**EEA051 - Digital Logic**
數位邏輯

# Chapter 9
# Asynchronous Sequential Logic

吳俊興
高雄大學 資訊工程學系

December 2004

## 9.1 Introduction

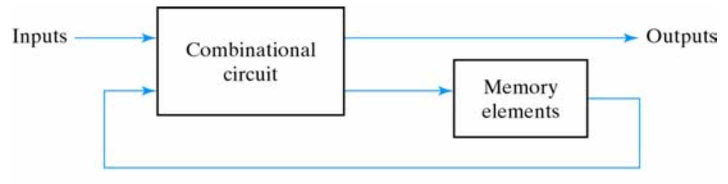


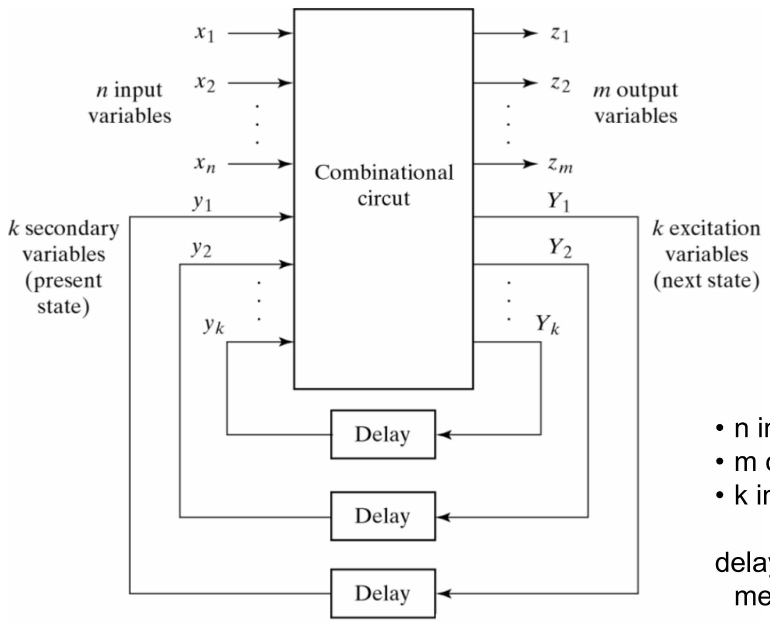
Fig. 5-1 Block Diagram of Sequential Circuit

Two major types of sequential circuits: depending on timing of their signals

- **Asynchronous sequential circuits**
  - The transition happens at ***any*** instant of time
  - Do not use clock pulses. Change of internal state occurs when there is a change in input variables
    - Instability problem: may become unstable at times
  - Storage elements work as time-delay device
    - May be regarded as a combinational circuit with feedback
- **Synchronous sequential circuits**
  - The transition happens at ***discrete*** instants of time
  - The circuit responds only to pulses on particular inputs
  - Storage elements are affected only with the arrival of each pulse

## Chapter 9 Asynchronous Sequential Logic

## Block Diagram of an Asynchronous Sequential Circuit



- n input variables
- m output variables
- k internal states

delay: like short-term memory

secondary variables and excitation variables

## Asynchronous Sequential Circuits

- Timing Problems
  - synchronous circuit: eliminated by triggering all flip-flops with the pulse edge
  - asynchronous circuit: change immediately after input changes
- Asynchronous Sequential Circuits
  - no clock pulse
  - difficult to design
  - delay elements: the propagation delay
  - must attain a stable state before the input is changed to a new value
- DO NOT use asynchronous sequential circuits unless it is absolutely necessary
  - e.g., in you exam

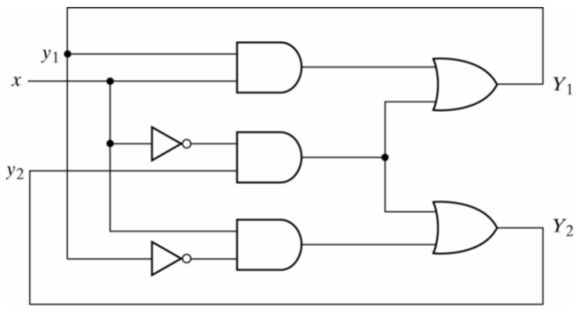
## 9-2 Analysis Procedure

- The procedure
  1. Determine all feedback loops
  2. Assign $Y_i$'s (excitation variables), $y_i$'s (secondary variables)
  3. Derive the ***Boolean functions*** of all $Y_i$'s
  4. Plot each Y function in ***a map***
     - the y variables for the rows
     - the external variable for the columns
  5. Combine all the maps into one ***transition table***
     - showing the value of $Y=Y_1Y_2\ldots Y_k$ inside each square
  6. Circle the stable states and derive the ***state table***
     - those values of Y that are equal to $y=y_1y_2\ldots y_k$ in the same row

Asynchronous sequential circuit (vs. sequential circuit)

- Total state of the circuit: combine internal state with input value
  - eg. Figure 9-3(c) has 4 stable total states: $y_1y_2x$=000, 011, 110, and 101, and 4 unstable total states: 001, 010, 111, and 100
- There usually is at least one stable state in each row

## Figure 9-2 Asynchronous Sequential Circuit Example



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

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

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

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

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

(c) Transition table

1. Excitation variables as outputs and secondary variables as inputs
   $Y_1=xy_1 + x'y_2$
   $Y_2=xy_1' + x'y_2$
2. Plot functions in a map
3. Combine all maps into a ***transition table***
   - stable state: $y=y_1y_2$ (circled)
4. Complete the state stable
   - check if unstable states will reach a stable state finally

**Table 9-1**
*State Table for the Circuit of Fig. 9-2*

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

## Flow Table

- A flow table
  - a state transition table with its internal state being symbolized with letters
  - Figure 9-4(a) is called ***a primitive flow table*** because it has only one stable state in each row
  - Figure 9-4(b): two states, a and b; two inputs, $x_1$ and $x_2$; and one output, z

| | $x$ = 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

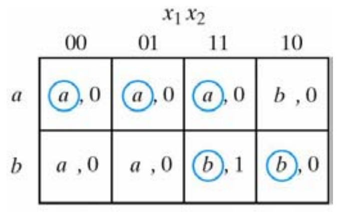
Figure 9-4. Examples of Flow Tables

Figure 9-4(b)

- If $x_1$=0, the circuit is in state a
- If $x_1$ goes to 1while $x_2$ is 0, the circuit goes to b
- With inputs $x_1x_2$=11, it may be in either in state a or state b, and output 0 or 1, respectively
- Maintain in state b if the inputs change from 10 to 11 and maintain in state a if the inputs changes from 01 to 11

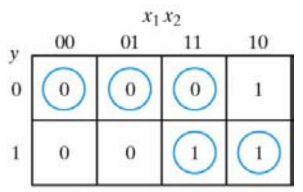
## Derivation of a Circuit Specified by Flow Table

- state assignment ⇒ state equation ⇒ logic diagram



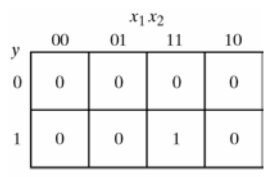
| | $x_1x_2$ 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| a | a, 0 | a, 0 | a, 0 | b, 0 |
| b | a, 0 | a, 0 | b, 1 | b, 0 |

(b) Two states with two inputs and one output



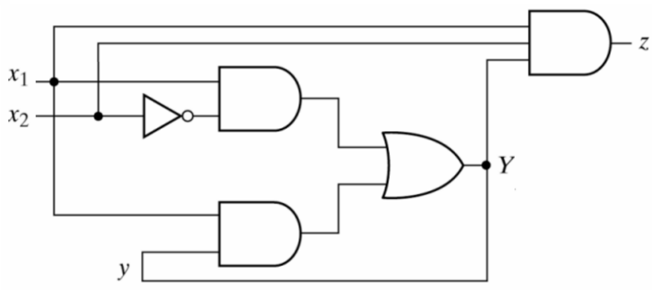
| y | $x_1x_2$ 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 |

(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
$z = x_1x_2y$



(c) Logic diagram

Figure 9-5

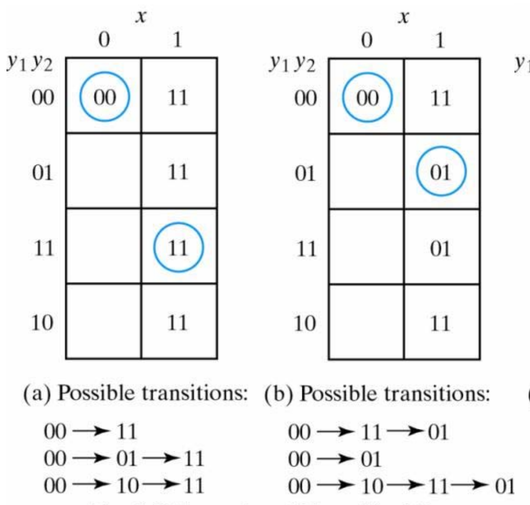
## Race Conditions

- Race condition
  - occur 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 state variables to change in an unpredictable manner
    - $y_1$, $y_2$, …, $y_i$ may change in unpredictable manner in response to a change in $x_1$
  - 00 → 11
    - 00 → 10 → 11 or 00 → 01 → 11
  - a noncritical race
    - if they reach the same final state
    - otherwise, a ***critical race***: end up in two or more different stable states

## Example of Race Conditions

Noncritical race

Critical race



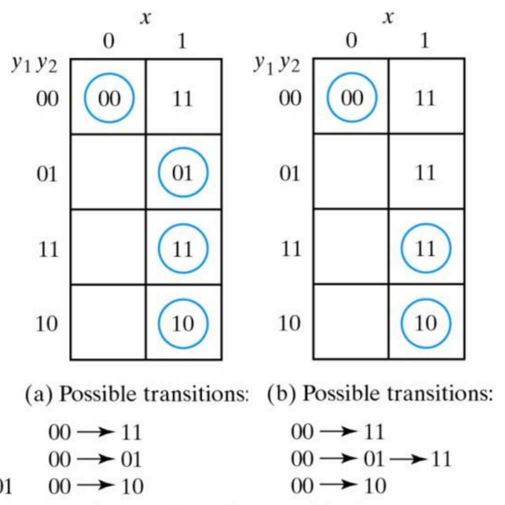
| $y_1y_2$ | $x$ 0 | 1 |
|---|---|---|
| 00 | 00 | 11 |
| 01 | | 11 |
| 11 | | 11 |
| 10 | | 11 |

(a) Possible transitions:
00 → 11
00 → 01 → 11
00 → 10 → 11

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

(b) Possible transitions:
00 → 11 → 01
00 → 01
00 → 10 → 11 → 01

Fig. 9-6 Examples of Noncritical Races



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

(a) Possible transitions:
00 → 11
00 → 01
00 → 10

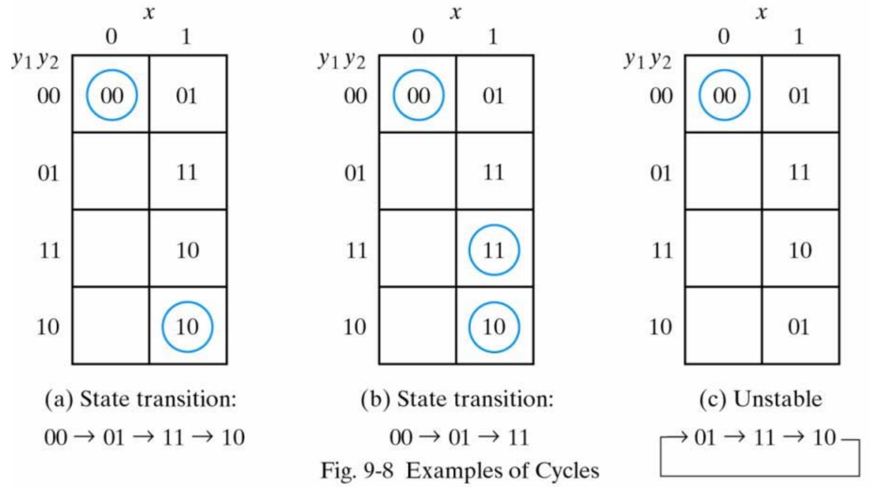
| $y_1y_2$ | $x$ 0 | 1 |
|---|---|---|
| 00 | 00 | 11 |
| 01 | | 11 |
| 11 | | 11 |
| 10 | | 10 |

(b) Possible transitions:
00 → 11
00 → 01 → 11
00 → 10

Fig. 9-7 Examples of Critical Races

## Figure 9-8 Examples of Cycles

- Races may be avoided
  - race-free assignment: only 1 state can change at any one time (Section 9-6)
  - Directing the circuit through inserting intermediate unstable states with a unique state-variable change
- A cycle: When a circuit goes through a unique sequence of unstable states



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

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

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

(b) State transition:
00 → 01 → 11

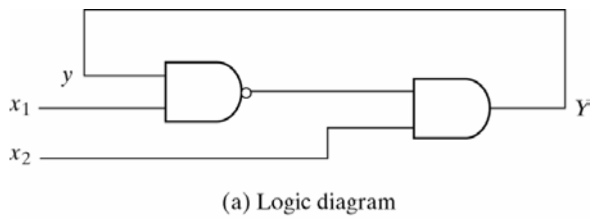
| $y_1y_2$ | $x$ 0 | 1 |
|---|---|---|
| 00 | 00 | 01 |
| 01 | | 11 |
| 11 | | 10 |
| 10 | | 01 |

(c) Unstable
→ 01 → 11 → 10 ⌐

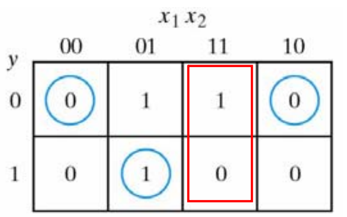
Fig. 9-8 Examples of Cycles

## Stability Considerations

- An unstable condition will cause the circuit to oscillate between unstable state
  - Care must be taken to ensure that the circuit does not become unstable
  - 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 |

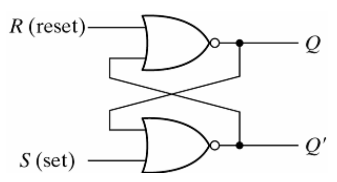
(b) Transition table

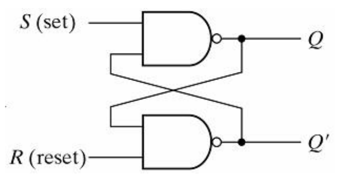
Fig. 9-9 Example of an Unstable Circuit

- Column 11 has no stable states: with input $x_1x_2$ fixed at 11, the values of Y and y are never the same
  - •State variable alternates between 0 and 1 indefinitely as long as input=11
- If each gate has a propagation delay of 5 ns, Y will be 0 for 10 ns and 1 for next 10 ns, resulting a square-wave waveform with 20 ns period, or 50Mhz

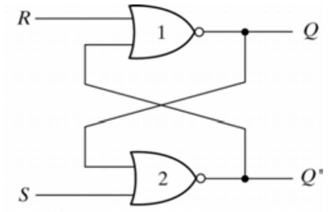
## 9-3 Circuits with Latches

- Asynchronous sequential circuits
  - were know and used before synchronous design
- SR Latch
  - the use of SR latches in asynchronous circuits produces a more orderly pattern
    - the memory elements clearly visible
    - reduce the circuit complexity
  - two cross-coupled NOR gates or NAND gates





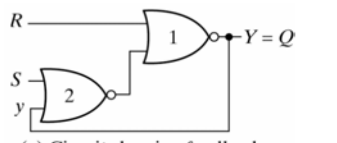
## SR Latch with Two Cross-coupled NOR Gates



(a) Crossed-coupled circuit

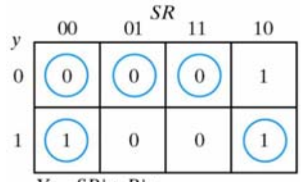
| S | R | Q | Q' | |
|---|---|---|---|---|
| 1 | 0 | 1 | 0 | |
| 0 | 0 | 1 | 0 | (After $SR = 10$) |
| 0 | 1 | 0 | 1 | |
| 0 | 0 | 0 | 1 | (After $SR = 01$) |
| 1 | 1 | 0 | 0 | |

(b) Truth table



(c) Circuit showing feedback

Fig. 9-10 *SR* Latch with NOR Gates



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

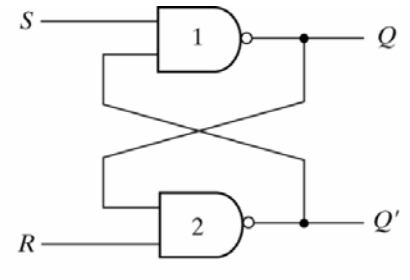
$Y = SR' + R'y$
$Y = S + R'y$ when $SR = 0$
(d) Transition table

–With SR=0
- SR' + SR = S(R'+R) = S
- Y = SR'+R'y = S + R'y

- Excitation variable:
  Y = ((S+y)'+R)' = (S+y)R' = SR'+R'y
- Derive the state transition table
  - With SR=10, output Q=Y=1
    - changing S to 0 ⇒ Q remains 1
  - With SR=01, output Q=Y=0
    - changing R to 0 ⇒ Q remains 0
  - With SR=11, Q=Q'=0
    - violate Q and Q' are the complement of each other
    - an unpredictable result when SR: 11 → 00
      - if S goes to 0 first, Q remains 0
      - if R goes to 0 first, Q goes to 1

•To analyze a circuit with an SR latch, first check the condition SR=0 holds at all times
•Then use the reduced excitation function Y=S+R'y to analyze the circuit
•If both S and R can be 1 at the same time, use the original excitation function

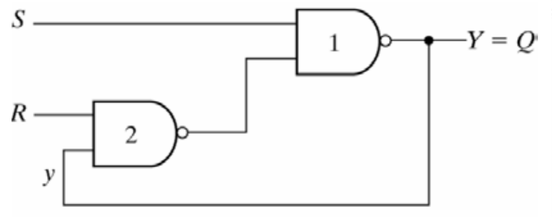
## S'R' Latch with NAND Gates



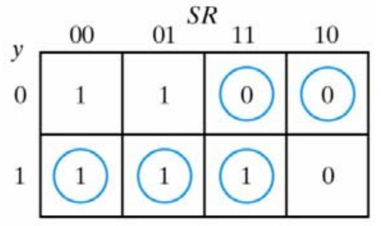
(a) Crossed-coupled circuit

| 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

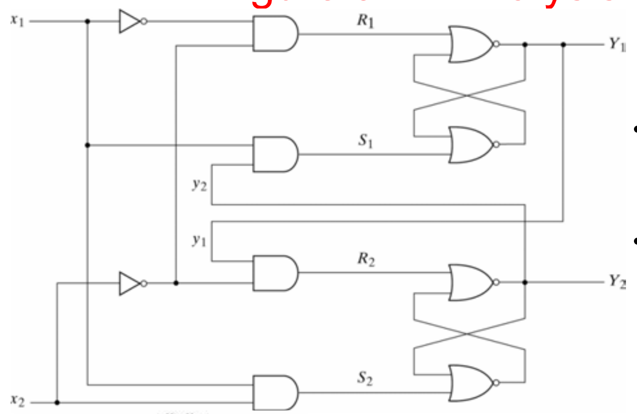


| y \ SR | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 |

$Y = S' + Ry$ when $S'R' = 0$
(d) Transition table

Figure 9-11 Excitation variable: Y = [S(Ry)']' = S' + Ry

## Figure 9-12 Analysis Example



- First obtain S and R inputs
  $S_1=x_1y_2$ $S_2=x_1x_2$
  $R_1=x_1'x_2'$ $R_2=x_2'y_1$
- Check if SR=0 is satisfied
  $S_1R_1=x_1y_2x_1'x_2'=0$
  $S_2R_2=x_1x_2x_2'y_1=0$

| $y_1y_2$ \ $x_1x_2$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 00 | 00 | 01 | 00 |
| 01 | 01 | 01 | 11 | 11 |
| 11 | 00 | 11 | 11 | 10 |
| 10 | 00 | 10 | 11 | 10 |

•Derive the excitation functions (by Y=S+R'y)
$Y_1=S_1+R_1'y_1=x_1y_2+(x_1+x_2)y_1=x_1y_2+x_1y_1+x_2y_1$
$Y_2=S_2+R_2'y_2=x_1x_2+(x_2+y_1')y_2=x_1x_2+x_2y_2+y_1'y_2$
•Derive the transition table

## Procedure for analyzing an asynchronous sequential circuit with SR latches

Logic circuit ⇒ transition table/map

1. Label each latch output with $Y_i$ and its external feedback path (if any) with $y_i$ for i = 1, 2, …, k
2. Derive the Boolean functions for $S_i$ and $R_i$ inputs in each latch
3. Check whether SR=0 for each NOR latch or whether S'R'=0 for each NAND latch
   - If not satisfied, it's possible that the circuit may not operate properly
4. Evaluate Y=S+R'y for each NOR latch or Y=S'+Ry for each NAND latch
5. Construct a map with the y's representing the rows and the x inputs representing the columns
6. Plot the value of $Y=Y_1Y_2…Y_k$ in the map
7. Circle all stable states where Y=y. The resulting map is then the transition table

## Latch Excitation Table

- excitation table: lists the required inputs S and R for each of the possible transitions from y to Y
  - To find the values of S and R during the design/implementation process

| $y$ \ $SR$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 0 | $\cancel{0}$ | 1 |
| 1 | 1 | 0 | $\cancel{0}$ | 1 |

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

Fig. 9-10 (d) Transition table

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

(b) Latch excitation table
Fig. 9-14

Derived from the latch transition table of Fig. 9-10(d)
•Remove the unstable condition SR=11
•i.e. to change from y=0 to Y=0, SR can be either 00 or 01⇒ S must be 0

## Implementation Example

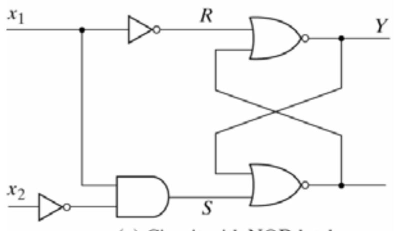
Transition table ⇒ Logic circuit

| $y$ \ $x_1x_2$ | 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



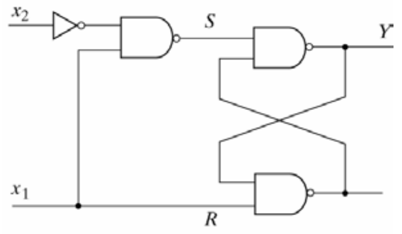
(e) Circuit with NOR latch

| $y$ \ $x_1x_2$ | 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 $R = x'_1$



(f) Circuit with NAND latch

•Derive (c) and (d) from (a) by referencing (b)
•Use the complemented values for S and R of NOR latch to derive the circuit for NAND latch: $S=(x_1x_2')'$ and $R = x_1$

## Procedure for implementing a circuit with SR latches from a given transition table

1. Given a transition table that specifies the excitation function $Y=Y_1Y_2 \ldots Y_k$, derive a pair of maps for $S_i$ and $R_i$

2. Derive the simplified Boolean functions for each $S_i$ and $R_i$
   - DO NOT make $S_i$ and $R_i$ equal to 1 in the same minterm square

3. Draw the logic diagram
   - for NAND latches, use the complemented values of those $S_i$ and $R_i$

## Debounce Circuit

- Mechanical switch: as input signal
- Debounce circuit
  - remove the series of pulses that result form a contact bounce and produce a single smooth transition of the binary signal

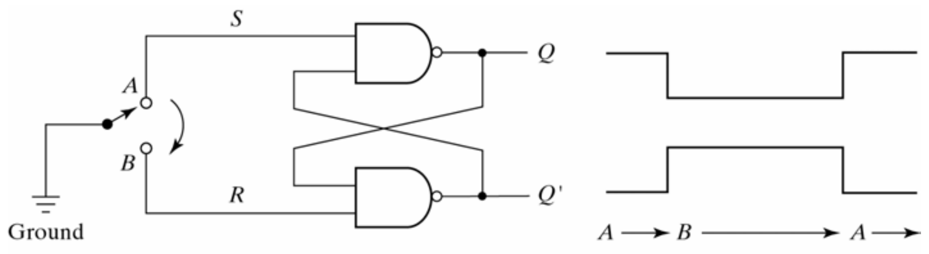


Fig. 9-15 Debounce Circuit

## 9-4 Design Procedure

- Start from the statement of problem and culminate in a logic diagram

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
- 1st step: derive transition table and flow table
  - no simultaneous transitions of two variables
  - state a: after inputs DG=01
  - state b: after inputs DG=11
  - only one stable state in each row

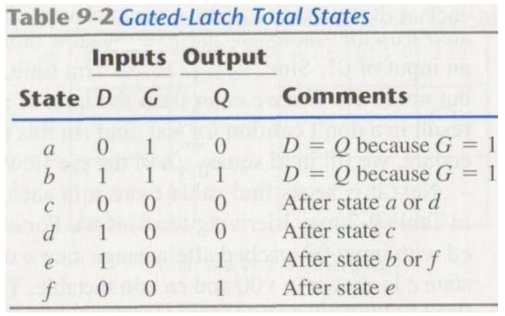


Table 9-2 *Gated-Latch Total States*

| | Inputs | | Output | |
|---|---|---|---|---|
| State | D | G | Q | Comments |
| *a* | 0 | 1 | 0 | *D* = *Q* because *G* = 1 DG=01 |
| *b* | 1 | 1 | 1 | *D* = *Q* because *G* = 1 DG=11 |
| *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* |

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

| 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

## 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 columns (formal procedure is given in next section)
  - Primitive flow table is separated into two parts of three rows each

| 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, – |

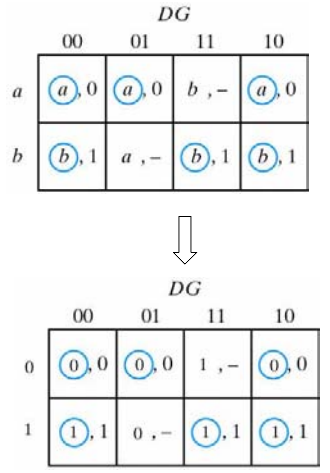
(a) States that are candidates for merging

| 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 |

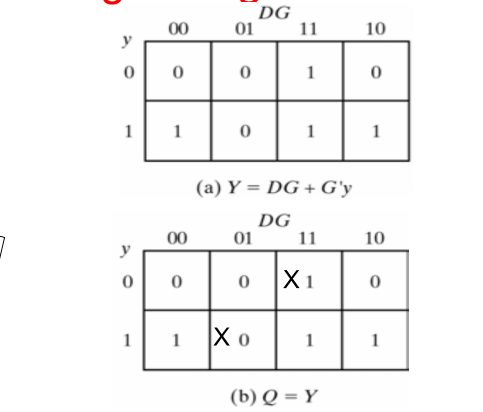
(b) Reduced table (two alternatives)

## Transition Table and Logic Diagram



| DG | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| a | a, 0 | a, 0 | b , – | a, 0 |
| b | b, 1 | a , – | b, 1 | b, 1 |

| DG | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0, 0 | 0, 0 | 1 , – | 0, 0 |
| 1 | 1, 1 | 0 , – | 1, 1 | 1, 1 |



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

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

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

(b) $Q = Y$

Fig. 9-18 Transition Table and Output Map for Gated Latch

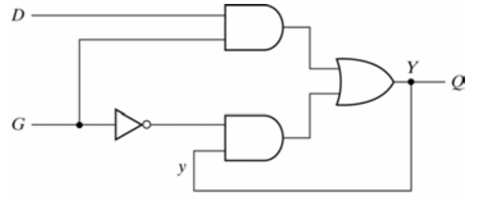
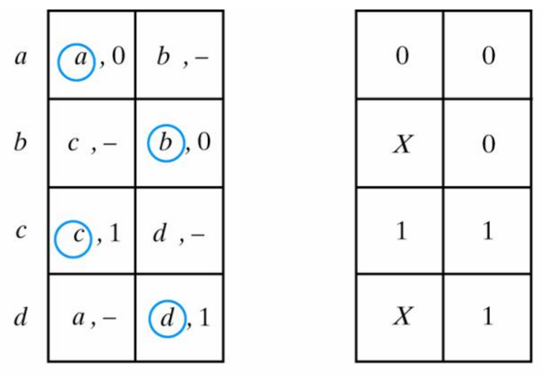


Fig. 9-19 Gated-Latch Logic Diagram

- State assignment
  - discussed in details in Sec. 9-6
  - a:0, b:1
  - Assign don't care: X=1 for y=0 and X=0 for y=1 ⇒ Q=Y

## Assign Outputs to Unstable States

•the unstable states have unspecified output values
•no momentary false outputs occur when circuit switches between stable states
  0→0 ⇒ 0 : assign 0 if the transient state between two 0 stable states
  1→1 ⇒ 1 : assign 1 if the transient state between two 1 stable states
  0→1, 1→0 ⇒ don't care: assign don't care if the transient state between two different stable states



| | | |
|---|---|---|
| a | a, 0 | b , – |
| b | c , – | b, 0 |
| c | c, 1 | d , – |
| d | a , – | d, 1 |

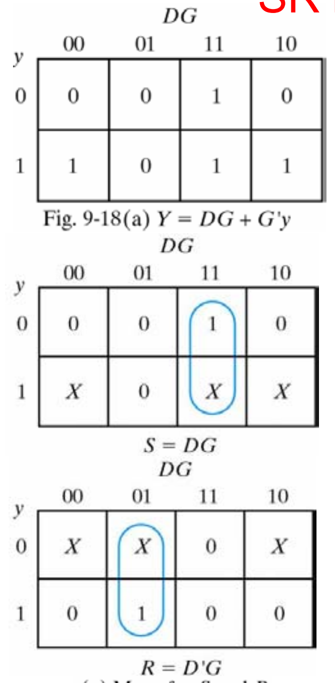
(a) Flow table

| | |
|---|---|
| 0 | 0 |
| X | 0 |
| 1 | 1 |
| X | 1 |

(b) Output assignment

Fig. 9-21 Assigning Output Values to Unstable States

## SR Latch Implementation



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

Fig. 9-18(a) $Y = DG + G'y$

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

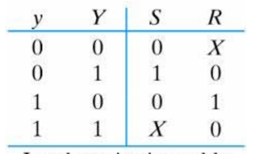
$S = DG$

| 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

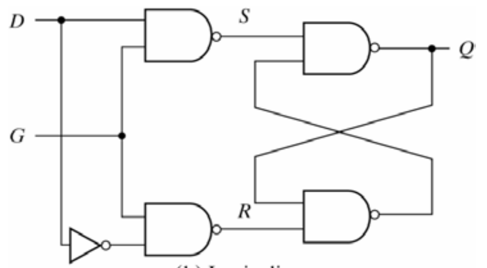
Use the procedure outlined in Sec. 9-3
- Obtain S=DG and R=D'G from Fig.9-18(a) by referencing the latch excitation table



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

Latch excitation table

- Draw the circuit with SR latch



(b) Logic diagram

FIG. 9-20 Circuit with SR Latch

## Summary of Design Procedure

0. Problem definition: state the design specifications
1. Interpretation: Obtain a *primitive flow table* from the given design specifications (Section 9-4; most difficult)
2. State reduction: reduce flow table by merging rows in primitive flow table (Section 9-5; *implication table*, *merger diagram*)
   –Reduce *equivalent states* and *compatible states*
3. State assignment: assign binary state variables to each row of the reduced flow table to obtain the *transition table*
   –Eliminates any possible critical races (Section 9-6)
4. Output assignment: assign output values to the dashes associated with the unstable states to obtain the *output maps*
5. Simplification: Simplify the Boolean functions of the excitation and output variables and draw the *logic diagram*
   –can be drawn using SR latches (Section 9-3)

## 9-5 Reduction of State and Flow Tables

- Reduction of state and flow tables
  - Equivalent states
  - Compatible states: there are unspecified states/outputs
- Equivalent states: for each input, two states
  - give exactly the same output and
  - go to the same next states or to equivalent next states
- Demonstrate
  - (a,b) are equivalent if (c,d) are equivalent
  - (a,b) ***imply*** (c,d)
  - (c,d) ***imply*** (a,b)
  - both pairs are equivalent

**Table 9-3** *State Table to Demonstrate Equivalent States*

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

## Implication Table

**Table 9-4** *State Table to Be Reduced*

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

check each pair of states for possible equivalence

(1) Mark 'x' for pairs with different outputs

|  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|
| *b* |  |  |  |  |  |  |
| *c* | × | × |  |  |  |  |
| *d* | × | × | × |  |  |  |
| *e* | × | × | × |  |  |  |
| *f* |  |  | × | × | × |  |
| *g* | × | × | × |  |  | × |
|  | *a* | *b* | *c* | *d* | *e* | *f* |

(2) For same outputs, mark 'v' for pairs with same next states, or enter next states to be checked

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

(3) Make successive passes to determine equivalences of remaining pairs

(d,e) implied by (a,b), (d,g) and (e,g)

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

Fig. 9-22 Implication Table

## Equivalent and Reduced States

- Equivalent states
  - (a,b)
  - (d,e), (d,g), (e,g) ⇒ (d,e,g)
- Reduced states
  - (a,b), (c), (d,e,g), (f)
- State table

**Table 9-4** *State Table to Be Reduced*

| Present State | Next State x = 0 | Next State x = 1 | Output x = 0 | Output x = 1 |
|---|---|---|---|---|
| *a* | *d* | x a | 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~~ |

**Table 9-5**
*Reduced State Table*

| Present State | Next State x = 0 | Next State x = 1 | Output x = 0 | Output 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

- Consider the don't-care conditions
  - combinations of inputs or input sequences may never occur
- **compatible**: two incompletely specified states that can be combined, even not equivalent
  - for each possible input:
    - they have the same output whenever specified and
    - their next states are compatible whenever they are specified

- Procedure for finding a suitable group of compatibles for merging a flow table
  1. determine all ***compatible pairs*** by using the ***implication table***
  2. find the ***maximal compatibles*** using a ***merger diagram***
  3. find a ***minimal collection*** of compatibles that cover all the states and is closed

## Step 1 of 3. Find Compatible Pairs

| | 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

Fig. 9-23 Flow and Implication Tables

Compatible pairs: (a,b) (a,c) (a,d) (b,e) (b,f) (c,d) (e,f)

(1) Check if compatible, or enter next states to be checked

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

(2) Make successive passes to determine compatibility of remaining pairs (no implied states)

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

(b) Implication table

## Step 2 of 3. Find Maximal Compatibles

Maximal compatibles: a group of compatibles that contains all the possible combinations of compatible states

**merger diagram**

– an isolated dot: a state that is not compatible to any other state

– a line: a compatible pair

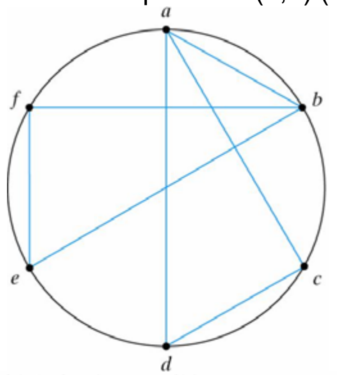
– a triangle: a compatible with three states

– an n-state compatible: an n-sided polygon with *all its diagonals connected*

Figure 9-23 Example

•Compatible pairs: (a,b) (a,c) (a,d) (b,e) (b,f) (c,d) (e,f)

•Maximal compatibles: (a,b) (a,c,d) (b,e,f)



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

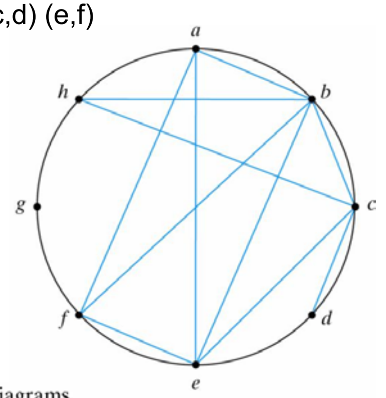


Fig. 9-24 Merger Diagrams

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

## Step 3 of 3. Closed Covering Condition

- **Closed covering**: the set of chosen compatibles must *cover all the states* and must *be closed*
  - closed: no implied states or the implied states are included within the set
  - implied states: entered in the checked square of the implication table
- Figure 9-23 / Figure 9-24(a) Example
  - Compatible pairs: (a,b) (a,c) (a,d) (b,e) (b,f) (c,d) (e,f)
  - Maximal compatibles: (a,b) (a,c,d) (b,e,f)
  - no implied states
  - Chosen set: (a,c,d) (b,e,f)
    - all six states are included: covering all states
    - no implied states: closed

| | 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 |

Fig. 9-17 (b) Reduced table (two alternatives)

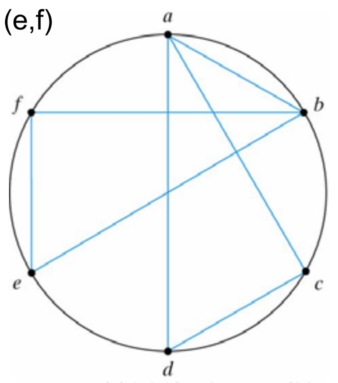
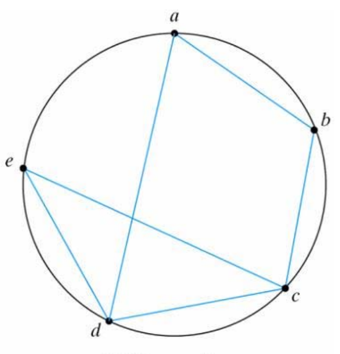


Fig. 9-24 (a) Maximal compatible: (a, b,) (a, c, d) (b, e, f)

## Closed and Unclosed

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

(a) Implication table



(b) Merger diagram

- Figure 9-25 Example (given (a) implication table)
  - compatible pairs
    (a,b) (a,d) (b,c) (c,d) (c,e) (d,e)
  - maximal compatibles
    (a,b) (a,d) (b,c) (c,d,e)
- Case I - chosen compatibles: (a,b) (c,d,e)
  - cover all the states
  - not closed: (b,c), implied by (a,b), not included
- Case II – chosen compatibles: (a,d) (b,c) (c,d,e)
  - cover all the states
  - closed: implied states (b,c) (d,e) (a,d) included
  - the same state can be repeated more than once

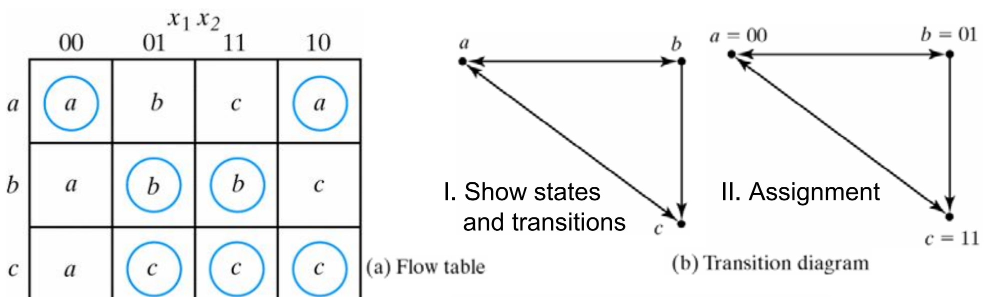
| Compatibles | (a, b) | (a, d) | (b, c) | (c, d, e) |
|---|---|---|---|---|
| Implied states | (b, c) | (b, c) | (d, e) | (a, d)<br>(b, c) |

(c) Closure table

## 9-6 Race-Free State Assignment

- *Race-free*: avoiding critical races
  - Only one variable changes at any given time
  - may allow noncritical race
- *Adjacent assignment*
  - Condition: binary values of states between which transitions occur only differ in one variable
    - tedious process: test and verify each possible transition between two stable states
  - $m$ variables required for a flow table with $n$ rows: $2^m \geq n$
    - No critical race for assigning a single variable to a flow table with two rows
- *Transition diagram*: pictorial representation of all required transitions between rows
  - Try to find only one binary variable changes during each state transition
  - If critical races exist, add extra rows to obtain race-free assignment
- Two methods for race-free state assignment
  - *shared-row method*
  - *multiple-row method*

## Three-Row Flow-Table Example



| $x_1x_2$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| a | a | b | c | a |
| b | a | b | b | c |
| c | a | c | c | c |

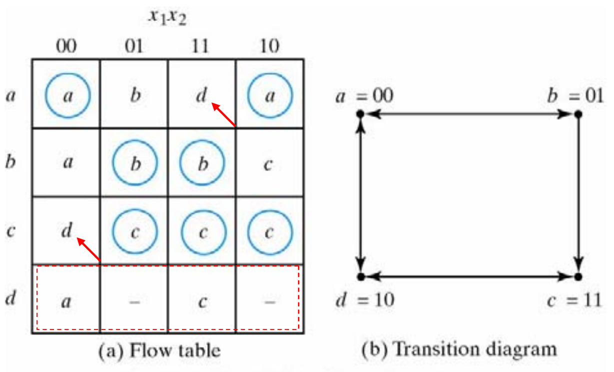
(a) Flow table (b) Transition diagram

Fig. 9-26 Three-Row Flow-Table Example

1. Derive the transition diagram from the flow table
   - Uni-directed line: one-way transition
   - Bi-directed line: two-way transition
2. State assignment: assign a=00, b=01, c=11
   - critical race: transition a → c
   - noncritical race: transition c → a

Race-free assignment: add an extra row to the flow table to avoid the critical race

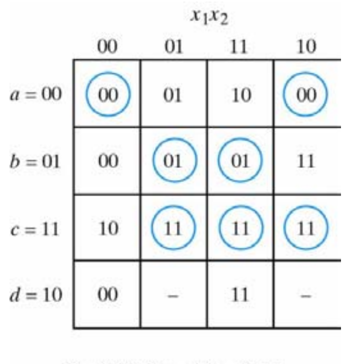
## Flow Table with an Extra Row



| $x_1x_2$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| a | a | b | d | a |
| b | a | b | b | c |
| c | d | c | c | c |
| d | a | – | c | – |

(a) Flow table (b) Transition diagram

Fig. 9-27 Flow Table with an Extra Row

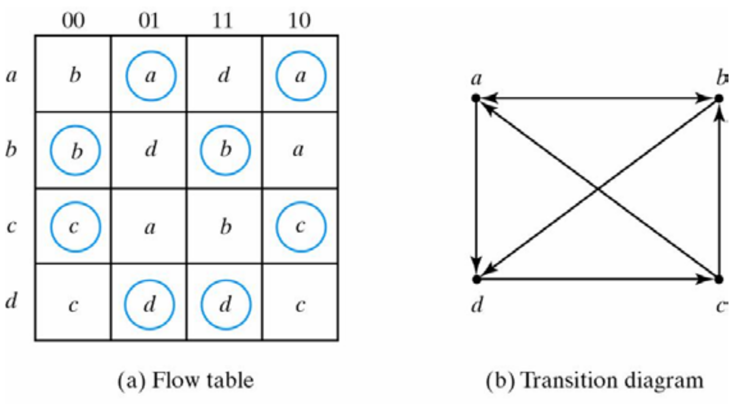


| $x_1x_2$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| a = 00 | 00 | 01 | 10 | 00 |
| b = 01 | 00 | 01 | 01 | 11 |
| c = 11 | 10 | 11 | 11 | 11 |
| d = 10 | 00 | – | 11 | – |

Fig. 9-28 Transition Table

- An extra row labeled d is added
  - critical-race transition a → c becomes a=00 → d=10 → c=11
  - noncritical-race transition c → a becomes c=11 → d=10 → a=00
  - no stable state in row d: two dashes represent unspecified states that
    - can be considered don't-care conditions
    - must not be d=10, or becomes stable state

## Four-Row Flow-table Example



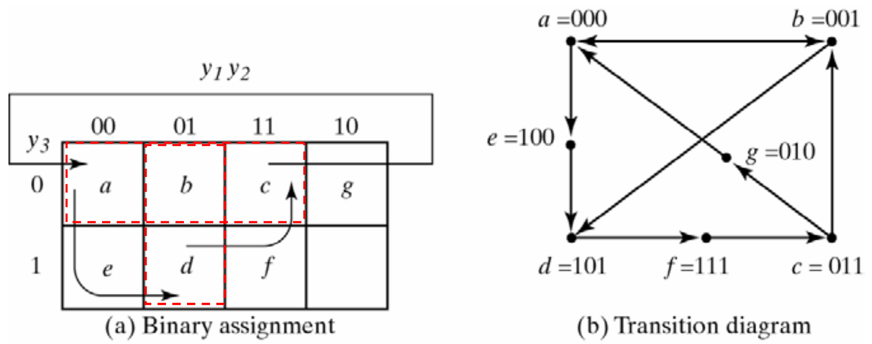
| | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| a | b | a | d | a |
| b | b | d | b | a |
| c | c | a | b | c |
| d | c | d | d | c |

(a) Flow table (b) Transition diagram

Fig. 9-29 Four-Row Flow-Table Example

Figure 9-29 Example: 4 states/rows
- Require a minimum of two state variables
- Diagonal transitions c→a and b→d make adjacent assignment impossible
- Therefore, at least 3 binary state variables are needed

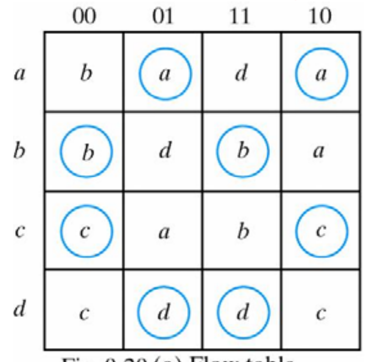
## Assignment for Four-Row Flow Table



(a) Binary assignment (b) Transition diagram
Fig. 9-30 Choosing Extra Rows for the Flow Table

- Figure 9-30: assignment for the 4-row flow table
  - Original states: a, b, c and d
  - Extra states: e, f and g
- Expanded to a seven-row table that is free of critical races
  - a→d ⇒ a→e→d
  - d→c ⇒ d→f→c
  - c→a ⇒ c→g→a
- It is suitable for any four-row flow table

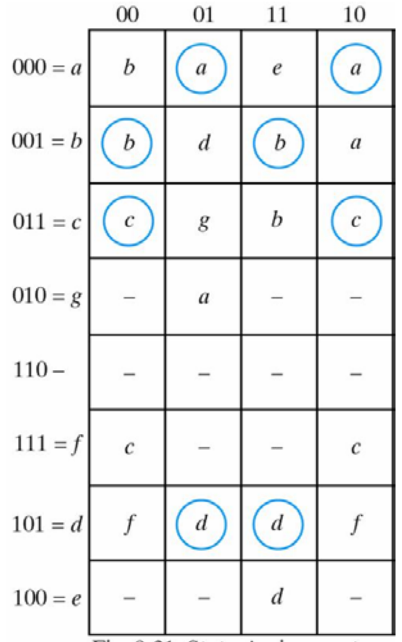
## State Assignment to Modified Flow Table



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

Fig. 9-29 (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 | – |

Fig. 9-31 State Assignment to Modified Flow Table

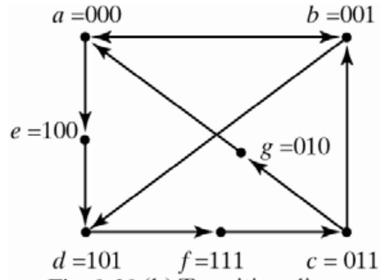
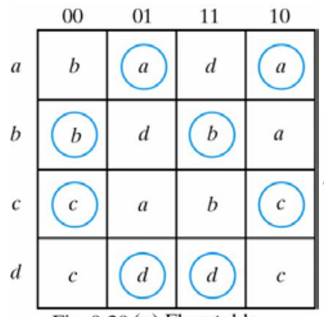


Fig. 9-30 (b) Transition diagram

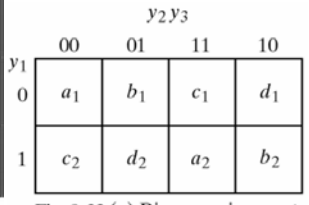
## Multiple-Row Method

- Methods for race-free assignment
  - shared-row method: adding extra rows
  - multiple-row method: multiple equivalent states for each stat
    - less efficient but easier to apply
- Multiple-row method for 4-row flow table
  - original state a is replaced by a1 and a2
  - each original state is adjacent to three states



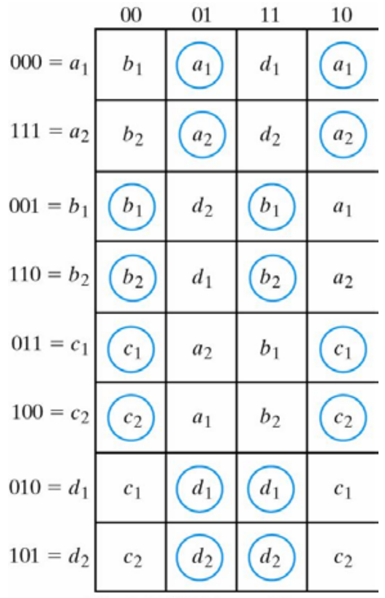
| | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| a | b | a | d | a |
| b | b | d | b | a |
| c | c | a | b | c |
| d | c | d | d | c |

Fig. 9-29 (a) Flow table



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

Fig. 9-32 (a) Binary assignment



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

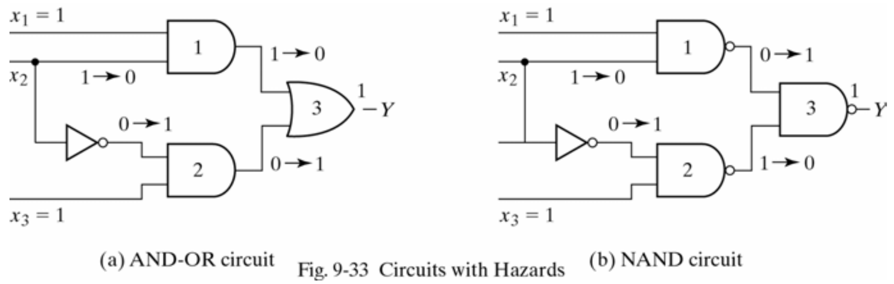
Fig. 9-32 (b) Flow table

## 9-7 Hazards

- In the design of asynchronous sequential circuit, the circuit
  - must be operated in fundamental mode with *only one input changing at any time*, and
  - must be free of critical races
- Hazards: unwanted switching transients at the output
  - because different paths exhibit different propagation delays
  - May cause the circuit to malfunction
    - in combinational circuits: may cause temporary false-output value
    - in asynchronous sequential circuits: may result in a transition to a wrong stable state
  - Need to check for possible hazards and determine whether causing improper operations

## Hazards in Combinational Circuits

•hazard: a condition where a single variable change produces a momentary output change when no output change should occur

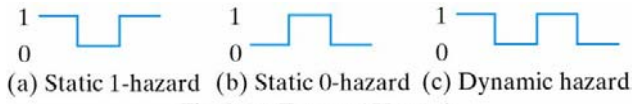


(a) AND-OR circuit Fig. 9-33 Circuits with Hazards (b) NAND circuit

•Assume all inputs are initially set to 1
– gate1 = 1, gate2 = 0 ⇒ gate3 = 1
•Consider a change of $x_2$ from 1 to 0
– gate1 = 0, gate2 = 1 ⇒ gate3 = 1
•Hazard: inverter delay may cause gate1=0 to change before gate2=1
– gate1 = 0, gate2 = 0 ⇒ gate3 = 0
– momentary gate3=1→0→1!

•Fig.9-33(b) is a NAND implementation of Fig.9-33(a)
– $Y=x_1x_2+x_2'x_3 = (x_1+x_2')(x_2+x_3)$

## Types of Hazards



(a) Static 1-hazard (b) Static 0-hazard (c) Dynamic hazard
Fig. 9-34 Types of Hazards

‖
change three or more when 0→1 or 1→0

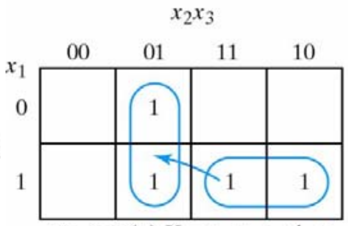


Fig. 9-35(a) $Y = x_1x_2 + x'_2x_3$
(both product terms have $x_2$)

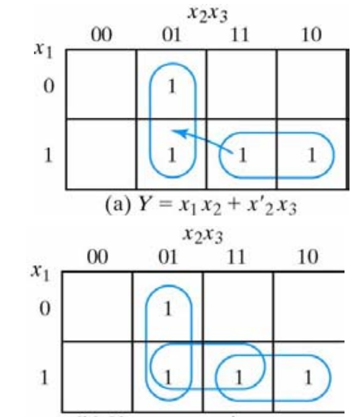
• Whenever the circuit must move from one product term to another, there is a possibility of a momentary interval when neither term is equal to 1, giving rise to an undesirable 0 output

• Detected by inspecting the map: the change of input results in different product term covering the two minterms
  – minterm 111 in gate 1 and minterm 101 in gate 2

• When a circuit is implemented in sum of products (AND-OR or NAND gates), the removal of static 1-hazard guarantees that no static 0-hazards or dynamic hazards will occur
  – If the momentary input causes the OR output to change from 0 to 1, the output will maintain at 1 after the propagation

## Hazard-Free Circuit

• The remedy: enclose the two minterms in question with another product term
  – the circuit moves from one product term to another
  – additional redundant gate
• General solution: cover any two minterms with a product term common to path



(a) $Y = x_1x_2 + x'_2x_3$
(b) $Y = x_1x_2 + x'_2x_3 + x_1x_3$
Fig. 9-35 Maps Demonstrating a Hazard and its Removal

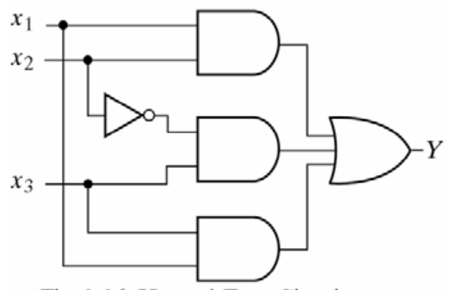
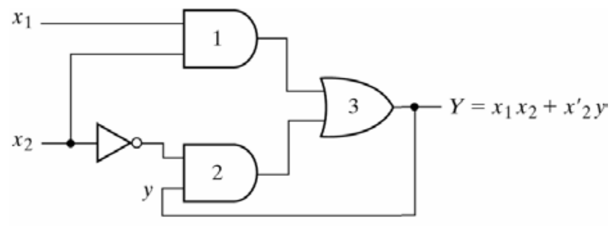


Fig. 9-36 Hazard-Free Circuit

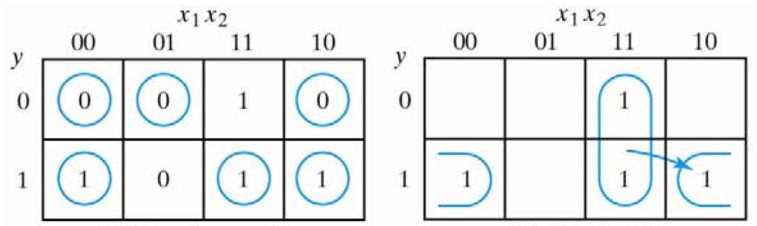
## Hazards in Sequential Circuits

In general, no problem for synchronous design, but a momentary incorrect signal fed back in asynchronous sequential circuit may cause the circuit to go to the wrong stable state



(a) Logic diagram

Figure 9-37 Example:
•state $yx_1x_2$=111 and input $x_2$ 1→0
•next state should be 110
•hazard: output Y may go to 0 momentarily
  • feeds back to gate 2 before $x_2$' enter gate 2
  • the circuit will switch to incorrect stable state 010



(b) Transition table (c) Map for $Y$
Fig. 9-37 Hazard in an Asynchronous Sequential Circuit

## Implementation with SR latches

Asynchronous sequential circuits with SR latches

- A third input to the gate from the complemented side of the latch Q' avoids static hazards (maintained at 1 or 0)
  - a momentary 0 signal at the S or R inputs of NOR latch has no effect
  - a momentary 1 signal at the S or R inputs of NAND latch has no effect

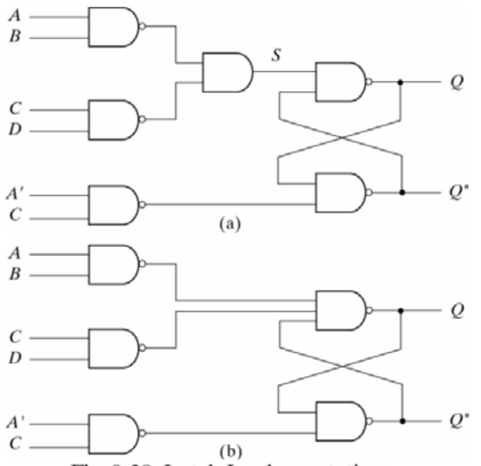


Fig. 9-38 Latch Implementation

Figure 9-38 Example:

- Consider a NAND SR latch
  - S=AB+CD and R=A'C
  - complement the inputs for NAND
    - S=(AB+CD)'=(AB)'(CD)'
    - R=(A'C)'
  - shown in Fig. 9-38(a)
- Boolean function for output Q
  - Q=(Q'S)'=[Q'(AB)'(CD)']'
  - generated in Fig. 9-38(b) with 2-levels of NAND gates

## Essential Hazards

- Essential Hazards: due to unequal delays along two or more paths that originate from the *same input*
  - Another type of hazard may occur in asynchronous sequential circuits
    - Static or dynamic hazards are resulted from delays of different inputs
    - It cannot be corrected by adding redundant gates
  - Solution: adjust the amount of delay in the affected path
    - the delay of feedback loops > delays of other signals that originate from the input terminals
    - Tends to be specialized

## 9-8 Design Example

Summary of design procedure

1. Problem definition: state the design specifications
2. Interpretation: derive the primitive flow table (Section 9-4)
   - total states: depend on # of input variables and # of secondary variables
3. State reduction: reduce the flow table by merging the rows (Section 9-5)
   - Reduce equivalent states and compatible states by using implication table and merger diagram to meet the closed covering condition
4. Race-free state assignment (Section 9-6)
   - adjacent assignment with transition diagram to avoid critical races
   - shared-row method and multiple-row method
5. Obtain the transition table and output map
   - Simplify the Boolean functions of the excitation and output variables
6. Obtain the logic diagram using SR latches (Section 9-3)

Example: design a negative-edge-triggered T flip-flop

## 1. Design Specifications

Design a negative-edge-triggered T flip-flop

- Variables
  - Two inputs, T (toggle) and C (clock), and
  - one output, Q
- Functions
  - Output state is complemented if
    - T=1 and
    - the clock C changes from 1 to 0 (negative-edge triggering)
  - Otherwise, output Q remains unchanged
    - under any other input condition

Note that this circuit can be used as a flip-flop in clocked sequential circuits, the internal design of the flip-flop is an asynchronous problem

## 2. Primitive Flow Table

Table 9-6 *Specification of Total States*

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

Differ by one variable and
- a,c: T=1, C=↑ ⇒ Q=initial values
- b,d: T=1, C=↓ ⇒ Q=Q'
- e,f,g,h: T=0 ⇒ Q=unchanged

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

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

Fig. 9-39 Primitive Flow Table

1. Fill in one square in each row belonging to the stable state
2. Enter dashes in those squares whose input differs by two variables from the input corresponding to the stable state
3. Unstable conditions are determined by utilizing Table 9-6

## 3. Merging of the Flow Table

- Derive Fig. 9-40 implication table from Fig. 9-39 flow table
  - Compatible pairs:
    (a,f) (b,g) (b,h) (c,h) (d,e) (d,f) (e,f) (g,h)
  - no implied states

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

Fig. 9-39 Primitive Flow Table

(1) Mark 'x' for pairs with different outputs

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *b* | | | | | | | |
| *c* | × | | | | | | |
| *d* | | × | | | | | |
| *e* | | | | | | | |
| *f* | | | | | | | |
| *g* | | | | | × | | |
| *h* | | | | | | × | |
| | *a* | *b* | *c* | *d* | *e* | *f* | *g* |

(2) mark 'v' for pairs with same next states, or enter next states to be checked

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *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* | × | ✓ |
| | *a* | *b* | *c* | *d* | *e* | *f* | *g* |

(3) Make successive passes to determine equivalences of remaining pairs

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *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* × | × | ✓ |
| | *a* | *b* | *c* | *d* | *e* | *f* | *g* |

Fig. 9-40 Implication Table

## Merging of the Flow Table (cont.)

Derive the reduced flow table (Fig. 9-42)
- Compatible pairs:
  (a,f) (b,g) (b,h) (c,h) (d,e) (d,f) (e,f) (g,h)

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

Fig. 9-39 Primitive Flow Table

- Maximal compatible set
  (a,f) (b,g,h) (c,h) (d,e,f)
  – covering all states
    (states h and f are repeated)
  – closed: no implied states

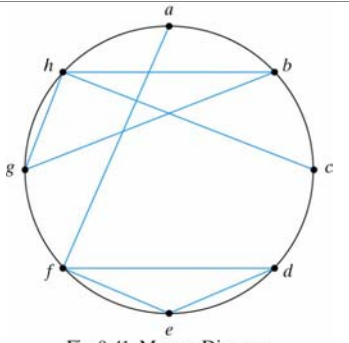


Fig. 9-41 Merger Diagram

⇒

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

(a)

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

(b)

Fig. 9-42 Reduced Flow Table

## 4. Race-free State Assignment

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

Fig. 9-42(b) Reduced Flow Table

⇒

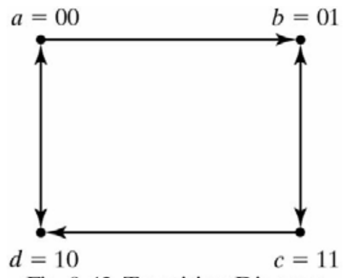
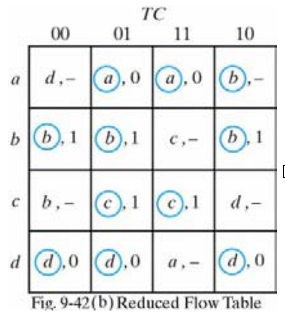


Fig. 9-43 Transition Diagram

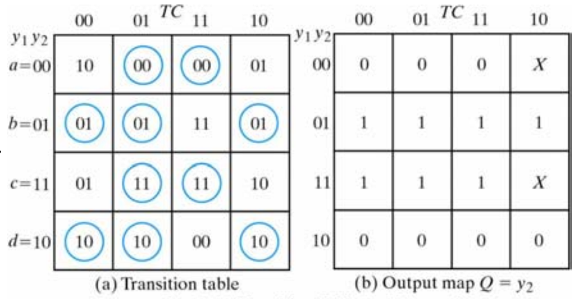
- Draw transition diagram Fig. 9-43 from the reduced flow table
  - four stable states
  - no diagonal lines
- Find the race-free state assignment by adjacent assignment
  - a=00, b=01, c=11, d=10

## 5. Obtain the Transition Table and Output Map



| | 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 |

Fig. 9-42(b) Reduced Flow Table



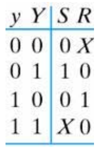
| $y_1y_2$ | TC 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| a=00 | 10 | 00 | 00 | 01 |
| b=01 | 01 | 01 | 11 | 01 |
| c=11 | 01 | 11 | 11 | 10 |
| d=10 | 10 | 10 | 00 | 10 |

(a) Transition table

| $y_1y_2$ | 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 |

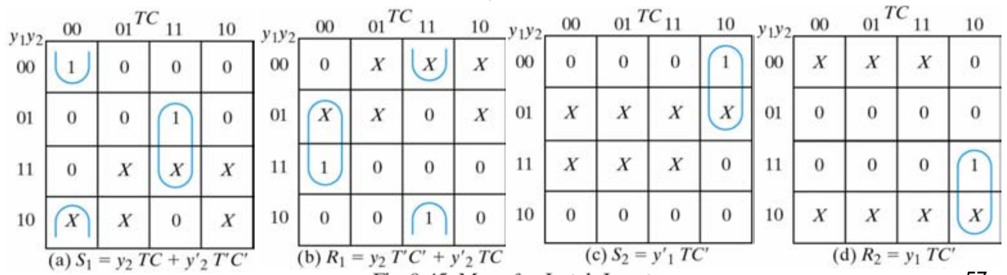
(b) Output map $Q = y_2$

Fig. 9-44 Transition Table and Output Map



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

Fig. 9-14(b) Latch excitation table



| $y_1y_2$ | 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) $R_2 = y_1 TC'$

Fig. 9-45 Maps for Latch Inputs

## 6. Obtain the Logic Diagram Using SR Latch

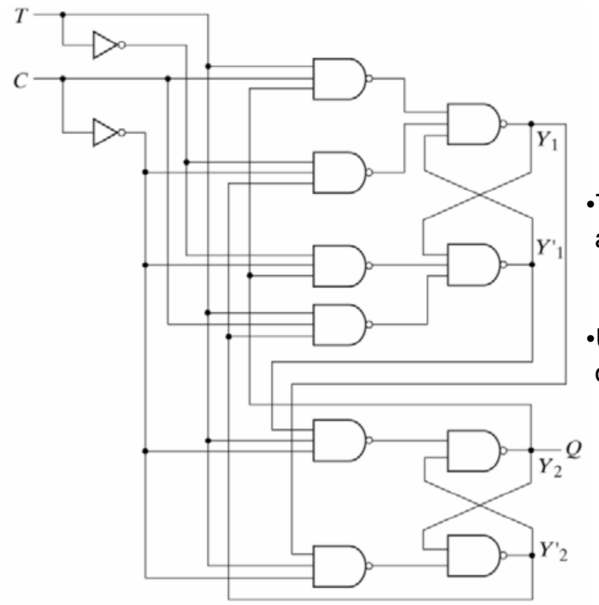


Fig. 9-46 Logic Diagram of Negative-Edge-Triggered T Flip-Flop

- Two state variables, $Y_1$ and $Y_2$, and one output, Q
  - two SR latches, one for each state variable
- Use two NAND latches with two or three inputs in each gate

## Summary