Digital Logic Design

4190.201

2014 Spring Semester

9. Working with Finite State Machines

Naehyuck Chang
Dept. of CSE
Seoul National University
naehyuck@snu.ac.kr

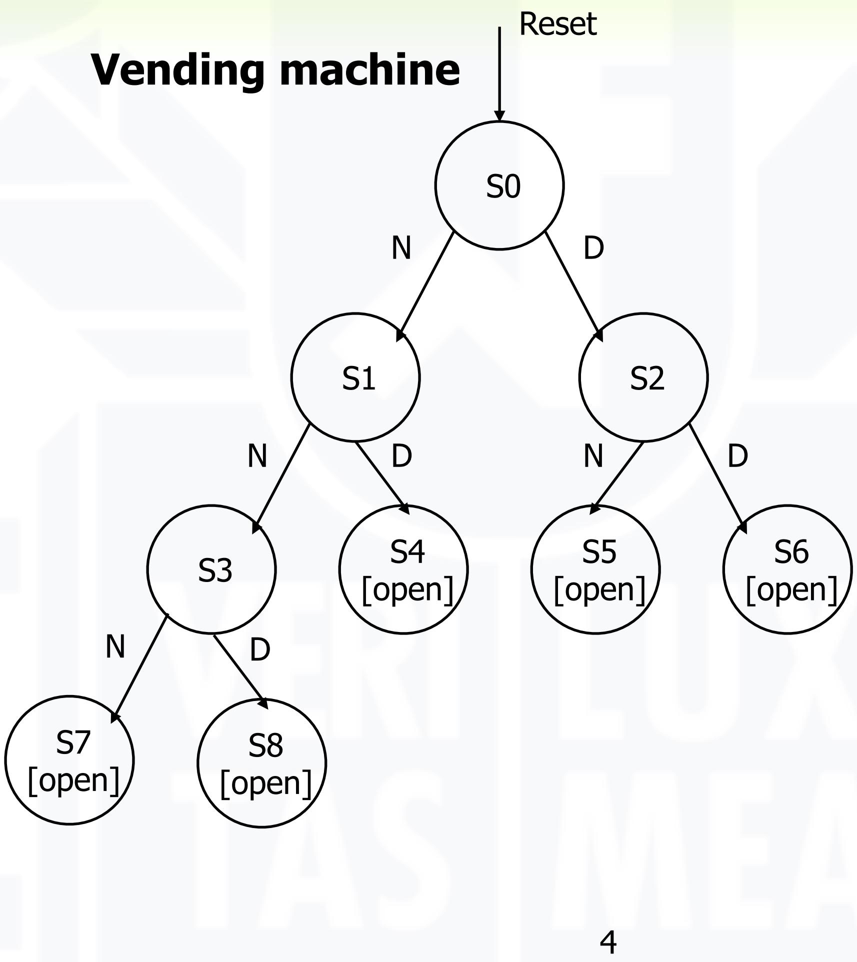
Finite state machine optimization

- State minimization
  - Fewer states require fewer state bits
  - Fewer bits require fewer logic equations
- Encodings: state, inputs, outputs
  - State encoding with fewer bits has fewer equations to implement
    - However, each may be more complex
  - State encoding with more bits (e.g., one-hot) has simpler equations
    - Complexity directly related to complexity of state diagram
  - Input/output encoding may or may not be under designer control
- Combinational logic optimization
  - Next state forming logic
  - Output forming logic

Algorithmic approach to state minimization

- State minimization/reduction
  - Goal – identify and combine states that have equivalent behavior
- Equivalent states:
  - Same output
  - For all input combinations, states transition to same or equivalent states
- Algorithm with the symbolic state transition table
  - Group states together with the same output
  - Examine transitions that have the same next state for every input combination
  - Row matching and implicant charts

Vending machine

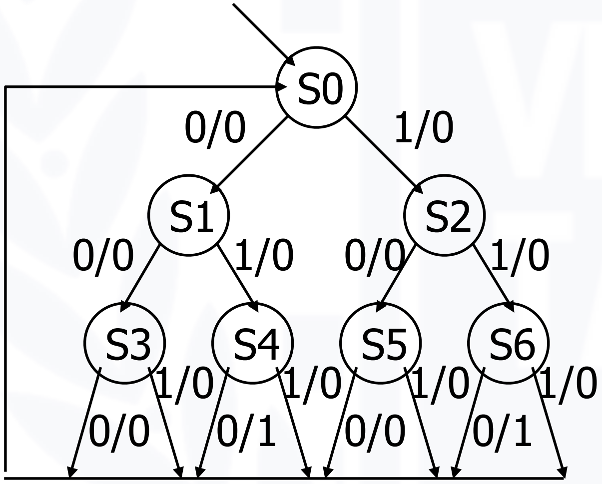


Embedded Low-Power Laboratory

State minimization example

- Sequence detector for 010 or 110

Input Sequence	Present State	Next State X=0	Next State X=1	Output X=0	Output 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



Method of successive partitions

Input Sequence	Present State	Next State X=0	Next State X=1	Output X=0	Output 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

(S0 S1 S2 S3 S4 S5 S6)

(S0 S1 S2 S3 S5) (S4 S6) S4 is equivalent to S6

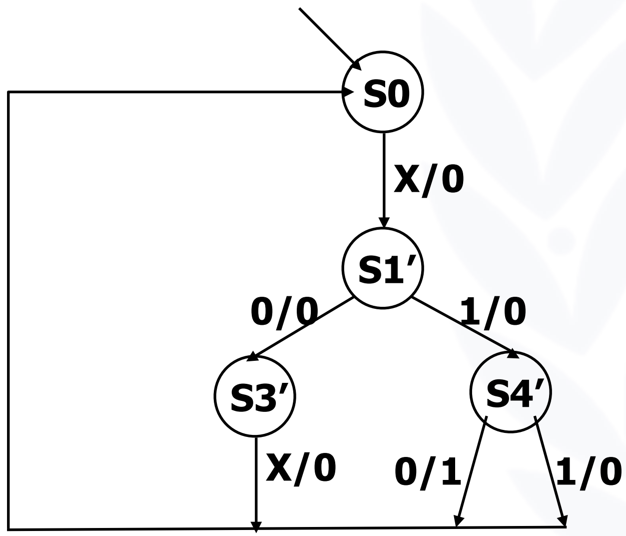
(S0 S3 S5) (S1 S2) (S4 S6) S3 is equivalent to S5

(S0) (S3 S5) (S1 S2) (S4 S6) S1 is equivalent to S2

Minimized FSM

- State minimized sequence detector for 010 or 110

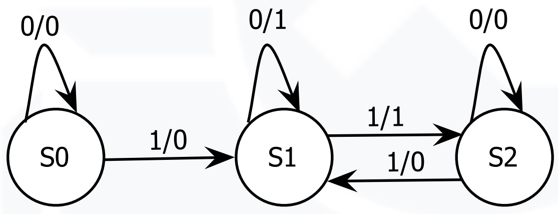
Input Sequence	Present State	Next State X=0	Next State X=1	Output X=0	Output X=1
Reset	S0	S1'	S1'	0	0
0 + 1	S1'	S3'	S4'	0	0
X0	S3'	S0	S0	0	0
X1	S4'	S0	S0	1	0



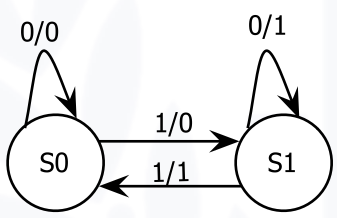
Minimized FSM

- Row matching does not always yield the most-reduced state table
- Counter example
  - Odd parity checker
  - S0 and S2 cannot be merged by row matching due to the self loop transitions on input 0

Present state	Next state		Output
	X=0	X=1	
S0	S0	S1	0
S1	S1	S2	1
S2	S2	S1	0



Original



Reduced

Embedded Low-Power Laboratory

Minimized FSM

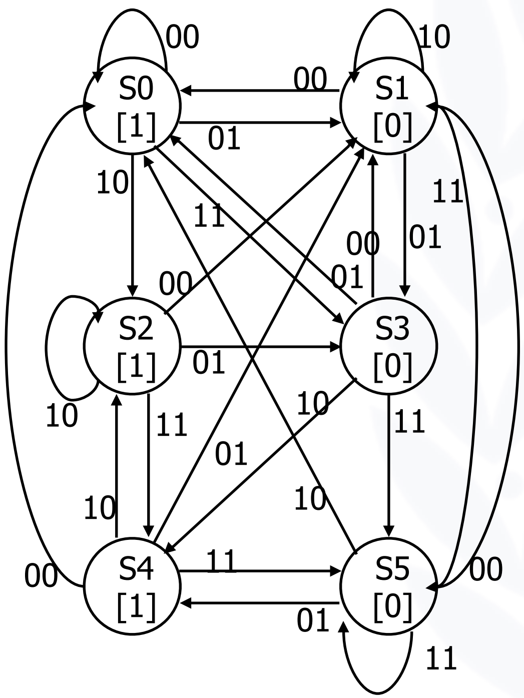
- Implicant chart method
  - More systematic approach to finding the state that can be combined into a single reduced state
  - More complex and better suited for machine implementation than hand use
  - (n2-n)/2 cells for n states
  - Xij: row is labeled by Si and column is labeled by Sj (Mark if Si, Sj can be combined)

S0	X							
S1		X						
S2			X					
S3				X				
S4					X			
S5						X		
S6							X	
S7								X
	S0	S1	S2	S3	S4	S5	S6	S7

S1							
S2							
S3							
S4							
S5							
S6							
S7							
	S0	S1	S2	S3	S4	S5	S6

More complex state minimization

- Multiple input example



Inputs here

present state	next state 00	01	10	11	output
S0	S0	S1	S2	S3	1
S1	S0	S3	S1	S4	0
S2	S1	S3	S2	S4	1
S3	S1	S0	S4	S5	0
S4	S0	S1	S2	S5	1
S5	S1	S4	S0	S5	0

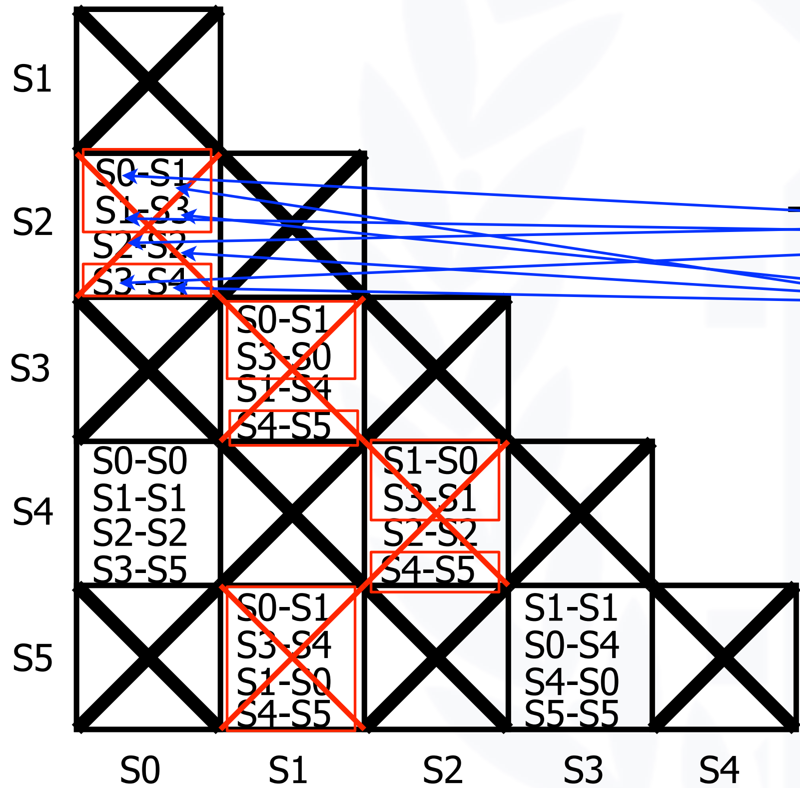
Symbolic state transition table

Embedded Low-Power Laboratory

Minimized FSM

- Implication chart method
  - Cross out incompatible states based on outputs (mark X if the output is different)
  - Complete the initial implicant chart
  - Cross out more cells if indexed chart entries are already crossed out

S0-S1
S0-S3
S0-S5
S1-S3
S1-S5
S2-S3
S2-S5
S3-S4
S4-S5



present state	next state 00	01	10	11	output
S0	S0	S1	S2	S3	1
S1	S0	S3	S1	S4	0
S2	S1	S3	S2	S4	1
S3	S1	S0	S4	S5	0
S4	S0	S1	S2	S5	1
S5	S1	S4	S0	S5	0

Equivalent states with don't cares

- Equivalence of states is transitive when machine is fully specified
  - a=b, b=c, then a=c
- But its not transitive when don't cares are present

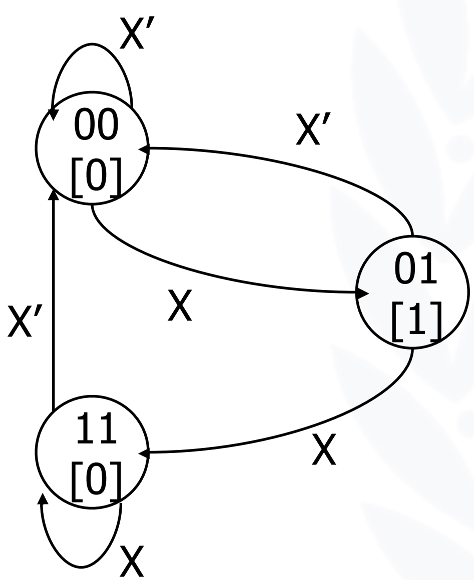
e.g.,

state	output	
S0	– 0	S1 is compatible with both S0 and S2
S1	1 –	but S0 and S2 are incompatible
S2	– 1	

- No polynomial time algorithm exists for determining best grouping of states into equivalent sets that will yield the smallest number of final states

Minimizing states may not yield best circuit

- Example: edge detector - outputs 1 when last two input changes from 0 to 1



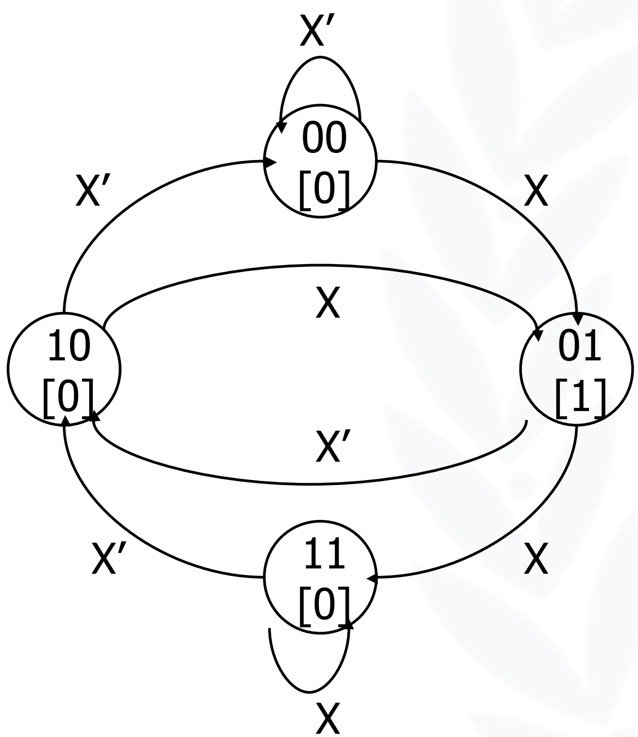
X	Q_1	Q_0	Q_1^+	Q_0^+
0	0	0	0	0
0	0	1	0	0
0	1	1	0	0
1	0	0	0	1
1	0	1	1	1
1	1	1	1	1
–	1	0	0	0

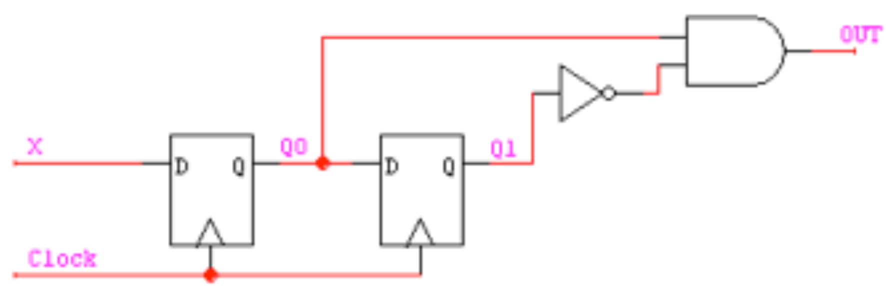
$$Q_1^+ = X \ (Q_1 \text{ xor } Q_0)$$

$$Q_0^+ = X \ Q_1' \, Q_0'$$

Another implementation of edge detector

- "Ad hoc" solution - not minimal but cheap and fast





State assignment

- Choose bit vectors to assign to each “symbolic” state
  - With n state bits for m states there are 2n! / (2n – m)!
    [log n <= m <= 2n]
  - 2n codes possible for 1st state, 2n–1 for 2nd, 2n–2 for 3rd, …
  - Huge number even for small values of n and m
    - Intractable for state machines of any size
    - Heuristics are necessary for practical solutions
  - Optimize some metric for the combinational logic
    - Size (amount of logic and number of FFs)
    - Speed (depth of logic and fanout)
    - Dependencies (decomposition)

State assignment strategies

- Possible strategies
  - Sequential – just number states as they appear in the state table
  - Random – pick random codes
  - One-hot – use as many state bits as there are states (bit=1 –> state)
  - Output – use outputs to help encode states
  - Heuristic – rules of thumb that seem to work in most cases
- No guarantee of optimality – another intractable problem

One-hot state assignment

- Simple
  - Easy to encode
  - Easy to debug
- Small logic functions
  - Each state function requires only predecessor state bits as input
- Good for programmable devices
  - Lots of flip-flops readily available
  - Simple functions with small support (signals its dependent upon)
- Impractical for large machines
  - Too many states require too many flip-flops
  - Decompose FSMs into smaller pieces that can be one-hot encoded
- Many slight variations to one-hot
  - One-hot + all-0

Heuristic methods

- CAD is preferred for good state encodings
  - Hand enumeration using trial and error becomes tedious
  - n-state FSM has n! different encodings even when densely encoded
- More tractable guidelines
  - Heuristics
    - To reduce the distance in Boolean n-space between related states
    - Encodings should differ by as few bits as possible
- State maps
  - Similar concept to K-maps

Output-based encoding

- Reuse outputs as state bits - use outputs to help distinguish states
  - Why create new functions for state bits when output can serve as well
  - Fits in nicely with synchronous Mealy implementations (Basic machine)

Inputs			Present State	Next State	Outputs		
C	TL	TS			ST	H	F
0	–	–	HG	HG	0	00	10
–	0	–	HG	HG	0	00	10
1	1	–	HG	HY	1	00	10
–	–	0	HY	HY	0	01	10
–	–	1	HY	FG	1	01	10
1	0	–	FG	FG	0	10	00
0	–	–	FG	FY	1	10	00
–	1	–	FG	FY	1	10	00
–	–	0	FY	FY	0	10	01
–	–	1	FY	HG	1	10	01

HG = ST′ H1′ H0′ F1 F0′ + ST H1 H0′ F1′ F0
HY = ST H1′ H0′ F1 F0′ + ST′ H1′ H0 F1 F0′
FG = ST H1′ H0 F1 F0′ + ST′ H1 H0′ F1′ F0′
HY = ST H1 H0′ F1′ F0′ + ST′ H1 H0′ F1′ F0

Output patterns are unique to states, we do not need ANY state bits – implement 5 functions (one for each output) instead of 7 (outputs plus 2 state bits)

Heuristic methods

- Comparison with two assignments
  - May results in significant bit change differences during state transition

State name	Assignment		
	Q2	Q1	Q0
S0	0	0	0
S1	1	0	1
S2	1	1	1
S3	0	1	0
S4	0	1	1

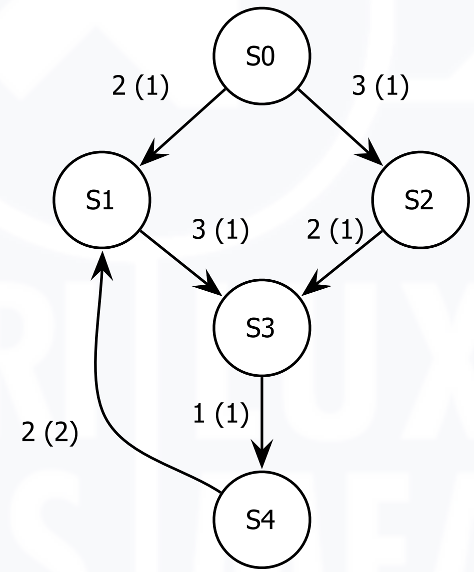
Assignment 1

Q2 \ Q1Q0	00	01	11	10
0	S0		S4	S3
1		S1	S2	

State name	Assignment		
	Q2	Q1	Q0
S0	0	0	0
S1	0	0	1
S2	0	1	0
S3	0	1	1
S4	0	1	1

Assignment 2

Q2 \ Q1Q0	00	01	11	10
0	S0	S1	S3	S2
1			S4	

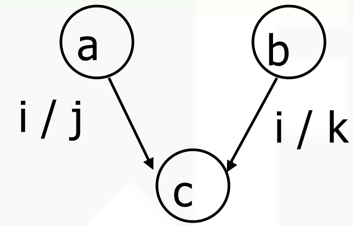


Heuristic methods

- Highest priority: adjacent codes to states that share a common next state

I	Q	Q^+	O
i	a	c	j
i	b	c	k

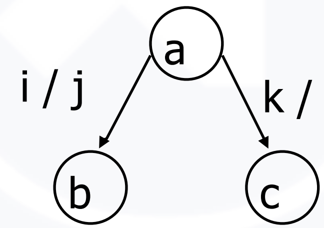
$$c = i * a + i * b$$



- Medium priority: adjacent codes to states that share a common ancestor state

I	Q	Q^+	O
i	a	b	j
k	a	c	l

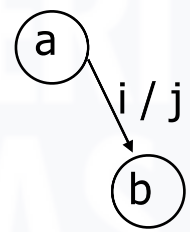
$$b = i * a$$
$$c = k * a$$

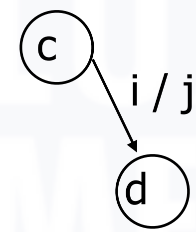


- Lowest priority: adjacent codes to states that have a common output behavior

I	Q	Q^+	O
i	a	b	j
i	c	d	j

$$j = i * a + i * c$$
$$b = i * a$$
$$d = i * c$$





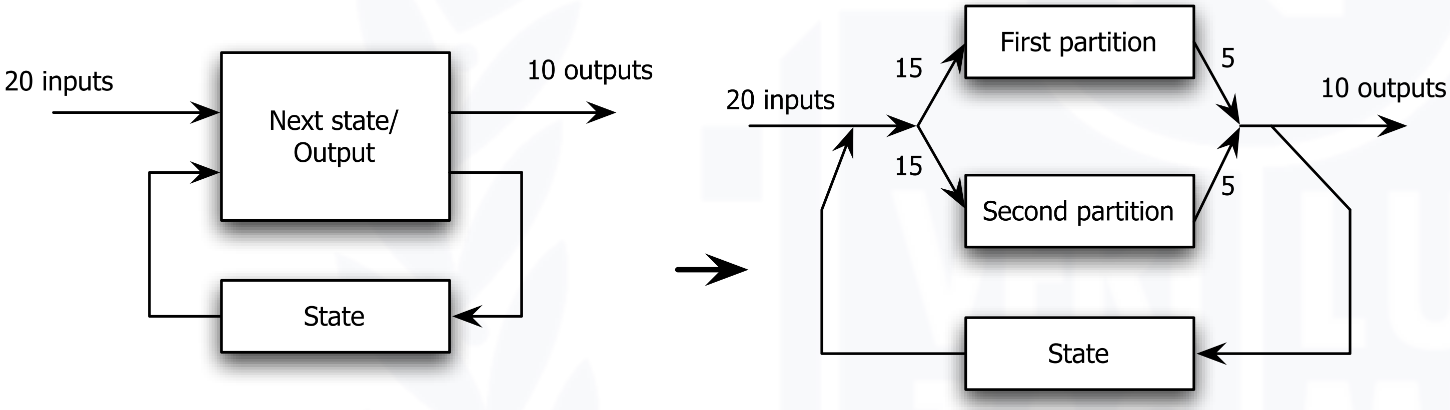
Embedded Low-Power Laboratory

Current state assignment approaches

- For tight encodings using close to the minimum number of state bits
  - Best of 10 random seems to be adequate (averages as well as heuristics)
  - Heuristic approaches are not even close to optimality
  - Used in custom chip design
- One-hot encoding
  - Easy for small state machines
  - Generates small equations with easy to estimate complexity
  - Common in FPGAs and other programmable logic
- Output-based encoding
  - Ad hoc - no tools
  - Most common approach taken by human designers
  - Yields very small circuits for most FSMs

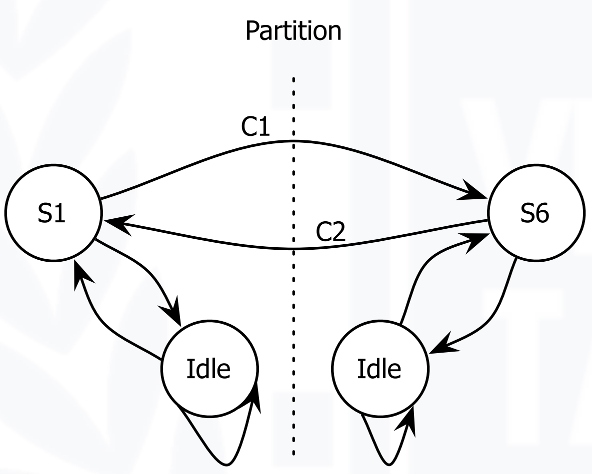
FSM partitioning

- Large-size FSM
  - Impossible to implement in a single programmable logic
  - Hard to optimize it due to the exponentially increasing complexity
- The following is not always possible



FSM partitioning

- FSM partitioning by introducing idle states
  - Additional idle or coordination states
  - When a state transition from S1 to S16 occurs
    - The left-hand partition is lost control
    - S1 is no longer active, but all the other states in the left-hand partition are not active, either
    - Should be in an idle state until a state transition from S6 to S1 occurs



Sequential logic optimization summary

- State minimization
  - Straightforward in fully-specified machines
  - Computationally intractable, in general (with don't cares)
- State assignment
  - Many heuristics
  - Best-of-10-random just as good or better for most machines
  - Output encoding can be attractive (especially for PAL implementations)