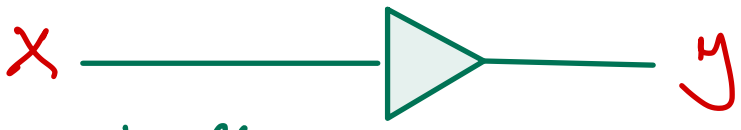
Other Gate Types

* Buffers and Tristate Buffers
* NAND and NOR gates
* XOR and XNOR gates
* Parity Generation & Checking

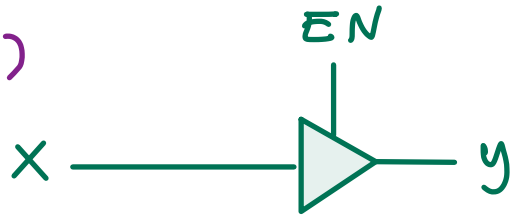
Buffers and Tristate Buffers

* Buffers are used to amplify signals
* We need signal amplification to enable it to drive multiple gates
* Buffer symbol is similar to NOT, but without the bubble

x	y
0	0
1	1

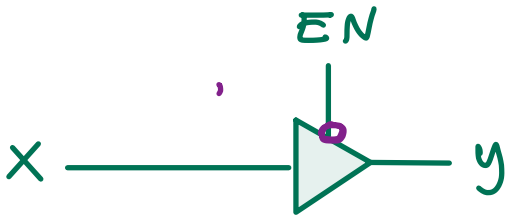


* Mathematically, $y = x$, but y can drive larger load
* Tristate buffer:
  - Buffer with an additional input called enable (EN)
  - Symbol



  - Best described using function table

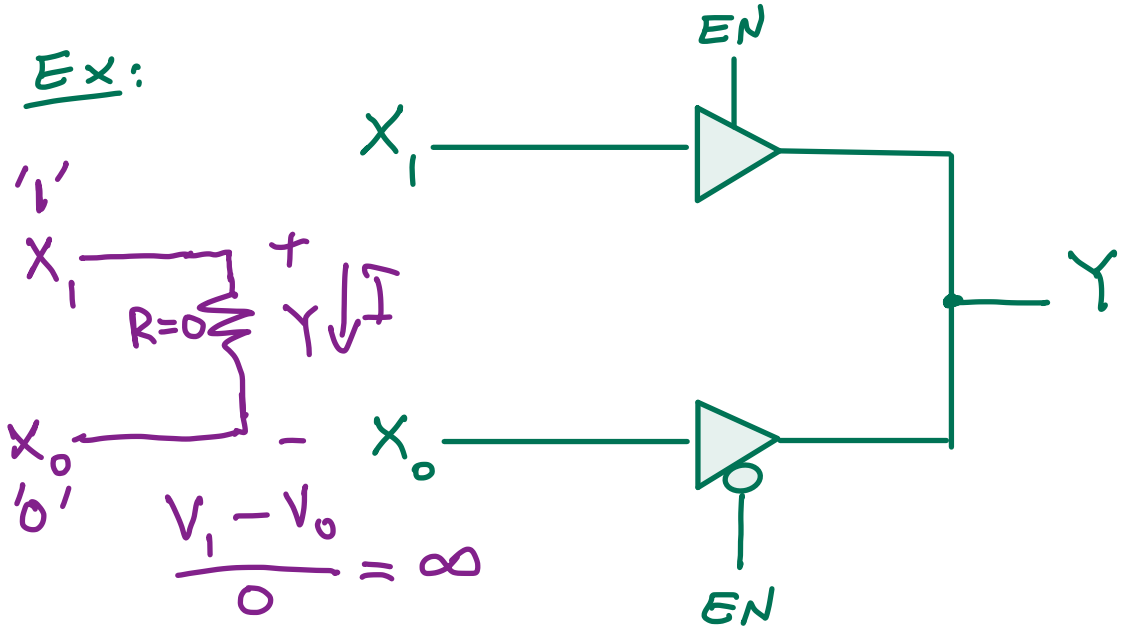
EN	y
0	Hi-Z
1	X



EN	y
0	X
1	Hi-Z

* Tristate buffers can be used when an output line is used by more than one driver, in which only one driver is enabled and rest are disabled, e.g. data bus in computers

Ex:



EN	Y
0	X_0
1	X_1

* Top tristate buffer will bypass x_1 to y if EN=1

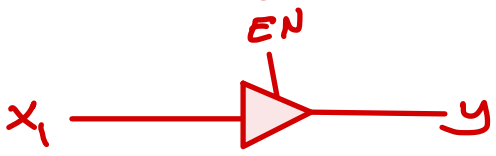
Bottom " " " " x_0 to y if EN=0
{Note the bubble}

EN	Y
0	X_0
1	X_1

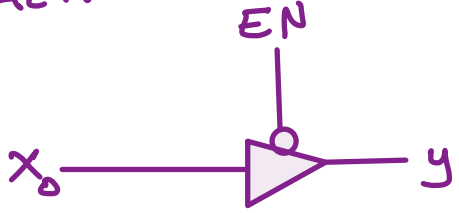
$$Y = \begin{cases} x_0 & \text{iff } EN == 0 \\ x_1 & \text{iff } EN == 1 \end{cases}$$

EN lets us select between x_0 and x_1 to connect to y

Active-high control



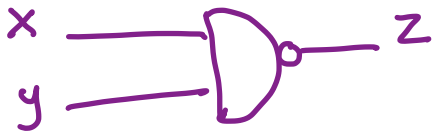
Active-low control



NAND and NOR Gates

NAND

* This gate is composed of AND followed by NOT



{AND-NOT}

* The bubble at the output of the gate corresponds to the inverter part

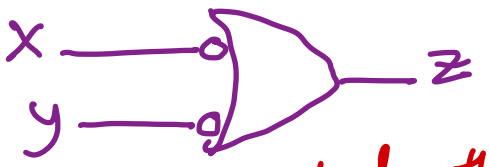
* Algebraically,

$$z = (xy)'$$

By DeMorgan's law,

$$z = \bar{x} + \bar{y}$$

⇒ Another symbol for NAND



{NOT-OR}

* As if we pushed the bubble of the {AND-NOT} form into the inputs, and at the same time, the AND turns into OR

* NAND is commutative

$$(xy)' = (yx)'$$

* NAND is NOT associative

$$((xy)' \cdot z)' \neq (x \cdot (yz)')'$$

⇒ Good exercise to show this using cannonical forms (or truth table)

* NAND is universal

Def: A universal gate is a gate that can be used to implement any Boolean function.

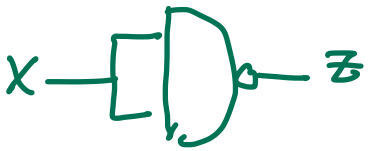
* We can easily prove universality by showing that the gate can implement all logic operations: NOT, AND, and OR.

*NAND is universal because we can implement AND, OR, and NOT using NAND gate

NOT

$Z = \overline{X}$

$\Rightarrow$

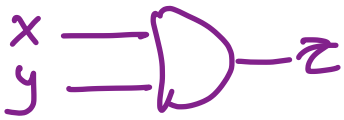


$\downarrow$

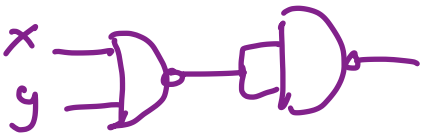
$Z = (xx)'$

$= \overline{x}$

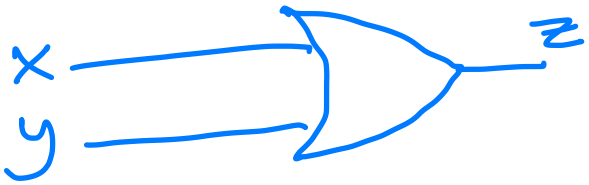
AND



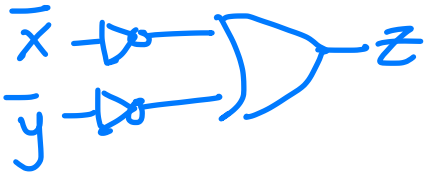
$\Rightarrow$



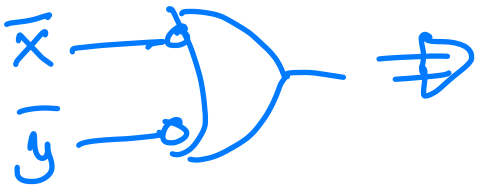
OR



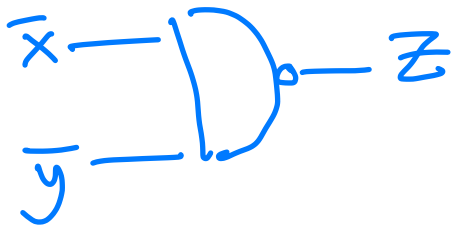
$\Rightarrow$



$\Rightarrow$



$\Rightarrow$



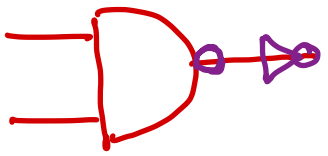
CONCLUSION

We can implement any logic circuit using NAND gates only

Steps for converting circuits into NAND

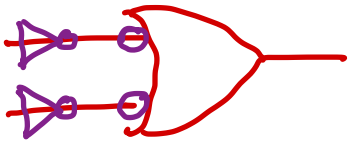
1. No matter what we do, we MUST ENSURE that we DO NOT CHANGE THE FUNCTIONALITY OF THE CIRCUIT.

2. For AND gates, just put a bubble followed by inverter at gate's output



{Func. still the same}

3. For OR gates, put bubble preceded by inverter at every input

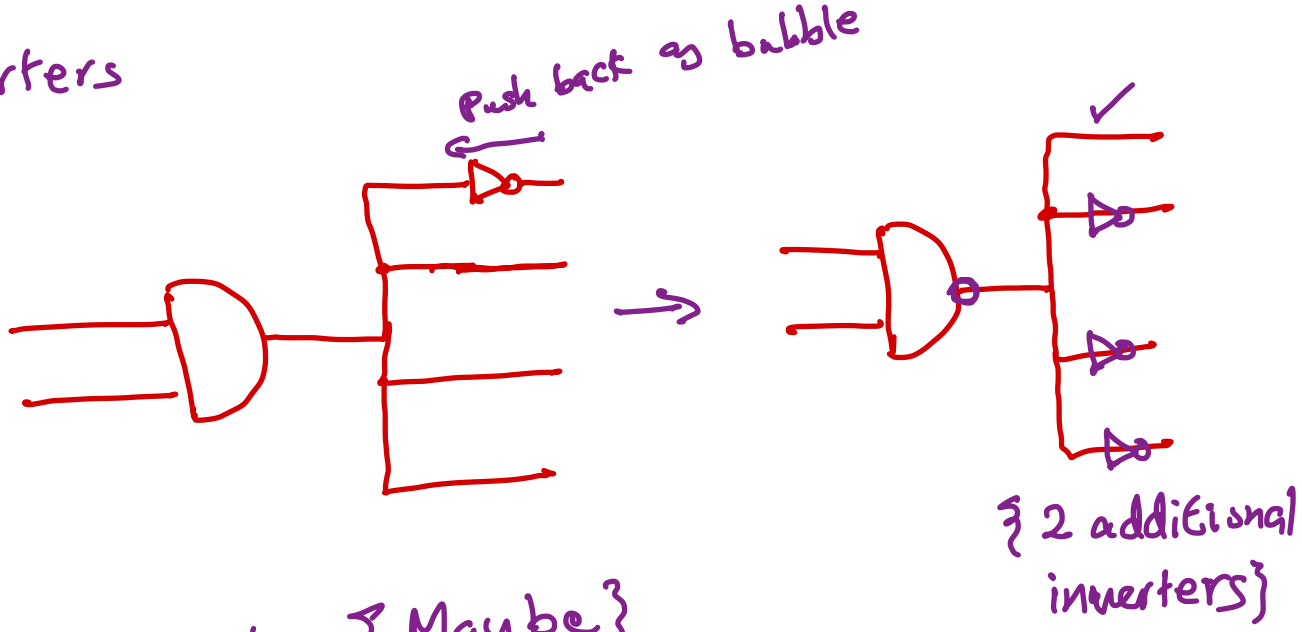


{≡ still OR gate}

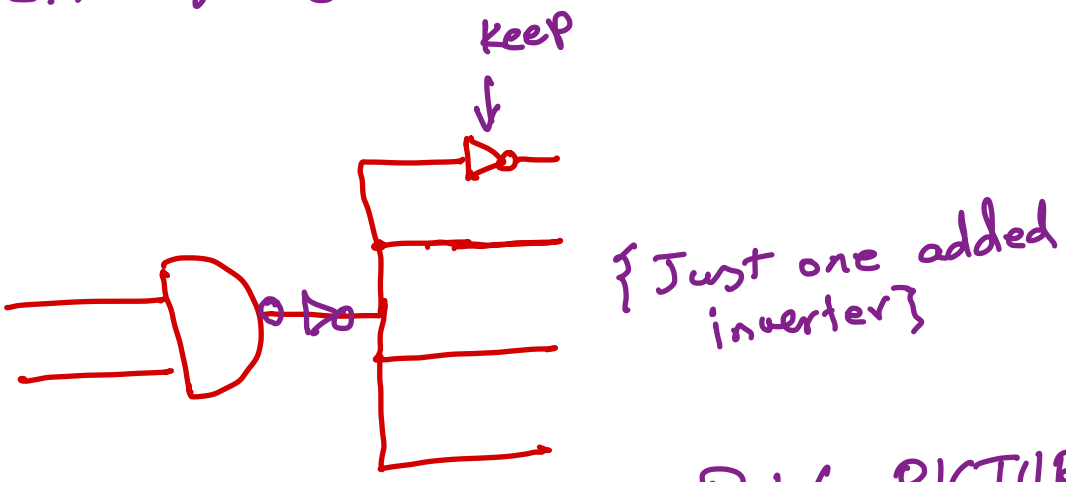
4. In ≡, if an inverter is already at output of AND, then just use it to convert to NAND

Same goes for ≡ also.

5. For fanout branches, make sure the option you choose results in minimum number of inverters

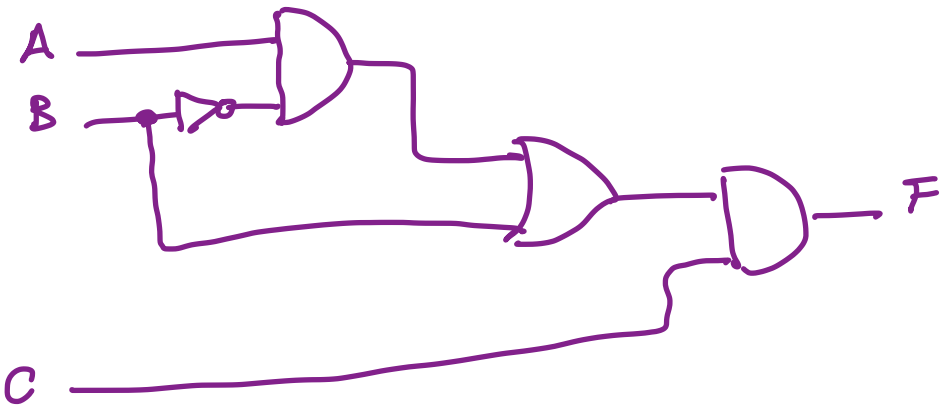


OR better {Maybe}

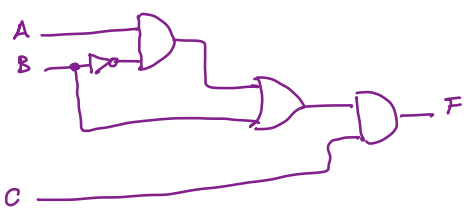


* But we must see the BIG PICTURE as the extra inverters may prove useful in simplifying the overall circuit

Ex Without simplification, show a NAND implementation of



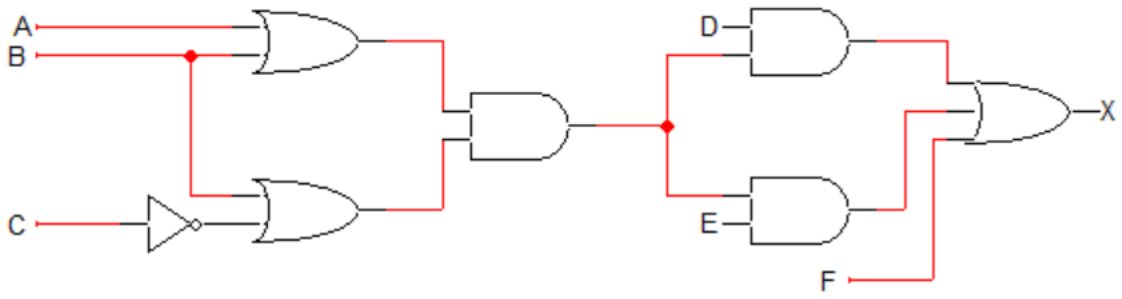
Use the minimum # of NAND gates.



Major 2 – Q 2 – 191

Question 2. **(10 points)**

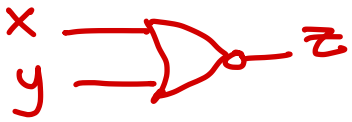
a) **(4 points)** Given the following circuit diagram with AND/OR/NOT gates:



Redraw the above circuit diagram **using only NAND gates** and a minimum number of inverters (NOT gates). You may insert inverters only when necessary.

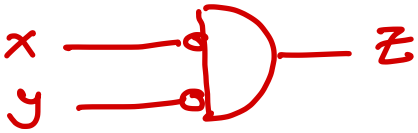
NOR

* OR followed by NOT



{OR-NOT}

≡

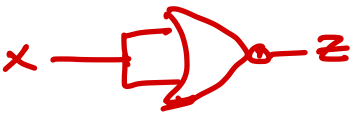


{NOT-AND}

* Commutative, not associative, and universal

Proof of universality of NOR:

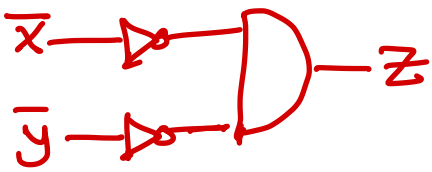
NOT 

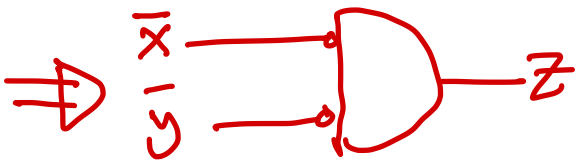
⇒ 

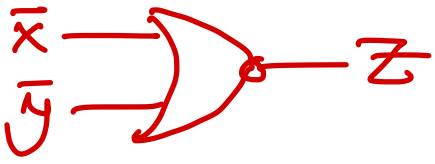
OR 

⇒ 

AND 

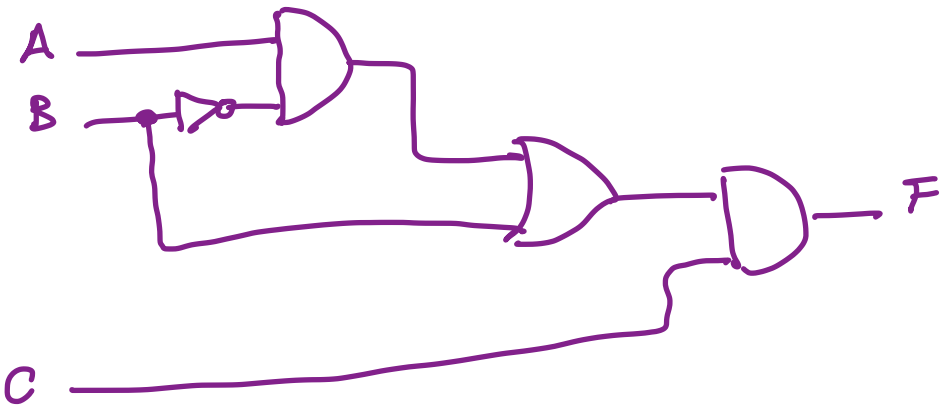
⇒ 

⇒ 

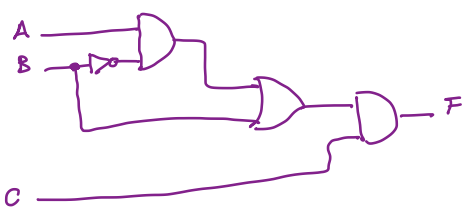
⇒ 

Ex Without simplification, show a NOR implementation

of



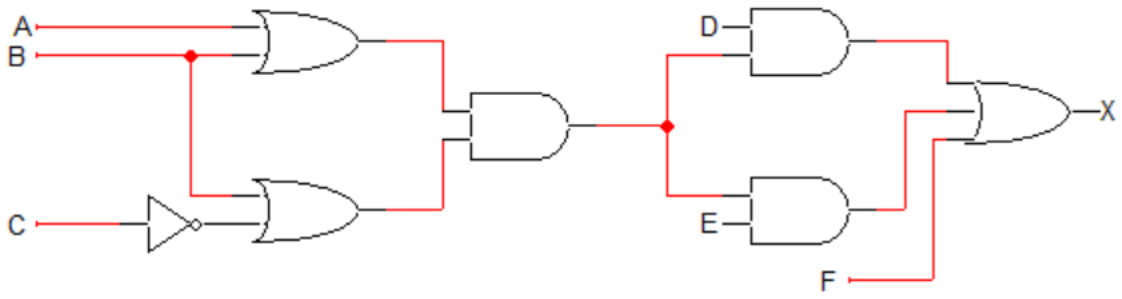
Use the minimum # of NOR gates.



Major 2 – Q 2 – 191

Question 2. **(10 points)**

a) **(4 points)** Given the following circuit diagram with AND/OR/NOT gates:

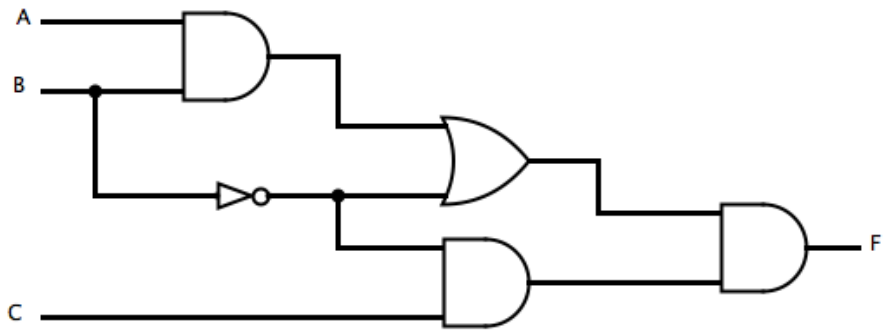


Redraw the above circuit diagram **using only ~~NAND~~ NOR gates** and a <u>minimum</u> number of inverters (NOT gates). You may insert inverters only when necessary.

COE 202–03 Digial Logic Design **Name:** _____
191
Quiz 4
Nov 21, 2019
Time Limit: 20 Minutes **ID:** _____

This quiz contains 2 pages and 2 questions. Calculators, phones, tablets, and all other technologies are not allowed during the quiz.

1. (5 points) Consider the following logic circuit



Assuming the availability of input signals in true and complemented forms, implement this circuit using 2–input NAND gates only. Do not manipulate the circuit in any way.

XOR and XNOR Gates

* The XOR and XNOR gates are composite gates

* TTs:

a	b	XOR $a \oplus b$	XNOR $(a \oplus b)'$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

* In terms of primitive gates:

{XOR} $a \oplus b = \bar{a}b + a\bar{b}$ {SoM}

This form is minimal already {Verify}

{XNOR} $(a \oplus b)' = \bar{a}\bar{b} + ab$ {SoM}

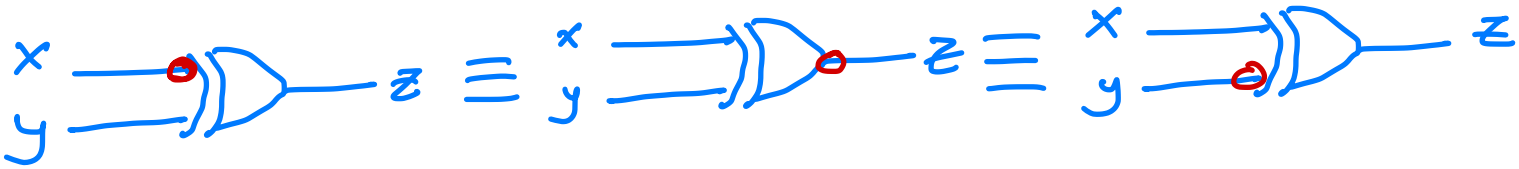
Minimal too {Verify}

We can easily show that XOR and XNOR operations are: commutative and associative, i.e.,

$x \oplus y = y \oplus x$ and $(x \oplus y) \oplus z = x \oplus (y \oplus z)$

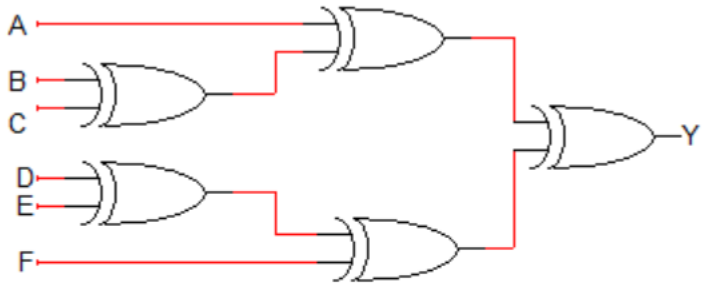
* Properties of XOR operation :

1. $x \oplus 0 = x$
2. $x \oplus 1 = \bar{x}$
3. $x \oplus \bar{x} = 1$
4. $x \oplus x = 0$
5. $x \oplus \bar{y} = \bar{x} \oplus y = (x \oplus y)'$



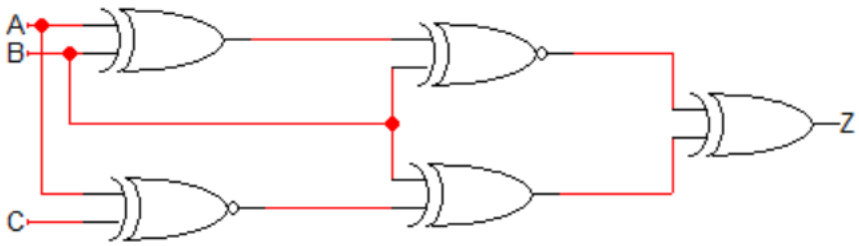
Major 2 – Q 2 – 191

b) (3 points) Given the following circuit diagram with 2-input XOR gates:



Redraw the above circuit using **only 2-input XNOR gates**. Minimize the number of 2-input XNOR gates used.

c) (3 points) Reimplement the circuit given below using **minimum number of 2-input XOR gates**:



2. (5 points) Given the following Boolean equation is true:

$$A \oplus B \oplus AC' = B \oplus C \tag{1}$$

Use the properties of the XOR gate to manipulate Equation 1, and then use logic reasoning to imply the value of the term $A + C'$, i.e., the value of A ORed with C'. *Hint:* Note that the final answer of $A + C'$ can be either 0, 1, or unknown. However, in order to get credit for this question, you must show all your steps.

Odd & Even Functions

Def

An odd function equals 1 iff the number of 1s in its inputs is odd.

Therefore, odd functions can be used to generate even parity bit.

Def

An even function equals 1 iff the number of 1s in its inputs is even.

Therefore, even functions can be used to generate odd parity bit.

To implement Odd function → XOR gate

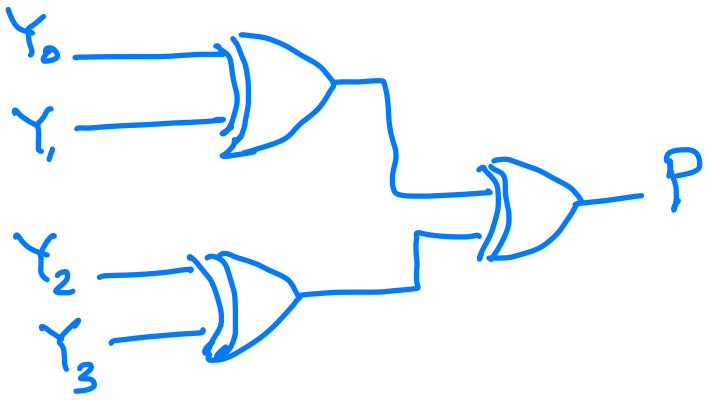
To implement Even function → XNOR gate

*An even parity generator is implemented using XOR.

Ex Show an even parity generator for a BCD 8421 code.

Input

output



⇒ Code generator output is

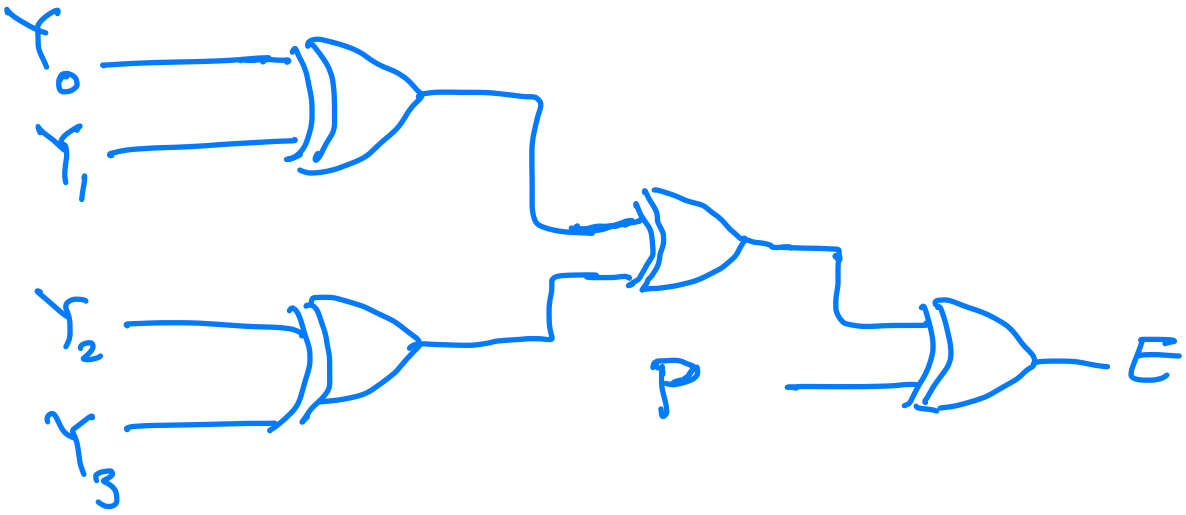
$\{P, Y_3, Y_2, Y_1, Y_0\}$

Q How to check for the correctness of the parity code?

A We use a parity checker.

We know we generated an even parity code, therefore we can generate an error signal E such that E=1 iff the code contains an odd number of 1s ⟹ Odd Function ⇒ XOR

Parity checker {even}



Summary

* for even parity generation and checking
  → use odd function → XOR

* for odd parity generation and checking
  → use even function → XNOR

Important

* Never use more than 2-inputs for XOR and XNOR
* To implement 4-input XNOR

$$Z = (a \oplus b \oplus c \oplus d)' =$$
$$[(a \oplus b) \oplus (c \oplus d)]'$$