
Computer Science 141

Computing Hardware

Fall 2009

Harvard University

Instructor: Prof. David Brooks

dbrooks@eecs.harvard.edu

Today's Lecture

- FSM minimization
 - One more FSM example
- Next Time Implementing Digital Designs
 - How to implement these designs in different technologies?
 - Full-Custom and Standard Cell Design
 - FPGA-based designs (e.g. CS141 labs)

FSM optimization

- try to minimize
 - number of states — fewer FFs
 - logic for next state/output calculation — faster clock rate
- important for Programmable Logic Device (PLD) design
- optimization based on fact that 2 FSMs equivalent if output behavior identical for all input sequences

Basic design approach

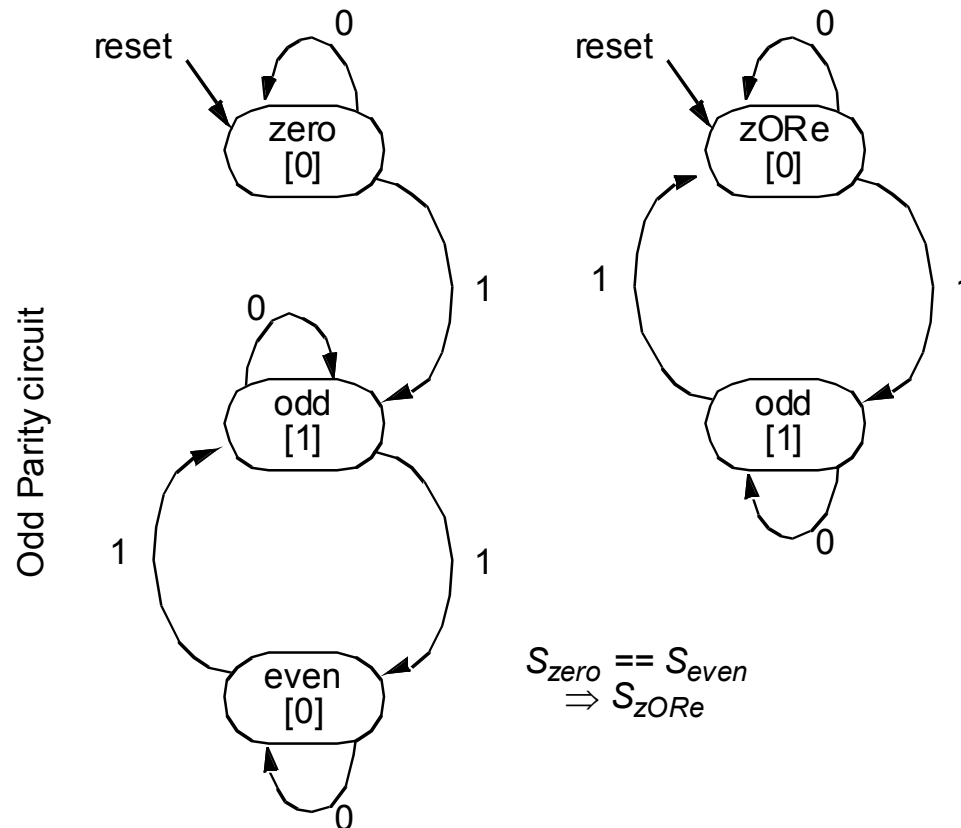
1. understand the problem
 - make assumptions to complete design specification
 - identify inputs and outputs
 - draw a block diagram of FSM
 - enumerate possible inputs sequences and system states
2. obtain abstract representation of FSM
 - draw state (transition) diagram
 - or alternative state machine representation
3. perform state minimization (*new step*)

Basic design approach

4. perform state assignment
 - encode states
 - build state transition table.
5. choose FF type
 - remap next state into required FF inputs
6. implement
 - simplify next state and output logic equations
 - wire the circuit together

State minimization/reduction

- two states equivalent if
 - output behavior identical for all inputs
 - next states same or equivalent for all inputs



Techniques for state minimization

1. row matching

- good for pencil-and-paper approach
- not always obtain best solution

2. implication charts

- very algorithmic and more complex
- guarantee to find best solution

Example using row-matching method

- systematic approach to finding equivalent states
 - build (slightly strange) state transition table
 - 1 row per present state
 - next states organized horizontally by inputs
 - outputs organized horizontally by inputs
 - group any rows with identical next state and output values into a single new state
 - replace old equivalent state names with new single state name
 - iterate

example continued

Input Sequence	Present State	Next State		Output	
		X=0	X=1	X=0	X=1
reset	S0	S1	S2	0	0
0	S1	S3	S4	0	0
1	S2	S5	S6	0	0
00	S3	S7	S8	0	0
01	S4	S9	S10	0	0
10	S5	S11	S12	0	0
11	S6	S13	S14	0	0
000	S7	S0	S0	0	0
001	S8	S0	S0	0	0
010	S9	S0	S0	0	0
011	S10	S0	S0	1	0
100	S11	S0	S0	0	0
101	S12	S0	S0	1	0
110	S13	S0	S0	0	0
111	S14	S0	S0	0	0

- S10 identical to S12 \Rightarrow S10'
- update table and continue

example continued

Input Sequence	Present State	Next State		Output	
		X=0	X=1	X=0	X=1
reset	S0	S1	S2	0	0
0	S1	S3	S4	0	0
1	S2	S5	S6	0	0
00	S3	S7	S8	0	0
01	S4	S9	S10'	0	0
10	S5	S11	S10'	0	0
11	S6	S13	S14	0	0
000	S7	S0	S0	0	0
001	S8	S0	S0	0	0
010	S9	S0	S0	0	0
011/101	S10'	S0	S0	1	0
100	S11	S0	S0	0	0
110	S13	S0	S0	0	0
111	S14	S0	S0	0	0

- S7, S8, S9, S11, S13, S14 identical \Rightarrow S7'
- update table and continue

example continued

Input Sequence	Present State	Next State		Output	
		X=0	X=1	X=0	X=1
reset	S0	S1	S2	0	0
0	S1	S3	S4	0	0
1	S2	S5	S6	0	0
00	S3	S7'	S7'	0	0
01	S4	S7'	S10'	0	0
10	S5	S7'	S10'	0	0
11	S6	S7'	S7'	0	0
!(011/101)	S7'	S0	S0	0	0
011/101	S10'	S0	S0	1	0

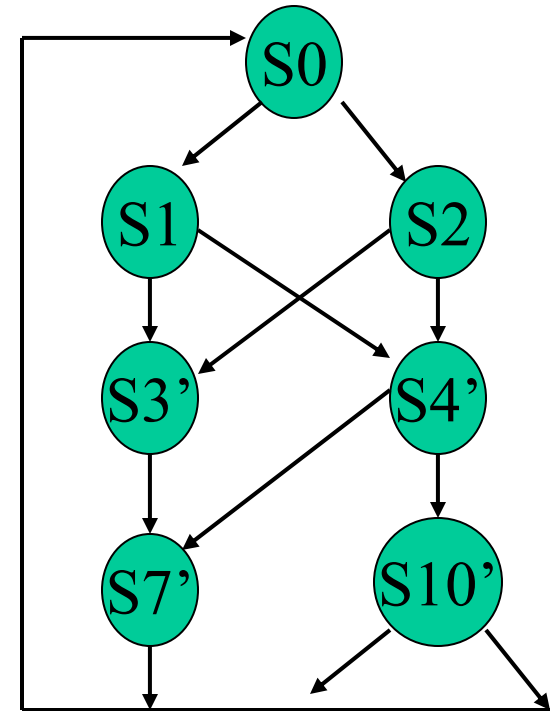
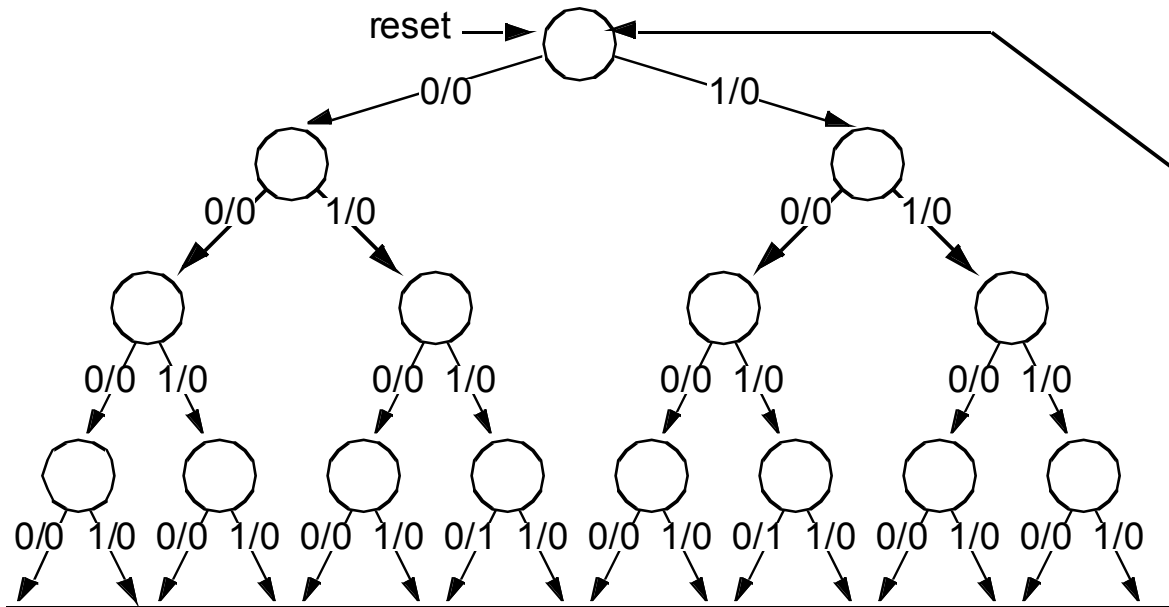
- S3 and S6 identical \Rightarrow S3' S4 and S5 identical \Rightarrow S4'
- update table and continue

example continued

Input Sequence	Present State	Next State		Output	
		X=0	X=1	X=0	X=1
reset	S0	S1	S2	0	0
0	S1	S3'	S4'	0	0
1	S2	S4'	S3'	0	0
00/11	S3'	S7'	S7'	0	0
01/10	S4'	S7'	S10'	0	0
!(011/101)	S7'	S0	S0	0	0
011/101	S10'	S0	S0	1	0

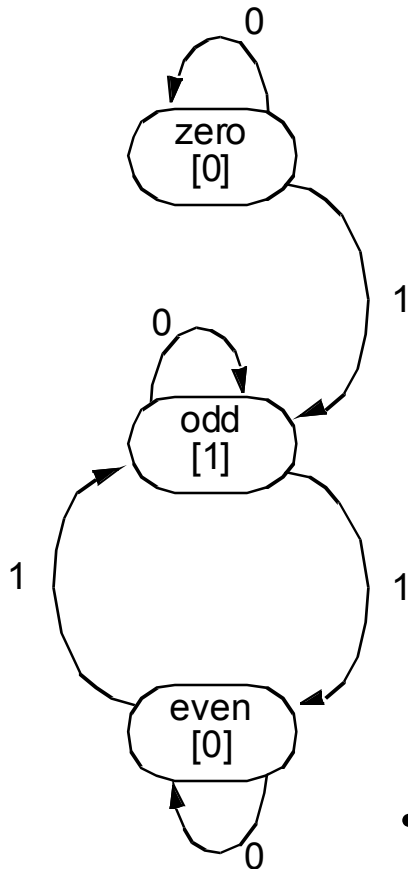
- Done! No more combining possible

Example: final state diagram



limitations of row-matching method

- self loops present a combining problem

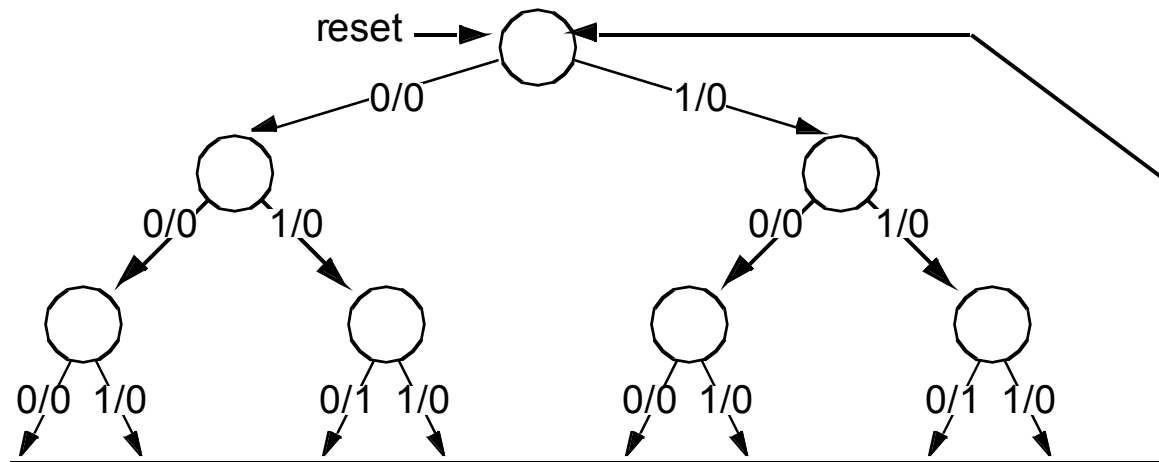


Present State	Next State		Output
	X=0	X=1	
zero	zero	odd	0
odd	odd	even	1
even	even	odd	0

- S_{zero} not identical to S_{even} under row matching

Example using implication chart method

- problem: build another sequence-detecting FSM
 - detects the 3-bit sequence 010 or 110
 - machine reset after each 3-bit sequence
 - 1 input, 1 output — output 1 only after sequence detected
 - Mealy machine state diagram



Example using implication chart method

- systematic approach to finding equivalent states
 - construct implication chart
 - 1 square for each combination of 2 states
 - if outputs of states different, “X” square
 - ⇒ states cannot be equivalent
 - otherwise, fill square with next state pairs for all input combinations
 - systematically advance through squares
 - for each next state pair in square, check that square for “X”
 - if “X”-ed, this square “X”-ed
 - ⇒ if next states not equivalent, then these states not equivalent
 - iterate until no changes
 - remaining unmarked squares represent equivalent states

state transition table

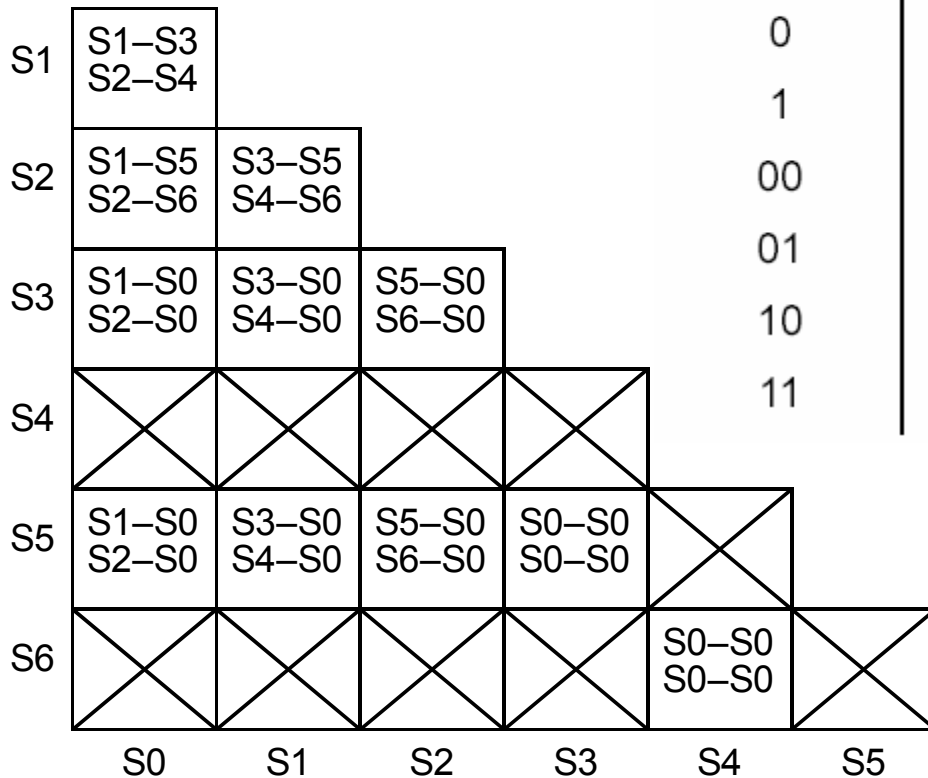
Input Sequence	Present State	Next State		Output	
		X=0	X=1	X=0	X=1
reset	S0	S1	S2	0	0
0	S1	S3	S4	0	0
1	S2	S5	S6	0	0
00	S3	S0	S0	0	0
01	S4	S0	S0	1	0
10	S5	S0	S0	0	0
11	S6	S0	S0	1	0

an empty implicant chart

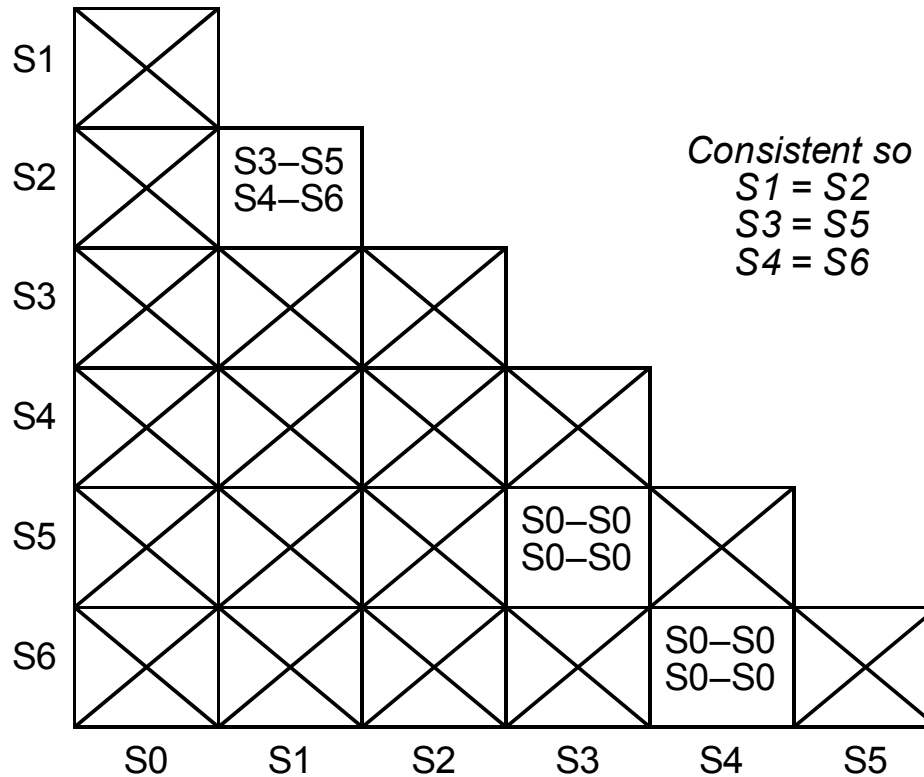
	Input		Present		Next State		Output	
	Sequence		State		X=0	X=1	X=0	X=1
	reset		S0		S1	S2	0	0
S1	0		S1		S3	S4	0	0
	1		S2		S5	S6	0	0
S2	00		S3		S0	S0	0	0
	01		S4		S0	S0	1	0
S3	10		S5		S0	S0	0	0
	11		S6		S0	S0	1	0
S4								
S5								
S6								
			S0	S1	S2	S3	S4	S5

the initial implicant chart

Input Sequence	Present State	Next State		Output	
		X=0	X=1	X=0	X=1
reset	S0	S1	S2	0	0
0	S1	S3	S4	0	0
1	S2	S5	S6	0	0
00	S3	S0	S0	0	0
01	S4	S0	S0	1	0
10	S5	S0	S0	0	0
11	S6	S0	S0	1	0



results of first marking pass



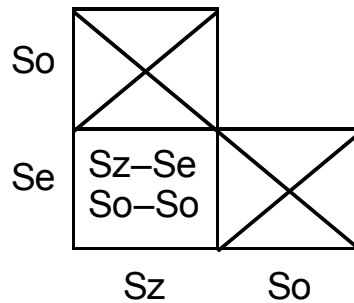
Back to the state transition table

Input Sequence	Present State	Next State		Output	
		X=0	X=1	X=0	X=1
reset	S0	S1	S2	0	0
0	S1	S3	S4	0	0
1	S2	S5	S6	0	0
00	S3	S0	S0	0	0
01	S4	S0	S0	1	0
10	S5	S0	S0	0	0
11	S6	S0	S0	1	0

$S1=S2, S3=S5, S4=S6$

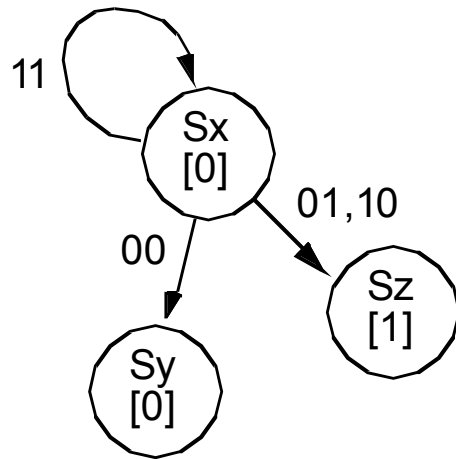
implicant chart method works for self loops

Present State	Next State		Output
	X=0	X=1	
zero	zero	odd	0
odd	odd	even	1
even	even	odd	0



Sz equivalent to Se!

Multiple inputs

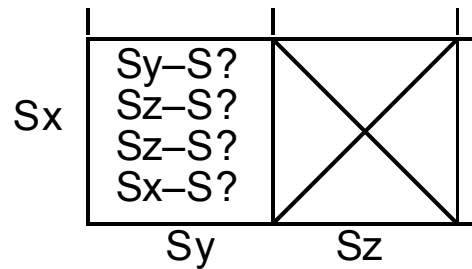


Partial State Diagram

Present State	Next State				Output
	00	01	10	11	
...
S_x	S_y	S_z	S_z	S_x	0
...

Partial State Table

Partial Implicant Chart



State assignment

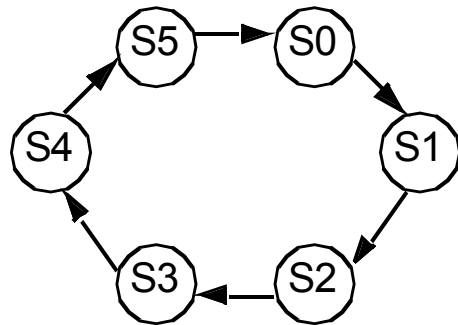
- next state and output logic depends strongly on how states are encoded
- only known way to solve this problem is through exhaustive search
- problem: search spaces grows quickly ($n!$ where n is number of states)
- heuristics developed:
 - minimum bit change
 - one hot encodings
 - etc.
- number of computer tools available for exploring this search space

State Assignment Heuristics

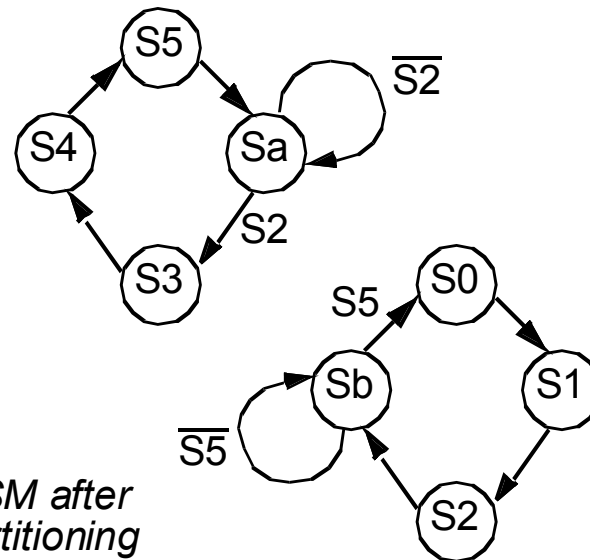
- Choose an initial state that is easy to reset (00...00 or 11...11)
- Minimize number of state variables that change on each transition
- Find groups of related states (where most transitions stay in the group) → then maximize number of state variables that don't change in these groups
- If unused state bits, utilize these bits to achieve the above goals

FSM partitioning

- useful technique for PLDs
- key idea:
 - introduction of idle states
 - represents hand-off of control between partial FSMs



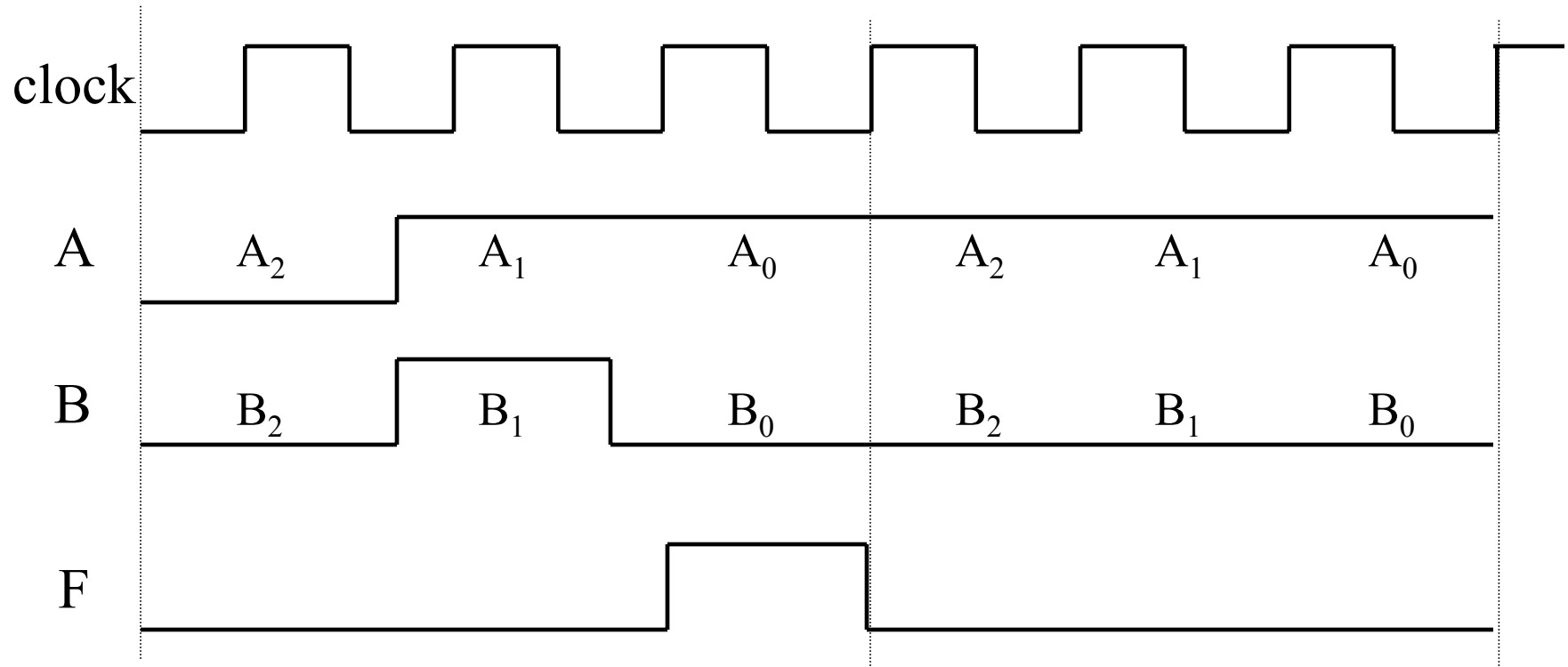
FSM before partitioning



FSM after partitioning

Design Problem

- Design a circuit which serially accepts two 3-bit two's complement numbers with the sign-bit delivered first. If input A is greater than input B, assert output after reading 3rd bit. The circuit continually samples the input lines comparing 3-bit numbers.



State Transition Diagram (Moore)

State Transition Table for state minimization

	Next State				
Present State	00	01	10	11	Output
0					
1					
2					
3					
4					
5					
6					
7					
8					
9					

	Next State				
Present State	00	01	10	11	Output
0					
1					
2					
3					
4					
5					
6					
7					

State Encoding

- Heuristic: : Minimize Bit Changes

S0 =

S4 =

S1 =

S5 =

S2 =

S6 =

S3 =

S7 =

State Transition Table

	AB		
Present State	Input	Next State	Output
