Combinational Logic Circuits
Part IV -Theoretical Foundations

Overview

- Basic Boolean Functions
  - Basic Boolean functions of 1 and 2 binary variables
- Logic Basis and Conversions
  - Basis NAND, Basis NOR
- Logic Gates (NOT, AND, OR, NAND, NOR, XOR, XNOR)
- Combinational Logic Circuits from Boolean Functions
  - Circuits with AND-OR-NOT gates, with NAND gates, with NOR gates

Basic Boolean Functions

- Given N binary variables there exist 2^{2^N} different Boolean functions of these N variables
- Some functions of 1 and 2 variables we will call Basic Boolean Functions
- There exist 4 Boolean functions of 1 variable
- The truth tables and names of these functions are given below:
  - F1(X) = 0 -- function *constant 0*
  - F2(X) = X' -- function inversion (NOT)
  - F3(X) = X -- function identity
  - F4(X) = 1 -- function constant 1

X		F1	F2	F3	F4
0		0	1	0	1
1		0	0	1	1

Boolean Functions of 2 Variables (1)

- There exist 16 different functions of 2 variables
  - G1(X,Y) = 0 -- constant 0
  - G2(X,Y) = (X+Y)’ -- NOT-OR (NOR)
  - G3(X,Y) = X’Y
  - G4(X,Y) = X’
  - G5(X,Y) = XY’
  - G6(X,Y) = Y’
  - G7(X,Y) = X’Y+XY’ -- Exclusive-OR (XOR)
  - G8(X,Y) = (X•Y)’ -- NOT-AND (NAND)

X	Y		**G1**	**G2**	G3	G4
0	**0**		**0**	**1**	0	1
0	**1**		**0**	**0**	1	1
1	**0**		**0**	**0**	0	0
1	**1**		**0**	**0**	0	0

X	Y		G5	G6	**G7**	**G8**
0	**0**		0	1	**0**	**1**
0	**1**		0	0	**1**	**1**
1	**0**		1	1	**1**	**1**
1	**1**		0	0	**0**	**0**

Boolean Functions of 2 Variables (2)

- There exist 16 different functions of 2 variables
  - G9(X,Y) = X•Y -- AND
  - G10(X,Y) = X'Y'+XY -- Exclusive-NOR (XNOR)
  - G11(X,Y) = Y
  - G12(X,Y) = X' + Y
  - G13(X,Y) = X
  - G14(X,Y) = X + Y'
  - G15(X,Y) = X + Y -- OR
  - G16(X,Y) = 1 -- constant 1

X	Y		**G9**	**G10**	G11	G12
0	**0**		**0**	**1**	0	1
0	**1**		**0**	**0**	1	1
1	**0**		**0**	**0**	0	0
1	**1**		**1**	**1**	1	1

X	Y		G13	G14	**G15**	**G16**
0	**0**		0	1	**0**	**1**
0	**1**		0	0	**1**	**1**
1	**0**		1	1	**1**	**1**
1	**1**		1	1	**1**	**1**

Logic Basis

- Definition: Logic Basis is a minimal set of basic Boolean functions with which an arbitrary Boolean function can be represented
- Function set = {AND, OR, NOT}
  - This set consists of functions G9, G15, and F2
    - G9(X,Y) = X•Y -- this function equals to logic operation AND
    - G15(X,Y) = X+Y -- this function equals to logic operation OR
    - F2(X) = X' -- this function equals to logic operation NOT
  - We have seen in previous lectures that using the basic logic operations (AND,OR,NOT) we can represent any Boolean function
  - Is this set a logic basis?

Logic Basis NAND

- Basic Boolean function NAND (G8) is a logic basis!
- Proof: with function NAND, we can represent basic logic operations AND, OR, and NOT, which implies that we can represent any Boolean function
  - Function NAND is $G8(X,Y) = (X \bullet Y)'$
  - Function NOT is $F2(X) = X' = (X \bullet X)' = G8(X,X)$
  - Function AND is $G9(X,Y) = X \bullet Y = ((X \bullet Y)')' =$
    $= ((X \bullet Y)' \bullet (X \bullet Y)')' =$
    $= G8(G8(X,Y) , G8(X,Y))$
  - Function OR is $G15(X,Y) = X+Y = ((X+Y)')' = (X' \bullet Y')' =$
    $= ((X \bullet X)' \bullet (Y \bullet Y)')' =$
    $= G8(G8(X,X) , G8(Y,Y))$

Logic Basis NOR

- Basic Boolean function NOR (G2) is a logic basis!
- Proof: with function NOR, we can represent the basic logic operations AND, OR, and NOT which implies that we can represent any Boolean function
  - Function NOR is $G2(X,Y) = (X+Y)'$
  - Function NOT is $F2(X) = X' = (X+X)' = G2(X,X)$
  - Function AND is $G9(X,Y) = X \bullet Y = ((X \bullet Y)')' = (X'+Y')' =$
    $= ((X+X)' + (Y+Y)')' =$
    $= G2(G2(X,X) , G2(Y,Y))$
  - Function OR is $G15(X,Y) = X+Y = ((X+Y)')' =$
    $= ((X+Y)' + (X+Y)')' =$
    $= G2(G2(X,Y) , G2(X,Y))$

Converting Boolean Functions from set AND-OR-NOT to basis NAND

- Any Boolean function in set AND-OR-NOT can be converted to basis NAND using the DeMorgan's theorem
- Examples:

$$H1(W,X,Y,Z) = W'X'+W'Y'+WXY+W'Z =$$
$$= ((W'X'+W'Y'+WXY+W'Z)')' =$$
$$= ((W'X')' \bullet (W'Y')' \bullet (WXY)' \bullet (W'Z)')'$$

$$H2(X,Y,Z) = (X+Y+Z) \bullet (Y'+Z') =$$
$$= ((X+Y+Z)')' \bullet ((Y'+Z')')' =$$
$$= (X'Y'Z')' \bullet (YZ)' =$$
$$= (((X'Y'Z')' \bullet (YZ)')')'$$

Converting Boolean Functions from set AND-OR-NOT to basis NOR

- Any Boolean function in set AND-OR-NOT can be converted to basis NOR using the DeMorgan's theorem
- Examples:

$$H1(W,X,Y,Z) = W'X'+W'Y'+WXY+ W'Z =$$
$$= ((W'X')')' + ((W'Y')')' + ((WXY)')' + ((W'Z)')' =$$
$$= (W+X)' + (W+Y)' + (W'+X'+Y')' + (W+Z')' =$$
$$= ((\ (W+X)' + (W+Y)' + (W'+X'+Y')' + (W+Z')'\ )')'$$

$$H2(X,Y,Z) = (X+Y+Z) \bullet (Y'+Z') =$$
$$= ((\ (X+Y+Z) \bullet (Y'+Z')\ )')' =$$
$$= (\ (X+Y+Z)' + (Y'+Z')'\ )'$$

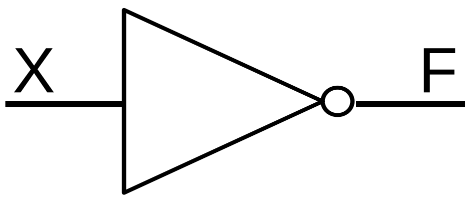
Logic Gates

- Digital Systems are made out of digital circuits
- Digital circuits are hardware components that manipulate binary information
- Certain well defined basic (small) digital circuits are called Logic Gates
- Logic Gates are electronic components that operate
  - on one or more input signals
  - to produce an output signal
- Logic Gates implement basic Boolean functions of one or more variables!
- Let us look at some Logic Gates

Inverter (NOT Gate)

- The NOT gate implements the basic Boolean function inversion (F2)

Graphical Symbol



Algebraic Equation

F = X’

Truth Table

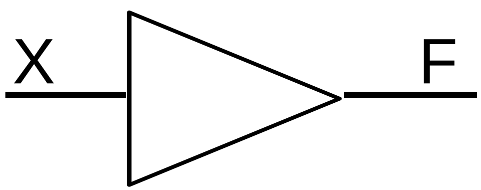
X	F
0	1
1	0

- Operation of the NOT gate:
  - Output F is 1 if input X = 0;
  - Output F is 0 if input X = 1;

Buffer

- The Buffer gate implements the basic Boolean function identity (F3)

Graphical Symbol



Algebraic Equation

$$F = X$$

Truth Table

X	F
0	0
1	1

- Operation of the Buffer gate:
  - Output F is 1 if input X = 1;
  - Output F is 0 if input X = 0;

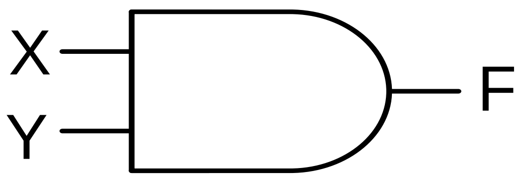
Why is this gate useful?!

- This gate is used to amplify the input electric signal X to permit more gates to be attached to the output

AND Gate

- The AND gate implements the basic Boolean function AND (G9)

Graphical Symbol



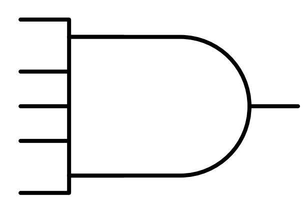
Algebraic Equation

$$F = X \bullet Y$$

Truth Table

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

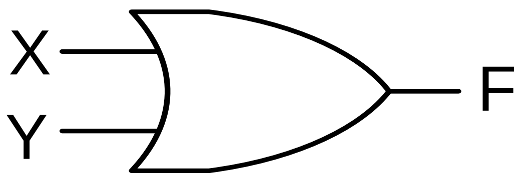
- Operation of the AND gate:
  - Output F is 1 *if and only if* input X=1 and input Y=1;
- There are AND gates with more than two inputs
  - $F = X_1 \bullet X_2 \bullet \ldots \bullet X_n$
  - Output F is 1 *if and only if* all inputs X_j are 1



OR Gate

- The OR gate implements the basic Boolean function OR (G15)

<u>Graphical Symbol</u>



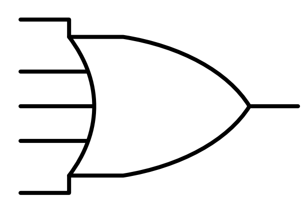
<u>Algebraic Equation</u>

$$F = X + Y$$

<u>Truth Table</u>

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

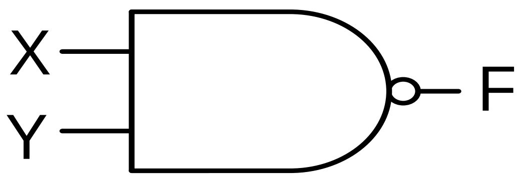
- Operation of the OR gate:
  - Output F is 0 <u>*if and only if*</u> input X=0 and input Y=0;
- There are OR gates with more than two inputs
  - $F = X_1 + X_2 + \ldots + X_n$
  - Output F is 0 <u>*if and only if*</u> all inputs X_j are 0



NAND Gate

- The NAND gate implements the basic Boolean function NOT-AND (G8)

Graphical Symbol



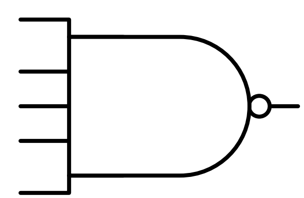
Algebraic Equation

$$F = (X \bullet Y)'$$

Truth Table

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

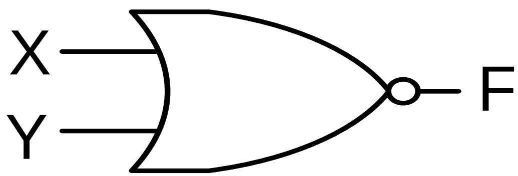
- Operation of the NAND gate:
  - Output F is 0 *if and only if* input X=1 and input Y=1;
- There are NAND gates with more than two inputs
  - $F = (X_1 \bullet X_2 \bullet \ldots \bullet X_n)'$
  - Output F is 0 *if and only if* all inputs X_j are 1



NOR Gate

- The NOR gate implements the basic Boolean function NOT-OR (G2)

Graphical Symbol



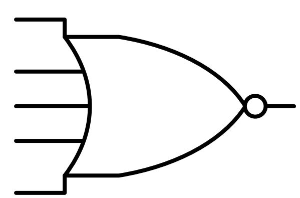
Algebraic Equation

$$F = (X + Y)'$$

Truth Table

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

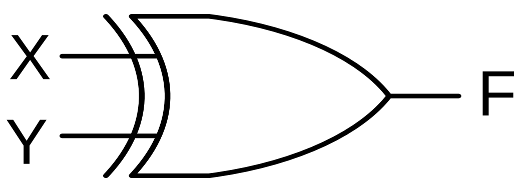
- Operation of the NOR gate:
  - Output F is 1 *if and only if* input X=0 and input Y=0;
- There are NOR gates with more than two inputs
  - $F = (X_1 + X_2 + \ldots + X_n)'$
  - Output F is 1 *if and only if* all inputs X_j are 0



XOR Gate

- The XOR gate implements the basic Boolean function Exclusive-OR (G7)

Graphical Symbol



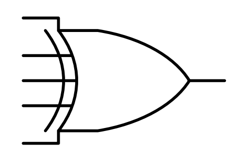
Algebraic Equation

$$F = XY' + X'Y = X \oplus Y$$

Truth Table

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

- Operation of the XOR gate:
  - Output F is 1 *if and only if* input X is not equal to input Y
- There are XOR gates with more than two inputs
  - $F = (X_1 \oplus X_2 \ \ldots \oplus X_n)$ (it is called ***odd*** function)

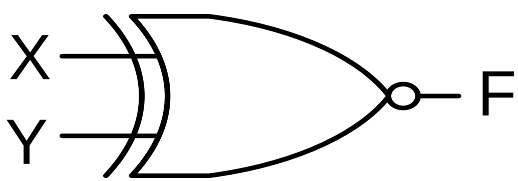


  - Output F is 1 *if and only if* **odd** number of inputs X_j are 1

XNOR Gate

- The XNOR gate implements the basic Boolean function Exclusive-NOR (G10)

<u>Graphical Symbol</u>

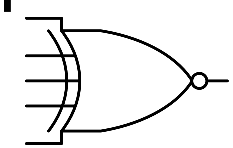


<u>Algebraic Equation</u>

$$F = XY + X'Y' = (X \oplus Y)'$$

<u>Truth Table</u>

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

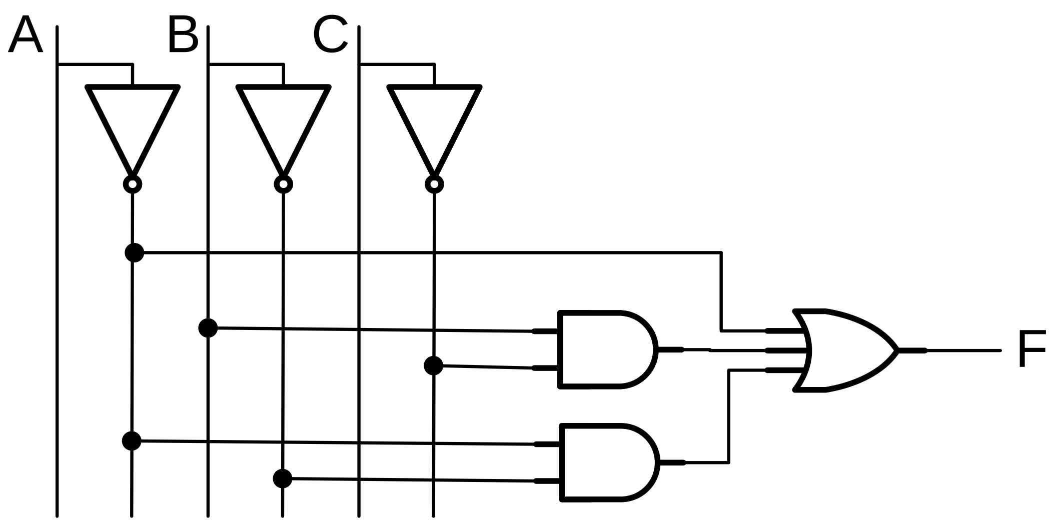
- Operation of the XNOR gate:
  - Output F is 1 <u>*if and only if*</u> input X is equal to input Y
- There are XNOR gates with more than two inputs.
  - $F = (X_1 \oplus X_2 \; \dots \oplus X_n)'$ (it is called ***even*** function) 
  - Output F is 1 <u>*if and only if*</u> **even** number of inputs X_j are 1

Combinational Logic Circuits from Boolean Functions

- Combinational Logic Circuits implement Boolean functions
- Any Boolean function can be represented using:
  - AND-OR-NOT basic functions
  - NAND basic function (Basis NAND)
  - NOR basic function (Basis NOR)
- Logic Gates implement the basic Boolean functions
- Thus, any Boolean function can be implemented using: AND, OR, NOT, NAND, NOR gates!
  - Outputs of Logic Gates are connected to inputs of other gates to form a Combinational Logic Circuit

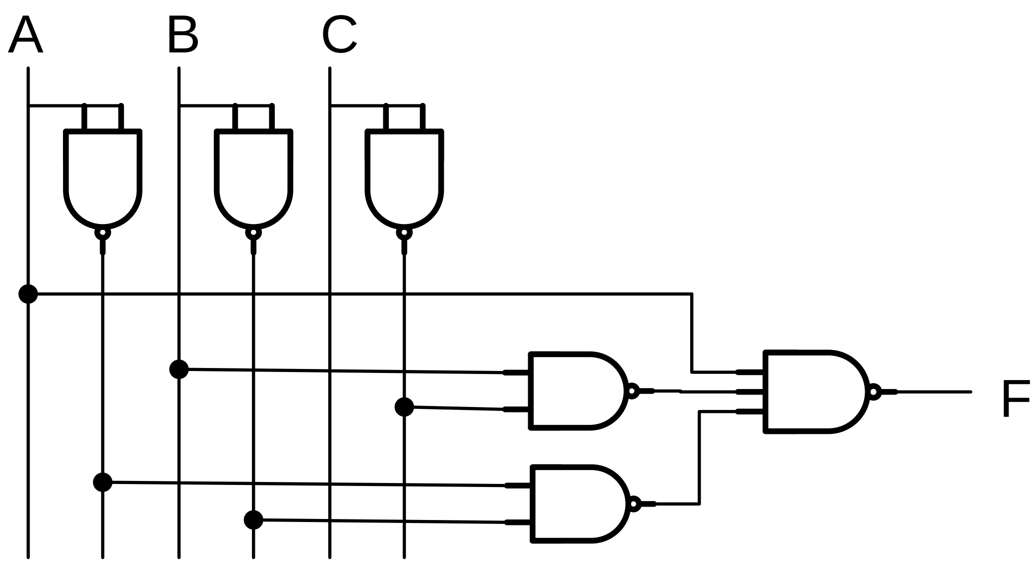
Combinational Logic Circuits from Boolean Functions using AND-OR-NOT

- Any Boolean function can be implemented using AND, OR and NOT gates
- Consider Boolean function **F = A' + B•C' + A'•B'**
- A combinational logic circuit can be constructed to implement F, by appropriately connecting input signals and logic gates:
  - Circuit input signals → from function variables (A, B, C)
  - Circuit output signal → function output (F)
  - Logic gates → from logic operations



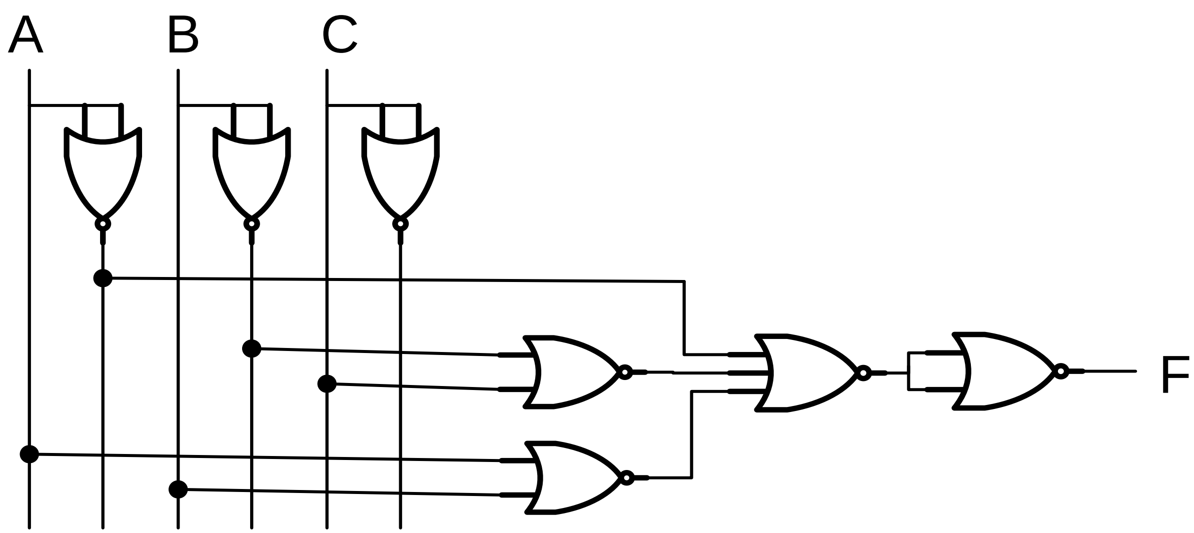
Combinational Logic Circuits from Boolean Functions in basis NAND

- Any Boolean function can be implemented using only NAND gates
- Consider Boolean function **F = A' + B•C' + A'•B'**
- Convert **F** in basis NAND
  F = ((A' + B•C' + A'•B')')' = (A • (B•C')' • (A'•B')')'



Combinational Logic Circuits from Boolean Functions in basis NOR

- Any Boolean function can be implemented using only NOR gates
- Consider Boolean function **F = A’ + B•C’ + A’•B’**
- Convert **F** in basis NOR
  F = A’+((B•C’)’)’+((A’•B’)’)’ = ((A’+(B’+C)’+(A+B)’)’)’



Combinational Logic Circuits from Boolean Functions (cont.)

- To design a cost-effective and efficient circuit, we must simplifying the corresponding Boolean function to be implemented

- Observe the truth table of
  - **F = A' + B•C' + A'•B'** and
  - **G = A' + B•C'**

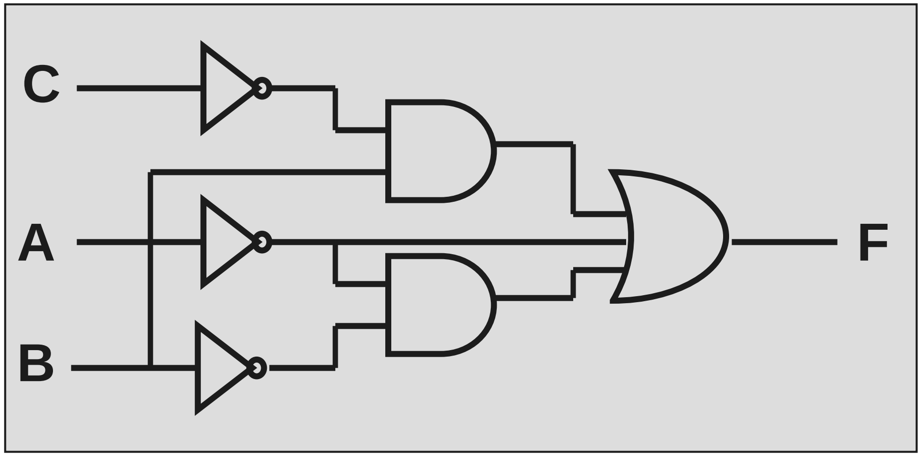
- Truth tables for F and G are identical
  - F and G represent the same function

- Use G to implement the logic circuit!
  - less components (gates) are needed

A	B	C	F	G
0	0	0	1	1
0	0	1	1	1
0	1	0	1	1
0	1	1	1	1
1	0	0	0	0
1	0	1	0	0
1	1	0	1	1
1	1	1	0	0

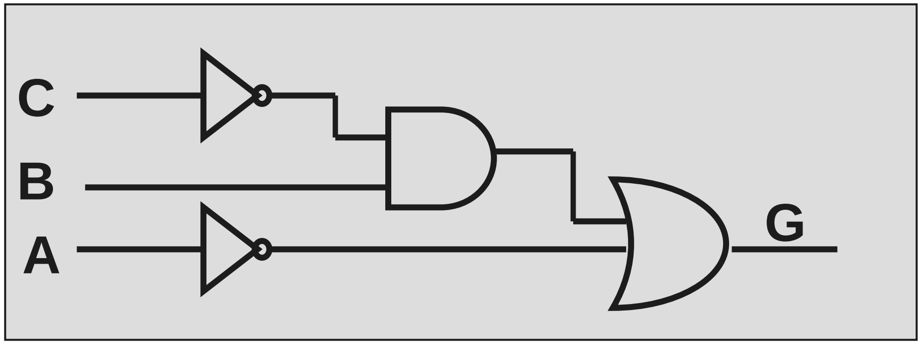
Combinational Logic Circuits from Boolean Functions (cont.)

F = A’ + B•C’ + A’•B’



Simplify F

G = A’ + B•C’



Digital Logic Circuits
Physical Implementation Basics

Overview

- Integrated Circuits
  - Form Sand to Integrated Circuits (Chips)
- CMOS Circuits Technology
  - MOS Transistors as Switches
  - Basic Gates as CMOS circuits
- Propagation Delay of Gates and Logic Circuits
- Basic Assumption for Logic Gates

Integrated Circuits (ICs)

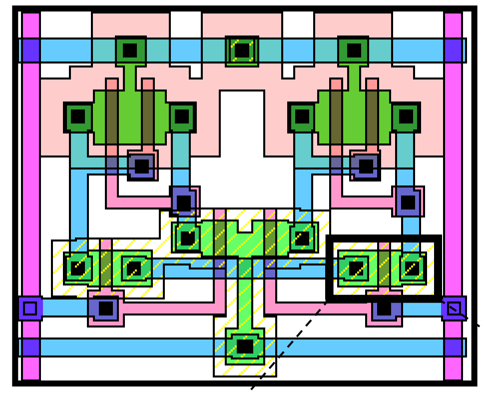
- Digital Systems are made out of digital circuits
- Certain well defined basic (small) digital circuits are called Logic Gates
- Gates have inputs and outputs and perform specific mathematical logic operations
- Outputs of gates are connected to inputs of other gates to form a digital logic circuit
- Digital logic circuits are physically implemented using transistors and interconnections in complex semiconductor devices called integrated circuits

From Sand to Integrated Circuits

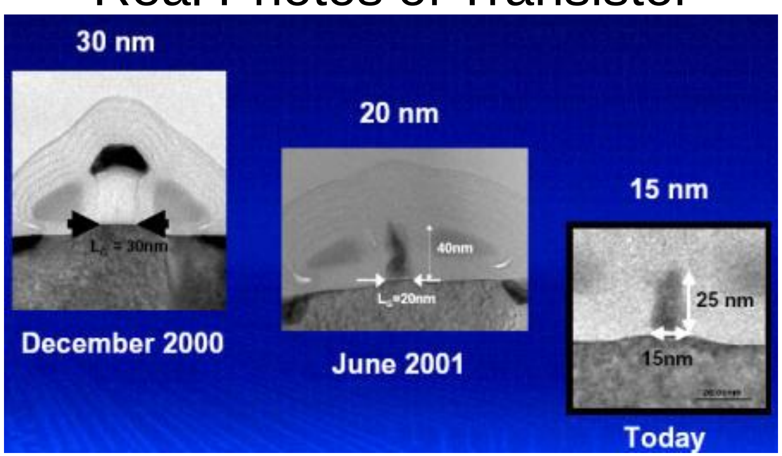
From Sand to Silicon- the Making of a Chip.mp4

Integrated Circuit:

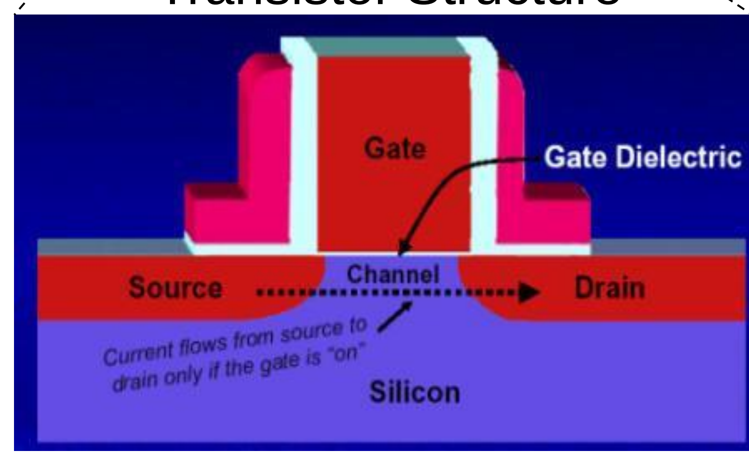
- Network of Transistors



Real Photos of Transistor

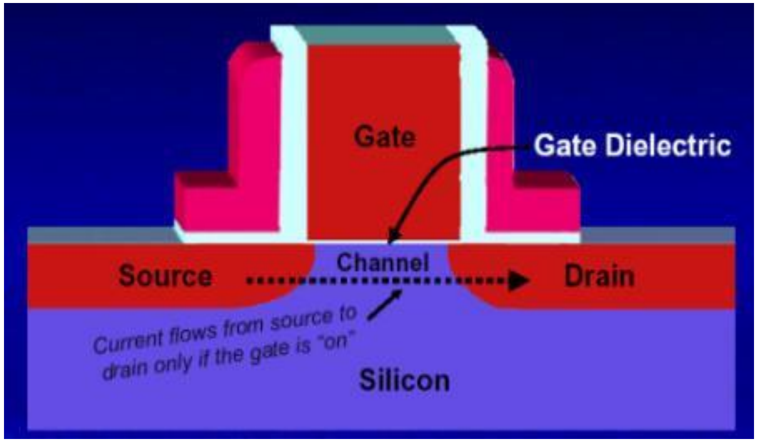


Transistor Structure



CMOS Circuits Technology

- How are logic gates physically implemented using CMOS technology?
- Basic element: **MOS transistor**
- Transistor Structure
  - 3 terminals in MOS transistors
    - **G**: Gate
    - **S**: Source
    - **D**: Drain



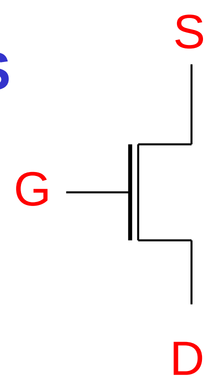
- 2 types of transistors:
  - **n-channel (nMOS)** and **p-channel (pMOS)**
  - Type depends on the semiconductor materials used to implement the transistor (beyond our scope…).

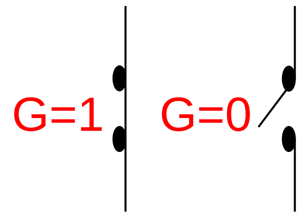
MOS Transistors as Switches

- To understand the behavior of a MOS transistor we view pMOS and nMOS transistors as <u>switches</u>

<u>Transistor Symbol</u> <u>Switch Model of MOS Transistors</u>

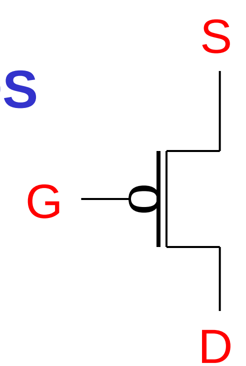
nMOS

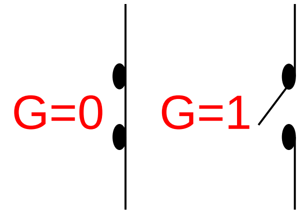




if G = 1 then switch is ON
If G = 0 then switch is OFF

pMOS

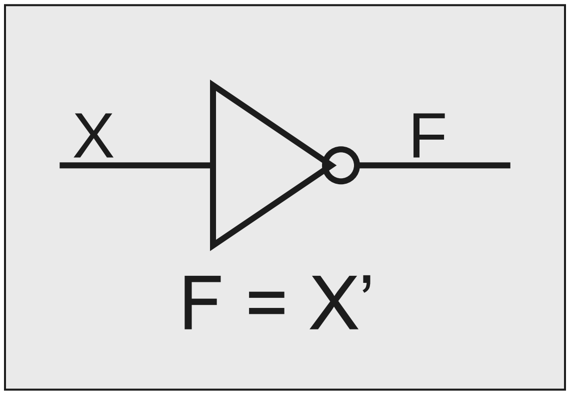




if G = 0 then switch ON
If G = 1 then switch is OFF

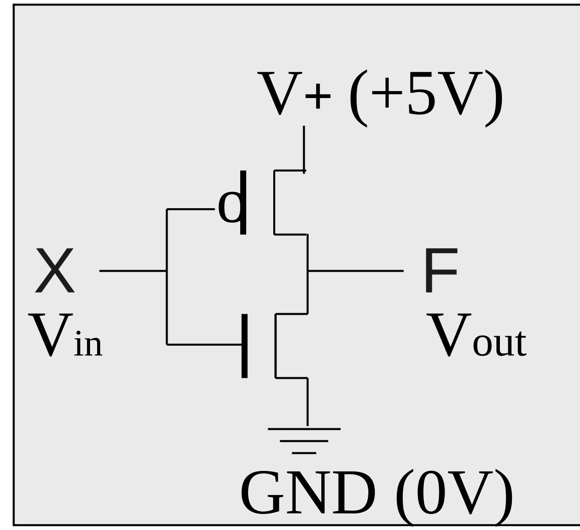
NOT (Inverter) Gate as CMOS circuit

Logic Symbol



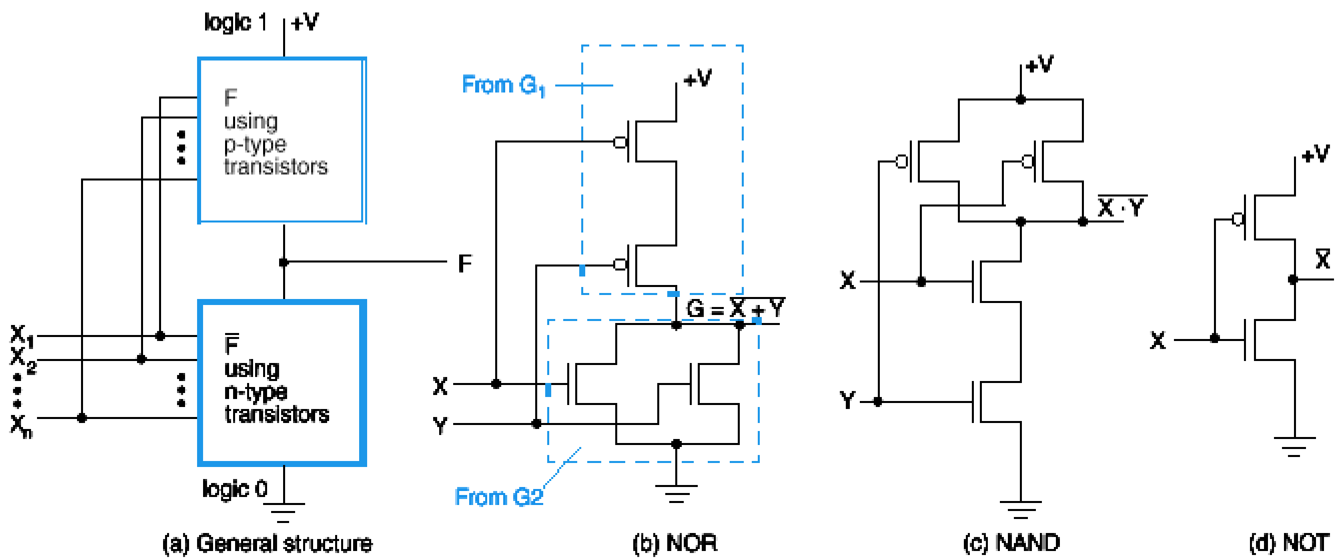
implementation

Transistor-level CMOS circuit



- CMOS Circuit Operation:
  - X=0 → pMOS switch is ON. nMOS switch OFF., i.e., conducts and draws from V+ → F=1
  - X=1 → pMOS switch is OFF. nMOS switch is ON, i.e., conducts and draws from GRD → F=0

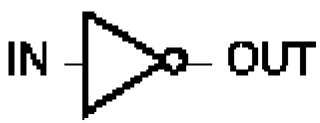
Basic Gates as CMOS Circuits



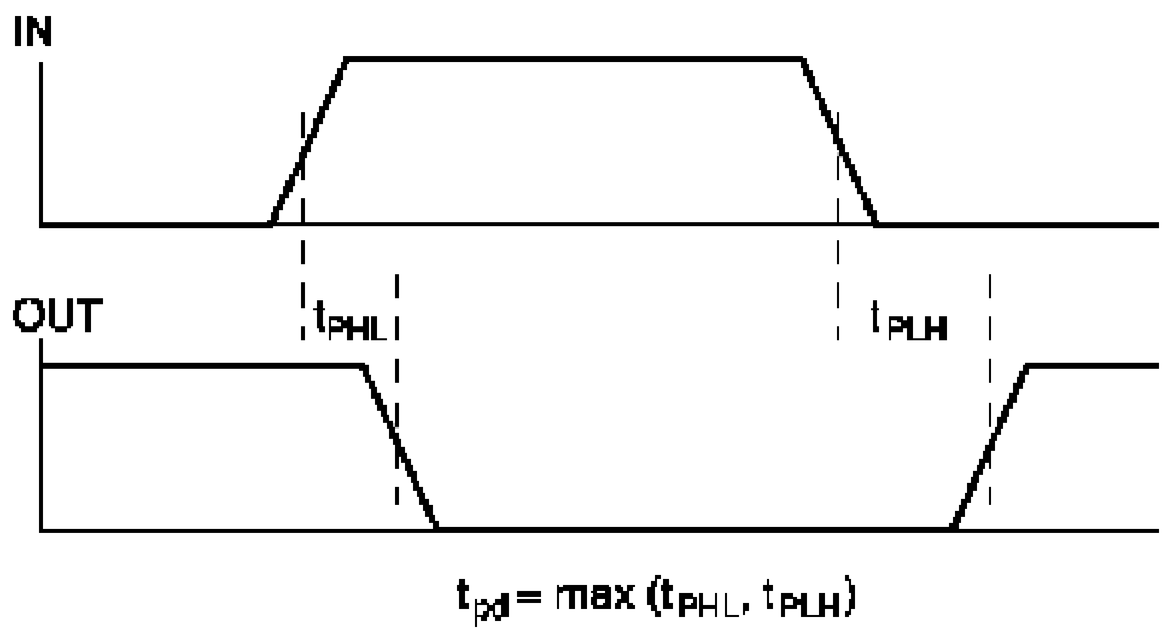
- CMOS technology implements physically digital logic circuits using NAND, NOR, and NOT gates, i.e., logic basis NAND and/or NOR is used. Why?
- Because NAND, NOR, and NOT gates are very easy to build as CMOS circuits (see above).

Propagation Delay of Gates

- One of the most important design parameters
- The propagation delay (t_{pd}) determines the gate's speed
- t_{PHL}: high-to-low propagation time
- t_{PLH}: low-to-high propagation time
- $t_{pd} = \max(t_{PHL}, t_{PLH})$



IN		OUT
0		1
1		0

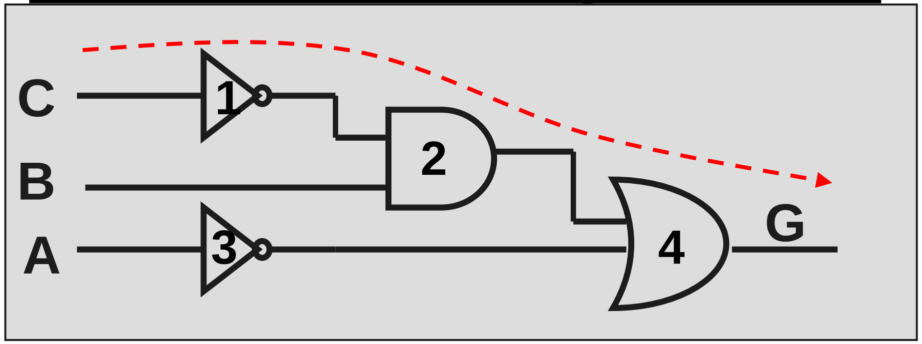


Propagation Delay for an Inverter

Propagation Delay of Logic Circuits

- The propagation delay (T_{pd}) determines the circuit's speed
- T_{pd} can be calculated as follow:
  - Find the longest path from an input of the circuit to an output of the circuit
  - Make a sum of the propagation delays (t_{pd}) of the gates on the longest path
- Example:
  - Longest Path
    - From input C to output G
  - Propagation Delay (T_{pd})
    - $\mathbf{T_{pd} = t^1_{pd} + t^2_{pd} + t^4_{pd}}$

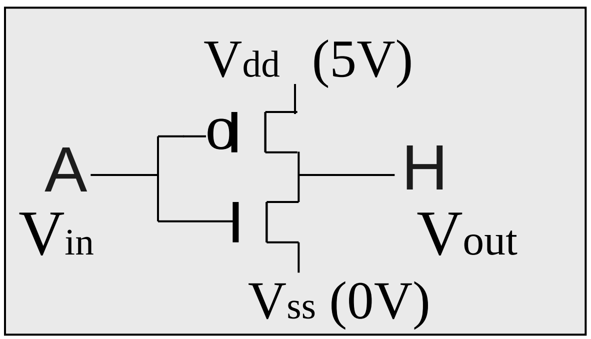
Combinational Logic Circuit



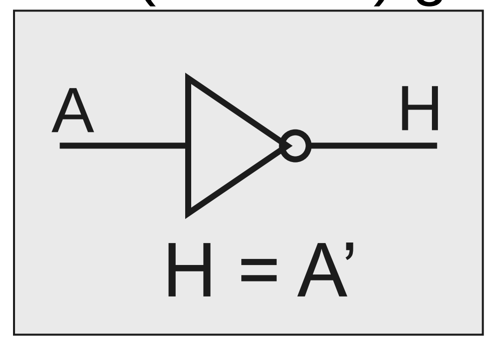
Basic Assumption for Logic Gates

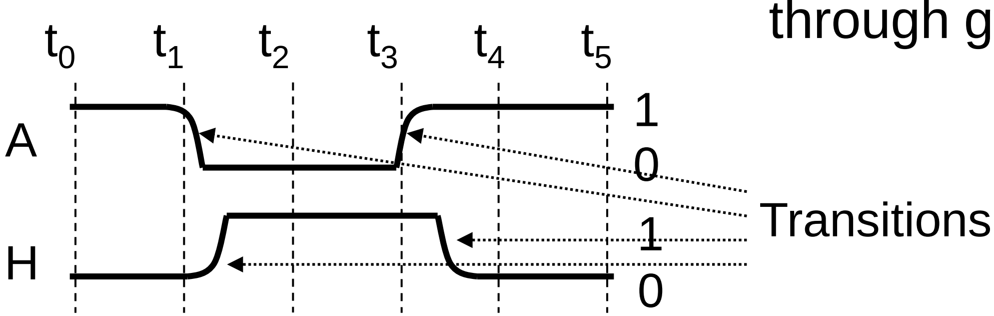
- We have seen that real logic gates have propagation delay
- However, when we design and functionally analyze logic circuits at gate level we assume that gates are ideal,
  - i.e., do not suffer from the physical propagation delays that are inherent in transistors

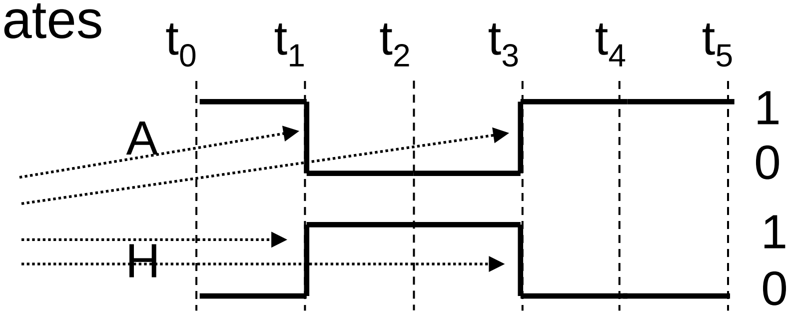
NOT (Inverter) gate



Basic Assumption:
Zero time for signals to propagate through gates







Response is not instantaneous

Response is instantaneous