CS8351 DIGITAL PRINCIPLES AND SYSTEM DESIGN

UNIT IV

ASYNCHRONOUS SEQUENTIAL LOGIC

Prof G ELANGOVAN
Professor and Head
Department of Electrical and Electronics Engineering
NPR College of Engineering and Technology
Natham, Dindigul Dist. 624 401

gurugovan@yahoo.com

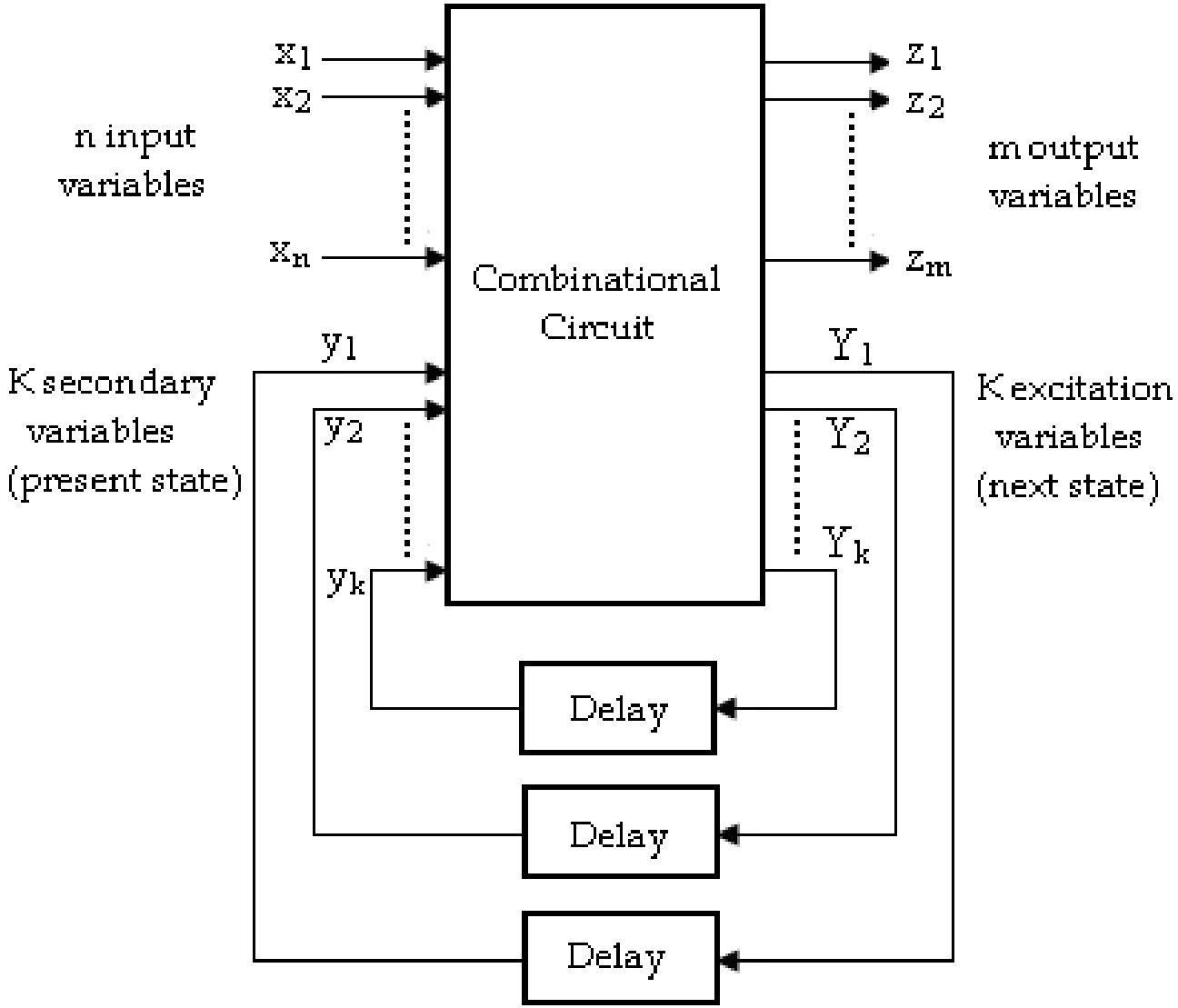
UNIT IV
ASYNCHRONOUS SEQUENTIAL LOGIC

- Analysis of Asynchronous Sequential Circuits
- Design of Asynchronous Sequential Circuits
- Reduction of State and Flow Tables
- Race-free State Assignment
- Hazards

Introduction

- A sequential circuit is specified by a time sequence of inputs, outputs and internal states. The output changes whenever a clock pulse is applied. The memory elements are clocked flip-flops.
- Asynchronous sequential circuits do not use clock pulses. The memory elements in asynchronous sequential circuits are either unclocked flip-flops (Latches) or time-delay elements.

S No	Synchronous sequential circuits	Asynchronous sequential circuits
1	Memory elements are clocked flip-flops	Memory elements are either unclocked flip-flops or time delay elements.
2	The change in input signals can affect memory element upon activation of clock signal.	The change in input signals can affect memory element at any instant of time.
3	The maximum operating speed of clock depends on time delays involved. Therefore synchronous circuits can operate slower than asynchronous.	Because of the absence of clock, it can operate faster than synchronous circuits.
4	Easier to design	More difficult to design



Block diagram of Asynchronous sequential circuits

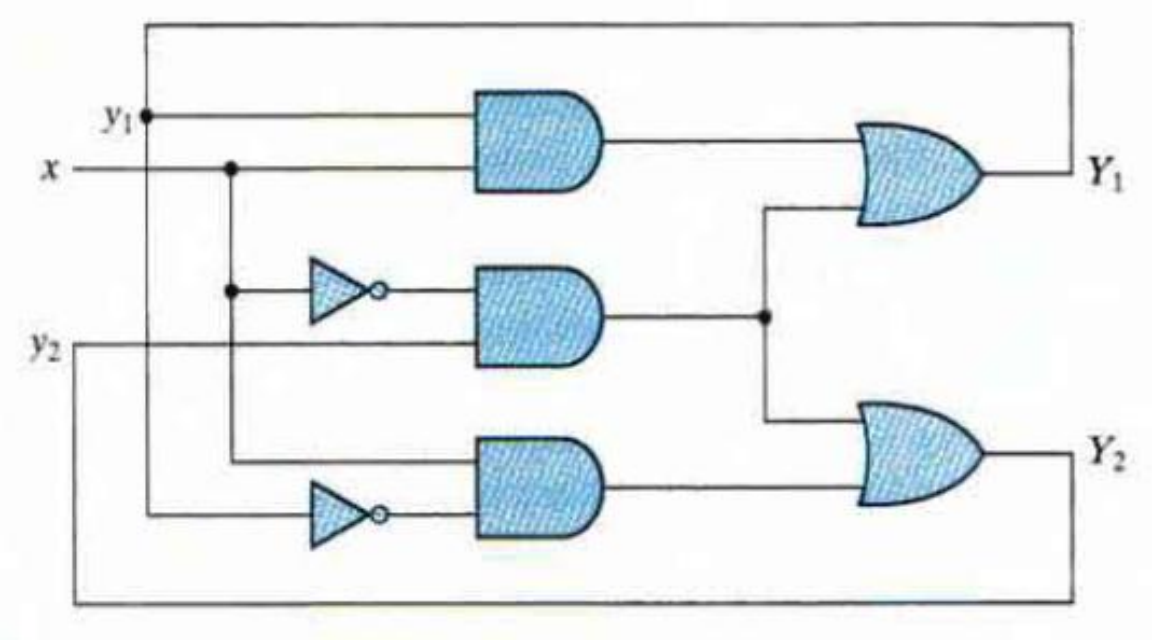
According to input variables there are two types

- **Fundamental mode circuit**
  - The input variables change only when the circuit is stable. Only one input variable can change at a given time.
  - Inputs are levels (0, 1) and not pulses.
- **Pulse mode circuit**
  - The input variables are pulses (True, False) instead of levels.
  - The width of the pulses is long enough for the circuit to respond to the input.
  - The pulse width must not be so long that it is still present after the new state is reached.

Analysis of Asynchronous Sequential Circuits

The analysis of asynchronous sequential circuits 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.

Analysis Procedure

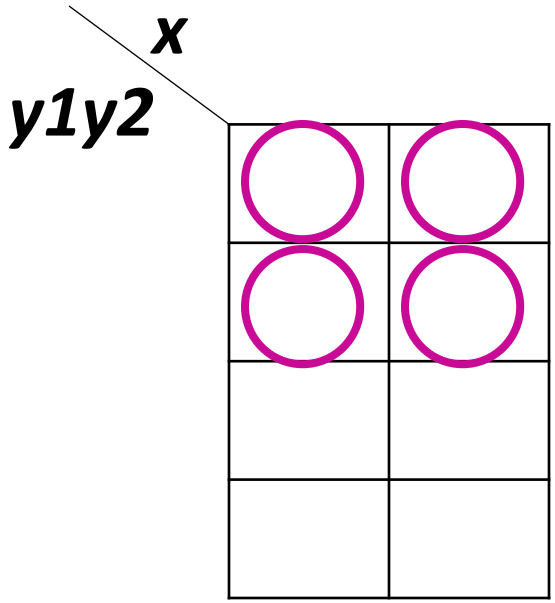


$$Y_1 = xy_1 + x'y_2$$

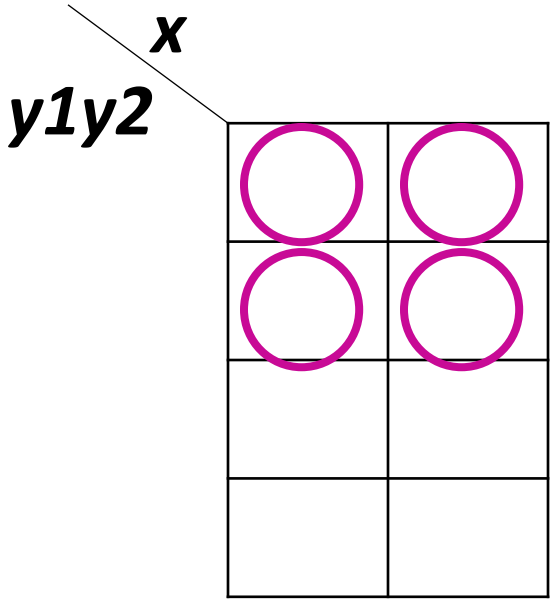
$$Y_2 = xy_1' + x'y_2$$

Example of an asynchronous sequential circuit

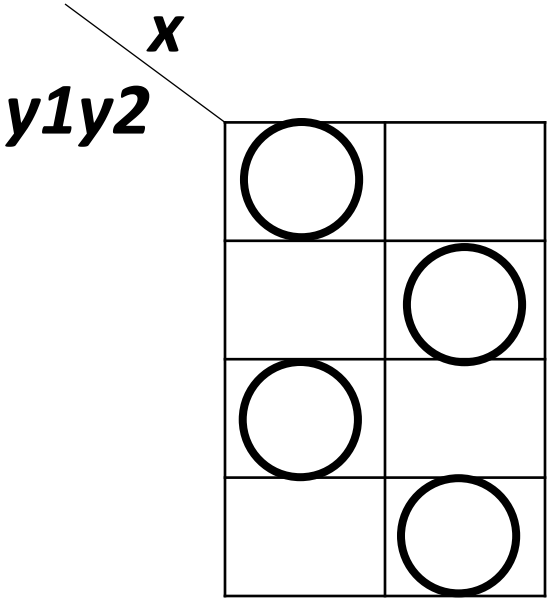
Transition Table



Map for
$Y_1 = xy_1 + x'y_2$



Map for
$Y_2 = xy'_1 + x'y_2$



Transition table

Total State

Four stable total states – ***y1y2x*** = 000, 011, 110, and 101

Four unstable total states – 001, 010, 111, and 100

The transition table of asynchronous sequential circuit is similar to the state table used for synchronous circuits

y1y2 \ x	0	1
00	(00)	01
01	11	(01)
11	(11)	10
10	00	(10)

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

The procedure for obtaining a transition table from the given circuit diagram is as follows.

1. Determine all feedback loops in the circuit.
2. Designate the output of each feedback loop with variable Y1 and its corresponding inputs *y1, y2,….yk*, where *k* is the number of feedback loops in the circuit.
3. Derive the Boolean functions of all Y's as a function of the external inputs and the y's.
4. Plot each *Y* function in a map, using *y* variables for the rows and the external inputs for the columns.
5. Combine all the maps into one table showing the value of ***Y= Y1, Y2,….Yk*** inside each square.
6. Circle all stable states where ***Y=y***. The resulting map is the transition table.

Once the transition table is available, the behavior of the circuit can be analyzed by observing the stale transition as a function of changes in the input variables.

Flow Table

- During the design of asynchronous sequential circuits, it is more convenient to name the states by letter symbols than binary values.
- Such a table is called *an flow table* and is similar to a transition table, except that the internal states are symbolized with letters rather than binary numbers.
- The flow table also includes the output values of the circuit for each stable state.

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

(a) Four states with one input

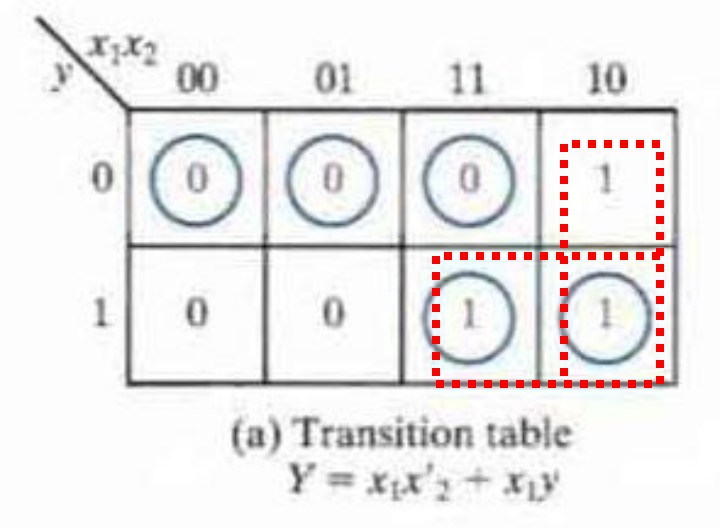
x_1x_2	00	01	11	10
a	(a) 0	(a) 0	(a) 0	b , 0
b	a , 0	a , 0	(b) 1	(b) 0

(b) Two states with two inputs and one output

If a transition table has only one stable state in each row then it is called as ***primitive flow table***

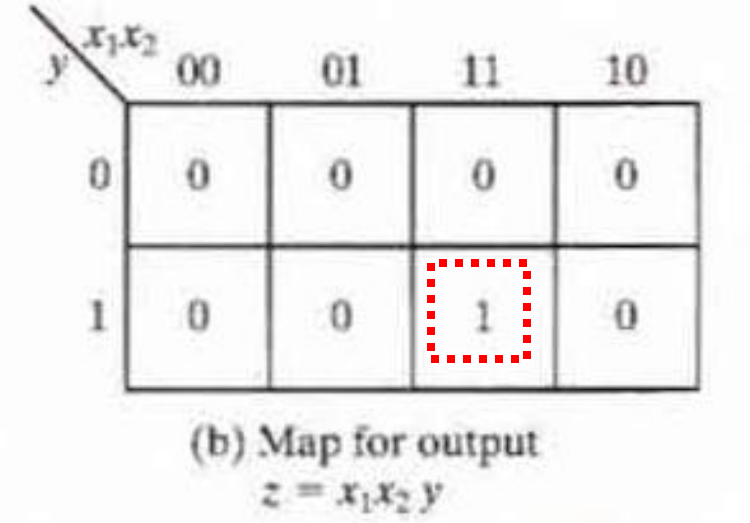
- Figure (a) is called a *primitive flow table because it has only one stable state in each row.*
- Figure (b) shows a now table with more than one stable state in the same row.
- The binary value of the output variable is indicated inside the square next to the state symbol and is separated from the state symbol by a comma.

- To obtain the circuit described by a flow table, it is necessary to assign a distinct binary value to each state.
- Such an assignment converts the flow table into a transition table from which we can derive the logic diagram.
- Assign 0 to state a and 1 to state b, the result is the transition table
- The output map is obtained directly from the output values in the flow table



(a) Transition table

$Y = x_1x'_2 + x_1y$

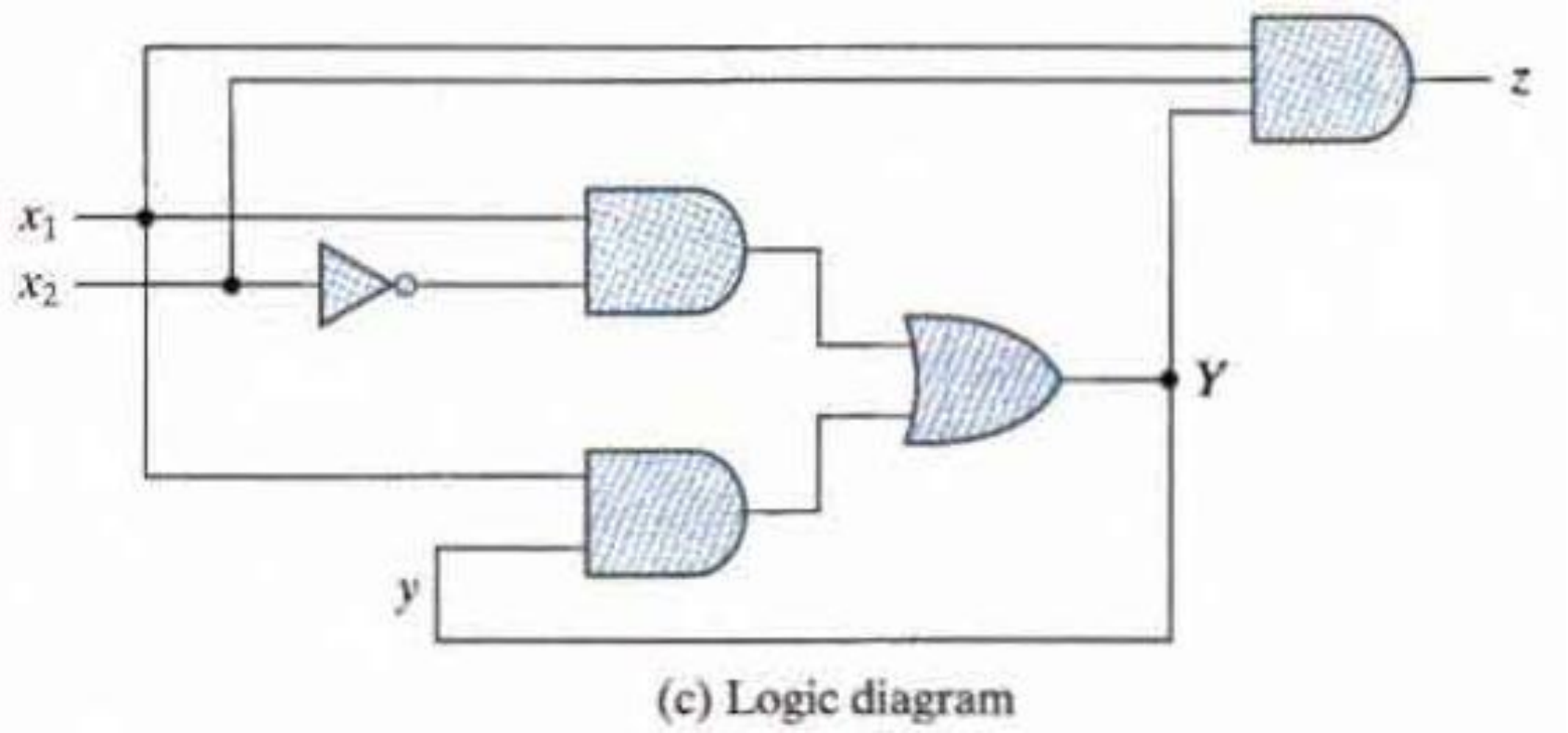


(b) Map for output

$z = x_1x_2\,y$

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

$$z = x_1 x_2 y$$



(c) Logic diagram

An asynchronous sequential circuit is described by the following excitation and output function,

Y= x1x2+ (x1+x2) y

Z= Y

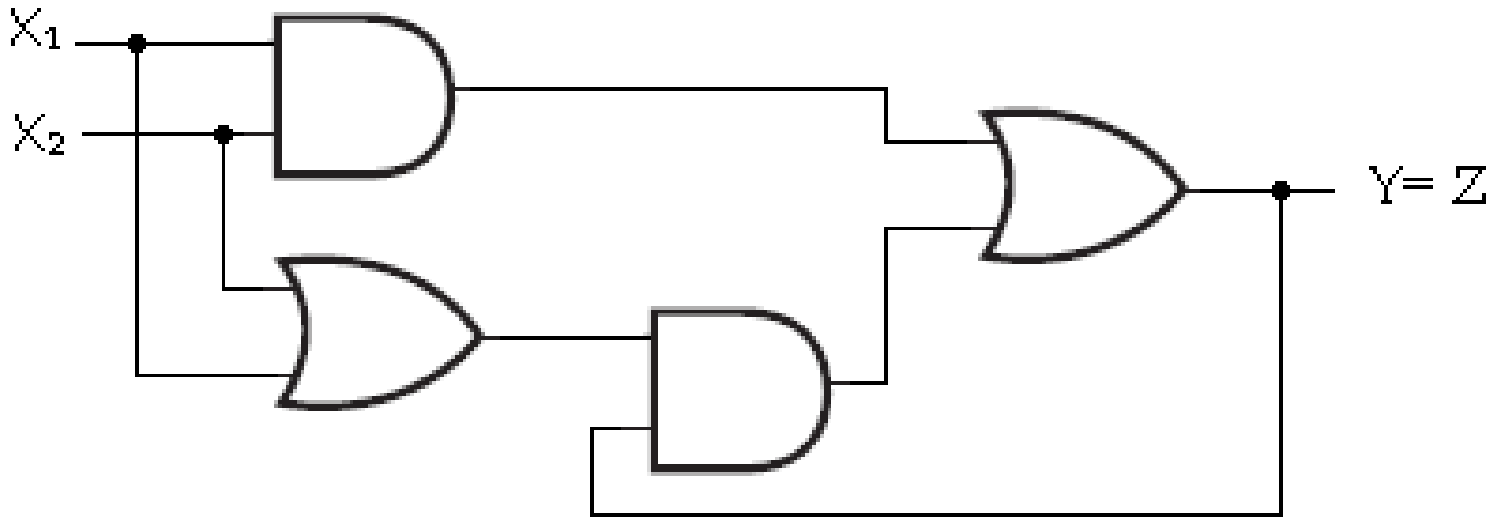
a) Draw the logic diagram of the circuit.

b) Derive the transition table, flow table and output map.

c) Describe the behavior of the circuit.

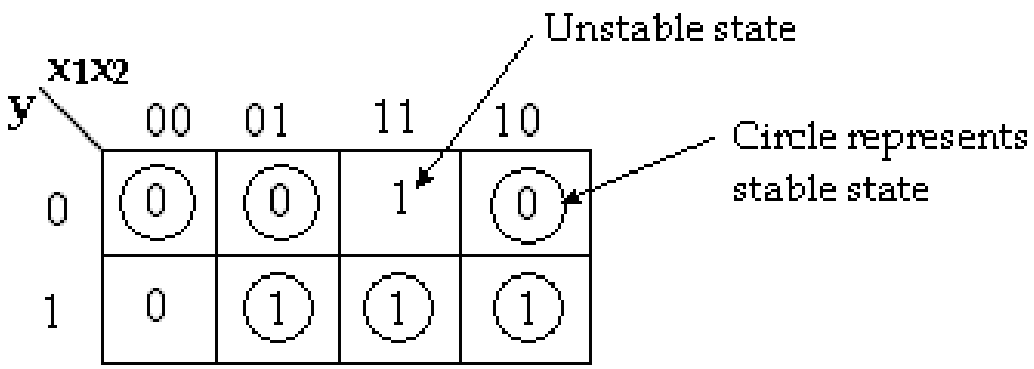
Y= x1x2+ (x1+x2) y

Z= Y

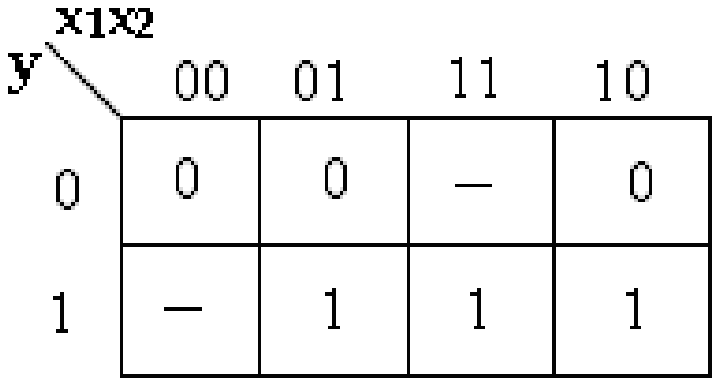


Logic diagram

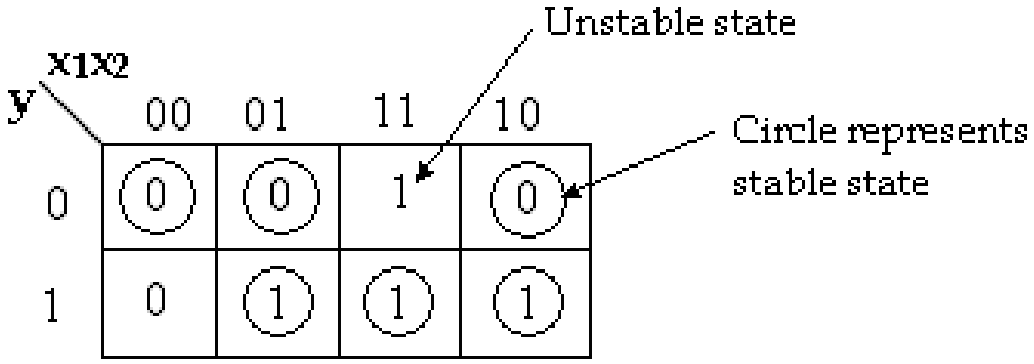
y	x_1	x_2	x_1x_2	$(x_1+x_2)y$	$Y= x_1x_2+ (x_1+x_2)y$	$Z= Y$
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	1	0	1	1
1	0	0	0	0	0	0
1	0	1	0	1	1	1
1	1	0	0	1	1	1
1	1	1	1	1	1	1



Transition table



Output map



Assign a= 0; b= 1

y \ x1x2	00	01	11	10
0	(a)	(a)	b	(a)
1	a	(b)	(b)	(b)

Flow table

The circuit gives carry output of the full adder circuit

Race Conditions

- A race condition is said to exist in an asynchronous sequential circuit when two or more binary state variables change value in response to a change in an input variable.
- When unequal delays are encountered, a race condition may cause the stale variables to change in an unpredictable manner.

- Races are classified as:

  i. Non-critical races

  ii. Critical races.

Non-critical races

- If the final stable state that the circuit reaches does not depend on the order in which the state variables change, the race is called a non-critical race.
- If a circuit, whose transition table starts with the total stable state y1y2x= *000* and then change the input from 0 to 1. The state variables must then change from 00 to 11, which define a race condition.

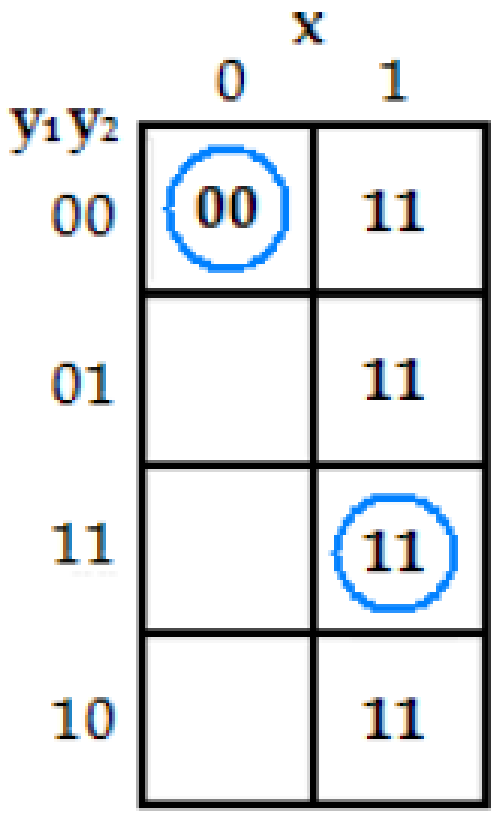
  The possible transitions are:

  00 → 11

  00 → 01 → 11

  00 → 10 → 11

- In all cases, the final state is the same, which results in a non-critical condition .

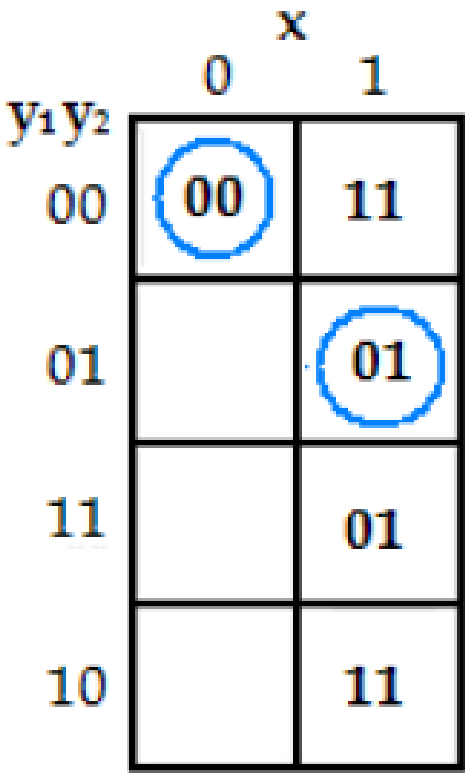


(a) Possible transitions:

00 → 11

00 → 01 → 11

00 → 10 → 11



(b) Possible transitions:

00 → 11 → 01

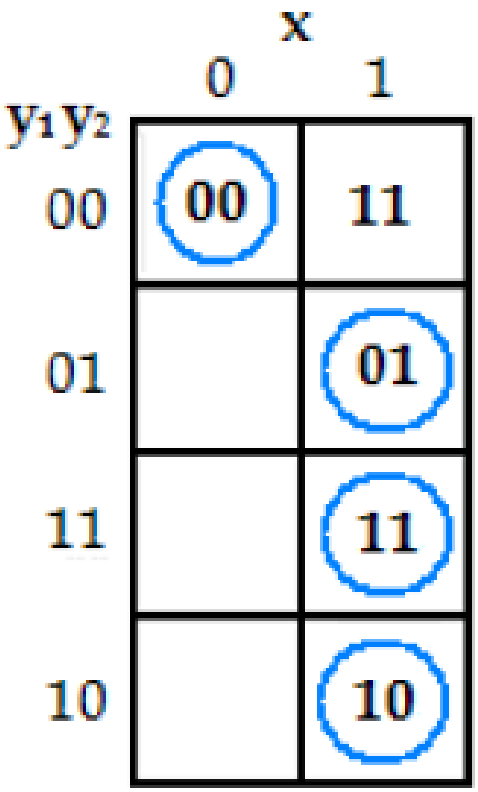
00 → 01

00 → 10 → 11 → 01

Examples of Non-critical Races

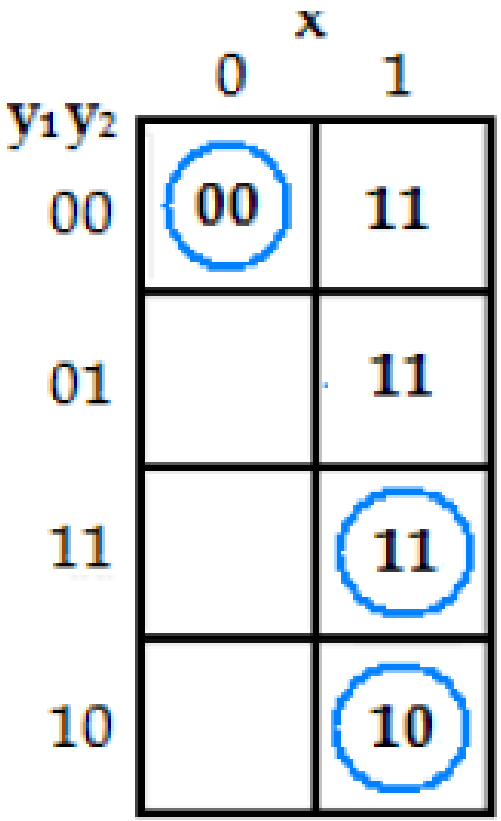
Critical races

- A race becomes critical if the correct next state is not reached during a state transition. If it is possible to end up in two or more different stable states, depending on the order in which the state variables change, then it is a critical race. For proper operation, critical races must be avoided.
- Stable state (y1y2x= 000), and then change the input from 0 to 1. The state variables must then change from 00 to 11. If they change simultaneously, the final total stable state is 111.
- If, because of unequal propagation delay, Y2 changes to 1 before Y1 does, then the circuit goes to the total stable state 011 and remains there.
- If, however, Y1 changes first, the internal state becomes 10 and the circuit will remain in the stable total state 101.
- Hence, the race is critical because the circuit goes to different stable states, depending on the order in which the state variables change.



(a) Possible transitions:

00 → 11
00 → 01
00 → 10



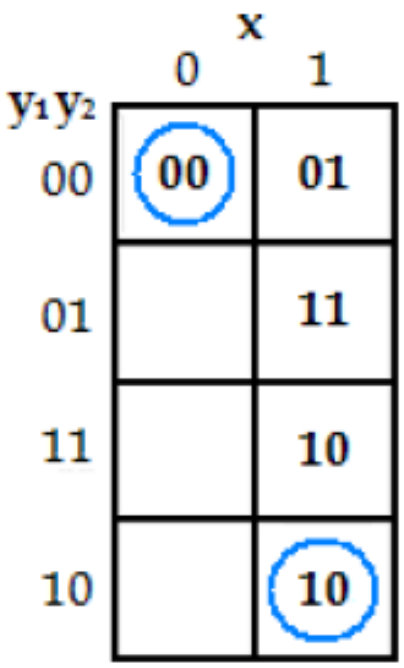
(b) Possible transitions:

00 → 11
00 → 01 → 11
00 → 10

Examples of Critical Races

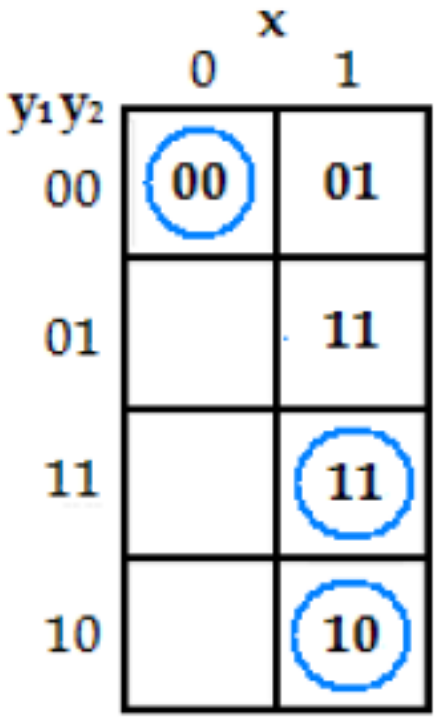
CYCLES

- Races can be avoided by directing the circuit through intermediate unstable states with a unique state-variable change.
- When a circuit goes through a unique sequence of unstable states, it is said to have a ***cycle.***
- Care must be taken when using a cycle that terminates with a stable state.
- If a cycle does not terminate with a stable state, the circuit will keep going from one unstable state to another, making the entire circuit unstable.



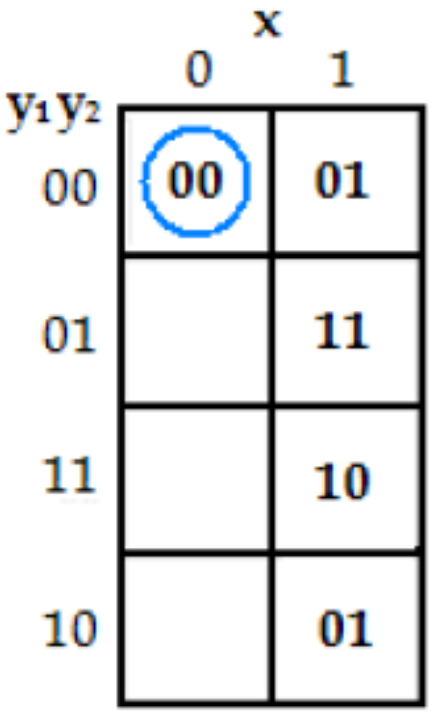
(a) State transition:

00 → 01 → 11 → 10

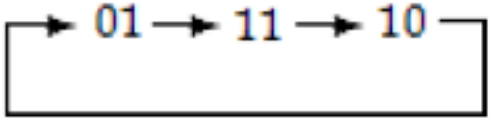


(b) State transition:

00 → 01 → 11



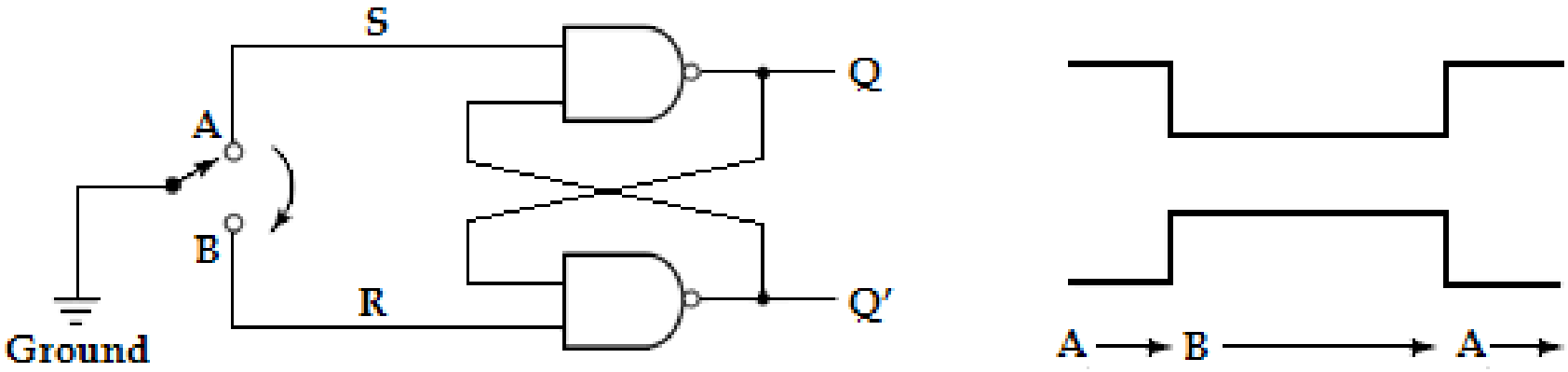
(c) Unstable:



Examples of cycles

Debounce Circuit

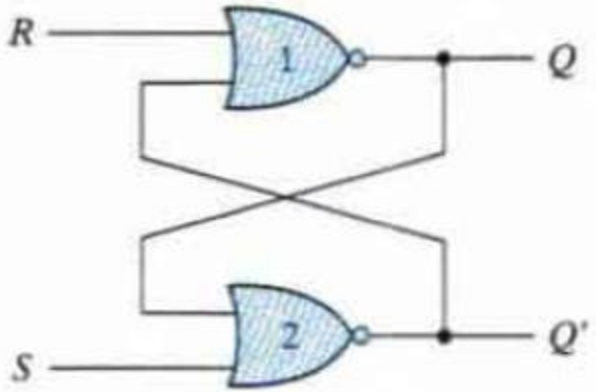
- A debounce circuit is a circuit which removes the series of pulses that result from a contact bounce and produces a single smooth transition of the binary signal from *0 to 1* or from *1 to 0.*
- One such circuit consists of a single-pole, double-throw switch connected to an SR latch.



Debounce Circuit

Circuits With Latches

SR Latch

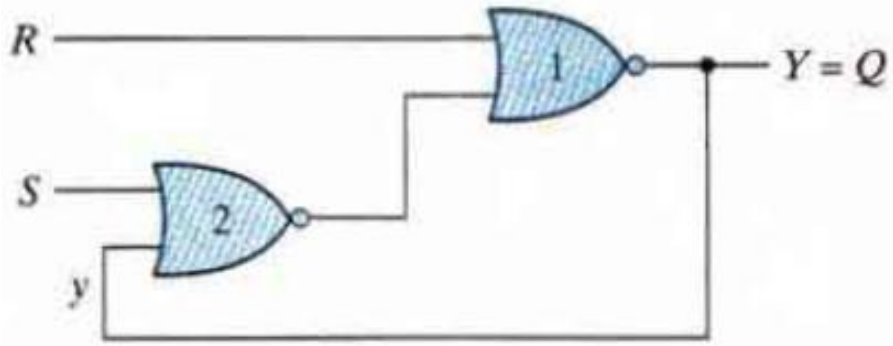


(a) Cross-coupled circuit

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

$$Y = [(S + y)' + R]' = (S + y)R' = SR' + R'y$$



(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

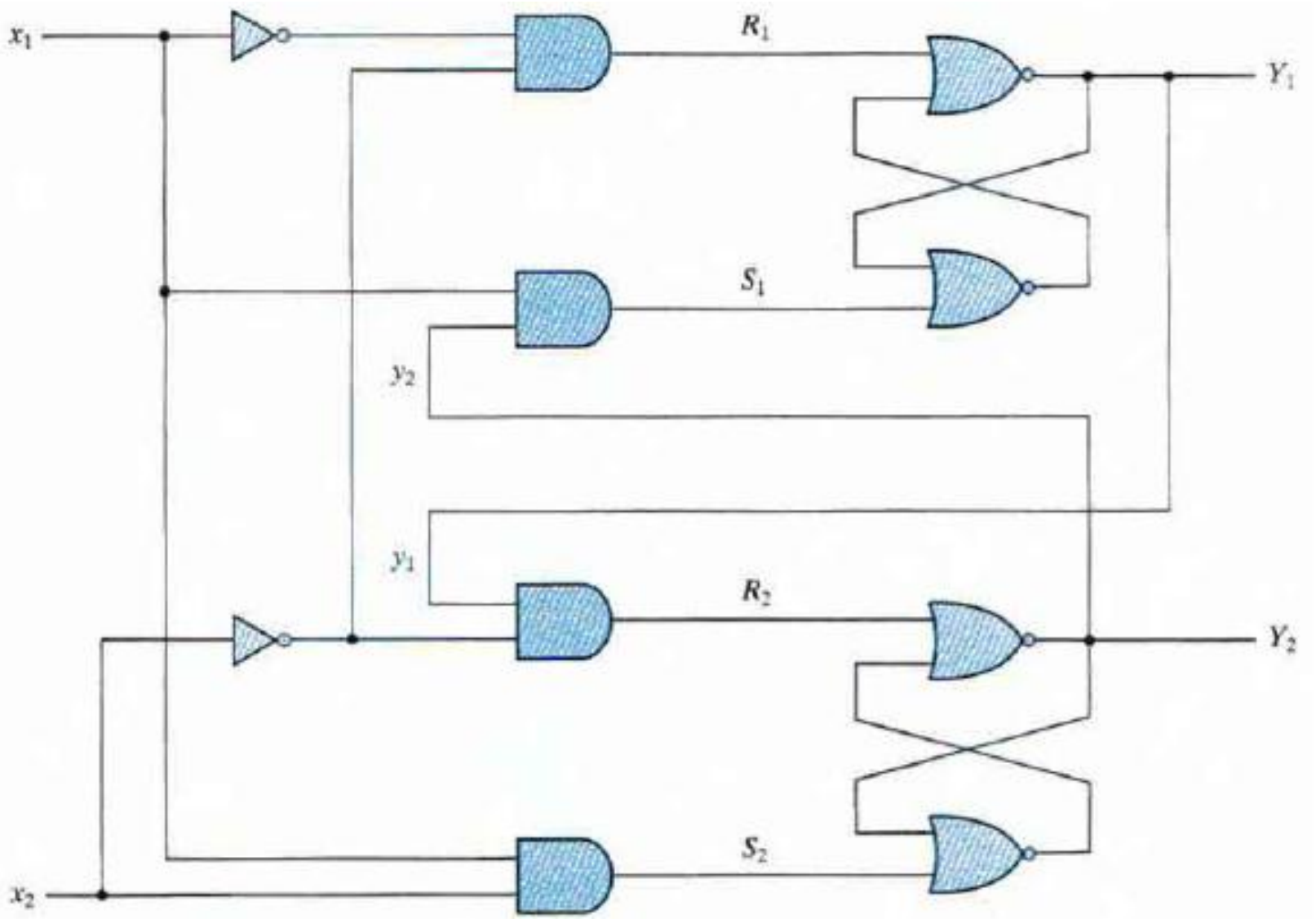
- The circuit exhibits some difficulty when both S and R are equal to 1 (Q = Q' = 0)
- From the transition table, we note that going from SR = 11 to *SR = 00* produces an unpredictable result
- Make sure that 1's are not applied to both the S and *R* inputs simultaneously. ➔ SR = 0

$$SR' + SR = S(R' + R) = S$$

$SR' = S$ when $SR = 0$.

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

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



Example of a circuit with *SR Latches*

- The Boolean functions for the S and R inputs in each latch are

  S1 = x1y2 **S2 = x1x2**

  R1 = x1'x2' **R2 = x2'y1**

- Check whether the conditions SR= 0 is satisfied to ensure proper operation of the circuit.

  S1R1 = x1y2 x1'x2' = 0

  S2R2 = x1x2 x2'y1 = 0 **(**x1x1' = x2x2' = 0)

- Evaluate Y1 and Y2. The excitation functions are derived from the relation **Y= S+ R'y.**

  Y1= S1+ R1'y1 = x1y2 +(x1'x2')' y1

  = x1y2 +(x1+ x2) y1 = x1y2 +x1y1+ x2y1

  Y2= S2+ R2'y2 = x1x2+ (x2'y1)'y2

  = x1x2+ (x2+ y1') y2 = x1x2+ x2y2+ y1'y2

y1	y2	x1	x2	x1y2	x1y1	x2y1	x1x2	x2y2	y1′y2	Y1	Y2

Map for Y_1

y_1y_2 \ x_1x_2	00	01	11	10
00	0	0	0	0
01	0	0	1	1
11	0	1	1	1
10	0	1	1	1

Map for Y_2

y_1y_2 \ x_1x_2	00	01	11	10
00	0	0	1	0
01	1	1	1	1
11	0	1	1	0
10	0	0	1	0

Map for Y_1Y_2

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)

Transition Table

The procedure for analyzing an asynchronous sequential circuit with SR latches

1. Label each latch output with Yi and its external feedback path (if any) with yi for *i = 1,2 ,..,, k.*
2. Derive the Boolean functions for the Si 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 condition 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= Y1Y2 ……Yk in the map.
7. Circle all stable states such that ***Y = y.*** The resulting map is the transition table.

Design of Asynchronous Sequential Circuits

- The design of an asynchronous sequential circuit starts from the statement of the problem and concludes in a logic diagram. The design steps must be carried out in order to minimize the circuit complexity and to produce a stable circuit without critical races.

The design steps are as follows:

1. State the design specifications.
2. Obtain a primitive flow table from the given design specifications.
3. Reduce the flow table by merging rows in the primitive flow table.
4. Assign binary state variables to each row of the reduced flow table to obtain the transition table. The procedure of state assignment eliminates any possible critical races.
5. Assign output values to the dashes associated with the unstable states to obtain the output maps.
6. Simplify the Boolean functions of the excitation and output variables and draw the logic diagram.

- Design a gated latch circuit with inputs, G (gate) and D (data), and one output, Q.
- Binary information present at the D input is transferred to the Q output when G is equal to 1.
- 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 occurred is retained at the Q output.
- The gated latch is a memory element that accepts the value of D when G= 1 and retains this value after G goes to 0, a change in D does not change the value of the output Q.

- From the design specifications, we know that

  Q= 0 if DG= 01

  and Q= 1 if DG= 11

  because D must be equal to Q when G= 1.
- When G goes to 0, the output depends on the last value of D.
- Thus, if the transition is from 01 to 00 to 10, then Q must remain 0 because D is 0 at the time of the transition from 1 to 0 in G.
- If the transition of DG is from 11 to 10 to 00, then Q must remain 1.
- This information results in six different total states, as shown in the table

State	Inputs		Output	Comments
	D	G	Q	

- A ***primitive flow table*** is a flow table with only one stable total state in each row. It has one row for each state and one column for each input combination.
- A total stale consists of the internal state combined with the input

States

Primitive Flow Table

Fill in one square in each row belonging to the stable state in that row. These entries are determined from Table.

Next, both inputs are nor allowed to change simultaneously, enter dash marks in each row that differs in two or more variables from the input variables associated with the stable state.

Next, it is necessary to find values for two more squares in each row. The comments listed in Table may help in deriving the necessary information

Reduction of the Primitive Flow Table

- The primitive flow table has only stable state in each row.
- The table can be reduced to a smaller number of rows if two or more stable states are placed in the same row of the flow table.
- The grouping of stable states from separate rows into one common row is called ***merging.***
- Two or more rows in the primitive flow table can be merged into one row if there are non conflicting states and outputs in each of the columns.
- Whenever one state symbol and don't -care entries are encountered in the same column, the state is listed in the merged row.
- If the state is circled in one of the rows. it is also circled in the merged row.
- The output value is included with each stable state in the merged row.

DG

	00	01	11	10
a	c, –	ⓐ, 0	b, –	–, –
c	ⓒ, 0	a, –	–, –	d, –
d	c, –	–, –	b, –	ⓓ, 0

DG

	00	01	11	10
b	–, –	a, –	ⓑ, 1	e, –
e	f, –	–, –	b, –	ⓔ, 1
f	ⓕ, 1	a, –	–, –	e, –

States that are candidates for merging

DG

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

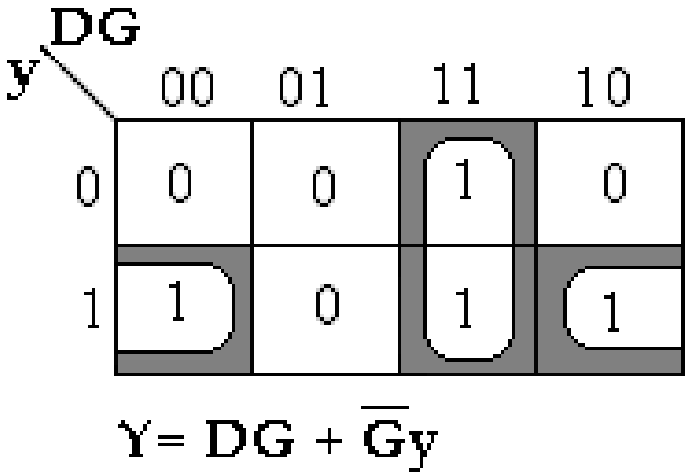
Reduced Table- 1

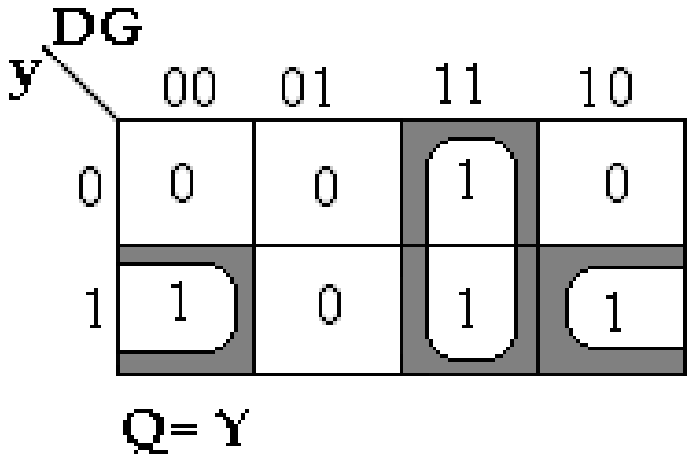
DG

	00	01	11	10
a	ⓐ, 0	ⓐ, 0	b, –	ⓐ, 0
b	ⓑ, 1	a, –	ⓑ, 1	ⓑ, 1

Reduced Table- 2

- Assign distinct binary value to each state.
- This assignment converts the flow table into a transition table.
- A binary state assignment must be made to ensure that the circuit will be free of critical races.
- Assign 0 to state a, and 1 to state b in the reduced state table.

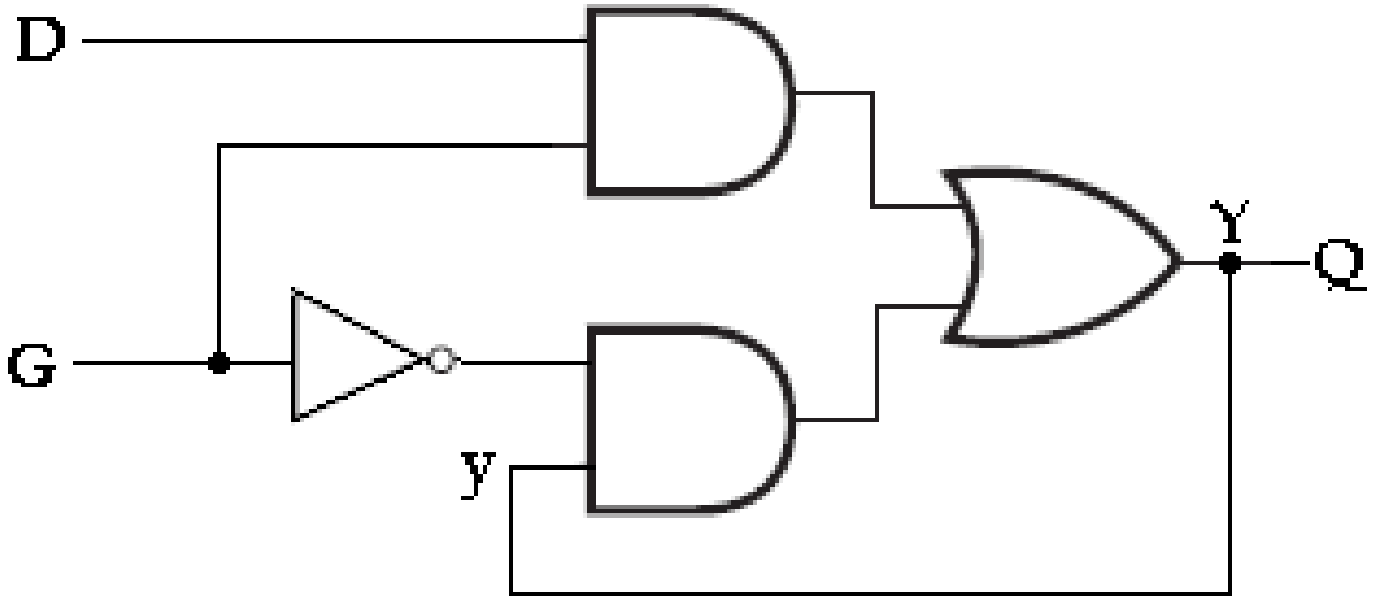




Transition Table and Output map

$$Y = DG + \overline{G}y$$

$$Q = Y$$



Gated-Latch Logic diagram

Assigning Outputs to Unstable States

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 .

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

(a) Flow table

0	
	0
1	
	1

(b) Output assignment

Reduction of State and Flow Tables

- The procedure for reducing the number of internal states in an asynchronous sequential circuit resembles the procedure that is used for synchronous circuits.
- An algorithm for the state reduction of a completely specified state table, state-reduction method that uses an implication table.
- The algorithm and the implication table will then be modified to cover the state reduction of incompletely specified state tables.

Implication Table and Implied State

- There are occasions when a pair of states do not have the same next states, but, nonetheless, go to equivalent next states.
- The present states *a* and *b* have the same output for the same input.
- Their next states are c and d for x = 0 and b and *a for x = 1* .
- If the pair of states (c, d) are equivalent, then the pair of states *(a , b)* will also be equivalent.
- Then ***(a, b) imply (c, d)***
- If (a, b) imply (c, d) and (c, d) imply (a , b), then a and b are equivalent, and so are c and d.

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

- The checking of each pair of states for possible equivalence in a table with a large number of states can be done systematically by means of an implication table,
- It is a chart that consists of squares.
- One for every possible pair of states, that provide spaces for listing any possible implied states.
- By judicious use of the table, it is possible to determine all pairs of equivalent states.
- On the left side along the vertical are listed all the states defined in the state table except the first, and across the bottom horizontally are listed all the states except the last.
- The result is a display of all possible combinations of two states, with a square placed in the intersection of a row and a column where the two states can be tested for equivalence.
- Two states having different outputs for the same input are not equivalent.

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

State Table to Be Reduced

put
$x = 1$
0
0
1
0
0
0
0

Implication table

(a, b) (d, e) (d, g) (e, g)

- The equivalent states are

  (a, b) (d, e) (d, g) (e, g)

- The last three pairs can be combined into a set of three equivalent states ***(d, e, g)***
- The final partition of the 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)

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

- When the state table for a sequential circuit is incompletely specified.
- This happens when certain combinations of inputs or input sequences never occur because of external or internal constraints.
- The next states and outputs as don't care conditions
- Incompletely specified states can be combined to reduce the number of states in the flow table.
- Such stares cannot be called equivalent because the formal definition of equivalence requires that all outputs and next states be specified for all inputs.
- Instead, two incompletely specified states that can be combined are said to be ***Compatible***.

- Two states are compatible if, for each possible input, they have the same output whenever it is specified and their next states are compatible whenever they are specified.
- All don’t-care conditions marked with dashes have no effect in the search for compatible states , as they represent unspecified conditions .
- The process that must be applied in order to find a suitable group of compatibles for the purpose of merging a flow tab le can be divided into three steps:

1. Determine all compatible pairs by using the implication table.
2. Find the maximal compatibles with the use of a merger diagram.
3. Find a minimal collection of compatibles that covers all the states and is closed.

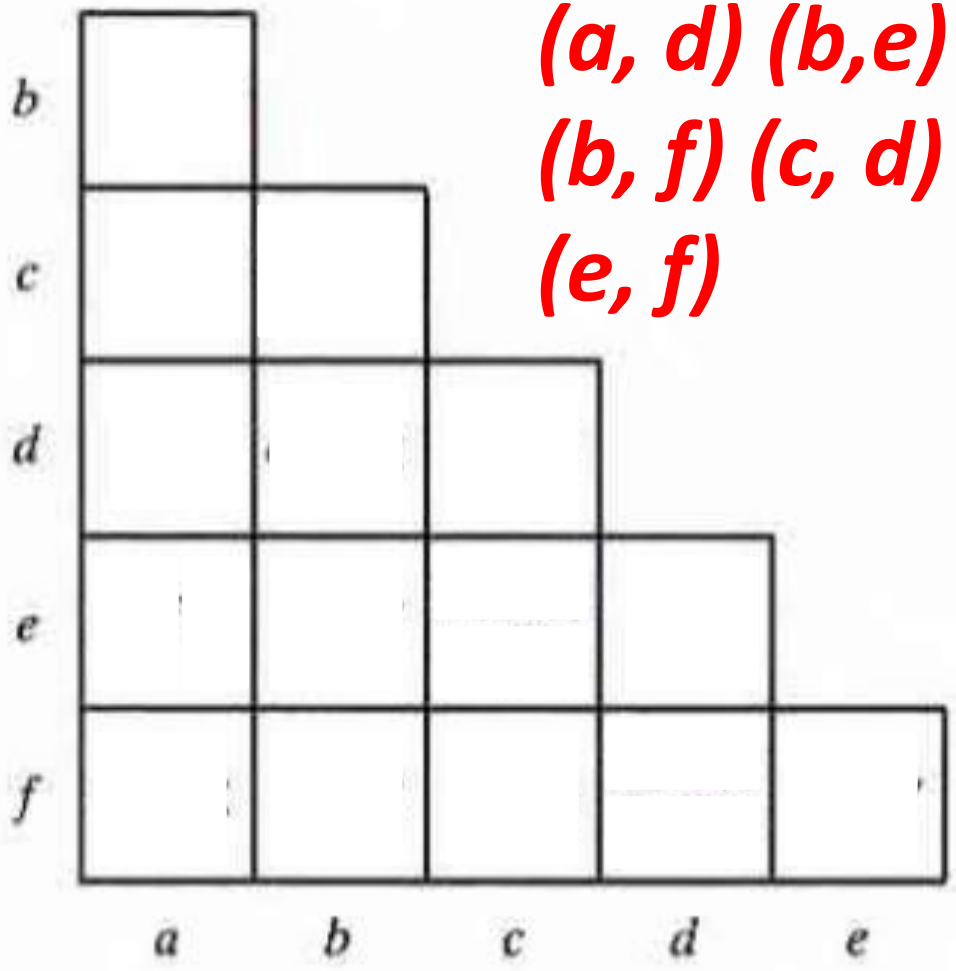
- The minimal collection of compatibles is then used to merge the rows of the flow table.

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

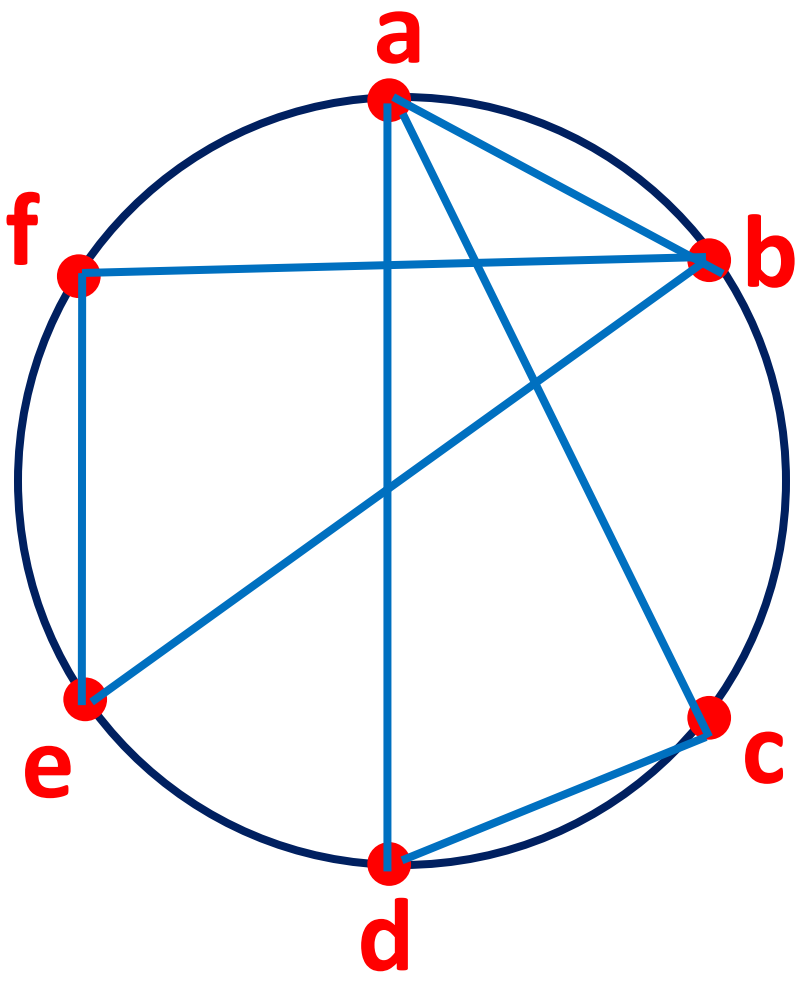


(b) Implication table

Maximal Compatibles

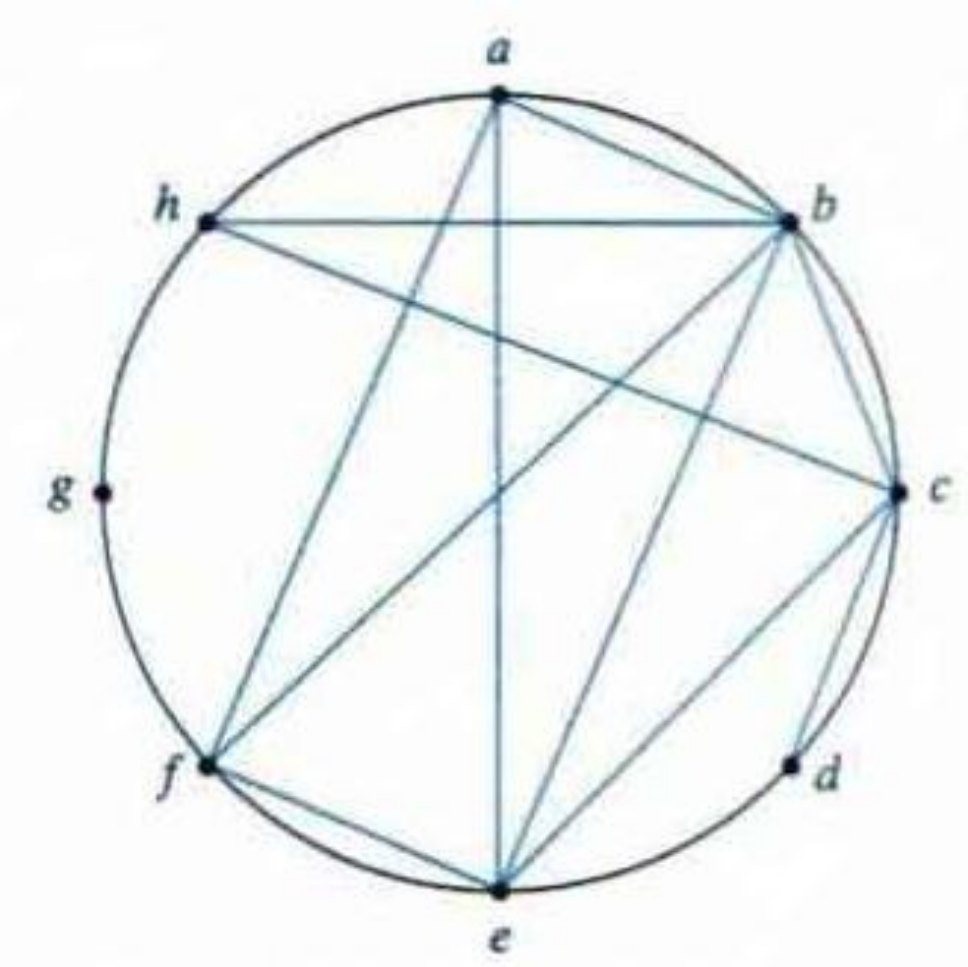
- The ***maximal compatible*** is a group of compatibles that contains all the possible combinations of compatible states.
- The maximal compatible can be obtained from a ***merger diagram***.
- The merger diagram is a graph in which each state is represented by a dot placed along the circumference of a circle.
- Lines are drawn between any two corresponding dots that form a compatible pair.
- All possible compatibles can be obtained from geometrical patterns in which states are connected to each other.
- An isolated dot represents a state that is not compatible with any other state.
- A line represents a compatible pair.
- A triangle constitutes a compatible with three states .
- An n-state compatible is represented in the merger diagram by an n-sided polygon with all its diagonals connected.

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



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

Maximal Compatible



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

Maximal Compatible

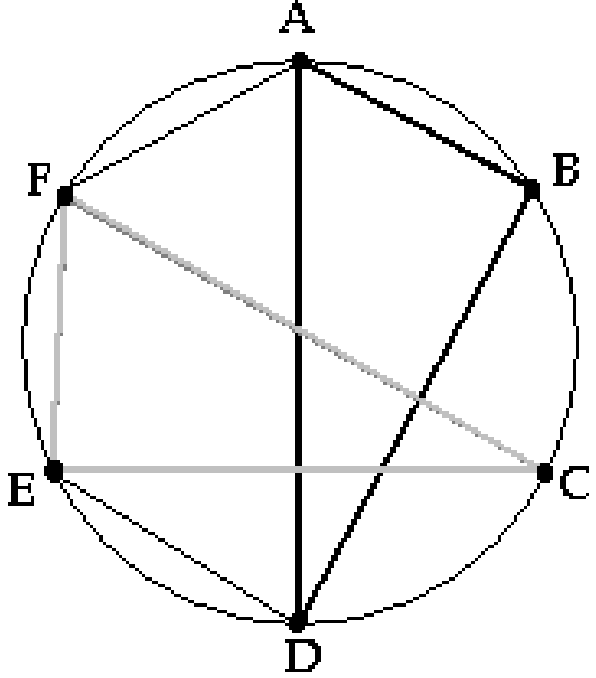
Closed covering condition

- The condition that must be satisfied for merging rows is that the set of chosen compatibles must ***cover*** all the states and must be ***closed***.
- The set will cover all the states if it includes all the states of the original state table.
- The closure condition is satisfied if there are no implied states or if the implied states are included within the set.
- A closed set of compatibles that covers all the states is called a ***closed covering***.

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

- If we remove *(a, b), we are left with* a set of two compatibles: *(a, c, d) (b, e, f)*
- All six states from the flow table are included in this set.
- Thus, the set satisfies the covering condition
- Therefore, the primitive flow table can be merged into two rows, one for each of the compatibles.

	A	B	C	D	E
B	✓				
C	A,E ×	A,E × D,F ×			
D	✓	✓	D,F ×		
E	A,E ×	A,E ×	✓	✓	
F	✓	D,F ×	✓	D,F ×	✓



- The compatible pairs derived from the implication table are
  (A, B) (A, D) (A, F) (B, D) (C, E) (C, F) (D, E) (E, F)
- The maximal compatibles:
  (A, B, D) (C, E, F) (A, F) (D, E)
- If we remove (A, F*) and (D, E*), we are left with a set of two compatibles
  (A, B, D) (C, E, F)
- All six states from the primitive flow table are included in this set

Design a negative-edge triggered T flip-flop.

The circuit has two inputs, T (toggle) and G (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 any other input condition, the output Q remains unchanged.

State	Inputs		Output	Comments
	T	G	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 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

Specifications of total states

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, –	–, –

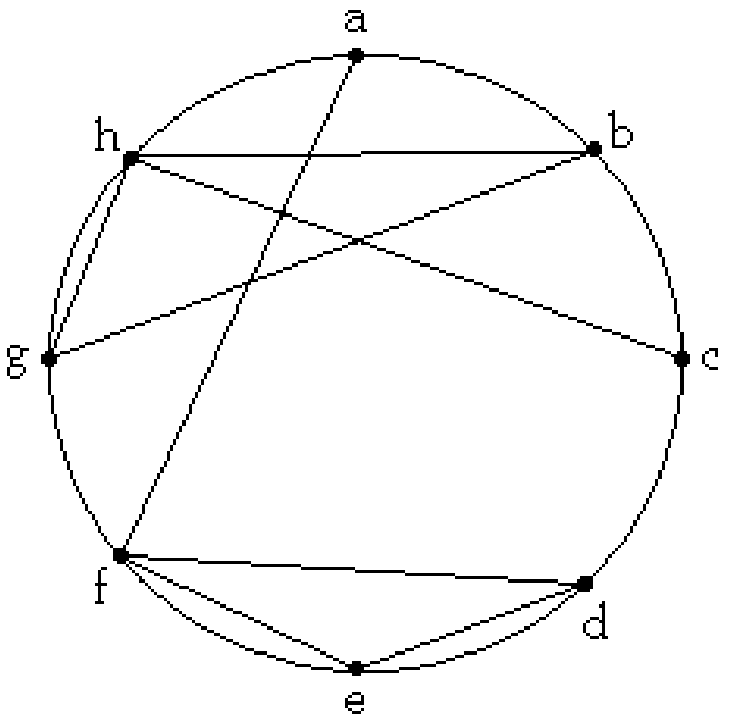
Primitive flow table

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

Implication table

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



Merger Diagram

The maximal compatibles are:

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

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, –	(h), 1	(c), 1	d, –
d, e, f	(e), 0	(f), 0	a, –	(d), 0

Reduced Flow table

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

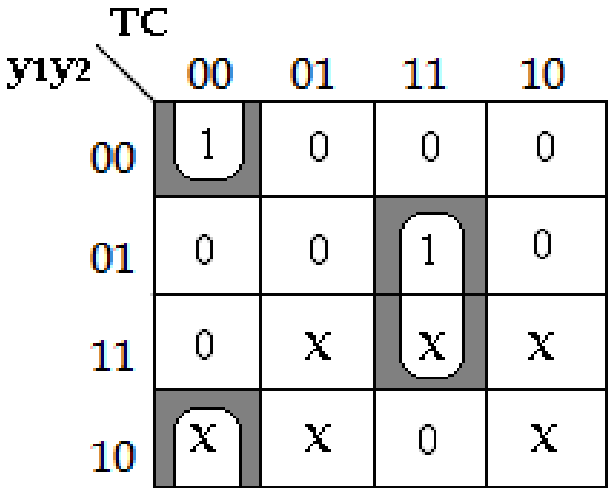
Final Reduced Flow table

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

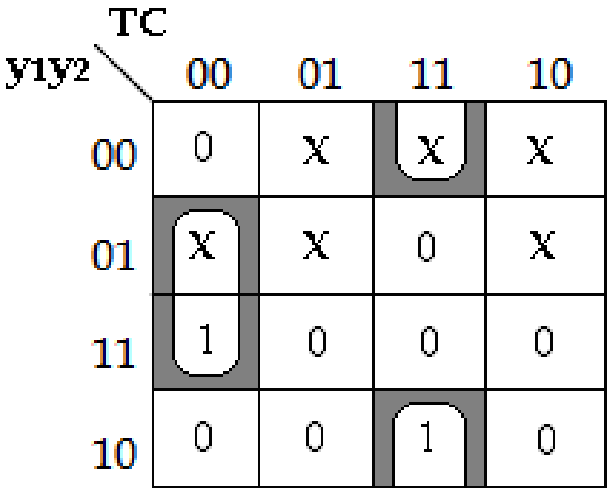
Transition table

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

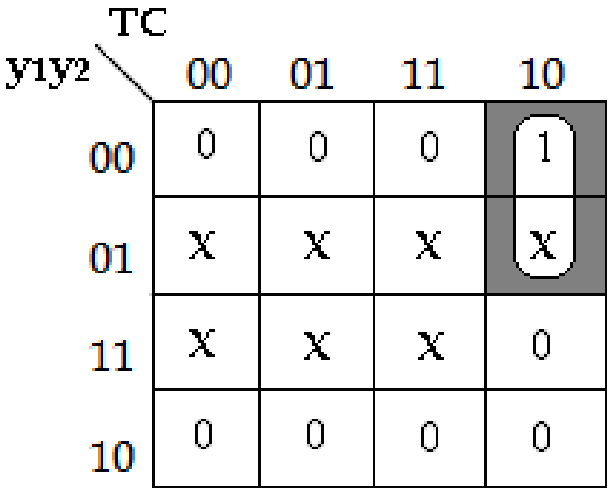
Output map Q= y_2



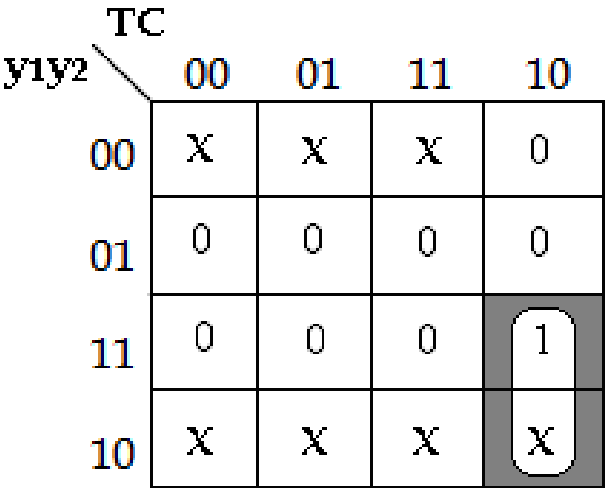
$S_1 = y_2TC + y'_2T'C'$



$R_1 = y_2T'C' + y_2'TC$

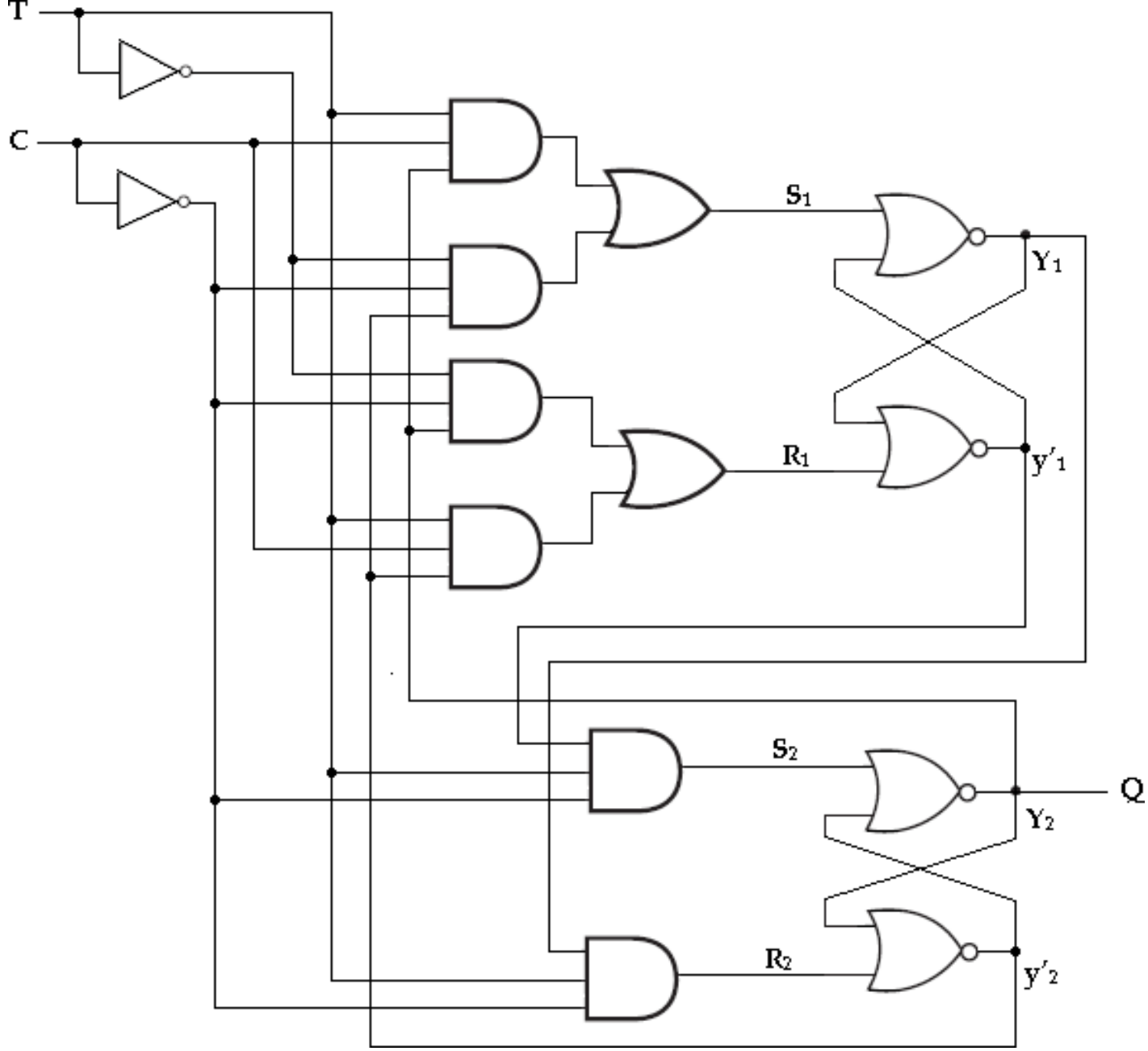


$S_2 = y'_1TC'$



$R_2 = y_1TC'$

Maps for Latch Inputs


T
C
S1
R1
Y1
y'1
S2
R2
Y2
y'2
Q

Race-free State Assignment

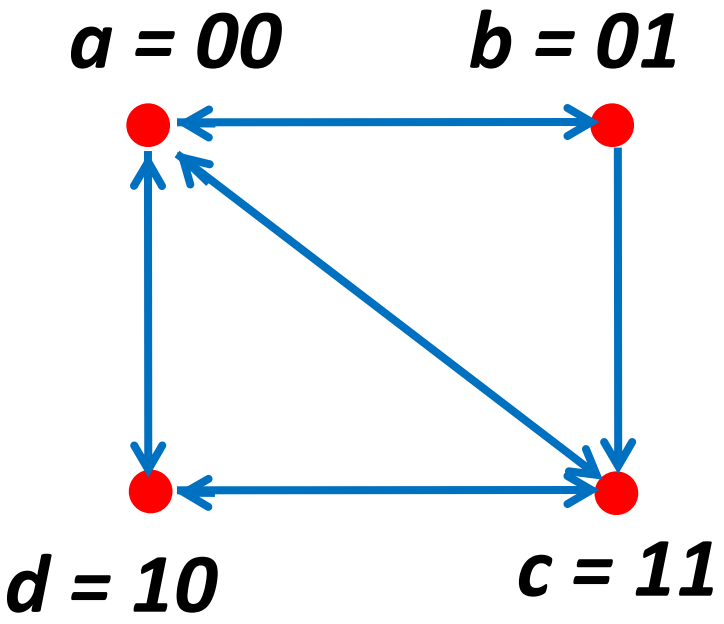
- In synchronous sequential circuit design after reducing the flow table ,assign binary variables to each stable state.
- The primary objective in choosing a proper binary state assignment is the prevention of critical races
- Critical races can be avoided by making a binary state assignment in such a way that only one variable changes at any given time when a state transition occurs in the flow tab le.
- To accomplish this objective, it is necessary that states between which transitions occur be given adjacent assignments.
- Two binary values are said to be *adjacent if they differ in only one variable.*
- For example. 010 and 011 are adjacent because they differ only in the third bit .

Three-Row Flow Table Example

- The assignment of a single binary variable to a flow table with two rows does not impose critical race problems.
- A flow table with three rows requires an assignment of two binary variables.
- Inspection of row *a* reveals that there is a transition from state *a* to state *b* in column 01 and from state *a* to state *c* in column 11.
- This information is transferred into a transition diagram.

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

(a) Flow table



Transition diagram

	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	–

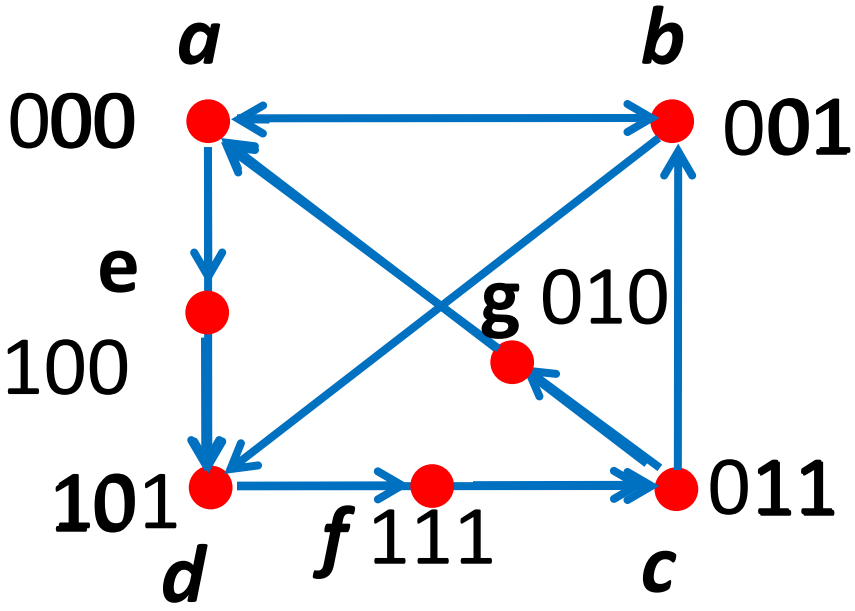
Transition table

Four-Row Flow-Table Example

- A flow table with four rows requires a minimum of two state variables.
- Although a race-free assignment is sometimes possible with only two binary state variables, in many cases the requirement of extra rows to avoid critical races will dictate the use of three binary state variables.

	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



Transition diagram

y_1y_2 / y_3	00	01	11	10
0	a	b	c	g
1	e	d	f	

(a) Binary assignment

	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*	–

State assignment to modified flow table

Multiple-Row Method

- Adding extra rows in the flow table, is referred to as the ***shared-row method***.
- A second method. called the ***multiple-row method,*** is not as efficient, but is easier to apply.
- In multiple-row assignment, each state in the original flow table is replaced by two or more combinations of state variables.
- There are two binary state variables for each stable state, each variable being the logical complement of the other.
- For example, the original slate *a is* replaced with two equivalent states *a1=000* and *a2 =111.*

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

- Hazards are unwanted switching transients that may appear at the output of a circuit because different paths exhibit different propagation delays.
- Hazards occur in combinational circuits, where they may cause a temporary false-output value.
- When this condition occurs in asynchronous sequential circuits, it may result in a transition to a wrong stable state.

Hazards in Combinational Circuits

- A hazard is a condition where a single variable change produces a momentary output change when no output change should occur.
- **Types of Hazards:**

  Static hazard

  Dynamic hazard
- **Static Hazard**

  In digital systems, there are only two possible outputs, a ‘0’ or a ‘1’. The hazard may produce a wrong ‘0’ or a wrong ‘1’. Based on these observations, there are three types,

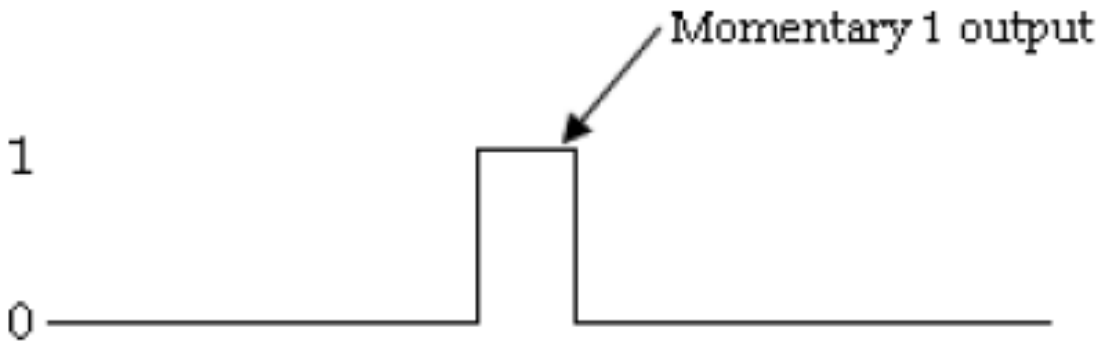
  Static- 0 hazard,

  Static- 1 hazard,

  Dynamic Hazard

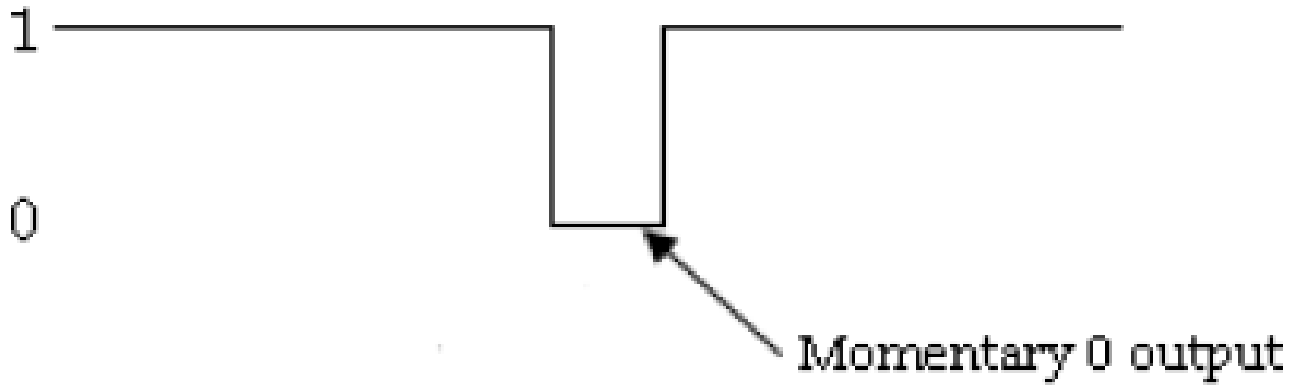
- **Static- 0 hazard**

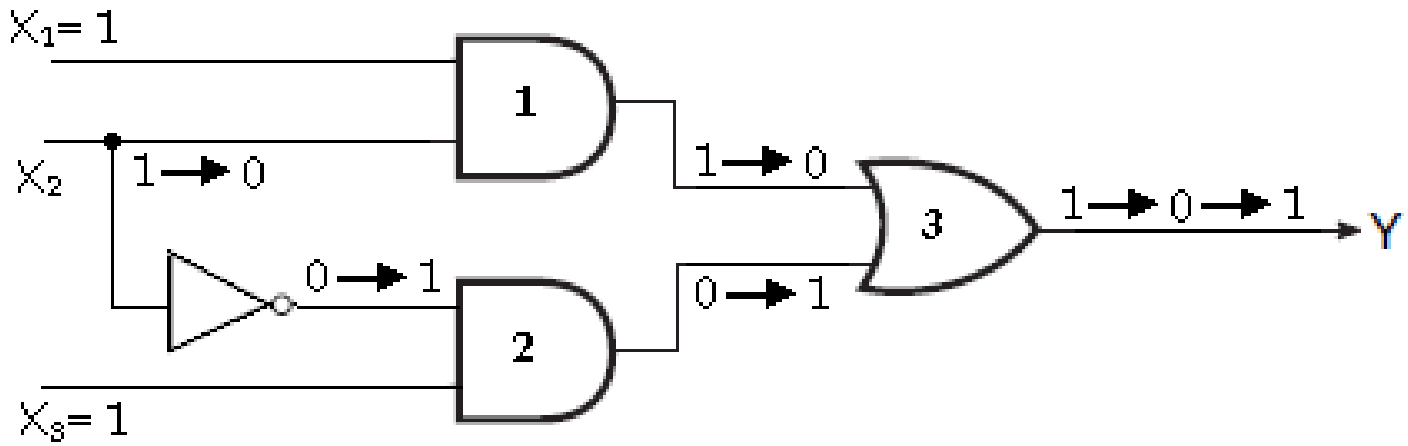
  When the output of the circuit is to remain at 0, and a momentary 1 output is possible during the transmission between the two inputs, then the hazard is called a static 0-hazard.



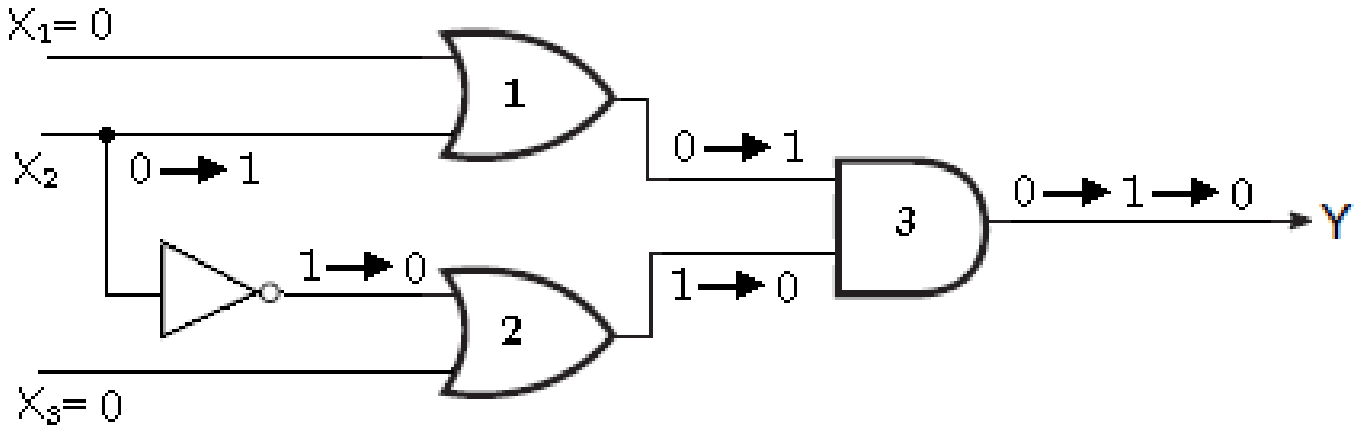
- **Static- 1 hazard**

  When the output of the circuit is to remain at 1, and a momentary 0 output is possible during the transmission between the two inputs, then the hazard is called a static 1-hazard.

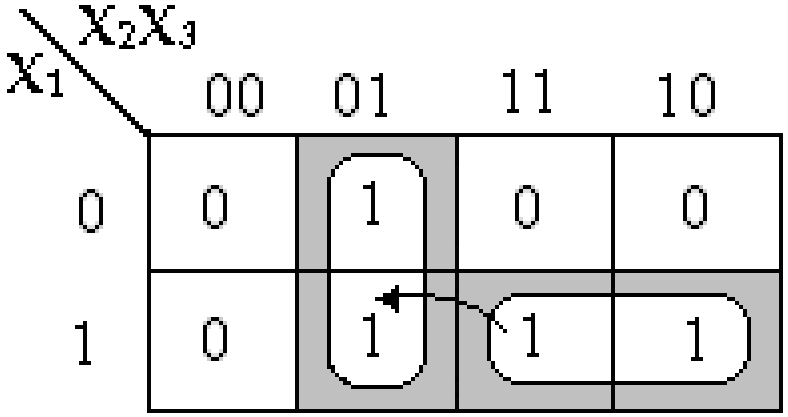




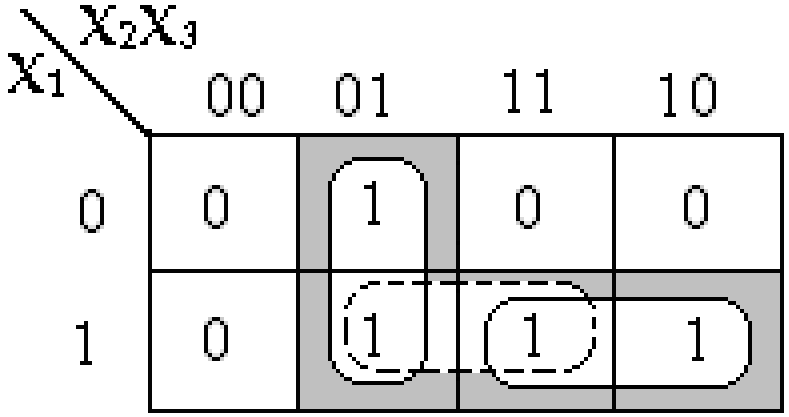
Circuit with static-1 hazard



Circuit with static-0 hazard

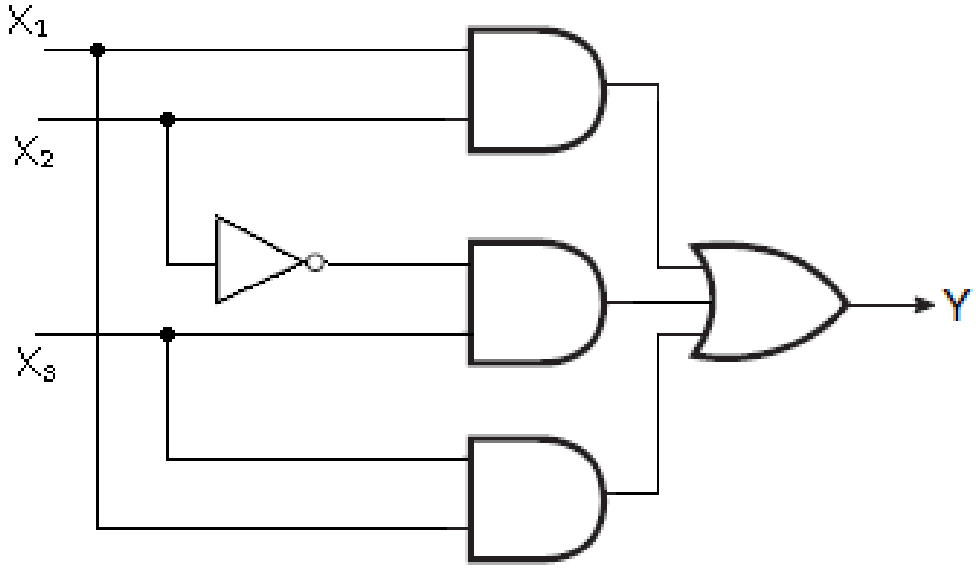


$Y = X_1X_2 + X_2'X_3$



$Y = X_1X_2 + X_2'X_3 + X_1X_3$

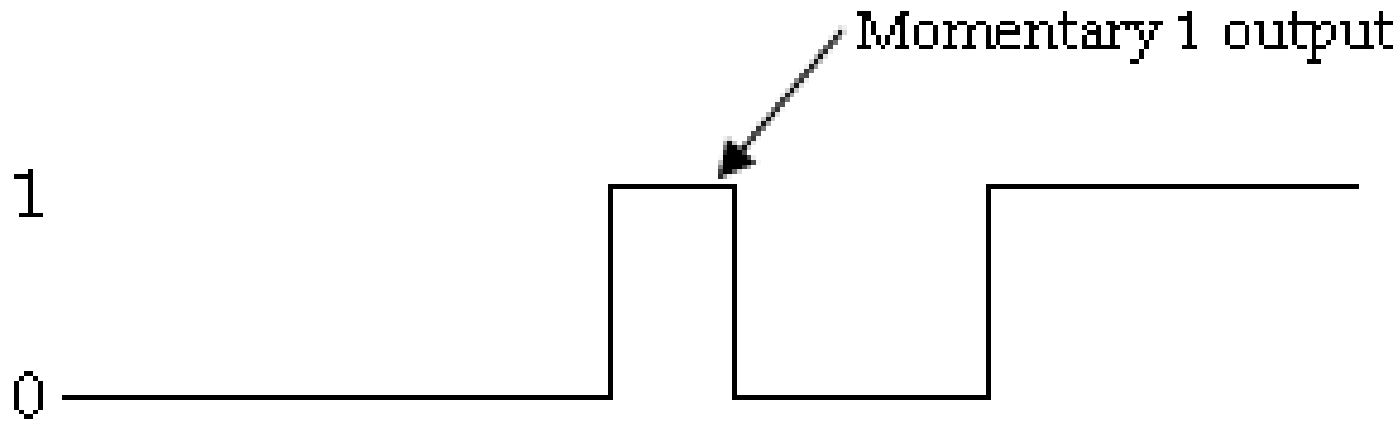
Maps demonstrating a Hazard and its Removal



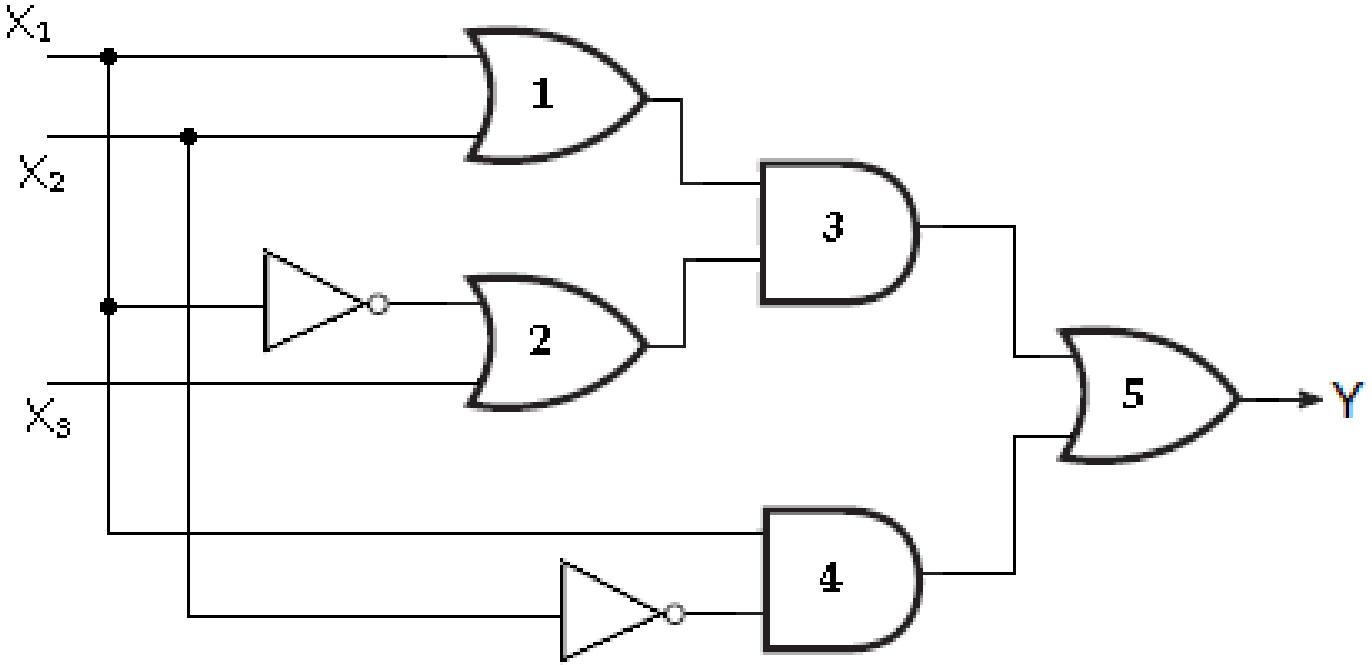
Hazard-free Circuit

- **Dynamic Hazard**

  A transient change occurring three or more times at an output terminal of a logic network when the output is supposed to change only once during a transition between two input states differing in the value of one variable.



Dynamic hazard



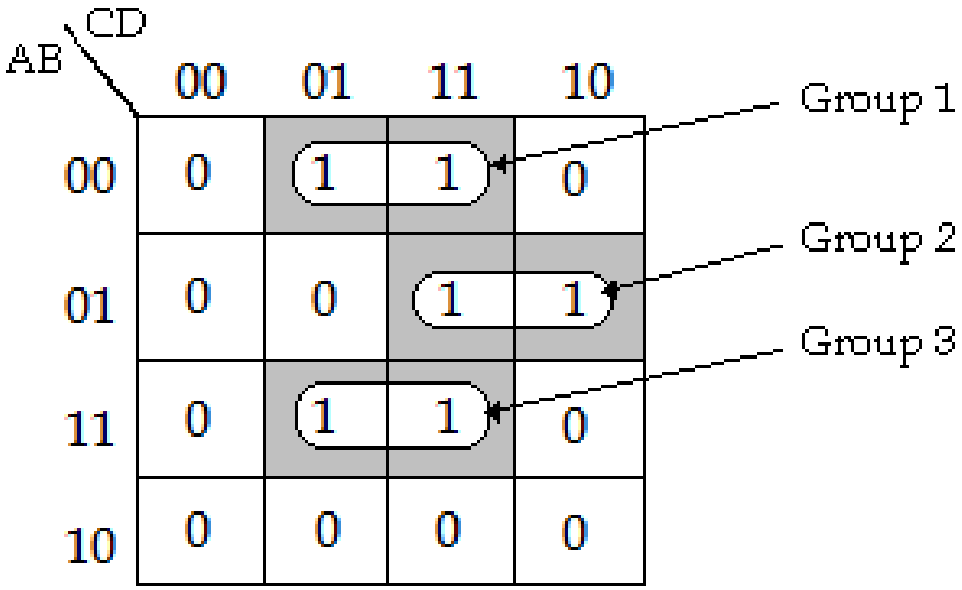
Circuit with Dynamic hazard

- **Essential Hazard**
- An essential hazard is caused by unequal delays along two or more paths that originate from the same input. An excessive delay through an inverter circuit in comparison to the delay associated with the feedback path may cause such a hazard.
- Essential hazards can be eliminated by adjusting the amount of delays in the affected path. To avoid essential hazards, each feedback loop must be handled with individual care to ensure that the delay in the feedback path is long enough compared with delays of other signals that originate from the input terminals.

Design of Hazard Free Circuits

Design a hazard-free circuit to implement the following function.

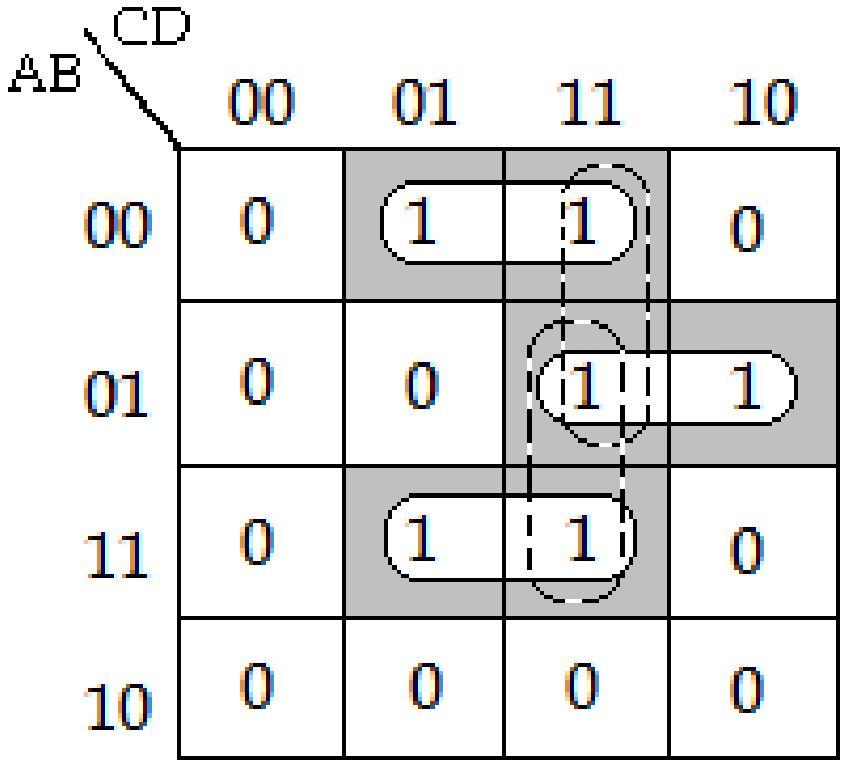
F (A, B, C, D) = Σm (1, 3, 6, 7, 13, 15)



F=A'B'D+ A'BC+ ABD

K-map Implementation and grouping

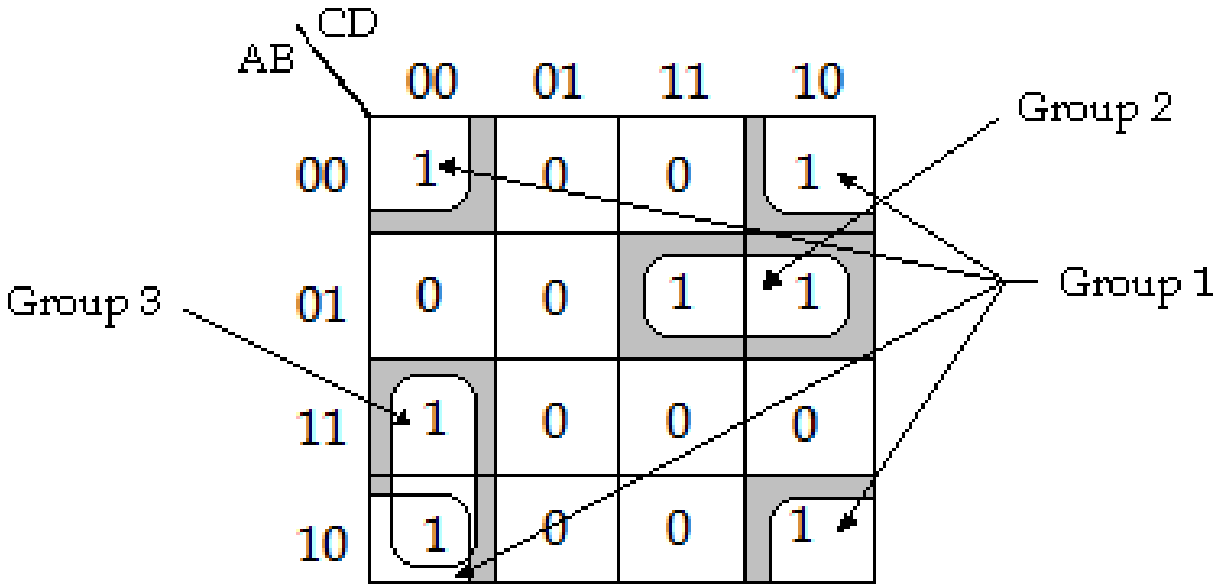
- Hazard- free realization



F=A'B'D+ A'BC+ ABD+ A'CD+ BCD

Design a hazard-free circuit to implement the following function.

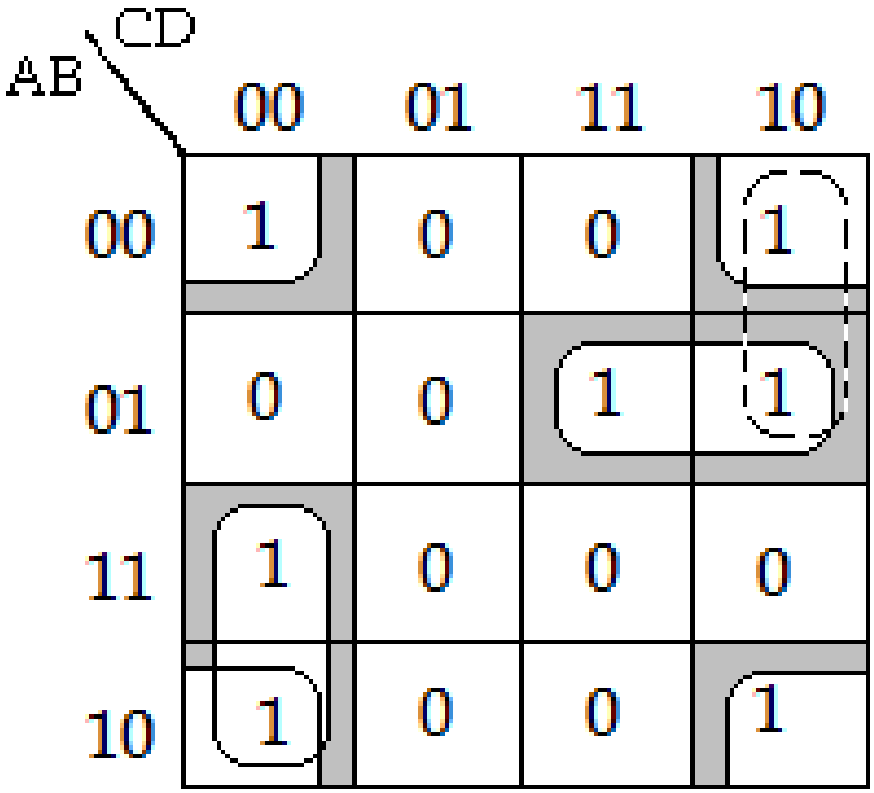
F (A, B, C, D) = Σm (0, 2, 6, 7, 8, 10, 12).



F= B'D'+ A'BC+ AC'D'

K-map Implementation and grouping

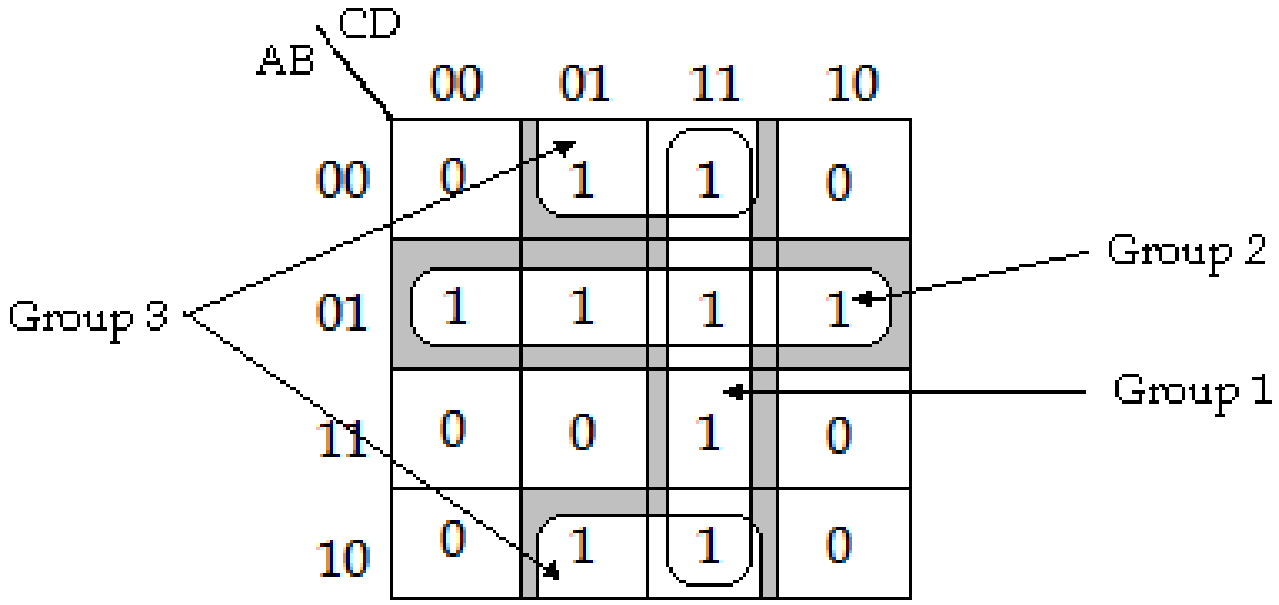
- Hazard- free realization



F= B'D'+ A'BC+ AC'D'+ A'CD'

Design a hazard-free circuit to implement the following function.

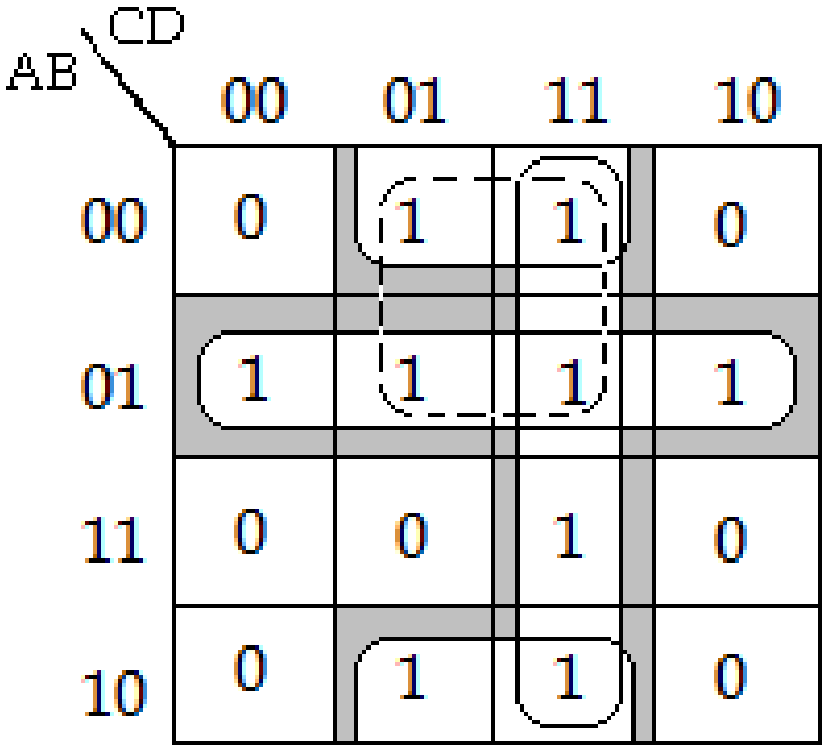
F (A, B, C, D) = Σm (1, 3, 4, 5, 6, 7, 9, 11, 15).



F= CD+ A'B+ B'D

K-map Implementation and grouping

- Hazard- free realization



F= CD+ A'B+ B'D+ A'D

End of Unit IV