Department of Electrical & Computer Engineering

COEN 212:
DIGITAL SYSTEMS DESIGN
Lecture 5.1: Other two level logic implementations

Instructor: Dr. Reza Soleymani, Office: EV-5.125,
Telephone: 848-2424 ext.: 4103.

Lecture 5.1:
Objectives of this lecture

Concordia University
Department of Electrical & Computer Engineering

- In this lecture, we see:
  - NAND-AND implementation.
  - AND-NOR implementation.
  - OR-NAND implementation.
  - NOR-OR implementation.
  - AND-OR-INVERT or OR-AND-INVERT

- We will also talk about XOR and its application

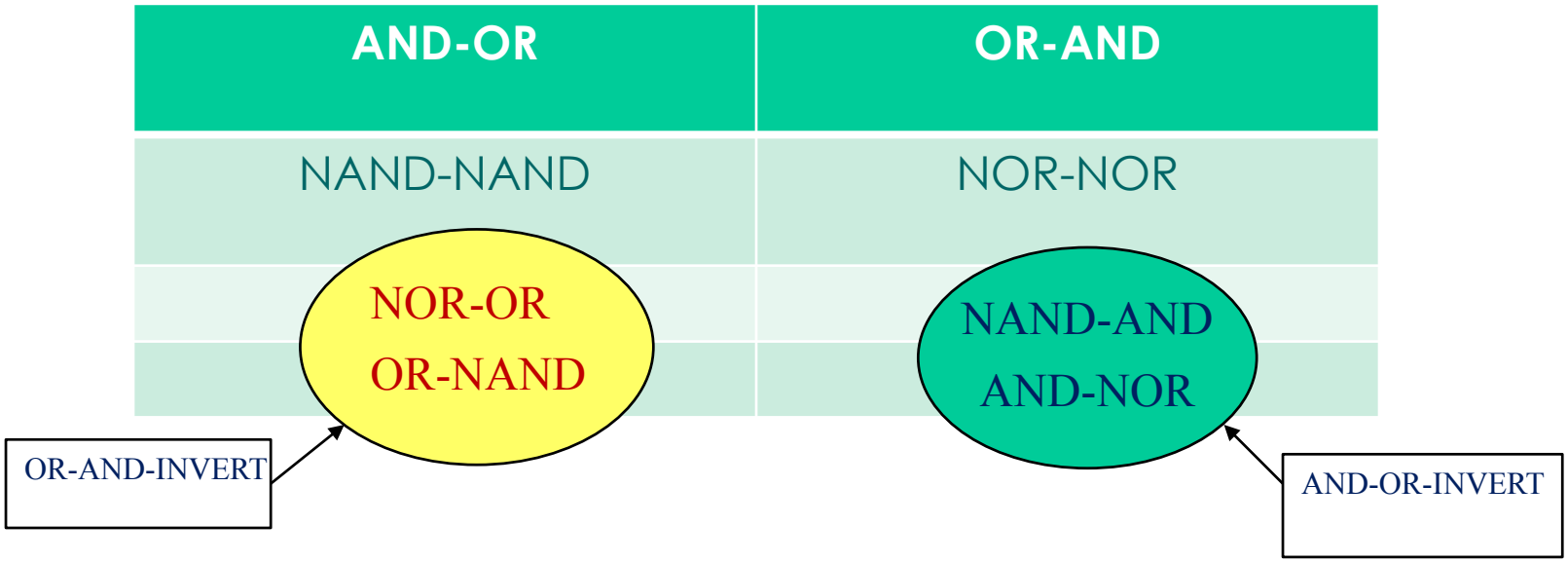
Lecture 5.1:
Reading for this lecture

- Digital Design by M. Morris R. Mano and Michael D. Ciletti, 6th Edition, Pearson, 2018:
  - Chapter 3 (3.7 and 3.8)

Lecture 5.1:
Other two-level logic combinations

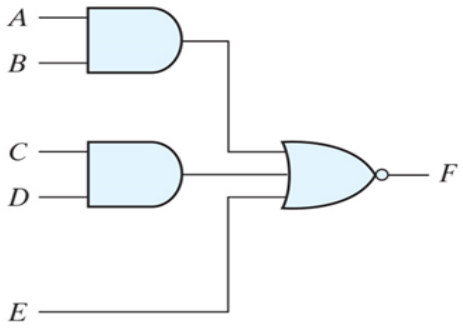
Concordia University
Department of Electrical & Computer Engineering

- Consider the four basic logic gates AND, OR, NAND, NOR.
- There are 16 possible two level arrangements.
- 8 of these combinations such as AND-AND, OR-OR degenerate into single operations.
- The non-degenerate cases are:



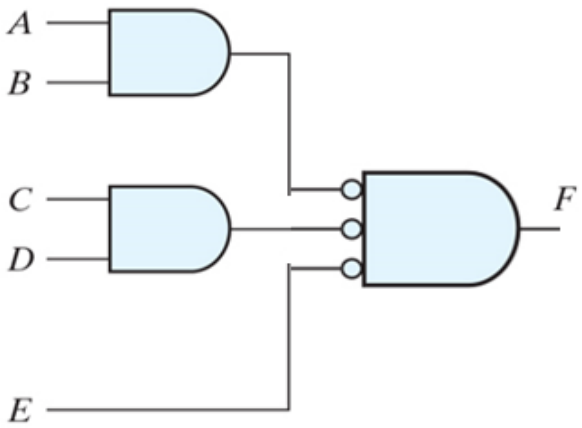
Lecture 5.1:
AND-OR-INVERT Implementation

- The AND-NOR combination, or equivalently, the AND-OR-INVERT is equivalent to the NAND-AND configuration.
- Procedure: Simply design the sum of products for the complement of the function and invert the output.
- Example: implementing $F = (AB + CD + E)'$:
  - AND-OR-INVERT:

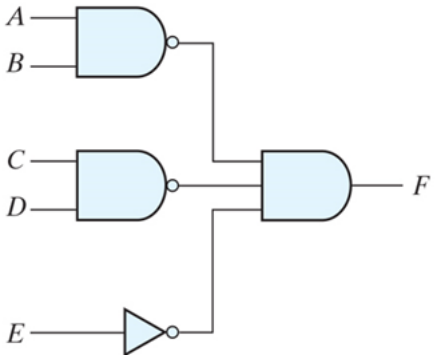


Lecture 5.1:
AND-OR-INVERT Implementation

- The circuit can be transformed into NAND-AND by:
  - Using invert-AND for NOR gate



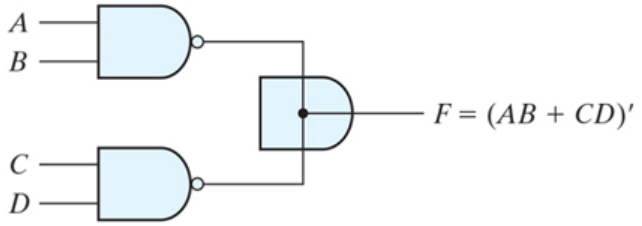
  - And moving
  - the inverters to the
  - Beginning of the line:



Lecture 5.1:
Wired Logic

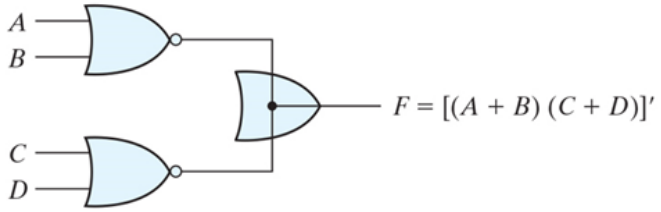
Concordia University
Department of Electrical & Computer Engineering

- Two NAND gates implemented using open collector TTL when tied together, their outputs will be AND with each other. This is called a wired AND operation.



$$F = (AB)'.(CD)' = (A' + B').(C' + D') = (AB + CD)',$$

- Also When two NOR gates implemented in ECL (Emitter Coupled Logic) are wired together, their outputs are OR'd (wired OR).

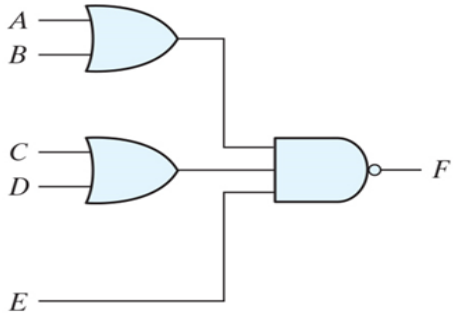


$$F = (A + B)' + (C + D)' = A'B' + C'D' = [(A + B)(C + D)]'$$

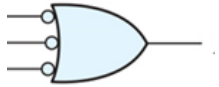
Lecture 5.1:
OR-AND-INVERT implementation

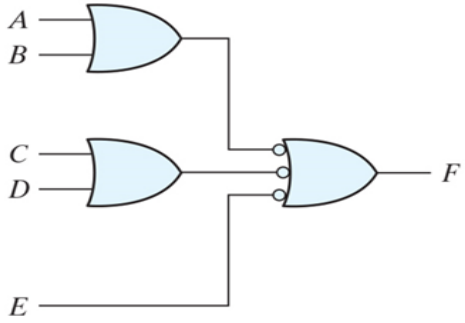
- The OR-NAND configuration is OR-AND and an inverter,
- i.e., a product of sum circuit whose output is inverted.
- It takes care of OR-NAND and NOR-OR cases.
- Procedure: Simply design the product of sums for the complement of the function and invert the output.
- Example: $F = [(A + B)(C + D)E]'$

  – OR-NAND implementation:

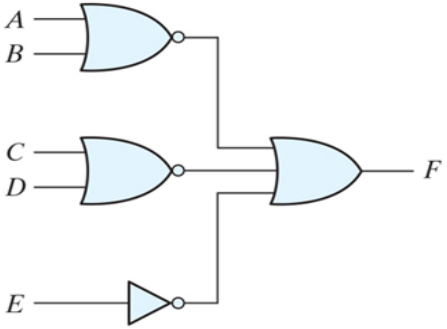


Lecture 5.1:
OR-AND-INVERT implementation

- Replacing the NAND with: 
  - We get



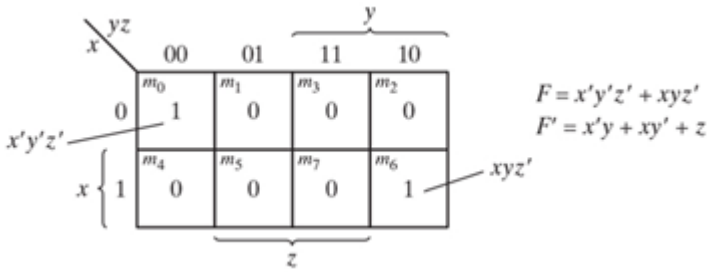
  - Or



Lecture 5.1: Example

Concordia University
Department of Electrical & Computer Engineering

- Implement the function F using:
  - a) OR-NAND, b) NOR-OR, c) AND-NOR, d) NAND-AND



We have:

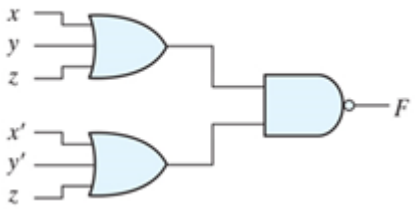
1) $F = x'y'z' + xyz' \Longrightarrow F' = (x + y + z)(z' + y' + z)$

and

2) $F' = z + xy' + x'y \Longrightarrow F = (z + xy' + x'y)'$

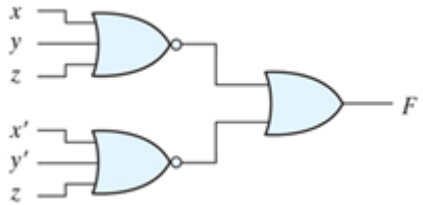
The first one gives us $F = [(x + y + z)(x' + y' + z)]'$: OR-NAND.

a) OR-NAND

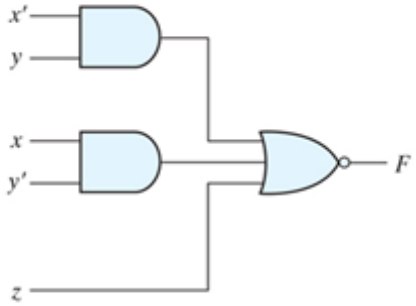


Lecture 5.1: Example

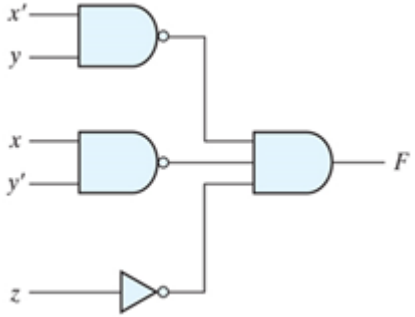
- OR-NAND can be easily changed into: NOR-OR:
  - b) NOR-OR



- $F = (z + xy' + x'y)'$ gives:
  -
  - c) AND-NOR



- That can be changed
  - d) NAND-AND

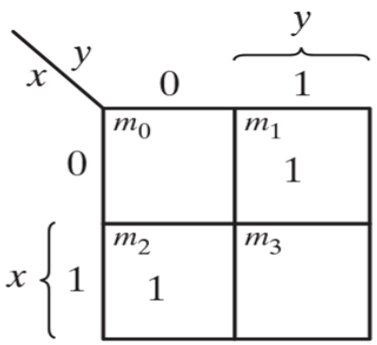


Lecture 5.1: Exclusive-OR: XOR

- The output is 1 if $x \neq y$

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

K-map



So: $x \oplus y = m_1 + m_2 = x'y + xy'$

- X-Nor is defined as the complement of XOR:

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

- X-Nor is an indicator of equality of its inputs.

$$x \oplus 0 = x$$

$$x \oplus 1 = x'$$

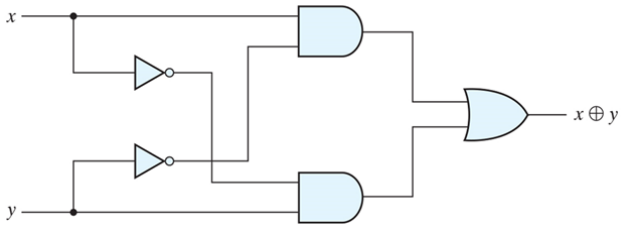
$$x \oplus x = 0$$

$$x \oplus x' = 1$$

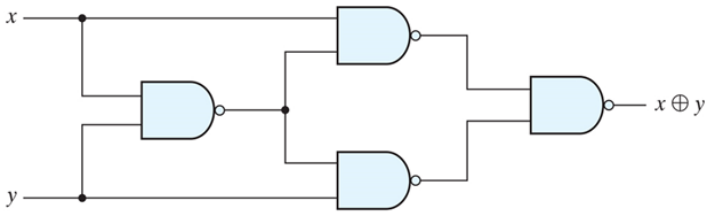
$$x \oplus y' = x' \oplus y = (x \oplus y)'$$

Lecture 5.1: XOR

- X-OR is both commutative and associative, i.e.,
  - $x \oplus y = x \oplus y$ and $(x \oplus y) \oplus z = x \oplus (y \oplus z)$
  - This assures multi-input XOR's can be implemented using 2-inpt XOR's.
  - From the equality: $x \oplus y = xy' + x'y$ we have the implementation



  - XOR can also be implemented using 4 NAND gates (verify)

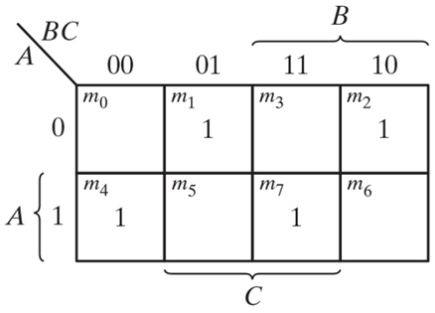


Lecture 5.1:
Using XOR: Odd Functions

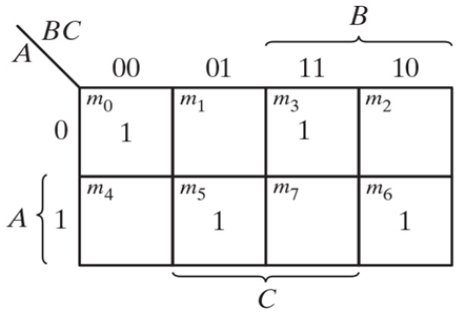
- Consider an XOR with three inputs x, y, z:

$$x \oplus y \oplus z = (x \oplus y)'z + (x \oplus y)z' = (xy' + x'y)'z + (xy' + x'y)z'$$
$$= (x'y' + xy)z + (xy' + x'y)z'$$
$$= x'y'z + xyz + xy'z' + x'yz' = \sum(1,2,4,7)$$

- The k-map for $x \oplus y \oplus z$ is:



- The k-map for X-NOR $(x \oplus y \oplus z)'$ is:

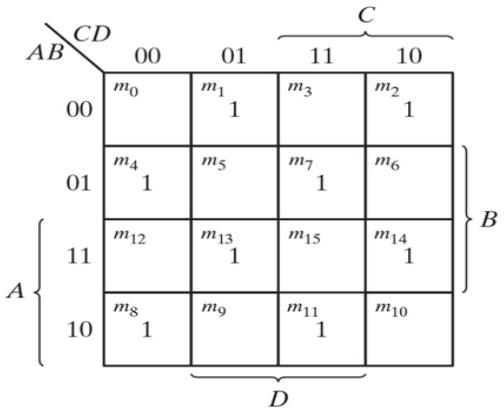


Lecture 5.1:
Using XOR: Odd Functions

- XOR of four variables $w, x, y,$ and z is:

$$w \oplus x \oplus y \oplus z = \sum (1, 2, 4, 7, 8, 11, 13, 14)$$

- The k-map for $w \oplus x \oplus y \oplus z$ is:



- This is true for any number of bits.

Lecture 5.1:
Using XOR: Error Detection

Concordia University
Department of Electrical & Computer Engineering

- Consider writing data bytes in a storage: a hard disk, CD or DVD.
- A byte consists of 8 bits so it take 256 values.
- One bit being flipped in a byte we get another byte and make an error without knowing.
- If we had kept the record of the number of ones being odd or even, we could detect that something had gone wrong. For example, for the byte 11010011, the parity is odd since $1 \oplus 1 \oplus 0 \oplus 1 \oplus 0 \oplus 0 \oplus 1 \oplus 1 = 1$
- We add this parity bit and store 110100111 making the parity even.
- When reading back the bits, we XOR the bits and if we get zero, we are OK, else, we detect an error.

Lecture 5.1:
Using XOR: Error Correction

Concordia University
Department of Electrical & Computer Engineering

- Example: (7, 4) Hamming Code:

w x y z	$p_1 = w \oplus x \oplus y$	$p_1 = x \oplus y \oplus z$	$p_1 = w \oplus y \oplus z$
0 0 0 0	0	0	0
0 0 0 1	0	1	1
0 0 1 0	1	1	1
0 0 1 1	1	0	0
0 1 0 0	1	1	0
0 1 0 1	1	0	1
0 1 1 0	0	0	1
0 1 1 1	0	1	0
1 0 0 0	1	0	1
1 0 0 1	1	1	0
1 0 1 0	0	1	0
1 0 1 1	0	0	1
1 1 0 0	0	1	1
1 1 0 1	0	0	0
1 1 1 0	1	0	0
1 1 1 1	1	1	1

Lecture 5.1:
Using XOR: Error Correction

- Hamming Code: