

ECE 274 Digital Logic – Fall 2009

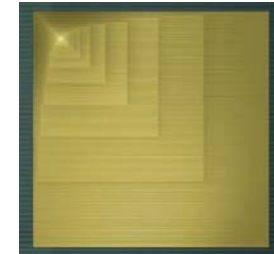
Basic Logic Gates *Digital Design 2.1 – 2.6*



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>

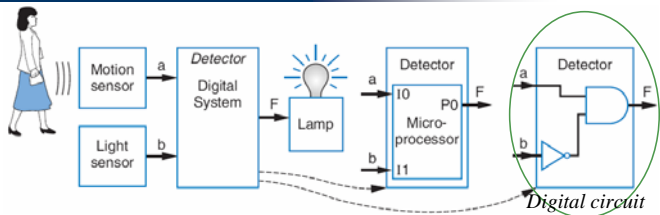


Copyright © 2007 Frank Vahid

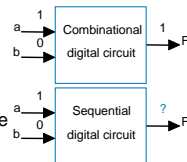
Instructors of courses requiring Vahid's Digital Design textbook (published by John Wiley and Sons) have permission to modify and use these slides for customary course-related activities, subject to keeping this copyright notice in place and unmodified. These slides may be posted as [unannotated pdf](#) 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.

Digital Logic – Combinational Logic *Introduction*

2.1



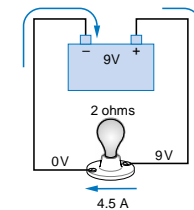
- Let's learn to design digital circuits
- We'll start with a simple form of circuit:
 - **Combinational circuit**
 - A digital circuit whose outputs depend solely on the present combination of the circuit inputs' values



Digital Logic – Combinational Logic *Switches*

2.2

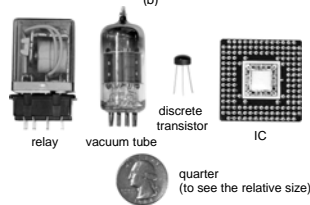
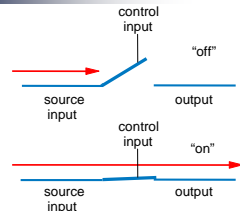
- Electronic switches are the basis of binary digital circuits
 - Electrical terminology
 - **Voltage:** Difference in electric potential between two points
 - Analogous to water pressure
 - **Current:** Flow of charged particles
 - Analogous to water flow
 - **Resistance:** Tendency of wire to resist current flow
 - Analogous to water pipe diameter
 - $V = I * R$ (Ohm's Law)



Digital Logic – Combinational Logic

Switches

- A switch has three parts
 - Source input, and output
 - Current wants 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...

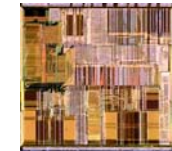
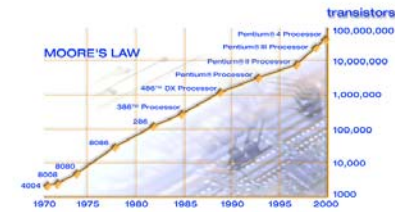


5

Digital Logic – Combinational Logic

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
 - 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

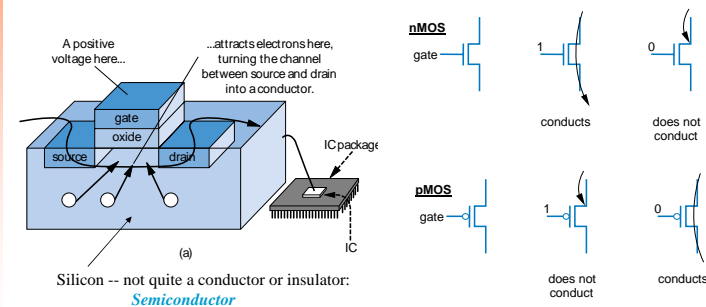
6

Digital Logic – Combinational Logic

CMOS Transistor

2.3

- CMOS transistor
 - Basic switch in modern ICs



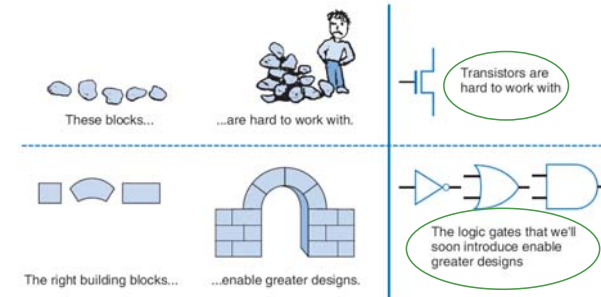
7

Digital Logic – Combinational Logic

Boolean Logic Gates - Building Blocks for Digital Circuits

2.4

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



8

Digital Logic – Combinational Logic

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 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 $a=1$)

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

9

Digital Logic – Combinational Logic

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
 - $m=1, j=0, s=1 \rightarrow F = (1 \text{ OR } 0) \text{ AND NOT}(1) = 1 \text{ AND } 0 = 0$
 - 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

10

Digital Logic – Combinational Logic

Evaluating Boolean Equations

- Evaluate the Boolean equation: $F = (a \text{ AND } b) \text{ OR } (c \text{ AND } d)$, where $a=1, b=1, c=1, d=0$. What is the value of F ?
 - 1
 - 0

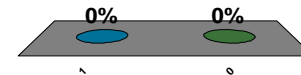


11

Digital Logic – Combinational Logic

Evaluating Boolean Equations

- Evaluate the Boolean equation: $F = (a \text{ AND } b) \text{ OR } (c \text{ AND } d)$, where $a=0, b=1, c=0, d=1$. What is the value of F ?
 - 1
 - 0



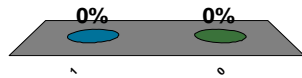
12

Digital Logic – Combinational Logic

Evaluating Boolean Equations

- Evaluate the Boolean equation: $F = (a \text{ AND } b) \text{ OR } (c \text{ AND } d)$, where $a=1, b=1, c=1, d=1$. What is the value of F?

- 1
- 0



13

Digital Logic – Combinational Logic

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

Transistor circuit

Symbol

NOT: $x \rightarrow \neg x = F$

OR: $x, y \rightarrow x \vee y = F$

AND: $x, y \rightarrow x \wedge y = F$

Truth table

x	y	F
0	1	0
1	0	1

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

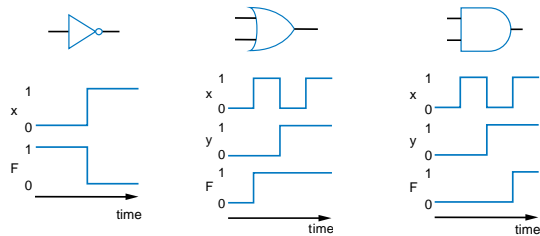
Note: These OR/AND implementations are inefficient; we'll show why, and show better ones later.

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

14

Digital Logic – Combinational Logic

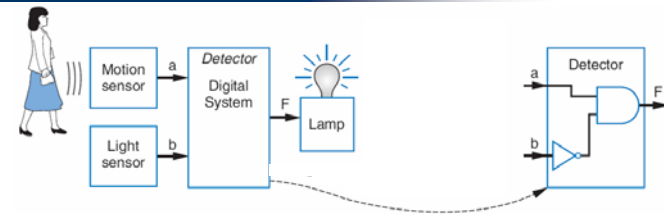
NOT/OR/AND Logic Gate Timing Diagrams



15

Digital Logic – Combinational Logic

Building Circuits Using Gates



- Motion-in-dark Detector
 - 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!**

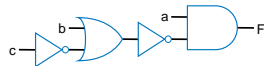
16

Which circuit corresponds to the Boolean Equation:
 $F = a \text{ AND NOT}(b \text{ OR NOT}(c))$

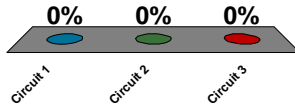
1. Circuit 1



2. Circuit 2



3. Circuit 3



17

Digital Logic – Combinational Logic

Example: Seat Belt Warning Light System

o Design circuit for warning light

o Sensors

- o s=1: seat belt fastened
- o k=1: key inserted
- o p=1: person in seat



o Capture Boolean equation

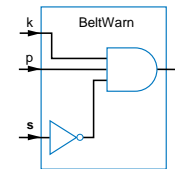
- o person in seat, and seat belt not fastened, and key inserted

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

o Convert equation to circuit

o Notice

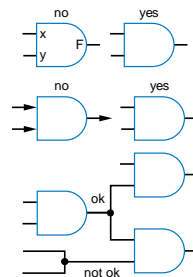
- o Boolean algebra enables easy capture as equation and conversion to circuit
 - o How design with switches?
 - o Of course, logic gates are built from switches, but we think at level of logic gates, not switches



18

Digital Logic – Combinational Logic

Some Circuit Drawing Conventions



19

Digital Logic – Combinational Logic

Boolean Algebra

2.5

o By defining logic gates based on Boolean algebra, we can use algebraic methods to manipulate circuits

- o So let's learn some Boolean algebraic methods

o Start with notation: Writing a AND b, a OR b, and NOT(a) is cumbersome

- o Use symbols: $a * b$, $a + b$, and a' (in fact, $a * b$ can be just ab).
 - o Original: $w = (p \text{ AND NOT}(s) \text{ AND } k) \text{ OR } t$
 - o New: $w = ps'k + t$
 - o Spoken as "w equals p and s prime and k, or t"
 - o Or even just "w equals p s prime k, or t"
 - o s' known as "complement of s"
 - o While symbols come from regular algebra, *don't* say "times" or "plus"

20

Digital Logic – Combinational Logic

Boolean Algebra Operator Precedence

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

- Evaluate the following Boolean equations, assuming $a=1$, $b=1$, $c=0$, $d=1$.
 - $F = ab + c$.
 - Answer: the problem is identical to the previous problem, using the shorthand notation for $*$.
 - $F = ab'$.
 - Answer: we first evaluate b' because NOT has precedence over AND, resulting in $F = 1 * (1') = 1 * (0) = 1 * 0 = 0$.
 - $F = (ac)'$.
 - Answer: we first evaluate what is inside the parentheses, then we NOT the result, yielding $(1*0)' = (0)' = 0' = 1$.

21

Digital Logic – Combinational Logic

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.

22

Digital Logic – Combinational Logic

Boolean Algebra Properties

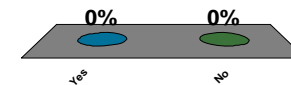
- Commutative
 - $a + b = b + a$
 - $a * b = b * a$
 - Distributive
 - $a * (b + c) = a * b + a * c$
 - $a + (b * c) = (a + b) * (a + c)$
 - (this one is tricky!)
 - Associative
 - $(a + b) + c = a + (b + c)$
 - $(a * b) * c = a * (b * c)$
 - Identity
 - $0 + a = a + 0 = a$
 - $1 * a = a * 1 = a$
 -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

23

Digital Logic – Combinational Logic

Boolean Algebra

- Can $xx' + xy(x'+y)$ ever evaluate to 1?
 1. Yes
 2. No

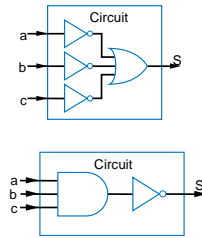


24

Digital Logic – Combinational Logic

Boolean Algebra 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
- Aircraft lavatory sign example
 - 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
 - Both are equivalent



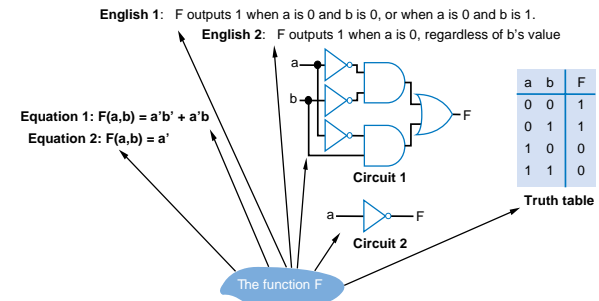
25

Digital Logic – Combinational Logic

Representations of Boolean Functions

2.6

- A function can be represented in different ways
 - Here are seven representations of the same function using four different methods: English, Equation, Circuit, and Truth Table



26

Digital Logic – Combinational Logic

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

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	0
1	1	0	1
1	1	1	1

27

Digital Logic – Combinational Logic

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		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	0
1	1	0	1
1	1	1	1

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

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

28

Digital Logic – Combinational Logic

Standard Representation: Truth Table

- How can we determine if two functions are the same?
 - Is $f = c'hp + c'hp' + c'h$ the same as $f = hc' + h'pc'$?
 - Use 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?)

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

29

Digital Logic – Combinational Logic

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
 - $ax^2 + bx + c$ ($3x^2 + 4x + 2x^2 + 3 + 1 \rightarrow 5x^2 + 4x + 4$)
 - Boolean algebra: create sum of minterms
 - Minterm**: product term with every function variable 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)

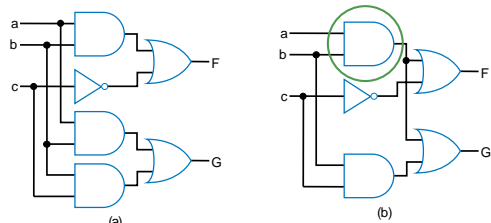
$$F = ab + a'b + a'b'$$
 (SAME -- same three terms as other equation)

30

Digital Logic – Combinational Logic

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

31

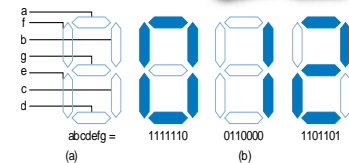
Digital Logic – Combinational Logic

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	0	1	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

COMBINATIONAL LOGIC



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

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

32

Digital Logic – Combinational Logic

In Class Exercise

- Convert the following Boolean equations to a digital circuit, sharing gates wherever possible.
 - $F(a,b,c) = abc + a'b'c + bc'$
 - $G(a,b) = ab + a'b'$