Today's lecture

- What's left?
- Remaining two topic talks.
- Pitches.
- Feature freeze.
- Optional AFSM lecture.

What's left?

- 14 April: Outline of course topics, which is useful when preparing for the final.
- 14 April: Submit poster draft PDF via Gradescope for feedback. P/F.
- 17 April: Polish project. Print poster. Practice pitch.
- 18 April
  - Demo Day: Come prepared with project, poster, and pitch.
  - Submit project report via Gradescope.
  - Submit final version of poster via Gradescope.
  - Team member evaluation.
- 19 April: Practice exam material posted.
- 26 April: Final exam
  - Comprehensive, short.
  - Optional review sessions covering no new material.

Why pitches? Selling your ideas

- Actual selling.
- Team building.
- Fundraising.
- Management support.
- Layered approach: 30 second, five minute, one hour.

Feature freeze

- T1: No new features.
- T2: No new non bug fix code.
- T3: No new code.
- Huge impact on whether demo/product/etc., achieves goal.

Thanks for taking this seriously!

Working on hard problems with people...

- ... who don't really care = torture
- ... who care, lead, push hard, and get results = **FUN!**

Keep in contact!

Asynchronous Finite State Machine Synthesis

Teacher:	Robert Dick
Office:	2417-E EECS
Email:	dickrp@umich.edu
Phone:	847–530-1824

13 April 2017

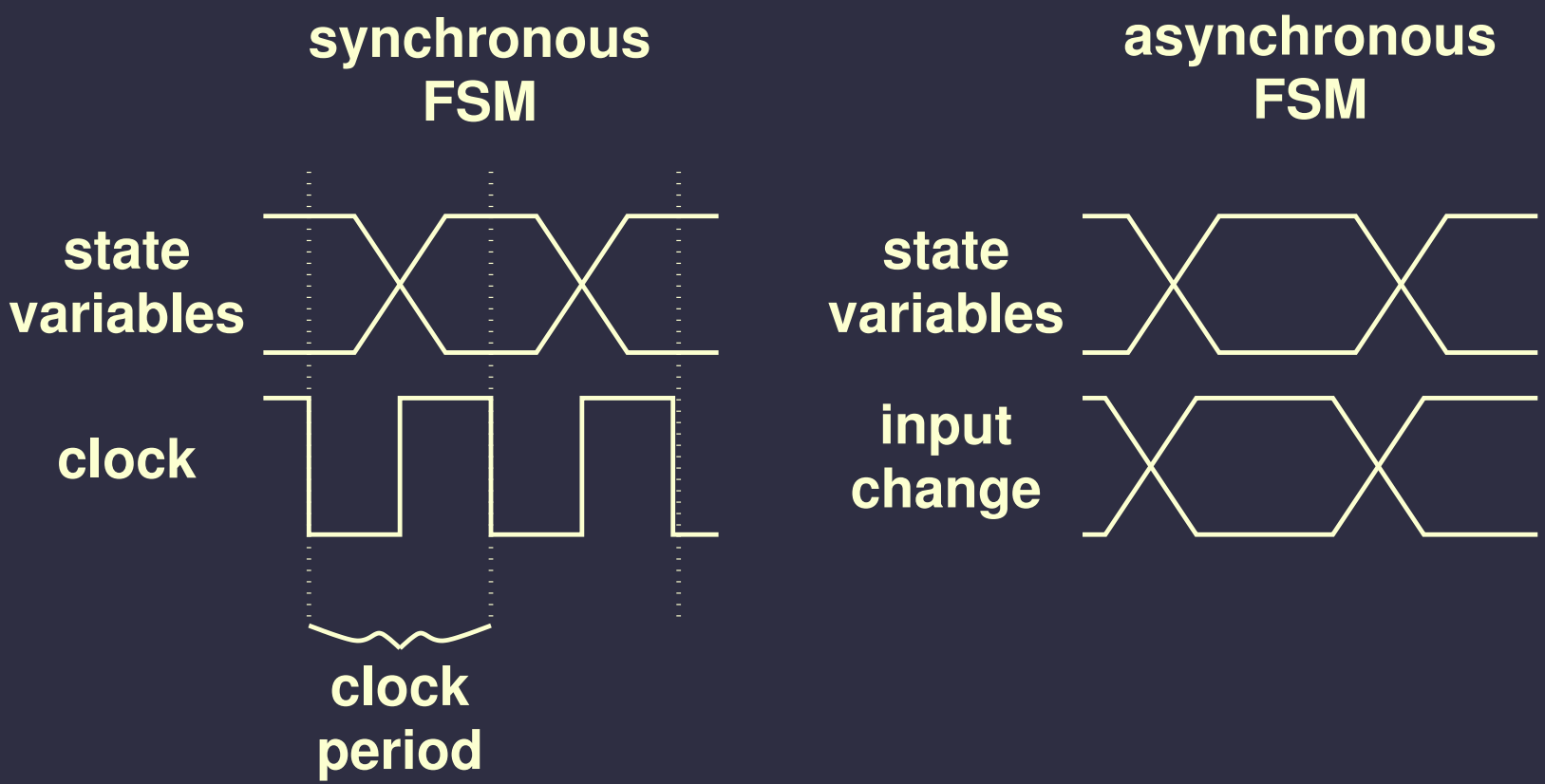
- Develop fully asynchronous circuits, e.g., microprocessors.
  - Why don't people do this often? Poor CAD tool support. Greater design complexity.
- Interface synchronous circuits with different clock periods and phases.
- Interface synchronous systems with asynchronous sensors.
- Methodically design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

Outline

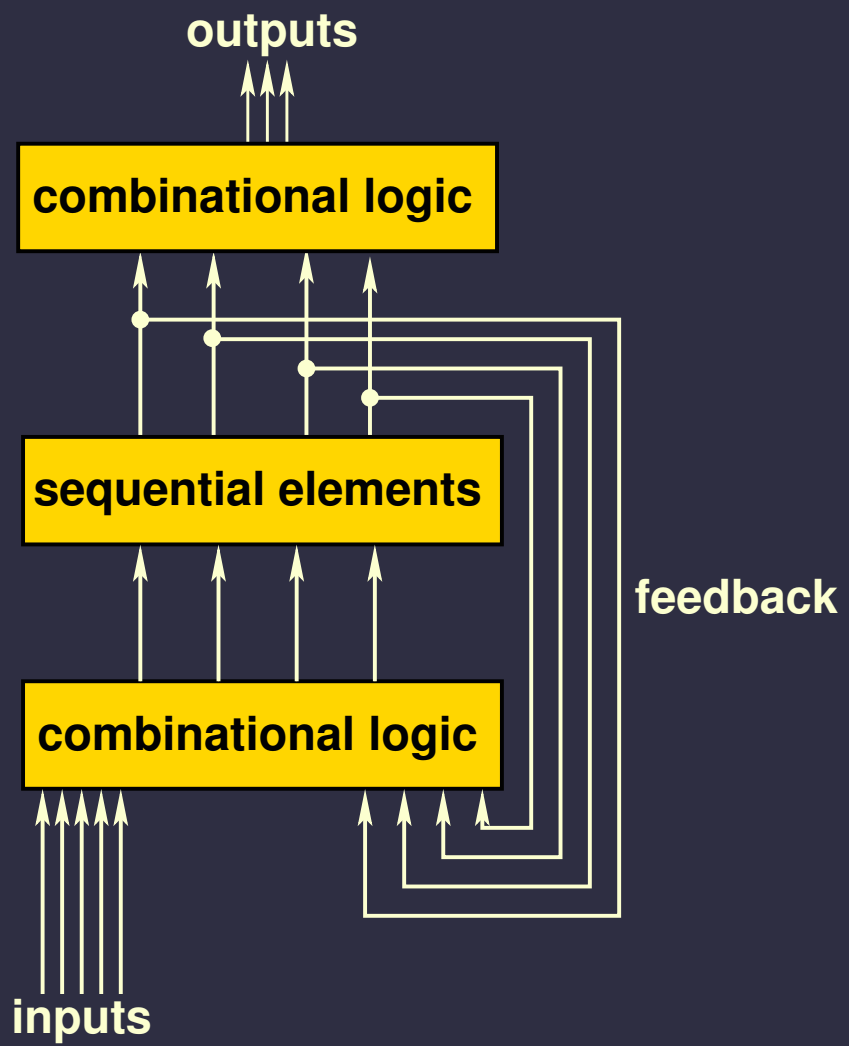
Synchronous vs. asynchronous design

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

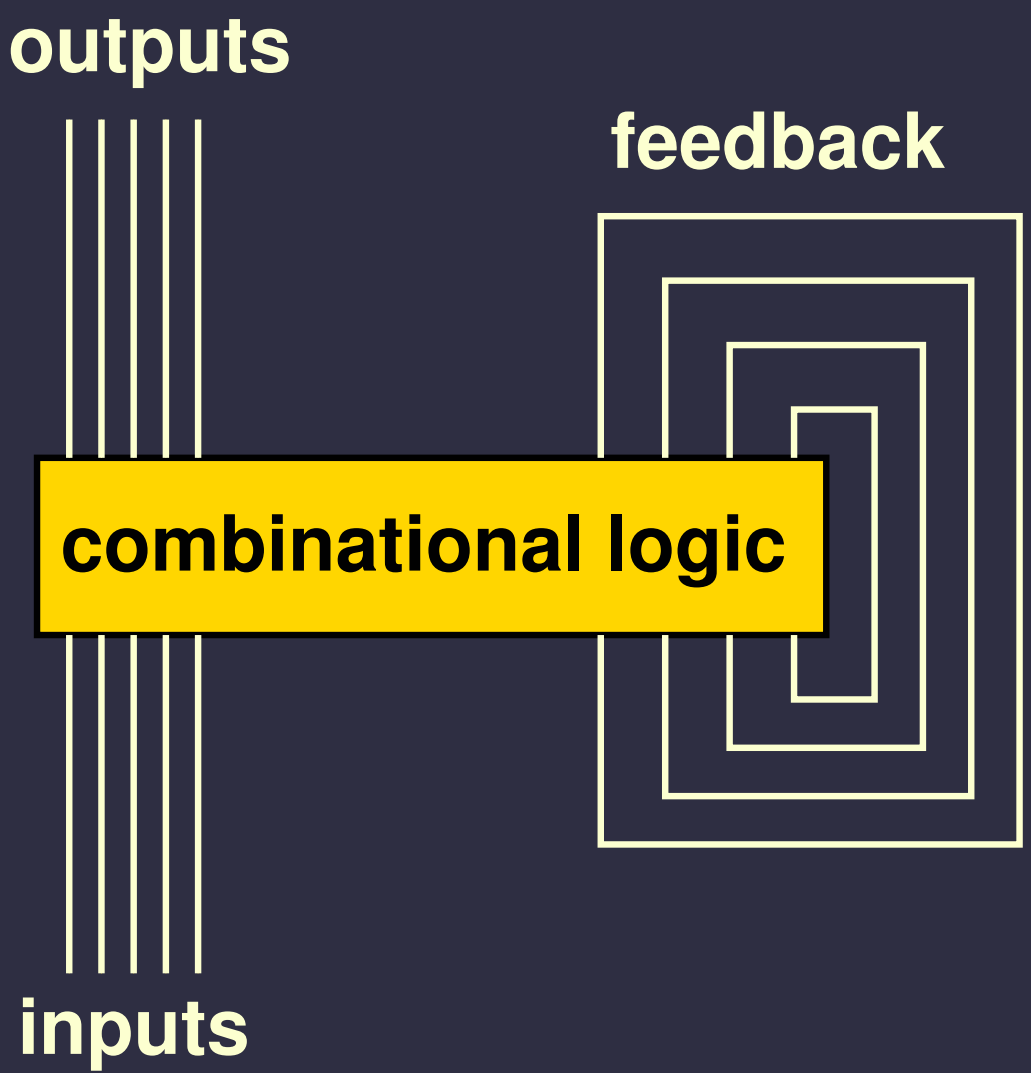
Synchronous vs. asynchronous FSMs



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

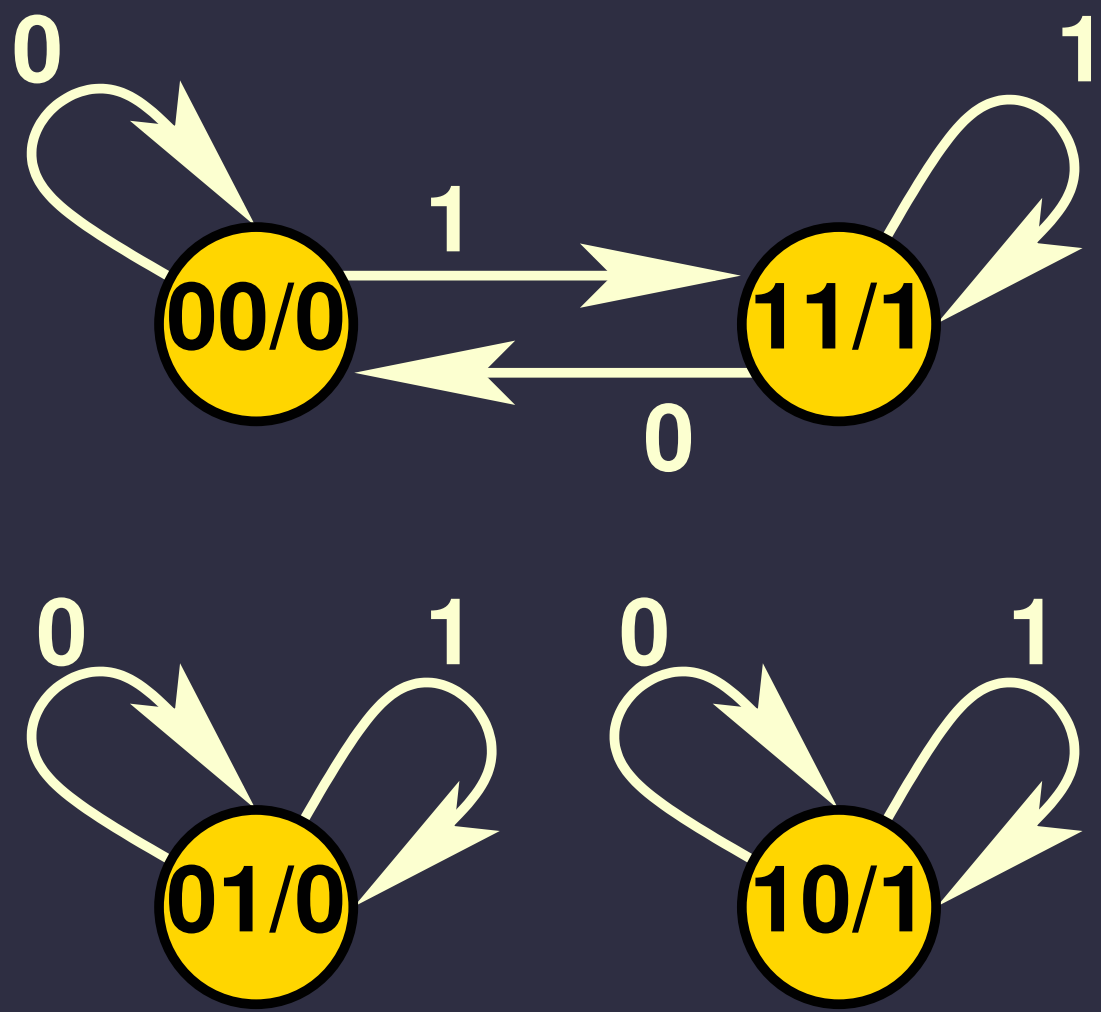
- State splitting, if necessary
- State assignment, preserving encoding adjacency
- Hazard-free state variable synthesis
- Potentially hazard-free output variable synthesis
- Technology mapping
- Disable any CAD tool optimizations that may eliminate hazard-covering cubes

Outline

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

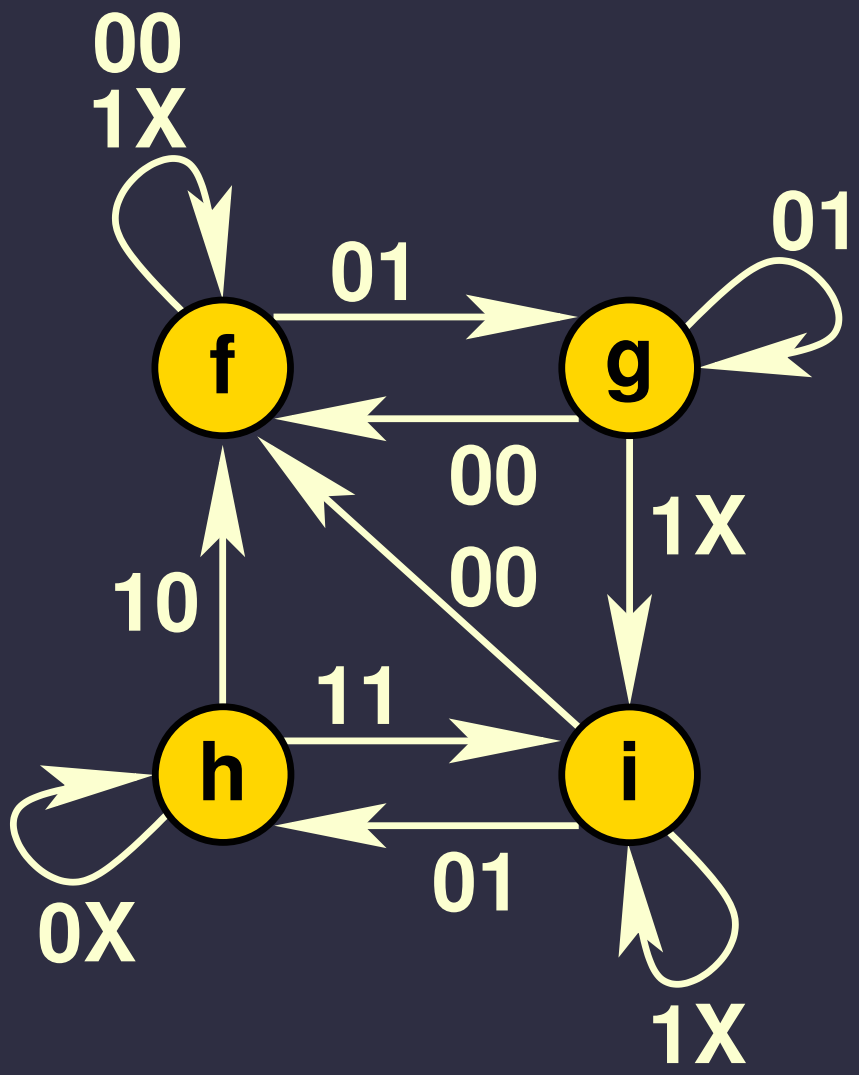


Asynchronous FSM state assignment

	s^+		
s	0	1	Q
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



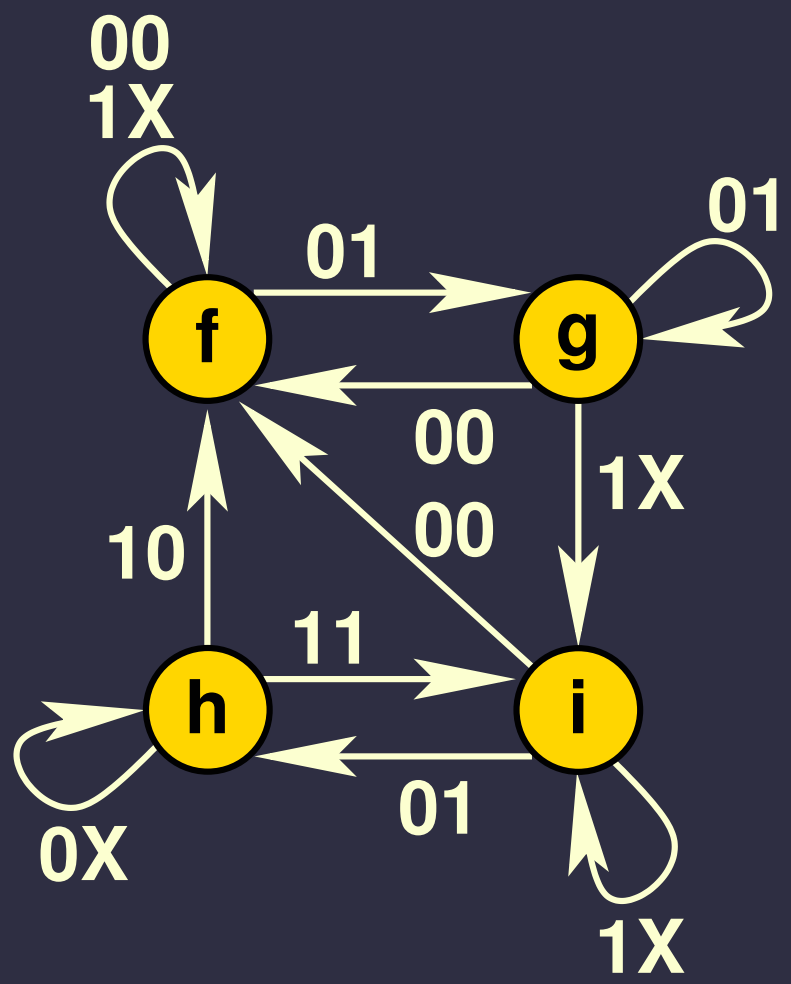
Asynchronous FSM adjacency

- 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

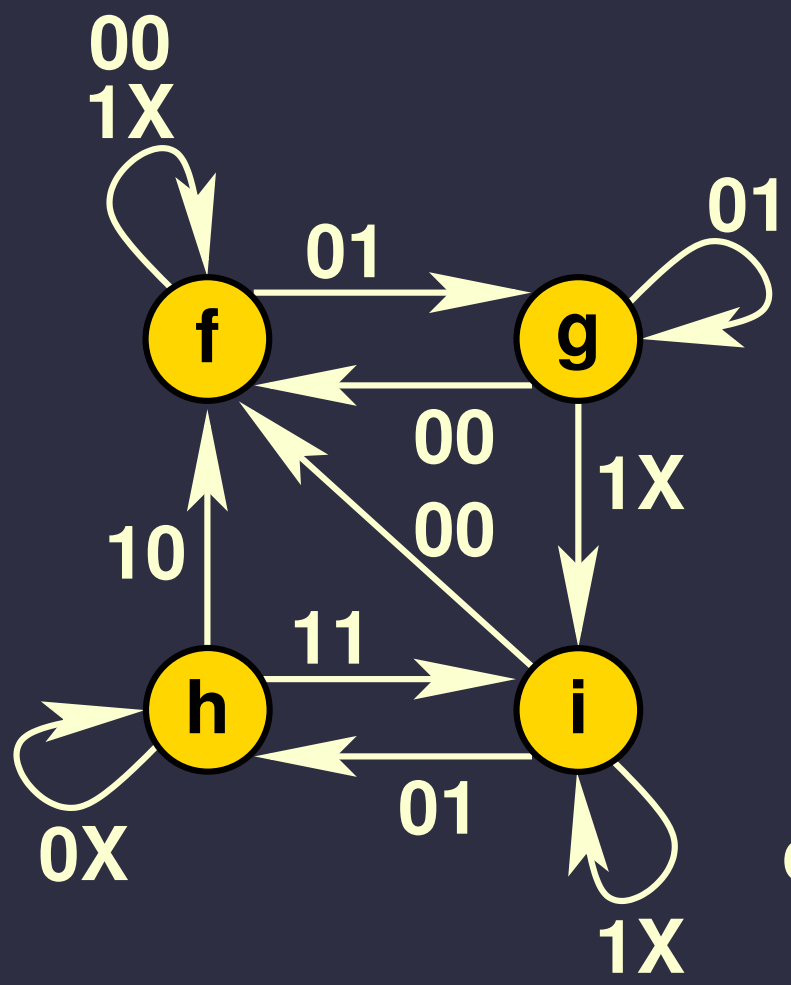


b \ a	0	1
0	f	g
1	h	i

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



b \ a	0	1
0	f	g
1	h	i

c \ ab	00	01	11	10
0	g	f	i2	i2
1	X	h1	h2	X

Asynchronous FSM adjacency

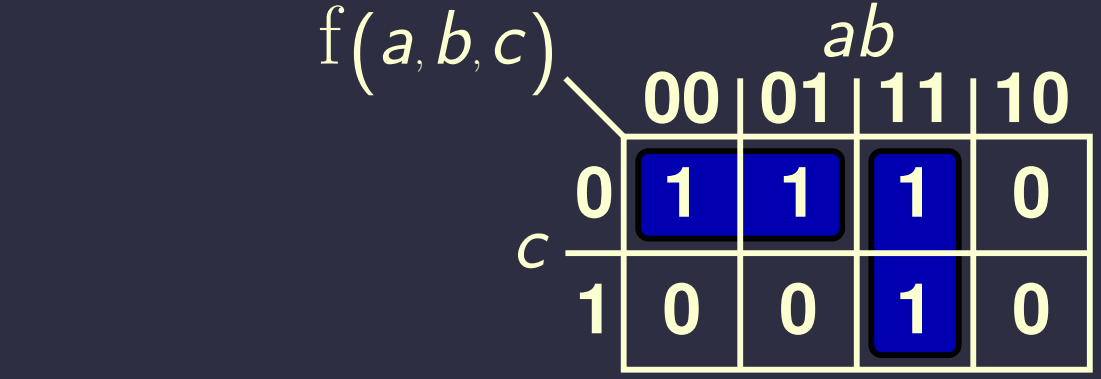
current state	next state			
	00	01	10	11
f	f	g	f	f
g	f	g	i_2	i_2
h_1	h_1	h_1	f	h_2
h_2	h_2	h_2	h_1	i_1
i_1	f	h_2	i_1	i_1
i_2	i_1	i_1	i_2	i_2

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 me covered by at least one cube
- Hazards may cause incorrect operation for AFSMs

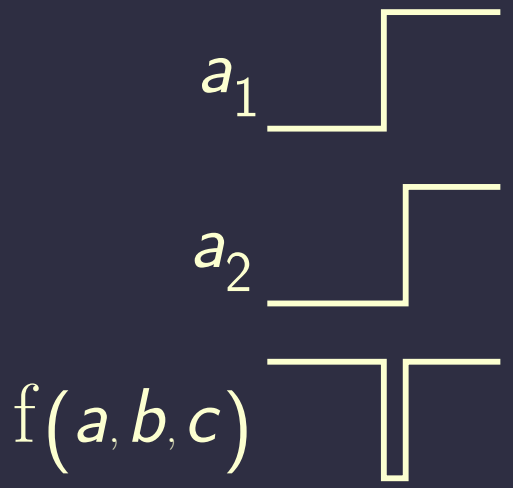
AFSM transition stability

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



$$\overline{a}\ \overline{c} + a\ \overline{c}$$

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



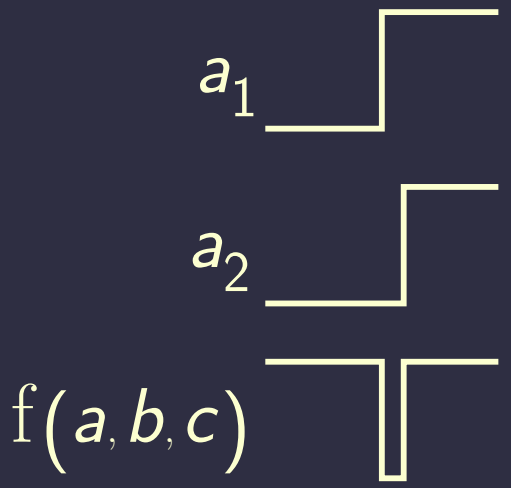
AFSM transition stability

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

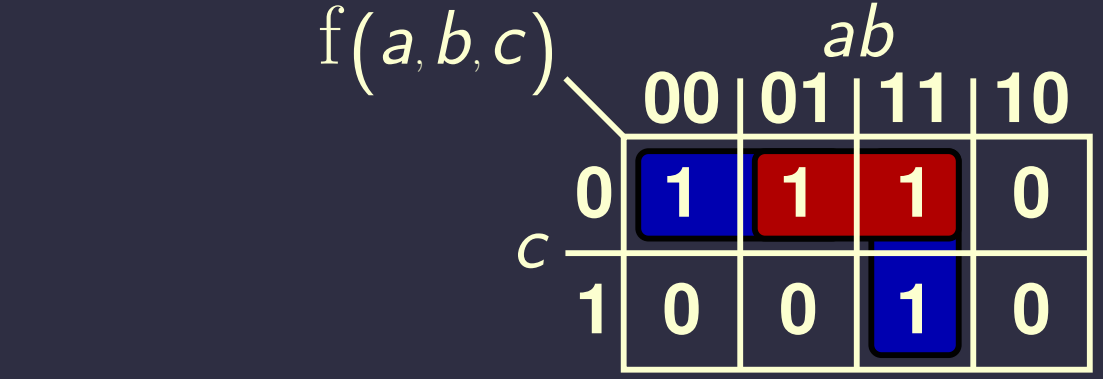
$$\overline{a_1}\ \overline{c} + a_2\ b$$

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



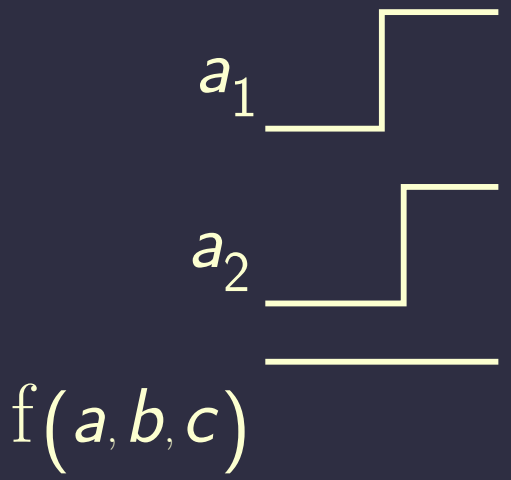
AFSM transition stability

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



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

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



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

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