



Digital Design

Chapter 2: Combinational Logic Design

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

Copyright © 2007 Frank Vahid

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.

Review Demorgan's Theorem

- $F = A'B$
 - Compliment terms: $= AB'$
 - Change AND to OR, OR to AND: $= A+B'$
 - Compliment entire function: $F = (A+B')'$, $F' = A+B'$
- $F = A'B + C'D'$
 - Compliment terms: $= (A'B)' + (C'D')'$
 - Change AND to OR, OR to AND: $= (A'B)'(C'D')'$
 - Compliment entire function: $F = ((A'B)'(C'D')')'$, $F' = (A'B)'(C'D')'$
- Theorem can be applied to portions of the equation
 - $F = (A+B')' + C'D'$
 - $F = (A+B')' + (C+D)'$
 - Apply theorem to above equation: $F' = (A+B') (C+D)$

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

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



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)



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



Canonical Form -- Sum of Minterms

- Truth tables too big for numerous inputs
- Use standard form of equation instead
 - Known as **canonical form**
 - 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 same function as $F(a,b)=a'b'+a'b+ab$, by converting first equation to canonical form (second already in canonical form)

$F = ab+a'$ (already sum of products)

$F = ab + a'(b+b')$ (expanding term)

$F = ab + a'b + a'b'$ (SAME -- same three terms as other equation)

a



Minterms

a	b	c	d	F
0	0	0	0	$a'b'c'd'$
0	0	0	1	$a'b'c'd$
0	0	1	0	$a'b'cd'$
0	0	1	1	$a'b'cd$
0	1	0	0	$a'bc'd'$
0	1	0	1	$a'bc'd$
0	1	1	0	$a'bcd'$
0	1	1	1	$a'bcd$
1	0	0	0	$ab'c'd'$
1	0	0	1	$ab'c'd$
1	0	1	0	$ab'cd'$
1	0	1	1	$ab'cd$
1	1	0	0	$abc'd'$
1	1	0	1	$abc'd$
1	1	1	0	$abcd'$
1	1	1	1	$abcd$

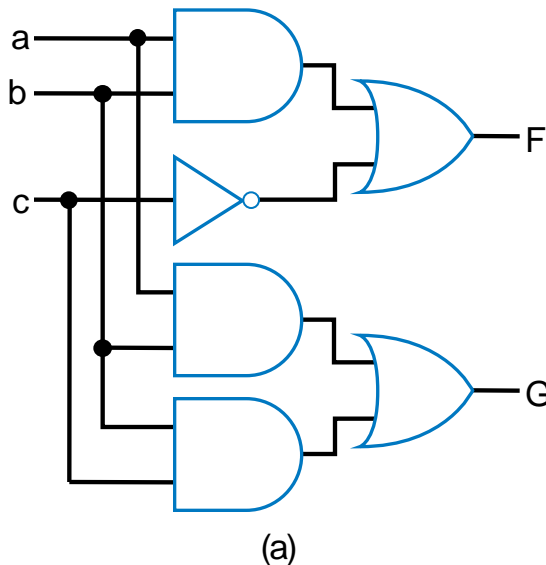
Determine SOP Equation

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

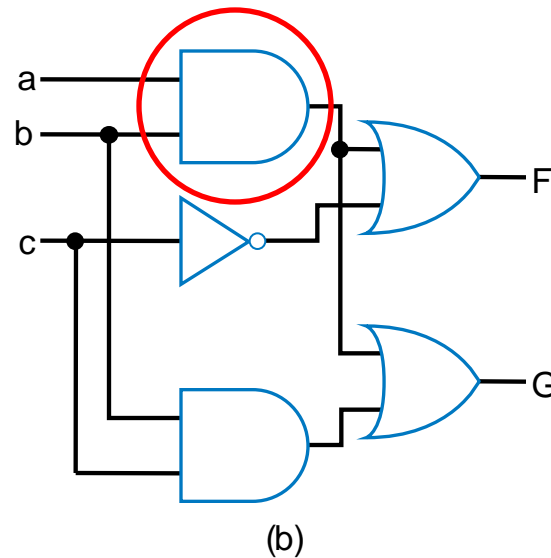
$$F = a'b'c'd + a'bc'd' + ab'cd' + ab'cd$$

Multiple-Output Circuits

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



Option 1: Separate circuits



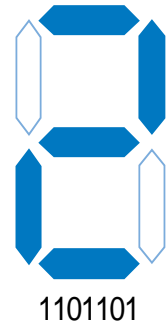
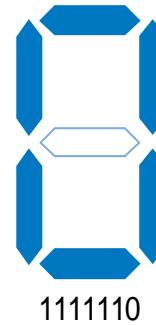
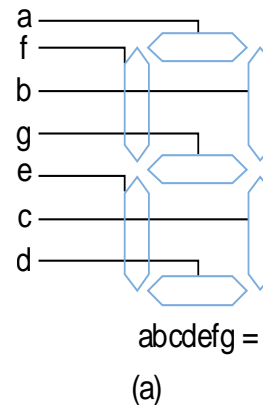
Option 2: Shared gates



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



Determine an SOP equation for the a and b segments

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

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



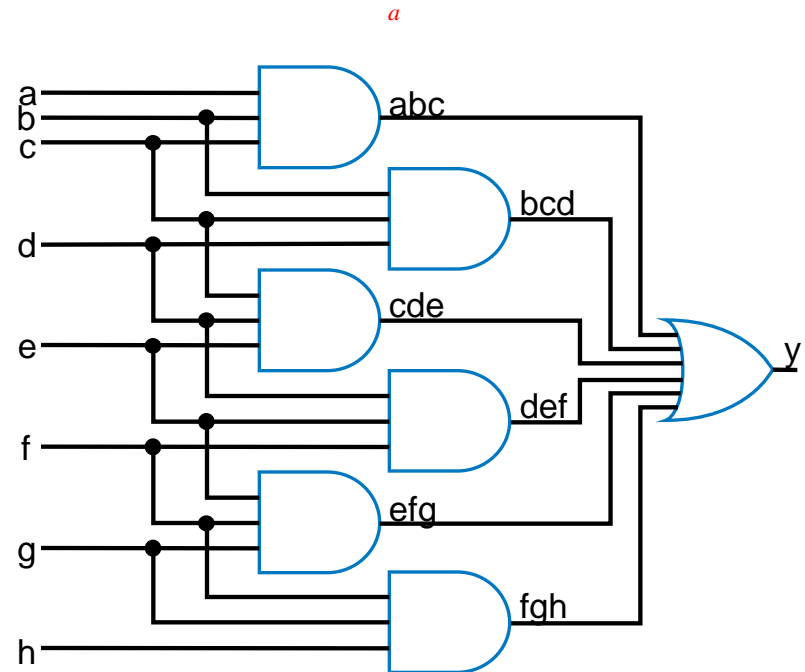
Combinational Logic Design Process

Step	Description
Step 1 Capture the function	Create a truth table or equations, <i>whichever is most natural for the given problem</i> , to describe the desired behavior of the combinational logic.
Step 2 Convert to equations	This step is only necessary if you captured the function using a truth table instead of equations. Create an SOP equation. Simplify the equations if desired.
Step 3 Implement as a gate-based circuit	For each output, create a circuit corresponding to the output's equation. (Sharing gates among multiple outputs is optional.)



Example: Three 1s Detector

- Problem: Detect three consecutive 1s in 8-bit input: abcdefgh
 - 00011101 \rightarrow 1 10101011 \rightarrow 0
 - 11110000 \rightarrow 1
- **Step 1: Capture** the function
 - Truth table or equation?
 - Truth table too big: $2^8=256$ rows
 - Equation: create terms for each possible case of three consecutive 1s
 - $y = abc + bcd + cde + def + efg + fgh$
- **Step 2: Convert** to equation -- already done
- **Step 3: Implement** as a gate-based circuit



Example: Number of 1s Count

- Problem: Output in binary on two outputs yz the number of 1s on three inputs

- 010 → 01 101 → 10 000 → 00

- **Step 1: Capture** the function

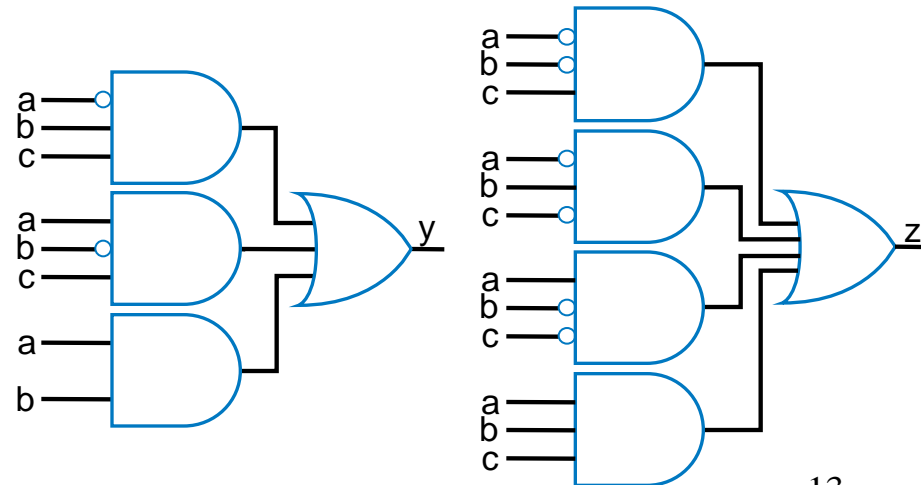
- Truth table or equation?
 - Truth table is straightforward

- **Step 2: Convert** to equation

- $y = a'bc + ab'c + abc' + abc$
 - $z = a'b'c + a'bc' + ab'c' + abc$

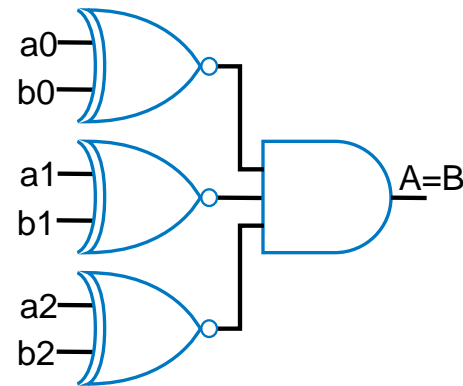
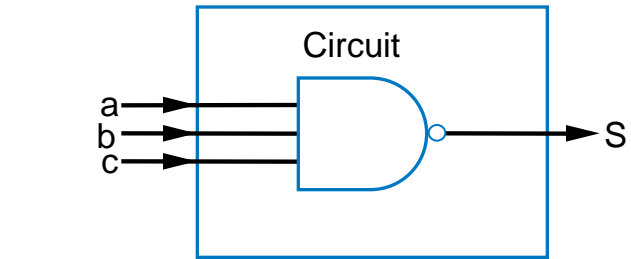
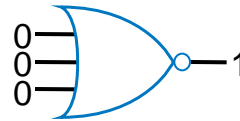
- **Step 3: Implement** as a gate-based circuit

Inputs			(# of 1s)	Outputs	
a	b	c		y	z
0	0	0	(0)	0	0
0	0	1	(1)	0	1
0	1	0	(1)	0	1
0	1	1	(2)	1	0
1	0	0	(1)	0	1
1	0	1	(2)	1	0
1	1	0	(2)	1	0
1	1	1	(3)	1	1



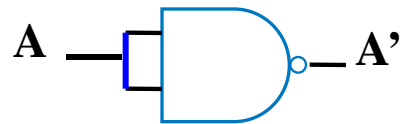
More Gates: Example Uses

- Aircraft lavatory sign example
 - $S = (abc)'$
- Detecting all 0s
 - Use NOR
- Detecting equality
 - Use XNOR
- Detecting odd # of 1s
 - Use XOR
 - Useful for generating “parity” bit common for detecting errors

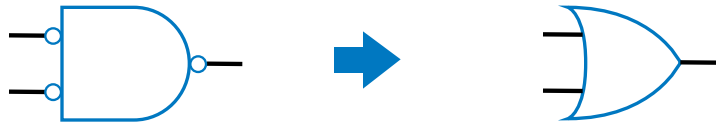


Completeness of NAND

- Any Boolean function can be implemented *using just NAND gates*. Why? DeMorgan's theorem
 - Need AND, OR, and NOT
 - NOT: NAND with inputs tied together)



- AND: NAND followed by NOT
- OR: NAND preceded by NOTs



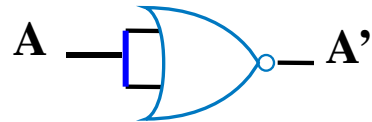
- DeMorgan's theorem: $(A' B')' = (A'' + B'')'' = A + B$



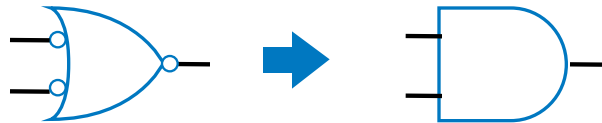
Completeness of NOR

- Any Boolean function can be implemented *using just NOR gates*.

- Need AND, OR, and NOT
- NOT: NOR with inputs tied together



- AND: NOR followed by NOT
- OR: NOR preceded by NOTs



- DeMorgan's theorem: $(A' + B')' = (A'' B'')'' = AB$

Homework

- Chapter 2: 38, 40, 53, 55, 67
- Homework is due on Thursday, January 28

Chapter Summary

- Combinational circuits
 - Circuit whose outputs are function of present inputs
 - No “state”
- Switches: Basic component in digital circuits
- Boolean logic gates: AND, OR, NOT -- Better building block than switches
 - Enables use of Boolean algebra to design circuits
- Boolean algebra: uses true/false variables/operators
- Representations of Boolean functions: Can translate among
- Combinational design process: Translate from equation (or table) to circuit through well-defined steps
- More gates: NAND, NOR, XOR, XNOR also useful

