Redundant States in Sequential Circuits

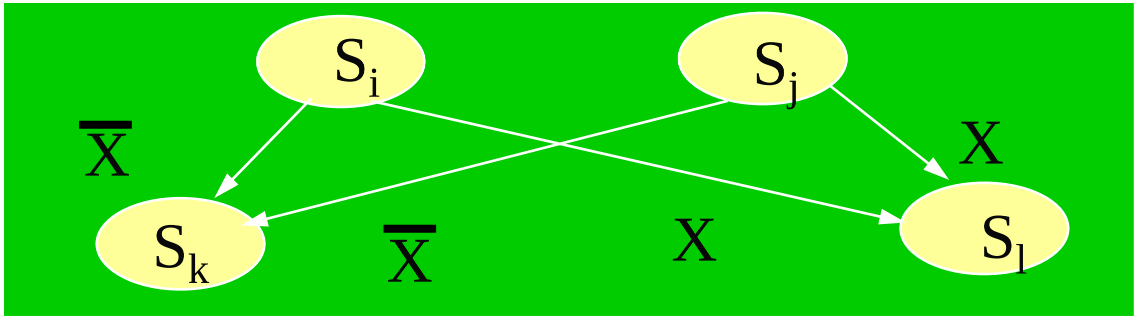
Removal of redundant states is important because

- ***Cost***: the number of memory elements is directly related to the number of states
- ***Complexity***: the more states the circuit contains, the more complex the design and implementation becomes
- ***Aids failure analysis***: diagnostic routines are often predicated on the assumption that no redundant states exist

Equivalent States

- Let S_i and S_j be states of a completely specified sequential circuit. Let S_k and S_l be the next states of S_i and S_j, respectively for input I_p.

  S_i and S_j are *equivalent* if and only if for every possible I_p the following conditions are satisfied:
  - the outputs produced by S_i and S_j are the same;
  - the next states S_k and S_l are equivalent.



  - States S_i and S_j are equivalent and are combined to one state by pointing all arrows that go to S_j to state S_i and removing S_j with its all arrows

Equivalence and Partitions

- ***Equivalence relation***: let *R* be a relation on a set *S*. *R* is an equivalence relation on *S* if and only if it is reflexive, symmetric, and transitive.
- An equivalence relation on a set partitions the set into disjoint equivalence classes.
- A partition consists of one or more blocks, where each block comprises a subset of states that may be equivalent, but the states in a given block are definitely not equivalent to the states in the other blocks.
- The partitioning method initially assumes that all states are equivalent and then proceeds to determine those state which are not equivalent by analyzing each states k-successors.

Finding Equivalent States By Inspection

	x = 0	x = 1
A	B/0	C/1
B	C/0	A/1
C	D/1	B/0
D	C/0	A/1

(a)

	x = 0	x = 1
A	B/0	C/1
B	C/0	A/1
C	B/1	B/0

(b)

	x = 0	x = 1
A	B/0	C/1
B	B/0	A/1
C	D/1	B/0
D	D/0	A/1

(c)

	x = 0	x = 1
A	B/0	C/1
B	B/0	A/1
C	B/1	B/0

(d)

	x = 0	x = 1
A	B/0	C/1
B	D/0	A/1
C	D/1	B/0
D	B/0	A/1

(e)

Partitioning Minimization Procedure

✓ PROCEDURE:

1) all states belong to the initial partition p_1
2) p_1 is partitioned in blocks such that the states in each block generate the same output.
3) continue to perform new partitions by testing whether the k-successors of the states in each block are contained in one block. Those states whose k-successors are in different blocks cannot be in one block.
4) prcedure ends when a new partition is the same as the previous partition

Finding Equivalent States by Partitioning

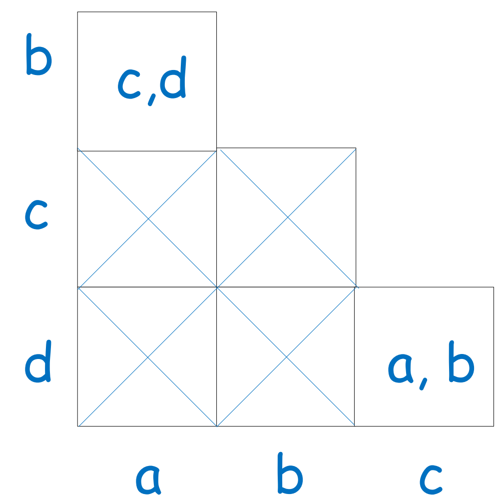
	Partition blocks				Action
Partition P_0	$(ABCDE)$				
Output for $x = 0$	11100				Separate (ABC) and (DE)
Output for $x = 1$	00011				Separate (ABC) and (DE)
Partition P_1	(ABC)		(DE)		
Next state for $x = 0$	CCB		DE		
Next state for $x = 1$	BEE		BA		Separate (A) and (BC)
Partition P_2	(A)	(BC)	(DE)		
Next state for $x = 0$	C	CB	DE		
Next state for $x = 1$	B	EE	BA		Separate (D) and (E)
Partition P_3	(A)	(BC)	(D)	(E)	
Next state for $x = 0$	C	CB	D	E	
Next state for $x = 1$	B	EE	B	A	
Partition $P_4 = P_3$	(A)	(BC)	(D)	(E)	

States B and C are equivalent.

Implication Table

- ✓ 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.

Present State	Next State		Output	
	X=0	X=1	X=0	X=1
a	*c*	*b*	0	1
b	*d*	*a*	0	1
c	*a*	*d*	1	0
d	*b*	*d*	1	0



(c,d) ⇔(a,b) ⟹ (c,d) && (a,b)

both pairs are equivalent

Implication Table

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

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

- the equivalent states
  - (a,b), (d,e), (d,g), (e,g)
- the reduced states
  - (a,b), (c), (d,e,g), (f)
- the 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

Incompletely specified circuits

✓ Finding compatible states by inspection

	$x = 0$	$x = 1$
A	$B/$0	$E/0$
B	$B/1$	$E/$1
C	$F/0$	$C/0$
D	$B/1$	$A/1$
E	$D/0$	$C/$0
F	$D/$1	$C/1$

	$x = 0$	$x = 1$
A'	$A'/1$	$D'/0$
B'	$D'/0$	$B'/0$
C'	$A'/1$	$A'/1$
D'	$C'/0$	$B'/1$

	$x = 0$	$x = 1$
A'	$B'/0$	$A'/0$
B'	$B'/1$	$A'/1$

Merging of the Flow Table

- The state table may be incompletely specified: combinations of inputs or input sequences may never occur (Some next states and outputs are don't care).
- Multi-input **primitive flow tables** are always incompletely specified
  - Several synchronous circuits also have this property
- Incompletely specified states are not "equivalent" as in completely specified circuits. Instead, we are going to find **compatible** states

Equivalent and Compatible States

- ✓ completely specified state table ⇨ equivalence

  If a and b are equivalent, and b and c are equivalent then also a and c are equivalent:

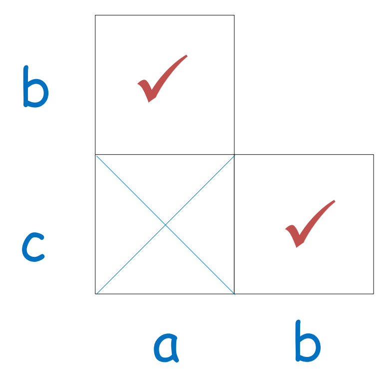
  (a,b), (b, c) ⇨(a, c)

- ✓ uncompletely specified state table ⇨ compatibility

  If a and b are compatible, and b and c are compatible not necessarily a and c are compatible:

y \	0	1
a	A,0	C,1
b	B,-	C,-
c	C,1	C,1

- ✓ **Compatibility relation**: let *R* be a relation on a set *S*. *R* is a compatibility relation on *S* if and only if it is reflexive and symmetric. *A* compatibility relation on a set partitions the set into compatibility classes. They are typically not disjoint.



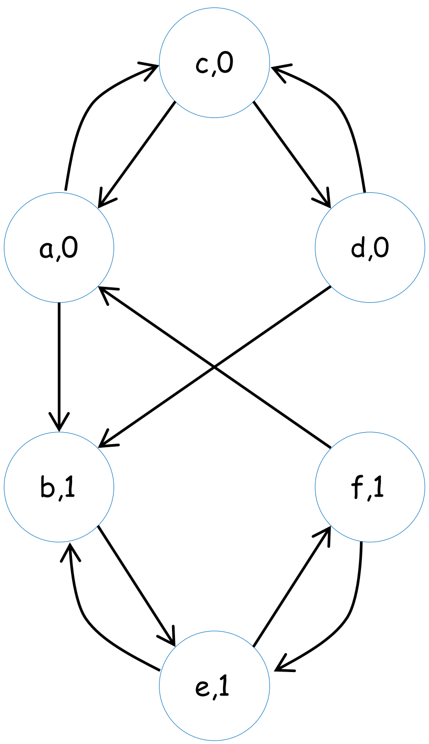
Incompletely specified circuits: partition method

- ✓ The partitioning minimization procedure which was applied to completely specified state tables can also be applied to incompletely specified state tables.
- ✓ To perform the partitioning process, we can assume that the unspecified outputs have a specific value.
- ✓ The partitioning method is equally applicable to Mealy type FSMs in the same way as for Moore-type FSMs.

Compatible Pairs (DG sequential circuit)

✓ 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 ×	✓

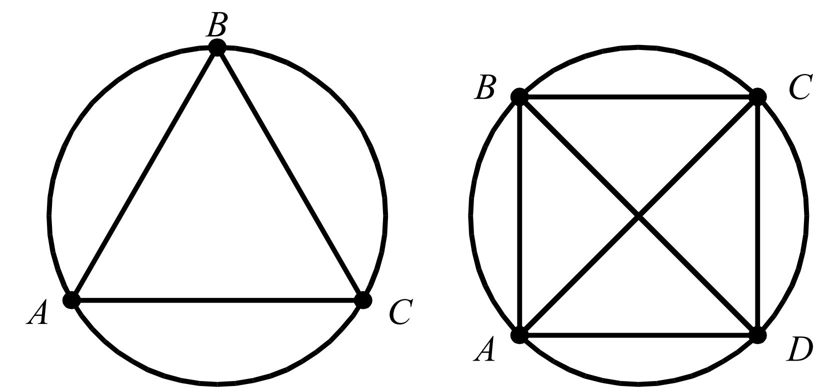
(b) Implication table

The compatible pairs are :

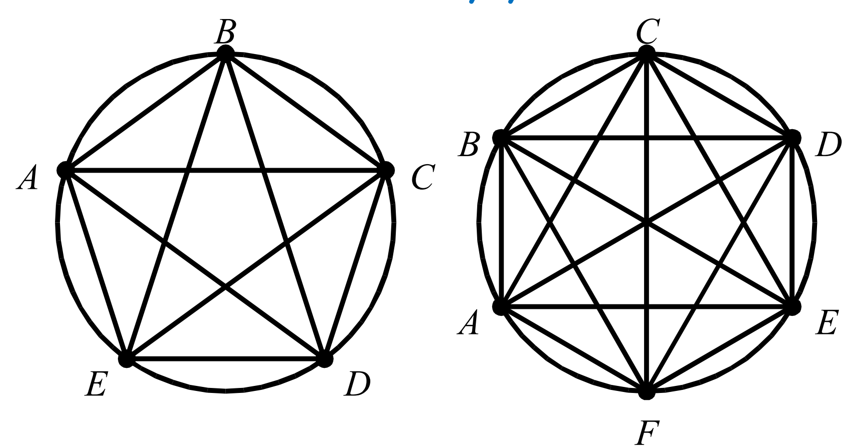
(a , b)
(a , c)
(a , d)
(b , e)
(b , f)
(c ,d)
(e , f)

Merger diagrams

- ✓ States are represented as dot in a circle
- ✓ Lines connect states couples compatible
- ✓ Maximal sets can be identified as those sets in which every states is connected to every other state by a line segment



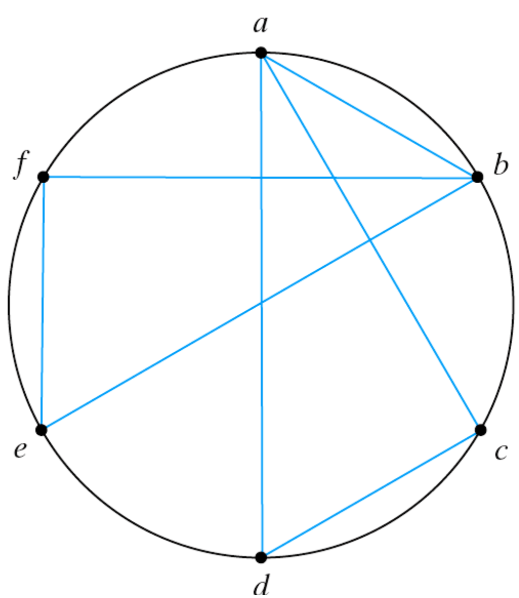
Maximal sets with 3,4,5 and 6 states

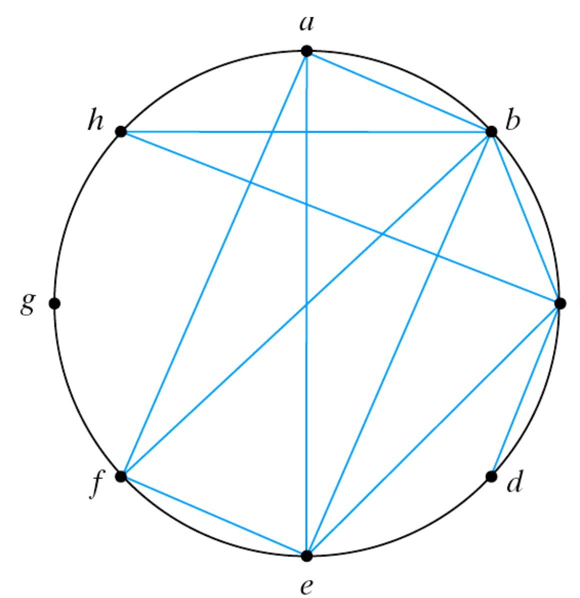


Maximal Compatibles

- ✓ Maximal Compatibles: a group of compatibles that contains all the possible combinations of compatible states.
- ✓ n-state compatible ⇨ n-sided fully connected polygon.
  - All its diagonals connected.
  - Not all maximal compatibles are necessary
  - 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





Closed Covering Condition

- ✓ The condition that must be satisfied is that the set of chosen compatibles must:
  - Cover all states.
  - Be closed: the closure condition is satisfied if there are no implied states or if the implied states are included within a set
- ✓ In the example of the DG sequential circuit, the maximal compatibles are:

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

  (a, c, d), (b, e , f):
  - All the six states are included in this set.
  - There are no implied states for (a,c); (a,d);(c,d);(b,e);(b,f) and (e,f) [you can check the implication table] . The closer condition is satisfied

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

Closed Covering Condition (Example)

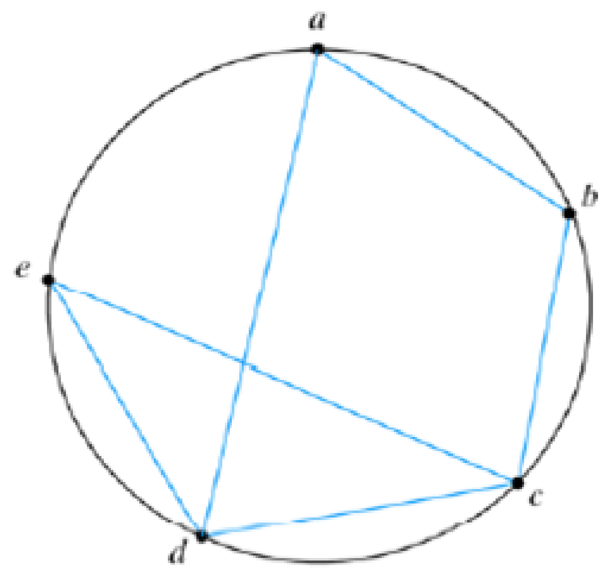
- From the aside implication table, we have the following compatible: pairs: (a, b) (a , d) (b, c) (c , d) (c , e) (d , e)
- From the merger diagram, we determine the maximal compatibles: (a , b) (a , d) (b , c) (c , d , e)
- If we choose the two compatibles: (a , b) (c, d, e)

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

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

Closure table

- All the 5 states are included in this set.
- The implied states for (a, b) are (b, c). But (b, c) are not include in the chosen set. This set is not closed.
- *A set of compatibles that will satisfy the closed covering condition is (a, **d**) (b, **c**) (**c**, **d**, e)*
- Note that: the same state can be repeated more than once



Minimization of campatible classes

Select a set of compatibility classes so that the following conditions are satisfied:

- ✓ Completeness: all states of the original machine must be covered
- ✓ Consistency: the chosen set of compatibility classes must be closed
- ✓ Minimality: the smallest number of compatibility classes is used

Bounding the number of states

- ✓ Unfortunately the process of selecting the set of compatibility classes that meets the three conditions must be done by **trial and error**.
- ✓ Let ***U*** be the **upper bound** on the number of states needed in the minimized circuit.

  Then U = minimum (NSMC, NSOC)

  - where NSMC = the number of sets of maximal compatibles
  - and NSOC = the number of states in the original circuit
- ✓ Let ***L*** be the **lower bound** on the number of states needed in the minimized circuit

  Then L = maximum($NSMI_1$, $NSMI_2$,…, $NSMI_i$)

  - where $NSMI_i$ = the number of states in the i^{th} group of the set of maximal **incompatibles** of the original circuit.

State Reduction Algorithm

- ✓ Step 1 -- find the maximal compatibles
- ✓ Step 2 -- find the maximal incompatibles
- ✓ Step 3 -- Find the upper and lower bounds on the number of states needed
- ✓ Step 4 -- Find a set of compatibility classes that is complete, consistent, and minimal
- ✓ Step 5 -- Produce the minimum state table

Example -- State reduction problem

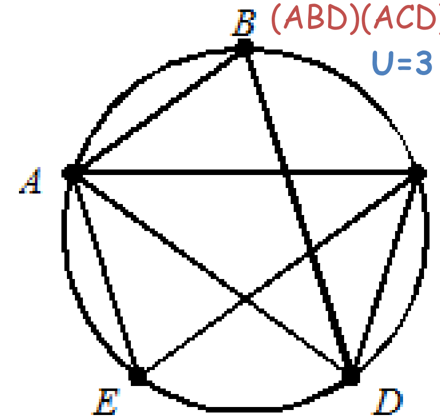
	x = 0	1
A	A/-	-/-
B	C/1	B/0
C	D/0	-/1
D	-/-	B/-
E	A/0	C/1

Compatibles pairs
$(AB)(AC)(AD)(AE)(BD)(CD)(CE)$

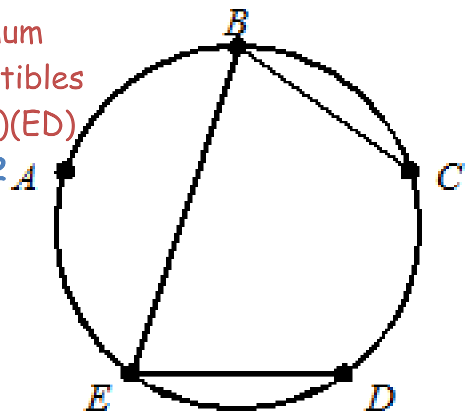
Incompatibles pairs
$(BC)(BE)(DE)$

	A	B	C	D
B	AC			
C	AD	X		
D	√	√	√	
E	√	X	AD	BC

Maximum Compatibles
(ABD)(ACD)(ACE)
U=3



Maximum Incompatibles
(BE)(BC)(ED)
L=2



	x = 0	1
(ABD)	AC	B
~~(ACD)~~	~~AD~~	~~B~~
(ACE)	AD	C

Closure table

	x = 0	1
A'	B'/1	A'/0
B'	A'/0	B'/1

$A' = (ABD)$
$B' = (ACE)$

Reduced state table

Another state table reduction problem

	x = 0	1
A	$B/1$	$D/0$
B	–/–	$B/0$
C	$E/0$	$D/–$
D	$B/1$	$A/0$
E	–/–	$C/1$
F	–/0	$E/1$

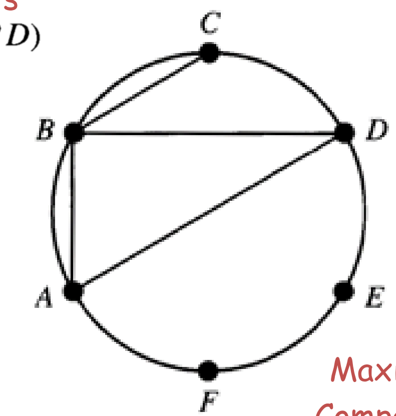
(a)

B	BD				
C	×	BD			
D	√	AB	×		
E	×	×	CD	×	
F	×	×	DE	×	CE
	A	B	C	D	E

(b)

Compatibles pairs

$(AB)(AD)(BC)(BD)$



(c)

Maximum Compatibles

(ABD)(C)(E)(F)

U=4

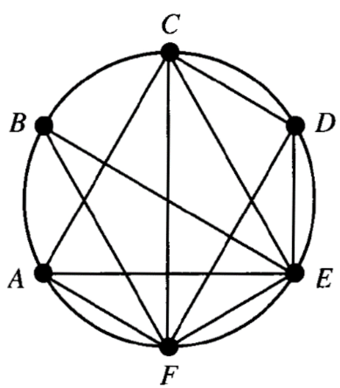
Closure table

	x = 0	1
(ABD)	B	ABD
(C)	E	D
(E)	–	C
(F)	–	E

(d)

Incompatibles pairs

$(AC)(AE)(AF)(BE)(BF)(CD)(CE)(CF)(DE)(DF)(EF)$



(e)

Maximum Incompatibles

(ACEF)(CDEF)

(BEF)

L=4

	0	1
A'	$A'/1$	$A'/0$
B'	$C'/0$	$A'/0$
C'	–/–	$B'/1$
D'	–/0	$C'/1$

(f)

Reduced state table

Note that the resultant state table has considerable flexibility.

This property will serve to simplify the hardware realization

	x = 0	1
(ABD)	B	ABD
(BC)	E	BD
(E)	–	C
(F)	–	E

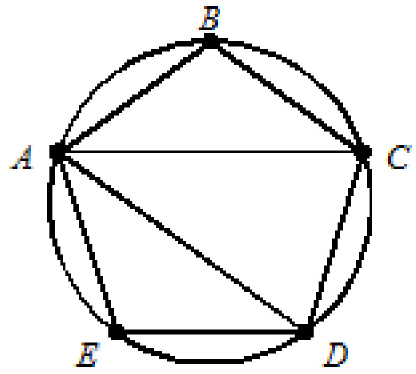
	x = 0	1
A'	$A', B'/1$	$A'/0$
B'	$C'/0$	$A'/0$
C'	–/–	$B'/1$
D'	–/0	$C'/1$

Yet another state reduction problem

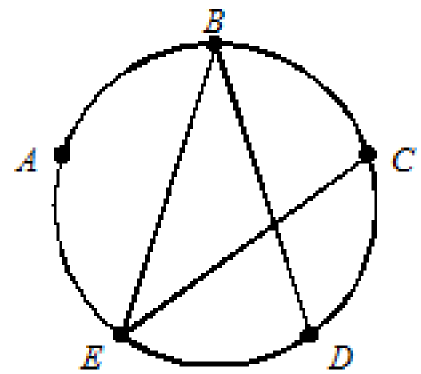
	x = 0	x = 1
A	D/-	A/-
B	E/0	A/-
C	D/0	B/-
D	C/-	C/-
E	C/1	B/-

	A	B	C	D
B	DE			
C	AB	AB DE		
D	AC CD	AC CE	BC	
E	AB CD	X	X	BC

Maximum Compatibles



Maximum Incompatibles



Maximum Compatibles (ABC)(ACD)(DE)
U=3

Maximum Incompatibles (BD)(BE)(CE)
L=2

	x = 0	x = 1
(ABC)	DE	AB
~~(ACD)~~	~~CD~~	~~ABC~~
(ADE)	CD	ABC

Closure table Maximum Compatible

	x = 0	x = 1
(ABC)	DE	AB
(DE)	C	BC

Closure table

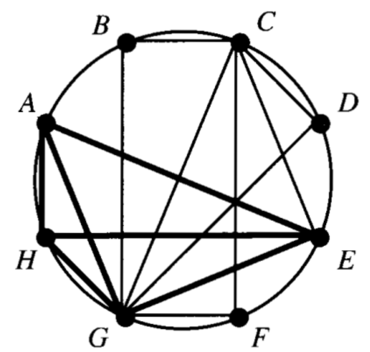
	x = 0	x = 1
A'	B'/0	A'/-
B'	A'/1	A'/-

Reduced state table

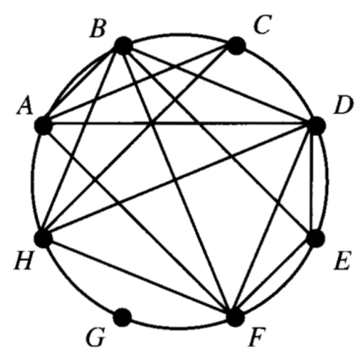
Generating Maximal Compatibles and Incompatibles

	x = 0	1
A	A/–	C/1
B	B/–	A/–
C	G/–	E/0
D	C/1	C/–
E	A/1	C/–
F	D/–	A/–
G	G/–	G/–
H	H/–	D/–

	A	B	C	D	E	F	G
B	AC						
C		BG AE					
D	AC	BC AC	CG CE				
E	√	AB AC	AG	AC			
F	AD AC	BD	DG AE	CD AC	AD AC		
G	CG	AG	EG	CG	AG CG	DG AG	
H	CD	AD	GH DE	CH CD	AH CD	DH AD	DG



Maximum Compatibles
(AEGH)(BCG)(CDG)
(CEG)(CFG)
U=5



Maximum
Incompatibles
(ABDF)(AC)(BDEF)
(CH)(BDFH)
L=4

Reduced state table

	$x=0$	$x=1$
A	$A/-$	$C/1$
B	$B/-$	$A/-$
C	$G/-$	$E/0$
D	$C/1$	$C/-$
E	$A/1$	$C/-$
F	$D/-$	$A/-$
G	$G/-$	$G/-$
H	$H/-$	$D/-$

- ✓ Closure table: treat maximum compatibles as states and find their sets of next states

	$x=0$	$x=1$
$(AEGH)$	AGH	CDG
(BCG)	BG	AEG
(CDG)	CG	CEG
(CEG)	AG	CEG
(CFG)	DG	AEG

Closure table

	$x=0$	$x=1$
A'	$A'/1$	$C'/1$
B'	$B'/-$	$A'/0$
C'	$B', C', D', E'/1$	$D'/0$
D'	$A'/1$	$D'/0$
E'	$C'/-$	$A'/0$

Reduced state table

- ✓ All maximal compatibles are used as states of the reduced machine. Hence, the final five states are:

$$A' = (AEGH), \quad D' = (CEG)$$
$$B' = (BCG), \quad E' = (CFG)$$
$$C' = (CDG)$$

State Assignment

- ✓ Primary Objective of Synchronous Networks
  - Simplification of Logic and Improvement of Performance
  - Improvement of Testability
  - Minimization of Power Consumption.
- ✓ Primary Objective of Asynchronous Networks
  - Prevention of Critical Races
  - Simplification of Logic

Race-Free State Assignment

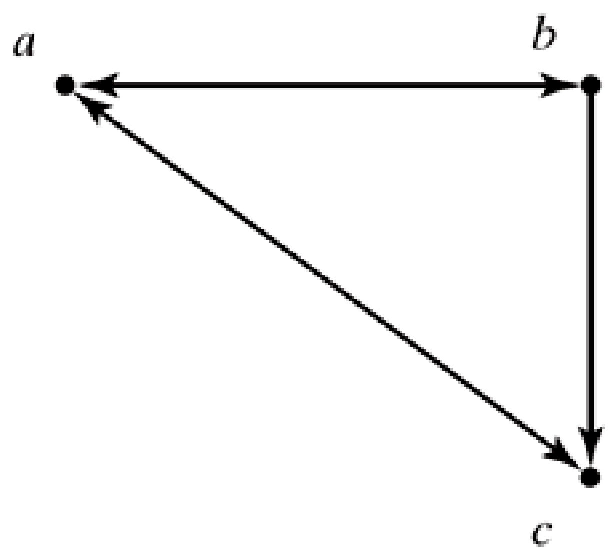
- ✓ Objective: choose a proper binary state assignment to prevent critical races
- ✓ Only one variable can change at any given time when a state transition occurs
- ✓ States between which transitions occur will be given adjacent assignments
  - Two binary values are said to be adjacent if they differ 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 (in the flow table outputs are omitted for simplicity)
- ✓ Representation by a transition diagram
- ✓ a and c are not adjacent in such an assignment!
  - Impossible to make all states adjacent if only 3 states are 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



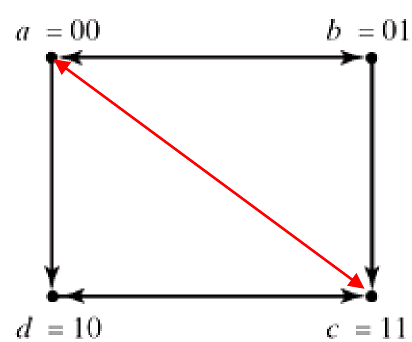
(b) Transition diagram

3-Row Flow-Table Example

- ✓ A race-free assignment can be obtained if we add an extra row to the flow table
- ✓ Only provide a race-free transition between the stable states
- ✓ The transition from a to c must now go through d

  00 ⇨ 10 ⇨ 11 (no race condition)
- ✓ Note that no stable state can be introduced in row d

	x_1x_2 = 00	01	11	10
a	a	b	d	a
b	a	b	b	c
c	d	c	c	c
d	a	–	c	–



	x_1x_2 = 00	01	11	10
a = 00	00	01	10	00
b = 01	00	01	01	11
c = 11	10	11	11	11
d = 10	00	–	11	–

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

Shared-Row Method

- ✓ The shared row is not assigned to any specific stable state
- ✓ Used to convert a critical race into a cycle that goes through adjacent transitions between two stable states
- ✓ May require more extra rows

State adjacencies for assignments

UNIQUE STATE ASSIGNMENTS

	Assignments		
	1	2	3
States	y_1y_2	y_1y_2	y_1y_2
A	00	00	00
B	01	11	10
C	11	01	01
D	10	10	11

$y_1 \backslash y_2$	0	1
0	A	B
1	D	C

$y_1 \backslash y_2$	0	1
0	A	C
1	D	B

$y_1 \backslash y_2$	0	1
0	A	C
1	B	D

Unique State Assignments

Present state	x = 0	x = 1
A	$A/0$	$B/0$
B	$A/0$	$C/0$
C	$C/0$	$D/0$
D	$C/1$	$A/0$

(a)

States	Assignments 1 $y_1\,y_2$	2 $y_1\,y_2$	3 $y_1\,y_2$
A	00	10	00
B	01	11	10
C	11	01	11
D	10	00	01

(b)

$$D_1 = y_1\bar{x} + y_2 x \qquad D_2 = y_1\bar{x} + \bar{y}_1 x$$

$$D_1 = y_1\bar{x} + \bar{y}_2 x \qquad D_2 = \bar{y}_1\bar{x} + y_1 x$$

$$D_1 = y_2\bar{x} + \bar{y}_2 x \qquad D_2 = y_2\bar{x} + y_1 x$$

NUMBER OF STATE ASSIGNMENTS.

N_S	N_{FF}	N_{SA}	N_{UA}
1	0	—	—
2	1	2	1
3	2	24	3
4	2	24	3
5	3	6,720	140
6	3	20,160	420
7	3	40,320	840
8	3	40,320	840
9	4	4.15×10^9	10,810,800
10	4	2.91×10^{10}	75,675,600

$$N_{UA} = \frac{(2^{N_{FF}} - 1)!}{(2^{N_{FF}} - N_S)!\,N_{FF}!}$$

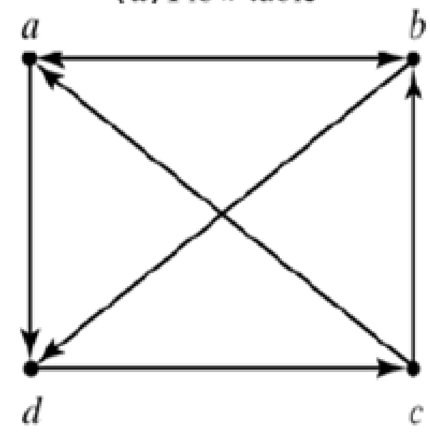
4-Row Flow-Table Example

- ✓ 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 general 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



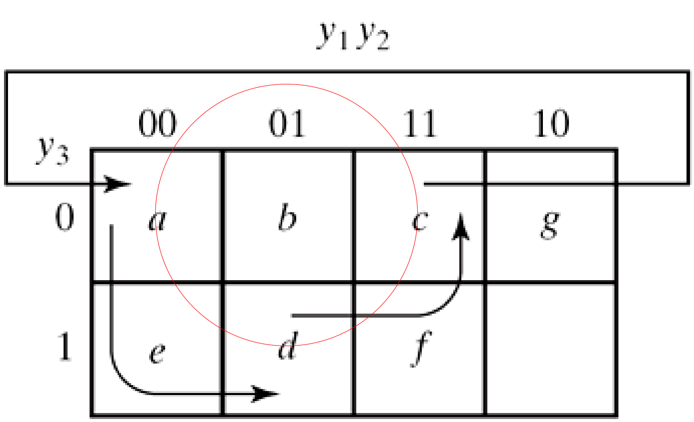
(b) Transition diagram

4-Row Flow-Table Example

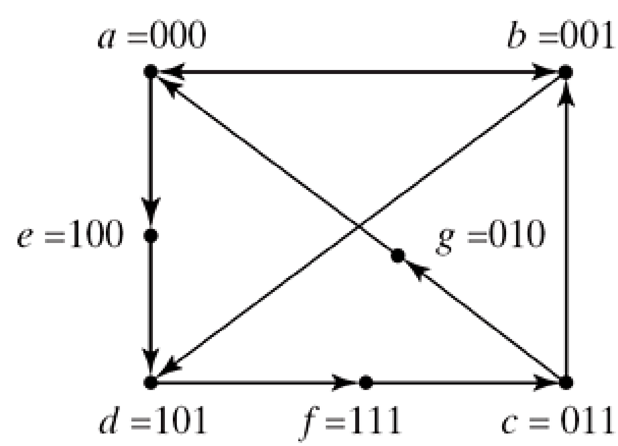
- The following state assignment map is suitable for any table.
  - a, b, c, and d are the original states.
  - e, f, and g are extra states.
  - States placed in adjacent squares in the map will have adjacent assignments
  - **Please note that state variable order in figures is $y_3y_1y_2$**

	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



(a) Binary assignment



(b) Transition diagram

4-Row Flow-Table Example

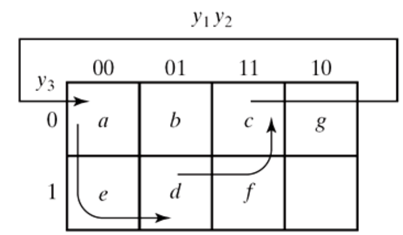
✓ To produce cycles:

- 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 among the two possibilities

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

(a) Binary assignment

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

(b) Flow table

Don't Care Assignment

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

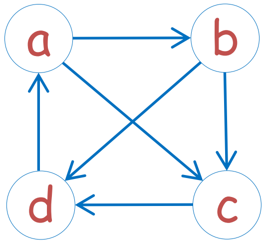
column 00: $d \rightarrow a$

column 01: $a \rightarrow b, \quad c \rightarrow d$

column 11: $b \rightarrow c$

column 10: $a \rightarrow c, \quad b \rightarrow d$

Needed Transitions



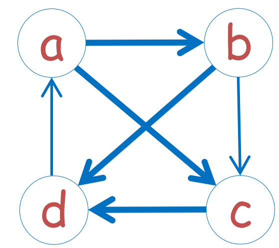
- ✓ All possible transitions between pairs of rows are needed
- ✓ Fill in don't care to eliminate races
- ✓ Direct transitions for columns 01, 10 (No don't care)

Don't Care Assignment

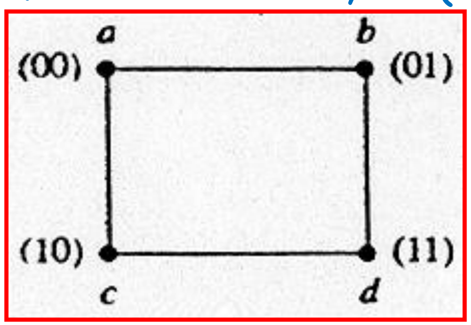
	00	01	11	10
a	a	b	–	c
b	b	b	c	d
c	–	d	c	c
d	a	d	d	d

column 00: $d \rightarrow a$
column 01: $a \rightarrow b, \quad c \rightarrow d$
column 11: $b \rightarrow c$
column 10: $a \rightarrow c, \quad b \rightarrow d$

Needed Transitions



- ✓ All possible transitions between pairs of rows are needed
- ✓ Fill in don't care to eliminate races
- ✓ Direct transitions for columns 01, 10 (No don't care)

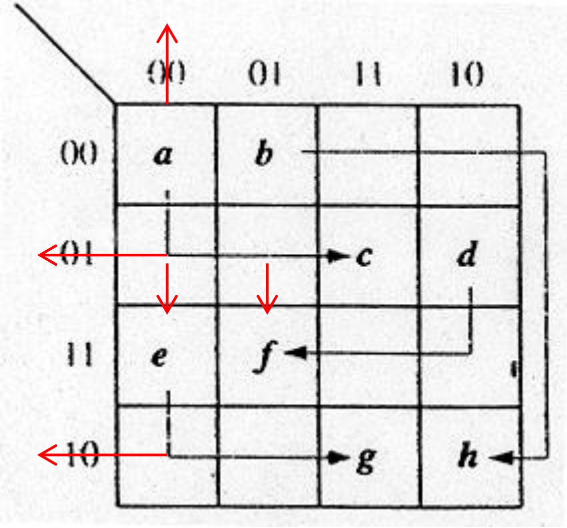


		00	01	11	10
00	a	a	b	c	c
01	b	b	b	a	d
10	c	a	d	c	c
11	d	c	d	d	d

In column 00: d⇨a changed to d⇨c⇨a.
In column 11: b⇨c changed to b⇨a⇨c.

State Assignment

- ✓ Universal Assignment for 8-Row Tables
  - Require 4 state variables.
  - Slow due to several successive changes needed.

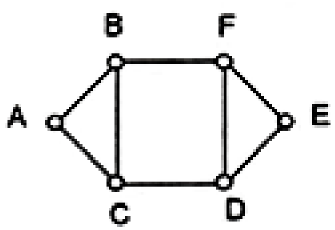


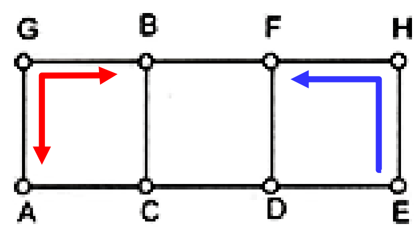
- ✓ Universal Assignment as the Last Resort.
  - Try to use smaller number of state variables.
  - Try to take advantage of don't care.
  - Allow several state variables change simultaneously.
    - Make all races noncritical.
    - Faster.

Extra raw flow table

I	I_1	I_2	I_3	I_4
a	C,1	B,0	A,1	C,-
b	C,1	B,0	A,1	A,-
c	C,1	A,-	C,0	D,0
d	E,0	D,1	C,0	D,0
e	E,0	D,1	F,-	E,1
f	D,-	D,1	B,-	F,0

	I_1	I_2	I_3	I_4
a	C,1	G,0	A,1	C,-
b	C,1	B,0	G,1	A,-
c	C,1	A,-	C,0	D,0
d	E,0	D,1	C,0	D,0
e	E,0	D,1	H,-	E,1
f	D,-	D,1	B,-	F,0
g	-,-	B,-	A,-	-,-
h	-,-	-,-	F,-	-,-



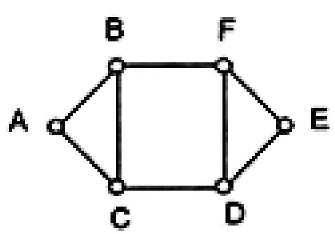


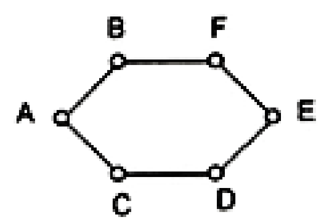
y_3 \ y_1y_2	00	01	11	10
0	G	B	F	H
1	A	C	D	E

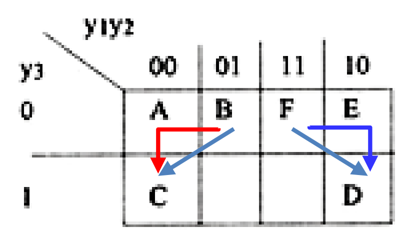
Alternative solution: transition changes

I	I_1	I_2	I_3	I_4
a	C,1	B,0	A,1	C,-
b	C,1	B,0	A,1	A,-
c	C,1	A,-	C,0	D,0
d	E,0	D,1	C,0	D,0
e	E,0	D,1	F,-	E,1
f	D,-	D,1	B,-	F,0

I	I_1	I_2	I_3	I_4
a	C, 1	B, 0	A, 1	C, -
b	A•, 1	B, 0	A, 1	A, -
c	C, 1	A, -	C, 0	D, 0
d	E, 0	D, 1	C, 0	D, 0
e	E, 0	D, 1	F, -	E, 1
f	E•, -	D, 1	B, -	F, 0







Summary

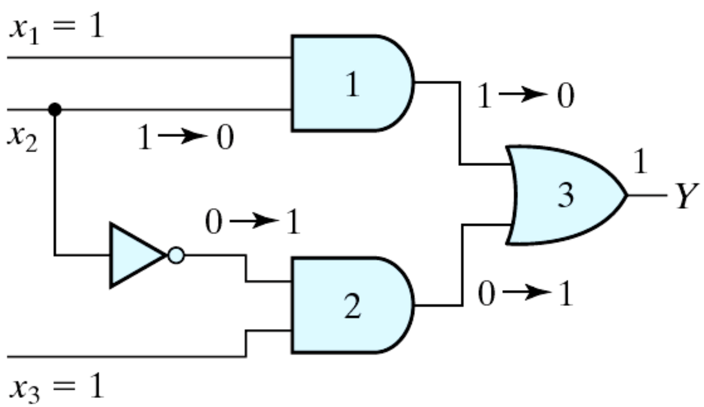
- ✓ Shared row method
- ✓ Multiple row method
- ✓ Don't care assignment
- ✓ Universal states assignment
- ✓ Transition changes
- ✓ Other approaches

[1] VP Nelson et al., not presented here

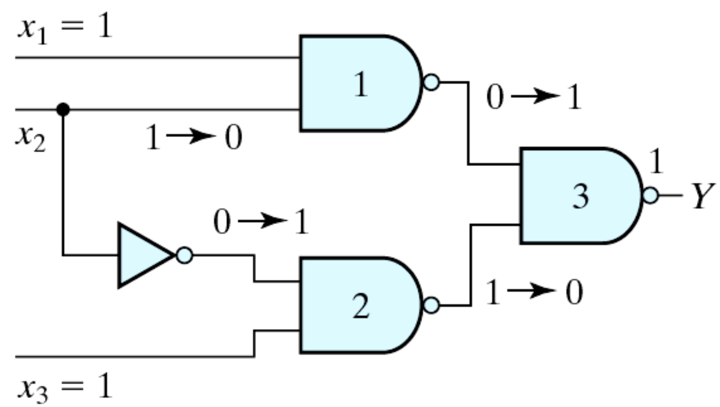
Hazards

- A timing problem arises due to gate and wiring delays
- Hazards: Unwanted switching transients at the network output, caused by input changes and due to different paths through the network from input to output that may have different propagation delays
- Hazards occur 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.
- Asynchronous Sequential Circuits:
  - Hyphothesis:
    - Operated in fundamental mode with only one input changing at any time
  - Objectives:
    - Free of critical races
    - Free of hazards

Hazards in combinational circuits



(a) AND–OR circuit

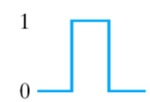


(b) NAND circuit

✓ Static 1-hazard (sum of products)

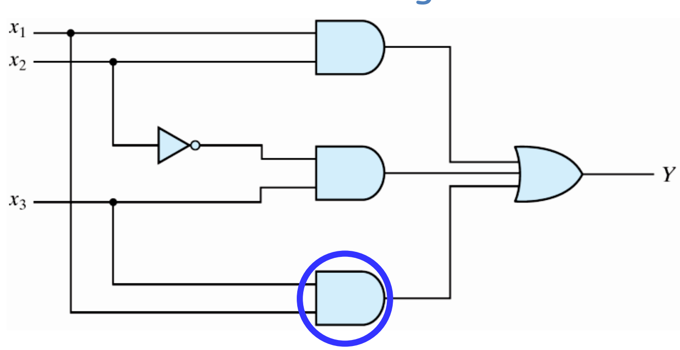
- The remedy
  - the circuit moves from one product term to another
  - additional redundant gate

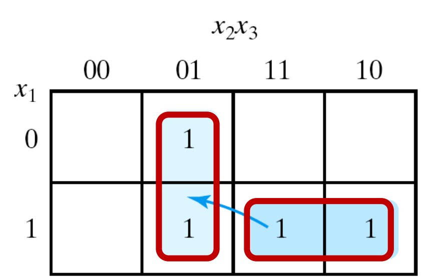
(a) Static 1-hazard



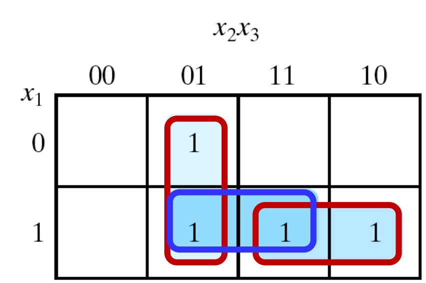
(b) Static 0-hazard

(c) Dynamic hazard





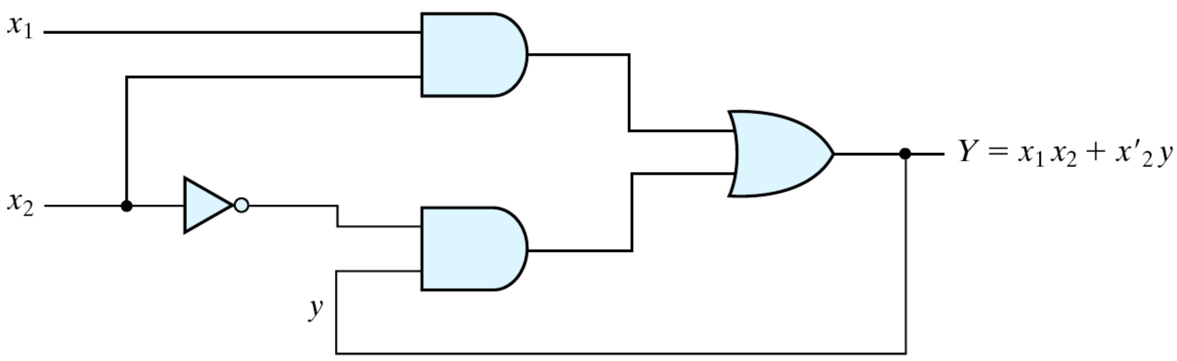
(a) $Y = x_1x_2 + x'_2x_3$



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

Hazards in sequential circuits

✓ An asynchronous example:



(a) Logic diagram

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

(b) Transition table

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

(c) Map for Y

$111 \rightarrow 110$

$111 \rightarrow 010$

Remove Hazards with Latches

- The implementation of the asynchronous circuits with SR latches can remove static hazards

- SR Latch (NOR type)
  - Allow 1-hazard (a momentary 0 has no effect)
  - The network realizing S and R must be free of 0-hazards.

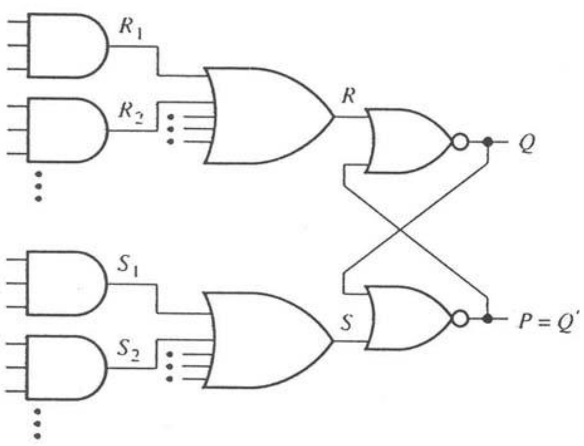
- S'R' Latch (NAND type)
  - Allow 0-hazard (a momentary 1 has no effect)
  - The network realizing S and R must be free of 1-hazards.

- Note that a sum-of-products implementation is automatically free of static 0-hazards and a product-of-sums implementation is free of static 1-hazards.

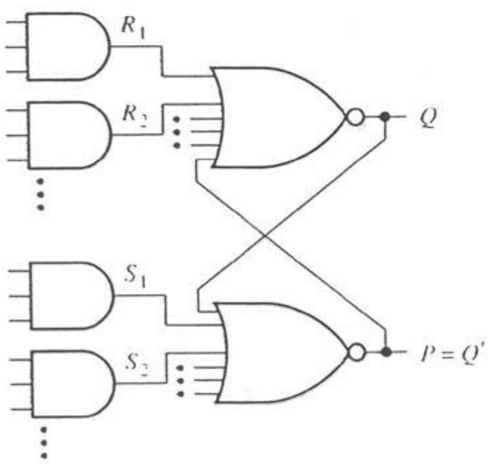
Hazard-Free Realization

✓ S-R Latch (NOR type):

S-R Flip-Flop Driven by
2-level AND-OR Networks



Equivalent Network Structure
(in general faster)



Example

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

  $S = AB + CD$

  $R = A'C'$

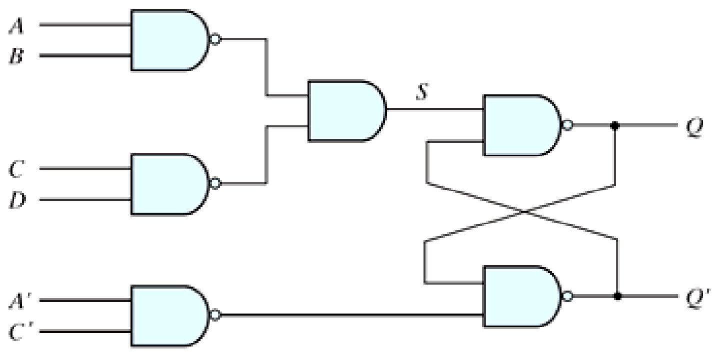
- ✓ If we want to use a NAND latch we must complement the value for S and R

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

  $R = (A'C')'$

- ✓ The Boolean function for output is

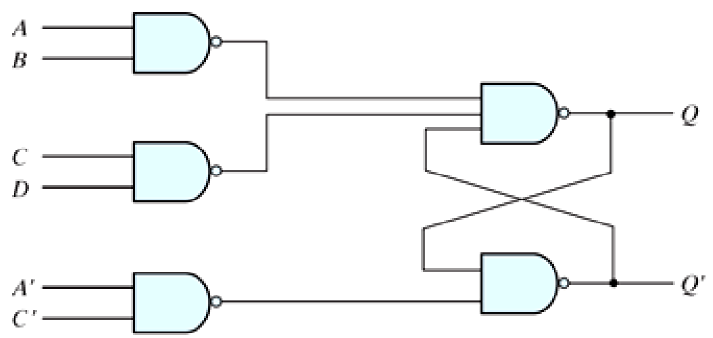
  $Q = (Q'S)' = [Q' (AB)'(CD)']'$



(a)

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

  A product-of-sums implementation is automatically free of static 1-hazards.



(b)

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

Essential Hazards

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

Design Example

Recommended Design Procedure:

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

Design Example

1) Design Specifications:

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

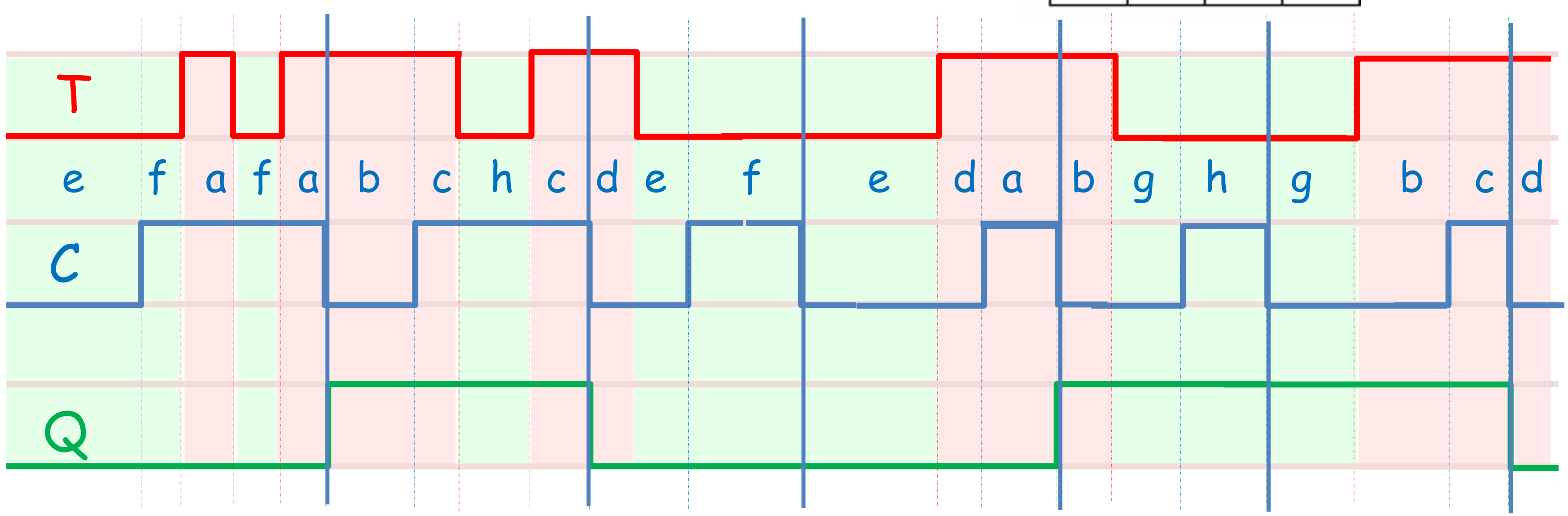
- A Negative-Edge-Triggered T FF
- Two inputs : T, C
- Flip-Flop changes state when T = 1 and C changes from 1 to 0
- Q remains constant under all other conditions
- T and C do not change simultaneously

Design Example

2) Primitive Flow Table

State	Inputs		Output
	T	C	Q
a	1	1	0
b	1	0	1
c	1	1	1
d	1	0	0
e	0	0	0
f	0	1	0
g	0	0	1
h	0	1	1

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



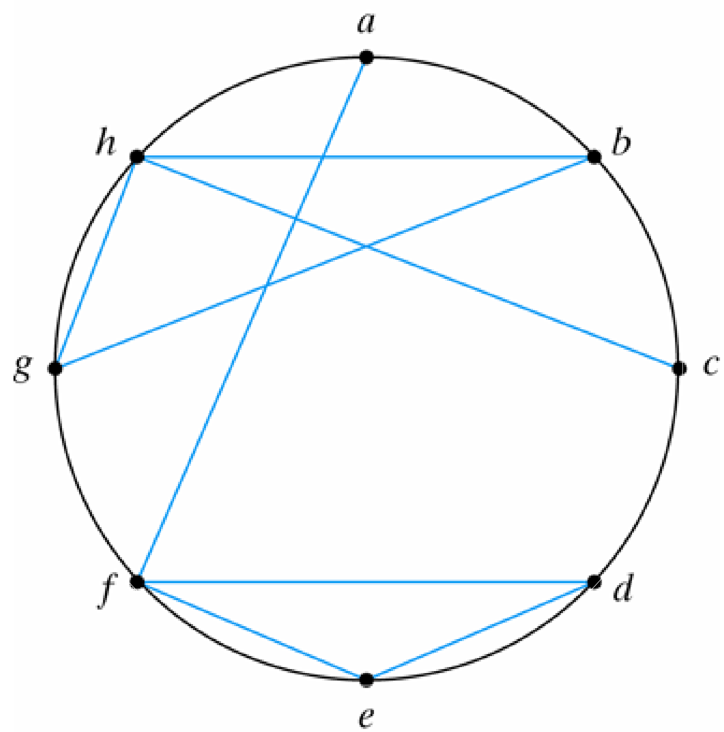
Design Example

3) Merging of the Flow Table

Implication Table

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*

Merger Diagram



Compatible pairs:

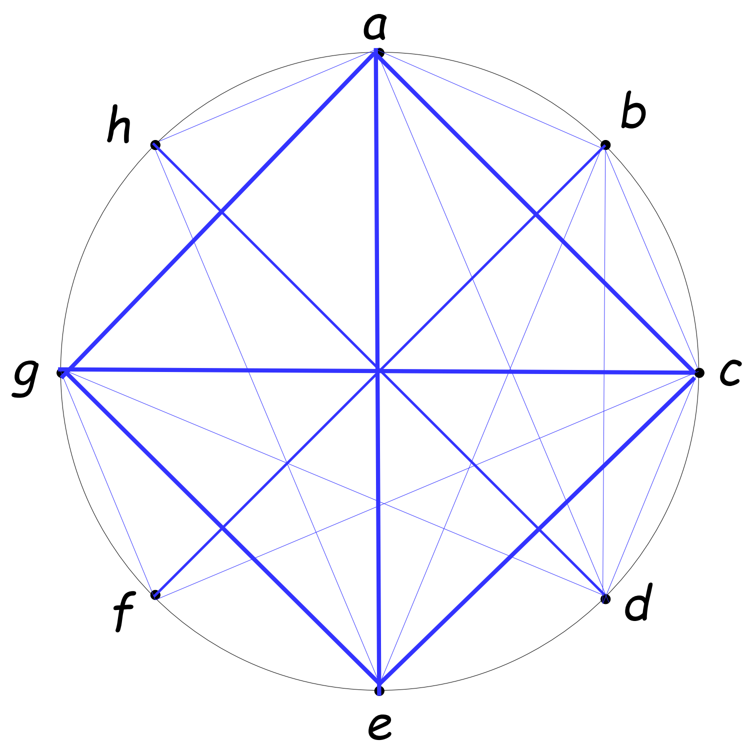
$(a,f)\ (b,g)\ (b,h)\ (c,h)\ (d,e)\ (d,f)\ (e,f)\ (g,h)$

The maximal compatibles pairs are:

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

U=4

Maximal Incompatibles



L=4

	a	b	c	d	e	f	g
b	a, c ×						
c	×	b, d ×					
d	b, d ×	×	a, c ×				
e	b, d ×	e, g × b, d ×	f, h ×	✓			
f	✓	e, g × a, c ×	f, h × a, c ×	✓	✓		
g	f, h ×	✓	b, d ×	e, g × b, d ×	×	e, g × f, h ×	
h	f, h × a, c ×	✓	✓	d, e × c, f ×	e, g × f, h ×	×	✓

Design Example

✓ In this particular example, the minimal collection of compatibles is also the maximal compatibles set that satisfy also the closed condition:

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

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

(a)

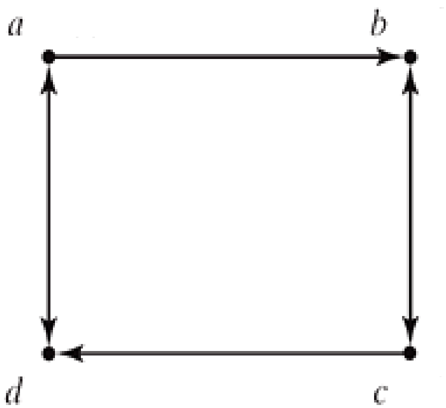
	TC 00	01	11	10
a	d, –	a, 0	a, 0	b, –
b	b, 1	b, 1	c, –	b, 1
c	b, –	c, 1	c, 1	d, –
d	d, 0	d, 0	a, –	d, 0

(b)

Design Example

4) State Assignment and Transition Table

✓ No diagonal lines in the transition diagram: No need to add extra states (**race-free binary state assignment!)**



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 $Q = y_2$

Design Example

5) Logic Diagram

y_1y_2 \ TC	00	01	11	10
00	1	0	0	0
01	0	0	1	0
11	0	X	X	X
10	X	X	0	X

(a) $S_1 = y_2\,TC + y'_2\,T'C'$

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) $R_2 = y_1\,TC'$