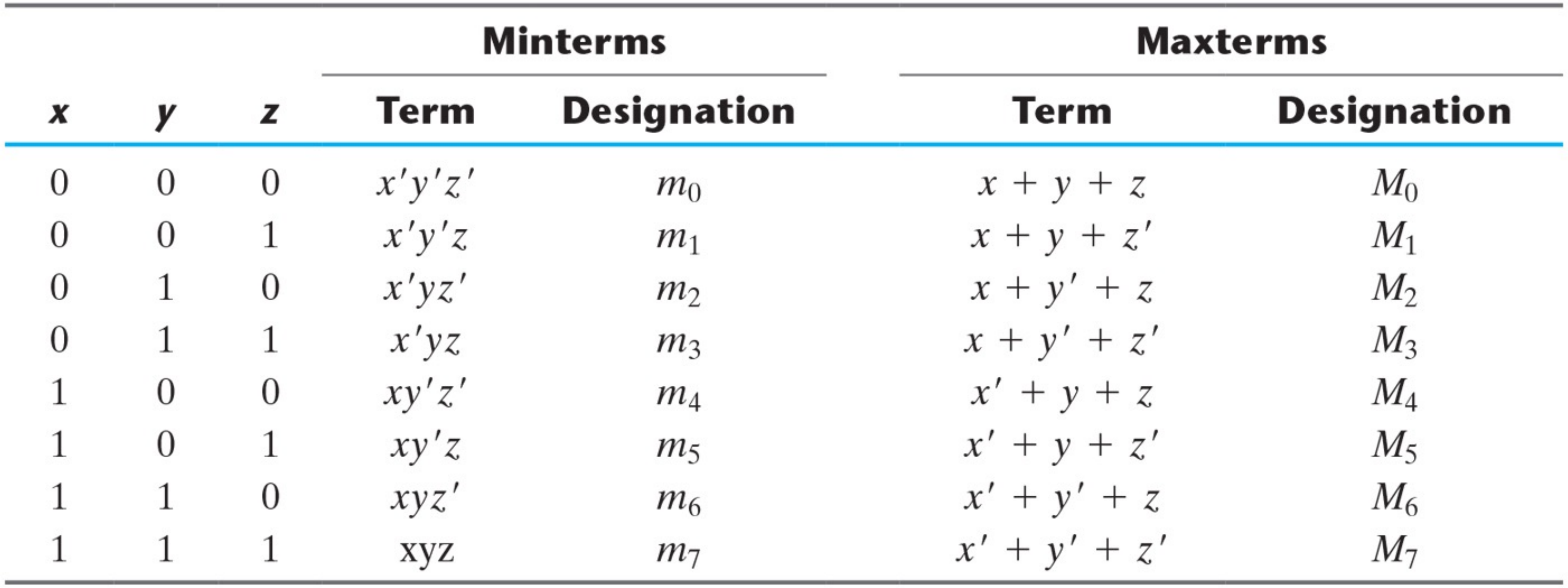
THE UNIVERSITY OF TEXAS AT DALLAS

CS4341 Digital Logic & Computer Design

Lecture Notes 6

Omar Hamdy
Assistant Professor
Department of Computer Science

Review: 3-variable Minterms & Maxterms



			Minterms		Maxterms	
x	y	z	Term	Designation	Term	Designation
0	0	0	$x'y'z'$	m_0	$x + y + z$	M_0
0	0	1	$x'y'z$	m_1	$x + y + z'$	M_1
0	1	0	$x'yz'$	m_2	$x + y' + z$	M_2
0	1	1	$x'yz$	m_3	$x + y' + z'$	M_3
1	0	0	$xy'z'$	m_4	$x' + y + z$	M_4
1	0	1	$xy'z$	m_5	$x' + y + z'$	M_5
1	1	0	xyz'	m_6	$x' + y' + z$	M_6
1	1	1	xyz	m_7	$x' + y' + z'$	M_7

Review: Representing a Boolean Function

- We can express any function by "ORing" the minterms corresponding to the ‘1’ outputs in the truth table.
- We can express any function by "ANDing" the maxterms corresponding to ‘0’ outputs in the truth table
- The same Boolean function can be expressed in two canonical ways: Sum-of-Minterms (SOM) and Product-of-Maxterms (POM).
- If a Boolean function has fewer ‘1’ outputs, then the SOM canonical form will contain fewer terms than POM. However, if it has fewer ‘0’ outputs then the POM form will have fewer terms than SOM.

Converting to SOM Form

- If a function is expressed in a non-standard SOM form, then it can be converted to a standard SOM using:
  - Truth Table
    - Describe the function using the truth table
    - Identify the minterms from the table and generate the SOM
  - Algebraic manipulations
    - For each missing literal x in a term, multiply the term by (x + x’)
    - Simplify the expression as needed
    - Re-group the terms in order

Example: Converting to SOM Form

- Express $F = A + B'C$ in the SOM canonical form
  - The function has three variables, and hence, each term consists of 3 literals
  - $A = A.(B + B') = AB + AB'$
    - $AB = AB(C + C') = ABC + ABC'$
    - $AB' = AB'(C + C') = AB'C + AB'C'$
    - $A = ABC + ABC' + AB'C + AB'C'$
  - $B'C = B'C(A + A') = B'CA + B'CA' = AB'C + A'B'C$
  - $F = ABC + ABC' + AB'C + AB'C' + AB'C + A'B'C$
  - Simplify: $F = ABC + ABC' + AB'C + AB'C' + A'B'C$
  - Order: $F = A'B'C + AB'C' + AB'C + ABC' + ABC$
  - $F = m_1 + m_4 + m_5 + m_6 + m_7 = \sum(1,4,5,6,7)$

Solution Using Truth Table

- Express `F = A + B'C` in the SOM canonical form

A	B	C	F	Minterm
0	0	0	0	
0	0	1	1	$m_1 = A'B'C$
0	1	0	0	
0	1	1	0	
1	0	0	1	$m_4 = AB'C'$
1	0	1	1	$m_5 = AB'C$
1	1	0	1	$m_6 = ABC'$
1	1	1	1	$m_7 = ABC$

- $F = m_1 + m_4 + m_5 + m_6 + m_7 = \sum(1,4,5,6,7)$

Converting to POM Form

- If a function is expressed in a non-standard POM form, then it can be converted to a standard POM using:
  - Truth Table
    - Describe the function using the truth table
    - Identify the maxterms from the table (rows with output 0) and generate the POM
  - Algebraic manipulations
    - Re-structure the expression in product terms such that each term is composed of sum of individual literals only.
    - This can be done using the algebraic equivalence $a + bc = (a + b)(a + c)$ how?
    - Complete missing literals in each term by adding AA', then use the same distribution equivalence above.

Example: Converting to POM Form

- Express $F = A + B'C$ in the POM canonical form
  - The function has three variables, and hence, each term consists of 3 literals
  - $F = (A + B')(A + C)$
  - $A + B' = A + B' + CC' = (A + B' + C)(A + B' + C')$
  - $A + C = A + BB' + C = (A + B + C)(A + B' + C)$
  - $F = (A + B' + C)(A + B' + C')(A + B + C)(A + B' + C)$
  - Simplify: $F = (A + B' + C)(A + B' + C')(A + B + C)$
  - Order: $F = (A + B + C)(A + B' + C)(A + B' + C')$
  - $F = M_0 \, . \, M_2 \, . \, M_3 = \prod(0,2,3)$

Solution Using Truth Table

- Express F = A + B′C in the POM canonical form

A	B	C	F	Minterm
0	0	0	**0**	$M_0 = (A+B+C)$
0	0	1	1	
0	1	0	**0**	$M_2 = (A+B'+C)$
0	1	1	**0**	$M_3 = (A+B'+C')$
1	0	0	1	
1	0	1	1	
1	1	0	1	
1	1	1	1	

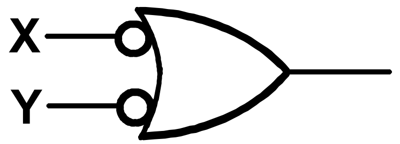
- $F = M_0 \,.\, M_2 \,.\, M_3 = \prod(0,2,3)$

Function Complement

- The complement of a function expressed as a sum of minterms is constructed by selecting the minterms missing in the sum-of-minterms canonical form
- Alternatively, the complement of a function expressed by a Sum of Minterms form is simply the Product of Maxterms with the same indices
- Example: Given $F(x, y, z) = \sum(1,3,5,7)$
- $F'(x, y, z) = \sum(0,2,4,6)$
- $F'(x, y, z) = \prod(1,3,5,7)$

NAND Gate

- The NAND gate is implemented efficiently in CMOS technology in terms of chip area and speed.
- NAND represents NOT AND. The small “bubble” circle represents the invert function.
- Applying DeMorgan's Law, a NAND gate is also Invert-OR

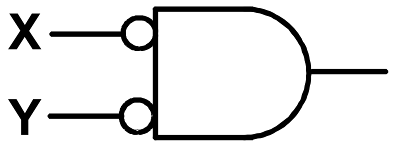


$\overline{X} + \overline{Y} = \overline{X \cdot Y} = \text{NAND}$

- Pushing bubble forward or backward flips the gate and inverts inputs.
- NAND gate can implement all basic gates (universality). Example, an OR gate is implemented using a NAND gate and inverted inputs

NOR Gate

- NOR represents NOT OR. Similarly, Applying DeMorgan's Law, a NOR gate is also Invert-AND (bubble push backwards)



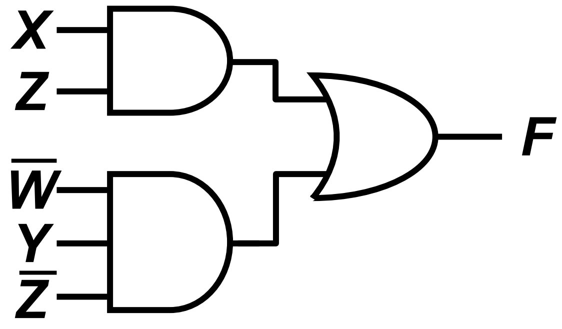
$$\overline{X} \cdot \overline{Y} = \overline{X + Y} = \text{NOR}$$

- NOR gate can implement all basic gates (universality). Example, an AND gate is implemented using a NOR gate and inverted inputs
- Unlike AND and OR, the NAND and NOR operations are NOT associative
  - (X NAND Y) NAND Z ≠ X NAND (Y NAND Z)
  - (X NOR Y) NOR Z ≠ X NOR (Y NOR Z)

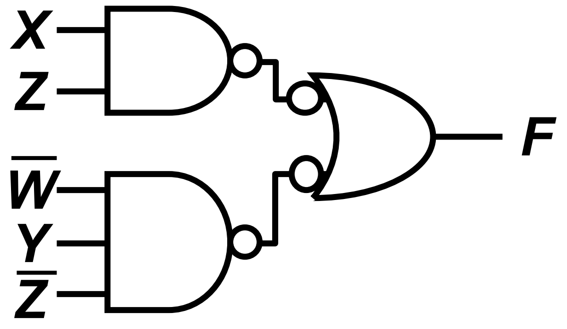
NAND-NAND Implementation

- ➢ Consider the Following SOP Expression: $F = XZ + W'YZ'$
- ➢ A 2-level AND-OR circuit can be converted easily to a NAND-NAND implementation

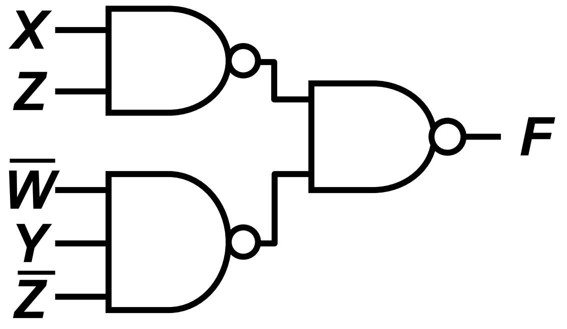
Step 1: Build the circuit as is from the SOP expression



Step 2: Add redundant bubbles on the same line as needed



Step 3: Push bubbles forward as needed to get NAND only gates



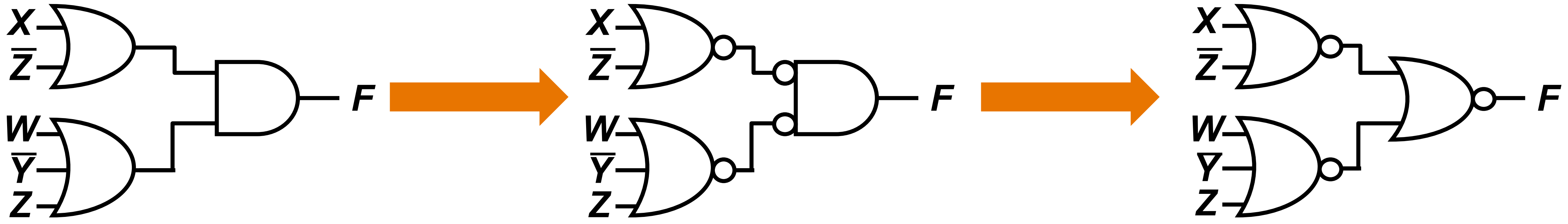
NOR-NOR Implementation

- Consider the Following POS Expression: `F = (X+Z')(W+Y'+Z)`
- A 2-level OR-AND circuit can be converted easily to a NOR-NOR implementation

Step 1: Build the circuit as is from the POS expression

Step 2: Add redundant bubbles on the same line as needed

Step 3: Push bubbles forward as needed to get NOR only gates



Schematic Design

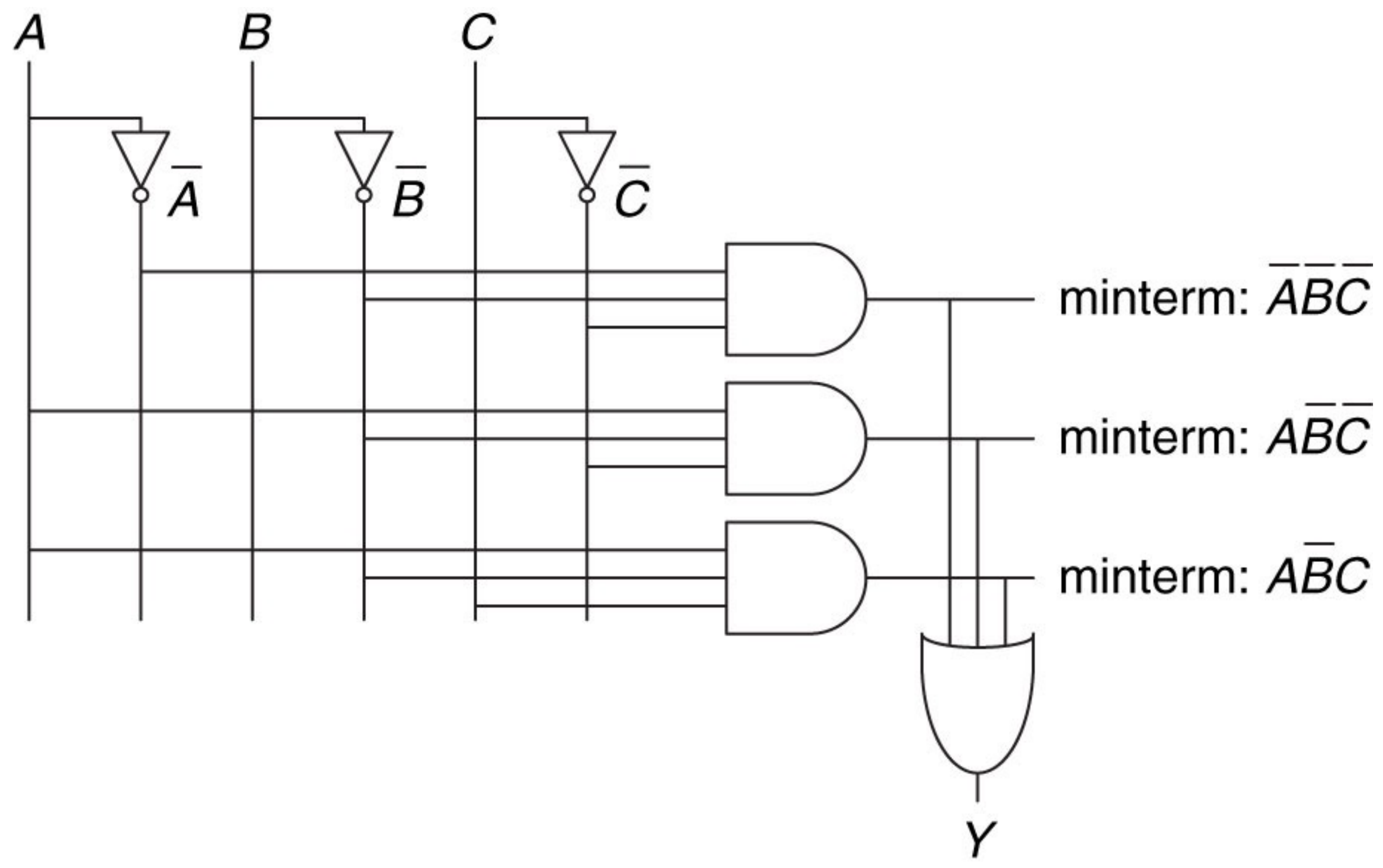
- It is crucial to use clear guidelines when drawing digital circuits from the logic function or truth table.
  - Inputs are on the left (or top) side of a schematic
  - Outputs are on the right (or bottom) side of a schematic
  - Whenever possible, gates should flow from left to right (or top to bottom)
  - Straight wires are better to use than wires with multiple corners
  - Wires always connect at a T junction
  - A dot where wires cross indicates a connection between the wires; otherwise, they do not connect

SOP Schematic Design Steps

- SOP diagrams can be systematically drawn following the Programmable Logic Array (PLA) method:
  - Draw columns for all inputs, with inverters in adjacent columns (if needed)
  - Draw rows of AND gates for each minterm
  - Draw an OR gate to connect each output of the AND gates

SOP Schematic Design Example

Draw the logic function Y = A’B’C’ + AB’C’ + AB’C


A
B
C
$\overline{A}$
$\overline{B}$
$\overline{C}$
minterm: $\overline{A}\,\overline{B}\,\overline{C}$
minterm: $A\overline{B}\,\overline{C}$
minterm: $A\overline{B}C$
Y

Example: Multiple Output Circuits

- In many cases, there are more than one output required to carry out the desired function behavior.
- Example, 4-input priority circuit
  - The circuit has 4-outputs that signals which input should be given the priority when more than one input are requesting it.
  - The circuit allows only one output signal to be high at any time.
  - Assume inputs A_3 (highest priority), A_2, A_1, A_0(lowest priority)
  - Assume outputs Y_3 indicating A_3 gets the priority, Y_2 indicating A_2 gets the priority, and so on.

4-Input Priority Circuit Design Approach 1

- One approach is to build the truth table, determine the minterms for each output, express each output in SOM form, then draw the circuit.

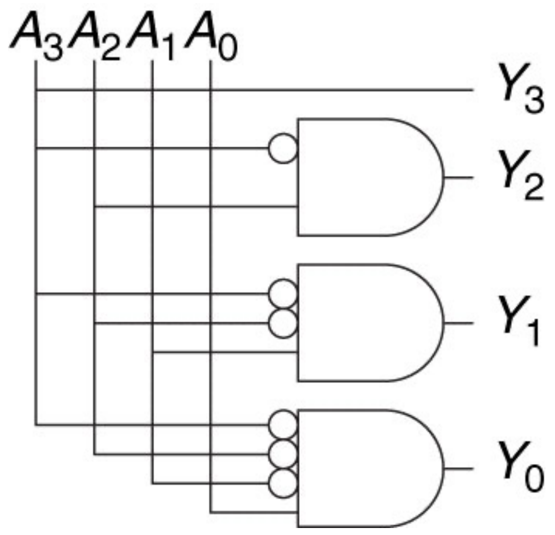
A_3	A_2	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	0
0	1	0	0	0	1	0	0
0	1	0	1	0	1	0	0
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	0
1	0	0	0	1	0	0	0
1	0	0	1	1	0	0	0
1	0	1	0	1	0	0	0
1	0	1	1	1	0	0	0
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	0
1	1	1	0	1	0	0	0
1	1	1	1	1	0	0	0

- $Y_0 = m_1 = A'_3A'_2A'_1A_0$.
- $Y_1 = m_2 + m_3 = A'_3A'_2A_1A'_0 + A'_3A'_2A_1A_0$
  - Simplify = $A'_3A'_2A_1$ ... why?
- $Y_2 = m_4 + m_5 + m_6 + m_7 = A'_3A_2A'_1A'_0 + A'_3A_2A'_1A_0 + A'_3A_2A_1A'_0 + A'_3A_2A_1A_0$
  - Simplify = A'_3A_2 ... why?
- $Y_3 = m_8 + m_9 + m_{10} + m_{11} + m_{12} + m_{13} + m_{14} + m_{15} = A_3A'_2A'_1A'_0 + A_3A'_2A'_1A_0 + A_3A'_2A_1A'_0 + A'_3A'_2A_1A_0 + ...$
  - Simplify = A_3 ... why?
- Draw the circuit using the PLA approach

4-Input Priority Circuit Design Approach 2

- Careful review of the circuit function, we notice:
  - Y_3 output is 1 when A_3 is 1 and does not care about the values of any other input.
  - Therefore, $Y_3 = A_3$
  - Y_2 output is 1 when 1) A_3 is 0, 2) A_2 is 1 and does not care about the input values of A_1 or A_0
  - Y_1 output is 1 when 1) A_3 is 0, 2) A_2 is 0, 3) A_1 is 1 and does not care about the input value of A_0
  - Y_0 output is 1 when 1) A_3 is 0, 2) A_2 is 0, 3) A_1 is 0 and 4) A_0 is 1
- The concept of "Don't Care" can be represented with symbol X in the truth table, and can simplify the design process

A_3	A_2	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	X	0	0	1	0
0	1	X	X	0	1	0	0
1	X	X	X	1	0	0	0

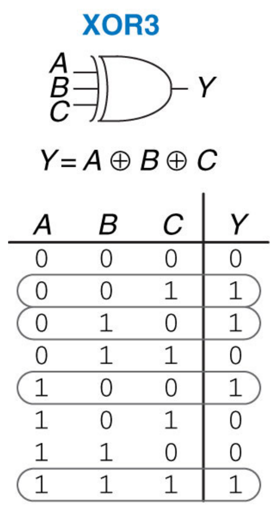


Multilevel Combinational Logic Design

- In many cases, multi-level circuit designs can reduce the number of true gates required.
- Example: Implementation of 3-input XOR function (remember, there is no actual 3-input XOR gate)
- Truth table out is 1 if the inputs have odd number of 1s

$A \oplus B \oplus C = m_1 + m_2 + m_4 + m_7 = A'B'C + AB'C' + ABC$ (8 gates)

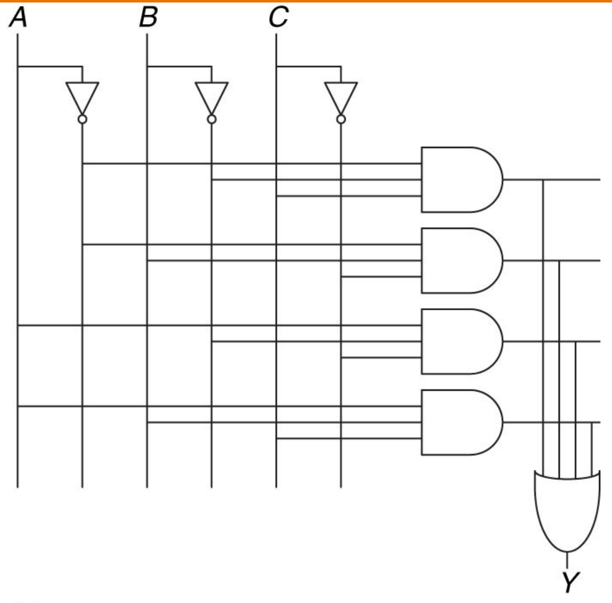
$A \oplus B \oplus C = (A \oplus B) \oplus C$ (2 gates)



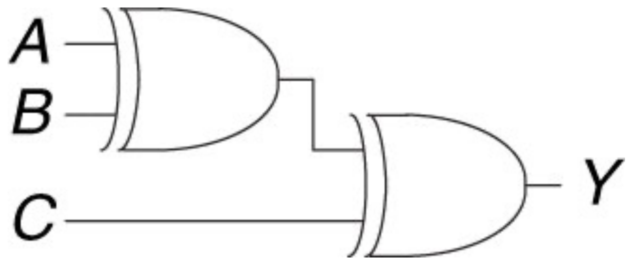
$Y = A \oplus B \oplus C$

A	B	C	Y
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

(a)



(b)



To Do List

- ➢Review lecture notes, and try the examples yourself
- ➢Study chapter 2 until 2.7
- ➢Start working on homework 1