LECTURE 19

DIGITAL LOGIC FAMILIES

Debounce Circuit

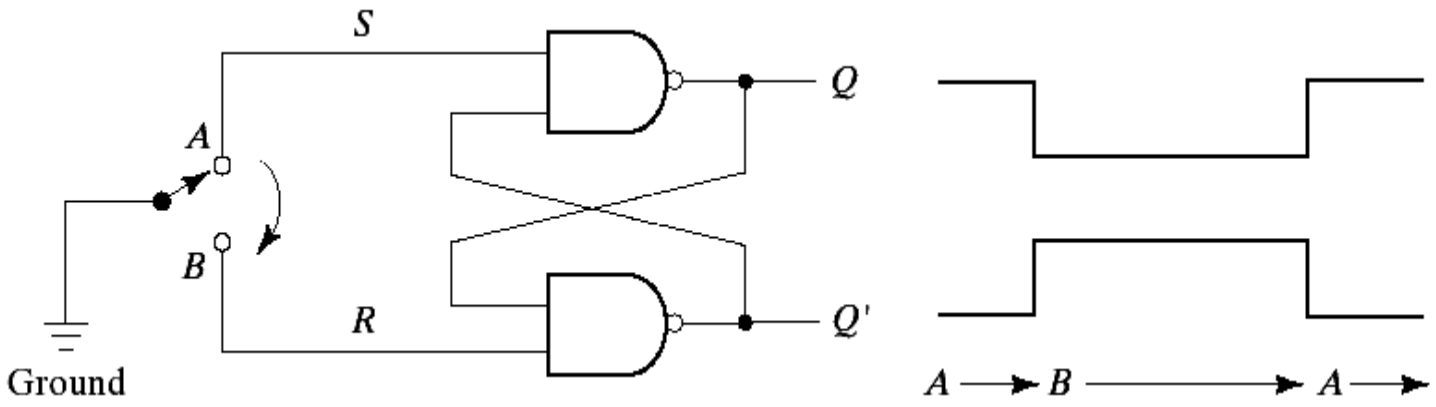
Mechanical switches are often used to generate binary signals to a digital circuit

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

-Cause the signal to oscillate between 1 and 0

A debounce circuit can remove the series of pulses from a contact bounce and produce a single smooth transition

-Position A(SR=01) → bouncing(SR=11) → Position B(SR=10)
Q = 1(set) → Q = 1(no change) → Q = 0 (reset)



Design procedure

(i) Obtain a primitive table from specifications

(ii) Reduce flow table by merging rows in the primitive flow table

(iii) Assign binary state variables to each row of reduced table

(iv) Assign output values to dashes associated with unstable states to obtain the output map

(v) Simplify Boolean functions for excitation and output variables;

(vi) Draw the logic diagram

Design Example:

Problem Statement:

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

Design Example:

1-Primitive Flow Table

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

	Input		Output	
State	D	G	Q	Comments
a	0	1	0	D=Q because G=1
b	1	1	1	D=Q because G=1
c	0	0	0	After states a or d
d	1	0	0	After state c
e	1	0	1	After states b or f
f	0	0	1	After state e

Design Example:

1-Primitive Flow Table

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

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

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

All outputs associated with unstable states are marked with a dash to indicate don't care conditions.

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

Design Example:

2-Reduction of the Primitive Flow Table

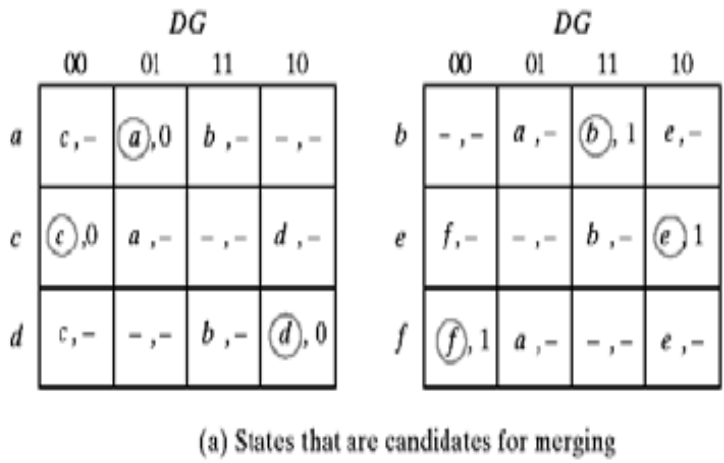
Two or more rows can be merged into one row if there are non-conflicting states and outputs in every columns.

After merged into one row:

Don’t care entries are overwritten

Stable states and output values are included

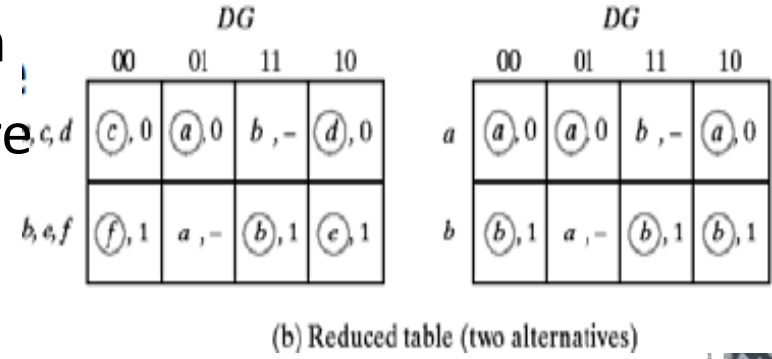
A common symbol is given to the merged row



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

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

(a) States that are candidates for merging



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

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

(b) Reduced table (two alternatives)

Design Example:

3-Transition Table and Logic Diagram

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

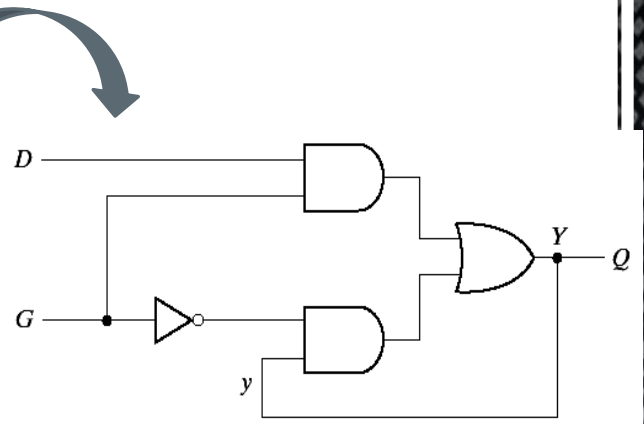
a=0, b=1 in this example

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

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

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

(b) $Q = Y$



Design Example:

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

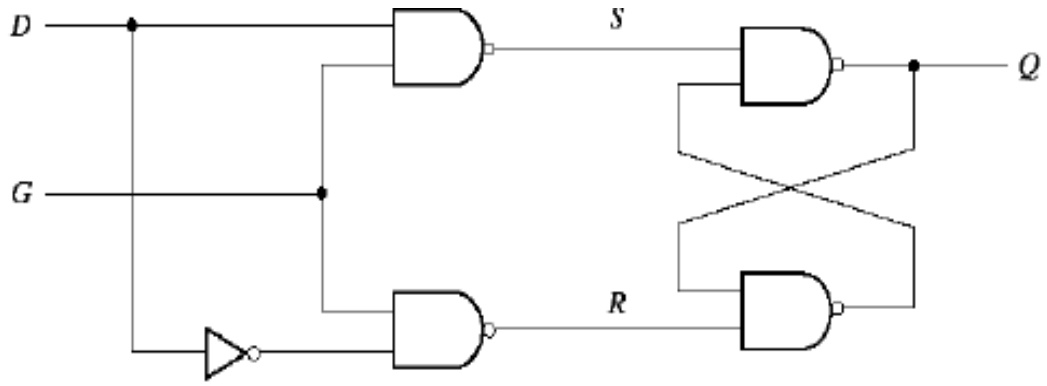
(a) $S = DG$

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

$R = D'G$

Implementation with SR Latch

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



4- Assigning Outputs to Unstable States

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

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

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

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

Design Example:

4- Assigning Outputs to Unstable States

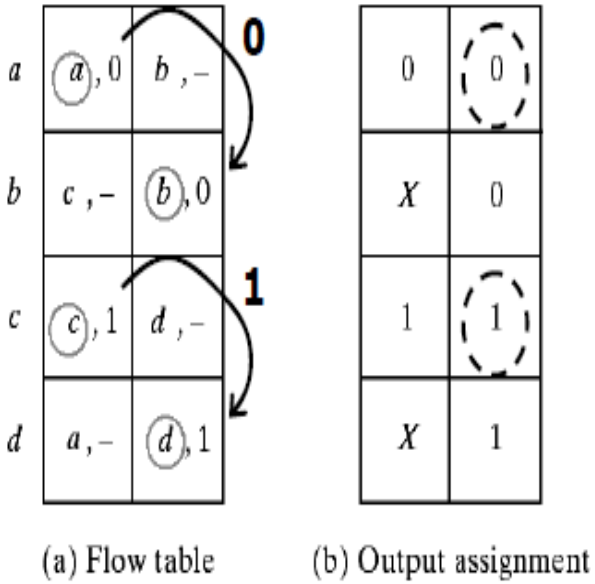
Ex:

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

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

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

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



Reduction of States and Flow Tables

Implication Table

Merging of the Flow Table

Compatible Pairs

Maximal Compatibles

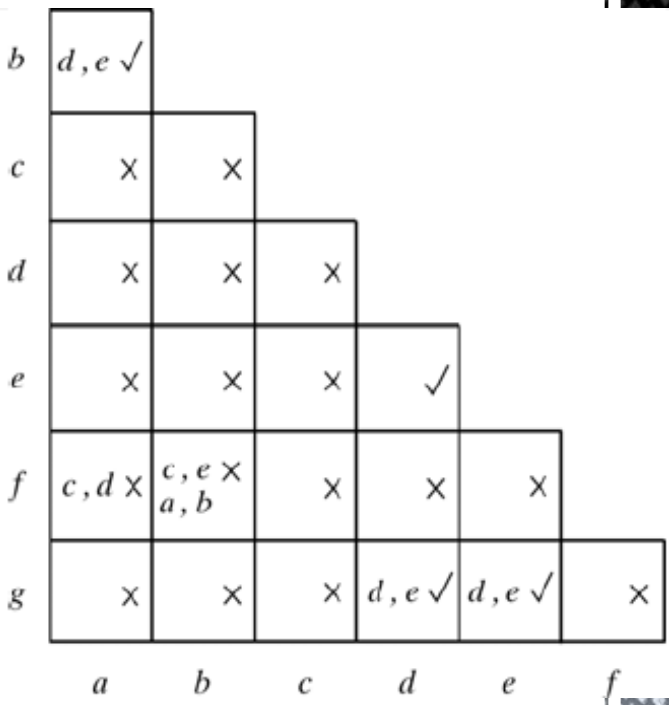
Closed Covering Condition

Implication Table (Example):

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

State Table to Be Reduced

Present State	Next State		Output	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
a	d	b	0	0
b	e	a	0	0
c	g	f	0	1
d	a	d	1	0
e	a	d	1	0
f	c	b	0	0
g	a	e	1	0

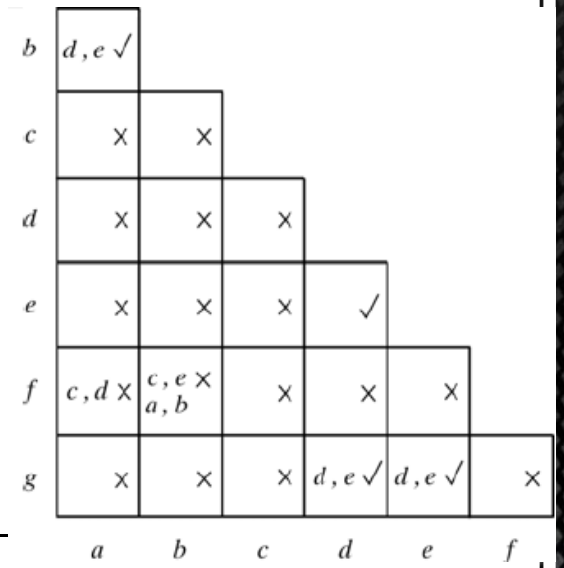


Implication Table (Example):

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

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

So the original flow table can be reduced to:



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

Merging of the Flow Table

The state table may be incompletely specified(Some next states and outputs are don't care).

Primitive flow tables are always incompletely specified

-Several synchronous circuits also have this property

Incompletely specified states are not "equivalent" Instead, we are going to find "compatible" states

→Two states are compatible if they have the same output and compatible next states whenever specified Three procedural steps:

-Determine all compatible pairs

- Find the maximal compatibles

-Find a minimal closed collection of compatible

Compatible Pairs

Implication tables are used to find compatible states.

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

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

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

(a) Primitive flow table

	a	*b*	*c*	*d*	*e*
b	✓				
c	✓	*d* , *e* ×			
d	✓	*d* , *e* ×	✓		
e	*c* , *f* ×	✓	*d* , *e* × *c* , *f* ×	×	
f	*c* , *f* ×	✓	×	*d* , *e* × *c* , *f* ×	✓

(b) Implication table

The compatible pairs are :

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

Maximal Compatibles

- A group of compatibles that contains all the possible combinations of compatible states.

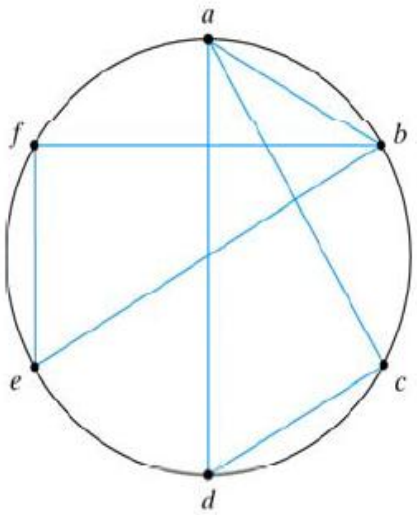
  -Obtained from a merger diagram.

  -A line in the diagram represents that two states are compatible.

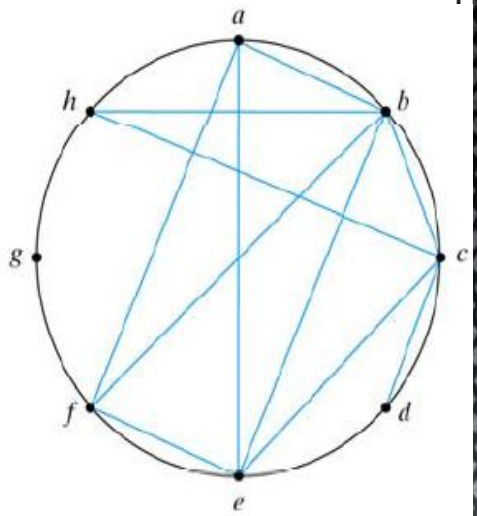
- n-state compatible → n-sided fully connected polygon.

  -All its diagonals connected.

- Not all maximal compatibles are necessary.



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



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

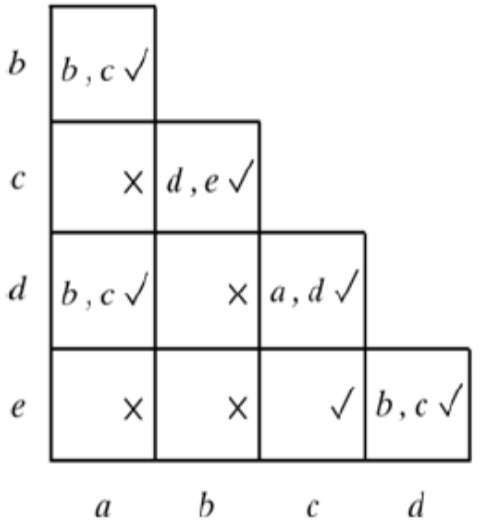
Closed Covering Condition (Example)

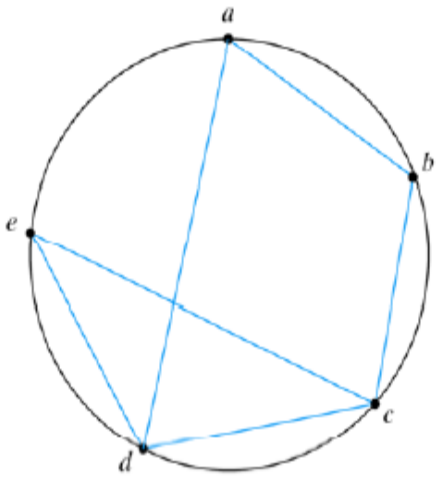
- From the given implication table, we have the following compatible: pairs: (a , b) (a , d) (b , c)(c , d)(c , e) (d , e)
- From the merger diagram, we determine the maximal compatibles: (a , b) (a , d) (b , c) (c , d , e)
-

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

(c) Closure table

d , e)





-All the 5 states are included in this set.

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

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

Race-Free State Assignment

- Objective: choose a proper binary state assignment to prevent critical races
- Only one variable can change at any given time when a state transition occurs
- States between which transitions occur will be given adjacent assignments

  -Two binary values are said to be adjacent if they differ in only one variable
- To ensure that a transition table has no critical races, every possible state transition should be checked

  -A tedious work when the flow table is large

  -Only 3-row and 4-row examples are demonstrated

3-Row Flow-Table Example

Three states require two binary variables

Outputs are omitted for simplicity

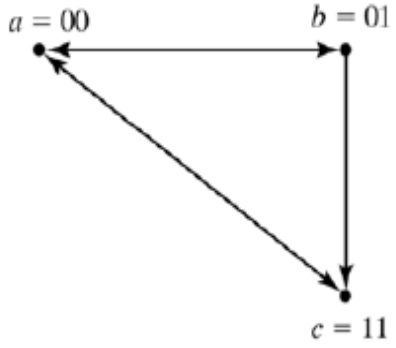
Adjac nted by a transition diagram

a and nt in such a

-Imp tates adjace re used

	$x_1 x_2$ = 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

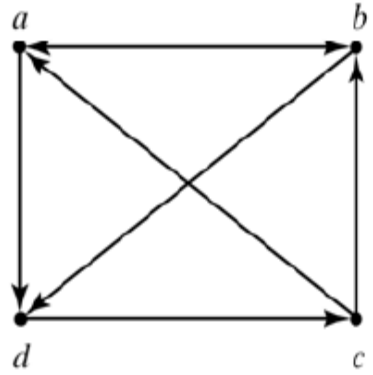
4-Row Flow-Table Example

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

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

	00	01	11	10
a	*b*	(*a*)	*d*	(*a*)
b	(*b*)	*d*	(*b*)	*a*
c	(*c*)	*a*	*b*	(*c*)
d	*c*	(*d*)	(*d*)	*c*

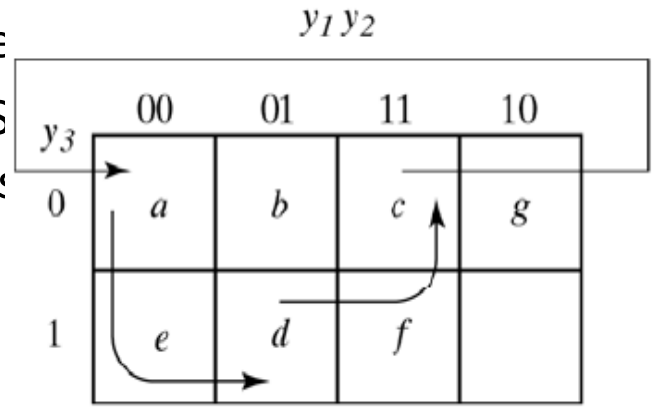
(a) Flow table



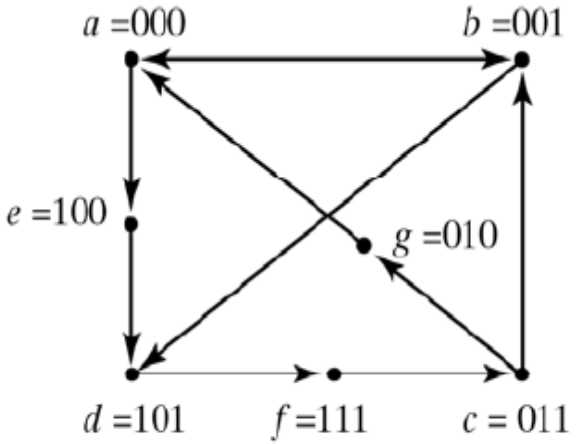
(b) Transition diagram

4-Row Flow-Table Example

- The following state assignment map is suitable for any 4-row flow table.
  - a
  - e
  - S
  ass



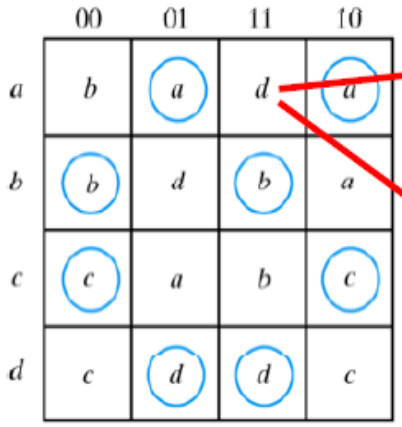
(a) Binary assignment



(b) Transition diagram

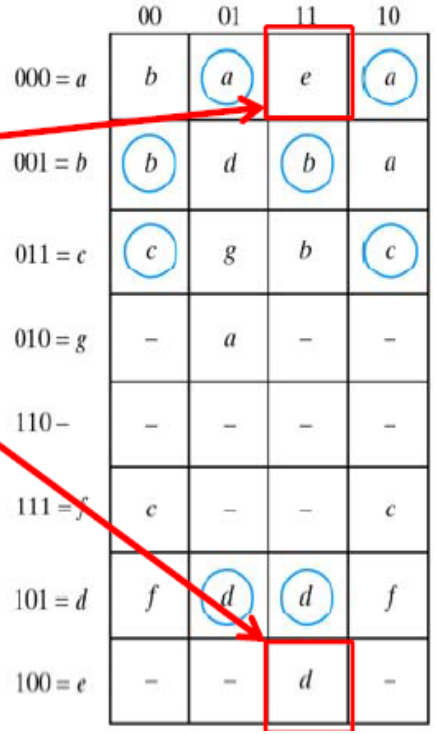
4-Row Flow-Table Example

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



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

(a) Flow table



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

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

Multiple Row Method

- Multiple-row method is easier

  May not as efficient as in above shared-row method
- Each stable state is duplicated with exactly the same output

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

y_1	y_2y_3 = 00	01	11	10
0	a_1	b_1	c_1	d_1
1	c_2	d_2	a_2	b_2

(a) Binary assignment

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

(b) Flow table

Hazards

Hazards: are unwanted switching transients that may appear at the output of a circuit because different paths exhibit different propagation delay.

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

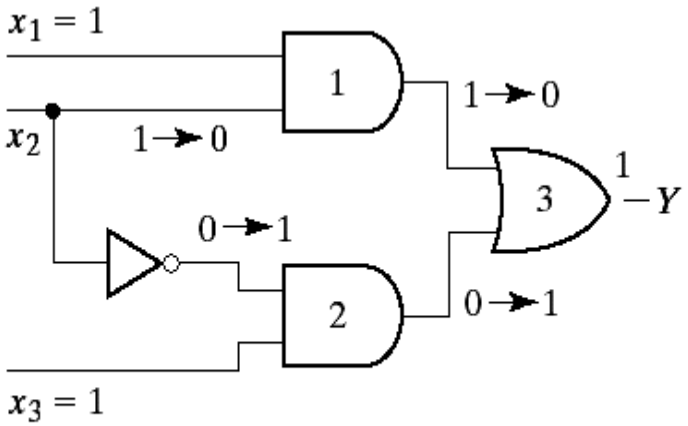
Hazards

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

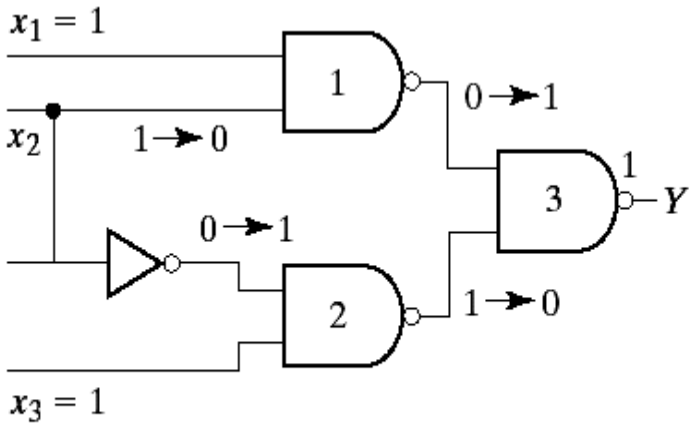
If implemented in sum of products:

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

Two examples for static 1-hazard:



(a) AND-OR circuit



(b) NAND circuit