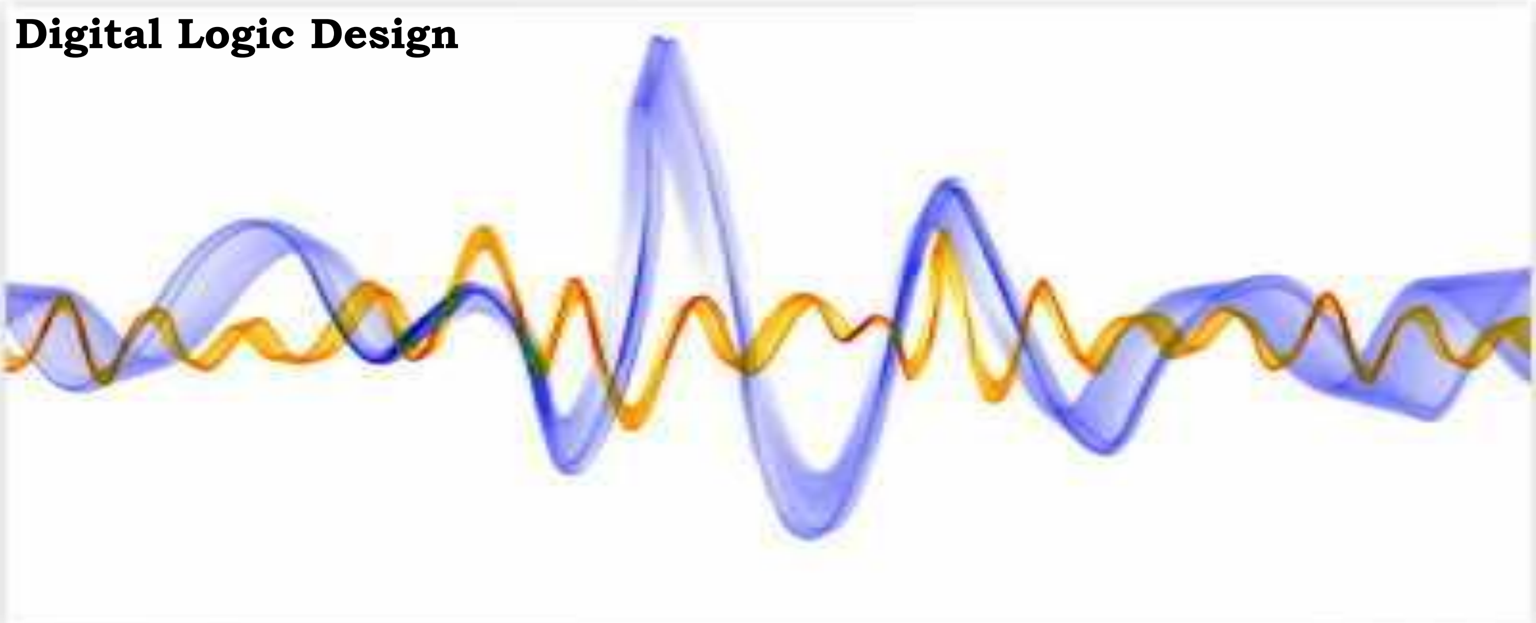
Digital Logic Design



Lecture 2:

Chapter 1&2: Arithmetic Operations, Boolean Algebra and Logic Gates

Mirvat Al-Qutt, Ph.D
Computer Systems Department , FCIS,
Ain Shams University

Arithmetic Operations (**Addition**)

- Arithmetic operations with numbers in base ***r*** **follow the same rules as for decimal numbers.** When a base other than the familiar base **10** is used, one must be careful to use **only the *r*-allowable digits**.

- Example add **3758 and 4657**

$$\begin{array}{r} 3758 \\ +\ \underline{4657} \end{array}$$

Arithmetic Operations (**Addition**)

- Example add **3758 and 4657**

$$
\begin{array}{r}
\color{red}{1\,1\,1} \\
3\,7\,5\,8 \\
+\;\underline{4\,6\,5\,7} \\
8\,4\,1\,5
\end{array}
$$

What just happened?

	1	1	1		(carry)
	3	7	5	8	
+	4	6	5	7	
	8	14	11	15	(sum)
-		10	10	10	(subtract the base)
	8	4	1	5	

- when the sum of a column is equal to or greater than the base, we subtract the base from the sum, record the difference, and carry one to the next column to the left.

Arithmetic Operations (**Addition**)

- In Binary ….. Just like in decimal
- **Rules:**
  - **0+0 = 0**
  - **0+1 = 1**
  - **1+0 = 1**
  - **1+1 = 2_{10} (2-2 =0 , result in binary 0 with carry 1)**
  - **1+1+1 = 3_{10} (3-2=1 , result in binary 1 with carry 1)**

- when the sum of a column is equal to or greater than the base, we subtract the base from the sum, record the difference, and carry one to the next column to the left.

Arithmetic Operations (**Addition**)

- In Binary ….. Just like in decimal
- Add 110111 + 011100

$$
\begin{array}{ccccccccc}
 & 1 & 1 & 1 & 1 & & & \\
 & & 1 & 1 & 0 & 1 & 1 & 1 \\
+ & & 0 & 1 & 1 & 1 & 0 & 0 \\
\hline
 & 1 & 0 & 1 & 0 & 0 & 1 & 1
\end{array}
$$

$$
\begin{array}{cccccccc}
 & 1 & 1 & 1 & 1 & & & \\
 & & 1 & 1 & 0 & 1 & 1 & 1 \\
+ & & 0 & 1 & 1 & 1 & 0 & 0 \\
\hline
 & & 2 & 3 & 2 & 2 & & \\
- & & 2 & 2 & 2 & 2 & & . \\
\hline
 & 1 & 0 & 1 & 0 & 0 & 1 & 1
\end{array}
$$

Arithmetic Operations (**Addition**)

- Try it your self
- Example 2:

$$
\begin{array}{r}
1\ 1\ 1\ 1\ 0\ 1_2 \\
+\ 0\ 0\ 1\ 1\ 0\ 1_2 \\
\hline
\end{array}
$$

- Example 3:

$$
\begin{array}{r}
1\ 0\ 0\ 1\ 0\ 1\ 1\ 1_2 \\
+\ 1\ 1\ 0\ 1\ 0\ 1\ 0\ 1_2 \\
\hline
\end{array}
$$

Arithmetic Operations (**Addition**)

- Try it your self
- Example 2:

$$
\begin{array}{ccccccc}
\color{red}{1} & \color{red}{1} & \color{red}{1} & \color{red}{1} & & \color{red}{1} & \\
 & 1 & 1 & 1 & 1 & 0 & 1_2 \\
+ & 0 & 0 & 1 & 1 & 0 & 1_2 \\
\hline
1 & 0 & 0 & 1 & 0 & 1 & 0_2
\end{array}
\rightarrow
\begin{array}{r}
61_{10} \\
+13_{10} \\
\hline
74_{10}
\end{array}
$$

- Example 3:

$$
\begin{array}{ccccccccc}
\color{red}{1} & & & \color{red}{1} & & \color{red}{1} & \color{red}{1} & \color{red}{1} & \\
 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1_2 \\
+ & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1_2 \\
\hline
1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0
\end{array}
\rightarrow
\begin{array}{r}
151_{10} \\
+213_{10} \\
\hline
364_{10}
\end{array}
$$

Arithmetic Operations (**Subtraction**)

- Example subtract **8025 and 4657**

$$
\begin{array}{rcccc}
 & 8 & 0 & 2 & 5 \\
- & 4 & 6 & 5 & 7 \\
\hline
\end{array}
$$

Arithmetic Operations (**Subtraction**)

- Example subtract **8025 and 4657**

$$\begin{array}{rcccc} & & & 1 & \\ & 8 & 0 & \cancel{2} & 15 \\ - & 4 & 6 & 5 & 7 \\ \hline & & & & 8 \end{array}$$

Arithmetic Operations (**Subtraction**)

- Example subtract **8025 and 4657**

$$
\begin{array}{ccccc}
 & 7 & 9 & 11 & \\
 & \cancel{8} & \cancel{10} & \cancel{2} & 15 \\
- & 4 & 6 & 5 & 7 \\
\hline
 & & & 6 & 8
\end{array}
$$

Arithmetic Operations (**Subtraction**)

- Example subtract **8025 and 4657**

$$\begin{array}{rrrrr} & 7 & 9 & 11 & \\ & \cancel{8} & 1\cancel{0} & \cancel{2} & 15 \\ - & 4 & 6 & 5 & 7 \\ \hline & 3 & 3 & 6 & 8 \end{array}$$

Arithmetic Operations (**Subtraction**)

- In Binary ….. Just like in decimal
- In binary, the base unit is 2,
- So when you cannot subtract, you borrow from the column to the left.
- The amount borrowed is **2**.
- The 2 is added to the original column value, so you will be able to subtract.

Arithmetic Operations (**Subtraction**)

- In Binary ….. Just like in decimal
- Example Subtract 110011 - 11100

$$
\begin{array}{r}
1\ 1\ 0\ 0\ 1\ 1 \\
-\quad\ 1\ 1\ 1\ 0\ 0 \\
\hline
\end{array}
$$

Arithmetic Operations (**Subtraction**)

- In Binary ….. Just like in decimal
- Example Subtract 110011 - 11100

$$\begin{array}{rrrrrrr} & 1 & 1 & 0 & 0 & 1 & 1 \\ - & & 1 & 1 & 1 & 0 & 0 \\ \hline & & & & & 1 & 1 \end{array}$$

Arithmetic Operations (**Subtraction**)

- In Binary ….. Just like in decimal
- Example Subtract 110011 - 11100

$$
\begin{array}{ccccccc}
 & & & 1 & & & \\
 & & 0 & \cancel{2} & 2 & & \\
 & 1 & \cancel{1} & \cancel{0} & \cancel{0} & 1 & 1 \\
- & & 1 & 1 & 1 & 0 & 0 \\
\hline
 & & & & 1 & 1 & 1
\end{array}
$$

Arithmetic Operations (**Subtraction**)

- In Binary ….. Just like in decimal
- Example Subtract 110011 - 11100

$$
\begin{array}{rcccccc}
 & & 2 & 1 & & & \\
 & 0 & \cancel{0} & \cancel{2} & 2 & & \\
 & \cancel{1} & \cancel{1} & \cancel{0} & \cancel{0} & 1 & 1 \\
- & & 1 & 1 & 1 & 0 & 0 \\
\hline
 & & 1 & 0 & 1 & 1 & 1
\end{array}
$$

Arithmetic Operations (**Subtraction**)

- Try it your self
- Example 2:

$$\begin{array}{r} 110101_2 \\ -\ 101011_2 \\ \hline \end{array}$$

- Example 3:

$$\begin{array}{r} 1001101_2 \\ -\ 110111_2 \\ \hline \end{array}$$

Arithmetic Operations (**Subtraction**)

- Try it your self
- Example 2:

$$
\begin{array}{cccccccc}
 & & 0 & 2 & 0 & 2 & & \\
 & 1 & \cancel{1} & 0 & \cancel{1} & 0 & 1_2 \\
- & 1 & 0 & 1 & 0 & 1 & 1_2 \\
\hline
 & 0 & 0 & 1 & 0 & 1 & 0_2
\end{array}
\rightarrow
\begin{array}{cc}
 & 53_{10} \\
- & 43_{10} \\
\hline
 & 10_{10}
\end{array}
$$

- Example 3:

$$
\begin{array}{cccccccc}
 & & 1 & & & 2 & & \\
 & 0 & \cancel{2} & 2 & 0 & \cancel{0} & 2 & \\
 & \cancel{1} & 0 & 0 & \cancel{1} & \cancel{1} & 0 & 1_2 \\
- & & 1 & 1 & 0 & 1 & 1 & 1_2 \\
\hline
 & & 0 & 1 & 0 & 1 & 1 & 0_2
\end{array}
\rightarrow
\begin{array}{cc}
 & 77_{10} \\
- & 55_{10} \\
\hline
 & 22_{10}
\end{array}
$$

Arithmetic Operations (**Hexadecimal)**

Addition

$$
\begin{array}{rcccc}
 & 1 & 1 & & \\
 & 7 & C & 3 & 9_{16} \\
+ & 3 & 7 & F & 2_{16} \\
\hline
 & & 20 & 18 & 11 \\
- & & 16 & 16 & \\
\hline
 & B & 4 & 2 & B_{16}
\end{array}
$$

(subtract Base (16))

Arithmetic Operations (**Hexadecimal)**

Subtraction

$$
\begin{array}{ccccc}
 & & B & 16 & \\
 & 7 & \cancel{C} & \underline{3} & 9_{16} \\
 & & & 19 & \\
- & 3 & 7 & F & 2_{16} \\ \hline
 & 4 & 4 & 4 & 7_{16}
\end{array}
$$

Arithmetic Operations (**Octal**)

Addition

$$
\begin{array}{rcccccl}
 & & 1 & 1 & & & \\
 & & 6 & 4 & 3 & 7_8 & \\
+ & & 2 & 5 & 1 & 0_8 & \\
\hline
 & & 9 & 9 & & & \\
- & & 8 & 8 & & & \text{(subtract Base (8))} \\
\hline
 & 1 & 1 & 1 & 4 & 7_8 &
\end{array}
$$

Arithmetic Operations (**Octal**)

Subtraction

$$
\begin{array}{cccccc}
 & & 8 & & & \\
 & 0 & 0 & 8 & & \\
 & \cancel{1} & \underline{\cancel{1}} & \underline{1} & 4 & 7_8 \\
 & & 8 & 9 & & \\
- & & 6 & 4 & 3 & 7_8 \\
\hline
 & & 2 & 5 & 1 & 0_8
\end{array}
$$

Arithmetic Operations (**Multiplication**)

- Bit by bit

				1	0	1	1	1
x					1	0	1	0
				0	0	0	0	0
			1	0	1	1	1	
		0	0	0	0	0		
	1	0	1	1	1			
	1	1	1	0	0	1	1	0

Complements

- There are **two types of complements for each base-*r* system**

Diminished Radix Complement
(r-1)'s Complement

Given a number N in base r having n digits,
the $(r-1)$'s complement *of* N is defined as:
$\boldsymbol{(r^n - 1) - N}$

Radix Complement
r's complement

Given n-digit number N in base r
the r's complement *of* N is defined as
$\mathbf{r^n - N}$ for $N \neq 0$ and as **0** for $N = 0$.

Comparing with the (r – 1) 's complement, we note that the r's complement is obtained by adding 1 to the (r – 1) 's complement, since $r^n - N = [(r^n - 1) - N] + 1$.

Complements

- **Diminished Radix Complement - (r-1)'s Complement**
  - Given a number N in base r having n digits, the $(r-1)$'s complement *of* N is defined as:

$$(r^n - 1) - N$$

- **Example for 6-digit decimal numbers**:
  - 9's complement is $(r^n - 1) - N = (10^6 - 1) - N = 999999 - N$
  - 9's complement of 546700

$$\begin{array}{r} 999999 \\ -\quad 546700 \\ \hline 453299 \end{array}$$

Complements

- **Diminished Radix Complement , (r-1)'s Complement**

- **Example for 7-digit <u>binary</u> numbers:**
  - **1's** complement is $(r^n - 1) - N = (2^7 - 1) - N = 1111111 - N$
  - **1's** complement of 1011000 is

$$\begin{array}{rr} & 1111111 \\ - & 1011000 \\ \hline & 0100111 \end{array}$$

Observation:

- Subtraction from $(r^n - 1)$ will never require a borrow
- Diminished radix complement can be computed digit-by-digit
- For binary: $1 - 0 = 1$ and $1 - 1 = 0$

Complements

- **1's Complement (*Diminished Radix* Complement)**
  - All '0's become '1's
  - All '1's become '0's

Example $(\mathbf{10110000})_2$

$\Rightarrow (\mathbf{01001111})_2$

If you add a number and its 1's complement …

$$
\begin{array}{r}
1\,0\,1\,1\,0\,0\,0\,0 \\
+\ 0\,1\,0\,0\,1\,1\,1\,1 \\
\hline
1\,1\,1\,1\,1\,1\,1\,1
\end{array}
$$

Complements

- There are **two types of complements for each base-*r* system**

Diminished Radix Complement
(r-1)'s Complement

Given a number *N* in base *r* having *n* digits,
the (*r*–*1*)'s complement *of N* is defined as:
$(r^n - 1) - N$

Radix Complement
r's complement

Given n-digit number N in base r
the *r*'s complement *of N* is defined as
$\mathbf{r^n - N}$ for $N \neq 0$ and
as **0** for $N = 0$.

Complements

- Radix Complement
- Example: Base-10
  - The 10's complement of 012398 is 987602
  - The 10's complement of 246700 is 753300

$$\begin{array}{rr} & 1000000 \\ - & 012398 \\ \hline & 987602 \end{array} \qquad \begin{array}{rr} & 1000000 \\ - & 246700 \\ \hline & 753300 \end{array}$$

Comparing with the (r – 1) 's complement, we note that the r's complement is obtained by adding 1 to the (r – 1) 's complement, since $r^n - N = [(r^n - 1) - N] + 1$.

Complements

- Radix Complement
- Example: Base-2
  - The 2's complement of 1101100 is 0010100
  - The 2's complement of 0110111 is 1001001

$$\begin{array}{rr} & 10000000 \\ - & 1101100 \\ \hline & 0010100 \end{array} \qquad \begin{array}{rr} & 10000000 \\ - & 0110111 \\ \hline & 1001001 \end{array}$$

Comparing with the (r – 1) 's complement, we note that the r's complement is obtained by adding 1 to the (r – 1) 's complement, since $r^n - N = [(r^n - 1) - N] + 1$.

Complements

- 2's Complement (*Radix* Complement)
  - Take 1's complement then add 1

OR
  - Toggle all bits to the left of the first '1' from the right

***Example*:**

Number:	1 0 1 1 0 0 0 0	1 0 1 1 0 0 0 0
1's Comp.:	0 1 0 0 1 1 1 1	
+	1	
	0 1 0 1 0 0 0 0	0 1 0 1 0 0 0 0

Complements

- **Subtraction with Complements**
  - The subtraction of two *n*-digit unsigned numbers $M - N$ in base *r* can be done as follows:

1. Add the minuend M to the r's complement of the subtrahend N. Mathematically, $M + (r^n - N) = M - N + r^n$.
2. If $M \geqq N$, the sum will produce and end carry r^n, which can be discarded; what is left is the result $M - N$.
3. If $M < N$, the sum does not produce an end carry and is equal to $r^n - (N - M)$, which is the r's complement of $(N - M)$. To obtain the answer in a familiar form, take the r's complement of the sum and place a negative sign in front.

Complements

- Example 1.7
  - Given the two binary numbers perform the subtraction
    - ***X* = 1010, *Y* = 0110,**
  - (a) X – ; (b) Y – X , using complement.

X-Y		Y-X	
1's Comp	2's Comp	1's Comp	2's Comp
1010	1010	0110	0110
−0110	−0110	−1010	−1010
1010	1010	0110	0110
+1001	+1010	+0101	+0110
10011	10100	1011	1100
+1			
0100	0100	−0100	−0100

Complements

1's Complement	2's Complement
Subtract N from (2^n-1)	Subtract N from (2^n)
Inverting 0's to be 1's and 1's to be 0's Bitwise toggling	Toggle all bits to the left of the first '1' from the right
Subtraction M-N is done By: - Get 1's Complement of N - Add M + N - If carry then Add carry to summation - If no carry then result = - 1's complement of result	**Subtraction M-N is done By:** - Get 2's Complement of N - Add M + N - If carry then discard carry - If no carry then result = - 2's complement of result

Digital Logic Gates

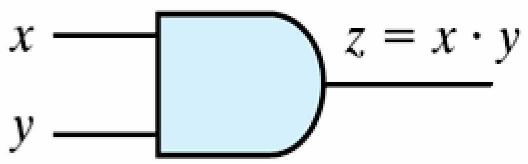
- Definition of Binary Logic
  - Binary logic consists of binary variables and a set of logical operations.
  - The variables are designated by letters of the alphabet, such as *A*, *B*, *C*, *x*, *y*, *z*, etc, with each variable having two and only two distinct possible values: 1 and 0,
  - Three basic logical operations: AND, OR, and NOT.

1. AND: This operation is represented by a dot or by the absence of an operator. For example, $x \cdot y = z$ or $xy = z$ is read "x AND y is equal to z," The logical operation AND is interpreted to mean that $z = 1$ if only $x = 1$ and $y = 1$; otherwise $z = 0$. (Remember that x, y, and z are binary variables and can be equal either to 1 or 0, and nothing else.)
2. OR: This operation is represented by a plus sign. For example, $x + y = z$ is read "x OR y is equal to z," meaning that $z = 1$ if $x = 1$ or $y = 1$ or if both $x = 1$ and $y = 1$. If both $x = 0$ and $y = 0$, then $z = 0$.
3. NOT: This operation is represented by a prime (sometimes by an overbar). For example, x' = z (or $\bar{x} = z$) is read "not x is equal to z," meaning that z is what z is not. In other words, if $x = 1$, then $z = 0$, but if $x = 0$, then $z = 1$, The NOT operation is also referred to as the complement operation, since it changes a 1 to 0 and a 0 to 1.

Digital Logic Gates

- Truth Tables, Boolean Expressions, and Logic Gates

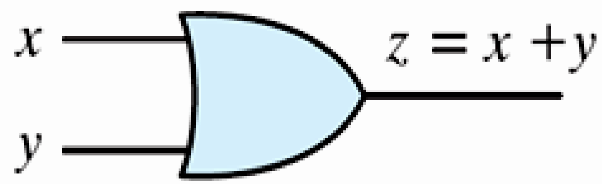
AND



(a) Two-input AND gate

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

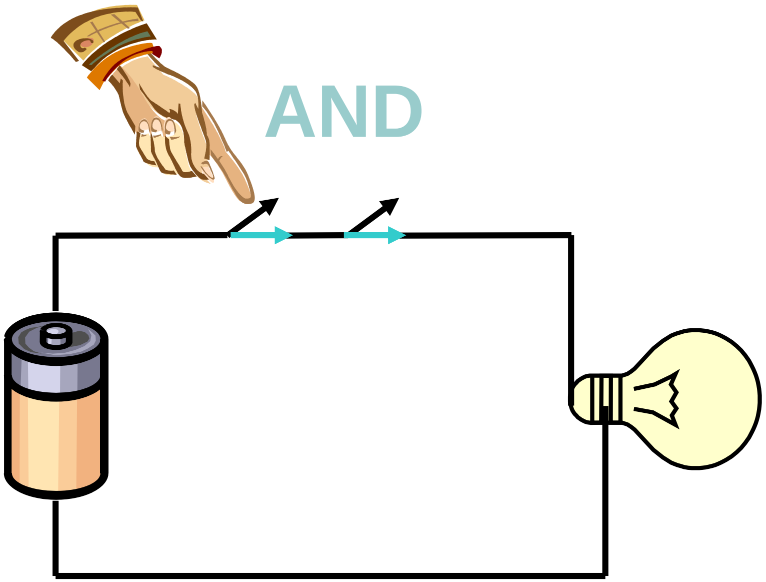
OR

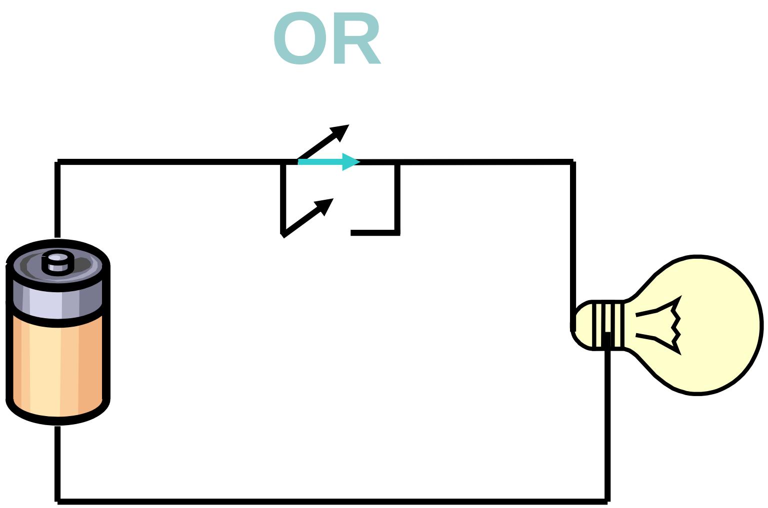


(b) Two-input OR gate

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

Switching Circuits

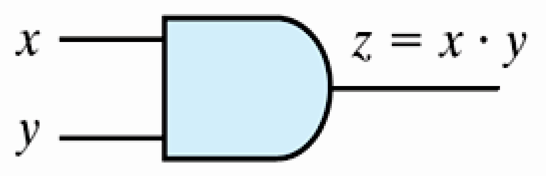




Digital Logic Gates

- Truth Tables, Boolean Expressions, and Logic Gates

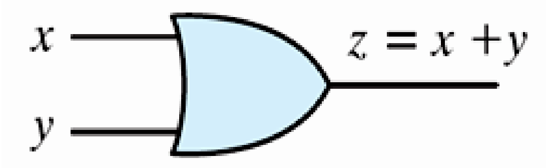
AND



(a) Two-input AND gate

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

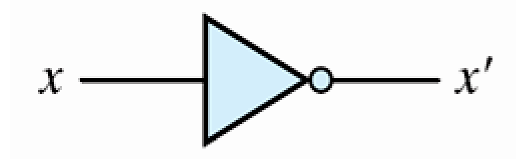
OR



(b) Two-input OR gate

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

NOT



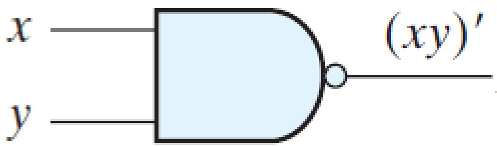
(c) NOT gate or inverter

x	F
0	1
1	0

Digital Logic Gates

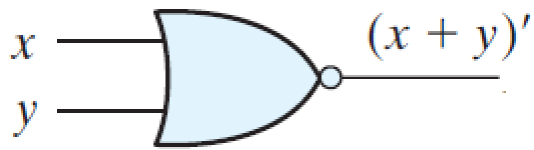
- Truth Tables, Boolean Expressions, and Logic Gates

NAND



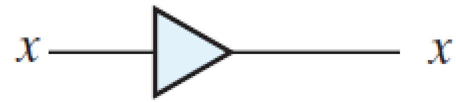
x	y	F
0	0	1
0	1	1
1	0	1
1	1	0

NOR



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

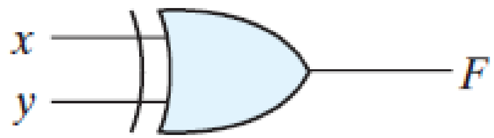
Buffer



x	F
0	0
1	1

Digital Logic Gates

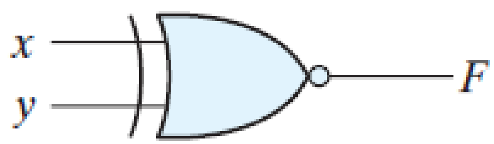
Exclusive-OR
(XOR)



$$F = xy' + x'y = x \oplus y$$

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

Exclusive-NOR
or
equivalence



$$F = xy + x'y' = (x \oplus y)'$$

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

Digital Logic Gates

- Logic gates
  - Graphic Symbols and Input-Output Signals for Logic gates:

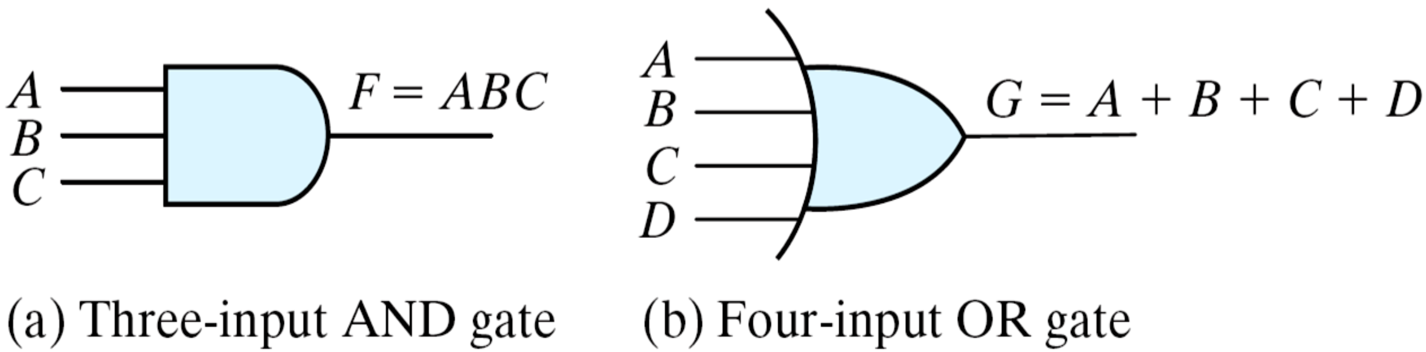


Fig. 1.6 Gates with multiple inputs

Chapter 2:

Boolean Algebra and Logic Gates

Outlines

1. Basic Definitions
2. Axiomatic Definition of Boolean Algebra
3. Basic Theorems and Properties of Boolean Algebra
4. Boolean Functions
5. Canonical and Standard Forms
6. Other Logical Operations

Boolean Algebra

- Finding **simpler** and **cheaper**, but **equivalent**, realizations of a circuit can reap huge payoffs in reducing the overall cost of the design.

- Mathematical methods that simplify circuits rely primarily on Boolean algebra.

- Therefore, this chapter provides a basic vocabulary and a brief foundation in Boolean algebra that will **enable you to optimize simple circuits**

Algebras

- **What is an algebra?**
  - Mathematical system consisting of
    - Set of elements (example: $N = \{1,2,3,4,\ldots\}$)
    - Set of operators (+, -, ×, ÷)
    - Axioms or postulates (associativity, distributivity, closure, identity elements, etc.)
- **Why is it important?**
  - Defines rules of "calculations"
- **Note: operators with two inputs are called *binary***
  - Does not mean they are restricted to binary numbers!
  - Operator(s) with one input are called *unary*

Axiomatic Definition of Boolean Algebra

- We need to define algebra for binary values
  - Developed by George Boole in 1854

- Huntington postulates (1904) for Boolean algebra :
- $B = \{0, 1\}$ and two binary operations, (+) and (.)

- Terminology:
  - *Literal:* A variable or its complement
  - *Product term:* literals connected by (·)
  - *Sum term:* literals connected by (+)

Basic Definitions

- **The Postulates Boolean Algebra**
  - Closure (+ and ·)
  - The identity elements
    - + → 0
    - · → 1

AND

x	y	x.y
0	0	0
0	1	0
1	0	0
1	1	1

OR

x	y	x+y
0	0	0
0	1	1
1	0	1
1	1	1

NOT

x	x’
0	1
1	0

Basic Definitions

- **The Postulates Boolean Algebra**
  - The commutative laws $x+y = y+x$, $x.y = y.x$
  - The distributive laws $x . (y+z) = (x.y)+(x.z)$

x	y	z
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Basic Definitions

- **The Postulates Boolean Algebra**
  - The commutative laws $x+y = y+x$, $x.y = y.x$
  - The distributive laws $x.(y+z) = (x.y)+(x.z)$

x	y	z	$y+z$	$x \cdot (y+z)$	$x \cdot y$	$x \cdot z$	$(x \cdot y)+(x \cdot z)$
0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	0	0	0	0
1	0	1	1	1	0	1	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

Basic Definitions

- **The Postulates Boolean Algebra**
  - The distributive laws $x + (y.z) = (x+y).(x+z)$

x	y	z	$y.z$	$x+(y.z)$	$x+y$	$x+z$	$(x+y).(x+z)$
0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0
0	1	1	1	1	1	1	1
1	0	0	0	1	1	1	1
1	0	1	0	1	1	1	1
1	1	0	0	1	1	1	1
1	1	1	1	1	1	1	1

Basic Definitions

- **The Postulates Boolean Algebra**
  - Complement
  - $x + x' = 1$, since
    - $0+0'=0+1=1$;
    - $1+1'=1+0=1$
  - $x \cdot x' = 0$, since
    - $0 \cdot 0'=0 \cdot 1=0$;
    - $1 \cdot 1'=1 \cdot 0=0$

Basic Definitions

- **Duality Principle (DeMorgan's Law)**
  - Every algebraic expression deducible from the postulates of Boolean algebra remains valid if the operators and identity elements are interchanged.
  - To get dual form:
    - Interchange OR(+) and AND(.)
    - Toggle 0's and 1's

Basic Definitions

- **Duality Principle (DeMorgan's Theorem)**
- Verify DeMorgan's Theorem

$$(x+y)' = x'y'$$
$$(x\ y)' = x'+y'$$

x	y	x'	y'	$x+y$	$(x+y)'$	$x'y'$	Xy	$x'+y'$	$(xy)'$
0	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	0	0	1	1
1	0	0	1	1	0	0	0	1	1
1	1	0	0	1	0	0	1	0	0

Basic Definitions

- **The Postulates Boolean Algebra**

Table 2.1
Postulates and Theorems of Boolean Algebra

Postulate 2	(a)	$x + 0 = x$	(b)	$x \cdot 1 = x$
Postulate 5	(a)	$x + x' = 1$	(b)	$x \cdot x' = 0$
Theorem 1	(a)	$x + x = x$	(b)	$x \cdot x = x$
Theorem 2	(a)	$x + 1 = 1$	(b)	$x \cdot 0 = 0$
Theorem 3, involution		$(x')' = x$		
Postulate 3, commutative	(a)	$x + y = y + x$	(b)	$xy = yx$
Theorem 4, associative	(a)	$x + (y + z) = (x + y) + z$	(b)	$x(yz) = (xy)z$
Postulate 4, distributive	(a)	$x(y + z) = xy + xz$	(b)	$x + yz = (x + y)(x + z)$
Theorem 5, DeMorgan	(a)	$(x + y)' = x'y'$	(b)	$(xy)' = x' + y'$
Theorem 6, absorption	(a)	$x + xy = x$	(b)	$x(x + y) = x$

Basic Definitions

- **Consensus Theorem**

$$xy + x'z + yz = xy + x'z$$

Proof:

$xy + x'z + yz$

$= xy + x'z + 1.yz$

$= xy + x'z + (x+x')yz$

$= xy + x'z + xyz + x'yz$

$= (xy + xyz) + (x'z + x'zy)$

$= xy\,(1+z) + x'z\,(1+ y)$

$= xy + x'z$

$$(x+y)\cdot(x'+z)\cdot(y+z) = (x+y)\cdot(x'+z)$$

Proof:

$(x+y)\bullet(x'+z)\bullet(y+z)$

$= (x+y)\bullet(x'+z)\bullet(0+y+z)$

$= (x+y)\bullet(x'+z)\bullet((xx')+y+z)$

$= (x+y)\bullet(x'+z)\bullet(x+y+z)\bullet(x'+y+z)$

$= (x+y)\bullet(0\bullet z)(x'+z)\bullet(0\bullet y)$

$= (x+y)(x'+z)$

Operator Precedence

- The operator precedence for evaluating Boolean Expression is
  - Parentheses
  - NOT
  - AND
  - OR
- Examples
  - $x\,y' + z$
  - $(x\,y + z)'$

Boolean Functions

- A Boolean function my include:
  - Binary variables
  - Binary operators OR and AND
  - Unary operator NOT
  - Parentheses
- Examples
  - $F_1 = x\, y\, z'$
  - $F_2 = x + y'z$
  - $F_3 = x'\, y'\, z + x'\, y\, z + x\, y'$
  - $F_4 = x\, y' + x'\, z$

- **The truth table of 2^n entries (n=number of variables)**
- **Two Boolean expressions may specify the same function $F_3 = F_4$**

x	y	z	F_1	F_2	F_3	F_4
0	0	0	0	0	0	0
0	0	1	0	1	1	1
0	1	0	0	0	0	0
0	1	1	0	0	1	1
1	0	0	0	1	1	1
1	0	1	0	1	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	0

Boolean Functions

- Different representation of Boolean Function
  - Boolean Expression (Many)
  - Truth Table (Unique)
  - Logic Gates Diagram (Many)

- Examples
  - $F_1 = x\,y\,z'$
  - $F_2 = x + y'z$
  - $F_3 = x'\,y'\,z + x'\,y\,z + x\,y'$
  - $F_4 = x\,y' + x'\,z$

x	y	z	F_1	F_2	F_3	F_4
0	0	0	0	0	0	0
0	0	1	0	1	1	1
0	1	0	0	0	0	0
0	1	1	0	0	1	1
1	0	0	0	1	1	1
1	0	1	0	1	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	0

Boolean Functions

- Implementation with logic gates

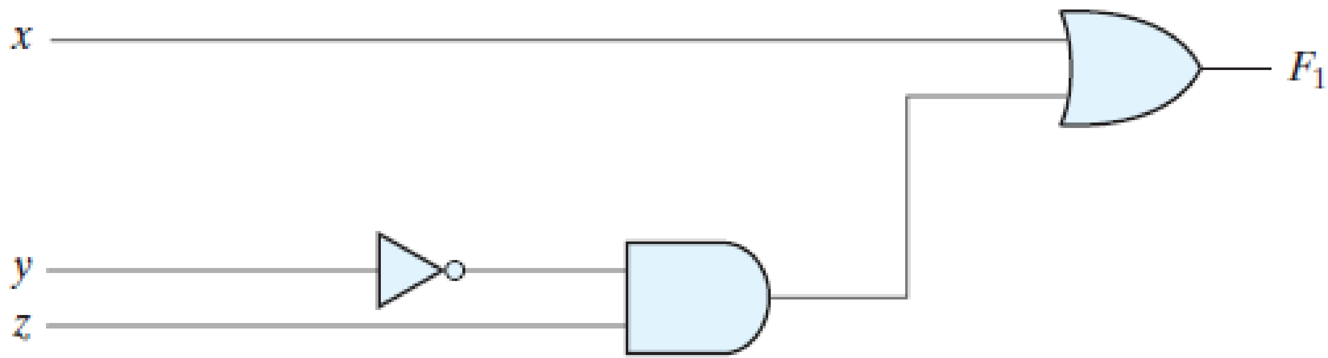


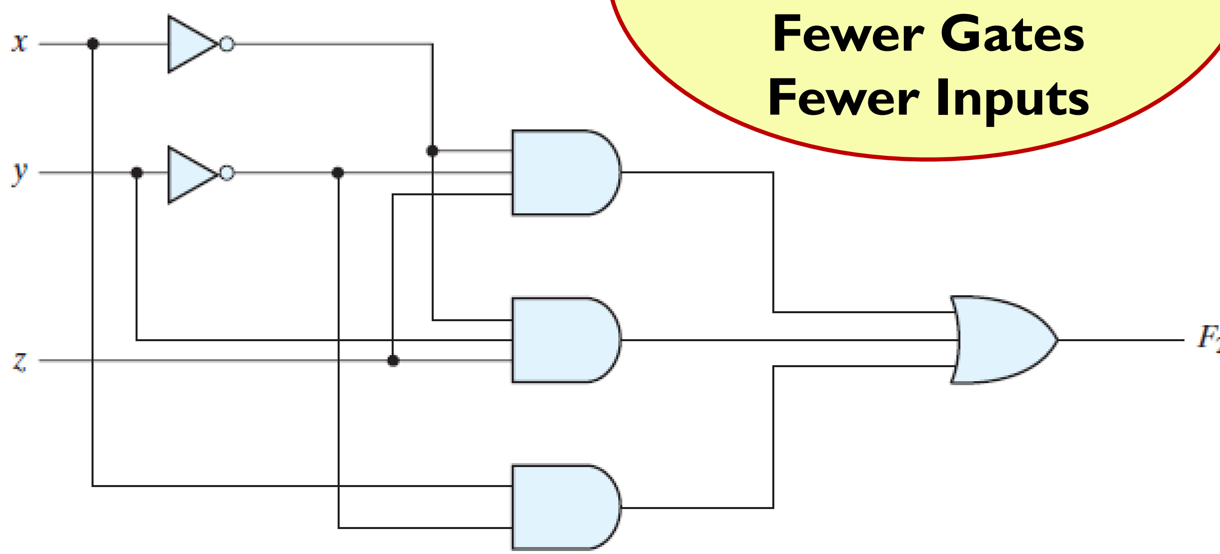
FIGURE 2.1
Gate implementation of $F_1 = x + y'z$

Boolean Functions

Economical
Simpler ,
Less Cost
Fewer Gates
Fewer Inputs

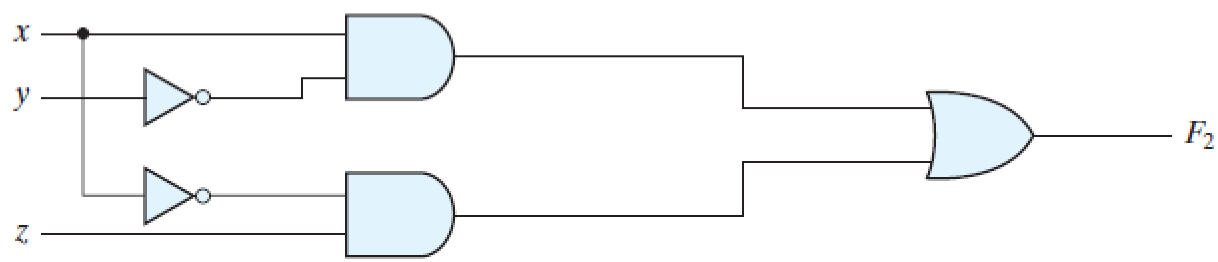
- Implementation with logic gates

$$F_2 = x'\,y'\,z + x'\,y\,z + x\,y'$$
$$= x'\,z\,(y'+y) + x\,y'$$
$$= x'\,z\,(1) + x\,y'$$
$$= x'\,z + x\,y'$$



(a) $F_2 = x'y'z + x'yz + xy'$

Simplification



(b) $F_2 = xy' + x'z$

FIGURE 2.2
Implementation of Boolean function F_2 with gates

Boolean Functions

- Simplify the following functions

$F = x(x' + y)$ $= xx' + xy$ $= 0 + xy$ $= xy$	$F = x + x'y$ $= (x + x')(x + y)$ $= 1(x + y)$ $= (x + y)$
$F = (x + y)(x + y')$ $= x + xy + xy' + yy'$ $= x(1 + y + y')$ $= x$	$F = xy + x'z + yz$ $= xy + x'z + yz(x + x')$ $= xy + x'z + xyz + x'yz$ $= xy(1+z) + x'z(1+y)$ $= xy + x'z$ **Consensus Theorem**

Complement of a Function

- The complement of a function F is F' and is obtained from an interchange of 0's for 1's and 1's for 0's in the value of F.

- The complement of a function may be derived algebraically with aid of DeMorgan's theorems,
  - 3 variables DeMorgan's theorem
  - $(A+B+C)' = (A+X)'$ //let $B+C = X$
    $= A'X'$ //by theorem 5(a) (DeMorgan's)
    $= A'(B+C)'$ //substitute $B+C = X$
    $= A'(B'C')$ //by DeMorgan's theorem
    $= A'B'C'$ //by associative theorem

Complement of a Function

- The complement of a function F is F' and is obtained from an interchange of 0's for 1's and 1's for 0's in the value of F.

- The complement of a function may be derived algebraically with aid of DeMorgan's theorems,
  - 3 variables DeMorgan's theorem
  - $(A+B+C)' = (A+X)'$ let $B+C = X$

$$
\begin{aligned}
(A+B+C)' &= (A+X)' && \text{let } B+C = X \\
&= A'X' && \text{by theorem 5(a) (DeMorgan's)} \\
&= A'(B+C)' && \text{substitute } B+C = X \\
&= A'(B'C') && \text{by DeMorgan's theorem} \\
&= A'B'C' && \text{by associative theorem}
\end{aligned}
$$

Complement of a Function

- The complement of a function *F* is *F'* and is obtained from an interchange of 0's for 1's and 1's for 0's in the value of *F*.

- Generalization: a function is obtained by interchanging AND and OR operators and complementing each literal.
  - F = $A+B+C+D+\ldots$ Then $F' = (A+B+C+D+\ldots)' = A'B'C'D'\ldots$
  - F = $ABCD\ldots$ Then $F' = (ABCD\ldots)' = A'+B'+C'+D'\ldots$

The complement of a function may be derived algebraically with aid of DeMorgan's theorems

Complement of a Function

- Find the Complement of the following functions
- $F_1 = x'\,y\,z' + x'\,y'\,z$
- $F_2 = x(y'\,z' + y\,z)$

- $F_1' = (x'yz' + x'y'z)' = (x'yz')'\,(x'y'z)' = (x+y'+z)\,(x+y+z')$

- $F_2' = [x(y'z'+yz)]' = x' + (y'z'+yz)' = x' + (y'z')'\,(yz)'$
  $= x' + (y+z)\,(y'+z') = x' + yz'+y'z$


Thank You!