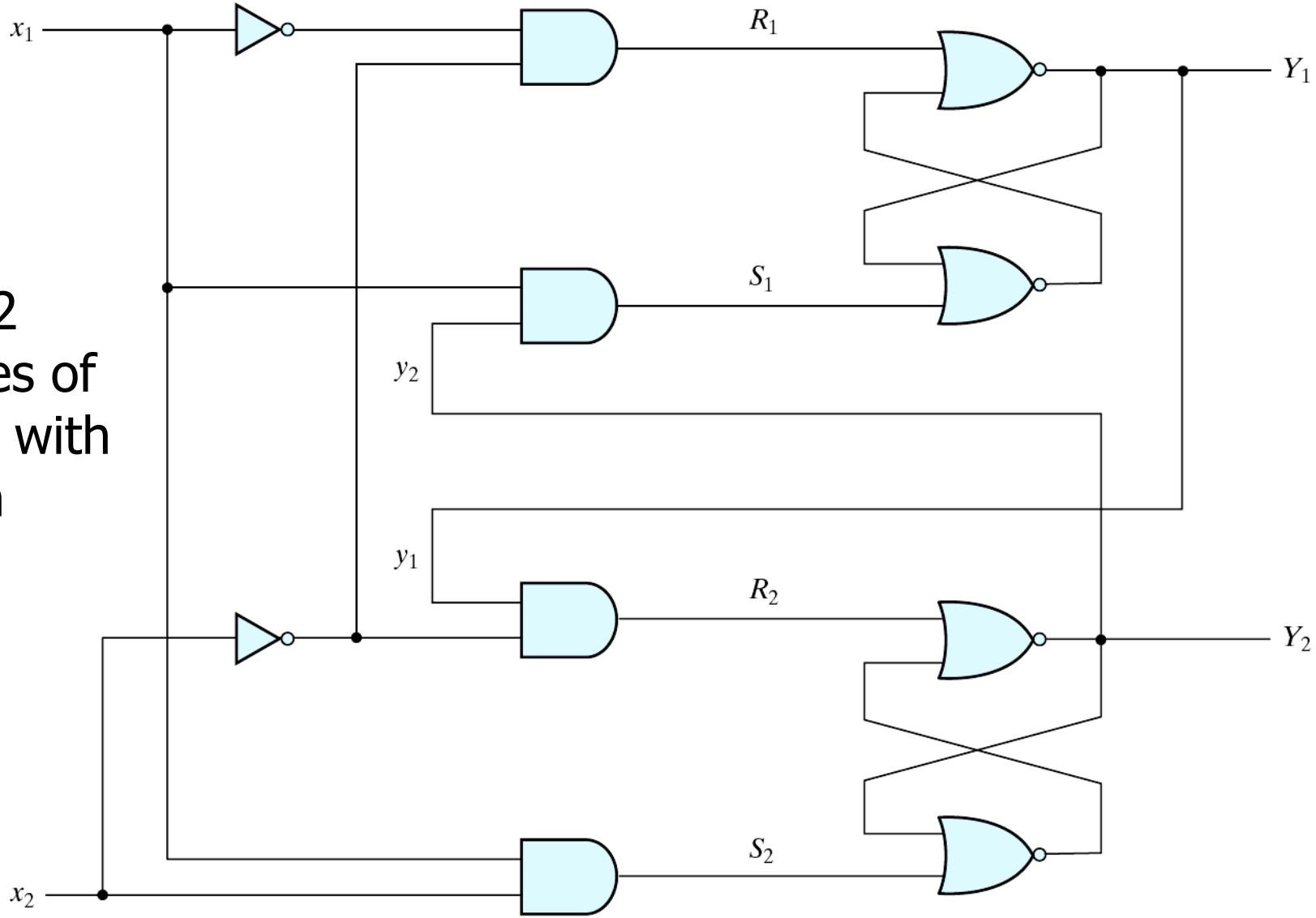


Fig. 9.12
Examples of a circuit with *SR* latch

Analysis Example (Continued):

- Label each latch output with Y_i and its external feedback path with y_i
- Derive the Boolean functions for the S_i and R_i
  - $S_1 = x_1 y_2$
  - $R_1 = x'_1 x'_2$
  - $S_2 = x_1 x_2$
  - $R_2 = x'_2 y_1$
- Check whether SR=0 for each NOR latch or whether S'R'=0 for each NAND latch
  - $S_1 R_1 = x_1 y_2 x'_1 x'_2 = 0$
  - $S_2 R_2 = x_1 x_2 x'_2 y_1 = 0$

Analysis Example (Continued):

- Evaluate Y = S +R'y for each NOR latch
  or Y = S' + Ry for each NAND latch
  - $Y_1 = x_1y_2 + (x_1+x_2)y_1 = x_1y_2+x_1y_1+x_2y_1$
  - $Y_2 = x_1x_2 + (x_2+y'_1)y_2 = x_1x_2+x_2y_2+y'_1y_2$
- Construct the state transition table
- Circle all stable states

Race example:

- Let initial state is

  $y_1y_2x_1x_2$=1101

  input x_2 is changed to 0

  if Y_1 change to 0 before Y_2

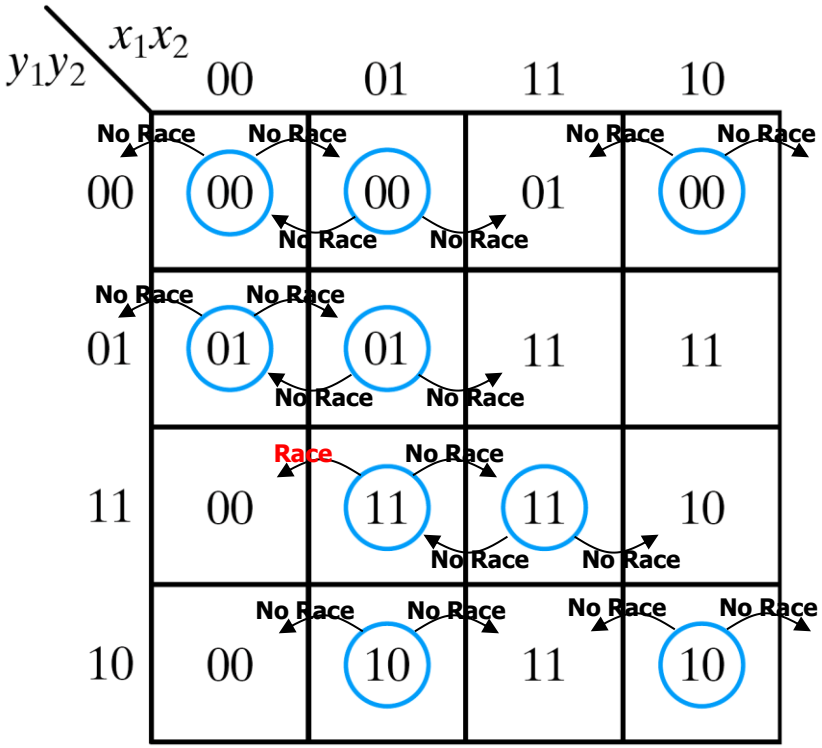
  then $y_1y_2x_1x_2$=0100

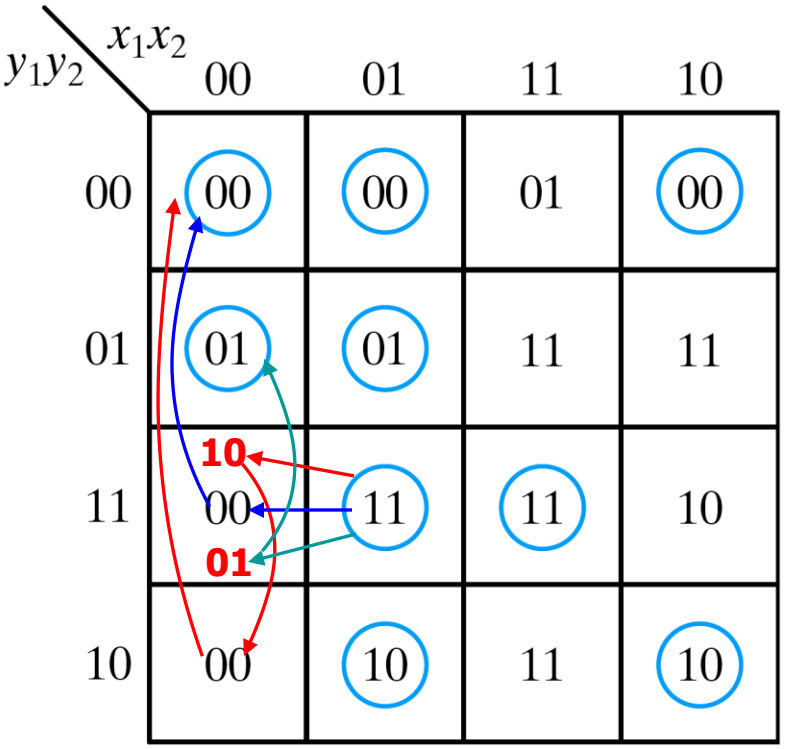
  instead of 0000

y_1y_2 \ x_1x_2	00	01	11	10
00	(00)	(00)	01	(00)
01	(01)	(01)	11	11
11	00	(11)	(11)	10
10	00	(10)	11	(10)

Fig. 9.13
Transition table for the circuit of Fig. 9.12

Analysis Example (Continued):





11 → 00
11 → 10 → 00
11 → 01

→ Critical Race

Analysis Procedure (With Latches)

The procedure for analyzing an asynchronous sequential circuit with *SR* latches can be summarized as follows:

1. Label each latch output with Y_i and its external feedback path (if any) with y_i for $i = 1, 2, \ldots, k$.
2. Derive the Boolean functions for the S_i and R_i inputs in each latch.
3. Check whether $SR = 0$ for each NOR latch or whether $S'R' = 0$ for each NAND latch. If either of these conditions is not satisfied, there is a possibility that the circuit may not operate properly.
4. Evaluate $Y = S + R'y$ for each NOR latch or $Y = S' + Ry$ for each NAND latch.
5. Construct a map, with the y's representing the rows and the x inputs representing the columns.
6. Plot the value of $Y = Y_1 Y_2 \cdots Y_k$ in the map.
7. Circle all stable states such that $Y = y$. The resulting map is then the transition table.

Latch Excitation Table

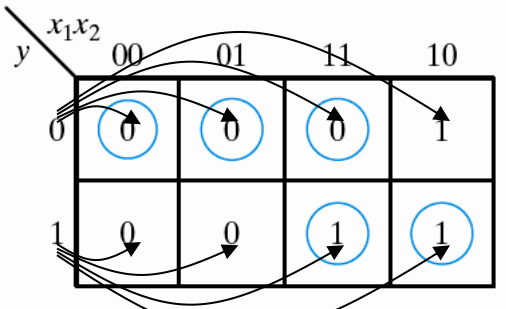
- For *SR* latch:

y	Y	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

(b) Latch excitation table

Implementation (With a Latch)

Fig. 9.14
Derivation of a latch circuit from a transition table

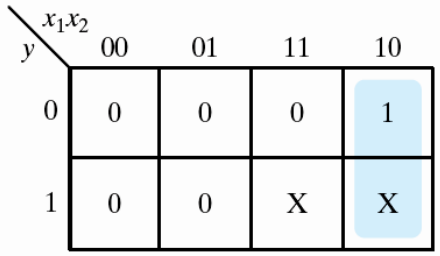


(a) Transition table

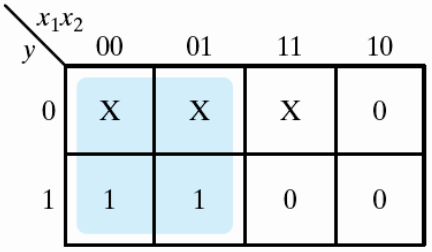
$Y = x_1x'_2 + x_1y$

y	Y	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

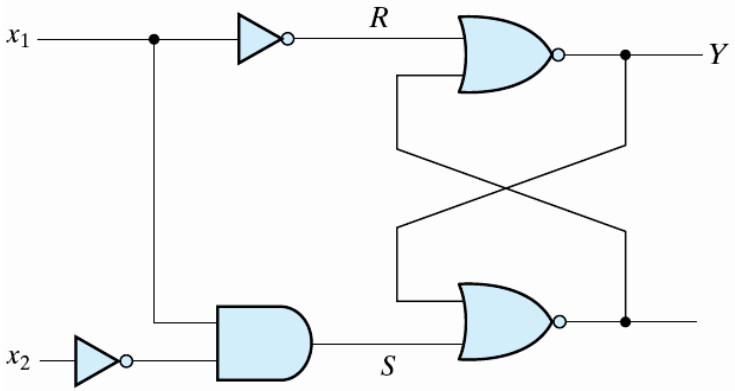
(b) Latch excitation table



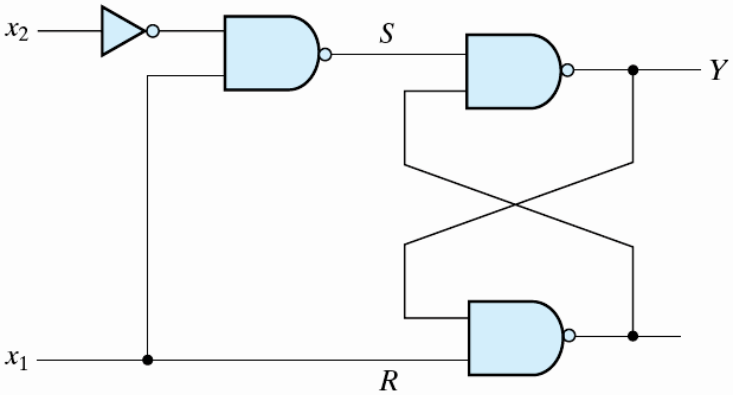
(c) Map for $S = x_1x'_2$



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



(e) Circuit with NOR latch



(f) Circuit with NAND latch

Implementation Example

- Determine the Boolean functions for the S and R inputs of each latch
- Given a transition table
  - From maps: the simplified Boolean functions are

$S = x_1 x_2'$ and $R = x_1'$ ⇒ NOR latch

$S = (x_1 x_2')'$ and $R = x_1$ ⇒ NAND latch

General Procedure for Implementing a Circuit with SR Latches

- Derive a pair of maps for S_i and R_i
- Derive the simplified Boolean functions for each S_i and R_i
- DO NOT make S_i and R_i equal to 1 in the same minterm square
- Draw the logic diagram
  - for NAND latches, use the complemented values of those S_i and R_i

Debounce Circuit

- Remove the series of pulses that result form a contact bounce and produce a single smooth transition of the binary signal

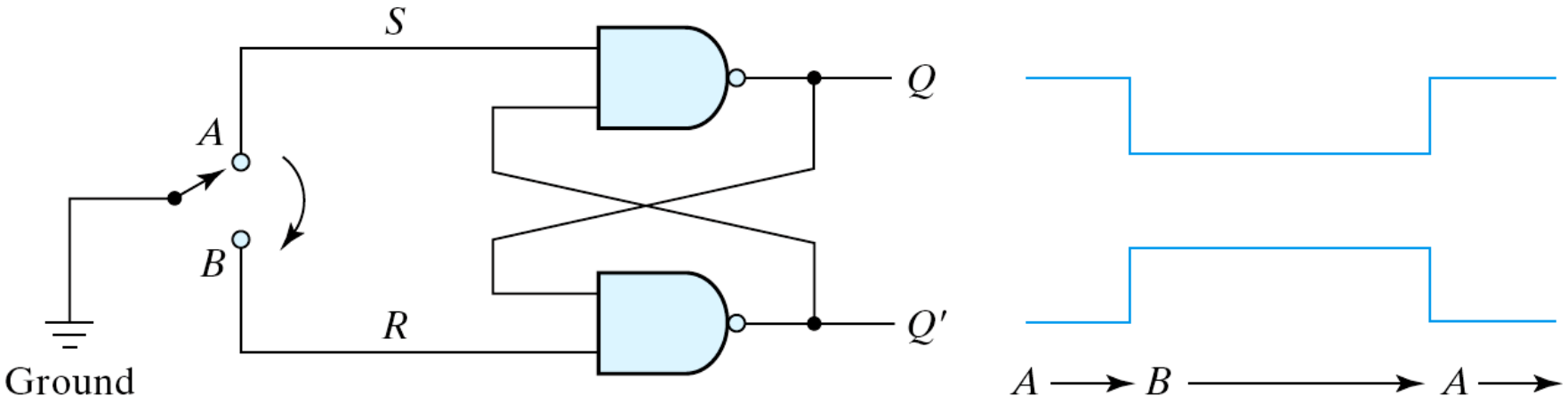


Fig. 9.15
Debounce circuit

- TTL: input = logic-1 when open

9-4 Design Procedure

- Design specifications
  - a gated latch
  - two inputs, G (gate) and D (data)
  - one output, Q
  - G = 1: Q follows D
  - G = 0 : Q remains unchanged

Design Procedure

- All the total states
  - combinations of the inputs and internal states

Table 9.2
Gated-Latch Total States

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

  - simultaneous transitions of two input variables are not allowed

Design Procedure

- Primitive flow table

Table 9.2
Gated-Latch Total States

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

States	Inputs *DG* 00	01	11	10
a	*c* , –	(*a*), 0	*b* , –	– , –
b	– , –	*a* , –	(*b*), 1	*e* , –
c	(*c*), 0	*a* , –	– , –	*d* , –
d	*c* , –	– , –	*b* , –	(*d*), 0
e	*f* , –	– , –	*b* , –	(*e*), 1
f	(*f*), 1	*a* , –	– , –	*e* , –

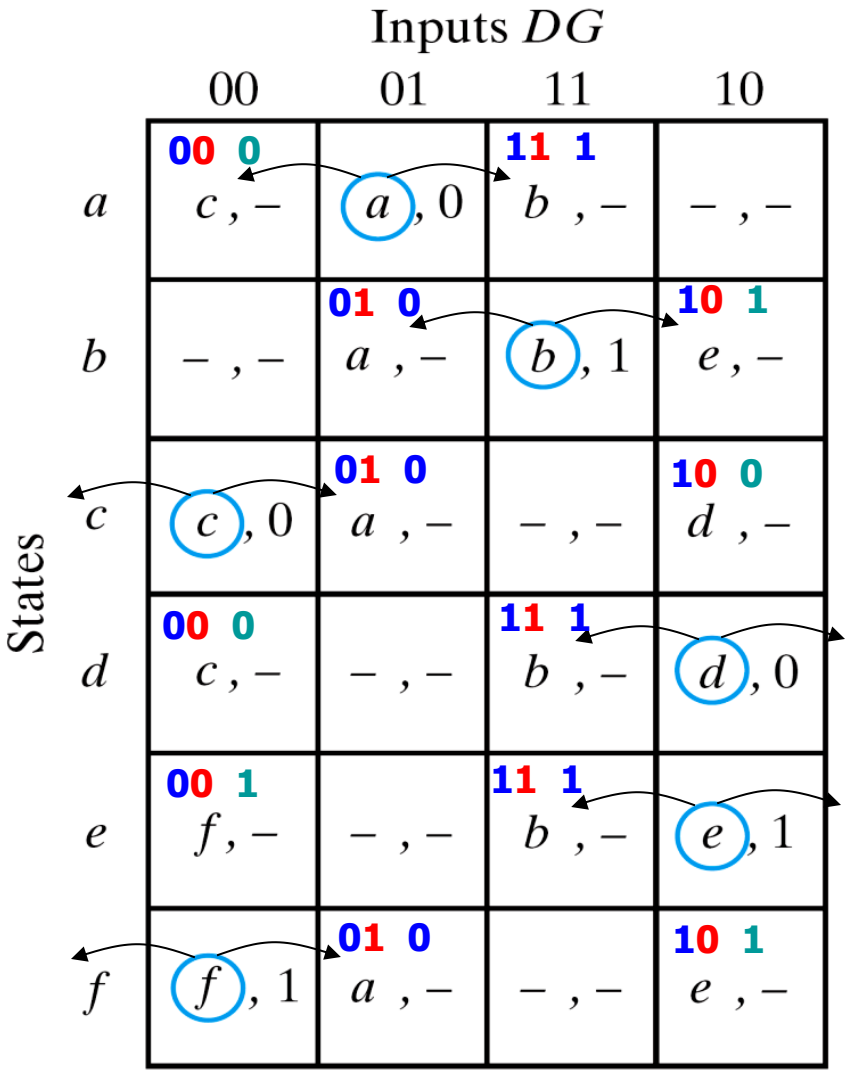
Fig. 9.16
Primitive flow table

- dash marks in each row that differs in two or more variables from the input variables associated with the stable state
- don't care condition for the next state and output

Design Procedure

Table 9.2
Gated-Latch Total States

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



Design Procedure

- Reduction of the primitive flow table
  - two or more rows in the primitive flow table can be merged if there are non-conflicting states and outputs in each of the columns

Design Procedure

States	Inputs DG: 00	01	11	10
a	c, –	(a), 0	b, –	–, –
b	–, –	a, –	(b), 1	e, –
c	(c), 0	a, –	–, –	d, –
d	c, –	–, –	b, –	(d), 0
e	f, –	–, –	b, –	(e), 1
f	(f), 1	a, –	–, –	e, –

States	DG: 00	01	11	10
a	c, –	(a), 0	b, –	–, –
c	(c), 0	a, –	–, –	d, –
d	c, –	–, –	b, –	(d), 0

States	DG: 00	01	11	10
b	–, –	a, –	(b), 1	e, –
e	f, –	–, –	b, –	(e), 1
f	(f), 1	a, –	–, –	e, –

(a) States that are candidates for merging

States	DG: 00	01	11	10
a, c, d	(c), 0	(a), 0	b, –	(d), 0
b, e, f	(f), 1	a, –	(b), 1	(e), 1

States	DG: 00	01	11	10
a	(a), 0	(a), 0	b, –	(a), 0
b	(b), 1	a, –	(b), 1	(b), 1

(b) Reduced table (two alternatives)

Fig. 9.17
Reduction of the primitive flow table

Design Procedure (No Latches)

- Transition table and logic diagram
  - State assignment
    - discussed in details in Sec. 9-6
  - a:0, b:1

States \ DG	00	01	11	10
a	(a), 0	(a), 0	b , –	(a), 0
b	(b), 1	a , –	(b), 1	(b), 1

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

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

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

(b) $Q = Y$

Fig. 9.18
Transition table and output map for gated latch

Design Procedure (No Latches)

- the output
- logic diagram

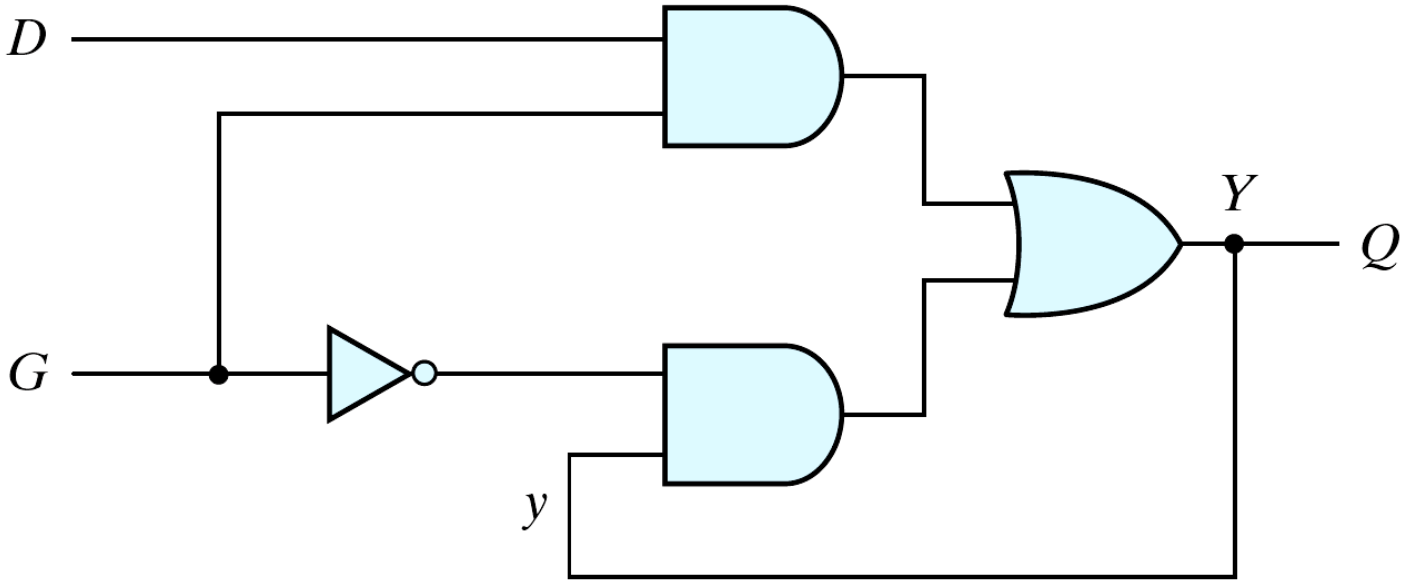


Fig. 9.19
Gated-latch logic diagram

Design Procedure (With a Latch)

- SR latch implementation

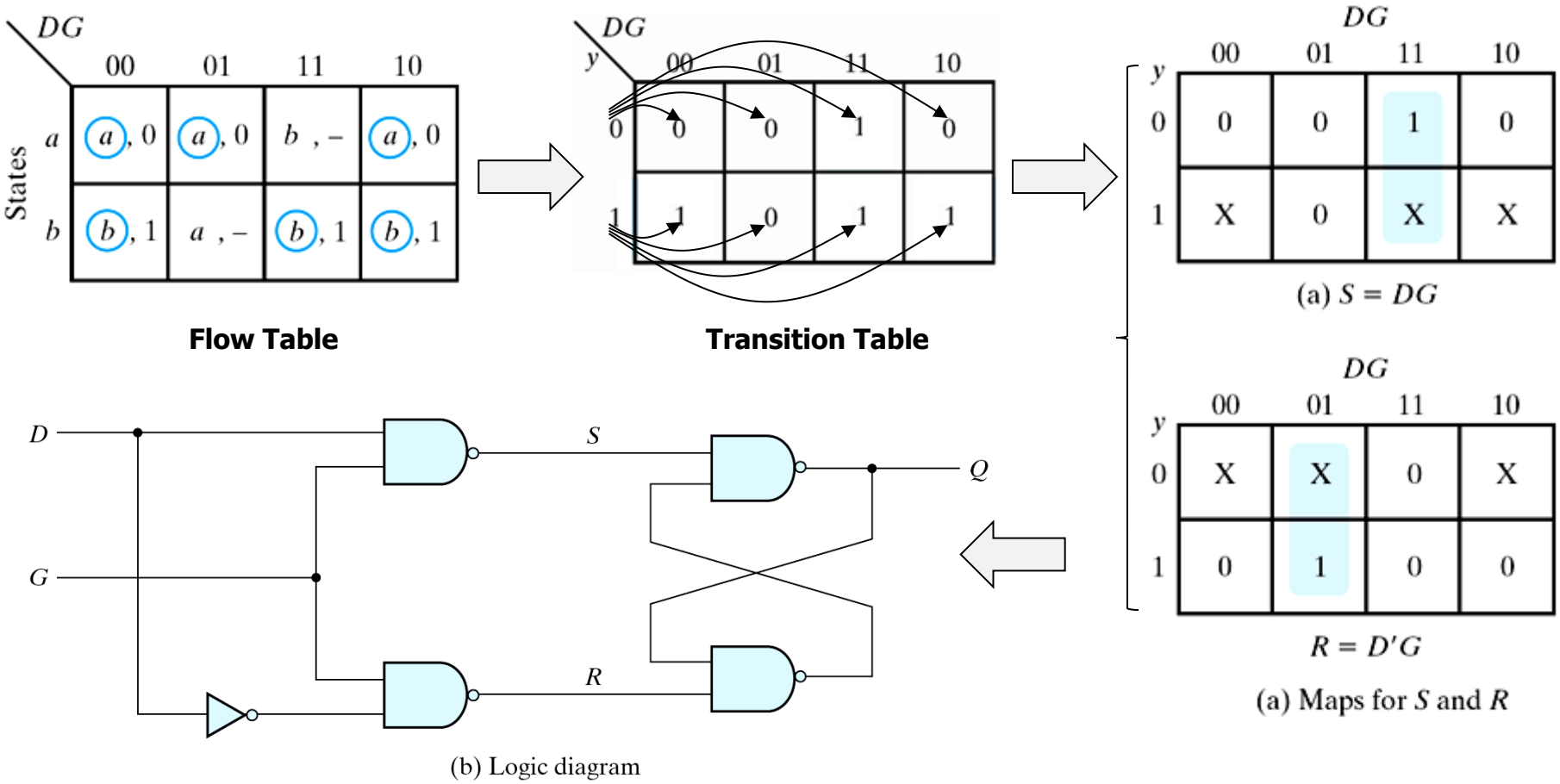


Fig. 9.20
Circuit with *SR* latch

Design Procedure

- Assign outputs to unstable states
  - the unstable states have unspecified output values
  - no momentary false outputs occur when the circuit switches between stable states

- $0 \rightarrow 0$: 0
- $1 \rightarrow 1$: 1
- $0 \rightarrow 1, 1 \rightarrow 0$: -

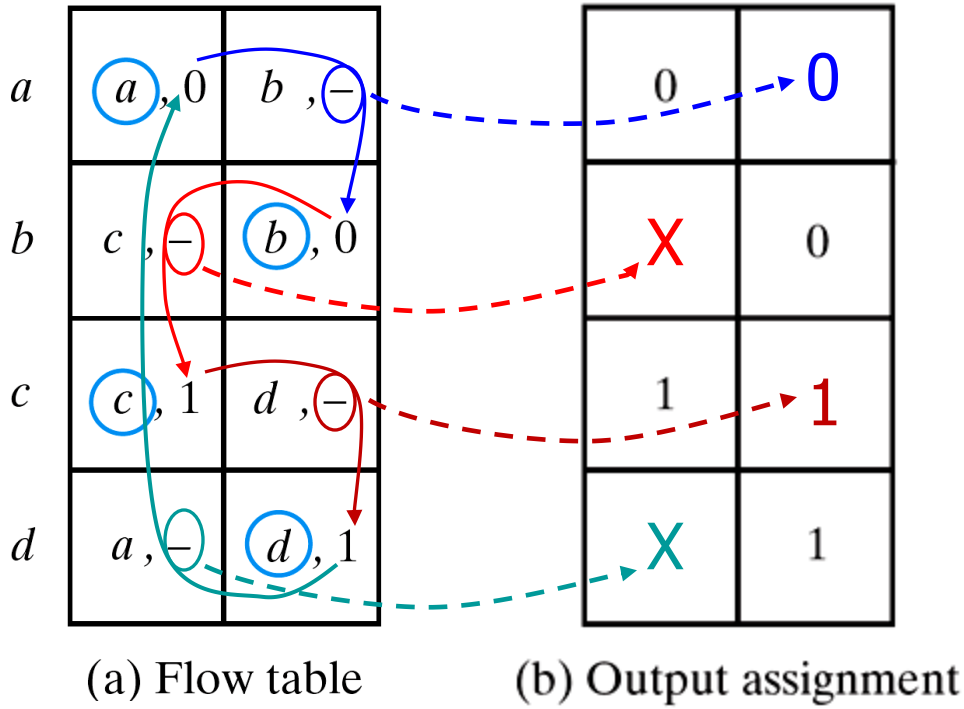


Fig. 9.21
Assigning output values to unstable states

Design Procedure

- The procedure for making the assignment to outputs associated with unstable states can be summarized follows:

1. Assign a 0 to an output variable associated with an unstable state which is a transient state between two stable states that have a 0 in the corresponding output variable.
2. Assign a 1 to an output variable associated with an unstable state which is a transient state between two stable states that have a 1 in the corresponding output variable.
3. Assign a don't-care condition to an output variable associated with an unstable state which is a transient state between two stable states that have different values (0 and 1, or 1 and 0) in the corresponding output variable.

Design Procedure

- **Summary**
  - a primitive flow table
  - state reduction
  - state assignment
  - output assignment
  - Simplify the Boolean functions of the excitation and output variables and draw the logic diagram

9-5 Reduction of State and Flow Tables

- **Equivalent states**
  - for each input, two states give exactly the same output and go to the same next states or to equivalent next states

Table 9.3
State Table to Demonstrate Equivalent States

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

Reduction of State and Flow Tables

- (a,b) are equivalent if (c,d) are equivalent
- (a,b) imply (c,d)
- (c,d) imply (a,b)
- both pairs are equivalent

Table 9.3

State Table to Demonstrate Equivalent States

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

Reduction of State and Flow Tables

- The checking of each pair of states for possible equivalence

Table 9.4
State Table to Be Reduced

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

Reduction of State and Flow Tables

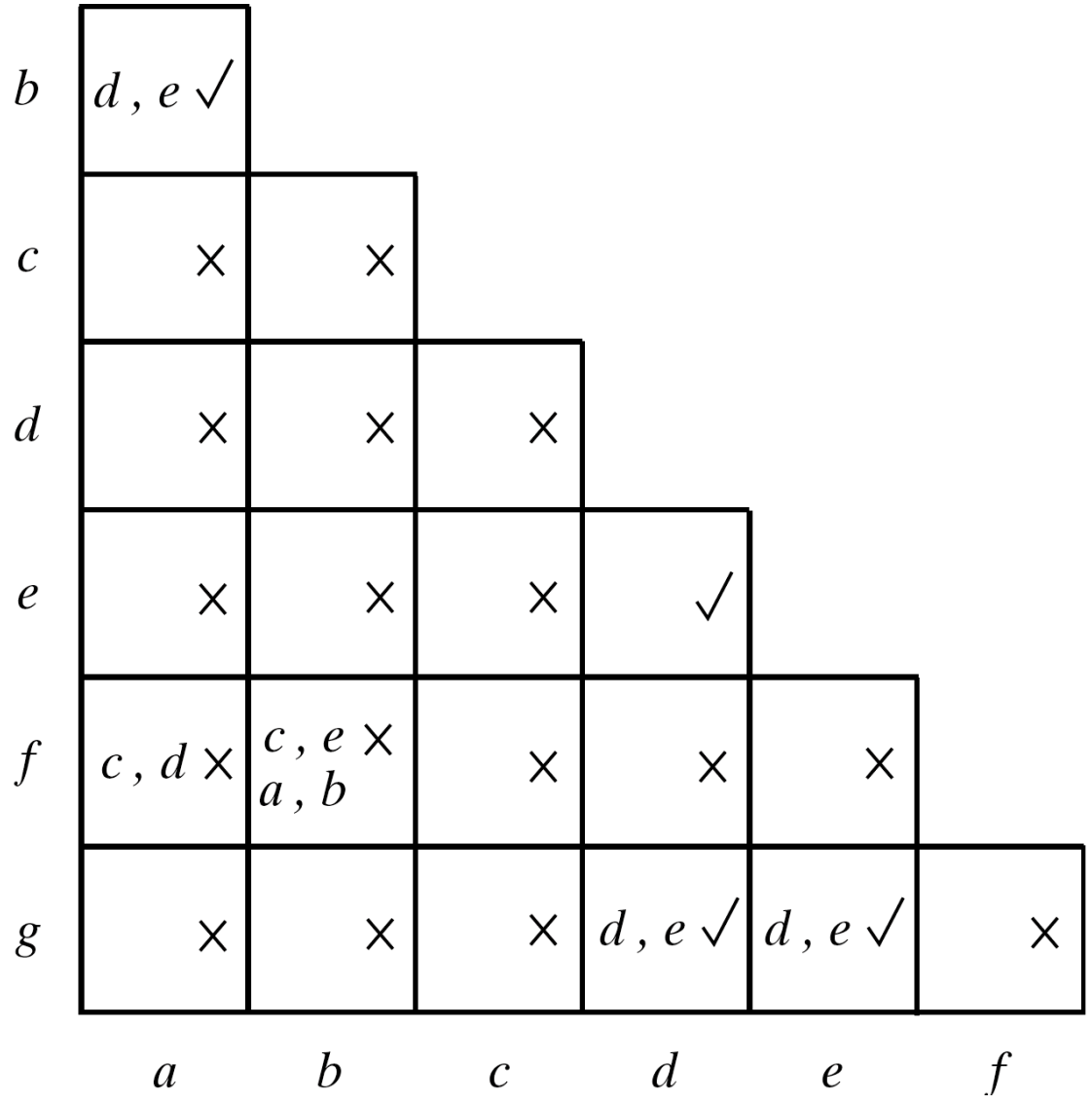
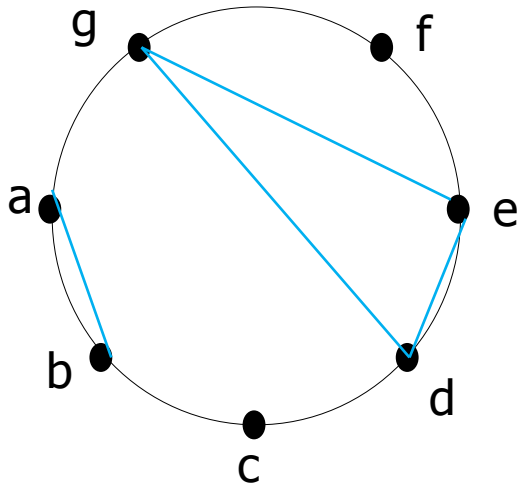


Fig. 9.22
Implication table

Reduction of State and Flow Tables

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



Compatibles	(a,b)	(c)	(d,e,g)	(f)
Implied states	(d,e)	-	(d,e)	-

Table 9.5
Reduced State Table

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

(a,b), (c), (d,e,g), (f)
- cover all the states
- closed

Reduction of State and Flow Tables

- **Merging of the Flow Table**
  - consider the don't-care conditions
  - combinations of inputs or input sequences may never occur
  - two incompletely specified states that can be combined are said to be compatible
  - for each possible input they have the same output whenever specified and their next states are compatible whenever they are specified
  - **determine all compatible pairs**
  - **find the maximal compatibles**
  - **find a minimal closed covering**

Reduction of State and Flow Tables

- **Compatible pairs**
  - (a,b) (a,c) (a,d) (b,e) (b,f) (c,d) (e,f)

	00	01	11	10
a	*c* , –	(*a*), 0	*b* , –	– , –
b	– , –	*a* , –	(*b*), 1	*e* , –
c	(*c*), 0	*a* , –	– , –	*d* , –
d	*c* , –	– , –	*b* , –	(*d*), 0
e	*f* , –	– , –	*b* , –	(*e*), 1
f	(*f*), 1	*a* , –	– , –	*e* , –

(a) Primitive flow table

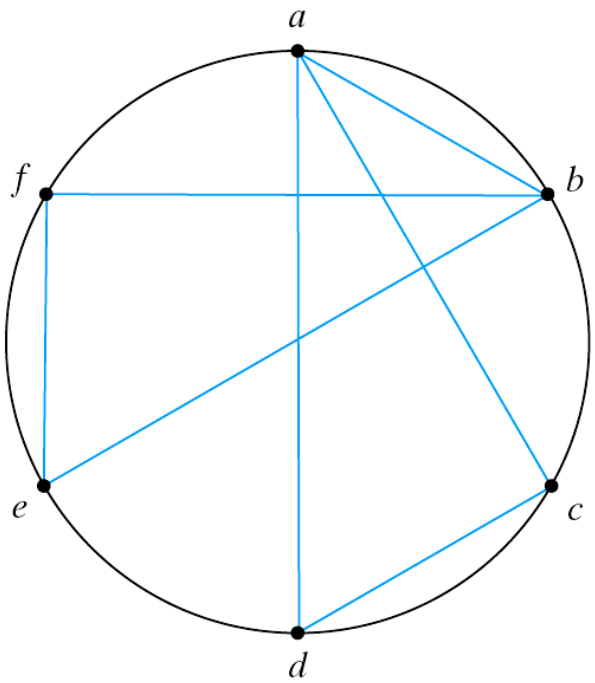
	a	*b*	*c*	*d*	*e*
b	✓				
c	✓	*d* , *e* ×			
d	✓	*d* , *e* ×	✓		
e	*c* , *f* ×	✓	*d* , *e* × *c* , *f* ×	×	
f	*c* , *f* ×	✓	×	*d* , *e* × *c* , *f* ×	✓

(b) Implication table

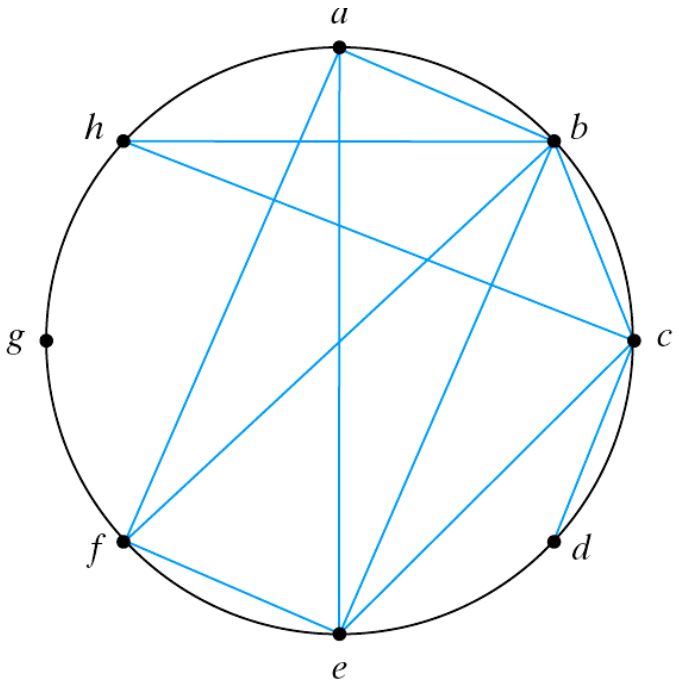
Fig. 9.23
Flow and
implication tables

Reduction of State and Flow Tables

- **Maximal Compatibles**
  - a group of compatibles that contains all the possible combinations of compatible states
  - **Merger Diagram**



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

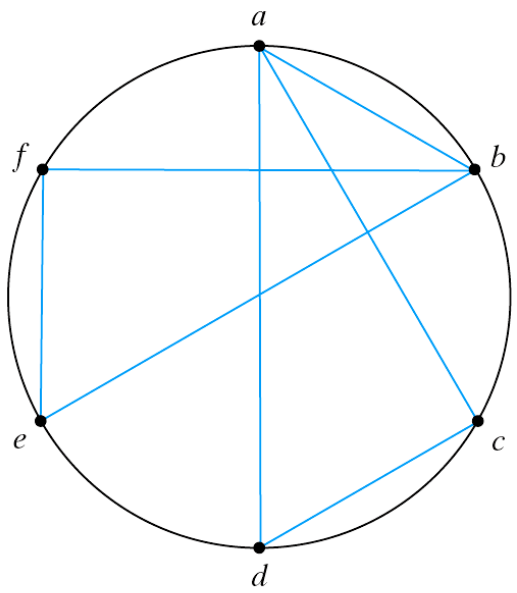


(b) Maximal compatible:
(a, b, e, f) (b, c, h) (c, d) (g)

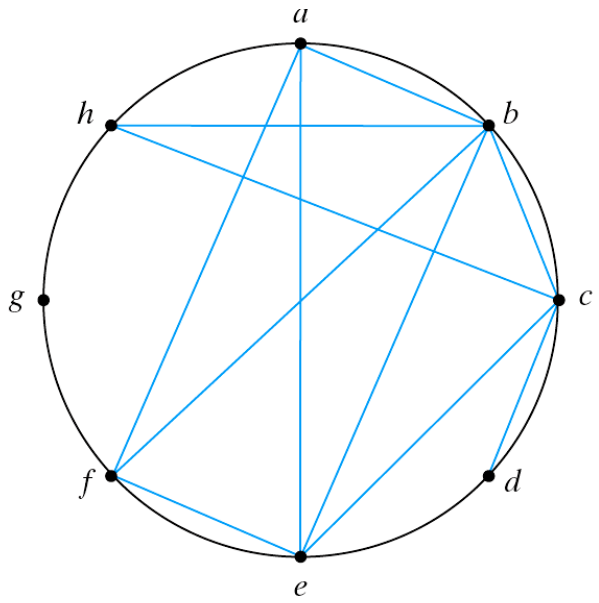
Fig. 9.24
Merger diagram

Reduction of State and Flow Tables

- **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)

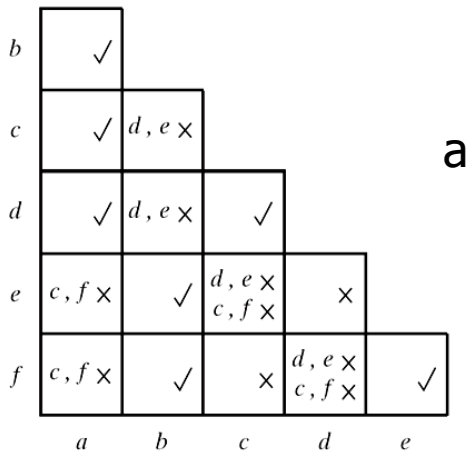


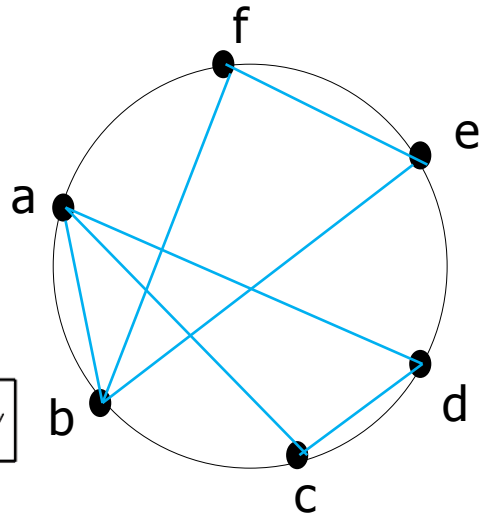
(b) Maximal compatible:
(a, b, e, f) (b, c, h) (c, d) (g)

Fig. 9.24
Merger diagram

Reduction of State and Flow Tables

- **Closed covering condition**
  - cover all the states
  - closed
    - no implied states or the implied states are included within the set
  - (a,c,d) (b,e,f)
    - cover all the states
    - no implied states





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

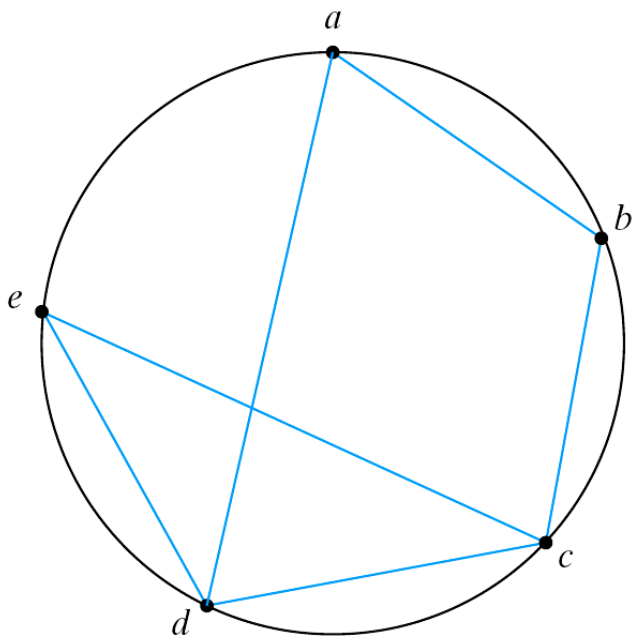
Reduction of State and Flow Tables

another example

Primitive flow table is not shown

	a	b	c	d
b	b , c ✓			
c	×	d , e ✓		
d	b , c ✓	×	a , d ✓	
e	×	×	✓	b , c ✓

(a) Implication table



(b) Merger diagram

Fig. 9.25
Choosing a set of compatibles

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

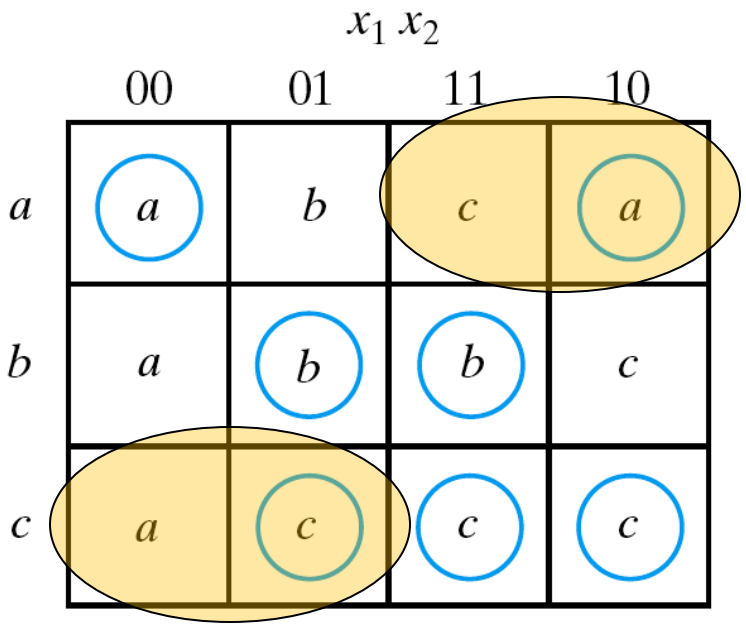
(c) Closure table

Reduction of State and Flow Tables

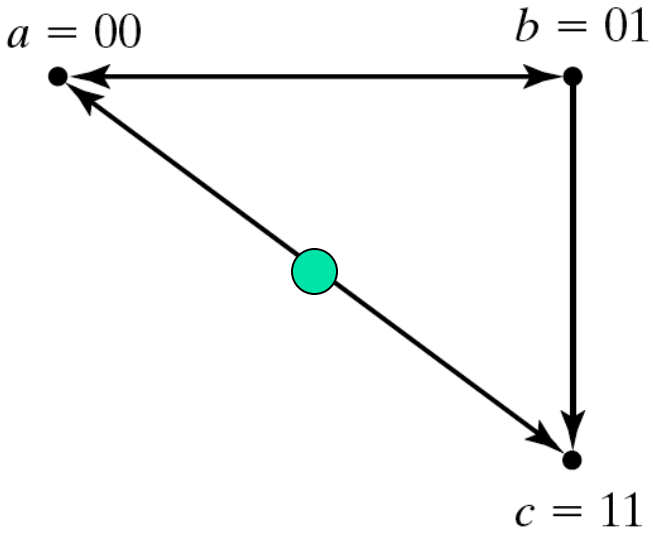
- (a,b) (c,d,e)
  - cover all the states
  - but not closed
    - (b,c) are implied but not included
- (a,d) (b,c) (c,d,e)
  - cover all the states
  - closed
    - implied states: (b,c) (d,e) (a,d)
  - the same state can be repeated more than once
- The original flow table (not shown) can be reduced from five rows to three rows by merging rows a and b, b and c, and c, d, and e.

9-6 Race-Free State Assignment

- To avoid critical races
  - only one variable changes at any given time
- **Three-row flow-table example**
  - flow-table and transition diagram example



(a) Flow table



(b) Transition diagram

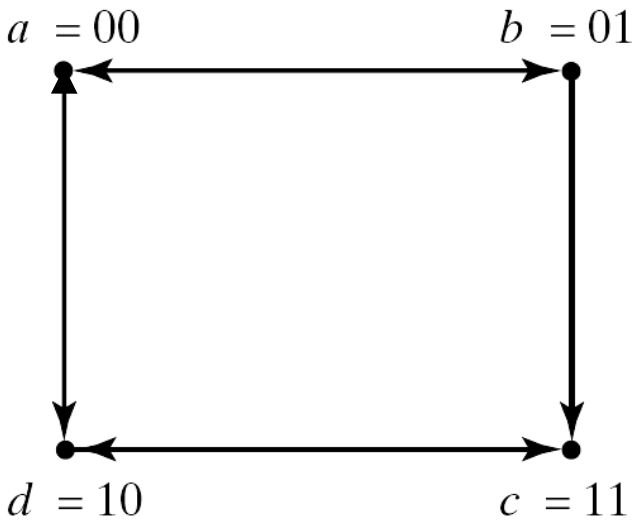
Fig. 9.26
Three-row flow-table example

Race-Free State Assignment

- an extra row is added
- no stable state 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



(b) Transition diagram

Fig. 9.27
Flow-table with an extra row

Race-Free State Assignment

Transition 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	–

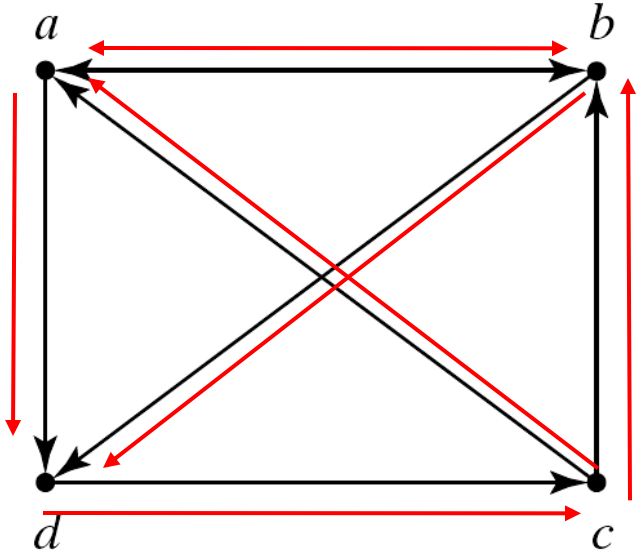
Fig. 9.28
Flow-table with an extra row

Race-Free State Assignment

- **Four-row flow-table example**
  - flow-table and transition diagram

	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

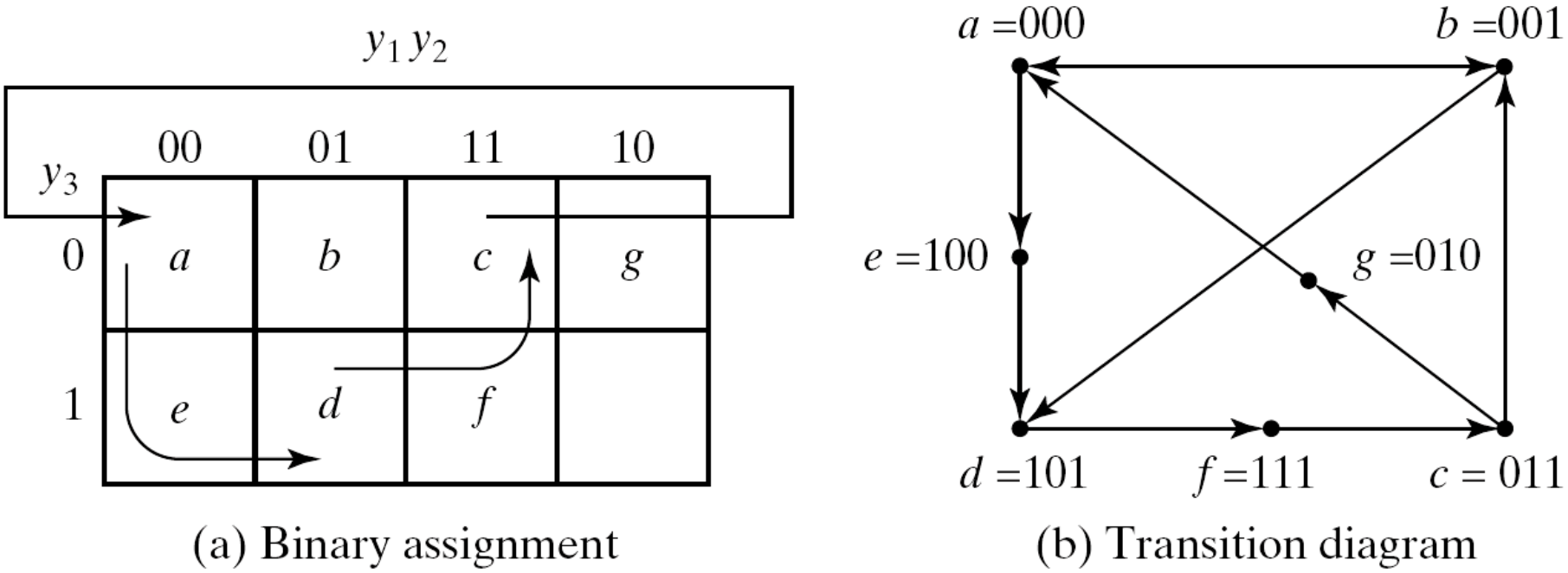


(b) Transition diagram

Fig. 9.29
Four-row flow-table example

Race-Free State Assignment

- add extra rows

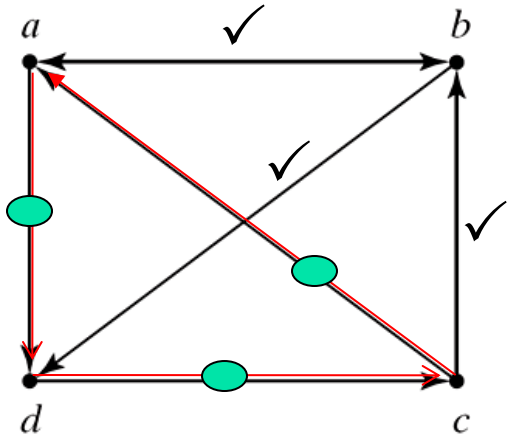


(a) Binary assignment

(b) Transition diagram

Fig. 9.30
Choosing extra rows for the flow table

Race-Free State Assignment



	00	01	11	10
0	a	b	c	
1		d		

	00	01	11	10
0	a	b	c	
1	e	d		

	00	01	11	10
0	a	b	c	g
1	e	d		

	00	01	11	10
0	a	b	c	g
1	e	d	f	

- State b is assigned to 001 (randomly).
- State b is adjacent to the other three original states (a, c, d).
- Transition a(000) → d(101) ⇨ directed thru e.
- Transition c(011) → a(000) ⇨ directed thru g.
- Transition d(101) → c(011) ⇨ directed thru f.

Race-Free State Assignment

- The modified flow table

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

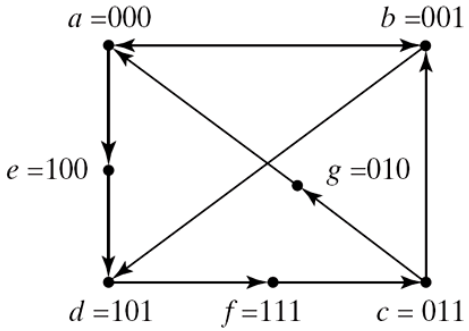


Fig. 9.31
State assignment to modified flow table

	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	–

Race-Free State Assignment

Multiple-row method

	00	01	11	10
a	*b*	(*a*)	*d*	(*a*)
b	(*b*)	*d*	(*b*)	*a*
c	(*c*)	*a*	*b*	(*c*)
d	*c*	(*d*)	(*d*)	*c*

- less efficient
- multiple equivalent states for each state

y_1 \ y_2y_3	00	01	11	10
0	a_1	b_1	c_1	d_1
1	c_2	d_2	a_2	b_2

(a) Binary assignment

	00	01	11	10
000 = a_1	b_1	(a_1)	d_1	(a_1)
111 = a_2	b_2	(a_2)	d_2	(a_2)
001 = b_1	(b_1)	d_2	(b_1)	a_1
110 = b_2	(b_2)	d_1	(b_2)	a_2
011 = c_1	(c_1)	a_2	b_1	(c_1)
100 = c_2	(c_2)	a_1	b_2	(c_2)
010 = d_1	c_1	(d_1)	(d_1)	c_1
101 = d_2	c_2	(d_2)	(d_2)	c_2

(b) Flow table

Fig. 9.32
Multiple-row assignment

Race-Free State Assignment

Multiple-row method

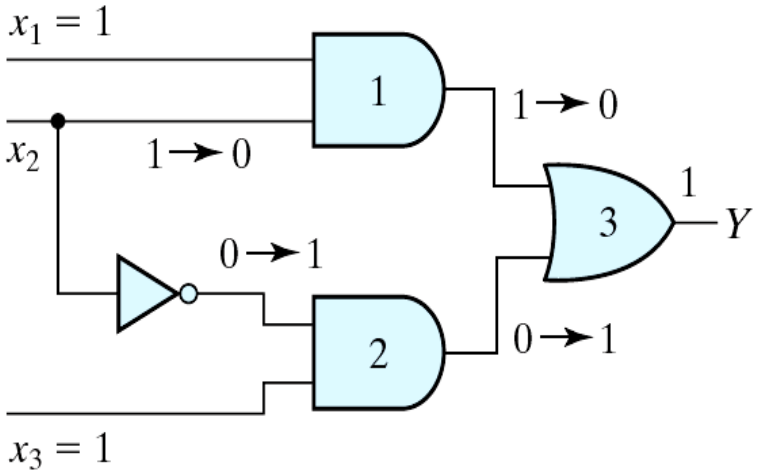
- The expanded table is formed by replacing each row of the original table with two rows.
- For example, row b is replaced by rows b_1 and b_2.
- Stable state b is entered in columns 00 and 11 in both b_1 and b_2.
- After all the stable states have been entered, the unstable states are filled in by reference to the binary assignment
- When choosing the next state for a given present state, a state which is adjacent to the present state is selected from the map.

9-7 Hazards

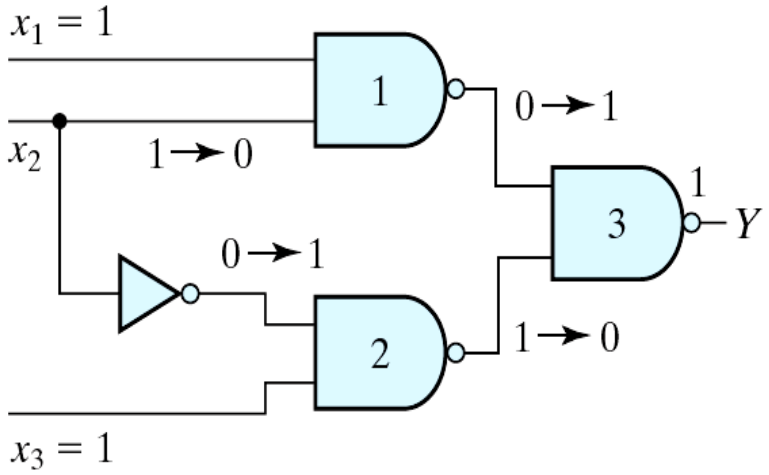
- Unwanted switching transients at the output
  - different paths exhibit different propagation delays
  - temporary false-output value in combinational circuits
  - may result in a transition to a wrong stable state in asynchronous sequential circuits

Hazards

- Hazards in combinational circuits
  - examples



(a) AND–OR circuit



(b) NAND circuit

Fig. 9.33
Circuits with hazards

Hazards

- static 1-hazard (sum of products)
  - the removal of static 1-hazard guarantees that no static 0-hazards or dynamic hazards

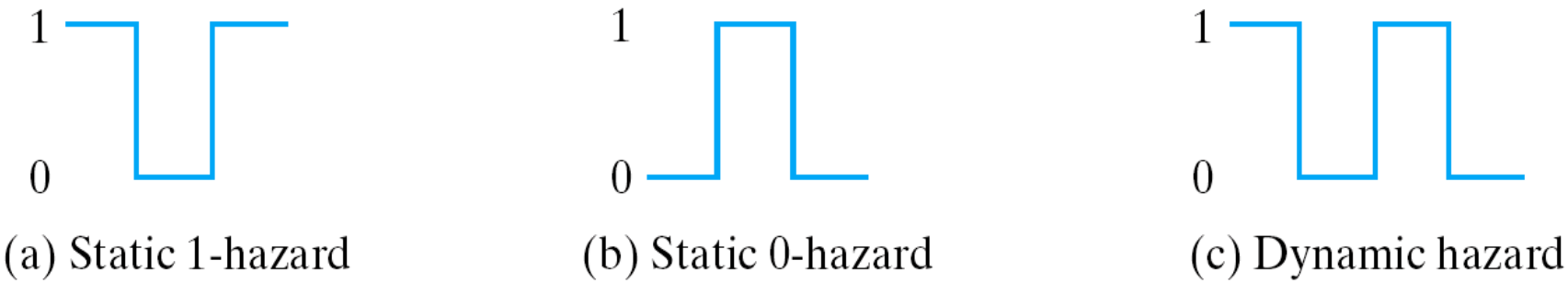


Fig. 9.34
Types of hazards

Hazards

- static 0-hazard (product of sum)
  - $Y = (x_1+x_2')(x_2+x_3)$
- The remedy
  - the circuit moves from one product term to another
  - additional redundant gate

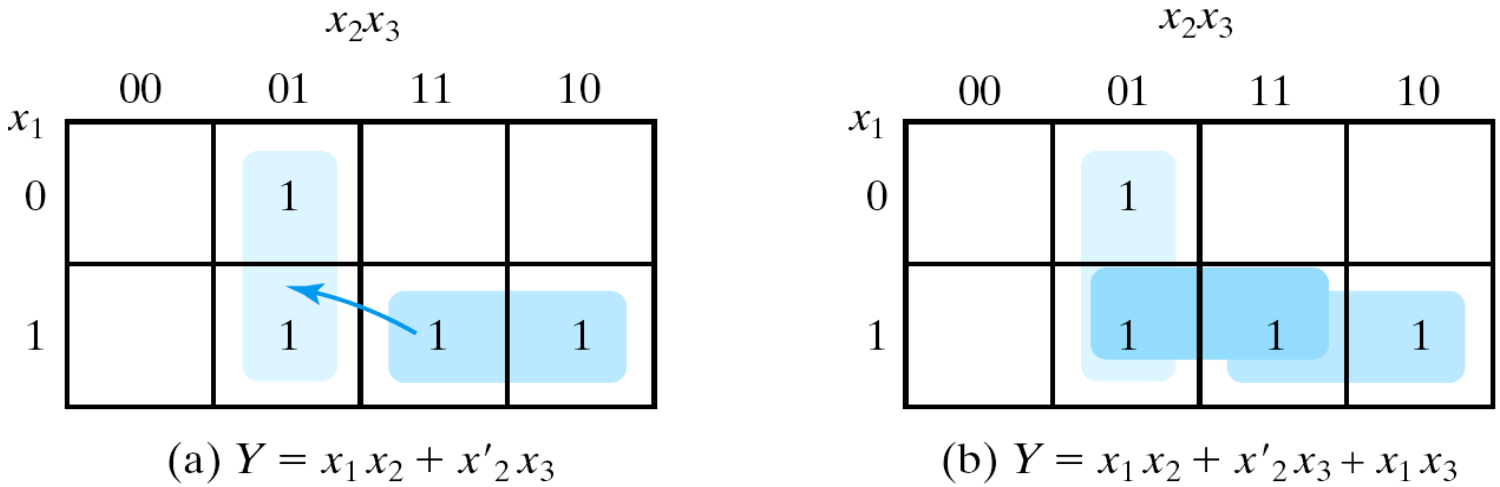


Fig. 9.35
Maps illustrating a hazard and its removal

Hazards

- Hazard-free circuit

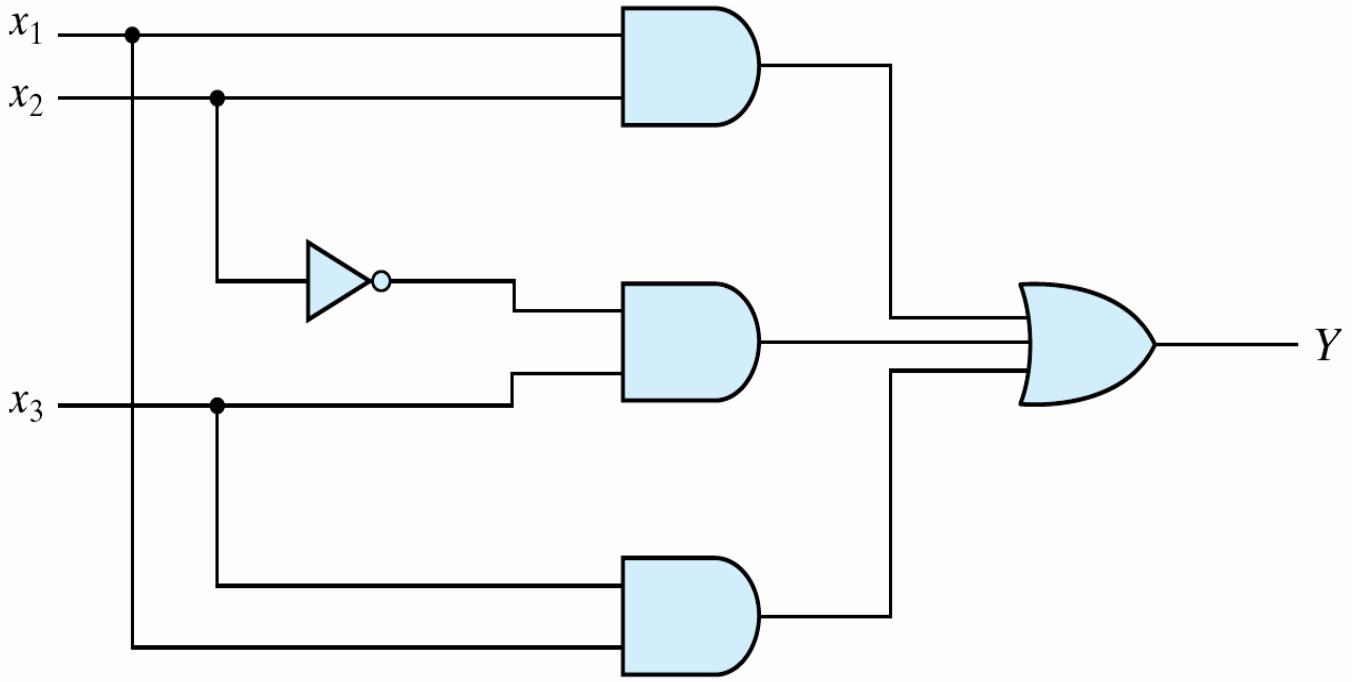
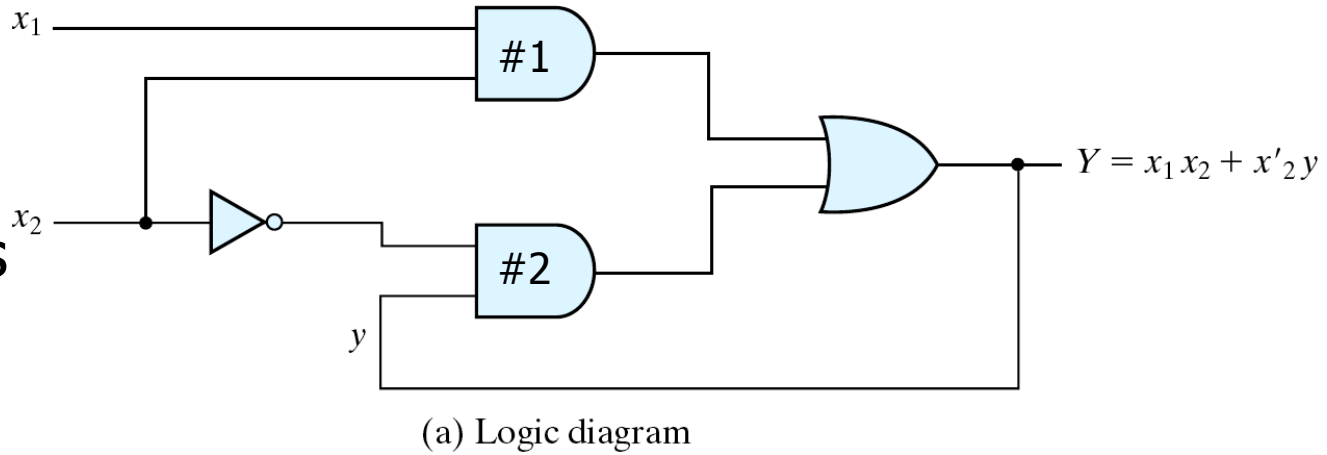


Fig. 9.36
Hazard-free circuit

Hazards

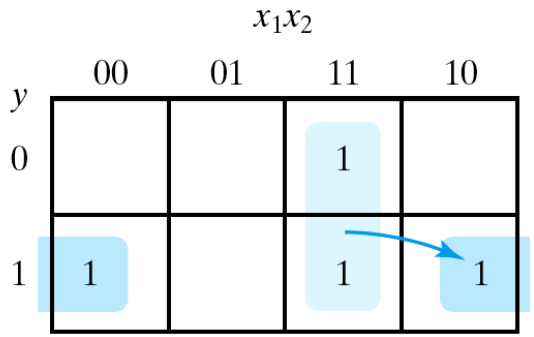
- Hazards in sequential circuits
  - in general, no problem for synchronous design
  - an asynchronous example



(a) Logic diagram

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

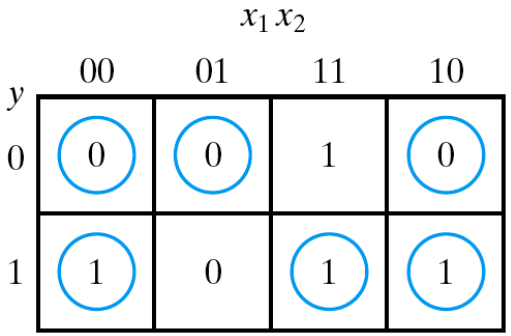
(b) Transition table



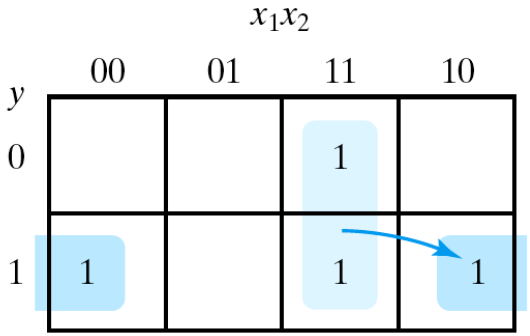
(c) Map for Y

Fig. 9.37 Hazard in a asynchronous sequential circuit

Hazards

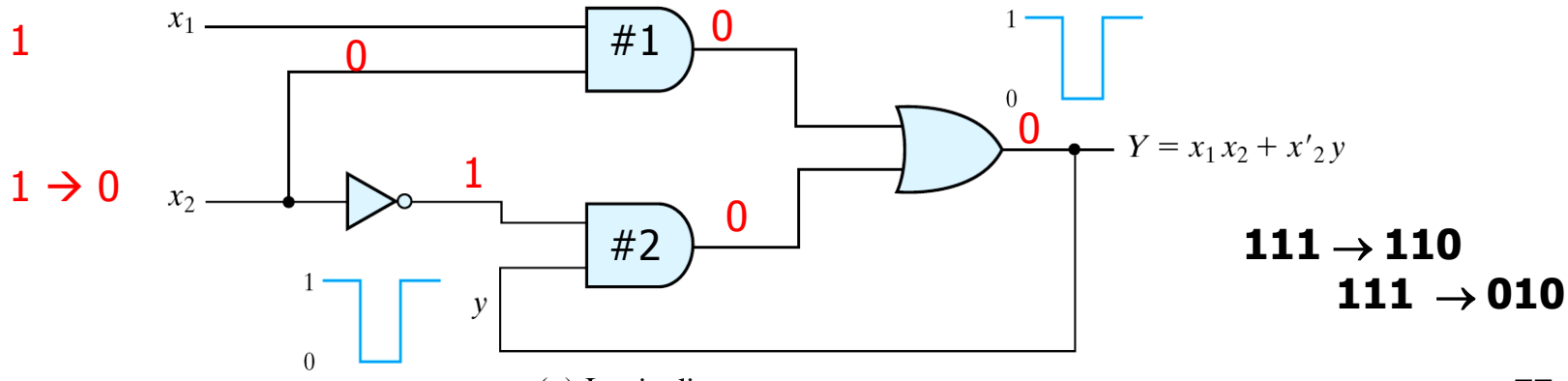


(b) Transition table



(c) Map for Y

- Because of hazard, output Y may go to 0 momentarily.
- If the false signal feed back into the AND gate #2 before the output of the inverter goes to 1, the output of gate 2 will remain at 0 and the circuit will switch to the incorrect total stable state 010.
- The malfunction can be eliminated by adding an extra gate.



(a) Logic diagram

Hazards

- 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

Hazards

- Implementation with SR latches

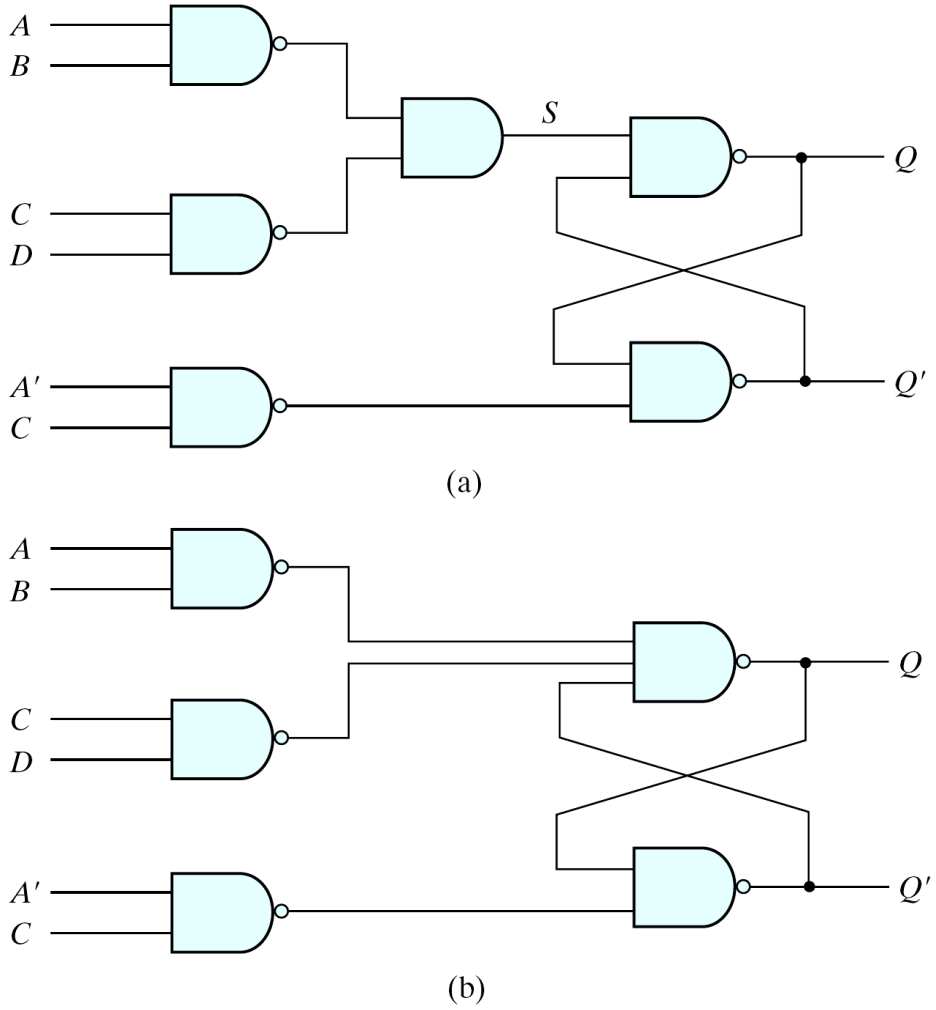


Fig. 9.38
Latch implementation

Hazards

- Implementation with SR latches
  - For NAND SR latch:

Boolean functions for S and R:

$$S = AB + CD$$

$$R = A'C$$

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

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

The Boolean function for output Q is

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

Hazards

- **Essential Hazards**
  - asynchronous sequential circuits
  - unequal delays along two or more paths that originate from the same input
  - cannot be corrected by adding redundant gates
  - the delay of feedback loops > delays of other signals that originate from the input terminals

9.8 Design Example

- Summary of design procedure
  - State the design spec.
  - Derive the primitive flow table
  - Reduce the flow table by merging the rows
  - Race-free state assignment
  - Obtain the transition table and output map
  - Obtain the logic diagram using SR latches

Design Specification

Table 9.6
Specification of Total States

	Inputs		Output	
State	***T***	**C**	**Q**	**Comments**
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 state *d* or *f*
f	0	1	0	After state *e* or *a*
g	0	0	1	After state *b* or *h*
h	0	1	1	After state *g* or *c*

Primitive Flow table

	TC = 00	TC = 01	TC = 11	TC = 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 , $-$	$-$, $-$

Fig. 9.39
Primitive flow table

Merging of the Flow Table

- **Compatible pairs:**

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

- **Maximal compatible set:**

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

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

Fig. 9.40
Implication table

Merging of the Flow Table

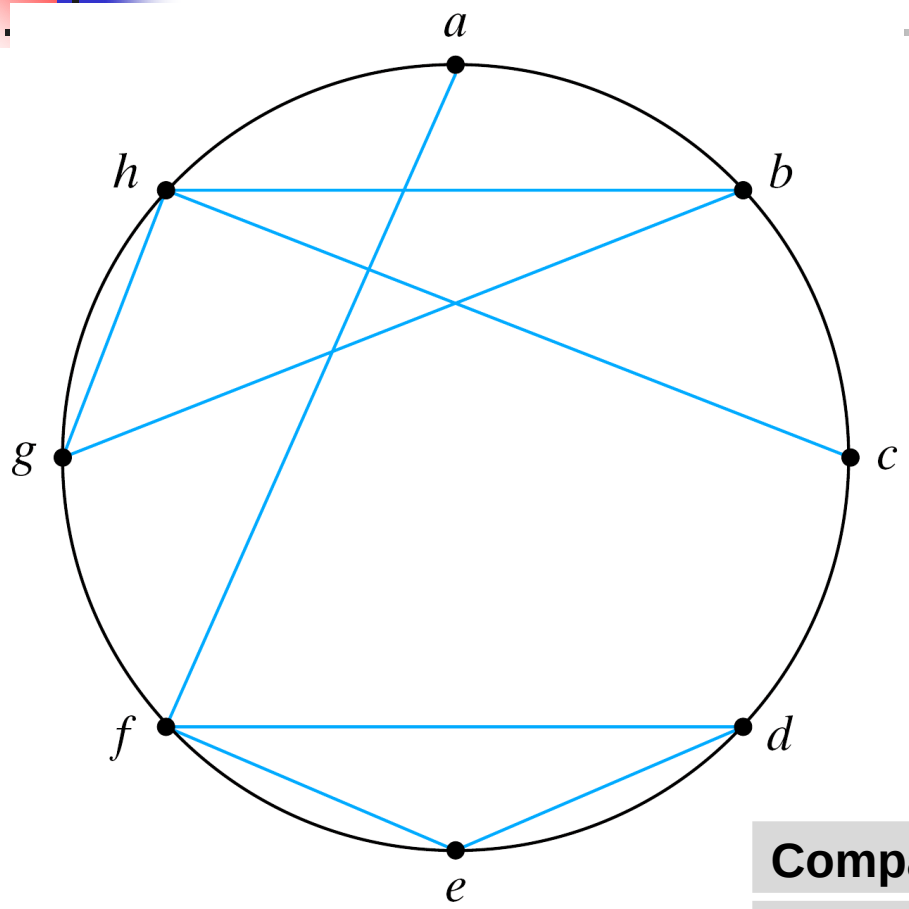


Fig. 9.41
Merger diagram

Closure Table

Compatibles	(a,f)	(b,g,h)	(c,h)	(d,e,f)
Implied states	-	-	-	-

(a,f), (b,g,h), (c,h), (d,e,f)
- cover all the states
- closed

Merging of the Flow Table

	TC 00	01	11	10
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)

	TC 00	01	11	10
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)

Fig. 9.42
Reduced flow table

State Assignment and Transition Table

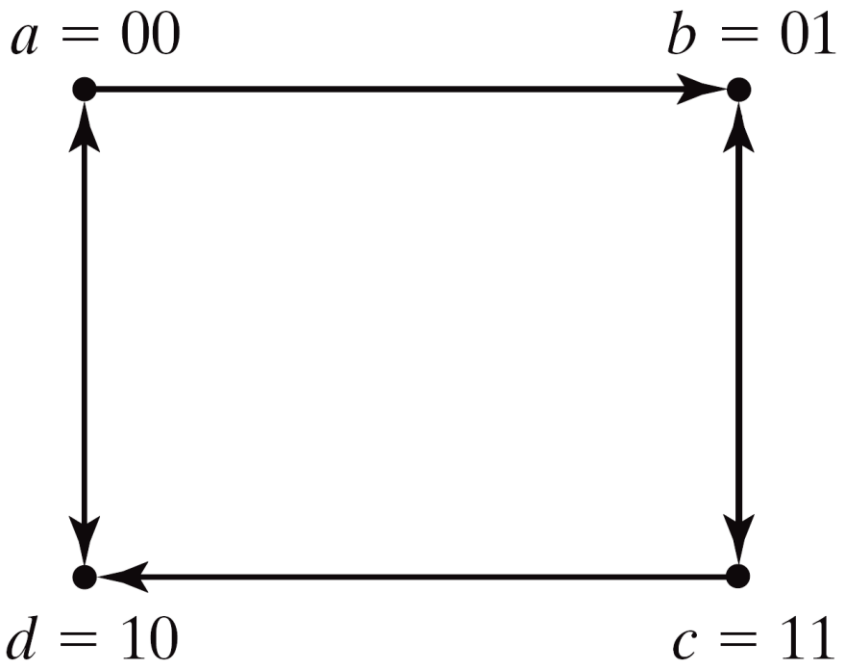
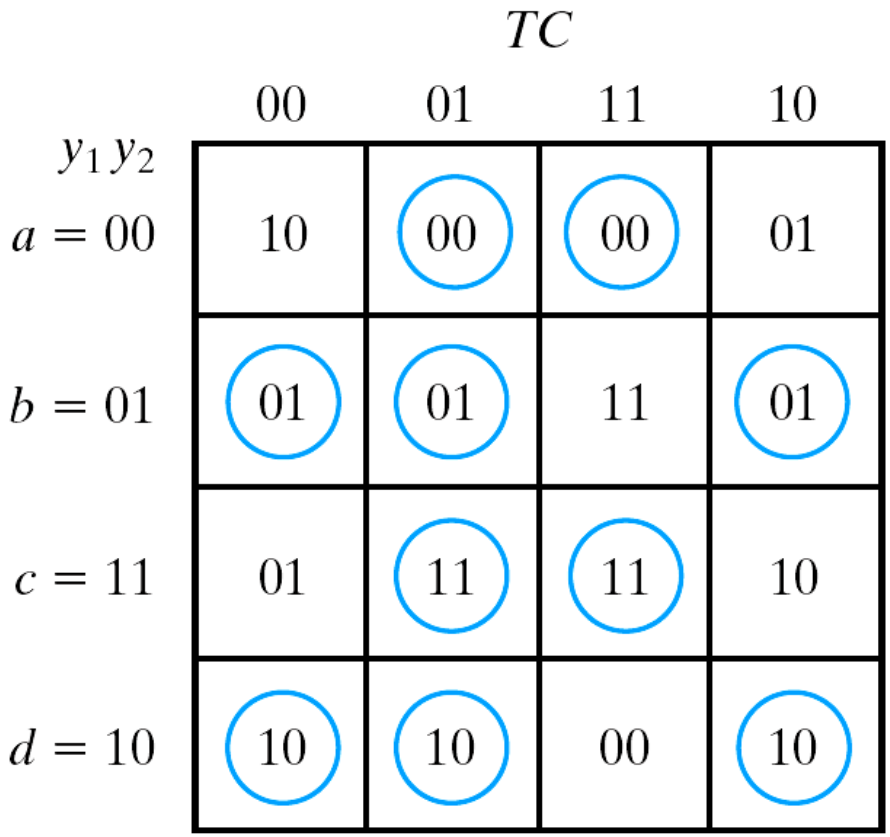
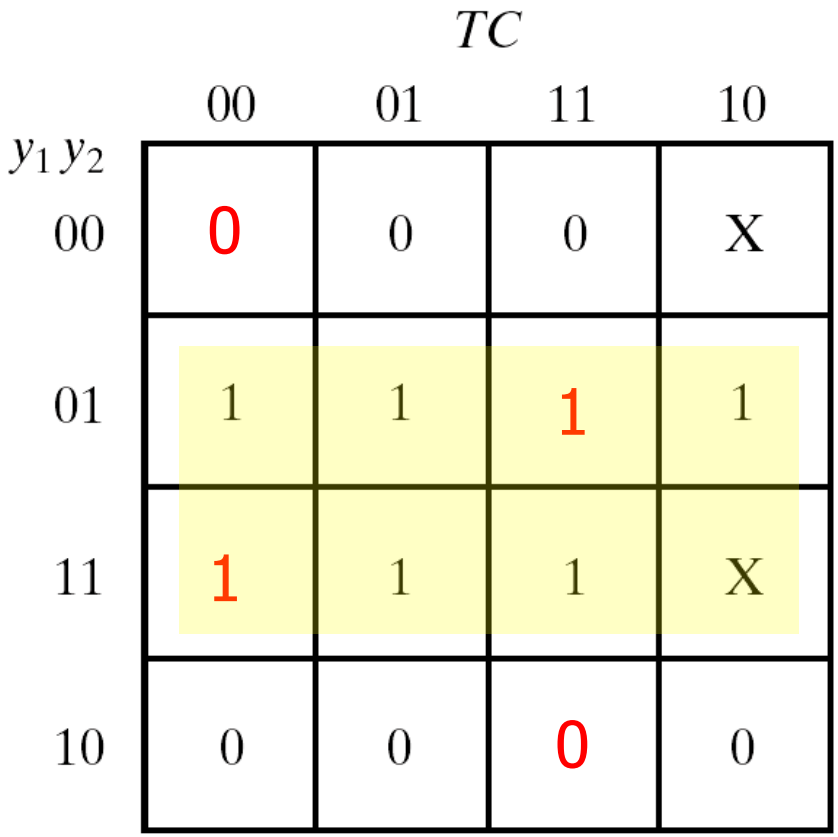


Fig. 9.43
Transition diagram

State Assignment and Transition Table



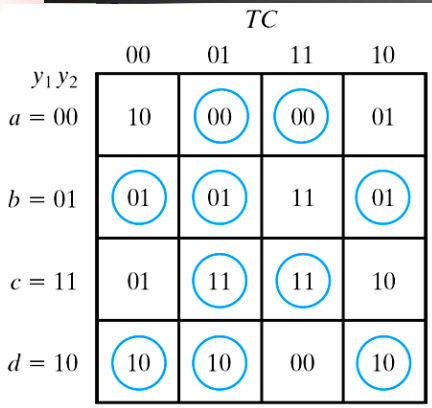
(a) Transition table



(b) Output map $Q = y_2$

Fig. 9.44
Transition table and output map

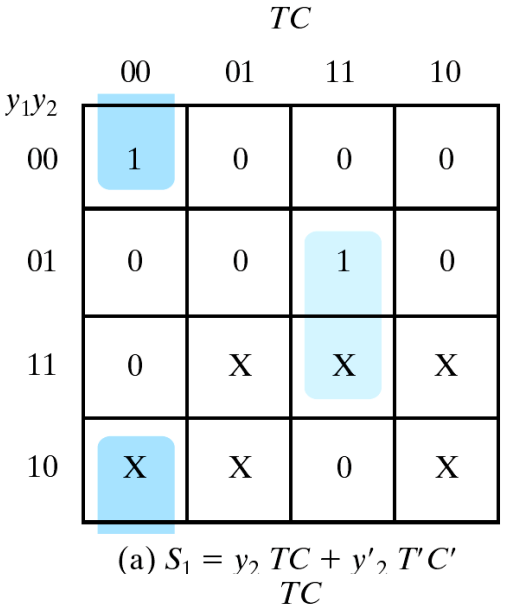
Logic Diagram



TC / y_1y_2	00	01	11	10
$a = 00$	10	00	00	01
$b = 01$	01	01	11	01
$c = 11$	01	11	11	10
$d = 10$	10	10	00	10

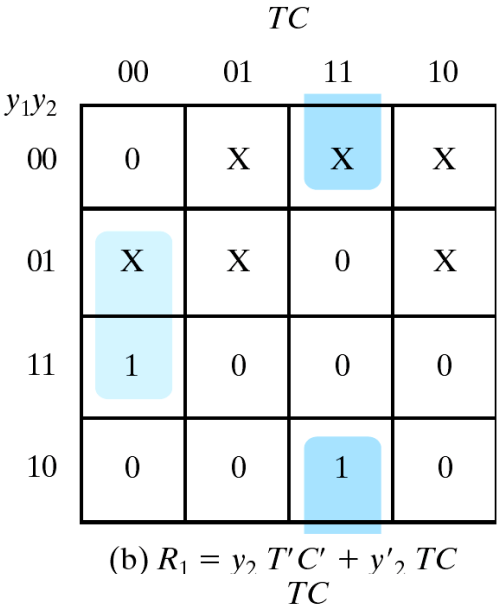
y	Y	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

(b) Latch excitation table



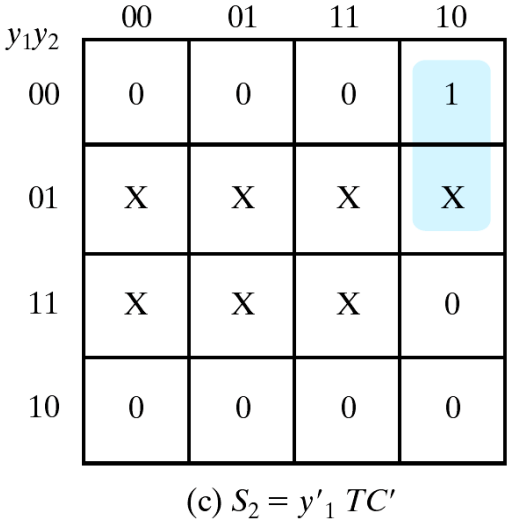
TC / y_1y_2	00	01	11	10
00	1	0	0	0
01	0	0	1	0
11	0	X	X	X
10	X	X	0	X

(a) $S_1 = y_2\,TC + y'_2\,T'C'$



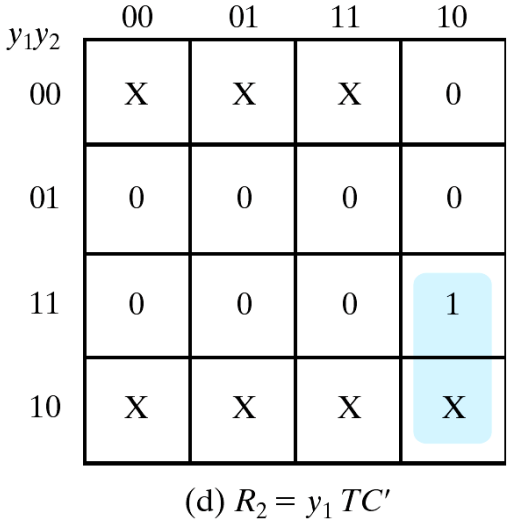
TC / y_1y_2	00	01	11	10
00	0	X	X	X
01	X	X	0	X
11	1	0	0	0
10	0	0	1	0

(b) $R_1 = y_2\,T'C' + y'_2\,TC$



TC / y_1y_2	00	01	11	10
00	0	0	0	1
01	X	X	X	X
11	X	X	X	0
10	0	0	0	0

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



TC / y_1y_2	00	01	11	10
00	X	X	X	0
01	0	0	0	0
11	0	0	0	1
10	X	X	X	X

(d) $R_2 = y_1\,TC'$

Fig. 9.45
Maps for latch inputs

Logic Diagram

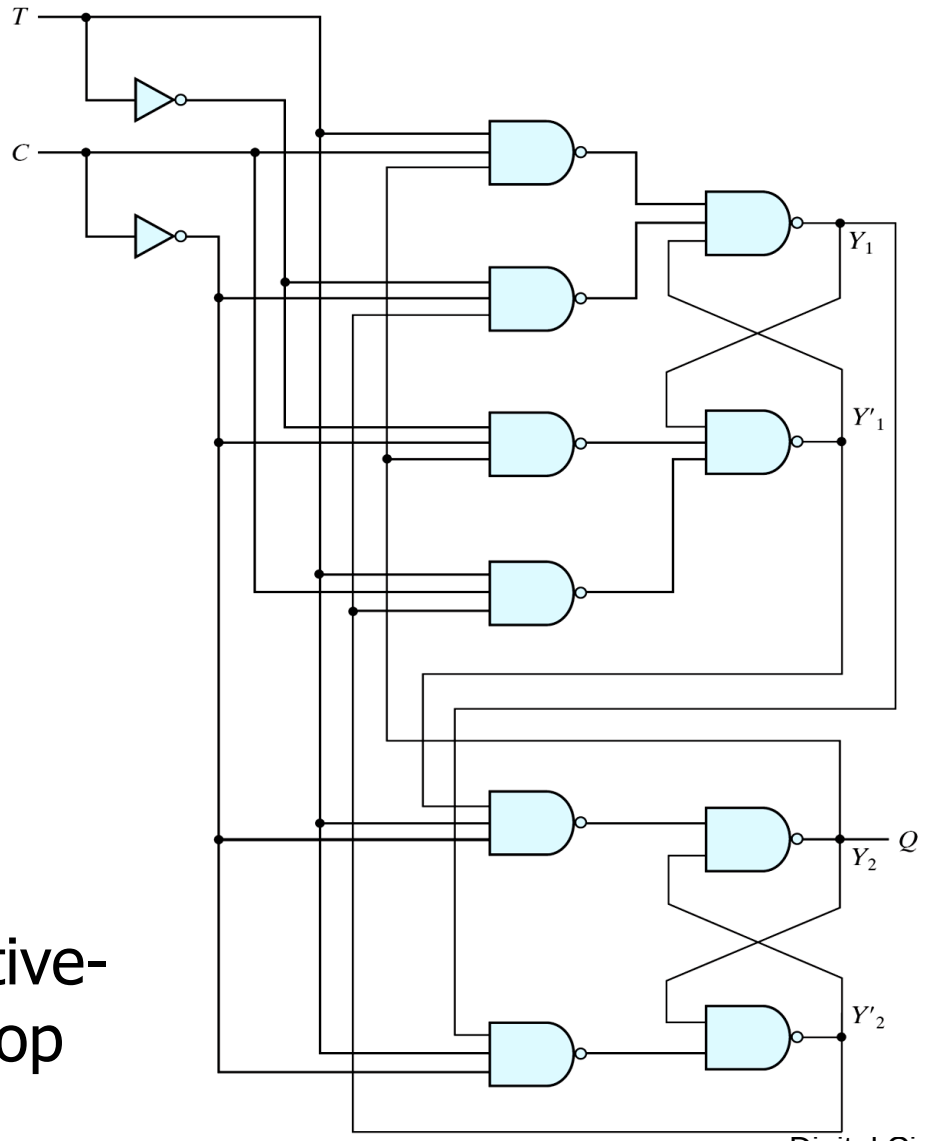


Fig. 9.46
Logic diagram of negative-edge-triggered *T* flip-flop