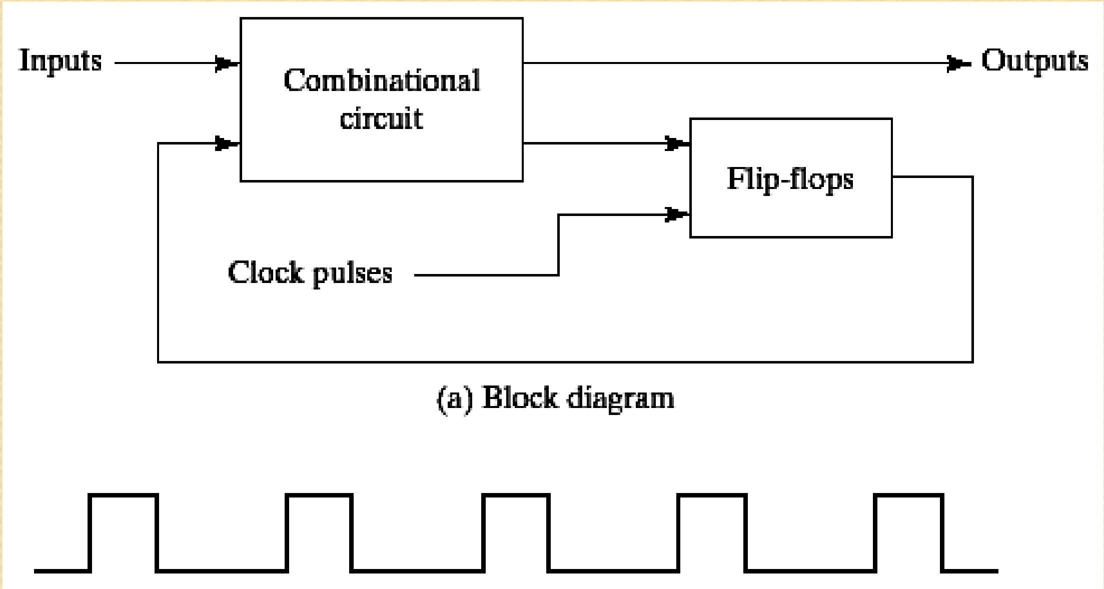
# Asynchronous Sequential Circuits

# Synchronous Sequential Circuits

- The change of internal state occurs in response to the synchronized clock pulses.
- The memory elements are flip-flops.



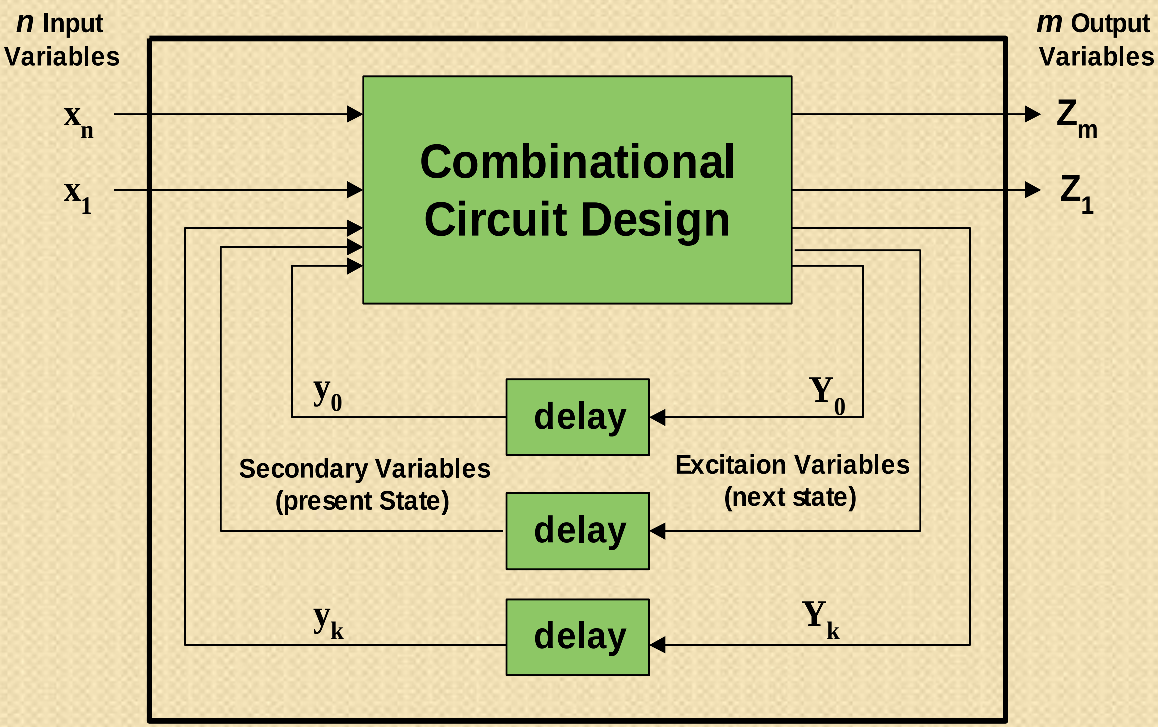

(a) Block diagram

# Asynchronous Sequential Circuits

**Asynchronous sequential circuits**

- 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

-The memory elements are either unclocked FF's or time-delay elements.
-The design of these circuits is more difficult than the design of synchronous circuits due to the timing problem.



# Why Asynchronous Circuits?

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

2-Used when the input signals change independently of the clock pulses.

3- Simplify the circuit in the small independent circuits.

4- Used to communicate two circuits each have its own clock.

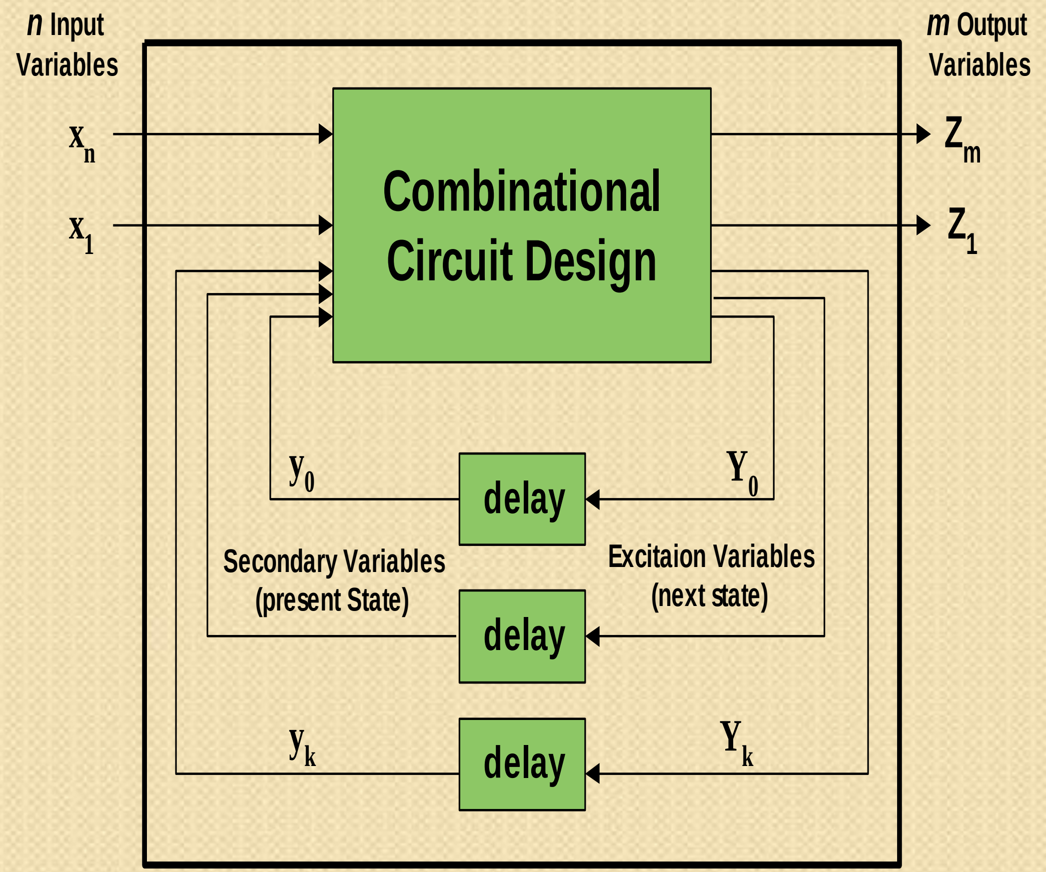
# Asynchronous Circuits

- The delay elements provide short-term memory for the sequential circuits.

- Present state variables [y1..yk] are called secondary variables

- Next state variables [Y1..Yk] are called excitation variables.

- When an input variable changes, it takes a certain time to propagate through the combinational circuit to change Y, and then Y takes a certain time to propagate through the delay element to become a new state.



# Asynchronous Circuits

- The circuit reaches a steady-state condition when $y_i = Y_i$ for $i=1,2,\ldots, K$.

- Stable System:

  for a given value of input variables, the system is stable if the circuit reaches a steady state condition.

- Fundamental-mode operation:

  this mode assumes that the one input signal changes at a time and only when the circuit is in stable condition.

- The time between two input changes must be longer than the time it takes the circuit to reach a stable state.

# Analysis Procedure

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

**Transition Table**

**Flow Table**

**Stability Consideration**

# Transition Table

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

1. Determine all feedback loops in the circuits

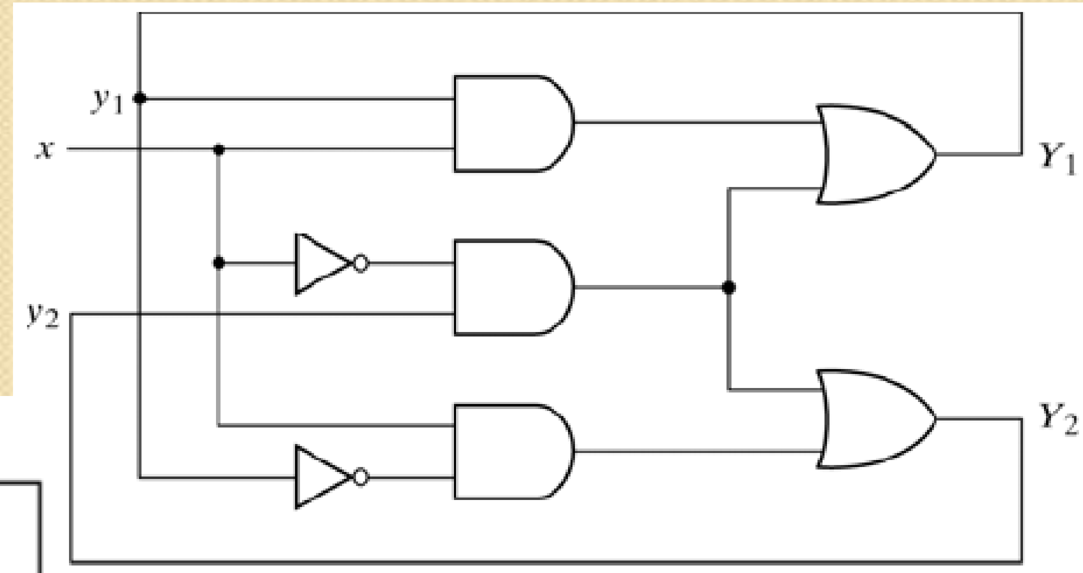
2. Mark the input (yi) and output (Yi) of each feedback loop

3. Derive the Boolean functions of all Y's

4. Plot each Y function in a map and combine all maps into one table

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

# Transition Table




| $y_1y_2$ \ $x$ | 0 | 1 |
|---|---|---|
| 00 | 0 | 0 |
| 01 | 1 | 0 |
| 11 | 1 | 1 |
| 10 | 0 | 1 |

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

| $y_1y_2$ \ $x$ | 0 | 1 |
|---|---|---|
| 00 | 0 | 1 |
| 01 | 1 | 1 |
| 11 | 1 | 0 |
| 10 | 0 | 0 |

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

Y1 = xy1 + x’y2
Y2 = xy’1 + x’y2

# Transition Table

-If y=00 and x= 0 ➡ Y ==00 (Stable)

-If x changes from 0 to 1 while y=00, the circuit changes Y to 01 which is temporary unstable condition (Y != y)

-As soon as the signal propagates to make Y = 01, the feedback path causes a change in y to 01. (transition form the first row to the second row)

-If the input repeatedly alternates between 0 and 1, the circuit will repeat the sequence of states

| $y_1 y_2$ | $x$ = 0 | $x$ = 1 |
|---|---|---|
| 00 | 00 | 01 |
| 01 | 11 | 01 |
| 11 | 11 | 10 |
| 10 | 00 | 10 |

*00 → 01 → 11 → 10*

# Transition Table

In an asynchronous sequential circuit, the internal state can change immediately after a change in the input.

It is sometimes convenient to combine the internal state with input

value together and call it the **Total State of the circuit.**

(Total state = Internal state + Inputs)

In the last example , the circuit has

- 4 **stable total states: (y1y2x= 000, 011, 110, and 101)**
- 4 **unstable total states: (y1y2x= 001,010,111, and 100)**

# Flow Table

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

| | $x$ = 0 | $x$ = 1 |
|---|---|---|
| $a$ | $a$ | $b$ |
| $b$ | $c$ | $b$ |
| $c$ | $c$ | $d$ |
| $d$ | $a$ | $d$ |

(a) Four states with one input

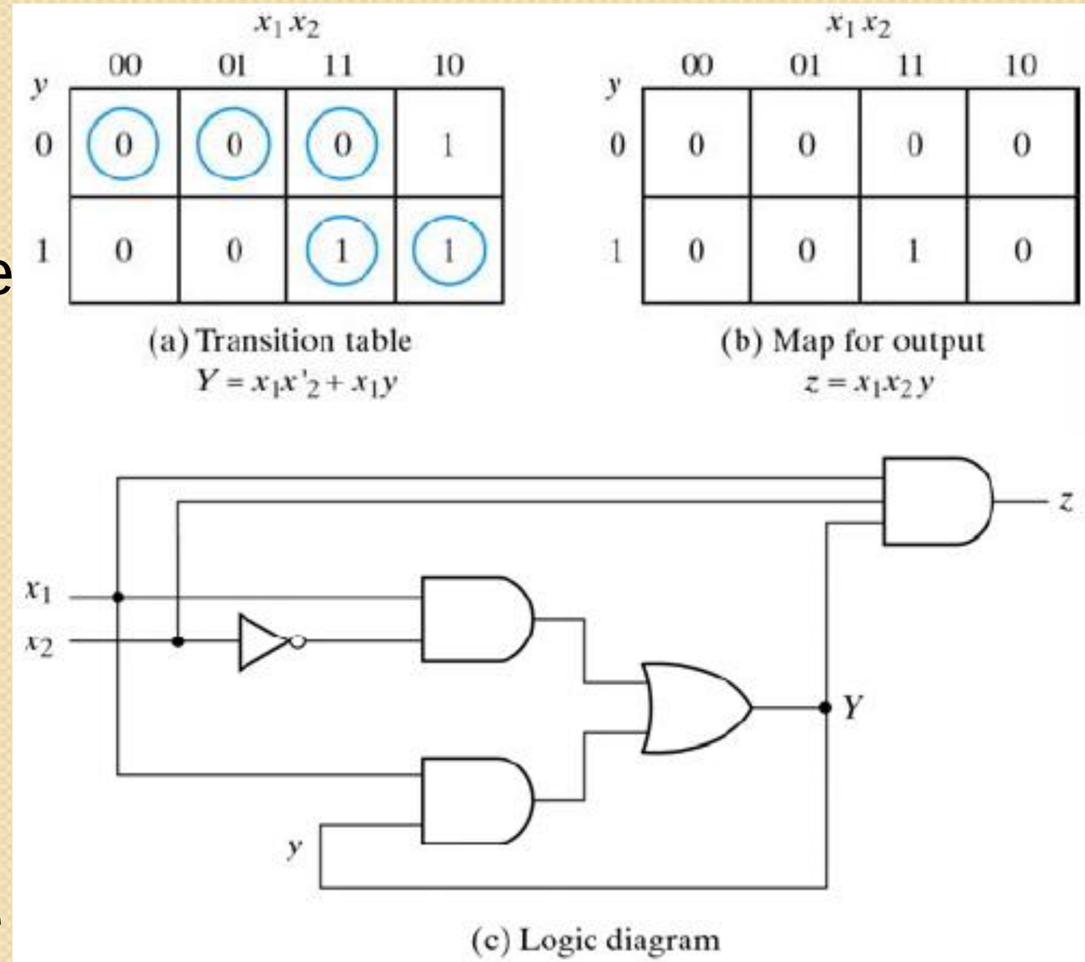
| | $x_1x_2$ = 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| $a$ | $a$, 0 | $a$, 0 | $a$, 0 | $b$, 0 |
| $b$ | $a$, 0 | $a$, 0 | $b$, 1 | $b$, 0 |

(b) Two states with two inputs and one output

# Flow Table

- In order to obtain the circuit described by a flow table, it is necessary to convert the flow table into a transition table from which we can derive the logic diagram .

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



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

(b) Map for output $z = x_1x_2y$

(c) Logic diagram

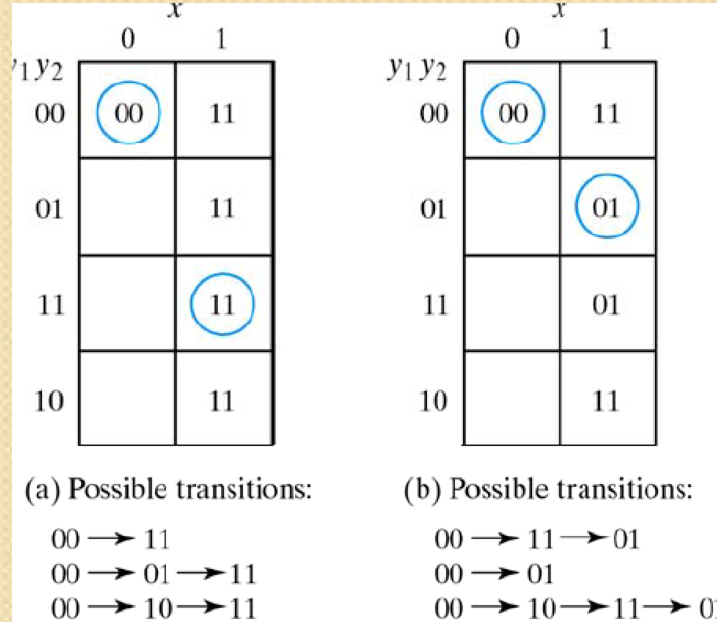
# Race condition

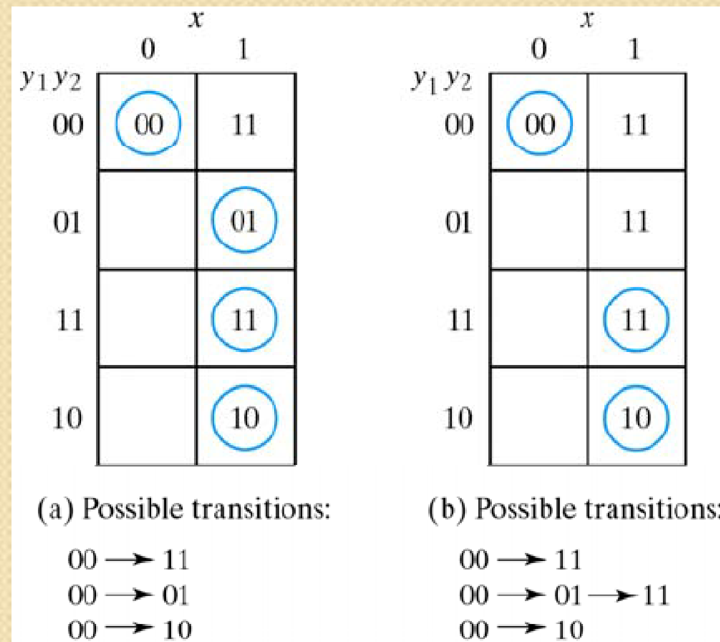
Two or more binary state variables will change value when one input variable changes.

Cannot predict state sequence if unequal delay is encountered.

**<u>Non-critical race</u>**: The final stable state does not depend on the change order of state variables

**<u>Critical race</u>**: The change order of state variables will result in different stable states Should be avoided !!



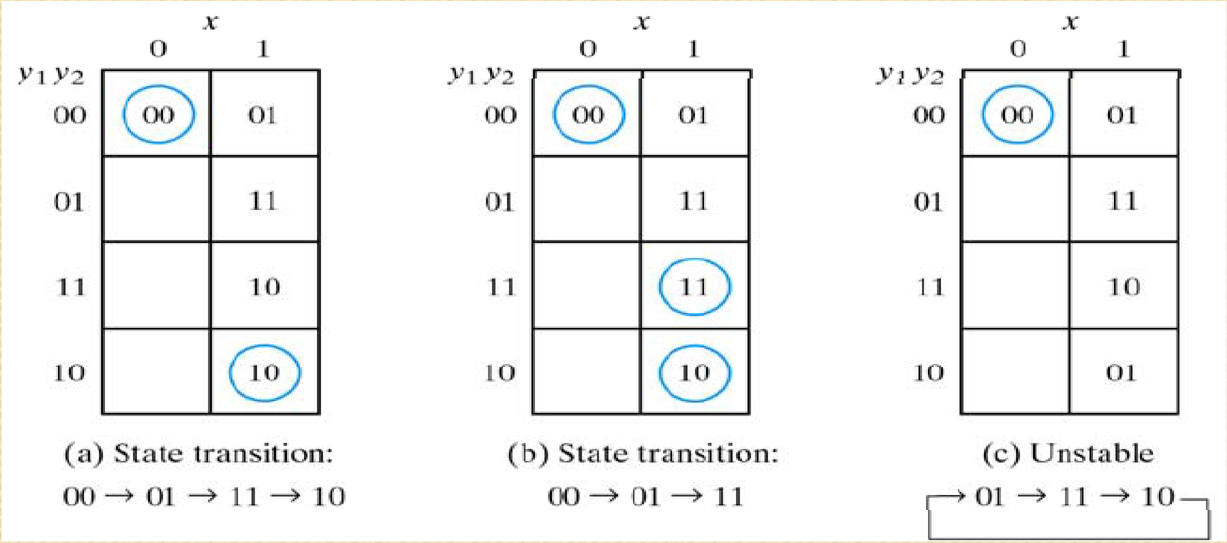



# Race Solution

It can be solved by making a proper binary assignment to the state variables.

The state variables must be assigned binary numbers in such a way that only one state variable can change at any one time when a state transition occurs in the flow table.

It will be discussed later.



# Stability Check

Asynchronous sequential circuits may oscillate between unstable states due to the feedback

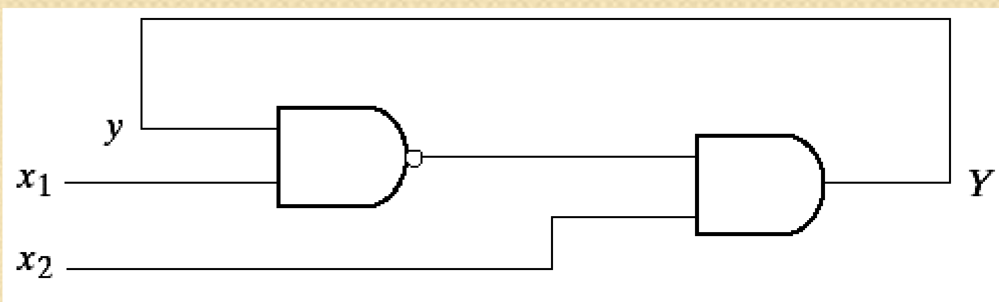
-Must check for stability to ensure proper operations

Can be easily checked from the transition table

-Any column has no stable states → unstable

-Ex: when x1x2=11 in Fig. 9-9(b), Y and y are never the same

Y = x'1x2 + x2y'




(a) Logic diagram

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

(b) Transition table

# Latches in Asynchronous Circuits

-The traditional configuration of asynchronous circuits is using one or more feedback loops

- No real delay elements.

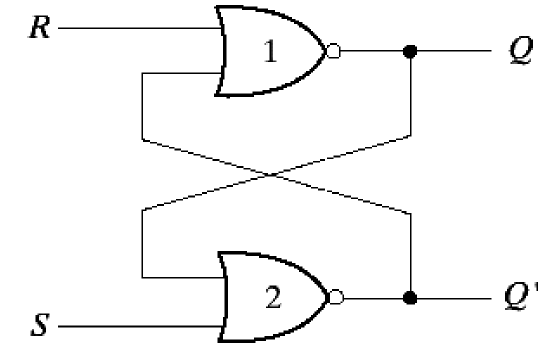
-It is more convenient to employ the SR latch as a memory element in asynchronous circuits

- Produce an orderly pattern in the logic diagram with the memory elements clearly visible.

-SR latch is also an asynchronous circuit

- Will be analyzed first using the method for asynchronous circuits.

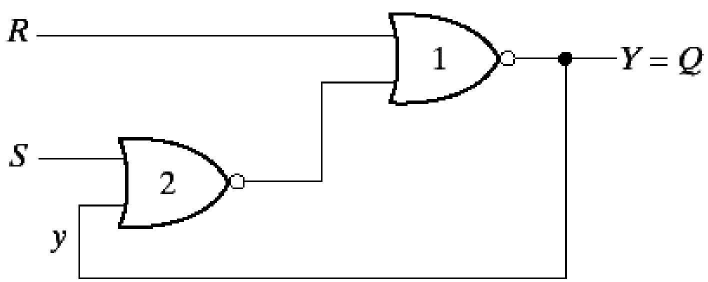
# SR Latch with NOR Gates




(a) Crossed-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 |

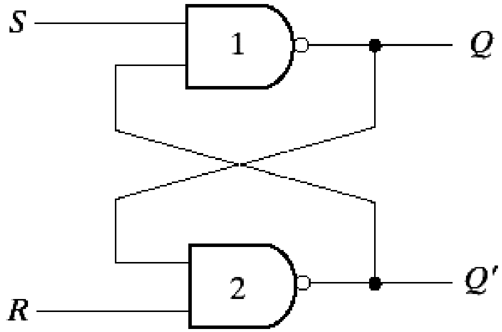
$Y = SR' + R'y$

$Y = S + R'y$ when $SR = 0$ ⇒ *** should be carefully** checked first

(d) Transition table

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

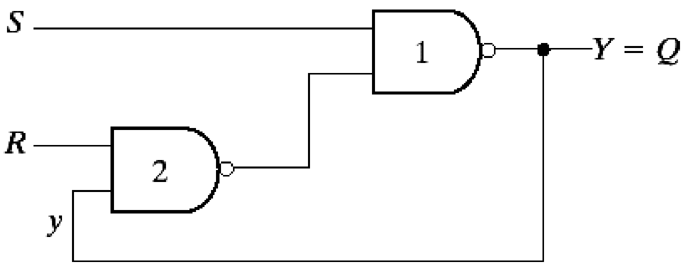
# SR Latch with NAND Gates




(a) Crossed-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 |

$Y = S' + Ry$ when $S'R' = 0$

(d) Transition table

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

* **should be carefully** checked first

# Analysis Procedure

**Analysis Procedure for NOR latch based asynchronous circuit**

(i) Label each latch o/p with $Y_i$ and feed back path with $y_i$

(ii) Derive Boolean functions for $S_i$ and $R_i$

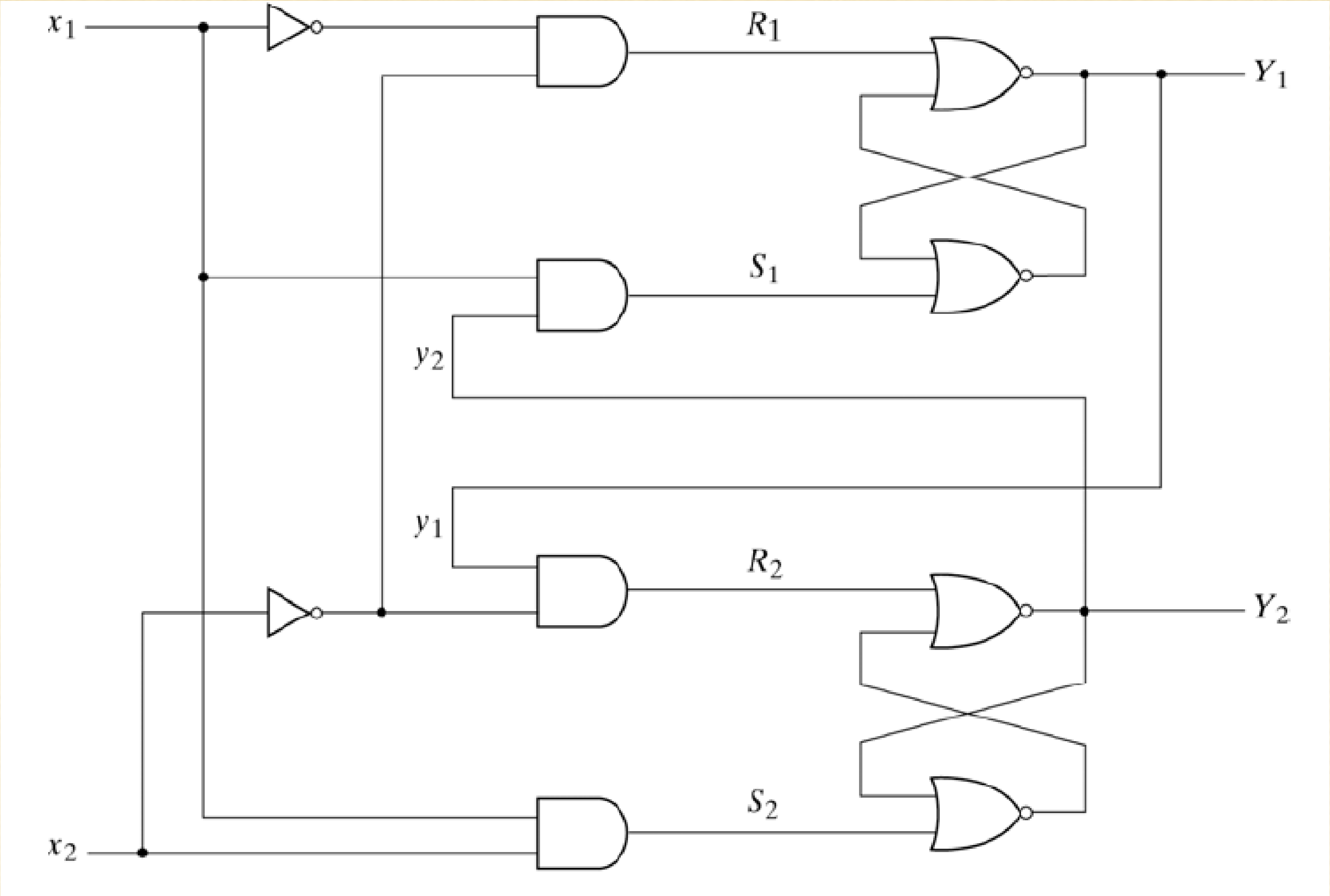
(iii) Check $SR = 0$ for each NOR latch

(iv) Evaluate $Y = S + R'y$ for each latch

(v) Construct the transition table

(vi) Circle all stable states

# Analysis Example



# Analysis Example

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

1. Label each latch output with Yi and its external feedback path with yi for i=1,2,…,k

2. Derive the Boolean functions for the Si and Ri inputs in each latch.

$$S_1 = x_1 y_2 \qquad S_2 = x_1 x_2$$

$$R_1 = x_1' x_2' \qquad R_2 = x_2' y_1$$

# Analysis Example

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

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

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

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

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

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

# Analysis Example

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 result is then the transition table.

- The transition table shows that the circuit is **stable**
- Race Conditions: there is a **critical race** condition when the circuit is initially in total state y1y2x1x2 = 1101 and x2 changes from 1 to 0.

-The circuit should go to the total state 0000.

-If Y1 changes to 0 before Y2, the circuit goes to total state 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 |

Transition Table

# Implementation Procedure

**Procedure to implement an asynchronous sequential circuits with SR latches:**

1. Given a transition table that specifies the excitation function Y = Y1Y2…Yk, derive a pair of maps for each Si and Ri using the latch excitation table
2. Derive the Boolean functions for each Si and Ri (do not to make Si and Ri 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 Procedure

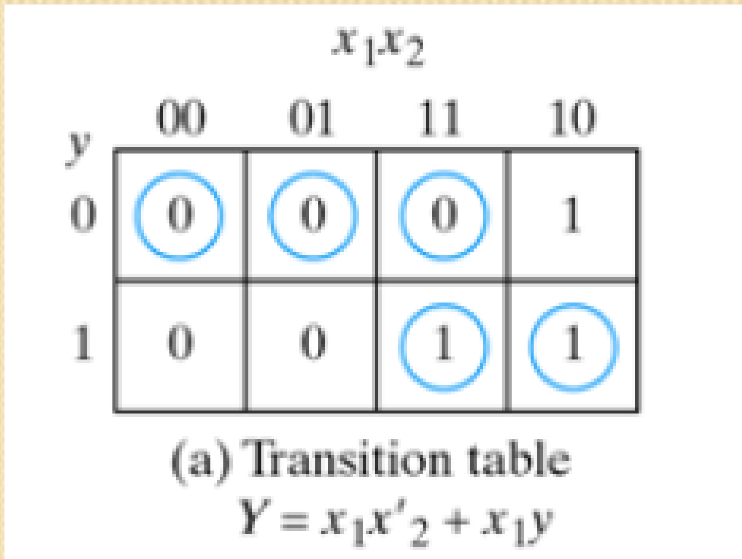
**Latch Excitation Table**

- During the implementation process, the transition table of the circuit is available and we wish to find the values of S and R .
- Excitation table: Lists the required inputs S and R for each of the possible transition from y to Y.

| $y$ | $Y$ | $S$ | $R$ |
|---|---|---|---|
| 0 | 0 | 0 | $X$ |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | $X$ | 1 |

# Implementation Example

- Given a transition table that specifies the excitation function Y=Y1Y2…Yk, then the general procedure for implementing a circuit with SR latches can be summarized as follows:



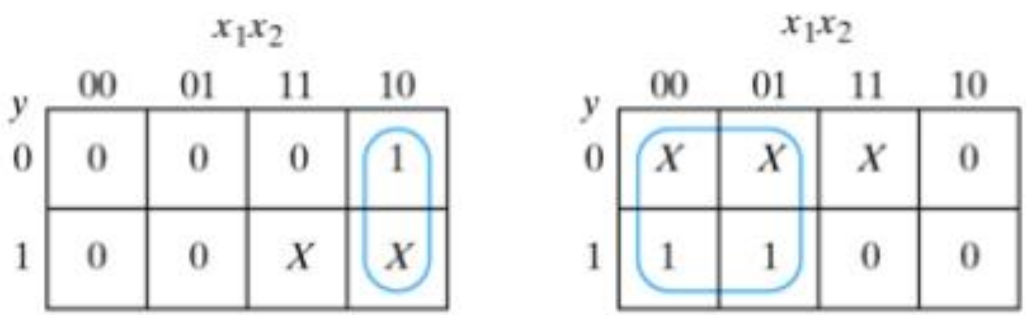
| $y \backslash x_1x_2$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 |

(a) Transition table

$Y = x_1x'_2 + x_1y$

# Implementation Example

1. Derive a pair of maps for Si and Ri for each I = 1, 2,…,k. (This is done by using the latch excitation table)

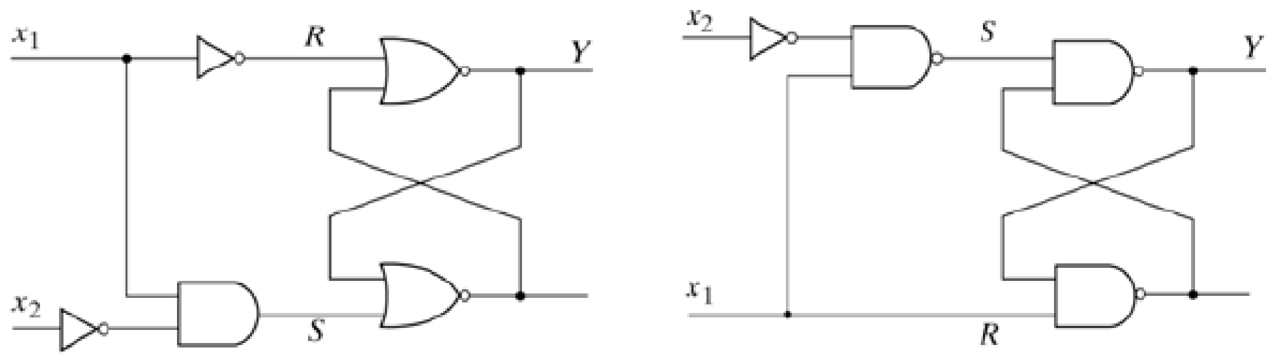



(c) Map for $S = x_1x'_2$ (d) Map for $R = x'_1$

# Implementation Example

2. Draw the logic diagram, using k latches together with the gates required to generate the S and R Boolean functions obtained in step1 (for NAND latches, use the complemented values)

| | NOR Latch | NAND Latch |
|---|---|---|
| S= | X1X'2 | (X1X'2)' |
| R= | X'1 | X1 |




(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

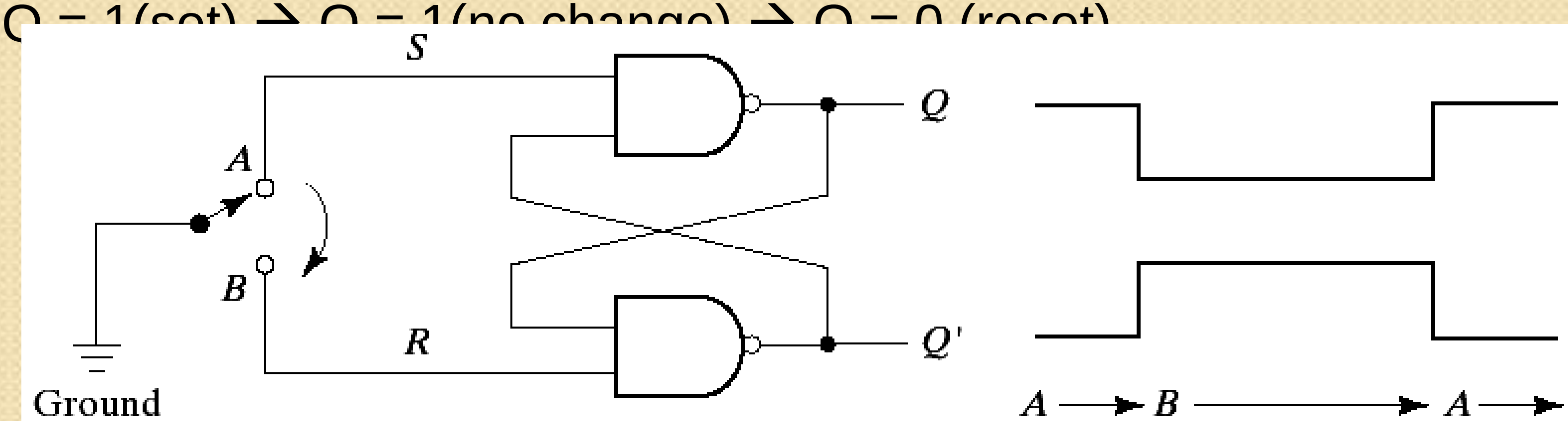
-It may vibrate or bounce several times before going to a final rest

-Cause the signal to oscillate between 1 and 0

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)



# Design procedure

(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 Example:

Problem Statement:

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

# Design Example:

## 1-Primitive Flow Table

- A primitive flow table is a flow table with only one stable total state (internal state + input) in each row.
- In order to form the primitive flow table , we first form a table with all possible total states.

| | Input | | 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 states a or d |
| d | 1 | 0 | 0 | After state c |
| e | 1 | 0 | 1 | After states b or f |
| f | 0 | 0 | 1 | After state e |

# Design Example:

## 1-Primitive Flow Table

First, we fill in one square in each row belonging to the stable state in that row.

Next we note that both inputs are not allowed to change at the same time, we enter dash marks in each row that differs in two or more variables from the input variables associated with the stable state.

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

All outputs associated with unstable states are marked with a dash to

| | 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 , – |

# Design Example:

## 2-Reduction of the Primitive Flow Table

Two or more rows can be merged into one row if there are non-conflicting states and outputs in every columns.

After merged into one row:

Don’t care entries are overwritten

Stable states and output values are included

A common symbol is given to the merged row

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

| | 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

| | 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 |

| | 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)

# Design Example:

## 3-Transition Table and Logic Diagram

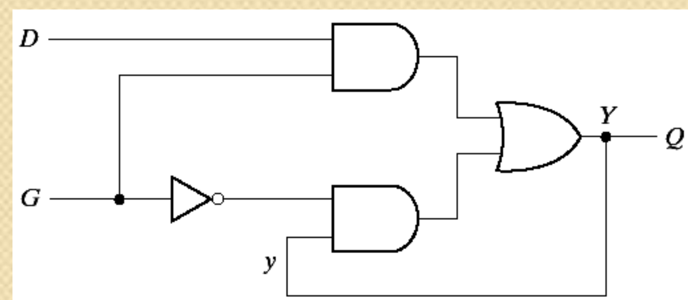
- In order to obtain the circuit described by the reduced flow table, it is necessary to assign a distinct binary value to each state.
- This converts the flow table to a transition table.
- A binary state assignment must be made to ensure that the circuit will be free of critical race. (This problem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

| $y$ \ $DG$ | 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) $Q = Y$



# Design Example:

## Implementation with SR Latch

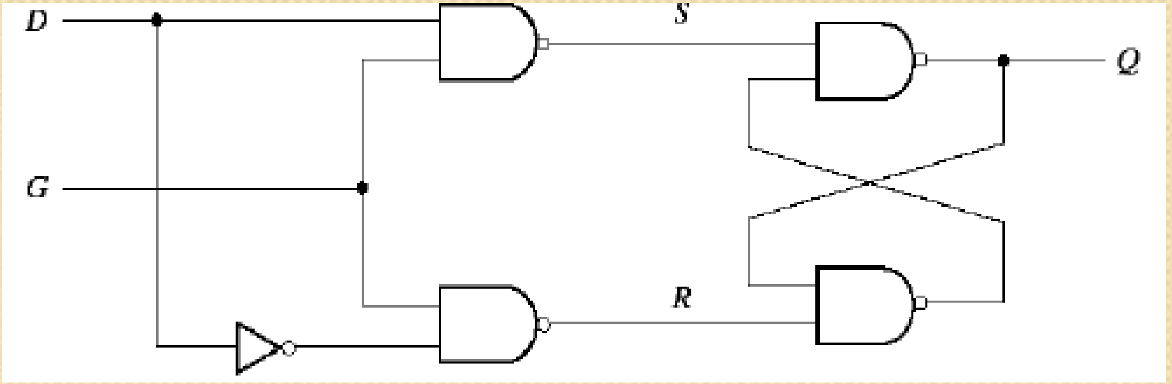
| $y$ \ $DG$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 |
| 1 | $X$ | 0 | $X$ | $X$ |

(a) $S = DG$

| $y$ \ $DG$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | $X$ | $X$ | 0 | $X$ |
| 1 | 0 | 1 | 0 | 0 |

$R = D'G$

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



# Design Example:

## 4- Assigning Outputs to Unstable States

- While the stable states in a flow table have specific output values associated with them, the unstable states have unspecified output entries designated by a dash.

These unspecified output values must be chosen so that no momentary false outputs occur when the circuit switches between stable states.

---

*If the two stable states have the save output value, then an unstable states that is a transient state between them must have the same output.*

*If an output variable is to change as a result of a state change, then this variable is assigned a don’t care condition.*

# Design Example:

## 4- Assigning Outputs to Unstable States

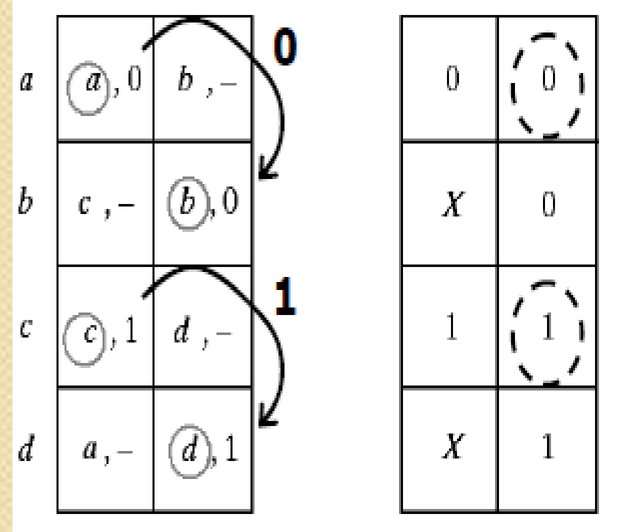
Ex:

- If a changes to b, the two stable states have the same output value =0

the transient unstable state b in the first row must have the same output value = 0

- If b changes to c, the two stable states have different output values

the transient unstable state c in the second row is assigned a don't care condition




(a) Flow table (b) Output assignment

# Reduction of States and Flow Tables

**Implication Table**

**Merging of the Flow Table**

**Compatible Pairs**

**Maximal Compatibles**

**Closed Covering Condition**

# Implication Table

**<u>Equivalent States</u>**: Two states are equivalent if, for each possible input, they give exactly the same output and go to the same next states or to equivalent next states.

→Equivalent states can be combined into one sate in the state table.

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**.

**<u>Implication Table:</u>** It is a chart that consists of squares, one for every possible pair of states.

# Implication Table (Example):

1. Place a cross in any square corresponding to a pair whose outputs are not equal
2. Enter in the remaining squares the pairs of states that are implied by the pair of states representing the squares. (Start form the top square in the left column and going down and then proceeding with the next column to the right).
3. Make successive passes through the table to determine whether any additional squares should be marked with a ‘x’.

**State Table to Be Reduced**

| Present State | Next State $x = 0$ | Next State $x = 1$ | Output $x = 0$ | Output $x = 1$ |
|---|---|---|---|---|
| $a$ | $d$ | $b$ | 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 |

| | | | | | | |
|---|---|---|---|---|---|---|
| $b$ | $d, e$ ✓ | | | | | |
| $c$ | × | × | | | | |
| $d$ | × | × | × | | | |
| $e$ | × | × | × | ✓ | | |
| $f$ | $c, d$ × | $c, e$ × $a, b$ | × | × | × | |
| $g$ | × | × | × | $d, e$ ✓ | $d, e$ ✓ | × |
| | $a$ | $b$ | $c$ | $d$ | $e$ | $f$ |

# Implication Table (Example):

Its clear that (e,d) are equivalent. And this leads (a,b) and (e,g) to be equivalent too.

Finally we have [(a,b) , c , (e,d,g) , f ]→4 states.

So the original flow table can be rec

| | | | | | | |
|---|---|---|---|---|---|---|
| $b$ | $d, e$ ✓ | | | | | |
| $c$ | × | × | | | | |
| $d$ | × | × | × | | | |
| $e$ | × | × | × | ✓ | | |
| $f$ | $c, d$ × | $c, e$ × $a, b$ | × | × | × | |
| $g$ | × | × | × | $d, e$ ✓ | $d, e$ ✓ | × |
| | $a$ | $b$ | $c$ | $d$ | $e$ | $f$ |

| Present State | Next State | | Output | |
|---|---|---|---|---|
| | x=0 | x =1 | x=0 | x=1 |
| a | d | a | 0 | 0 |
| c | d | f | 0 | 1 |
| d | a | d | 1 | 0 |
| f | c | a | 0 | 0 |

# 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 compatible

# Compatible Pairs

Implication tables are used to find compatible states.

-We can adjust the dashes to fit any desired condition.

-Must have no conflict in the output values to be merged.

| | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| $a$ | $c$, – | $a$, 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

| | $a$ | $b$ | $c$ | $d$ | $e$ |
|---|---|---|---|---|---|
| $b$ | ✓ | | | | |
| $c$ | ✓ | $d, e$ × | | | |
| $d$ | ✓ | $d, e$ × | ✓ | | |
| $e$ | $c, f$ × | ✓ | $d, e$ × <br> $c, f$ × | × | |
| $f$ | $c, f$ × | ✓ | × | $d, e$ × <br> $c, f$ × | ✓ |

(b) Implication table

The compatible pairs are :

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

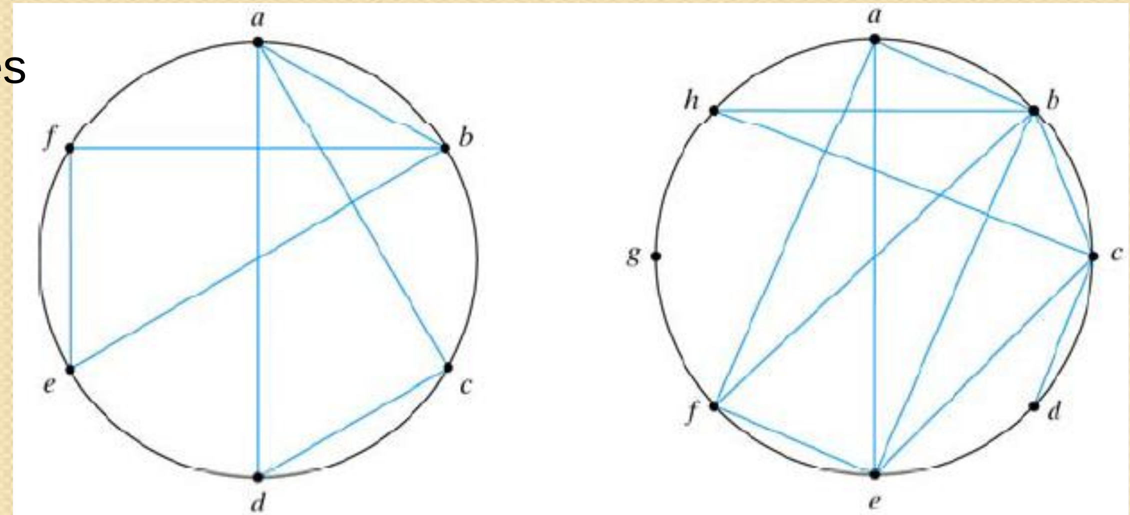
# Maximal Compatibles

- 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.




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

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

# Closed Covering Condition

- The condition that must be satisfied for row merging is that the set of chosen compatibles must:

1. Cover all states.
2. Be closed: ( the closure condition is satisfied if there are no implied states or if the implied states are included within the set)

→In the last example, the maximal compatibles are (a , b) (a , c , d),    (b , e , f)

- if we remove (a , b), we get a set of two compatibles: (a , c , d) ,     (b , e , f)

-All the six states are included in this set.

-There are no impiled states for (a,c); (a,d);(c,d);(b,e);(b,f) and (e,f) [you can check the implication table] . the closer condition is satisfied

→***The original primitive flow table can be merged into two rows, one for each of the compatibles.***

# Closed Covering Condition (Example)

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

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

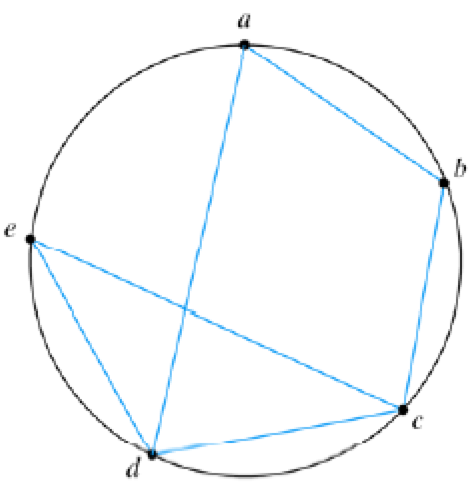
(c) Closure table

-All the 5 states are included in this set.

- The implied states for (a,b) are (b,c). But (b,c) are not include in the chosen set  This set is not closed.

-A set of compatibles that will satisfy the closed covering condition is ( a , d ) ( b , c ) ( c , d , e )

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



# 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

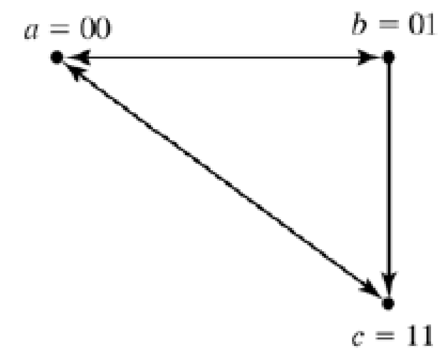
Adjacent info. are represented by a transition diagram

a and c are still not adjacent in such an assignment !!

-Impossible to make all states adjacent if only 3 states are used

| | $x_1x_2$ = 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

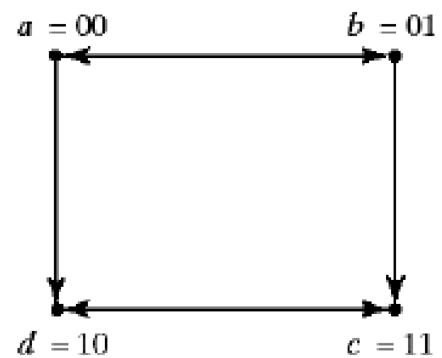
# 3-Row Flow-Table Example

A race-free assignment can be obtained if we add anextra 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

| | $x_1x_2$ 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 | – |

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

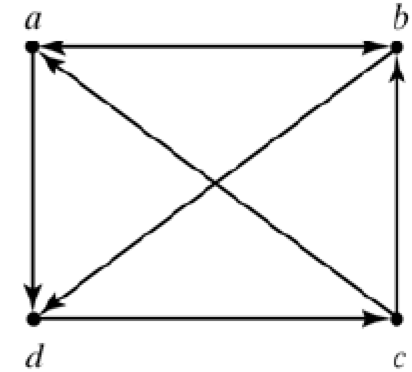
# 4-Row Flow-Table Example

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

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

| | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| *a* | *b* | *a* | *d* | *a* |
| *b* | *b* | *d* | *b* | *a* |
| *c* | *c* | *a* | *b* | *c* |
| *d* | *c* | *d* | *d* | *c* |

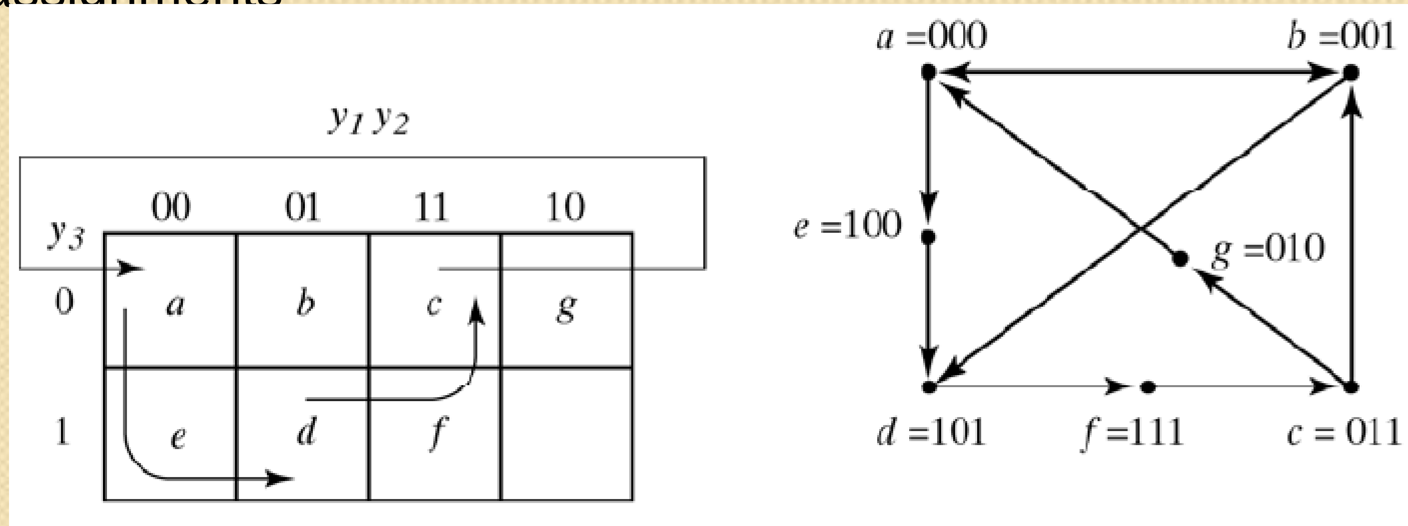
(a) Flow table




(b) Transition diagram

# 4-Row Flow-Table Example

- The following state assignment map is suitable for any 4-row flow table.
  - a, b, c, and d are the original states.
  - e, f, and g are extra states.
  - States placed in adjacent squares in the map will have adjacent assignments




(a) Binary assignment (b) Transition diagram

# 4-Row Flow-Table Example

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

| | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| *a* | *b* | *a* | *d* | *a* |
| *b* | *b* | *d* | *b* | *a* |
| *c* | *c* | *a* | *b* | *c* |
| *d* | *c* | *d* | *d* | *c* |

(a) Flow table

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

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

# 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

  Behaviors are still the same
- While choosing the next states, choose the adjacent one

| $y_1 \backslash y_2y_3$ | 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 delay.

- Hazards occur in in combinational and asynchronous circuits:
  - In combination circuits, they may cause a temporarily false output value.
  - In asynchronous circuits, they may result in a transition to a wrong stable state.

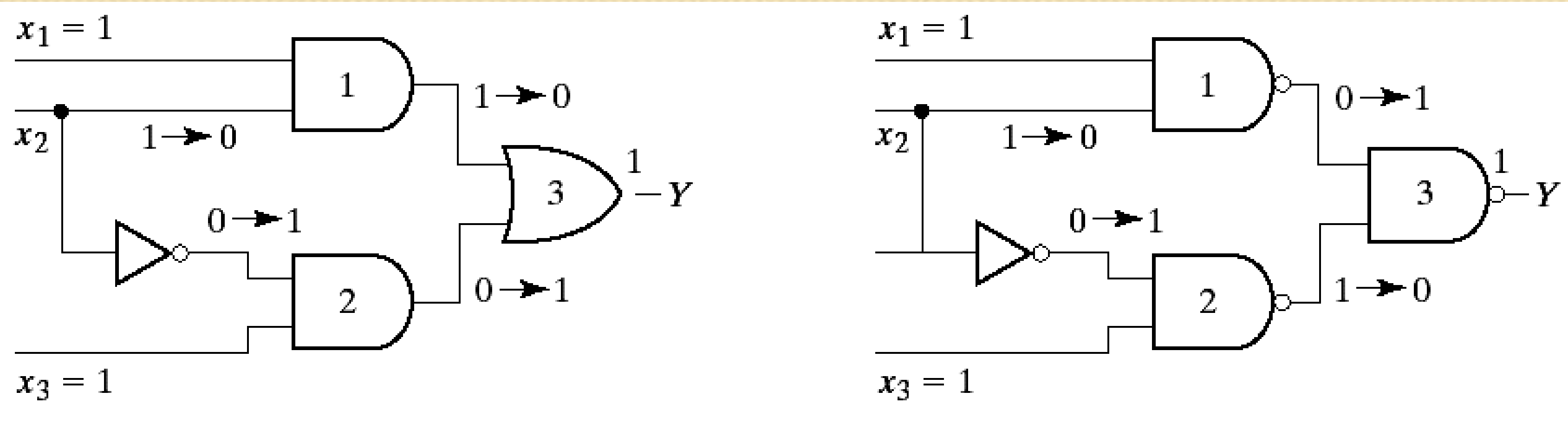
# Hazards

Static hazard: a momentary output change when no output change should occur

If implemented in sum of products:

-no static 1-hazard → no static 0-hazard or dynamic hazard

Two examples for static 1-hazard:

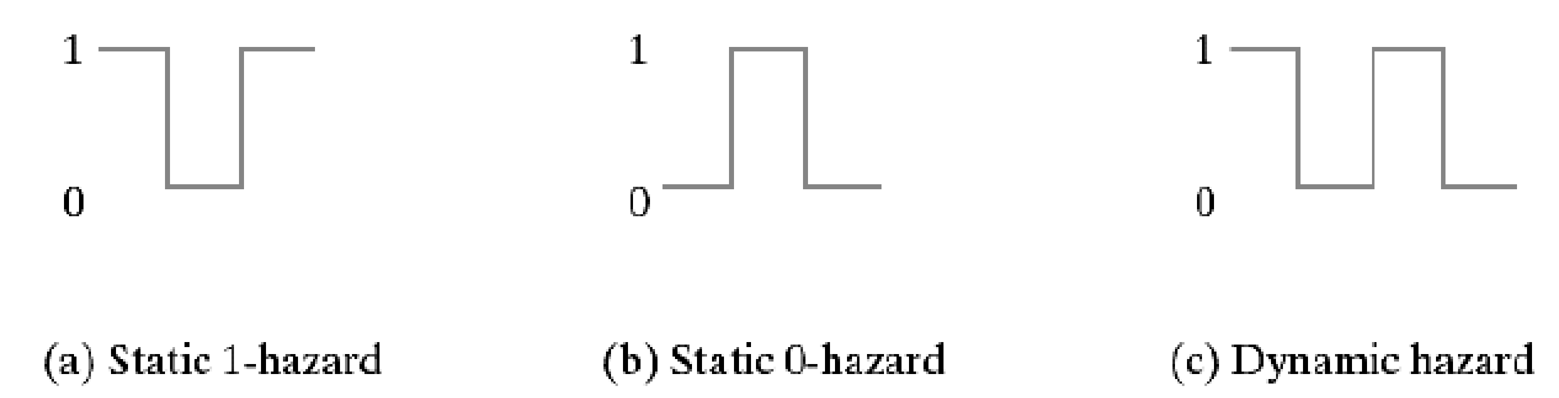



(a) AND-OR circuit (b) NAND circuit

# Hazards

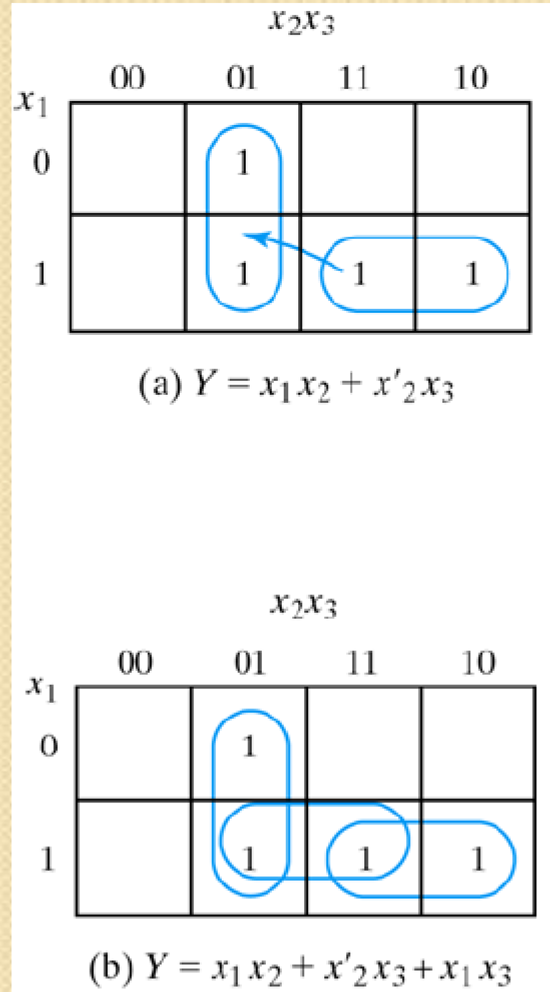
- The **dynamic hazard** causes the output to change two, three or four times when it should change from 1 to 0 or from 0 to 1.

- The occurrence of the hazard can be detected by inspecting the map of a particular circuit.



# Hazards Free Circuit

- The change in x2 from 1 to 0 moves the circuit from minterm 111 to minterm 101.
- The hazard exists because the change of input results in a different product term covering the two minterms.
- Whenever the circuit must move from one product term to another, there is a possibility of a momentary interval when neither term is equal to 1, giving rise to undesirable 0 output.
- The solution is to enclose the minterms with another product term that overlaps both groupings.

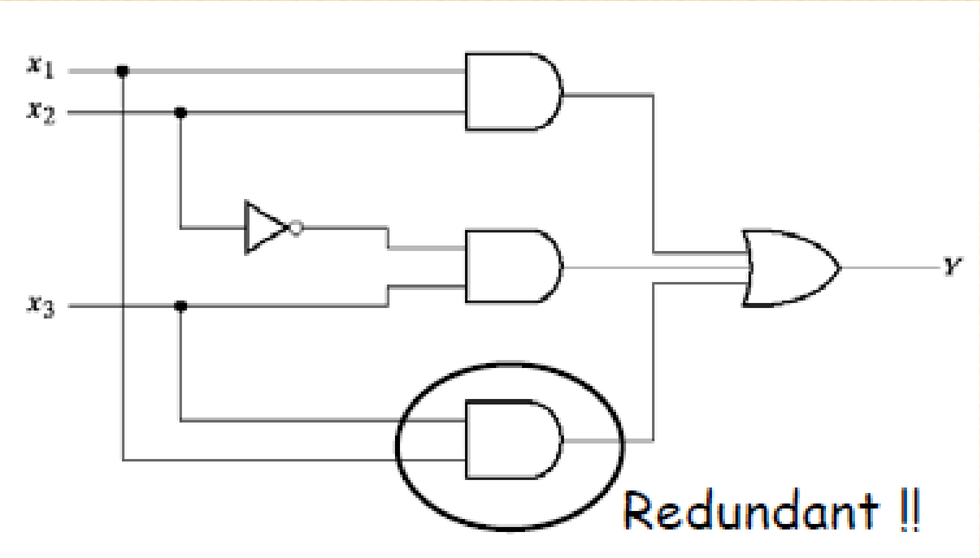



(a) $Y = x_1x_2 + x'_2x_3$



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

# Hazard Free Circuit



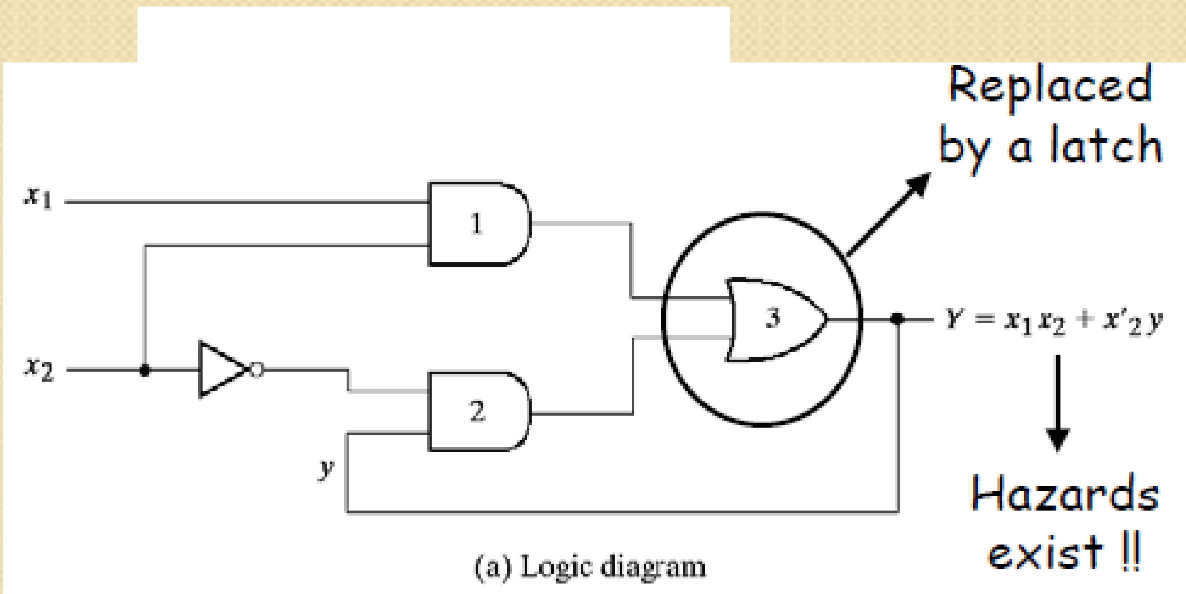

The removal of hazards requires the addition of redundant gates to the circuit.

# Remove Hazards with Latches

Implement the asynchronous circuit with SR latches can also remove static hazards

A momentary 0 has no effects to the S and R inputs of a NOR latch

A momentary 1 has no effects to the S and R inputs of a NAND latch



(a) Logic diagram

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

(b) Transition table

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

(c) Map for $Y$

# Example

- Consider a NAND SR-latch with the following Boolean functions for S and R

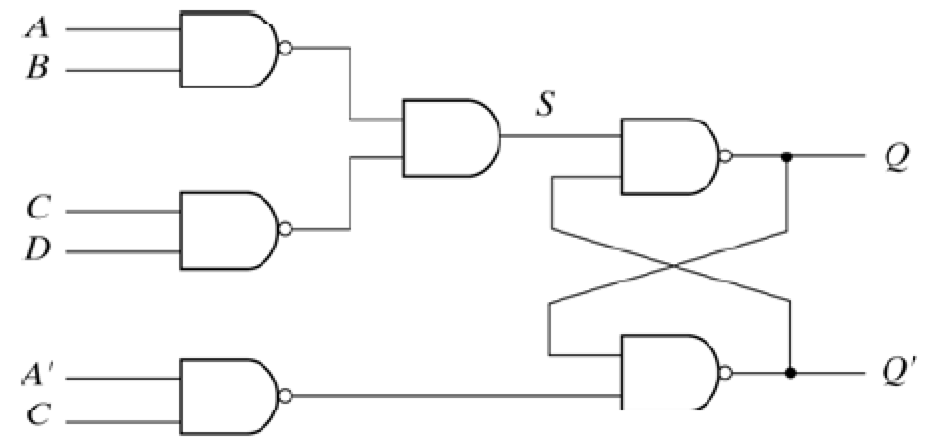
**S = AB + CD**

**R = A'C**

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

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

**R = (A'C)'**



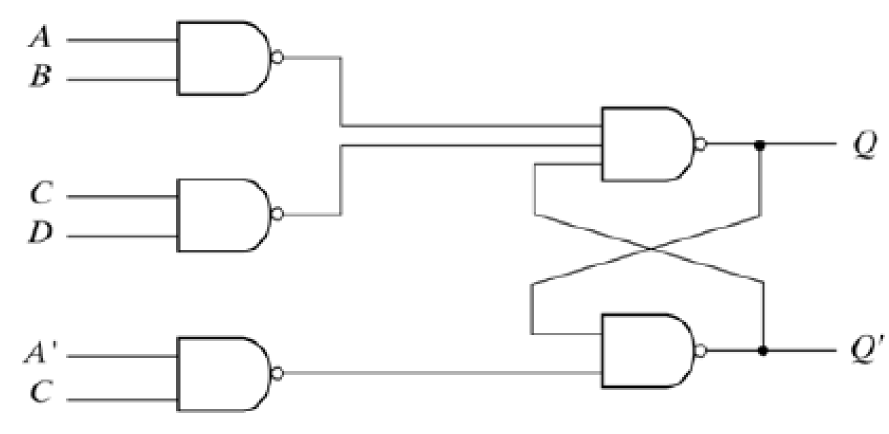

(a)

# Example

- The Boolean function for output is

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

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




(b)

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

# Essential Hazards

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

  **-** Each feedback path should be examined carefully !!

# Design Example

Recommended Design Procedure:

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

# Design Example

1) **Design Specifications:**

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

# Design Example

## 2) Primitive Flow Table

| | 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 |

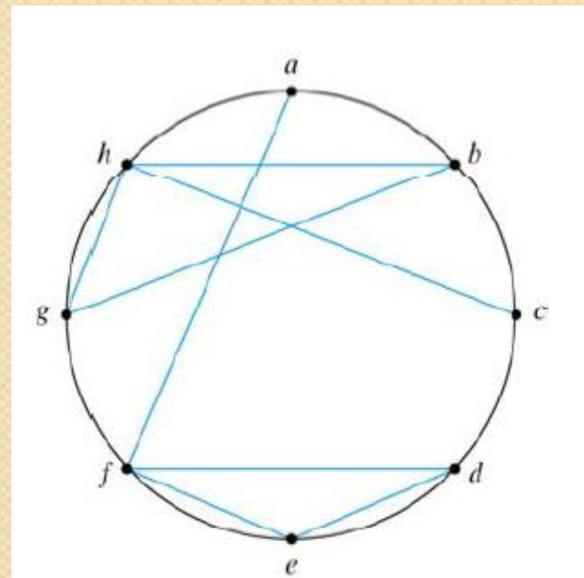
| | TC 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| *a* | –, – | *f*, – | *a*, 0 | *b*, – |
| *b* | *g*, – | –, – | *c*, – | *b*, 1 |
| *c* | –, – | *h*, – | *c*, 1 | *d*, – |
| *d* | *e*, – | –, – | *a*, – | *d*, 0 |
| *e* | *e*, 0 | *f*, – | –, – | *d*, – |
| *f* | *e*, – | *f*, 0 | *a*, – | –, – |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

# Design Example

## 3) Merging of the Flow Table

| | 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 × | × | ✓ |

Implication Table




Merger Diagram

→The maximal compatibles pairs are: (a , f) (b , g , h) (c , h) (d , e , f)

# Design Example

In this particular example, the minimal collection of compatibles is also the maximal compatibles set:

(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, 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 |

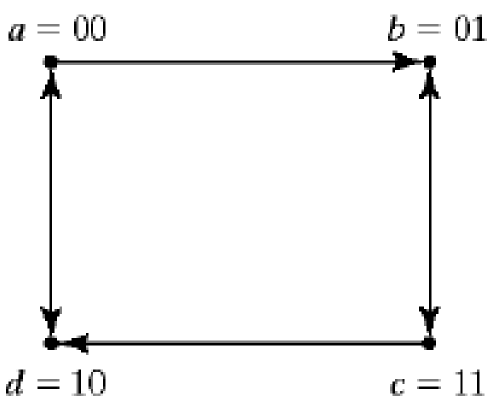
(b)

# Design Example

**4) State Assignment and Transition Table**

No diagonal lines in the transition diagram:

→No need to add extra states




| $y_1 y_2$ \ $TC$ | 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

| $y_1 y_2$ \ $TC$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | 0 | 0 | $X$ |
| 01 | 1 | 1 | 1 | 1 |
| 11 | 1 | 1 | 1 | $X$ |
| 10 | 0 | 0 | 0 | 0 |

(b) Output map $Q = y_2$

# Design Example

## 5) 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) $S_1 = y_2TC + y'_2T'C'$

| $y_1y_2$ \ $TC$ | 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) $R_1 = y_2T'C' + y'_2TC$

| $y_1y_2$ \ $TC$ | 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) $S_2 = y'_1TC'$

| $y_1y_2$ \ $TC$ | 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) $R_2 = y_1TC'$