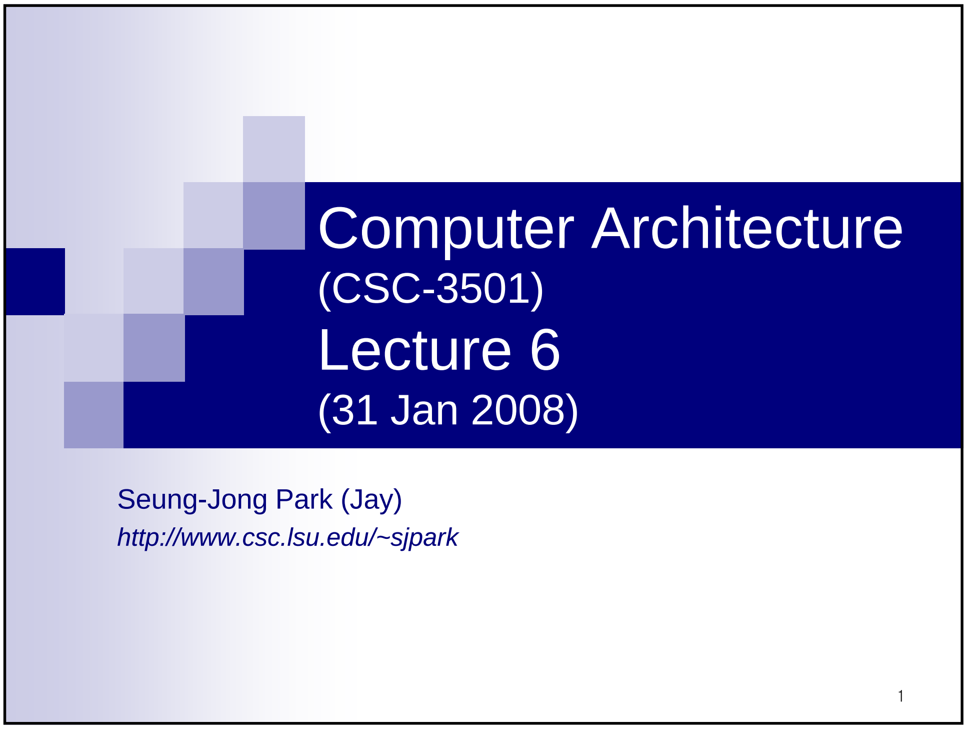


Computer Architecture

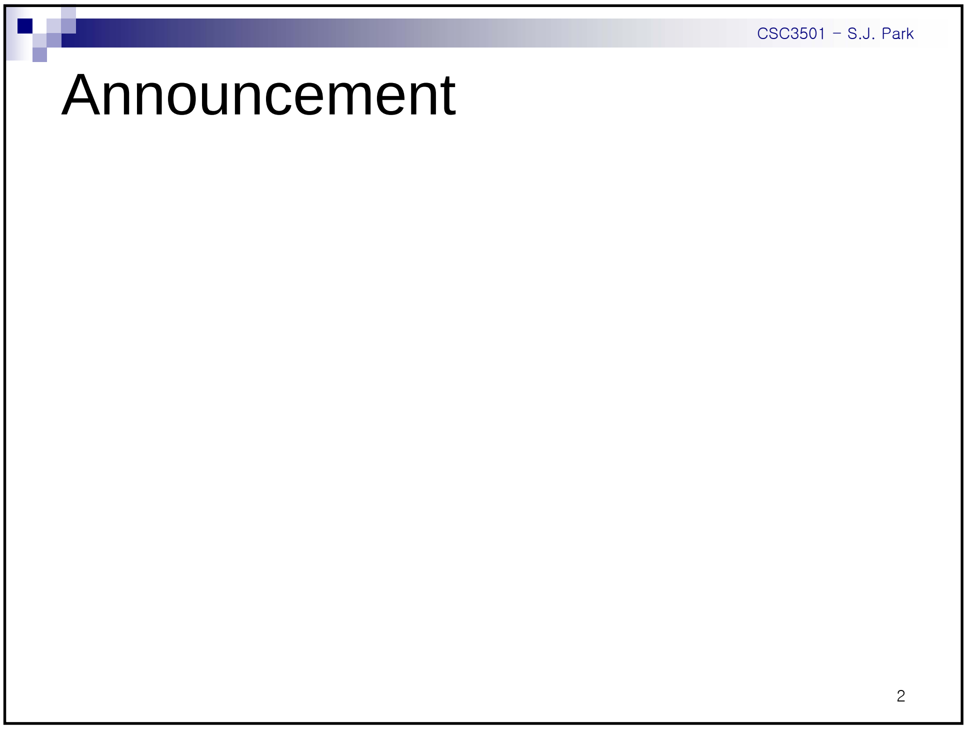
(CSC-3501)

Lecture 6

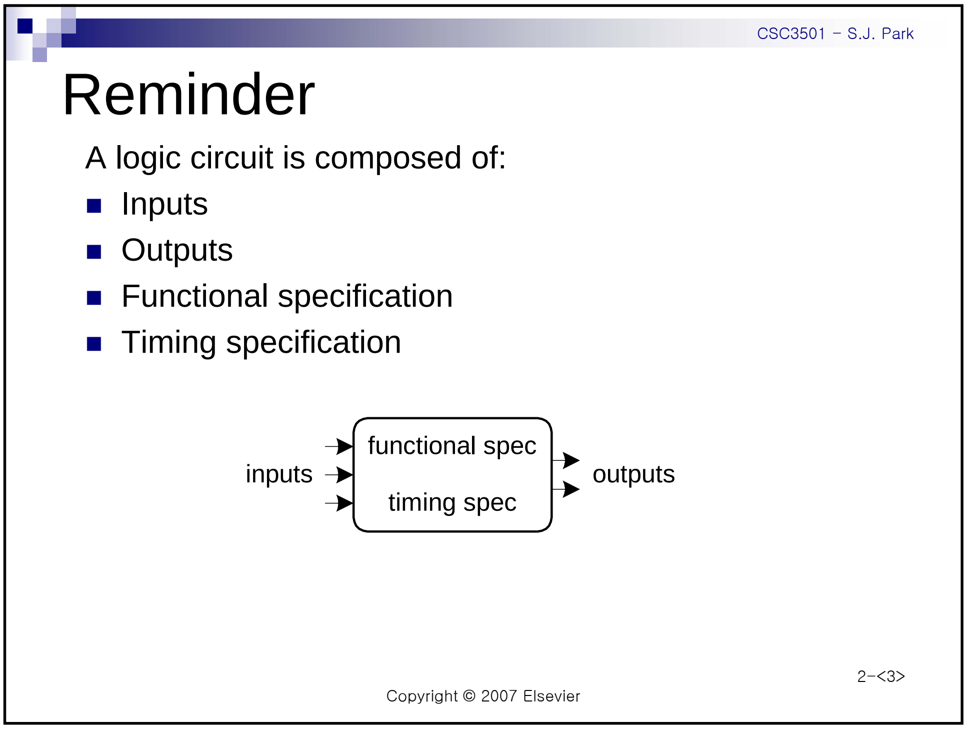
(31 Jan 2008)

Seung-Jong Park (Jay)

http://www.csc.lsu.edu/~sjpark



Announcement



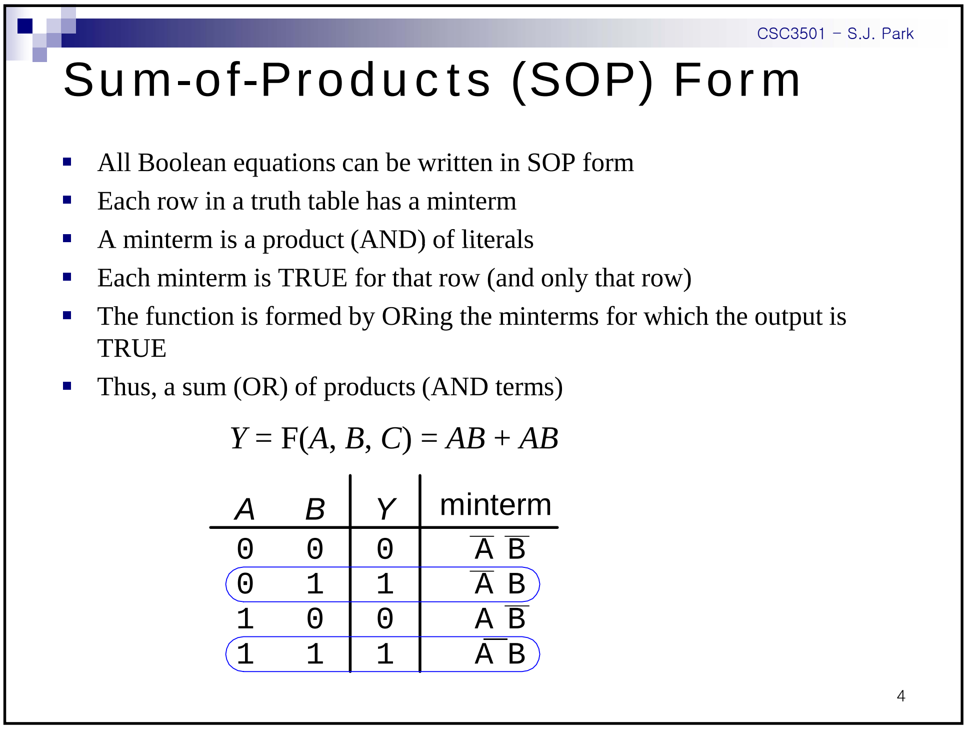
Reminder

A logic circuit is composed of:

- Inputs
- Outputs
- Functional specification
- Timing specification

inputs
functional spec
timing spec
outputs

Copyright © 2007 Elsevier



Sum-of-Products (SOP) Form

- All Boolean equations can be written in SOP form
- Each row in a truth table has a minterm
- A minterm is a product (AND) of literals
- Each minterm is TRUE for that row (and only that row)
- The function is formed by ORing the minterms for which the output is TRUE
- Thus, a sum (OR) of products (AND terms)

$$Y = \mathrm{F}(A, B, C) = AB + AB$$

A	B	Y	minterm
0	0	0	$\overline{A}\ \overline{B}$
0	1	1	$\overline{A}\ B$
1	0	0	$A\ \overline{B}$
1	1	1	$\overline{A}\ B$

Product-of-Sums (POS) Form

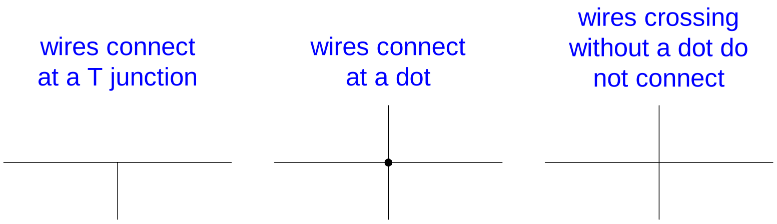
- All Boolean equations can be written in POS form
- Each row in a truth table has a maxterm
- A maxterm is a sum (OR) of literals
- Each maxterm is FALSE for that row (and only that row)
- The function is formed by ANDing the maxterms for which the output is FALSE
- Thus, a product (AND) of sums (OR terms

$$Y = \mathrm{F}(A, B, C) = (A + B)(A + B)$$

A	B	Y	maxterm
0	0	0	$A + B$
0	1	1	$A + \overline{B}$
1	0	0	$\overline{A} + B$
1	1	1	$\overline{A} + \overline{B}$

Circuit Schematic Rules

- Wires always connect at a T junction
- A dot where wires cross indicates a connection between the wires
- Wires crossing *without* a dot make no connection



wires connect at a T junction

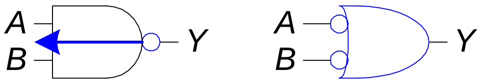
wires connect at a dot

wires crossing without a dot do not connect

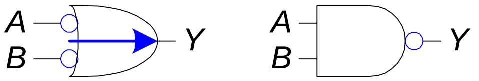
Copyright © 2007 Elsevier

Bubble Pushing

- Think DeMorgan's law
- Pushing bubbles backward (from the output) or forward (from the inputs) changes the body of the gate from AND to OR or vice versa.
- Pushing a bubble from the output back to the inputs puts bubbles on all gate inputs.

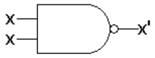


- Pushing bubbles on *all* gate inputs forward toward the output puts a bubble on the output and changes the gate body.



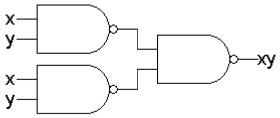
NANDs are special!

- The NAND gate is universal: it can replace all other gates!
  - □



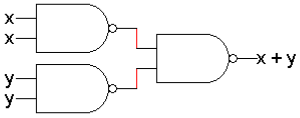
$(xx)' = x'$ [because $xx = x$]

  - □

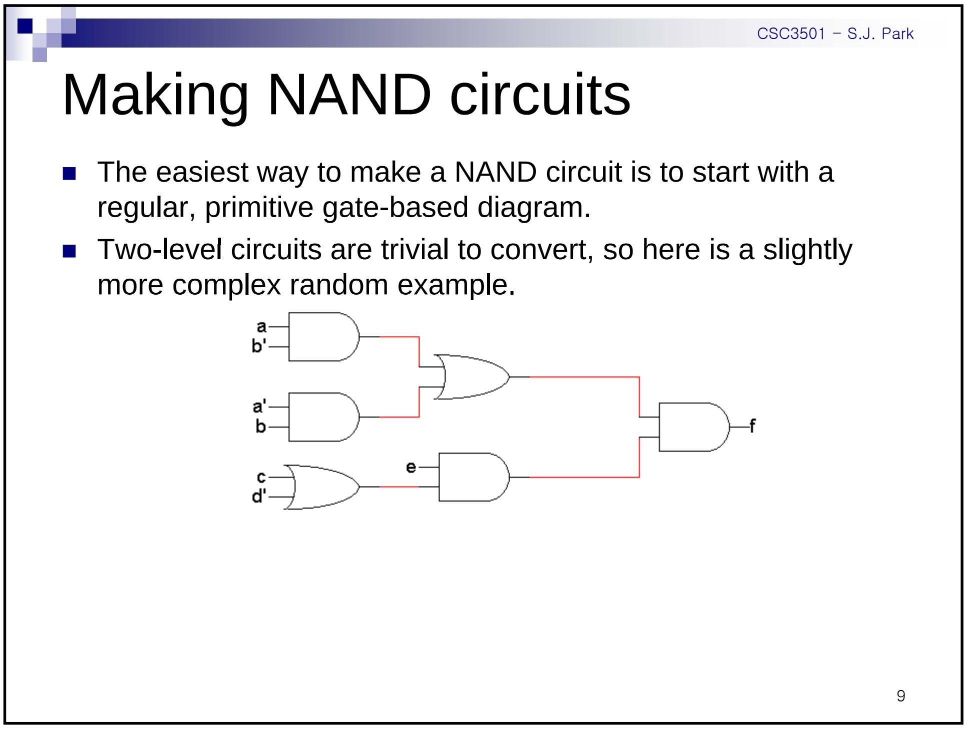


$((xy)'\,(xy)')' = xy$ [from NOT above]

  - □

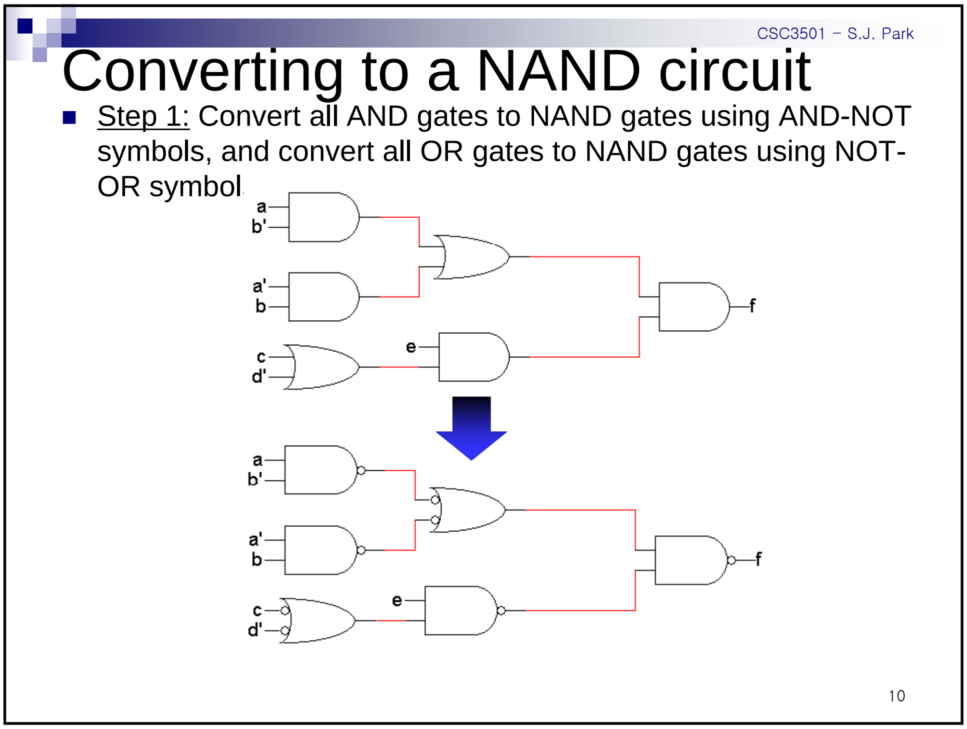


$((xx)'\,(yy)')' = (x'\,y')'$ [$xx = x$, and $yy = y$]
$= x + y$ [DeMorgan's law]



Making NAND circuits

- The easiest way to make a NAND circuit is to start with a regular, primitive gate-based diagram.
- Two-level circuits are trivial to convert, so here is a slightly more complex random example.



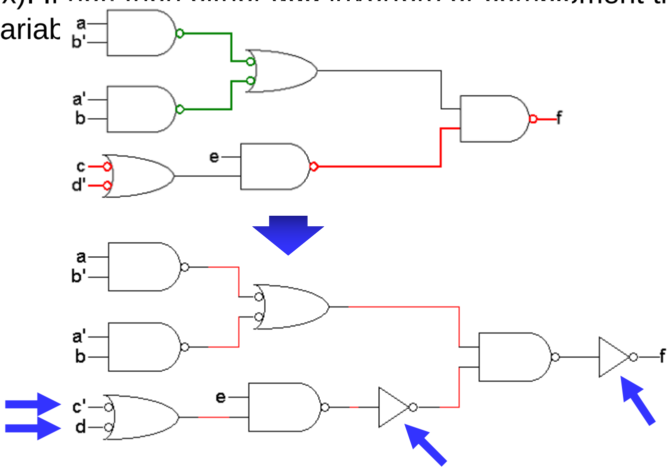
Converting to a NAND circuit

- Step 1: Convert all AND gates to NAND gates using AND-NOT symbols, and convert all OR gates to NAND gates using NOT-OR symbol

a
b'
a'
b
c
d'
e
f
a
b'
a'
b
c
d'
e
f

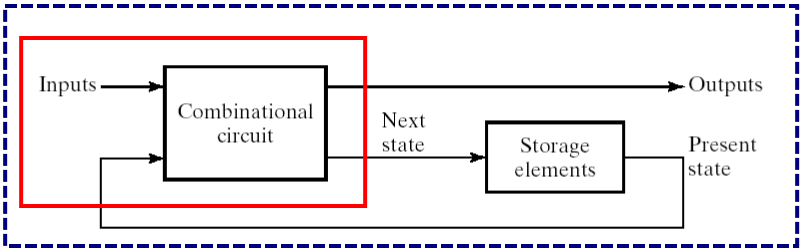
Converting to NAND, concluded

- Step 2: Make sure you added bubbles along lines *in pairs* ((x′)′ = x). If not, then either add inverters or complement the input variab



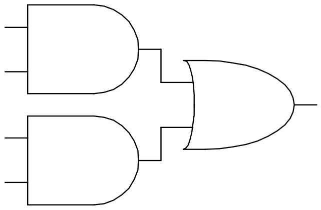
Combinational vs. Sequential Logic

- Combinational Logic
  - Memoryless
  - Outputs determined by current values of inputs
- Sequential Logic
  - Has memory
  - Outputs determined by previous and current values of inputs
- Reference
  - “Logic and Computer Design Fundamentals” by Mano http://www.writphotec.com/mano/



Combinational Logic

- Every circuit element is itself combinational
- Every node of the circuit is either designated as an input to the circuit or connects to exactly one output terminal of a circuit element
- The circuit contains no cyclic paths: every path through the circuit visits each circuit node at most once
- Example:

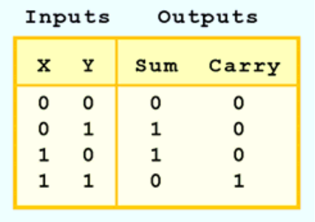


Basic Combinational Logics

- Arithmetic Functions
  - ADD, Subtract, Multiply, Division
- Encoder/Decoder
- Multiplexer(Selector) /Demultiplexer

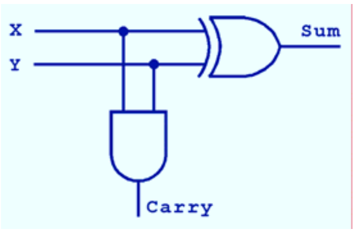
Half Adder

- Combinational logic
- One of the simplest is the *half adder*, which finds the sum of two bits.
- We can gain some insight as to the construction of a half adder by looking at its truth table, shown at the right.



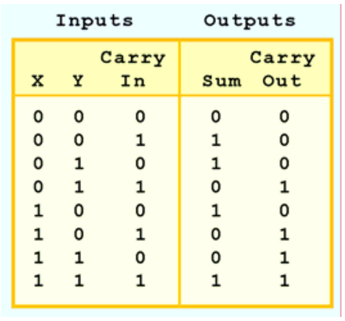
Inputs		Outputs	
X	Y	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

- As we see, the sum can be found using the XOR operation and the carry using the AND operation.

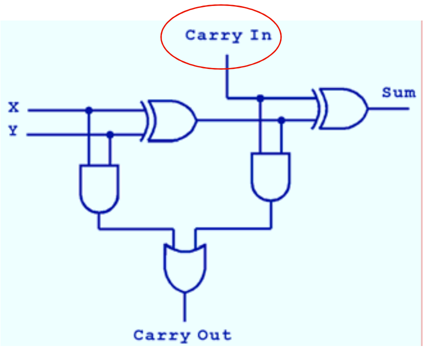


Full Adder

- We can change our half adder into to a full adder by including gates for processing the carry bit.
- The truth table for a full adder is shown at the right.



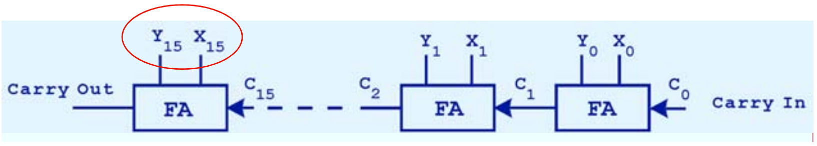
Inputs			Outputs	
X	Y	Carry In	Sum	Carry Out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



Ripple Carry Adder

- Just as we combined half adders to make a full adder, full adders can connected in series.
- The carry bit “ripples” from one adder to the next; hence, this configuration is called a *ripple-carry adder*.

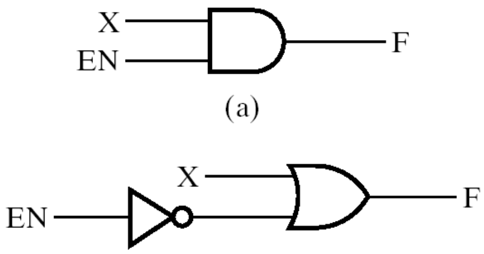
Should wait for previous results



- **Today’s systems employ more efficient adders.**
  - Carry look-ahead adder

Enable

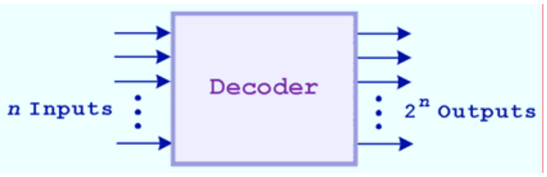
- *Enable* is a common input to logic functions
  - See it in memories, and today’s blocks
  - *Change value inside buffer or memory*



(a)

Decoder

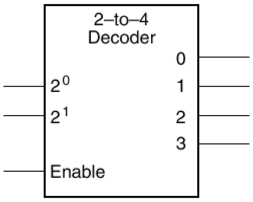
- Decoders are another important type of combinational circuit.
- Among other things, they are useful in selecting a memory location according a binary value placed on the address lines of a memory bus.
- Address decoders with *n* inputs can select any of 2^n locations.

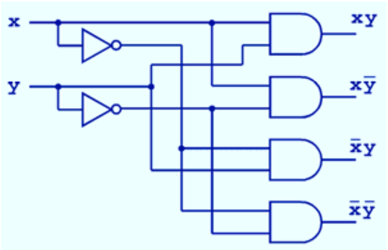


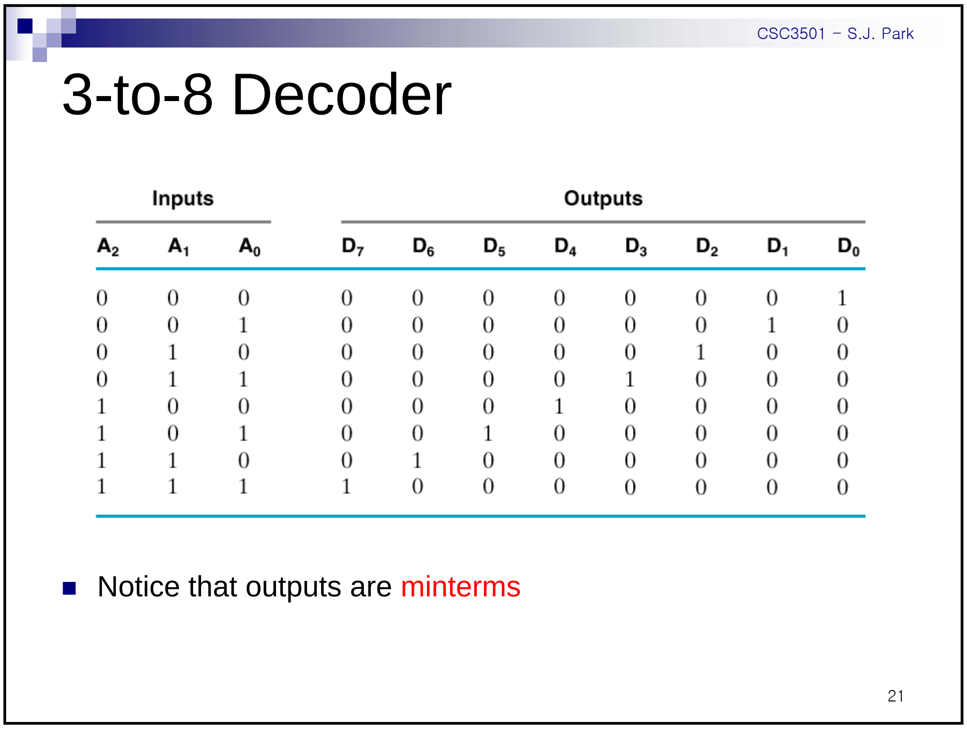
2-to-4 Decoder

- Truth table for 2-to-4 decoder ?

x	y	xy	$x\bar{y}$	$\bar{x}y$	$\overline{xy}$
0	0				
0	1				
1	0				
1	1				



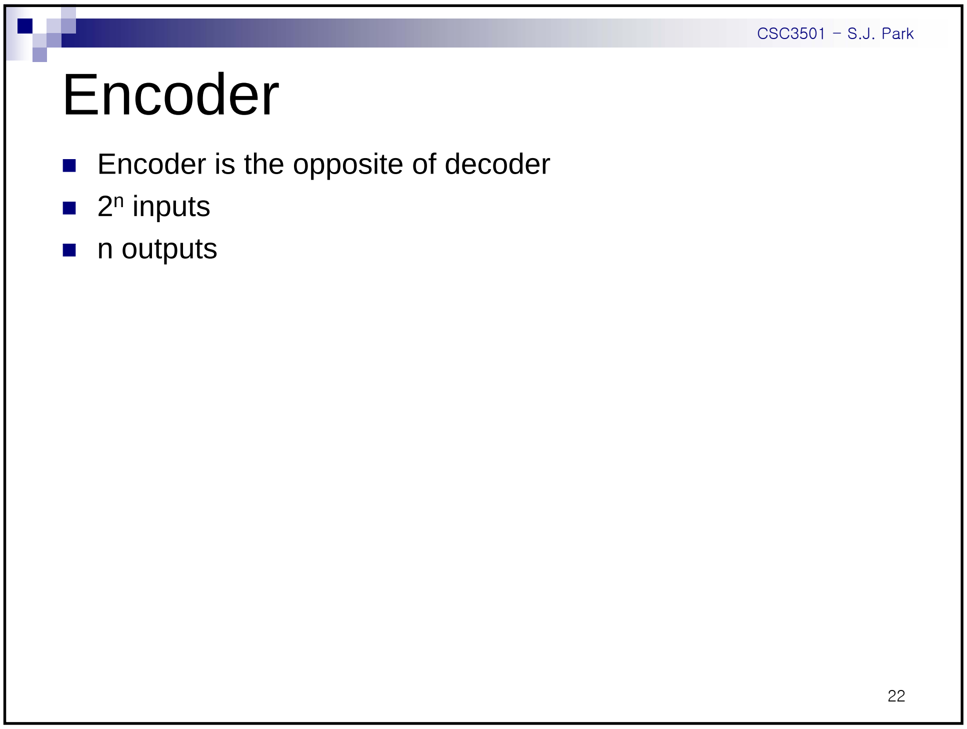




3-to-8 Decoder

Inputs			Outputs							
A_2	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

- Notice that outputs are minterms



Encoder

- Encoder is the opposite of decoder
- 2^n inputs
- n outputs

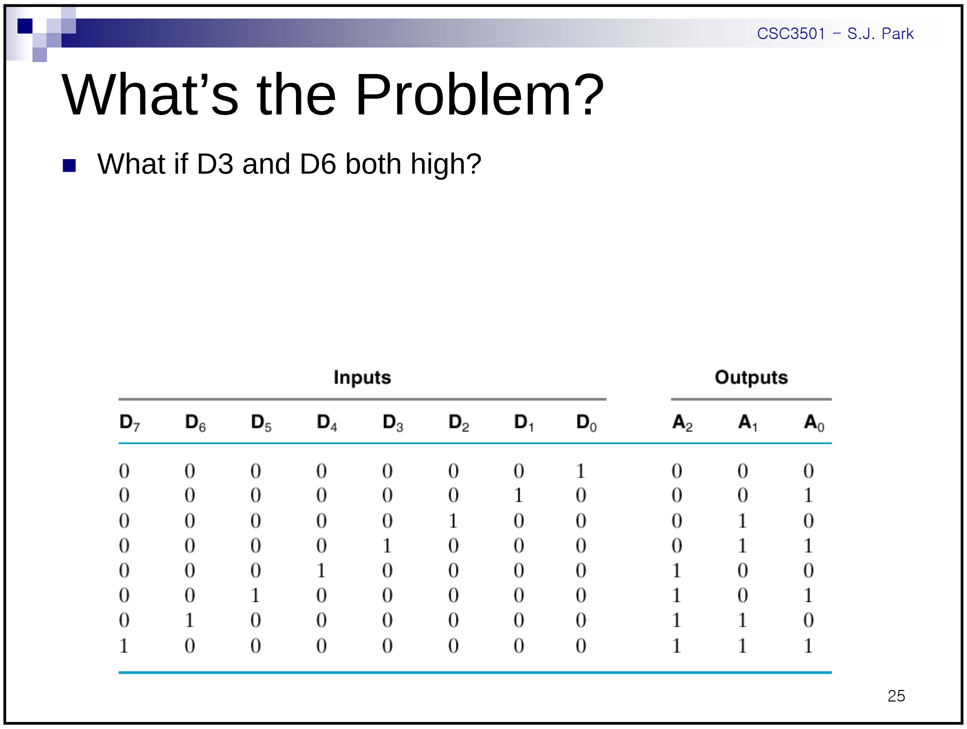
Truth Table of 8-to-3 Encoder

Inputs								Outputs		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_2	A_1	A_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

Inputs are Minterms

- Can OR them together appropriately
- $A_0 = D_1 + D_3 + D_5 + D_7$

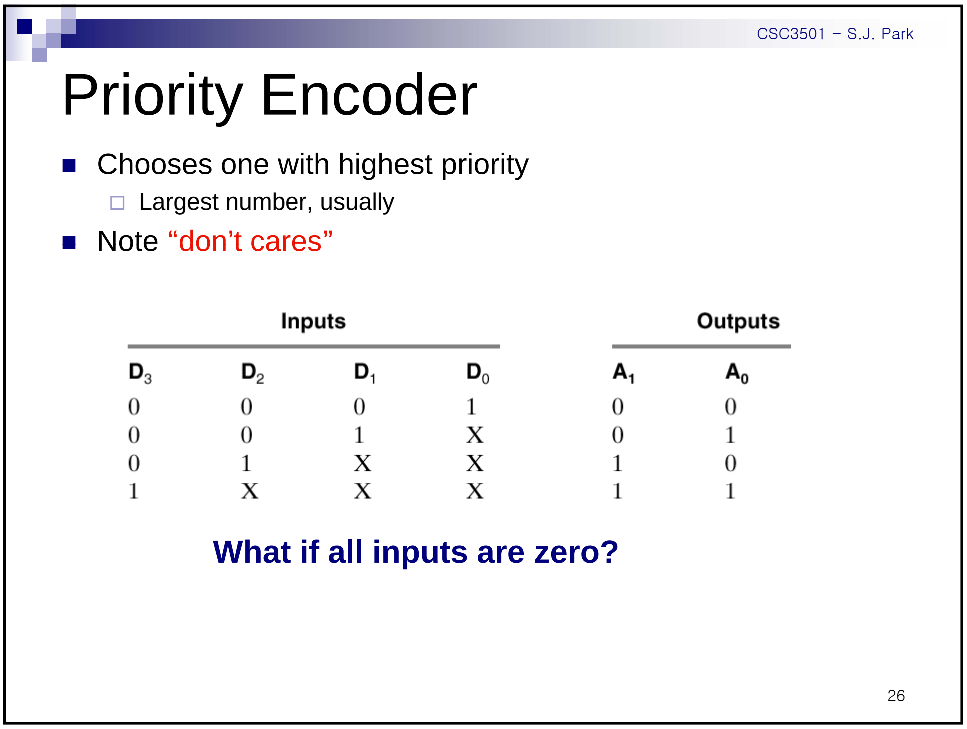
Inputs								Outputs		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_2	A_1	A_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1



What's the Problem?

- What if D3 and D6 both high?

Inputs								Outputs		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_2	A_1	A_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1



Priority Encoder

- Chooses one with highest priority
  - Largest number, usually
- Note "don't cares"

Inputs				Outputs	
D_3	D_2	D_1	D_0	A_1	A_0
0	0	0	1	0	0
0	0	1	X	0	1
0	1	X	X	1	0
1	X	X	X	1	1

What if all inputs are zero?

Need Another Output

Inputs				Outputs		
D_3	D_2	D_1	D_0	A_1	A_0	V
0	0	0	0	X	X	0
0	0	0	1	0	0	1
0	0	1	X	0	1	1
0	1	X	X	1	0	1
1	X	X	X	1	1	1

Valid is OR of inputs

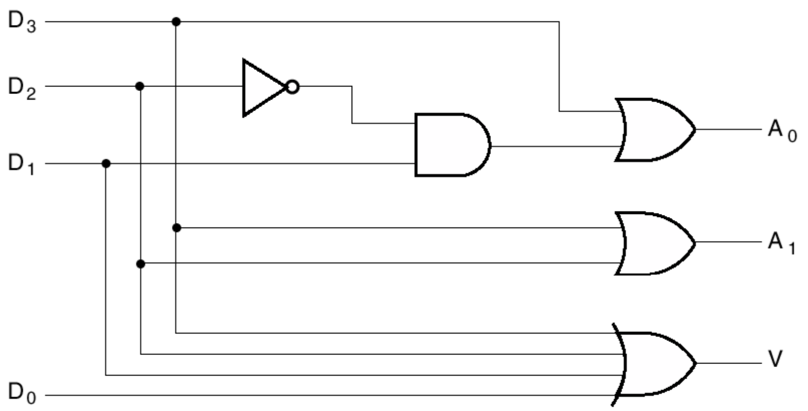
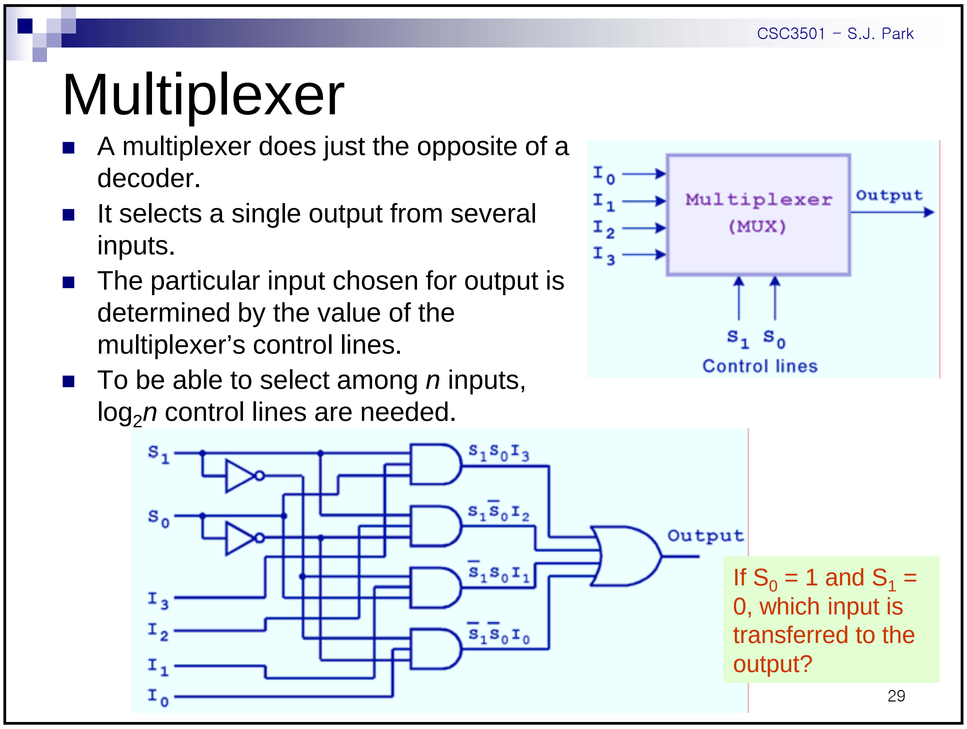


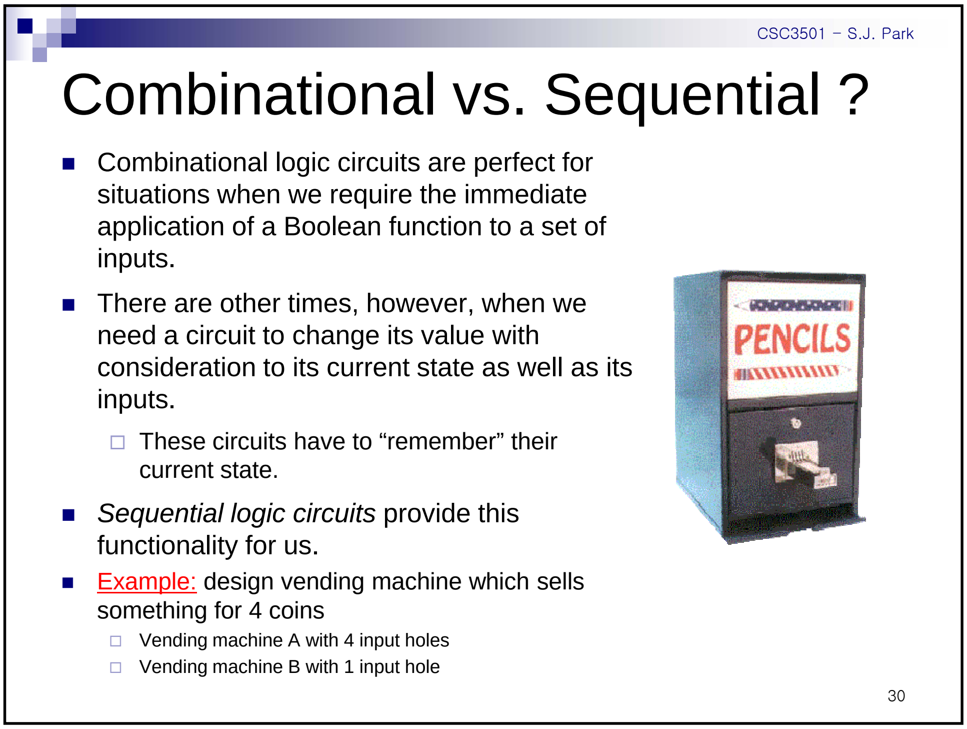
Fig. 3-18 Logic Diagram of a 4-Input Priority Encoder



Multiplexer

- A multiplexer does just the opposite of a decoder.
- It selects a single output from several inputs.
- The particular input chosen for output is determined by the value of the multiplexer's control lines.
- To be able to select among *n* inputs, $\log_2 n$ control lines are needed.

If S_0 = 1 and S_1 = 0, which input is transferred to the output?



Combinational vs. Sequential ?

- Combinational logic circuits are perfect for situations when we require the immediate application of a Boolean function to a set of inputs.
- There are other times, however, when we need a circuit to change its value with consideration to its current state as well as its inputs.
  - These circuits have to "remember" their current state.
- *Sequential logic circuits* provide this functionality for us.
- Example: design vending machine which sells something for 4 coins
  - Vending machine A with 4 input holes
  - Vending machine B with 1 input hole