Synchronous Sequential Circuit

- ✓ The change of internal state occurs in response to the synchronized clock pulses.
- ✓ Data are read during the clock pulse (e.g. rising-edge triggered)
- ✓ It is supposed to wait long enough after the external input changes for all flip-flop inputs to reach a steady value before the next clock pulse
- ✓ Unsuitable Situations:
  - Inputs can change at any time and cannot be synchronized with a clock
  - Circuit is large, a cost in time of transitions can not be avoided

Asynchronous Circuits

- ✓ Not synchronized by a common clock
- ✓ States change immediately after input changes
- ✓ For a given value of input variables, the system is stable if the circuit reaches a steady state condition.
- ✓ The circuit reaches a steady-state condition when $y_i = Y_i$ for all i.
- ✓ A transition from one stable state to another occurs only in response to a change in an input variable
- ✓ **Fundamental-mode operation**
  - The input signals change only when the circuit is in a **stable condition**
  - The input signals change **one at a time**
- ✓ The time between two input changes must be longer than the time it takes the circuit to reach a stable state.
- ✓ Timing is a Major Problem because of **unequal delays** through various paths in the circuit

Why Asynchronous Sequential Circuits?

Asynchronous sequential circuits basics

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

Why Asynchronous Circuits?

- ✓ Accelerate the speed of the machine (no need to wait for the next clock pulse).
- ✓ Simplify the circuit in the small independent gates.
- ✓ Necessary when having multi circuits each having its own clock.

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.

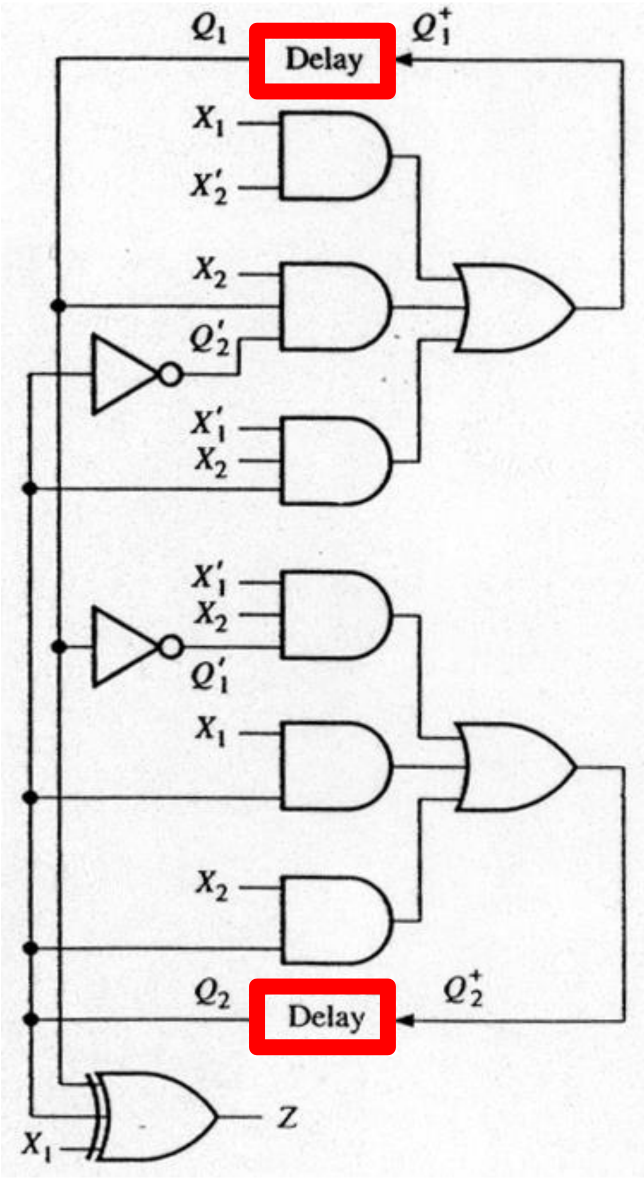
Example Circuit

- ✓ First construction of Asynchronous Circuits:
  - using only gates
  - with **feedback paths**
- ✓ Analysis:
  - Lump all of the delay associated with each feedback path into a "delay" box
  - Associate a state variable with each delay output
  - Construct the flow table
- ✓ Network equations

$$Q_1^+ = X_1X_2' + X_1'X_2Q_2 + X_2Q_1Q_2'$$

$$Q_2^+ = X_1'X_2Q_1' + X_1Q_2 + X_2Q_2$$

$$Z = X_1 \oplus Q_1 \oplus Q_2$$



Example Circuit: Output Table

- ✓ 1. Starting in total state
  $X_1X_2Q_1Q_2$=0000
- ✓ 2. Input changes to 01
  - Internal state changes to 01 and then to 11.
- ✓ 3. Input changes to 11.
  - Go to unstable total state 1111 and then to 1101.
- ✓ 4. Input changes to 10.
  - Go to unstable total state 1001 and then to 1011.
- ✓ The output sequence:
  0 (0) (1) 0 (1) 0 (0) 1
  - Condensed to the form
    0 (1) 0 (1) 0 1.
  - Two transient 1 outputs is dangerous can be eliminated by proper design.

$Q_1^+ = X_1X_2' + X_1'X_2Q_2 + X_2Q_1Q_2'$

$Q_2^+ = X_1'X_2Q_1' + X_1Q_2 + X_2Q_2$

Q_1Q_2 \ X_1X_2	00	01	11	10
00	00	01	00	10
01	00	11	01	11
11	00	11	01	11
10	00	10	10	10

Q_1Q_2 \ X_1X_2	00	01	11	10
00	0	0	1	1
01	1	1	0	0
11	0	0	1	1
10	1	1	0	0

Z

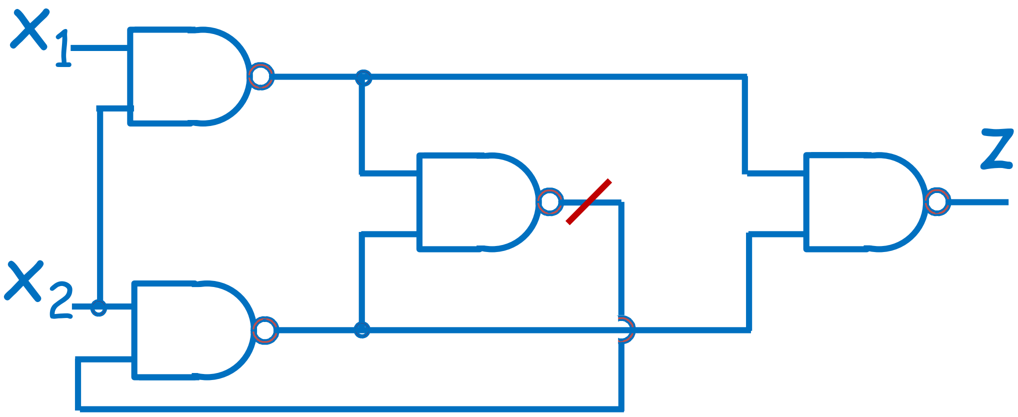
$Z = X_1 \oplus Q_1 \oplus Q_2$

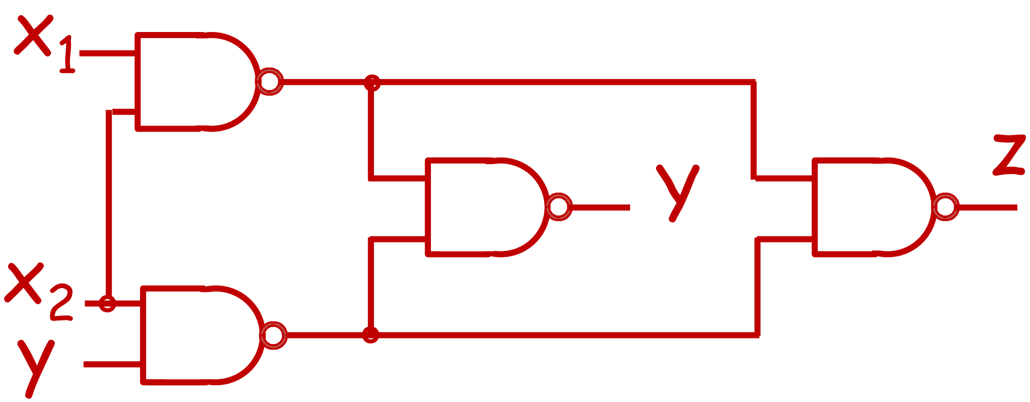
Transition Table

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

1. Determine all feedback loops in the circuits
2. Mark the input (y_i) and output (Y_i) of each feedback loop
3. Derive the Boolean functions of all Y's
4. Plot each Y function in a map and combine all maps into one table (**flow table**)
5. Circle those values of Y in each square that are equal to the value of y in the same row (stable states)

Asynchronous Sequential Analysis

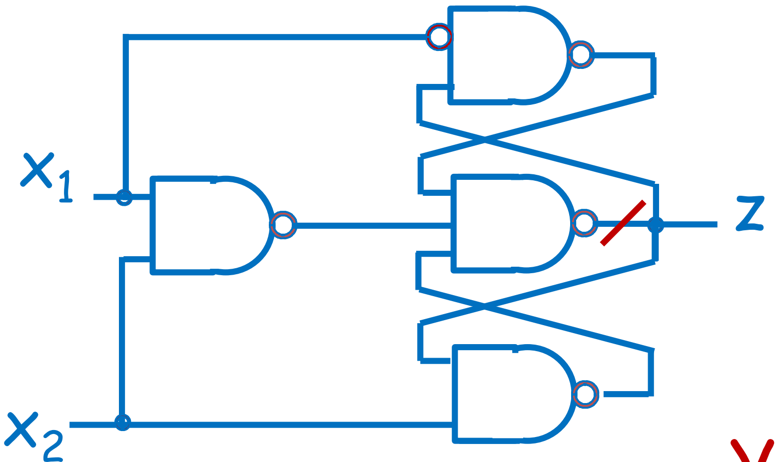


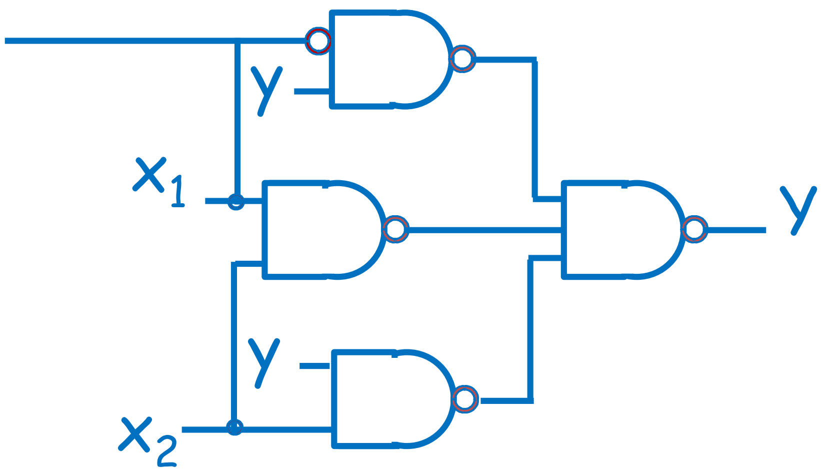


$$Y = \overline{\overline{x_1 x_2}\;\;\overline{x_2 y}} = x_1 x_2 + x_2 y$$

$$z = y$$

Asynchronous Sequential Analysis

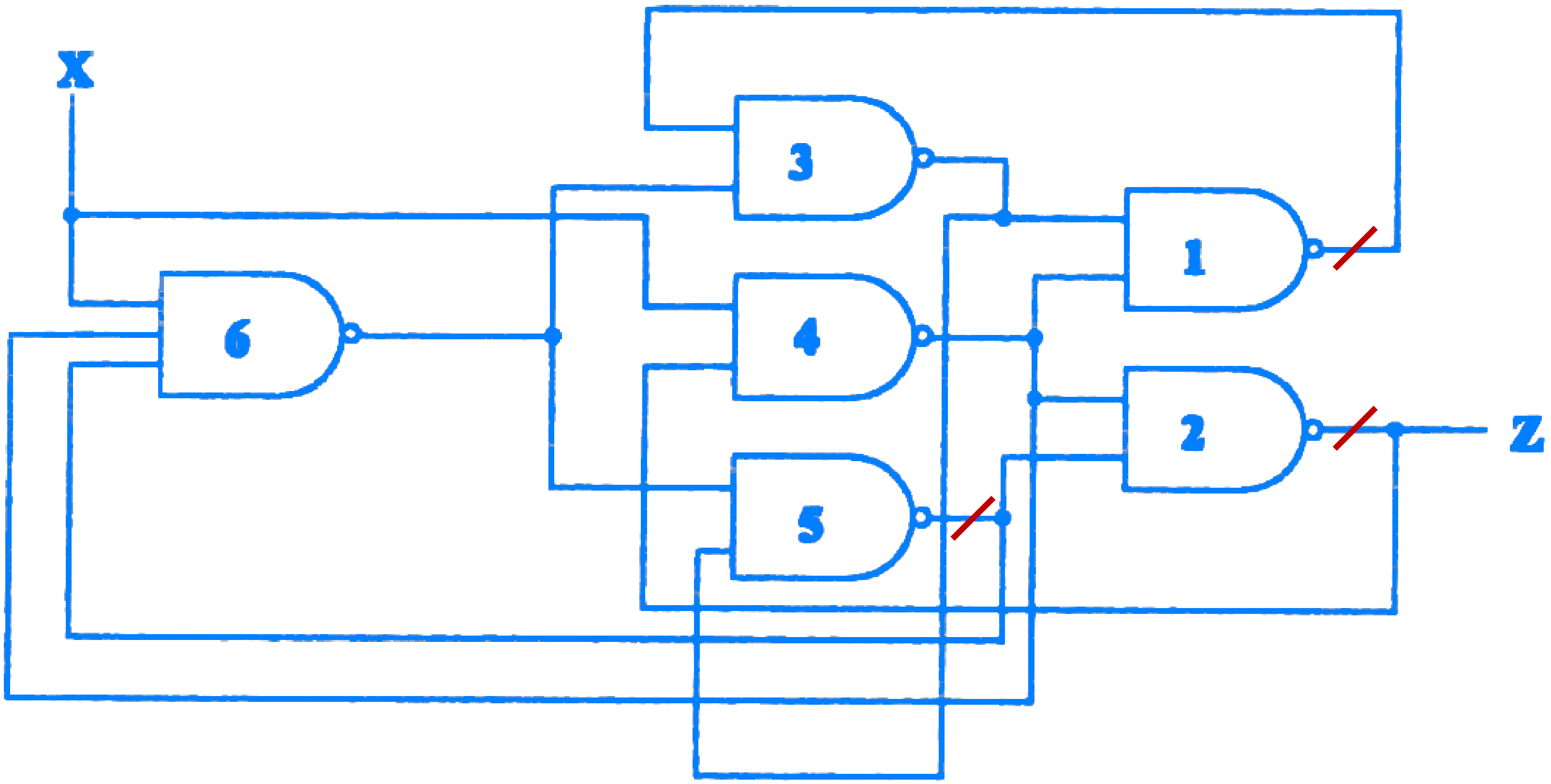




$$Y=\overline{\overline{x_1x_2}\ \overline{x_2y}\ \overline{y\overline{x}_1}}=x_1x_2+x_2y+y\overline{x}_1$$

$$z=y$$

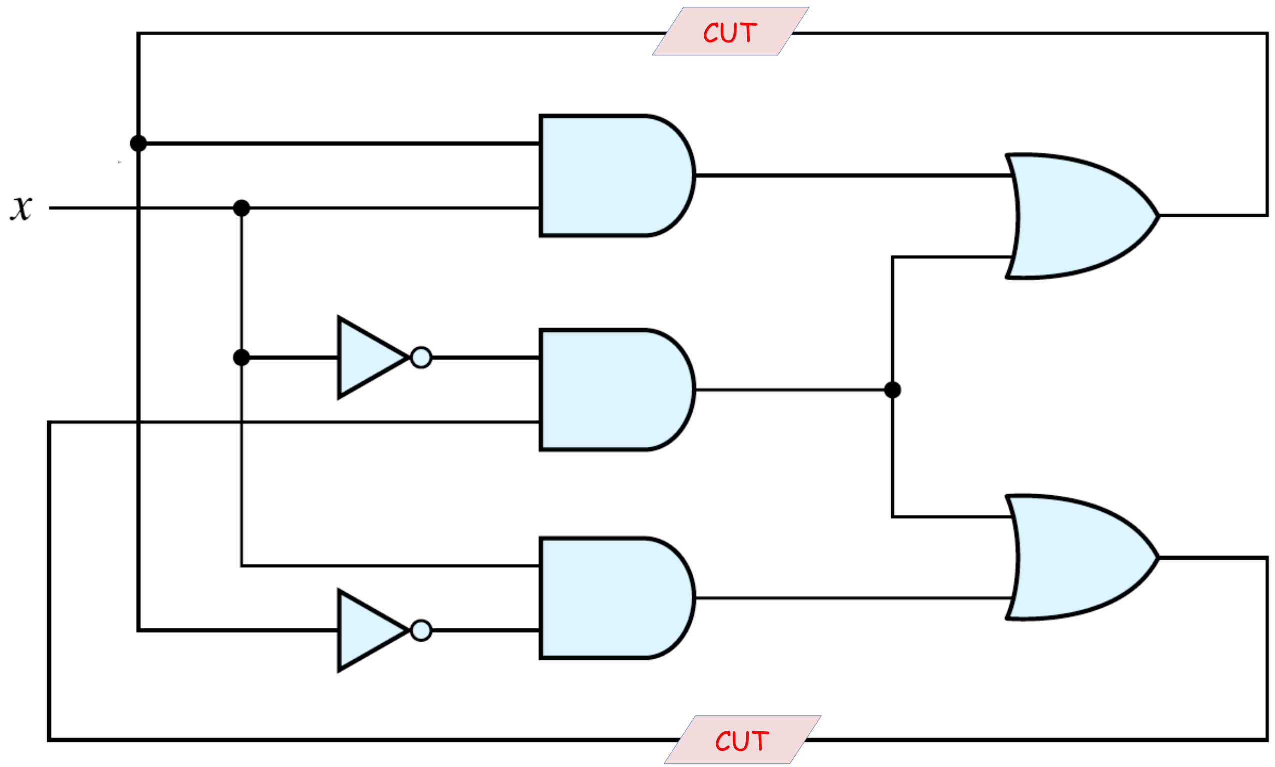
Asynchronous Sequential Analysis



Asynchronous Sequential Circuit

✓ The state variables: Y_1 and Y_2

- $Y_1 = xy_1 + \overline{x}y_2$
- $Y_2 = x\overline{y}_1 + \overline{x}y_2$



Transition Table

- Combine the internal state with input variables
  - Stable total states:
    y_1y_2x = 000, 011, 110 and 101

$Y_1 = xy_1 + \overline{x}y_2$

$Y_2 = x\overline{y}_1 + \overline{x}y_2$

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)

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 example , the circuit has
  - 4 **stable total states**: (y_1y_2x= 000, 011, 110, and 101)
  - 4 **unstable total states**: (y_1y_2x= 001, 010, 111, and 100)

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

Transition Table

- ✓ If y=00 and x=0➡Y=00 (Stable state)
- ✓ 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 alternates between 0 and 1, the circuit will repeat the sequence of states:

00 → 01 → 11 → 10

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

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.

$y \backslash x$	0	1
a	a	b
b	c	b
c	c	d
d	a	d

(a) Four states with one input

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

(b) Two states with two inputs and one output

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.

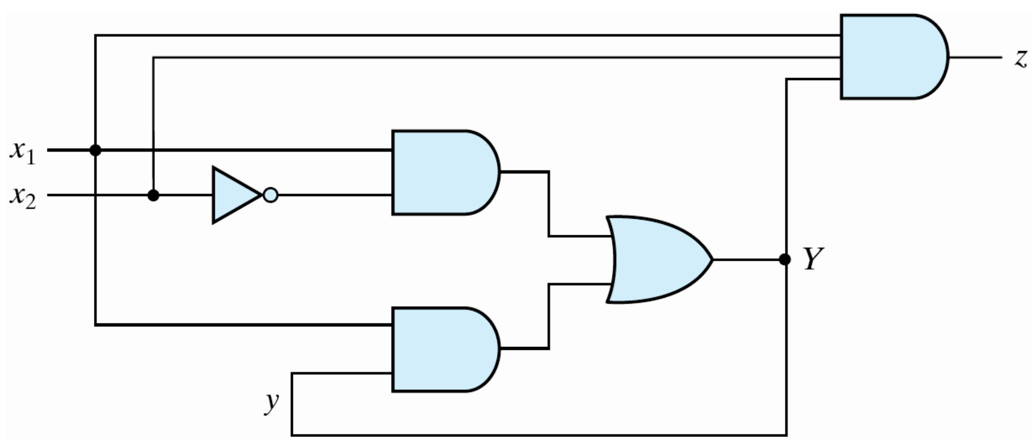
x_1x_2	00	01	11	10
a	a, 0	a, 0	a, 0	b, 0
b	a, 0	a, 0	b, 1	b, 0

Assignements:
A≡0 B≡1

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

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

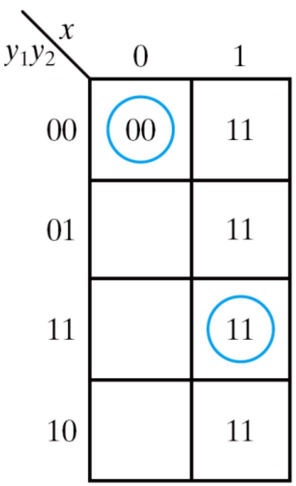
- ✓ This can be done through the assignment of a distinct binary value to each state.



(c) Logic diagram

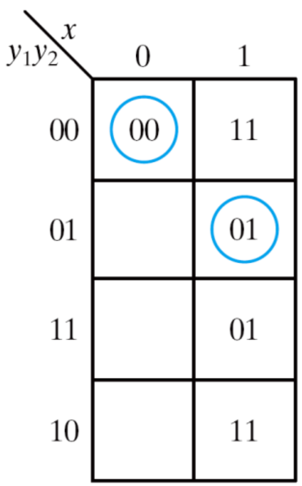
Race condition

- ✓ Two or more binary state variables will change value when one input variable changes.
- ✓ Cannot predict state sequence if unequal delay is encountered.
- ✓ **Non-critical race**: The final stable state does not depend on the change order of state variables
- ✓ **Critical race**: The change order of state variables will result in different stable states. Must be avoided !!



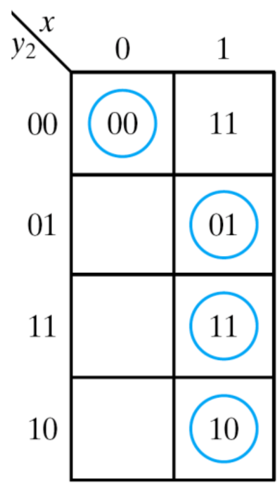
(a) Possible transitions:

00 ⟶ 11
00 ⟶ 01 ⟶ 11
00 ⟶ 10 ⟶ 11



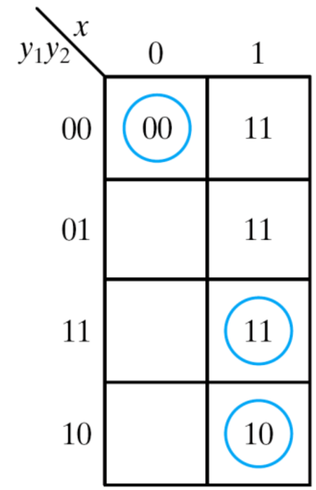
(b) Possible transitions:

00 ⟶ 11 ⟶ 01
00 ⟶ 01
00 ⟶ 10 ⟶ 11 ⟶ 01



(a) Possible transitions:

00 ⟶ 11
00 ⟶ 01
00 ⟶ 10

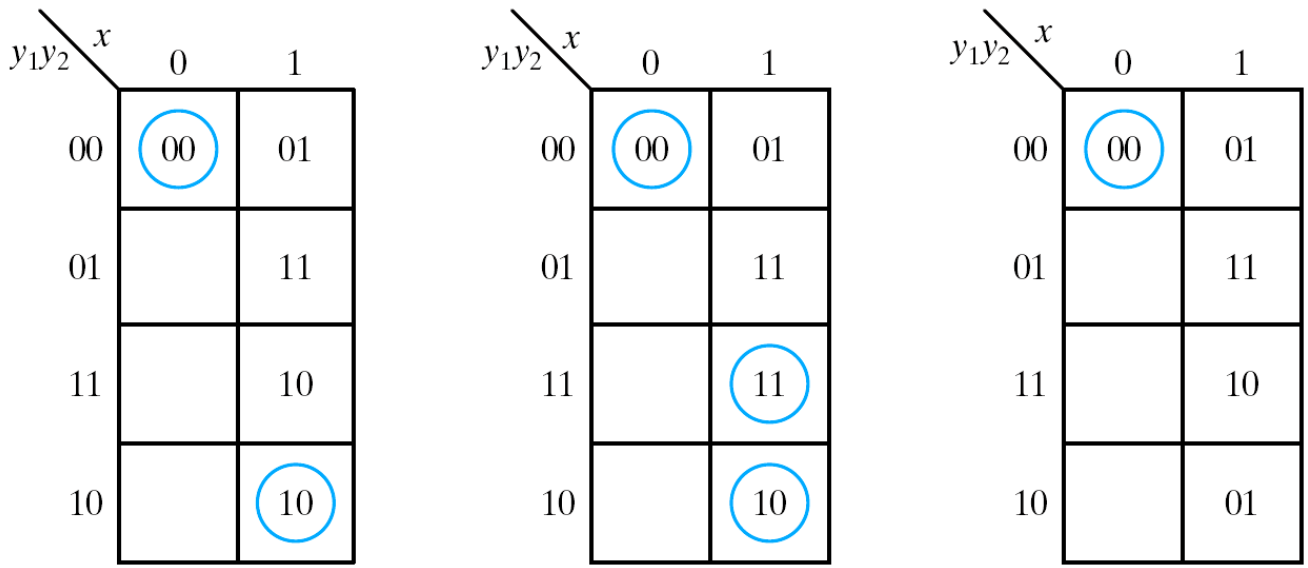


(b) Possible transitions:

00 ⟶ 11
00 ⟶ 01 ⟶ 11
00 ⟶ 10

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.



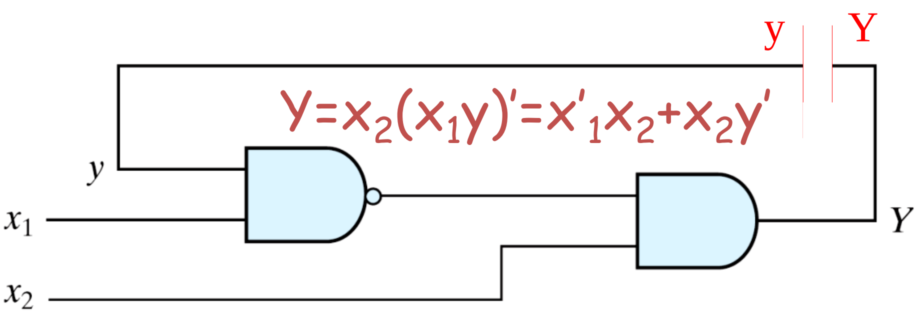
(a) State transition: 00 → 01 → 11 → 10

(b) State transition: 00 → 01 → 11

(c) Unstable → 01 → 11 → 10 ↩

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 x_1x_2=11 in (b), Y and y are never the same



(a) Logic diagram

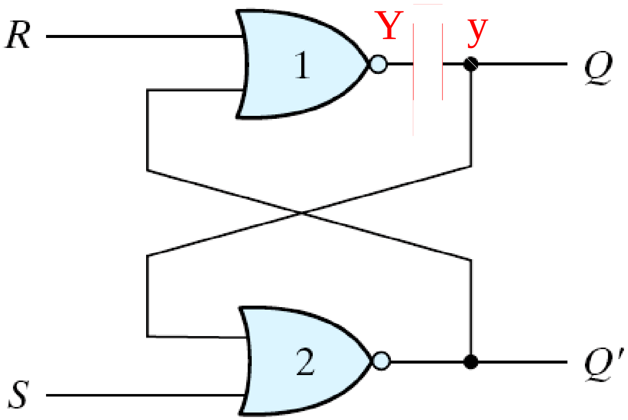
$y \backslash x_1x_2$	00	01	11	10
0	0	1	1	0
1	0	1	0	0

(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 an asynchronous circuit
  - So will be analyzed first using the method for asynchronous circuits.

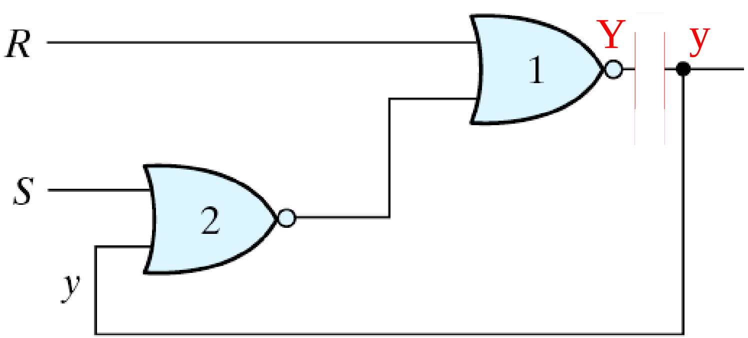
SR Latch with NOR Gates



(a) Cross-coupled circuit

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

(b) Truth table



(c) Circuit showing feedback

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

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

(d) Transition table

Constraints on Inputs

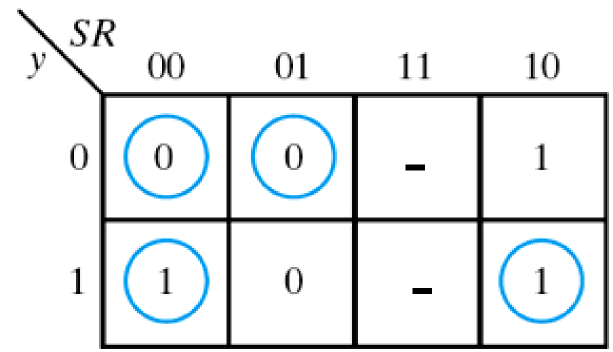
The condition to be avoided is that both S and R inputs must not be 1 simultaneously. This condition is avoided when $SR = 0$ (i.e., ANDing of S and R must always result in 0).

When $SR = 0$ holds at all times, the excitation function derived previously:

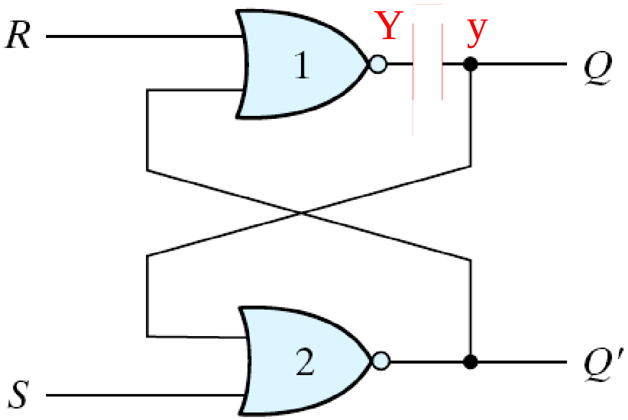
$$Y = SR' + R'y$$

can be expressed as:

$$\boxed{Y = S + R'y}$$



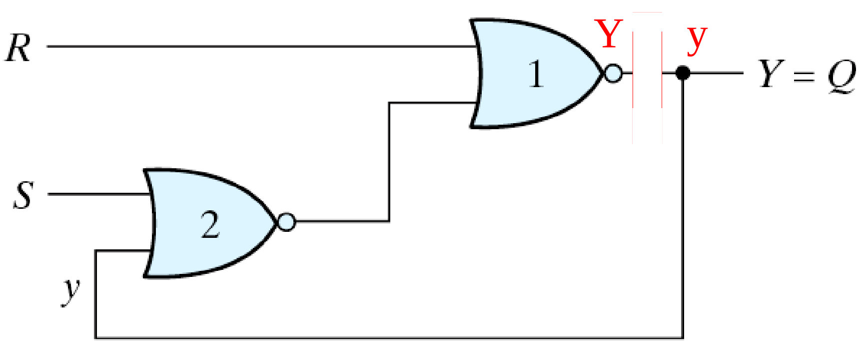
SR Latch with NOR Gates



(a) Cross-coupled circuit

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

(b) Truth table



(c) Circuit showing feedback

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

$Y = SR' + R'y$

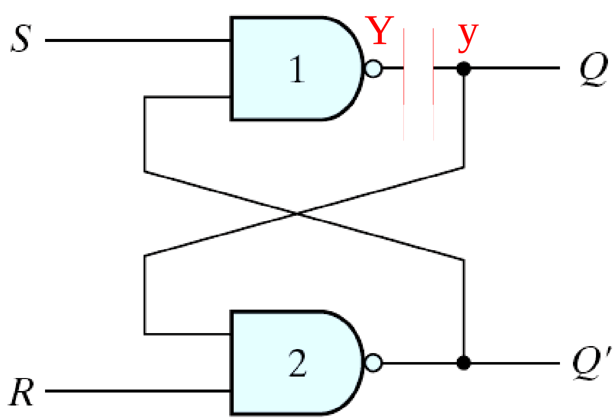
$Y = S + R'y$ when $SR = 0$ ⇨

(d) Transition table

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

should be carefully checked first

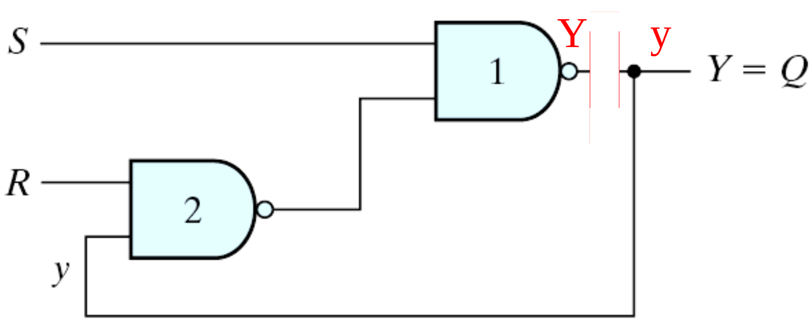
SR Latch with NAND Gates



(a) Cross-coupled circuit

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

(b) Truth table



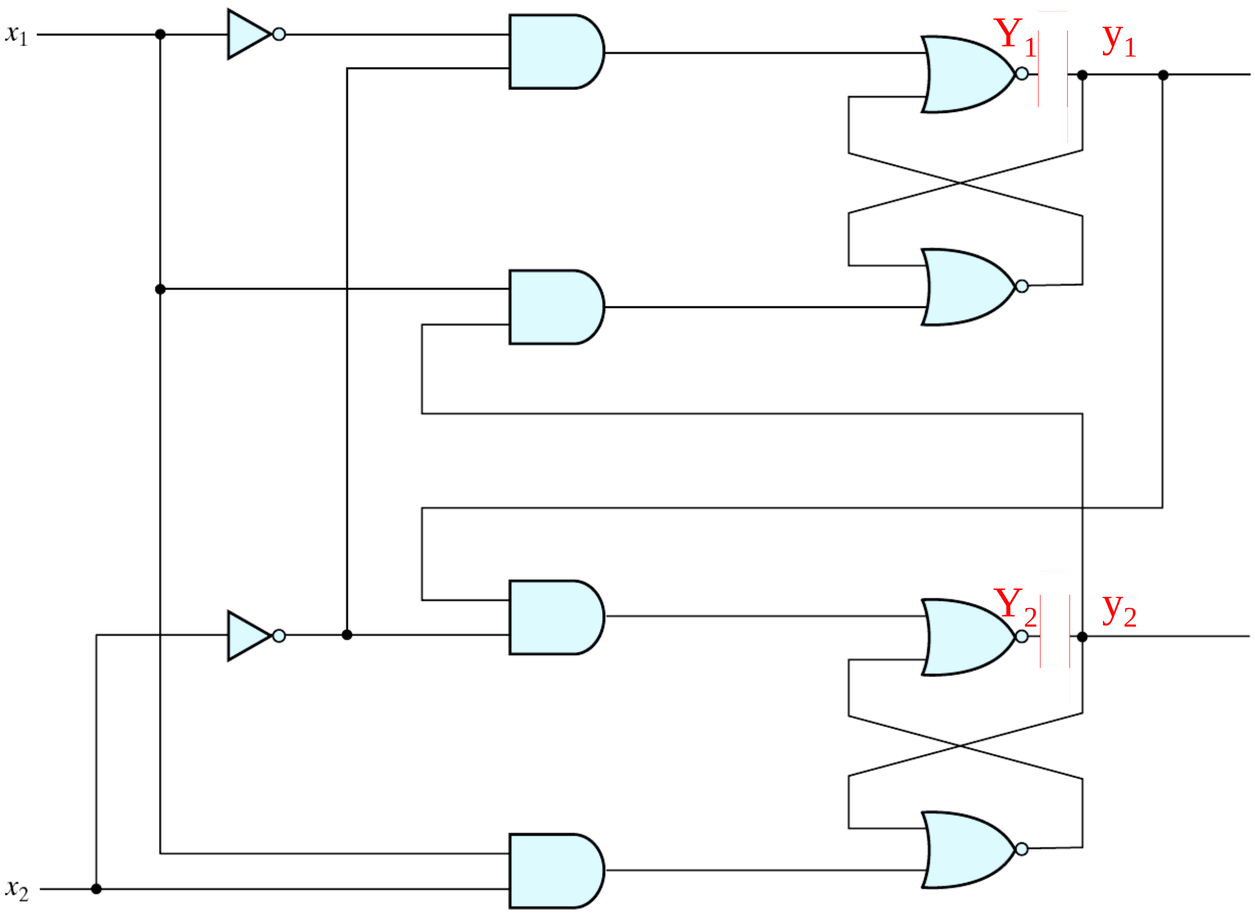
(c) Circuit showing feedback

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

(d) Transition table

S=0, R=0 (S+R=0) should not be used ⇒ S+R=1 is normal mode (eq. S’R’=0) **should be carefully** checked first, so it is obtained
Y = S′ + Ry

Analysis Example



Analysis Example

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

- Label each latch output with Y_i and its external feedback path with y_i for i=1,2,…,k
- Derive the Boolean functions for the S_i and R_i inputs in each latch.

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

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

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

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

Analysis Example

- 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)

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

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

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

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

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

Analysis Example

- Construct a map, with the y's representing the rows and the x inputs representing the columns.
- Plot the value of $Y=Y_1Y_2 \ldots Y_k$ in the map.
- 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 $y_1y_2x_1x_2$ = 1101 and x_2 changes from 1 to 0.
- The circuit should go to the total state 0000.
- If Y_1 changes to 0 before Y_2, 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

$$Y_1 = x_1y_2 + x_1y_1 + x_2y_1$$
$$Y_2 = x_1x_2 + x_2y_2 + y_1'y_2$$

Implementation Procedure

- ✓ Procedure to implement an asynchronous sequential circuits with SR latches:
  - Given a transition table that specifies the excitation function $Y = f(y_1,-,y_n, x_1,-, x_m)$ derive a pair of maps for each S_i and R_i using the latch excitation table
  - Derive the Boolean functions for each S_i and R_i (do not to make S_i and R_i equal to 1 in the same minterm square; for NAND latch, use the complemented values)
  - Draw the logic diagram using k latches together with the gates required to generate the S and R

Implementation Example

- Given a transition table $Y = f(y_1,-,y_n, x_1,-, x_m)$, then the general procedure for implementing a circuit with SR latches is specified by the excitation function, and can be summarized as follows:
  - Given a transition table

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

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

  - Determine the Boolean functions for the S and R inputs of each latch (this is done by using the latch excitation table)

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

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

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

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

Implementation Example

- From maps: the simplified Boolean functions are

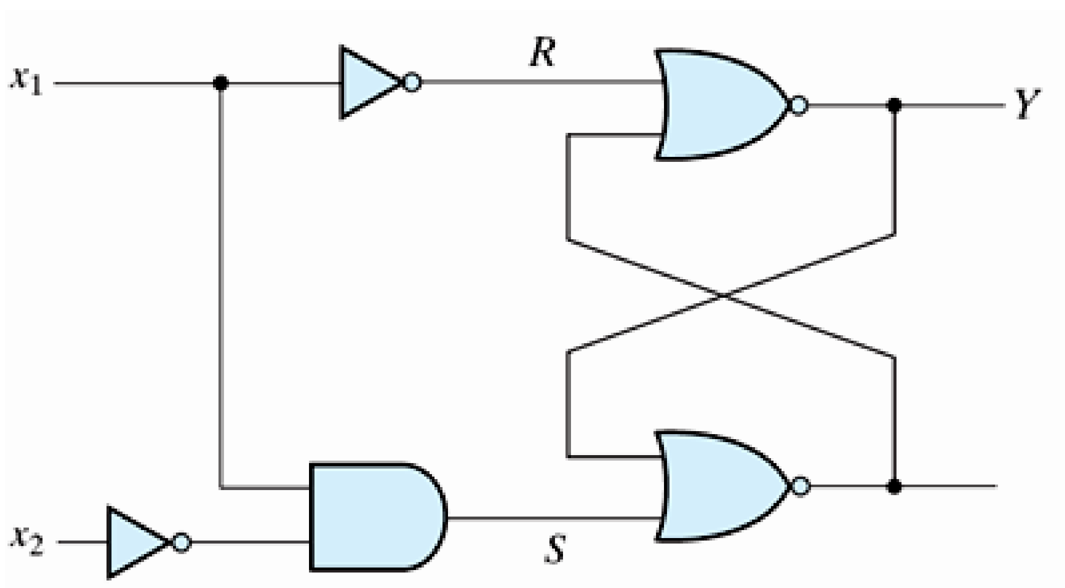
$$S = x_1 x_2' \qquad \text{and} \qquad R = x_1'$$

→ NOR latch

- Check whether SR=0 for each NOR latch or whether S'R'=0 for each NAND latch:

$$SR = x_1 x_2' x_1' = 0$$

- 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)



Primitive Flow Table

- ✓ Primitive flow table - has exactly one stable total state (internal state + input) per row
- ✓ To avoid the timing problems:
  - Only one input variable changes at a time
  - Networks reach a stable total state between input changes (Fundamental Mode)
- ✓ Every change in input changes the state

Design procedure

1. Obtain a **primitive table** from specifications
2. Reduce flow table by merging rows in the primitive flow table
3. Assign binary state variables to each row of reduced table
4. Assign output values to dashes associated with unstable states to obtain the output map
5. Simplify Boolean functions for excitation and output variables;
6. Draw the logic diagram

Design Example 1:

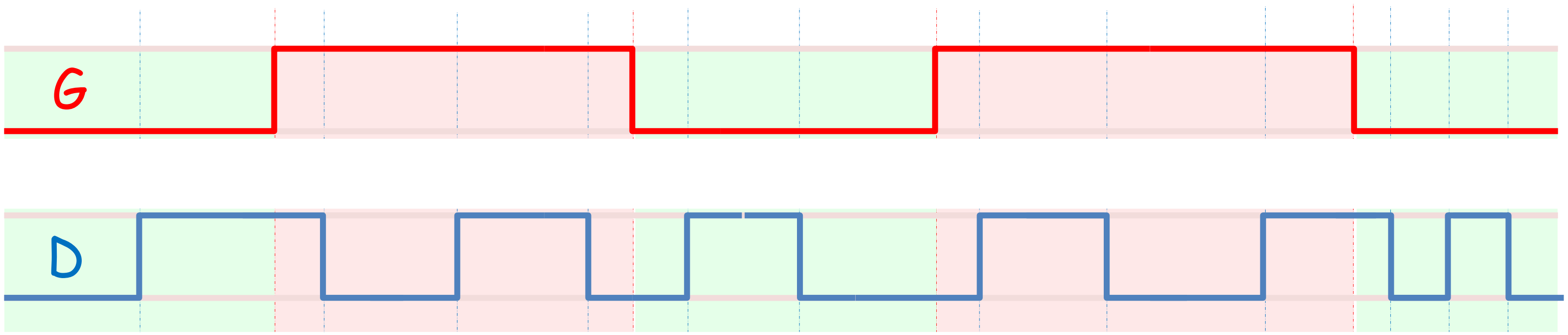
- Problem Statement:
  - Design a gated latch circuit (memory element) with two inputs, G(gate) and D(Data) and one output Q.
  - The Q output will follow the D input as long as G=1. When G goes to 0, the information that was present at the D input at the time of transition is retained at the Q output.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

Design Example 1:

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, combinations of the inputs and internal states, simultaneous transitions of two input variables are not allowed

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

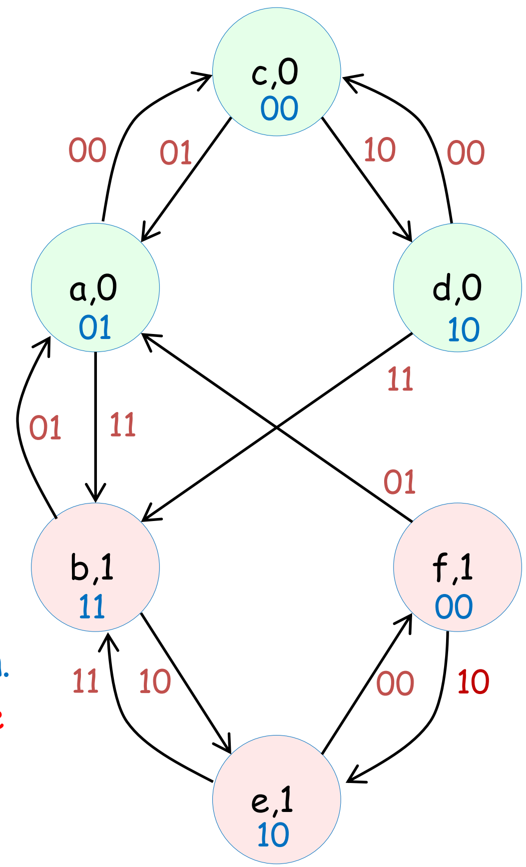


	Inputs		Output	
State	**D**	**G**	**Q**	**Comments**
a	0	1	0	$D = Q$ because $G = 1$
b	1	1	1	$D = Q$ because $G = 1$
c	0	0	0	After state a or d
d	1	0	0	After state c
e	1	0	1	After state b or f
f	0	0	1	After state e

Design Example 1

1-Primitive Flow Table

- One square in each row is a stable state for that row.
- First, 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 the two squares adjacent to the stable state in each row.
  - the previous table may support in deriving the necessary information.
- All outputs associated with unstable states are don't care conditions
  - We marked them with a dash.



	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 1

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.

Candidates states for merging:

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

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

- 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

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

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

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

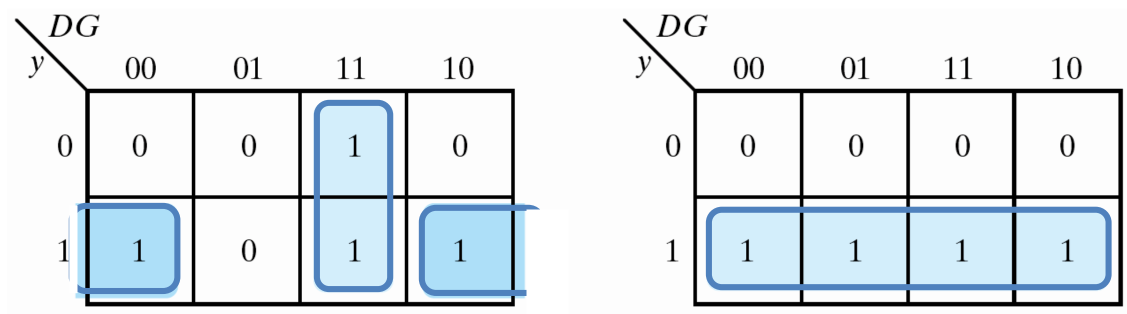
Design Example 1

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.

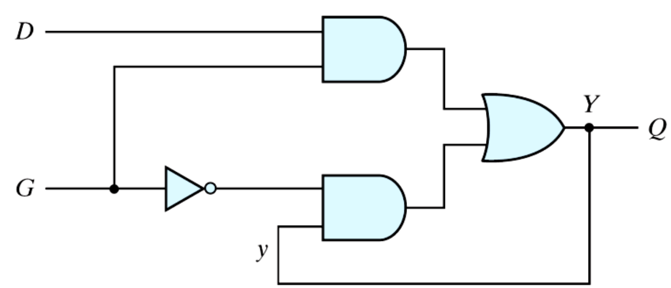
a=0, b=1 in this example

Transition table and output map



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

Gated-latch logic diagram



Design Example 1

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

4. Implementation with SR Latch

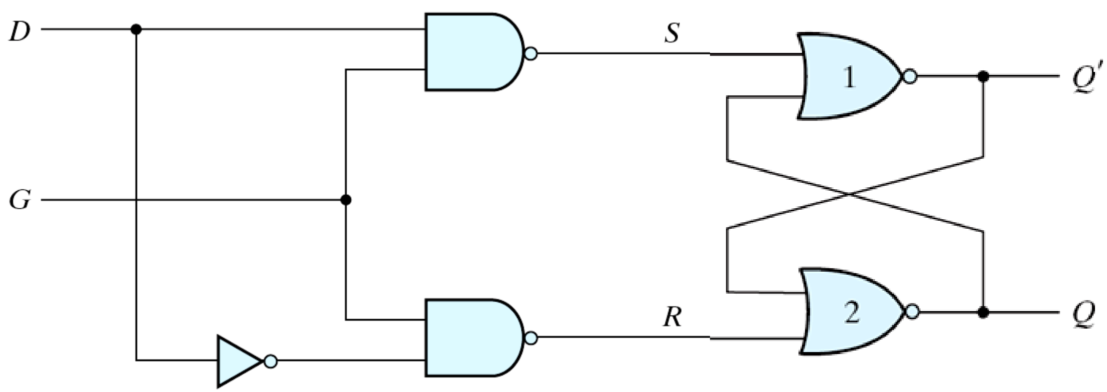
y \ DG	00	01	11	10
0	0	0	1	0
1	X	0	X	X

(a) $S = DG$

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

$R = D'G$

Circuit with *SR* latch



(b) Logic diagram

Design Example:

5- *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 are 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 1

5- Assigning Outputs to Unstable States

Example:

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

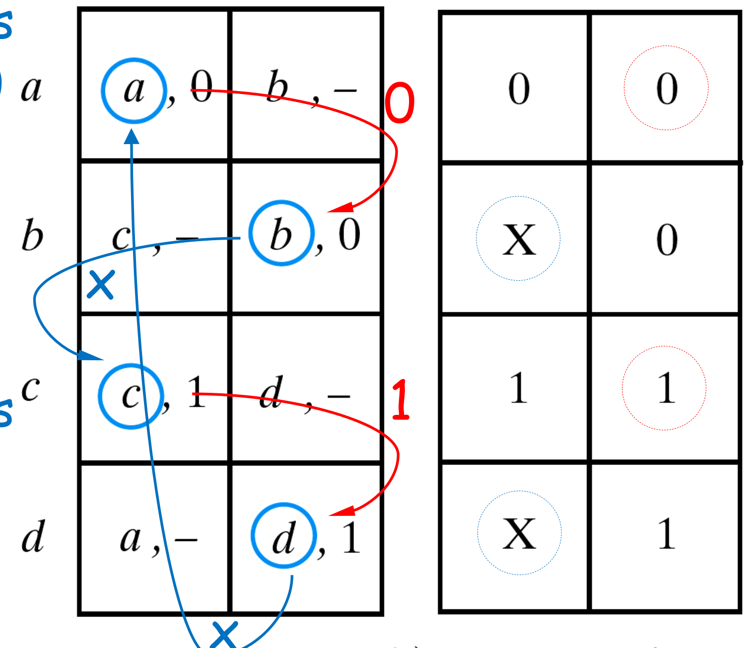
  the transient unstable state b in the first row must have the same output value = 0

  if c changes to d same for 1⇨1: 1

- If b changes to c, the two stable states have different output values 0⇨1: x

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

  if d changes to a same for 1⇨0: x



(a) Flow table b) Output assignment