Boolean Algebra – Relationship with Digital Design

- To understand the benefits of "logic gates" vs. switches, we should first understand Boolean algebra
- "Traditional" algebra
  - Variables represent real numbers
  - Operators operate on variables, return real numbers

Boolean Algebra – Relationship with Digital Design

- ***Boolean Algebra***
  - Variables represent 0 or 1 only
  - Operators return 0 or 1 only
  - Basic operators
    - AND: *a AND b* returns 1 only when both a=1 and b=1
    - OR: *a OR b* returns 1 if either (or both) a=1 or b=1
    - NOT: *NOT a* returns the opposite of a (1 if a=0, 0 if a=1)

Boolean Algebra – Relationship with Digital Design

- Developed mid-1800's by George Boole to formalize human thought
  - Ex: "I'll go to lunch if Mary goes OR John goes, AND Sally does not go."
    - Let F represent my going to lunch (1 means I go, 0 I don't go)
    - Likewise, m for Mary going, j for John, and s for Sally
    - Then **F = (m OR j) AND NOT(s)**
  - Nice features
    - Formally evaluate
      - m=1, j=0, s=1 --> F = (1 OR 0) AND NOT(1) = 1 AND 0 = **0**
    - Formally transform
      - F = (m and NOT(s)) OR (j and NOT(s))
        - » Looks different, but same function
        - » We'll show transformation techniques soon

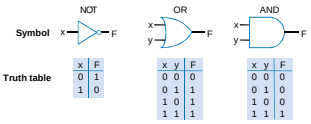
Relating Boolean Algebra to Digital Design

Boolean algebra (mid-1800s) — *Boole's intent: formalize human thought*

Switches (1930s) — *For telephone switching and other electronic uses*

Shannon (1938) — *Showed application of Boolean algebra to design of switch-based circuits*

Digital design



- Implement Boolean operators using transistors
  - Call those implementations ***logic gates***.
  - ***Let's us build circuits by doing math*** -- powerful concept

Boolean Algebra

- By defining logic gates based on Boolean algebra, we can use algebraic methods to manipulate circuits.
- Different from regular algebra, but use the same symbols.
- While symbols come from regular algebra, don't say "times" or "plus" – but "and" or "or". However, we do use "sums" and "products".

Boolean Algebra (cont.)

- Algebraic tools allow:
  - simplifying equations, thus circuits that are represented by them;
  - checking equivalency between two equations;
  - easily inverting an equation;
- Truth tables are tabular definitions of a Boolean function

Equation Notation for Boolean Operations

a and b	a or b	not a
a AND b	a OR b	NOT a
a * b	a + b	a’
a · b	a | b	‘a
ab		$\overline{a}$
a & b		~ a

Verilog notation is illustrated in red.

Equation Notation Examples

f = a or (b and (not c)) and d
f = a + (b · (‘c)) · d
f = a + b’c d
f = a + b $\overline{c}$ d

Verilog (continuous assignment):
assign f = a | (b & (~c)) & d;
assign f = a | b & ~c & d;

Boolean Algebra Precedence

Highest Precedence: top to bottom

Symbol	Name	Description
()	parenthesis	Evaluate expressions nested in parenthesis first
‘	NOT	Evaluate from left to right
*	AND	Evaluate from left to right
+	OR	Evaluate from left to right

Boolean Algebra Terminology

- Example equation: **F(a,b,c) = a’bc + abc’ + ab + c**
- *Variable* – represents a value (0 or 1)
  - three variables: **a, b, c**
- *Literal* – the *appearance* of a variable, in true or complemented form
  - Nine literals (appearances): **a’, b, c, a, b, c’, a, b, c**
- *Product term* – a product of literals
  - Four product terms: **a’bc, abc’, ab, c**
- *Sum of Products* (SOP) – equation written as the OR of product terms, as described above
  - Not SOP equation: **F = (ab + bc)(b + c)**