Boolean Algebra and Its Relation to Gates

An Introduction to CS233

CS 398 this semester

PICK UP HANDOUT

233 in one slide!

- **The class consists roughly of 4 quarters:**
  1. You will build a simple computer processor
  2. You will learn how high-level language code executes on a processor
  3. You will learn why computers perform the way they do
  4. You will learn about hardware mechanisms for parallelism

- **We will have a SPIMbot contest!**

- **Section begins this week, so I must teach you something!**
  - More on class mechanics on Wednesday…

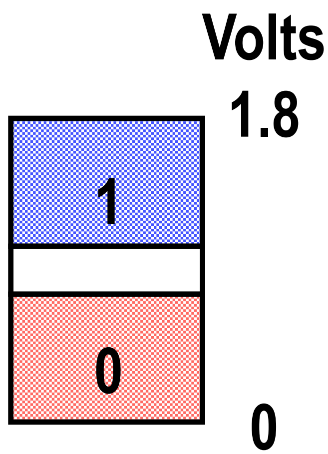
Today's lecture

- **Basic Boolean expressions**
  - Booleans
  - AND, OR and NOT
  - Expressing Boolean functions:
    - as **truth tables**
    - as mathematical **expressions**
    - as digital circuits made of **gates**
    - using **hardware description languages**

Computing: It is all just ones and zeros

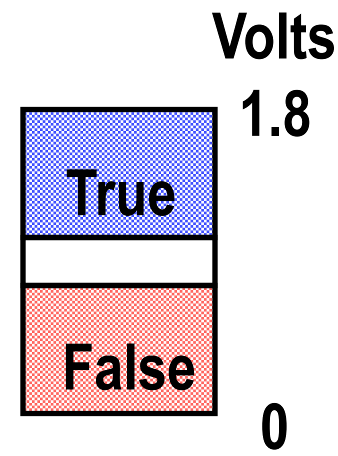
- **Computers use voltages to represent information.**

- **For reliability and ease of design, however, we group ranges of voltages into two discrete, or digital, values: 1 and 0.**
  - This two-valued domain is referred to as **BINARY**
  - Often **1** is used for **TRUE** and **0** for **FALSE**.



- **Everything in digital computers is represented with ones and zeros**
  - We'll show you how.

Boolean values



概念

- **If we think of our digital voltages as two *logical* values, true and false, we call these "Booleans"**
  - After the mathematician George Boole

- **For simplicity, we often still write digits instead:**
  - 1 is true
  - 0 is false

- **Boolean algebra is the mathematics defined over this binary domain.**

Boolean functions

- **Just like in other mathematics, we can define functions:**

$$y = f(x)$$

- **The output is specified purely by the function & inputs**

- **Because there are a finite number (2) of boolean values…**
  - **There are a finite number of boolean functions**

- **For 1-input functions (e.g., f(x)) there are only 4 possible**
  - (let's first see how to represent these…)

Truth tables

- **A truth table shows all possible inputs & outputs of a function.**
- **Each input variable is either 1 or 0. (so are the outputs.)**
  - Because there are only a finite number of values (1 and 0), truth tables themselves are finite.
  - A function with *n* variables has 2^n possible combinations of inputs.

x	y	z	f(x,y,z)
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

1-input Boolean functions

$$y = f(x)$$

- **A 1-input Boolean function has 2^1 = 2 possible inputs:**

x	f(x)
0	f(0)
1	f(1)

- **There are $2^{2^{(\# \text{ of inputs})}}$ possible functions**
  - For each input, there are 2 possible outputs
  - The outputs are independent for each input (hence the multiplication)
- **The 4 possible 1-input Boolean functions**

return ∅

x	$f_0(x)$
0	0
1	0

x	$f_1(x)$
0	0
1	1

return input

x	$f_2(x)$
0	1
1	0

inversion
not

return 1

x	$f_3(x)$
0	1
1	1

2-input Boolean functions

$$z = f(x,y)$$

- **4 possible inputs, 16 possible functions:**

x	y	f0	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11	f12	f13	f14	f15
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

AND

OR

- **We'll focus on 2 for now**

Basic Boolean operations

- There are three basic operations for logical values.

Operation:	AND (product) of two inputs	OR (sum) of two inputs	NOT (complement) on one input
Expression Notation:	xy, or $x \cdot y$	$x + y$	x' or $\bar{x}$

Truth table:

x	y	xy
0	0	0
0	1	0
1	0	0
1	1	1

x	y	x+y
0	0	0
0	1	1
1	0	1
1	1	1

x	x'
0	1
1	0

These are sufficient to implement any Boolean function

Basic Boolean operations

- There are three basic operations for logical values.

	AND (product) of two inputs	OR (sum) of two inputs	NOT (complement) on one input
Operation:	AND (product) of two inputs	OR (sum) of two inputs	NOT (complement) on one input
Expression Notation:	xy, or $x \cdot y$	$x + y$	x' or $\overline{x}$

Truth table:

x	y	xy
0	0	0
0	1	0
1	0	0
1	1	1

x	y	x+y
0	0	0
0	1	1
1	0	1
1	1	1

x	x'
0	1
1	0

These are sufficient to implement any Boolean function

Boolean expressions (formally)

- **Use these basic operations to form more complex expressions:**

$$f(x,y,z) = (x + y')z + x'$$

- **Some terminology and notation:**
  - f is the name of the function.
  - (x,y,z) are the input variables, each representing 1 or 0. Listing the inputs is optional, but sometimes helpful.
  - A literal is any occurrence of an input variable or complement. The function above has four literals: x, y', z, and x'.
- **Precedences are important, but not too difficult.**
  - NOT has the highest precedence, followed by AND, and then OR.
  - Fully parenthesized, the function above would be kind of messy:

$$f(x,y,z) = (((x +(y'))z) + x')$$

Boolean expressions to Truth tables

- **To compute a truth table given a Boolean expression:**
  - Evaluate the function for every combination of inputs.

$f(x,y,z) = (x + y')z + x'$

$f(0,0,0) = (0 + 1)0 + 1 = 1$

$(1)0 + 1$

$0 + 1 = 1$

$f(1,0,1) = (1 + 1)1 + 0 = 1$

$(1)1 + 0$

$1 + 0 = 1$

x	y	z	f(x,y,z)
0	0	0	1
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	1
1	1	0	
1	1	1	

a)	0
b)	1

Boolean expressions to Truth tables

- **To compute a truth table given a Boolean expression:**
  - Evaluate the function for every combination of inputs.

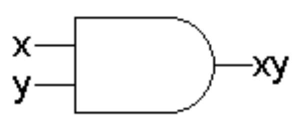
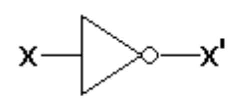
$f(x,y,z) = (x + y')z + x'$

f(0,0,0)	= (0 + 1)0 + 1	= 1
f(0,0,1)	= (0 + 1)1 + 1	= 1
f(0,1,0)	= (0 + 0)0 + 1	= 1
f(0,1,1)	= (0 + 0)1 + 1	= 1
f(1,0,0)	= (1 + 1)0 + 0	= 0
f(1,0,1)	= (1 + 1)1 + 0	= 1
f(1,1,0)	= (1 + 0)0 + 0	= 0
f(1,1,1)	= (1 + 0)1 + 0	= 1

x	y	z	f(x,y,z)
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

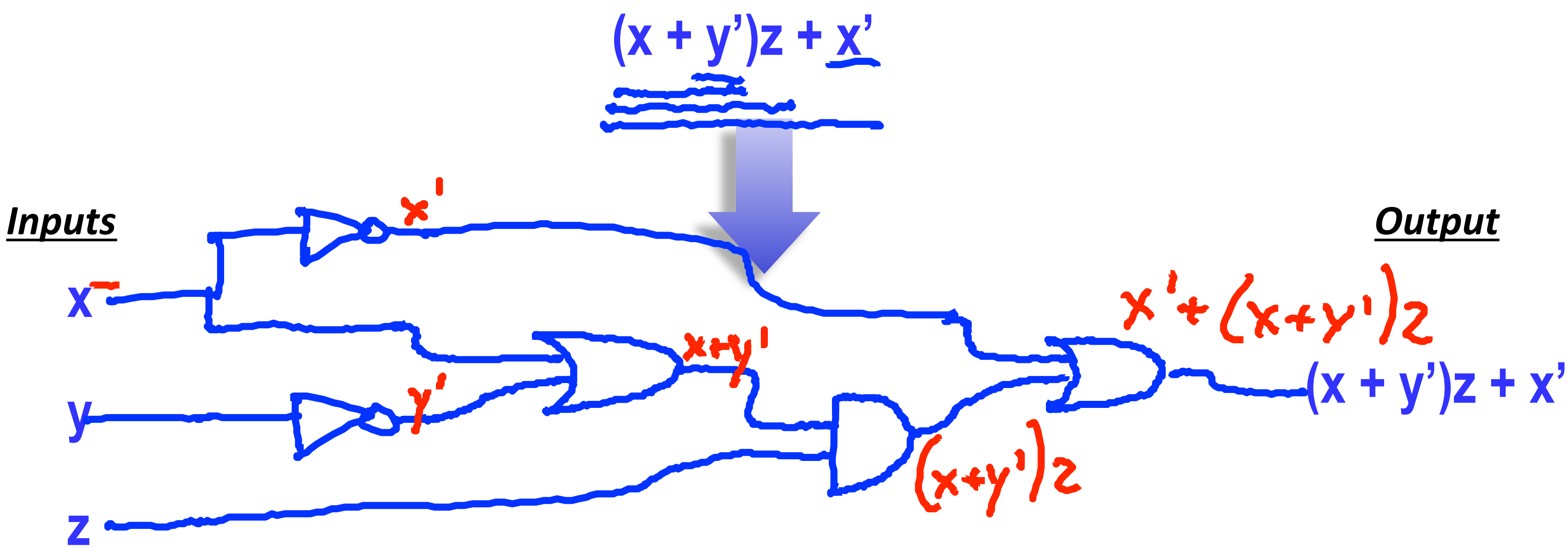
Primitive logic gates

- **Each of our basic operations can be implemented in hardware using a primitive logic gate.**
  - Symbols for each of the logic gates are shown below.
  - These gates output the product, sum or complement of their inputs.

Operation:	**AND (product) of two inputs**	**OR (sum) of two inputs**	**NOT (complement) on one input**
Expression:	**xy, or x•y**	**x + y**	**x’**
Logic gate:			

Boolean expressions to circuits

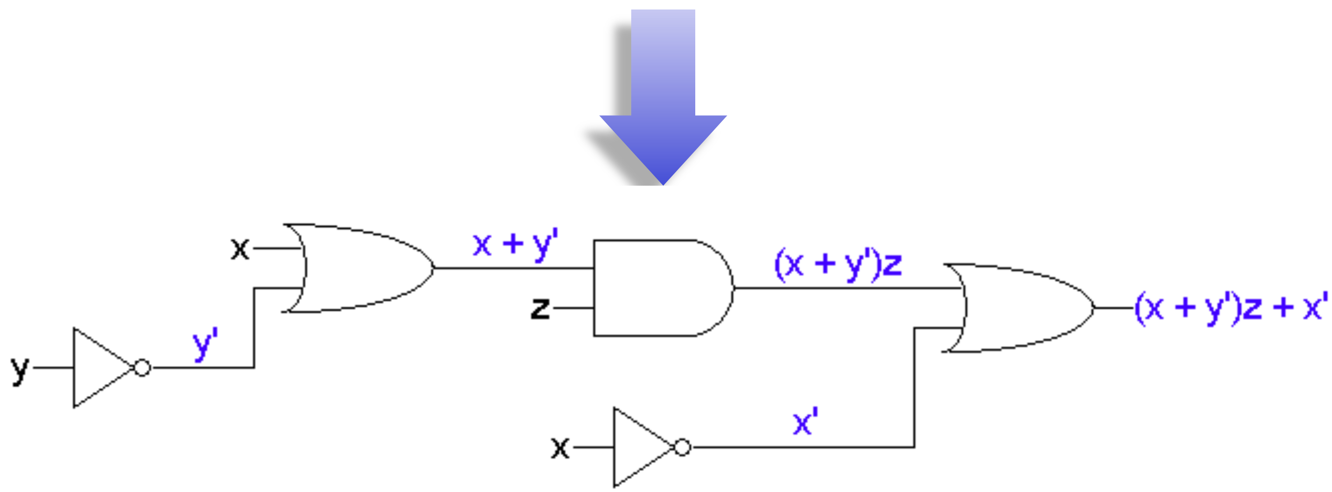
- *Any* Boolean expression can be converted into a circuit in a straightforward way.
  - Write a gate for each operation in the expression in precedence order.
  - We typically draw circuits with inputs on left and outputs on right.



Boolean expressions to circuits

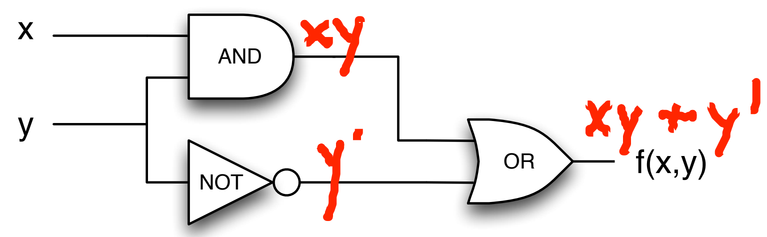
- ***Any* Boolean expression can be converted into a circuit in a straightforward way.**
  - Write a gate for each operation in expression in precedence order.
  - We typically draw circuits with inputs on left and outputs on right.

(x + y')z + x'



Converting circuits to expressions

- **What Boolean expression does this circuit implement?**



a) (x + y)y’

b) x + y + y’

c) xy’ + y

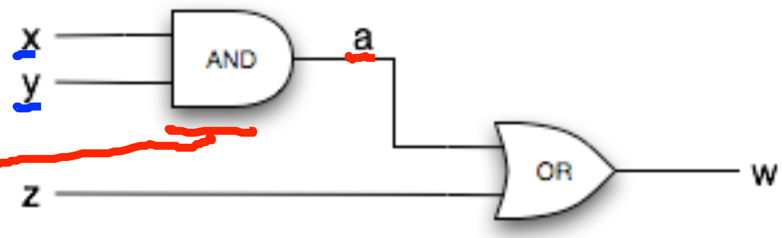
d) (xy) + y’

e) (x+y)(x+y’)

Hardware Description Languages (HDL)

- **Textual descriptions of circuits**
  - (We're very good at manipulating text…)

A Circuit:



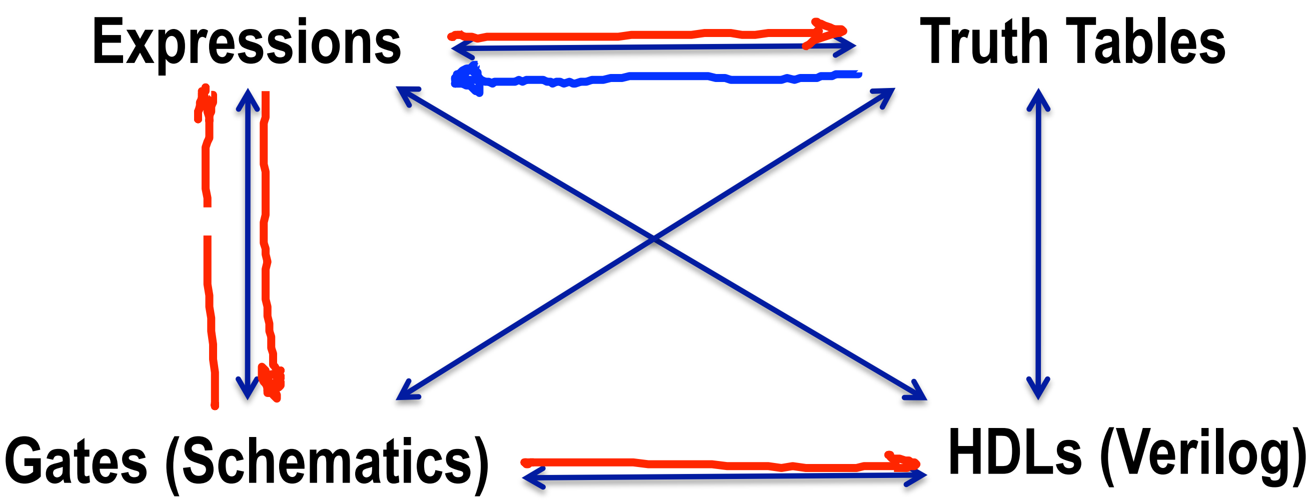
Verilog HDL Code:

```
wire  x, y, z, a, w;
and a1(a, x, y);        // gatetype name(out, in1, in2);
or  o1(w, a, z);
```

- **Not like a normal programming language**
  - Each statement describes one or more gates and/or wires.

Boolean functions summary

- **We can interpret high and low voltages as true and false.**
- **A Boolean variable can be either 1 or 0.**
- **AND, OR, and NOT are the basic Boolean operations.**
- **We can express Boolean functions in many ways:**
  - Expressions, truth tables, circuits, and HDL code
  - These are different representations for equivalent things



Discussion Section starts this week!

- **We'll introduce you to the tools designing, testing, and debugging digital logic circuits**
  - Verilog
  - Waveform Viewers

Class Organization

- **Piazza**
- **Weekly Labs**
- **3 Exams**
  - 2nd chance testing
- **Short final, not yet scheduled**
- **Course web page:**
  - https://wiki.engr.illinois.edu/display/cs398fa12