Lecture #20

ANNOUNCEMENT

- Midterm 2 thurs. april 15, 9:40-11am.
- A-M initials in 10 Evans
- N-Z initials in Sibley auditorium
- Closed book, except for two 8.5 x 11 inch cheat sheets
- Covers HW's 5-9; L's,C's, 1st-order ckts,semiconductor devices, diode ckts, mosfet model, common source amplifier

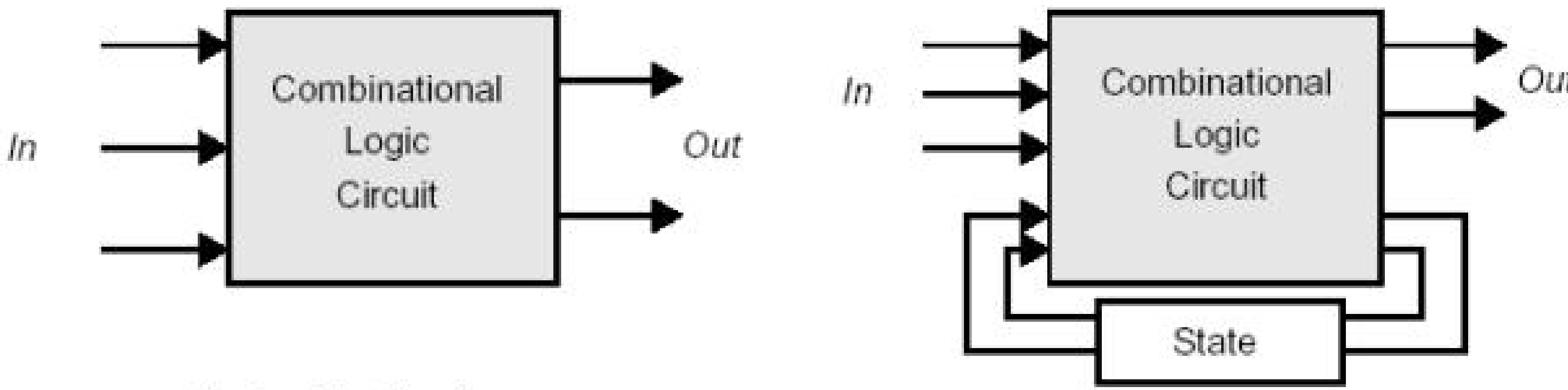
OUTLINE

– Synthesis of logic circuits
– Minimization of logic circuits

Reading: Hambley Ch. 7 through 7.5

Combinational Logic Circuits

- Logic gates combine several logic-variable inputs to produce a logic-variable output.
- ***Combinational logic circuits*** are "memoryless" because their output value at a given instant depends only on the input values at that instant.



(a) Combinational (b) Sequential

- Next time, we will study ***sequential logic circuits*** that possess memory because their present output value depends on previous as well as present input values.

Boolean Algebra Relations

$A \cdot A = A$ $\quad$ $A + A = A$

$A \cdot \overline{A} = 0$ $\quad$ $A + \overline{A} = 1$

$A \cdot 1 = A$ $\quad$ $A + 1 = 1$

$A \cdot 0 = 0$ $\quad$ $A + 0 = A$

$A \cdot B = B \cdot A$ $\quad$ $A + B = B + A$

$A \cdot (B \cdot C) = (A \cdot B) \cdot C$ $\quad$ $A + (B + C) = (A + B) + C$

$$A \cdot (B + C) = A \cdot B + A \cdot C$$

De Morgan's laws:

$$\overline{A \cdot B} = \overline{A} + \overline{B}$$

$$\overline{A} \cdot \overline{B} = \overline{A + B}$$

Boolean Expression Example

$$F = A \cdot \overline{B} \cdot C + A \cdot B \cdot C + (C+D) \cdot (\overline{D}+E)$$

$$F = C \cdot (A+\overline{D}+E) + D \cdot E$$

Logical Sufficiency of NAND Gates

- If the inputs to a NAND gate are tied together, an inverter results

- From De Morgan's laws, the OR operation can be realized by inverting the input variables and combining the results in a NAND gate.

- Since the basic logic functions (AND, OR, and NOT) can be realized by using only NAND gates, NAND gates are sufficient to realize any combinational logic function.

Logical Sufficiency of NOR Gates

- Show how to realize the AND, OR, and NOT functions using only NOR gates

- Since the basic logic functions (AND, OR, and NOT) can be realized by using only NOR gates, NOR gates are sufficient to realize any combinational logic function.

Synthesis of Logic Circuits

Suppose we are given a truth table for a logic function.

Is there a method to implement the logic function using basic logic gates?

Answer: There are lots of ways, but one simple way is the "**sum of products**" implementation method:

1) Write the sum of products expression based on the truth table for the logic function
2) Implement this expression using standard logic gates.

- We may not get the most efficient implementation this way, but we can simplify the circuit afterwards…

Logic Synthesis Example: Adder

Input			Output	
A	B	C	S_1	S_0
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

S_1 using sum-of-products:

1) Find where S_1 is **1**

2) Write down each product of inputs which create a **1**

$\bar{A}BC \quad A\bar{B}C$

$AB\bar{C} \quad ABC$

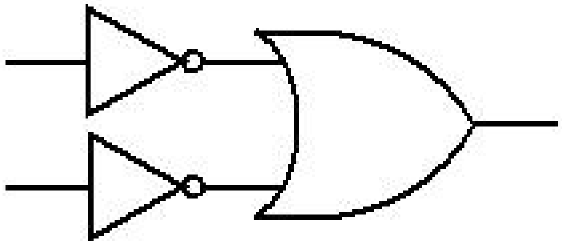
3) Sum all of the products

$$\bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$$

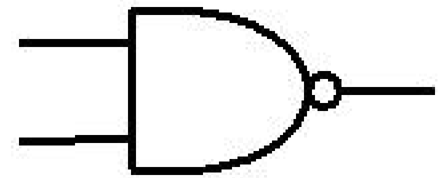
4) Draw the logic circuit

NAND Gate Implementation

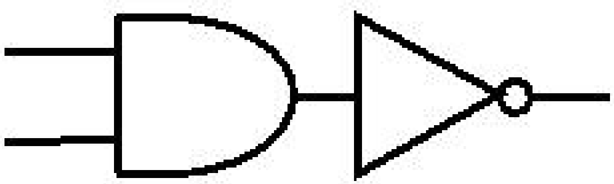
- De Morgan's law tells us that



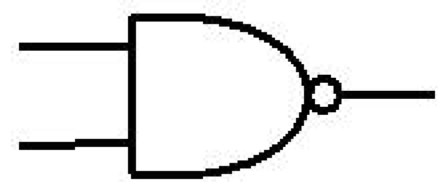
is the same as



- By definition,



is the same as



→ **All sum-of-products expressions can be implemented with only NAND gates.**

Creating a Better Circuit

What makes a digital circuit better?

- Fewer number of gates
- Fewer inputs on each gate
  - multi-input gates are slower

- Let's see how we can simplify the sum-of-products expression for S_1, to make a better circuit…
  - Use the Boolean algebra relations

Karnaugh Maps

- Graphical approach to minimizing the number of terms in a logic expression:
  1. Map the truth table into a Karnaugh map (see below)
  2. For each **1**, circle the biggest block that includes that **1**
  3. Write the product that corresponds to that block.
  4. Sum all of the products

<u>2-variable Karnaugh Map</u>

A \ B	0	1
0		
1		

<u>3-variable Karnaugh Map</u>

A \ BC	00	01	11	10
0				
1				

<u>4-variable Karnaugh Map</u>

AB \ CD	00	01	11	10
00				
01				
11				
10				

Karnaugh Map Example

Input			Output	
A	B	C	S_1	S_0
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Simplification of expression for S_1:

A \ BC	00	01	11	10
0	0	0	1	0
1	0	1	1	1

BC AC AC AB

$$S_1 = AB + BC + AC$$

Further Comments on Karnaugh Maps

- The algebraic manipulations needed to simplify a given expression are not always obvious. Karnaugh maps make it easier to minimize the number of terms in a logic expression.
- Terminology:
  - "2-cube: 2 squares that have a common edge (-> product of 3 variables)
  - "4-cube: 4 squares with common edges (-> product of 2 variables)
- In locating cubes on a Karnaugh map, the map should be considered to fold around from top to bottom, and from left to right.
  - Squares on the right-hand side are considered to be adjacent to those on the left-hand side.
  - Squares on the top of the map are considered to be adjacent to those on the bottom.
  - Example:
    The four squares in the map corners form a 4-cube