

Chapter 3

Boolean Algebra & Digital Logic

Modified by M.L.Malone, Feb 05

Chapter 3 Outcomes. At the end of this chapter, the successful student will be able to...

- Explain the relationship between Boolean logic and digital computer circuits.
- Design simple logic circuits.
- Explain how digital circuits work together to form complex computer systems.
- Explain the *simplicity* that constitutes the essence of the machine.

2

3.1 Introduction

- George Boole (late 19th c.)
 - English mathematician
 - Suggested that **logical thought** could be represented through mathematical equations.
 - Incensed philosophers and mathematicians
 - *How dare anyone suggest that human thought could be encapsulated and manipulated like an algebraic formula?*
- Computers, as we know them today, are implementations of Boole's *Laws of Thought*.

3

3.1 Introduction

- Mid 20th c.
 - Computers commonly known as “thinking machines” and “electronic brains.”
 - Many people fearful of them.
- Today...
 - Digital computers part of our lives.
 - Still feared by some.

4

3.2 Boolean Algebra

- A mathematical system for the manipulation of variables
- Can have one of two values:
 - (in formal logic) **“true” and “false.”**
 - (in digital systems)
 - **“on” and “off”**
 - **1 and 0**
 - **“high” and “low.”**

5

3.2 Boolean Expressions

- Created by performing operations on Boolean variables.
- Common Boolean operators include –
 - AND
 - OR
 - NOT

6

3.2 Boolean Algebra

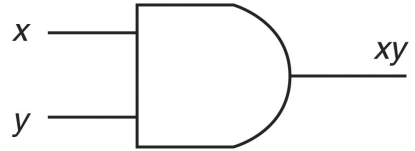
- A Boolean operator can be completely described using a truth table.
- Truth table for AND
- AND operator represents a Boolean product (using "•" sign)
 - Two notations: $x \cdot y, xy$

X AND Y		
X	Y	XY
0	0	0
0	1	0
1	0	0
1	1	1



7

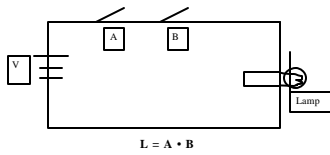
3.2 Boolean Algebra



AND Gate

8

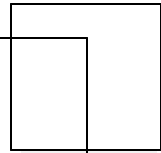
The AND operation



9

3.2 Boolean Algebra

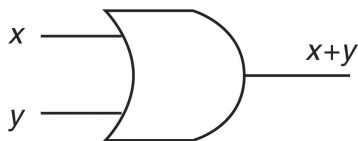
- Truth table for OR
- OR operator represents a Boolean "sum" (using "+" sign)



X OR Y		
X	Y	X+Y
0	0	0
0	1	1
1	0	1
1	1	1

10

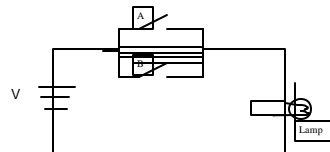
3.2 Boolean Algebra



OR Gate

11

The OR Operation



12

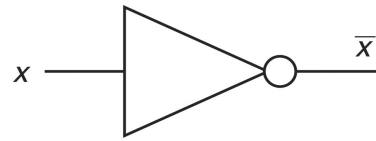
3.2 Boolean Algebra

- Truth table for the Boolean NOT operator
- Notations for the NOT operation
 - (most common) overbar, as in table to right
 - (sometimes) a prime mark (') or an "elbow" (^).

NOT x	
x	\bar{x}
0	1
1	0

13

3.2 Boolean Algebra



NOT Gate
aka an inverter

14

3.2 Boolean Algebra

- A Boolean function has--
 - At least one Boolean variable
 - At least one Boolean operator
 - At least one input from the set {0,1}.
- Its output is also a member of the set {0,1}.

15

Animations of these gates...

- <http://isweb.redwoods.cc.ca.us/INSTRUC T/CalderwoodD/diglogic/index.htm>

16

3.2 Boolean Algebra

- The truth table for the Boolean function:
 $F(x, y, z) = x\bar{z} + y$
- The shaded column shows subparts of the problem to make it easier to evaluate.

$F(x, y, z) = x\bar{z} + y$					
x	y	z	\bar{z}	$x\bar{z}$	$x\bar{z} + y$
0	0	0	1	0	0
0	0	1	0	0	0
0	1	0	1	0	1
0	1	1	0	0	1
1	0	0	1	1	1
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	0	0	1

17

3.2 Boolean Algebra

- Rules of precedence from highest priority to lowest
 - NOT
 - AND
 - OR
- Knowing rules helps in evaluating a function, as on right.

$F(x, y, z) = x\bar{z} + y$					
x	y	z	\bar{z}	$x\bar{z}$	$x\bar{z} + y$
0	0	0	1	0	0
0	0	1	0	0	0
0	1	0	1	0	1
0	1	1	0	0	1
1	0	0	1	1	1
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	0	0	1

18

3.2 Boolean Algebra

Let's create the truth table for this Boolean function:

$$F(x, y, z) = \overline{x}y + yz$$

19

3.2 Boolean Algebra

- Digital computers contain circuits that implement Boolean functions !!!
- The **simpler** that we can make a Boolean function, the **smaller** the circuit that will result..
 - cheaper to build
 - consume less power
 - run faster than complex circuits

20

3.2 Boolean Algebra

- Always “reduce” a Boolean function to its simplest form.
- Boolean “identities” help us do this.

21

3.2 Boolean Algebra

- A few Boolean identities
 - NOT the same as algebraic identities!

Identity Name	AND Form	OR Form
Identity Law	$1x = x$	$0 + x = x$
Null Law	$0x = 0$	$1 + x = 1$
Idempotent Law	$xx = x$	$x + x = x$
Inverse Law	$x\overline{x} = 0$	$x + \overline{x} = 1$

22

3.2 Boolean Algebra

- More Boolean identities
 - Associative and Commutative Laws same as in algebra!

Identity Name	AND Form	OR Form
Commutative Law	$xy = yx$	$x+y = y+x$
Associative Law	$(xy)z = x(yz)$	$(x+y)+z = x+(y+z)$
Distributive Law	$x+yz = (x+y)(x+z)$	$x(y+z) = xy+xz$

23

3.2 Boolean Algebra

- Even more Boolean identities
 - These perhaps the most useful !

Identity Name	AND Form	OR Form
Absorption Law	$x(x+y) = x$	$x + xy = x$
DeMorgan's Law	$(\overline{xy}) = \overline{x} + \overline{y}$	$\overline{(x+y)} = \overline{x}\overline{y}$
Double Complement Law	$\overline{(\overline{x})} = x$	

24

3.2 Boolean Algebra

- Using Boolean identities to simplify a function:

$$F(X, Y, Z) = (X + Y)(X + \bar{Y})(\bar{X}\bar{Z})$$

$(X + Y)(X + \bar{Y})(\bar{X}\bar{Z})$	Idempotent Law (Rewriting)
$(X + Y)(X + \bar{Y})(\bar{X} + Z)$	DeMorgan's Law
$(XX + X\bar{Y} + XY + Y\bar{Y})(\bar{X} + Z)$	Distributive Law
$((X + Y\bar{Y}) + X(Y + \bar{Y}))(\bar{X} + Z)$	Commutative & Distributive Laws
$((X + 0) + X(1))(\bar{X} + Z)$	Inverse Law
$X(\bar{X} + Z)$	Idempotent Law
$X\bar{X} + XZ$	Distributive Law
$0 + XZ$	Inverse Law
XZ	Idempotent Law

25

3.2 Boolean Algebra

- DeMorgan's law provides an easy way of finding the complement of a Boolean function.
- Recall DeMorgan's law states:

$$\overline{(xy)} = \bar{x} + \bar{y} \quad \text{and} \quad \overline{(x+y)} = \bar{x}\bar{y}$$

- Replace each variable by its complement and change all ANDs to ORs and all ORs to ANDs.

26

3.2 Boolean Algebra

- More on DeMorgan's law
 - Can be extended to any number of variables.
 - Example: The complement of

$$F(X, Y, Z) = (XY) + (\bar{X}Z) + (Y\bar{Z})$$

is:
$$\begin{aligned} \bar{F}(X, Y, Z) &= \overline{(XY) + (\bar{X}Z) + (Y\bar{Z})} \\ &= (\overline{XY})(\overline{\bar{X}Z})(\overline{Y\bar{Z}}) \\ &= (\bar{X} + \bar{Y})(X + Z)(\bar{Y} + Z) \end{aligned}$$

27

3.2 Boolean Algebra

- "Synonymous" forms
 - Logically equivalent
 - Have identical truth tables.
- To eliminate confusion, designers express Boolean functions in **standardized** or **canonical** form.

28

3.2 Boolean Algebra

- Two canonical forms for Boolean expressions
 - sum-of-products (SOP)
 - product-of-sums (POS)
- In sum-of-products form, ANDed variables are ORed together.
 - For example: $F(x, y, z) = xy + xz + yz$
- In product-of-sums form, ORed variables are ANDed together:
 - For example: $F(x, y, z) = (x+y)(x+z)(y+z)$

29

3.2 Boolean Algebra

To convert a function to sum of-products form using its truth table:

$$F(x, y, z) = x\bar{z} + y$$

x	y	z	$x\bar{z} + y$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

- Look at values of variables that make the function true (=1).
- List the values of the variables that result in a true function value.
- Each group of variables is then ORed together.

30

3.2 Boolean Algebra

- The sum-of-products form for our function is:

$$F(x, y, z) = \bar{x}y\bar{z} + \bar{x}yz + x\bar{y}\bar{z} + x\bar{y}z + xy\bar{z} + xyz$$

Note: This function is not in simplest terms. That was NOT the goal here! Our aim was to rewrite it in canonical sum-of-products form.

$F(x, y, z) = x\bar{z} + y$

x	y	z	$x\bar{z} + y$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

31

3.2 Boolean Algebra

Let's try another!

Write the canonical sum of products form for this truth table →

x	y	z	F(x,y,z)
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

32

3.2 Boolean Algebra

$F(x,y,z) = ??$

x	y	z	F(x,y,z)
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

33

3.2 Boolean Algebra

- Sum of products
 - Easier to work with
 - Any truth table can be represented in this form!
 - We'll use this... in simplifying circuits using K-maps.
- Product of sums
 - Generally preferred when Boolean expression evaluates to true more times than it evaluates to false

34

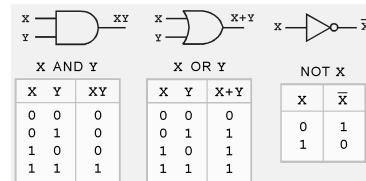
3.3 Logic Gates

- Boolean functions
 - Can be implemented in digital computer circuits called "gates".
- "Gate"
 - An electronic device that produces a result based on one or more input values.
 - Consists of a set of transistors
 - Integrated circuits (ICs) contain collections of gates suited to a particular purpose.

35

3.3 Logic Gates

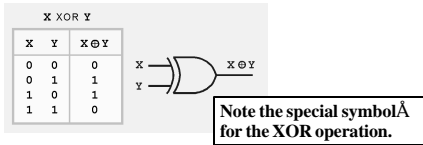
- The three simplest gates :



36

3.3 Logic Gates

- Exclusive OR (XOR) gate
 - Also quite useful
- Output true only when the values of the inputs differ.

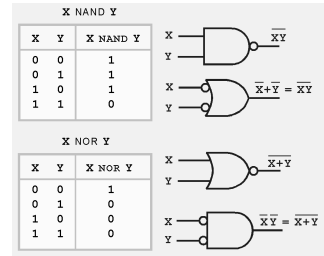


37

3.3 Logic Gates

- NAND and NOR gates
 - Very important!

Why?



38

3.3 Logic Gates

NAND and NOR gates

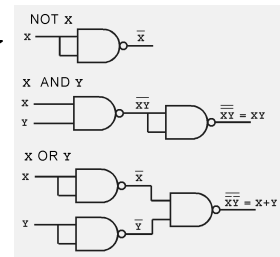
- Inexpensive to manufacture
- Known as *universal gates* because—
 - **Any Boolean function** can be constructed using only NAND or only NOR gates !
 - Complex ICs often easier to build using same building block rather than a collection of different ones

39

3.3 Logic Gates

Using the NAND gate to implement..

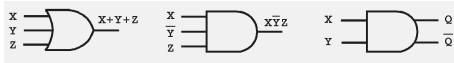
- a NOT gate
- an AND gate
- an OR gate



40

3.3 Logic Gates

- Can have multiple inputs & more than one output.
 - A second output can be provided for the complement of the operation.
 - More on this later...

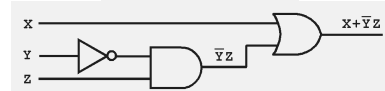


41

3.4 Digital Components

- Combinations of gates implement Boolean functions as circuits !
- Ex:
 - Circuit below implements the Boolean function:

$$F(X, Y, Z) = X + \bar{Y}Z$$



Why simplify Boolean expressions?

42

3.4 Digital Components

- Boolean algebra
 - Allows us to analyze, design, & simplify digital circuits
 - Cheaper than building a circuit with gates and then trying to simplify!
- K- Maps (Karnaugh Maps)
 - Used to simplify Boolean expressions
 - See section at end of Chap. 3 (*read/study this !*)

43

3.5 Combinational Circuits

- The circuit (from a few slides ago...) implemented the Boolean function:

$$F(X, Y, Z) = X + \bar{Y}Z$$
 - It's an example of a *combinational logic* circuit.
 - Produces a specified output (almost) at the instant when input values are applied.
- Later, we'll explore sequential circuits where this is not the case.*

44

3.5 Combinational Circuits

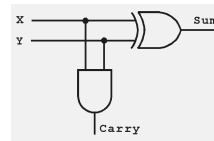
- The Half Adder
 - A simple example of a useful combinational circuit:
 - Finds the sum of two bits.
- Truth table for half-adder.

Inputs		Outputs	
X	Y	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

45

3.5 Combinational Circuits

- Use XOR to find sum
- Use AND to find carry



Inputs		Outputs	
X	Y	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

See animation: <http://www.redwoods.cc.ca.us/INSTRUCT/CalderwoodD/digitalogic/index.htm>

46

3.5 Combinational Circuits

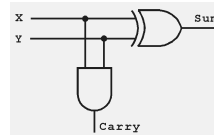
- A Full Adder?
 - Modify the half adder by including gates for processing the carry bit.
- Truth table for a full adder

Inputs			Outputs	
X	Y	Carry In	Sum	Carry Out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

47

3.5 Combinational Circuits

- How can we change the half adder shown below to make it a full adder?

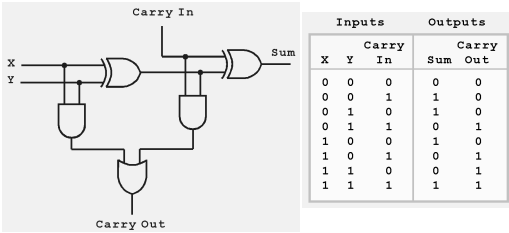


Inputs			Outputs	
X	Y	Carry In	Sum	Carry Out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

48

3.5 Combinational Circuits

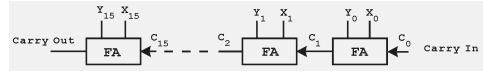
The completed full adder.



Animation:
<http://isweb.redwoods.cc.ca.us/INSTRUCT/CalderwoodD/digilogic/index.htm>

3.5 Combinational Circuits

- We just combined half adders to make a full adder
- We can combine full adders, connected in series, to make a *ripple-carry adder*.
 - The carry bit "ripples" from one adder to the next.
- What will this ripple-carry adder allow us to do?



Today's systems employ more efficient adders.

3.5 Combinational Circuits

- The ripple-carry adder (from previous slide)
 - Not normally implemented **but** easy to understand
 - Good because gives idea of how to add larger numbers
 - Improvements (newer adders) improve speeds by 40-90% by performing some adds in parallel

Before continuing...

let's do some circuit reducing using Karnaugh Maps (aka K-Maps)... We'll then return to the rest of Chapter 3.

READ / STUDY section on K-Maps at the end of Chapter 3.

3.5 Combinational Circuits

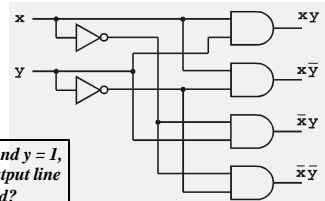
- Decoders
 - Important type of combinational circuit.
 - Useful in selecting a memory location according a binary value placed on the address lines of a memory bus.
- Address decoders with n inputs can select any of 2^n locations.
 - That is, **one unique line is set to 1**

This is a block diagram for a decoder.



3.5 Combinational Circuits

- This is what a 2-to-4 decoder looks like on the inside.

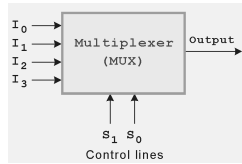


If $x = 0$ and $y = 1$, which output line is enabled?

3.5 Combinational Circuits

Multiplexer

- Opposite of a decoder.
- Selects a single output from several inputs.
- Input chosen for output determined by the value of the multiplexer's control lines.
 - To select among n inputs, $\log_2 n$ control lines needed.

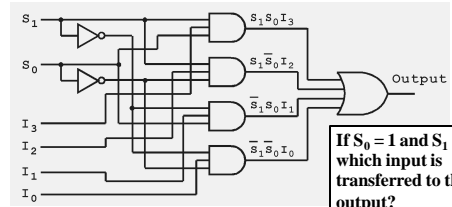


This is a block diagram for a multiplexer.

55

3.5 Combinational Circuits

- This is what a 4-to-1 multiplexer looks like on the inside.



If $S_0 = 1$ and $S_1 = 0$, which input is transferred to the output?

56

3.6 Sequential Circuits

- Combinational logic circuits
 - Perfect when we require the immediate application of a Boolean function to a set of inputs.
 - **BUT... they have no "memory"**
- Sequential logic circuits provide this functionality for us.

57

3.6 Sequential Circuits

- Defines its output as a function of both its current inputs and its previous inputs
- Output depends on past inputs
- Must have a storage element
 - Called a "flip flop" (more on this later...)

58

3.6 Sequential Circuits

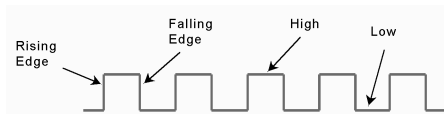
- Require a means by which events can be sequenced.
- State changes controlled by clocks.
- "clock"
 - Special circuit that sends electrical pulses through a circuit.
 - Produces electrical waveforms (see below)



59

3.6 Sequential Circuits

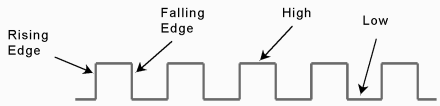
- State changes occur in sequential circuits only when the clock ticks.
- Circuits can change state on the rising edge, falling edge, or when the clock pulse reaches its highest voltage.



60

3.6 Sequential Circuits

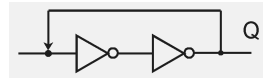
- *Edge-triggered* circuits
 - Change state on the rising edge or falling edge of the clock pulse
- *Level-triggered* circuits
 - Change state when the clock voltage reaches its highest or lowest level.



61

3.6 Sequential Circuits

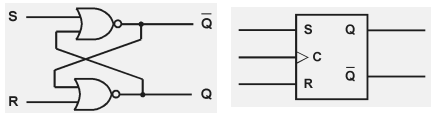
- Rely on **feedback** to retain their state values
- Feedback occurs when an output is looped back to the input.
- Example:
 - If Q is 0, will it always be 0? Why? Why not?
 - If Q is 1, will it always be 1. Why? Why not?



62

3.6 Sequential Circuits

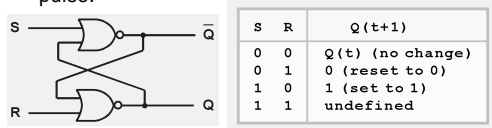
- You can see how feedback works by examining the most basic sequential logic components, the SR flip-flop.
 - The “SR” stands for set/reset.
- The internals of an SR flip-flop are shown below, along with its block diagram.



63

3.6 Sequential Circuits

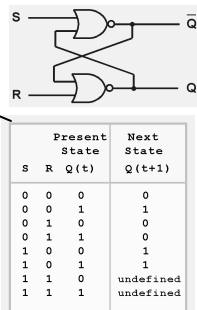
- SR flip-flop behavior described by a “characteristic table”.
- Q(t) means the value of the output at time t.
- Q(t+1) is the value of Q after the next clock pulse.



64

3.6 Sequential Circuits

- SR flip-flop
 - has three inputs:
 - S
 - R
 - **and its current output, Q.**
 - Truth table at right.
 - Notice two undefined values.
 - When both S and R are 1, the SR flip-flop is unstable.



65

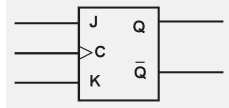
SR Flip-flop animation

<http://isweb.redwoods.cc.ca.us/INSTRUCT/CalderwoodD/diglogic/srflip.htm>

66

3.6 Sequential Circuits

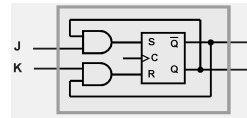
- If we can be sure that the inputs to an SR flip-flop will never both be 1, we will never have an unstable circuit. This may not always be the case.
- The SR flip-flop can be modified to provide a stable state when both inputs are 1.
- This modified flip-flop is called a JK flip-flop, shown at the right.
 - The "JK" is in honor of Jack Kilby.



67

3.6 Sequential Circuits

- At the right, we see how an SR flip-flop can be modified to create a JK flip-flop.
- The characteristic table indicates that the flip-flop is stable for all inputs.

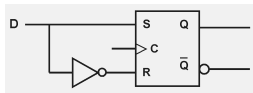


J	K	Q (t+1)
0	0	Q (t) (no change)
0	1	0 (reset to 0)
1	0	1 (set to 1)
1	1	$\bar{Q}(t)$

68

3.6 Sequential Circuits

- The D flip-flop
 - Another modification of the SR flip-flop
 - The fundamental circuit for computer memory**
 - See the "characteristic table" below.
 - Output of flipflop remains the same during subsequent clock pulses. The output changes only when the value of D changes.

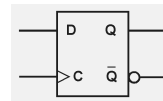


D	Q (t+1)
0	0
1	1

69

3.6 Sequential Circuits

- More on the D flip-flop**
 - Usually illustrated using the block diagram shown below.
 - Combined to create a register (next slide...)

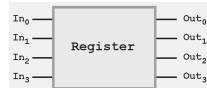


D	Q (t+1)
0	0
1	1

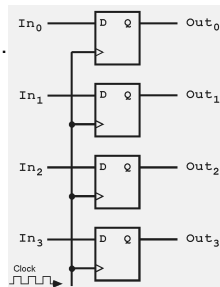
70

3.6 Sequential Circuits

- A 4-bit register consisting of D flip-flops.
- The block diagram (below).



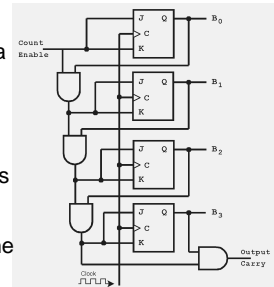
A larger memory configuration is in your text.



71

3.6 Sequential Circuits

- A binary counter
 - another example of a sequential circuit.
 - The low-order bit is complemented at each clock pulse.
 - Whenever it changes from 0 to 1, the next bit is complemented, and so on through the other flip-flops.



72

3.7 Designing Circuits

- We have seen digital circuits from two points of view
 - **Digital analysis**
 - Explores relationship between a circuit's inputs and its outputs.
 - **Digital synthesis**
 - Creates logic diagrams using the values specified in a truth table.

73

3.7 Designing Circuits

- Digital systems designers must —
 - Be mindful of the physical behaviors of circuits
 - Include minute propagation delays that occur between the time when a circuit's inputs are energized and when the output is accurate and stable.
 - Rely on specialized software to create efficient circuits.

74

3.7 Designing Circuits

- A hardware solution often preferred when...
 - We need to implement a simple, specialized algorithm
 - Its execution speed must be as fast as possible
- Embedded Systems
 - Small special-purpose computers that we find in many everyday things.
 - Require special programming that demands an understanding of the operation of digital circuits

75

Chapter 3 Conclusion

- Computers are implementations of Boolean logic.
- Boolean functions are completely described by truth tables.
- Logic gates are small circuits that implement Boolean operators.
- The basic gates are AND, OR, and NOT.
- The "universal gates" are NOR, and NAND.

76

Chapter 3 Conclusion

- Computer circuits consist of combinational logic circuits and sequential logic circuits.
- Combinational circuits produce outputs (almost) immediately when their inputs change.
- Sequential circuits require clocks to control their changes of state.
- The basic sequential circuit unit is the flip-flop:
 - The behaviors of SR, JK, and D flip-flops are the most important to know.

77

What's Ahead?

- Complete Chapter 3 homework.
- Read/study Chapter 4.

78