Advanced Digital Logic Design – EECS 303

http://ziyang.eecs.northwestern.edu/eecs303/

Teacher:	Robert Dick
Office:	L477 Tech
Email:	dickrp@northwestern.edu
Phone:	847–467–2298

NORTHWESTERN
UNIVERSITY

Outline

Minimization of incompletely specified FSMs

CS	NS(0)	NS(1)	OUT
A	A	X	1
B	B	C	1
C	C	A	X
D	A	D	0

Reason for prime compatibles

Consider the following maximal compatibles

AB

BC

CD

BE $\longrightarrow$ BC

Minimization stages

- State table
- Implication chart
- Maximal cliques (for larger problems)
  - Largest fully connected subgraphs
- Maximal compatibles
- Prime compatibles
- Binate covering

Outline

Registers and counters

- Once you understand flip-flops and FSM design, registers and counters are easy
- Shift registers can shift contents left or right
- Registers
  - Commonly a group of D flip-flops written and read simultaneously
- Counters
  - FSMs that have only a clock input
  - Can count up or down in some binary number system
  - Can also cyclicly shift a one through flip-flops (ring counter)

Multiple-output pseudo-NFAs

- Similar to standard NFAs
- Have multiple accept states
- Simple translation to Moore machines
- Going from DFAs to Mealy machines is more complicated

Outline

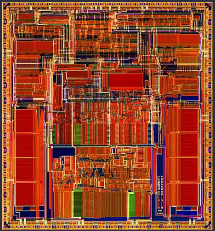
1. Review of state minimization
2. Registers and counters
3. Asynchronous finite state machines

Section outline

Synchronous vs. asynchronous design

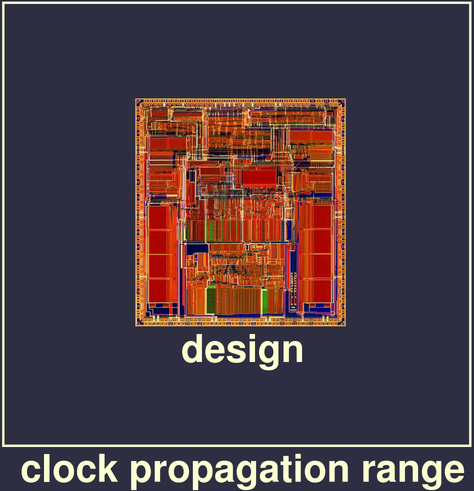
- Synchronous design makes a lot of problems disappear
- Glitches not fatal
- FSM design easier
- However, things are likely to change soon

Current SOC clocking and communication

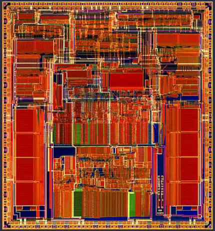


design

Future SOC clocking and communication

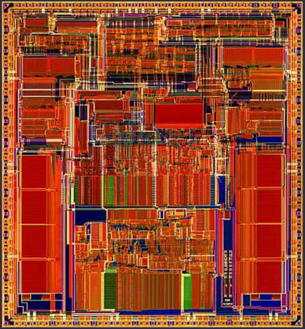


Future SOC clocking and communication



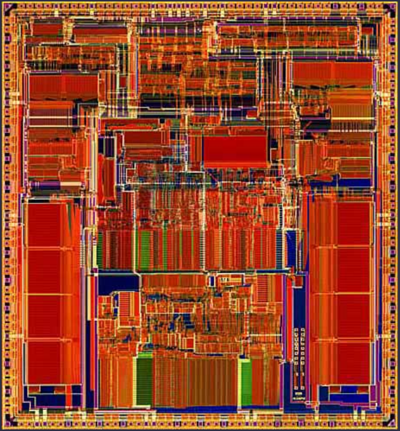
design

Future SOC clocking and communication



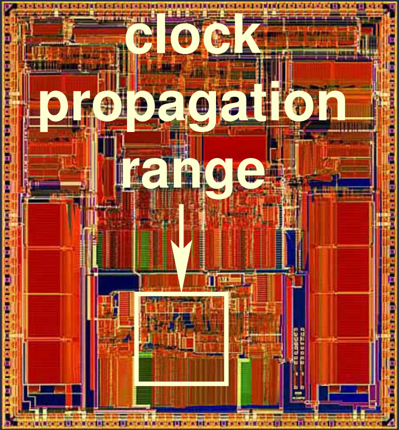
design

Future SOC clocking and communication



design

Future SOC clocking and communication

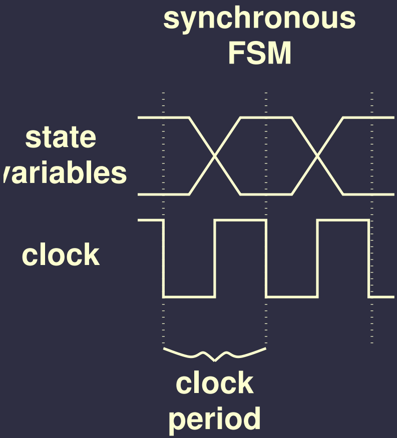


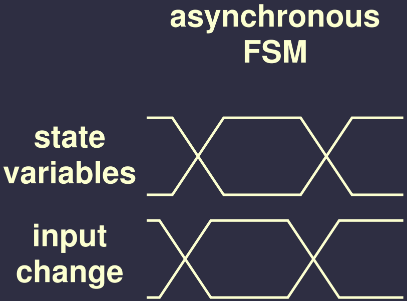
design

Globally asynchronous, locally synchronous (GALS)

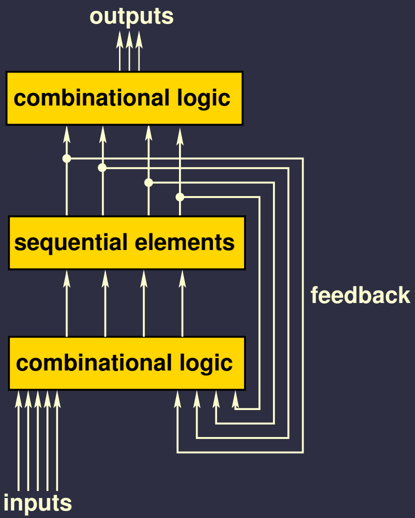
- Complete flexibility in region frequencies
- Reputation for inefficient communication
- However, results always improving
- Asynchronous circuits traditionally skipped
- However, you will encounter them in interface circuits and are likely to encounter them more and more frequently
- Asynchronous design likely to become increasingly important

Synchronous vs. asynchronous FSMs





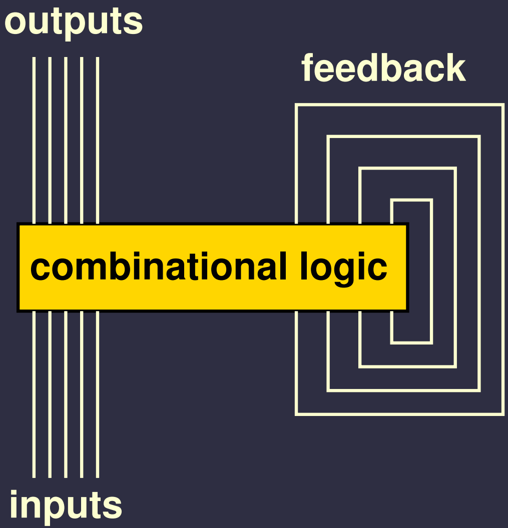
Synchronous system



Differences from synchronous circuits

- Avoid critical races (more later)
- Avoid glitches
- State can be a function of input as well as state variables
- May need to do state splitting

Asynchronous machine block diagram

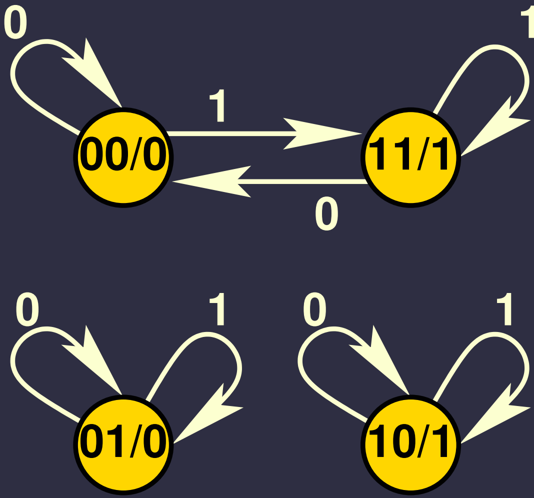


Section outline

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



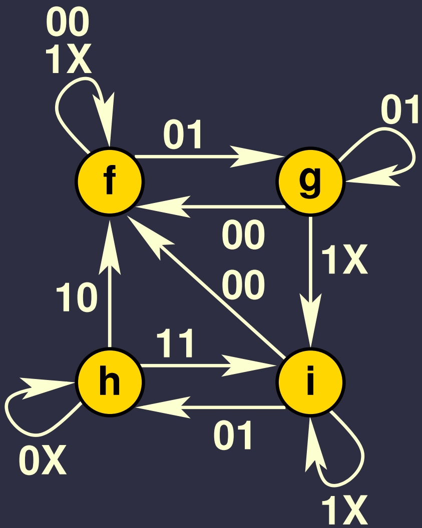
Asynchronous FSM state assignment

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

Consider 00 $\longrightarrow$ 11 transition

- Becomes trapped in 01 or 10
- Which one?
  - Random

Asynchronous FSM adjacency



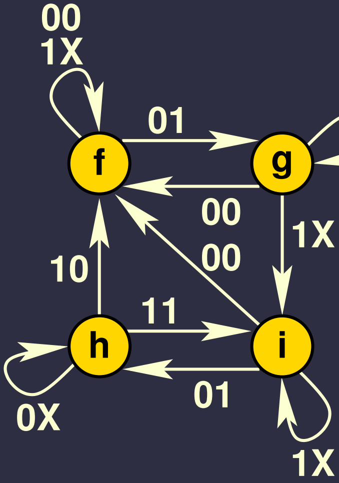
Asynchronous FSM adjacency

- 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



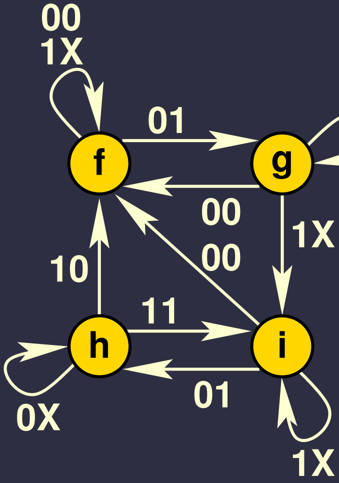
$$\begin{tikzcd}
f \arrow[r, bend left, "01"] \arrow[loop above, "00\ 1X"] & g \arrow[l, bend left, "00"] \arrow[d, "1X"] \arrow[loop right, "01"] \\
h \arrow[u, "10"] \arrow[r, bend left, "11"] \arrow[loop left, "0X"] & i \arrow[ul, "00"] \arrow[l, bend left, "01"] \arrow[loop below, "1X"]
\end{tikzcd}$$

		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



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

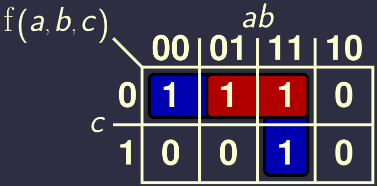
current state	next state 00	01	10	11
f	f	g	f	f
g	f	g	i_2	i_2
h_1	h_1	h_1	f	h_2
h_2	h_2	h_2	h_1	i_1
i_1	f	h_2	i_1	i_1
i_2	i_1	i_1	i_2	i_2

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

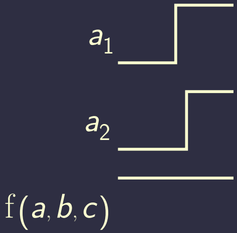
AFSM transition stability

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



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

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



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

Section outline

Debouncing

- Recall previous method of debouncing switch
- Consider SPDT (single pole, double throw) switch
- Pull-up/Pull-down resistors?
- Latches?

Glitches on state variables in AFSMs

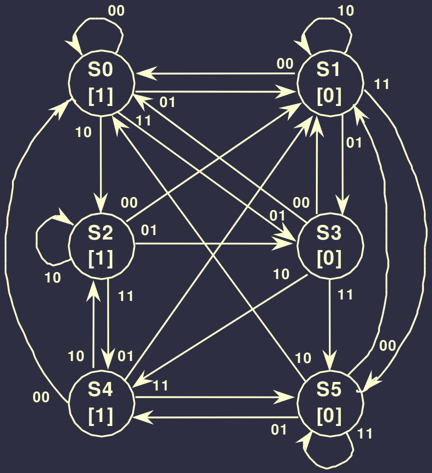
- Can uses latches in similar way
- Additional advantage: Separate sequential and combinational logic. feedback requirements reduced
- Disadvantage: May require more logic
- Consider the example

AFSM design example

- Design a two–input machine (LM)
    - Output 1 iff L is low and M was high at some time during most recent L high period
    - Output 0 otherwise
    - Let's build two AFSMs to solve this problem
    - One will use global feedback
    - One will use RS latches

Multiple input example

Initial multiple input FSM state diagram

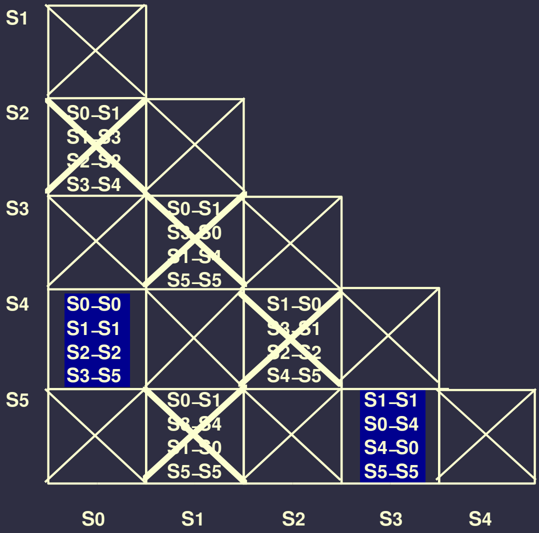


State Diagram

State table

Present State	Next State 00	01	10	11	Output
S_0	S_0	S_1	S_2	S_3	1
S_1	S_0	S_3	S_1	S_5	0
S_2	S_1	S_3	S_2	S_4	1
S_3	S_1	S_0	S_4	S_5	0
S_4	S_0	S_1	S_2	S_5	1
S_5	S_1	S_4	S_0	S_5	0

Implication chart



Simplified state table

Present State	Next State 00	01	10	11	Output
S_6	S_6	S_1	S_2	S_7	1
S_1	S_6	S_7	S_1	S_7	0
S_2	S_1	S_7	S_2	S_6	1
S_7	S_1	S_6	S_6	S_7	0