

# **CHAPTER II**

## **SWITCH NETWORKS AND SWITCH DESIGN**

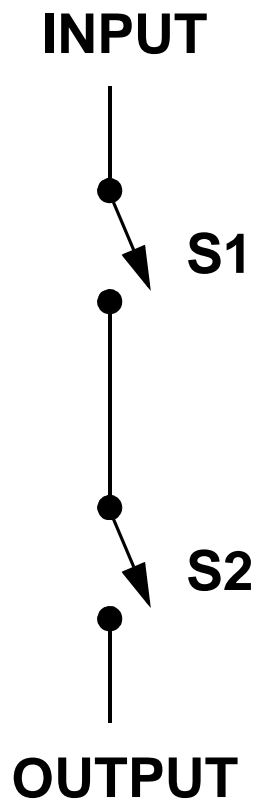
- Simplest structure in a computing system is a switch

**IDEAL SWITCH**



- Path exists between INPUT and OUTPUT if Switch is CLOSED or ON
- Path does not exist between INPUT and OUTPUT if SWITCH is OPEN or OFF

### SWITCHES IN SERIES

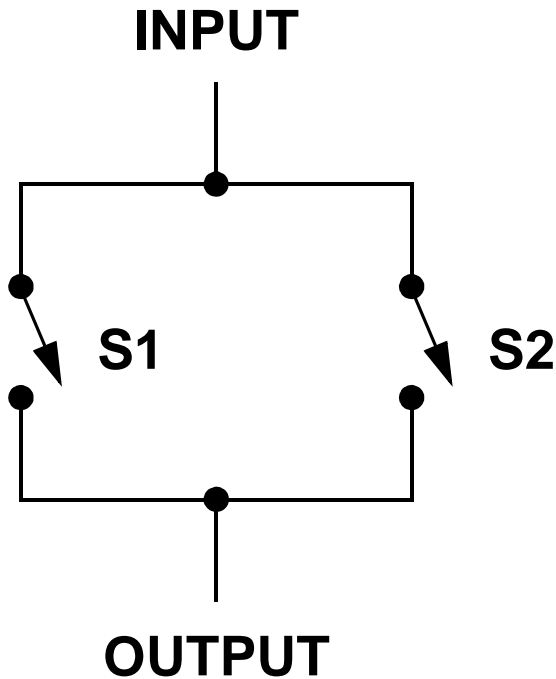


Truth Table

S1	S2	PATH?
OFF	OFF	NO
OFF	ON	NO
ON	OFF	NO
ON	ON	YES

- AND configuration

### SWITCHES IN PARALLEL



Truth Table

S1	S2	PATH?
OFF	OFF	NO
OFF	ON	YES
ON	OFF	YES
ON	ON	YES

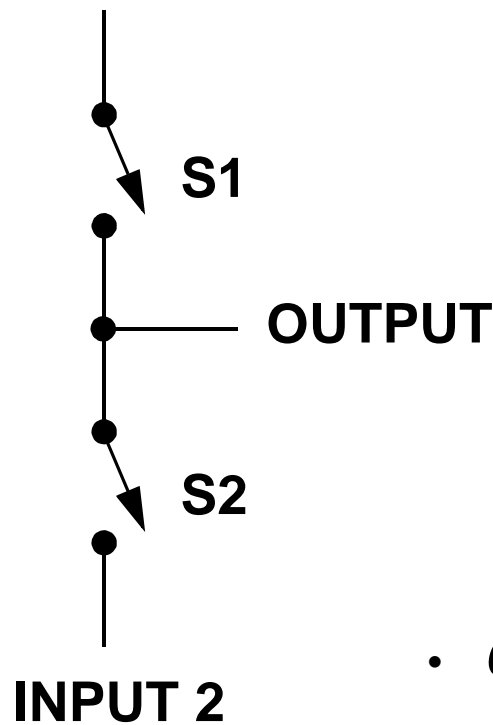
- OR configuration

# SWITCH NETWORKS

## INPUT SELECTOR

- SWITCH NETWORKS
- BASIC SWITCH
- SWITCHES IN SERIES
- SWITCHES IN PARALLEL

INPUT 1



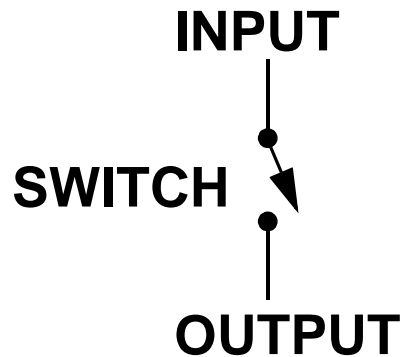
Truth Table

S1	S2	OUTPUT
OFF	OFF	NONE
OFF	ON	INPUT 2
ON	OFF	INPUT 1
<u>ON</u>	<u>ON</u>	<u>UNKNOWN</u>

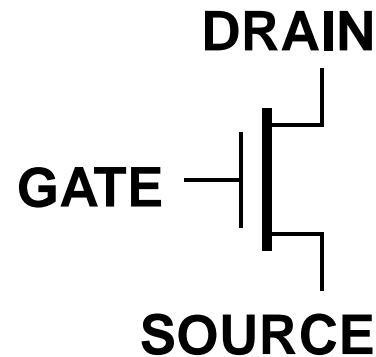
- *Crowbarred* level where logic level is indeterminate. Likely avoid this case.

- The idea is to use the series and parallel switch configurations to route signals in a desired fashion.
- Unfortunately, it is difficult to implement an ideal switch as given.
- Complementary Metal Oxide Semiconductor (CMOS) devices give us some interesting components.

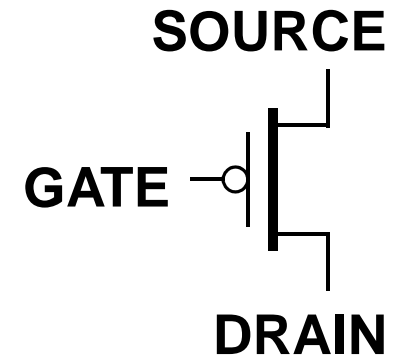
**IDEAL SWITCH**



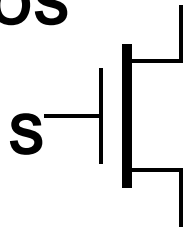
**nMOS transistor**



**pMOS transistor**



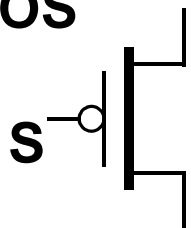
**nMOS**



S	SWITCH
0	OPEN
1	CLOSED

- nMOS when CLOSED
- Transmits logic level 0 well
- Transmits logic level 1 poorly

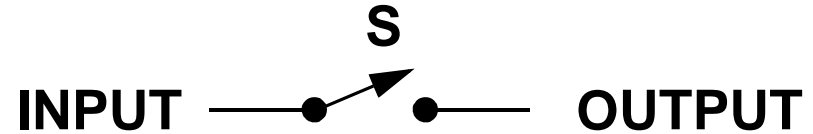
**pMOS**



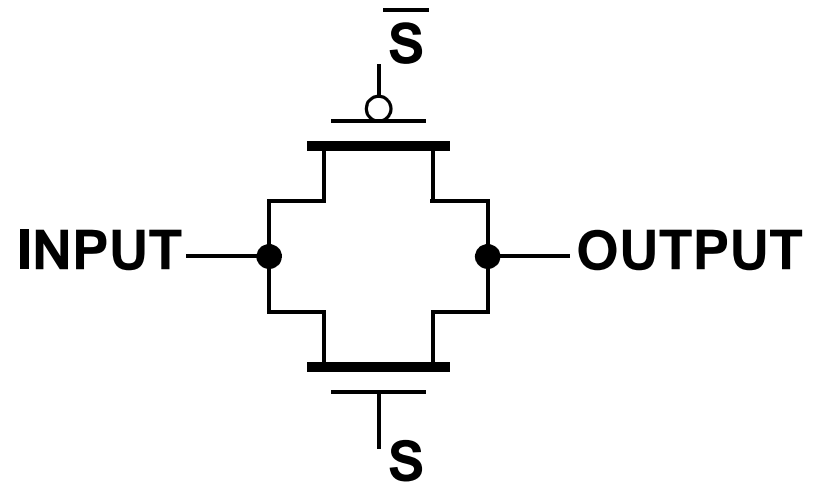
S	SWITCH
0	CLOSED
1	OPEN

- pMOS when CLOSED
- Transmits logic level 1 well
- Transmits logic level 0 poorly

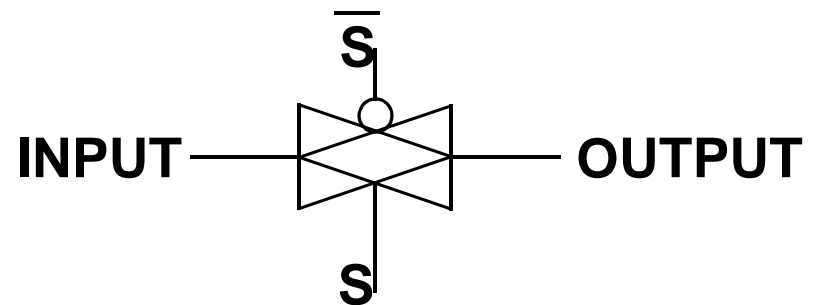
### IDEAL SWITCH



### CMOS TRANSMISSION GATE (SWITCH)

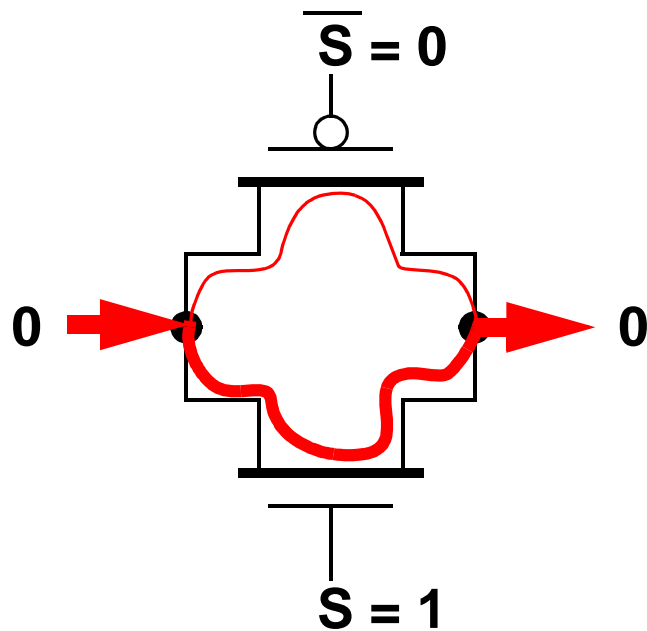


S	nMOS	pMOS	OUTPUT
0	OFF	OFF	Z
1	ON	ON	INPUT

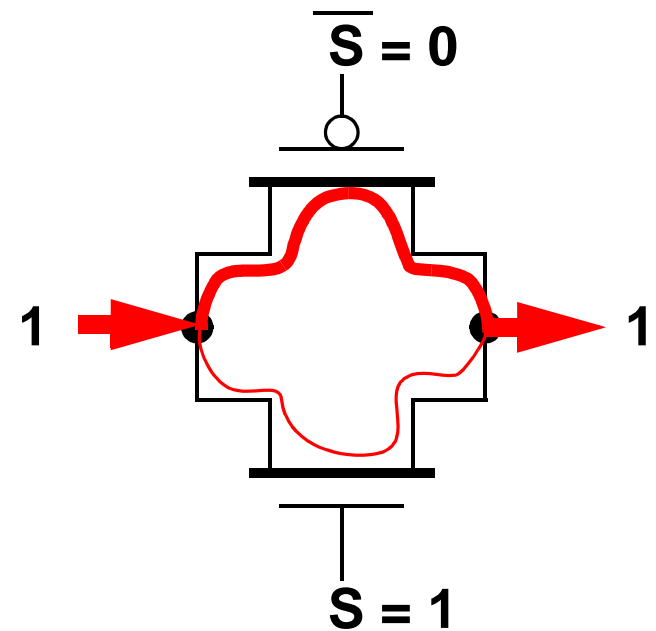


### SPLIT OF CURRENT ACROSS A TRANSMISSION GATE FOR LOGIC-0 AND LOGIC-1 INPUT

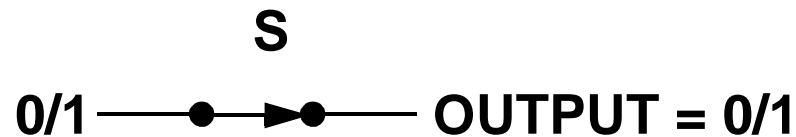
LOGIC-0 AT INPUT



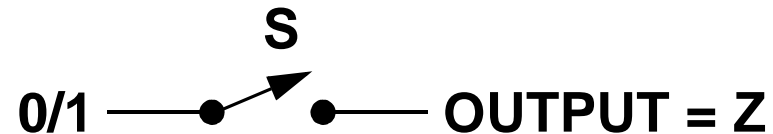
LOGIC-1 AT INPUT



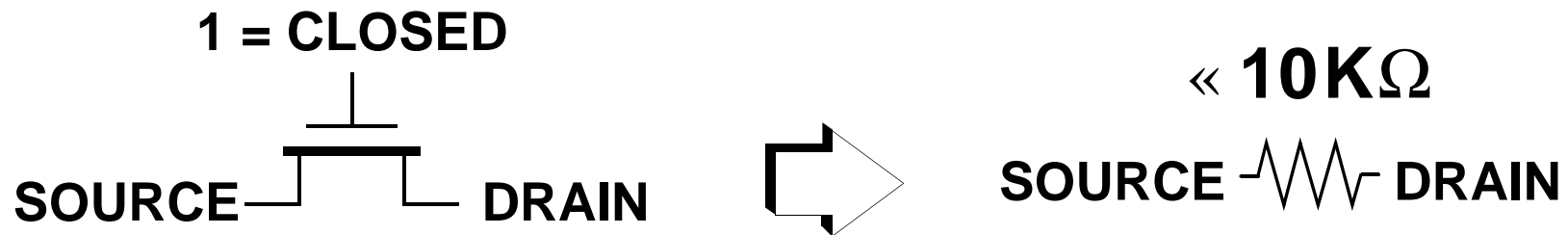
- With switches, we can consider three states for an output:
  - Logic-0
  - Logic-1
  - High Impedance **Z**
- Path exists for Logic-0 and Logic-1 when the switch is CLOSED.



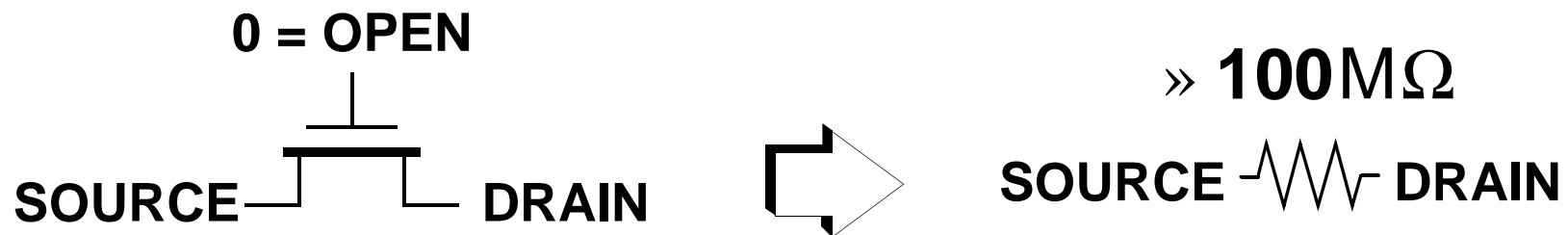
- High impedance is a state where the switch is OPEN.



- Another way of thinking of switches is as follows
  - Path exists for Logic-0 and Logic-1 when the switch is CLOSED, meaning that the **impedance/resistance is small** enough to allow amply flow of current.



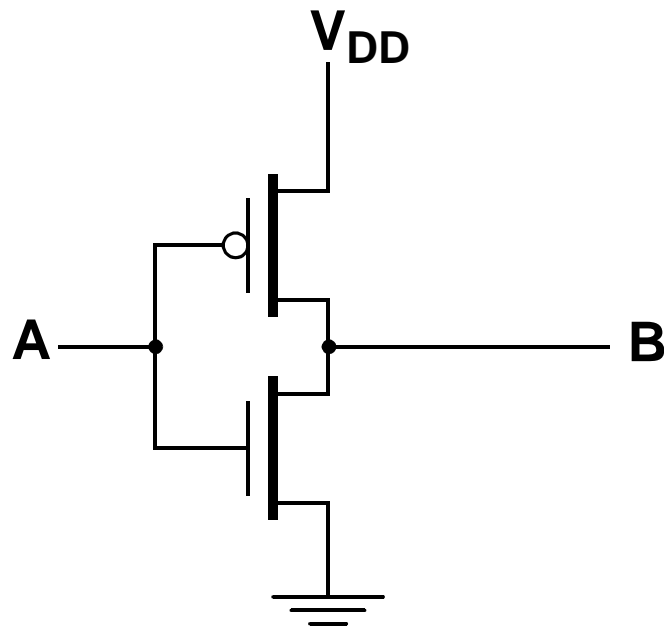
- High impedance is a state where the switch is OPEN, meaning that the **impedance/resistance is very large** allowing nearly no current flow.



# SWITCH NETWORKS

## INVERTER (NOT)

- CMOS
- SWITCH NETWORKS
- HIGH IMPEDANCE Z



$$B = \bar{A}$$

PULL-DOWN		PULL-UP	
A	B	A	B
0	Z	0	1
1	0	1	Z

→

A	B
0	1
1	0

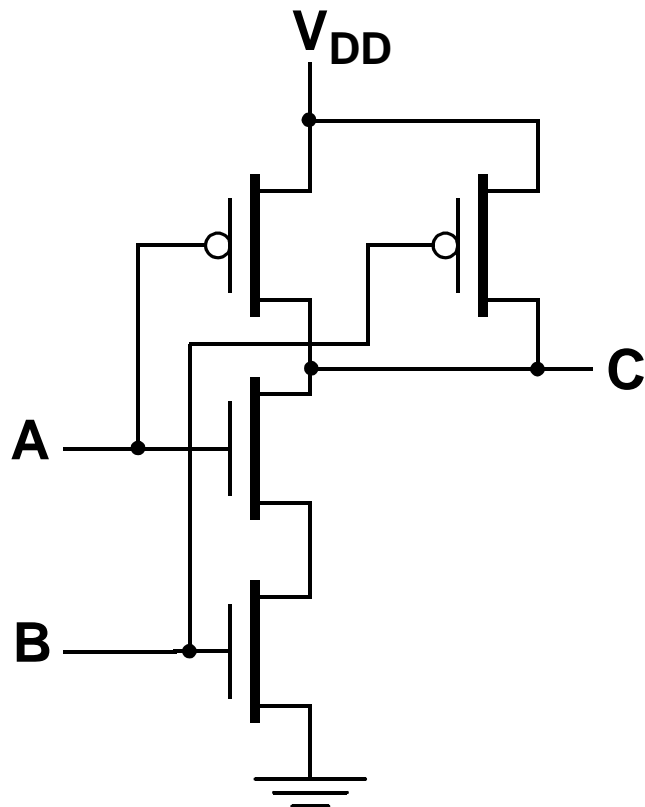
- This network inverts the binary input value.

# SWITCH NETWORKS

## NAND NETWORK

- CMOS
- SWITCH NETWORKS
- HIGH IMPEDANCE Z
- INVERTER

$$C = \overline{AB}$$



PULL-DOWN			PULL-UP		
A	B	C	A	B	C
0	0	Z	0	0	1
0	1	Z	0	1	1
1	0	Z	1	0	1
1	1	0	1	1	Z

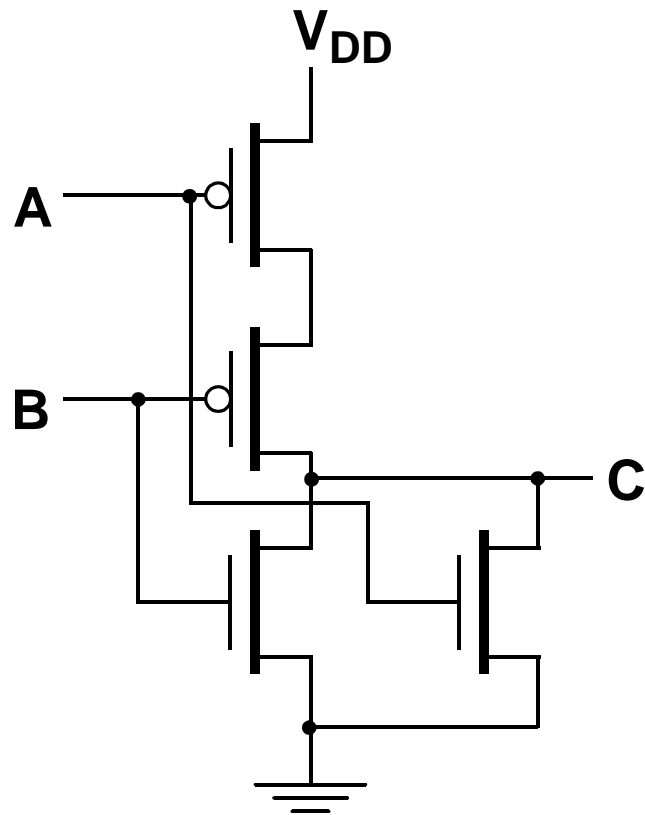


A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

# SWITCH NETWORKS

## NOR NETWORK

- SWITCH NETWORKS
- HIGH IMPEDANCE Z
- INVERTER
- NAND NETWORK



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

PULL-DOWN			PULL-UP		
A	B	C	A	B	C
0	0	Z	0	0	1
0	1	0	0	1	Z
1	0	0	1	0	Z
1	1	0	1	1	Z

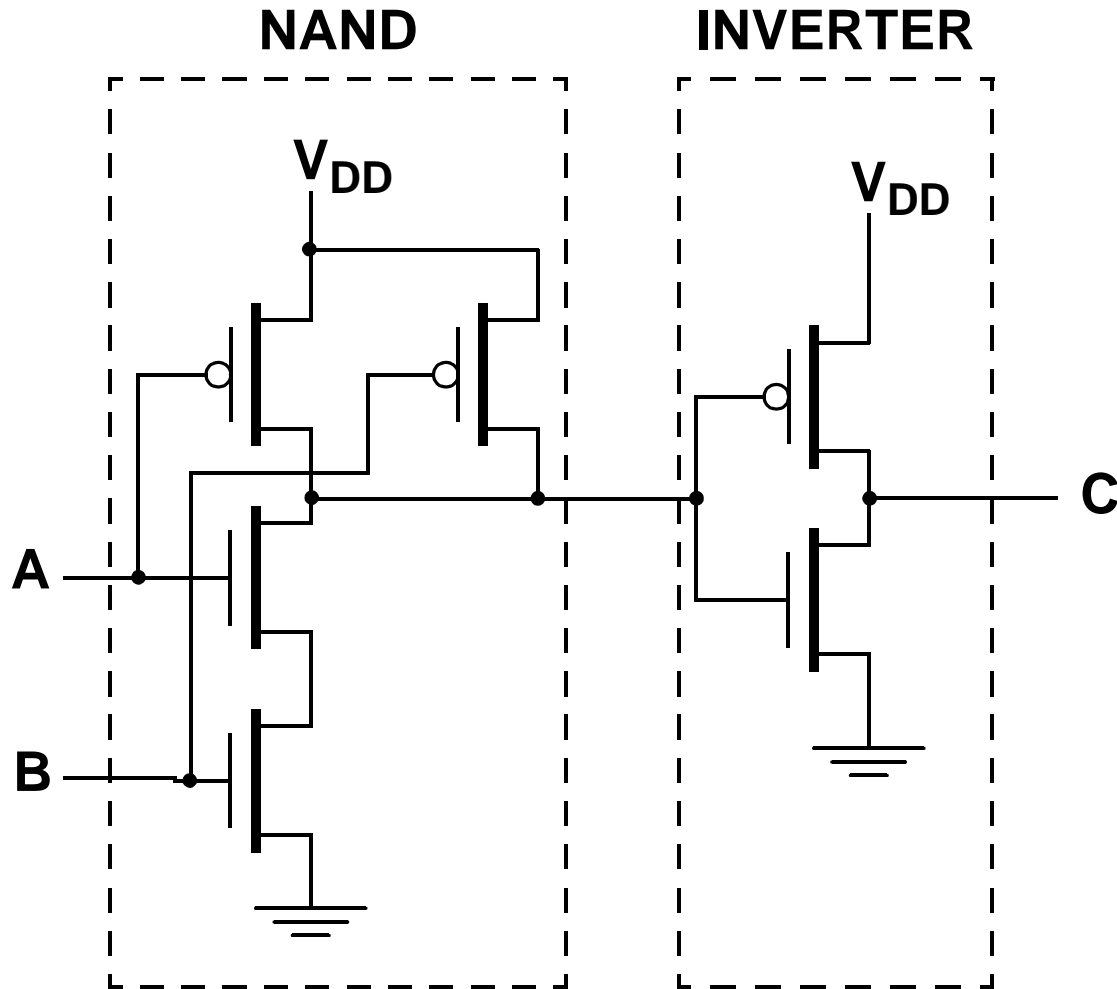


A	B	C
0	0	1
0	1	0
1	0	0
1	1	0

# SWITCH NETWORKS

## AND NETWORK

- SWITCH NETWORKS
- INVERTER
- NAND NETWORK
- NOR NETWORK



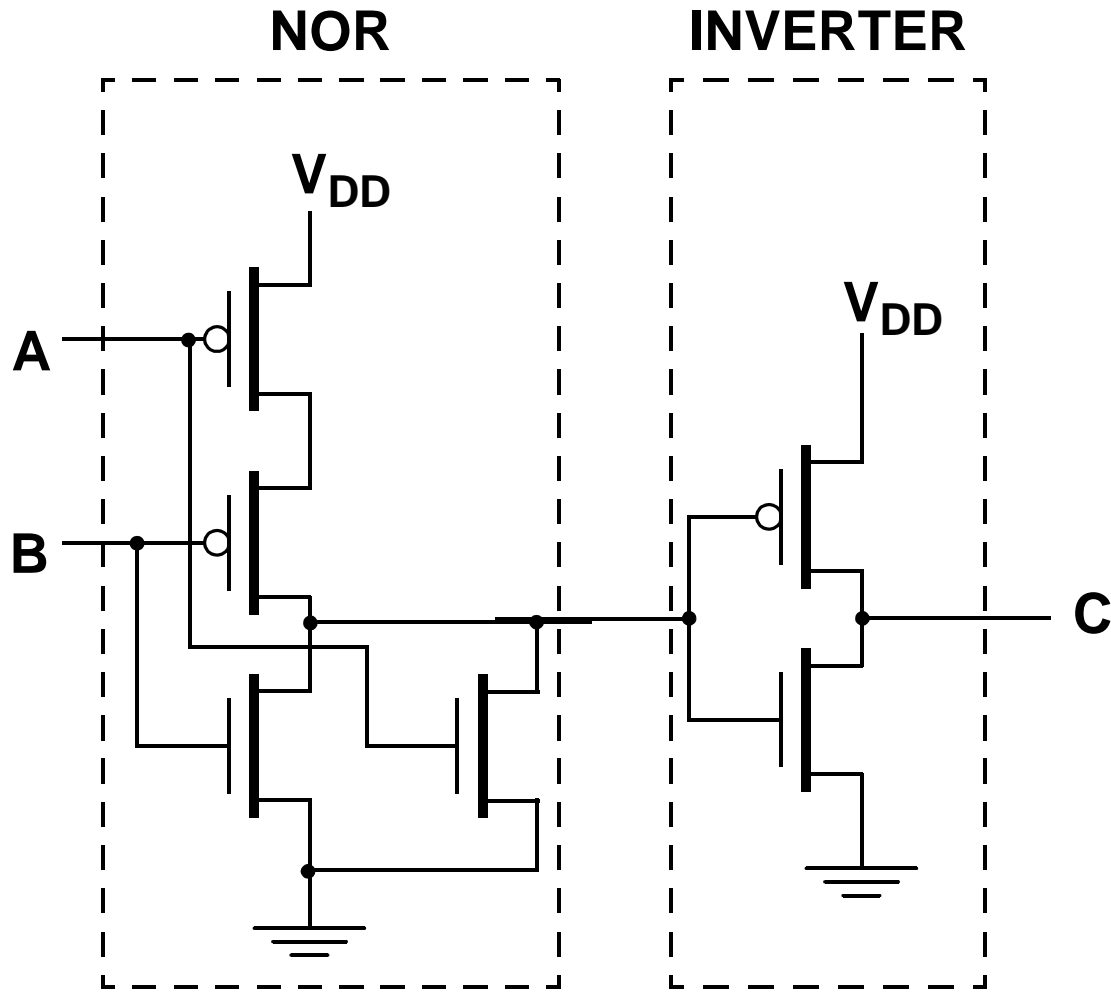
$$C = AB$$

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

# SWITCH NETWORKS

## OR NETWORK

- SWITCH NETWORKS
- NAND NETWORK
- NOR NETWORK
- AND NETWORK



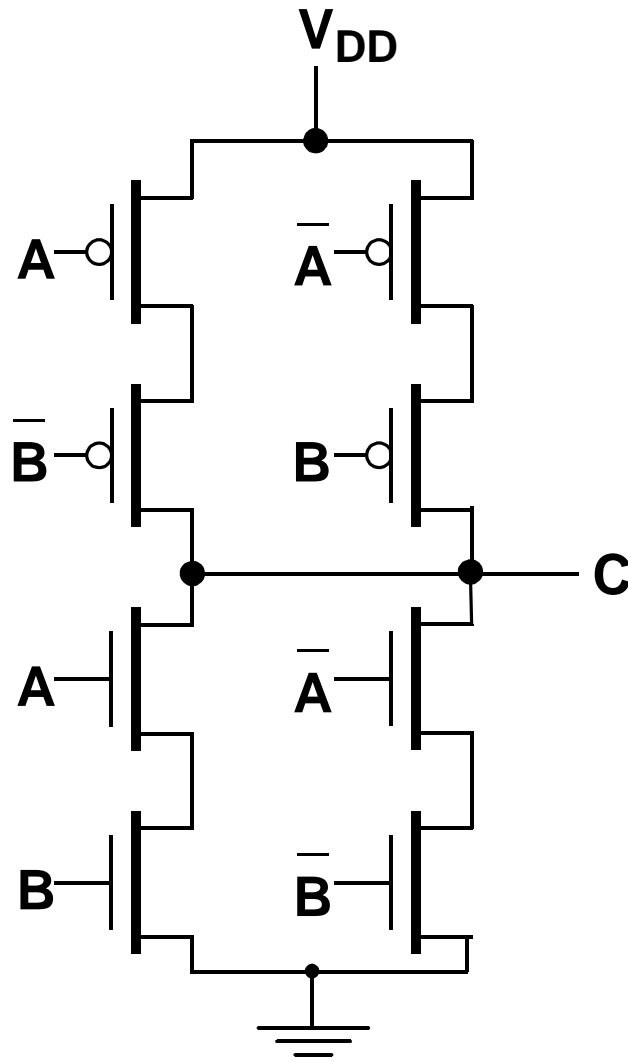
$$C = A + B$$

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

# SWITCH NETWORKS

## XOR NETWORK

- SWITCH NETWORKS
- NOR NETWORK
- AND NETWORK
- OR NETWORK

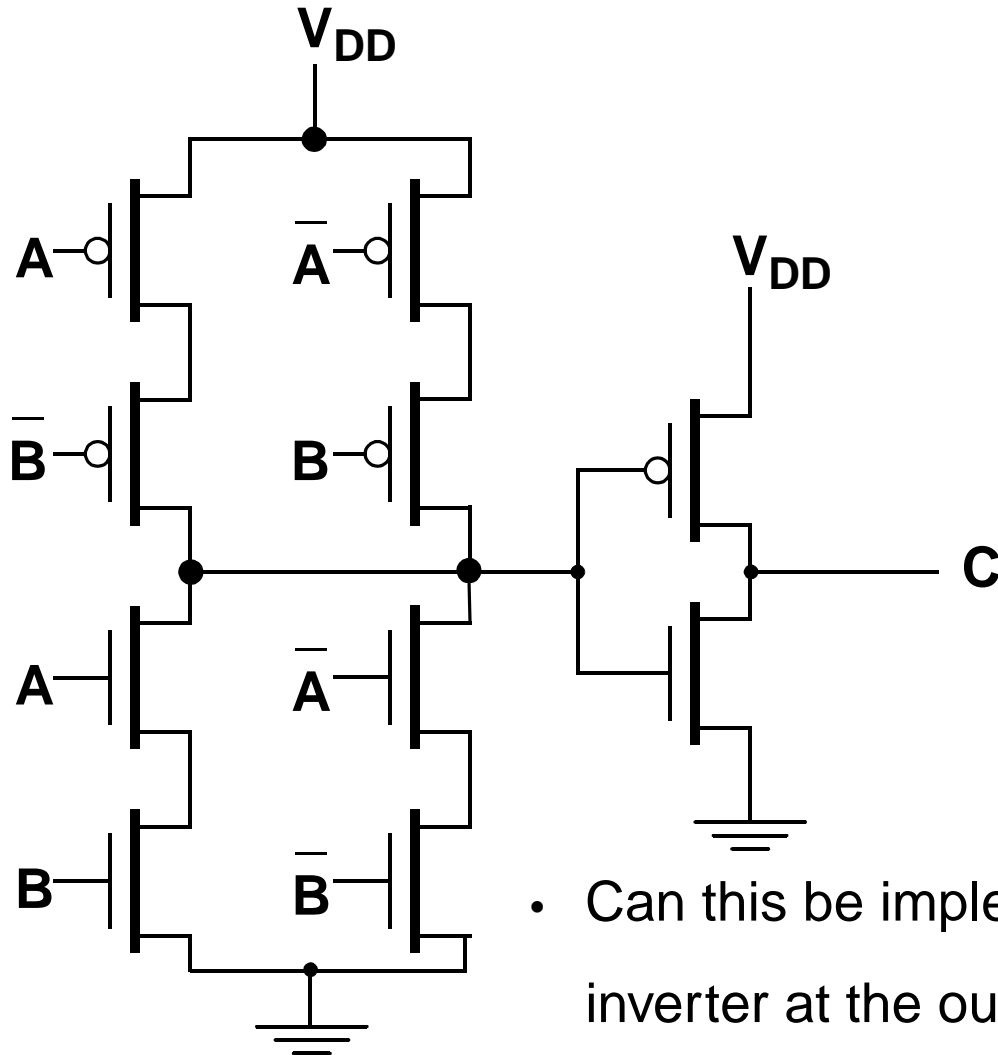


$$C = \bar{A}B + A\bar{B}$$

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

# SWITCH NETWORKS

## XNOR NETWORK



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

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

- Can this be implemented without the extra inverter at the output? Answer: Yes!

# SWITCH NETWORKS

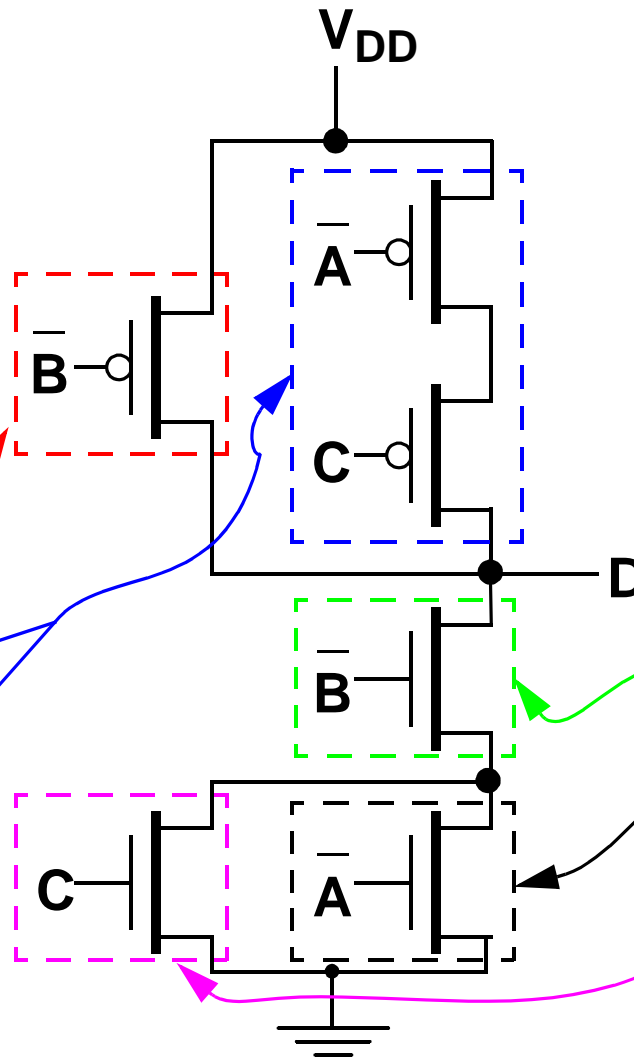
PULL-UP/PULL-DOWN

- SWITCH NETWORKS
- OR NETWORK
- XOR NETWORK
- XNOR NETWORK

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

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

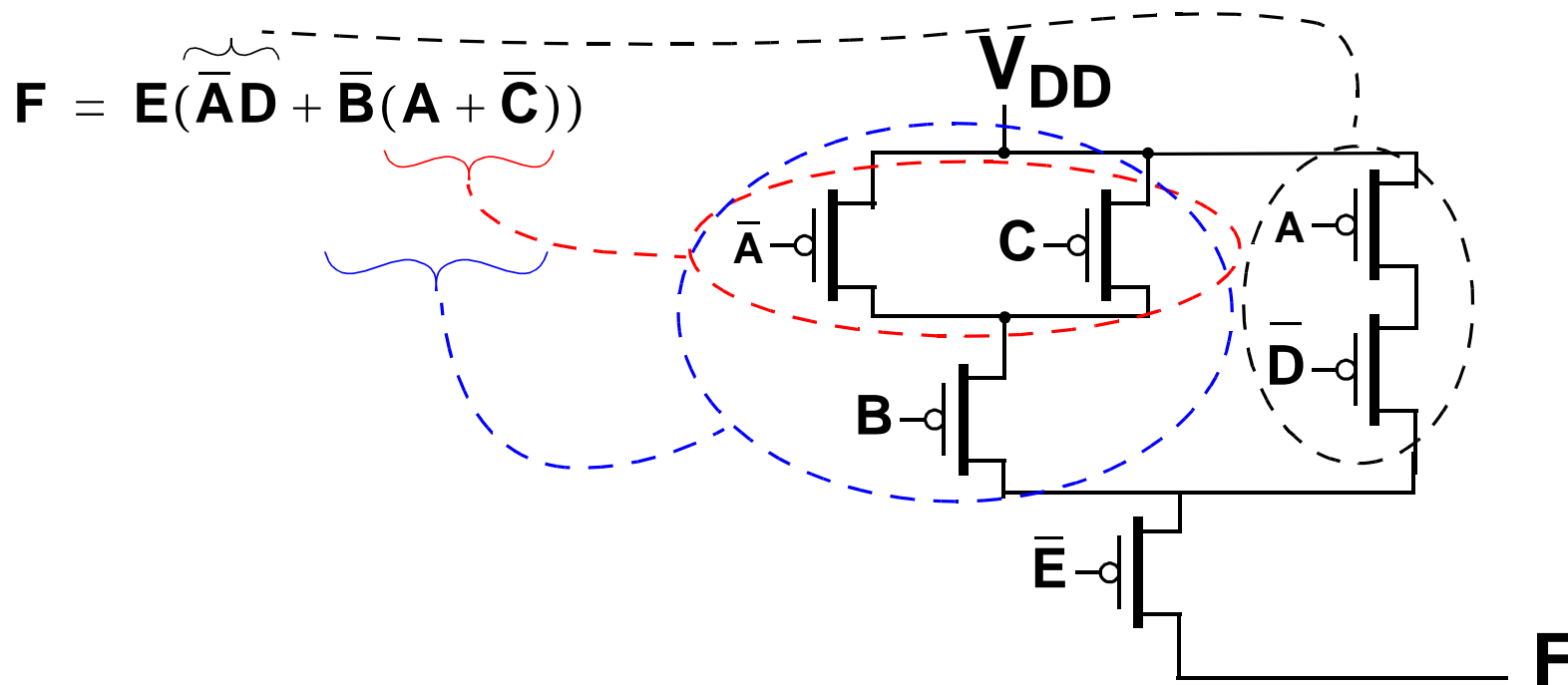
A	B	C	D
0	0	0	0
0	0	1	0
0	1	0	Z
0	1	1	Z
1	0	0	Z
1	0	1	0
1	1	0	Z
1	1	1	Z

- Most Boolean functions can be easily 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

# SWITCH NETWORKS

## EXAMPLE PULL-UP

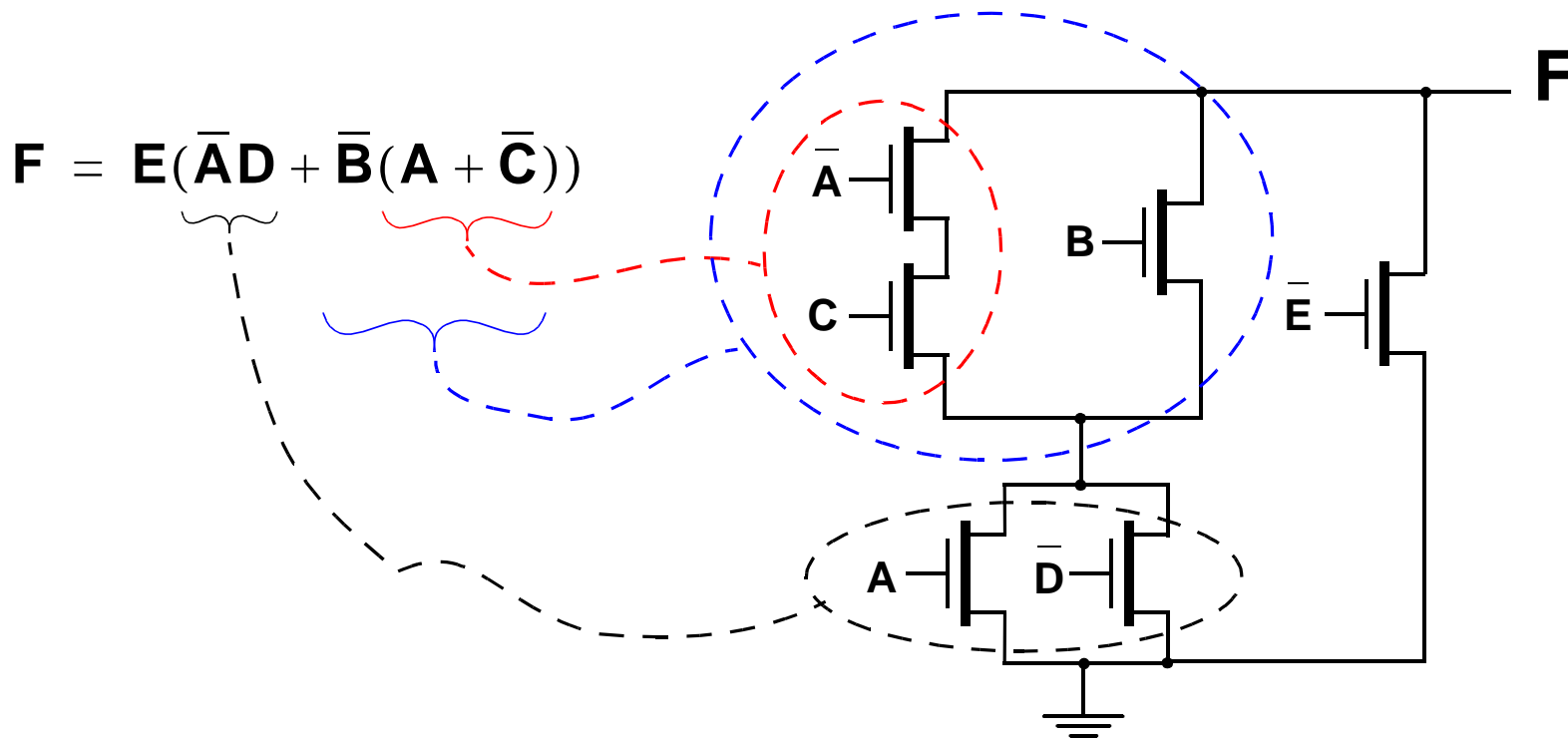
- To implement the Boolean function given below, the following pull-up network could be designed.



# SWITCH NETWORKS

## EXAMPLE PULL-DOWN

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



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

# SWITCH NETWORKS

COMPLETED EXAMPLE

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

