EDA322
Digital Design

Lecture 19:

Asynchronous Sequential Logic

Ioannis Sourdis

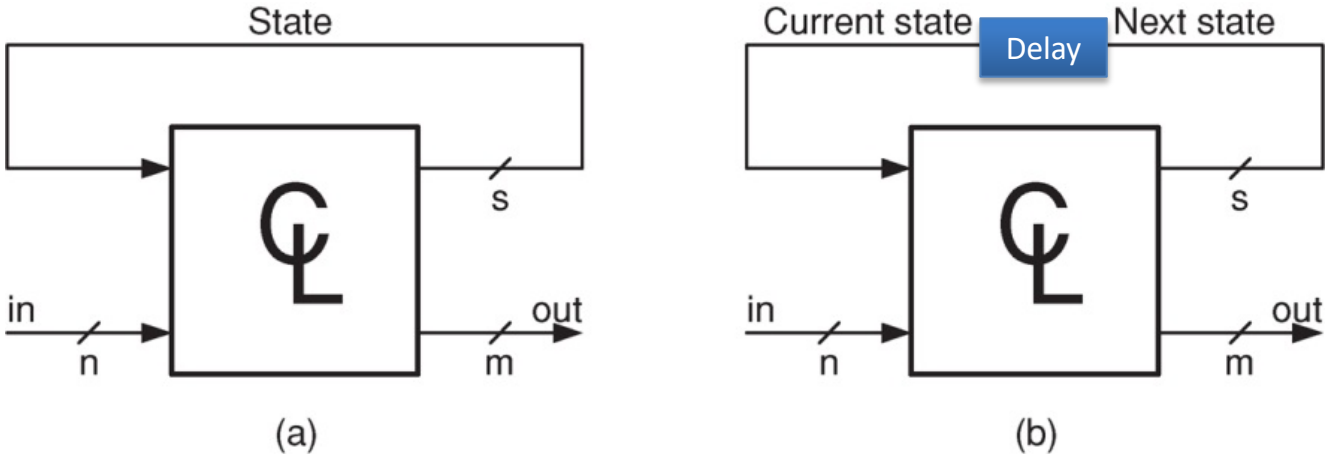
Outline of Lecture 19

- Introduction to asynchronous circuits
- Analysis and synthesis of asynchronous circuits
- Race conditions and state assignment
- State minimization
- Asynchronous design example
- Hazards

Disclaimer many slides of this lecture taken from ECE124 Digital Circuits and Systems course

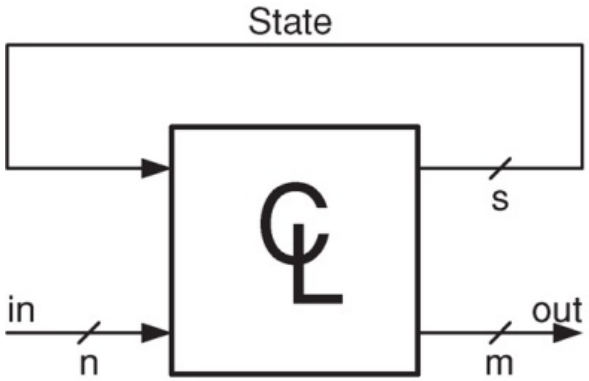
Asynchronous Sequential Circuits

- A type of circuit has state that is not synchronized with a clock **(therefore NO flip-flops)**
- They are realized by adding a **feedback** to the combinational logic that implements the next state
- State variables may change at any point in time

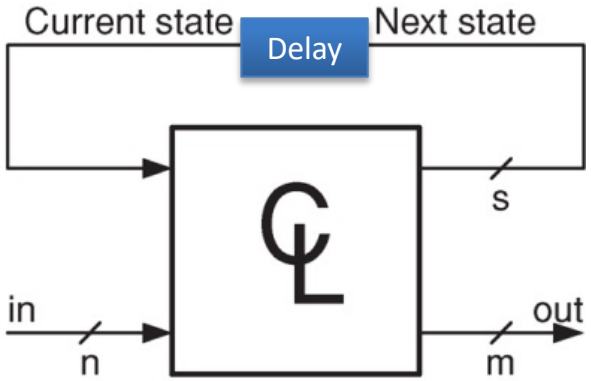


Asynchronous Sequential Circuits

- Should worry about
  - **Hazards:** glitches in the combinational logic may cause going to wrong state), and
  - **Races** between state bits: more than one state bits changing in a random order may cause going to the wrong state



(a)



(b)

Definitions important for asynchronous circuits

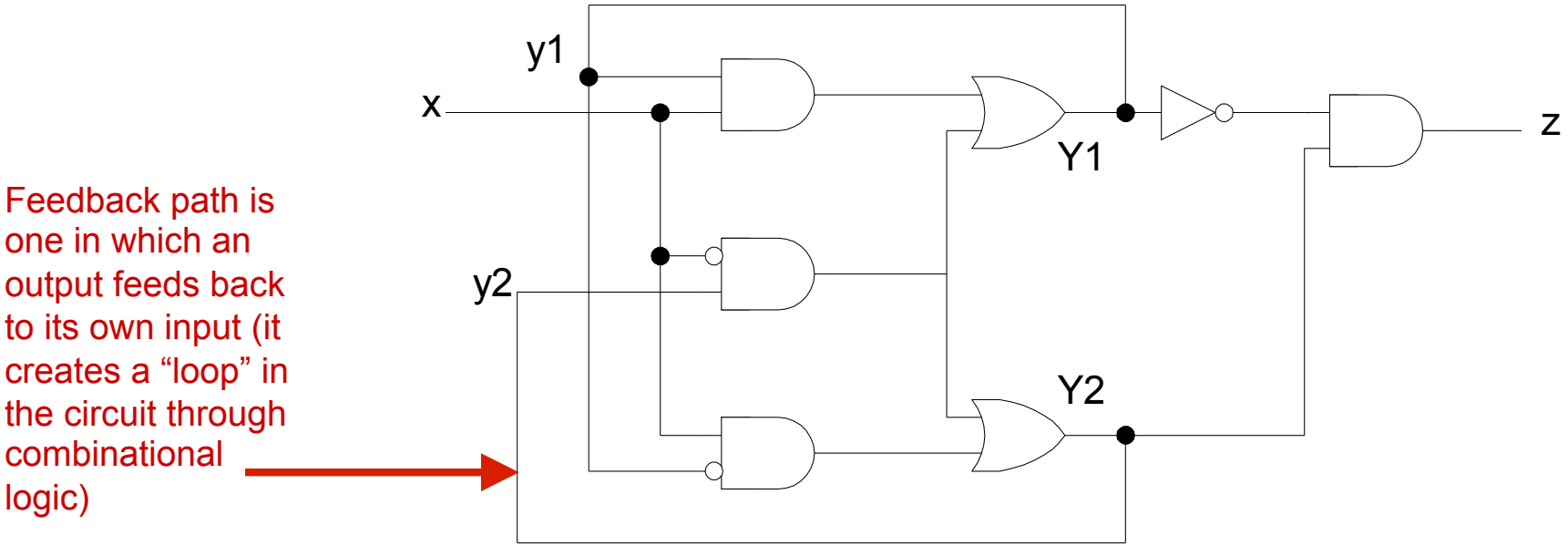
- ***Stability***:
  For a given set of inputs (i.e., values), the system is **stable** if the circuit eventually reaches **steady state** and the current and next state are equal and unchanging (secondary variables y = excitation variables Y), otherwise the circuit is **unstable**.

- ***Fundamental Mode Restriction***:
  A circuit is operating in fundamental mode if we assume/force the following restrictions on how the inputs can change:
  - Only **one** input is allowed to change at a time,
  - The input changes **only** after the circuit is **stable**.

Analysis of asynchronous circuits

Example of an asynchronous circuit

- Consider the following circuit that has combinatorial feedback paths (and is therefore identified as asynchronous). No apparent latches in the circuit:



- Circuit has one input (x), one output (z), two current state variables (y1, y2) and two next state variables (Y1, Y2).

Analysis (writing logic equations)

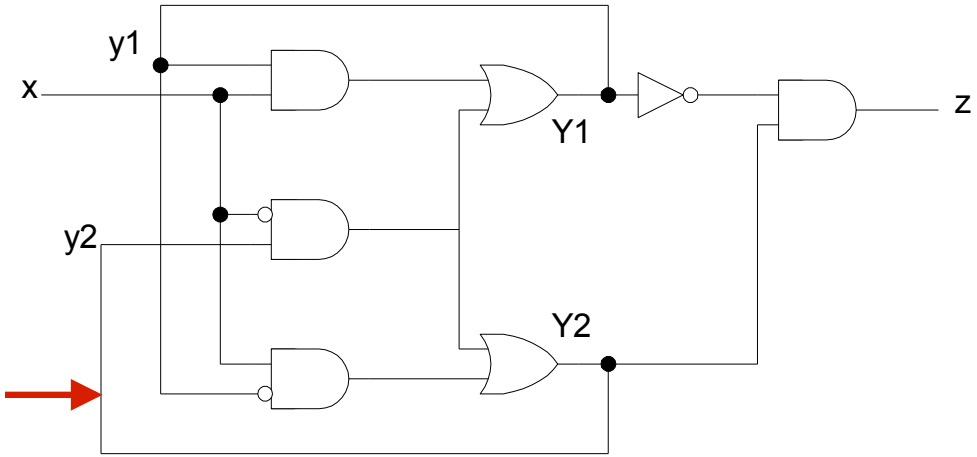
- Write logic equations for the next state in terms of the circuit inputs and current state:

$$Y_1 = xy_1 + \overline{x}y_2$$

$$Y_2 = \overline{x}y_2 + x\overline{y}_1$$

- Write logic equations for circuit outputs in terms of the circuit inputs and current state:

$$z = \overline{y}_1 y_2$$



Analysis (transition table)

- Using these equations, we can write a **transition table** that shows next state (Y1, Y2) and outputs (z) as a function of inputs (x) and current state (y1, y2):

curr state	next state		output	
	x=0	x=1	x=0	x=1
y2y1	Y2Y1	Y2Y1	z	z
00	00	10	0	0
01	00	01	0	0
10	11	10	1	1
11	11	01	0	0

- **Note that stable states (current state "y" equal to next state "Y") are circled.**

Analysis (flow table)

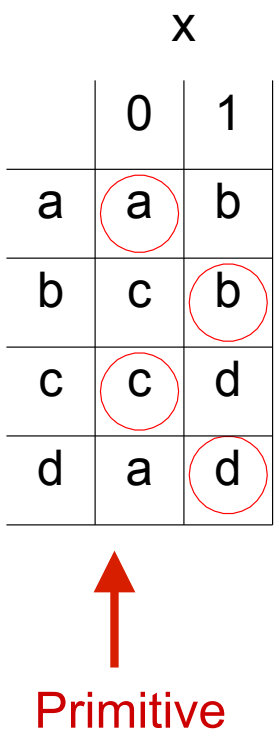
- We can also create a **flow table**, which is just the transition table with binary numbers replaced with symbols (e.g., let **a = 00, b = 01, c = 10** and **d = 11**):

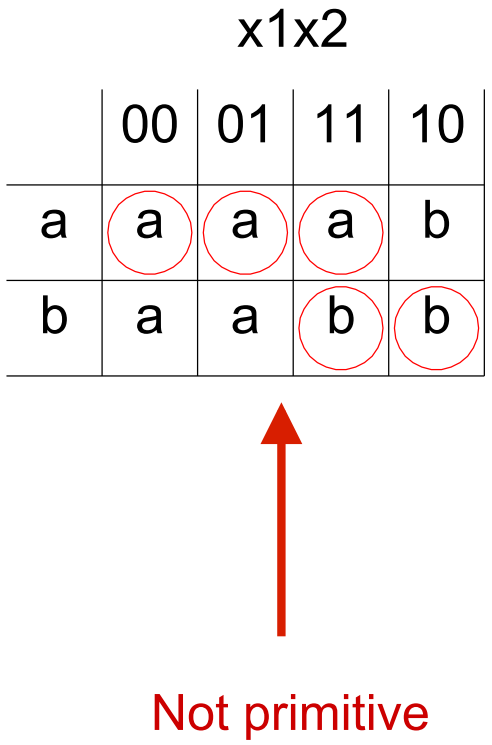
curr state	next state		output	
	x=0	x=1	x=0	x=1
y2y1	Y2Y1	Y2Y1	z	z
a	a	c	0	0
b	a	b	0	0
c	d	c	1	1
d	d	b	0	0

- We could proceed to draw something like a state diagram from this information, if we choose...

Primitive flow table

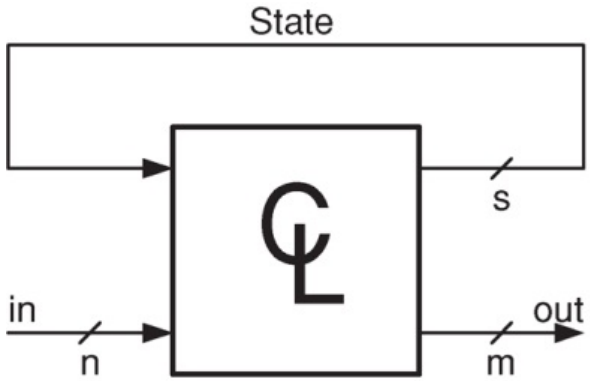
- Flow table with **only one stable state per row is called a primitive flow table**.



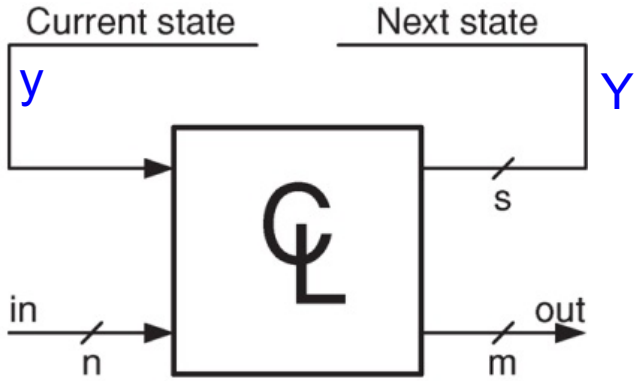


Flow-Table analysis

- Break the feedback path (fig. b)
- Write the equations of the next state (Y) as a function of current state (y) and inputs
- Make a flow table
- When multiple bits of the state are changing at the same time (at different order) it can cause a condition called **race.** The circuit may not reach a steady state and may oscillate indefinitely.

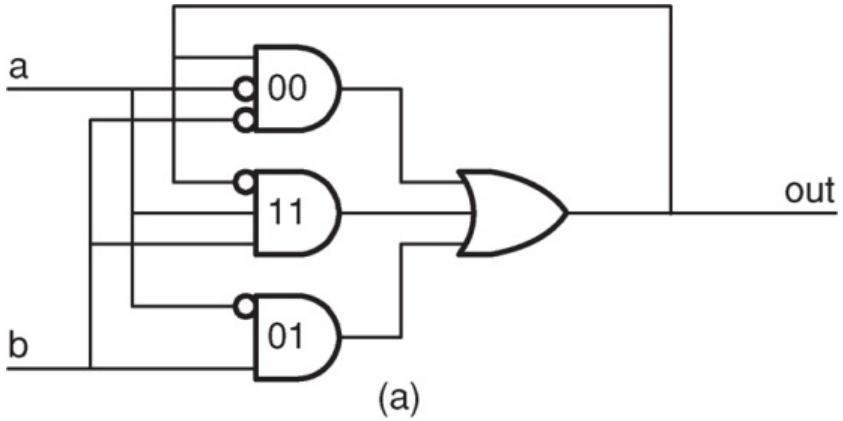


(a)



(b)

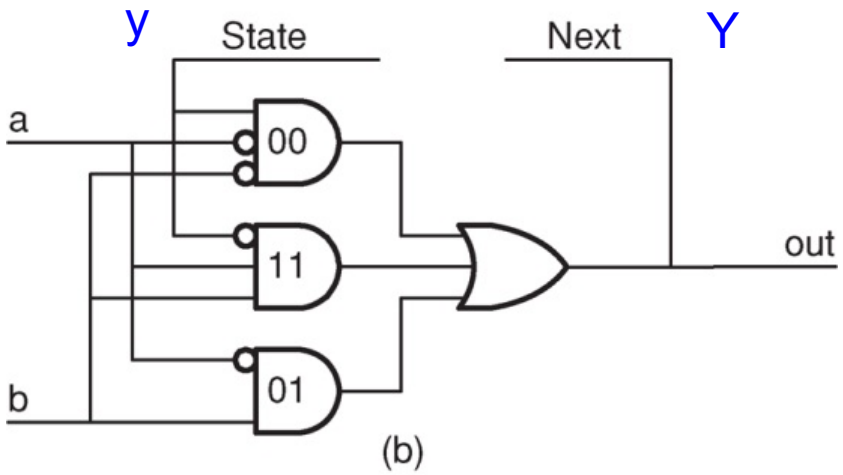
Another analysis example



A circle with transient states would cause an oscillation.

Hardly ever desired
e.g. ab=11

Break the feedback loop



Make the flow table

State	Next			
	00	01	11	10
0	(0)	1	1	(0)
1	(1)	(1)	0	0

(c)

When next state is the same as the current state it is a **stable state** and is circled
Otherwise it is an unstable or **transient state**

Summary of analysis

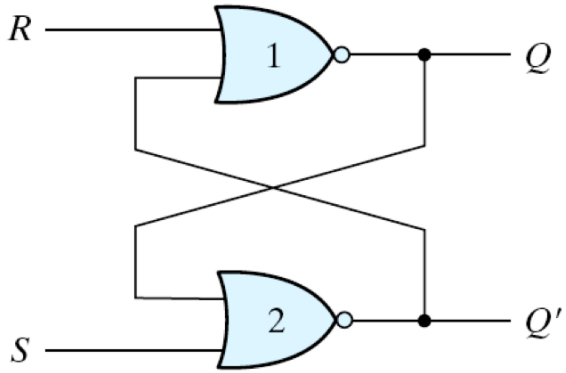
- Procedure to determine transition table and/or flow table from a circuit with combinatorial feedback paths:
  - Identify **feedback paths**.
  - Label Y (next state variables) at output and y (current state variables at input).
  - Derive logic expressions for Y (next state variables) in terms of circuit inputs and current state variables. Do the same for circuit outputs.
  - Create a transition table and flow table.
  - Circle **stable states** where Y (next state variables) are equal to y (current variables).

Revising latches

- Latches are simply asynchronous circuits. We can use the previous analysis technique to see how latches work.

Analysis of an SR latch (1)

- We can analyze an SR latch using the previous technique



(a) Cross-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	

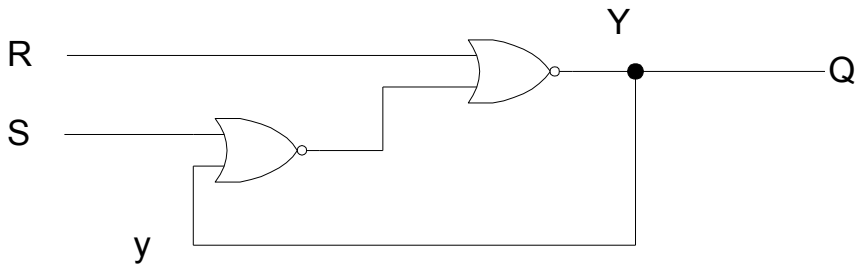
(b) Truth table

- Equations derived for secondary variable (same equation for output):

$$Y = \overline{R + \overline{(S + y)}} = S\overline{R} + \overline{R}y$$

- Since we want to avoid the SR=11 situation, we can write:

$$Y = S + \overline{R}y \quad \text{if} \quad S*R = 0$$

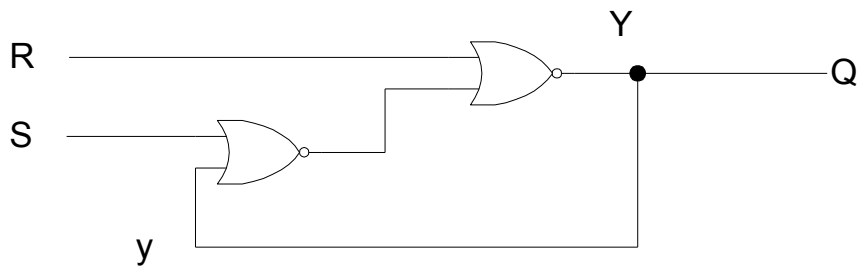


Analysis of an SR latch (2)

- Can derive the transition table and the flow table:

curr state	next state SR=00	01	11	10	output
y	Y	Y	Y	Y	
0	0	0	0	1	0
1	1	0	0	1	1

curr state	next state SR=00	01	11	10	output
y	Y	Y	Y	Y	
a	a	a	a	b	0
b	b	a	a	b	1



Analysis of an SR latch (3)

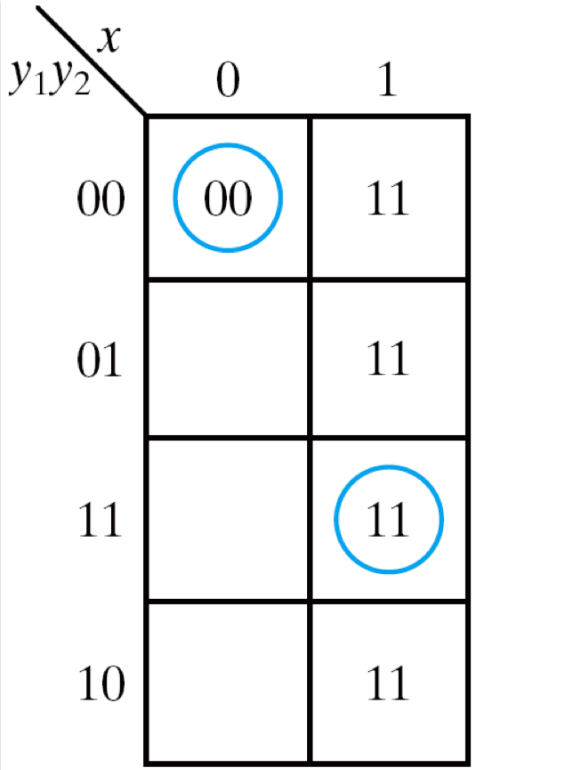
- Note: We can see the undesirable case when SR=11 and inputs change.
- Depending on the various delays and assuming SR=11 changes to SR=00...
  - If SR=11 -> SR=10 -> SR=00, we get stable state with output of 1.
  - If SR=11 -> SR=01 -> SR=00, we get stable state with output of 0.
- So the stable state is not predictable.
- Conclusion is that we need to be careful if we (possibly) need to transition from one state to another and we (somehow) pass through state 11.

Race conditions &
State assignment

Race conditions

- when two or more binary state variables change value
- $00 \rightarrow 11$
  - $00 \rightarrow 10 \rightarrow 11$ or $00 \rightarrow 01 \rightarrow 11$
- a non-critical race
  - if they reach the same final state
  - otherwise, a critical state

Non-critical race

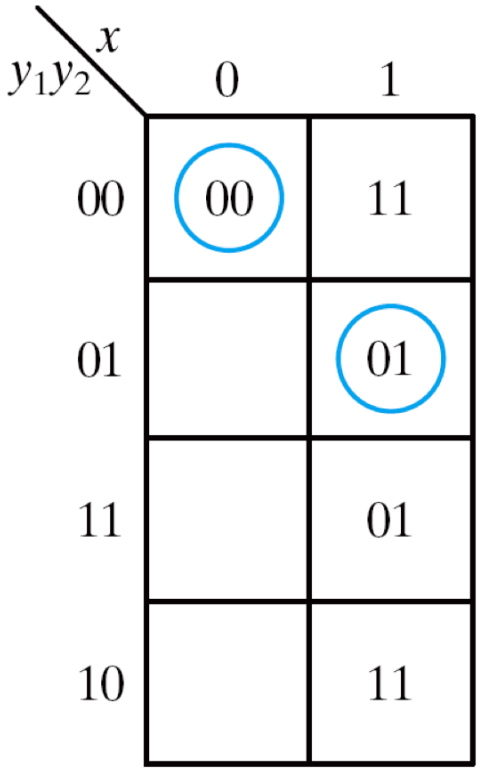


(a) Possible transitions:

$00 \rightarrow 11$

$00 \rightarrow 01 \rightarrow 11$

$00 \rightarrow 10 \rightarrow 11$



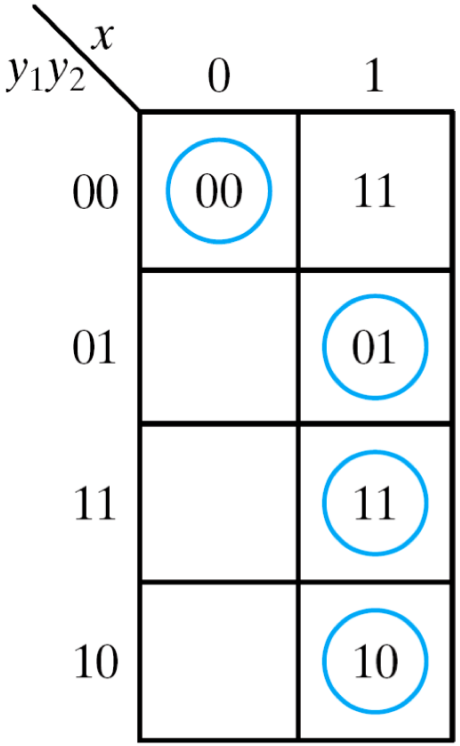
(b) Possible transitions:

$00 \rightarrow 11 \rightarrow 01$

$00 \rightarrow 01$

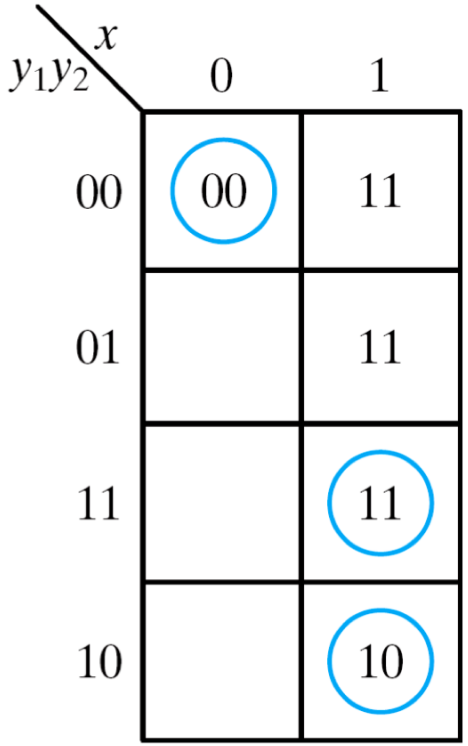
$00 \rightarrow 10 \rightarrow 11 \rightarrow 01$

Critical race



(a) Possible transitions:

$00 \rightarrow 11$
$00 \rightarrow 01$
$00 \rightarrow 10$



(b) Possible transitions:

$00 \rightarrow 11$
$00 \rightarrow 01 \rightarrow 11$
$00 \rightarrow 10$

Avoiding race conditions

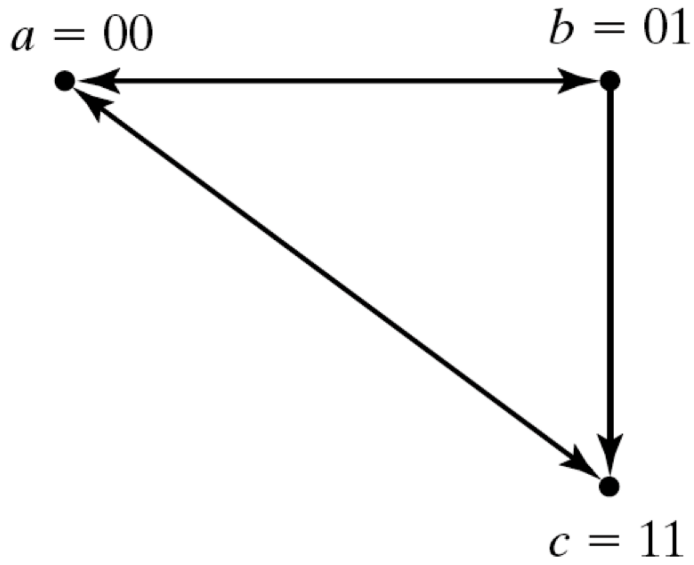
- Races may be avoided
  - race-free state-assignment: only one state-bit can change at any one time
  - insert intermediate unstable states with a unique state-variable change

Race-Free State Assignment

- To avoid critical races
  - only one variable changes at any given time
- Three-row flow-table example
  - flow-table and transition diagram example

	$x_1 x_2$ = 00	01	11	10
a	a	b	c	a
b	a	b	b	c
c	a	c	c	c

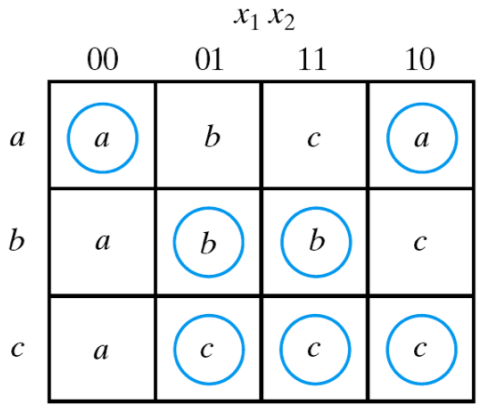
(a) Flow table



(b) Transition diagram

Race-Free State Assignment

- an extra row is added
- no stable state in row d

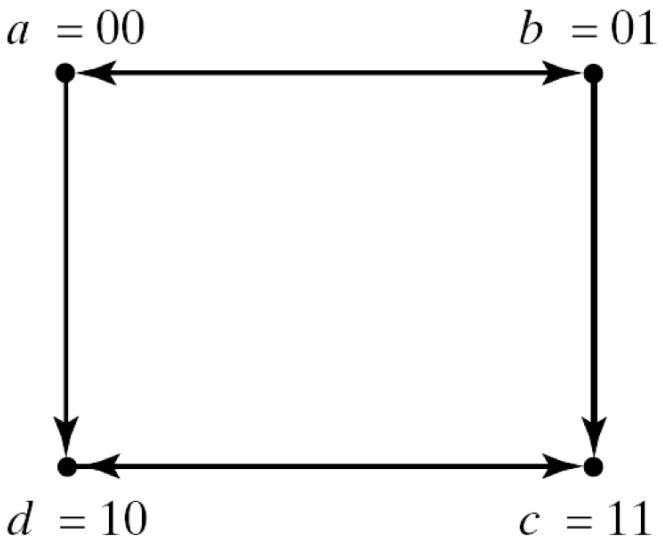


	x_1x_2 = 00	01	11	10
a	a	b	c	a
b	a	b	b	c
c	a	c	c	c

(a) Flow table

	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	–

(a) Flow table



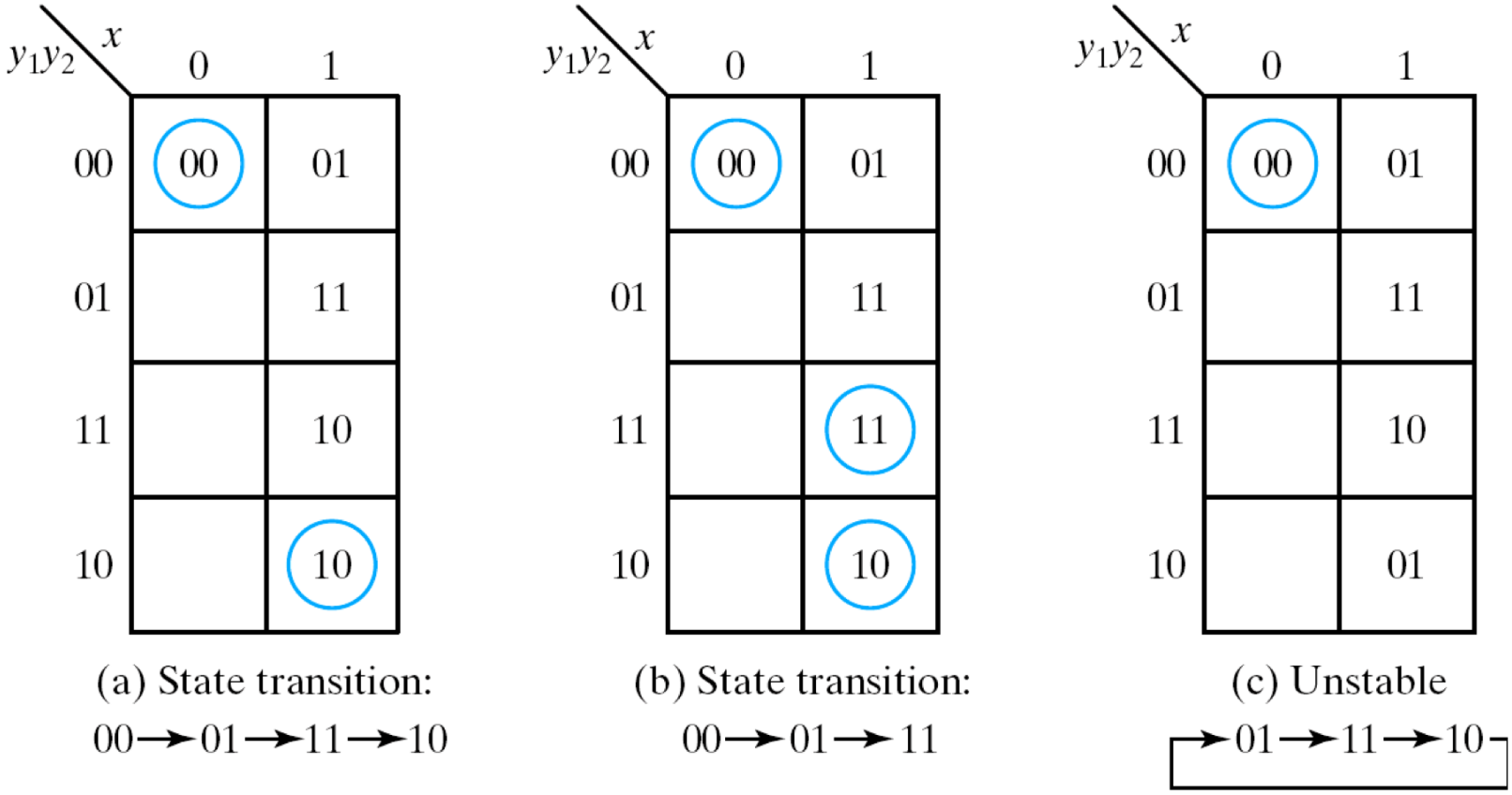
(b) Transition diagram

Transition Table

	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	–

Cycles

A cycle is a unique sequence of unstable states

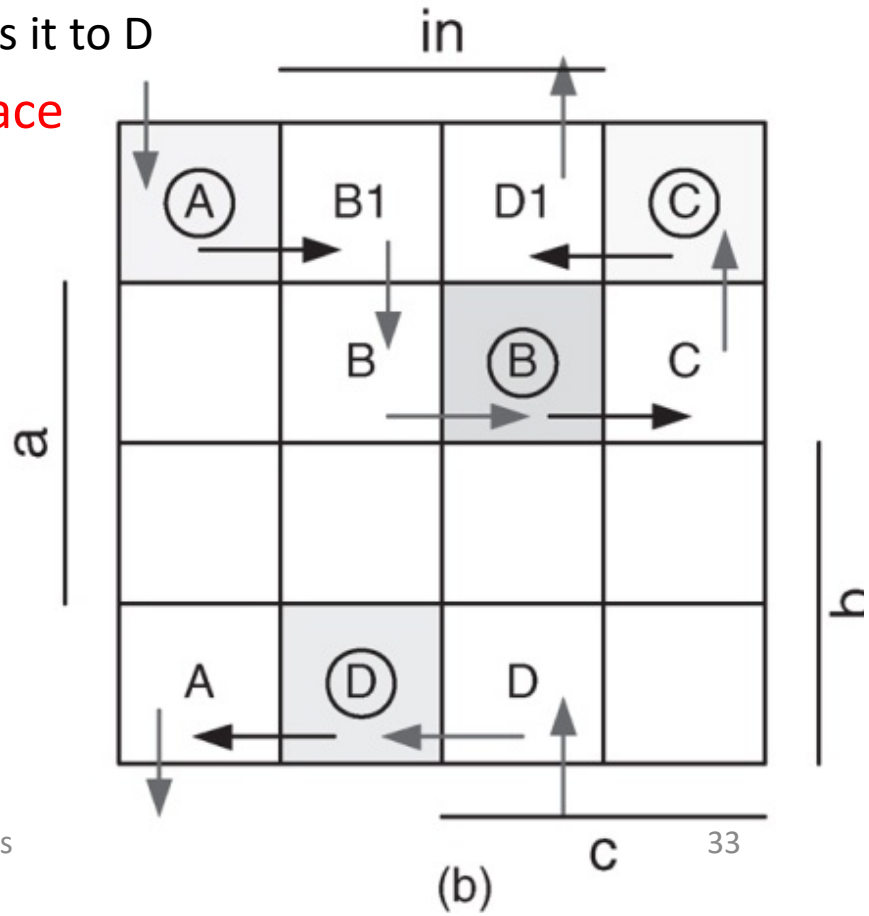


Race and state assignment

- Outputs a, b can serve as the two LSbits of the state encoding
- An extra state variable c needed to distinguish between A and C states
- Transition from A (cab=000) to B (110) has 2 state bits changing
  - If a changes first go to 010 and then to 110
  - If c changes first go to C (100) which then takes it to D
- Racing affects the end state so it is a critical race

State	Code (c,a,b)	Next (in) 0	Next (in) 1	Out (a,b)
A	000	(A)	B	00
B	110	C	(B)	10
C	100	(C)	D	00
D	001	A	(D)	01

(a)

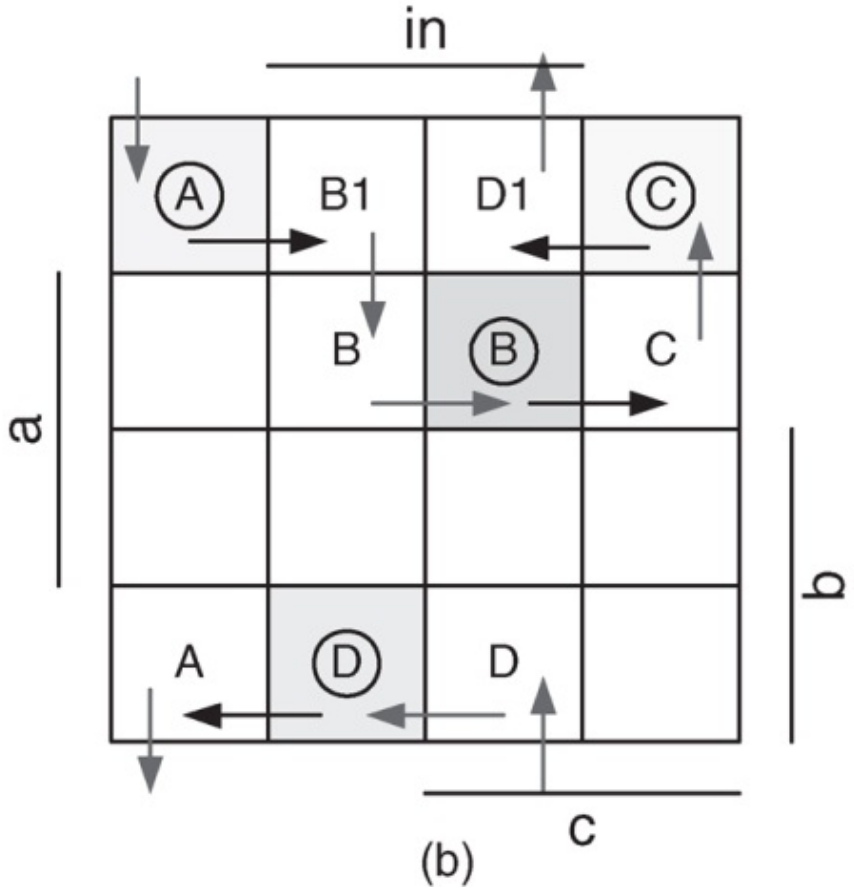


(b)

Race and state assignment

State	Code (c,a,b)	Next (in) 0	Next (in) 1	Out (a,b)
A	000	(A)	B	00
B	110	C	(B)	10
C	100	(C)	D	00
D	001	A	(D)	01

(a)



(b)

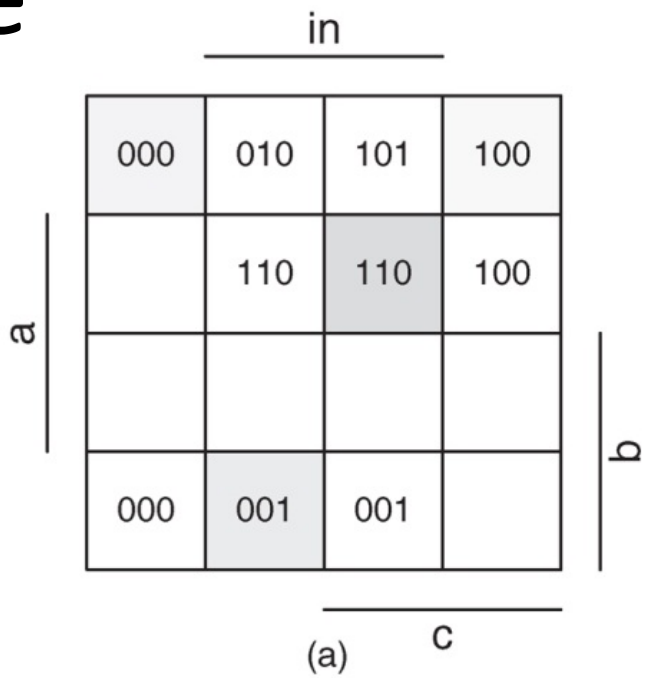
- Introduce B1 (010), when input changes to 1: A->B1, B1->B
- Transient state also required for C (100) to D (001)
- Introduce D1 (101)

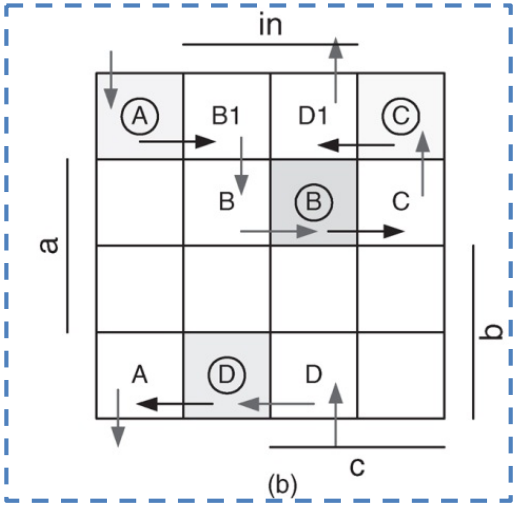
Race and state assignment

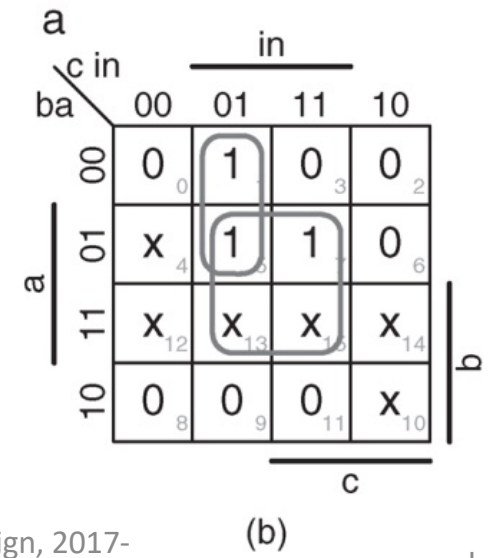
a = in*b'*c' + in*a

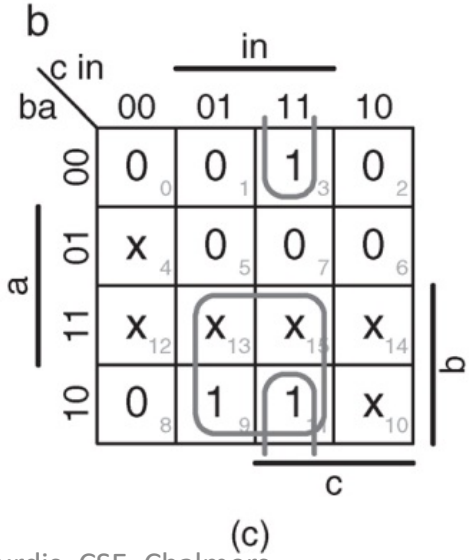
b = in*a'*c + in*b

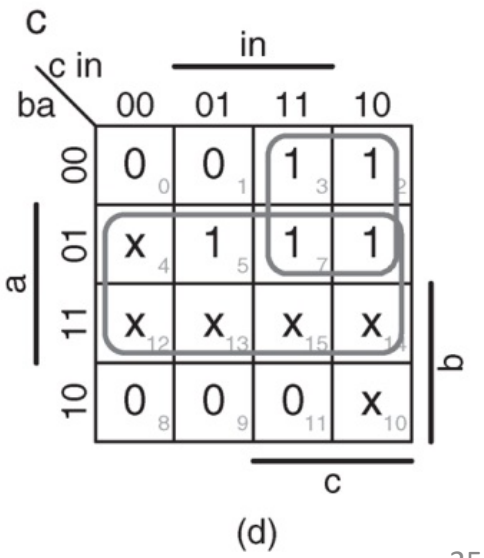
c = a + b'*c











Quiz 17-1

http://m.socrative.com/student/#joinRoom

room number: 713113

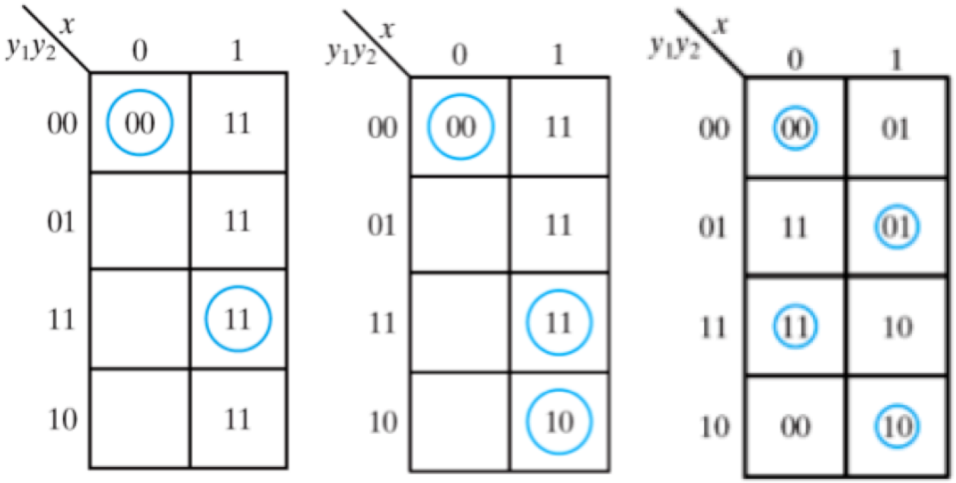
- Q1: Find the stable states in the following transition table:

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

(b) Transition table

- Q2: What is the output of the circuit for x_1x_2=11
  1. Logic 1
  2. Logic 0
  3. Consecutive logic 1s and 0s

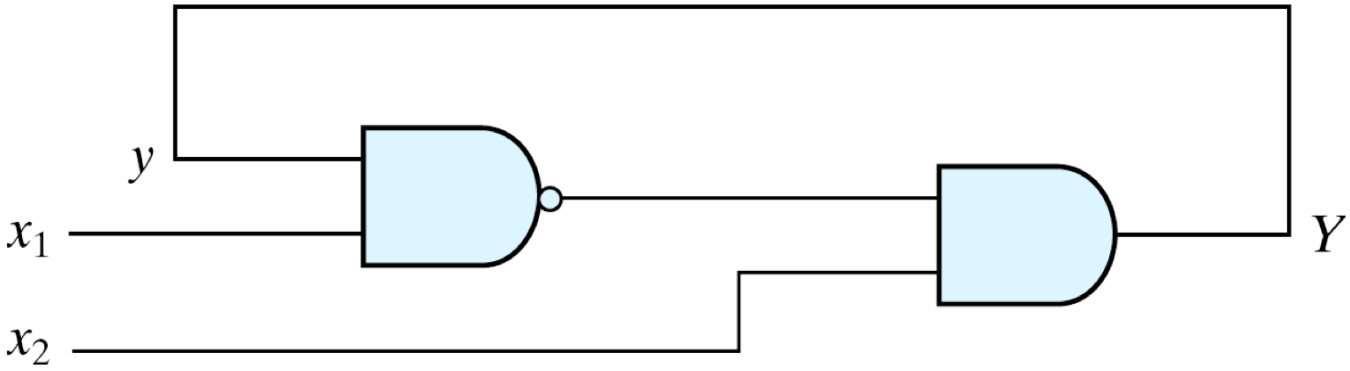
- Q3: Are there race conditions in the cases below? If not put (a), if there are critical ones put (b) if there are only non-critical put (c)



(i) (ii) (iii)

Stability Considerations

a square-wave generator (x_1x_2=11)?



(a) Logic diagram

y \ x_1x_2	00	01	11	10
0	(0)	1	1	(0)
1	0	(1)	0	0

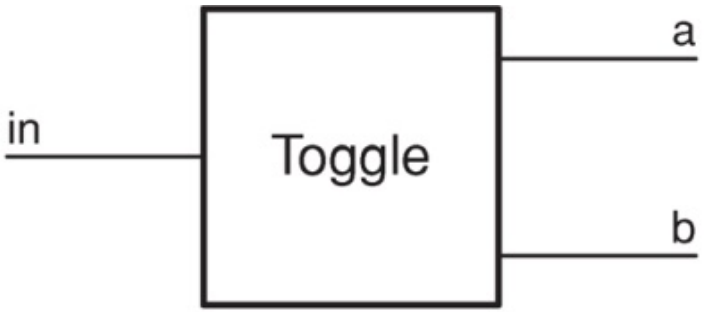
(b) Transition table

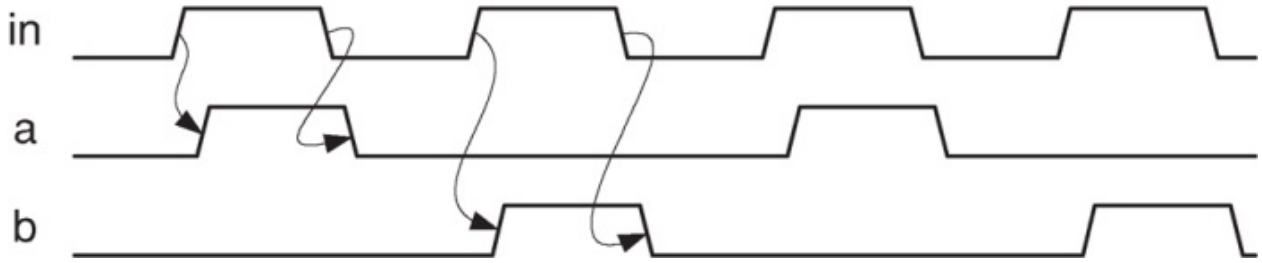
Example of an unstable circuit

Synthesis of Asynchronous circuits

Flow-Table Synthesis

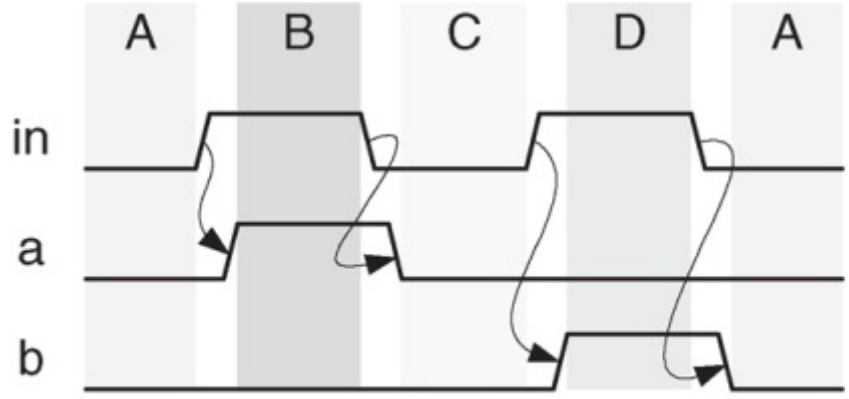
- Create a flow table from specifications of a circuit
- Use flow table to synthesize the circuit that realizes the specification





Flow-Table Synthesis

- Partition the waveform into potential states
- Write down the flow table



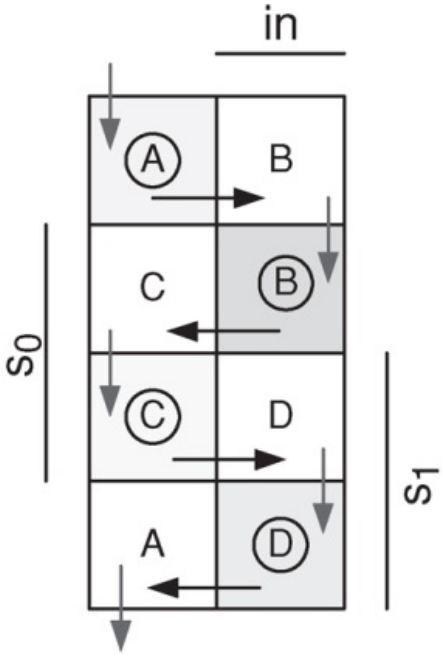
State	Next (in) 0	Next (in) 1	Out (a,b)
A	(A)	B	00
B	C	(B)	10
C	(C)	D	00
D	A	(D)	01

Flow-Table Synthesis

- State assignment: Assign binary codes to each state
- If two states differ in more than one bit, a transition between them needs to go through a transient state with one bit change
- Careful for races between two state bits

State	Code	Next (in) 0	Next (in) 1	Out (a,b)
A	00	(A)	B	00
B	01	C	(B)	10
C	11	(C)	D	00
D	10	A	(D)	01

(a)



(b)

Redraw flow table as Karnaugh map showing the state transitions (trajectory map)

Flow-Table Synthesis

- Redraw Karnaugh map replacing state symbols with their binary codes
- Make separate maps for each next state variable (S0, S1)
- From Karnaugh derive equations:

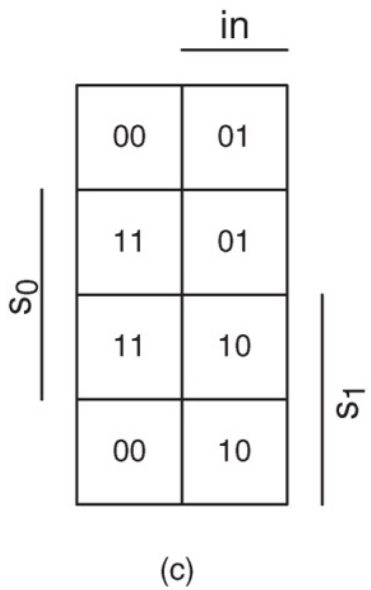
  S0 = s1'*in + s0*in' + s0*s1'

  S1 = s1*in + s0*in' + s0*s1
- Derive output equations:

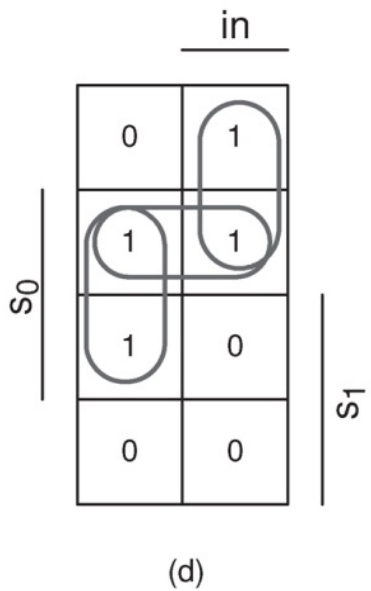
  a = s1'*s0

  b=s1*s0'

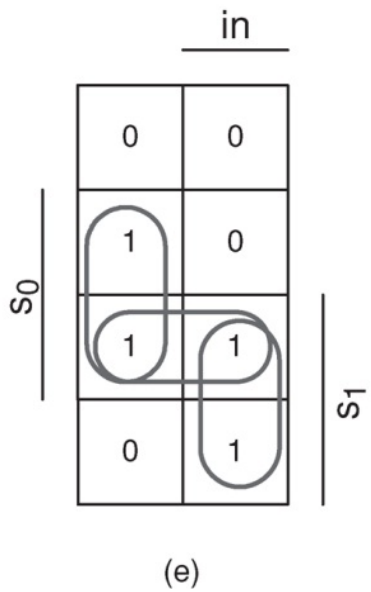
Last implicants are put to avoid hazards



(c)



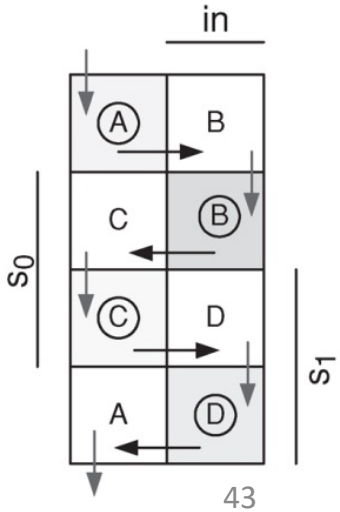
(d)



(e)

State	Code	Next (in) 0	Next (in) 1	Out (a,b)
A	00	(A)	B	00
B	01	C	(B)	10
C	11	(C)	D	00
D	10	A	(D)	01

(a)



(b)

State Minimization

State minimization in asynchronous circuits

- Similar to the minimization we did with synchronous sequential circuits.
- Lots of opportunity for state minimization in asynchronous flow tables:
  - Lots of don't care outputs for unstable states (since we won't stay in unstable states too long).
  - Don't care next state information if we assume fundamental mode operation (some transitions will not occur).

Compatible states

- With don't cares, **equivalency** is replaced with **compatibility**.

- Two states A and B are **compatible** if, for every input combination we find:
  - A and B produce the same outputs **wherever specified**, AND
  - A and B have compatible next states **wherever specified**.

- **Don't cares match with anything...**

Implication chart using compatible states (1)

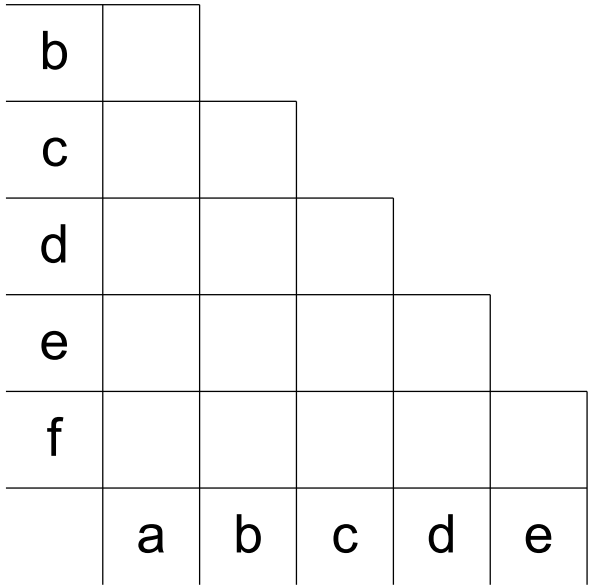
- Consider the following flow table with some unspecified next states and outputs (two inputs, one output, 6 states):

curr state	inputs (DG)				output			
	00	01	11	10	00	01	11	10
a	c	a	b	-	-	0	-	-
b	-	a	b	e	-	-	1	-
c	c	a	-	d	0	-	-	-
d	c	-	b	d	-	-	-	0
e	f	-	b	e	-	-	-	1
f	f	-	-	e	1	-	-	-

Implication chart using compatible states (2)

- Build the implication chart (list states along left and bottom side – like lower triangle of a matrix):

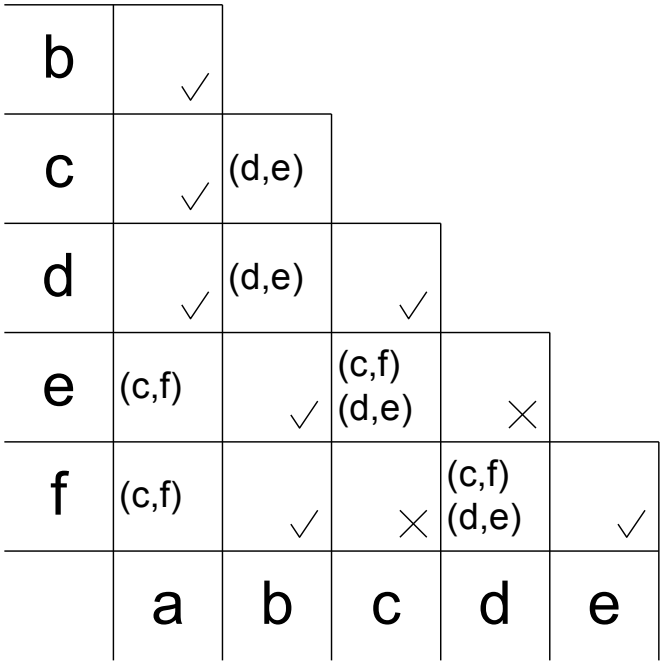
curr state	inputs (DG)				output			
	00	01	11	10	00	01	11	10
a	c	a	b	-	-	0	-	-
b	-	a	b	e	-	-	1	-
c	c	a	-	d	0	-	-	-
d	c	-	b	d	-	-	-	0
e	f	-	b	e	-	-	-	1
f	f	-	-	e	1	-	-	-



Implication chart using compatible states (3)

- Mark states incompatible due to different outputs with ”x” .
- Marking definitely compatible states with “v”.
- Marking possibly compatible states with implied decisions.

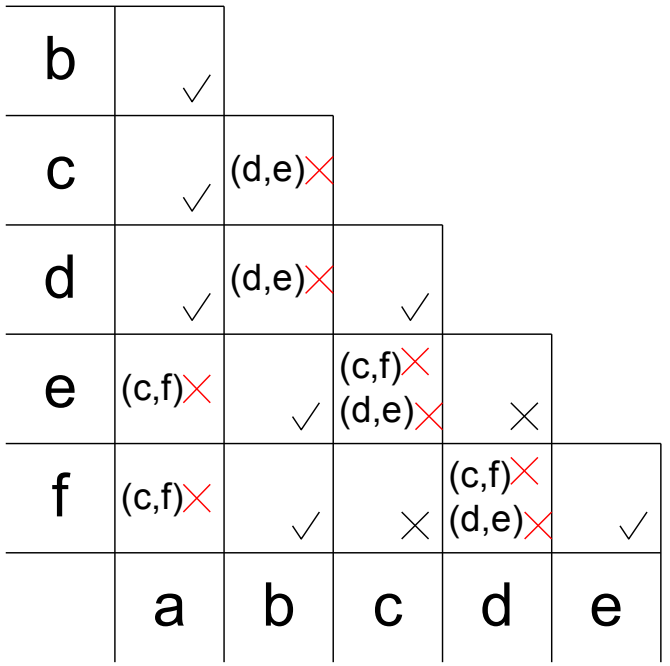
curr state	inputs (DG)				output			
	00	01	11	10	00	01	11	10
a	c	a	b	-	-	0	-	-
b	-	a	b	e	-	-	1	-
c	c	a	-	d	0	-	-	-
d	c	-	b	d	-	-	-	0
e	f	-	b	e	-	-	-	1
f	f	-	-	e	1	-	-	-



Implication chart using compatible states (4)

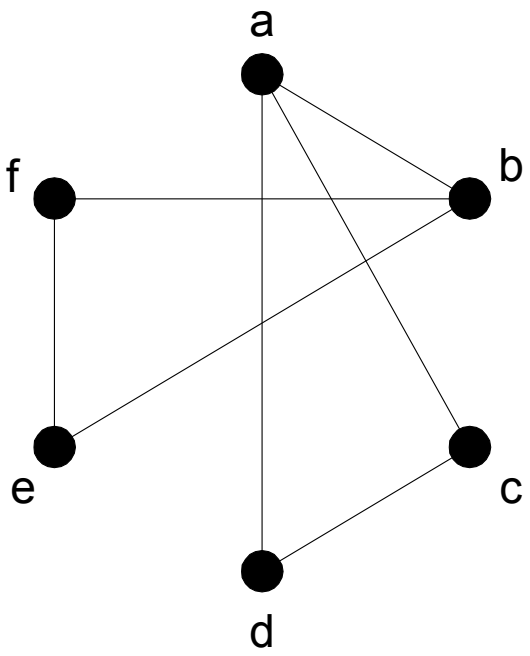
- Scan columns again and again, checking implied decisions to remove compatibilities...

curr state	inputs (DG)				output			
	00	01	11	10	00	01	11	10
a	c	a	b	-	-	0	-	-
b	-	a	b	e	-	-	1	-
c	c	a	-	d	0	-	-	-
d	c	-	b	d	-	-	-	0
e	f	-	b	e	-	-	-	1
f	f	-	-	e	1	-	-	-



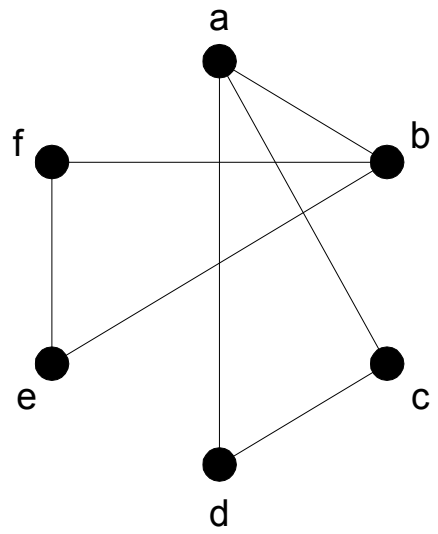
Merger diagram with compatible states (1)

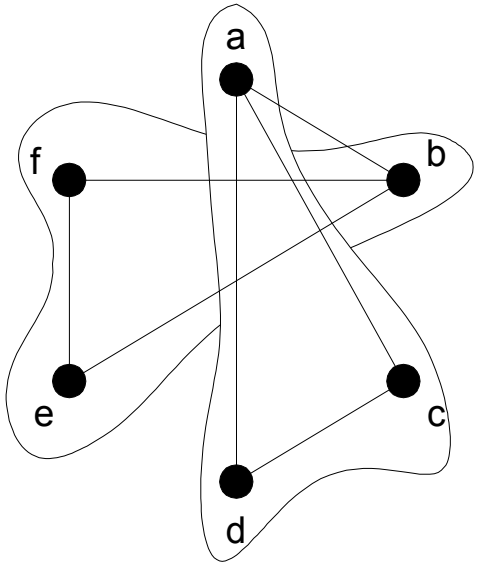
- Squares with any x are not compatible; those with all “v” are compatible (possibly under implications).
- Draw Merger Diagram:



Merger diagram with compatible states (2)

- We now look for large **cliques** in the graph (clique is part of the graph in which every node is connected to every other node)...





- We can now merge states (a,c,d) and (b,e,f). We have reduced 6 states down to 2 states by merging.

Final Result

curr state	inputs (DG)				output			
	00	01	11	10	00	01	11	10
a	c	a	b	-	-	0	-	-
b	-	a	b	e	-	-	1	-
c	c	a	-	d	0	-	-	-
d	c	-	b	d	-	-	-	0
e	f	-	b	e	-	-	-	1
f	f	-	-	e	1	-	-	-

curr state	inputs (DG)				output			
	00	01	11	10	00	01	11	10
a	a	a	b	b	0	0	-	0
b	b	a	b	b	1	-	1	1

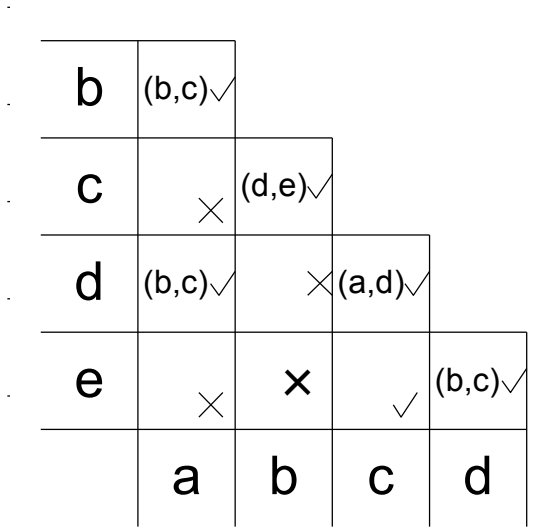
- We can now merge states (a,c,d) and (b,e,f). We have reduced 6 states down to 2 states by merging.
- Note that we still have some unspecified values in the flow table (which is no longer a primitive flow table).

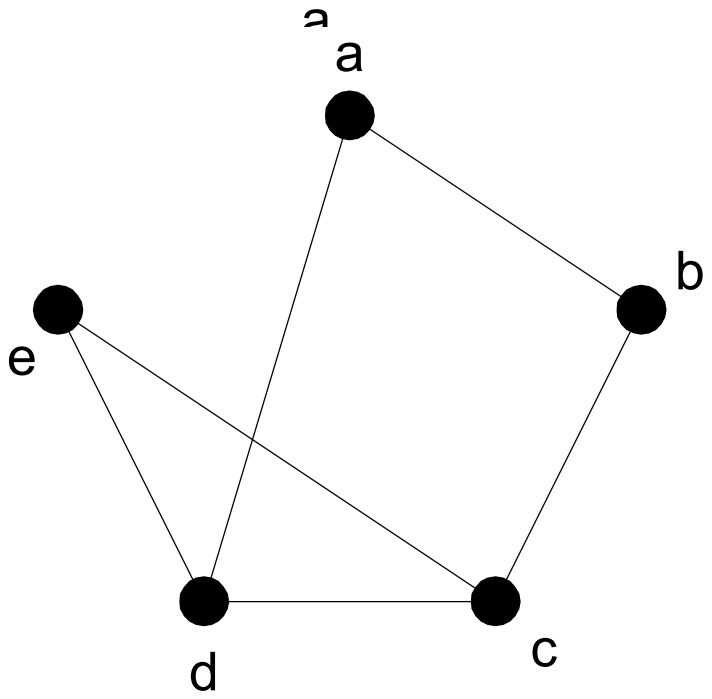
Important!!!

- Reminders:
  - We need to check that each state is included at least once.
  - We need to make sure that any implied compatibilities are true...
- For our solution...
  - (a,c,d) and (b,e,f) all states are included.
  - Implied compatibilities are true. In particular, (a,c,d) and (b,e,f) requires no implied compatibilities.

Revisiting the merger diagram (1)

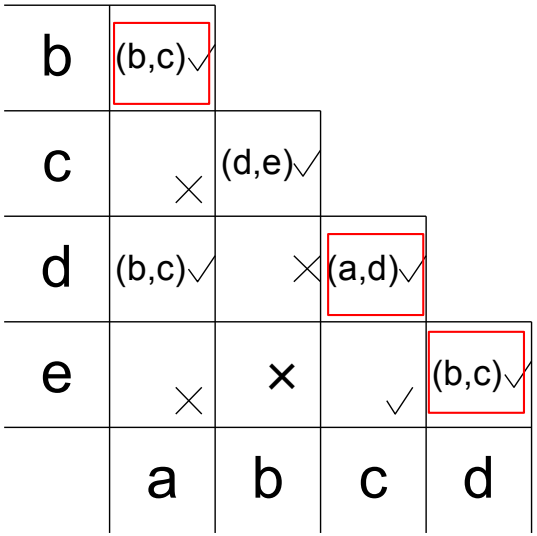
- Useful to illustrate an example where we need to be careful about mergings in the Merger Diagram.
- Consider the following implication chart and its merger diagram:

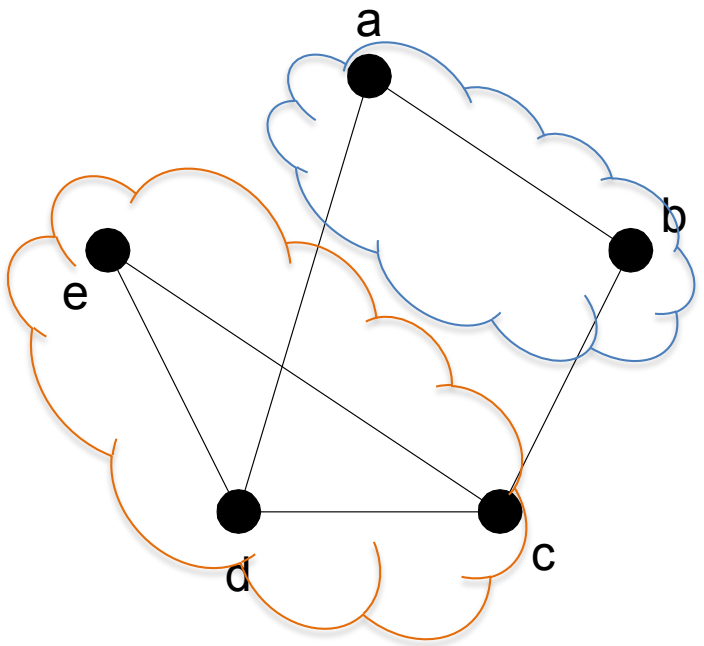




Revisiting the merger diagram (2)

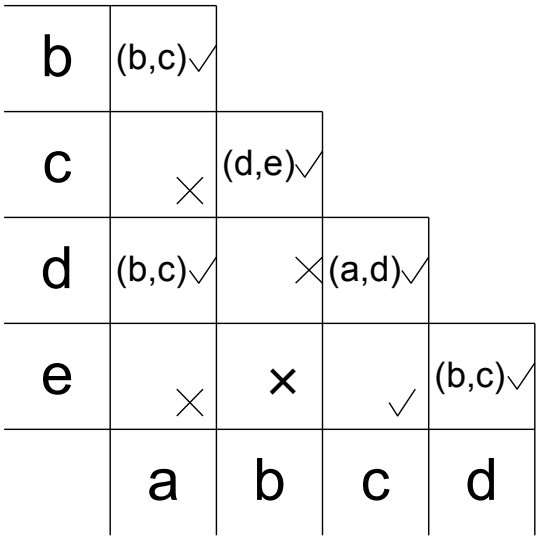
- Say we consider the merging (a,b) and (c,d,e)...
  - For (a,b), (d,e) we require that (b,c) get merged.
  - For (c,d) we require that (a,d) get merged.
- The implied compatibilities do not hold given our selected merging, so our merging is BAD (i.e., **wrong**).

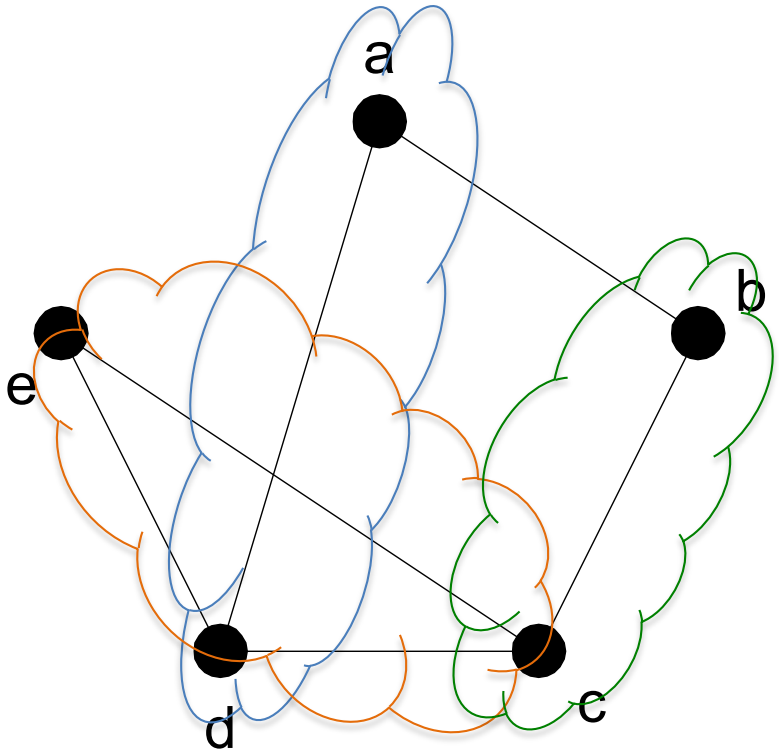




Revisiting the merger diagram (3)

- Say we consider (a,d), (b,c), (c,d,e)...
- The implied compatibilities do hold given our selected merging, so our merging is GOOD (i.e., right).





Asynchronous synthesis example

Asynchronous design example

- Consider a circuit with two inputs, D and G and one output, Q. Output Q follows D with G=1, otherwise Q holds its value.
  - Assume fundamental mode operation – only one input changes at a time.

state	D	G	Q	Condition
a	0	1	0	(follow 0)
b	1	1	1	(follow 1)
c	0	0	0	(hold 0; from a or d)
d	1	0	0	(hold 0)
e	0	0	1	(hold 1; from b or f)
f	1	0	1	(hold 1)

Asynchronous design example (primitive flow table)

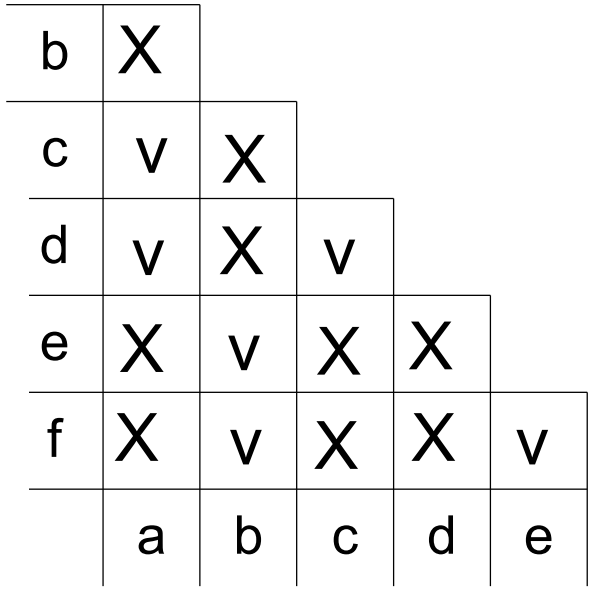
- Note: Outputs depend only on one state (Moore-like):

curr state	next state				output Q
	DG=00	DG=01	DG=10	DG=11	
a	c	(a)	-	b	0
b	-	a	e	(b)	1
c	(c)	a	d	-	0
d	c	-	(d)	b	0
e	f	-	(e)	b	1
f	(f)	a	e	-	1

- Note: Some unspecified entries due to the fundamental mode assumption (e.g., in state a, DG=01, so we never go from DG=01 -> DG=10)...

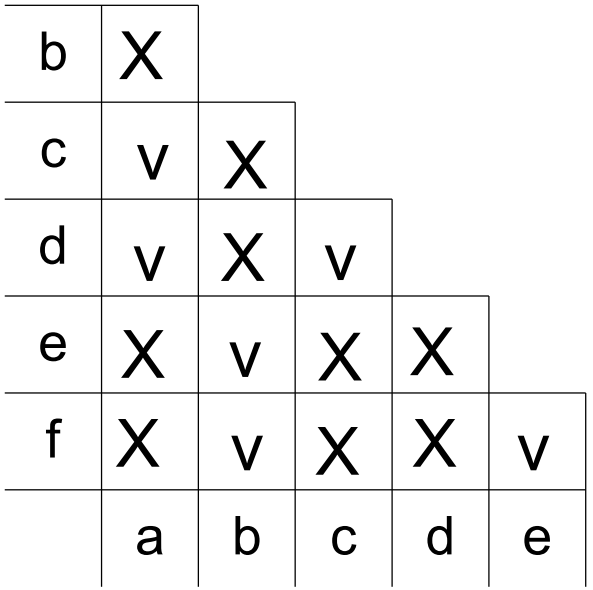
Asynchronous design example (reduced flow table)

curr state	next state				output Q
	DG=00	DG=01	DG=10	DG=11	
a	c	(a)	-	b	0
b	-	a	e	(b)	1
c	(c)	a	d	-	0
d	c	-	(d)	b	0
e	f	-	(e)	b	1
f	(f)	a	e	-	1



Asynchronous design example (reduced flow table)

curr state	next state				output Q
	DG=00	DG=01	DG=10	DG=11	
a	c	(a)	-	b	0
b	-	a	e	(b)	1
c	(c)	a	d	-	0
d	c	-	(d)	b	0
e	f	-	(e)	b	1
f	(f)	a	e	-	1



- Reduced flow table:

curr state	next state				output Q
	DG=00	DG=01	DG=10	DG=11	
a	(a)	(a)	(a)	b	0
b	(b)	a	(b)	(b)	1

a, c, d can be merged
b, e, f, can be merged

Asynchronous design example (state assignment and transition table)

- We only have two states, so we can let a=0, and b=1.
- Our transition table becomes:

curr state (y)	next state (Y)				output Q
	DG=00	DG=01	DG=10	DG=11	
0	0	0	0	1	0
1	1	0	1	1	1

Asynchronous design example (logic equations)

- We can make K-Maps to determine excitation variables (Y) and output (Z) in terms of circuit inputs and secondary variables (y):

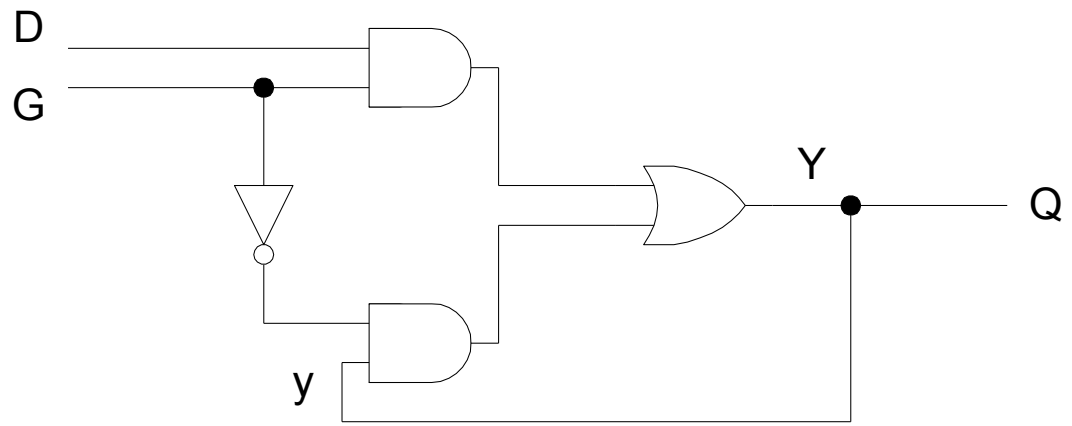
y \ DG	00	01	11	10
0	0	0	1	0
1	1	0	1	1

$$Y = DG + G'y$$

- Output equal to the secondary (state) variable.

Asynchronous design example (circuit)

- Can finally draw the circuit:



$Y = DG + G'y$

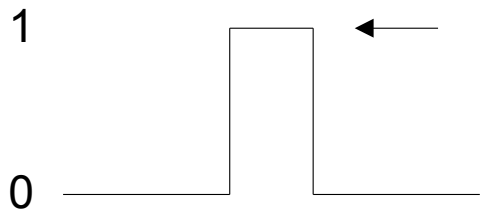
Hazards

Hazards

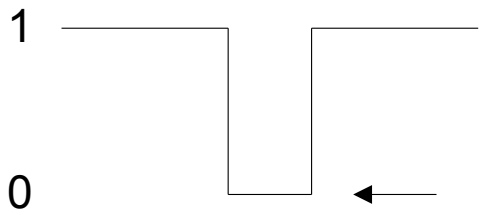
- A **hazard** is a momentary **unwanted** switching transient at a logic function's output (i.e., a glitch).
- Hazards/glitches occur due to unequal propagation delays along different paths in a combinational circuit.
- Can take steps to try and eliminate hazards.
- There are two types of hazards; **static and dynamic**.
- **For asynchronous circuits in particular, hazards can cause problems in addition to other issues like races and non-fundamental mode operation!**
  - ***Momentary false logic function values in an asynchronous circuit can cause a transition to an incorrect stable state!***

Static hazards

- Static-0 Hazard:
  - Occurs when output is 0 and should remain at 0, but temporarily switches to a 1 due to a change in an input.
- Static-1 Hazard:
  - Occurs when output is 1 and should remain at 1, but temporarily switches to a 0 due to a change in an input.



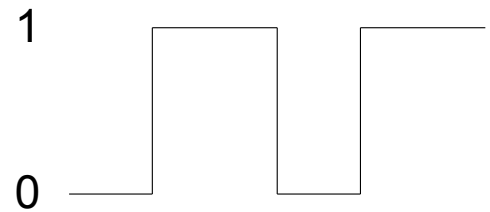
static-0 hazard (0->0)



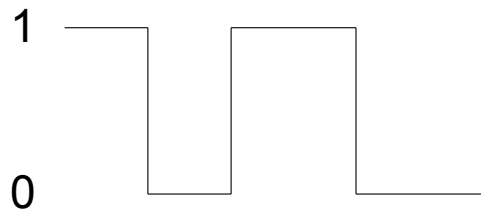
static-1 hazard (1->1)

Dynamic Hazards

- Dynamic Hazard:
  - Occurs when an input changes, and a circuit output should change **0 -> 1 or 1 -> 0,** but temporarily flips between values.



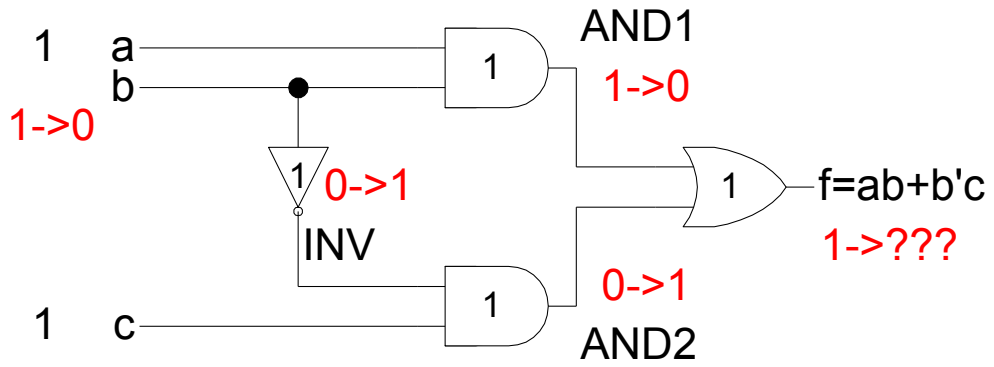
dynamic hazard (0->1)

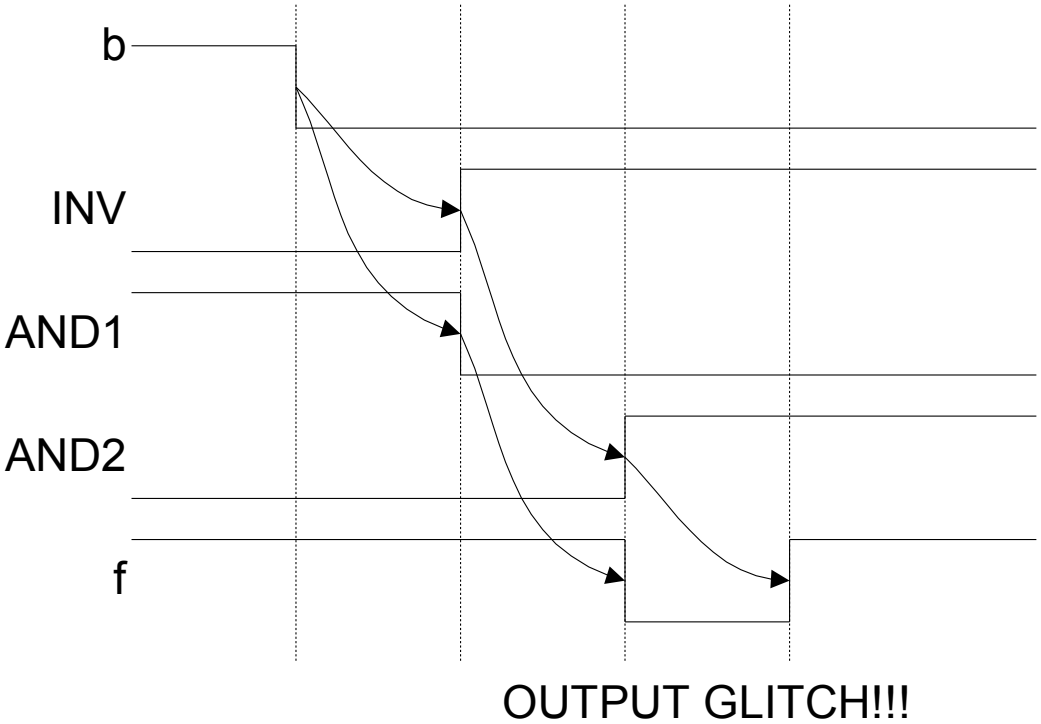


dynamic hazard (1->0)

Illustration

- Consider the following circuit with delays where only one input (input b) changes...
- Draw a timing diagram to see what happens at output with delays.
- From the logic expression, we see that b changing should result in the output remaining at logic level 1...
- Due to delay, the output goes 1->0- >1 and this is an output glitch; **we see a static-1 hazard.**





Fixing hazards (2-level circuits) (1)

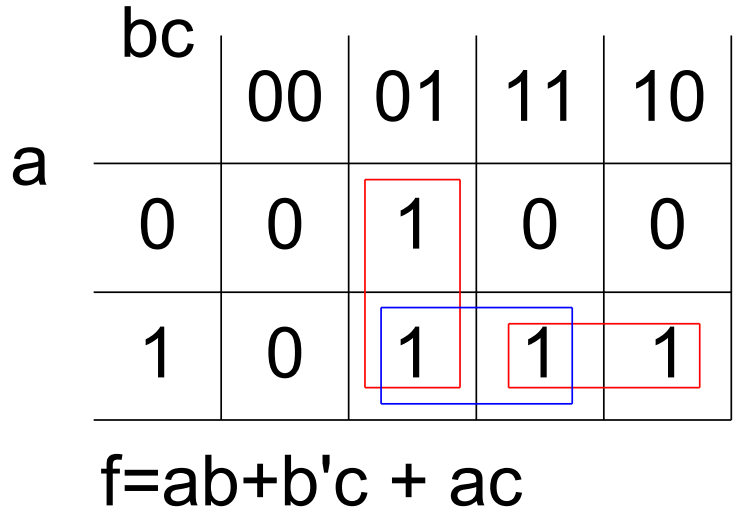
- When circuits are implemented as **2-level SOP (2-level POS)**, we can detect and remove hazards by inspecting the K-Map and ***adding redundant product (sum) terms***.

a \ bc	00	01	11	10
0	0	1	0	0
1	0	1	1	1

f=ab+b'c

- Observe that when input b changes from 1->0 (as in the previous timing diagram), that we "jump" from one product term to another product term.
  - ***If adjacent minterms are not covered by the same product term, then a HAZARD EXISTS!!!***

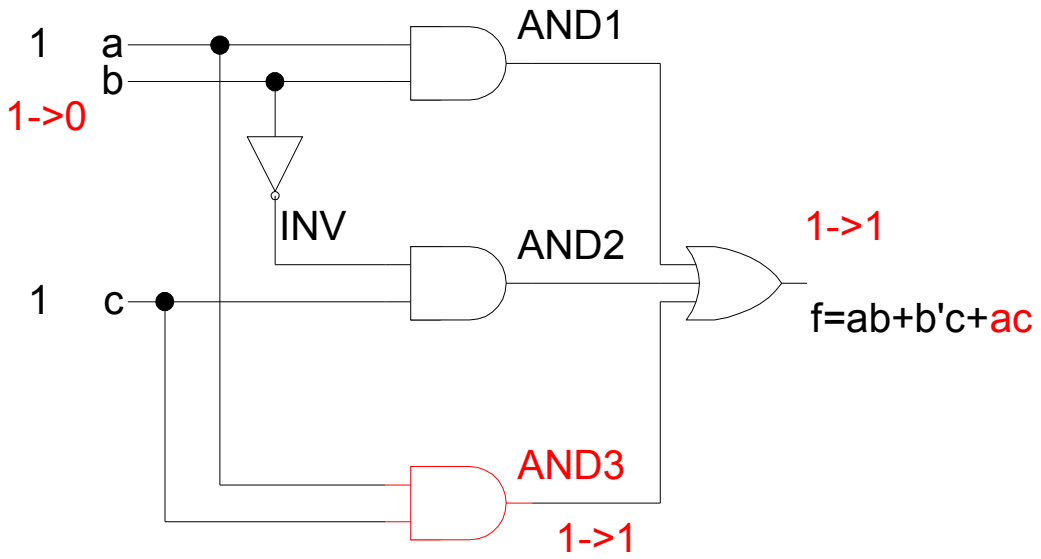
Fixing hazards (2-level circuits) (2)



- The extra product term does not include the changing input variable, and therefore serves to prevent possible momentary output glitches due to this variable.

Fixing hazards (2-level circuits) (3)

- The redundant product term is not influenced by the changing input.



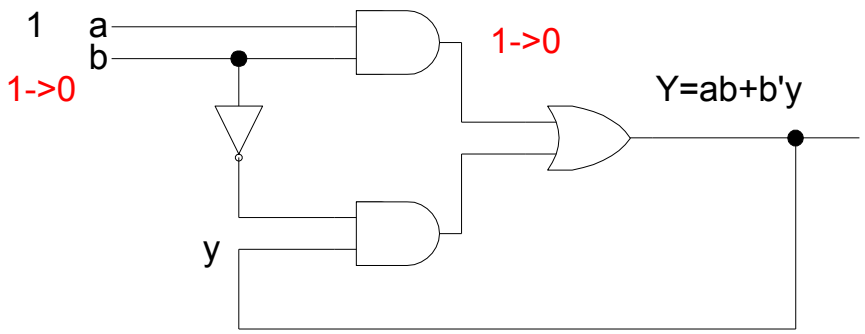
Fixing hazards (2-level circuits) (4)

- For 2-level circuits, if we remove all static-1 hazards using the K-Map (adding redundant product terms), we are guaranteed that there will be no static-0 hazards or dynamic hazards.

- If we work with Product-Of-Sums, we might find static-0 hazards when moving from one sum term to another sum term. We can remove these hazards by adding redundant sum-terms.

Hazards in asynchronous circuits

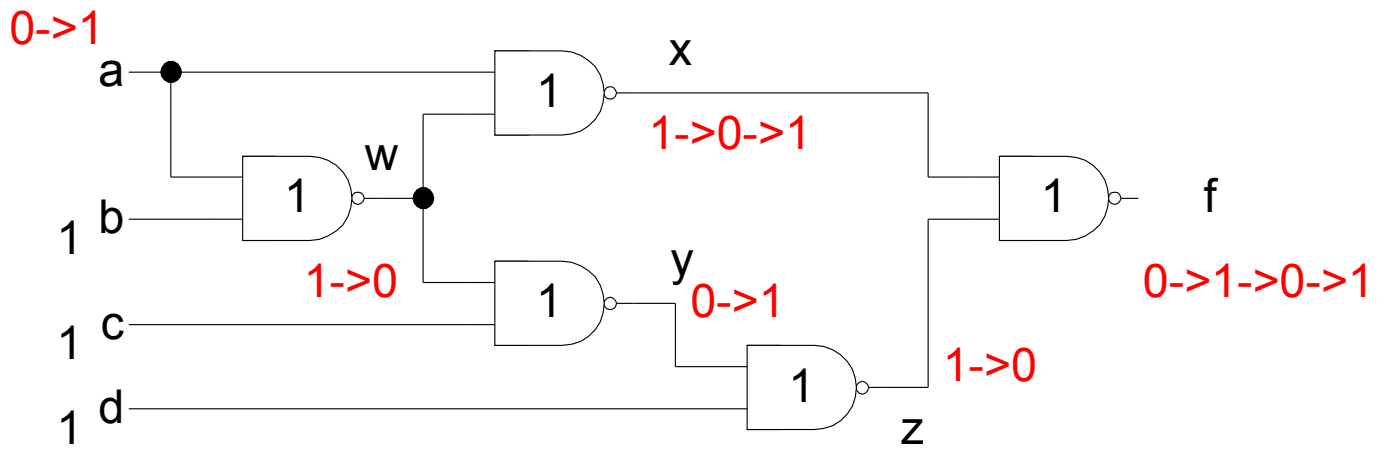
- Consider our first circuit with a hazard, but assume it is not combinatorial, but rather asynchronous.
- We can draw the transition table, and see that there is the potential to end up in an **incorrect** stable state.



curr state	next state				output
	ab=00	01	11	10	
y	Y	Y	Y	Y	
0	0	0	1	0	0
1	1	0	1	1	1

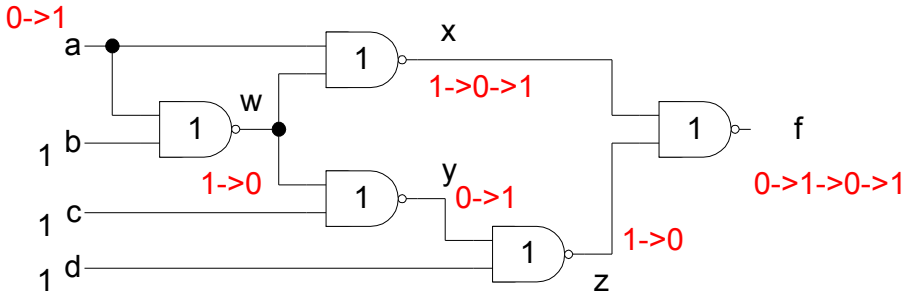
Hazards in multi-level circuits (1)

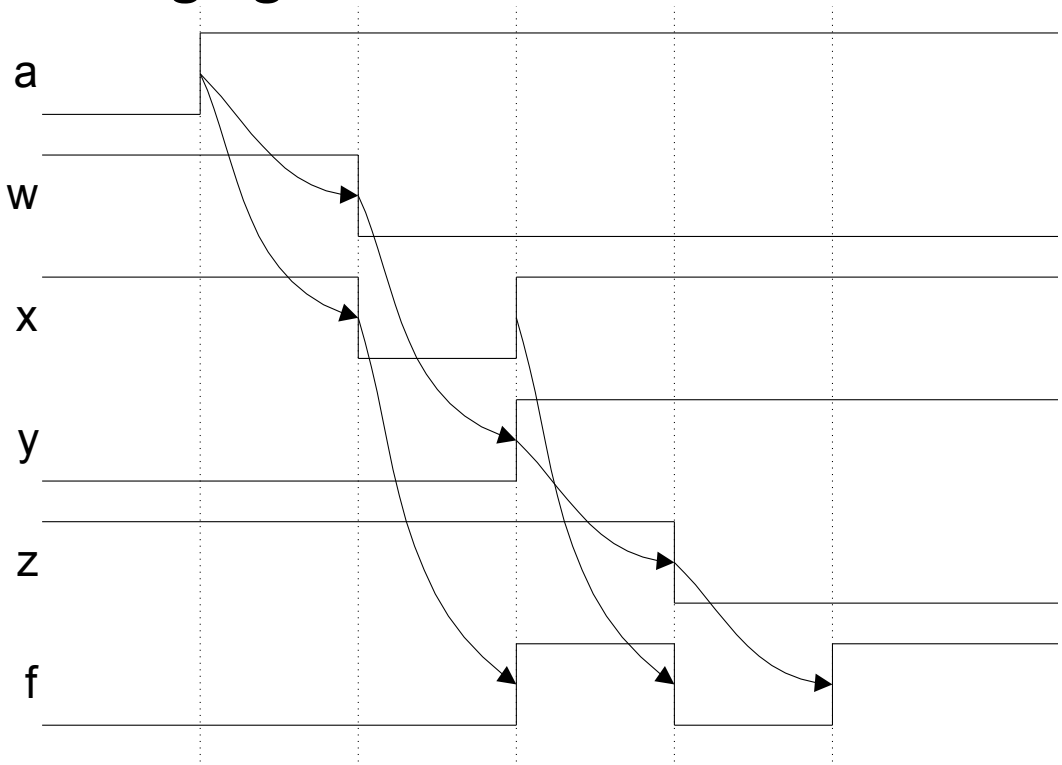
- 2-level circuits are easy to deal with and hazards can be removed…
- The situation is harder with multi-level circuits in which there are multiple paths from an input to an output:



Hazards in multi-level circuits (2)

- Timing diagram shows output changing 0->1->0->1.

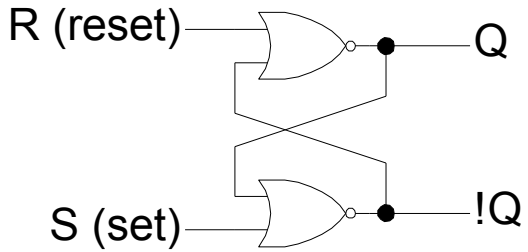


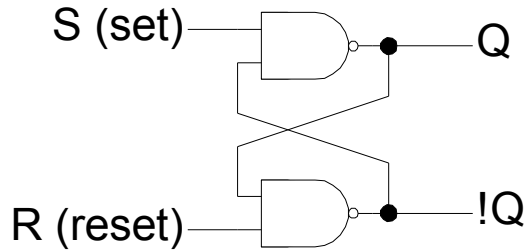


- Hazards like this are hard to fix. We could always find a 2-level version of the previous circuit and get something hazard free...

Fixing hazards with latches

- Can also fix hazards using SR (with NOR gates) or S'R' (with NAND gates) Latches
  - An SR Latch can tolerate momentary 0s appearing at its inputs (since we might momentarily move from a set or reset to a hold and then back).
  - An S'R' Latch can tolerate momentary 1s appearing at its inputs (since we might momentarily move from a set or reset to a hold and then back)



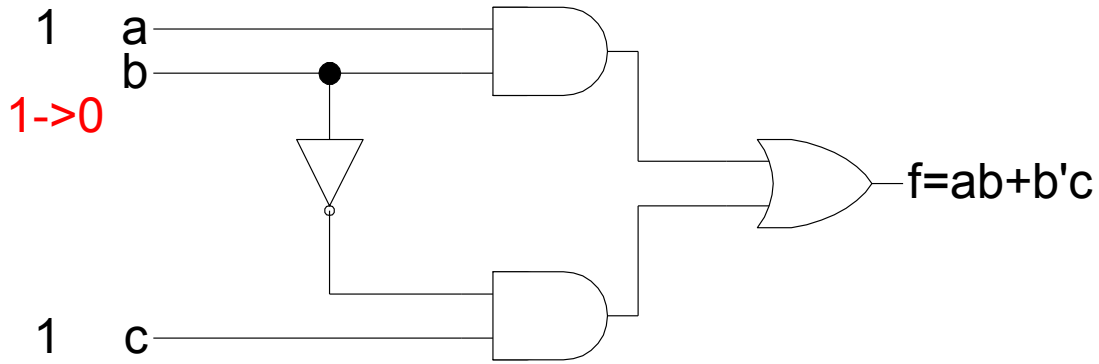


S	R	Q	$\overline{Q}$	
1	0	1	0	
0	0	1	0	(after $S = 1, R = 0$)
0	1	0	1	
0	0	0	1	(after $S = 0, R = 1$)
1	1	0	0	

S	R	Q	$\overline{Q}$	
1	0	0	1	
1	1	0	1	(after $S = 1, R = 0$)
0	1	1	0	
1	1	1	0	(after $S = 0, R = 1$)
0	0	1	1	

Fixing hazards with latches

- Consider our original circuit with a static-1 hazard (temporary 0 at output):



a \ bc	00	01	11	10
0	0	1	0	0
1	0	1	1	1

f=ab+b'c

Fixing hazards with latches

- Consider that we take our output f from the output of a latch.
- Since we are trying to fix static-1 hazards we need to be able to tolerate momentary 0s at latch inputs => **Use a SR Latch (NOR Latch).**
- To get the function f from the latch output, we need equations for S and R of the latch (so that the latch gets SET when f should be one, otherwise RESET).

a \ bc	00	01	11	10
0	0	1	0	0
1	0	1	1	1

Equation for S

$$S = ab + b'c$$

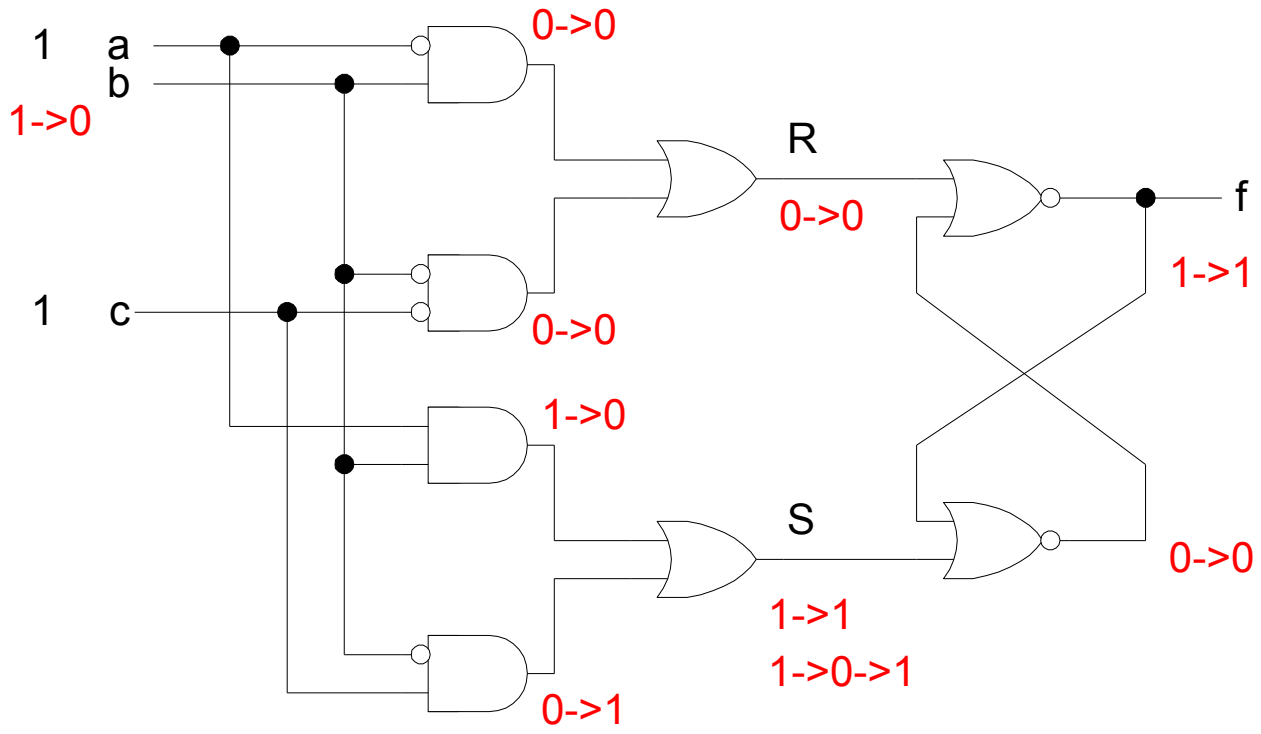
a \ bc	00	01	11	10
0	1	0	1	1
1	1	0	0	0

Equation for R

$$R = b'c' + a'b$$

Fixing hazards with latches

- Draw a circuit using the latch, and see that glitch in output due to the hazard is gone.



Output assignment in asynchronous circuits

- Flow and transition tables might have unspecified entries for outputs.
  - This might be a result of the fundamental mode assumption.
  - This might be a result of unstable states.
- Note: we always have output values assigned for stable states!
  We should think about what happens with the unspecified outputs...
  - They are, in effect, don't cares that we can exploit during minimization of the output logic equations.
  - But, we might temporarily pass through these values while transitioning from one stable state to another stable state.
- Depending on the output equations that we derive (due to minimization of the output equations), we might end up having **glitches** at our circuit outputs.
  - Glitches are bad; they could get fed into another circuit causing problems. They also waste power.

Avoiding output glitches

- Consider the following flow table with don't cares at some outputs (circuit has one input and one output):

curr state	next state		output	
	x=0	x=1	x=0	x=1
a	(a)	b	0	-
b	c	(b)	-	0
c	(c)	d	1	-
d	a	(d)	-	1

- Consider a transition between two stable states due to a change in an input value and how it might be best to assign the don't care value in an unstable intermediate state:
  - If both stable states produce a 0 output, make output 0 instead of a don't care.
  - If both stable states produce a 1 output, make output 1 instead of a don't care.
  - If stable states produce different outputs, the output can remain a don't care and be used to find a smaller output circuit.
- This will enable us to avoid output glitches when passing through unstable temporary states.

Example

- If we consider possible transitions, we see that some of the output don't cares should be changed to 0 or 1 to avoid glitches.

curr state	next state		output	
	x=0	x=1	x=0	x=1
a	(a)	b	0	-
b	c	(b)	-	0
c	(c)	d	1	-
d	a	(d)	-	1

curr state	next state		output	
	x=0	x=1	x=0	x=1
a	(a)	b	0	0
b	c	(b)	-	0
c	(c)	d	1	1
d	a	(d)	-	1

- The above changes will avoid temporary glitches at the outputs during transitions where the output should not change.

Quiz 17-2

http://m.socrative.com/student/#joinRoom

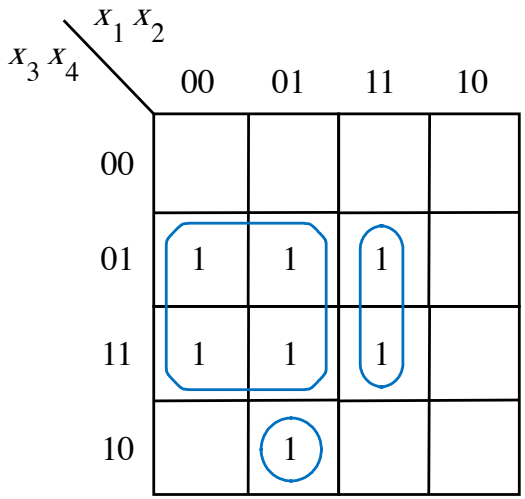
room number: 713113

- Q1: Apply state minimization to the following table:

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

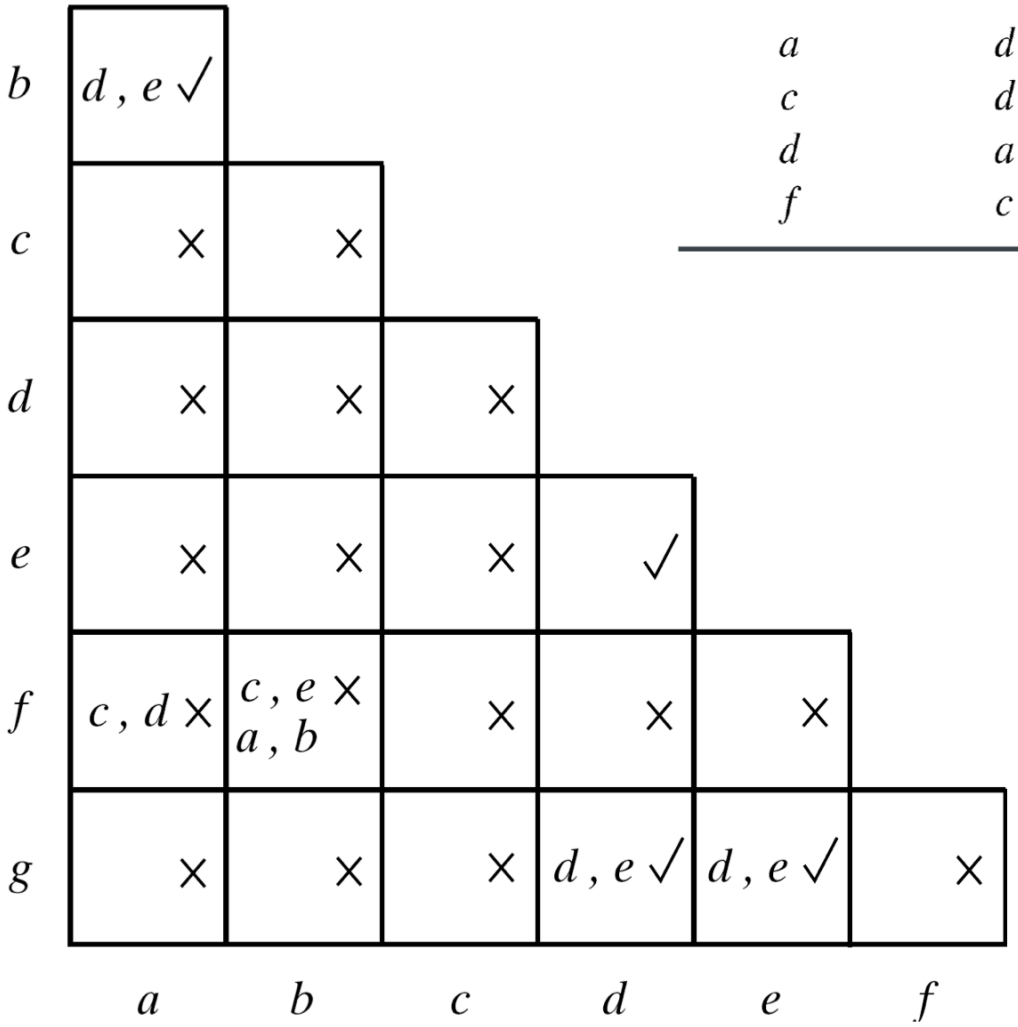
how many states remained?

- Q2: find and remove all hazards in the following k-map:



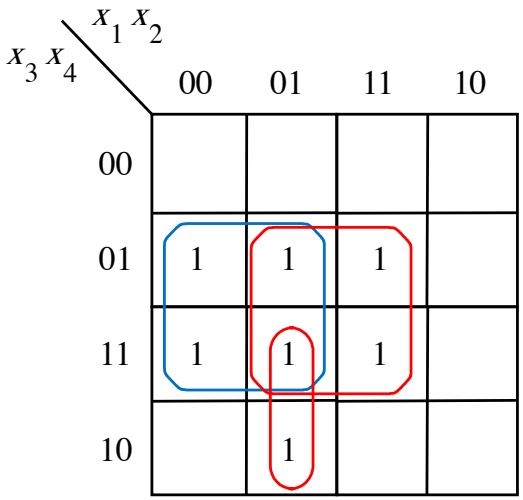
Q1:

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



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

Q2:



$F = x_1'x_2x_3x_4' + x_1'\,x_4 + + x_1x_2x_4$

Hazard free $F = x_1'x_2x_3 + x_1'\,x_4 + + x_2x_4$

Summary

- Introduction to asynchronous circuits
- Analysis and synthesis of asynchronous circuits
- Race conditions and state assignment
- State minimization
- Asynchronous design example
- Hazards

- This lecture is related to Chapter 26 and to 6.10

- Next Lecture:
  - Wrapup lecture

Summary of analysis when latches are present

- Procedure:
  - Label each latch output with **Yj** and its feedback path with **yj**.
  - Derive logic equations for latch inputs **Sj and R**j.
  - Check of SR=0 for NOR Latches and S'R'=0 for NAND Latches. If not satisfied, the circuit may not work correctly.
  - Create logic equations for latch outputs **Yj** using the known behavior of a latch (**Y=S+R'y** for NOR Latches and **Y=S'+Ry** for NAND Latches).
  - Construct a transition table using the logic equations for the latch outputs and circuit stable states.
  - Obtain a flow table, if desired.

Analysis of Asynchronous circuits

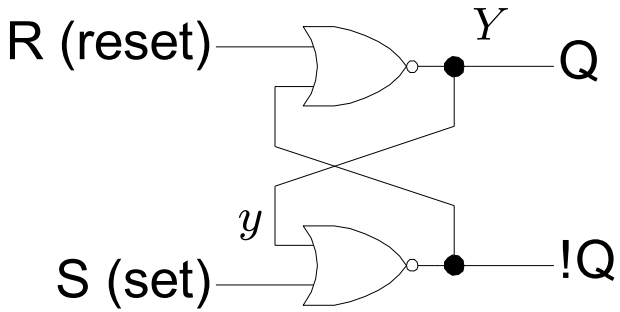
- We identify asynchronous circuits by (1) the presence of latches (un-clocked storage elements) and/or (2) combinational feedback paths (loops through logic gates).
- Analysis involves obtaining *a table or diagram that describes the sequence of internal states and outputs as a function of changes in the circuit inputs*.
- The tables we will try to obtain are **transition tables** and **flow tables** (more or less the same thing as a state table in synchronous circuit FSM design). We can also use state diagrams to describe asynchronous circuits.

Design with latches

- We can also implement asynchronous circuits using latches at the outputs.
- Given the map for each excitation variable Y, derive necessary equations for S and R of a latch to produce Y.
- Derive Boolean equations for S and R.
  - Need to make sure the S and R are never equal (potential problem in Latch).

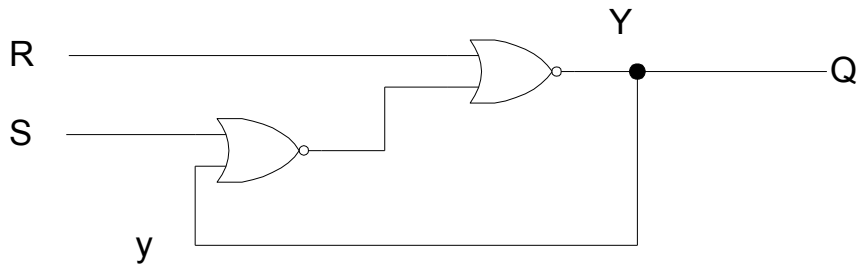
Revising SR latches (excitation table)

- Recall how a SR Latch (NOR) works:



S	R	Q	$\overline{Q}$	
1	0	1	0	
0	0	1	0	(after $S=1, R=0$)
0	1	0	1	
0	0	0	1	(after $S=0, R=1$)
1	1	0	0	

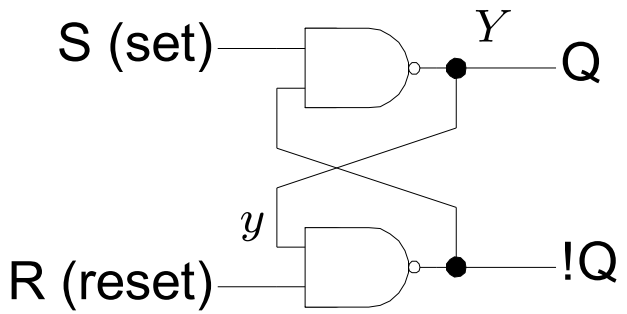
- Assuming we never have the SR=11 case. Can write **excitation table**:



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

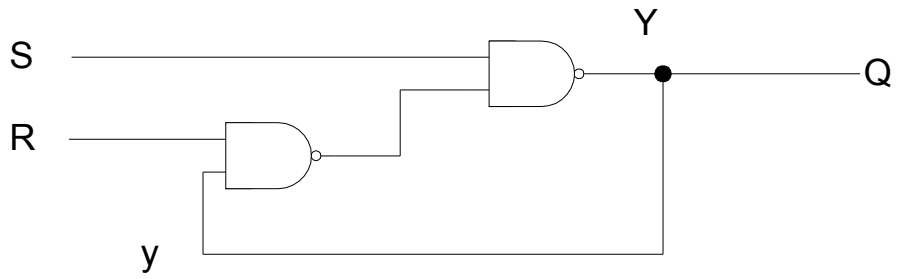
Revising S'R' latches (excitation table)

- Recall how a S'R' Latch (NAND) works:



S	R	Q	$\overline{Q}$	
1	0	0	1	
1	1	0	1	(after $S=1, R=0$)
0	1	1	0	
1	1	1	0	(after $S=0, R=1$)
0	0	1	1	

- Assuming we never have the SR=00 case. Can write **excitation table**:



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

Implementation using latches

- Consider our example again, and assume we want to use a S'R' latch:

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

$$Y = DG + G'y$$

- Need to figure out how to select S and R for the NAND Latch (while making sure never 0 at same time):

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

$$S = (DG)'$$

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

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

$$R = (D'G)'$$

Implementation using latches

- Can draw the circuit: