
DO NOT SHARE
SLIDES AND CLASS MATERIALS
ON ONLINE SITES
Course Hero

Advanced Logic Design
Asynchronous Sequential Circuit

Mingoo Seok
Columbia University

BV chapters: 9.1-9.3, 9.6

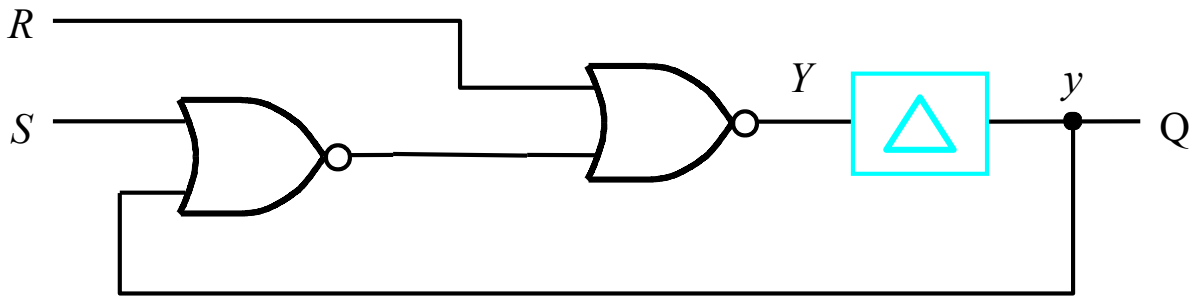
Asynchronous Behavior

- An asynchronous sequential circuit does not use flip-flops and a clock signal; its timing depends on propagation delays through the gates in feedback loops that implement the states of the circuit.
- Transitions between states take place when changes in input signals occur.

Assumptions/Requirements

- For reliable operation, the inputs to the circuit must change in a specific manner – we will impose a simplifying constraint that **the inputs change one at a time**.
- Moreover, there must be **sufficient time between the changes in input signals to allow the circuit to reach a stable state**, which is achieved when all internal signals stop changing.
- When changing from one stable state to another stable state, the circuit may pass through one or more unstable states.

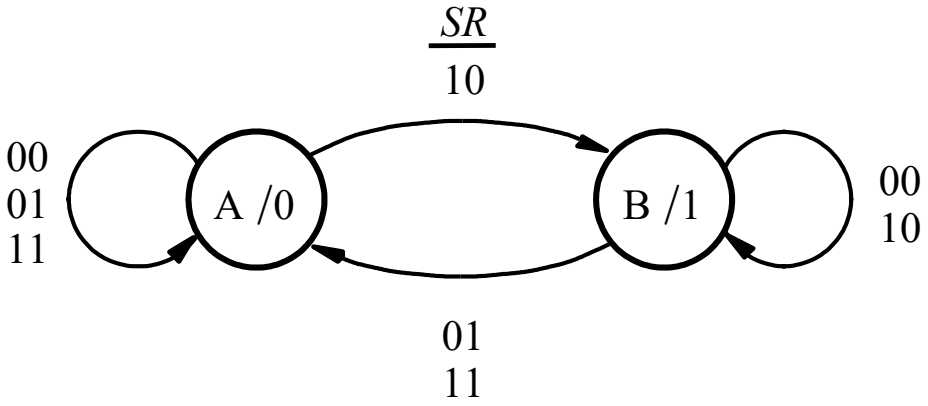
SR Latch



Present state y	Next state			
SR =	00	01	10	11
	Y	Y	Y	Y
0	0	0	1	0
1	1	0	1	0

Stable state

SR Latch



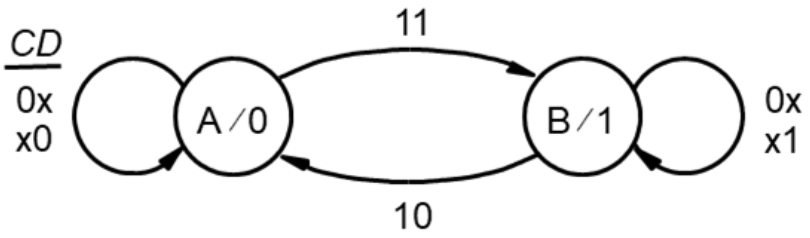
Present state	Next state				Output Q
	SR = 00	01	10	11	
A	(A)	(A)	B	(A)	0
B	(B)	A	(B)	A	1

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

- Moore machine
- State table is called a *flow table* or an *excitation table*

D Latch

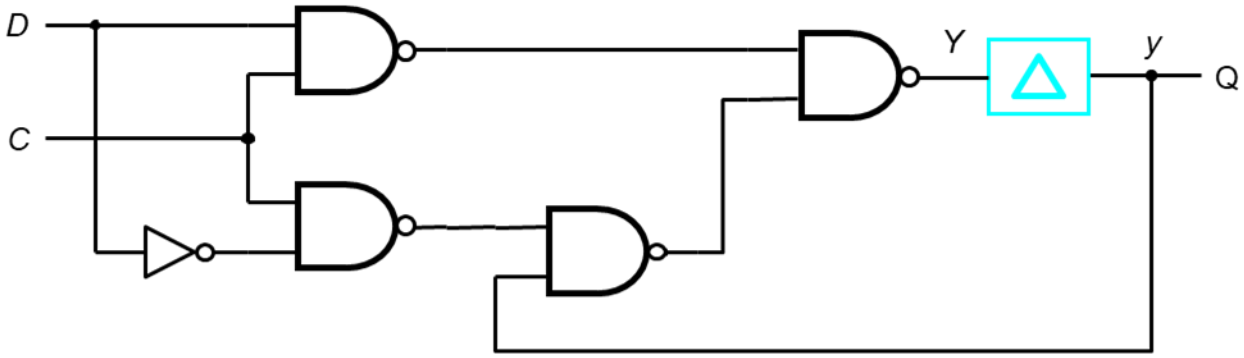
Present state y	Next state CD = 00	01	10	11	Q
	Y	Y	Y	Y	
0	(0)	(0)	(0)	1	0
1	(1)	(1)	0	(1)	1



$$Y = CD + \overline{C}y + Dy$$

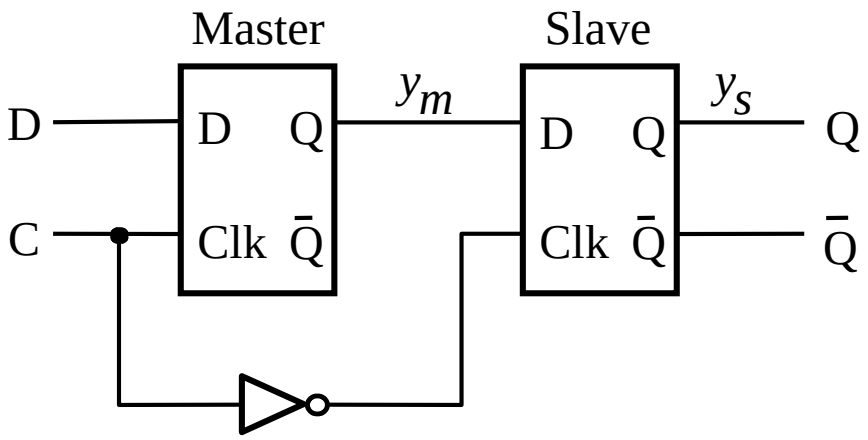
For hazard mitigation

D Latch



$$Y = CD + \overline{C}y + Dy$$

Master-Slave D Flip-Flop



Present state $y_m\ y_s$	Next state $Y_m\ Y_s$				Output Q
	CD = 00	01	10	11	
00	(00)	(00)	(00)	10	0
01	00	00	(01)	11	1
10	11	11	00	(10)	0
11	(11)	(11)	01	(11)	1

Excitation table

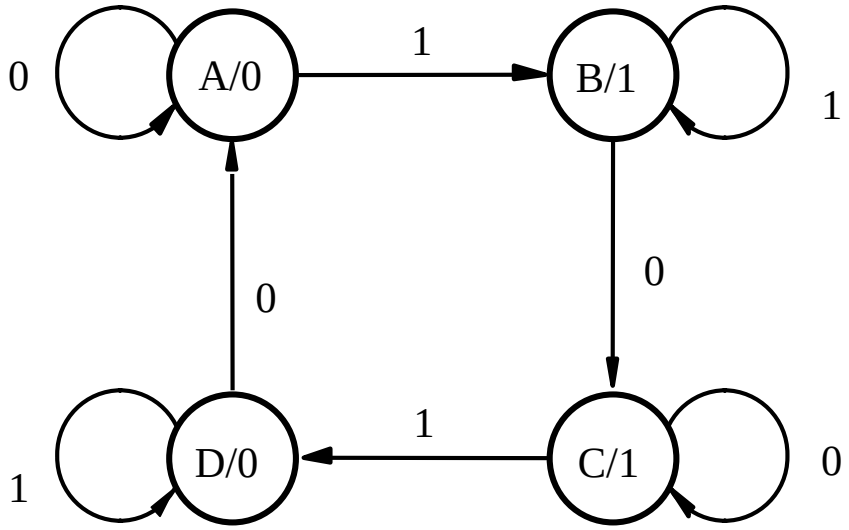
Unspecified Entries

Present state	Next state				Output Q
	CD = 00	01	10	11	
S1	(S1)	(S1)	(S1)	S3	0
S2	S1	S1	(S2)	S4	1
S3	S4	S4	S1	(S3)	0
S4	(S4)	(S4)	S2	(S4)	1

Present state	Next state				Output Q
	CD = 00	01	10	11	
S1	(S1)	(S1)	(S1)	S3	0
S2	S1	–	(S2)	S4	1
S3	–	S4	S1	(S3)	0
S4	(S4)	(S4)	S2	(S4)	1

- Suppose the MSFF is at the state S2 and CD=10
- Now it receives CD=01, which involves simultaneous changes in two signals → such changes are not allowed in asynchronous FSM
- Have an unspecified entry (-) in the excitation table

Serial Parity Generator



Present State	Next state		Output
	$w = 0$	$w = 1$	z
A	(A)	B	0
B	C	(B)	1
C	(C)	D	1
D	A	(D)	0

- Count the number of the input pulse *w* and produce the output *z=0* if the number is even and *z=1* otherwise

Serial Parity Generator

Present state $y_2 y_1$	Next state $Y_2 Y_1$		Output z
	$w = 0$	$w = 1$	
00	(00)	01	0
01	10	(01)	1
10	(10)	11	1
11	00	(11)	0

(a) Poor state assignment

Present state $y_2 y_1$	Next state $Y_2 Y_1$		Output z
	$w = 0$	$w = 1$	
00	(00)	01	0
01	11	(01)	1
11	(11)	10	1
10	00	(10)	0

(b) Good state assignment

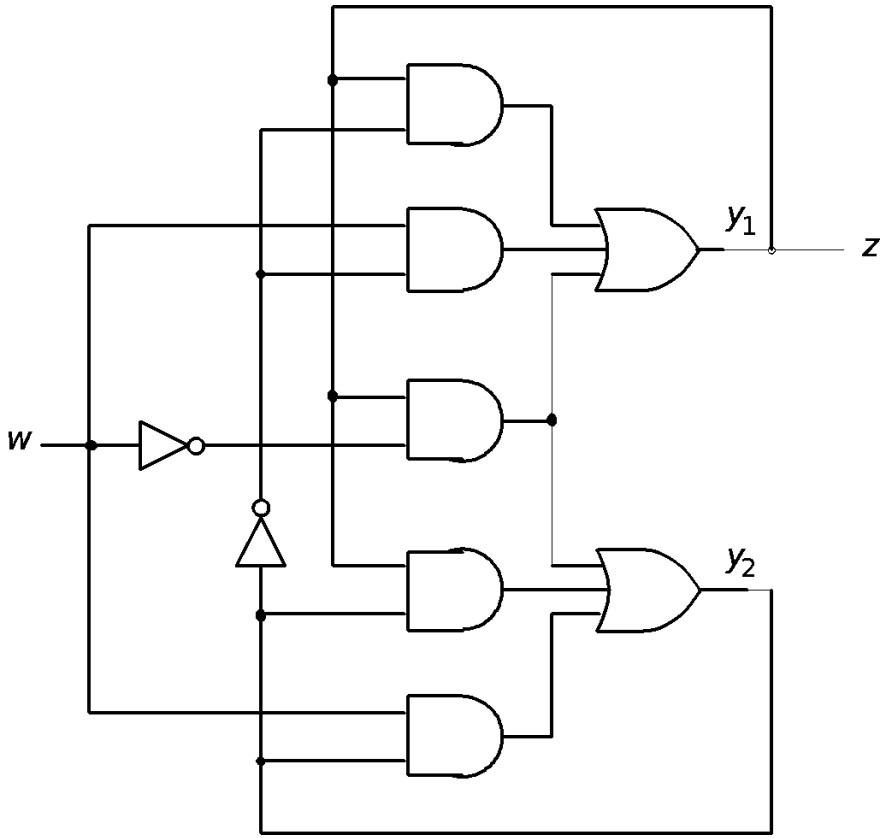
- State transition from B (01) to C (10) involves simultaneous changes of two state variables
- If each variable has slightly different delay, the FSM transits to a state other than C, e.g., D (11) and A (00), and may not be able to be stabilized to C (10)
- We must design for such corner case; here by assigning states so that every state involves only one signal transition

Serial Parity Generator

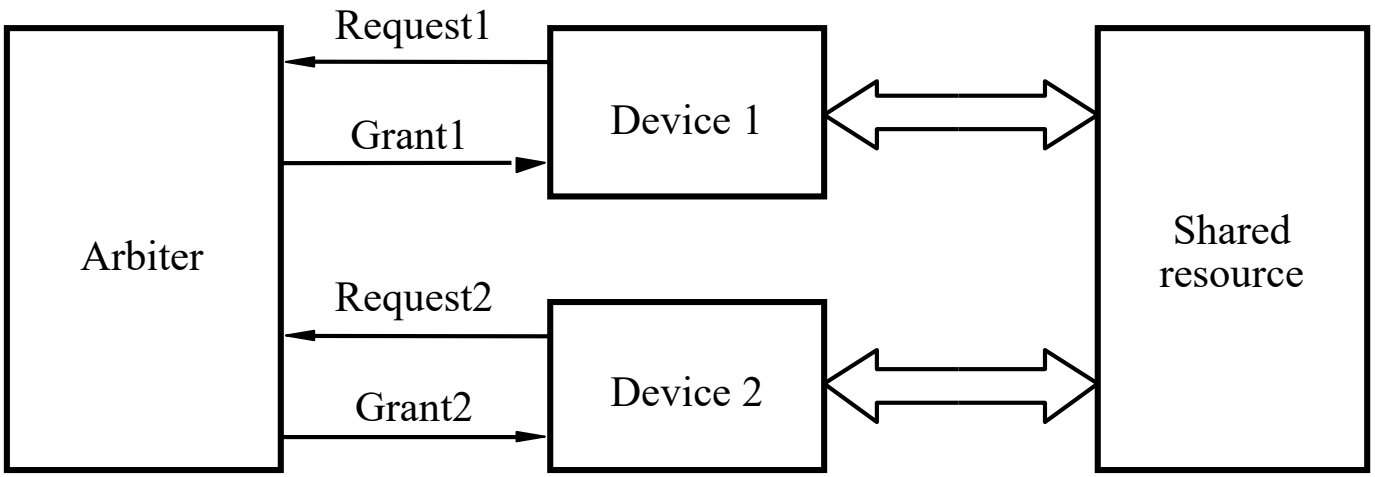
$$Y_1 = w\bar{y}_2 + \bar{w}y_1 + y_1\bar{y}_2$$

$$Y_2 = wy_2 + \bar{w}y_1 + y_1y_2$$

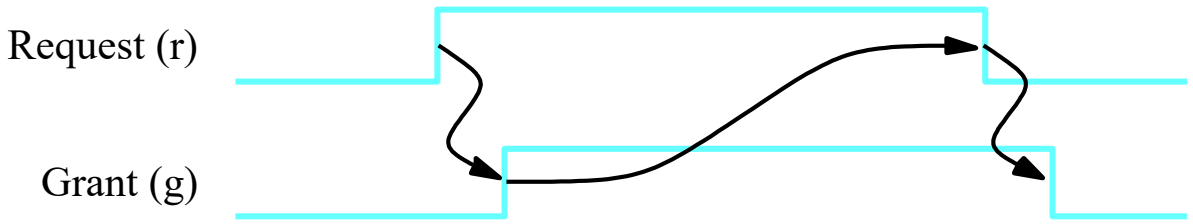
$$z = y_1$$



Simple Arbiter

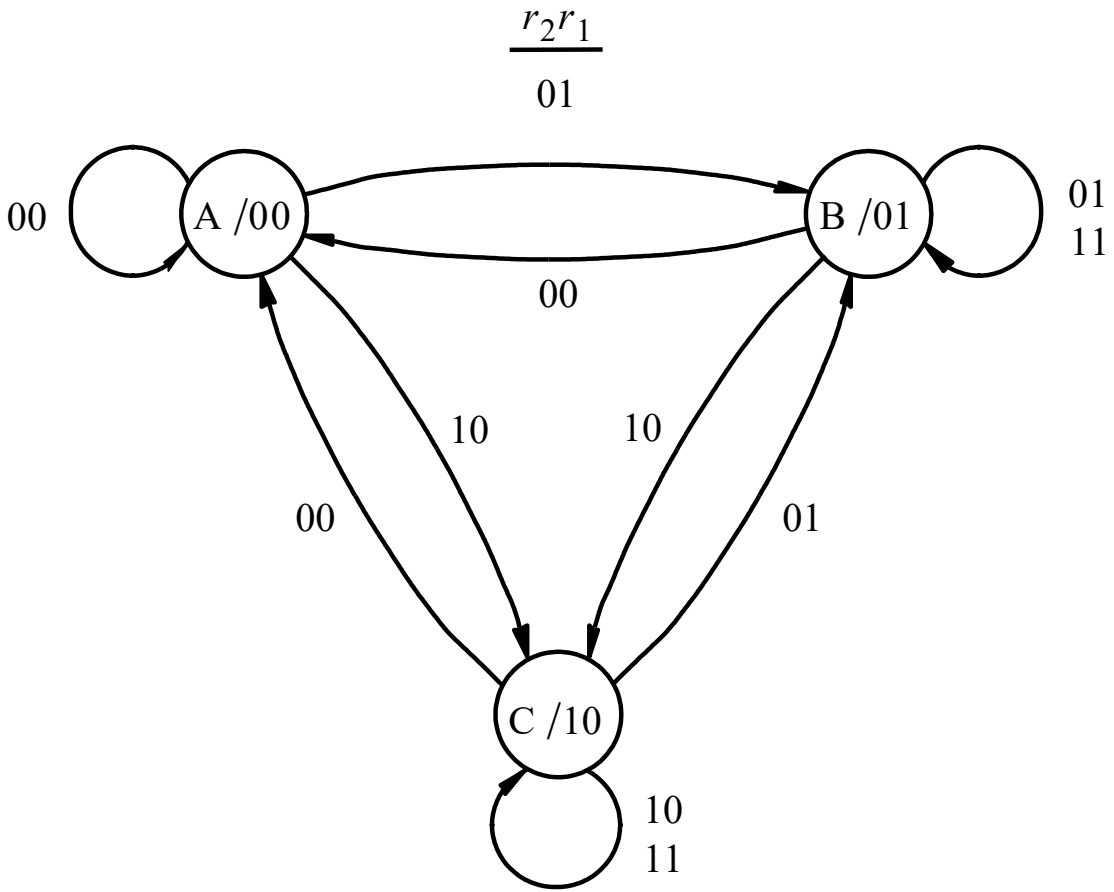


(a) Arbitration structure



(b) Handshake signaling

Simple Arbiter



- A: quiescent
- B: the device A is given permission to use the resource
- C: the device B is given permission to use the resource
- r: request (input to the arbiter)
- g: grant (output of the arbiter)

Simple Arbiter

Present state	Next state r_2r_1 = 00	01	10	11	Output g_2g_1
A	(A)	B	C	—	00
B	A	(B)	C	(B)	01
C	A	B	(C)	(C)	10

	Present state y_2y_1	Next state r_2r_1 = 00	01	10	11	Output g_2g_1
		Y_2Y_1				
A	00	(00)	01	10	—	00
B	01	00	(01)	10	(01)	01
C	10	00	01	(10)	(10)	10
D	11	—	01	10	—	dd

- Left: B→C involves the change of two state variables, posing a race condition! (Y_2Y_1 can be 11 or 00 momentarily before 10)
- Y_2Y_1 being 00 momentarily is okay as it eventually becomes 10; but being 11 is a problem since *the state 11* is not defined!
- Remedy: add a dummy state *D* that is *11*

Simple Arbiter

- If both ***r*** signals come at the same time
- This violates the key assumption, i.e., each signal change at a time in asynchronous FSM
- Mutual Exclusion (ME) element
  - Both inputs are 1, it makes one output go to 1 and keeps the other at 0
- It is still a challenging problem

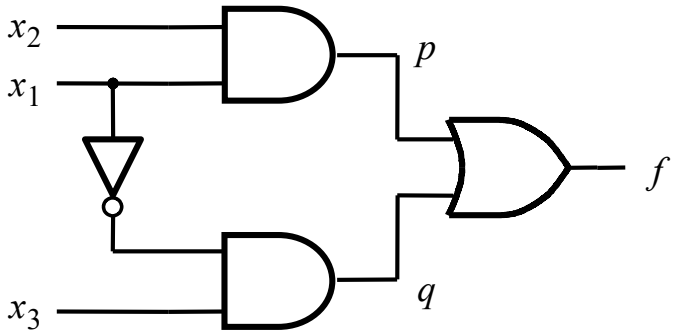
Hazard

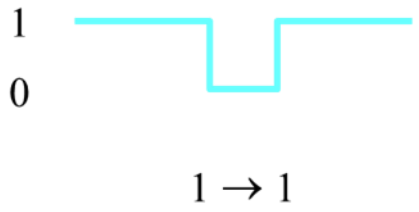
(a) Static hazard

(b) Dynamic hazard

- Hazard: the glitches caused by the structure of a given circuit and propagation delays in the circuits in asynchronous circuits
- Static hazard: a signal is supposed to remain at a particular logic value when an input variable changes its value, but instead the signal undergoes a momentary change in its required value
- Dynamic hazard: hazard when a signal changes

Static Hazard

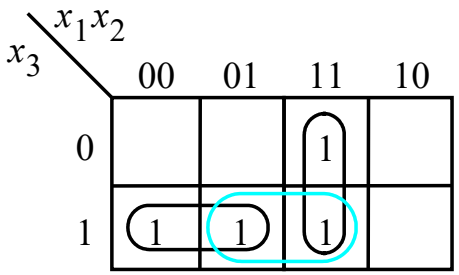




(a) Circuit with a hazard

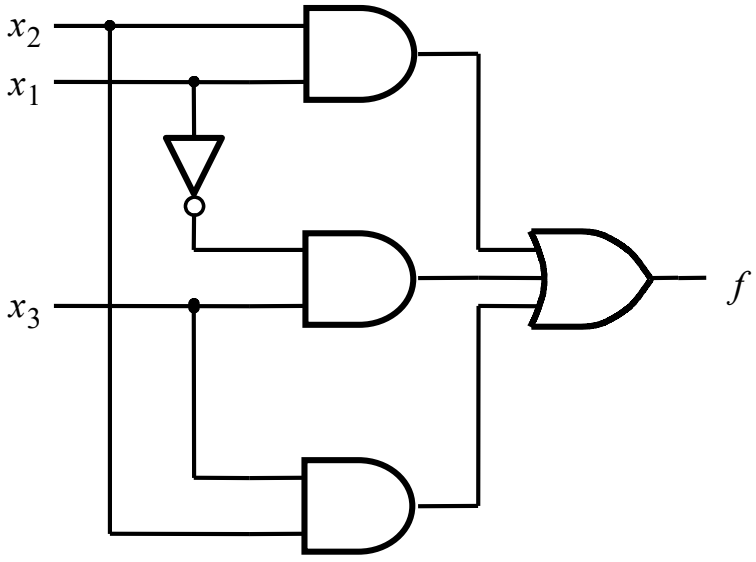
- Initially, $x_1x_2x_3$=111 and f=1
- x_1 changes from 1 to 0
- f is supposed to be 1 but will experience a dip to 0
- What is the time width of the dip?

Static Hazard



$$f = x_1x_2 + \bar{x}_1x_3$$

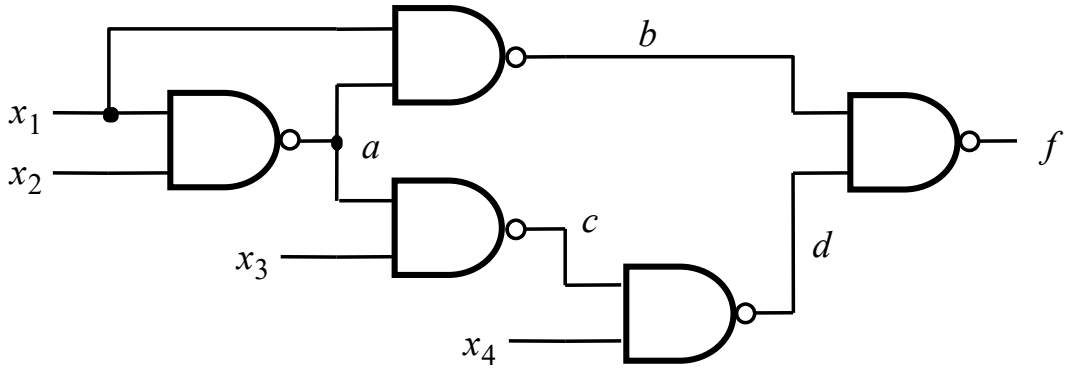
$$f = x_1x_2 + \bar{x}_1x_3 + x_2x_3$$

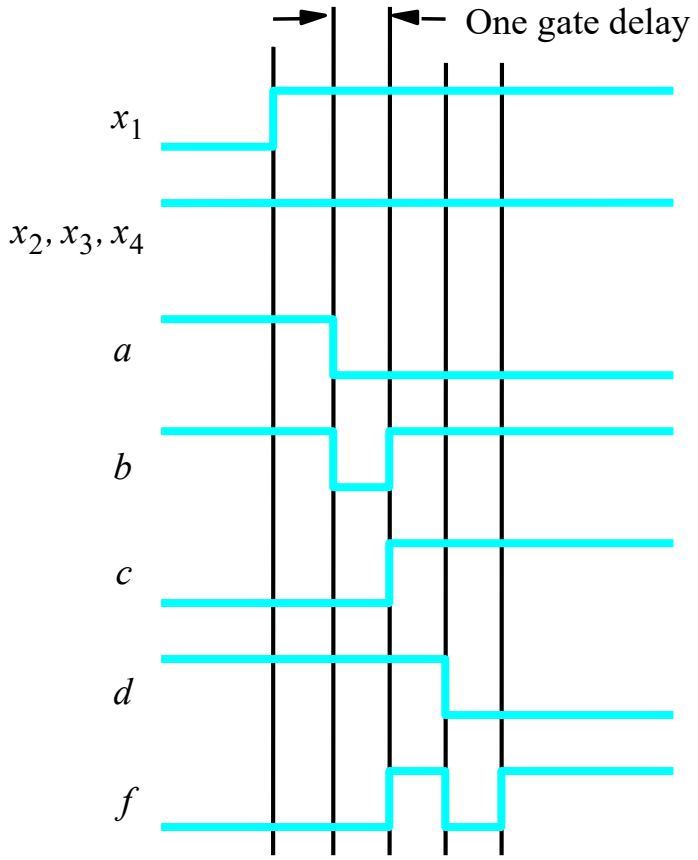


(c) Hazard-free circuit

- A potential hazard where two adjacent 1s are not covered by a single product term
- Add additional PIs would remove hazards

Dynamic Hazard Example





(b) Timing diagram

Dynamic Hazard

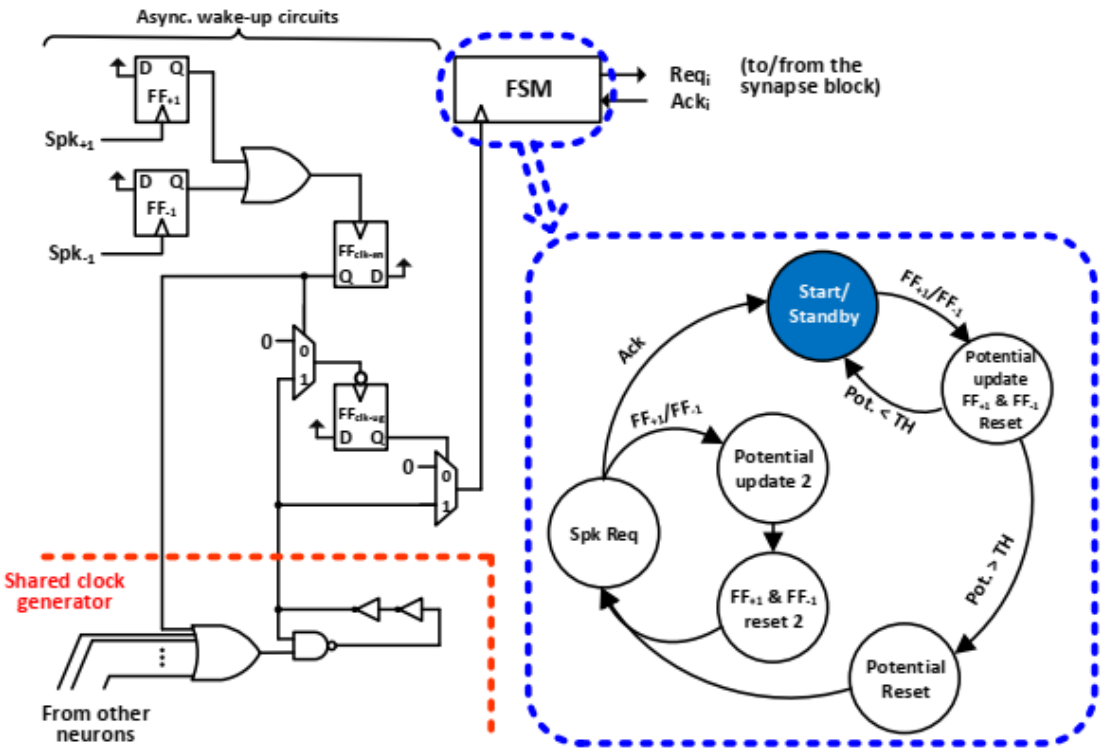
$$f = x_1\bar{x}_2 + \bar{x}_3x_4 + x_1x_4$$

- Usually occurred multilevel circuits obtained using factoring or decomposition techniques
- 2-level circuits would remove the dynamic hazards
- However, gate delay mismatches, e.g., by process variation, can still create hazard even in the 2-level circuits

More Verifiable Approach

- Asynchronously generate synchronous clock
- Disable it synchronously
- No need to care about the hazard
- Delay skews among signals will be ignored by sampling signals at a clock edge
- Delay to generate/enable clock

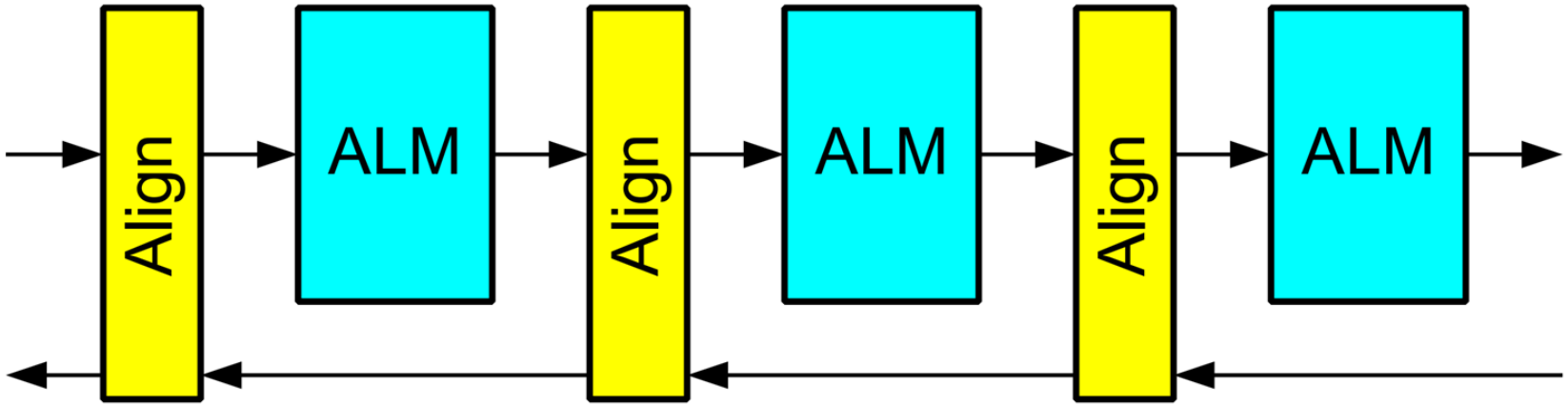
Example



Source: D. Wang et al. ASSCC20

- Asynchronously start the clock generator
  - Start the clock at Spk+1 pulse rising edge
- With the clock, start the synchronous FSM
- Disable the clock when FSM enters the idle state
- Reset the registers to zero, disabling the clock generator

Asynchronous Pipeline



Source: Daly

- Align blocks separate async logic modules
  - It waits for all inputs valid, then signals the previous align block
  - when signal received from forward align block, set inputs invalid (if RZ) and allow next set of inputs to enter

Asynchronous Pipeline

- Very different from synchronous pipelines
- Delay of pipe is sum of delays of stages
  - not rounded up to delay of longest stage
- The number of problems in the pipe is flexible
  - Different branches of a pipe need not have the same number of stages
  - The first problem may exit before the second problem enters regardless of the number of stages

Remarks

- Asynchronous circuits are much more complex design & verify than synchronous counterparts
- Asynchronous circuits are usually bulkier
- Unless absolutely necessary, better to stick to synchronous designs
- Some circumstances : multi-clock FIFO
- Maybe it is still better to fall back to hybrid design such as synchronous circuits w/ asynchronously generated clock