Unit 8

Minterm and Canonical Sums

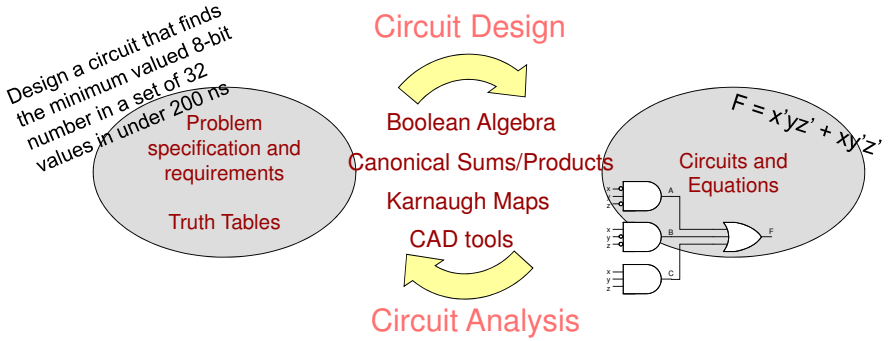
2- and 3-Variable Boolean Algebra Theorems

DeMorgan's Theorem

Simplification using Boolean Algebra

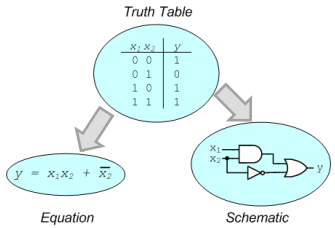
Circuit Design and Analysis

- There are two basic tasks as a digital design engineer…
  - Circuit Design/Synthesis: Take a set of requirements or functional descriptions and arrive at a logic circuit
  - Circuit Analysis: Given a logic circuit, find or verify the logic function it implements



Outcomes

- The goal of this unit is to teach you how you can construct a digital circuit to implement ANY logic function
  - Example: Given a truth table, build a circuit from ANDs and ORs that will implement the function specified by the truth table



SYNTHESIZING LOGIC FUNCTIONS

The Problem

- Given a logic function, how do we arrive at a circuit to implement this combinational function?
- Rather than trying to solve the problem all at once we will do it in two steps:
  - Use gates to convert raw inputs (e.g. x,y,z) to intermediate signals;
  - Convert the intermediate signals to produce the output (e.g. P)

Primes between 0-7

X	Y	Z	P
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

1's Count of Inputs

I3	I2	I1	C1	C0
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Using Decoders to Implement Functions

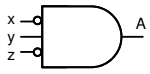
- Given an any logic function, it can be implemented with the superposition of decoders/checkers

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

Using Decoders to Implement Functions

- Given an any logic function, it can be implemented with the superposition of decoders

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

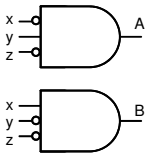


X	Y	Z	A
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Using Decoders to Implement Functions

- Given an any logic function, it can be implemented with the superposition of decoders

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

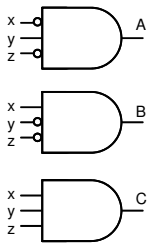


X	Y	Z	A	B
0	0	0	0	0
0	0	1	0	0
0	1	0	1	0
0	1	1	0	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	0	0

Using Decoders to Implement Functions

- Given an any logic function, it can be implemented with the superposition of decoders

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

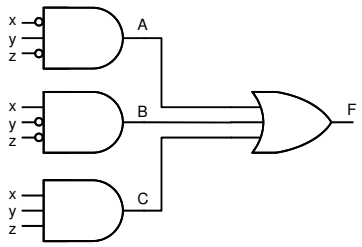


X	Y	Z	A	B	C
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	1	0	0
0	1	1	0	0	0
1	0	0	0	1	0
1	0	1	0	0	0
1	1	0	0	0	0
1	1	1	0	0	1

Using Decoders to Implement Functions

- Given an any logic function, it can be implemented with the superposition of decoders

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

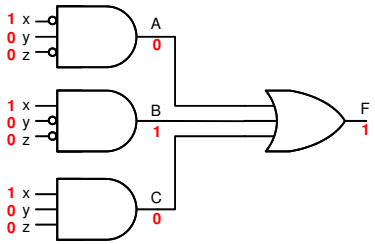


X	Y	Z	A	B	C	F
0	0	0	0	0	0	**0**
0	0	1	0	0	0	**0**
0	1	0	1	0	0	**1**
0	1	1	0	0	0	**0**
1	0	0	0	1	0	**1**
1	0	1	0	0	0	**0**
1	1	0	0	0	0	**0**
1	1	1	0	0	1	**1**

Using Decoders to Implement Functions

- Given an any logic function, it can be implemented with the superposition of decoders

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



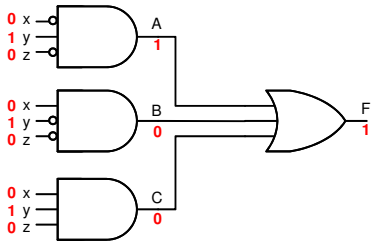
X	Y	Z	A	B	C	F
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	1	0	0	1
0	1	1	0	0	0	0
1	0	0	0	1	0	1
1	0	1	0	0	0	0
1	1	0	0	0	0	0
1	1	1	0	0	1	1

F(1,0,0) = 1

Using Decoders to Implement Functions

- Given an any logic function, it can be implemented with the superposition of decoders

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



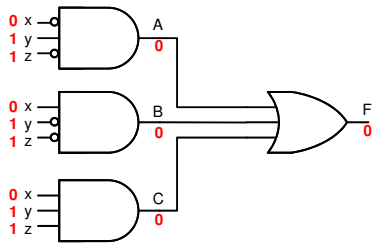
X	Y	Z	A	B	C	F
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	1	0	0	1
0	1	1	0	0	0	0
1	0	0	0	1	0	1
1	0	1	0	0	0	0
1	1	0	0	0	0	0
1	1	1	0	0	1	1

F(0,1,0) = 1

Using Decoders to Implement Functions

- Given an any logic function, it can be implemented with the superposition of decoders

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



X	Y	Z	A	B	C	F
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	1	0	0	1
0	1	1	0	0	0	0
1	0	0	0	1	0	1
1	0	1	0	0	0	0
1	1	0	0	0	0	0
1	1	1	0	0	1	1

F(0,1,1) = 0

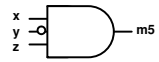
Defining Minterms

- We call an AND gate checking for one combination of a logic function (truth table) a **minterm**
- Write the expression for each minterm of 3 variables: x,y,z

						Minterms							
Row #		Minterm Expression	x	y	z	m_0	m_1	m_2	m_3	m_4	m_5	m_6	m_7
0	m_0		0	0	0	1	0	0	0	0	0	0	0
1	m_1		0	0	1	0	1	0	0	0	0	0	0
2	m_2		0	1	0	0	0	1	0	0	0	0	0
3	m_3		0	1	1	0	0	0	1	0	0	0	0
4	m_4		1	0	0	0	0	0	0	1	0	0	0
5	m_5		1	0	1	0	0	0	0	0	1	0	0
6	m_6		1	1	0	0	0	0	0	0	0	1	0
7	m_7		1	1	1	0	0	0	0	0	0	0	1

Checkers / Decoders

- The m_i functions on the previous slide are just AND gate checkers
  - That combination can be changed by adding inverters to the inputs
  - We can think of the AND gate as "checking" or "decoding" a specific combination and outputting a '1' when it matches.



AND gate decoding (checking for) combination 101

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

AND gate decoding (checking for) combination 000

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

Applying Minterms to Synthesize a Function

- Each numbered minterm checks whether the inputs are equal to the corresponding combination. When the inputs are equal, the minterm will evaluate to 1 and thus the whole function will evaluate to 1.

x	y	z	P	
0	0	0	0	
0	0	1	0	use…
0	1	0	1	m_2
0	1	1	1	m_3
1	0	0	0	
1	0	1	1	m_5
1	1	0	0	
1	1	1	1	m_7

$P = m_2 + m_3 + m_5 + m_7$
$= x'yz' + x'yz + xy'z + xyz$

when x,y,z = {0,1,0} = 2 then
$P = 0'\cdot1\cdot0' + 0'\cdot1\cdot0 + 0\cdot1'\cdot0 + 0\cdot1\cdot0$
$= 1 + 0 + 0 + 0 = 1$

when x,y,z = {1,0,1} = 5 then
$P = 1'\cdot0\cdot1' + 1'\cdot0\cdot1 + 1\cdot0'\cdot1 + 1\cdot0\cdot1$
$= 0 + 0 + 1 + 0 = 1$

when x,y,z = {0,0,1} = 5 then
$P = 0'\cdot0\cdot1' + 0'\cdot0\cdot1 + 0\cdot0'\cdot1 + 0\cdot0\cdot1$
$= 0 + 0 + 0 + 0 = 0$

Definitions

- **Literal:** A single bit _____ or its _____
  - **Good:** x, y', ALARM'
  - **Bad:** (x+y)
- **Product Term:** A _____ literal by itself or an _____'ing (not NAND'ing) of literals
  - **Good:** z, x•y, ALARM•PANIC•ENABLED
  - **BAD:** (x•y)', ALARM•(PANIC+ENABLED)
- **Minterm**: A product term where each input variable of a function appears as exactly one literal
  - f(x,y,z) =>
    - x'y'z
    - xyz
    - x'y
    - x+y+z

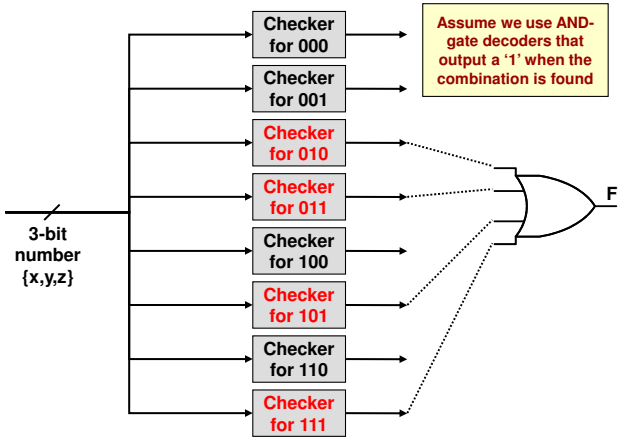
Minterms

- Consider F(A,B)
- One minterm per combination of the input variables
- Only *one* minterm can evaluate to 1 at any time

			m_0	m_1	m_2	m_3
A	B	F	A'•B'	A'•B	A•B'	A•B
0	0	0	1	0	0	0
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	0	0	0	0	1

Finding Equations/Circuits

- Given a function and checkers (called decoders) for each combination, we just need to OR together the checkers where F = 1



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

Canonical Sums

- Given a T.T., ___ together all the minterms where F = _
  - (Σ = SUM or OR of all the minterms)

$F = m_2+m_3+m_5+m_7$
$=(X'Y'Z')+(X'Y'Z)+$
$(XY'Z')+(XYZ)$

Canonical Sum:

$F = \Sigma_{xyz}(2,3,5,7)$

List the minterms where F is 1.

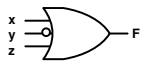
	X	Y	Z	F
m_0	0	0	0	0
m_1	0	0	1	0
m_2	0	1	0	1
m_3	0	1	1	1
m_4	1	0	0	0
m_5	1	0	1	1
m_6	1	1	0	0
m_7	1	1	1	1

Using OR-gate checkers

AN ALTERNATIVE

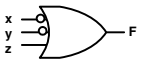
OR-Gate Checkers / Decoders

- An OR gate only outputs '0' for 1 combination
  - That combination can be changed by adding inverters to the inputs
  - We can think of the OR gate as "checking" or "decoding" a specific combination and outputting a '0' when it matches.



OR gate decoding (checking for) combination 010

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

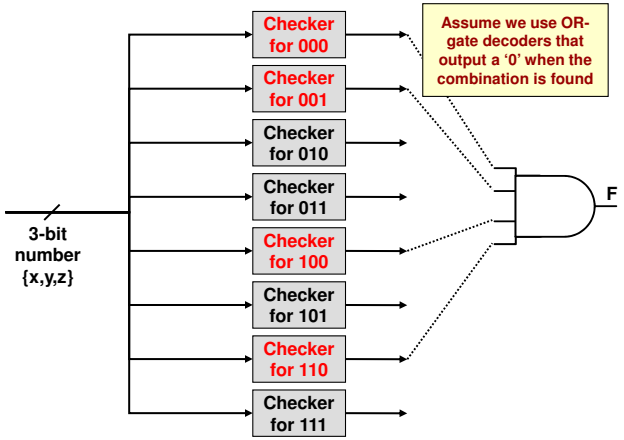


OR gate decoding (checking for) combination 110

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

Finding Equations/Circuits

- Given a function and checkers (called decoders) for each combination, we just need to AND together the checkers where F = 0



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

Applying Maxterms to Synthesize a Function

- Each output that should produce a '0' can be checked-for with an OR gate
  - We refer to that OR-gate checker as a Maxterm of the function (M_i)
- We then AND together the maxterms

x	y	z	P	use...
0	0	0	0	M_0
0	0	1	0	M_1
0	1	0	1	
0	1	1	1	
1	0	0	0	M_4
1	0	1	1	
1	1	0	0	M_6
1	1	1	1	

$P = M_0 \cdot M_1 \cdot M_4 \cdot M_6$
$= (x+y+z) \cdot (x+y+z') \cdot (x'+y+z) \cdot (x'+y'+z)$

when x,y,z = {0,0,1} = 1 then
$P = (0+0+1) \cdot (0+0+1') \cdot (0'+0+1) \cdot (0'+0'+1)$
$= 1 \cdot 0 \cdot 1 \cdot 1 = 0$

when x,y,z = {1,1,0} = 6 then
$P = (1+1+0) \cdot (1+1+0') \cdot (1'+1+0) \cdot (1'+1'+0)$
$= 1 \cdot 1 \cdot 1 \cdot 0 = 0$

when x,y,z = {1,1,1} = 7 then
$P = (1+1+1) \cdot (1+1+1') \cdot (1'+1+1) \cdot (1'+1'+1)$
$= 1 \cdot 1 \cdot 1 \cdot 1 = 1$

Canonical Products

- Given a T.T., AND together all the maxterms where F = 0

$F = M_0 \bullet M_1 \bullet M_4 \bullet M_6$
$= (X+Y+Z) \bullet (X+Y+Z') \bullet$
$(X'+Y+Z) \bullet (X'+Y'+Z)$

Canonical Product:

$F = \Pi_{xyz}(0,1,4,6)$

List the maxterms where F is 0.

	X	Y	Z	F
M_0	0	0	0	0
M_1	0	0	1	0
M_2	0	1	0	1
M_3	0	1	1	1
M_4	1	0	0	0
M_5	1	0	1	1
M_6	1	0	0	0
M_7	1	1	1	1

Definitions

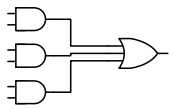
- **Literal:** A single bit variable or its inverse
  - **Good:** x, y', ALARM'
- **Sum Term:** A single literal by itself or an OR'ing (not NOR'ing) of literals
  - **Good:** z, x'+y, ALARM+PANIC+ENABLED
  - **BAD:** (x+y)', ALARM•(PANIC+ENABLED)
- **Maxterm**: A sum term where each input variable of a function appears as exactly one literal
  - f(x,y,z) =>
    - x'+y'+z
    - x+y+z
    - x'•y
    - x•y•z

You are only responsible for Canonical Sums (not Products)

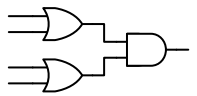
LOGIC FUNCTION NOTATION

Boolean Algebra Terminology

- **SOP (_____) Form**: An SOP expression is a logical sum (OR) of product terms
  - Correct Examples: [x'•y'•z + w + a'•b•c], [w + x'•z•y + y'z]
  - Incorrect Examples: [x'•y•z + w•(a+b)], [x•y + (y'•z)']
- **POS (_____) Form**: A POS expression is a logical product (AND) of sum terms.
  - Correct Examples: [(x+y'+z) • (w'+z) • (a)], [z'•(x+y)•(w'+y)]
  - Incorrect Examples: [x' + y•(x+w)], [(x+y)•(x+z)']
- SOP equations yield _____ circuits with AND gates in the 1st level with OR gates in the 2nd (aka AND-OR circuits)
- POS equations yield _____ circuits with OR gates in the 1st level with OR gates in the 2nd (aka OR-AND circuits)



AND-OR



OR-AND

Check Yourself

Expression	SOP / POS / Both / Neither
w•x•(y•z)' + xy'z + w	
xy+xz+(w'yz)	
(w+y'+z)(w+x)	
(w+y)x(w'+z)	
w'y + w'y + xy'	
w+x+y	

Canonical Sums & Products

- **Canonical Sum**: An ____ expression where all the product terms are _____ (i.e. have each literal in each product term)
- **Canonical Product**: A ____ expression where all the sum terms are _____ (i.e. each literal in each sum term)

Canonical Sum:

$F = \Sigma_{xyz}(2,3,5,7)$

$F = m_2+m_3+m_5+m_7$

$=(X'Y'Z')+(X'Y'Z)+(XY'Z')+(XYZ)$

Canonical Product:

$F = \Pi_{xyz}(0,1,4,6)$

$F = M_0 \bullet M_1 \bullet M_4 \bullet M_6 =$

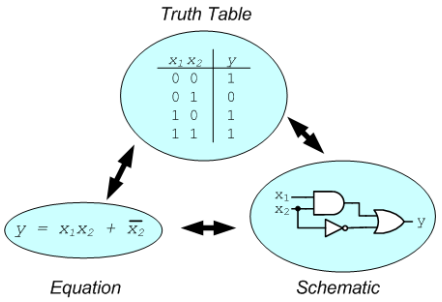
$(X+Y+Z)\bullet(X+Y+Z')\bullet(X'+Y+Z)\bullet(X'+Y'+Z)$

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

Logic Functions

- A logic function maps input combinations to an output value ('1' or '0')
- 3 possible representations of a function
  - Equation
  - Schematic
  - Truth Table
- Can convert between representations
- Truth table is only unique representation*

* Canonical Sums/Products (minterm/maxterm) representation provides a standard equation/schematic form that is unique per function



Unique Representations

- Canonical => Same functions will have same representations
- Truth Tables along with Canonical Sums and Products specify a function *uniquely*
- Equations/circuit schematics are NOT inherently canonical

Truth Table

x	y	z	P
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Canonical Sum

$P = \sum_{x,y,z}(2,3,5,7)$

ON-Set of P (minterms)

Yields SOP equation, AND-OR circuit

Canonical Product

$P = \prod_{x,y,z}(0,1,4,6)$

OFF-Set of P (maxterms)

Yields POS equation, OR-AND circuit

2 & 3 Variable Theorems

T6	X+Y = Y+X	**T6'**	X•Y = Y•X	Commutativity
T7	(X+Y)+Z = X+(Y+Z)	**T7'**	(X•Y)•Z = X•(Y•Z)	Associativity
T8	XY+XZ = X(Y+Z)	**T8'**	(X+Y)(X+Z) = X+YZ	Distribution & Factoring
T9	X + XY = X	**T9'**	X(X+Y) = X	Covering
T10	XY + XY' = X	**T10'**	(X+Y)(X+Y') = X	Combining
T11	XY + X'Z + YZ = XY + X'Z	**T11'**	(X+Y)(X'+Z)(Y+Z) = (X+Y)(X'+Z)	Consensus
DM	(X+Y)' = X'•Y'	**DM'**	(X•Y)'=X'+Y'	DeMorgan's

Proofs Through Other Theorems

T8	XY+XZ = X(Y+Z)	**T8'**	(X+Y)(X+Z) = X+YZ
T9	X + XY = X	**T9'**	X(X+Y) = X
T10	XY + XY' = X	**T10'**	(X+Y)(X+Y') = X
T11	XY + X'Z + YZ = XY + X'Z	**T11'**	(X+Y)(X'+Z)(Y+Z) = (X+Y)(X'+Z)

- Prove T9: X + XY = X
  - –
  - –
  - –
- Prove T10: XY + XY' = X
  - –
  - –
  - –
- Prove T10': (X+Y)(X+Y')=X
  - –
  - –
  - –
  - –

OR

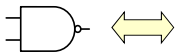
DeMorgan's Theorem

- Inverting output of an AND gate = inverting the inputs of an OR gate
- Inverting output of an OR gate = inverting the inputs of an AND gate

A function's inverse is equivalent to inverting all the inputs and changing AND to OR and vice versa

A	B	Out
0	0	1
0	1	1
1	0	1
1	1	0

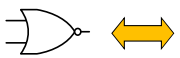
$\overline{A \bullet B}$



A	B	Out
0	0	1
0	1	1
1	0	1
1	1	0

A	B	Out
0	0	1
0	1	0
1	0	0
1	1	0

$\overline{A+B}$



A	B	Out
0	0	1
0	1	0
1	0	0
1	1	0

Analogy: Turning a gate "inside-out" (like your socks).

DeMorgan's Theorem

$$F = \overline{(\overline{X}+Y) + \overline{Z} \bullet (Y+W)}$$

Use DeMorgan's theorem to simplify "move" inversions (either to break-up "big bars" or join "small bars"

$$F = \overline{(\overline{X}+Y) + \overline{Z} \bullet (Y+W)}$$

$$F = \overline{(\overline{X}+Y)} \bullet \overline{(\overline{Z} \bullet (Y+W))}$$

$$F =$$

$$F =$$

Generalized DeMorgan's Theorem

$$F'(X_1,\ldots,X_n,+,\bullet) = F(X_1',\ldots,X_n',\bullet,+)$$

To find F', swap AND's and OR's and complement each literal. However, you must maintain the original order of operations.

Note: This parentheses doesn't matter (we are just OR'ing X', Y, and the following subexpression)

$$F = (\overline{X}+Y) + \overline{Z} \bullet (Y+W)$$

$$F = \overline{X}+Y + (\overline{Z} \bullet (Y+W))$$

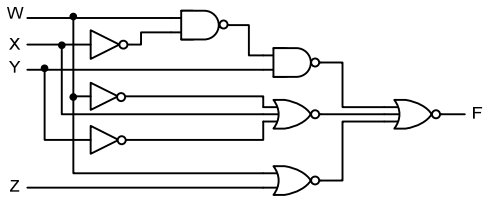
Fully parenthesized to show original order of ops.

$$\overline{F} = X \bullet \overline{Y} \bullet (Z + (\overline{Y} \bullet \overline{W}))$$

AND's & OR's swapped
Each literal is inverted

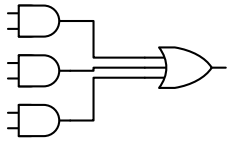
DeMorgan's Theorem Example

- Cancel as many bubbles as you can using DeMorgan's theorem.



AND-OR / NAND-NAND

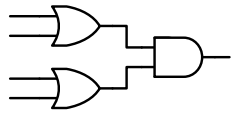
- Canonical Sums yield
  - AND-OR Implementation
  - NAND-NAND Implementation



II

OR-AND / NOR-NOR

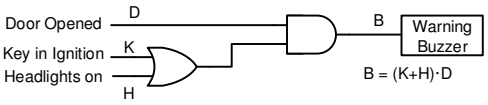
- Canonical Products yield
  - OR-AND Implementation
  - NOR-NOR Implementation

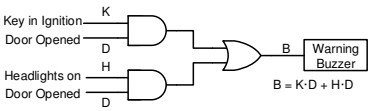


II

Example: Automobile Buzzer

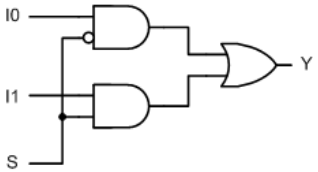
- Convert each implementation to use either just NOR or just NAND gates + inverters





Convert to NAND-NAND

- Convert the 2-to-1 mux below to use just NAND or NOR gates?



Logic Synthesis

- Describe the function
  - Usually with a truth table
- Find the sum of minterm (or product of maxterm) expression
  - In this class, you're only responsible for the sum of minterm approach
- Use Boolean Algebra (T8-T11) to find a simplified expression

Synthesize/Simplify Exercise 1

- Synthesize this function
  - First generate the canonical sum
  - Then use theorems to simplify

T8	XY+XZ = X(Y+Z)	**T8'**	(X+Y)(X+Z) = X+YZ
T9	X + XY = X	**T9'**	X(X+Y) = X
T10	XY + XY' = X	**T10'**	(X+Y)(X+Y') = X
T11	XY + X'Z + YZ = XY + X'Z	**T11'**	(X+Y)(X'+Z)(Y+Z) = (X+Y)(X'+Z)

Primes between 0-7

X	Y	Z	P
0	0	0	**0**
0	0	1	**0**
0	1	0	**1**
0	1	1	**1**
1	0	0	**0**
1	0	1	**1**
1	1	0	**0**
1	1	1	**1**

Synthesize/Simplify Exercise 2

- Synthesize each output separately
  - First generate the canonical sum
  - Then use theorems to simplify

T8	XY+XZ = X(Y+Z)	**T8'**	(X+Y)(X+Z) = X+YZ
T9	X + XY = X	**T9'**	X(X+Y) = X
T10	XY + XY' = X	**T10'**	(X+Y)(X+Y') = X
T11	XY + X'Z + YZ = XY + X'Z	**T11'**	(X+Y)(X'+Z)(Y+Z) = (X+Y)(X'+Z)

I3	I2	I1	M1	M0
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
0	1	1	1	0
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

Encode the highest input ID (ie. 3, 2, or 1) that is ON (=1)

Synthesize/Simplify Exercise 3 (Optional)

- Synthesize each output separately
  - First generate the canonical sum
  - Then use theorems to simplify

T8	XY+XZ = X(Y+Z)	**T8'**	(X+Y)(X+Z) = X+YZ
T9	X + XY = X	**T9'**	X(X+Y) = X
T10	XY + XY' = X	**T10'**	(X+Y)(X+Y') = X
T11	XY + X'Z + YZ = XY + X'Z	**T11'**	(X+Y)(X'+Z)(Y+Z) = (X+Y)(X'+Z)

1's Count of Inputs

I3	I2	I1	C1	C0
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1