Unit 8

Minterm and Canonical Sums

2- and 3-Variable Boolean Algebra Theorems

DeMorgan's Theorem

Simplification using Boolean Algebra

Duality

- As we progress in this unit, remember and look for the idea of duality at work
- Duality says: A new, true statement could be found from another by swapping:
  - _____ ⇔ _____
  - _____ ⇔ _____

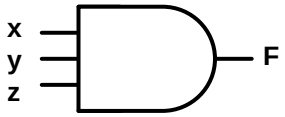
$X + 1 = 1$ ⇒ _____

Original equation **Dual**

CHECKERS / DECODERS

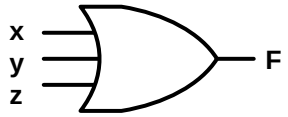
Gates

- Gates can have more than 2 inputs but the operations stay the same
  - AND = output = 1 if ALL inputs are 1
    - Outputs 1 for only 1 input combination
  - OR = output = 1 if ANY input is 1
    - Outputs 0 for only 1 input combination



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	0
1	1	0	0
1	1	1	1

3-input AND

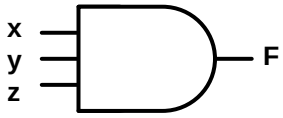


X	Y	Z	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

3-input OR

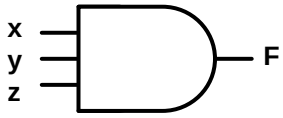
Checkers / Decoders

- An AND gate only outputs ‘1’ for 1 combination
  - 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.



Add inverters to create an 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

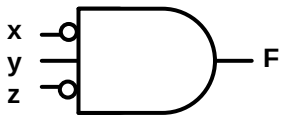


Add inverters to create an AND gate decoding (checking for) combination 000

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

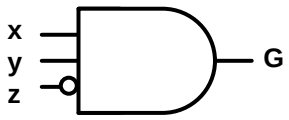
Checkers / Decoders

- Place inverters at the input of the AND gates such that
  - F produces '1' only for input combination {x,y,z} = {_____}
  - G produces '1' only for input combination {x,y,z} = {_____}



AND gate decoding (checking for) combination ____

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

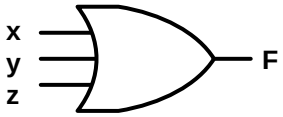


AND gate decoding (checking for) combination ____

X	Y	Z	G
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

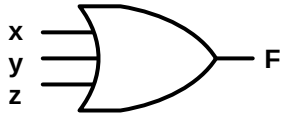
Checkers / Decoders

- 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.



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

Add inverters to create an OR gate decoding (checking for) combination 010



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

Add inverters to create an OR gate decoding (checking for) combination 110

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

Design a circuit that finds the minimum valued 8-bit number in a set of 32 values in under 200 ns

Problem specification and requirements

Truth Tables

X	Y	C_{in}	C_{out}	S
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

Circuit Design

Boolean Algebra

Canonical Sums/Products

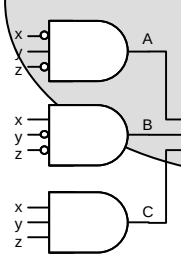
Karnaugh Maps

CAD tools

Circuit Analysis

$F = x'yz' + xy'z'$

Circuits and Equations



SYNTHESIZING LOGIC FUNCTIONS

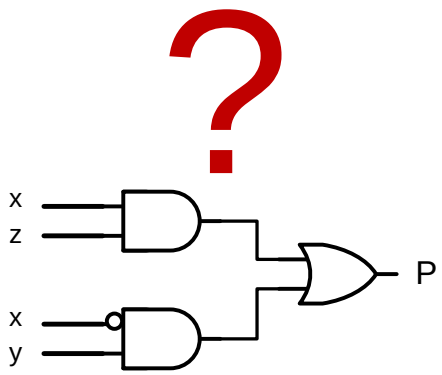
The Problem

- The goal of this unit is to teach you how you can take ***ANY*** logic function expressed as a _____ and design a digital circuit to implement that logic function

How can I find a circuit that implements this truth table?

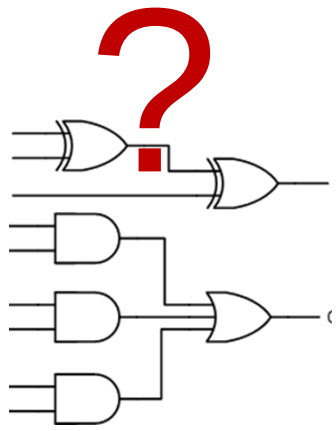
Primes between 0-7

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 (Addition) 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**



Two Approaches: Minterms & Maxterms

- Because of duality, there are at least two ways to implement any circuit
- Using _____ gate checkers (aka **product- or "_____" terms**)
  - Then combining their results with a single OR gate
- Using _____ gate checkers (aka **sum- or "_____" terms**)
  - Then combining their results with a single AND gate

Using AND Gates (Minterms) to Implement Functions

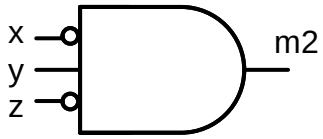
- Given an any logic function, it can be implemented with the superposition of AND gate decoders/checkers

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 AND Gates (Minterms) to Implement Functions

- Generate an **AND** gate checker ("minterm") for each combination *where the output of the logic function evaluates to **1** (i.e. F=1)*

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

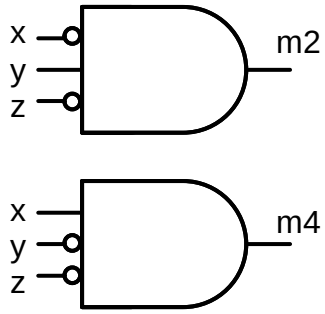


X	Y	Z	m2
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Using AND Gates (Minterms) to Implement Functions

- Generate an **AND** gate checker ("minterm") for each combination *where the output of the logic function evaluates to **1** (i.e. F=1)*

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

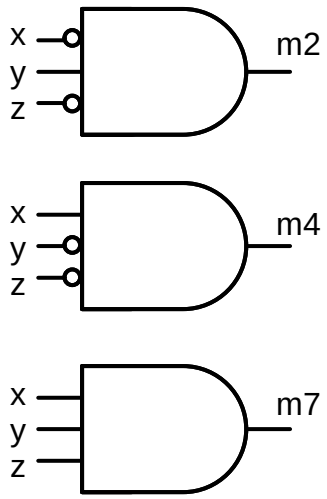


X	Y	Z	m2	m4
0	0	0	0	0
0	0	1	0	0
0	1	0	1	0
0	1	1	0	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	0	0

Using AND Gates (Minterms) to Implement Functions

- Generate an **AND** gate checker ("minterm") for each combination *where the output of the logic function evaluates to **1** (i.e. F=1)*

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

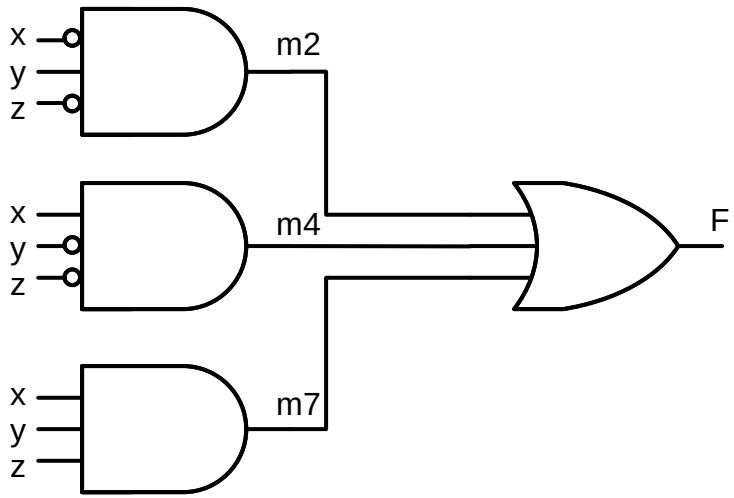


X	Y	Z	m2	m4	m7
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	1	0	0
0	1	1	0	0	0
1	0	0	0	1	0
1	0	1	0	0	0
1	1	0	0	0	0
1	1	1	0	0	1

Using AND Gates (Minterms) to Implement Functions

- Then, OR together all outputs of the AND gate checkers to form the overall function output

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

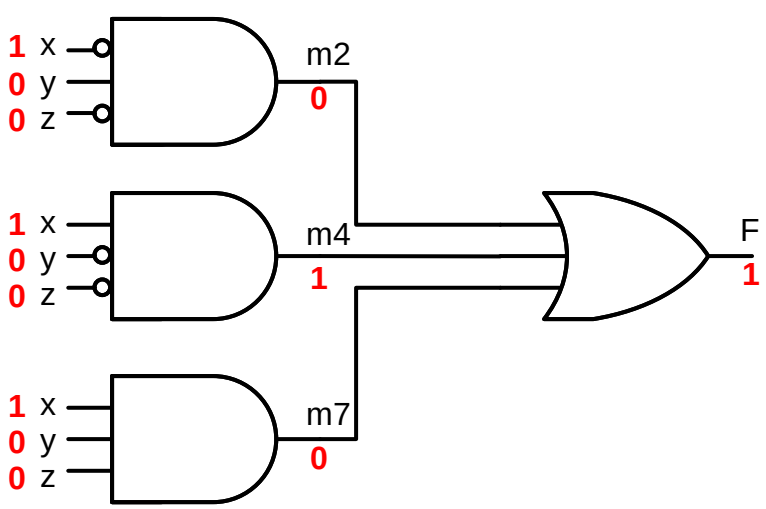


X	Y	Z	m2	m4	m7	F
0	0	0	0	0	0	**0**
0	0	1	0	0	0	**0**
0	1	0	1	0	0	**1**
0	1	1	0	0	0	**0**
1	0	0	0	1	0	**1**
1	0	1	0	0	0	**0**
1	1	0	0	0	0	**0**
1	1	1	0	0	1	**1**

Using AND Gates (Minterms) to Implement Functions

- Test it by plugging in combinations that should cause F=1
  - As long as one AND gate outputs 1, the output will be 1

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



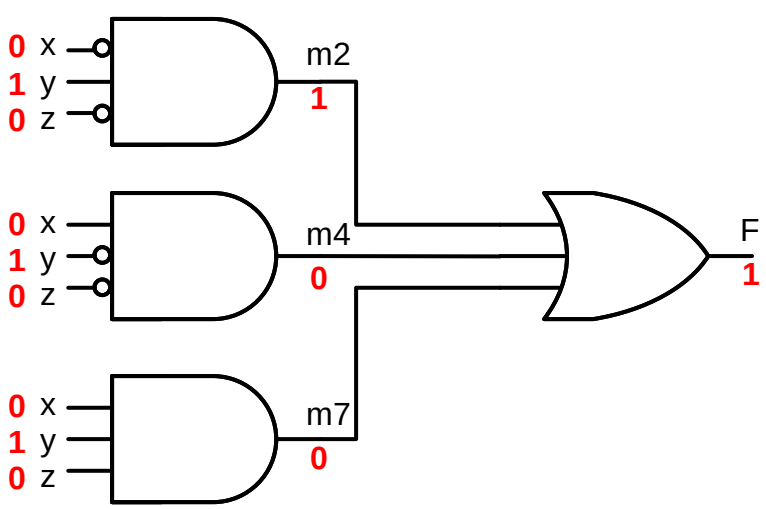
X	Y	Z	m2	m4	m7	F
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	1	0	0	1
0	1	1	0	0	0	0
1	0	0	0	1	0	1
1	0	1	0	0	0	0
1	1	0	0	0	0	0
1	1	1	0	0	1	1

F(1,0,0) = 1

Using AND Gates (Minterms) to Implement Functions

- Test it by plugging in combinations that should cause F=1
  - As long as one AND gate outputs 1, the output will be 1

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



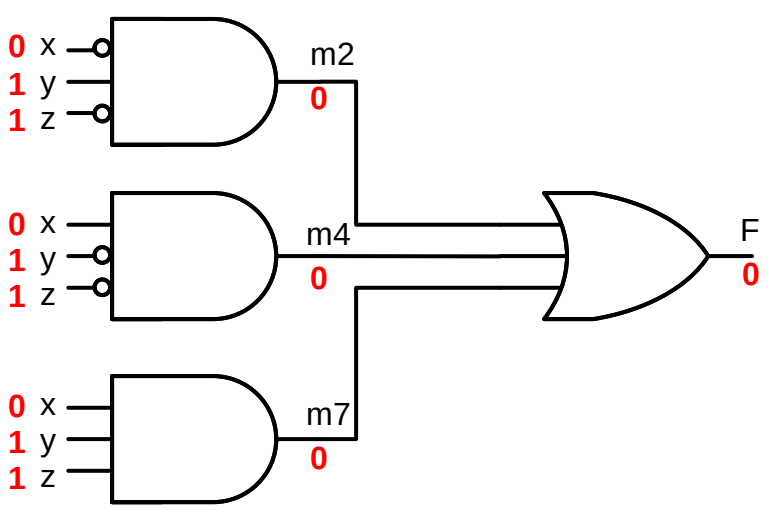
X	Y	Z	m2	m4	m7	F
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	1	0	0	1
0	1	1	0	0	0	0
1	0	0	0	1	0	1
1	0	1	0	0	0	0
1	1	0	0	0	0	0
1	1	1	0	0	1	1

F(0,1,0) = 1

Using AND Gates (Minterms) to Implement Functions

- Test it by plugging in combinations that should cause F=0
  - All AND gates output 0, thus the OR gate will output 0

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

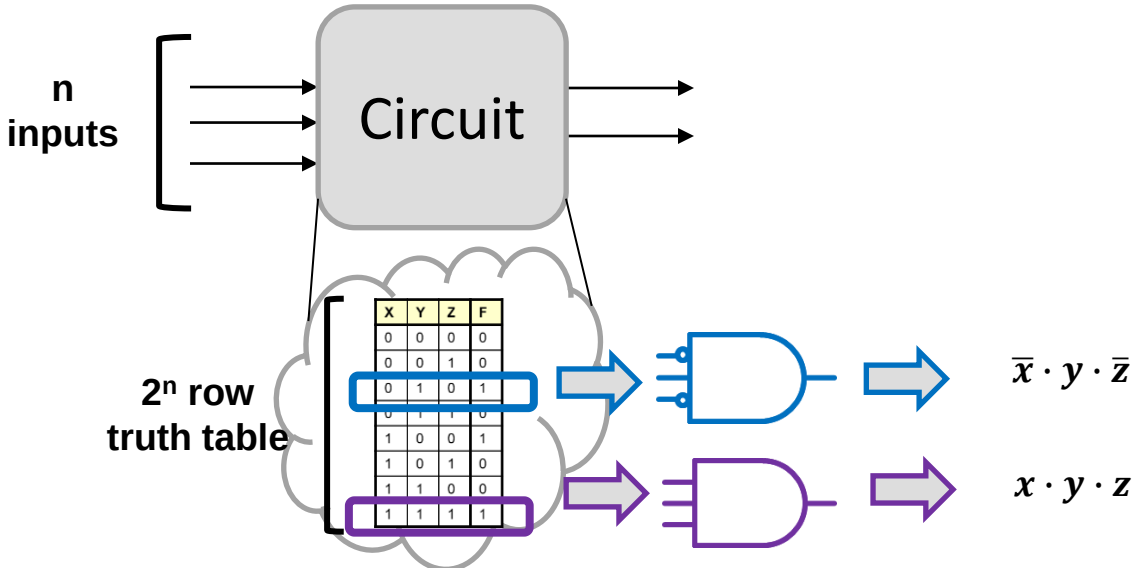


X	Y	Z	m2	m4	m7	F
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	1	0	0	1
0	1	1	0	0	0	0
1	0	0	0	1	0	1
1	0	1	0	0	0	0
1	1	0	0	0	0	0
1	1	1	0	0	1	1

F(0,1,1) = 0

Minterms

- An n-input combinational function can be described with 2^n row truth table
- Each row in the truth table (input combination) has a **unique logic expression** (i.e. an AND gate) that only evaluates to '1' for that combination
  - This logic expression is known as a **minterm**



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.

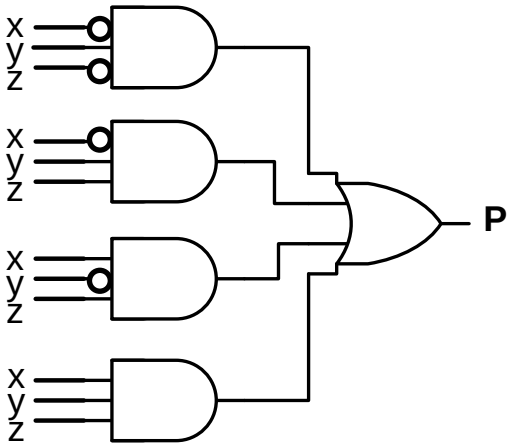
x	y	z	P	
0	0	0	0	
0	0	1	0	use…
0	1	0	1	m_2
0	1	1	1	m_3
1	0	0	0	
1	0	1	1	m_5
1	1	0	0	
1	1	1	1	m_7

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

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} = 1 then
$P = 0'\cdot0\cdot1' + 0'\cdot0\cdot1 + 0\cdot0'\cdot1 + 0\cdot0\cdot1$
$= 0 + 0 + 0 + 0 = 0$

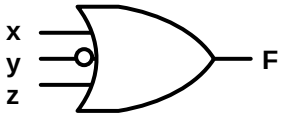


Using OR-gate checkers

AN ALTERNATIVE

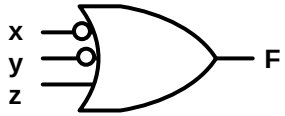
OR-Gate Checkers / Decoders

- An OR gate only outputs ‘0’ for a single 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

Using OR Checkers to Implement Functions

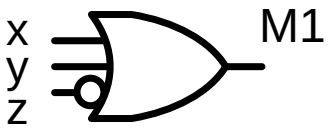
- Given an any logic function, it can be implemented with the superposition of OR-gate checkers/decoders (aka "sum or maxterms")

X	Y	Z	G
0	0	0	1
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

Using OR Checkers to Implement Functions

- Generate an **OR** gate checker ("maxterm") for each combination *where the output of the logic function evaluates to **0** (i.e. G=0)*

X	Y	Z	G
0	0	0	1
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

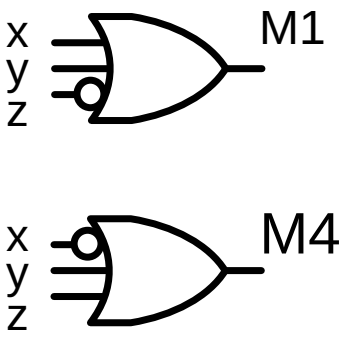


X	Y	Z	M1
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Using OR Checkers to Implement Functions

- Generate an **OR** gate checker ("maxterm") for each combination *where the output of the logic function evaluates to **0** (i.e. G=0)*

X	Y	Z	G
0	0	0	1
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

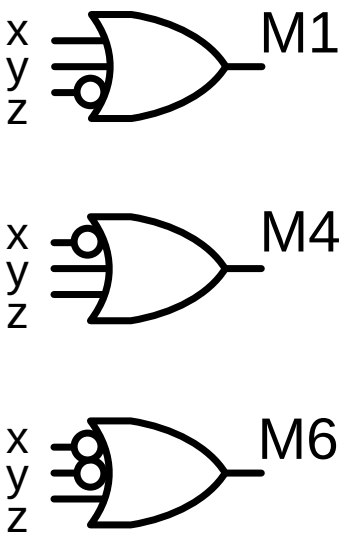


X	Y	Z	M1	M4
0	0	0	1	1
0	0	1	0	1
0	1	0	1	1
0	1	1	1	1
1	0	0	1	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

Using OR Checkers to Implement Functions

- Generate an **OR** gate checker ("maxterm") for each combination *where the output of the logic function evaluates to **0** (i.e. G=0)*

X	Y	Z	G
0	0	0	1
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

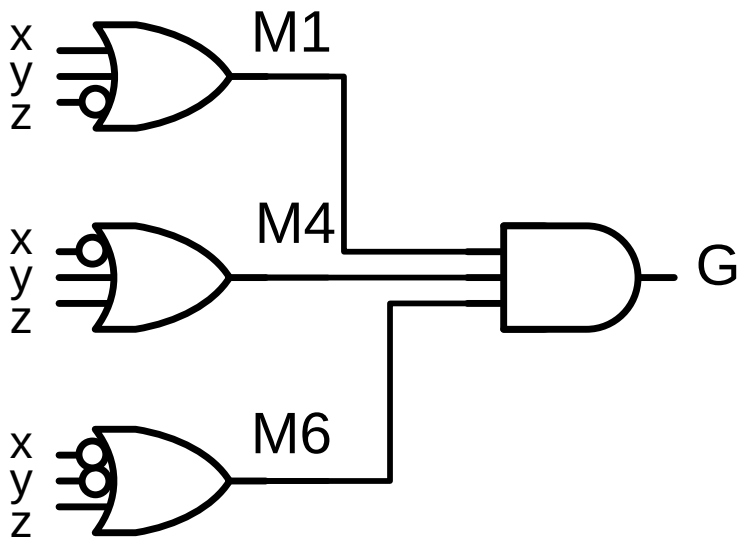


X	Y	Z	M1	M4	M6
0	0	0	1	1	1
0	0	1	0	1	1
0	1	0	1	1	1
0	1	1	1	1	1
1	0	0	1	0	1
1	0	1	1	1	1
1	1	0	1	1	0
1	1	1	1	1	1

Using OR Checkers to Implement Functions

- Then, AND together all outputs of the OR gate checkers to form the overall function output

X	Y	Z	G
0	0	0	1
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

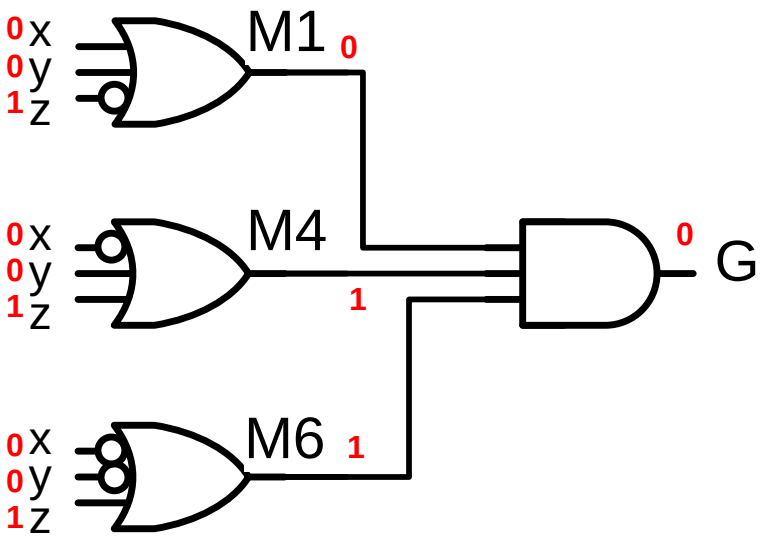


X	Y	Z	M1	M4	M6	G
0	0	0	1	1	1	**1**
0	0	1	0	1	1	**0**
0	1	0	1	1	1	**1**
0	1	1	1	1	1	**1**
1	0	0	1	0	1	**0**
1	0	1	1	1	1	**1**
1	1	0	1	1	0	**0**
1	1	1	1	1	1	**1**

Using OR Checkers to Implement Functions

- Test it by plugging in combinations that should cause G=0
  - As long as one OR gate outputs 0, the output will be 0

X	Y	Z	G
0	0	0	1
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



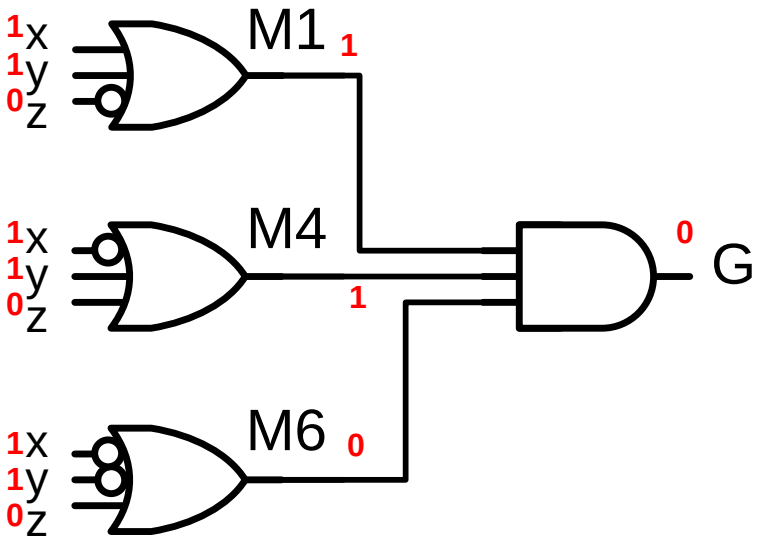
X	Y	Z	M1	M4	M6	G
0	0	0	1	1	1	**1**
0	0	1	0	1	1	**0**
0	1	0	1	1	1	**1**
0	1	1	1	1	1	**1**
1	0	0	1	0	1	**0**
1	0	1	1	1	1	**1**
1	1	0	1	1	0	**0**
1	1	1	1	1	1	**1**

F(0,0,1) = 0

Using OR Checkers to Implement Functions

- Test it by plugging in combinations that should cause G=0
  - As long as one OR gate outputs 0, the output will be 0

X	Y	Z	G
0	0	0	1
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



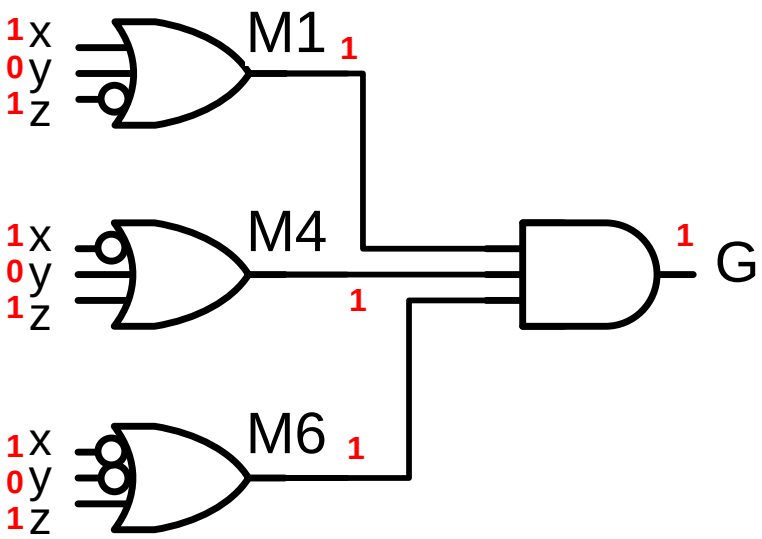
F(1,1,0) = 0

X	Y	Z	M1	M4	M6	G
0	0	0	1	1	1	**1**
0	0	1	0	1	1	**0**
0	1	0	1	1	1	**1**
0	1	1	1	1	1	**1**
1	0	0	1	0	1	**0**
1	0	1	1	1	1	**1**
1	1	0	1	1	0	**0**
1	1	1	1	1	1	**1**

Using OR Checkers to Implement Functions

- Test it by plugging in combinations that should cause G=1
  - All OR gates output 1, thus the AND gate will output 1

X	Y	Z	G
0	0	0	1
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



X	Y	Z	M1	M4	M6	G
0	0	0	1	1	1	**1**
0	0	1	0	1	1	**0**
0	1	0	1	1	1	**1**
0	1	1	1	1	1	**1**
1	0	0	1	0	1	**0**
1	0	1	1	1	1	**1**
1	1	0	1	1	0	**0**
1	1	1	1	1	1	**1**

F(1,0,1) = 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) where i represents the decimal value of the binary combination being checked
- We then AND together the maxterms

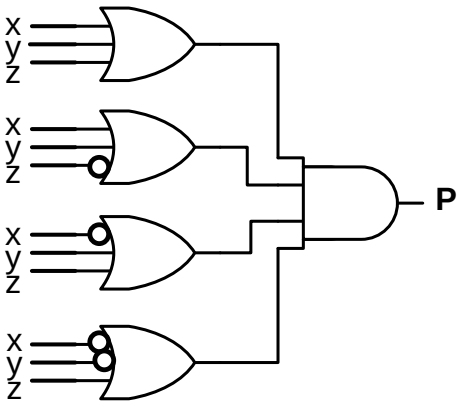
x	y	z	P	use…
0	0	0	0	M_0
0	0	1	0	M_1
0	1	0	1	
0	1	1	1	
1	0	0	0	M_4
1	0	1	1	
1	1	0	0	M_6
1	1	1	1	

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

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

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

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



Defining Min-/Max-terms

- Below are the min-/max-terms for a function of 3-inputs: x,y,z
- Given a desired output, the designer could choose to include the appropriate set of min-/max-terms

			Inputs			Min-/Max-term Outputs							
Row #	Abbrev	Min-/Max-term Expression	x	y	z	m_0/ M_0	m_1/ M_1	m_2/ M_2	m_3/ M_3	m_4/ M_4	m_5/ M_5	m_6/ M_6	m_7/ M_7
0	m_0/M_0	x'•y'•z' / x+y+z	0	0	0	1/0	0/1	0/1	0/1	0/1	0/1	0/1	0/1
1	m_1/M_1	x'•y'•z / x+y+z'	0	0	1	0/1	1/0	0/1	0/1	0/1	0/1	0/1	0/1
2	m_2/M_2	x'•y•z' / x+y'+z	0	1	0	0/1	0/1	1/0	0/1	0/1	0/1	0/1	0/1
3	m_3/M_3	x'•y•z / x+y'+z'	0	1	1	0/1	0/1	0/1	1/0	0/1	0/1	0/1	0/1
4	m_4/M_4	x•y'•z' / x'+y+z	1	0	0	0/1	0/1	0/1	0/1	1/0	0/1	0/1	0/1
5	m_5/M_5	x•y'•z / x'+y+z'	1	0	1	0/1	0/1	0/1	0/1	0/1	1/0	0/1	0/1
6	m_6/M_6	x•y•z' / x'+y'+z	1	1	0	0/1	0/1	0/1	0/1	0/1	0/1	1/0	0/1
7	m_7/M_7	x•y•z / x'+y'+z'	1	1	1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	1/0

Finding simplified equations and circuits

2- AND 3-VARIABLE THEOREMS

Why Boolean Algebra

- We can now convert ***any truth table*** into an equation and circuit by using **minterms or maxterms**
- But minterms/maxterms yield the _____ equation/circuit
- By starting with sum of minterm (product of maxterm) form and then using _____ to simplify, we can arrive and smaller (even minimal) circuits

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

Logic Synthesis

- Describe the function
  - Usually with a truth table
- Find the sum of minterm (or product of maxterm) expression
- 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)

- 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**

Synthesize/Simplify Exercise 2a

- Synthesize each output separately
  - First generate the canonical prod.
  - 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)

- G1 =
- G0 =

A	B	C	G1	G0
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 2b

- 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)

- G1 = m2 + m3 + m4 + m5 + m6 + m7
  - A'BC' + A'BC + AB'C' + AB'C + ABC' + ABC
  - [T3 (A = A + A) allows us to replicate m_6 and m_7 (ABC' + ABC)]
  - A'BC' + A'BC + ABC' + ABC + AB'C' + AB'C + ABC' + ABC
  - B(A'C' +A'C + AC' + AC) + A(B'C' + B'C + BC' + BC) [T8]
  - B(A'(C'+C) + A(C'+C)) + A(B'(C+C') + B(C' + C)) [T8/T5 or T10]
  - B(A'+ A) + A(B' + B) [T8/T5] = **B + A** [Final Answer]
- G0 = m1 + m4 + m5 + m6 + m7
  - A'B'C + AB'C' + AB'C + ABC' + ABC
  - A'B'C + AB'C + AB'C' + AB'C + ABC' + ABC [Use T3 to replicate m_5]
  - B'C(A' + A) + A(B'C' + B'C + BC' + BC) [T8]
  - B'C + A [T8/T5 or T10] = **B'C + A** [Final Answer]

A	B	C	G1	G0
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)

- C1 =

- C0 = m1 + m2 + m4 + m7
  - A'B'C + A'BC' + AB'C' + ABC [Not much to factor that will cause simplification]
  - A'(B'C + BC') + A(B'C' + BC) [But write the truth tables of B'C+BC' and B'C'+BC]
  - A'(B⊕C) + A(B⊕C)' [But if we let W=B⊕C then we have A'W + AW'…write its TT]
  - **A ⊕ B ⊕ C [Final Answer]**

1's Count of Inputs

A	B	C	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**

How to make faster circuits…

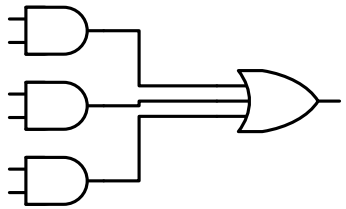
DEFINITIONS, EXPRESSION FORMS, SPEED, AND DEMORGAN'S

Definitions

- **Literal:** A single bit _____ or its _____
  - **Good:** `x, y', SLEEPING'`
  - **Bad:** `(x+y)`
- **Product Term:** A single literal by itself or an _____'ing (not _____'ing) of literals
  - **Good:** `z, x•y, AWAKE•LISTENING•THINKING`
  - **BAD:** `(x•y)', AWAKE•(LISTENING+THINKING)`
  - **The _____ we defined earlier are product terms where EACH input variable of a function is 1 literal in the product term**
- **Sum Term:** A single literal by itself or an _____'ing (not _____'ing) of literals
  - **Good:** `z, x'+y, CURIOUS+PERSISTENT`
  - **BAD:** `(x+y)', TIRED•(BORED+SLEEPY)`
  - **The _____ we defined earlier are sum terms where EACH input variable of a function is 1 literal in the sum term**

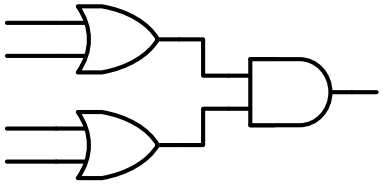
Expression/Circuit Forms

- **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)']
- SOP equations yield ___**-level circuits** with AND gates in the 1st level with an OR gate in the 2nd (aka _____ circuits)



SOP ⇔ AND-OR
(Sum of products yields AND-OR circuits)

- **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)']
- POS equations yield ___**-level circuits** with OR gates in the 1st level with an AND gate in the 2nd (aka _____ circuits)



POS ⇔ OR-AND
(Product of sums yields OR-AND circuits)

- 1 level circuits (i.e. a single gate) are generally BOTH SOP and POS

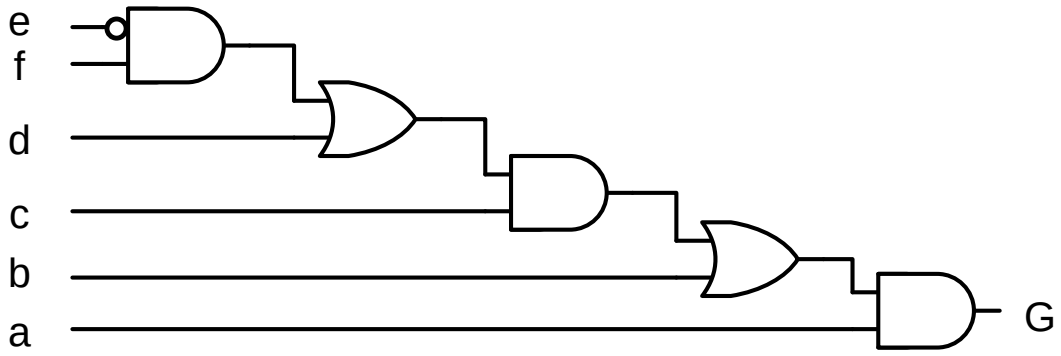
Check Yourself

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)	
wy + wy + xy’	
w+x+y	

Factoring and Distributing (Size vs. Speed)

- Factoring decreases _____
- Distributing decreases _____

$$G = a \cdot (b + (c \cdot (d + \bar{e}f))) =$$_____

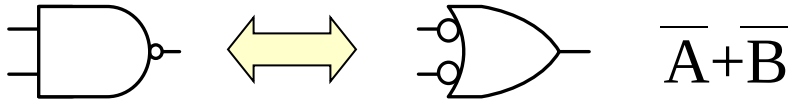


DeMorgan's Theorem

- 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

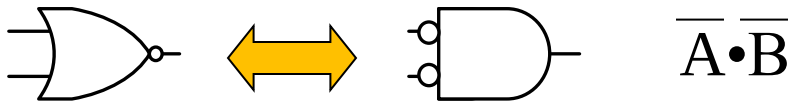
A function's inverse is equivalent to inverting all the inputs and changing AND to OR and vice versa

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

$\overline{A \bullet B}$  $\overline{A}+\overline{B}$

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

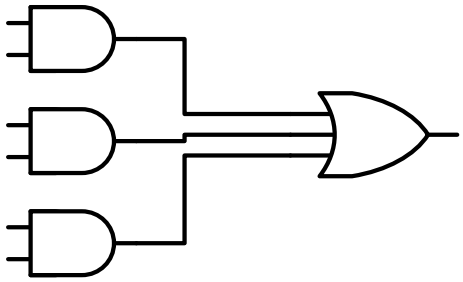
$\overline{A+B}$  $\overline{A} \bullet \overline{B}$

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

Analogy: Turning a gate "_____" (like your _____).

AND-OR / NAND-NAND

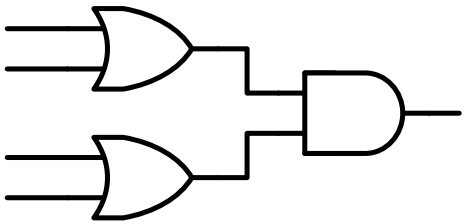
- Canonical Sums yield
  - AND-OR Implementation
  - _____ Implementation
- Recall inverting gates such as NAND gates may be "_____" or have desirable properties vs. typical AND/OR gates



||

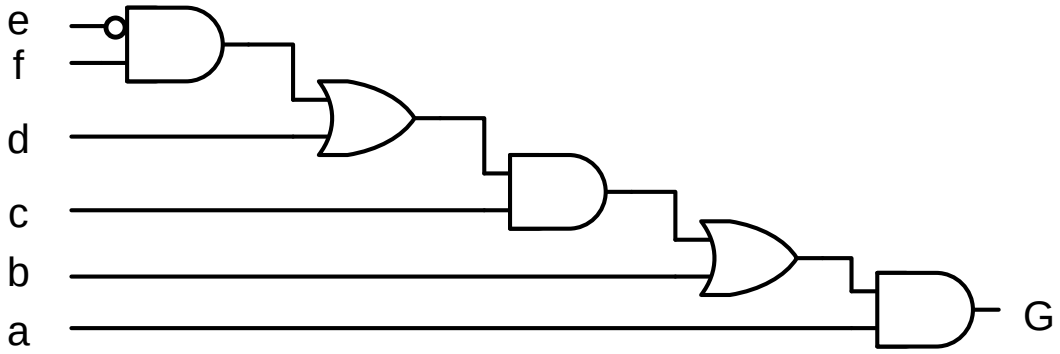
OR-AND / NOR-NOR

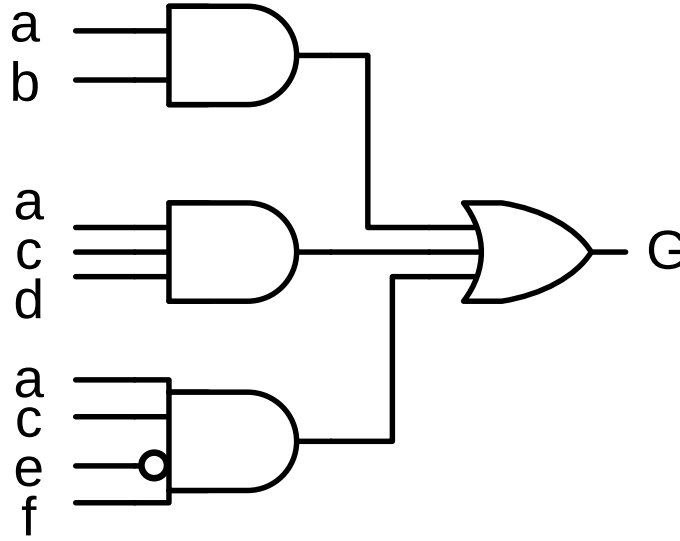
- Canonical Products yield
  - OR-AND Implementation
  - _____ Implementation
- Recall inverting gates such as NOR gates may be "faster" or have desirable properties vs. typical AND/OR gates



DeMorgan's Practice

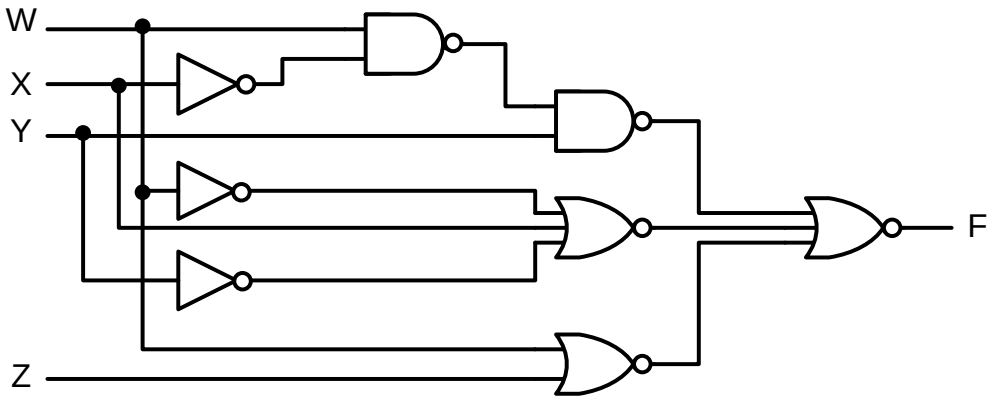
- Convert the circuits shown below to use only NAND or NOR gates?



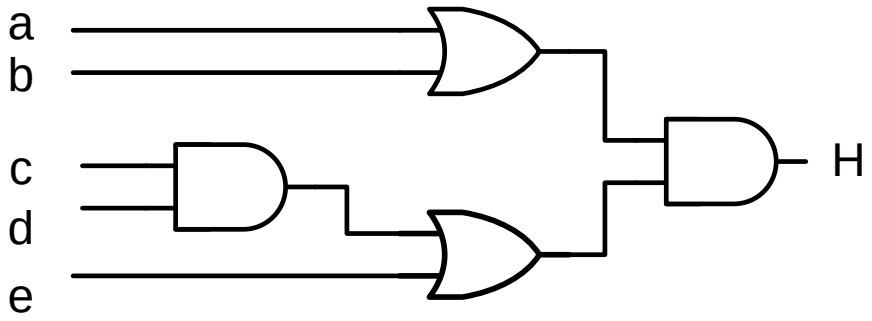


DeMorgan's Theorem Example

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



- Convert as many gates as possible to NOR gates. You are allowed to add additional inverters



DeMorgan's Theorem

- DeMorgan's let's us break large inversions (of whole expressions) into smaller inversions (of individual literals).
  - This is necessary to arrive at SOP or POS (which can only have inversions of literals)
- Recursively find the last (lowest precedence operation) and apply DeMorgan's theorem by flipping the operation and inverting the inputs

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

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

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

Additional Content

Not Tested

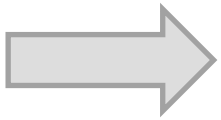
Canonical Sums and Products

LOGIC FUNCTION NOTATION

Canonical Sums and Products

- Truth tables require us to list all 2^n combinations of the n inputs
- A shorthand for a truth table is to describe the function using the canonical sum (sigma, ___) or product (pi, ____) notation
- These forms of expressing a function have all the information in the truth table but can be written more compactly
  - Though still may require listing 2^n input values
- We'll often use these shorthand notations in assignments/exams

	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



F = _____

Canonical Sums

- Given a T.T., use the minterms where F=1 and SUM them together
  - (Σ = SUM or OR of all the minterms)

	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

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

Canonical Sum:

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

List the variables in the _____ they would appear in the truth table and that you'd use to find the decimal values

List the minterms where F is 1, and just list their decimal number equivalent

Canonical Products

- Given a T.T., AND together all the maxterms where F = 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	1	0	0
M_7	1	1	1	1

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

Canonical Product:

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

List the variables in the order they would appear in the truth table and that you'd use to find the decimal values

List the maxterms where F is 0, and just list their decimal number equivalent

Canonical Sums & Products

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

Canonical Sum:

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

$F = m_2 + m_3 + m_5 + m_7$

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

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

Definitions

- **Minterm**: A **product** term where all the input variables of a function appear as exactly one literal

f(x,y,z)	Yes (m_i) / No	g(w,x,y,z)	Yes (m_i) / No
x•y			Yes, m4
x'•y•z'		w•x'•z'	No
x+y'•z'		w•x'•y•z	Yes, m11
(x•y'•z)'			Yes, m6

- **Maxterm**: A **sum** term where each input variable of a function appears as exactly one literal

f(a,b,c)	Yes (M_i) / No	g(v,w,x,y,z)	Yes (M_i) / No
a+b'+c			Yes, M12
a+b'		v+z	No
a'•(b'+c)		v'+w+x+y'+z'	Yes, M19
(a'+b+c)'			