

CSCE/ELEG 2114: Digital Design

Gate Design

Reading: Chapter 2.7-2.8, 3.6, 4.5-4.6

Courtesy of Dr. Brown, Dr. Vranesic, Dr. Harris, Dr. Zhou and Dr. Hadimioglu

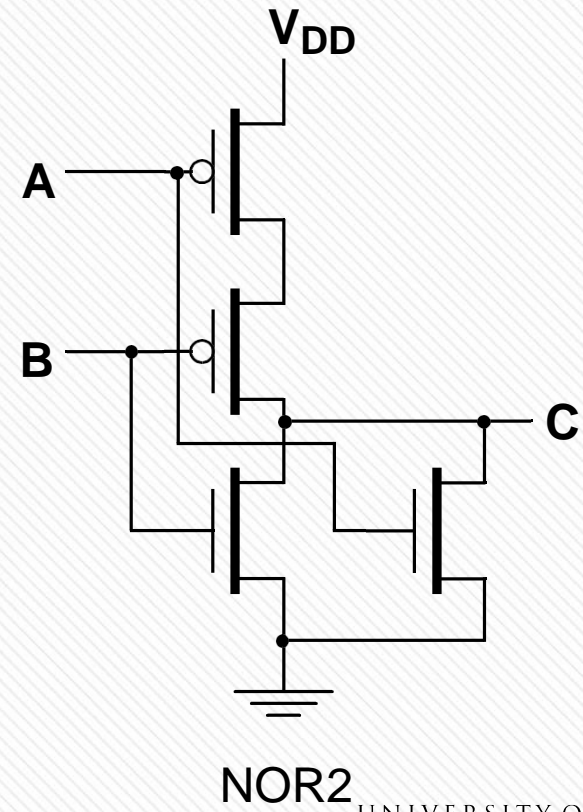
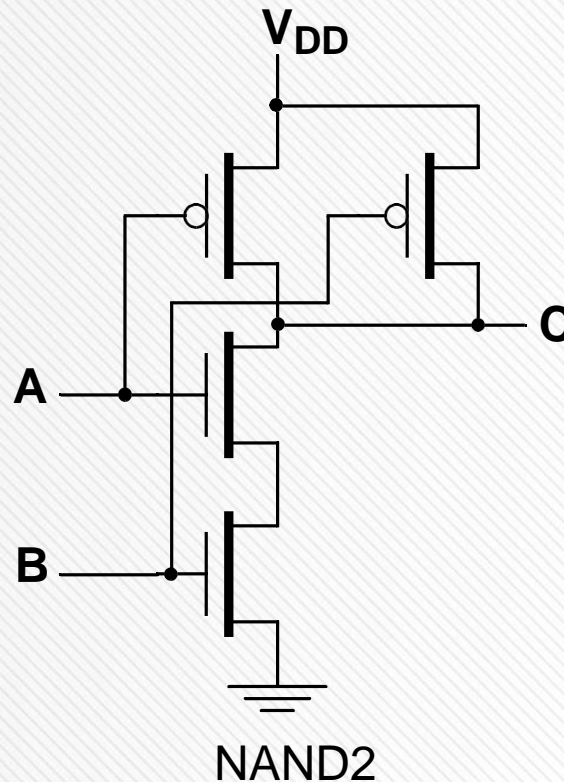
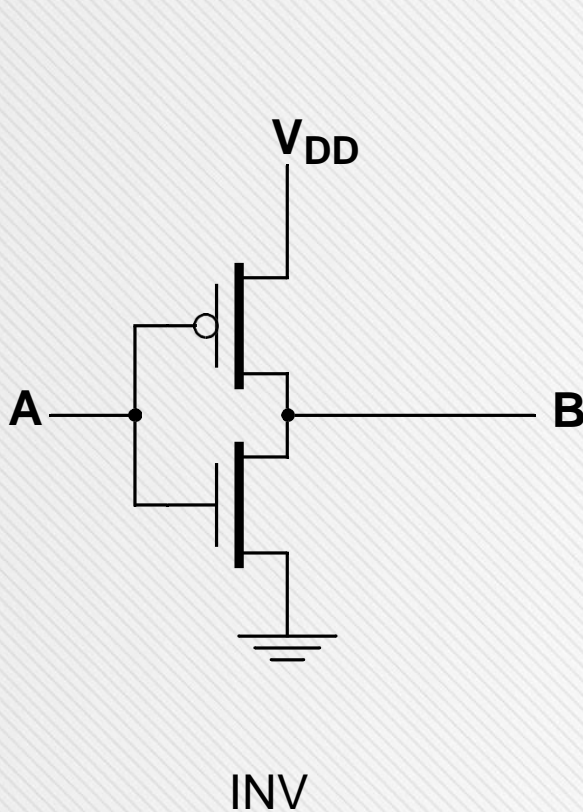


Basic Gates



□ We have learned INV, NAND2, NOR2 gates

- What about XOR/XNOR?
- What about any boolean function?



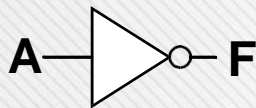


Inverters



□ Inverters can also be implemented with a NAND or NOR gate.

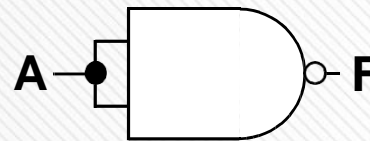
NOT



A	F
0	1
1	0

$$F = \bar{A}$$

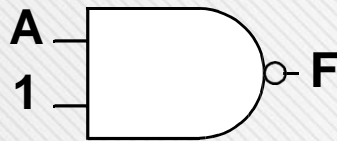
NAND



A	A	F
0	0	1
1	1	0

$$F = \overline{AA} = \bar{A}$$

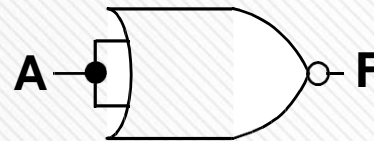
NAND



A	1	F
0	1	1
1	1	0

$$F = \overline{A \cdot 1} = \bar{A}$$

NOR



A	A	F
0	0	1
1	1	0

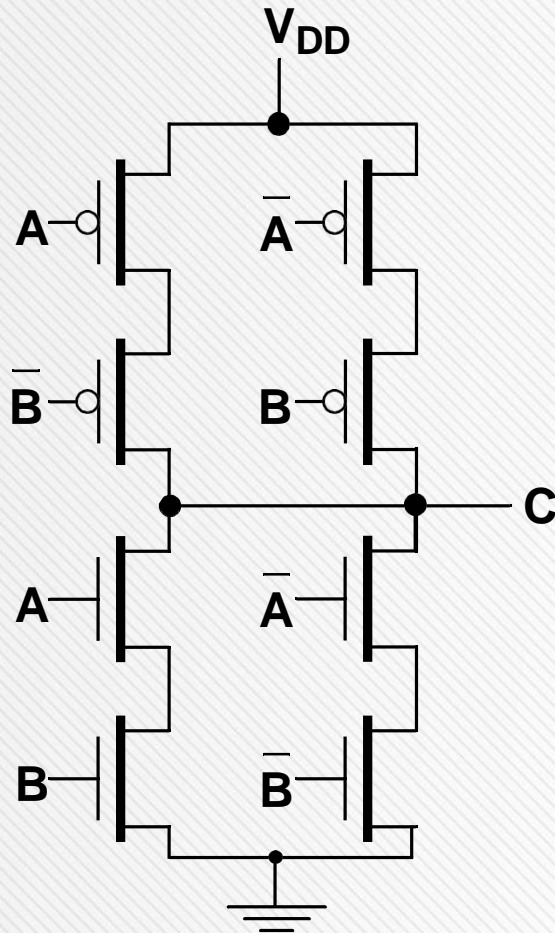
$$F = \overline{A + A} = \bar{A}$$



XOR Gate



□ 8T design



$$C = \overline{A}B + A\overline{B}$$

A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

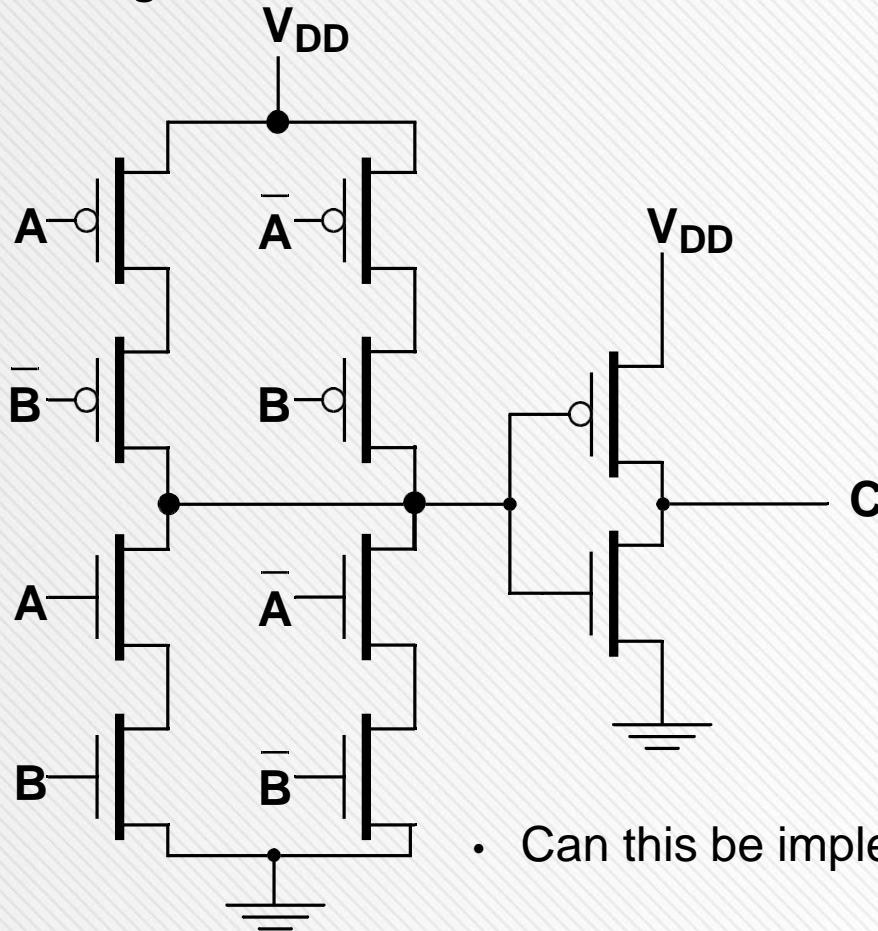


XNOR Gate



□ XNOR=XOR+INV

- 10T design



$$C = \overline{AB} + \overline{A\bar{B}}$$

A	B	C
0	0	1
0	1	0
1	0	0
1	1	1

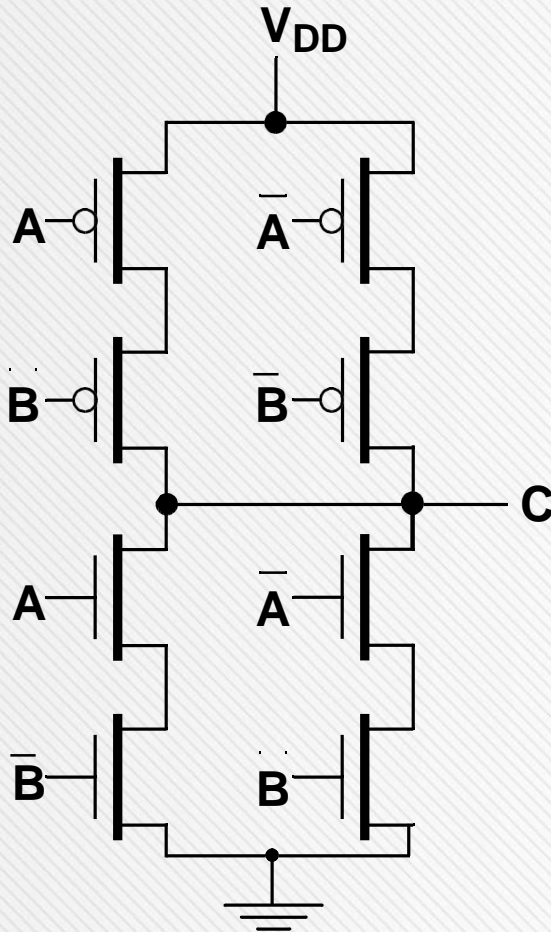
- Can this be implemented without the extra inverter?



XNOR Gate



□ 8T Design



$$C = AB + \overline{A}\overline{B}$$

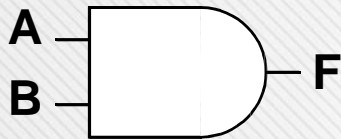
A	B	C
0	0	1
0	1	0
1	0	0
1	1	1



Non-Inverting Operators



AND



A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

$$F = AB$$

6 transistors

OR

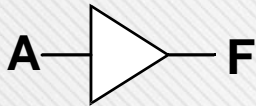


A	B	F
0	0	0
0	1	1
1	0	1
1	1	1

$$F = A + B$$

6 transistors

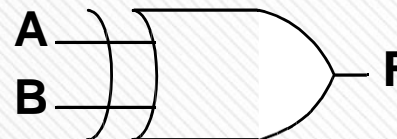
BUFFER



A	F
0	0
1	1

$$F = A$$

XOR



A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

$$F = A\bar{B} + \bar{A}B = A \oplus B$$

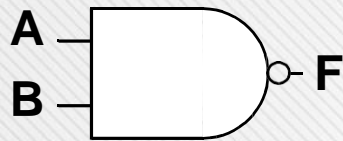
8 transistors



Inverting Operators



NAND

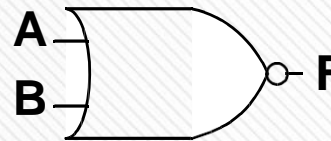


A	B	F
0	0	1
0	1	1
1	0	1
1	1	0

$$F = \overline{AB}$$

4 transistors

NOR

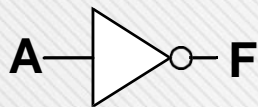


A	B	F
0	0	1
0	1	0
1	0	0
1	1	0

$$F = \overline{A + B}$$

4 transistors

NOT

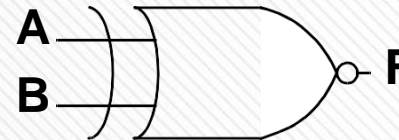


A	F
0	1
1	0

$$F = \overline{A}$$

2 transistors

XNOR



A	B	F
0	0	1
0	1	0
1	0	0
1	1	1

$$F = AB + \overline{A}\overline{B} = \overline{A \oplus B}$$

8 transistors



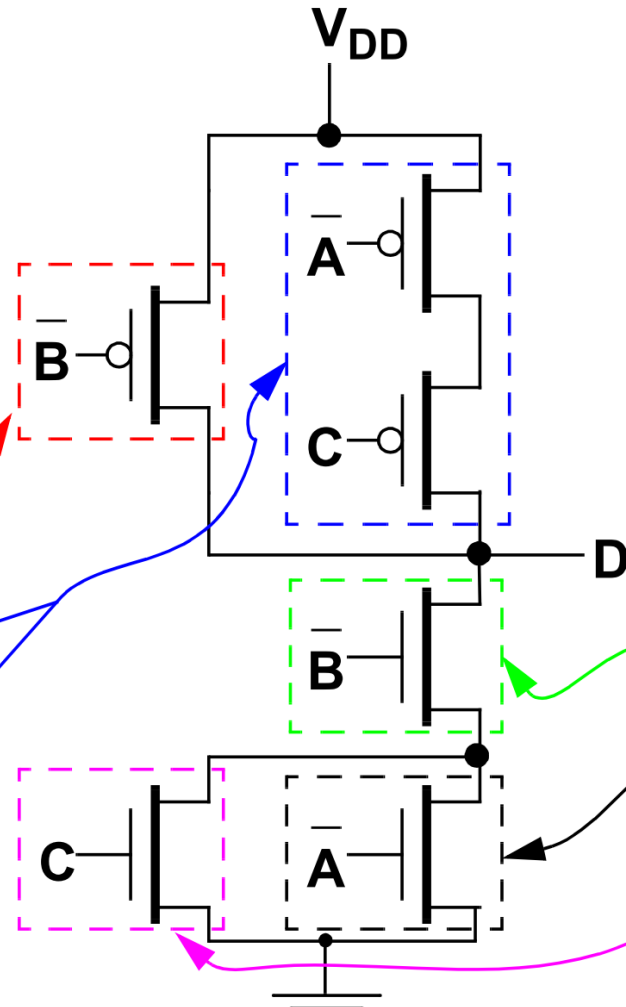
A Complex Three Input Gate



$$D = \overline{A}C + B$$

PULL-UP

A	B	C	D
0	0	0	Z
0	0	1	Z
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	Z
1	1	0	1
1	1	1	1



PULL-DOWN

A	B	C	D
0	0	0	0
0	0	1	0
0	1	0	Z
0	1	1	Z
1	0	0	Z
1	0	1	0
1	1	0	Z
1	1	1	Z



Boolean Functions to Transistors

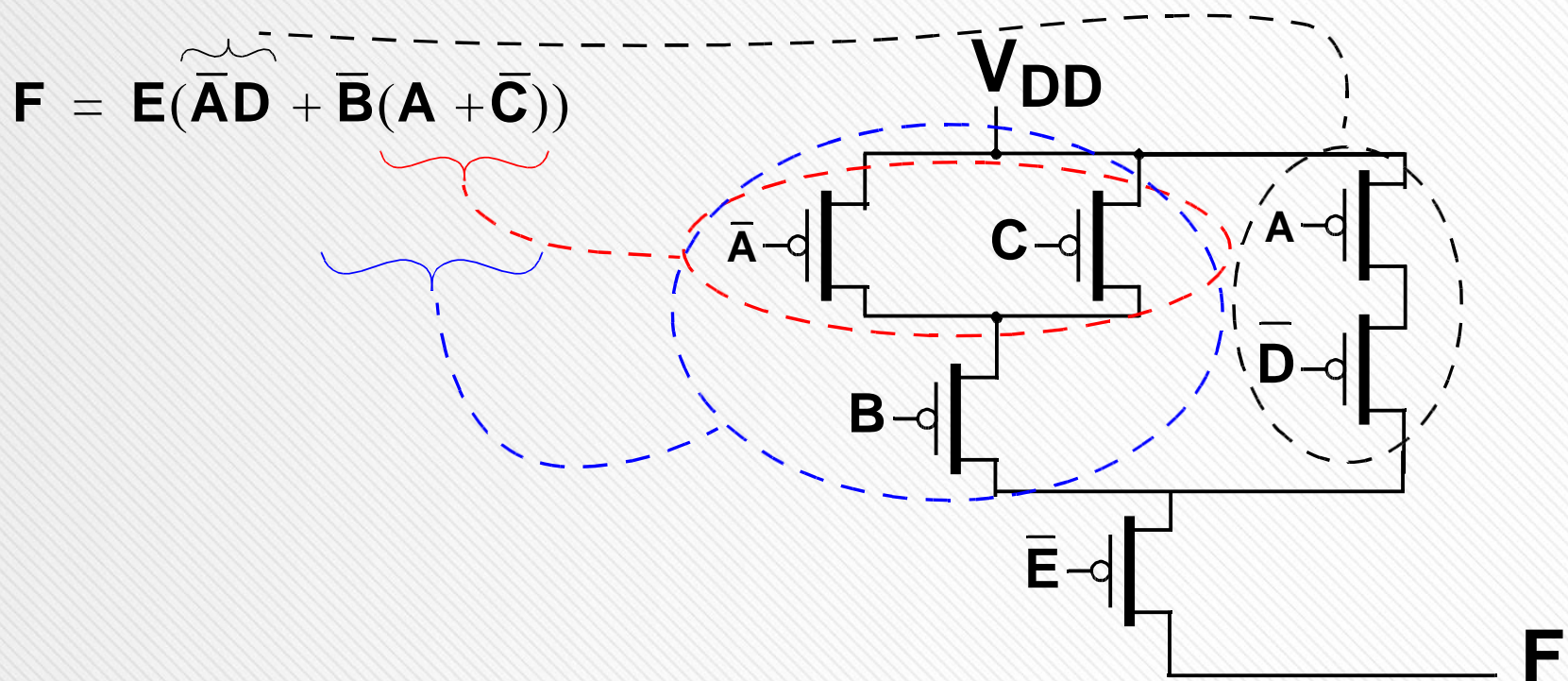


- ❑ **Most Boolean functions can be implemented using switches**
- ❑ **The basic rules are as follows**
 - **Pull-up section of switch network**
 - Use complements for all literals in expression
 - Use only pMOS devices
 - Form series network for an AND operation
 - Form parallel network for an OR operation
 - **Pull-down section of switch network**
 - Use complements for all literals in expression
 - Use only nMOS devices
 - Form parallel network for an AND operation
 - Form series network for an OR operation



Example Pull-Up

□ To implement the Boolean function given below, the following pull-up network could be designed.



- Notice how **AND** and **OR** become **series** and **parallel** circuits, respectively.

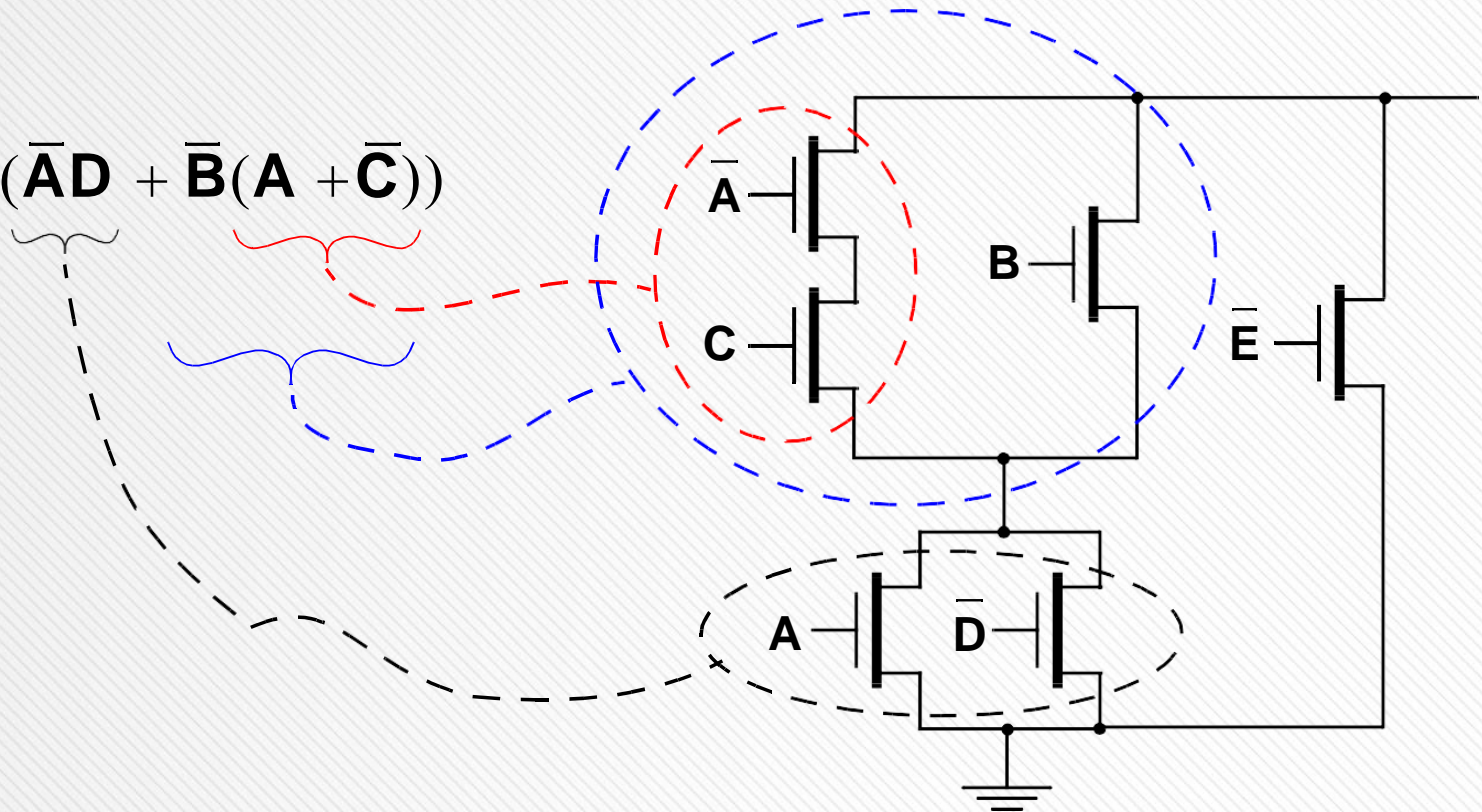


Example Pull-Down



- To complete the switch design, the pull-down section for the Boolean function must also be designed.

$$F = E(\bar{A}D + \bar{B}(A + \bar{C}))$$



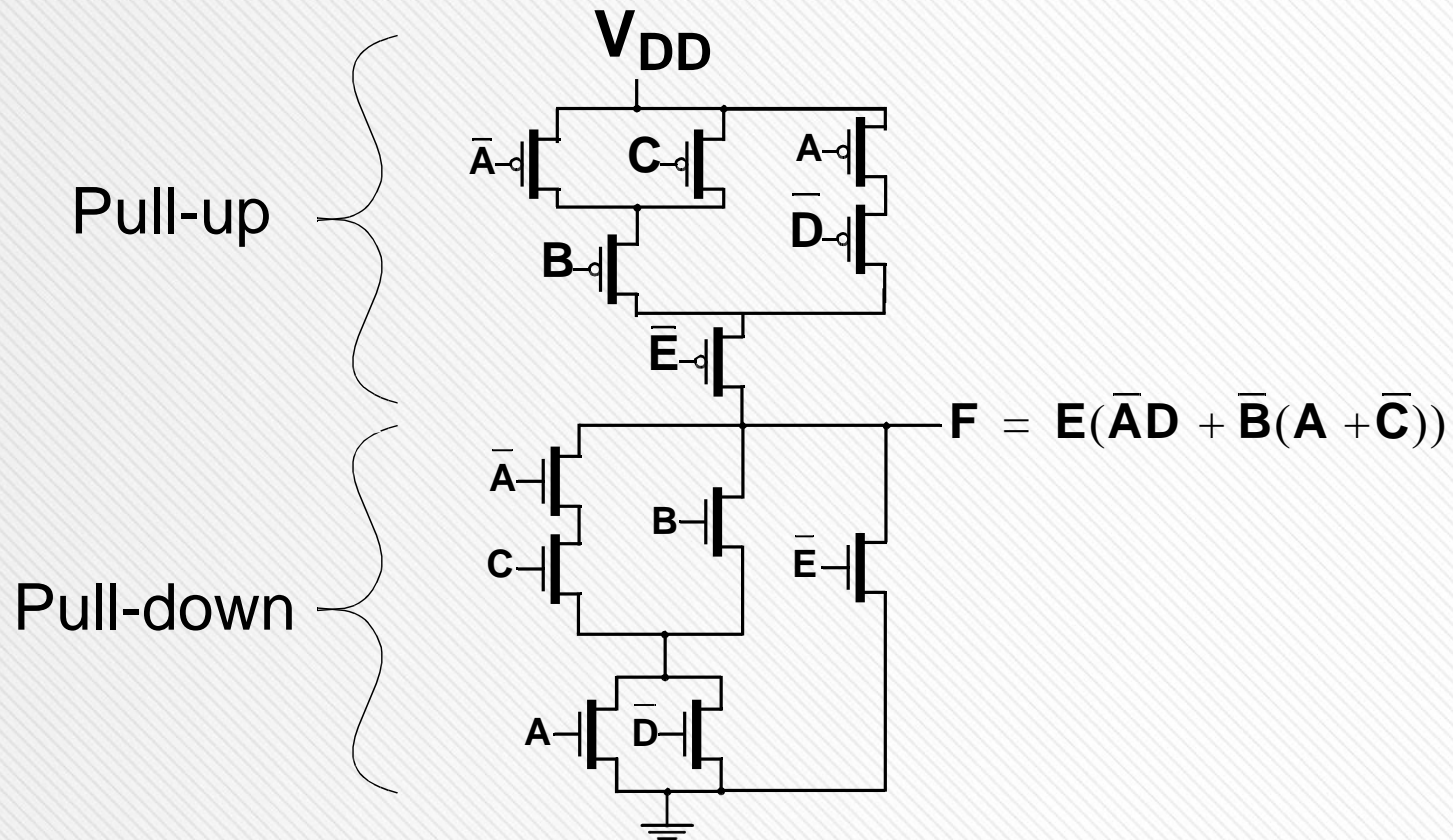
- Notice how **AND** and **OR** become **parallel** and **series** circuits, respectively.



Completed Example



- Putting the pull-up and pull-down pieces together gives the following CMOS switch implementation of the Boolean function.





Gate Networks



Gate network consists of

- Gates
- External inputs and outputs
- Connections

Gate inputs

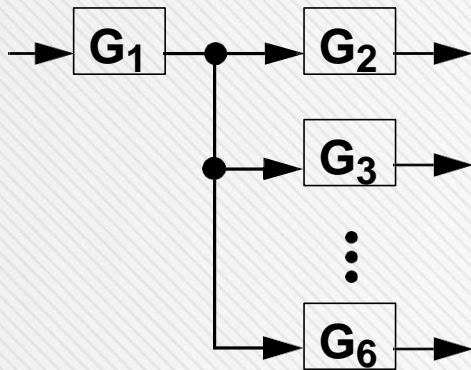
- Only one connection to input is allowed
 - Connected to constant value (0 or 1)
 - Connected to an external input
 - Connected to a gate output

Gate outputs

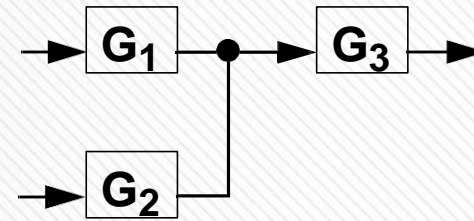
- Output load should not be greater than the fanout factor for the gate and technology being used.



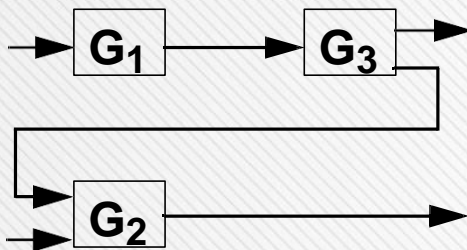
Valid/Invalid Networks



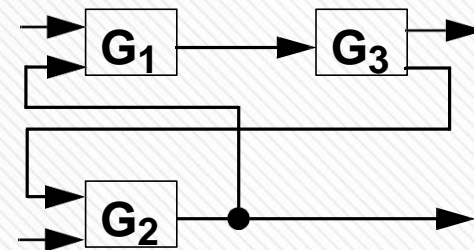
Valid or Invalid?



Valid or Invalid?



Valid or Invalid?



Valid or Invalid?



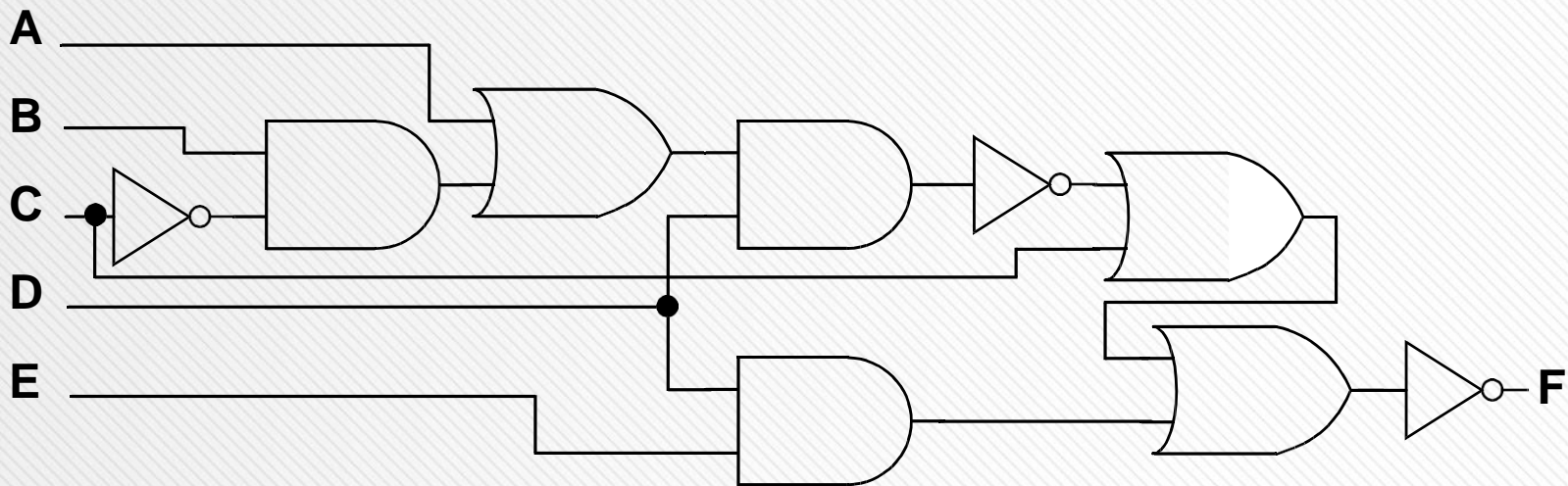
Boolean Functions to Gates



- Implement the following Boolean function using logic gates

$$F = \overline{\overline{(A + B\bar{C})D} + C + DE}$$

- Possible solution:



- $3 \times 6_{\text{AND}} + 3 \times 6_{\text{OR}} + 3 \times 2_{\text{NOT}} = 42$ transistors for CMOS technology.



Using Specific Gates



- ❑ **Because of various implementation reasons, it may be desired to use only specific sorts of logic gates in an implementation.**
 - For instance, many CMOS implementations use only NAND gates. Some implementations use on NOR gates.
 - This can be done in a number of manners. One is to rework the Boolean functions so that only the specific gates desired are used.
 - May reduce the physical number of transistors required if the appropriate types of gates are used.

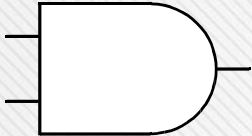
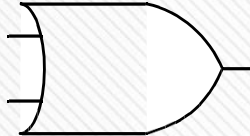

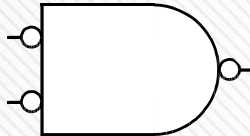

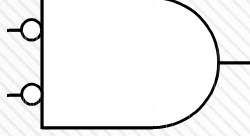
- ❑ **NAND and NOR are universal gates**
 - AND and OR need INVs to implement any function



Demorgan's Square



DeMorgan's Square

AND					OR																																		
	<table border="1"><thead><tr><th>A</th><th>B</th><th>F</th></tr></thead><tbody><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></tbody></table>	A	B	F	0	0	0	0	1	0	1	0	0	1	1	1					<table border="1"><thead><tr><th>A</th><th>B</th><th>F</th></tr></thead><tbody><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></tbody></table>	A	B	F	0	0	0	0	1	1	1	0	1	1	1	1			
A	B	F																																					
0	0	0																																					
0	1	0																																					
1	0	0																																					
1	1	1																																					
A	B	F																																					
0	0	0																																					
0	1	1																																					
1	0	1																																					
1	1	1																																					
																																							
	<table border="1"><thead><tr><th>A</th><th>B</th><th>F</th></tr></thead><tbody><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></tbody></table>	A	B	F	0	0	1	0	1	1	1	0	1	1	1	0					<table border="1"><thead><tr><th>A</th><th>B</th><th>F</th></tr></thead><tbody><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></tbody></table>	A	B	F	0	0	1	0	1	0	1	0	0	1	1	0			
A	B	F																																					
0	0	1																																					
0	1	1																																					
1	0	1																																					
1	1	0																																					
A	B	F																																					
0	0	1																																					
0	1	0																																					
1	0	0																																					
1	1	0																																					
																																							

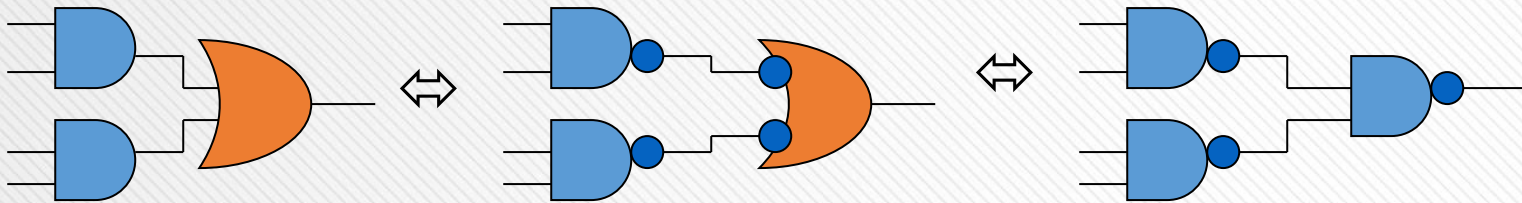


Two Level Synthesis (SOP)

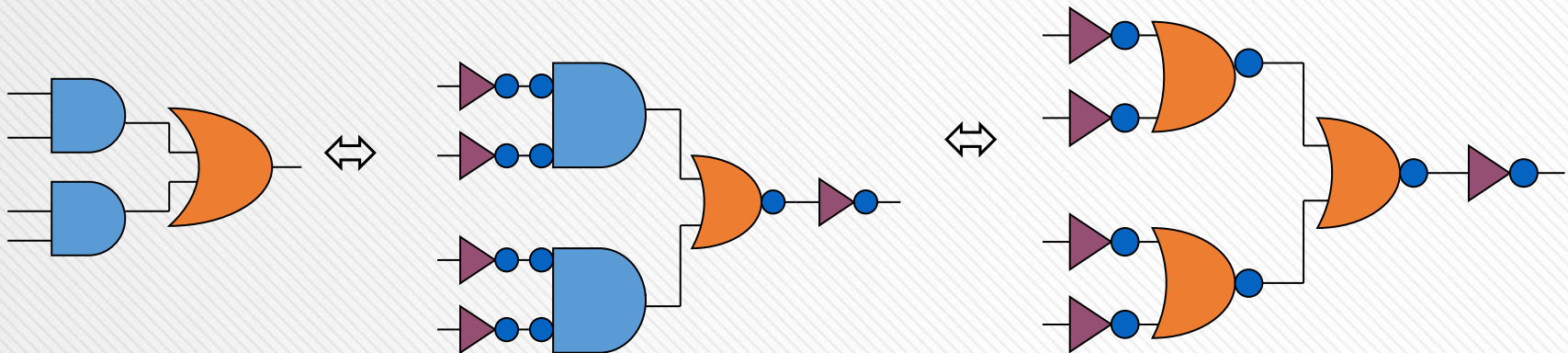


□ Sum of Products

- Using only NAND gates



- We create many bubbles with NOR gates



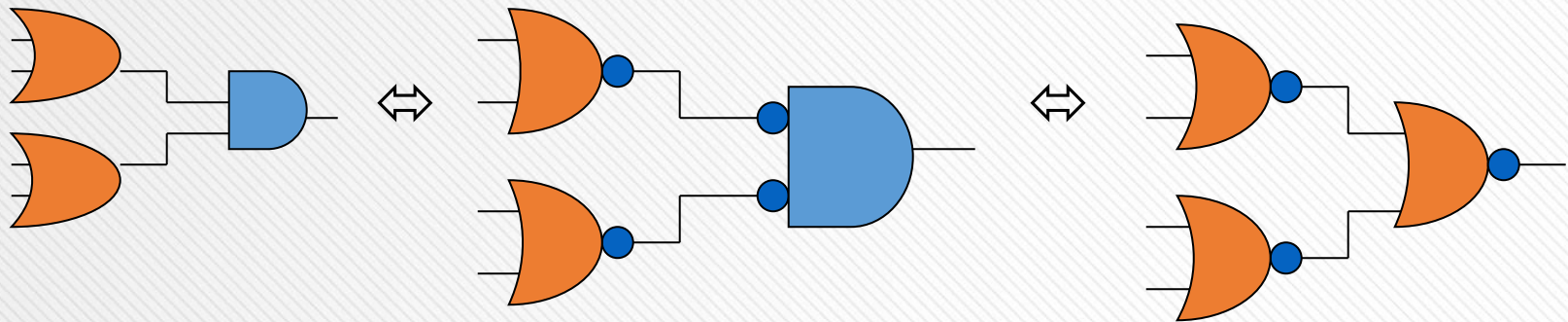


Two Level Synthesis (POS)



□ Product of Sums

- NOR gates only
- We will create many bubbles with NAND gates





Mixed Logic



- ❑ **Mixed logic is one approach that makes it easier to redesign a logic network to use desired types of gates.**

- ❑ **Mixed logic is also self-documenting**
 - This means that you can see what the original designer started with and see how the logic network was changed for the implementation.
 - The idea behind mixed logic is to diagram out the logic network from the Boolean equations given, and then make small changes to the logic network to achieve desired results for implementation.



Mixed Logic Procedure



- ❑ **Mixed logic is one approach that makes it easier to redesign a logic network to use desired types of gates**
- ❑ **The procedure for performing mixed logic conversions:**
 - **Draw the logic network for the given Boolean equation.**
 - Use only AND and OR gates.
 - Replace all complements with a bar (no bubbles or inverters yet!)
 - **Add complement bubbles and NOT gates within the network to appropriately convert logic gates to desired gate sets.**
 - **The rules in adding complement bubbles and NOT gates**
 - All bubbles must cancel each other out
 - Exactly one and only one bubble needed on each bar
 - **Extract the gates from the circuit, ignoring bars**



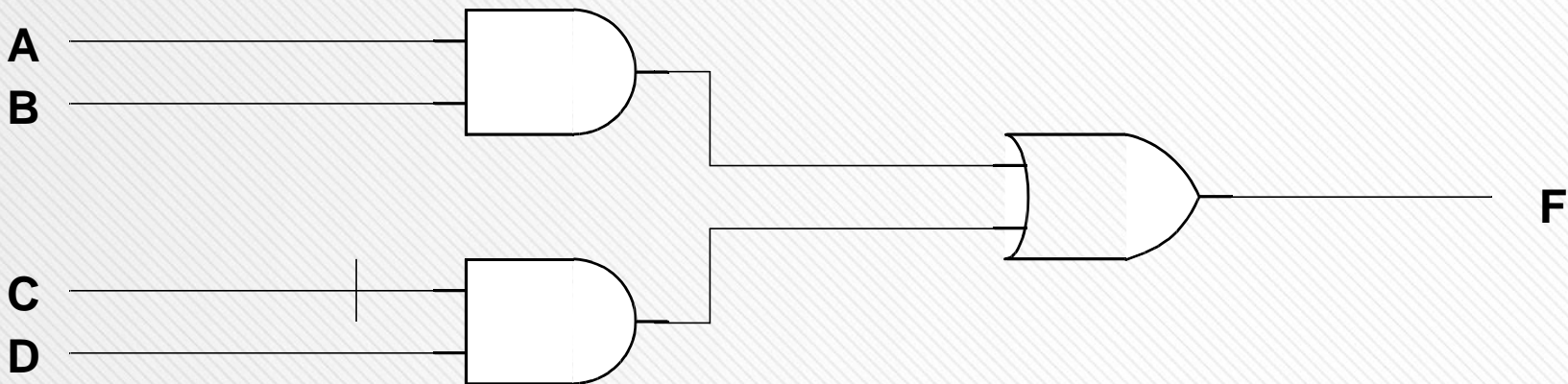
Example 1



- ❑ Implement the following Boolean function using only NAND gates first and then using only NOR gates.

$$F = AB + \bar{C}D$$

- ❑ Solution: Start by drawing the logic network for the Boolean function with the complements as bars.

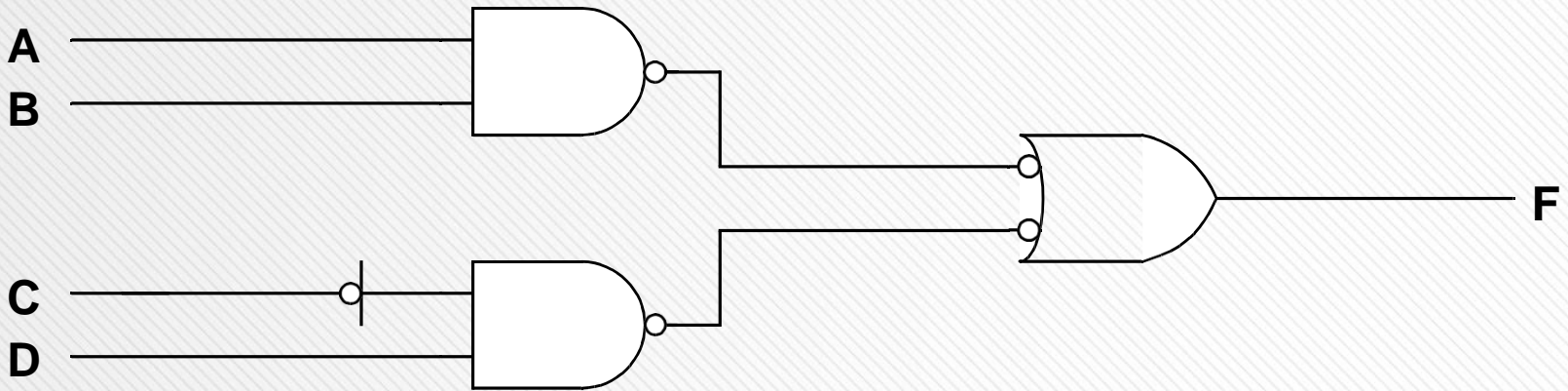




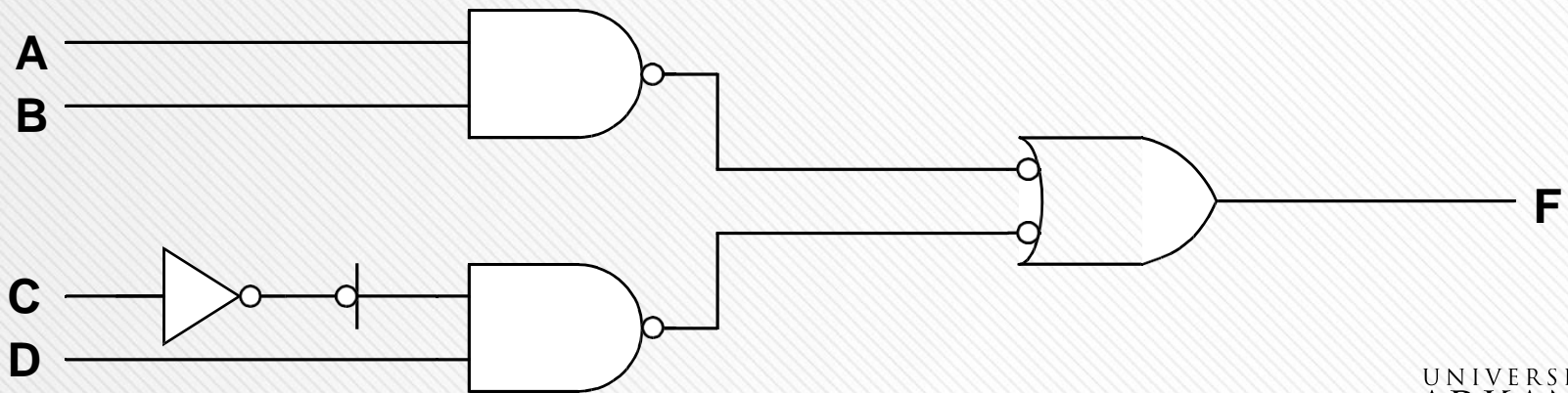
Example 1 using NAND



Step 2: Add bubbles to form NAND gates



Step 3: Add invertors to cancel out bubbles



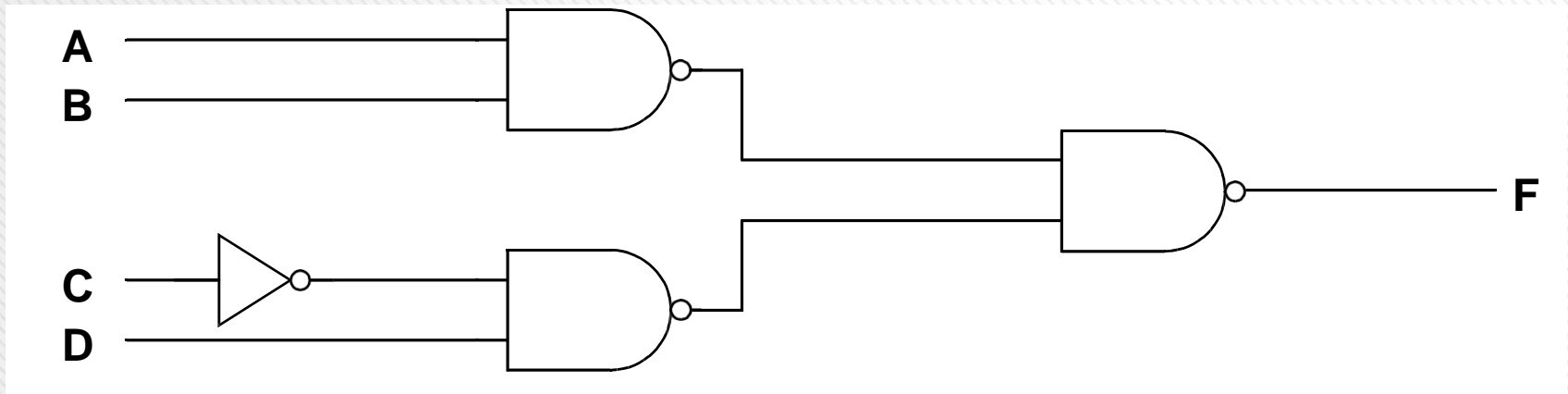


Example 1 using NAND



□ **Solution: This logic network now only uses NAND gates and INVs**

$$F(A, B, C, D, E) = \overline{\overline{AB}} \overline{\overline{CD}}$$

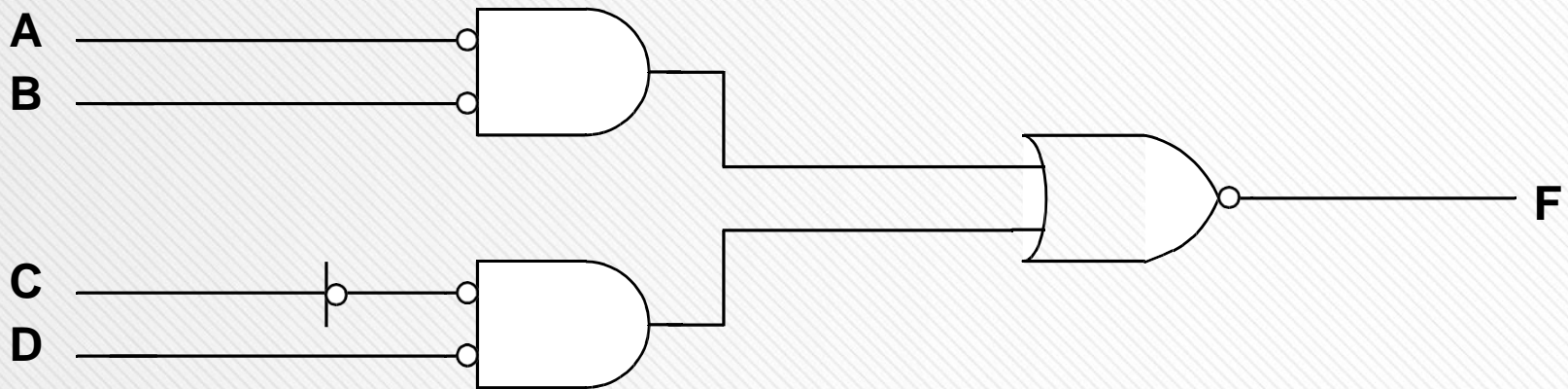




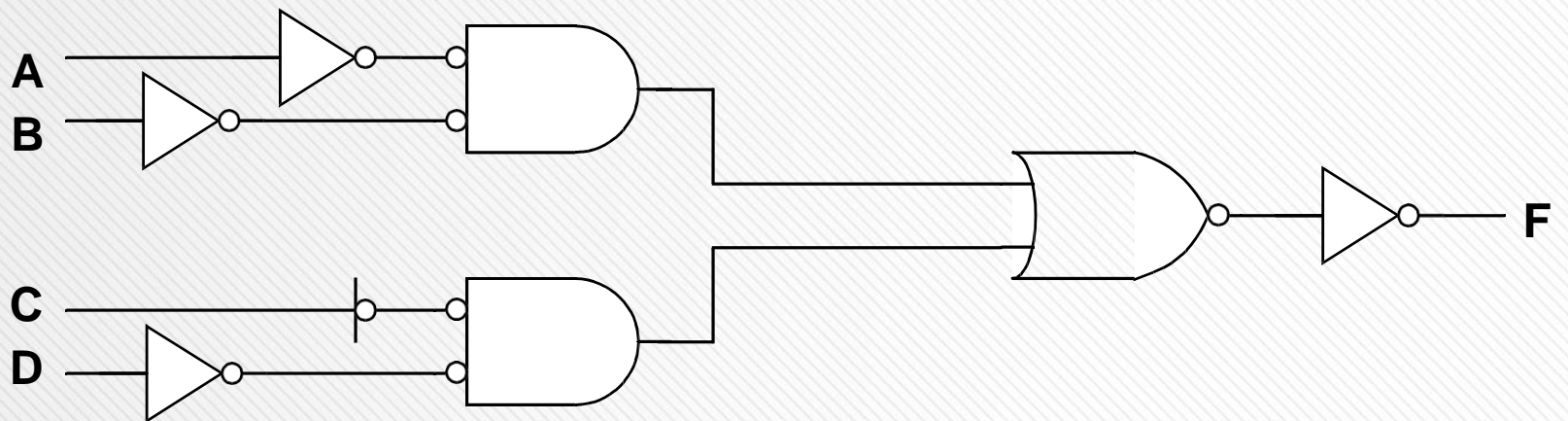
Example 1 using NOR



□ **Step 2: Add bubbles to form NOR gates (step 1 is the same)**



□ **Step 3: Add invertors to cancel out bubbles**



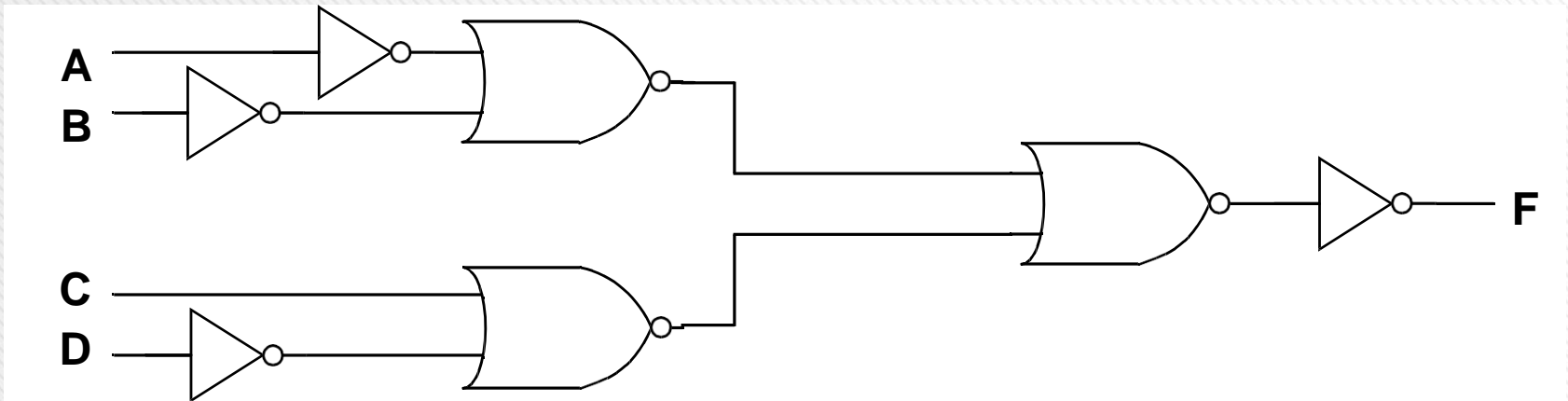


Example 1 using NOR



□ **Solution: This logic network now only uses NOR gates and INVs**

$$F(A, B, C, D, E) = \overline{\overline{\overline{\overline{\overline{A} + \overline{\overline{\overline{\overline{B} + \overline{\overline{\overline{\overline{C} + \overline{\overline{\overline{\overline{D}}}}}}}}}}}}}}}}$$





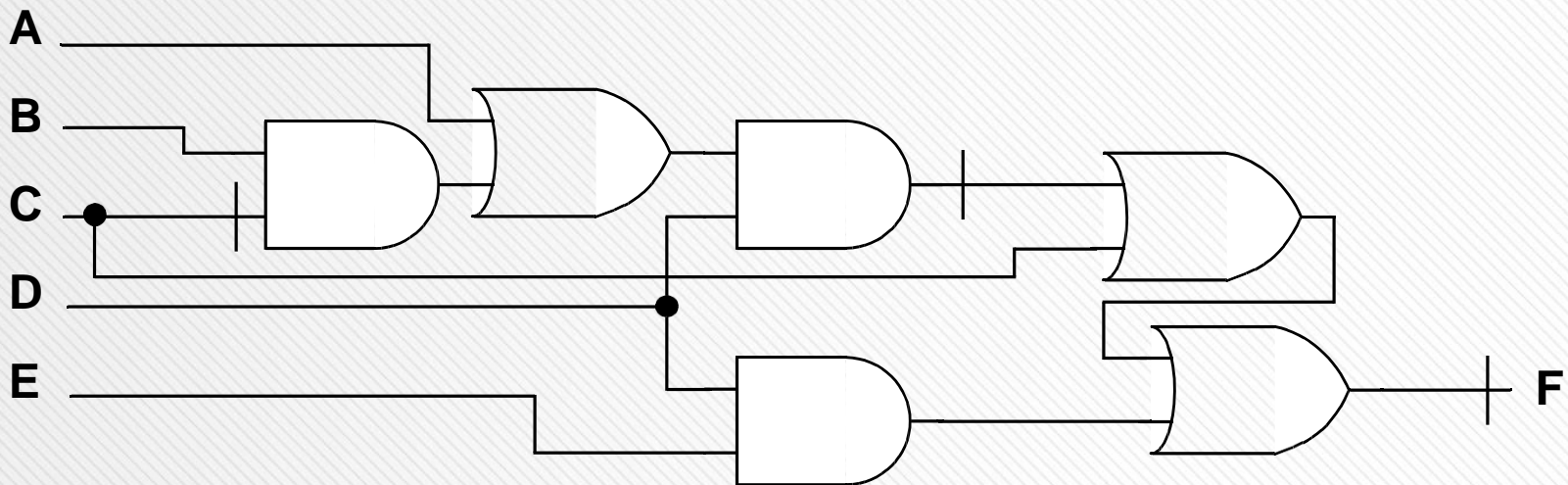
Example 2



□ Implement the following Boolean function using NAND gates

$$F = \overline{\overline{((A + B\bar{C})D) + C + DE}}$$

- Step1: Start by drawing the logic network for the Boolean function with the complements as bars.

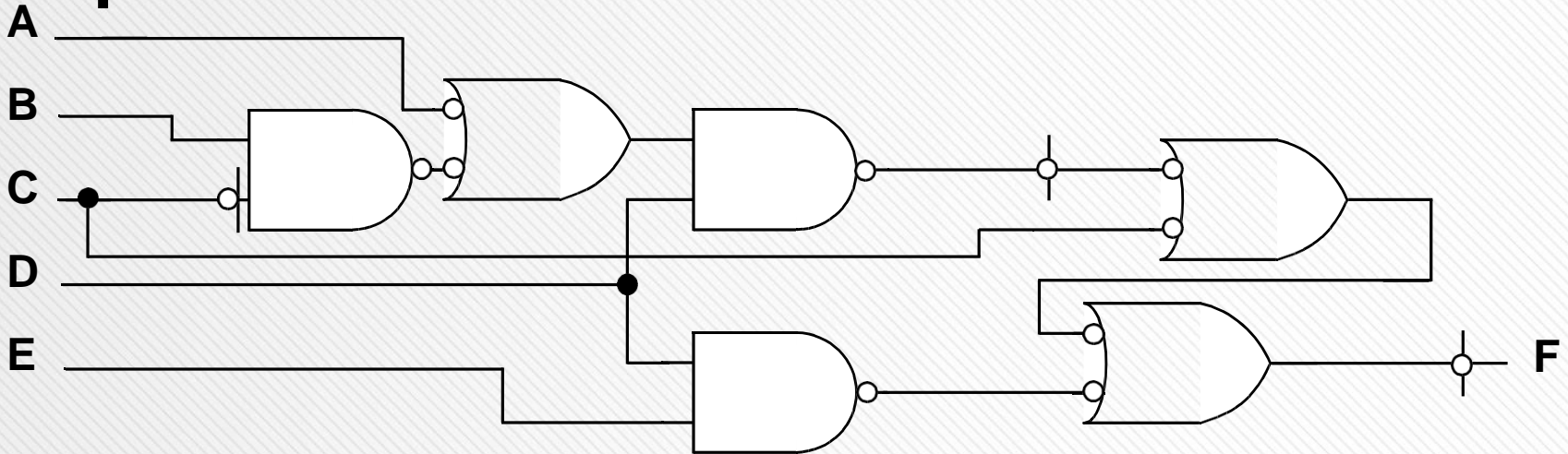




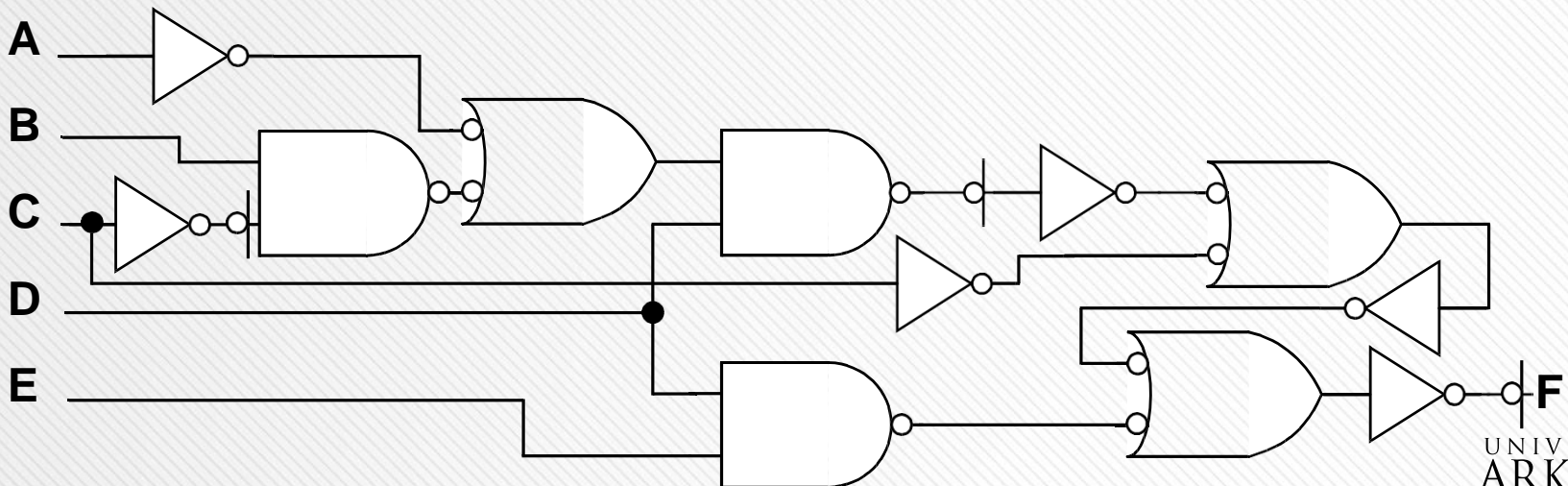
Example 2



Step 2: Add bubbles



Step 3: Add inverters to balance bubbles





Example 2



□ **Step4: This logic network now only uses NAND gates and INVs**

