University Of Diyala
College Of Engineering
Department of Computer Engineering

Digital System Design II
Asynchronous Sequential Logic
Part II

Dr. Yasir Al-Zubaidi

Third stage

2021

Outline

- Asynchronous Sequential Circuits
- Analysis Procedure
- Circuits with Latches
- **Design Procedure**
- Reduction of State and Flow Tables
- Race-Free State Assignment
- Hazards
- Design Example

Design Procedure

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

Primitive Flow Table

- Design example: gated latch
- Two Input (G = gate , D = Data).
- One output is Q , the gated latch is a memory element that;
  - Accept the value of D when G=1
  - Retain this value after G goes to 0 (D has no effects now)
  - Obtain the flow table by listing all possible states.
  - Dash marks are given when both inputs change simultaneously
  - Outputs of unstable states are don't care

	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

	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 , –

Reduce the 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
- Formal reduction procedure is given in next section

DG

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

DG

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

(a) States that are candidates for merging

DG

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

DG

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

(b) Reduced table (two alternatives)

Transition Table and Logic Diagram

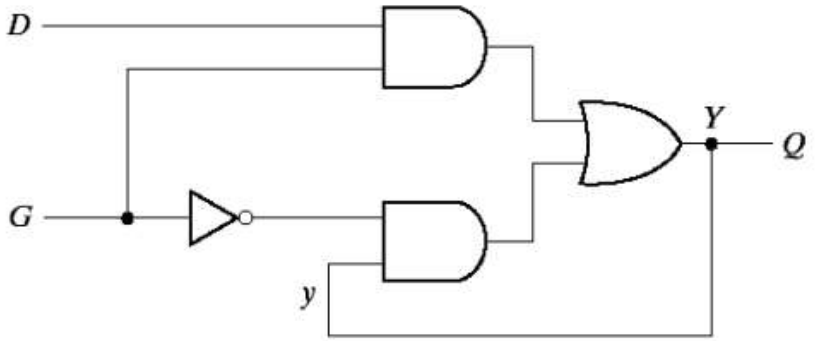
y \ DG	00	01	11	10
0	0	0	1	0
1	1	0	1	1

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

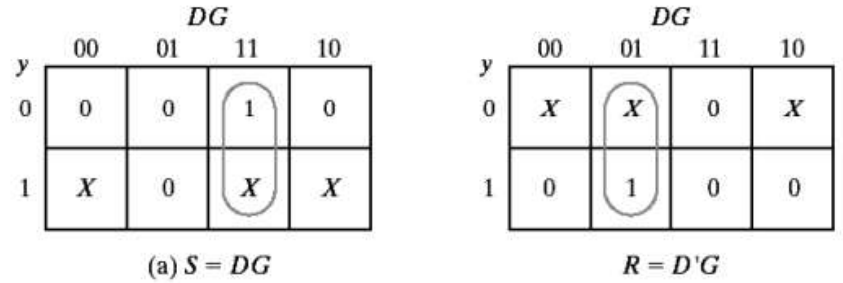
y \ DG	00	01	11	10
0	0	0	1	0
1	1	0	1	1

(b) $Q = Y$

- **Assign a binary value to each state to generate the transition table**
  - **a=0, b=1 in this example**
- **Directly use the simplified Boolean function for the excitation variable Y**
  - **An asynchronous circuit without latch is produced**



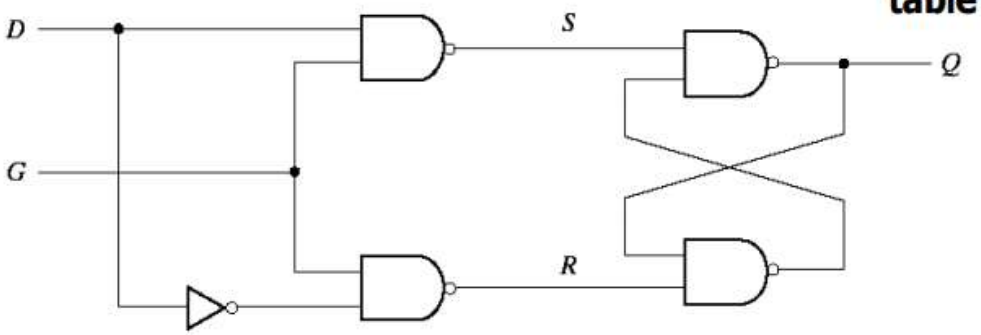
Implementation with SR Latch



(a) $S = DG$ $R = D'G$

(a) Maps for S and R

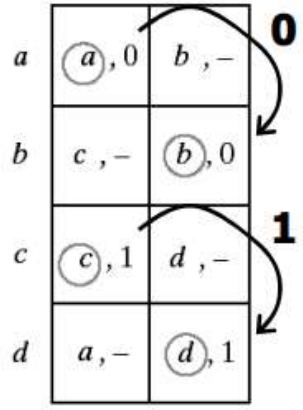
Listed according to the transition table and the excitation table of SR latch



(b) Logic diagram

Outputs for Unstable States

- **Objective: no momentary false outputs occur when the circuit switches between stable states**
- **If the output value is not changed, the intermediate unstable state must have the same output value**
  - **0 → 1 (unstable) → 0 (X)**
  - **0 → 0 (unstable) → 0 (O)**
- **If the output value changed, the intermediate outputs are don't care**
  - **It makes no difference when the output change occurs**



(a) Flow table

0	(0)
X	0
1	(1)
X	1

(b) Output assignment

Outline

- Asynchronous Sequential Circuits
- Analysis Procedure
- Circuits with Latches
- Design Procedure
- Reduction of State and Flow Tables
- Race-Free State Assignment
- Hazards
- Design Example

State Reduction

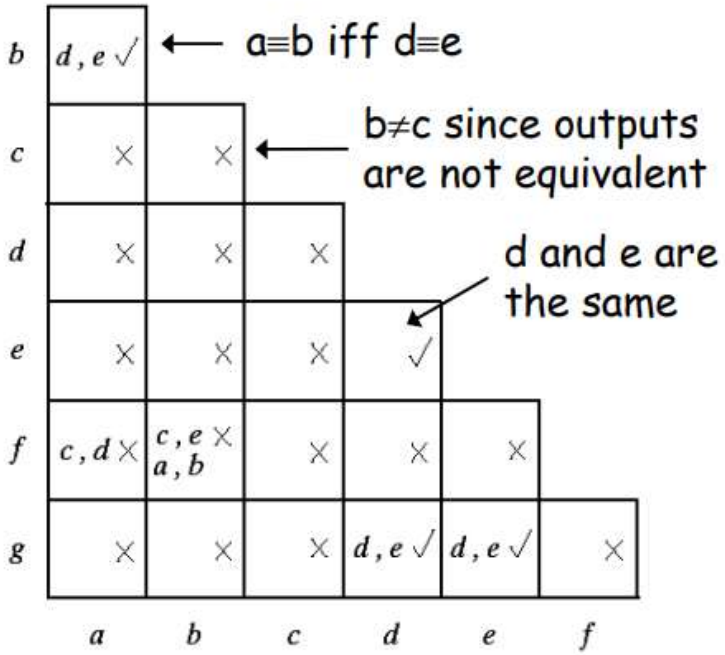
- Two states are equivalent if they have the ***same output*** and go to the ***same*** (equivalent) ***next states*** for each possible input
  - Ex: (a,b) are equivalent
    (c,d) are equivalent
- State reduction procedure is similar in both sync. & async. sequential circuits
  - For completely specified state tables:
    → use implication table
  - For incompletely specified state tables:
    → use compatible pairs

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

Implication Table Method (1/2)

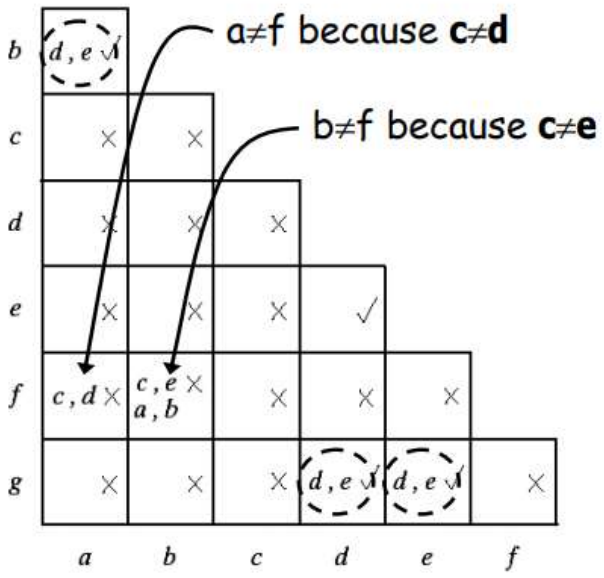
- **Step 1: build the implication chart**

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



Implication Table Method (2/2)

- **Step 2: delete the node with unsatisfied conditions**
- **Step 3: repeat Step 2 until equivalent states found**



equivalent states :
(a,b) (d,e) (d,g) (e,g)

d == e == g

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

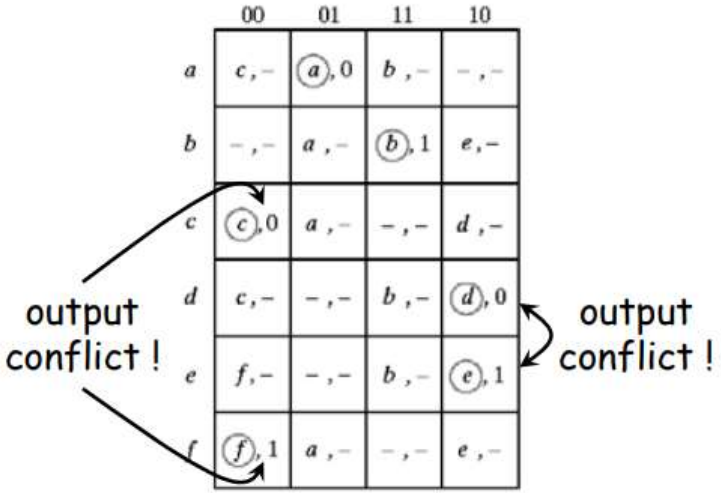
Reduced State Table

Merge the Flow Table

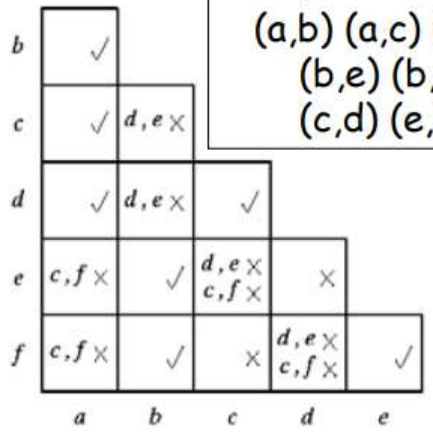
- The state table may be incompletely specified
  - Some next states and outputs are don't care
- Primitive flow tables are always incompletely specified
  - Several synchronous circuits also have this property
- Incompletely specified states are not "equivalent"
  - Instead, we are going to find "***compatible***" 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 compatibles

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**



(a) Primitive flow table

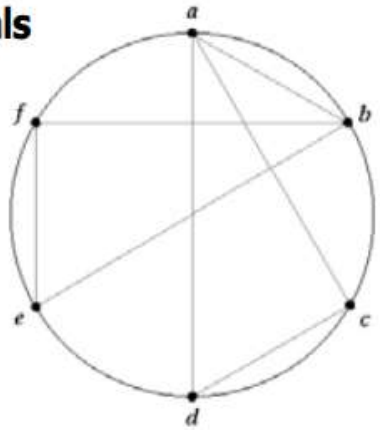


(b) Implication table

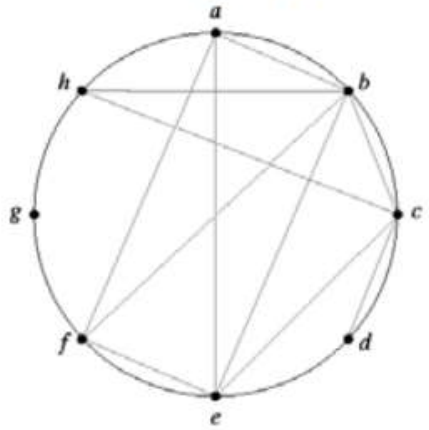
compatible pairs :
(a,b) (a,c) (a,d)
(b,e) (b,f)
(c,d) (e,f)

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



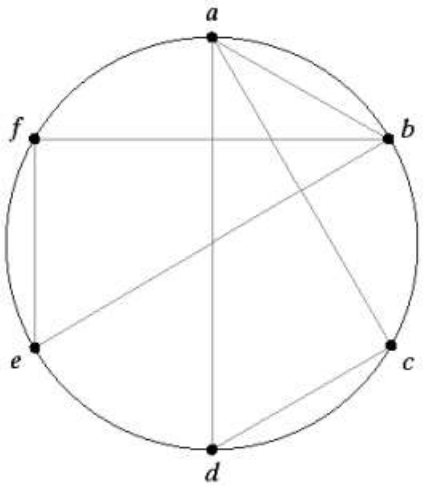
(a) Maximal compatible:
$(a, b,)\ (a, c, d)\ (b, e, f)$



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

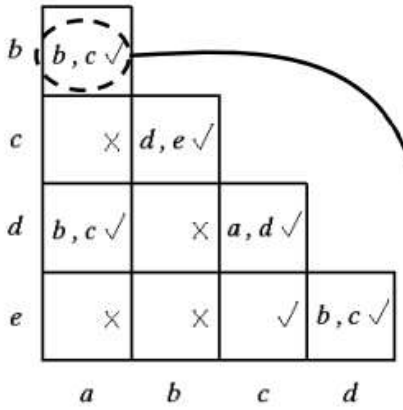
Closed Covering Condition

- The set of chosen compatibles must cover all the states and must be closed
  - Closed covering
- The closure condition is satisfied if
  - There are no implied states
  - The implied states are included within the set
- Ex: if remove (a,b) in the right
  - (a,c,d) (b,e,f) are left in the set
  - All six states are still included
  - No implied states according to its implication table 9-23(b)

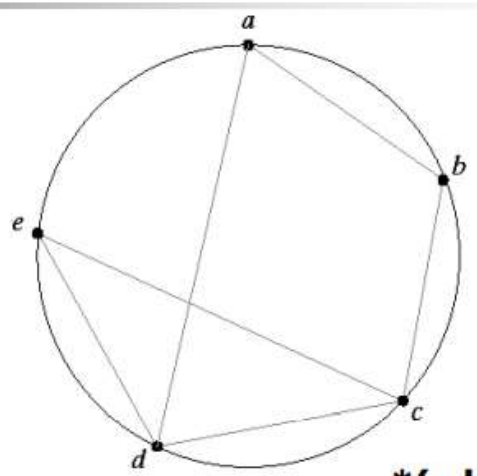


(a) Maximal compatible:
$(a, b,)$ (a, c, d) (b, e, f)

Closed Covering Example



(a) Implication table



(b) Merger diagram

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

***(a,b) (c,d,e) → (X) implied (b,c) is not included in the set**

*** better choice: (a,d) (b,c) (c,d,e) all implied states are included**

Outline

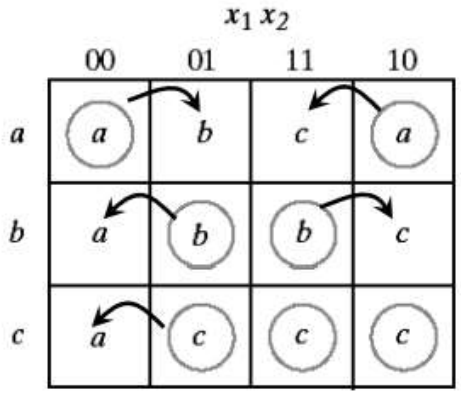
- Asynchronous Sequential Circuits
- Analysis Procedure
- Circuits with Latches
- Design Procedure
- Reduction of State and Flow Tables
- Race-Free State Assignment
- Hazards
- Design Example

Race-Free State Assignment

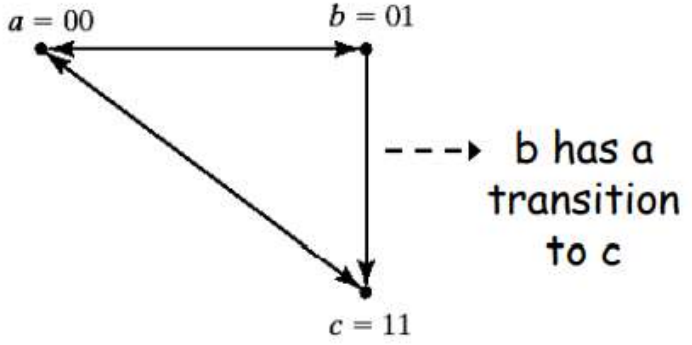
- Objective: choose a proper binary state assignment to ***prevent critical races***
- Only one variable can change at any given time when a state transition occurs
- States between which transitions occur will be given ***adjacent*** assignments
  - Two binary values are said to be adjacent if they 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 (1/2)

- Three states require two binary variables
- Outputs are omitted for simplicity
- Adjacent info. are represented by a transition diagram
- a and c are still not adjacent in such an assignment !!
  - Impossible to make all states adjacent if only 3 states are used



(a) Flow table

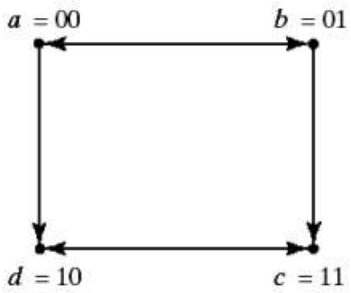


(b) Transition diagram

3-Row Flow Table Example (2/2)

- 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)

	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	–



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

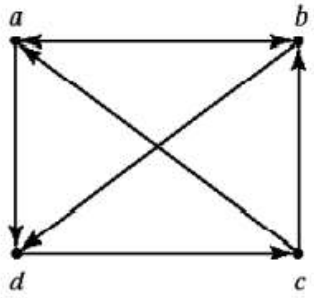
	x_1x_2 = 00	01	11	10
$a = 00$	(00)	01	10	(00)
$b = 01$	00	(01)	(01)	11
$c = 11$	10	(11)	(11)	(11)
$d = 10$	00	–	11	–

4-Row Flow Table Example (1/2)

- Sometimes, just one extra row may not be sufficient to prevent critical races
  - More binary state variables may also required
- With one or two diagonal transitions, there is no way of using two binary variables that satisfy all adjacency

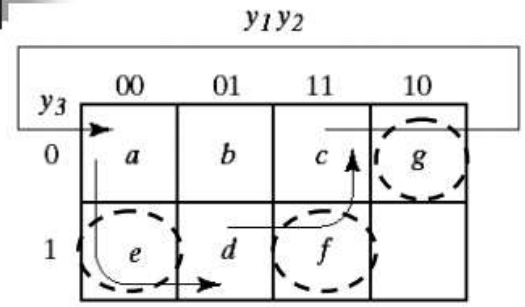
	00	01	11	10
a	*b*	(*a*)	*d*	(*a*)
b	(*b*)	*d*	(*b*)	*a*
c	(*c*)	*a*	*b*	(*c*)
d	*c*	(*d*)	(*d*)	*c*

(a) Flow table

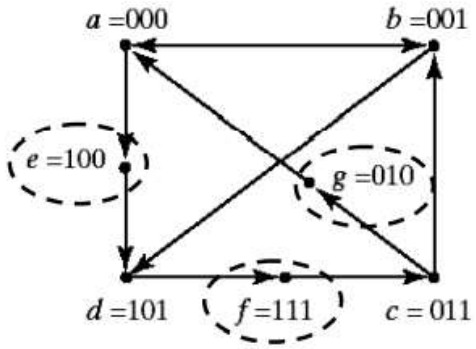


(b) Transition diagram

4-Row Flow Table Example (2/2)



(a) Binary assignment



(b) Transition diagram

	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*	–

still has 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**

can be used to any 4-row flow table

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

(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

Outline

- Asynchronous Sequential Circuits
- Analysis Procedure
- Circuits with Latches
- Design Procedure
- Reduction of State and Flow Tables
- Race-Free State Assignment
- Hazards
- Design Example

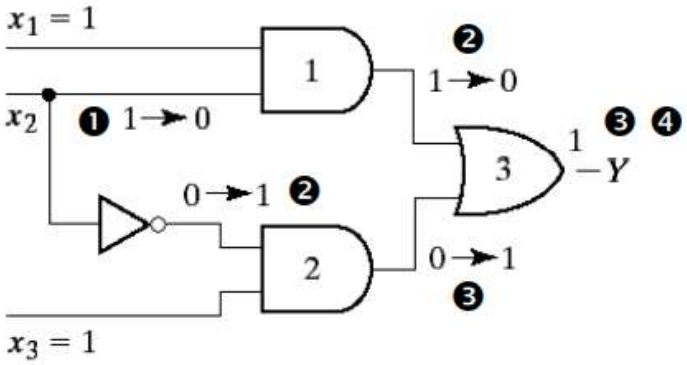
Hazards

- **Unwanted switching appears at the output of a circuit**
  - **Due to different propagation delay in different paths**
- **May cause the circuit to mal-function**
  - **Cause temporary false-output values in combinational circuits**
  - **Cause a transition to a wrong state in asynchronous circuits**
  - **Not a concern to synchronous sequential circuits**
- **Three types of hazards:**

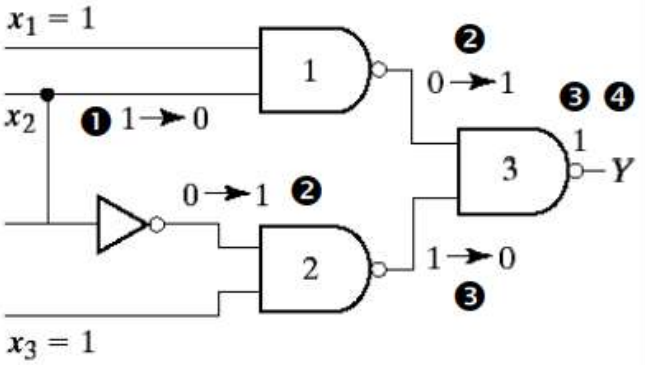
(a) Static 1-hazard (b) Static 0-hazard (c) Dynamic hazard

Circuits with 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

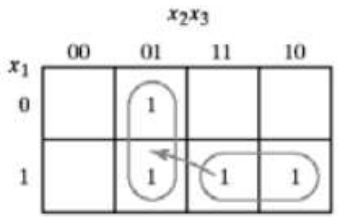


(b) NAND circuit

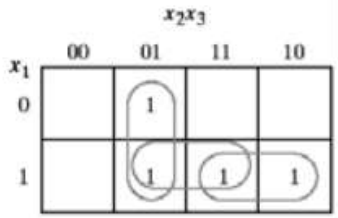
Hazard-Free Circuit

- Hazard can be detected by inspecting the map
- The change of input results in a change of covered product term

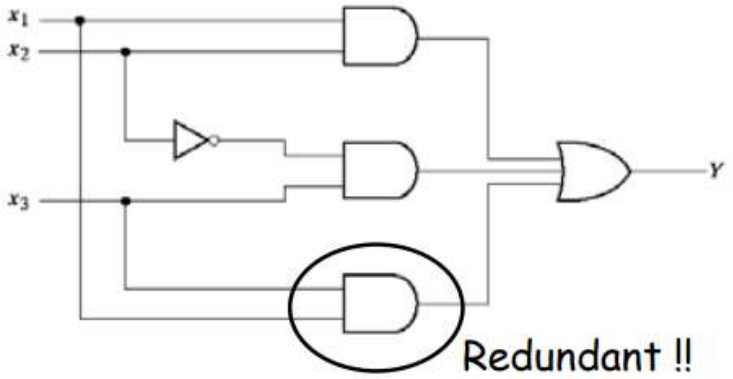
  → Hazard exists
  - Ex: 111 → 101 in (a)
- To eliminate the hazard, enclose the two minterms in another product term
  - Results in redundant gates



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

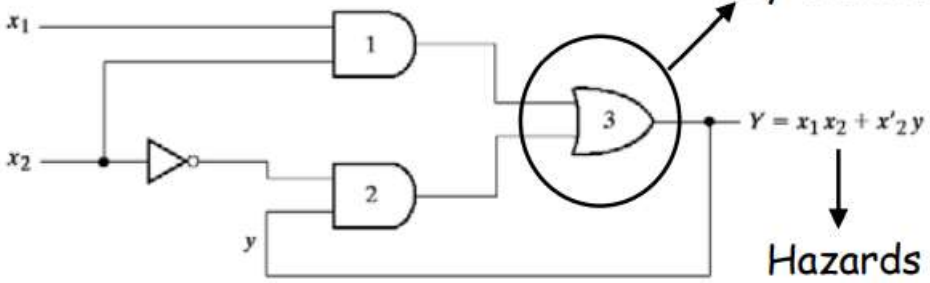


(b) $Y = x_1x_2 + x'_2x_3 + x_1x_3$



Remove Hazard with Latches

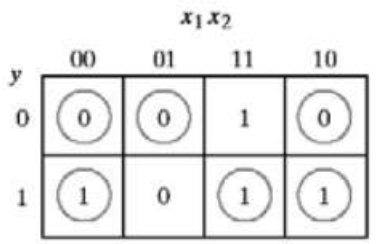
- **Implement the asynchronous circuit with SR latches can also remove static hazards**
  - **A momentary 0 has no effects to the S and R inputs of a NOR latch**
  - **A momentary 1 has no effects to the S and R inputs of a NAND latch**



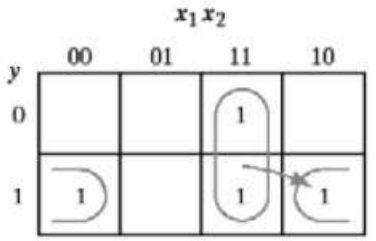
(a) Logic diagram

Replaced by a latch

Hazards exist !!



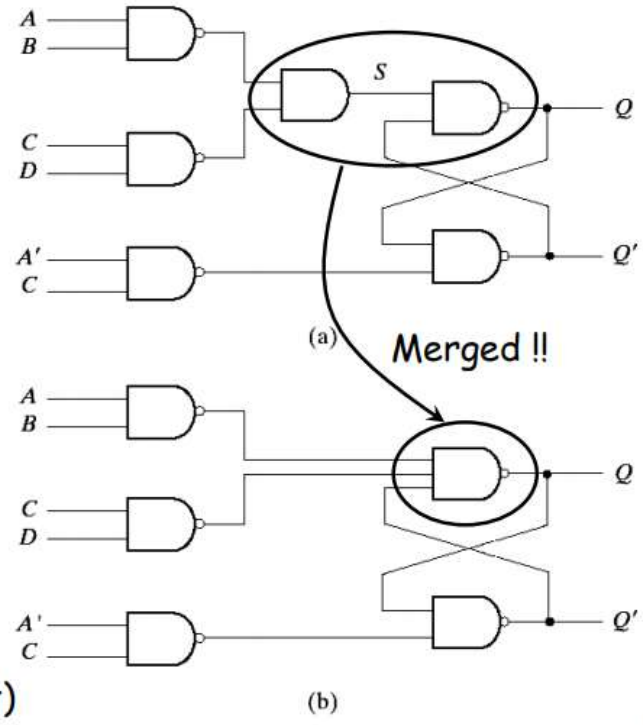
(b) Transition table



(c) Map for Y

Implementation with SR Latches

- **Given:**
  S = AB + CD
  R = A'C
- **For NAND latch, use complemented inputs**
  - **S' = (AB + CD)'**
    = (AB)'(CD)'
  - **R' = (A'C)'**
- **Q = (Q'S)'**
  = [Q'(AB)'(CD)']'
  → Two-level circuits
  (this is the output we want)



Essential Hazards

- Besides static and dynamic hazards, another type of hazard in asynchronous circuits is called ***essential hazard***
- 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 !!

Outline

- Asynchronous Sequential Circuits
- Analysis Procedure
- Circuits with Latches
- Design Procedure
- Reduction of State and Flow Tables
- Race-Free State Assignment
- Hazards
- Design Example

Recommended Design Procedure

1. State the design specifications
2. Derive a primitive flow table
3. Reduce the flow table by merging the 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

Primitive Flow Table

- Design a negative-edge-triggered T flip-flop
- Two inputs: T(toggle) and C(clock)
  - T=1: toggle, T=0: no change
- One output: Q

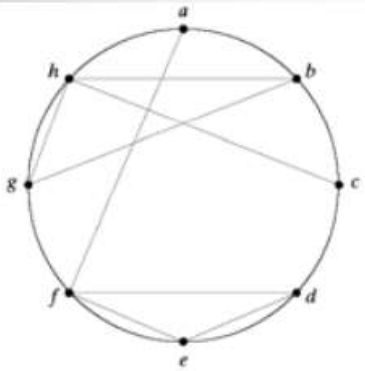
State	Input T	Input C	Output Q	Comments
a	1	1	0	Initial output is 0
b	1	0	1	After state a
c	1	1	1	Initial output is 1
d	1	0	0	After state c
e	0	0	0	After states d or f
f	0	1	0	After states e or a
g	0	0	1	After states b or h
h	0	1	1	After states g or c

	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 , –	– , –

Merging the Flow Table

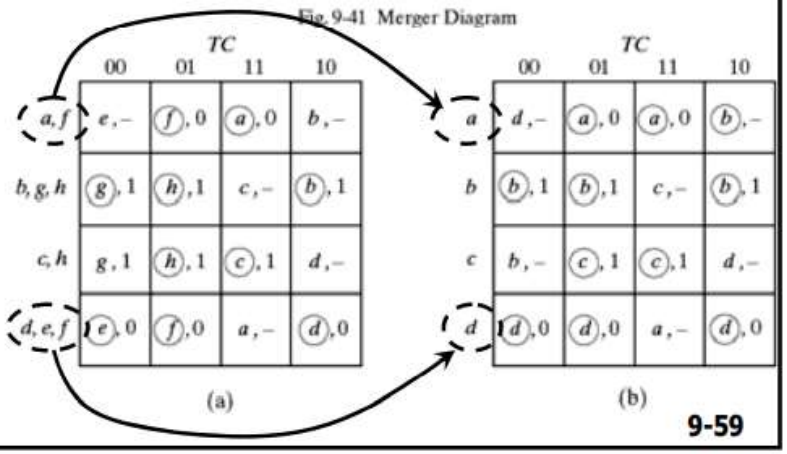
Compatible pairs:
(a,f) (b,g) (b,h) (c,h)
(d,e) (d,f) (e,f) (g,h)

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



	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 ×	×	✓

Fig. 9-41 Merger Diagram



State Assignment & Transition Table

- **No diagonal lines in the transition diagram**

  → **No need to add extra states**

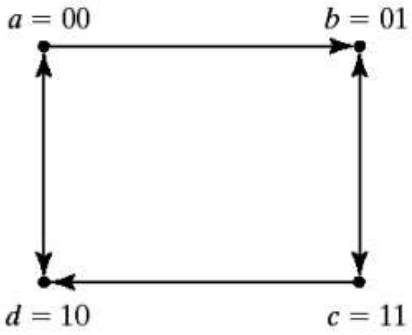


Fig. 9-43 Transition Diagram

$y_1 y_2$	TC = 00	TC = 01	TC = 11	TC = 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

$y_1 y_2$	TC = 00	TC = 01	TC = 11	TC = 10
00	0	0	0	*X*
01	1	1	1	1
11	1	1	1	*X*
10	0	0	0	0

(b) Output map $Q = y_2$

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) $R_2 = y_1\,TC'$