Lecture #16

OUTLINE

- Logic
- Binary representations
- Combinatorial logic circuits

Reading

- Chap 7-7.5

Digital Circuits – Introduction

- Analog: signal amplitude is continuous with time.
- Digital: signal amplitude is represented by a restricted set of discrete numbers.
  - Binary: only two values are allowed to represent the signal: High or low (i.e. logic 1 or 0).
- Digital word:
  - Each binary digit is called a bit
  - A series of bits form a word
    - Byte is a word consisting of 8-bits
- Advantages of digital signal
  - Digital signal is more resilient to noise→ can more easily differentiate high (1) and low (0)
- Transmission
  - Parallel transmission over a bus containing n wires.
    - Faster but short distance (internal to a computer or chip)
  - Serial transmission (transmit bits sequentially)
    - Longer distance

Binary Representation

- N bit can represent 2^N values: typically from 0 to 2^N-1
  - 3-bit word can represent 8 values: e.g. 0, 1, 2, 3, 4, 5, 6, 7
- Conversion
  - Integer to binary
  - Fraction to binary ($13.5_{10}=1101.1_2$ and $0.392_{10}=0.011001_2$)
- Octal and hexadecimal

Logic Gates and Memories

- Logic gates
  - Combine several logic variable inputs to produce a logic variable output
- Memory
  - Memoryless: output at a given instant depends the input values of that instant.
  - Momory: output depends on previous and present input values.

Boolean algebras

- are algebraic structures which "capture the essence" of the logical operations AND, OR and NOT as well as the corresponding set theoretic operations intersection, union and complement.
- They are named after George Boole, an English mathematician at University College Cork, who first defined them as part of a system of logic in the mid 19th century. Specifically, Boolean algebra was an attempt to use algebraic techniques to deal with expressions in the propositional calculus. Today, Boolean algebras find many applications in electronic design. They were first applied to switching by Claude Shannon in the 20th century.

Boolean algebras

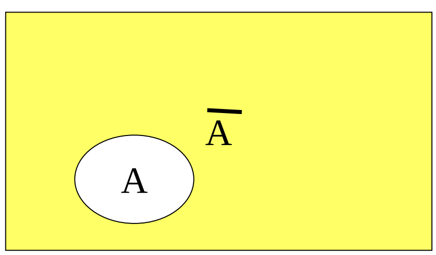
- The operators of Boolean algebra may be represented in various ways. Often they are simply written as AND, OR and NOT.
- In describing circuits, NAND (NOT AND), NOR (NOT OR) and XOR (eXclusive OR) may also be used.
- Mathematicians often use + for OR and · for AND (since in some ways those operations are analogous to addition and multiplication in other algebraic structures) and represent NOT by a line drawn above the expression being negated.

Boolean Algebra

- NOT operation (inverter) $A \cdot \bar{A} = 0$
  $A + \bar{A} = 1$
- AND operation
  $A \cdot A = A$
  $A \cdot 1 = A$
  $A \cdot 0 = 0$
  $A \cdot B = B \cdot A$
  $(A \cdot B) \cdot C = A \cdot (B \cdot C)$
- OR operation
  $A + A = A$
  $A + 1 = 1$
  $A + 0 = A$
  $A + B = B + A$
  $(A + B) + C = A + (B + C)$

EE40 Summer 2005: Lecture 16 Instructor: Octavian Florescu 7

Graphic Representation



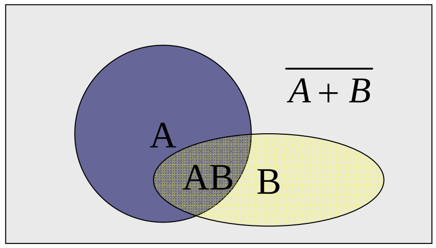
$A \cdot \bar{A} = 0$

$A + \bar{A} = 1$

Full square = complete set =1
Yellow part = NOT(A) = $\bar{A}$
White circle = A

EE40 Summer 2005: Lecture 16 Instructor: Octavian Florescu 8

Graphic Representation



$$A \oplus B = A\overline{B} + \overline{A}B = (A+B)\cdot(\overline{A}+\overline{B}) = \overline{A\cdot B + \overline{A+B}}$$

Exclusive OR=yellow and blue part – intersection/overlap part
=exactly when only one of the input is true

Boolean Algebra

- Distributive Property

$$A\cdot(B+C) = A\cdot B + A\cdot C$$

$$(A+B)\cdot C = (A+B)\cdot(A+C)$$

- De Morgan's laws

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

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

- An excellent web site to visit
  - http://en.wikipedia.org/wiki/Boolean_algebra

Examples

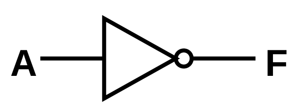
$$F = A \bullet \overline{B} \bullet C + A \bullet B \bullet C + (C+D) \bullet (\overline{D}+E)$$

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

Logic Functions, Symbols, & Notation

NAME | **SYMBOL** | **NOTATION** | **TRUTH TABLE**

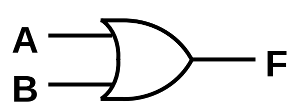
"NOT"



$F = \overline{A}$

A	F
0	1
1	0

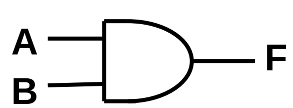
"OR"



$F = A+B$

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

"AND"

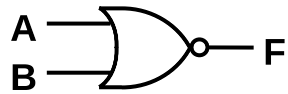


$F = A \bullet B$

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

Logic Functions, Symbols, & Notation 2

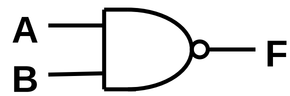
"NOR"



$F = \overline{A+B}$

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

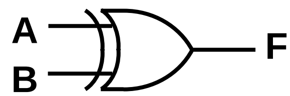
"NAND"



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

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

"XOR"
(exclusive OR)

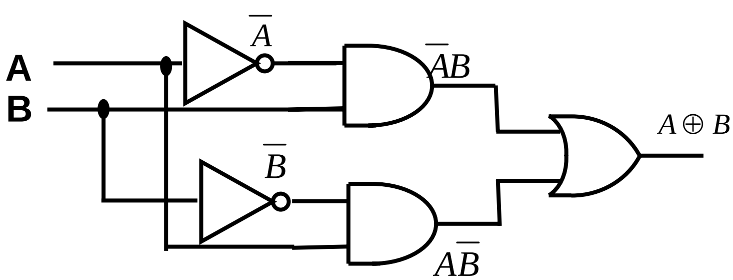


$F = A \oplus B$

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

Circuit Realization

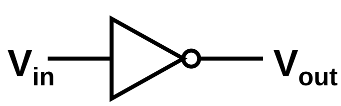
$$A \oplus B = A\overline{B} + \overline{A}B = (A+B)\cdot(\overline{A}+\overline{B}) = \overline{A\cdot B + \overline{A+B}}$$



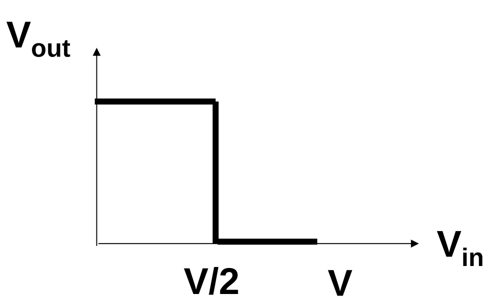
Fan in/Fan out

- Complex digital operations are formed with a variety of gates interconnected to yield the desired logic function.
- Sometimes a number of inputs are connected to one gate input and output of a gate may be connected to a number of gates.
- Fan-in: the maximum number of logic gates that can be connected at the input of a gate without altering its performance.
- Fan-out: the maximum number of logic gates that can be connected to the output of a gate without altering its performance.
- Typical fan-in and fan-out numbers are 3.

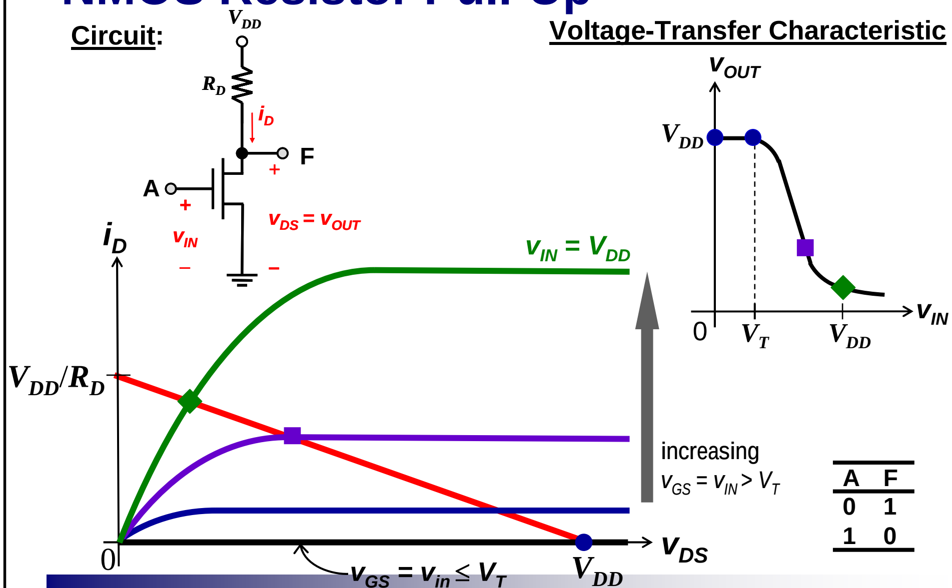
Inverter = NOT Gate



Ideal Transfer Characteristics



NMOS Resistor Pull-Up



A	F
0	1
1	0

Disadvantages of NMOS Logic Gates

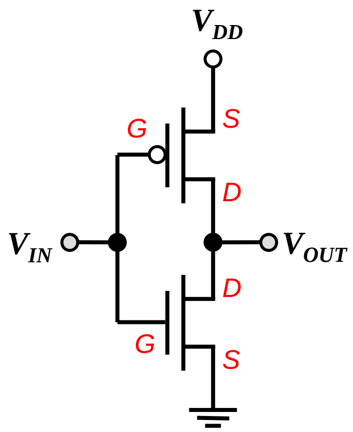
- Large values of $\boldsymbol{R_D}$ are required in order to
  - achieve a low value of V_{OL}
  - keep power consumption low

  → Large resistors are needed, but these take up a lot of space.

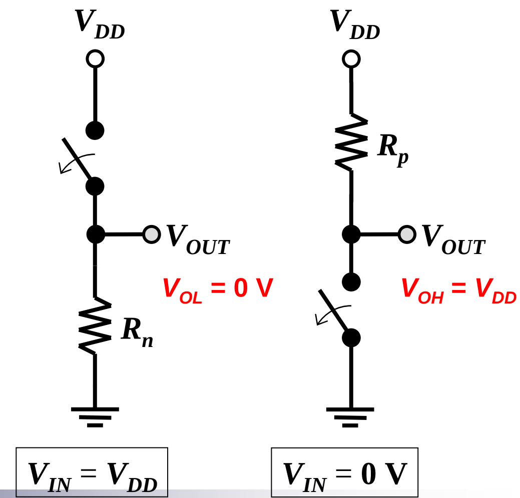
    - One solution is to replace the resistor with an NMOSFET that is always on.

The CMOS Inverter: Intuitive Perspective

CIRCUIT

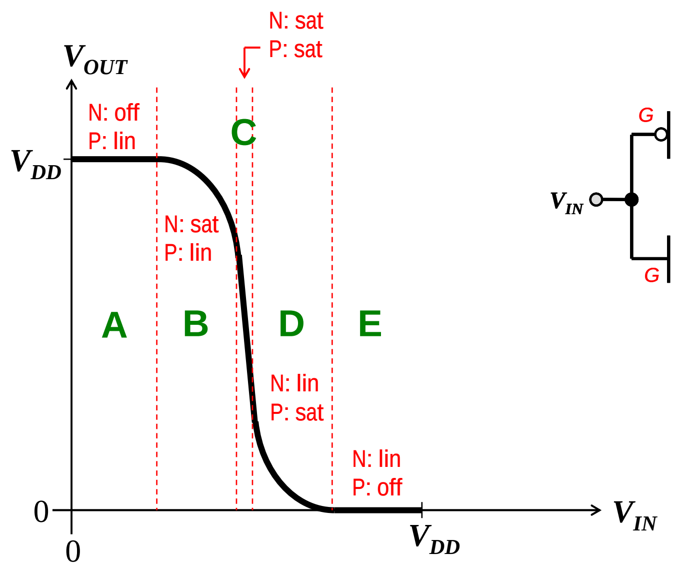


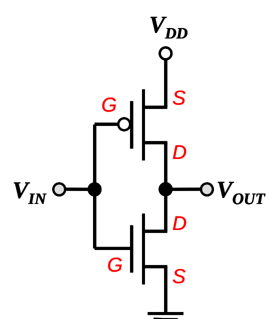
SWITCH MODELS



Low static power consumption, since one MOSFET is always off in steady state

CMOS Inverter Voltage Transfer Characteristic

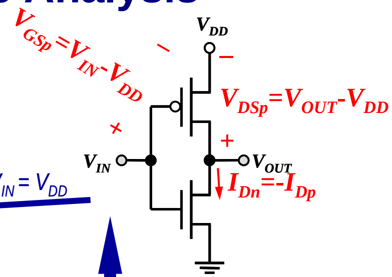


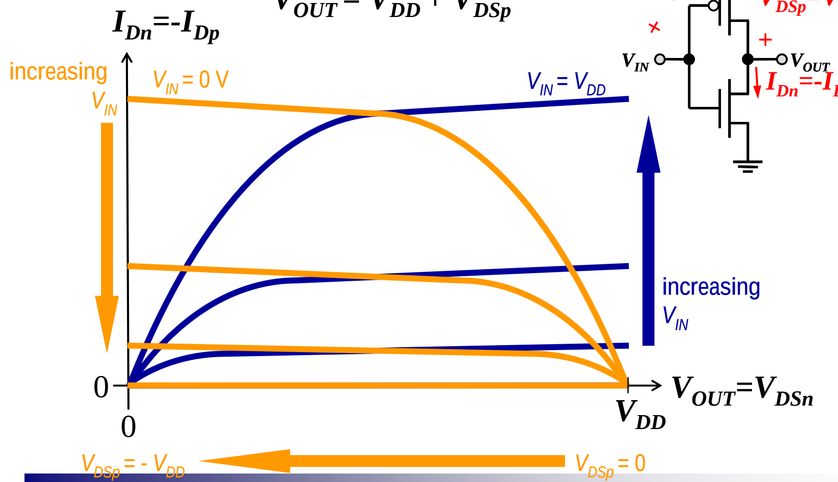


CMOS Inverter Load-Line Analysis

$V_{IN} = V_{DD} + V_{GSp}$

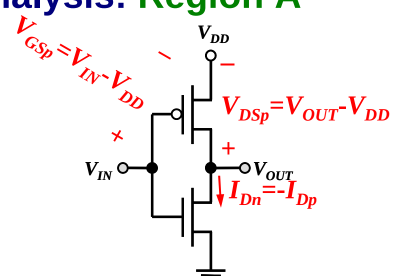
$V_{OUT} = V_{DD} + V_{DSp}$

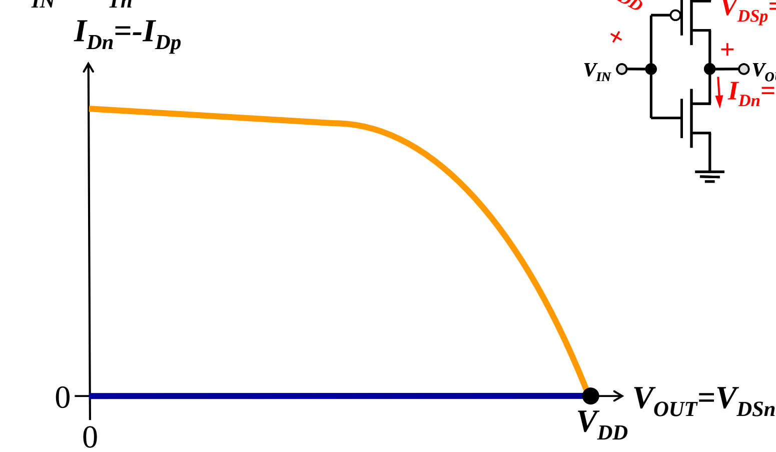




CMOS Inverter Load-Line Analysis: Region A

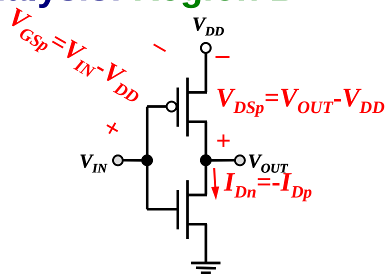
$V_{IN} \leq V_{Tn}$

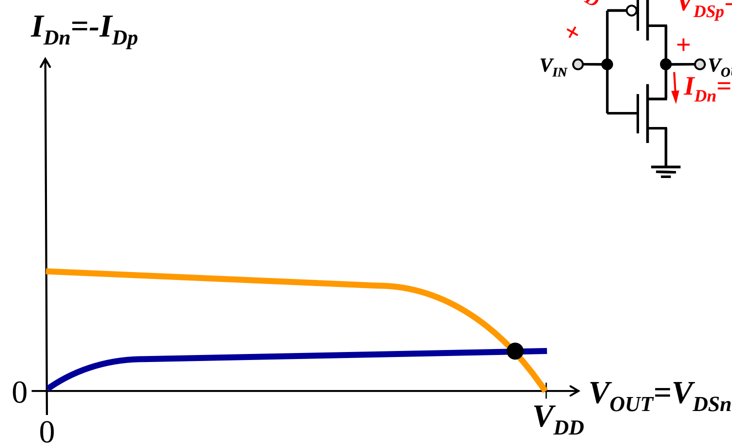




CMOS Inverter Load-Line Analysis: Region B

$V_{DD}/2 > V_{IN} > V_{Tn}$

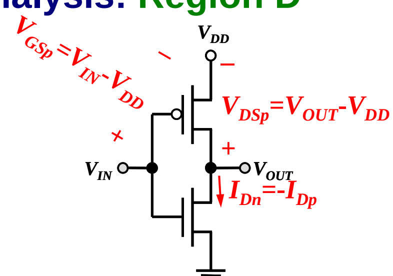


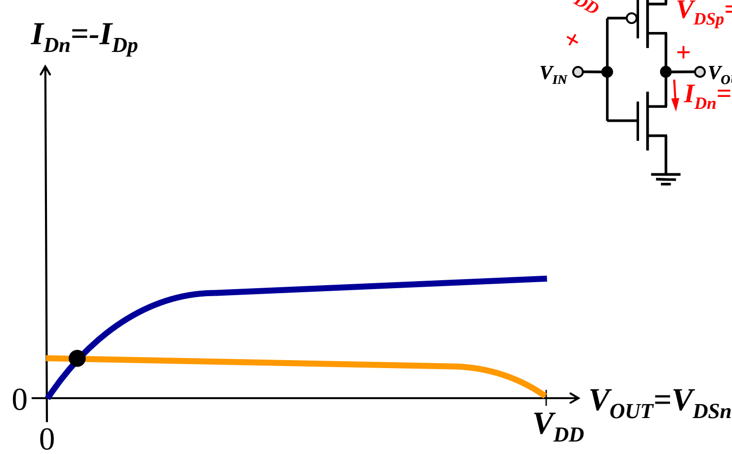


EE40 Summer 2005: Lecture 16 Instructor: Octavian Florescu 23

CMOS Inverter Load-Line Analysis: Region D

$V_{DD} - |V_{Tp}| > V_{IN} > V_{DD}/2$

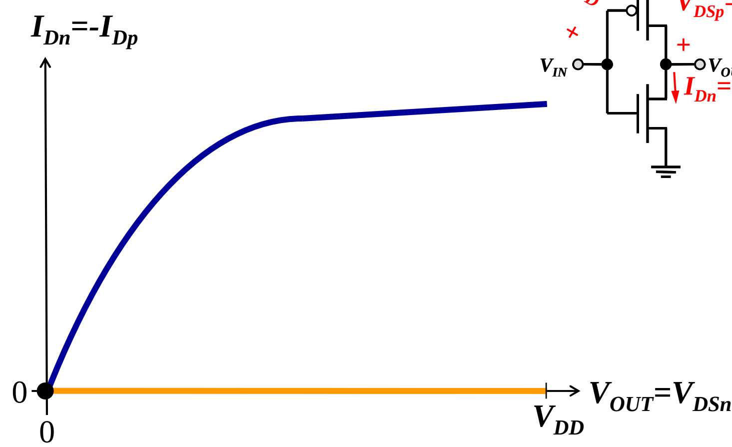


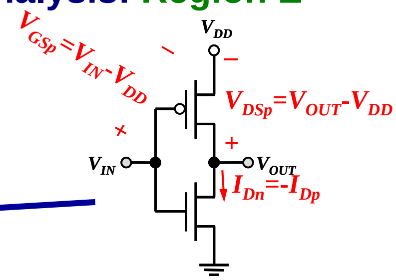


EE40 Summer 2005: Lecture 16 Instructor: Octavian Florescu 24

CMOS Inverter Load-Line Analysis: Region E

$V_{IN} > V_{DD} - |V_{Tp}|$

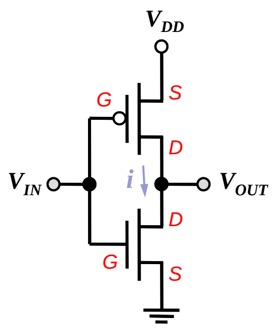


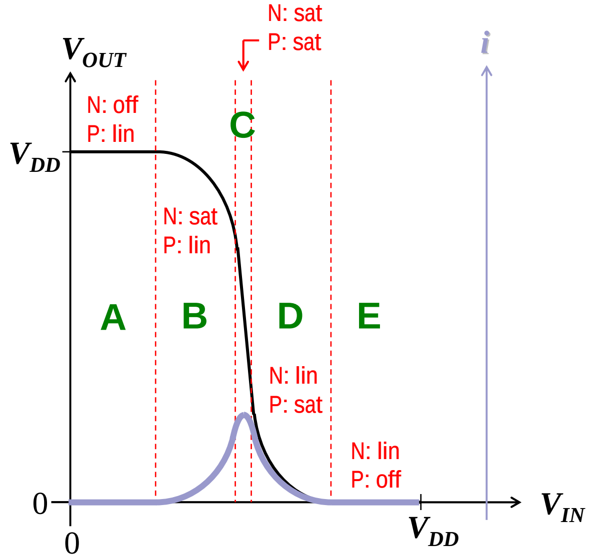


Features of CMOS Digital Circuits

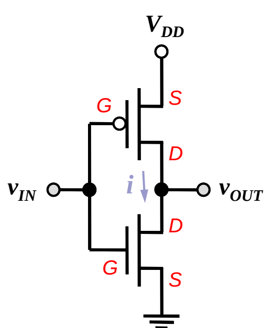
- The output is always connected to V_{DD} or **GND** in steady state
  - → Full logic swing; large noise margins
  - → Logic levels are not dependent upon the relative sizes of the devices (“ratioless”)
- There is no direct path between V_{DD} and **GND** in steady state
  - → no static power dissipation

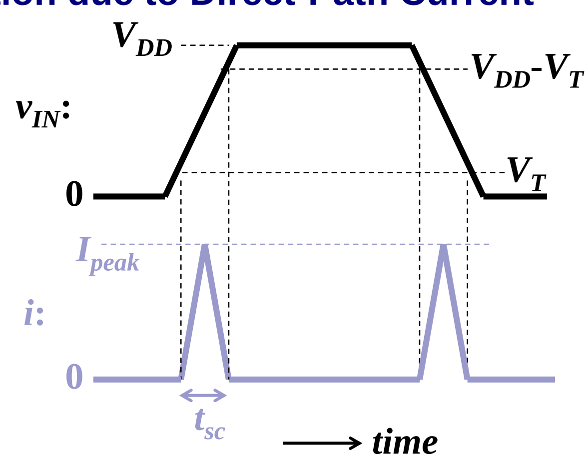
The CMOS Inverter: Current Flow during Switching





Power Dissipation due to Direct-Path Current

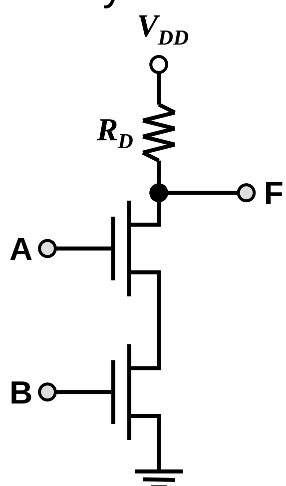




Energy consumed per switching period: $E_{dp} = t_{sc} V_{DD} I_{peak}$

NMOS NAND Gate

- Output is low only if both inputs are high

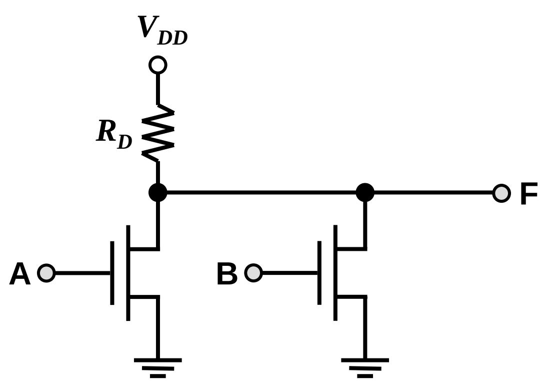


Truth Table

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

NMOS NOR Gate

- Output is low if either input is high

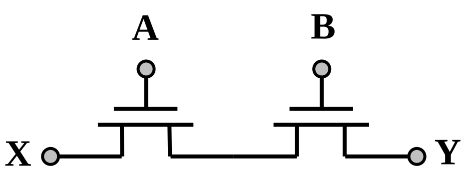


Truth Table

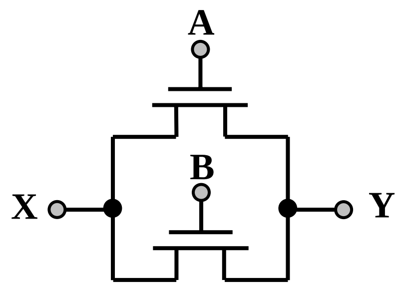
A	B	F
0	0	1
0	1	0
1	0	0
1	1	0

N-Channel MOSFET Operation

An NMOSFET is a closed switch when the input is high



Y = **X** if **A** and **B**

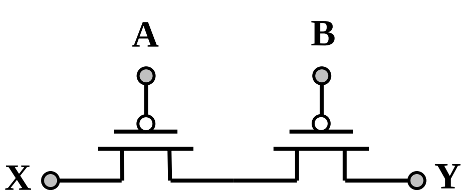


Y = **X** if **A** or **B**

NMOSFETs pass a “strong” **0** but a “weak” **1**

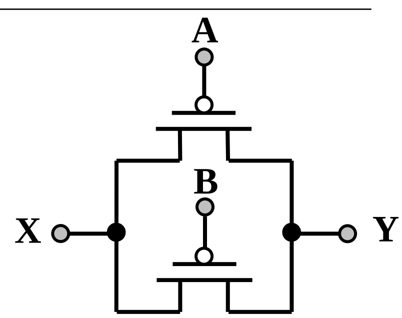
P-Channel MOSFET Operation

A PMOSFET is a closed switch when the input is low



Y = **X** if $\overline{A}$ and $\overline{B}$

$= (\overline{A + B})$



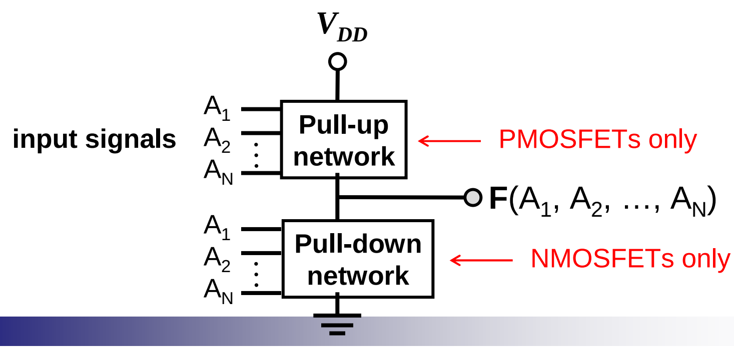
Y = **X** if $\overline{A}$ or $\overline{B}$

$= (\overline{AB})$

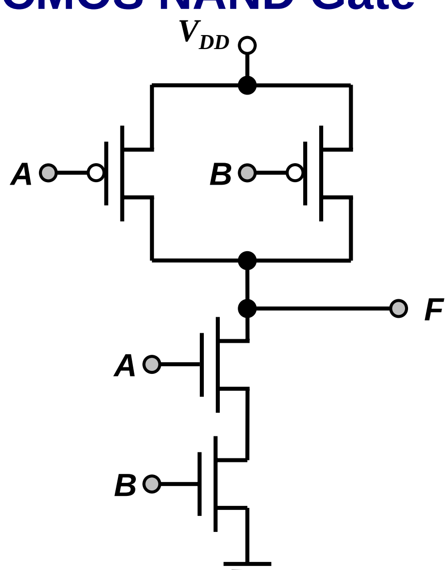
PMOSFETs pass a “strong” **1** but a “weak” **0**

Pull-Down and Pull-Up Devices

- In CMOS logic gates, NMOSFETs are used to connect the output to **GND**, whereas PMOSFETs are used to connect the output to V_{DD}.
  - An NMOSFET functions as a ***pull-down device*** when it is turned on (gate voltage = V_{DD})
  - A PMOSFET functions as a ***pull-up device*** when it is turned on (gate voltage = ***GND***)

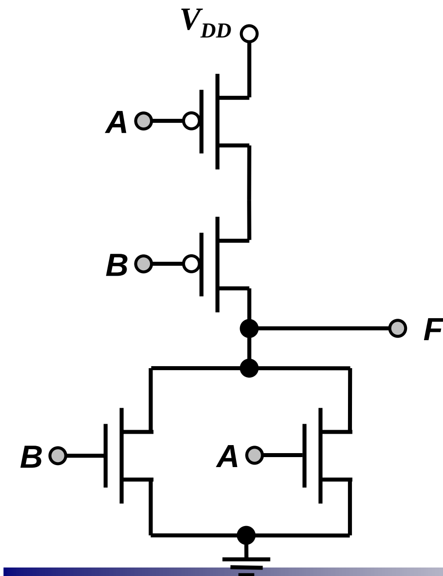


CMOS NAND Gate



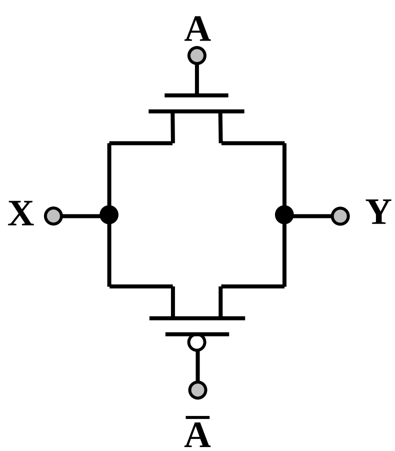
A	B	F
0	0	1
0	1	1
1	0	1
1	1	0

CMOS NOR Gate



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

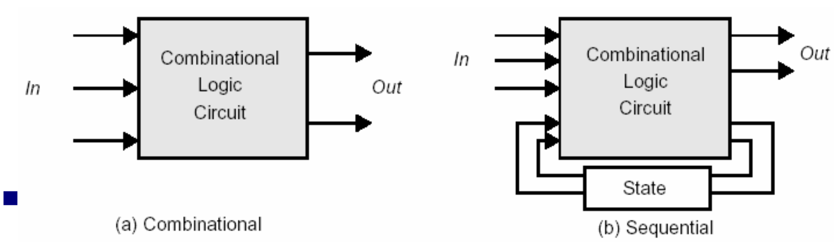
CMOS Pass Gate



Y = **X** if **A**

Combinational Logic Circuits

- Logic gates combine several logic-variable inputs to produce a logic-variable output.
- ***Combinational logic circuits*** are "memoryless" because their output value at a given instant depends only on the input values at that instant.



- depends on previous as well as present input values.

Logical Sufficiency of NAND Gates

- If the inputs to a NAND gate are tied together, an inverter results
- From De Morgan's laws, the OR operation can be realized by inverting the input variables and combining the results in a NAND gate.
- Since the basic logic functions (AND, OR, and NOT) can be realized by using only NAND gates, NAND gates are sufficient to realize any combinational logic function.

Logical Sufficiency of NOR Gates

- Show how to realize the AND, OR, and NOT functions using only NOR gates
- Since the basic logic functions (AND, OR, and NOT) can be realized by using only NOR gates, NOR gates are sufficient to realize any combinational logic function.

Synthesis of Logic Circuits

Suppose we are given a truth table for a logic function.

Is there a method to implement the logic function using basic logic gates?

Answer: There are lots of ways, but one simple way is the **"sum of products"** implementation method:

1) Write the sum of products expression based on the truth table for the logic function
2) Implement this expression using standard logic gates.

- We may not get the most efficient implementation this way, but we can simplify the circuit afterwards…

Logic Synthesis Example: Adder

Input			Output	
A	B	C	S_1	S_0
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

S_1 using sum-of-products:

1) Find where S_1 is **1**
2) Write down each product of inputs which create a **1**

   $\overline{A}\,B\,C$ $\quad A\,\overline{B}\,C$

   $A\,B\,\overline{C}$ $\quad A\,B\,C$

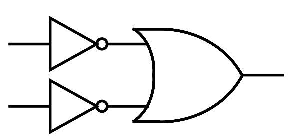
3) Sum all of the products

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

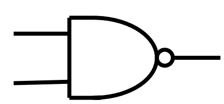
4) Draw the logic circuit

NAND Gate Implementation

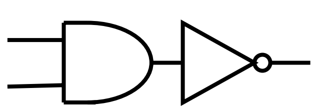
- De Morgan's law tells us that



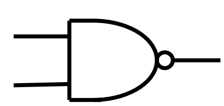
is the same as



- By definition,



is the same as



→ All sum-of-products expressions can be implemented with only NAND gates.

Creating a Better Circuit

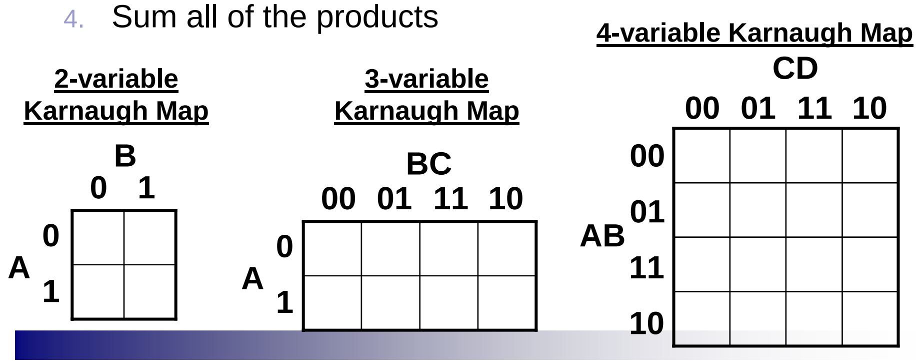
What makes a digital circuit better?

- Fewer number of gates
- Fewer inputs on each gate
  - multi-input gates are slower

- Let’s see how we can simplify the sum-of-products expression for S_1, to make a better circuit…
  - Use the Boolean algebra relations

Karnaugh Maps

- Graphical approach to minimizing the number of terms in a logic expression:
  1. Map the truth table into a Karnaugh map (see below)
  2. For each **1**, circle the biggest block that includes that **1**
  3. Write the product that corresponds to that block.
  4. Sum all of the products



Karnaugh Map Example

Input			Output	
A	B	C	S_1	S_0
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

Simplification of expression for S_1:

A \ BC	00	01	11	10
0	0	0	1	0
1	0	1	1	1

BC **AC** **AC** **AB**

$$S_1 = AB + BC + AC$$

Further Comments on Karnaugh Maps

- The algebraic manipulations needed to simplify a given expression are not always obvious. Karnaugh maps make it easier to minimize the number of terms in a logic expression.
- Terminology:
  - "2-cube: 2 squares that have a common edge (-> product of 3 variables)
  - "4-cube: 4 squares with common edges (-> product of 2 variables)
- In locating cubes on a Karnaugh map, the map should be considered to fold around from top to bottom, and from left to right.
  - Squares on the right-hand side are considered to be adjacent to those on the left-hand side.
  - Squares on the top of the map are considered to be adjacent to those on the bottom.
  - <u>Example</u>: The four squares in the map corners form a 4-cube

AB \ CD	00	01	11	10
00				
01				
11				
10				