Lecture #29

ANNOUNCEMENTS

- Lab project:
  - Bring a check ($50, payable to "UC Regents") to lab this week in order to receive your Tutebot kit. (You will receive this back, when you return the kit.)
  - Extra credit will be awarded if you endow your Tutebot with additional "intelligence"!
- Prof. King's office hour tomorrow (11/6) is cancelled

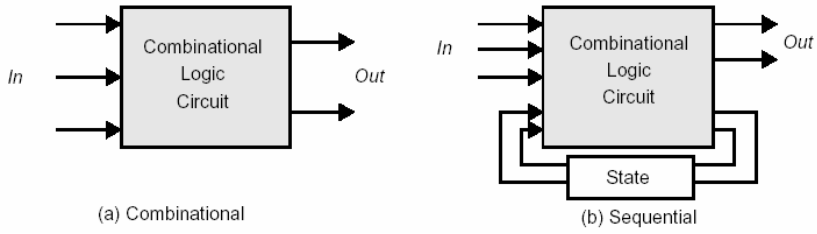
OUTLINE

- Synthesis of logic circuits
- Minimization of logic circuits

Reading: Schwarz & Oldham pp. 403-411

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.



- 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

$A \cdot A = A$	$A + A = A$
$A \cdot \overline{A} = 0$	$A + \overline{A} = 1$
$A \cdot 1 = A$	$A + 1 = 1$
$A \cdot 0 = 0$	$A + 0 = A$
$A \cdot B = B \cdot A$	$A + B = B + A$
$A \cdot (B \cdot C) = (A \cdot B) \cdot C$	$A + (B + C) = (A + B) + C$

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

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

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

De Morgan's laws

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**

$\overline{A}\,B\,C \qquad A\,\overline{B}\,C$

$A\,B\,\overline{C} \qquad A\,B\,C$

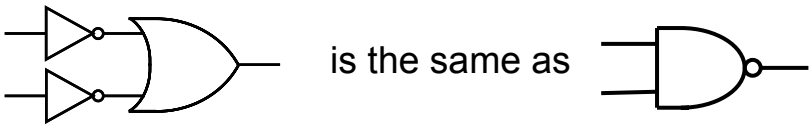
3) Sum all of the products

$\overline{A}\,B\,C + A\,\overline{B}\,C + A\,B\,\overline{C} + A\,B\,C$

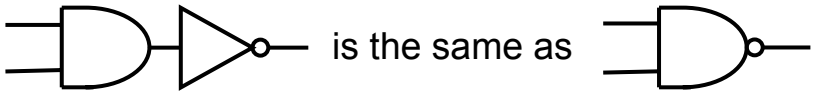
4) Draw the logic circuit

NAND Gate Implementation

- De Morgan's law tells us that



- By definition,



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

Creating a Better Circuit

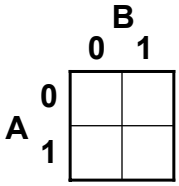
What makes a digital circuit better?

- 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

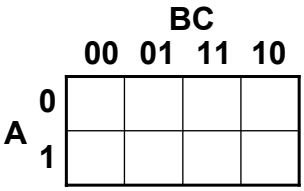
Karnaugh Maps

- 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

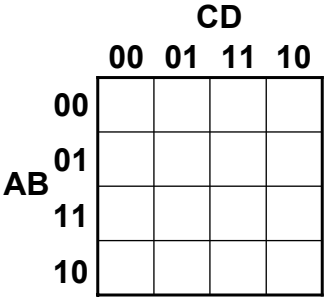
2-variable Karnaugh Map



3-variable Karnaugh Map



4-variable Karnaugh Map



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

$$S_1 = AB + BC + AC$$