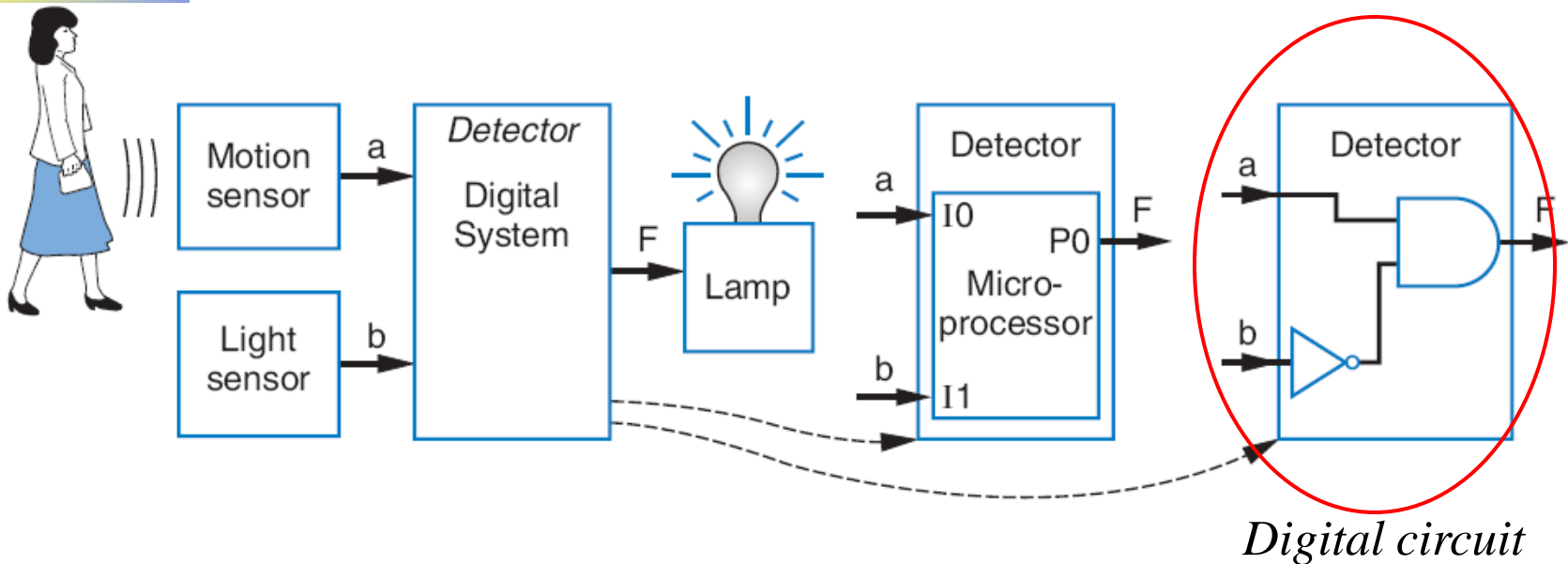




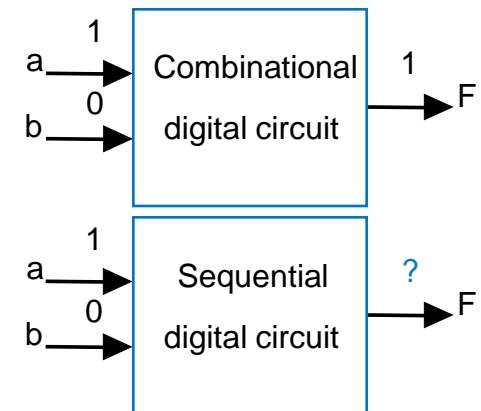
Slides to accompany the textbook *Digital Design*, First Edition,
by Frank Vahid, John Wiley and Sons Publishers, 2007.
<http://www.ddvahid.com>

Instructors of courses requiring Vahid's Digital Design textbook (published by John Wiley and Sons) have permission to modify and use these slides for customary course-related activities, subject to keeping this copyright notice in place and unmodified. These slides may be posted as unanimated pdf versions on publicly-accessible course websites. PowerPoint source (or pdf with animations) may not be posted to publicly-accessible websites, but may be posted for students on internal protected sites or distributed directly to students by other electronic means. Instructors may make printouts of the slides available to students for a reasonable photocopying charge, without incurring royalties. Any other use requires explicit permission. Instructors may obtain PowerPoint source or obtain special use permissions from Wiley – see <http://www.ddvahid.com> for information.

Introduction



- Let's learn to design digital circuits
- We'll start with a simple form of circuit:
 - **Combinational circuit**
 - A digital circuit whose outputs depend only on the present **combination** of the circuit input values



Boolean Algebra and its Relation to Digital Circuits

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

- **Boolean Algebra**

- Variables represent 0 or 1 only
- Operators return 0 or 1 only
- Basic operators
 - AND: $a \text{ AND } b$ returns 1 only when both $a=1$ and $b=1$
 - OR: $a \text{ OR } b$ returns 1 if either (or both) $a=1$ or $b=1$
 - NOT: $\text{NOT } a$ returns the opposite of a (1 if $a=0$, 0 if $a=1$)

a	b	AND
0	0	0
0	1	0
1	0	0
1	1	1

a	b	OR
0	0	0
0	1	1
1	0	1
1	1	1

a	NOT
0	1
1	0



Boolean Algebra and its Relation to Digital Circuits

- 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 **$F = (m \text{ OR } j) \text{ AND NOT}(s)$**
 - Nice features
 - Formally evaluate
 - $m=1, j=0, s=1 \rightarrow F = (1 \text{ OR } 0) \text{ AND NOT}(1) = 1 \text{ AND } 0 = \underline{0}$
 - Formally transform
 - $F = (m \text{ and NOT}(s)) \text{ OR } (j \text{ and NOT}(s))$
 - » Looks different, but same function
 - » We'll show transformation techniques soon

a	b	AND
0	0	0
0	1	0
1	0	0
1	1	1

a	b	OR
0	0	0
0	1	1
1	0	1
1	1	1

a	NOT
0	1
1	0



Evaluating Boolean Equations

- Evaluate the Boolean equation **$F = (a \text{ AND } b) \text{ OR } (c \text{ AND } d)$** for the given values of variables a, b, c, and d:
 - Q1: $a=1, b=1, c=1, d=0$.
 - Answer: $F = (1 \text{ AND } 1) \text{ OR } (1 \text{ AND } 0) = 1 \text{ OR } 0 = 1$.
 - Q2: $a=0, b=1, c=0, d=1$.
 - Answer: $F = (0 \text{ AND } 1) \text{ OR } (0 \text{ AND } 1) = 0 \text{ OR } 0 = 0$.
 - Q3: $a=1, b=1, c=1, d=1$.
 - Answer: $F = (1 \text{ AND } 1) \text{ OR } (1 \text{ AND } 1) = 1 \text{ OR } 1 = 1$.

a	b	AND
0	0	0
0	1	0
1	0	0
1	1	1

a	b	OR
0	0	0
0	1	1
1	0	1
1	1	1

a	NOT
0	1
1	0



Truth Table (Function Table)

$$F = (a \text{ AND } b) \text{ OR } (c \text{ AND } d)$$

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

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

Input values are usually arranged in ascending order of binary number value.

Truth Table (Function Table)

$$F = (a \text{ AND } b) \text{ OR } (c \text{ AND } d)$$

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

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

Converting to Boolean Equations

- Convert the following English statements to a Boolean equation
 - Q1. a is 1 and b is 1.
 - Answer: $F = a \text{ AND } b$
 - Q2. either of a or b is 1.
 - Answer: $F = a \text{ OR } b$
 - Q3. both a and b are not 0.
 - Answer:
 - (a) Option 1: $F = \text{NOT}(a) \text{ AND } \text{NOT}(b)$
 - (b) Option 2: $F = \text{NOT}(a \text{ OR } b)$
 - (c) Option 3: $\text{NOT } F = a \text{ or } b$
 - Q4. a is 1 and b is 0.
 - Answer: $F = a \text{ AND } \text{NOT}(b)$

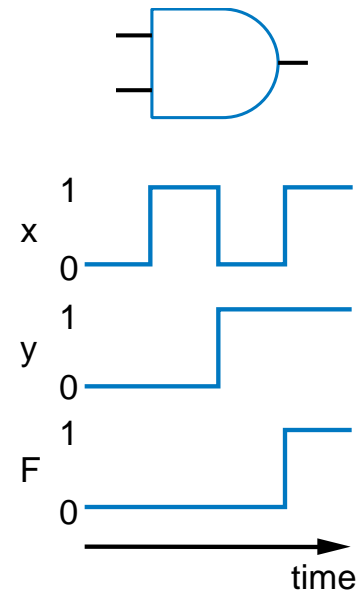
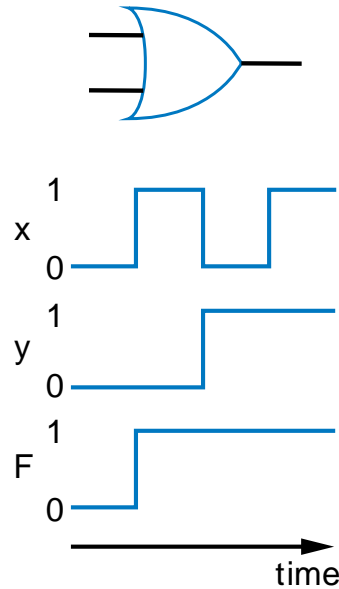
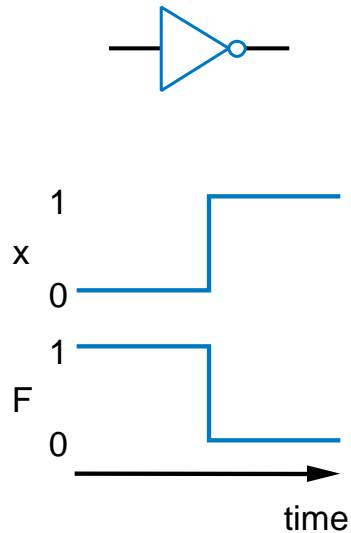


Converting to Boolean Equations

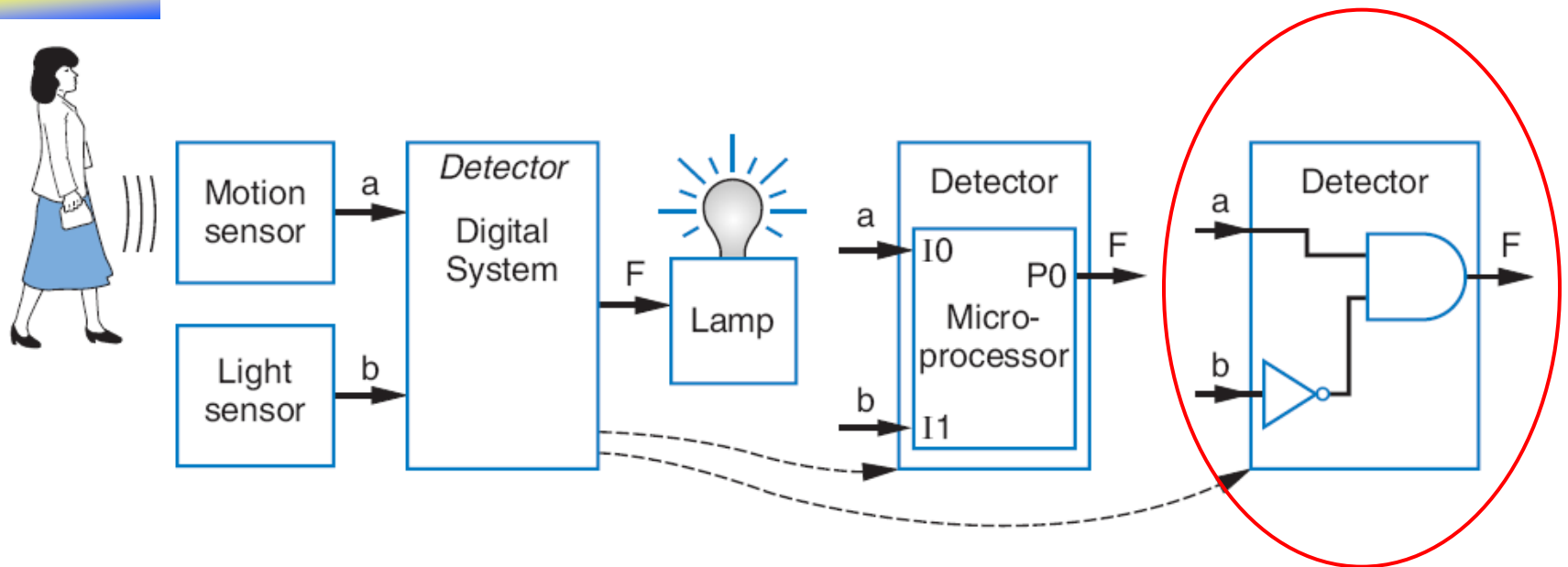
- Q1. A fire sprinkler system should spray water if high heat is sensed and the system is set to enabled.
 - Answer: Let Boolean variable h represent “high heat is sensed,” e represent “enabled,” and F represent “spraying water.” Then an equation is: $F = h \text{ AND } e$.
- Q2. A car alarm should sound if the alarm is enabled, and either the car is shaken or the door is opened.
 - Answer: Let a represent “alarm is enabled,” s represent “car is shaken,” d represent “door is opened,” and F represent “alarm sounds.” Then an equation is: $F = a \text{ AND } (s \text{ OR } d)$.
 - (a) Alternatively, assuming that our door sensor d represents “door is closed” instead of open (meaning $d=1$ when the door is closed, 0 when open), we obtain the following equation: $F = a \text{ AND } (s \text{ OR } \text{NOT}(d))$.



NOT/OR/AND Logic Gate Timing Diagrams



Building Circuits Using Gates

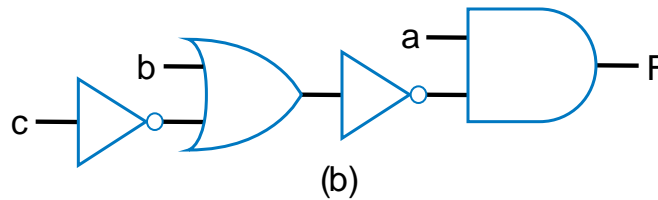
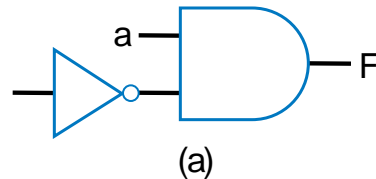


- Recall Chapter 1 motion-in-dark example
 - Turn on lamp ($F=1$) when motion sensed ($a=1$) and no light ($b=0$)
 - $F = a \text{ AND NOT}(b)$
 - Build using logic gates, AND and NOT, as shown
 - We just built our first digital circuit!



Example: Converting a Boolean Equation to a Circuit of Logic Gates

- Q: Convert the following equation to logic gates:
 $F = a \text{ AND NOT}(b \text{ OR NOT}(c))$

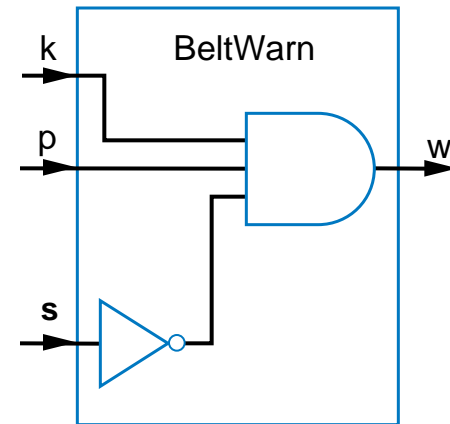


Example: Seat Belt Warning Light System

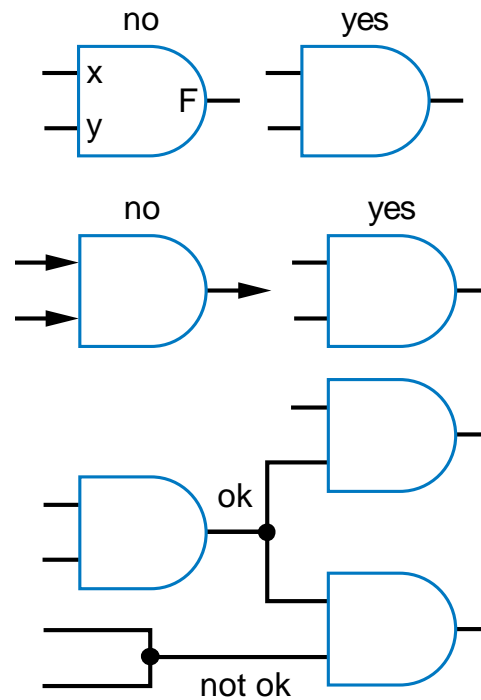
- Design circuit for warning light
- Sensors
 - $s=1$: seat belt fastened
 - $k=1$: key inserted
 - $p=1$: person in seat
- Capture Boolean equation
 - person in seat, and seat belt not fastened, and key inserted
- Convert equation to circuit
- Notice
 - Boolean algebra enables easy capture as equation and conversion to circuit
 - How design with switches?
 - Of course, logic gates are built from switches, but we think at level of logic gates, not switches



$$w = p \text{ AND NOT}(s) \text{ AND } k$$



Some Circuit Drawing Conventions



Boolean Algebra

- By defining logic gates based on Boolean algebra, we can use algebraic methods to manipulate circuits
 - So let's learn some Boolean algebraic methods
- Start with notation: Writing a AND b, a OR b, and NOT(a) is cumbersome
 - Use symbols: $a * b$, $a + b$, and a' (in fact, $a * b$ can be just ab).
 - Original: $w = (p \text{ AND NOT}(s) \text{ AND } k) \text{ OR } t$
 - New: $w = ps'k + t$
 - Spoken as “w equals p and s not and k, or t”
 - Or even just “w equals p s not k, or t”
 - s' known as “complement of s”
 - While symbols come from regular algebra, *don't* say “times” or “plus”

Boolean algebra precedence, highest precedence first.

Symbol	Name	Description
()	Parentheses	Evaluate expressions nested in parentheses first
'	NOT	Evaluate from left to right
*	AND	Evaluate from left to right
+	OR	Evaluate from left to right



Boolean Algebra Operator Precedence

- Evaluate the following Boolean equations, assuming $a=1$, $b=1$, $c=0$, $d=1$.
 - Q1. $F = a * b + c$.
 - Answer: $*$ has precedence over $+$, so we evaluate the equation as $F = (1 * 1) + 0 = (1) + 0 = 1 + 0 = 1$.
 - Q2. $F = ab + c$.
 - Answer: the problem is identical to the previous problem, using the shorthand notation for $*$.
 - Q3. $F = ab'$.
 - Answer: we first evaluate b' because NOT has precedence over AND, resulting in $F = 1 * (1') = 1 * (0) = 1 * 0 = 0$.
 - Q4. $F = (ac)'$.
 - Answer: we first evaluate what is inside the parentheses, then we NOT the result, yielding $(1*0)' = (0)' = 0' = 1$.
 - Q5. $F = (a + b') * c + d'$.
 - Answer: Inside left parentheses: $(1 + (1')) = (1 + (0)) = (1 + 0) = 1$. Next, $*$ has precedence over $+$, yielding $(1 * 0) + 1' = (0) + 1'$. The NOT has precedence over the OR, giving $(0) + (1') = (0) + (0) = 0 + 0 = 0$.



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, and c
- **Literal**
 - Appearance of a variable, in true or complemented form
 - Above equation has 9 literals: a', b, c, a, b, c', a, b, and c
- **Product term**
 - Product of literals
 - Four product terms: a'bc, abc', ab, c
- **Sum-of-products SOP**
 - Equation written as OR of product terms only
 - Above equation is in sum-of-products form.



Boolean Algebra Properties

- Commutative
 - $a + b = b + a$
 - $a * b = b * a$
- Distributive
 - $a * (b + c) = a * b + a * c$
 - $a + (b * c) = (a + b) * (a + c)$
 - (this one is tricky!)
- Associative
 - $(a + b) + c = a + (b + c)$
 - $(a * b) * c = a * (b * c)$
- Identity
 - $0 + a = a + 0 = a$
 - $1 * a = a * 1 = a$
- Complement
 - $a + a' = 1$
 - $a * a' = 0$
- To prove, just evaluate all possibilities

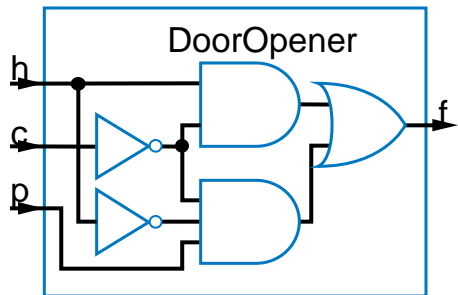
Example uses of the properties

- Show abc' equivalent to $c'ba$.
 - Use commutative property:
 - $a*b*c' = a*c'*b = c'*a*b = c'*b*a = c'ba$.
- Show $abc + abc' = ab$.
 - Use first distributive property
 - $abc + abc' = ab(c+c')$.
 - Complement property
 - Replace $c+c'$ by 1: $ab(c+c') = ab(1)$.
 - Identity property
 - $ab(1) = ab*1 = ab$.
- Show $x + x'z$ equivalent to $x + z$.
 - Second distributive property
 - Replace $x+x'z$ by $(x+x')*(x+z)$.
 - Complement property
 - Replace $(x+x')$ by 1,
 - Identity property
 - replace $1*(x+z)$ by $x+z$.



Example that Applies Boolean Algebra Properties

- Want automatic door opener circuit (e.g., for grocery store)
 - Output: $f=1$ opens door
 - Inputs:
 - $p=1$: person detected
 - $h=1$: switch forcing hold open
 - $c=1$: key forcing closed
 - Want open door when
 - $h=1$ and $c=0$, or
 - $h=0$ and $p=1$ and $c=0$
 - Equation: $f = hc' + h'pc'$



- Found inexpensive chip that computes:
 - $f = c'hp + c'hp' + c'h'p$
 - Can we use it?
 - Is it the same as $f = c'(p+h)$?
- Use Boolean algebra:

$$f = c'hp + c'hp' + c'h'p$$

$$f = c'h(p + p') + c'h'p \quad (\text{by the distributive property})$$

$$f = c'h(1) + c'h'p \quad (\text{by the complement property})$$

$$f = c'h + c'h'p \quad (\text{by the identity property})$$

$$f = hc' + h'pc' \quad (\text{by the commutative property})$$

Same!



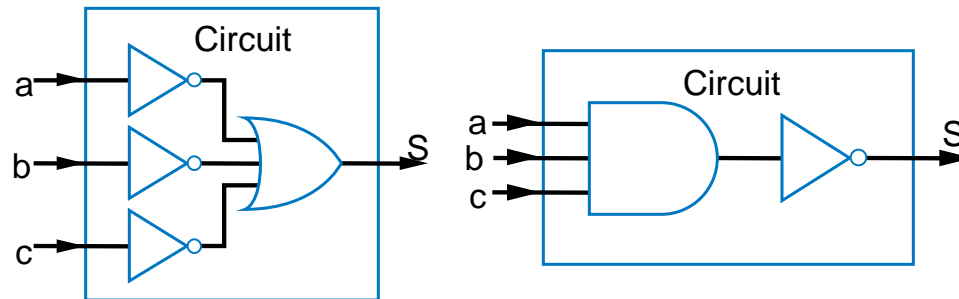
Boolean Algebra: Additional Properties

Aircraft lavatory sign example

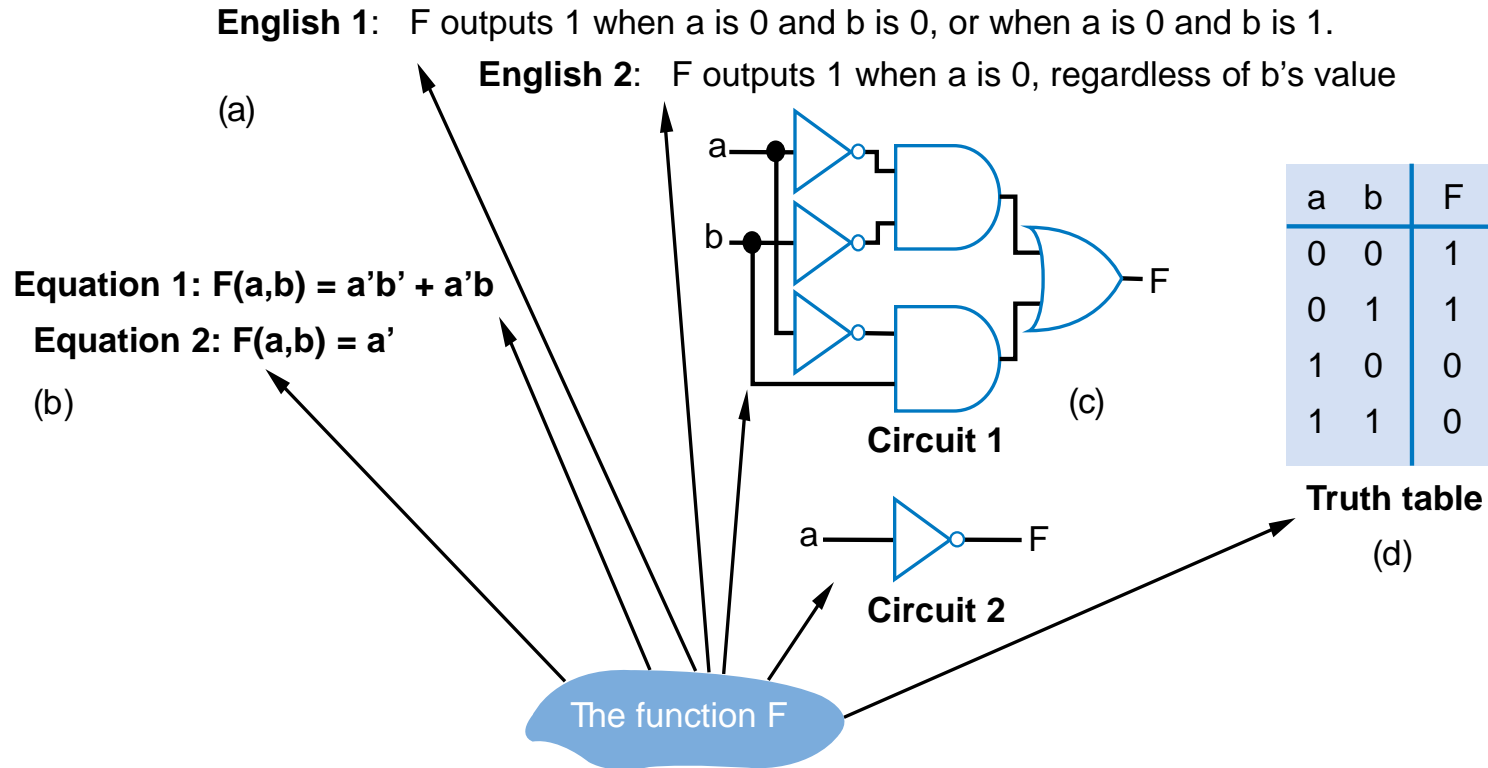
- Null elements
 - $a + 1 = 1$
 - $a * 0 = 0$
- Idempotent Law
 - $a + a = a$
 - $a * a = a$
- Involution Law
 - $(a')' = a$
- DeMorgan's Law
 - $(a + b)' = a'b'$
 - $(ab)' = a' + b'$
 - Very useful!
- To prove, just evaluate all possibilities

- Behavior
 - Three lavatories, each with sensor (a, b, c), equals 1 if door locked
 - Light “Available” sign (S) if any lavatory available
- Equation and circuit
 - $S = a' + b' + c'$
- Transform
 - $(abc)' = a' + b' + c'$ (by DeMorgan's Law)
 - $S = (abc)'$
- New equation and circuit

- Alternative: Instead of lighting “Available,” light “Occupied”
 - Opposite of “Available” function $S = a' + b' + c'$
 - So $S' = (a' + b' + c')'$
 - $S' = (a')' * (b')' * (c')'$ (by DeMorgan's Law)
 - $S' = a * b * c$ (by Involution Law)
 - Makes intuitive sense
 - Occupied if all doors are locked



Representations of Boolean Functions



- A function can be represented in different ways
 - Above shows seven representations of the same functions $F(a,b)$, using four different methods: English, Equation, Circuit, and Truth Table



Truth Table Representation of Boolean Functions

- Define value of F for each possible combination of input values
 - 2-input function: 4 rows
 - 3-input function: 8 rows
 - 4-input function: 16 rows
- Q: Use truth table to define function $F(a,b,c)$ that is 1 when abc is 5 or greater in binary

a	b	F
0	0	
0	1	
1	0	
1	1	

(a)

a	b	c	F
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

(b)

a	b	c	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

a

a	b	c	d	F
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

(c)



Converting among Representations

- Can convert from any representation to any other
- Common conversions
 - Equation to circuit (we did this earlier)
 - Truth table to equation (which we can convert to circuit)
 - Easy -- just OR each input term that should output 1
 - Equation to truth table
 - Easy -- just evaluate equation for each input combination (row)
 - Creating intermediate columns helps

Q: Convert to truth table: $F = a'b' + a'b$

Inputs				Output
a	b	$a'b'$	$a'b$	F
0	0	1	0	1
0	1	0	1	1
1	0	0	0	0
1	1	0	0	0

Inputs		Outputs	Term
a	b	F	F = sum of
0	0	1	$a'b'$
0	1	1	$a'b$
1	0	0	
1	1	0	

$$F = a'b' + a'b$$

Q: Convert to equation

a	b	c	F	
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	0	
1	0	0	0	
1	0	1	1	$ab'c$
1	1	0	1	abc'
1	1	1	1	abc

$$F = ab'c + abc' + abc$$

SOP Form



Standard Representation: Truth Table

- How can we determine if two functions are the same?
 - Recall automatic door example
 - Same as $f = hc' + h'pc'$?
 - Used algebraic methods
 - But if we failed, does that prove *not* equal? No.
- Solution: Convert to truth tables
 - Only ONE truth table representation of a given function
 - **Standard** representation -- for given function, only one version in standard form exists

$$f = c'h p + c'h p' + c'h'$$

$$f = c'h(p + p') + c'h'p$$

$$f = c'h(1) + c'h'p$$

$$f = c'h + c'h'p$$

(what if we stopped here?)

$$f = hc' + h'pc'$$

Q: Determine if $F=ab+a'$ is same function as $F=a'b'+a'b+ab$, by converting each to truth table first

F = ab + a'			F = a'b' + a'b + ab		
a	b	F	a	b	F
0	0	1	0	0	1
0	1	1	0	1	1
1	0	0	1	0	0
1	1	1	1	1	1

Same

a



Homework

- Chapter 2: 10,13,21,27,28.
- Due date: Thursday, January 21

