Introduction to Digital Design

Week 4: Boolean Algebra, Multi-Output Circuits

Yao Zheng
Assistant Professor
University of Hawai'i at Mānoa
Department of Electrical Engineering

Overview

- Boolean Operator
  - (), ', *, +.
  - precedence.
- Terminology
  - variable, literal, product term, sum-of-products.
- Properties
  - Commutative, distributive, associative, identity, complement.
  - Null elements, idempotent, involution, DeMorgan.
- Representation
  - Truth table, Boolean equation, circuit.
  - Canonical form, multiple-output circuits.

2

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 AND NOT(s) AND k) 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

3

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 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

4

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.

5

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.

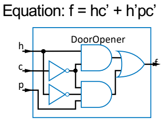
6

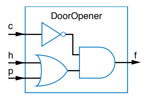
Example that Applies Boolean Algebra Properties

- 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)





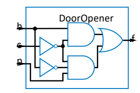
Simplified circuit

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

7

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 (by the distributive property)
f = c'h(1) + c'h'p (by the complement property)
f = c'h + c'h'p (by the identity property)
f = hc' + h'pc' (by the commutative property)
Same! Yes, we can use it.

8

Example that Applies Boolean Algebra Properties

Video

9

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

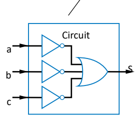
10

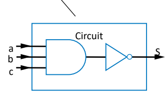
Example Applying DeMorgan's Law

(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

11

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

12

Complement of a Function

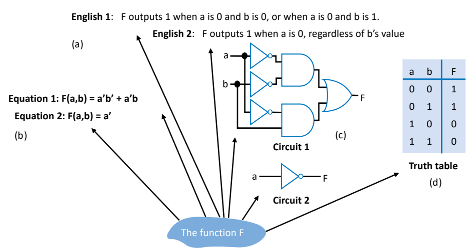
- Commonly want to find complement (inverse) of function F
  - 0 when F is 1;   1 when F is 0
- Use DeMorgan's Law repeatedly
  - Note: DeMorgan's Law defined for more than two variables, e.g.:
    - (a + b + c)' = (abc)'
    - (abc)' = (a' + b' + c')
- Complement of f = w'xy + wx'y'z'
  - f ' = (w'xy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

Example that Applies Boolean Algebra Properties

Video

2.6

Representations of Boolean Functions



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

Truth table

(d)

- 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)

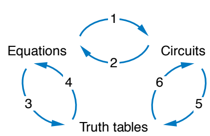
a	b	c	d	F
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

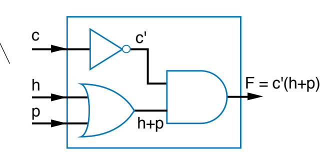
(c)

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**

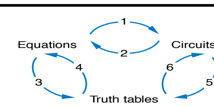
Converting among Representations

- 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 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

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**

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

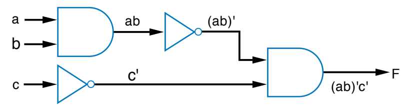
a	b	c	P
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Convert to eqn.

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

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

20

Example: Converting from Circuit to Truth Table

Video

21

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'		
a	b	F
0	0	1
0	1	1
1	0	0
1	1	1

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

Same

22

Truth Table Canonical Form

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

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

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

Not equivalent

23

Canonical Form – Sum of Minterms

- Truth tables too big for numerous inputs
- Use standard form of equation instead
  - Known as ***canonical form***
  - Regular algebra: group terms of polynomial by power
    - $ax^2 + bx + c$ ($3x^2 + 4x + 2x^2 + 3 + 1$ --> $5x^2 + 4x + 4$)
  - Boolean algebra: create sum of minterms
    - ***Minterm***: product term with every function literal appearing exactly once, in true or complemented form
    - Just multiply-out equation until sum of product terms
    - Then expand each term until all terms are minterms

Q: Determine if F(a,b)=ab+a' is equivalent to F(a,b)=a'b'+a'b+ab, by converting first equation to canonical form (second already is)

F = ab+a' (already sum of products)
F = ab + a'(b+b') (expanding term)
F = ab + a'b + a'b' (Equivalent – same three terms as other equation)

24

Canonical Form – Sum of Minterms

- Q: Determine whether the functions G(a,b,c,d,e) = abcd + a'bcde and H(a,b,c,d,e) = abcde + abcde' + a'bcde + a'bcde(a' + c) are equivalent.

G = abcd + a'bcde
G = abcd(e+e') + a'bcde
G = abcde + abcde' + a'bcde
G = a'bcde + abcde' + abcde *(sum of minterms form)*

Equivalent

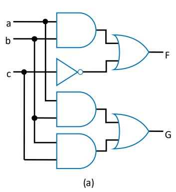
H = abcde + abcde' + a'bcde + a'bcde(a' + c)
H = abcde + abcde' + a'bcde + a'bcdea' + a'bcdec
H = abcde + abcde' + a'bcde + a'bcde + a'bcde
H = abcde + abcde' + a'bcde
H = a'bcde + abcde' + abcde

Compact Sum of Minterms Representation

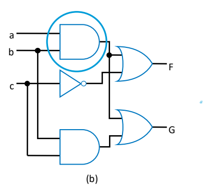
- List each minterm as a number
- Number determined from the binary representation of its variables' values
  - a'bcde corresponds to 01111, or 15
  - abcde' corresponds to 11110, or 30
  - abcde corresponds to 11111, or 31
- Thus, H = a'bcde + abcde' + abcde can be written as:
  - H = $\sum m(15,30,31)$
  - "H is the sum of minterms 15, 30, and 31"

Multiple-Output Circuits

- Many circuits have more than one output
- Can give each a separate circuit, or can share gates
- Ex: F = ab + c', G = ab + bc

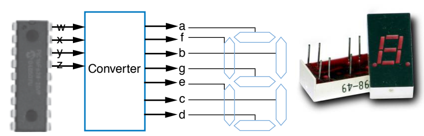


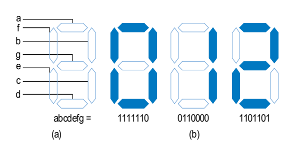
Option 1: Separate circuits



Option 2: Shared gates

Multiple-Output Example:
BCD to 7-Segment Converter

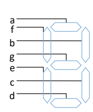




Multiple-Output Example:
BCD to 7-Segment Converter

TABLE 2-4 4-bit binary number to seven-segment display truth table

w	x	y	z	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1
1	0	1	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0



a = w'x'y'z' + w'x'yz' + w'x'yz + w'xy'z + w'xyz' + w'xyz + wx'y'z' + wx'y'z

b = w'x'y'z' + w'x'y'z + w'x'yz' + w'x'yz + w'xy'z' + w'xyz + wx'y'z' + wx'y'z

...

Example: Multiple-Output Circuits

Video

Summary

- Boolean Operator
  - (), ', *, +.
  - precedence.
- Terminology
  - variable, literal, product term, sum-of-products.
- Properties
  - Commutative, distributive, associative, identity, complement.
  - Null elements, idempotent, involution, DeMorgan.
- Representation
  - Truth table, Boolean equation, circuit.
  - Canonical form, multiple-output circuits.

31