Week 15

Logic Gates

Fall 2013

Student Responsibilities — Week 15

- **Reading**: Textbook, Section 11.1 – 11.3
- **Attendance**: Finally! Encouraged

Week 15 Overview

How Boolean logic and Boolean algebra relate to computer circuits and chips.

- Sec 11.1 Boolean Functions
- Sec 11.2 Representing Boolean Functions
- Sec 11.3 Logic Gates

9.1 Boolean Functions

Boolean Algebra provides the operations [*complement, product, sum*] and rules for working with the set $\{0, 1\}$.

Complement (denoted with bar): $\overline{0} = 1, \quad \overline{1} = 0.$

Product (denoted with `AND`, $\bullet$, or implicit):

$$1 \bullet 1 = 1 \quad 1 \bullet 0 = 0 \quad 0 \bullet 1 = 0 \quad 0 \bullet 0 = 0$$

Sum (denoted with `OR` or $+$):

$$1 + 1 = 1 \quad 1 + 0 = 1 \quad 0 + 1 = 1 \quad 0 + 0 = 0$$

Precedence of Operators: complement, product, sum

Example: $(1 + 0) \bullet \overline{(0 \bullet 1)} \;=\; 1 \bullet \overline{0} \;=\; 1 \bullet 1 \;=\; 1$

Boolean Functions

Let $B = \{0, 1\}$

- A **Boolean variable** x assumes values only from B.
- A **Boolean Function of Degree n** is a function from B^n, the set $\{(x_1, x_2, \ldots, x_n) | x_i \in B, 1 \leq i \leq n\}$, to B.

  Function values are often displayed in tables.

Boolean Expressions

- **Boolean Expressions**, which can represent Boolean functions, are made up from Boolean variables and operations.

- They are defined recursively as follows:
  - $0, 1, x_1, x_2, \ldots, x_n$ are Boolean expressions.
  - If E_1 and E_2 are Boolean expressions, then so are their complements, their product, and their sum:

$$\overline{E_1}, \quad (E_1 E_2), \quad \text{and} \quad (E_1 + E_2)$$

- Each Boolean expression represents a Boolean function.

Function Evaluation

To evaluate a function, we substitute 0's and 1's for the variables in the same way we did for Truth Tables.

Table 1. $F(x, y, z) = \overline{x} + yz$

x	y	z	$\overline{x}$	yz	$F(x, y, z) = \overline{x} + yz$
1	1	1	0	1	1
1	1	0	0	0	0
1	0	1	0	0	0
1	0	0	0	0	0
0	1	1	1	1	1
0	1	0	1	0	1
0	0	1	1	0	1
0	0	0	1	0	1

Equivalence of Boolean Functions

Two Boolean functions F and G are **equivalent** if and only if when they are evaluated on the variables $b_1, b_2, \ldots, b_n \in B$:

$$F(b_1, b_2, \ldots, b_n) \quad = \quad G(b_1, b_2, \ldots, b_n)$$

All these functions are equivalent: xy $\qquad xy + 0$ $\qquad xy \bullet 1$

Boolean Operators on Functions

- The **complement** of the Boolean function F is the function $\overline{F}$, where:

$$\overline{F}(x_1, \ldots, x_n) \;=\; \overline{F(x_1, \ldots, x_n)}$$

- The **Boolean sum** $F + G$ is defined by:

$$(F + G)(x_1, \ldots, x_n) \;=\; F(x_1, \ldots, x_n) + G(x_1, \ldots, x_n)$$

- The **Boolean product** FG is defined by:

$$(FG)(x_1, \ldots, x_n) \;=\; F(x_1, \ldots, x_n)G(x_1, \ldots, x_n)$$

Degree of a Function

The **degree** of a Boolean function is the number of different variables upon which it depends.

$$F(x_1, \ldots, x_n) \text{ has degree } n.$$

Table 3. Boolean Functions of Degree 2.

x	y	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8
1	1	1	1	1	1	1	1	1	1
1	0	1	1	1	1	0	0	0	0
0	1	1	1	0	0	1	1	0	0
0	0	1	0	1	0	1	0	1	0

x	y	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}	F_{16}
1	1	0	0	0	0	0	0	0	0
1	0	1	1	1	1	0	0	0	0
0	1	1	1	0	0	1	1	0	0
0	0	1	0	1	0	1	0	1	0

Table 4. The Number of Boolean Functions of Degree n

Degree	Number
1	4
2	16
3	256
4	65,536
5	4,294,967,296
6	18,446,744,073,709,551,616

Boolean Function with degree n

There are 2^n n-tuples of 0's and 1's — representing all possible combinations of the n variable values.

Each function is an assignment of 0's and 1's to each of these n–tuples

Hence, there are 2^{2^n} different Boolean functions of degree n.

Boolean Identities

Identity	**Name**
$\overline{\overline{x}} = x$	Law of Double Complement
$x + x = x$ $xx = x$	Idempotent Laws
$x + 0 = x$ $x(1) = x$	Identity Laws
$x + 1 = 1$ $x(0) = 0$	Dominance Laws
$x + y = y + x$ $xy = yx$	Commutative Laws
$x + (y + z) = (x + y) + z$ $x(yz) = (xy)z$	Associative Laws
$x + yz = (x + y)\,(x + z)$ $x(y + z) = xy + xz$	Distributive Laws
$\overline{xy} = \overline{x} + \overline{y}$ $\overline{x + y} = \overline{xy}$	De Morgan's Laws

Boolean Algebra

A **Boolean algebra** is a set B with:

- two binary operations, $\vee$ and $\wedge$
- elements 0 and 1
- a unary operation such that the following properties hold $\forall x, y, z \in B$:

$x \vee 0 = x$ $x \wedge 1 = x$	Identity Laws
$x \vee \overline{x} = 1$ $x \wedge \overline{x} = 0$	Dominance Laws
$(x \vee y) \vee z = x \vee (y \vee z)$ $(x \wedge y) \wedge z = x \wedge (y \wedge z)$	Associative Laws
$x \vee y = y \vee x$ $x \wedge y = y \wedge x$	Commutative Laws
$x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$ $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$	Distributive Laws

Equivalent Collections

Collections which satisfy all three Boolean Algebra properties include:

- $B = \{0, 1\}$, with $\{+, \bullet\}$ and the complement operator
- The set of propositions in n variables with the $\vee$ and $\wedge$ operators, `F` and `T`, and the negation operator.
- The set of subsets of a universal set U with $\cap$ and $\cup$, $\emptyset$, and set complementation operator.

They All Tie Together

To establish results about each of

Boolean expressions

Propositions

and

Sets,

we need only prove results about abstract Boolean Algebras!

Section 11.3 — Logic Gates

- Computer chips are made up of vast numbers of circuits.
- Circuits can be designed using the rules of Boolean algebra.
- The basic components of circuits are called gates and each type of gate implements a Boolean operation.
- We can use the rules of Boolean algebra to combine gates into circuits that perform various tasks. Input and output will both be from the set 0, 1.

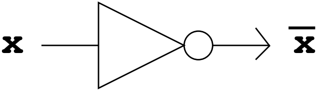
- The **combinatorial circuits** or **gating networks** we'll be studying depend only upon the **inputs**, and not on the **current state** of the circuit - i.e., they have no memory capabilities.

- The three types of elements we'll use to create circuits are:
  - the **inverter**, which produces the complement of its input value;
  - the **OR** gate, which produces the sum of its inputs, and
  - the **AND** gate, which produces the product of its inputs

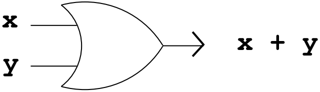
Symbolic Gates

The symbols used for these types of elements are shown below:

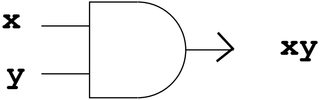
- **inverter**:



- **OR gate**:

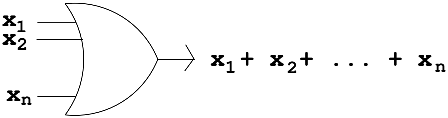


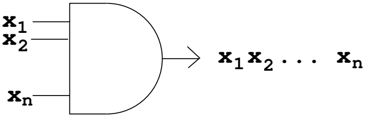
- **AND gate**:



Multiple Input Gates

- We can also have multiple input OR and AND gates. Examples of gates with *n* inputs are shown below:

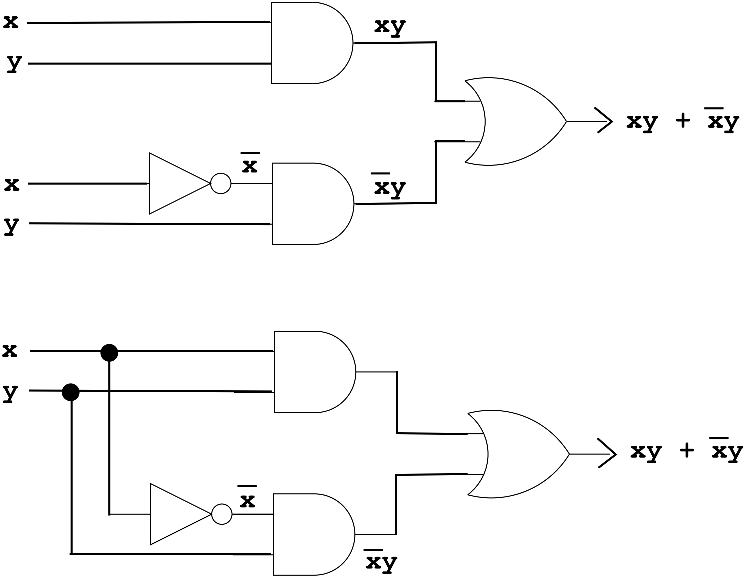




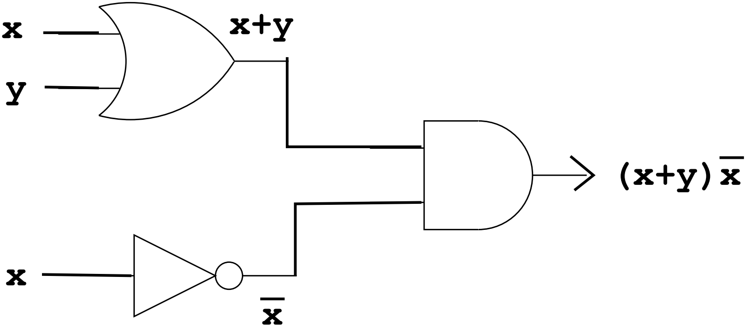
- Inputs enter inverters and gates from the their left sides and output is shown leaving from their right sides. There is only one way for current to flow through these components.

- Combinational circuits can be constructed using a combination of inverters, OR, and AND gates.

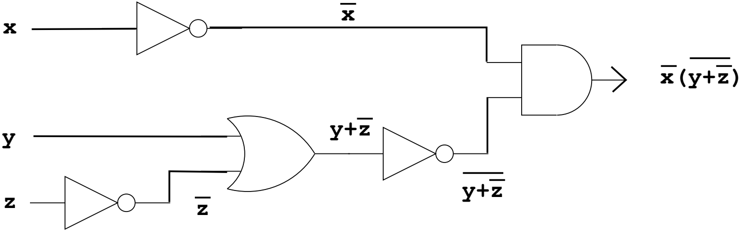
- When combinations of circuits are formed, some gates may share inputs. There are two common ways to show this:
  - One is to give the same name to the separate inputs for each gate, as shown in the first figure on the next slide.
  - The other is to use branches that indicate all gates using a given input. This is shown in the second figure.


x
y
xy
$xy + \overline{x}y$
x
y
$\overline{x}$
$\overline{x}y$
x
y
xy
$xy + \overline{x}y$
$\overline{x}$
$\overline{x}y$

Examples of Circuits: $(x + y)\bar{x}$



Examples of Circuits: $\bar{x}(\overline{y+\bar{z}})$



Two–Switch Light (0 = off, 1 = on)

Flipping either switch should turn the light on if it's off, and off if it's on.

2–switch light		
x	**y**	**F(x, y)**
1	1	1
1	0	0
0	1	0
0	0	1

Three–Switch Light (0 = off, 1 = on)

3–switch light			
x	**y**	**z**	**F(x, y, z)**
1	1	1	1
1	1	0	0
1	0	1	0
1	0	0	1
0	1	1	0
0	1	0	1
0	0	1	1
0	0	0	0

Three Voters with Majority Rule

3–Votes, Majority Rules			
x	**y**	**z**	**M(x, y, z)**
1	1	1	
1	1	0	
1	0	1	
1	0	0	
0	1	1	
0	1	0	
0	0	1	
0	0	0	

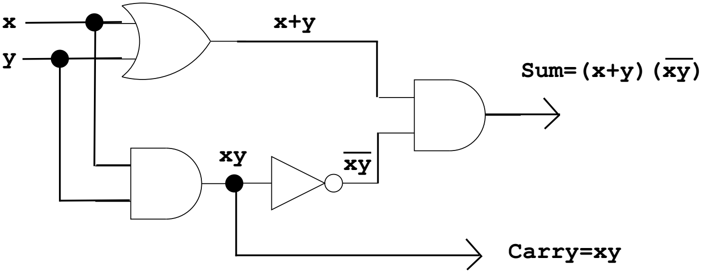
Adders

- One of the most common uses for a computer is numerical computation.

- We will next see how we can design circuits to carry out addition.

- A **half adder** adds two bits without considering any carry from a previous addition.
  - The **input** will be two values, x and y, each either 0 or 1.
  - The **output** will be the sum bit s, and the carry bit, c.
  - This circuit is called a **multiple output circuit** since it has more than one output.

Half–Adder I/O			
INPUT		OUTPUT	
x	y	s	c
1	1	0	1
1	0	1	0
0	1	1	0
0	0	0	0



Full Adder

- The **full adder** is used to compute the sum bit and carry bit when two bits and a carry are added.

- The **inputs** to the full adder are the bits x and y, and the carry c_i.

- The **outputs** are the sum bit s and the new carry c_{i+1}.

Full–Adder I/O

INPUT			OUTPUT	
x	y	c_i	s	c_{i+1}
1	1	1	1	1
1	1	0	0	1
1	0	1	0	1
1	0	0	1	0
0	1	1	0	1
0	1	0	1	0
0	0	1	1	0
0	0	0	0	0

This full adder utilizes half adders rather than building them from scratch: