

- Develop fully asynchronous circuits, e.g., microprocessors.
 - Why don't people do this often? Poor CAD tool support. Greater design complexity.
- Interface synchronous circuits with different clock periods and phases.
- Interface synchronous systems with asynchronous sensors.
- Methodically design flip-flop and latch like circuits of arbitrary complexity and specifications.

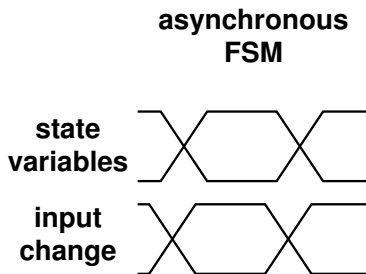
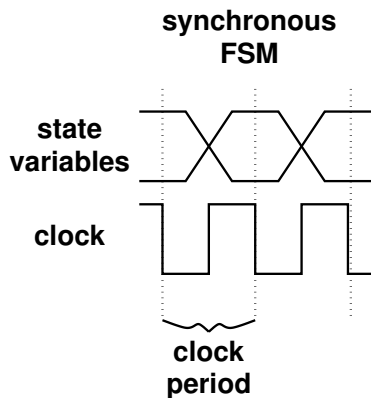
Outline

1. Synchronous vs. asynchronous design
2. Asynchronous synthesis techniques
3. Example

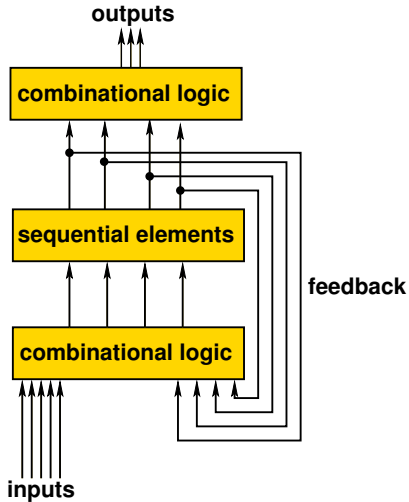
Synchronous vs. asynchronous design

- Synchronous design makes a lot of problems disappear
- Glitches not fatal
- FSM design easier

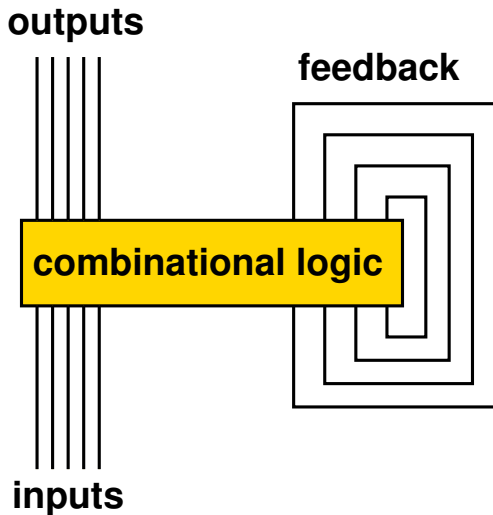
Synchronous vs. asynchronous FSMs



Synchronous system



Asynchronous machine block diagram



Synchronous FSM specification

Non-deterministic (multi-input pseudo-)FSA for complex specifications

State diagram

State table

Synchronous FSM minimization

Implication chart

Maximal cliques and compatibles

Prime compatibles

Binate covering formulation

Minimized state table

Synchronous FSM synthesis

State assignment

State variable synthesis

Output variable synthesis

Technology mapping

Asynchronous FSM specification

Non-deterministic (multi-output pseudo-)FSA for complex specifications

State diagram, **explicitly considering all clock-like inputs**

State table

Asynchronous FSM minimization

Implication chart

Maximal cliques and compatibles

Prime compatibles

Binate covering formulation

Minimized state table

Unchanged.

Asynchronous FSM synthesis

State splitting, if necessary

State assignment, preserving encoding adjacency

Hazard-free state variable synthesis

Potentially hazard-free output variable synthesis

Technology mapping

Disable any CAD tool optimizations that may eliminate hazard-covering cubes

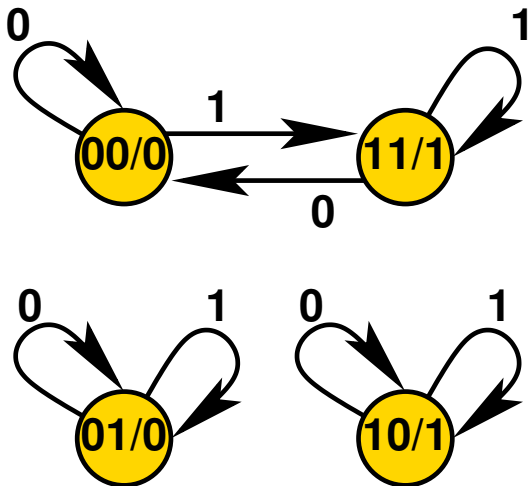
Outline

1. Synchronous vs. asynchronous design
2. Asynchronous synthesis techniques
3. Example

Asynchronous FSM state assignment

- For synchronous FSMs, state assignment impacts area and power consumption
- For asynchronous FSMs, incorrect state assignment results in incorrect behavior
- A *race* is a condition in which the behavior of the circuit is decided by the relative switching speeds of two state variables
- An asynchronous FSM with races will not behave predictably
- Avoid *critical races*, races which result in different end states depending on variable change order

Incorrect asynchronous assignment

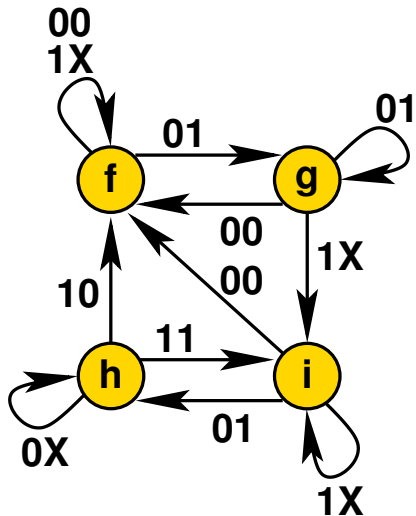


Asynchronous FSM state assignment

s	s ⁺		Q
	0	1	
00	00	11	0
01	01	01	0
10	10	10	1
11	00	11	1

- Consider 00 \rightarrow 11 transition
- Becomes trapped in 01 or 10
- Which one?
 - Random

Asynchronous FSM adjacency



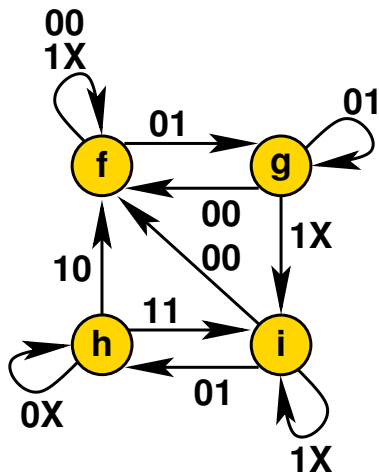
Asynchronous FSM adjacency

- Two input bits
- When a particular input leads to a state, maintaining that input should generally keep one in the state
 - E.g., 01 for g
- Will show exception later

Asynchronous FSM adjacency

- f adjacent to g , h , and i
- g adjacent to f and i
- h adjacent to f and i
- i adjacent to f , g , and h
- Four states $\rightarrow \lceil \lg(4) \rceil = 2$ state variables
- However, in 2D space, each point is adjacent to only two others
- Need at least 3D

Asynchronous FSM adjacency

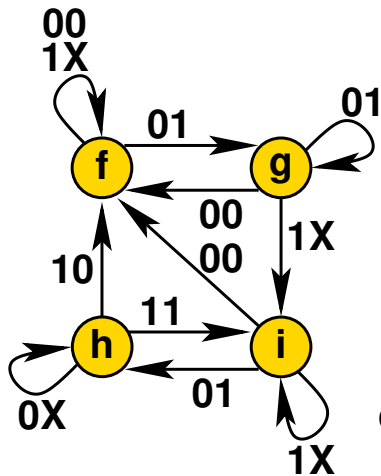


		a	
		0	1
b	0	f	g
	1	h	i

Asynchronous FSM adjacency

- Need all adjacent states in AFSM to be adjacent
- i to f transition could be trapped in g !
- What to do for a graph with too many connections?
- Split states and hop through some states to reach others

Asynchronous FSM adjacency



	a	
	0	1
b	0	1
	f	g
	1	i

	ab			
	00	01	11	10
c	0	1	2	3
	g	f	i1	i2
	1	X	h1	h2
				X

Asynchronous FSM adjacency

current state	next state			
	00	01	10	11
f	f	g	f	f
g	f	g	i ₂	i ₂
h ₁	h ₁	h ₁	f	h ₂
h ₂	h ₂	h ₂	h ₁	i ₁
i ₁	f	h ₂	i ₁	i ₁
i ₂	i ₁	i ₁	i ₂	i ₂

AFSM synthesis redundancy

- Even if AFSM has a fully connected adjacent state assignment there are still additional complications
- State variables must have stable transitions
- E.g., for a SOP implementation, every state pair that is connected in the state transition graph must be covered by at least one cube
- Hazards may cause incorrect operation for AFSMs

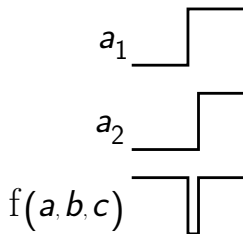
AFSM transition stability

Given that $f(a, b, c)$ is a state variable

		ab			
		00	01	11	10
c	0	1	1	1	0
	1	0	0	1	0

$$\bar{a} \bar{c} + a b$$

$$\bar{a} b \bar{c} \rightarrow a b \bar{c}$$



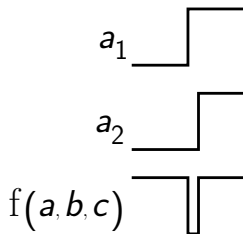
AFSM transition stability

Given that $f(a, b, c)$ is a state variable

		ab			
		00	01	11	10
c	0	1	1	1	0
	1	0	0	1	0

$$\bar{a}_1 \bar{c} + a_2 b$$

$$\bar{a} b \bar{c} \rightarrow a b \bar{c}$$



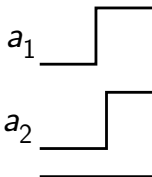
AFSM transition stability

Given that $f(a, b, c)$ is a state variable

$f(a, b, c)$		ab			
		00	01	11	10
c	0	1	1	1	0
	1	0	0	1	0

$$\bar{a} \bar{c} + a b + b \bar{c}$$

$$\bar{a} b \bar{c} \rightarrow a b \bar{c}$$



$f(a, b, c)$

AFSM design summary

- AFSMs immediately react to input changes
- No need to worry about clock
- However, design more complicated
- Stability
- Unstable states must have appropriate (no glitches) outputs
- Adjacent states must have adjacent assignments
- Glitches on state variables may be fatal

Outline

1. Synchronous vs. asynchronous design
2. Asynchronous synthesis techniques
3. Example

Example: D flip-flop

Design a falling edge triggered D flip-flop

Other examples

Design a two-input AFSM (LM)

- Output 1 iff L is low and M was high at some time during most recent L high period
- Output 0 otherwise

Design an inverting D flip-flop that is simultaneously rising-edge and falling-edge triggered

Design an interface queue between two synchronous domains with differing clock periods and phases