

# Outline

## Other Logic Gates and their uses

# Other Logic Operations









*Truth Tables for the 16 Functions of Two Binary Variables*

<i>x</i>	<i>y</i>	<i>F</i> <sub>0</sub>	<i>F</i> <sub>1</sub>	<i>F</i> <sub>2</sub>	<i>F</i> <sub>3</sub>	<i>F</i> <sub>4</sub>	<i>F</i> <sub>5</sub>	<i>F</i> <sub>6</sub>	<i>F</i> <sub>7</sub>	<i>F</i> <sub>8</sub>	<i>F</i> <sub>9</sub>	<i>F</i> <sub>10</sub>	<i>F</i> <sub>11</sub>	<i>F</i> <sub>12</sub>	<i>F</i> <sub>13</sub>	<i>F</i> <sub>14</sub>	<i>F</i> <sub>15</sub>
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

*Boolean Expressions for the 16 Functions of Two Variables*

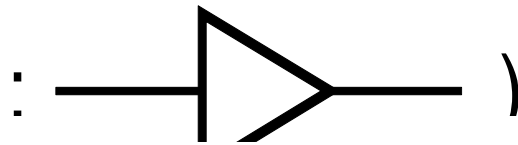
<b>Boolean Functions</b>	<b>Operator Symbol</b>	<b>Name</b>	<b>Comments</b>
$F_0 = 0$		Null	Binary constant 0
$F_1 = xy$	$x \cdot y$	AND	$x$ and $y$
$F_2 = xy'$	$x/y$	Inhibition	$x$ , but not $y$
$F_3 = x$		Transfer	$x$
$F_4 = x'y$	$y/x$	Inhibition	$y$ , but not $x$
$F_5 = y$		Transfer	$y$
$F_6 = xy' + x'y$	$x \oplus y$	Exclusive-OR	$x$ or $y$ , but not both
$F_7 = x + y$	$x + y$	OR	$x$ or $y$
$F_8 = (x + y)'$	$x \downarrow y$	NOR	Not-OR
$F_9 = xy + x'y'$	$(x \oplus y)'$	Equivalence	$x$ equals $y$
$F_{10} = y'$	$y'$	Complement	Not $y$
$F_{11} = x + y'$	$x \subset y$	Implication	If $y$ , then $x$
$F_{12} = x'$	$x'$	Complement	Not $x$
$F_{13} = x' + y$	$x \supset y$	Implication	If $x$ , then $y$
$F_{14} = (xy)'$	$x \uparrow y$	NAND	Not-AND
$F_{15} = 1$		Identity	Binary constant 1

# Digital logic gates

Name	Graphic symbol	Algebraic function	Truth table															
AND		$F = xy$	<table border="1"> <thead> <tr> <th>x</th> <th>y</th> <th>F</th> </tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </tbody> </table>	x	y	F	0	0	0	0	1	0	1	0	0	1	1	1
x	y	F																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
OR		$F = x + y$	<table border="1"> <thead> <tr> <th>x</th> <th>y</th> <th>F</th> </tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </tbody> </table>	x	y	F	0	0	0	0	1	1	1	0	1	1	1	1
x	y	F																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
Inverter		$F = x'$	<table border="1"> <thead> <tr> <th>x</th> <th>F</th> </tr> </thead> <tbody> <tr><td>0</td><td>1</td></tr> <tr><td>1</td><td>0</td></tr> </tbody> </table>	x	F	0	1	1	0									
x	F																	
0	1																	
1	0																	
Buffer		$F = x$	<table border="1"> <thead> <tr> <th>x</th> <th>F</th> </tr> </thead> <tbody> <tr><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td></tr> </tbody> </table>	x	F	0	0	1	1									
x	F																	
0	0																	
1	1																	
NAND		$F = (xy)'$	<table border="1"> <thead> <tr> <th>x</th> <th>y</th> <th>F</th> </tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>0</td></tr> </tbody> </table>	x	y	F	0	0	1	0	1	1	1	0	1	1	1	0
x	y	F																
0	0	1																
0	1	1																
1	0	1																
1	1	0																
NOR		$F = (x + y)'$	<table border="1"> <thead> <tr> <th>x</th> <th>y</th> <th>F</th> </tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>0</td></tr> </tbody> </table>	x	y	F	0	0	1	0	1	0	1	0	0	1	1	0
x	y	F																
0	0	1																
0	1	0																
1	0	0																
1	1	0																
Exclusive-OR (XOR)		$F = xy' + x'y$ $= x \oplus y$	<table border="1"> <thead> <tr> <th>x</th> <th>y</th> <th>F</th> </tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>0</td></tr> </tbody> </table>	x	y	F	0	0	0	0	1	1	1	0	1	1	1	0
x	y	F																
0	0	0																
0	1	1																
1	0	1																
1	1	0																
Exclusive-NOR or equivalence		$F = xy + x'y'$ $= (x \oplus y)'$	<table border="1"> <thead> <tr> <th>x</th> <th>y</th> <th>F</th> </tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </tbody> </table>	x	y	F	0	0	1	0	1	0	1	0	0	1	1	1
x	y	F																
0	0	1																
0	1	0																
1	0	0																
1	1	1																

## Buffer

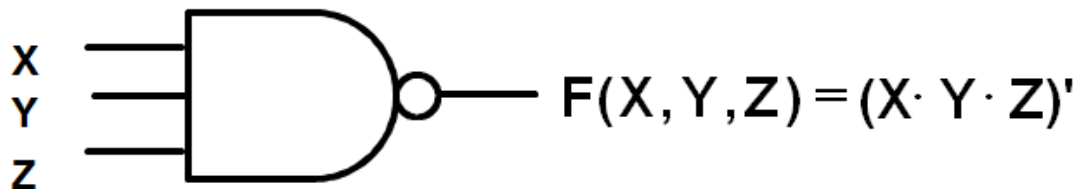
- A buffer is a gate with the function  $F = X$ :



- In terms of Boolean function, a buffer is the same as a connection!
- So why use it?
  - A buffer is an electronic amplifier used to improve circuit voltage levels and increase the speed of circuit operation.

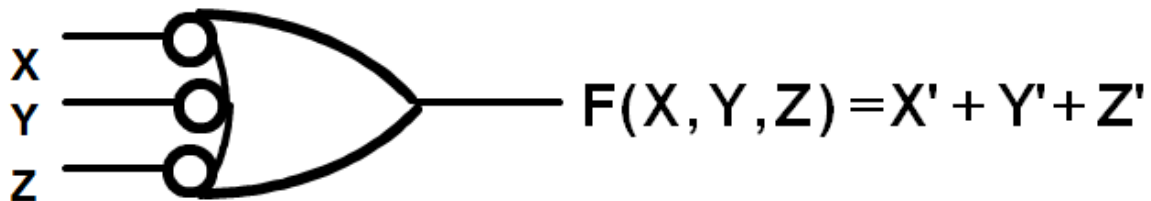
## NAND Gate

- The basic NAND gate has the following symbol, illustrated for three inputs:
  - AND-Invert (NAND)



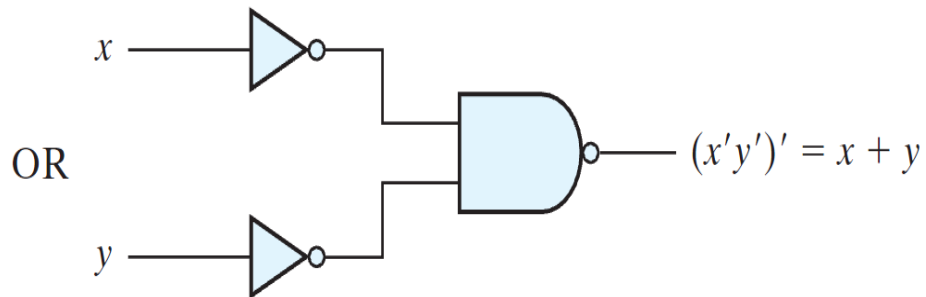
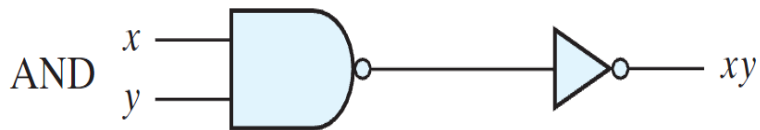
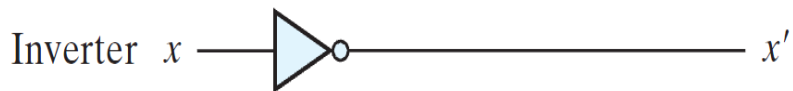
- NAND represents NOT AND, i. e., the AND function with a NOT applied. The symbol shown is an AND-Invert. The small circle (“bubble”) represents the invert function.

- Applying DeMorgan's Law gives Invert-OR (NAND)



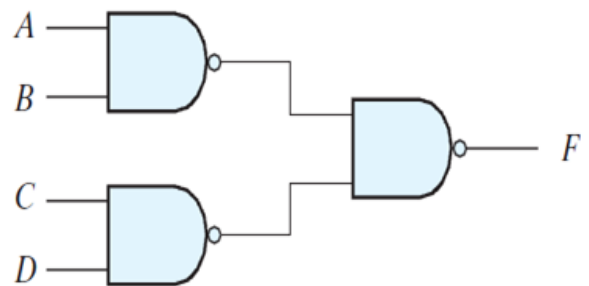
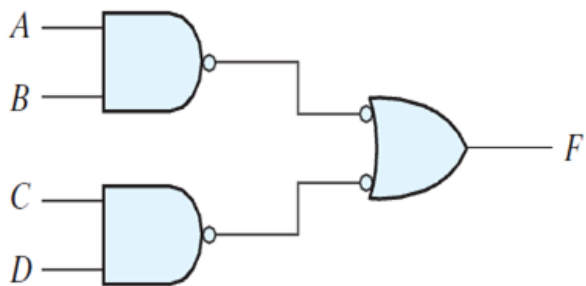
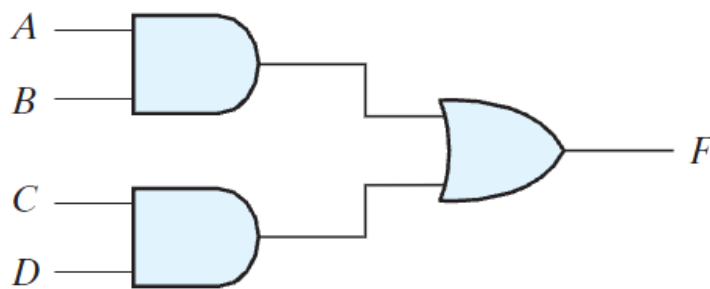
- This NAND symbol is called Invert-OR, since inputs are inverted and then ORed together.
- AND-Invert and Invert-OR both represent the NAND gate. Having both makes visualization of circuit function easier.
- A NAND gate with one input degenerates to an inverter.
- The NAND gate is the natural implementation for CMOS technology in terms of chip area and speed.

- *Universal gate* - a gate type that can implement any Boolean function.
- The NAND gate is a universal gate

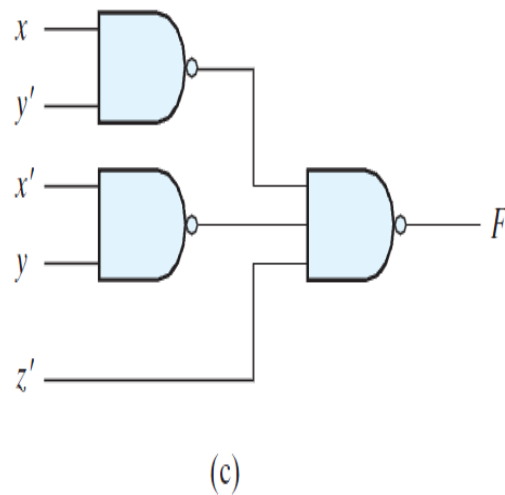
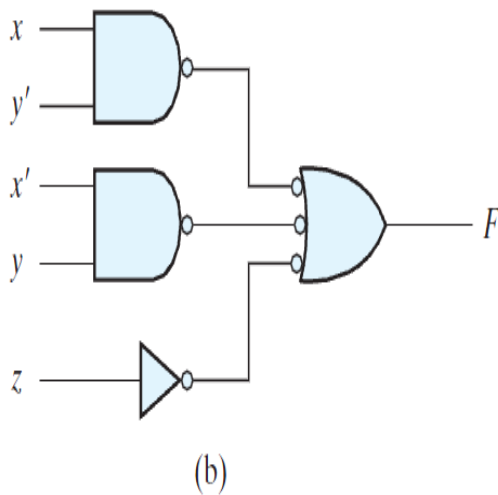


## NAND circuit

$$F = AB + CD$$



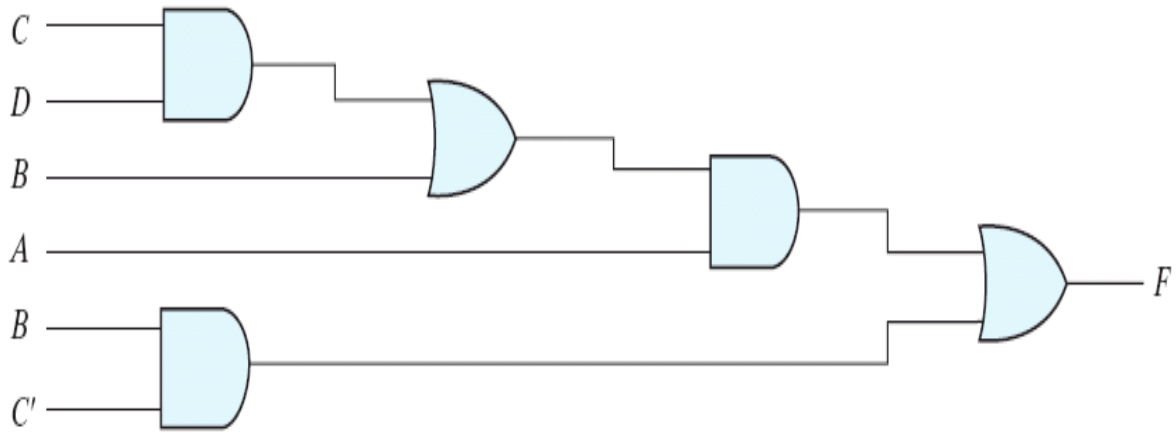
$$F = xy' + x'y + z$$



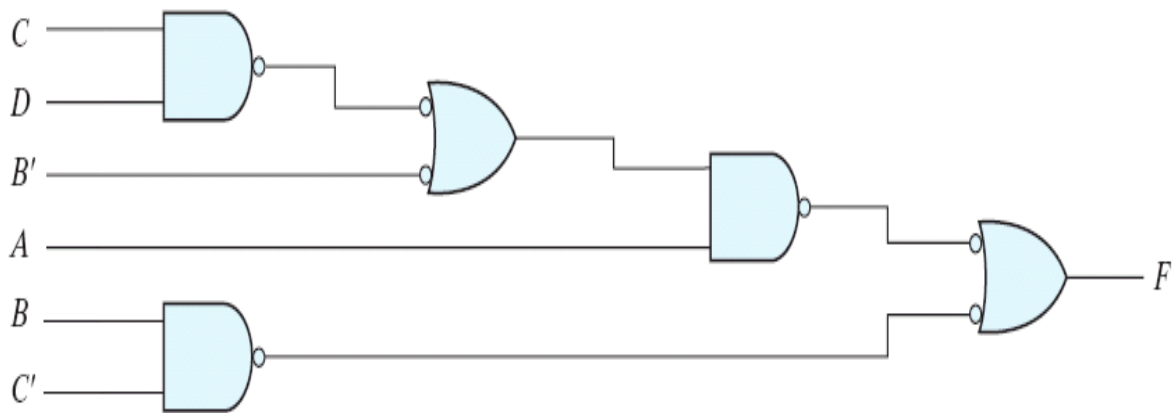
1. Convert all AND gates to NAND gates with AND-invert graphic symbols
2. Convert all OR gates to NAND gates with invert-OR graphic symbols
3. For every bubble that is not compensated by another small circle along the same line, insert an inverter or complement the input literal.



$$F=A(CD+B)+BC'$$



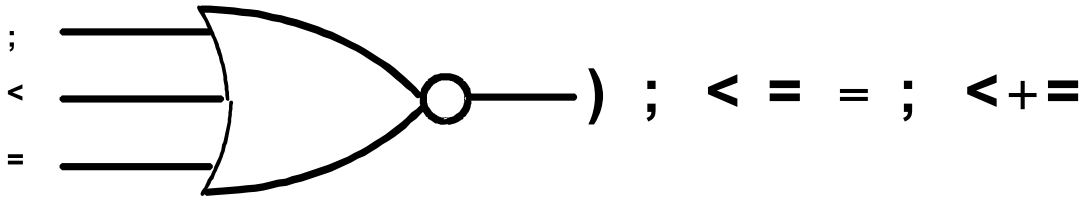
(a) AND-OR gates



(b) NAND gates

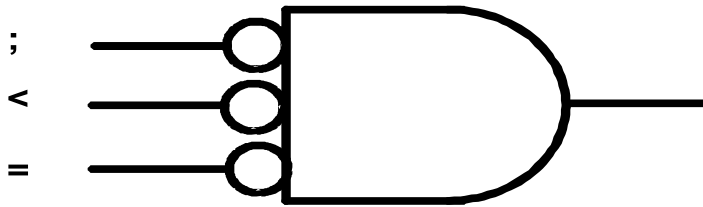
## NOR Gate

- The basic NOR gate has the following symbol, illustrated for three inputs:
  - OR-Invert (NOR)



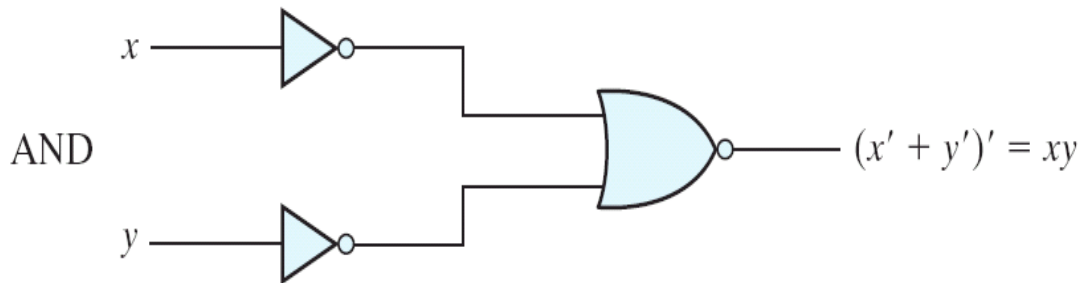
- NOR represents NOT - OR, i. e., the OR function with a NOT applied. The symbol shown is an OR-Invert. The small circle (“bubble”) represents the invert function.

- Applying DeMorgan's Law gives Invert-AND (NOR)



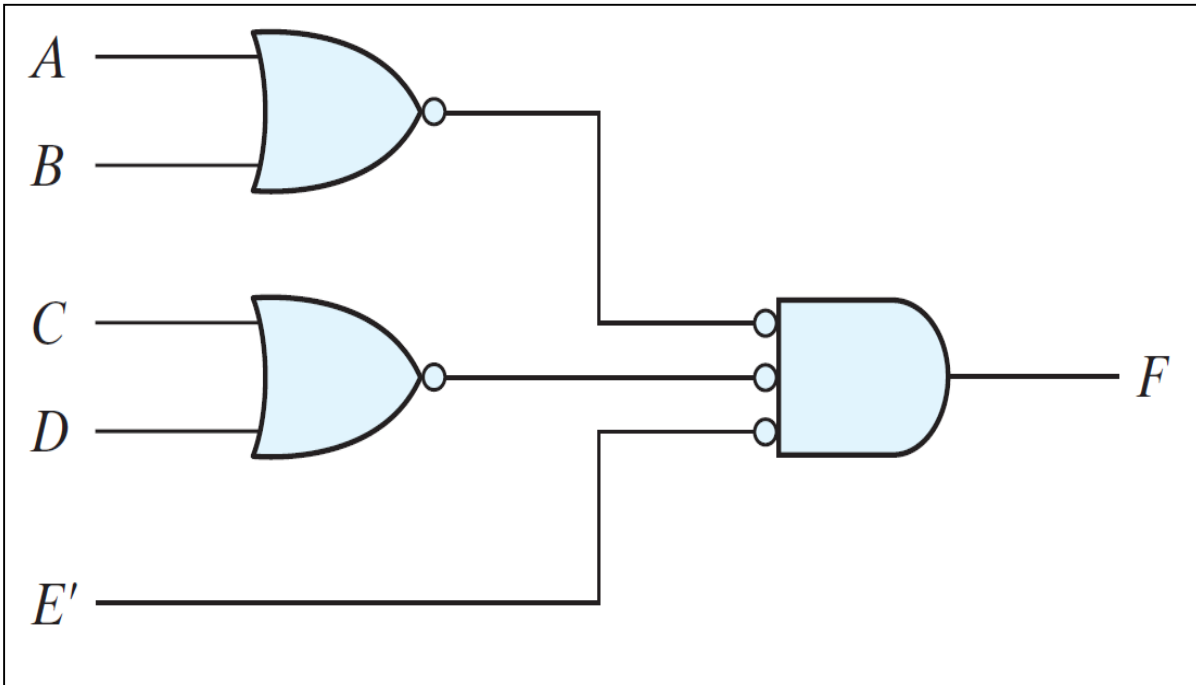
- This NOR symbol is called Invert-AND, since inputs are inverted and then ANDed together.
- OR-Invert and Invert-AND both represent the NOR gate. Having both makes visualization of circuit function easier.
- A NOR gate with one input degenerates to an inverter.
- The NOR gate is a natural implementation for some technologies other than CMOS in terms of chip area and speed.

- The NOR gate is a universal gate



# NOR circuit

$$F=(A+B)(C+D)E$$



- Convert all OR gates to NOR gates with OR-invert graphic symbols
- Convert all AND gates to NOR gates with invert-AND graphic symbols
- For every bubble that is not compensated by another small circle along the same line, insert an inverter or complement the input literal.

## Exclusive OR/ Exclusive NOR

- The *eXclusive OR (XOR)* function is an important Boolean function used extensively in logic circuits.
- The XOR function may be;
  - implemented directly as an electronic circuit (truly a gate) or
  - implemented by interconnecting other gate types (used as a convenient representation)
- The *eXclusive NOR* function is the complement of the XOR function
- Uses for the XOR and XNORs gate include:
  - Adders/subtractors/multipliers
  - Counters/incrementers/decrementers
  - Parity generators/checkers
- Definitions
  - The XOR function is:  $X \oplus Y = XY' + X'Y$
  - The eXclusive NOR (XNOR) function,  
otherwise  $(X \oplus Y) = X Y + X' Y'$

known as *equivalence*

## Truth Tables for XOR/XNOR

- Operator Rules: XOR XNOR

X	Y	$X \oplus Y$
0	0	0
0	1	1
1	0	1
1	1	0

;	<	$;\oplus<$ RU ; $\equiv$ <

- The XOR function means:  
X OR Y, but NOT BOTH
- Why is the XNOR function also known as the *equivalence* function, denoted by the operator  $\equiv$ ?

- The XOR function can be extended to 3 or more variables. For more than 2 variables, it is called an *odd function*.

$$X \oplus Y \oplus Z = X'Y'Z + X'YZ' + XY'Z' + XYZ$$

The Truth Table

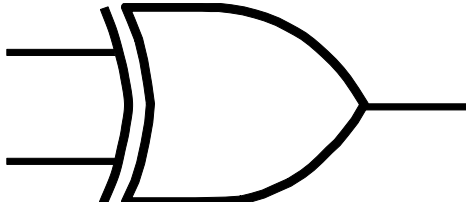
$x$	$y$	$z$	$F$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

- The complement of the odd function is the even function.
- The XOR identities:

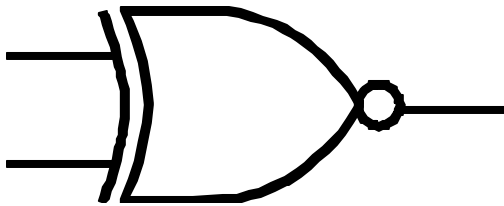
$$\begin{aligned}
 & ; \oplus = ; & ; \oplus = ; \\
 & ; \oplus ; = & ; \oplus ; = \\
 & ; \oplus < = < \oplus ; \\
 & ; \oplus < \oplus = = ; \oplus < \oplus = = ; \oplus < \oplus =
 \end{aligned}$$



- XOR symbol:



- XNOR symbol:



- Shaped symbols exist only for two inputs

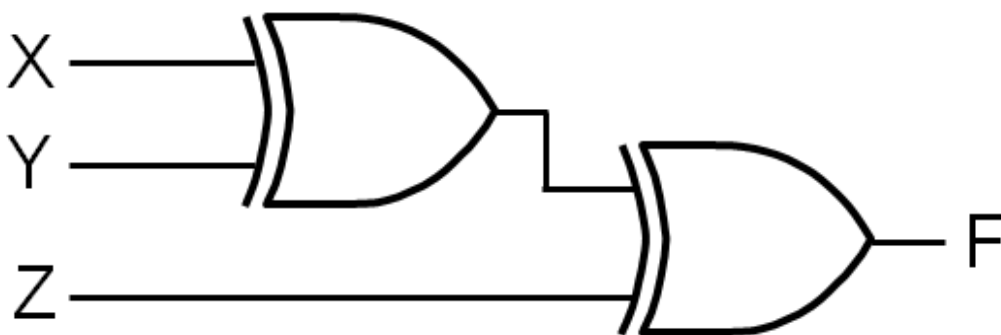
## Example: Odd Function Implementation

- Design a 3-input odd function with 2-input XOR gates

$$F = X \oplus Y \oplus Z$$

- $F = (X \oplus Y) \oplus Z$

- The circuit:

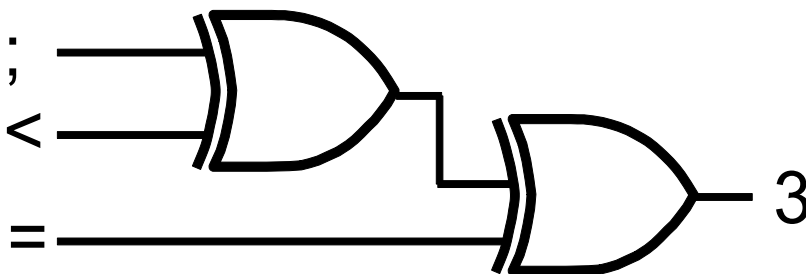


## Parity Generators and Checkers

- In Chapter 1, a parity bit added to n-bit code to produce an n + 1 bit code:
  - Add a parity bit to generate code words with even parity

*Even-Parity-Generator Truth Table*

Three-Bit Message			Parity Bit
<i>x</i>	<i>y</i>	<i>z</i>	<i>P</i>
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

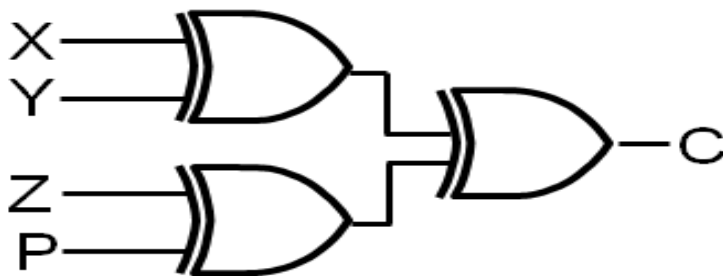


# PARITY CHECKERS

- Use parity checker circuit to check code words with even parity

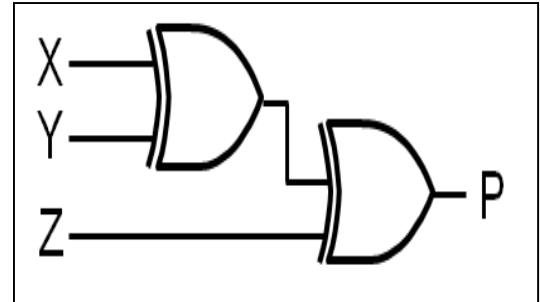
*Even-Parity-Checker Truth Table*

Four Bits Received				Parity Error Check
<i>x</i>	<i>y</i>	<i>z</i>	<i>P</i>	<i>C</i>
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

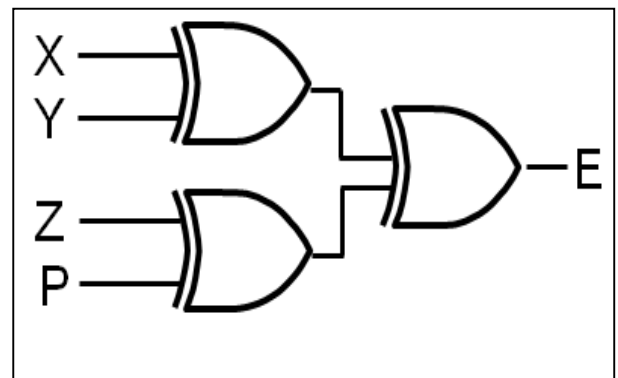


## PARITY GENERATORS AND CHECKERS

- Example:  $n = 3$ . Generate even parity code words of length four with 3-bit even parity generator:



- Check even parity code words Of length four with 4-bit even parity checker:

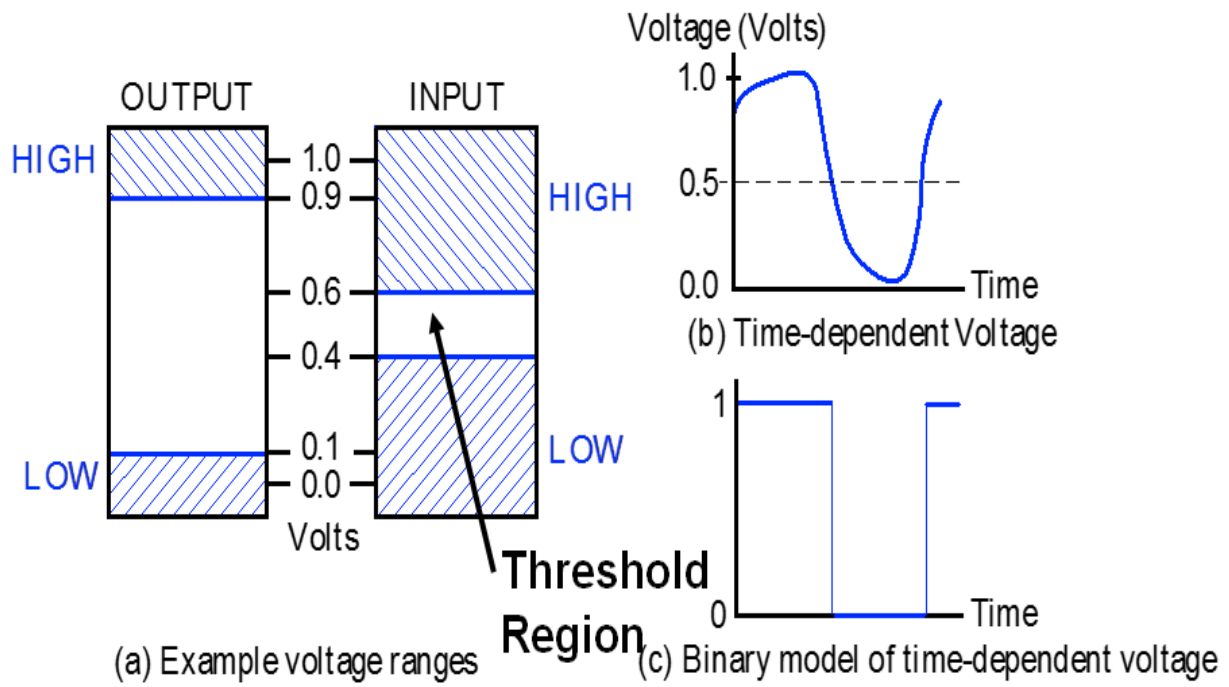


- Operation:  $(X, Y, Z) = (0, 0, 1)$  gives  $(X, Y, Z, P) = (0, 0, 1, 1)$  and  $E = 0$ .  
If Y changes from 0 to 1 between generator and checker, then  $E = 1$  indicates an error.

# INFORMATION REPRESENTATION – Signals

## Example – Physical Quantity: Voltage

- Binary values are represented by values or ranges of values of physical quantities



## Circuit Attributes

- **Power dissipation:** power consumed
- **Noise margin:** maximum external noise voltage added in an input does not cause an undesirable change in output
- **Fan-out**
  - The number of standard loads that can be connected to the output of the gate without degrading its normal operation.
  - A standard load is the amount of current needed by an input of another gate.

## Circuit Attributes

- In actual physical gates, if one or more input changes causes the output to change, the output change does not occur instantaneously.
- The delay between an input change(s) and the resulting output change is the **Propagation delay**.

