Chapter 9

Asynchronous Sequential Logic

9-1

Outline

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

9-2

Sequential Circuits

- Consist of a combinational circuit to which storage elements are connected to form a feedback path
- Specified by ***a time sequence of inputs***, ***outputs***, and ***internal states***
- Two types of sequential circuits:
  - Synchronous
  - Asynchronous

primary difference

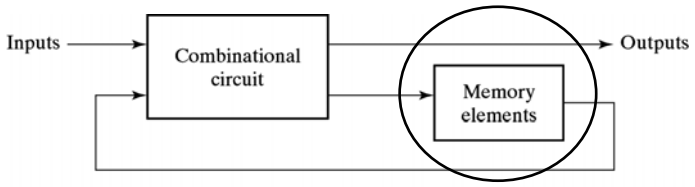
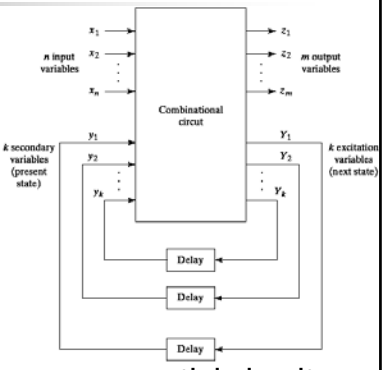
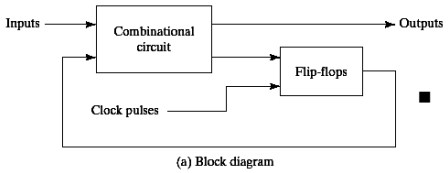


Fig. 5-1 Block Diagram of Sequential Circuit

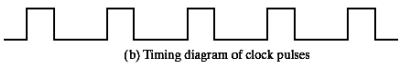
Synchronous vs. Asynchronous

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





(a) Block diagram



(b) Timing diagram of clock pulses

- Synchronous sequential circuits
  - Synchronized by a ***periodic*** train of clock pulses
  - Much easier to design (preferred design style)

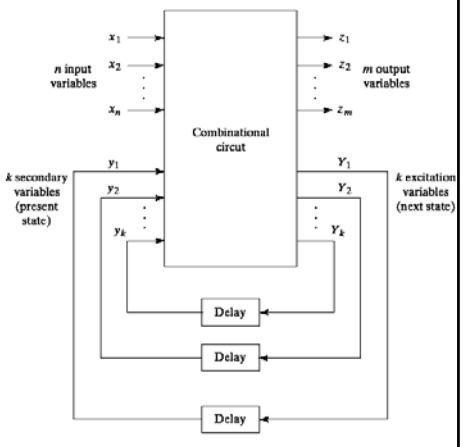
Why Asynchronous Circuits ?

- Used when speed of operation is important
  - Response quickly without waiting for a clock pulse
- Used in small independent systems
  - Only a few components are required
- Used when the input signals may change independently of internal clock
  - Asynchronous in nature
- Used in the communication between two units that have their own independent clocks
  - Must be done in an asynchronous fashion

9-5

Definitions of Asyn. Circuits

- Inputs / Outputs
- Delay elements:
  - Only a short term memory
  - May not really exist due to original gate delay
- Secondary variable:
  - Current state (small y)
- Excitation variable:
  - Next state (big Y)
  - Have some delay in response to input changes



9-6

Operational Mode

- Steady-state condition:
  - Current states and next states are the same
  - Difference between Y and y will cause a transition
- Fundamental mode:
  - No simultaneous changes of two or more variables
  - The time between two input changes must be longer than the time it takes the circuit to a stable state
  - The input signals change one at a time and only when the circuit is in a stable condition

9-7

Outline

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

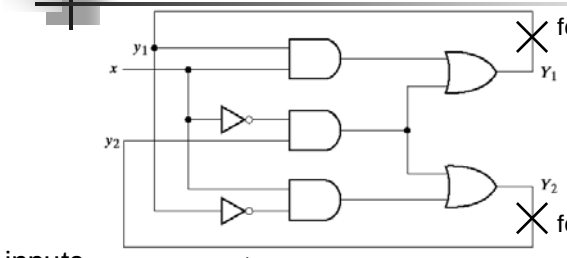
9-8

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
  5. Circle those values of Y in each square that are equal to the value of y in the same row

9-9

An Example of Transition Table



$Y_1 = xy_1 + x'y_2$

$Y_2 = xy'_1 + x'y_2$

$Y = Y_1Y_2$

inputs

current states

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

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

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

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

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

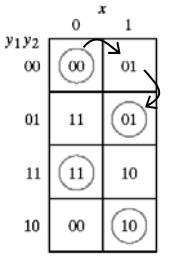
(c) Transition table

stable !!

9-10

State Table

- When input x changes from 0 to 1 while y=00:
  - Y changes to 01 → unstable
  - y becomes 01 after a short delay → stable at the second row
  - The next state is Y=01
- Each row must have ***at least one*** stable state
- Analyze each state in this way can obtain its state table



(c) Transition table

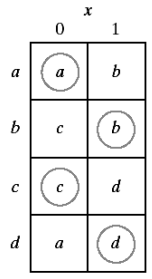
Present State		Next State			
		X=0		X=1	
0	0	0	0	0	1
0	1	1	1	0	1
1	0	0	0	1	0
1	1	1	1	1	0

y_1y_2x : total state

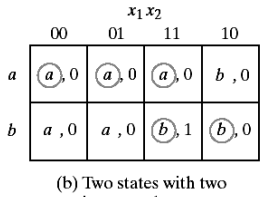
4 stable total states: 000,011, 110,101

Flow Table

- Similar to a transition table except the states are represented by ***letter symbols***
- Can also include the output values
- Suitable to obtain the logic diagram from it
- Primitive flow table: ***only one*** stable state in each row (ex: 9-4(a))



(a) Four states with one input



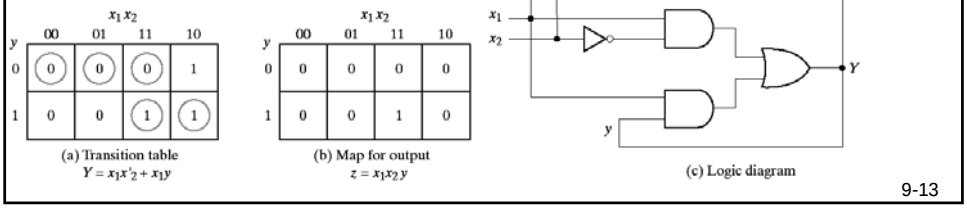
(b) Two states with two inputs and one output

Equivalent to 9-3(c) if a=00, b=01, c=11, d=10

Flow Table to Circuits

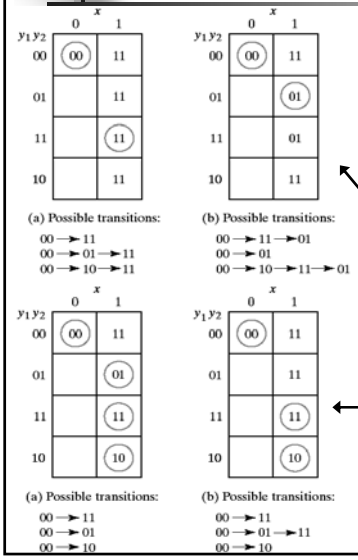
- Procedure to obtain circuits from flow table:
  - Assign to each state a distinct binary value (convert to a transition table)
  - Obtain circuits from the map
- Two difficulties:
  - The binary state assignment (to avoid race)
  - The output assigned to the unstable states

Ex: from the flow table 9-4(b)



9-13

Race Conditions

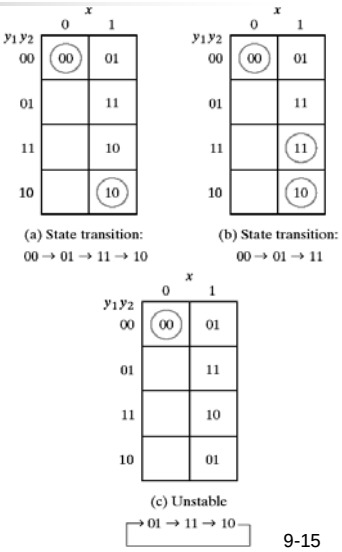


- Race condition:
  - ***two or more*** binary state variables will change value when one input variable changes
  - Cannot predict state sequence if unequal delay is encountered
- Non-critical race:
  - The final stable state ***does not*** depend on the change order of state variables
- Critical race: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

9-14

Race-Free State Assignment

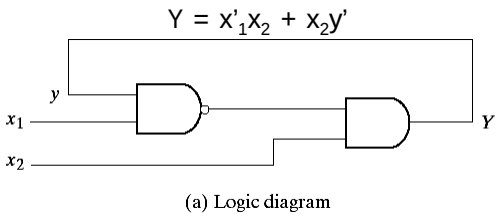
- Race can be avoided by proper state assignment
  - Direct the circuit through intermediate unstable states with a unique state-variable change
  - It is said to have a ***cycle***
- Must ensure that a cycle will terminate with a stable state
  - Otherwise, the circuit will keep going in unstable states
- More details will be discussed in Section 9-6

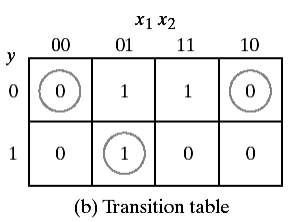


9-15

Stability Check

- Asynchronous sequential circuits may oscillate between unstable states due to the feedback
  - Must check for stability to ensure proper operations
- Can be easily checked from the transition table
  - Any column has no stable states → unstable
  - Ex: when x_1x_2=11 in Fig. 9-9(b), Y and y are never the same





9-16

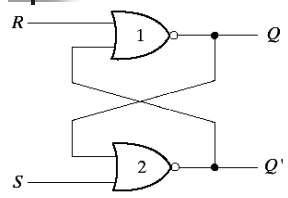
Outline

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

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 also an asynchronous circuit
  - Will be analyzed first using the method for asynchronous circuits

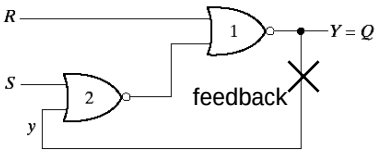
SR Latch with NOR Gates



(a) Crossed-coupled circuit

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

(b) Truth table



(c) Circuit showing feedback

SR

y	00	01	11	10
0	(0)	(0)	(0)	1
1	(1)	0	0	(1)

$Y = SR' + R'y$

$Y = S + R'y$ when $SR = 0$ → *

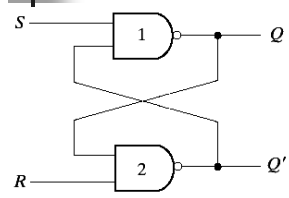
(d) Transition table

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

* should be carefully checked first

9-19

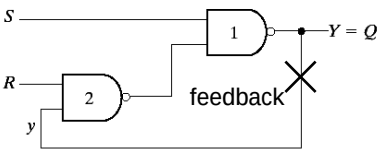
SR Latch with NAND Gates



(a) Crossed-coupled circuit

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

(b) Truth table



(c) Circuit showing feedback

SR

y	00	01	11	10
0	1	1	(0)	(0)
1	(1)	(1)	(1)	0

$Y = S' + Ry$ when $S'R' = 0$ → *

(d) Transition table

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

* should be carefully checked first

9-20

Analysis Procedure

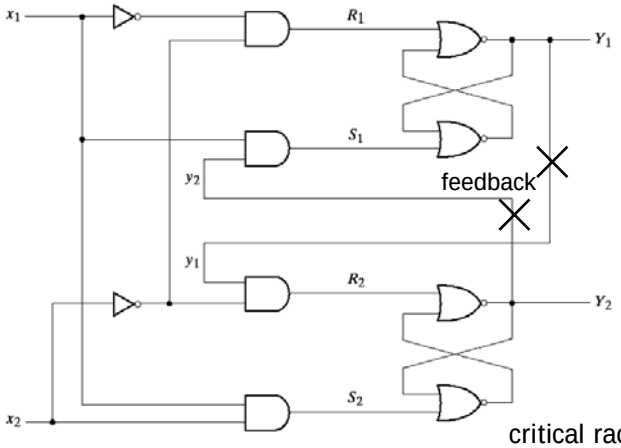
- Procedure to analyze an asynchronous sequential circuits with SR latches:
  1. Label each latch output with Y_i and its external feedback path (if any) with y_i
  2. Derive the Boolean functions for each S_i and R_i
  3. Check whether SR=0 (NOR latch) or S'R'=0 (NAND latch) is satisfied
  4. Evaluate Y=S+R'y (NOR latch) or Y=S'+Ry (NAND latch)
  5. Construct the transition table for $Y=Y_1Y_2...Y_k$
  6. Circle all stable states where Y=y

Analysis Example

$S_1=x_1y_2$ $R_1=x'_1x'_2 \Rightarrow S_1R_1 = x_1y_2x'_1x'_2 = 0$ (OK)
$S_2=x_1x_2$ $R_2=x'_2y_1 \Rightarrow S_2R_2 = x_1x_2x'_2y_1 = 0$ (OK)

$Y_1=S_1 + R'_1y_1$
$=x_1y_2 + (x_1+x_2)y_1$
$=x_1y_2+x_1y_1+x_2y_1$

$Y_2=S_2 + R'_2y_2$
$=x_1x_2 + (x_2+y'_1)y_2$
$=x_1x_2+x_2y_2+y'_1y_2$



y_1y_2 \ x_1x_2	00	01	11	10
00	(00)	(00)	01	(00)
01	(01)	(01)	11	11
11	00	(11)	(11)	10
10	00	(10)	11	(10)

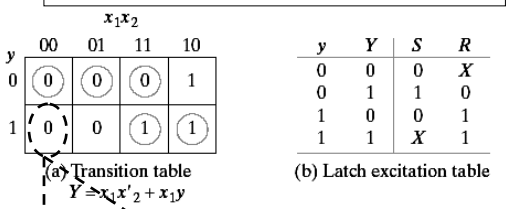
critical race !!

Implementation Procedure

- Procedure to implement an asynchronous sequential circuits with SR latches:
  1. Given a transition table that specifies the excitation function $Y = Y_1Y_2...Y_k$, derive a pair of maps for each S_i and R_i using the latch excitation table
  2. Derive the Boolean functions for each S_i and R_i
     (do not to make Si and Ri equal to 1 in the same minterm square)
  3. 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

Excitation table: list the required S and R for each possible transition from y to Y

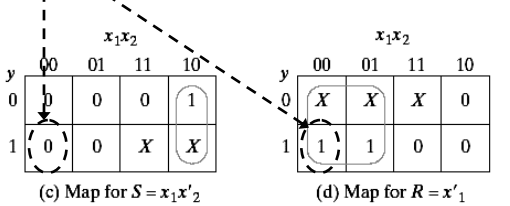


(a) Transition table

$Y = x_1x'_2 + x_1y$

y	Y	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	1

(b) Latch excitation table

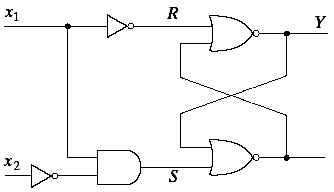


(c) Map for $S = x_1x'_2$

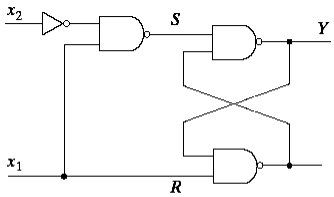
(d) Map for $R = x'_1$

y = 1 (outside) → 0 (inside)

∴ S=0, R=1 from excitation table



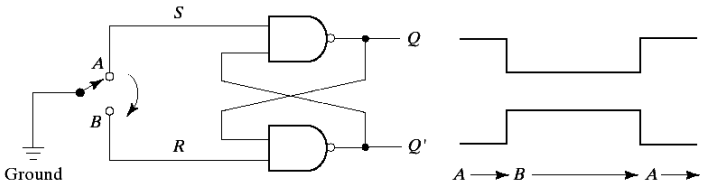
(e) Circuit with NOR latch



(f) Circuit with NAND latch

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)



9-25

Outline

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

9-26

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

9-27

Primitive Flow Table

- Design example: gated latch
  - 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, –

9-28

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

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

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

(b) Reduced table (two alternatives)

9-29

Transition Table and Logic Diagram

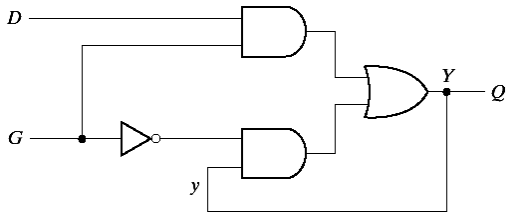
y \ DG	00	01	11	10
0	0	0	1	0
1	1	0	1	1

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

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

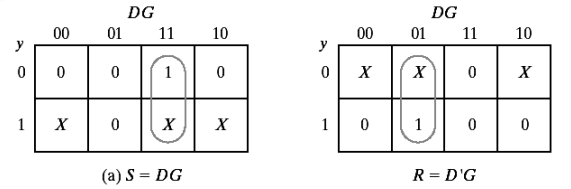
(b) $Q = Y$

- 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



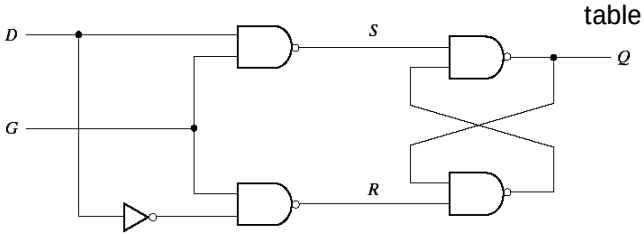
9-30

Implementation with SR Latch



(a) Maps for S and R

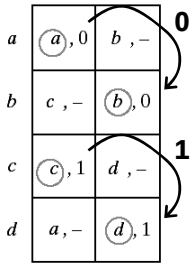
Listed according to the transition table and the excitation table of SR latch



(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

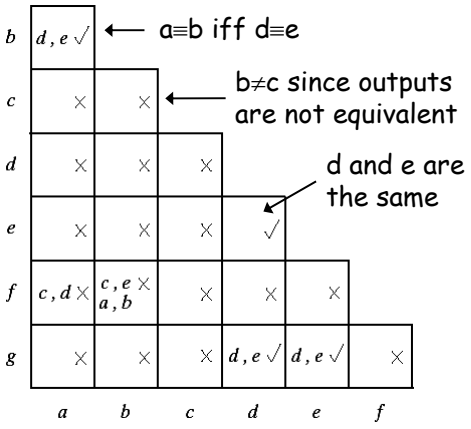
- Two states are equivalent if they have the ***same output*** and go to the ***same*** (equivalent) ***next states*** for each possible input
  - Ex: (a,b) are equivalent
    (c,d) are equivalent
- State reduction procedure is similar in both sync. & async. sequential circuits
  - For completely specified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

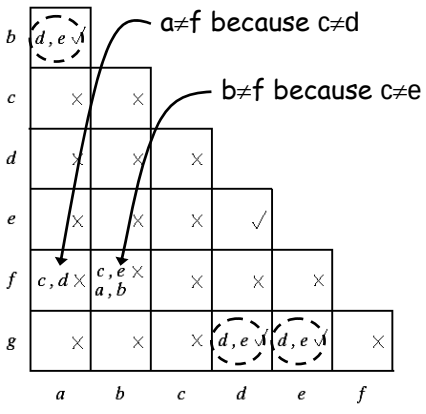
Present State	Next State x=0	Next State x=1	Output x=0	Output x=1
a	d	b	0	0
b	e	a	0	0
c	g	f	0	1
d	a	d	1	0
e	a	d	1	0
f	c	b	0	0
g	a	e	1	0



9-35

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

9-36

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

9-37

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

output conflict !

output conflict !

	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 ×	✓

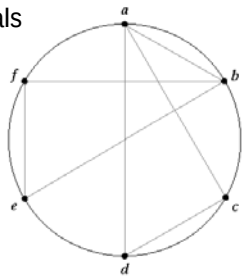
(b) Implication table

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

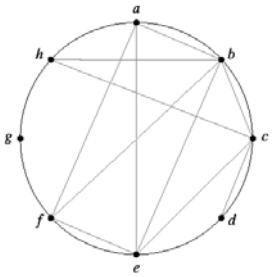
9-38

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

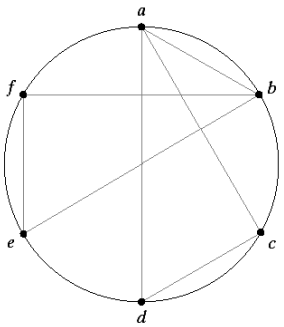


(b) Maximal compatible:
$(a, b, e, f)\ (b, c, h)\ (c, d)\ (g)$

9-39

Closed Covering Condition

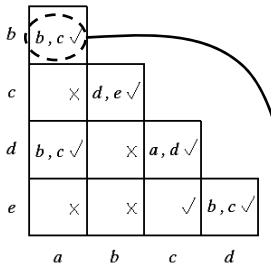
- The set of chosen compatibles must cover all the states and must be closed
  - Closed covering
- The closure condition is satisfied if
  - There are no implied states
  - The implied states are included within the set
- Ex: if remove (a,b) in the right
  - (a,c,d) (b,e,f) are left in the set
  - All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



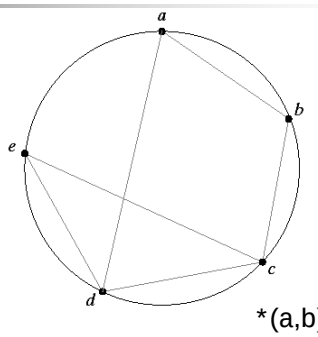
(a) Maximal compatible:
$(a, b,)\ (a, c, d)\ (b, e, f)$

9-40

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

9-41

Outline

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

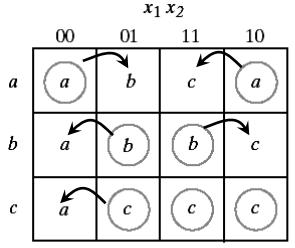
9-42

Race-Free State Assignment

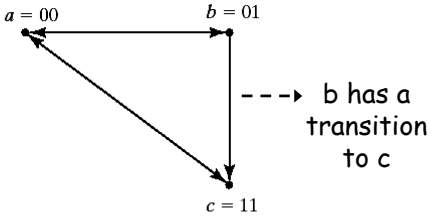
- Objective: choose a proper binary state assignment to ***prevent critical races***
- Only one variable can change at any given time when a state transition occurs
- States between which transitions occur will be given ***adjacent*** assignments
  - Two binary values are said to be adjacent if they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

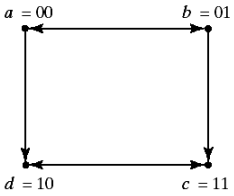


(b) Transition diagram

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)

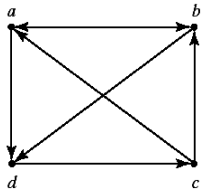
	x_1x_2 00	01	11	10
a = 00	(00)	01	10	(00)
b = 01	00	(01)	(01)	11
c = 11	10	(11)	(11)	(11)
d = 10	00	–	11	–

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

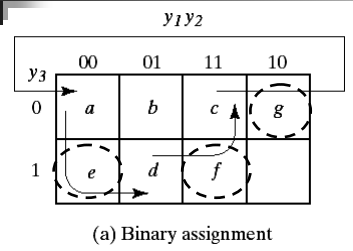
	00	01	11	10
a	b	(a)	d	(a)
b	(b)	d	(b)	a
c	(c)	a	b	(c)
d	c	(d)	(d)	c

(a) Flow table

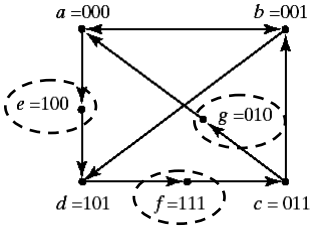


(b) Transition diagram

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

9-47

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

9-48

Outline

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

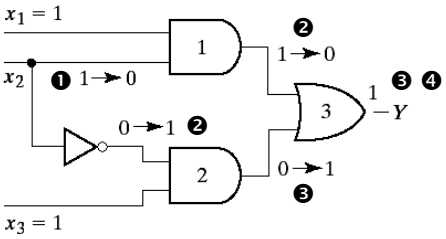
Hazards

- 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:

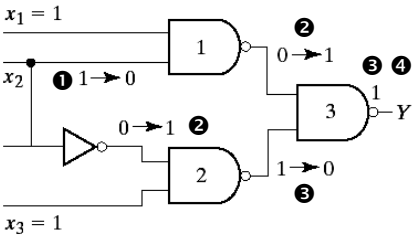
(a) Static 1-hazard (b) Static 0-hazard (c) Dynamic hazard

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

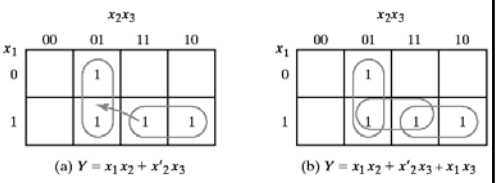


(b) NAND circuit

Hazard-Free Circuit

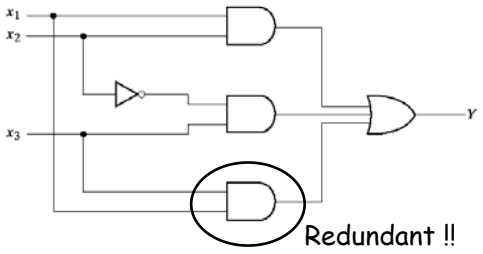
- Hazard can be detected by inspecting the map
- The change of input results in a change of covered product term

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



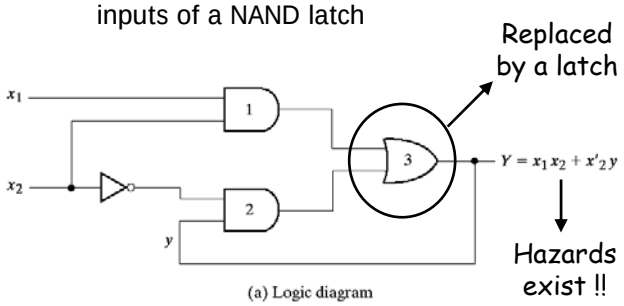
(a) $Y = x_1x_2 + x'_2x_3$

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

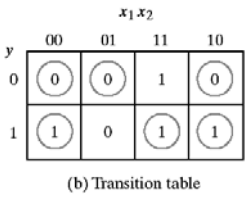


Remove Hazard with Latches

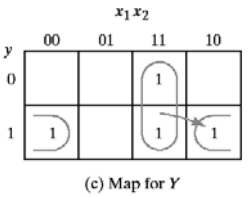
- Implement the asynchronous circuit with SR latches can also remove static hazards
  - A momentary 0 has no effects to the S and R inputs of a NOR latch
  - A momentary 1 has no effects to the S and R inputs of a NAND latch



(a) Logic diagram



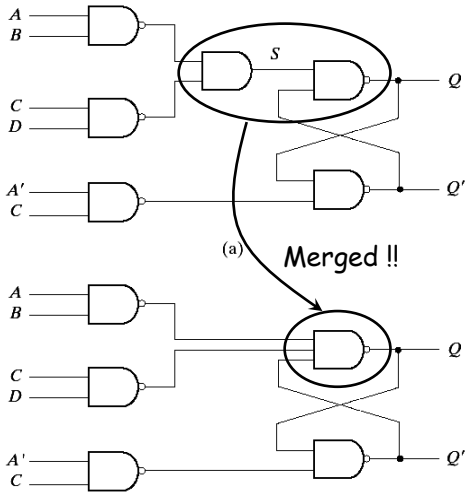
(b) Transition table



(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)



(b)

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 !!

9-55

Outline

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

9-56

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

9-57

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

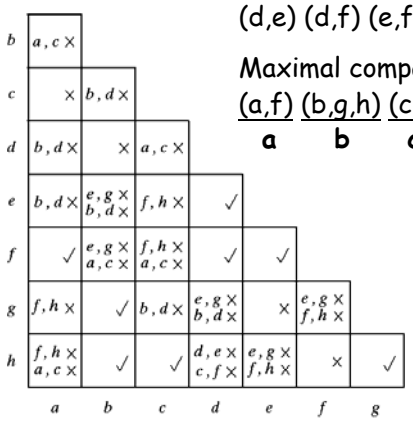
	TC = 00	01	11	10
a	– , –	f , –	(a), 0	b , –
b	g , –	– , –	c , –	(b), 1
c	– , –	h , –	(c), 1	d , –
d	e , –	– , –	a , –	(d), 0
e	(e), 0	f , –	– , –	d , –
f	e , –	(f), 0	a , –	– , –
g	(g), 1	h , –	– , –	b , –
h	g , –	(h), 1	c , –	– , –

9-58

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**



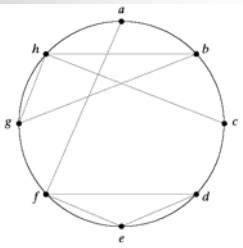
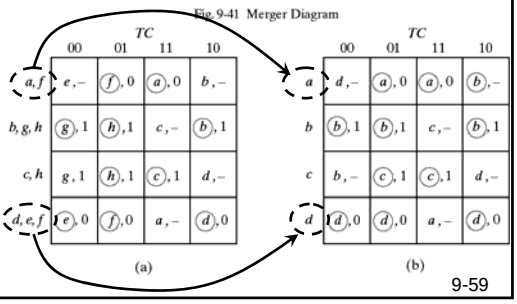


Fig. 9-41 Merger Diagram



State Assignment & Transition Table

- No diagonal lines in the transition diagram
  → No need to add extra states

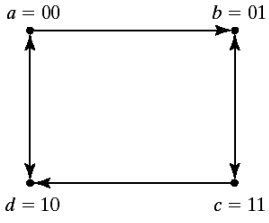


Fig. 9-43 Transition Diagram

TC

y_1y_2	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)

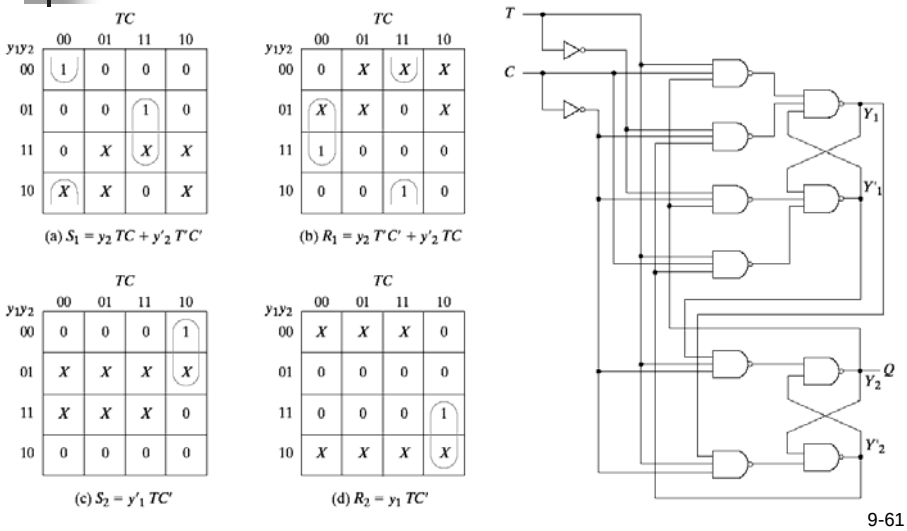
(a) Transition table

TC

y_1y_2	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 $Q = y_2$

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) $R_2 = y_1\,TC'$

9-61