

COM 251 Logic Design and Circuits

Combinational Logic Design: Logic Gates and Boolean Algebra

Prof. Dr. Halûk Gümüşkaya

haluk.gumuskaya@gediz.edu.tr

haluk@gumuskaya.com http://www.gumuskaya.com

Computer Engineering Department

GEDİZÜNİVERSİTESİ
izmir

Thursday, October 27, 2011

1

Acknowledgement

The slides have been based in-part upon original slides of a number of books including:

- *Digital Design with RTL Design, Verilog and VHDL*, 2nd ed., Frank Vahid, John Wiley, 2011.

2

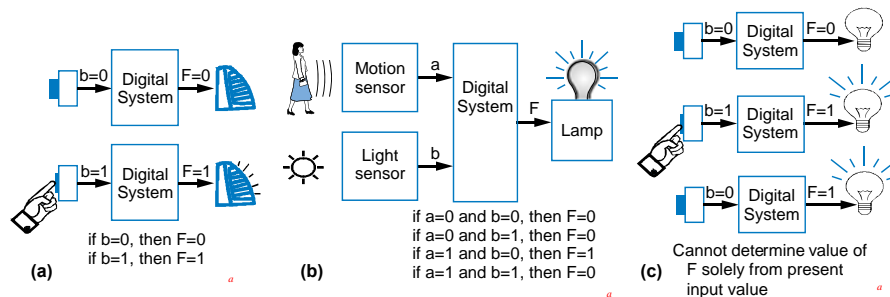
2.1

Introduction

- Let's learn to design digital circuits, starting with a simple form of circuit:

– **Combinational circuit**

- Outputs depend solely on the **present combination** of the circuit inputs' values
- Vs. sequential circuit: Has "memory" that impacts outputs too



Note: Slides with animation are denoted with a small red "a" near the animated items

3

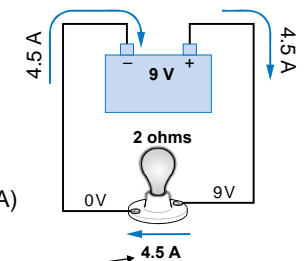
2.2

Switches

- Electronic switches are the basis of binary digital circuits

– Electrical terminology

- **Voltage:** Difference in electric potential between two points (volts, V)
 - Analogous to water pressure
- **Resistance:** Tendency of wire to resist current flow (ohms, Ω)
 - Analogous to water pipe diameter
- **Current:** Flow of charged particles (amps, A)
 - Analogous to water flow
- $V = I * R$ (Ohm's Law)
 - $9 V = I * 2 \text{ ohms}$
 - $I = 4.5 A$

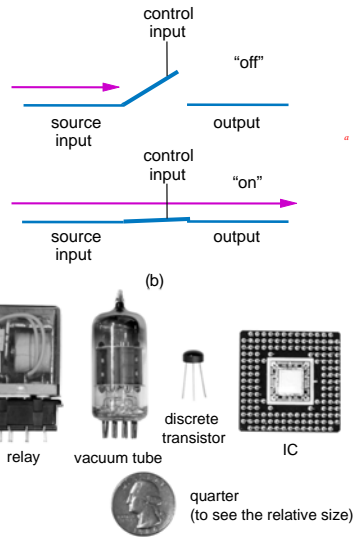


If a 9V potential difference is applied across a 2 ohm resistor, then 4.5 A of current will flow.

4

Switches

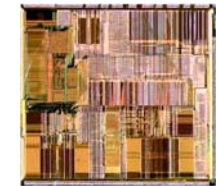
- A switch has three parts
 - Source input, and output
 - Current tries to flow from source input to output
 - Control input
 - Voltage that controls whether that current can flow
- The amazing shrinking switch
 - 1930s: Relays
 - 1940s: Vacuum tubes
 - 1950s: Discrete transistor
 - 1960s: Integrated circuits (ICs)
 - Initially just a few transistors on IC
 - Then tens, hundreds, thousands...



5

Moore's Law

- IC capacity doubling about every 18 months for several decades
 - Known as "Moore's Law" after Gordon Moore, co-founder of Intel
 - Predicted in 1965 predicted that components per IC would double roughly every year or so
 - Book cover depicts related phenomena
 - For a particular number of transistors, the IC area shrinks by half every 18 months
 - Consider how much shrinking occurs in just 10 years (try drawing it)
 - Enables incredibly powerful computation in incredibly tiny devices
 - Today's ICs hold *billions* of transistors
 - The first Pentium processor (early 1990s) needed only 3 million

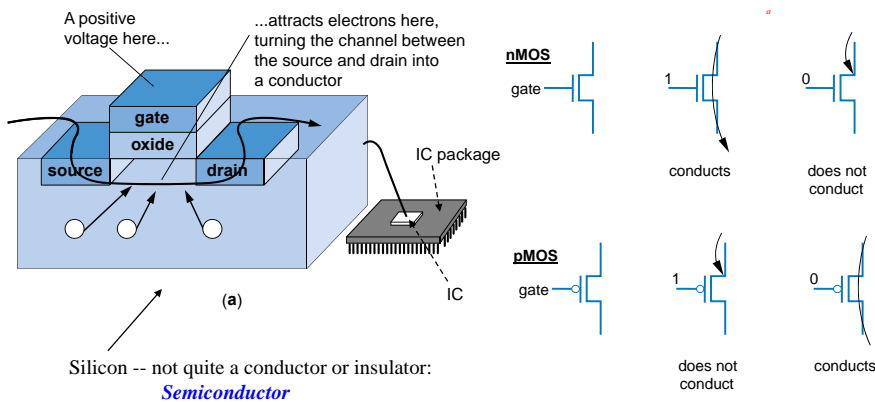


An Intel Pentium processor IC having millions of transistors

6

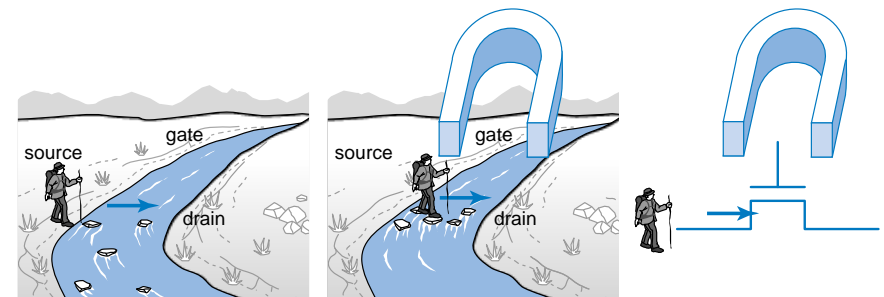
The CMOS Transistor

- CMOS transistor
 - Basic switch in modern ICs



7

CMOS Transistor Analogy



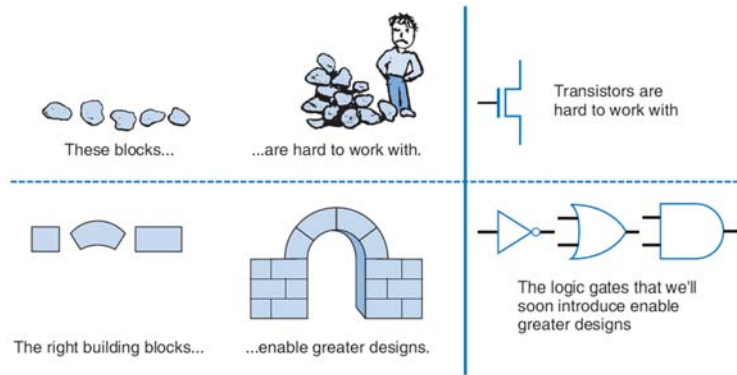
8

Boolean Logic Gates

Building Blocks for Digital Circuits

(Because Switches are Hard to Work With)

2.4



- “Logic gates” are better digital circuit building blocks than switches (transistors)
 - Why?...

9

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
 - Variables represent real numbers (x, y)
 - Operators operate on variables, return real numbers (2.5*x + y - 3)
- **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

10

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 OR j) AND NOT(s)**
 - Nice features
 - Formally evaluate
 - $m=1, j=0, s=1 \rightarrow F = (1 \text{ OR } 0) \text{ AND NOT}(1) = 1 \text{ AND } 0 = 0$
 - 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
 - Formally prove
 - Prove that if Sally goes to lunch ($s=1$), then I don't go ($F=0$)
 - $F = (m \text{ OR } j) \text{ AND NOT}(1) = (m \text{ OR } j) \text{ AND } 0 = 0$

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

11

Evaluating Boolean Equations

- Evaluate the Boolean equation **F = (a AND b) OR (c 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

12

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. a is 1 and b is 0.
 - Answer: $F = a \text{ AND NOT}(b)$
 - Q4. a is not 0.
 - Answer:
 - (a) Option 1: $F = \text{NOT}(\text{NOT}(a))$
 - (b) Option 2: $F = a$

13

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: $F = a \text{ AND } (s \text{ OR NOT}(d))$.

14

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

x	F
0	1
1	0

x	y	F
0	0	0
0	1	1
1	0	1
1	1	1

x	y	F
0	0	0
0	1	0
1	0	0
1	1	1

1 and 0 each actually corresponds to a voltage range

1.8 V "1"
1.2 V
0.6 V
0V "0"

Next slides show how these circuits work.
Note: The above OR/AND implementations are inefficient; we'll show why, and show better ones, later.

15

NOT gate

The NOT gate symbol is a triangle with a small circle at its tip. The circuit diagram shows a single transistor with its base connected to the input x, its emitter to ground, and its collector to the output F. When the input x is 0, the transistor is off, and the output F is 1. When the input x is 1, the transistor is on, and the output F is 0.

x	F
0	1
1	0

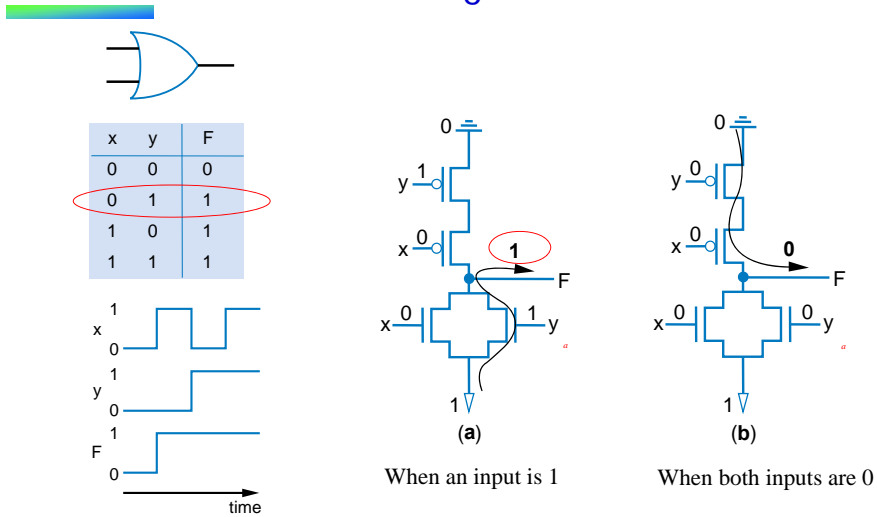
(a) When the input is 0

(b) When the input is 1

Timing diagram showing input x and output F over time. When x transitions from 0 to 1, F transitions from 1 to 0.

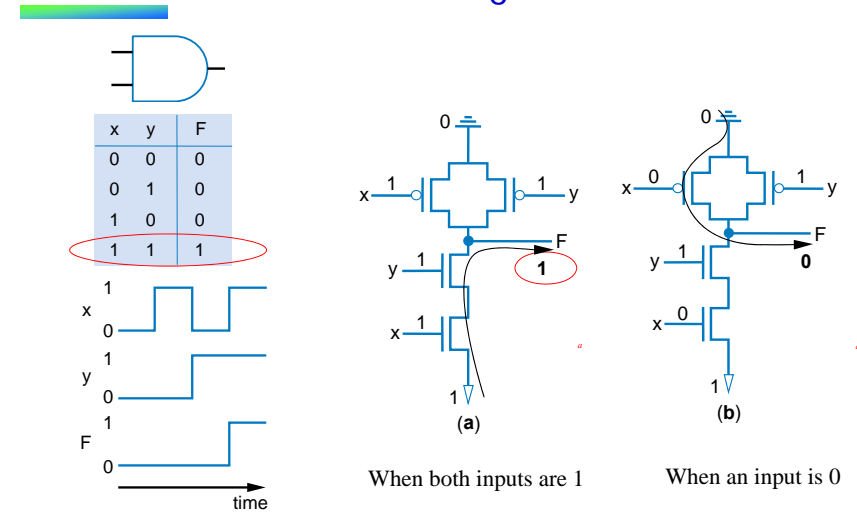
16

OR gate



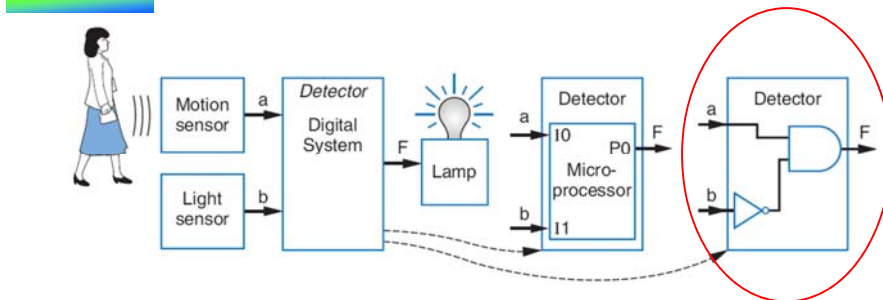
17

AND gate



18

Building Circuits Using Gates



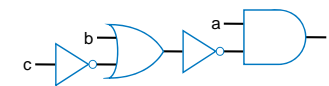
- Recall Chapter 1 motion-in-dark example
 - Turn on lamp ($F=1$) when motion sensed ($a=1$) and no light ($b=0$)
 - $F = a \text{ AND NOT}(b)$
 - Build using logic gates, AND and NOT, as shown
 - We just built our first digital circuit!

19

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

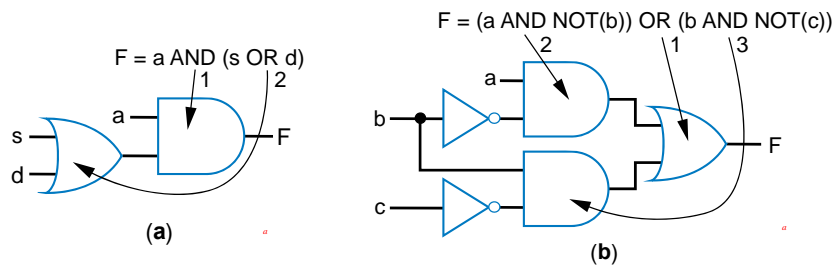
Start from the output, work back towards the inputs

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



20

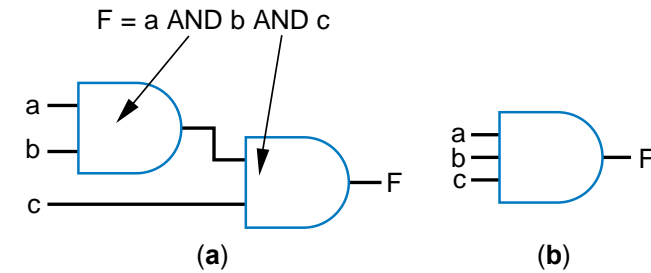
More examples



Start from the output, work back towards the inputs

21

Using gates with more than 2 inputs

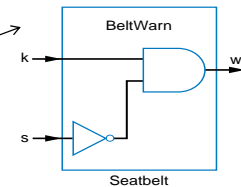


22

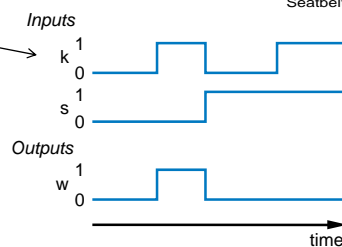
Example: Seat Belt Warning Light System

- Design circuit for warning light
- Sensors
 - $s=1$: seat belt fastened
 - $k=1$: key inserted
- Capture Boolean equation
 - seat belt not fastened, and key inserted
- Convert equation to circuit

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



- *Timing diagram* illustrates circuit behavior
 - We set inputs to any values
 - Output set according to circuit

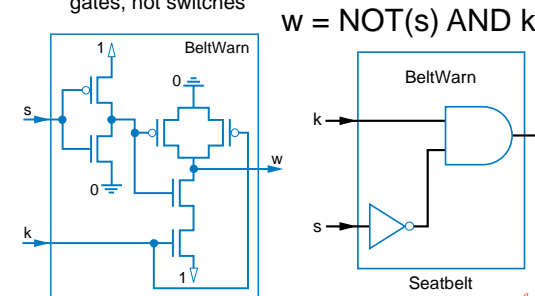
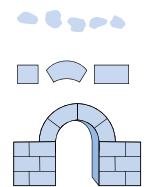


23

Gates vs. switches

Notice

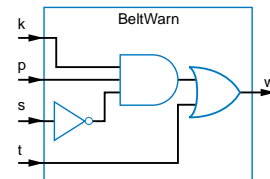
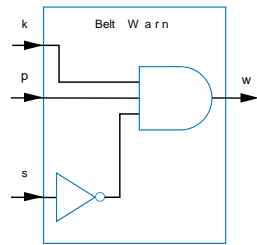
- Boolean algebra enables easy capture as equation and conversion to circuit
 - How design with switches?
 - Of course, logic gates are built from switches, but we think at level of logic gates, not switches



24

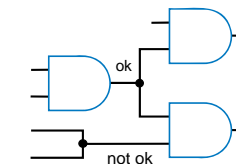
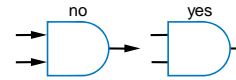
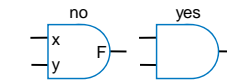
More examples: Seat belt warning light extensions

- Only illuminate warning light if person is in the seat ($p=1$), and seat belt not fastened and key inserted
- $w = p \text{ AND NOT}(s) \text{ AND } k$
- Given $t=1$ for 5 seconds after key inserted. Turn on warning light when $t=1$ (to check that warning lights are working)
- $w = (p \text{ AND NOT}(s) \text{ AND } k) \text{ OR } t$



25

Some Gate-Based Circuit Drawing Conventions



26

2.5

Boolean Algebra

- By defining logic gates based on Boolean algebra, we can *use algebraic methods to manipulate circuits*
- Notation: Writing a AND b, a OR b, NOT(a) is cumbersome
 - Use symbols: $a * b$ (or just ab), $a + b$, and a'
 - 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 prime and k, or t”
 - Or just “w equals p s prime k, or t”
 - s' known as “complement of s”
 - While symbols come from regular algebra, **don't** say “times” or “plus”
 - “product” and “sum” are OK and commonly used

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

27

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 $(0) + (1') = (0) + (0) = 0 + 0 = 0$.

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

28

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
 - Nine 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**
 - Equation written as OR of product terms only
 - Above equation is in sum-of-products form. "F = (a+b)c + d" is not.

29

Boolean Algebra Properties

- Commutative
 - $a + b = b + a$
 - $a * b = b * a$
- Distributive
 - $a * (b + c) = a * b + a * c$
 - Can write as: $a(b+c) = ab + ac$
 - $a + (b * c) = (a + b) * (a + c)$
 - (This second one is tricky!)
 - Can write as: $a+(bc) = (ab)(ac)$
- 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

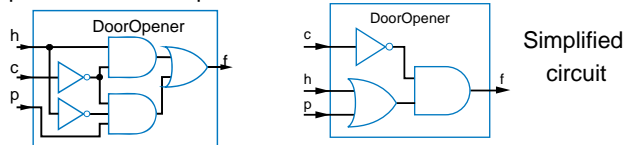
Example uses of the properties

- Show abc' equivalent to c'ba.
 - Use commutative property:
 - $a*b*c' = a*c'*b = c'*a*b = c'*b*a$
- 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 $x+z$.

30

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: $f = hc' + h'pc'$
- Can the circuit be simplified?
 - $f = hc' + h'pc'$
 - $f = c'h + c'h'p$ (by the commutative property)
 - $f = c'(h + h'p)$ (by the first distrib. property)
 - $f = c'((h+h')(h+p))$ (2nd distrib. prop.; tricky one)
 - $f = c'((1)*(h+p))$ (by the complement property)
 - $f = c'(h+p)$ (by the identity property)

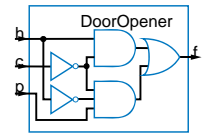


Simplification of circuits is covered in Sec. 2.11 / Sec 6.2.

31

Example that Applies Boolean Algebra Properties

- Found inexpensive chip that computes:
 - $f = c'hp + c'hp' + c'h'p$
 - Can we use it for the door opener?
 - Is it the same as $f = hc' + h'pc'$?
- Apply Boolean algebra:
 - Commutative
 - $a + b = b + a$
 - $a * b = b * a$
 - Distributive
 - $a * (b + c) = a * b + a * c$
 - $a + (b * c) = (a + b) * (a + c)$
 - 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
 - $a + a' = 1$
 - $a * a' = 0$



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

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

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

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

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

Same! Yes, we can use it.

32

Boolean Algebra: Additional Properties

- 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

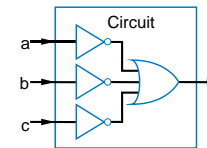
33

Example Applying DeMorgan's Law

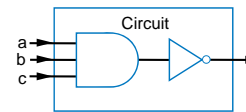
$$(a + b)' = a'b'$$

$$(ab)' = a' + b'$$

Aircraft lavatory sign example



- 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 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

34

Example Applying Properties

- Commutative
 - $a + b = b + a$
 - $a * b = b * a$
- Distributive
 - $a * (b + c) = a * b + a * c$
 - $a + (b * c) = (a + b) * (a + c)$
- 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
 - $a + a' = 1$
 - $a * a' = 0$
- Null elements
 - $a + 1 = 1$
 - $a * 0 = 0$
- Idempotent Law
 - $a + a = a$
 - $a * a = a$
- Involution Law
 - $(a)' = a$
- DeMorgan's Law
 - $(a + b)' = a'b'$
 - $(ab)' = a' + b'$
- For door opener $f = c'(h+p)$, prove door stays closed ($f=0$) when $c=1$
 - $f = c'(h+p)$
 - Let $c = 1$ (door forced closed)
 - $f = 1'(h+p)$
 - $f = 0(h+p)$
 - $f = 0h + 0p$ (by the distributive property)
 - $f = 0 + 0$ (by the null elements property)
 - $f = 0$

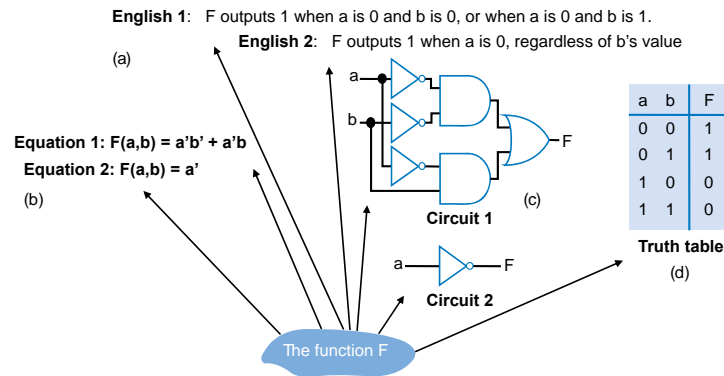
35

Complement of a Function

- Commonly want to find complement (inverse) of function F
 - 0 when F is 1; 1 when F is 0
- Use DeMorgan's Law repeatedly
 - Note: DeMorgan's Law defined for more than two variables, e.g.:
 - $(a + b + c)' = (abc)'$
 - $(abc)' = (a' + b' + c')$
- Complement of $f = w'xy + wx'y'z'$
 - $f' = (w'xy + wx'y'z)'$
 - $f' = (w'xy)'(wx'y'z)'$ (by DeMorgan's Law)
 - $f' = (w+x'+y')(w'+x+y+z)$ (by DeMorgan's Law)
- Can then expand into sum-of-products form

36

Representations of Boolean Functions



- A function can be represented in different ways
 - Above shows **7 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	

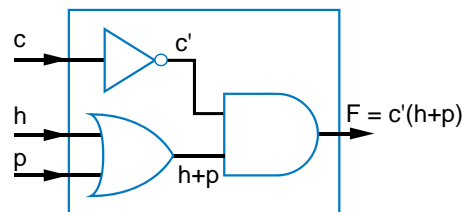
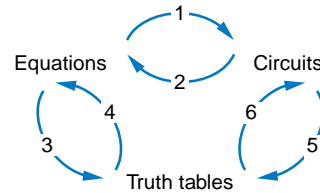
(b)

a	b	c	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)

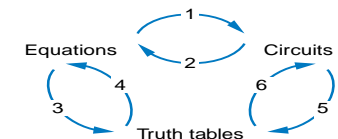
Converting among Representations

- Can convert from any representation to another
- Common conversions
 - Equation to circuit (we did this earlier)
 - Circuit to equation
 - Start at inputs, write expression of each gate output



Converting among Representations

- More common conversions
 - Truth table to equation (which we can then 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



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 truth table: $F = a'b' + a'b$

Inputs	Outputs	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

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
1	1	0	1
1	1	1	1

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

Example: Converting from Truth Table to Equation

- **Parity bit:** Extra bit added to data, intended to enable detection of error (a bit changed unintentionally)
 - e.g., errors can occur on wires due to electrical interference
- **Even parity:** Set parity bit so total number of 1s (data + parity) is even
 - e.g., if data is 001, parity bit is 1 → 0011 has even number of 1s
- Want equation, but easiest to start from truth table for this example

a	b	c	P
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

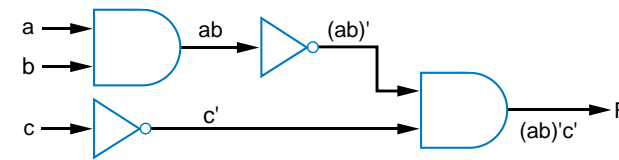
Convert to eqn.

$$P = a'b'c + a'bc' + ab'c' + abc$$

41

Example: Converting from Circuit to Truth Table

- First convert to circuit to equation, then equation to table



Inputs						Outputs
a	b	c	ab	(ab)'	c'	F
0	0	0	0	1	1	1
0	0	1	0	1	0	0
0	1	0	0	1	1	1
0	1	1	0	1	0	0
1	0	0	0	1	1	1
1	0	1	0	1	0	0
1	1	0	1	0	1	0
1	1	1	1	0	0	0

42

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

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

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

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

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

(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	1	1	1
1	0	0	1	0	0
1	1	1	1	1	1

Same

43

Truth Table Canonical Form

- Q: Determine via truth tables whether $ab+a'$ and $(a+b)'$ are equivalent

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

Not equivalent

44