

ENEE244-020x

Digital Logic Design

Lecture 6

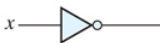
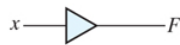
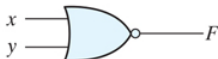
Announcements

- HW 2 up on course webpage, due on Wednesday, Sept. 23 @11:59pm
Extended.

Agenda

- Last time:
 - Boolean Functions and Complements (2.5)
 - Simplification of Boolean Formulas via Algebraic Manipulation (2.5)
 - Canonical and Standard Forms (2.6)
 - Digital Logic Gates (2.8)
- This time:
 - Incomplete Boolean Functions and Don't Care Conditions
 - Universal Gates (3.9.3)
 - NAND/NOR Gate Realizations (3.9.4-3.9.6)
 - Some examples of Synthesis Procedure
 - The simplification problem (4.1)
 - Prime Implicants (4.2)
 - Prime Implicates (4.3)

Digital Logic Gates

Name	Graphic symbol	Algebraic function	Truth table															
AND		$F = x \cdot y$	<table><tr><th>x</th><th>y</th><th>F</th></tr><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></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><tr><th>x</th><th>y</th><th>F</th></tr><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></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><tr><th>x</th><th>F</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	x	F	0	1	1	0									
x	F																	
0	1																	
1	0																	
Buffer		$F = x$	<table><tr><th>x</th><th>F</th></tr><tr><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td></tr></table>	x	F	0	0	1	1									
x	F																	
0	0																	
1	1																	
NAND		$F = (xy)'$	<table><tr><th>x</th><th>y</th><th>F</th></tr><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></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><tr><th>x</th><th>y</th><th>F</th></tr><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></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><tr><th>x</th><th>y</th><th>F</th></tr><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></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><tr><th>x</th><th>y</th><th>F</th></tr><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></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																

Some Terminology

- **System specification**: A description of the function of a system and of other characteristics required for its use.
 - A function (table, algebraic) on a finite set of inputs.
- **Double-rail logic***: both variables and their complements are considered as primary inputs. x, x'
- **Single-rail logic**: only variables are considered as primary inputs. Need inverters for their complements.

Incomplete Boolean Functions and Don't Care Conditions

Incomplete Boolean Functions and Don't Care Conditions

- n -variable incomplete Boolean function is represented by a truth table with $n+1$ columns and 2^n rows.
- For those combinations of values in which a functional value is not to be specified, a symbol, -, is entered.
- The complement of an incomplete Boolean function is also an incomplete Boolean function having the same unspecified rows of the truth table.

f	f'
1	0
0	1
-	-

Describing Incomplete Boolean Functions

X	Y	Z	F
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	--
1	0	0	0
1	0	1	--
1	1	0	0
1	1	1	1

Minterm canonical formula:

$$f(x, y, z) = \sum m(0,1,7) + dc(3,5)$$

Describing Incomplete Boolean Functions

X	Y	Z	F
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	--
1	0	0	0
1	0	1	--
1	1	0	0
1	1	1	1

Maxterm canonical formula:

$$f(x, y, z) = \Pi M(2,4,6) + dc(3,5)$$

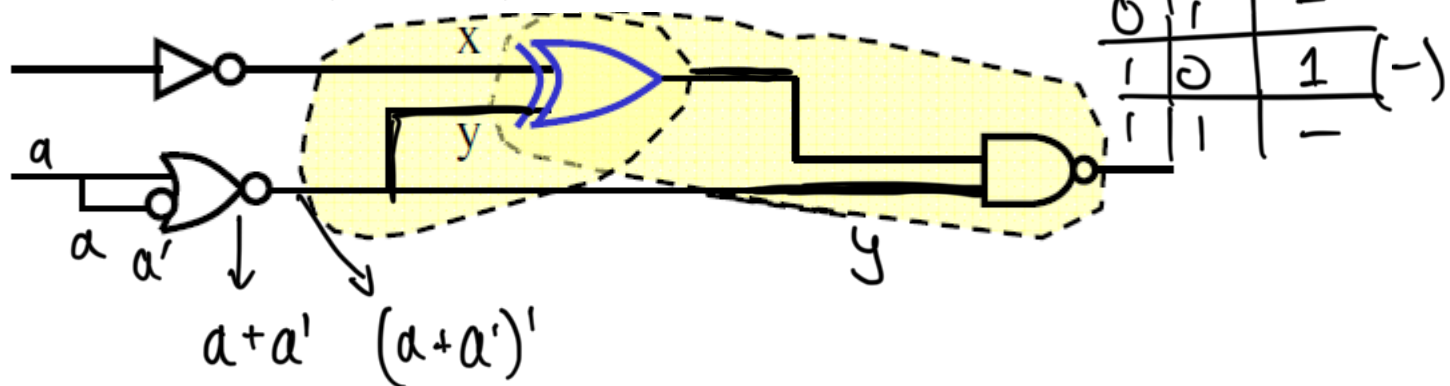
Describing Incomplete Boolean Functions

- Manipulating Boolean equations derived from incomplete Boolean functions is a very difficult task.
- In the next chapter, there are procedures for obtaining minimal expressions that can handle the don't care conditions.
- Can leverage don't care conditions to get simplified expressions for functions (smaller gate networks).

More on Don't Care Conditions

- Satisfiability don't cares of a subcircuit/subsystem consist of all input patterns that will never occur.
- Observability don't cares of a subcircuit/subsystem are the input patterns that represent situations when an output is not observed.

Example: $\{xy, \bar{x}y\}$ $\{x\bar{y}, \bar{x}\bar{y}\}$



Universal Gates

$$f: \{0,1\}^n \rightarrow \{0,1\}$$

x_1	x_2	$\dots$	x_n	$f(x_1, \dots, x_n)$
0	0	-	0	1
0	0	.	1	0
		,	.	
1	1	-	1	0

- A gate or set of gates is called universal if it can implement all Boolean functions.
- Standard universal gates:
 - AND, OR, NOT
 - Proof? Canonical forms give a constructive proof
- Theorem:
 - A set of gates $\{G_1, \dots, G_n\}$ is universal if it can implement AND, OR, NOT.

NAND is Universal

- Recall $NAND(x, y) = (xy)' = x' + y'$
- $NOT(x) =$

$\xrightarrow{\text{Idempotent}}$
 $x' = x' + x' = NAND(x, x) = (x \cdot x)'$

$\xleftarrow{\text{De Morgan's}}$
- $AND(x, y) =$
 $xy = (xy)'' = NOT((xy)')$

$\downarrow$
 Involution

 $= NOT(NAND(x, y))$
 $= NAND(NAND(x, y), NAND(x, y))$
- $OR(x, y) =$

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

 $x + y = (x + y)'' = (x'y')'$
 $= NAND(NOT(x), NOT(y))$
 $= NAND(NAND(x, x), NAND(y, y))$

NOR is Universal

- Recall $NOR = (x + y)' = x'y'$
- $NOT(x) =$
 $x' = x'x' = NOR(x, x)$
- $AND(x) =$
 $xy = (xy)'' = (x' + y')'$
 $= NOR(NOT(x), NOT(y))$
 $= NOR(NOR(x, x), NOR(y, y))$
- $OR(x) =$
 $x + y = (x + y)'' = NOT((x + y)')$
 $= NOT(NOR(x, y))$
 $= NOR(NOR(x, y), NOR(x, y))$

NAND-Gate Realizations

- Naïve approach: build network out of AND/OR/NOT gates, use the universal property above to replace each one with several NAND gates.
- A better approach: manipulate Boolean expression into the form $\text{NAND}(A, B, \dots, C)$

NAND-Gate Realizations

$$(x+y)'' = (\underbrace{(x+y)'}_//)'$$

- Basic idea:
 - Use the property: $x + y = (x'y')'$
 - i.e. $x + y = \text{NAND}(x', y')$
- Keep doing this recursively.

NAND-Gate Realizations

- Example:

$$\begin{aligned} f(w, x, y, z) &= w'z + wz'(x + y') \\ &= ((w'z)' \cdot (wz'(x + y'))')' \end{aligned}$$

- $(w'z)'$ --already in correct form, equivalent to $NAND(w', z)$

- $(wz'(x + y'))' = NAND(w, z', (x + y'))$

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

- $f(w, x, y, z) =$
 $NAND(NAND(w', z), NAND(w, z', (NAND(x', y))))$

NAND-Gate Realizations

- Only works if highest-order operation is an or-operation.
- **Highest-order operation** is the last operation that is performed when the expression is evaluated.
- What to do otherwise? Negate and repeat the procedure for f' . Then note that
$$f(x_1, \dots, x_n) = \text{NAND}(1, f'(x_1, \dots, x_n))$$

NAND-Gate Realization

- Example:

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

$$f(x, y, z) = \text{NAND}(1, f'(x, y, z))$$

$$- f'(x, y, z) = ((x + y)(y' + z))' = \text{NAND}(x + y, y' + z)$$

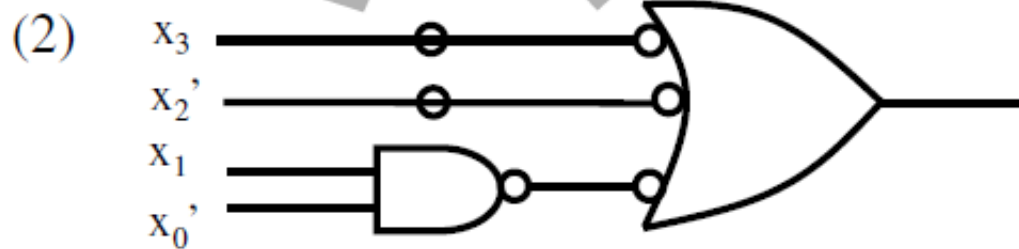
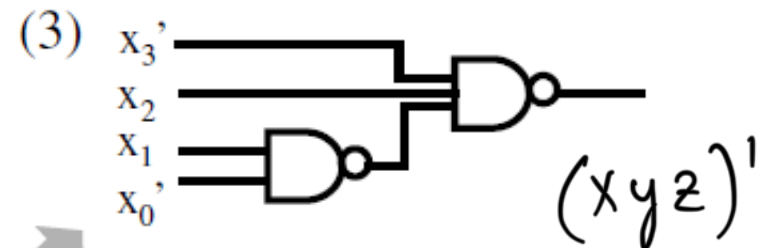
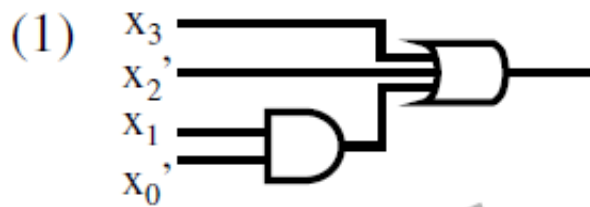
$$- x + y = (x'y)' = \text{NAND}(x', y')$$

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

$$- f(x, y, z) = \text{NAND}(1, \text{NAND}(\text{NAND}(x', y'), \text{NAND}(y, z')))$$

NAND Implementation Example

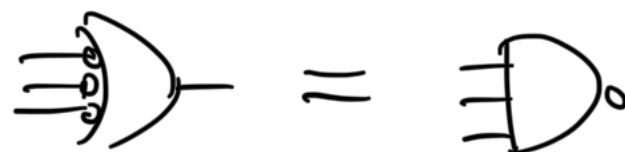
$$f(x_0, x_1, x_2, x_3) = x_3 + x'_2 + x_1 x'_0$$



$(xyz)'$
|| De Morgan's
 $x' + y' + z'$

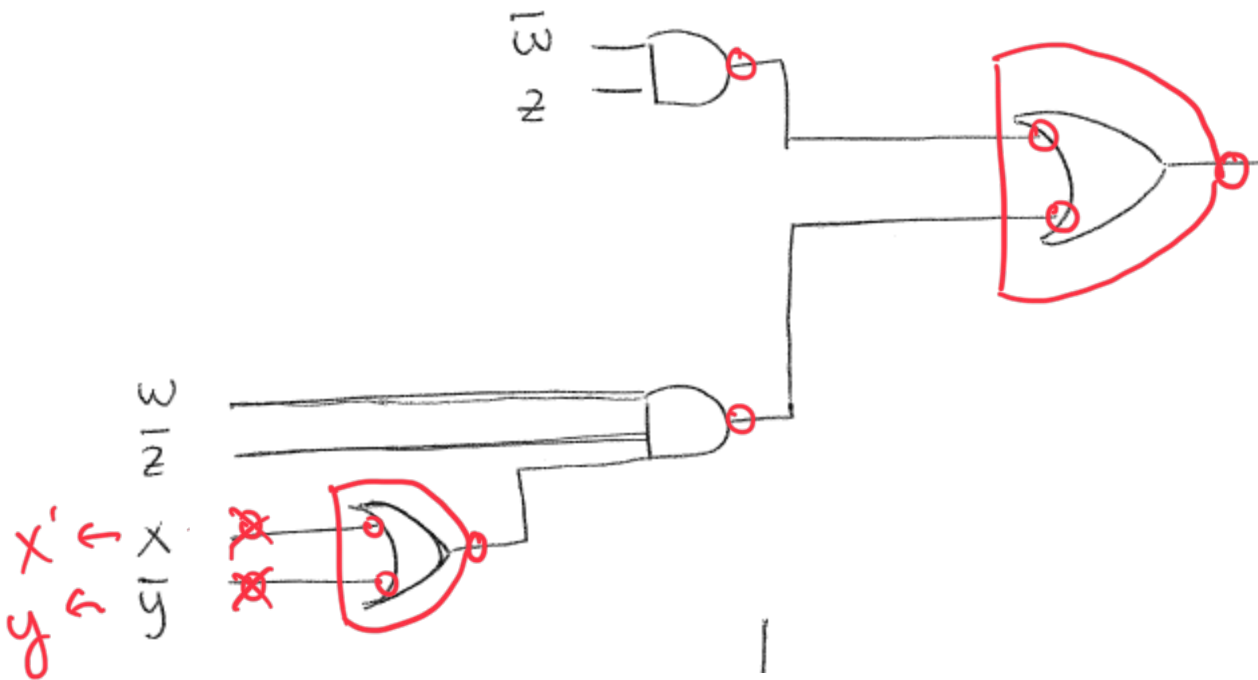
General NAND Implementation

- Draw the {AND,OR,NOT} implementation
- $\text{AND} \Rightarrow \text{AND-NOT-NOT} \Rightarrow \text{NAND-NOT}$
- $\text{OR} \Rightarrow \text{NOT-NOT-OR} \Rightarrow \text{NOT-NAND}$
- Delete “NOT-NOT”
- $\text{NOT} \Rightarrow \text{NAND}$

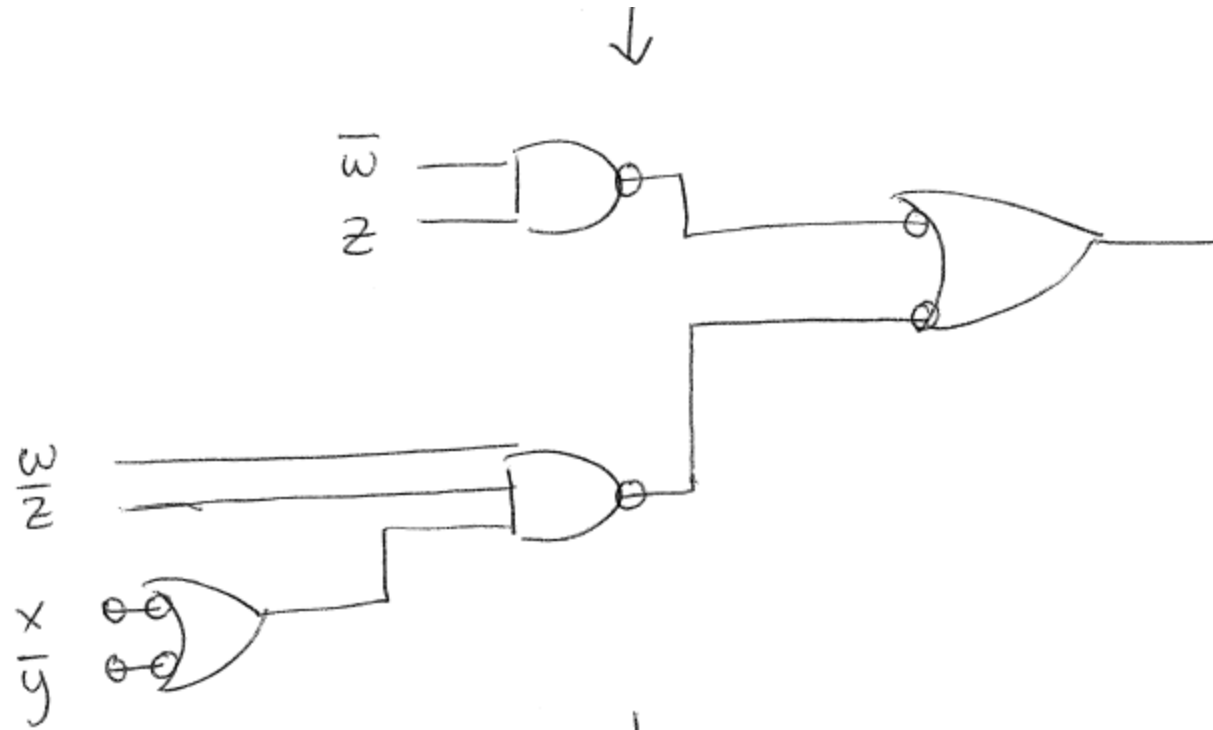


NAND-Gate Realizations

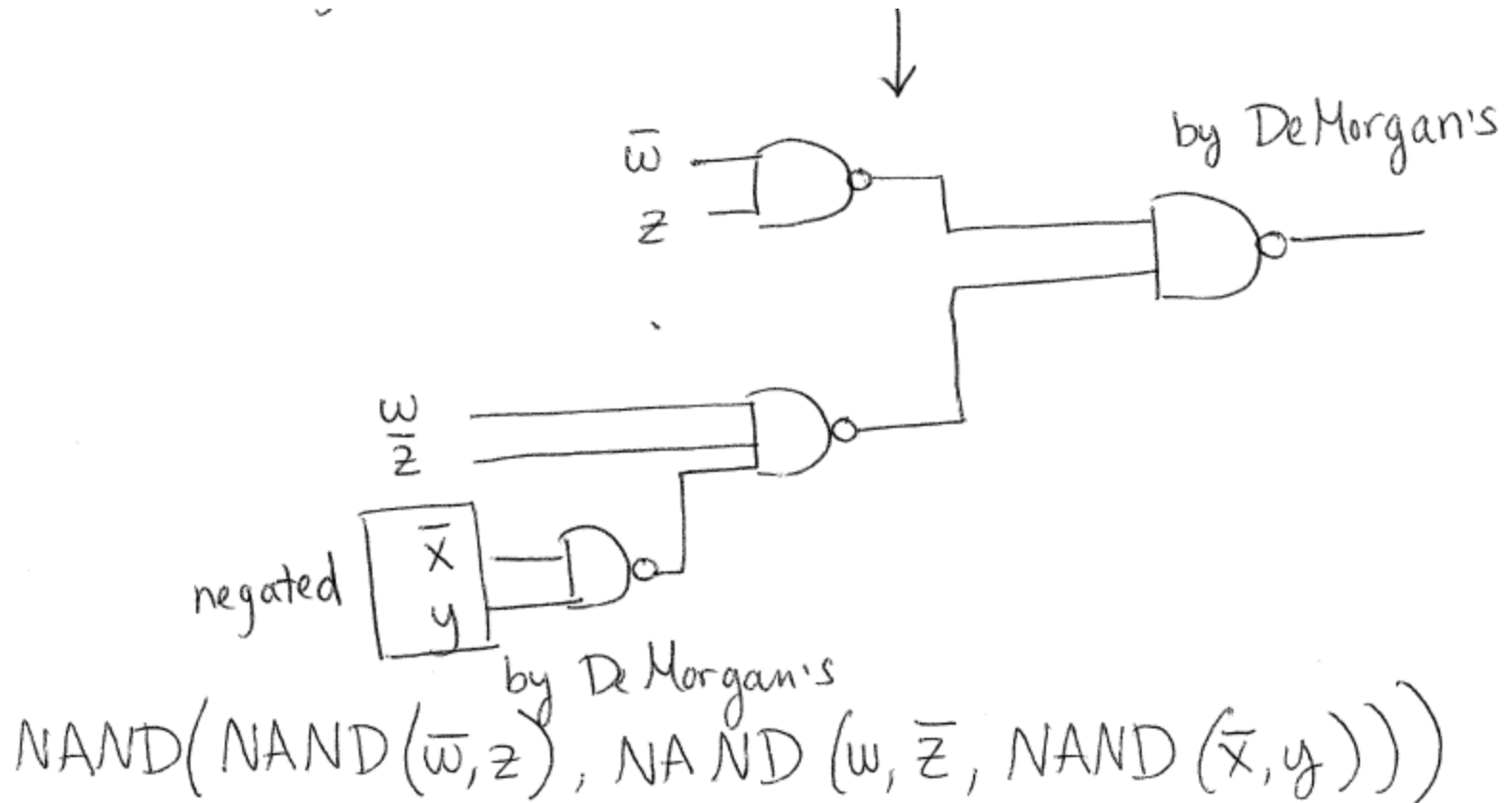
① $f(w, x, y, z) = \bar{w}z + w\bar{z}(x + \bar{y})$



NAND-Gate Realizations

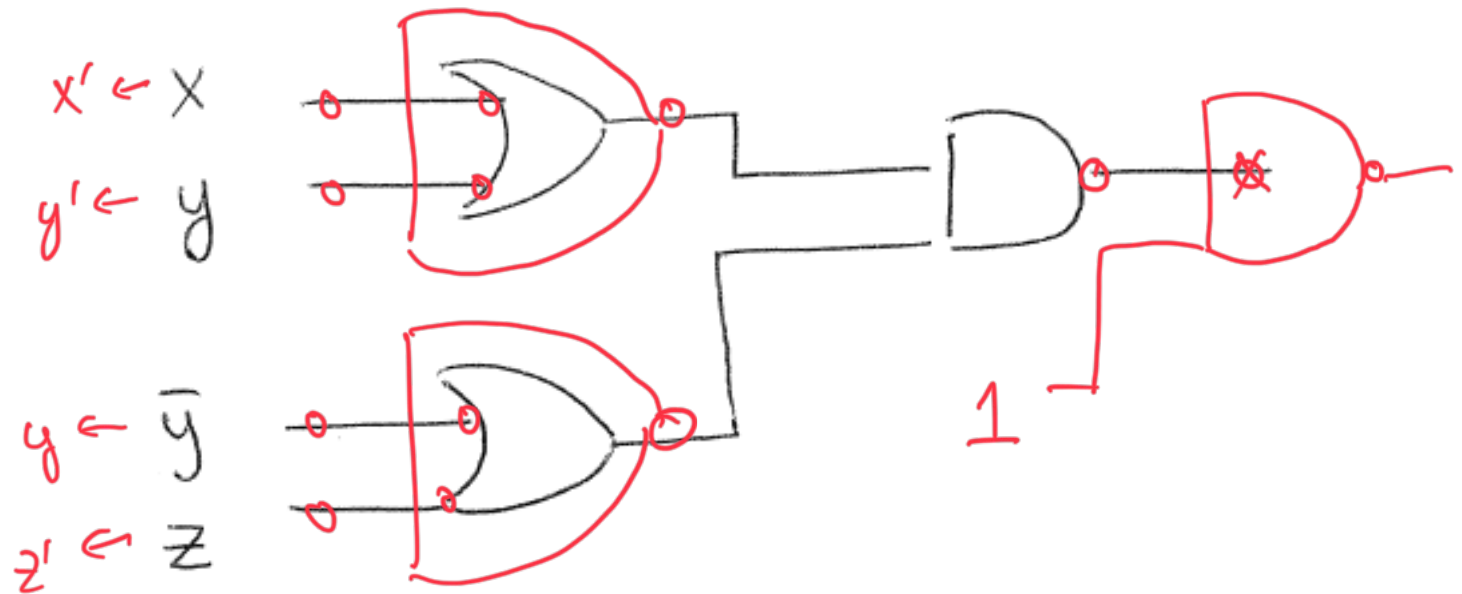


NAND-Gate Realizations

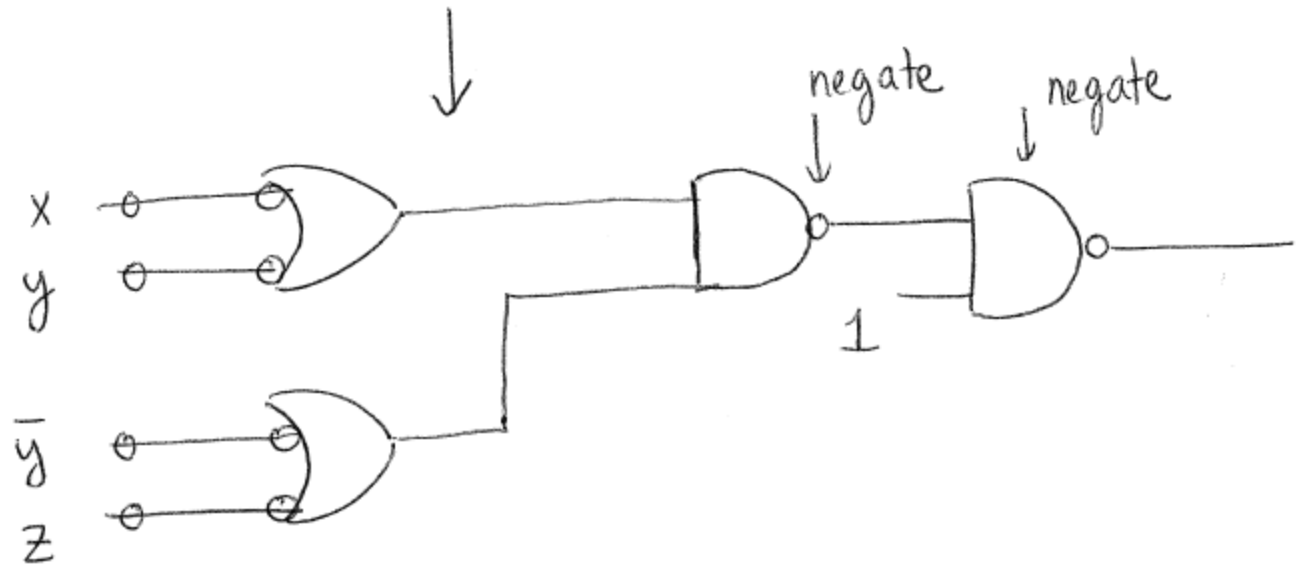


NAND-Gate Realizations

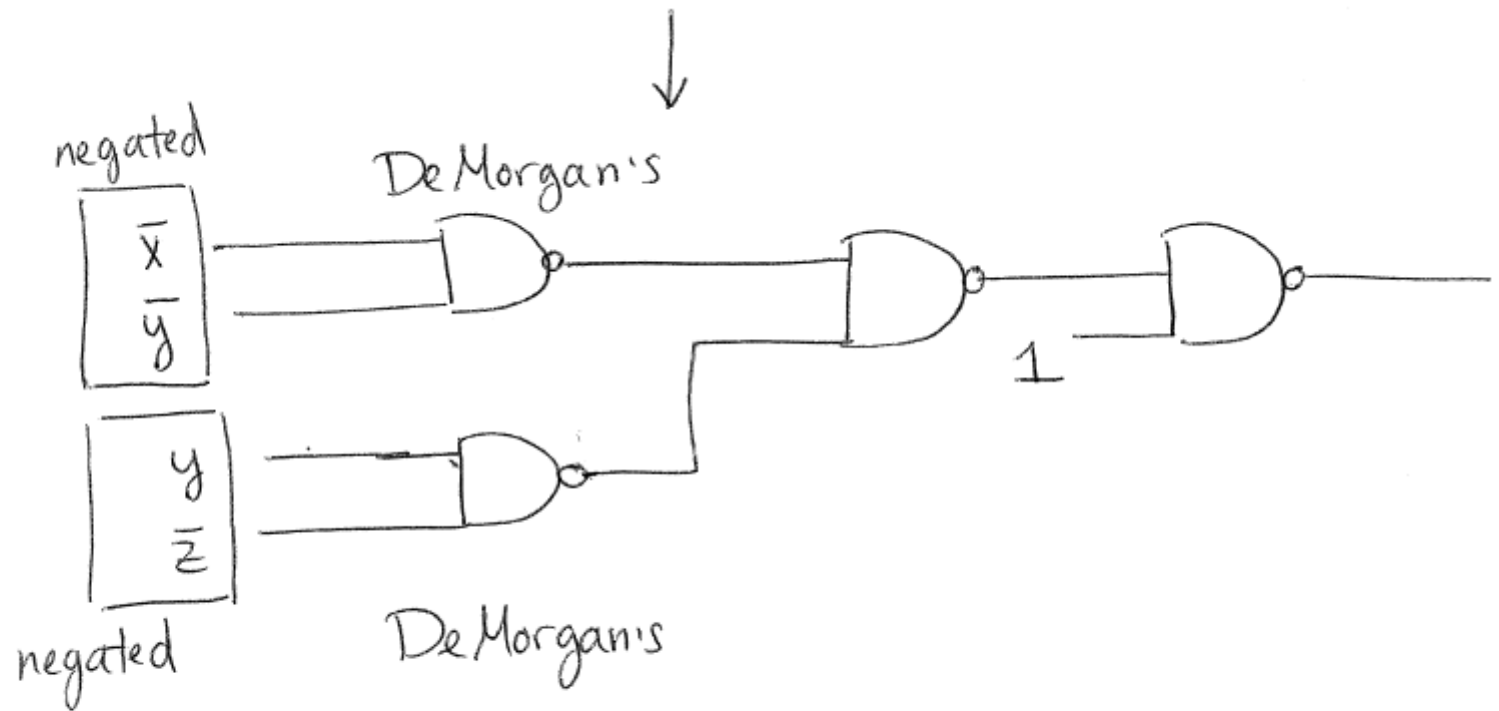
② $f(x, y, z) = (x + y)(\bar{y} + z)$



NAND-Gate Realizations



NAND-Gate Realizations



$$\text{NAND}(1, \text{NAND}(\text{NAND}(\bar{x}, \bar{y}), \text{NAND}(y, \bar{z})))$$

NOR-Gate Realizations

- Essentially the same procedure

In-Class Exercise

Synthesis Procedure Examples

Synthesis Procedure

- High-level description: A function with finite domain and range.
- Binary-level: All input-output variables are binary.

Example 1

A system takes one decimal digit and outputs 0 on even number and 1 on odd number.

- High-level:
 - Input: $x \in \{0,1,2,3,4,5,6,7,8,9\}$
 - Output: $z \in \{0,1\}$
 - Function:
$$z = F(x) = \begin{cases} 0 & x=0,2,4,6,8 \\ 1 & x=1,3,5,7,9 \end{cases}$$

x	0	1	2	3	4	5	6	7	8	9
z	0	1	0	1	0	1	0	1	0	1

Example 1 (cont'd)

- Binary-level (using BCD)

- Input variables: x_3, x_2, x_1, x_0

- Output variables: z

- Functions:

$$z(x_3, x_2, x_1, x_0) = x_0$$



x	$x_3x_2x_1x_0$	z
0	0 0 0 0	0
1	0 0 0 1	1
2	0 0 1 0	0
3	0 0 1 1	1
4	0 1 0 0	0
5	0 1 0 1	1
6	0 1 1 0	0
7	0 1 1 1	1
8	1 0 0 0	0
9	1 0 0 1	1

Example 2

A system takes one decimal digit and outputs its 9's complement.

- High-level:
 - Input: $x \in \{0,1,2,3,4,5,6,7,8,9\}$
 - Output: $z \in \{0,1,2,3,4,5,6,7,8,9\}$
 - Function: $z = F(x) = 9 - x$

x	0	1	2	3	4	5	6	7	8	9
z	9	8	7	6	5	4	3	2	1	0

Example 2 (cont'd)

- Binary-level (using BCD)

- Input variables: x_0, x_1, x_2, x_3
- Output variables: z_0, z_1, z_2, z_3
- Functions: (sum of minterms)

$$z_3(x_3, x_2, x_1, x_0) = \sum m(0, 1)$$

$$z_2(x_3, x_2, x_1, x_0) = \sum m(2, 3, 4, 5)$$

$$z_1(x_3, x_2, x_1, x_0) = \sum m(2, 3, 6, 7)$$

$$z_0(x_3, x_2, x_1, x_0) = \sum m(0, 2, 4, 6, 8)$$



X	$x_3x_2x_1x_0$	$z_3z_2z_1z_0$	Z
0	0 0 0 0	1 0 0 1	9
1	0 0 0 1	1 0 0 0	8
2	0 0 1 0	0 1 1 1	7
3	0 0 1 1	0 1 1 0	6
4	0 1 0 0	0 1 0 1	5
5	0 1 0 1	0 1 0 0	4
6	0 1 1 0	0 0 1 1	3
7	0 1 1 1	0 0 1 0	2
8	1 0 0 0	0 0 0 1	1
9	1 0 0 1	0 0 0 0	0

Simplification of Boolean Expressions

Formulation of the Simplification Problem

- What evaluation factors for a logic network should be considered?
 - Cost (of components, design, construction, maintenance)
 - Reliability (highly reliable components, redundancy)
 - Time it takes for network to respond to changes at its inputs.

Minimal Response Time

- Achieved by minimizing the number of levels of logic that a signal must pass through.
- Always possible to construct any logic network with at most two levels under the double-rail logic assumption.
 - Why?

Minimal Component Cost

- Assume this is the only other factor influencing the merit evaluation of a logic network.
- In general, there are many two-level realizations.
- Determine the normal formula with minimal component cost.
- **Number of gates** is one greater than the number of terms with more than one literal in the expression.
- **Number of gate inputs** is equal to the number of literals in the expression plus the number of terms containing more than one literal.
- Using these criteria can obtain a measure of a Boolean expression's complexity called the **cost** of the expression.

The Simplification Problem

- The determination of Boolean expressions that satisfy some criterion of minimality is the simplification or minimization problem.
- We will assume cost is determined by number of gate inputs.

Fundamental Terms

- A product or sum of literals in which no variable appears more than once.
- Can obtain a fundamental term by noting:

$$x + \bar{x} = 1$$

$$x \cdot \bar{x} = 0$$

$$x + x = x$$

$$x \cdot x = x$$

Prime Implicants

- f_1 implies f_2 ($f_1 \rightarrow f_2$)
 - There is no assignment of values to the n variables that makes f_1 equal to 1 and f_2 equal to 0.
 - Whenever f_1 equals 1, then f_2 must also equal 1.
 - Whenever f_2 equals 0, then f_1 must also equal 0.
- Concept can be applied to terms and formulas.

Examples

- $f_1(x, y, z) = xy + yz,$
 $f_2(x, y, z) = xy + yz + x'z$
- $f_3(x, y, z) = (x + y)(y + z)(x' + z),$
 $f_4(x + y)(y + z)$

Examples

- Case of Disjunctive Normal Formula
 - Sum-of-products form
 - Each of the product terms implies the function being described by the formula
 - Whenever product term has value 1, function must also have value 1.
- Case of Conjunctive Normal Formula
 - Product-of-sums form
 - Each sum term is implied by the function
 - Whenever the sum term has value 0, the function must also have value 0.

Subsumes

- A term t_1 is said to **subsume** a term t_2 iff all the literals of the term t_2 are also literals of the term t_1 .
- Example: $xy'z', xz'$
 $x + y' + z', x + z'$
- If a product term t_1 subsumes a product term t_2 , then t_1 implies t_2 .
 - Why?
- If a sum term t_3 subsumes a sum term t_4 , then t_4 implies t_1 .
 - Why?

Subsumes

- Theorem:
 - If one term subsumes another in an expression, then the subsuming term can always be deleted from the expression without changing the function being described.
- CNF: $(x + y)(x + y + z)$
- DNF: $xy + xyz$
- Which theorem of Boolean algebra implies this?