Lecture 4

- Logistics
  - HW1 due now
  - HW2 posted now and is due one week from today
  - Lab1 going on this week
  - Feedback on lectures, hw, lab, anything else
- Last lecture --- Boolean algebra
  - Axioms
  - Useful laws and theorems
  - Simplifying Boolean expressions
- Today's lecture
  - One more example of Boolean logic simplification
  - Logic gates and truth tables in detail
  - Implementing logic functions

The "WHY" slide

- Logic Gates and Truth Table
  - Now you know 0's and 1's and the basic Boolean algebra, now you are ready to go back and forth between truth table, Boolean expression, and logic gates. This ability to go back and forth is an extremely useful skill designing and optimizing computer hardware.
- Implementing Logic Functions
  - Now with these basic tools you learned, you can "implement" logic functions. You learned algebra in junior/high school and you use it for many things you do now. We use Boolean algebra to implement logic functions that are used in the computers. And these logic functions are used by computer programs you write.

One more example of logic simplification

- Example:

Z = A'BC + AB'C' + AB'C + ABC' + ABC

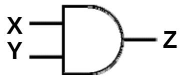
= A'BC + AB'(C’ + C) + AB(C' + C) distributive
= A'BC + AB’ + AB complementary
= A'BC + A(B' + B) distributive
= A'BC + A complementary

= BC + A absorption #2 Duality

(X • Y')+Y=X+Y} with X=BC and Y=A

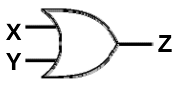
Logic gates and truth tables

- AND X•Y XY



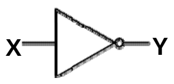
X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

- OR X+Y



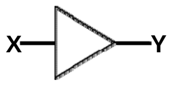
X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

- NOT $\overline{X}$ X’



X	Y
0	1
1	0

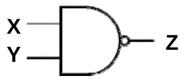
- Buffer X



X	Y
0	0
1	1

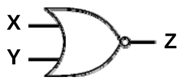
Logic gates and truth tables (con't)

- NAND $\overline{X \bullet Y}$ $\overline{XY}$



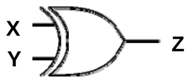
X	Y	Z
0	0	1
0	1	1
1	0	1
1	1	0

- NOR $\overline{X + Y}$



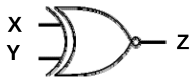
X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	0

- XOR $X \oplus Y$



X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	0

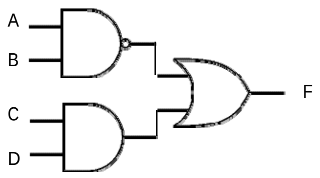
- XNOR $\overline{X \oplus Y}$



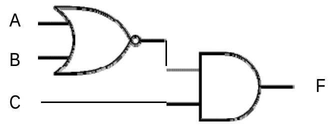
X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	1

Boolean expressions $\Rightarrow$ logic gates

- Example: F = (A•B)' + C•D



- Example: F = C•(A+B)'

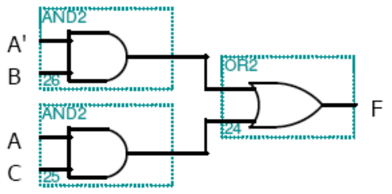


Truth tables ⇨ logic gates

- Given a truth table
  - Write the Boolean expression
  - Minimize the Boolean expression
  - Draw as gates
  - Example:

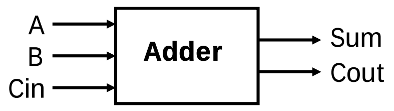
A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

$$\begin{aligned} F &= A'BC'+A'BC+AB'C+ABC \\ &= A'B(C'+C)+AC(B'+B) \\ &= A'B+AC \end{aligned}$$



Example: A binary full adder

- 1-bit binary adder
  - Inputs: A, B, Carry-in
  - Outputs: Sum, Carry-out



A	B	Cin	S	Cout
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

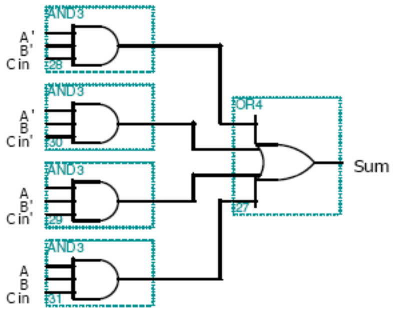
Sum = A'B'Cin + A'BCin' + AB'Cin' + ABCin
Cout = A'BCin + AB'Cin + ABCin' + ABCin

Both Sum and Cout can be minimized.

Full adder: Sum

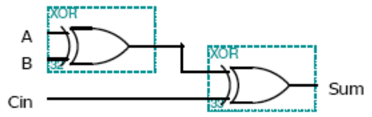
Before Boolean minimization

Sum = A'B'Cin + A'BCin'
+ AB'Cin' + ABCin



After Boolean minimization

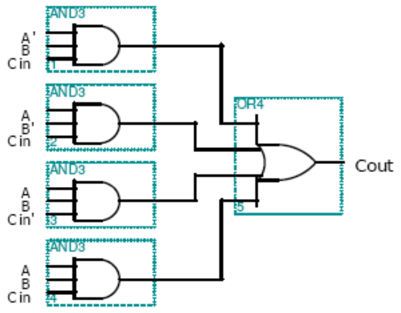
Sum = (A⊕B) ⊕ Cin



Full adder: Carry-out

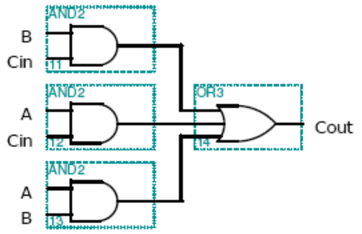
Before Boolean minimization

Cout = A'BCin + AB'Cin
+ ABCin' + ABCin

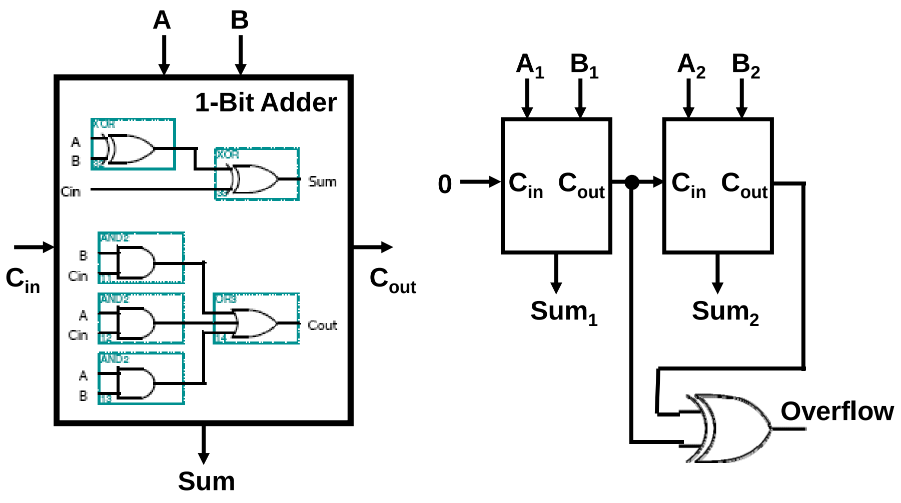


After Boolean minimization

Cout = BCin + ACin + AB

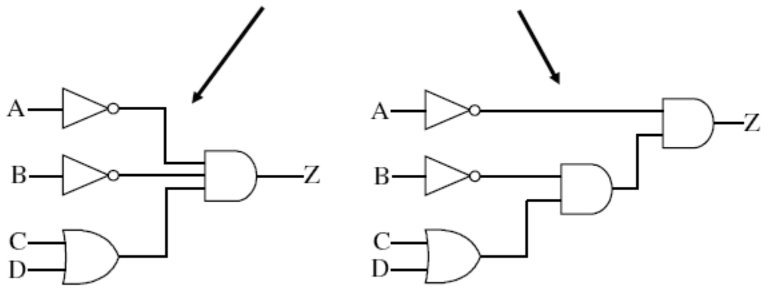


Preview: A 2-bit ripple-carry adder



Many possible mappings

- ◆ Many ways to map expressions to gates
  - Example: $Z = \overline{A} \bullet \overline{B} \bullet (C + D) = \overline{A} \bullet \ \overline{B} \bullet (C + D)$



What is the optimal gate realization?

- We use the axioms and theorems of Boolean algebra to "optimize" our designs
- Design goals vary
  - Reduce the number of gates?
  - Reduce the number of gate inputs?
  - Reduce number of chips and/or wire?
- How do we explore the tradeoffs?
  - Logic minimization: Reduce number of gates and complexity
  - Logic optimization: Maximize speed and/or minimize power
  - CAD tools

Minimal set

- We can implement any logic function from NOT, NOR, and NAND
  - Example: (X and Y) = not (X nand Y)
- In fact, we can do it with only NOR or only NAND
  - NOT is just NAND or NOR with two identical inputs

X	Y	X nor Y
0	0	1
1	1	0

X	Y	X nand Y
0	0	1
1	1	0

  - NAND and NOR are duals: Can implement one from the other
    - X nand Y = not ((not X) nor (not Y))
    - X nor Y = not ((not X) nand (not Y))