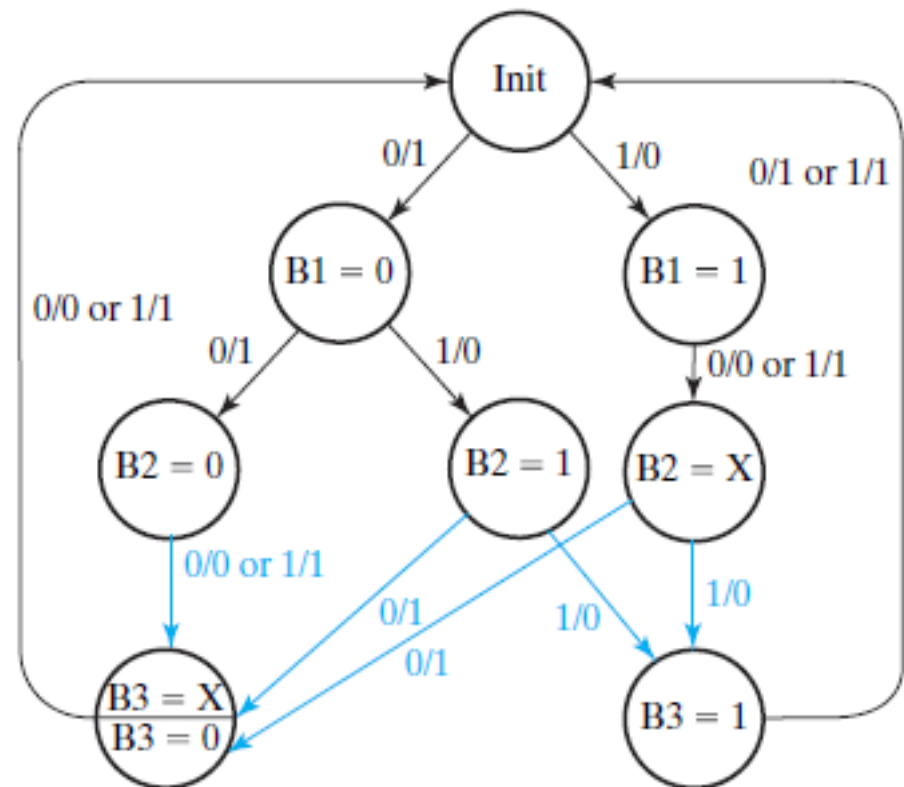


Sequential Code Converter BCD-Excess-3

BCD Input				Excess-3 Output			
1	2	3	4	1	2	3	4
0	0	0	0	1	1	0	0
0	0	0	1	1	1	0	1
0	0	1	0	1	1	1	0
0	1	0	0	1	0	1	0
0	1	1	0	1	0	0	1
1	0	0	0	0	0	1	0
1	0	0	1	0	0	1	1
1	0	1	0	0	0	0	1
1	1	0	0	0	1	1	0
1	1	1	0	0	1	0	1



Equivalent State Definitions

- ✓ Two states are **equivalent** if their response for each possible input sequence is an identical output sequence.
- ✓ Alternatively, two states are equivalent if their outputs produced for each input symbol is identical and their next states for each input symbol are the same or equivalent.
- ✓ Two states that are not equivalent are **distinguishable**
- ✓ The checking of each pair of states for possible equivalence in a table with a large number of states can be done systematically on an **Implication Table**.
- ✓ The **Implication Table** is a chart that consists of squares, one for every possible pair of states.

Implication Table (Example):

- ✓ Place a cross in any square corresponding to a pair whose outputs are not equal
- ✓ Enter in the remaining squares the pairs of states that are implied by the pair of states representing the squares. (Start from the top square in the left column and going down and then proceeding with the next column to the right).
- ✓ Make successive passes through the table to determine whether any additional squares should be marked with a 'x'.
- ✓ Finally, all the squares that have no crosses are recorded with check marks.

Present State	Next State		Output	
	X=0	X=1	X=0	X=1
<i>a</i>	<i>d</i>	<i>b</i>	0	0
<i>b</i>	<i>e</i>	<i>a</i>	0	0
<i>c</i>	<i>g</i>	<i>f</i>	0	1
<i>d</i>	<i>a</i>	<i>d</i>	1	0
<i>e</i>	<i>a</i>	<i>d</i>	1	0
<i>f</i>	<i>c</i>	<i>b</i>	0	0
<i>g</i>	<i>a</i>	<i>e</i>	1	0

<i>b</i>	<i>d, e</i> ✓					
<i>c</i>	x	x				
<i>d</i>	x	x	x			
<i>e</i>	x	x	x	✓		
<i>f</i>	<i>c, d</i> x	<i>c, e</i> x <i>a, b</i>	x	x	x	
<i>g</i>	x	x	x	<i>d, e</i> ✓	<i>d, e</i> ✓	x
	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>

Implication Table (Example):

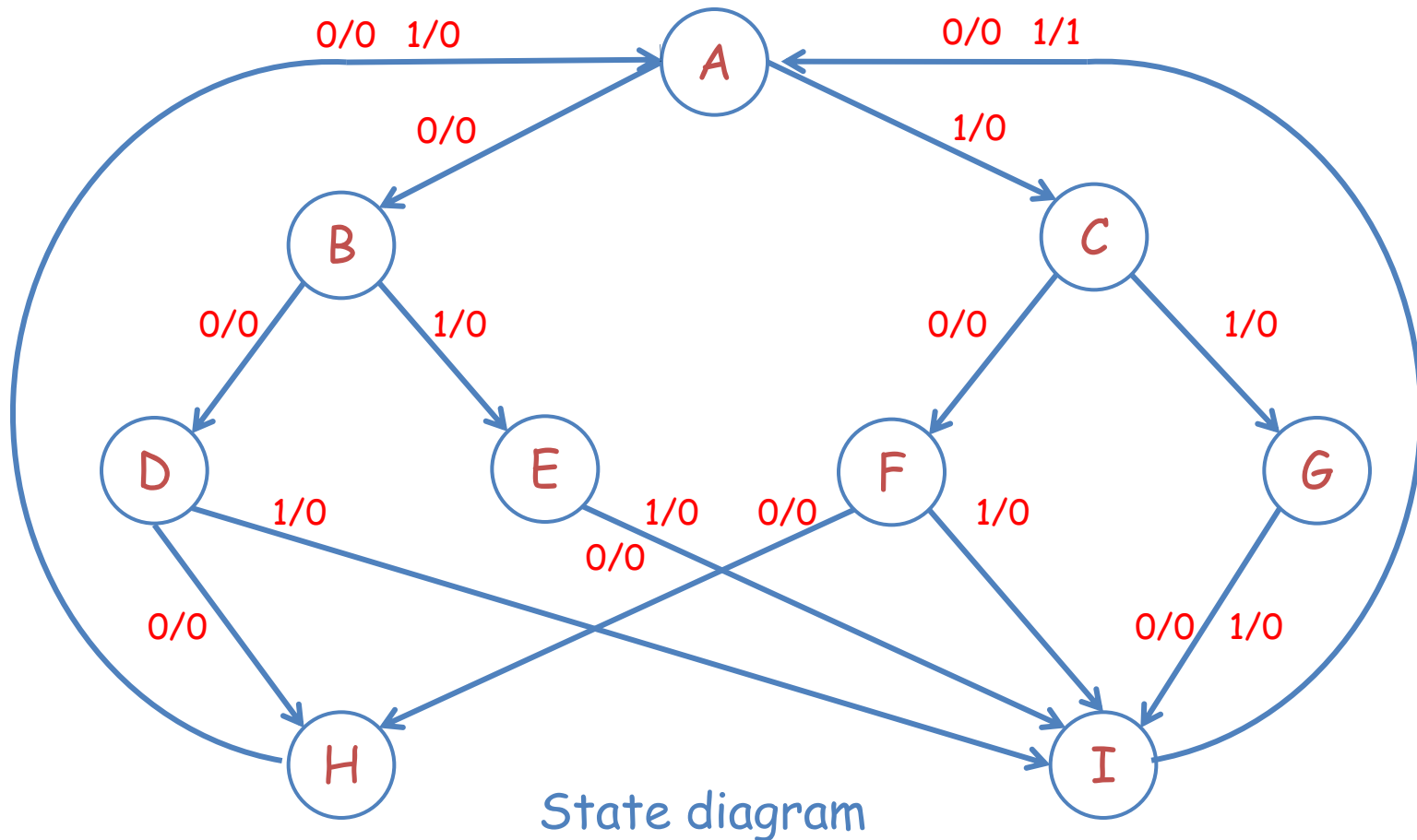
- ✓ Its clear that (e,d) are equivalent. And this leads (a,b) and (e,g) to be equivalent too.
- ✓ Finally we have [(a,b) , c , (e,d,g) , f] so four states.
- ✓ So the original flow table can be reduced to:

b	d, e ✓					
c	x	x				
d	x	x	x			
e	x	x	x	✓		
f	c, d x	c, e x a, b	x	x	x	
g	x	x	x	d, e ✓	d, e ✓	x
	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

Elimination of Redundant States

- ✓ When first setting up the state table, we will not be overly concerned with inclusion of extra states, and when the table is complete, we will eliminate any redundant states.
- ✓ Design a binary checker that has in input a sequence of BCD numbers and for every four bits (LSB order) has output 0 if the number is $0 \leq N \leq 9$ and 1 if $10 \leq N \leq 15$



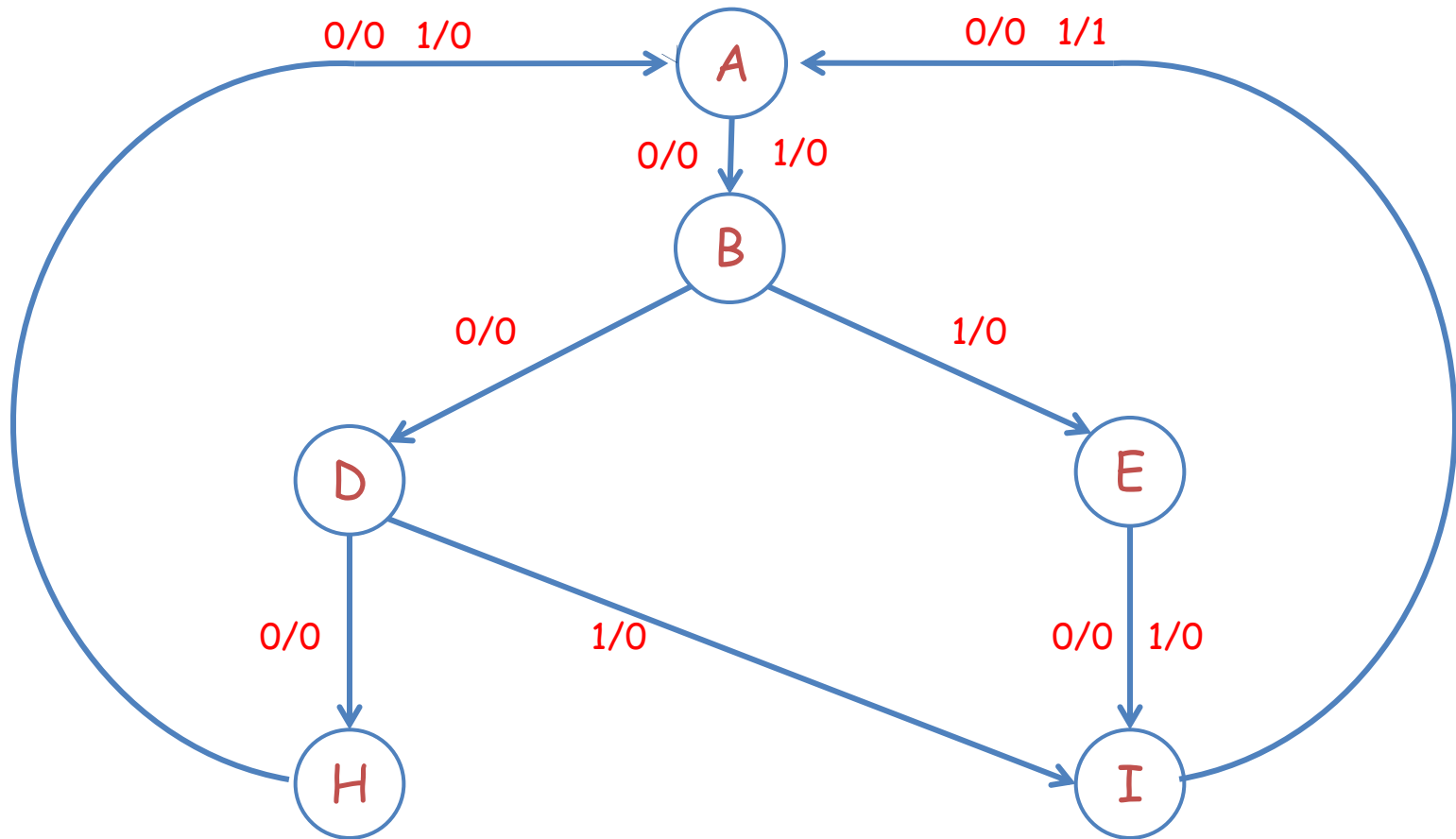
Elimination of Redundant States

Input Sequence	Present State	Next State		Present Output	
		$X = 0$	$X = 1$	$X = 0$	$X = 1$
reset	<i>A</i>	<i>B</i>	<i>C</i>	0	0
0	<i>B</i>	<i>D</i>	<i>E</i>	0	0
1	<i>C</i>	<i>F</i>	<i>G</i>	0	0
00	<i>D</i>	<i>H</i>	<i>I</i>	0	0
01	<i>E</i>	<i>I</i>	<i>I</i>	0	0
10	<i>F</i>	<i>H</i>	<i>I</i>	0	0
11	<i>G</i>	<i>I</i>	<i>I</i>	0	0
000 100	<i>H</i>	<i>A</i>	<i>A</i>	0	0
001 101	<i>I</i>	<i>A</i>	<i>A</i>	0	1
010 110					
011 111					

$\{B,C\}, \{D,F\}, \{E,G\}$

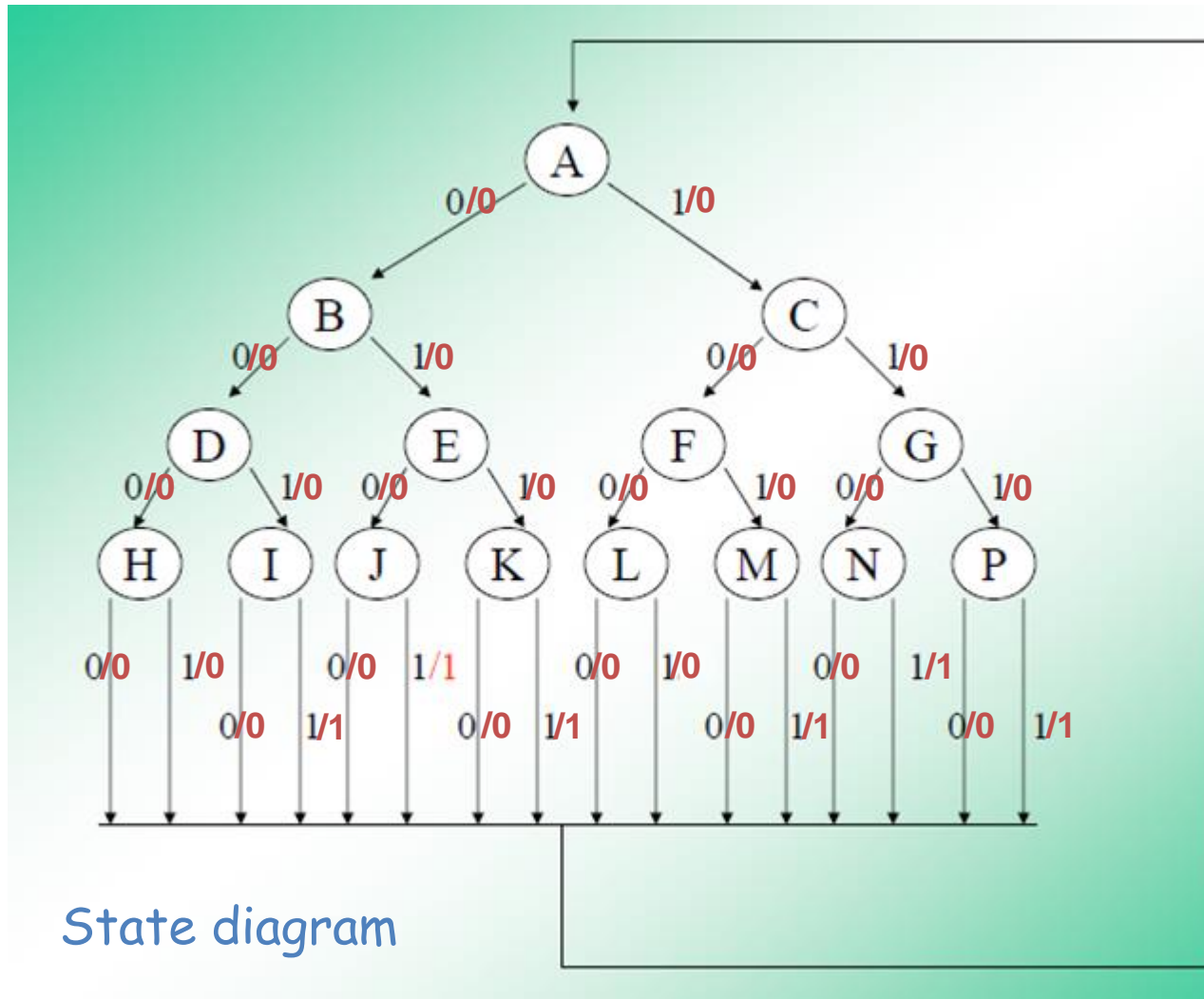
State table

Elimination of Redundant States



State diagram

Elimination of Redundant States



Elimination of Redundant States

- ✓ When first setting up the state table, we will not be overly concerned with inclusion of extra states, and when the table is complete, we will eliminate any redundant states.

Input Sequence	Present State	Next State		Present Output	
		$X = 0$	$X = 1$	$X = 0$	$X = 1$
reset	<i>A</i>	<i>B</i>	<i>C</i>	0	0
0	<i>B</i>	<i>D</i>	<i>E</i>	0	0
1	<i>C</i>	<i>F</i>	<i>G</i>	0	0
00	<i>D</i>	<i>H</i>	<i>I</i>	0	0
01	<i>E</i>	<i>J</i>	<i>K</i>	0	0
10	<i>F</i>	<i>L</i>	<i>M</i>	0	0
11	<i>G</i>	<i>N</i>	<i>P</i>	0	0
000	<i>H</i>	<i>A</i>	<i>A</i>	0	0
001	<i>I</i>	<i>A</i>	<i>A</i>	0	1
010	<i>J</i>	<i>A</i>	<i>A</i>	0	1
011	<i>K</i>	<i>A</i>	<i>A</i>	0	1
100	<i>L</i>	<i>A</i>	<i>A</i>	0	0
101	<i>M</i>	<i>A</i>	<i>A</i>	0	1
110	<i>N</i>	<i>A</i>	<i>A</i>	0	1
111	<i>P</i>	<i>A</i>	<i>A</i>	0	1

Synchronous Sequential Circuit

- ✓ The change of internal state occurs in response to the synchronized clock pulses.
- ✓ Input changes occur between clock pulses
- ✓ Data are read during the clock pulse
- ✓ It is supposed to wait long enough after the external input changes for all flip-flop inputs to reach a steady value before applying the new clock pulse
- ✓ Unsuitable Situations:
 - Inputs change at any time and cannot be synchronized with a clock
 - Circuit is large, Clock skew can not be avoided
 - High performance design

Asynchronous Circuits

- ✓ Not synchronized by a common clock
- ✓ States change immediately after input changes
- ✓ The circuit reaches a **steady-state condition** when $y_i = Y_i$ for all i .
- ✓ For a given value of input variables, **the system is stable if the circuit reaches a steady state condition.**
- ✓ 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

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.

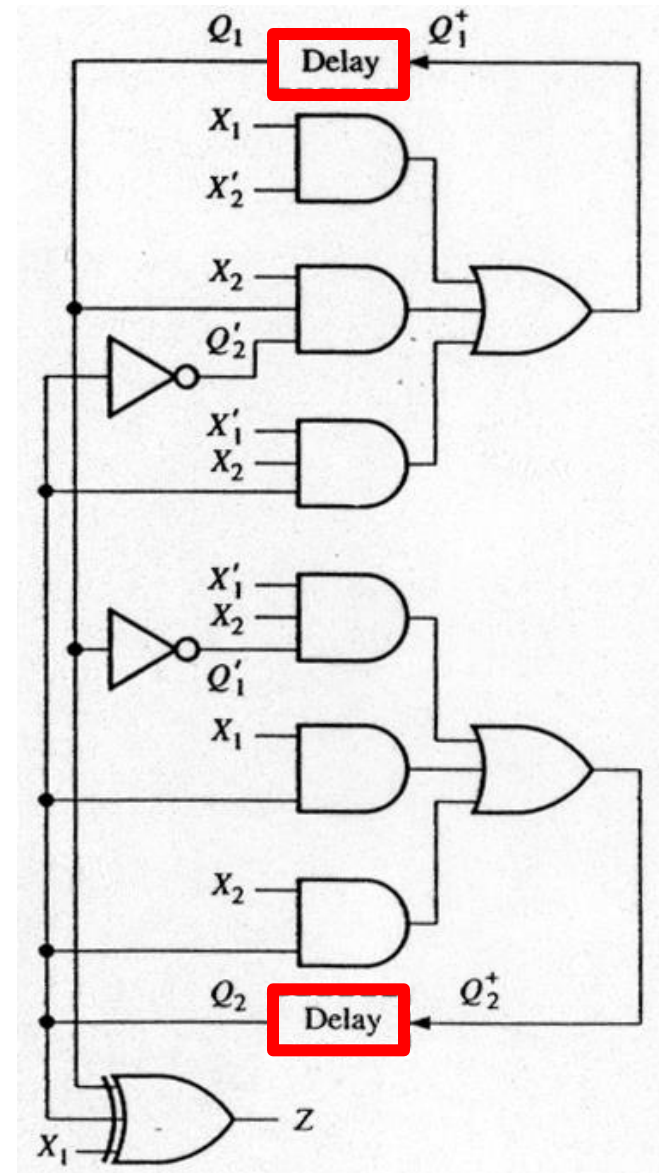
Example Circuit

- ✓ 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 can be eliminated by proper design.

$Q_1Q_2 \backslash X_1X_2$		00	01	11	10
		00	01	11	10
00	00	01	00	10	
01	00	11	01	11	
11	00	11	01	11	
10	00	10	10	10	

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

Z

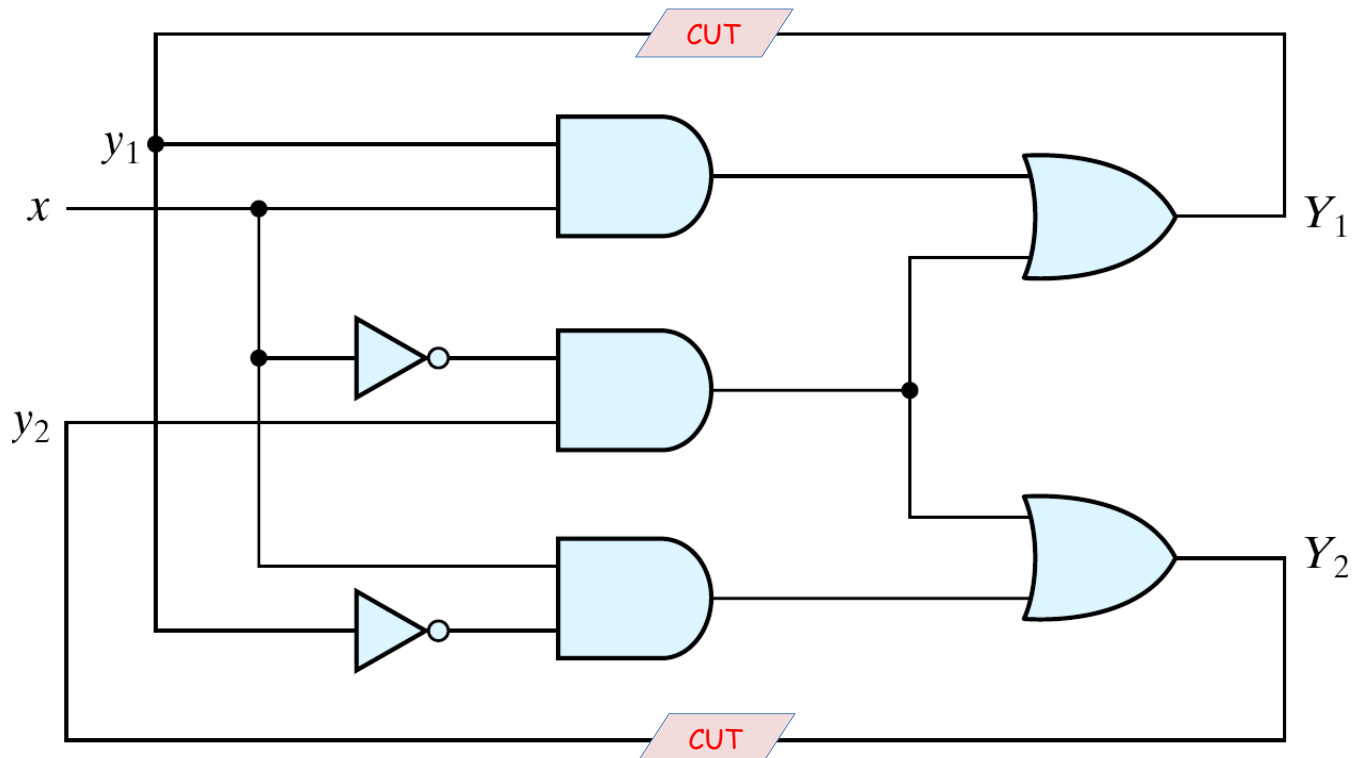
Transition Table

- ✓ 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

Asynchronous Sequential Circuit

✓ The excitation variables: Y_1 and Y_2

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



Transition Table

✓ Combine the internal state with input variables

- Stable total states:

$$y_1 y_2 x = 000, 011, 110 \text{ and } 101$$

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

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

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

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

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

(c) Transition table

Transition Table

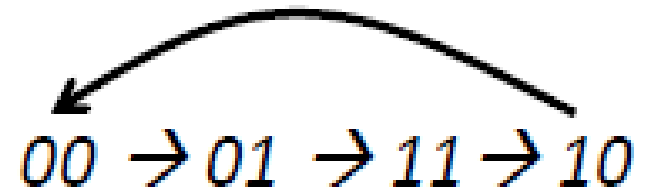
- ✓ In an asynchronous sequential circuit, the internal state can change immediately after a change in the input.
- ✓ It is sometimes convenient to combine the internal state with input value together and call it the **Total State of the circuit**. (Total state = Internal state + Inputs)
- ✓ In the example , the circuit has
 - 4 stable total states: ($y_1y_2x = 000, 011, 110, \text{ and } 101$)
 - 4 unstable total states: ($y_1y_2x = 001, 010, 111, \text{ and } 100$)

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

Transition Table

- ✓ If $y=00$ and $x=0 \Rightarrow Y=00$ (Stable state)
- ✓ If x changes from 0 to 1 while $y=00$, the circuit changes Y to 01 which is temporary unstable condition ($Y \neq y$)
- ✓ As soon as the signal propagates to make $Y=01$, the feedback path causes a change in y to 01. (transition from the first row to the second row)
- ✓ If the input alternates between 0 and 1, the circuit will repeat the sequence of states

		x	
		0	1
$y_1 y_2$	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	\textcircled{a}	b
b	c	\textcircled{b}
c	\textcircled{c}	d
d	a	\textcircled{d}

(a) Four states with one input

$x_1 x_2$	00	01	11	10
a	$\textcircled{a}, 0$	$\textcircled{a}, 0$	$\textcircled{a}, 0$	$b, 0$
b	$a, 0$	$a, 0$	$\textcircled{b}, 1$	$\textcircled{b}, 0$

(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.

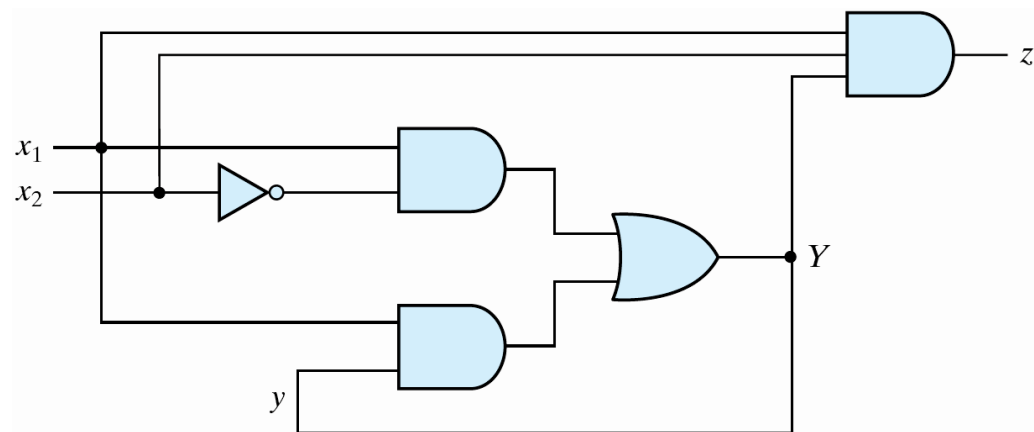
$y \backslash x_1x_2$		00	01	11	10
0	0	0	0	0	1
1	1	0	0	1	1

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

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

(b) Map for output
 $z = x_1x_2y$

- ✓ 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 !!**

		x	
		0	1
y ₁ y ₂	00	00	11
	01		11
	11		11
	10		11

(a) Possible transitions:

00 → 11
 00 → 01 → 11
 00 → 10 → 11

		x	
		0	1
y ₁ y ₂	00	00	11
	01		01
	11		01
	10		11

(b) Possible transitions:

00 → 11 → 01
 00 → 01
 00 → 10 → 11 → 01

		x	
		0	1
y ₂	00	00	11
	01		01
	11		11
	10		10

(a) Possible transitions:

00 → 11
 00 → 01
 00 → 10

		x	
		0	1
y ₁ y ₂	00	00	11
	01		11
	11		11
	10		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.

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

(a) State transition:
 $00 \rightarrow 01 \rightarrow 11 \rightarrow 10$

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

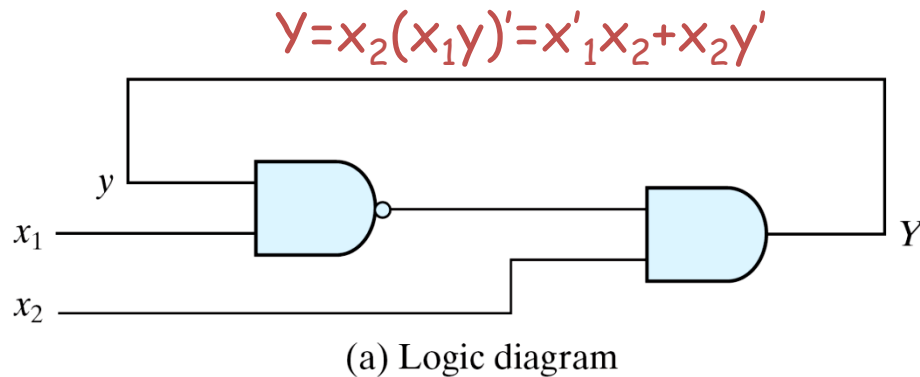
(b) State transition:
 $00 \rightarrow 01 \rightarrow 11$

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

(c) Unstable
 $\rightarrow 01 \rightarrow 11 \rightarrow 10 \rightarrow$

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 \longrightarrow unstable
Ex: when $x_1x_2=11$ in (b), Y and y are never the same



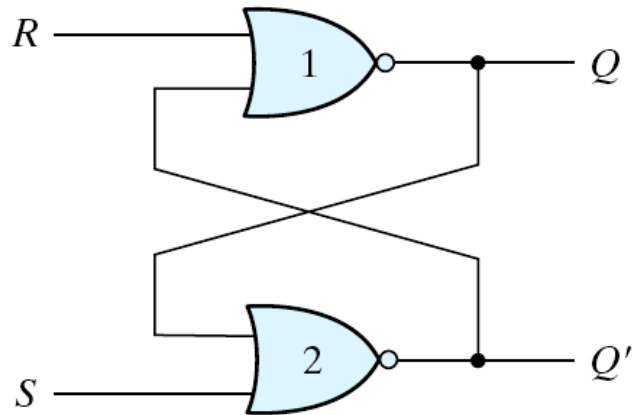
x_1x_2	00	01	11	10
0	0	1	1	0
1	0	1	0	0

(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



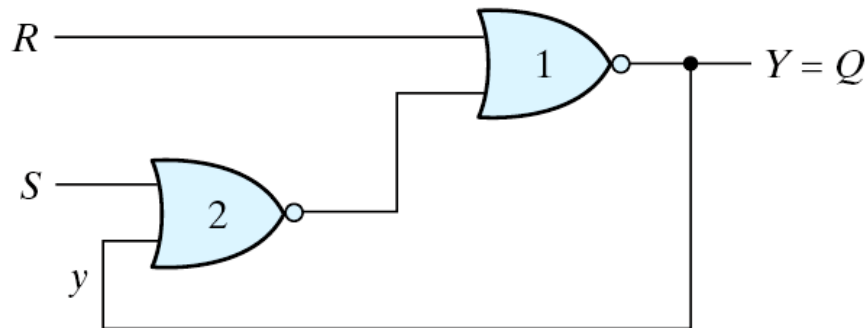
(a) Cross-coupled circuit

S	R	Q	Q'
1	0	1	0
0	0	1	0
0	1	0	1
0	0	0	1
1	1	0	0

(After $SR = 10$)

(After $SR = 01$)

(b) Truth table



(c) Circuit showing feedback

$y \backslash SR$		SR			
		00	01	11	10
y	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).

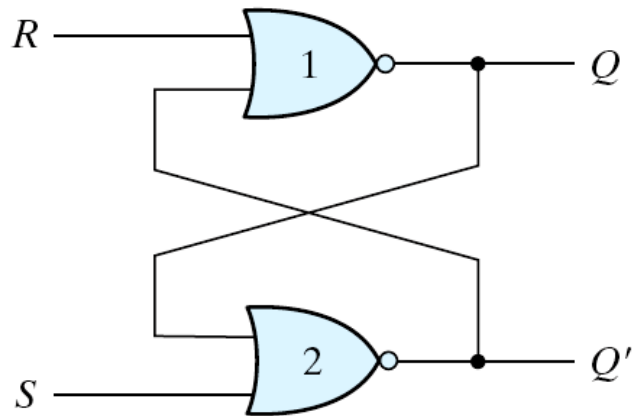
When $SR = 0$ holds at all times, the excitation function derived previously:

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

can be expressed as:

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

SR Latch with NOR Gates



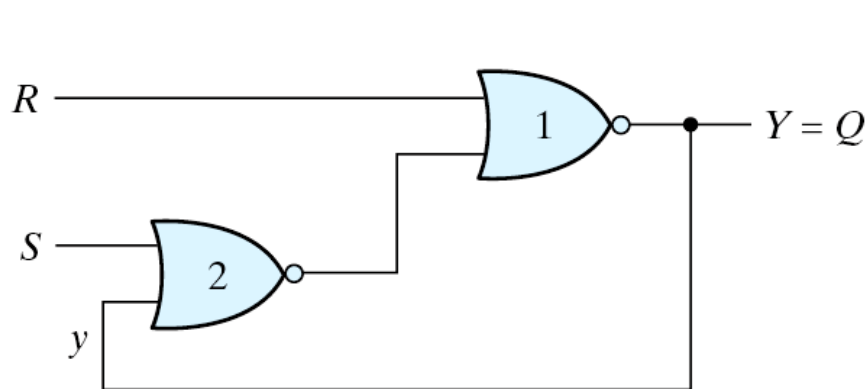
(a) Cross-coupled circuit

S	R	Q	Q'
1	0	1	0
0	0	1	0
0	1	0	1
0	0	0	1
1	1	0	0

(After $SR = 10$)

(After $SR = 01$)

(b) Truth table



(c) Circuit showing feedback

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

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

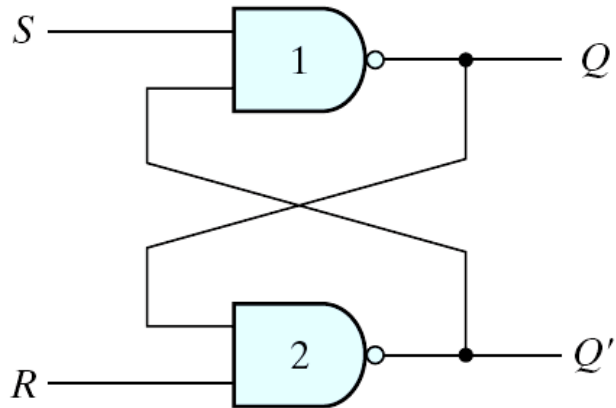
$$Y = SR' + R'y$$

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

**should be carefully
checked first**

(d) Transition table

SR Latch with NAND Gates



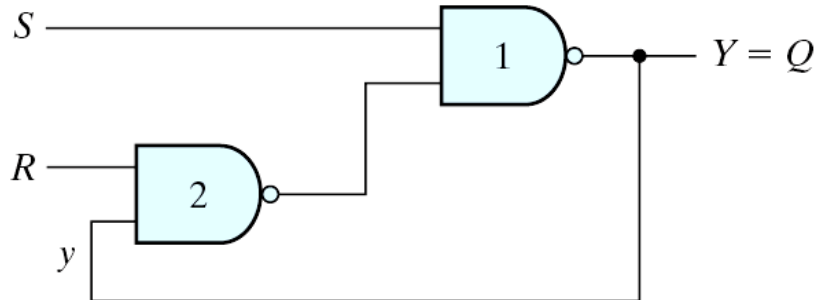
(a) Cross-coupled circuit

S	R	Q	Q'
1	0	0	1
1	1	0	1
0	1	1	0
1	1	1	0
0	0	1	1

(After $SR = 10$)

(After $SR = 01$)

(b) Truth table



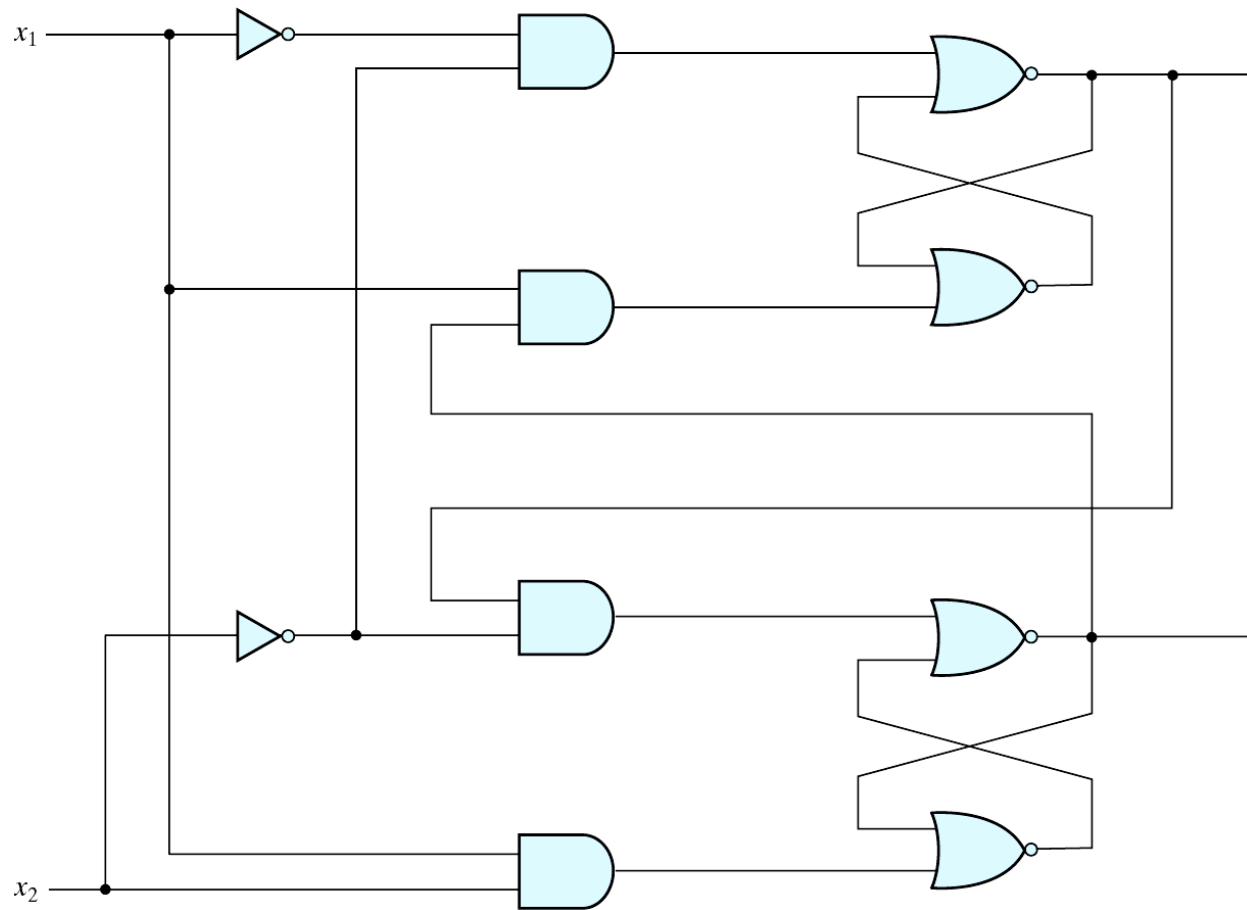
(c) Circuit showing feedback

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

(d) Transition table

$S=0, R=0$ ($S+R=0$)
should not be used
 $\Rightarrow S+R=1$ 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 $i=1,2,\dots,k$
 - Derive the Boolean functions for the S_i and R_i inputs in each latch.

$$S_1 = x_1 y_2$$

$$R_1 = x_1' x_2'$$

$$S_2 = x_1 x_2$$

$$R_2 = x_2' y_1$$

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)

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

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

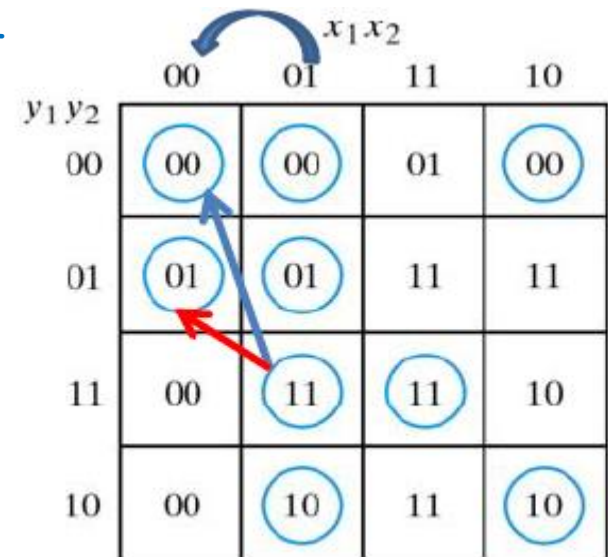
- Evaluate $Y = S + R'y$ for each NOR latch or $Y = S' + Ry$ for each NAND latch.

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

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

Analysis Example

- Construct a map, with the y 's representing the rows and the x inputs representing the columns.
- Plot the value of $Y=Y_1Y_2\dots Y_k$ 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 $y_1y_2x_1x_2 = \underline{1101}$ 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.



		x_1x_2			
		00	01	11	10
y_1y_2	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:
 - Given a transition table that specifies the excitation function $Y = Y_1 Y_2 \dots Y_k$, 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)
 - Draw the logic diagram using k latches together with the gates required to generate the S and R (for NAND latch, use the complemented values in step 2)

Implementation Example

- ✓ Given a transition table that specifies the excitation function $Y=Y_1Y_2...Y_k$, then the general procedure for implementing a circuit with SR latches can be summarized as follows:

- Given a transition table

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

(a) Transition table

$$Y = x_1x'_2 + x_1y$$

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

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

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

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

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

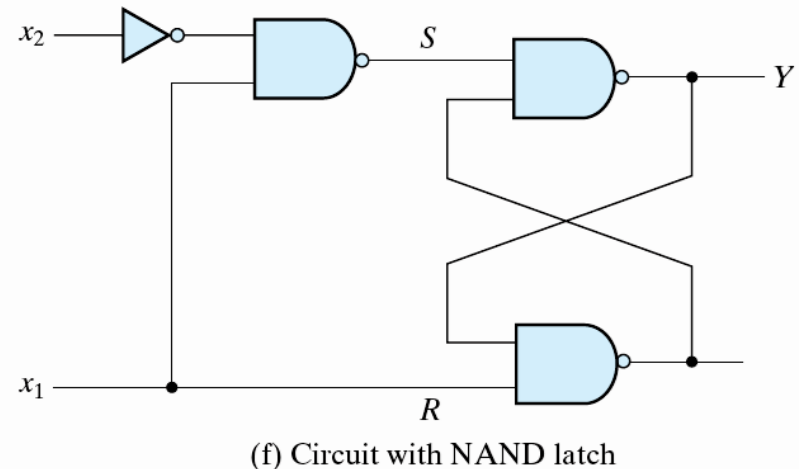
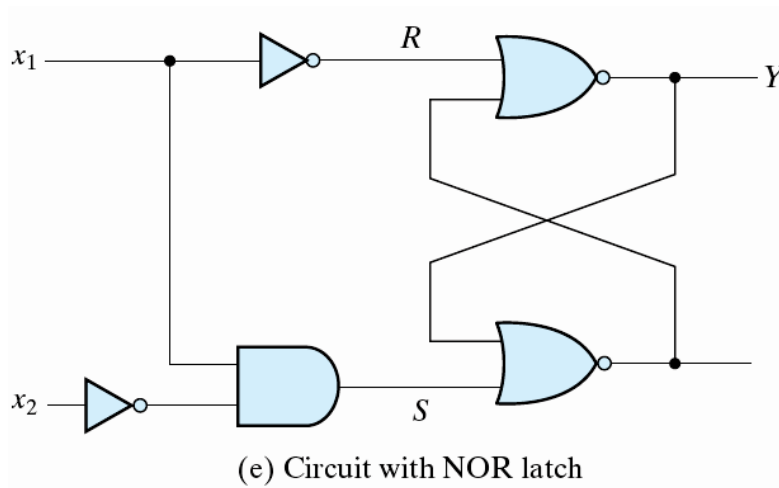
Implementation Example

- From maps: the simplified Boolean functions are

$$S = x_1 x'_2 \quad \text{and} \quad R = x'_1 \quad \Rightarrow \quad \text{NOR latch}$$

$$S = (x_1 x'_2)' \quad \text{and} \quad R = x_1 \quad \Rightarrow \quad \text{NAND latch}$$

- Check whether $SR=0$ for each NOR latch or whether $S'R'=0$ for each NAND latch:
 $SR = x_1 x'_2 x'_1 = 0$ $S'R' = (x_1 x'_2)' x_1 = 0$
- Draw the logic diagram, using k latches together with the gates required to generate the S and R Boolean functions obtained in step 1 (for NAND latches, use the complemented values)



Primitive Flow Table

- ✓ Primitive flow table - has exactly one stable total state (internal state + input) per row
 - Can be further reduced
- ✓ 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

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

Design Example 1

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.
- All outputs associated with unstable states are marked with a dash to indicate don't care conditions for the next state and output.
- 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.

		<i>DG</i>			
		00	01	11	10
<i>a</i>		<i>c</i> , -	<i>a</i> , 0	<i>b</i> , -	- , -
<i>b</i>		- , -	<i>a</i> , -	<i>b</i> , 1	<i>e</i> , -
<i>c</i>		<i>c</i> , 0	<i>a</i> , -	- , -	<i>d</i> , -
<i>d</i>		<i>c</i> , -	- , -	<i>b</i> , -	<i>d</i> , 0
<i>e</i>		<i>f</i> , -	- , -	<i>b</i> , -	<i>e</i> , 1
<i>f</i>		<i>f</i> , 1	<i>a</i> , -	- , -	<i>e</i> , -

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.
- 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
States	a	c, -	a , 0	b, -	-, -
	c	c , 0	a, -	-, -	d, -
	d	c, -	-, -	b, -	d , 0

DG		00	01	11	10
States	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
States	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
States	a	a , 0	a , 0	b, -	a , 0
	b	b , 1	a, -	b , 1	b , 1

(b) Reduced table (two alternatives)

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 for gated Latch

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

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

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

(b) $Q = Y$

Gated-latch logic diagram

