Additional Gates

COE 202

Digital Logic Design

Dr. Muhamed Mudawar

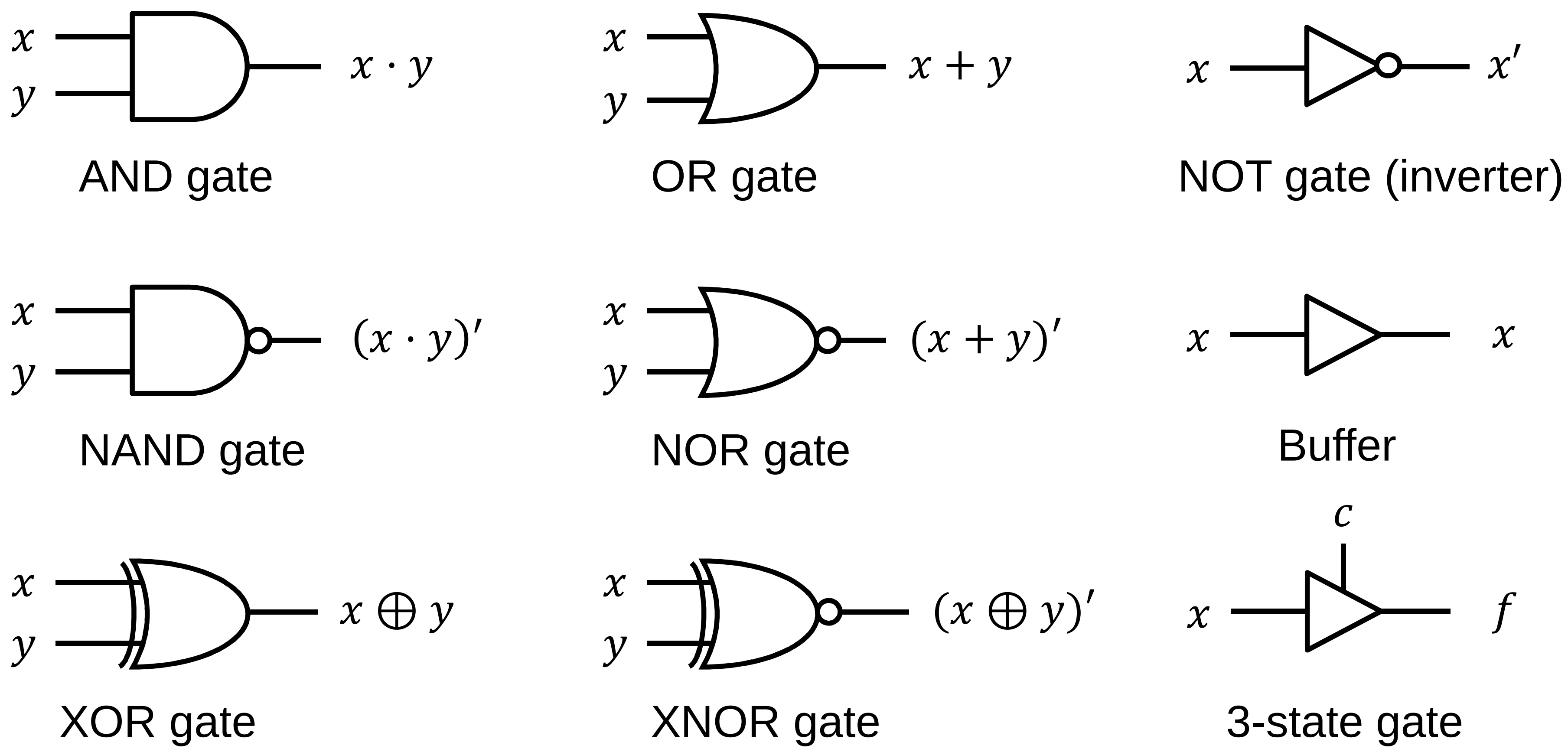
King Fahd University of Petroleum and Minerals

Presentation Outline

- Additional Gates and Symbols
- Universality of NAND and NOR gates
- NAND-NAND and NOR-NOR implementations
- Exclusive OR (XOR) and Exclusive NOR (XNOR) gates
- Odd and Even functions

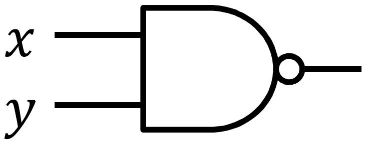
Additional Logic Gates and Symbols

- Why?
  - Low cost implementation
  - Useful in implementing Boolean functions



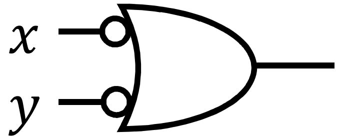
NAND Gate

- The NAND gate has the following symbol and truth table
- NAND represents **N**OT **AND**
- The small bubble circle represents the invert function



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

NAND gate



$x' + y'$

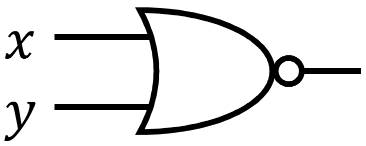
Another symbol for NAND

x	y	NAND
0	0	1
0	1	1
1	0	1
1	1	0

- NAND gate is implemented efficiently in CMOS technology
  - In terms of chip area and speed

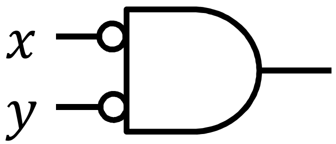
NOR Gate

- The NOR gate has the following symbol and truth table
- NOR represents **N**OT **OR**
- The small bubble circle represents the invert function



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

NOR gate



$x' \cdot y'$

Another symbol for NOR

x	y	NOR
0	0	1
0	1	0
1	0	0
1	1	0

- NOR gate is implemented efficiently in CMOS technology
  - In terms of chip area and speed

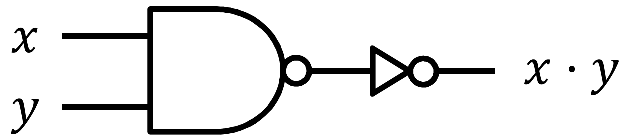
The NAND Gate is Universal

- NAND gates can implement any Boolean function
- NAND gates can be used as inverters, or to implement AND/OR
- A single-input NAND gate is an inverter

  x NAND x = $(x \cdot x)'$ = x'

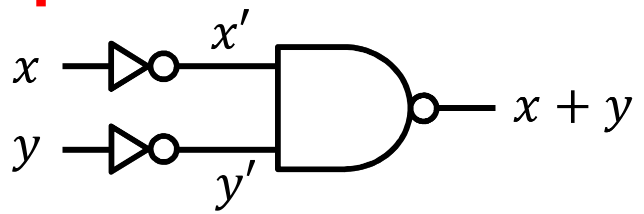
- AND is equivalent to NAND with **inverted output**

  $(x$ NAND $y)'$ = $((x \cdot y)')'$ = $x \cdot y$ (AND)



- OR is equivalent to NAND with **inverted inputs**

  $(x'$ NAND $y')$ = $(x' \cdot y')'$ = $x + y$ (OR)



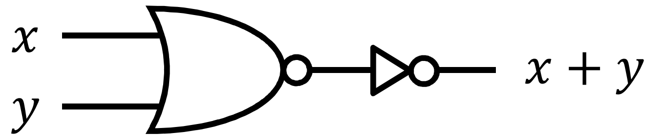
The NOR Gate is also Universal

- NOR gates can implement any Boolean function
- NOR gates can be used as inverters, or to implement AND/OR
- A single-input NOR gate is an inverter

  x NOR x = $(x + x)'$ = x'

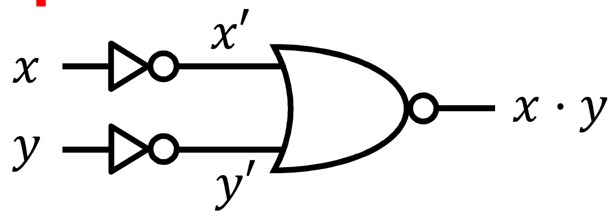
- OR is equivalent to NOR with **inverted output**

  $(x$ NOR $y)'$ = $((x + y)')'$ = $x + y$ (OR)



- AND is equivalent to NOR with **inverted inputs**

  $(x'$ NOR $y')$ = $(x' + y')'$ = $x \cdot y$ (AND)



Non-Associative NAND / NOR Operations

- Unlike AND, NAND operation is NOT associative

$(x \text{ NAND } y) \text{ NAND } z \neq x \text{ NAND } (y \text{ NAND } z)$

$(x \text{ NAND } y) \text{ NAND } z = ((xy)'z)' = ((x' + y')z)' = xy + z'$

$x \text{ NAND } (y \text{ NAND } z) = (x(yz)')' = (x(y' + z'))' = x' + yz$

- Unlike OR, NOR operation is NOT associative

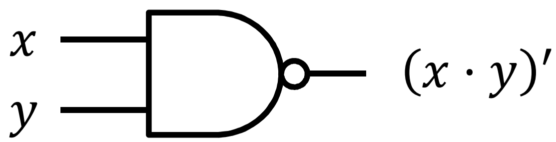
$(x \text{ NOR } y) \text{ NOR } z \neq x \text{ NOR } (y \text{ NOR } z)$

$(x \text{ NOR } y) \text{ NOR } z = \left((x + y)' + z\right)' = \left((x'y') + z\right)' = (x + y)z'$

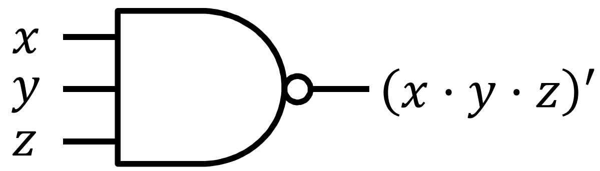
$x \text{ NOR } (y \text{ NOR } z) = (x + (y + z)')' = \left(x + (y'z')\right)' = x'(y + z)$

Multiple-Input NAND / NOR Gates

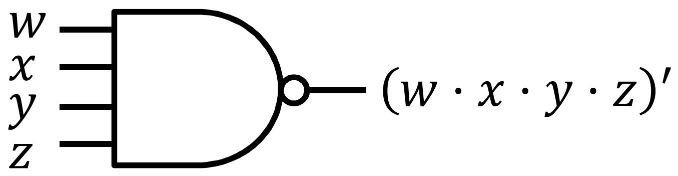
NAND/NOR gates can have multiple inputs, similar to AND/OR gates



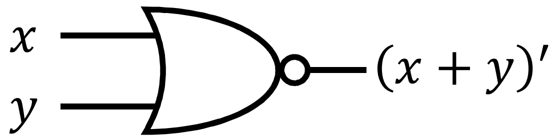
2-input NAND gate



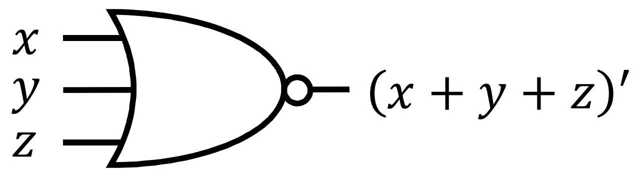
3-input NAND gate



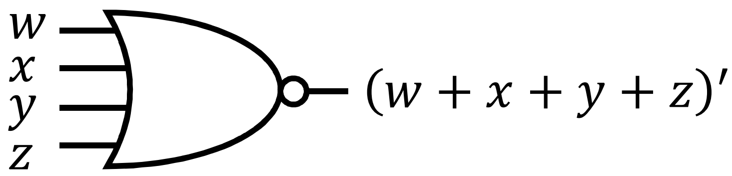
4-input NAND gate



2-input NOR gate



3-input NOR gate



4-input NOR gate

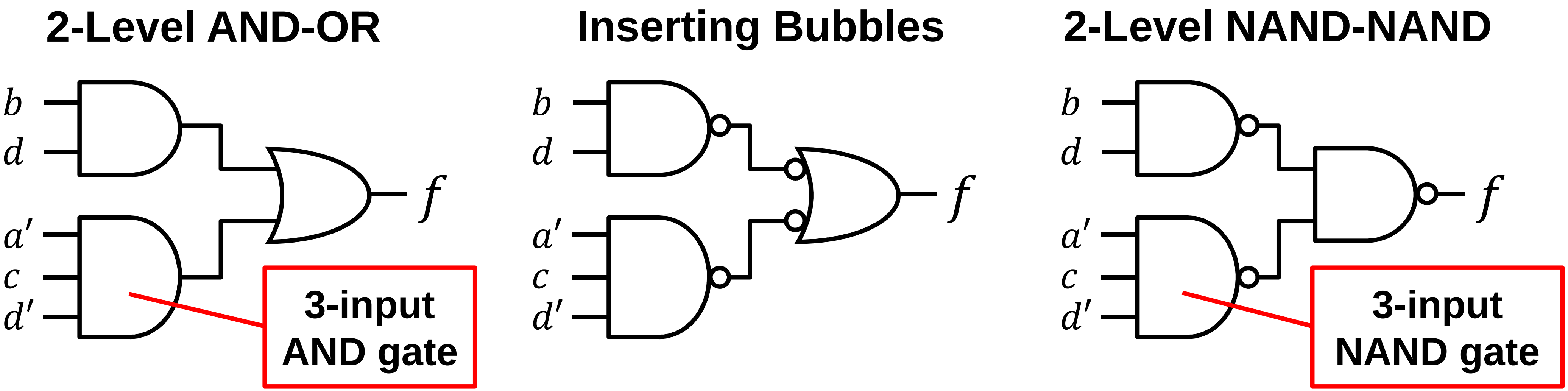
Note: a 3-input NAND is a single gate, NOT a combination of two 2-input gates.
The same can be said about other multiple-input NAND/NOR gates.

NAND - NAND Implementation

❖ Consider the following sum-of-products expression:

$$f = bd + a'cd'$$

❖ A 2-level **AND-OR** circuit can be converted easily to a 2-level **NAND-NAND implementation**



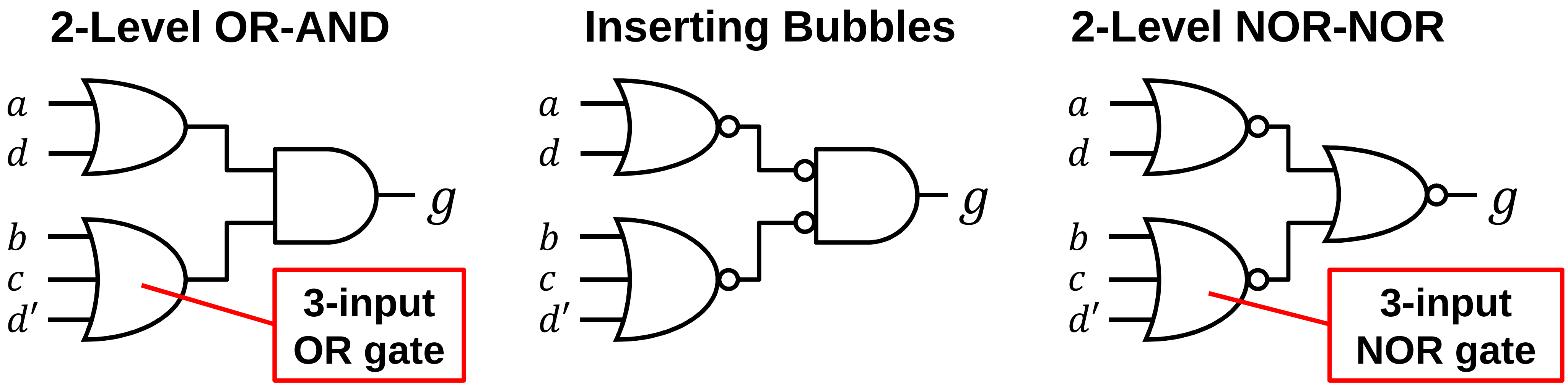
Two successive bubbles on same line cancel each other

NOR - NOR Implementation

❖ Consider the following product-of-sums expression:

$$g = (a + d)(b + c + d')$$

❖ A 2-level **OR-AND** circuit can be converted easily to a 2-level **NOR-NOR implementation**



Two successive bubbles on same line cancel each other

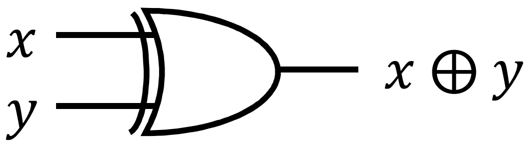
Exclusive OR / Exclusive NOR

- Exclusive OR (XOR) is an important Boolean operation used extensively in logic circuits
- Exclusive NOR (XNOR) is the complement of XOR

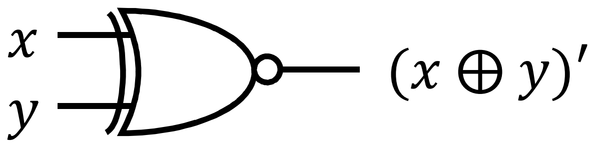
x y	XOR
0 0	0
0 1	1
1 0	1
1 1	0

x y	XNOR
0 0	1
0 1	0
1 0	0
1 1	1

XNOR is also known as **equivalence**



XOR gate



XNOR gate

XOR / XNOR Functions

- The XOR function is: $x \oplus y = xy' + x'y$
- The XNOR function is: $(x \oplus y)' = xy + x'y'$
- XOR and XNOR gates are complex
  - Can be implemented as a true gate, or by
  - Interconnecting other gate types
- XOR and XNOR gates do not exist for more than two inputs
  - For 3 inputs, use two XOR gates
  - The cost of a 3-input XOR gate is greater than the cost of two XOR gates
- Uses for XOR and XNOR gates include:
  - Adders, subtractors, multipliers, counters, incrementers, decrementers
  - Parity generators and checkers

XOR and XNOR Properties

- $x \oplus 0 = x$ $x \oplus 1 = x'$
- $x \oplus x = 0$ $x \oplus x' = 1$
- $x \oplus y = y \oplus x$
- $x' \oplus y' = x \oplus y$
- $(x \oplus y)' = x' \oplus y = x \oplus y'$

XOR and XNOR are **associative** operations

- $(x \oplus y) \oplus z = x \oplus (y \oplus z) = x \oplus y \oplus z$
- $\left((x \oplus y)' \oplus z\right)' = (x \oplus (y \oplus z)')' = x \oplus y \oplus z$

Odd Function

- Output is 1 if the **number of 1's is odd in the inputs**
- Output is the XOR operation on all input variables

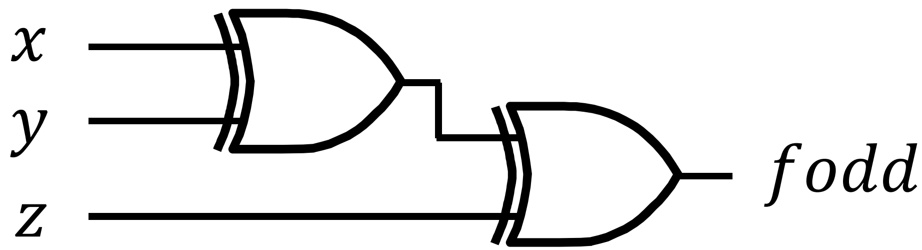
Odd Function with 3 inputs

x y z	fodd
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

$$fodd = \sum(1,2,4,7)$$

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

$$fodd = x \oplus y \oplus z$$



Implementation using two XOR gates

Even Function

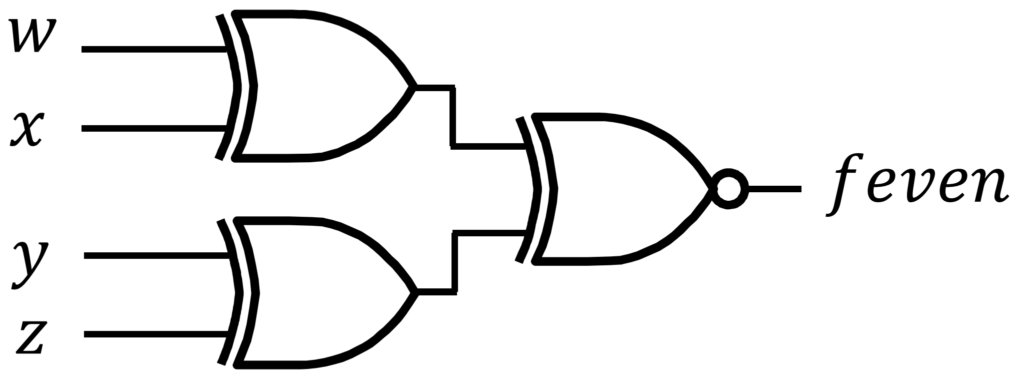
Even Function with 4 inputs

w x y z	feven
0 0 0 0	1
0 0 0 1	0
0 0 1 0	0
0 0 1 1	1
0 1 0 0	0
0 1 0 1	1
0 1 1 0	1
0 1 1 1	0
1 0 0 0	0
1 0 0 1	1
1 0 1 0	1
1 0 1 1	0
1 1 0 0	1
1 1 0 1	0
1 1 1 0	0
1 1 1 1	1

- Output is 1 if the **number of 1's is even** in the inputs (complement of odd function)
- Output is the XNOR operation on all inputs

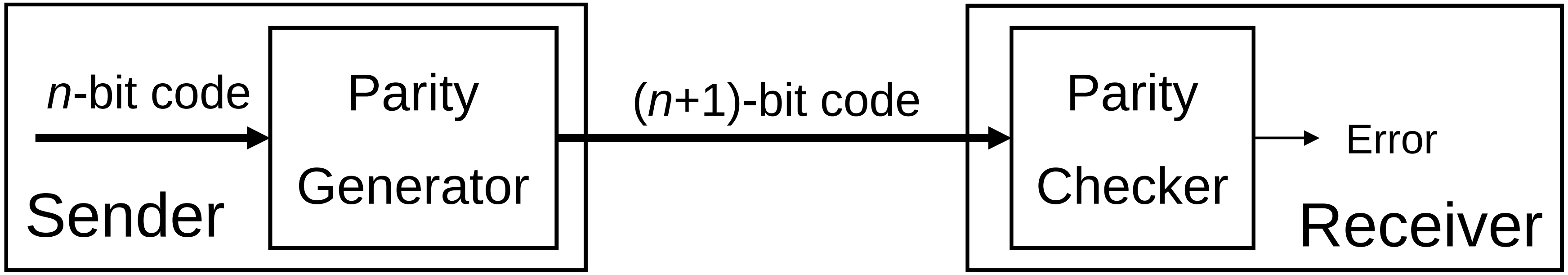
$$feven = \sum(0, 3, 5, 6, 9, 10, 12, 15)$$

$$feven = (w \oplus x \oplus y \oplus z)'$$



Implementation using two XOR gates and one XNOR

Parity Generators and Checkers



- A parity bit is added to the *n*-bit code
  - Produces (*n*+1)-bit code with an odd (or even) count of 1's
- **Odd parity:** count of 1's in the (*n*+1)-bit code is **odd**
  - Use an **even function** to generate the **odd parity bit**
  - Use an **even function** to check the (*n*+1)-bit code
- **Even parity:** count of 1's in the (*n*+1)-bit code is **even**
  - Use an **odd function** to generate the **even parity bit**
  - Use an **odd function** to check the (*n*+1)-bit code

Example of Parity Generator and Checker

❖ Design even parity generator & checker for 3-bit codes

❖ Solution:

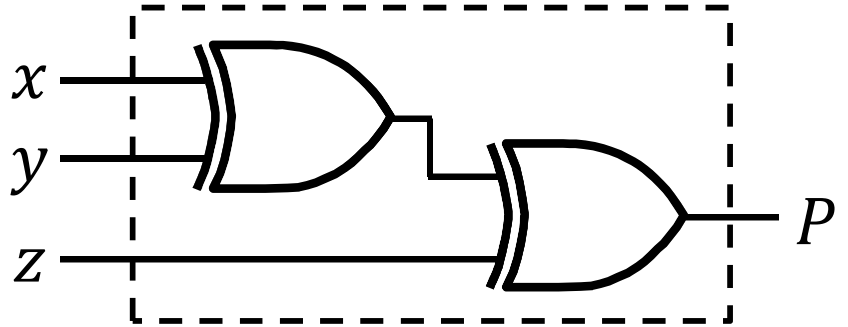
✧ Use **3-bit odd function** to generate even parity bit P.

✧ Use **4-bit odd function** to check if there is an error E in even parity.

✧ Given that: xyz = 001 then P = 1. The sender transmits $Pxyz$ = 1001.

✧ If y changes from 0 to 1 between generator and checker, the parity checker receives $Pxyz$ = 1011 and produces E = 1, indicating an error.

Parity Generator



Parity Checker