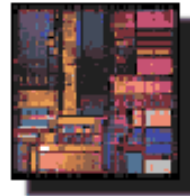


ECE 595Z
Digital VLSI Design Automation
Module 7 (Lectures 24-27): Sequential Logic
Optimization
Lecture 26



Anand Raghunathan
MSEE 318
raghunathan@purdue.edu

Summary

- Sequential logic minimization
 - State minimization
 - Completely specified FSMs
 - Identify and merge equivalent states
 - Efficient algorithm ($O(n \log n)$)
 - Incompletely specified FSMs
 - Identify minimum set of compatibles that is closed and complete
 - Problem is NP-hard [Pfleeger 1973]
 - State encoding
 - Combinational logic synthesis

State Encoding (a.k.a. State Assignment)

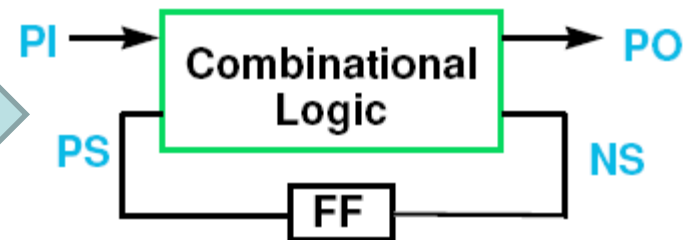
- Assign binary representation to “symbolic” states.
 - Defines the next state and output functions

Symbolic State Table

PI	PS	NS	PO
000	S0	S0	01
100	S0	S1	01
010	S0	S3	01
--1	S0	S0	10
100	S1	S1	01
0-0	S1	S0	01
110	S1	S2	01
--1	S1	S0	10
110	S2	S2	01
100	S2	S1	01
---	S3	S3	--

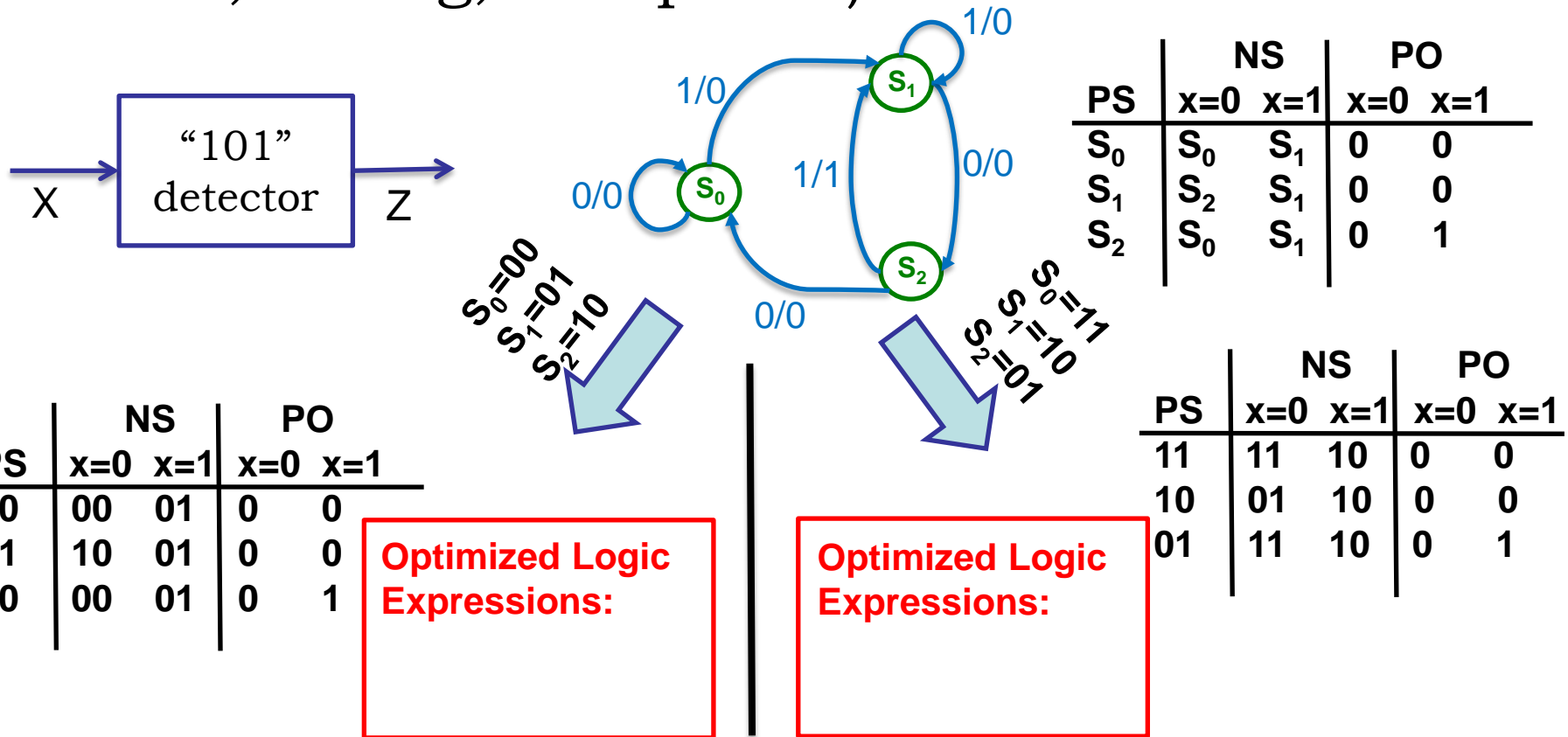
State Encoding

S0	01	S1	10
S2	00	S3	11



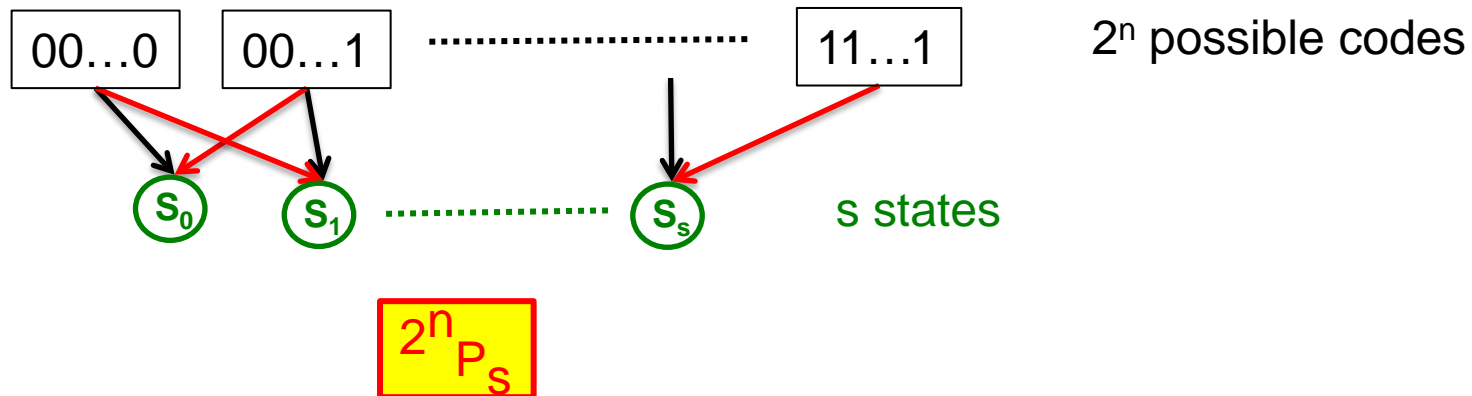
State Encoding: Example

- State encoding has a strong impact on the combinational logic complexity (and hence, area, timing, and power)




Complexity of State Encoding

- How many possible ways to encode an FSM that has s states using n bits?



- What if permutations of state bits are considered equivalent?

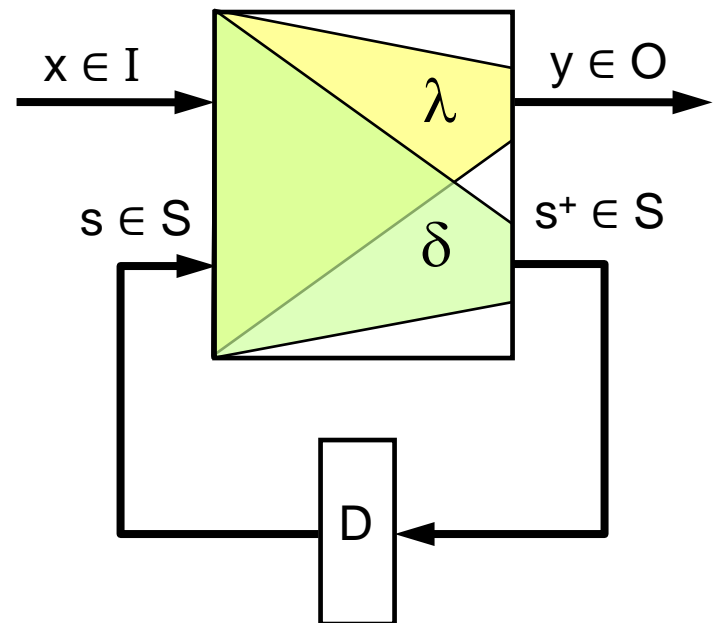
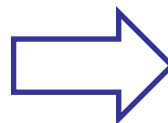
PS	NS		PO		↔	PS	NS		PO	
	x=0	x=1	x=0	x=1			x=0	x=1	x=0	x=1
00	00	01	0	0		00	00	10	0	0
01	10	01	0	0		10	01	10	0	0
10	00	01	0	1		01	00	10	0	1



State Encoding to Minimize Combinational Logic Complexity

- **Key idea:** Perform encoding so as to create opportunities for logic minimization in the next state and output functions
 - Techniques differ depending on whether target implementation of next-state & output logic is two-level or multi-level

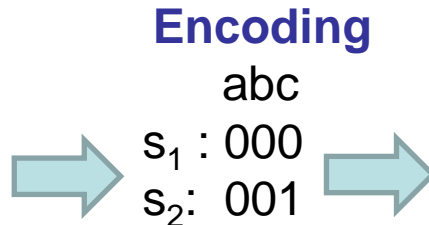
PI	PS	NS	PO
000	S0	S0	01
100	S0	S1	01
010	S0	S3	01
--1	S0	S0	10
100	S1	S1	01
0-0	S1	S0	01
110	S1	S2	01
--1	S1	S0	10
110	S2	S2	01
100	S2	S1	01
---	S3	S3	--



Guidelines for State Encoding

- States that have the same next state for the same input value should be given **adjacent** assignments

PS	NS		PO	
	x=0	x=1	x=0	x=1
...
s ₁	s ₃
...
s ₂	s ₃
...



Transition condition (s₃)

$$= x'a'b'c' + x'a'b'c$$

$$= x'a'b' (c'+c) = x'a'b'$$

Benefit: Potential for combining cubes in the next-state logic

$$NS(i) = \sum_{\text{all states } s_j \text{ with bit } i = 1} \text{Transition Condition}(s_j)$$

- Same applies for states that have the same output for the same input value

Guidelines for State Encoding

- States that have the same next state (for any input value) should be given **adjacent** assignments

PS	NS		PO	
	x=0	x=1	x=0	x=1
...
s₁	s₃
...
s₂	...	s₃
...

Encoding

abc
 $s_1 : 000$
 $s_2 : 001$

Transition condition (s_3)

$$\begin{aligned}
 &= x'a'b'c' + xa'b'c \\
 &= a'b'(x'c' + xc)
 \end{aligned}$$

Benefit: Potential for common factors in the next-state logic

What if we choose a different encoding?

abc
 $s_1 : 000$
 $s_2 : 111$

Transition condition (s_3)

$$= x'a'b'c' + xabc$$

abc
 $s_1 : 000$
 $s_2 : 011$

Transition condition (s_3)

$$\begin{aligned}
 &= x'a'b'c' + xa'bc \\
 &= a'(x'b'c' + xbc)
 \end{aligned}$$

Guidelines for State Encoding

- Next states that result from the same previous state should be given adjacent assignments

PS	NS		PO	
	x=0	x=1	x=0	x=1
...
s ₁	s ₂	s ₃
...
...
...

Encoding

abc
s₁ : 000
s₂ : 001
s₃ : 011

Transition condition (s₂)

$$= x'a'b'c' + \dots$$

Transition condition (s₃)

$$= xa'b'c' + \dots$$

$$c^+ = x'a'b'c' + \dots + xa'b'c' + \dots$$

a'b'c'

Benefit: Potential for combining cubes or common factors in the next-state logic

PS	NS		PO	
	x ₁ x ₂			
	00	11	00	11
...
s ₁	s ₂	s ₃
...
...
...

Transition condition (s₂)

$$= x_1'x_2'a'b'c' + \dots$$

Transition condition (s₃)

$$= x_1x_2a'b'c' + \dots$$

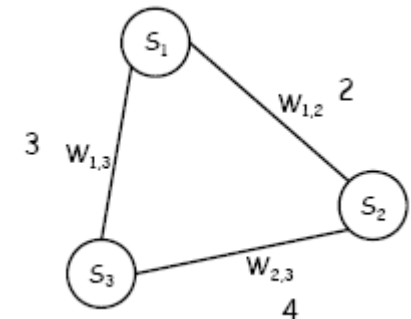
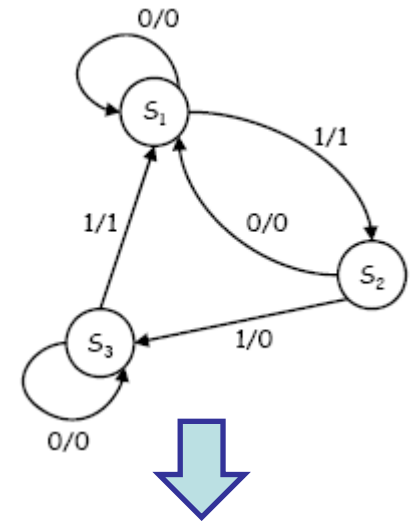
$$c^+ = x_1'x_2'a'b'c' + \dots +$$

$$x_1x_2a'b'c' + \dots$$

$$a'b'c'(x_1'x_2' + x_1x_2)$$

State Encoding Algorithm

- **General approach:** Construct a complete graph with nodes representing states, and weighted edges representing “affinity”
 - $\text{Affinity}(s_i, s_j)$ should reflect the potential benefit of assigning adjacent codes to states s_i and s_j
 - Label the vertices of the graph based on the edge weights



Two different approaches to computing edge weights – fanout-oriented and fanin-oriented

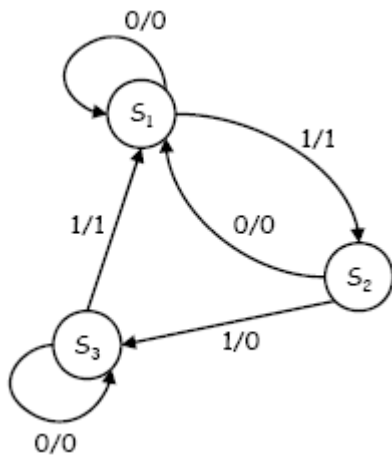
S. Devadas, et al, “MUSTANG: state assignment of finite state machines for optimal multi-level logic implementations,” IEEE Transactions on Computer-Aided Design, Dec. 1988.

State Encoding Algorithm: Computing Edge Weights (Fanout-Oriented)

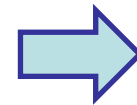
- **Fanout-oriented heuristic:** Present states that result in similar outputs and produce similar sets of next states are given high affinity
 - Intuition: Maximize the size of the most commonly occurring cube factors in the next-state and output logic

Next state set: Matrix that captures how often a (PS,NS) pair occurs

Output set: How often an output bit is asserted in each PS



PS (y_1y_2)	NS (Y_1Y_2)		PO (z)	
	$x=0$	$x=1$	$x=0$	$x=1$
S_1	S_1	S_2	0	1
S_2	S_1	S_3	0	0
S_3	S_3	S_1	0	1



$$NS_SET = \begin{pmatrix} & S_1^n & S_2^n & S_3^n \\ S_1^p & 1 & 1 & 0 \\ S_2^p & 1 & 0 & 1 \\ S_3^p & 1 & 0 & 1 \end{pmatrix}$$

$$OUT_SET = \begin{pmatrix} & z \\ S_1^p & 1 \\ S_2^p & 0 \\ S_3^p & 1 \end{pmatrix}$$

State Encoding Algorithms: Computing Edge Weights (Fanout-Oriented)

- Formula to compute edge weights

$$W_{i,j} = \frac{N_b}{2} \cdot NS_SET(i) \cdot NS_SET(j)^T + OUT_SET(i) \cdot OUT_SET(j)^T$$

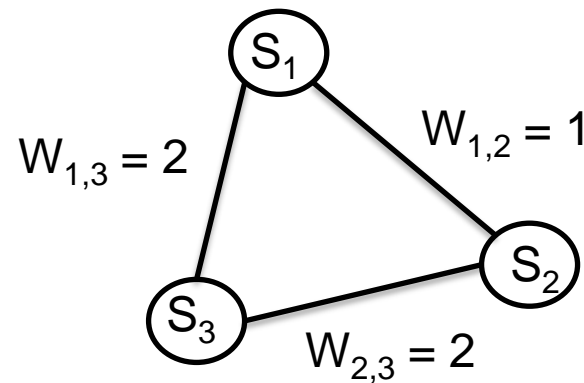
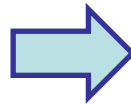
N_b : # of encoding bits

$NS_SET(i)$: i^{th} row of NS_SET matrix

$OUT_SET(i)$: i^{th} row of OUT_SET matrix

$$NS_SET = \begin{pmatrix} S_1^n & S_2^n & S_3^n \\ S_1^p & 1 & 1 & 0 \\ S_2^p & 1 & 0 & 1 \\ S_3^p & 1 & 0 & 1 \end{pmatrix}$$

$$OUT_SET = \begin{pmatrix} z \\ S_1^p & 1 \\ S_2^p & 0 \\ S_3^p & 1 \end{pmatrix}$$



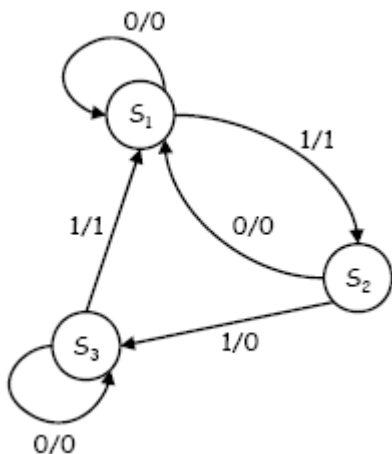
$$W_{2,3} = 1 \cdot [1 \ 0 \ 1][1 \ 0 \ 1]^T + [0][1]^T = 2$$

State Encoding Algorithm: Computing Edge Weights (Fanin-Oriented)

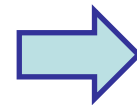
- **Fanin-oriented heuristic:** Next states that are produced by similar inputs and similar sets of present states are given high affinity

Present state set: Matrix that captures how often a (NS,PS) pair occurs

Input set: How often a next state is caused for each input value



PS (y ₁ y ₂)	NS (Y ₁ Y ₂)		PO (z)	
	x=0	x=1	x=0	x=1
S ₁	S ₁	S ₂	0	1
S ₂	S ₁	S ₃	0	0
S ₃	S ₃	S ₁	0	1



$$PS_SET = \begin{pmatrix} & S_1^p & S_2^p & S_3^p \\ S_1^n & 1 & 1 & 1 \\ S_2^n & 1 & 0 & 0 \\ S_3^n & 0 & 1 & 1 \end{pmatrix}$$

$$IN_SET = \begin{pmatrix} & x & x' \\ S_1^n & 1 & 2 \\ S_2^n & 1 & 0 \\ S_3^n & 1 & 1 \end{pmatrix}$$

State Encoding Algorithm: Computing Edge Weights (Fanin-Oriented)

- Formula to compute edge weights

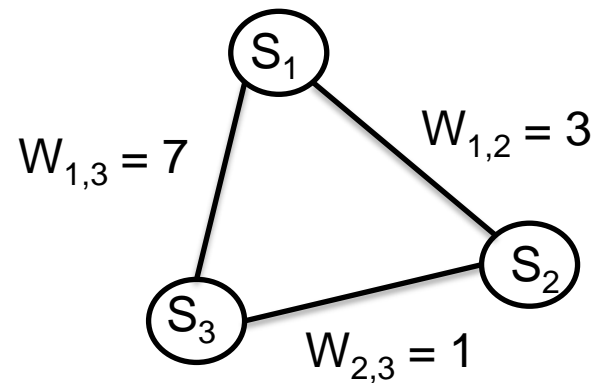
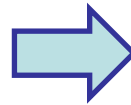
$$W_{i,j} = N_b \cdot PS_SET(i) \cdot PS_SET(j)^T + IN_SET(i) \cdot IN_SET(j)^T$$

N_b : # of encoding bits

$NS_SET(i)$: i^{th} row of NS_SET matrix

$OUT_SET(i)$: i^{th} row of OUT_SET matrix

$$PS_SET = \begin{pmatrix} & S_1^p & S_2^p & S_3^p \\ S_1^n & 1 & 1 & 1 \\ S_2^n & 1 & 0 & 0 \\ S_3^n & 0 & 1 & 1 \end{pmatrix}$$



$$IN_SET = \begin{pmatrix} x & x' \\ S_1^n & 1 & 2 \\ S_2^n & 1 & 0 \\ S_3^n & 1 & 1 \end{pmatrix}$$

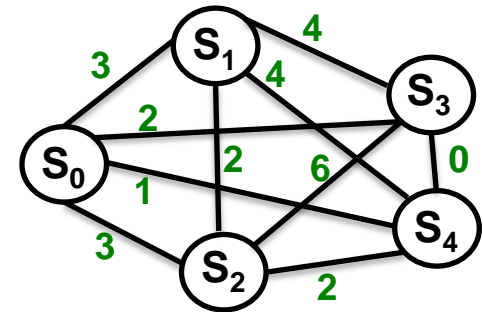
$$W_{1,3} = 2 \cdot [1 \ 1 \ 1][0 \ 1 \ 1]^T + [1 \ 2][1 \ 1]^T = 7$$

State Encoding Algorithm

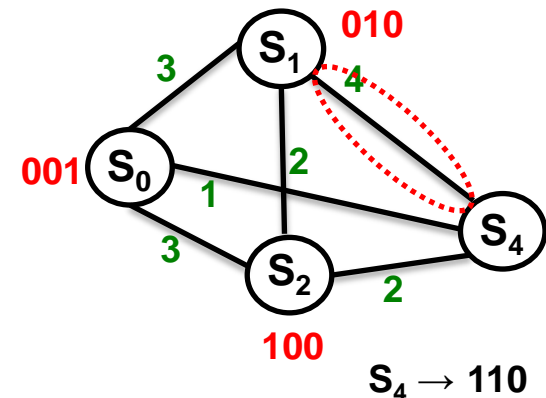
- **Algorithm for computing state encoding**

1. Select the state for which sum of weights of N_b heaviest incident edges is maximum
2. Arbitrarily assign a code to it and assign adjacent codes to N_b adjacent states
 - If some adjacent states have already been assigned codes, consider them when assigning a code to the selected state
3. Remove the state and edges selected in step 1 from the graph
4. Go to 1 and repeat, until graph is empty

- How well does this work in practice?
 - 30-40% lower literal count in the combinational logic (after multi-level optimization) compared to random state encoding



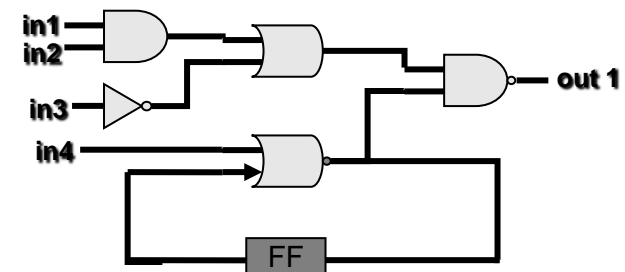
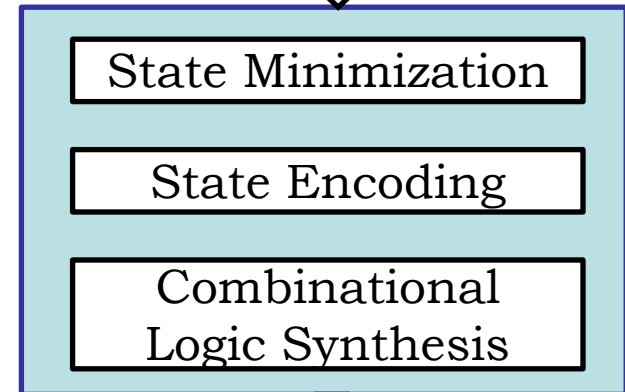
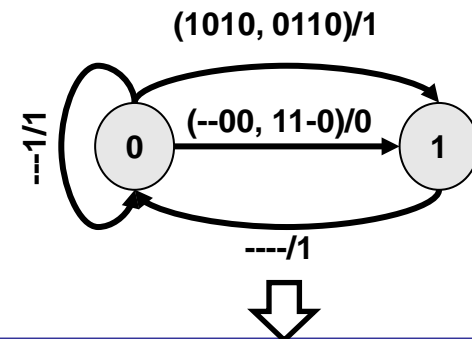
$N_b = 3$
 Pick S_3 (6+2+4)
 $S_3 \rightarrow 000$
 $S_0 \rightarrow 001$
 $S_1 \rightarrow 010$
 $S_2 \rightarrow 100$



S. Devadas, et al, "MUSTANG: state assignment of finite state machines for optimal multi-level logic implementations," IEEE Transactions on Computer-Aided Design, Dec. 1988.

Summary: FSM synthesis

- State minimization
 - Completely specified FSMs: equivalent states
 - Incompletely specified: compatible states
- State encoding
 - Create opportunities for two-level and multi-level minimization algorithms to optimize the next state and output logic
- FSM-based synthesis is usually used only for control logic



Optimizing Structural Representations of Sequential Networks

Limitations of FSM synthesis

- FSM representation is too large for most circuits
 - Only parts of the design (e.g., control logic) with small state spaces can be represented as an FSM
 - Data-paths have HUGE state spaces
- Two key advances have extended the scale of FSMs that can be handled
 - Implicit representations (BDDs)
 - Network of interacting FSMs
- Even with these advances, FSM synthesis is not applicable to large circuits (> 1000s of FFs)

Structural Approaches to Sequential Circuit Optimization

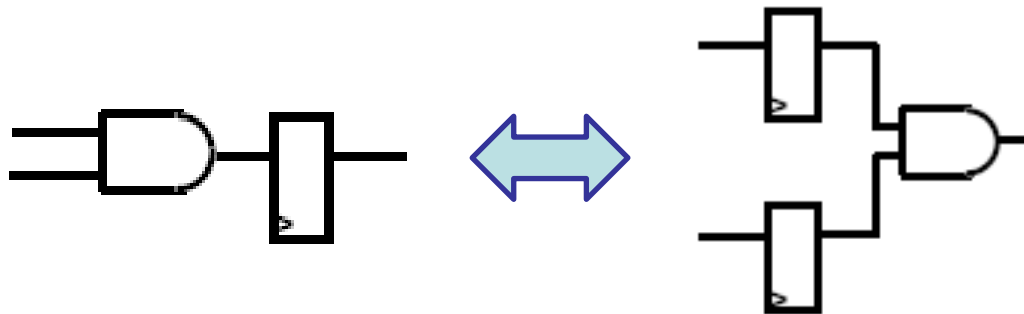
- Optimize combinational logic using sequential Don't Cares
- **Retiming**
- **Retiming & Re-synthesis**

Retiming

- Recall De Morgan's law?
 - Moving “bubbles” across gates



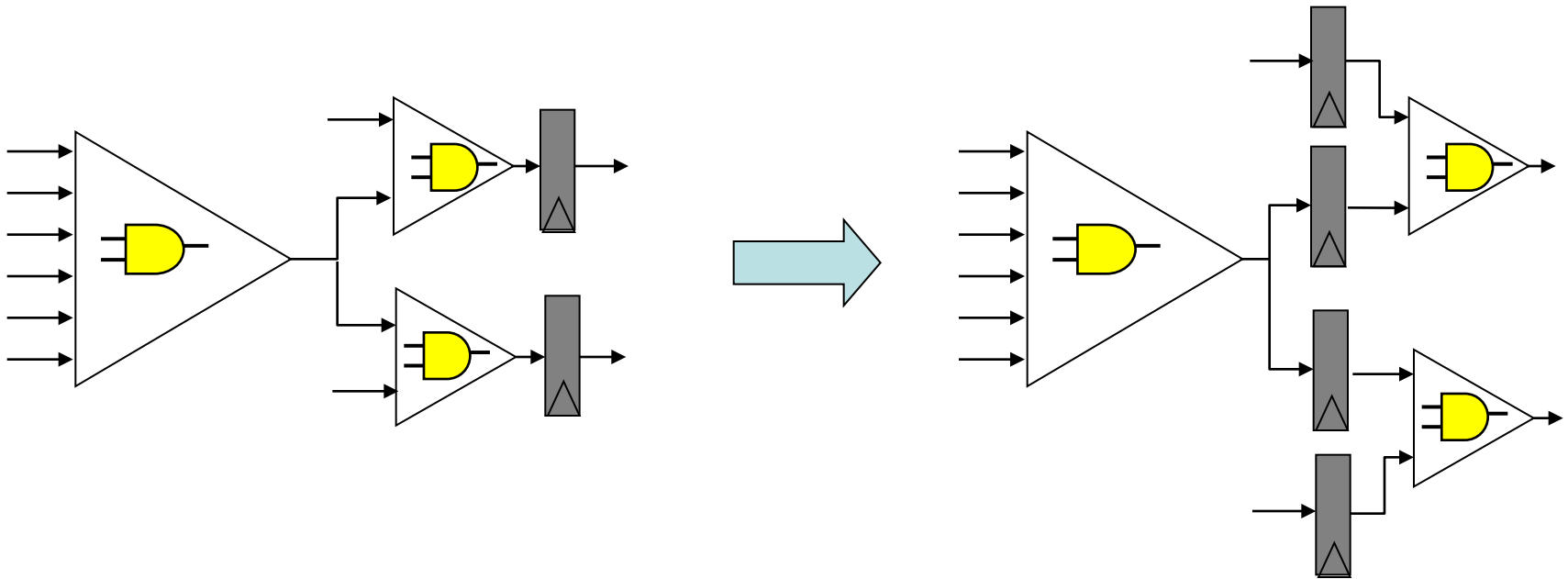
- It turns out you can do the same thing with flip-flops!
 - Does not change I/O behavior



C. E. Leiserson, F. M. Rose, and J. B. Saxe, “Optimizing synchronous circuitry by retiming,” Proc. 3rd Caltech Conf. on VLSI, 1983.

Retiming: Why?

- Re-position the flip-flops in the circuit to more “optimal” points
 - Increase the clock frequency
 - Reduce the number of registers
 - ...



Retiming: Example

