CSE 205: Digital Logic Design

Dr. Tanzima Hashem
Associate Professor
CSE, BUET

Sequential Circuits

- Consist of a combinational circuit to which storage elements are connected to form a feedback path
- Specified by **a time sequence of inputs, outputs** and **internal states**

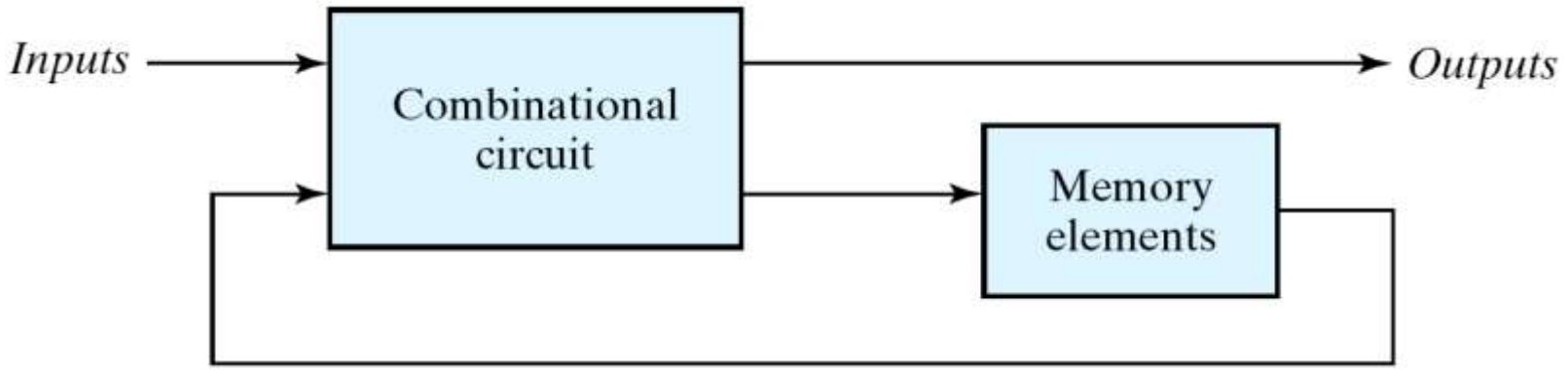


Fig. 5-1 Block Diagram of Sequential Circuit

Synchronous Sequential Circuit

- Synchronous sequential circuits
  - Synchronized by a **periodic train** of clock pulses
  - Much easier to design (preferred design style)

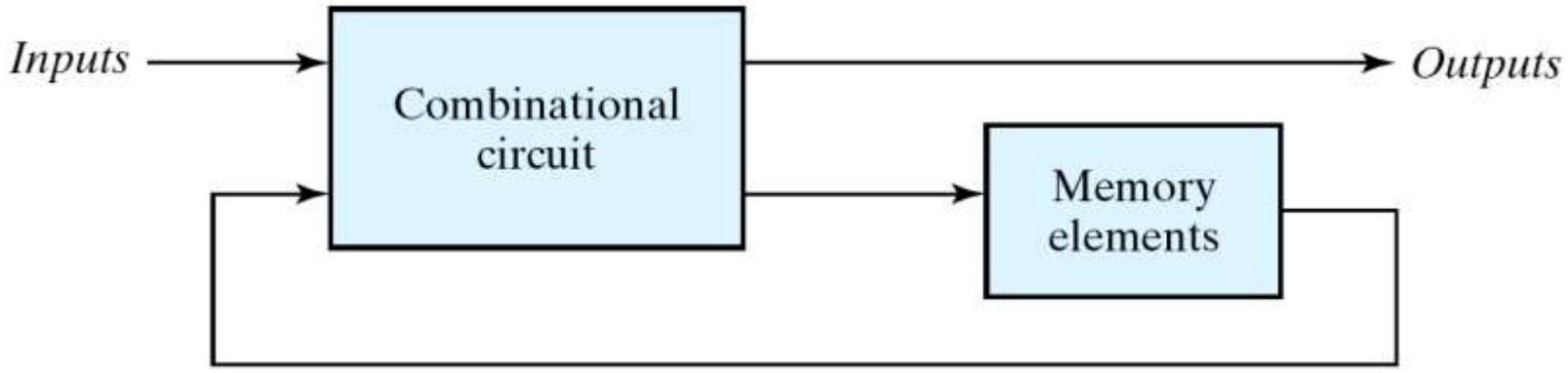
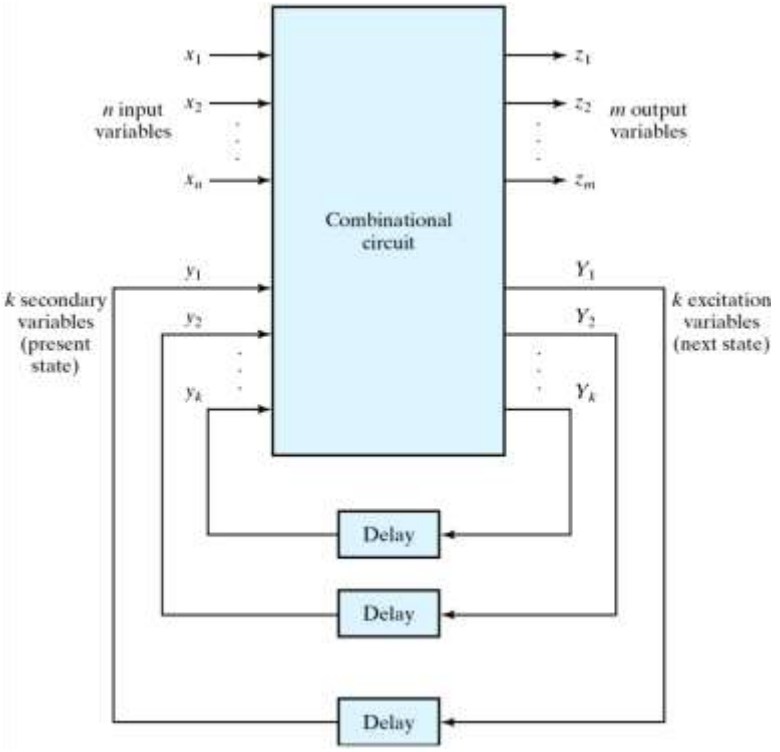


Fig. 5-1 Block Diagram of Sequential Circuit

Asynchronous Sequential Circuit

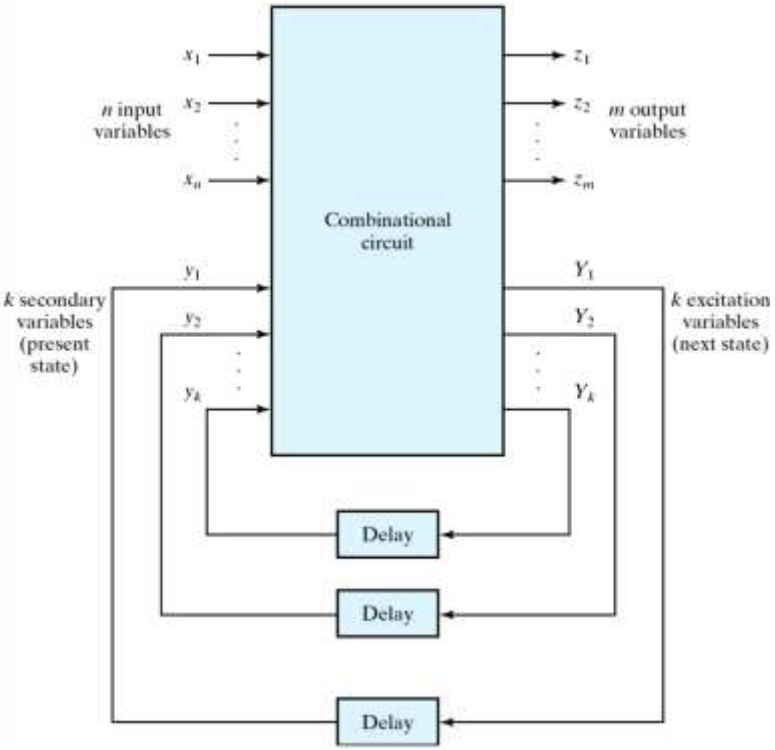
- Inputs / Outputs
- Delay elements:
  - Only a short term memory
  - May not really exist due to original gate delay
- Secondary variable:
  - Current state (small y)
- Excitation variable:
  - Next state (big Y)
  - Have some delay in response to input changes



Block diagram of an asynchronous sequential circuit

Asynchronous Sequential Circuit

- Internal states can change at **any instant of time when there is a** change in the input variables
- **No clock signal is required**
- Have better performance but hard to design due to timing problems



Block diagram of an asynchronous sequential circuit

Why Asynchronous Circuits?

- Used when speed of operation is important
  - Response quickly without waiting for a clock pulse
- Used in small independent systems
  - Only a few components are required
- Used when the input signals may change independently of internal clock
  - Asynchronous in nature
- Used in the communication between two units that have their own independent clocks
  - Must be done in an asynchronous fashion

Operational Mode

- Steady-state condition:
  - Current states and next states are the same
  - Difference between Y and y will cause a transition
- Fundamental mode:
  - No simultaneous changes of two or more variables
  - The time between two input changes must be longer than the time it takes the circuit to a stable state
  - The input signals change one at a time and only when the circuit is in a stable condition

Analysis Procedure 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
  5. Circle those values of Y in each square that are equal to the value of y in the same row

An Example of Transition Table

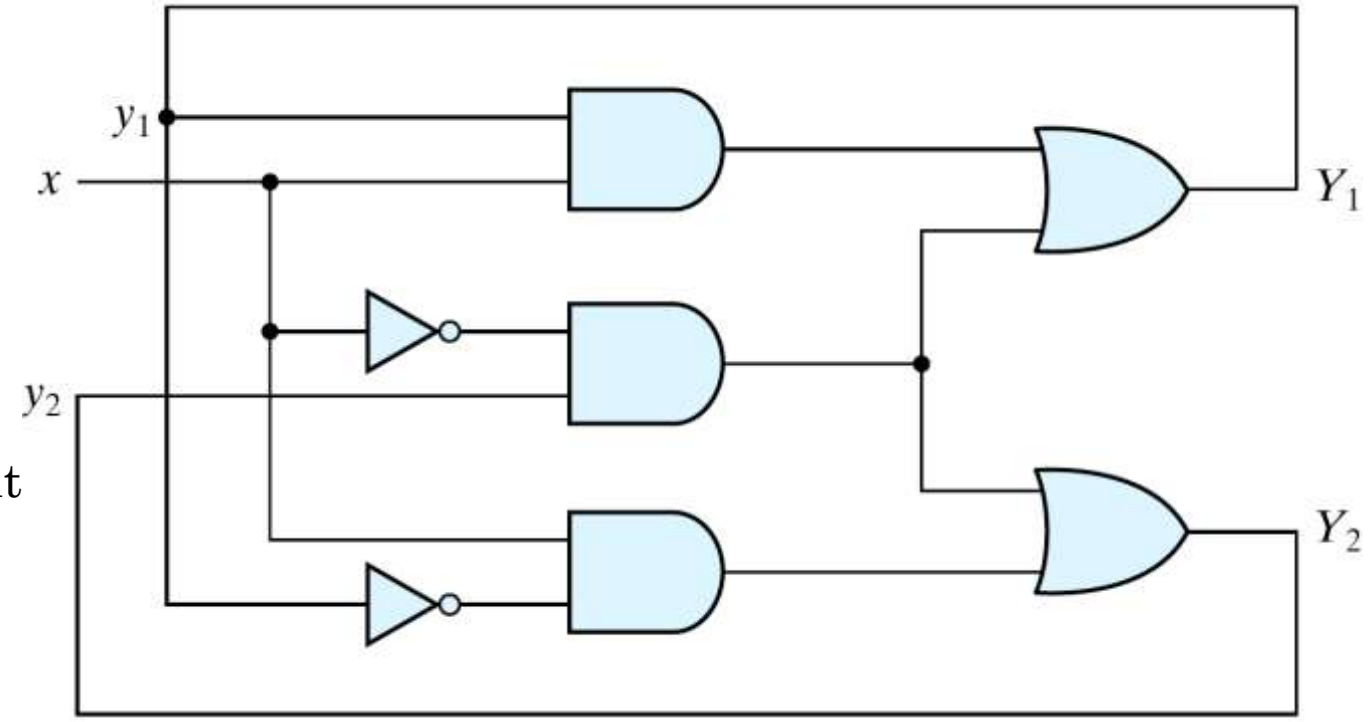


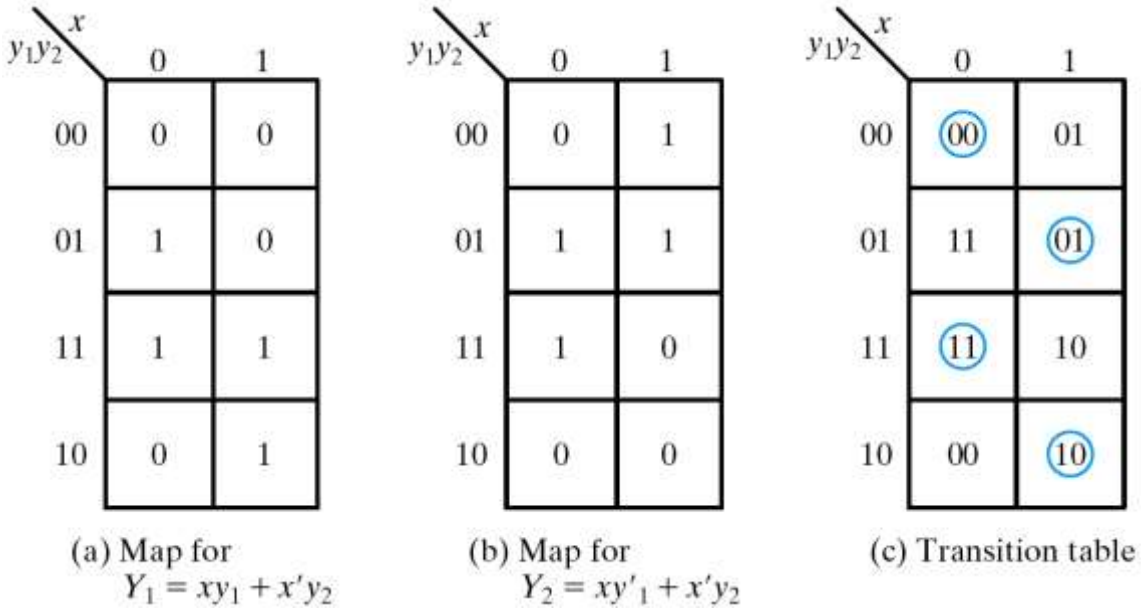
Fig. 9.2
Example of an asynchronous sequential circuit

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

An Example of Transition Table

- Maps and transition table

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



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

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

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

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

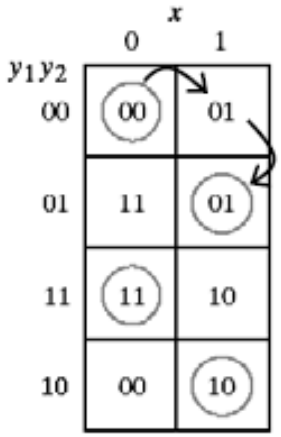
(c) Transition table

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

- y variables for the rows
- External variable for the columns
- Circle the stable states
  - $Y = y$

State Table

- When input x changes from 0 to 1 while y=00:
  - Y changes to 01 → unstable
  - y becomes 01 after a short delay → stable at the second row



(c) Transition table

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

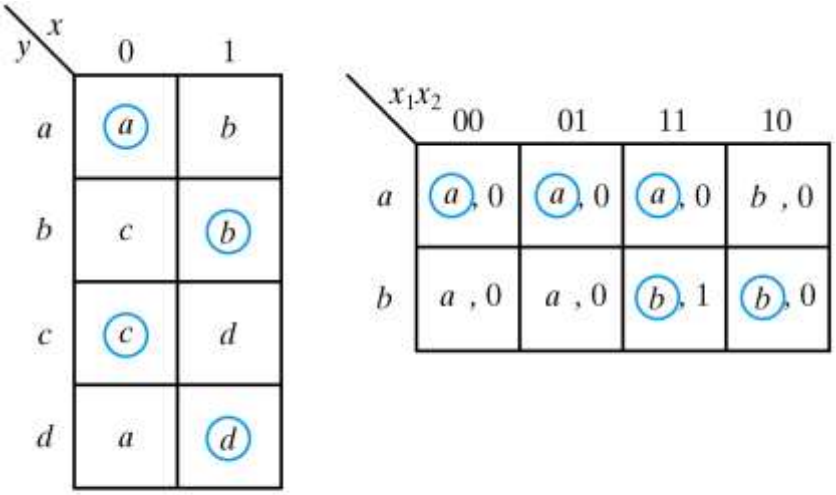
y_1y_2x : total state

4 stable total states: 000,011, 110,101

Flow Table

- Similar to a transition table except the states are represented by **letter symbols**
- Can also include the output values
- Suitable to obtain the logic diagram from it
- Primitive flow table:
  - **only one stable** state in each row

Equivalent to 9-3(c) if
a=00, b=01, c=11, d=10



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

Flow Table to Circuits

- Procedure to obtain circuits from flow table:
  - Assign to each state a distinct binary value (convert to a transition table)
  - Obtain circuits from the map

- Two difficulties:
  - The binary state assignment (to avoid race)
  - The output assigned to the unstable states

Flow Table to Circuits

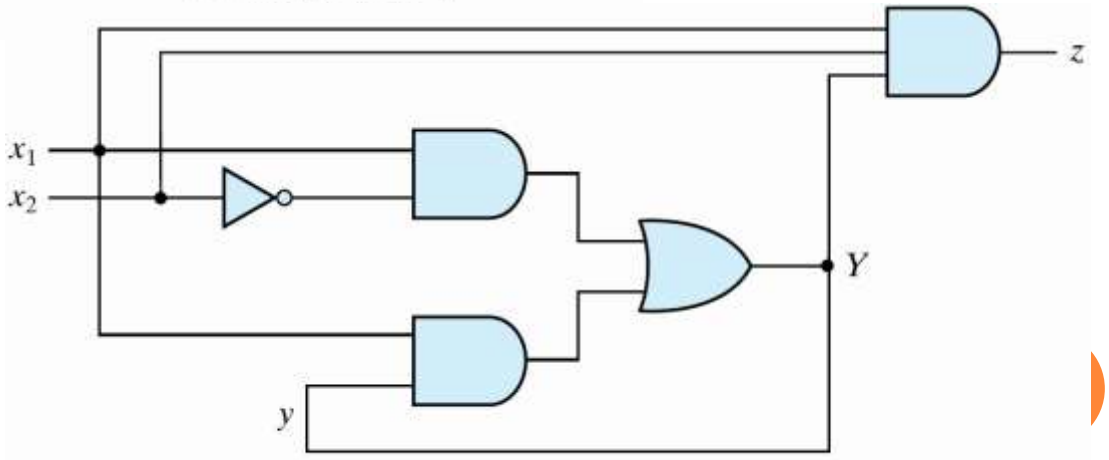
y \ x_1x_2	00	01	11	10
0	0	0	0	1
1	0	0	1	1

(a) Transition table

$Y = x_1x'_2 + x_1y$

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

(b) Map for output



(c) Logic diagram

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

Race Conditions

- 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
  - $00 \rightarrow 11$
    - $00 \rightarrow 10 \rightarrow 11$ or $00 \rightarrow 01 \rightarrow 11$
- 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**
  - Should be avoided !!

Noncritical Race: Examples

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

(a) Possible transitions:

$00 \rightarrow 11$

$00 \rightarrow 01 \rightarrow 11$

$00 \rightarrow 10 \rightarrow 11$

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

(b) Possible transitions:

$00 \rightarrow 11 \rightarrow 01$

$00 \rightarrow 01$

$00 \rightarrow 10 \rightarrow 11 \rightarrow 01$

Critical Race: Examples

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

(a) Possible transitions:

$00 \rightarrow 11$
$00 \rightarrow 01$
$00 \rightarrow 10$

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

(b) Possible transitions:

$00 \rightarrow 11$
$00 \rightarrow 01 \rightarrow 11$
$00 \rightarrow 10$

Race Conditions

- Race can be avoided by proper state assignment
  - Direct the circuit through intermediate unstable states with a unique state-variable change
  - It is said to have a **cycle**
- Must ensure that a cycle will terminate with a stable state
  - Otherwise, the circuit will keep going in unstable states

Cycles

- A cycle
  - a unique sequence of unstable states

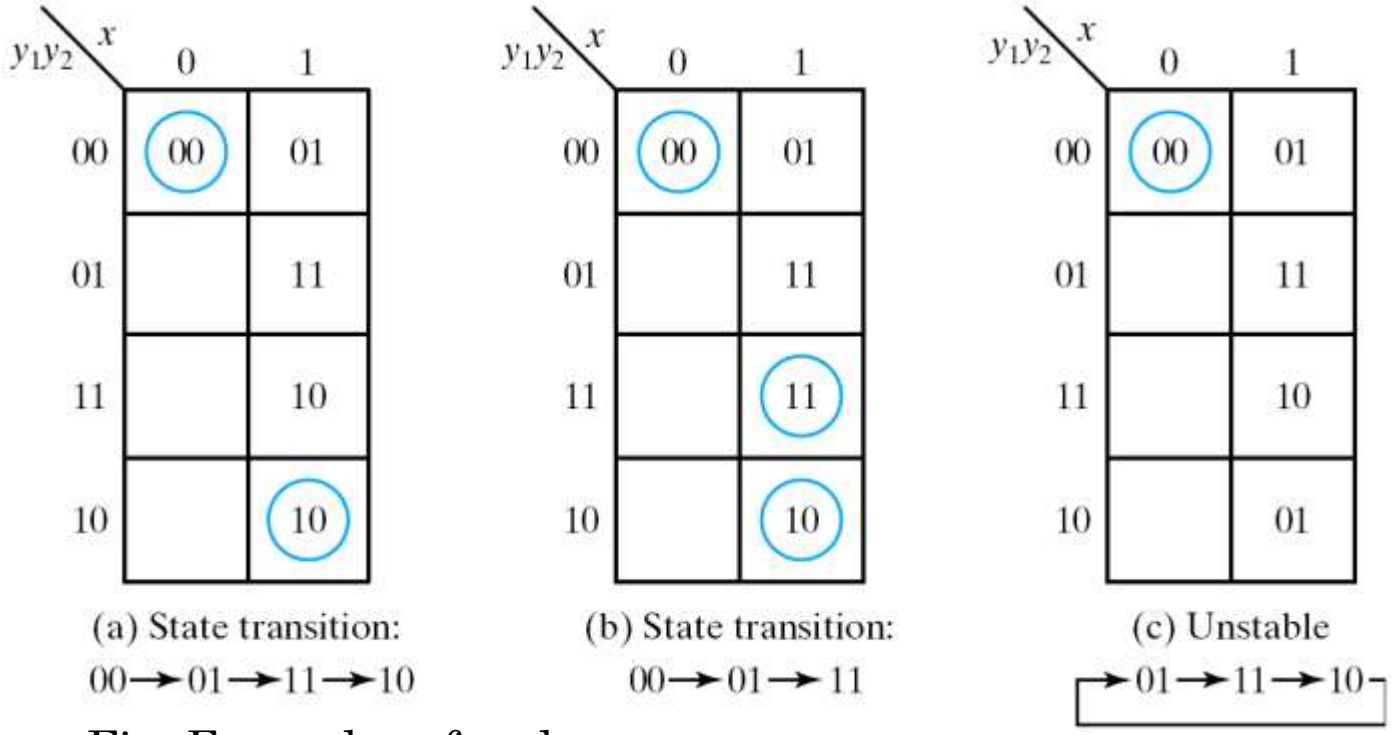


Fig: Examples of cycles

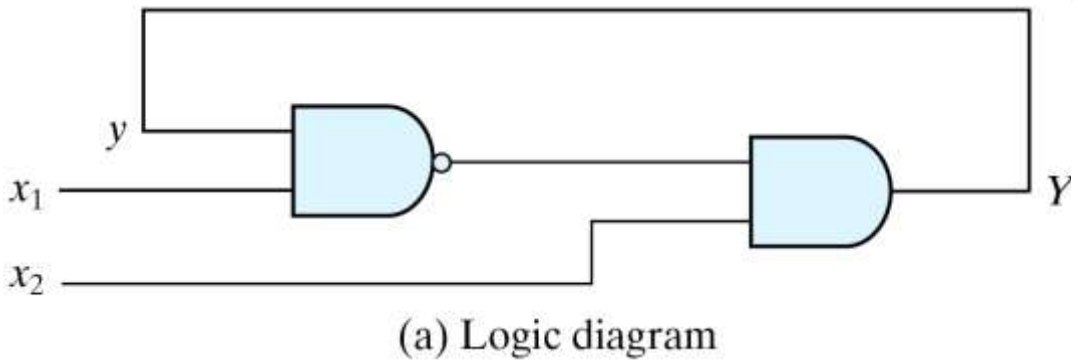
Stability Check

- Asynchronous sequential circuits may oscillate between unstable states due to the feedback
  - Must check for stability to ensure proper operations

- Can be easily checked from the transition table
  - Any column has no stable states → unstable

Example of an Unstable Circuit

- When x_1x_2=11 in Fig. 9-9(b), Y and y are never the same
- a square waveform generator?



(a) Logic diagram

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

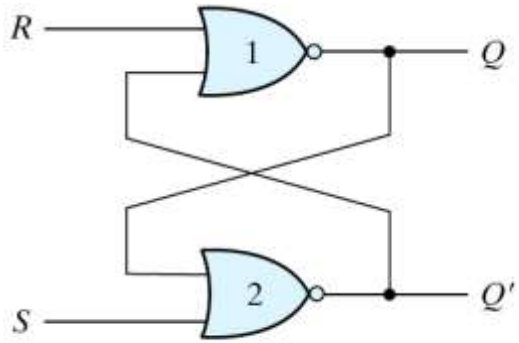
(b) Transition table

Fig. 9.9: Example of an unstable circuit

Circuits with Latches

- 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 also an asynchronous circuit
  - Will be analyzed first using the method for asynchronous circuits

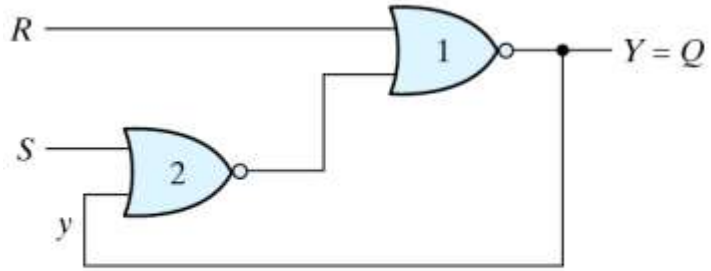
SR Latch with NOR Gates



(a) Cross-coupled circuit

S	R	Q	Q′	
1	0	1	0	
0	0	1	0	(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	00	01	11	10
0	0	0	0	1
1	1	0	0	1

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

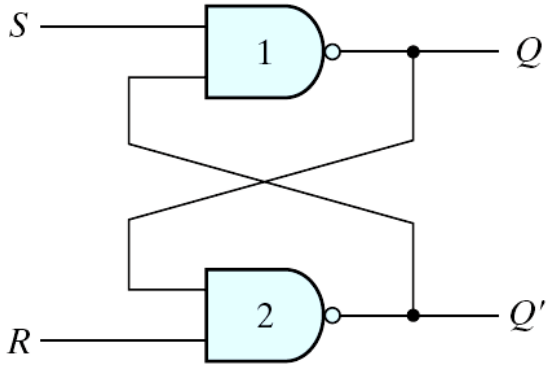
(d) Transition table

Fig. 9.10: *SR* latch with NOR gates

SR Latch with NOR Gates

- $Y = ((S+y)'+R)' = (S+y)R' = SR'+R'y$
- An unpredictable result when SR: $11 \rightarrow 00$
- $SR = 0$ in operation
- $Y = S + R'y$ when $SR = 0$

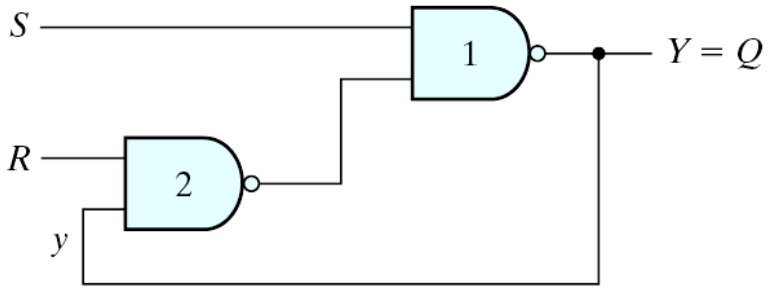
SR Latch with NAND Gates



(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

SR Latch with NAND Gates

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

Analysis Procedure

- Procedure to analyze an asynchronous sequential circuits with SR latches:
  1. Label each latch output with Y_i and its external feedback path (if any) with y_i
  2. Derive the Boolean functions for each S_i and R_i
  3. Check whether $SR=0$ (NOR latch) or $S'R'=0$ (NAND latch) is satisfied
  4. Evaluate $Y=S+R'y$ (NOR latch) or $Y=S'+Ry$(NAND latch)
  5. Construct the transition table for $Y=Y_1Y_2\ldots Y_k$
  6. Circle all stable states where $Y=y$

Analysis Example

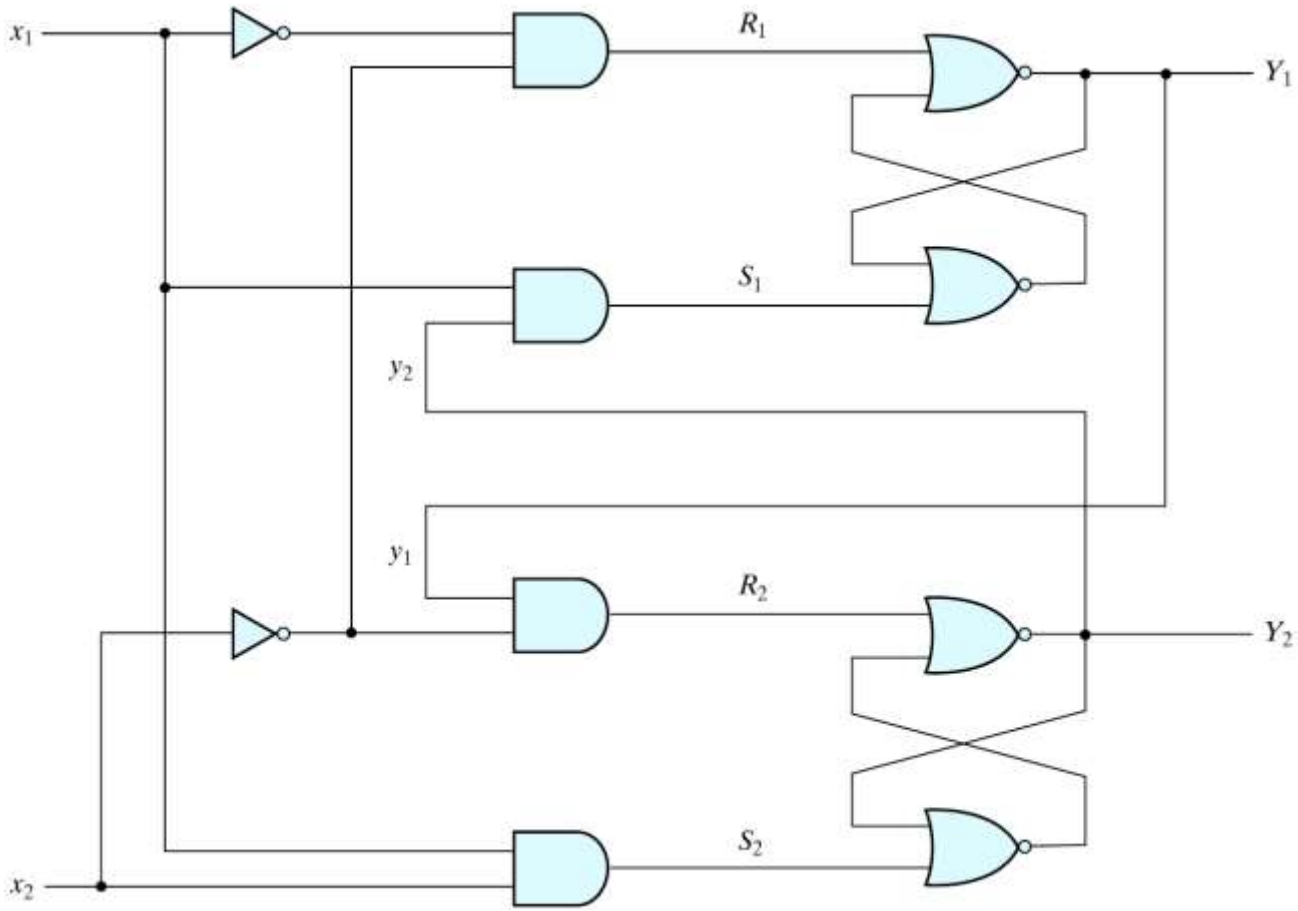


Fig. 9.12: Examples of a circuit with *SR* latch

Analysis Example

- 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

- 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

Analysis Example

- 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.

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 Procedure

- Procedure to implement an asynchronous sequential circuits with SR latches:
  1. Given a transition table that specifies the excitation function $Y = Y_1 Y_2 \ldots Y_k$, derive a pair of maps for each S_i and R_i using the latch excitation table
  2. 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)
  3. 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

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

$S = x_1x_2'$ and $R = x_1'$ → NOR latch

$S = (x_1x_2')'$ and $R = x_1$ → NAND latch

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

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

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

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

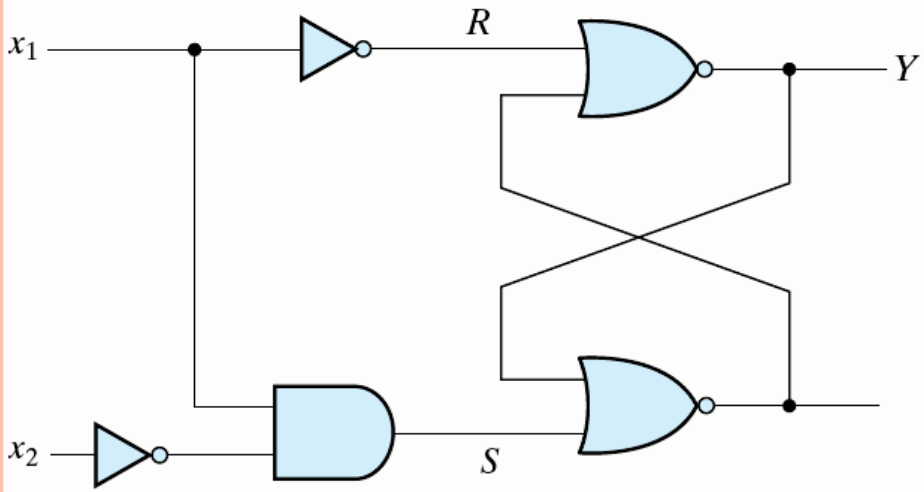
(b) Latch excitation table

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

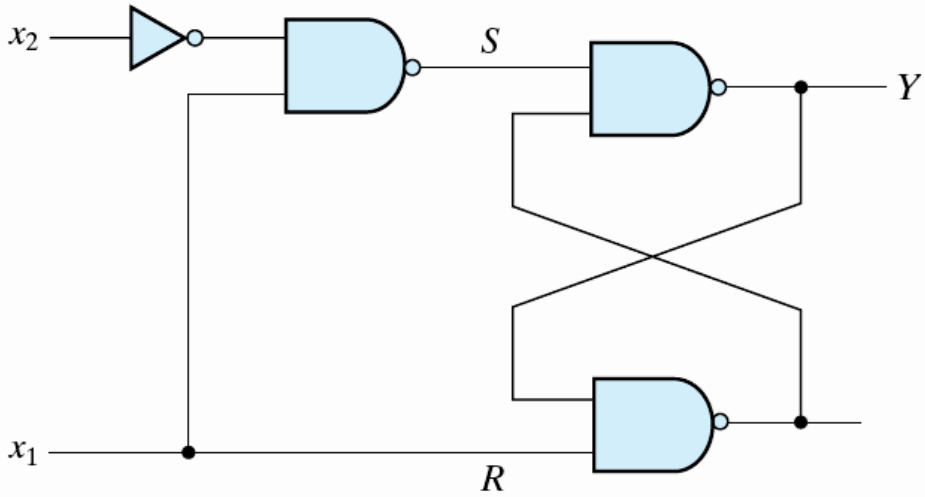
(c) Map for $S = x_1x'_2$

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

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



(e) Circuit with NOR latch



(f) Circuit with NAND latch

Debounce Circuit

- Mechanical switches are often used to generate binary signals to a digital circuit
  - It may vibrate or bounce several times before going to a final rest
  - Cause the signal to oscillate between 1 and 0

Debounce Circuit

- A debounce circuit can remove the series of pulses from a contact bounce and produce a single smooth transition
  - Position A (SR=01) → bouncing (SR=11)→ Position B (SR=10)
  - Q = 1 (set) → Q = 1 (no change) → Q = 0 (reset)

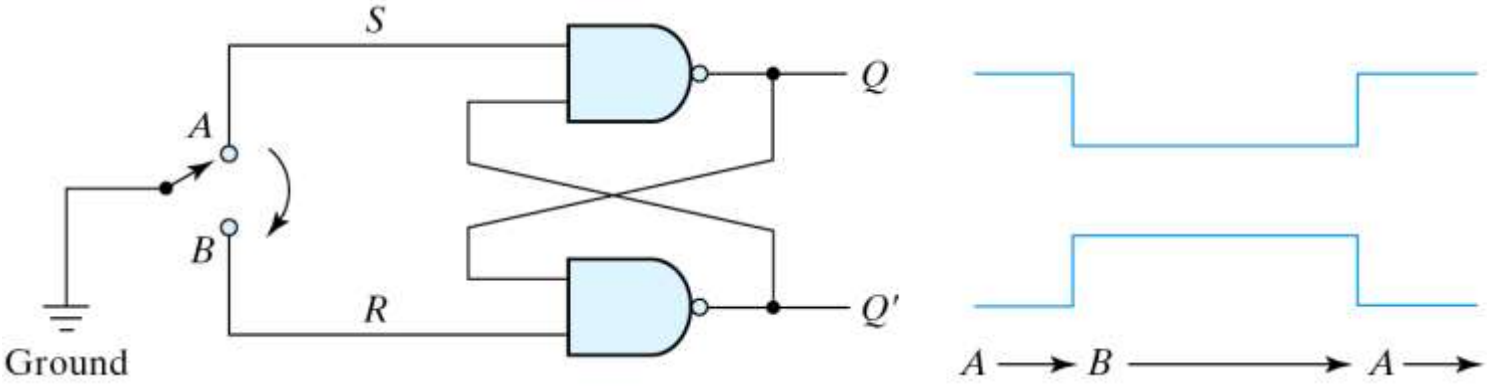


Fig. 9.15: Debounce circuit

Design Procedure

- Design specifications

  (i) Obtain a primitive table from specifications

  (ii) Reduce flow table by merging rows in the primitive flow table

  (iii) Assign binary state variables to each row of reduced table

  (iv) Assign output values to dashes associated with unstable states to obtain the output map

  (v) Simplify Boolean functions for excitation and output variables;

  (vi) Draw the logic diagram

Design Procedure: Example

- 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

- Assume fundamental mode operation – only one input changes at a time.

Design Procedure: Example

- Form a table with all possible total 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*

Primitive Flow Table

- Obtain the flow table by listing all possible states
  - Dash marks are given when both inputs change simultaneously
  - Outputs of unstable states are don't care

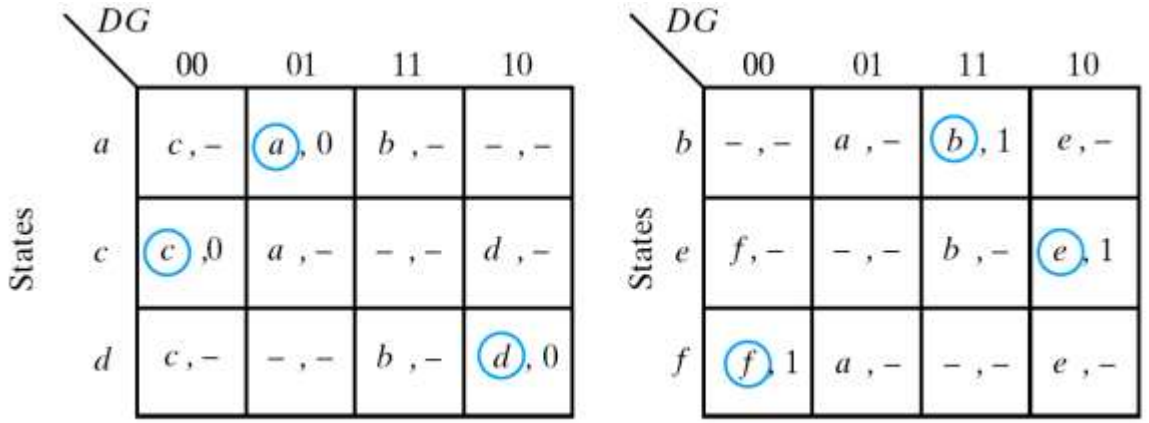
States	Inputs DG 00	01	11	10
a	c, –	(*a*), 0	b, –	–, –
b	–, –	a, –	(*b*), 1	e, –
c	(*c*), 0	a, –	–, –	d, –
d	c, –	–, –	b, –	(*d*), 0
e	f, –	–, –	b, –	(*e*), 1
f	(*f*), 1	a, –	–, –	e, –

Fig. 9.16
Primitive flow table

Reduce the Flow Table

- 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

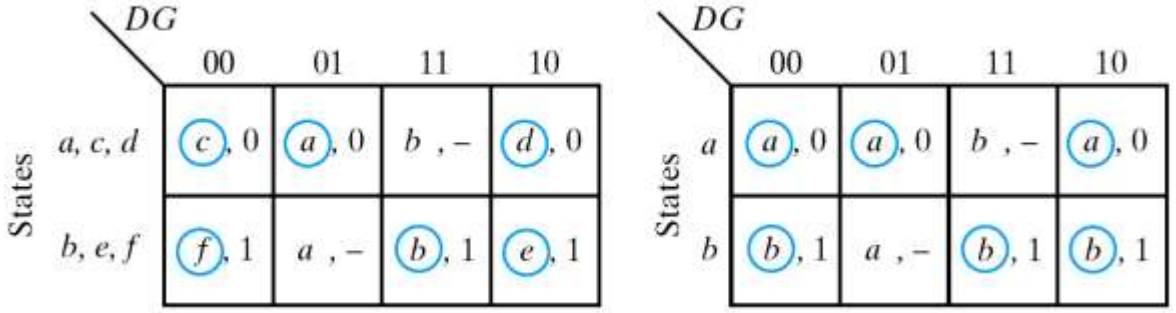
Reduce the Flow Table



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



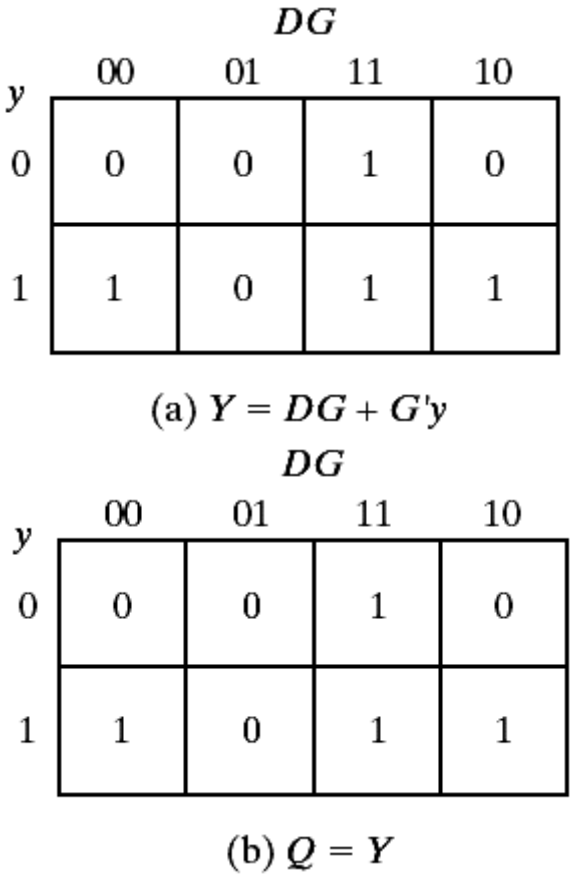
States	DG 00	01	11	10
a, c, d	(*c*), 0	(*a*), 0	b, –	(*d*), 0
b, e, f	(*f*), 1	a, –	(*b*), 1	(*e*), 1

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

Transition Table and Output Map



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	1	0
1	1	0	1	1

(b) $Q = Y$

Fig. 9.18: Transition table and output map for gated latch

- Assign a binary value to each state to generate the transition table
  - a=0, b=1 in this example

Logic Diagram

- Directly use the simplified Boolean function for the excitation variable Y
  - An asynchronous circuit without latch is produced

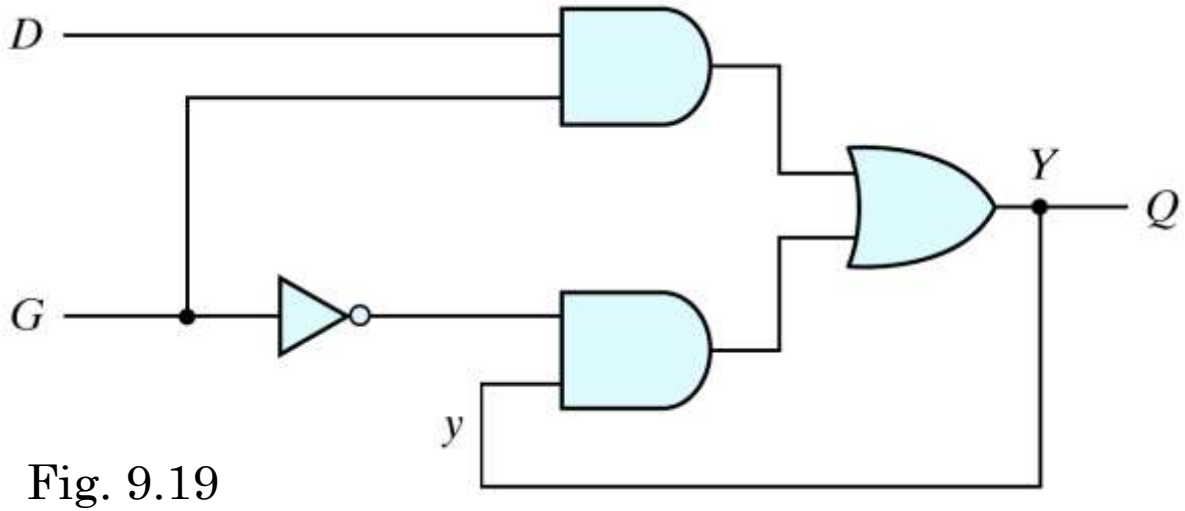


Fig. 9.19
Gated-latch logic diagram

Design with *SR* Latches

- We can also implement asynchronous circuits using latches at the outputs.
- Given the map for each excitation variable Y, derive necessary equations for S and R of a latch to produce Y.
- Derive Boolean equations for S and R.

DESIGN WITH *SR* LATCHES

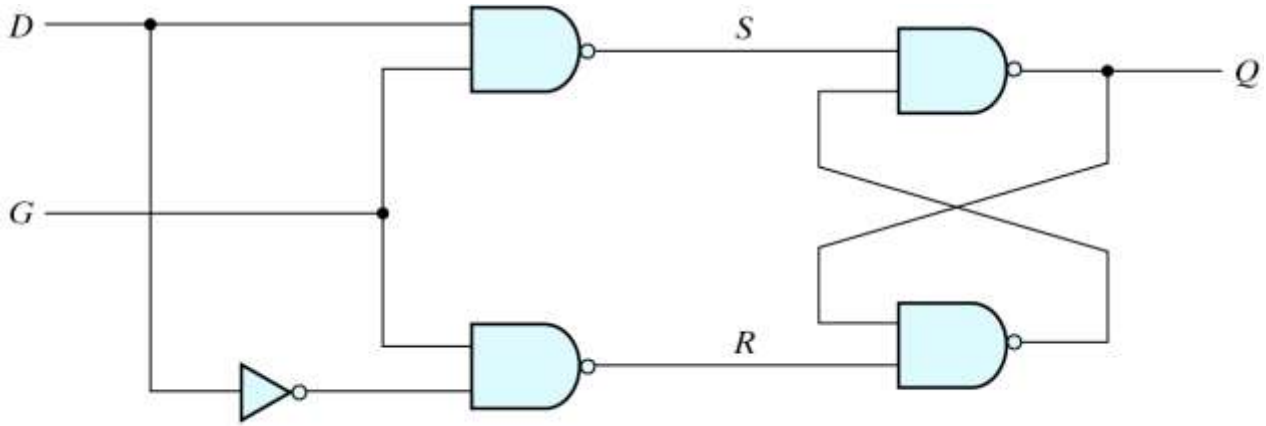
y \ *DG*	00	01	11	10
0	0	0	1	0
1	X	0	X	X

(a) $S = DG$

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

$R = D'G$

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



(b) Logic diagram

Fig. 9.20
Circuit with *SR* latch

Outputs for Unstable States

- Objective: no momentary false outputs occur when the circuit switches between stable states
- If the output value is not changed, the intermediate unstable state must have the same output value
  - 0 → 1 (unstable) → 0 (X)
- If the output value changed, the intermediate outputs are don't care
- It makes no difference when the output change occurs

a	(*a*), 0	*b* , –
b	*c* , –	(*b*), 0
c	(*c*), 1	*d* , –
d	*a* , –	(*d*), 1

(a) Flow table

0	0
X	0
1	1
X	1

(b) Output assignment

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

Reduction of State and Flow Table

- State reduction procedure is similar in both sync. & async. sequential circuits
- For completely specified state tables:
  - use implication table
- For incompletely specified state tables:
  - use compatible pairs states

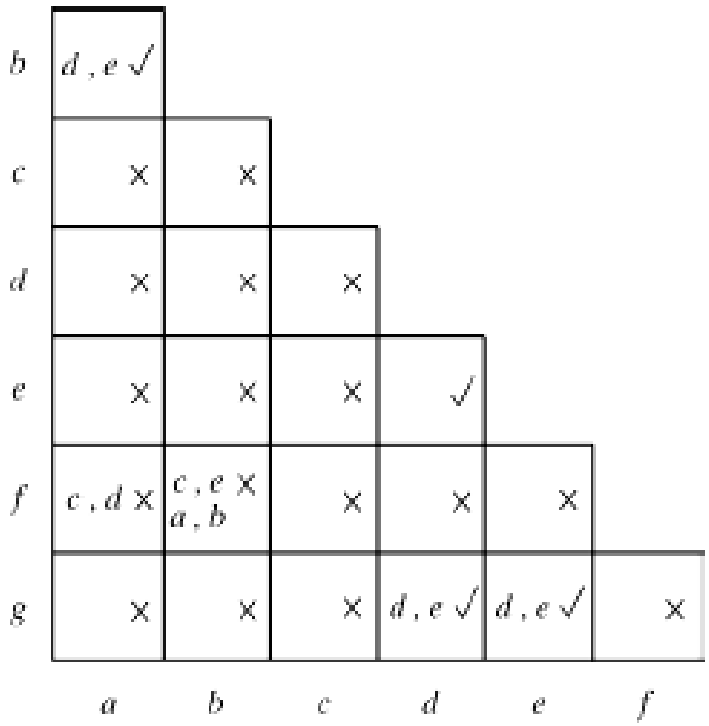
Reduction of State and Flow Table

- Two states are equivalent if they have the **same output** and go to the **same (equivalent) next states for each** possible input
- Example:
  - (a,b) are equivalent if (c,d) are equivalent
  - (a,b) imply (c,d)
  - (c,d) imply (a,b)

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

State Reduction:Implication Table



Present State	Next State		Output	
	x = 0	x = 1	x = 0	x = 1
a	*d*	6	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

State Reduction: Implication Table

- Finally, all the squares that have no crosses are recorded with check marks. The equivalent states are: (a, b), (d, e), (d, g), (e, g).

- We now combine pairs of states into larger groups of equivalent states. The last three pairs can be combined into a set of three equivalent states (d, e, g) because each one of the states in the group is equivalent to the other two.

State Reduction: Implication Table

- The final partition of these states consists of the equivalent states found from the implication table, together with all the remaining states in the state table that are not equivalent to any other state: (*a*, *b*) (*c*) (*d*, *e*, *g*) (*f*)

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

Merging of the Flow Table

- The state table may be incompletely specified
  - 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"
  - Instead, we are going to find "**compatible" states**
  - Two states are compatible if they have the **same output** and **compatible next states whenever specified**
- Three procedural steps:
  - Determine all compatible pairs
  - Find the maximal compatibles
  - Find a minimal closed collection of compatibles

Compatible Pairs

- 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

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

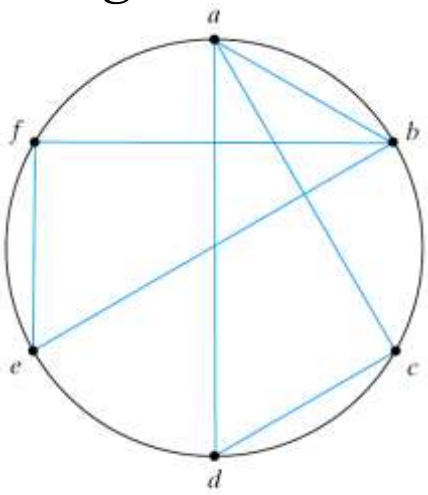
(b) Implication table

Maximal Compatibles

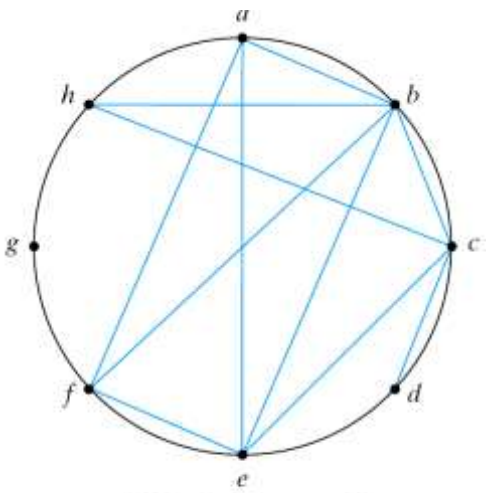
- A group of compatibles that contains all the possible combinations of compatible states
  - Obtained from a merger diagram
  - A line in the diagram represents that two states are compatible
- *n*-state compatible → n-sided fully connected polygon
  - All its diagonals connected
- Not all maximal compatibles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

Maximal Compatibles

- 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)

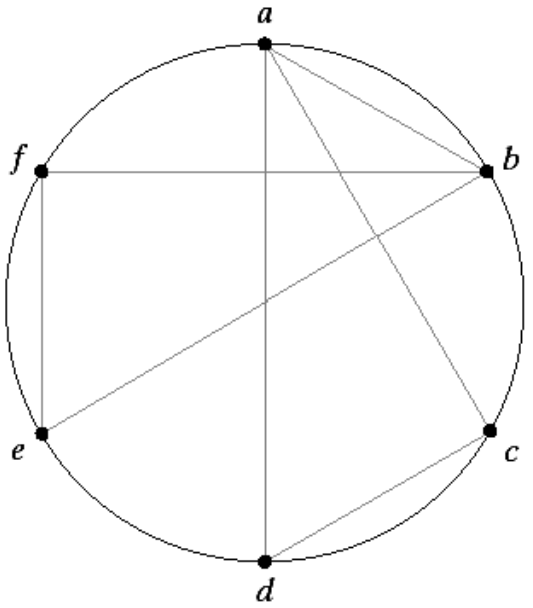


(b) Maximal compatible:
(a, b, e, f) (b, c, h) (c, d) (g)

Fig. 9.24
Merger diagram

Closed Covering Condition

- The set of chosen compatibles must cover all the states and must be closed
- The closure condition is satisfied if
  - no implied states or the implied states are included within the set
- Ex: if remove (a,b) in the right
  - (a,c,d) (b,e,f) are left in the set
  - All six states are still included
  - No implied states according to its implication table

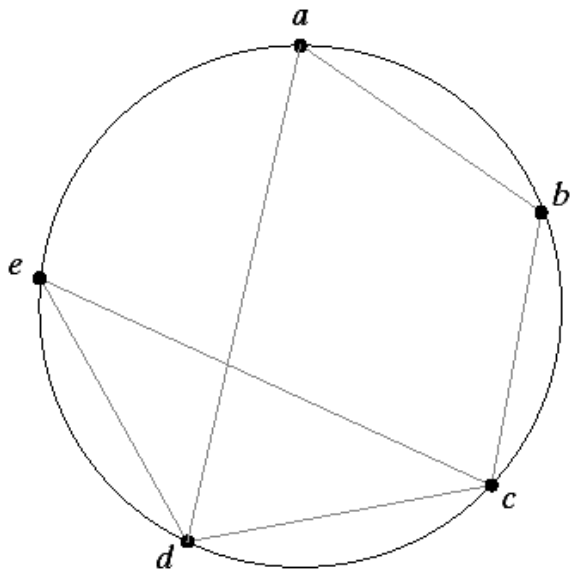


(a) Maximal compatible:
$(a, b,)\ (a, c, d)\ (b, e, f)$

Closed Covering Condition

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

(a) Implication table



(b) Merger diagram

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

*(a,b) (c,d,e) → (X) implied (b,c) is not included in the set

*better choice: (a,d) (b,c) (c,d,e) all implied states are included

Race-Free State Assignment

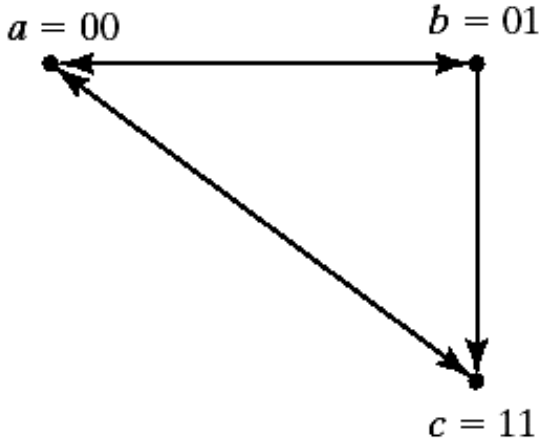
- Objective: choose a proper binary state assignment to **prevent critical races**
- Only one variable can change at any given time when a state transition occurs
- States between which transitions occur will be given **adjacent assignments**
  - Two binary values are said to be adjacent if they differ 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
- Outputs are omitted for simplicity
- Adjacent info. are represented by a transition diagram
- a and c are still 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

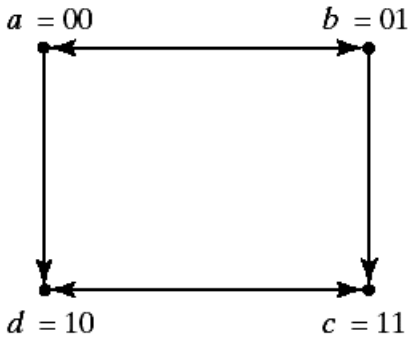


(b) Transition diagram

3-Row Flow Table Example

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

	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	–



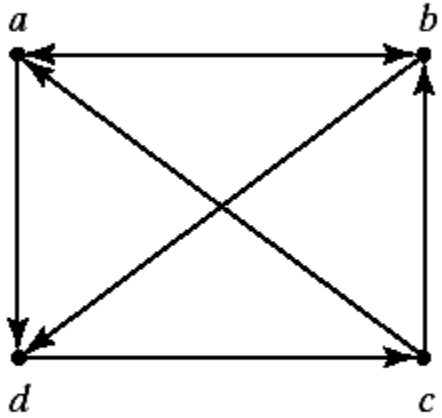
	x_1x_2 = 00	01	11	10
a = 00	(00)	01	10	(00)
b = 01	00	(01)	(01)	11
c = 11	10	(11)	(11)	(11)
d = 10	00	–	11	–

4-Row Flow Table Example

- Sometimes, just one extra row may not be sufficient to prevent critical races
  - More binary state variables may also required
- With one or two diagonal transitions, there is no way of using two binary variables that satisfy all adjacency

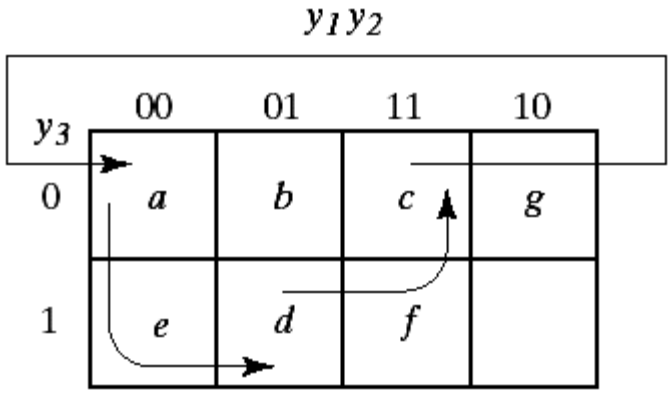
	00	01	11	10
a	*b*	(*a*)	*d*	(*a*)
b	(*b*)	*d*	(*b*)	*a*
c	(*c*)	*a*	*b*	(*c*)
d	*c*	(*d*)	(*d*)	*c*

(a) Flow table

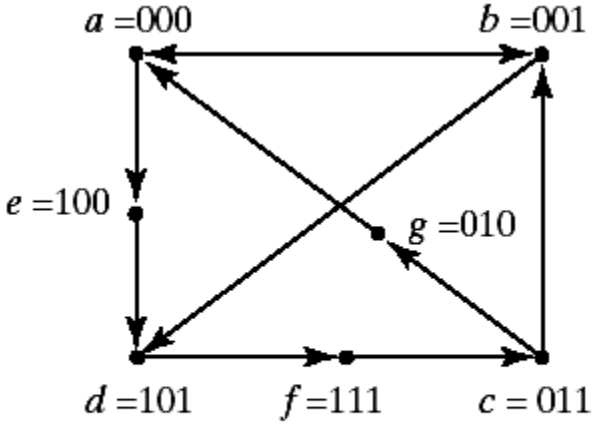


(b) Transition diagram

4-Row Flow Table Example



(a) Binary assignment



(b) Transition diagram

	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*	–

Multiple-Row Method

- Multiple-row method is easier
  - May not as efficient as in above **shared-row method**
- Each stable state is duplicated with exactly the same output
  - Behaviours are still the same
- While choosing the next states, choose the adjacent one

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

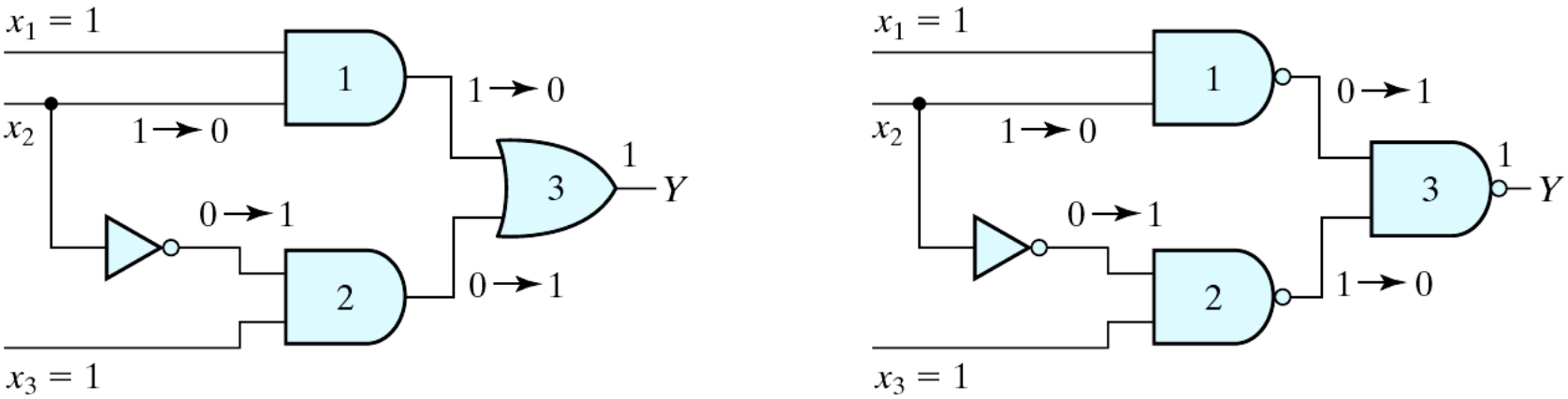
	00	01	11	10
$000 = a_1$	b_1	(a_1)	d_1	(a_1)
$111 = a_2$	b_2	(a_2)	d_2	(a_2)
$001 = b_1$	(b_1)	d_2	(b_1)	a_1
$110 = b_2$	(b_2)	d_1	(b_2)	a_2
$011 = c_1$	(c_1)	a_2	b_1	(c_1)
$100 = c_2$	(c_2)	a_1	b_2	(c_2)
$010 = d_1$	c_1	(d_1)	(d_1)	c_1
$101 = d_2$	c_2	(d_2)	(d_2)	c_2

(b) Flow table

Hazards

- Unwanted switching appears at the output of a circuit
  - Due to different propagation delay in different paths
- May cause the circuit to mal-function
  - Cause temporary false-output values in combinational circuits
  - Cause a transition to a wrong state in asynchronous circuits
  - Not a concern to synchronous sequential circuits

Hazards in Combinational Circuits



(a) AND–OR circuit

(b) NAND circuit

Fig. 9.33
Circuits with hazards

Types of Hazards

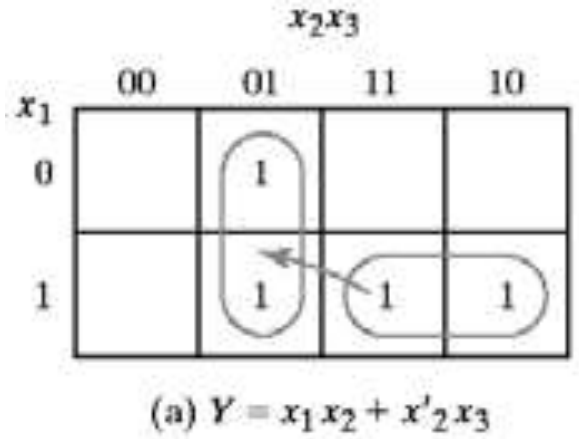
- Three types of hazards:

Fig. 9.34
Types of hazards

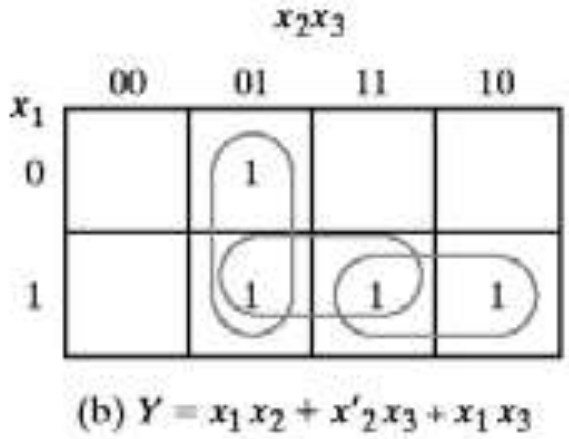
Hazard Free Circuit

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

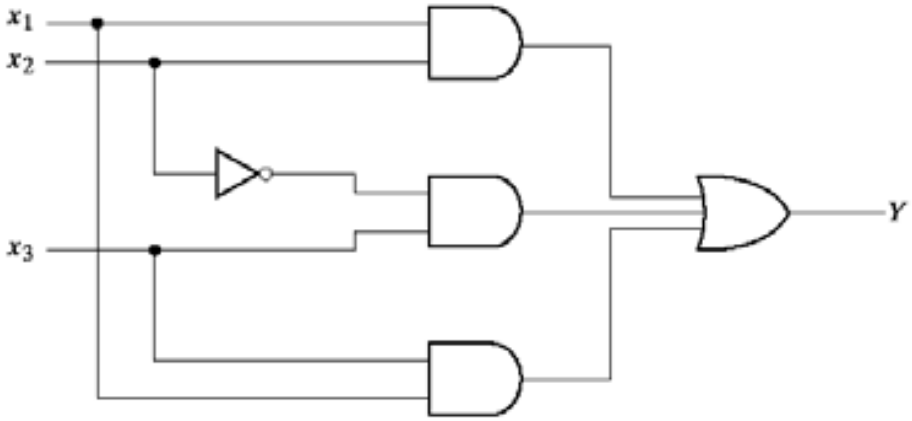
Hazard Free Circuit



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

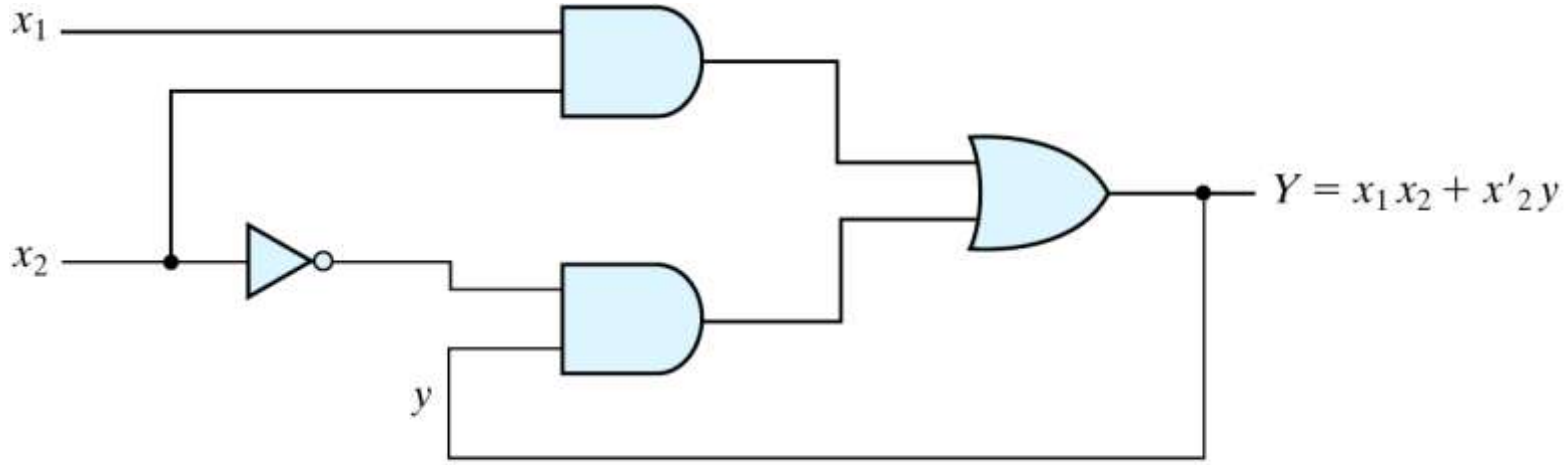


(b) $Y = x_1x_2 + x'_2x_3 + x_1x_3$



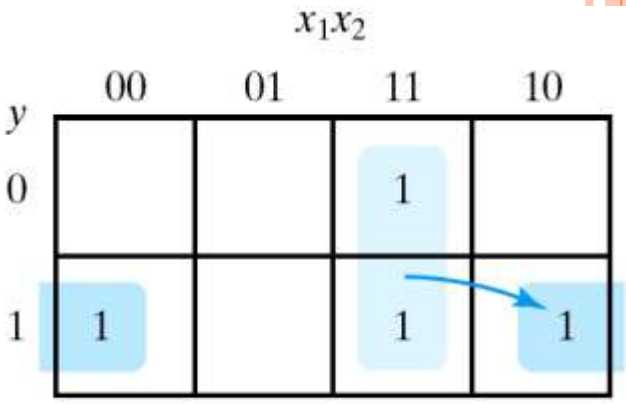
Hazard in a Asynchronous Circuit

- In general, no problem for synchronous design



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

(b) Transition table

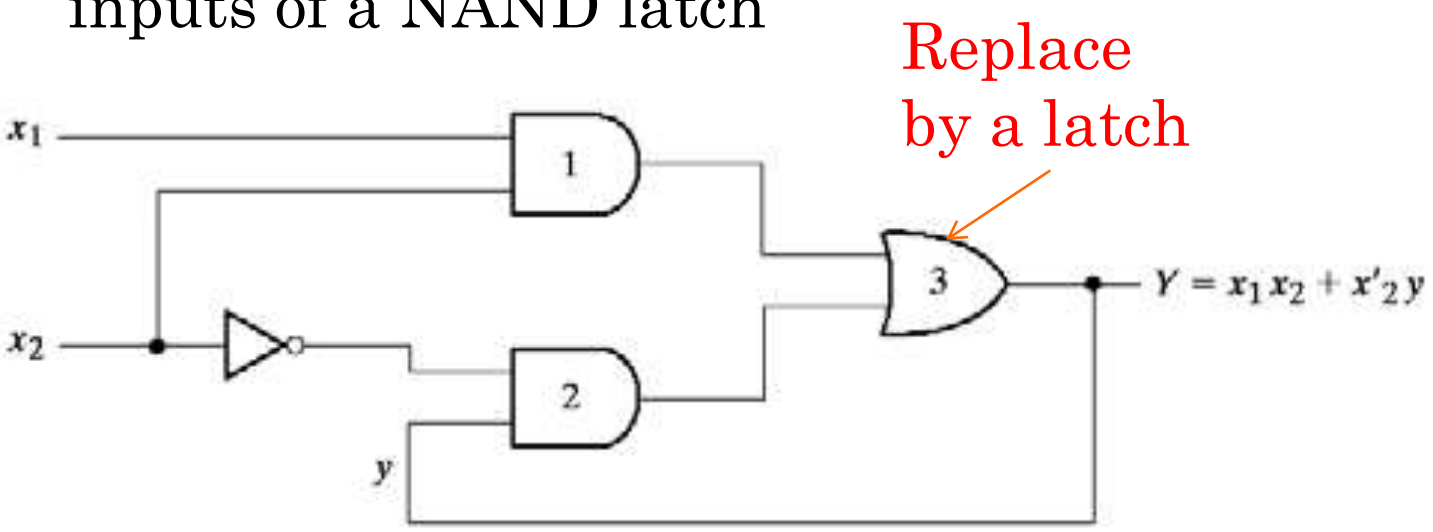


(c) Map for Y

Fig. 9.37
Hazard in a asynchronous sequential circuit

Remove Hazard with Latches

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

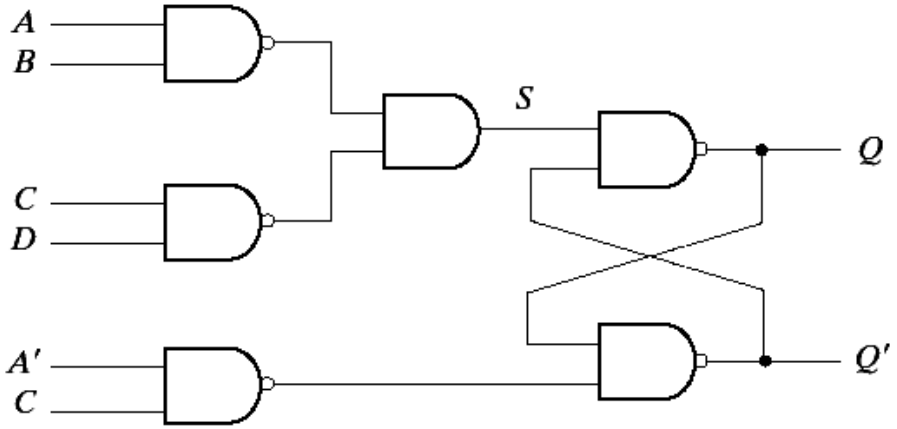


(a) Logic diagram

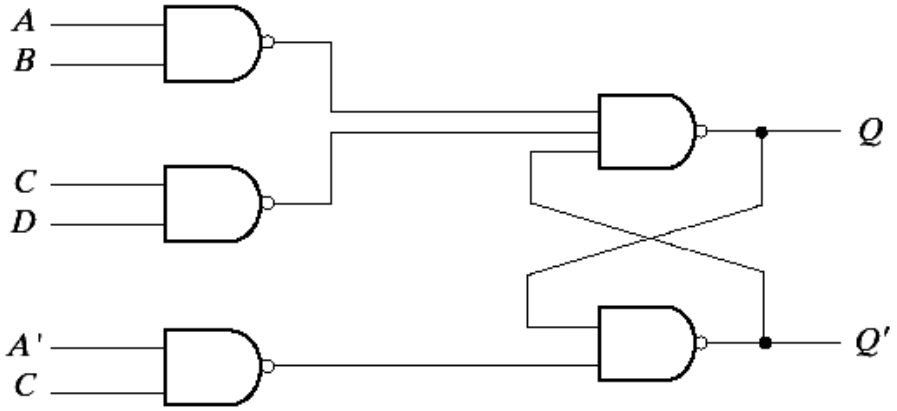
Implementation with SR latches

- Given:
  - $S = AB + CD$
  - $R = A'C$
- For NAND latch, use complemented inputs
  - $S' = (AB + CD)' = (AB)'(CD)'$
  - $R' = (A'C)'$
- $Q = (Q'S)' = [Q'(AB)'(CD)']'$
- Two-level circuits (this is the output we want)

Implementation with SR latches



(a)



(b)

Essential Hazards

- Besides static and dynamic hazards, another type of hazard in asynchronous circuits is called **essential hazard**
- 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 !!

Summary of Design Procedure

1. State the design specifications
2. Derive a primitive flow table
3. Reduce the flow table by merging the 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 Specification

- Design a negative-edge-triggered T flip-flop
  - Two inputs: T(toggle) and C(clock)
  - T=1: toggle, T=0: no change
  - One output: Q

Primitive Flow Table

Table 9.6
Specification of Total States

State	Inputs T	Inputs C	Output 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*

	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* , –	– , –

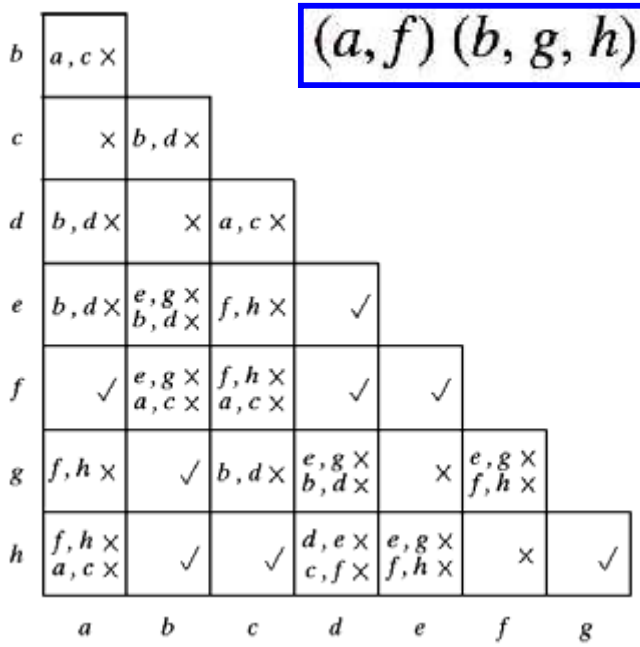
Merging of the Flow Table

- Compatible pairs:

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

- Maximal compatible set:

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



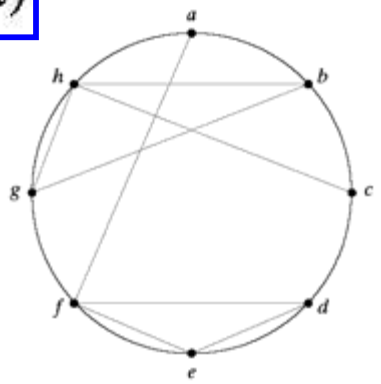
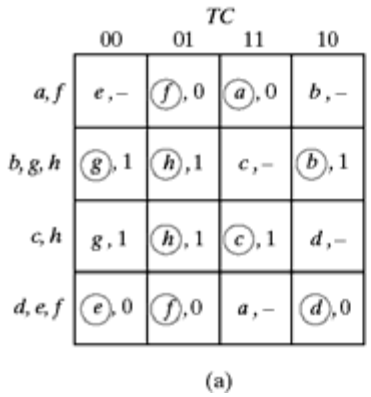
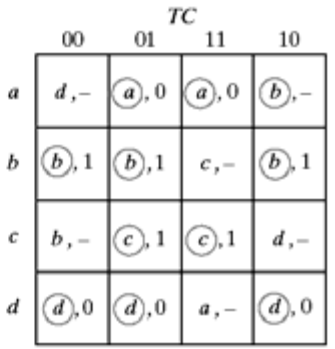


Fig. 9-41 Merger Diagram



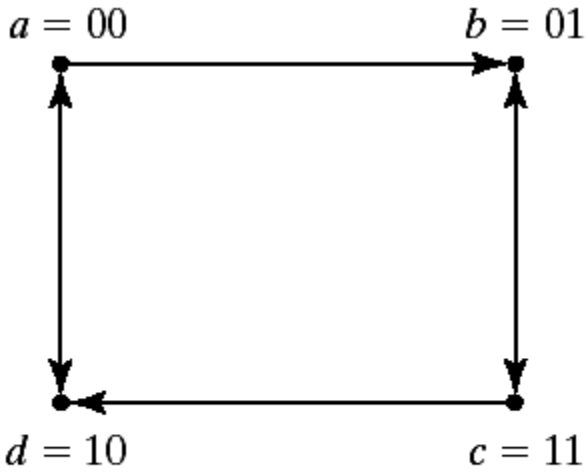
(a)



(b)

State Assignment and Transition Table

- No diagonal lines in the transition diagram
  - No need to add extra states



TC

$y_1 y_2$	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)

(a) Transition table

TC

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

(b) Output map $Q = y_2$

Fig. 9-43 Transition Diagram

Logic Diagram

y_1y_2 \ TC	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) $S_1 = y_2\,TC + y'_2\,T'C'$

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

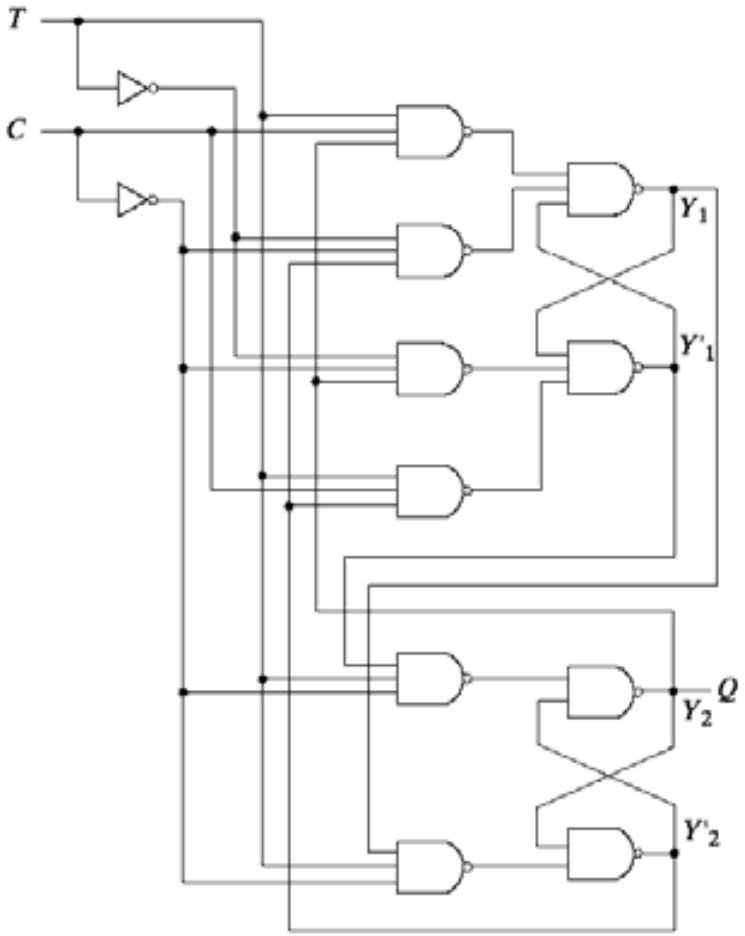
(b) $R_1 = y_2\,T'C' + y'_2\,TC$

y_1y_2 \ TC	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) $S_2 = y'_1\,TC'$

y_1y_2 \ TC	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) $R_2 = y_1\,TC'$



Syllabus

- Chapter 9