Lecture 3

Logic Gates

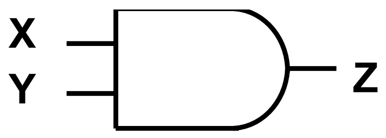
Learning outcomes

- Describe the operation of logic gates
- Express the operation of logic gates using truth table and Boolean Algebra
- Construct timing diagrams showing the proper time relationship between inputs and outputs for logic gates
- Define gate propagation delay, fan-in and fan-out
- Understand logic optimization trade-offs

Combinational logic gates

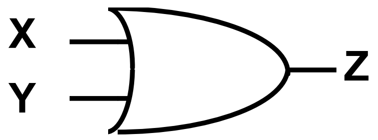
- We have seen that the three basic Boolean operations (OR, AND, NOT) can be used to describe any logic circuit but there are the logic gates.

- AND $X \bullet Y$



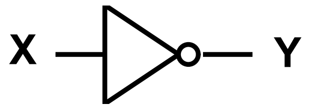
X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

- OR $X + Y$



X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

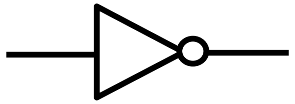
- NOT $\overline{X}$

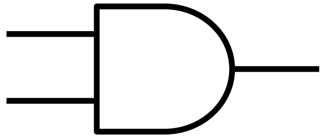


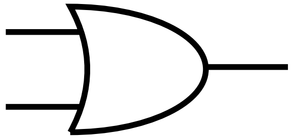
X	Y
0	1
1	0

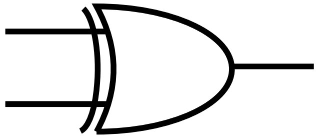
Logic Gates and symbols

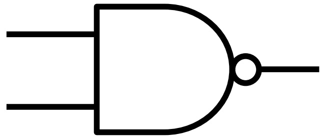
- Basic building blocks to implement Boolean functions

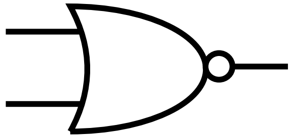

NOT gate (inverter)

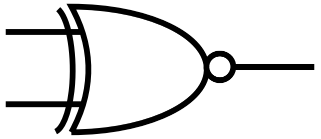

AND gate


OR gate


XOR gate

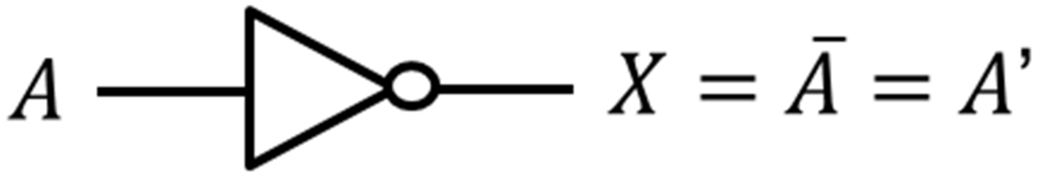

NAND gate


NOR gate


XNOR gate

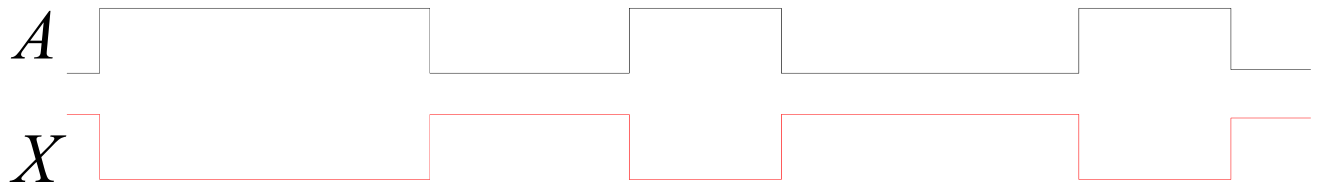
The inverter

- The inverter has a single input and a single output.
- The inverter performs the Boolean **NOT** operation. When the input is LOW (0), the output is HIGH (1); when the input is HIGH (1), the output is LOW (0).
- The **NOT** operator is denoted by (') or an overbar (‾).



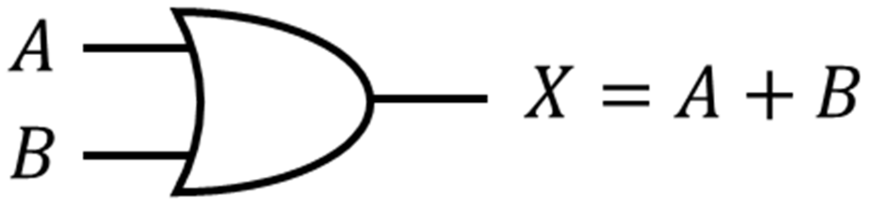
$X = \bar{A} = A'$

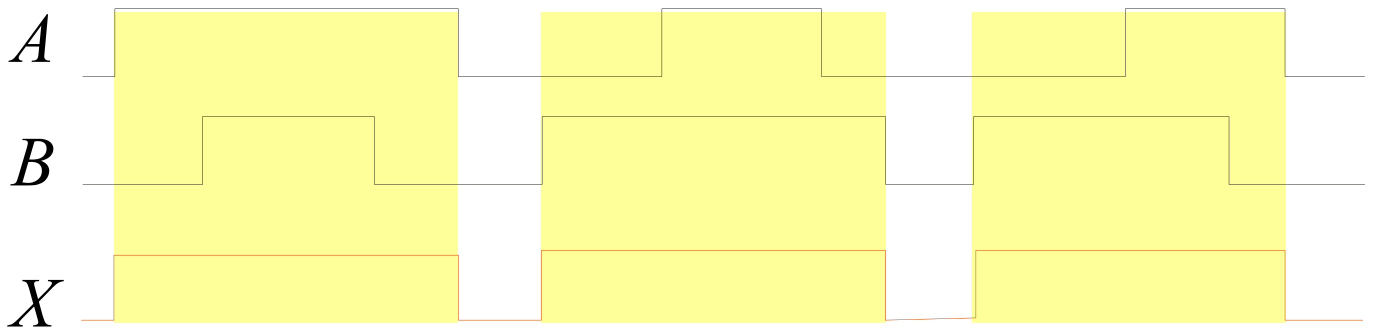
A	X
0	1
1	0



The OR gate

- The **OR gate** produces a HIGH output if any input is HIGH; if all inputs are LOW, the output is LOW.
- The truth table of a 2-input OR gate is given below.
- The **OR** operation is shown with a plus sign (+) between the variables. Thus, the OR operation is written as $X = A + B$.

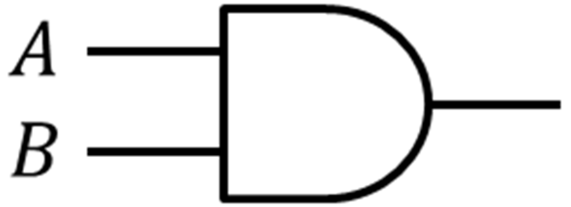




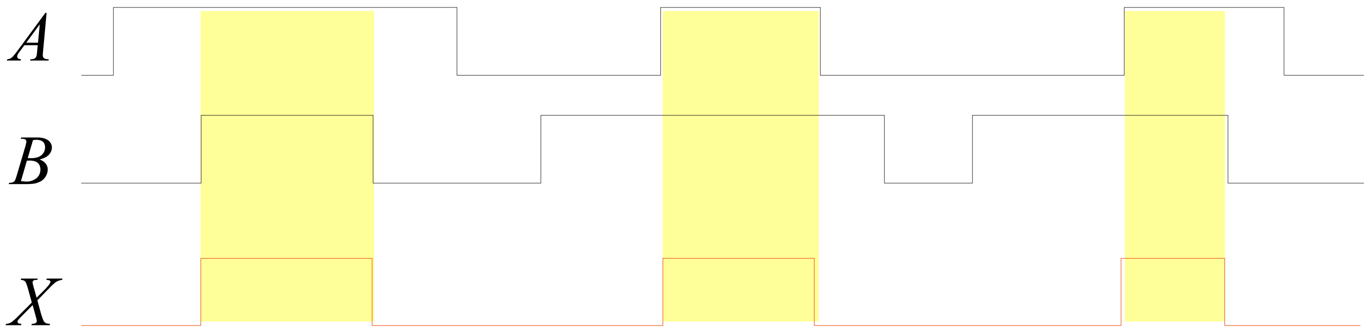
A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

The AND gate

- The **AND gate** produces a HIGH output when all inputs are HIGH; otherwise, the output is LOW.
- The truth table of a 2-input AND gate is given below.
- The **AND** operation is usually shown with a dot between the variables but it may be implied (no dot). Thus, the AND operation is written as $X = A \cdot B$ or $X = AB$.



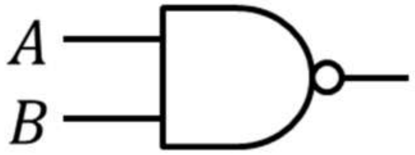
$$X = A.B = AB$$



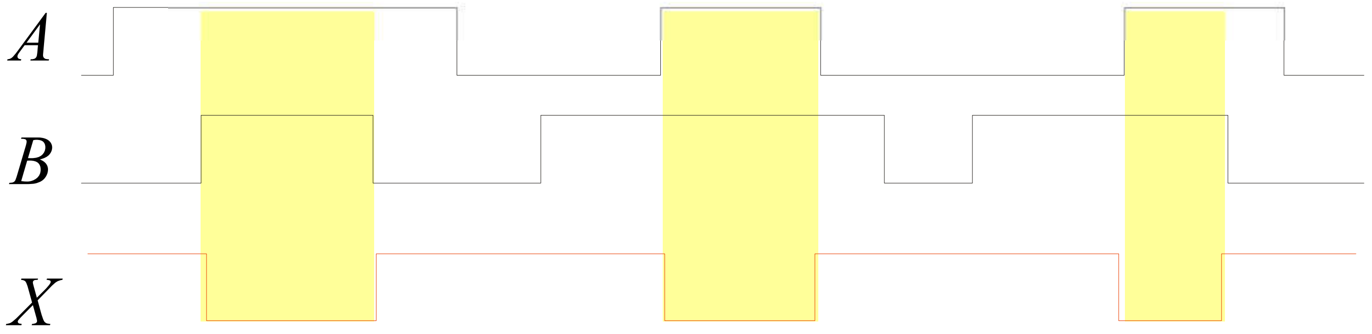
A B	X
0 0	0
0 1	0
1 0	0
1 1	1

The NAND gate

- NAND represents **N**OT **AND.** The small bubble circle represents the inversion function.
- The **NAND gate** produces a LOW output when all inputs are HIGH; otherwise, the output is HIGH. The truth-table of a 2-input NAND gate is given below:
- The **NAND** operation is shown with dot (.) between the variables, but it may be implied (no dot). The **NOT** operator is denoted by (') or an overbar (‾).



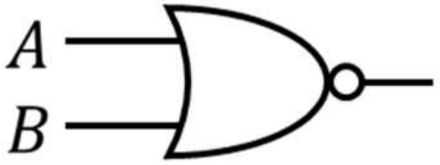
$$X = \overline{A.B} = \overline{AB} = (AB)'$$



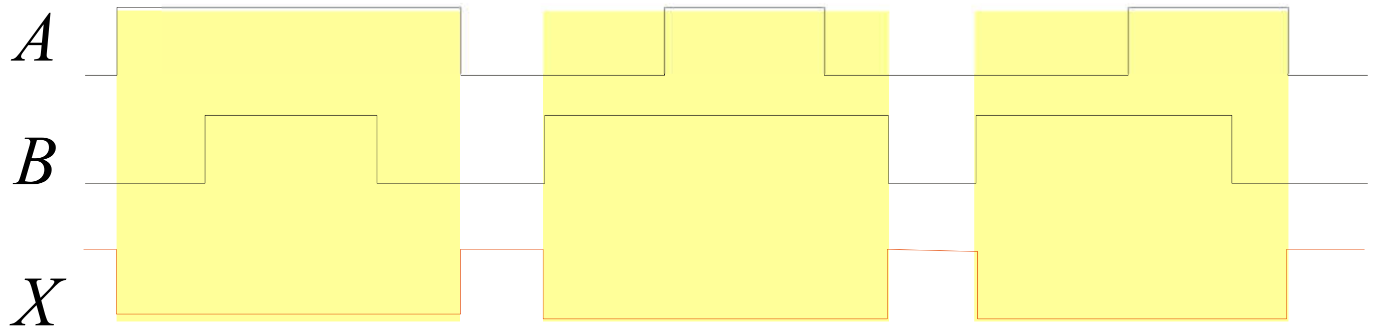
A B	X
0 0	1
0 1	1
1 0	1
1 1	0

The NOR gate

- NOR represents **N**OT **OR.** The small bubble circle represents the inversion function.
- The **NOR gate** produces a LOW output if any input is HIGH; if all inputs are HIGH, the output is LOW. The truth-table of a 2-input NOR gate is given below:
- The **NOR** operation is shown with a plus sign (+) between the variables and the **NOT** operator is denoted by (') or an overbar (‾).



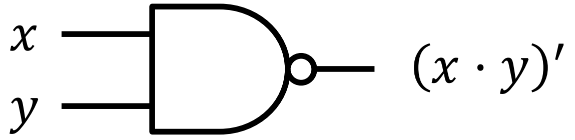
$$X = \overline{A + B} = (A + B)'$$



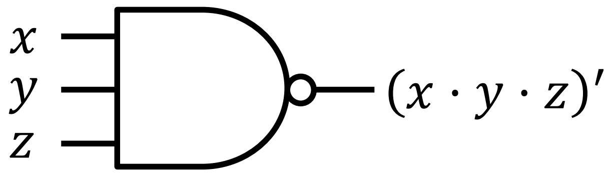
A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

Mutiple-input Logic Gates

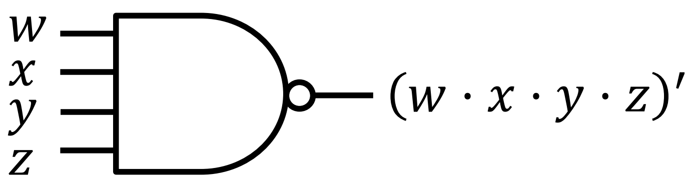
- Gates can have multiples inputs. Below are examples of multiple-inputs NAND and NOR gates:



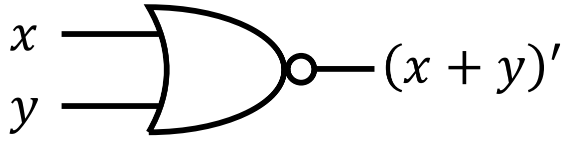
2-input NAND gate



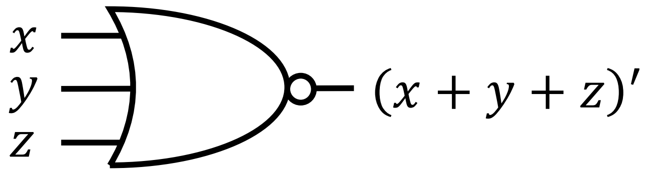
3-input NAND gate



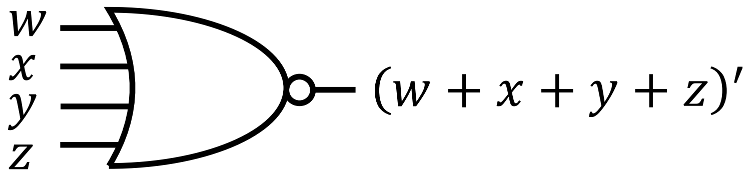
4-input NAND gate



2-input NOR gate



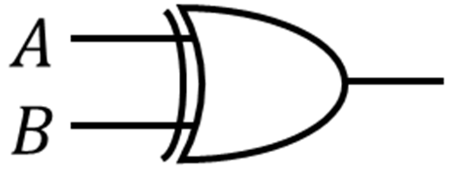
3-input NOR gate



4-input NOR gate

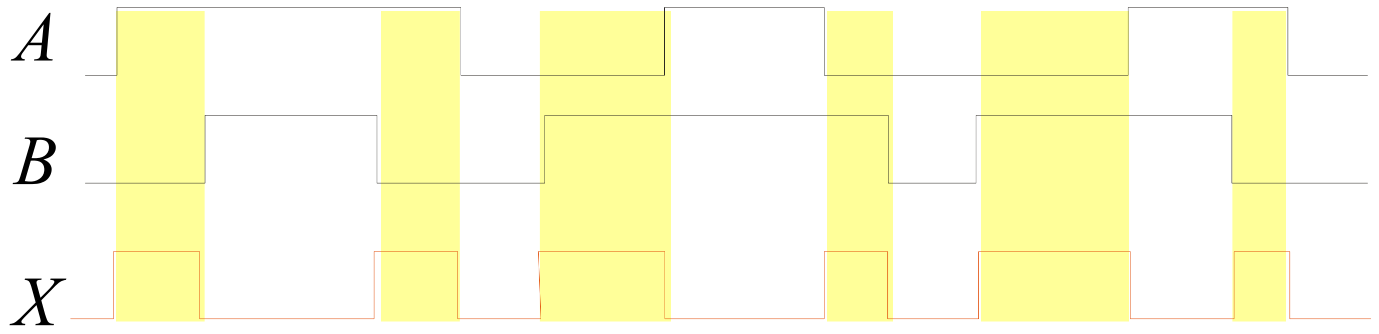
The XOR gate

- eXclusive **OR** (**XOR**) is an important Boolean operation used extensively in logic circuits
- The **XOR gate** produces a HIGH output only when both inputs are at opposite logic levels. The truth table of a 2-input XOR gate is given below.
- The **XOR** operation can be written with a circled plus sign between the variables: *X* = *A* ⊕ *B.*



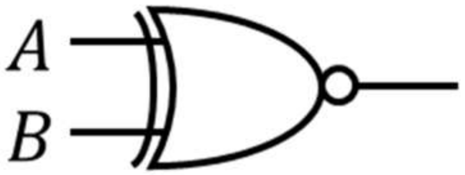
$$X = \overline{A}B + A\overline{B} = A \oplus B$$

A	B	X
0	0	0
0	1	1
1	0	1
1	1	0



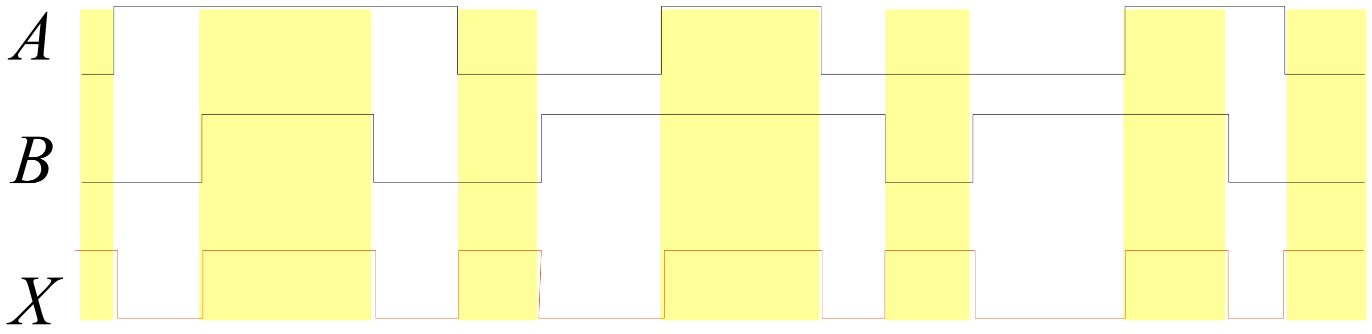
The XNOR gate

- Exclusive NOR (XNOR) is the complement of XOR.
- The **XNOR gate** produces a HIGH output only when both inputs are at the same logic level.
- The truth table of a 2-input XNOR gate is given below.
- The **NOT** operator is denoted by (') or an overbar (‾).



$$X = \bar{A}\bar{B} + AB = \overline{A \oplus B} = (A \oplus B)'$$

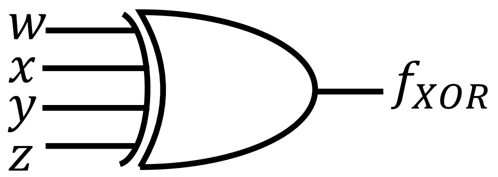
A B	X
0 0	1
0 1	0
1 0	0
1 1	1



N-Input XOR/XNOR

- The **XOR gate** produces a **HIGH (1)** output when an **odd** number of inputs are **HIGH (1)**.

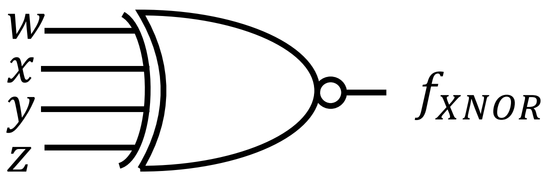
$$f_{XOR} = f_{Odd} = \sum(1,2,4,7)$$



$$f_{XOR} = x'y'z + x'yz' + xy'z' + xyz$$

$$f_{XOR} = x \oplus y \oplus z$$

- The **XNOR gate** produces a **HIGH (1)** output when an **even** number of inputs are **HIGH (1)**.



$$f_{XNOR} = f_{Even} = \sum m(0,3,5,6,9,10,12,15)$$

$$f_{XNOR} = (w \oplus x \oplus y \oplus z)'$$

w x y z	XOR	XNOR
0 0 0 0	0	1
0 0 0 1	1	0
0 0 1 0	1	0
0 0 1 1	0	1
0 1 0 0	1	0
0 1 0 1	0	1
0 1 1 0	0	1
0 1 1 1	1	0
1 0 0 0	1	0
1 0 0 1	0	1
1 0 1 0	0	1
1 0 1 1	1	0
1 1 0 0	0	1
1 1 0 1	1	0
1 1 1 0	1	0
1 1 1 1	0	1

XOR and XNOR properties

- The XOR function is: $x \oplus y = xy' + x'y$
- The XNOR function is: $(x \oplus y)' = xy + x'y'$
- $x \oplus 0 = x$ $\quad\quad x \oplus 1 = x'$
- $x \oplus x = 0$ $\quad\quad x \oplus x' = 1$
- $x \oplus y = y \oplus x$
- $x' \oplus y' = x \oplus y$
- $(x \oplus y)' = x' \oplus y = x \oplus y'$

XOR and XNOR are **associative** operations

- $(x \oplus y) \oplus z = x \oplus (y \oplus z) = x \oplus y \oplus z$
- $\left((x \oplus y)' \oplus z\right)' = (x \oplus (y \oplus z)')' = x \oplus y \oplus z$

Exclusive OR/ Exclusive NOR

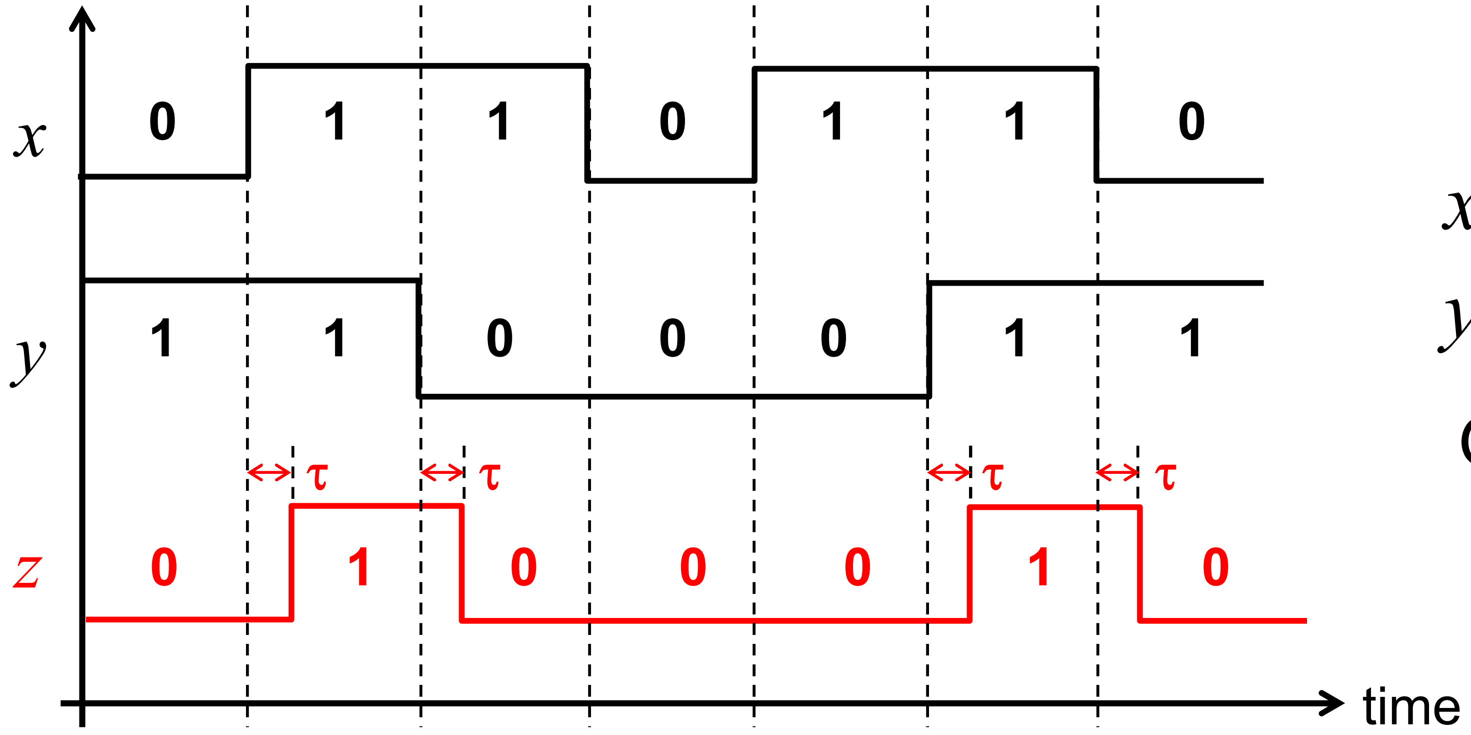
- The *eXclusive OR* (*XOR*) function is an important Boolean function used extensively in logic circuits:
  - Adders/subtractors/multipliers
  - Counters/incrementers/decrementers
  - Parity generators/checkers

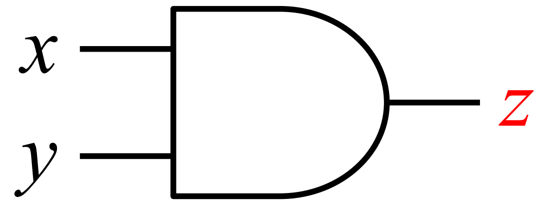
- The *eXclusive NOR* function (XNOR) is the complement of the XOR function

- XOR and XNOR gates are complex gates (built from simpler gates, such as AND, Not, etc).

Gate Delay

- In actual physical gates, if one or more input changes causes the output to change, the output change **does not occur instantaneously**.
- The delay between an input change(s) and the resulting output change is the **gate delay** denoted by τ.



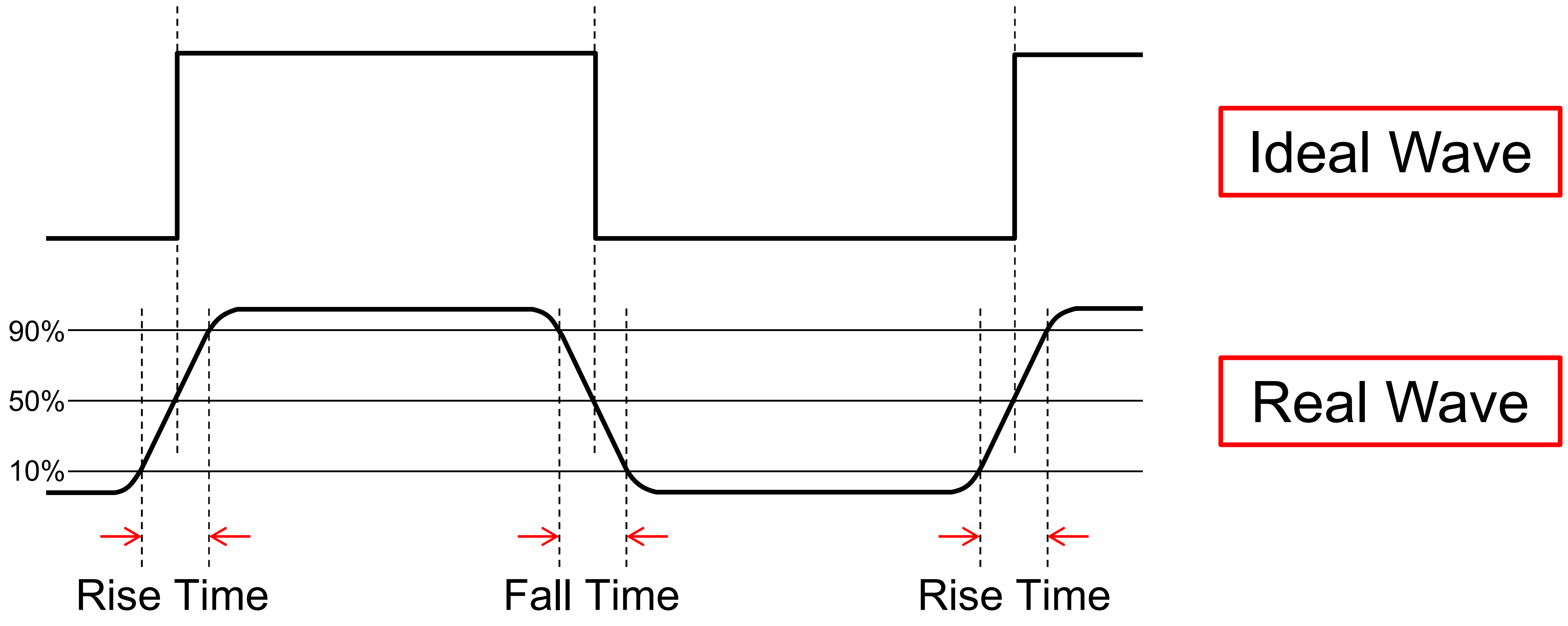


Gate delay = τ

Rise-Time and Fall-Time

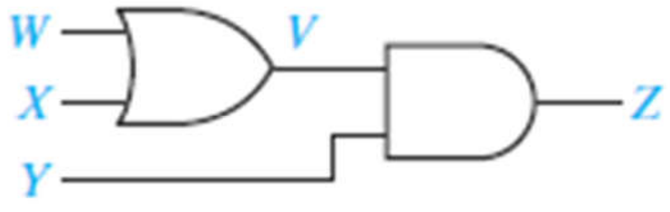
- In logic simulators, a waveform is drawn as an **ideal wave**
- The change from 0 to 1 (or from 1 to 0) is instantaneous
- In reality, a signal has a non-zero **rise-time** and **fall-time**

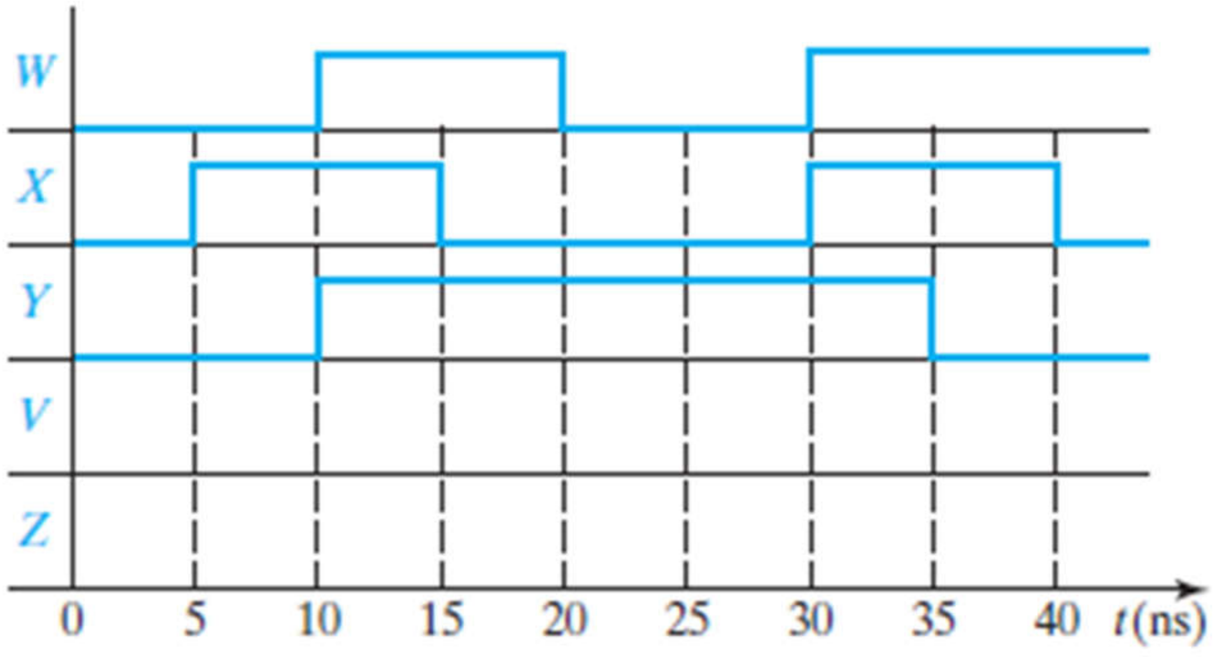
  Time taken to change from 10% to 90% of High voltage (and vice versa)



Timing Diagrams

- The timing diagrams depict the **functional behavior** of a logic circuit in a form that can be observed when the circuit is tested using instruments such as logic analyzers and oscilloscopes.
- Complete the timing diagrams for V and Z. Assume that outputs are initially at 0 and gates have a propagation delay of 5ns:



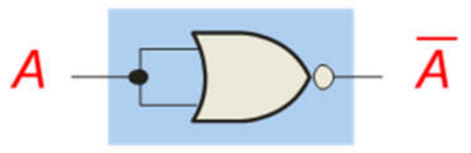


Universal Gates

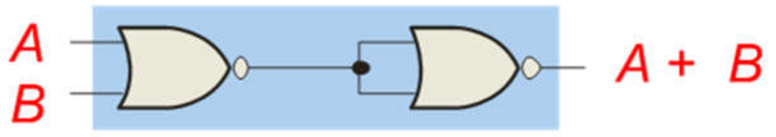
- ***Any Boolean expression can be implemented using combinations of OR gates, AND gates and INVERTERs.***
- ***Universal gate*** - a gate type that can implement any Boolean function.
- NAND/NOR gates are **universal gates** because one can implement any logic function using only NOR or only NAND:
  - NOT can be implemented using only NAND or only NOR gates
  - AND can be implemented using only NAND or only NOR gates
  - OR can be implemented using only NAND or only NOR gates
- NAND/NOR (rather than AND/OR) gates are natural gate implementations for CMOS technology considering chip area and speed. More on this later.

Universal Gate - NOR

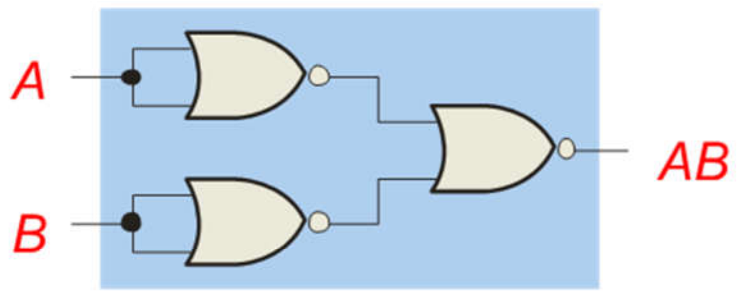
- NOR is a universal gate because one can implement any logic function using only NOR gates.



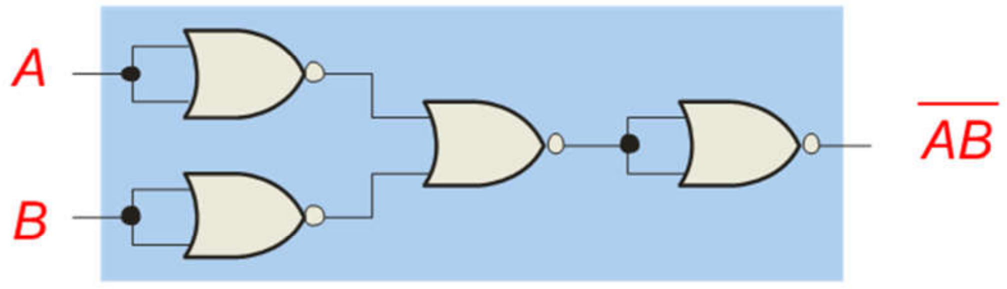
Inverter



OR gate



AND gate

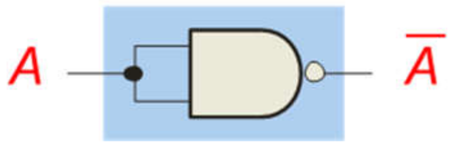


NAND gate

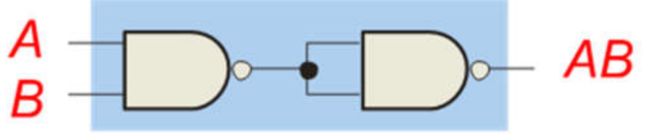
- NOR gate is preferred to OR gate because it can be implemented more efficiently in CMOS technology
  - In terms of chip area and speed

Universal Gate - NAND

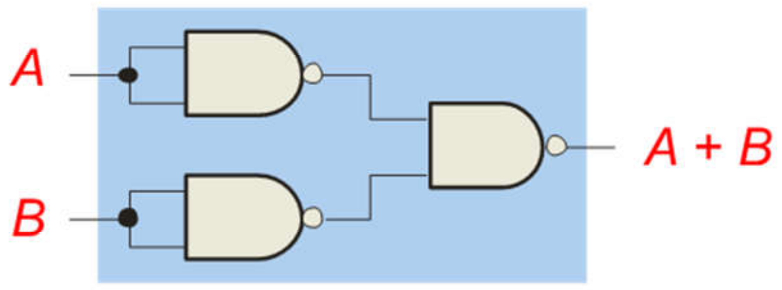
- NAND is a universal gate because one can implement any logic function using only NAND gates



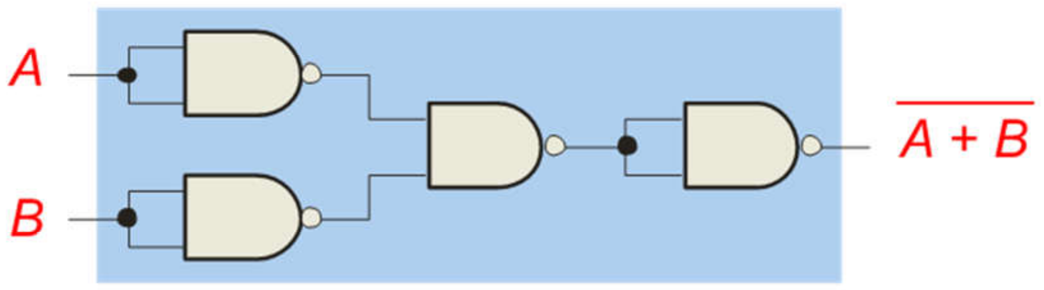
Inverter



AND gate



OR gate



NOR gate

- NAND gate is preferred to AND gate because it can be implemented more efficiently in CMOS technology
  - In terms of chip area and speed

Standard Library

- Library of gates and their physical characteristics
- Example:

Gate	Delay (ps)	Area (μ^2)
Inverter	20	10
Buffer	40	20
AND2	50	25
NAND2	30	15
OR2	55	26
NOR2	35	16
AND4	90	40
NAND4	70	30
OR4	100	42
NOR4	80	32

- Observations:

  In current technology (CMOS), inverting gates are faster and smaller

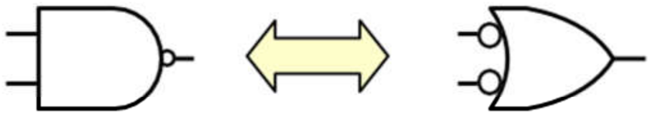
  Delay and area grow with number of inputs

DeMorgan Equivalents

- Inverting output of AND gate = inverting the inputs of OR gate
- Inverting output of OR gate = inverting the inputs of AND gate
- A function’s inverse is equivalent to inverting all the inputs and changing AND to OR and vice versa

A	B	Out
0	0	1
0	1	1
1	0	1
1	1	0

$\overline{A \bullet B}$

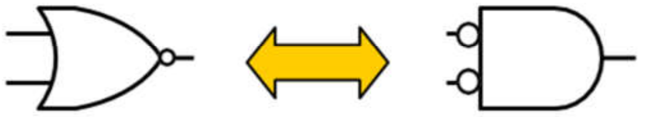


$\overline{A} + \overline{B}$

A	B	Out
0	0	1
0	1	1
1	0	1
1	1	0

A	B	Out
0	0	1
0	1	0
1	0	0
1	1	0

$\overline{A + B}$



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

A	B	Out
0	0	1
0	1	0
1	0	0
1	1	0

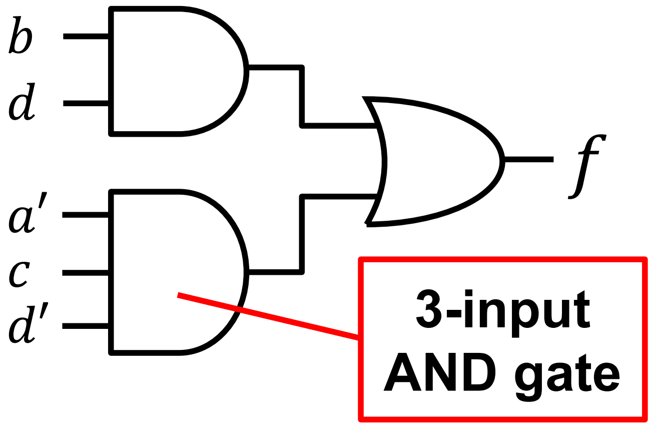
NAND-NAND implementation

- Consider the following sum-of-products expression:

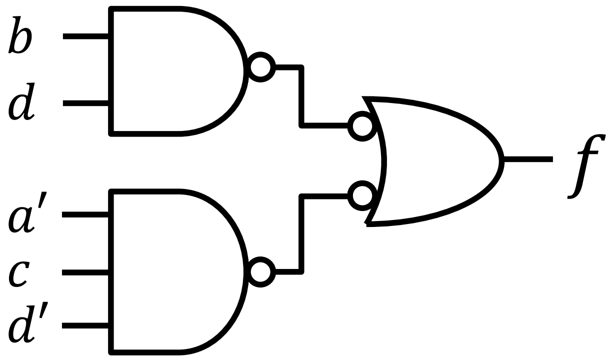
$$f = bd + a'cd'$$

- The corresponding 2-level **AND-OR** circuit can easily be converted to a 2-level **NAND-NAND implementation**

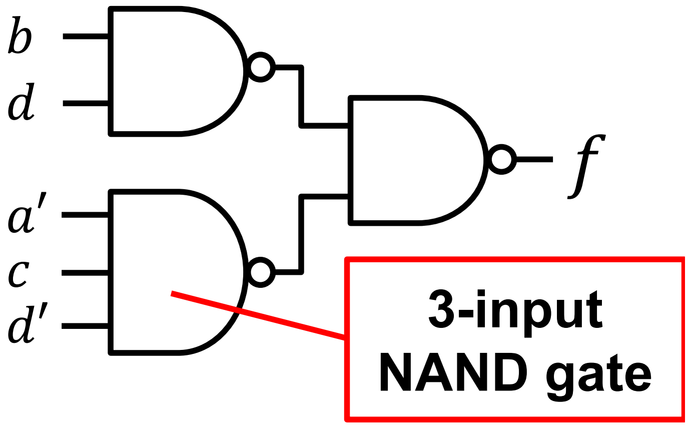
2-Level AND-OR



Inserting Bubbles



2-Level NAND-NAND



Two successive bubbles (inverters) on same line cancel each other

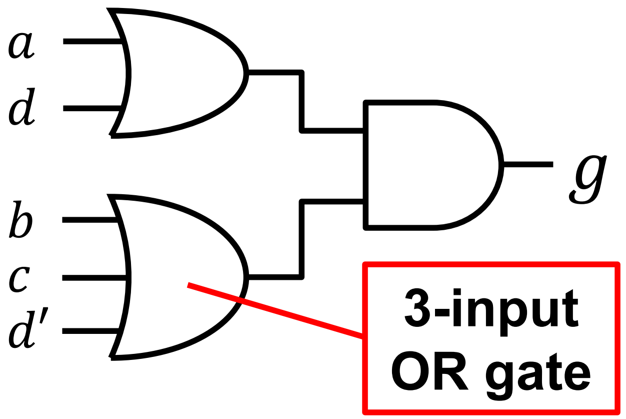
NOR-NOR implementation

- Consider the following product-of-sums expression:

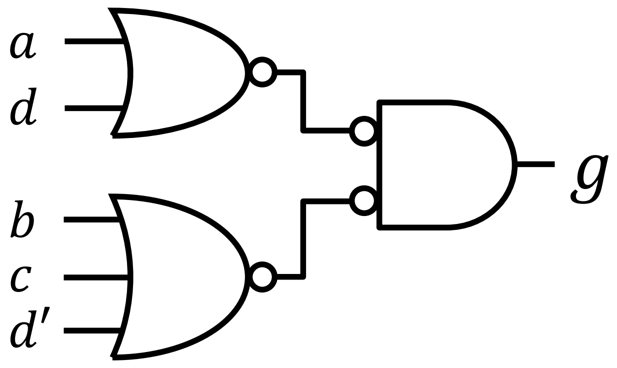
$$g = (a + d)(b + c + d')$$

- The corresponding 2-level **OR-AND** circuit can easily be converted to a 2-level **NOR-NOR implementation**

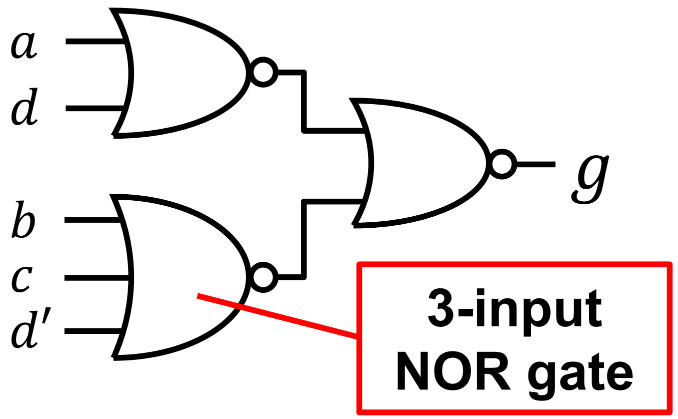
2-Level OR-AND



Inserting Bubbles



2-Level NOR-NOR



Two successive bubbles (inverters) on same line cancel each other

Fan-in of a logic gate

- The fan-in of a gate is its number of inputs.
- A 3-input OR gate has a fan-in of 3.
- The propagation delay of a gate increases significantly (e.g. quadratically) with the fan-in of the gate: a 3-input NAND gate will be slower than a 2-input NAND gate fabricated with the same technology.
- Logic gates with higher fan-in are more complex as they require more transistors to implement. The increased propagation delay of these gates is due to the higher resistance and capacitance introduced by transistors.
- Logic gates with a fan-in greater than 4 should be avoided.
- Logic gates with higher fan-in can help reduce the depth (number of levels) of a logic circuit.

Standard Library

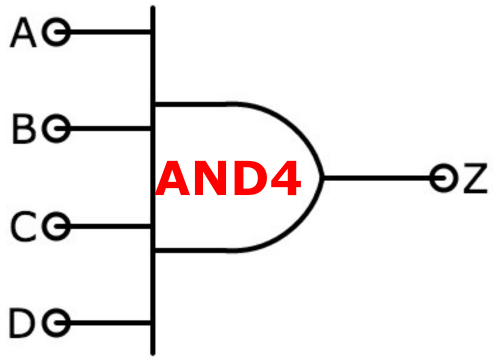
- Library of gates and their physical characteristics
- Example:

Gate	Delay (ps)	Area (μ^2)
Inverter	20	10
Buffer	40	20
AND2	50	25
NAND2	30	15
OR2	55	26
NOR2	35	16
AND4	90	40
NAND4	70	30
OR4	100	42
NOR4	80	32

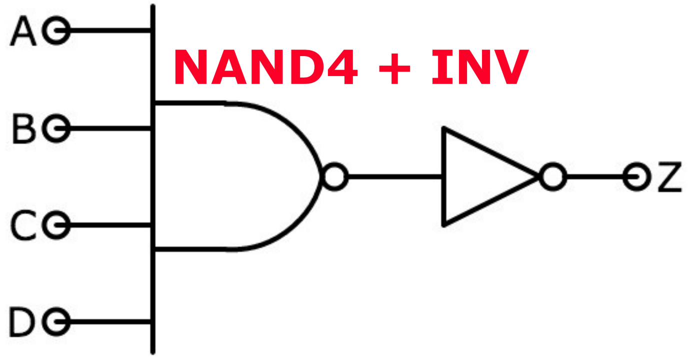
- Observations:
  - In current technology (CMOS), inverting gates are faster and smaller
  - Delay and area grow with number of inputs

Design Tradeoffs: Delay vs Size

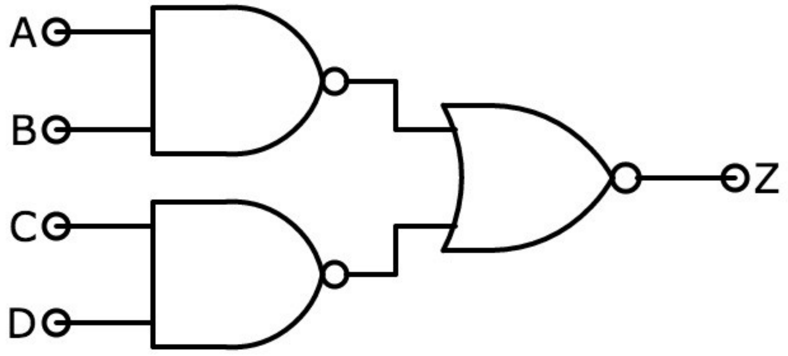
Gate	Delay (ps)	Area (μ^2)
Inverter	20	10
Buffer	40	20
AND2	50	25
NAND2	30	15
OR2	55	26
NOR2	35	16
AND4	90	40
NAND4	70	30
OR4	100	42
NOR4	80	32



t_{PD} = 90 ps
size = 40μ^2



t_{PD} = 90 ps
size = 40μ^2

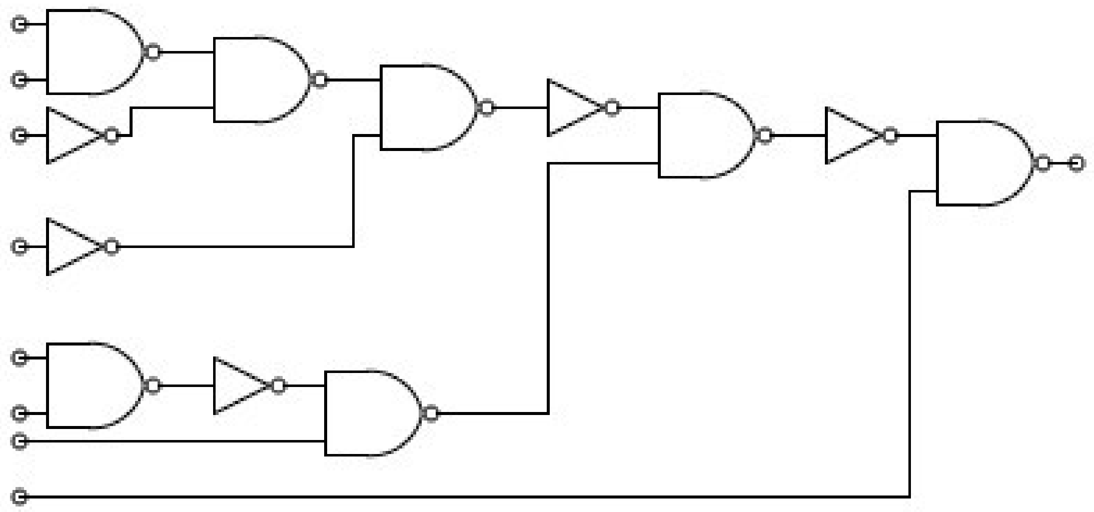


t_{PD} = 1 NAND2 + NOR2 = 65 ps
size = 2 NAND2 + NOR2 = 46μ^2

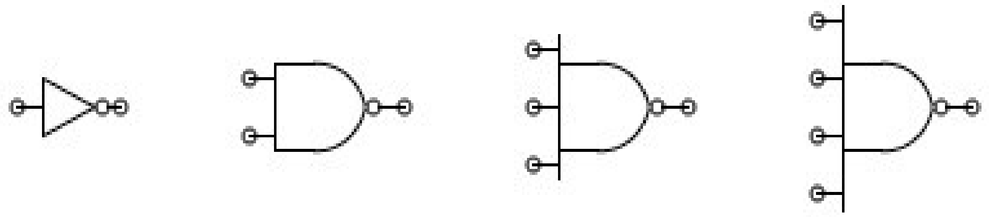
2×NAND2 + NOR2

Mapping a Circuit to a Standard Cell Library

- We would like to minimize the area required for the following circuit implementation:



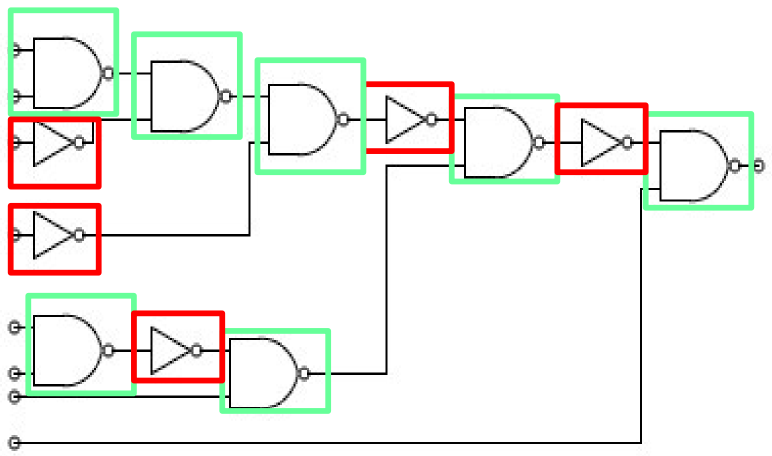
- You need to use gates from the following standard cell library:



- Area: 2 3 4 5

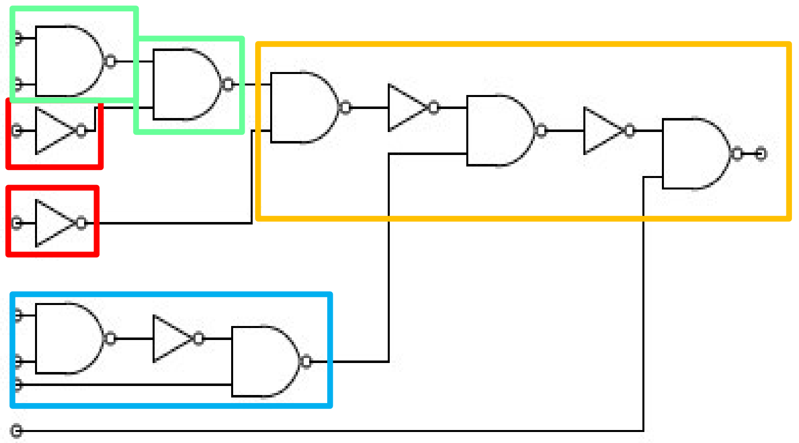
Mapping a Circuit to a Standard Cell Library

- Possible implementations:



7 NAND2 (3) = 21
5 INV (2) = 10

Total area cost: 31



2 INV = 4
2 NAND2 = 6
1 NAND3 = 4
1 NAND4 = 5

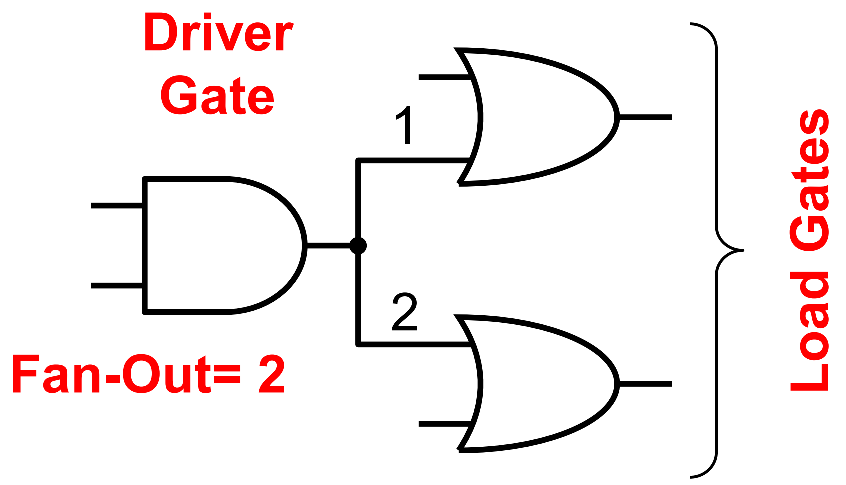
Total area cost: 19

Logic Optimization Takeaways

- Synthesizing an optimized circuit is a very complex problem
  - Boolean simplification
  - Mapping to cell libraries with many gates
  - Multi-dimensional tradeoffs (e.g., minimize area-delay-power product)
- Only possible to do by hand for small circuits.
- Instead, hardware designers write circuits in a **hardware description language**, and use a **synthesis tools** to derive optimized implementations

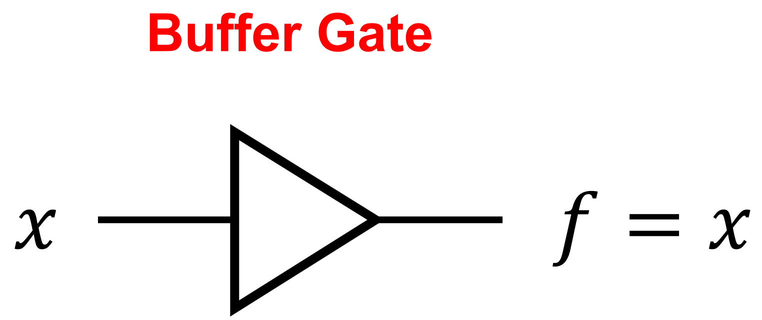
Fan-out of a logic gate

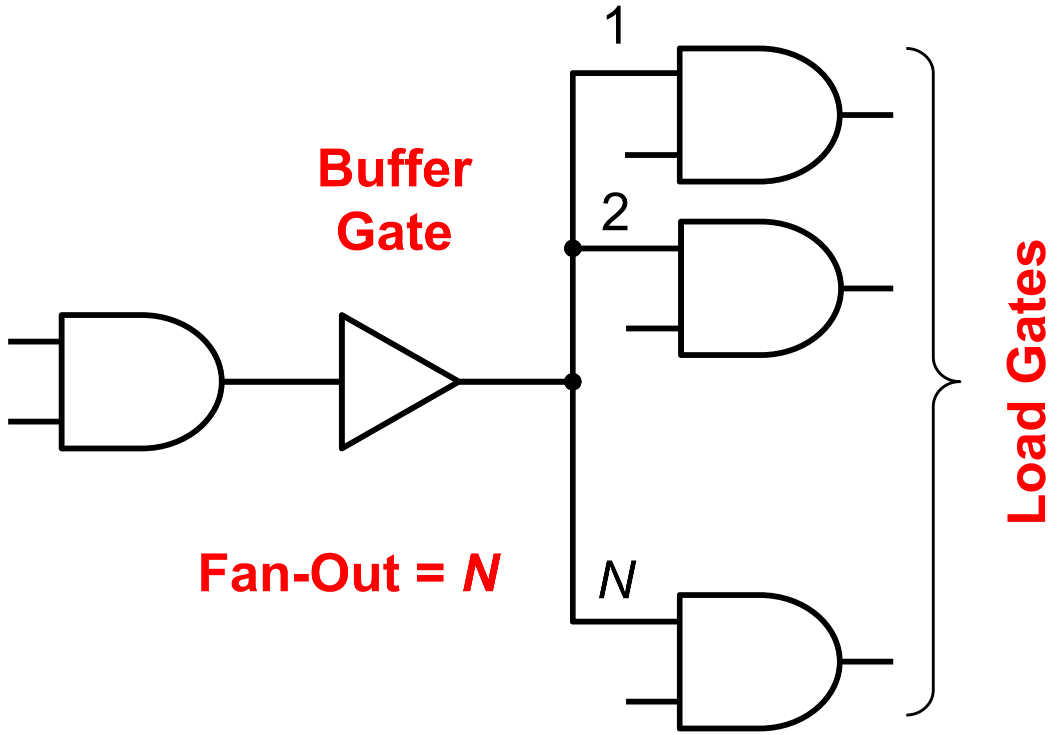
- In digital circuits, it is common for the output of one gate (called driver gate) to be connect to the inputs of several load gates
- The fan-out of a gate is the number of gate inputs it can feed
- There is a limit on the maximum fan-out of a gate
- The output of a driver gate can supply a limited amount of current.
- Each input of a load gate consumes a certain amount of current.
- Therefore, the driver gate can only feed a limited number of load gates.



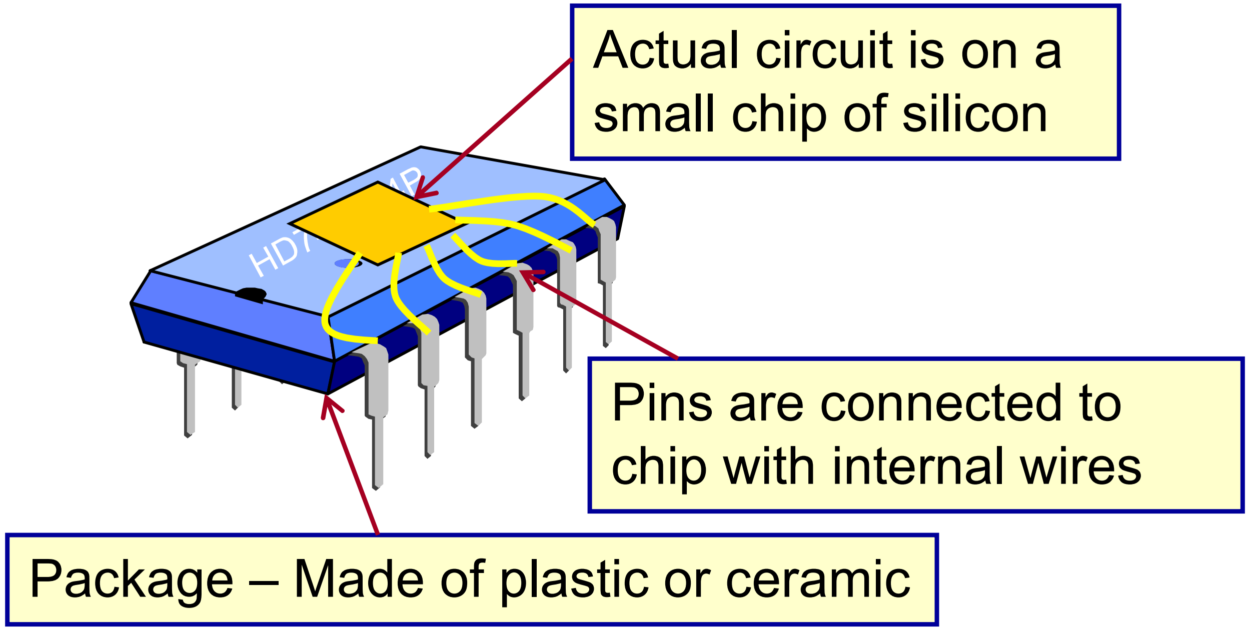
Buffer Gate

- A buffer is used to improve circuit voltage levels and increase fan-out and speed of circuit.
- Buffer Gate
  - Output f = Input x
- Buffer provides drive capability
  - Used to amplify an input signal
  - High current output
  - Increases the Fan-Out
- Buffer gate increases the propagation delay of a circuit

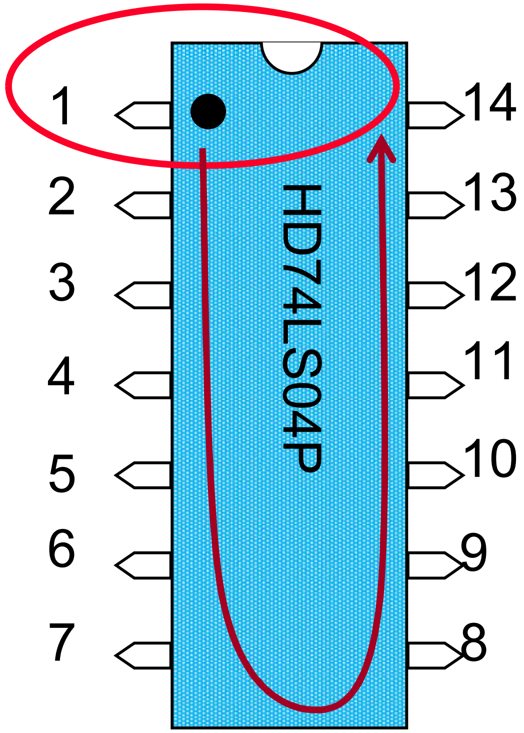




Logic Chips



Actual circuit is on a small chip of silicon
Pins are connected to chip with internal wires
Package – Made of plastic or ceramic
HD7
4P

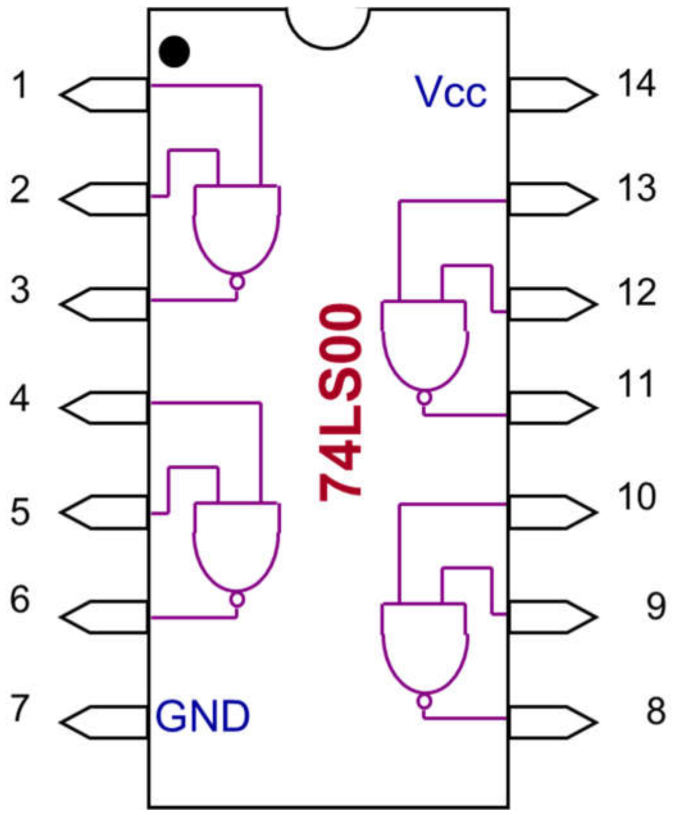


- Pin number one is identified with a dot or notch
- Pins are numbered in a counter-clockwise fashion
- Standard chips use either **TTL** (Transistor-Transistor Logic), or **CMOS** transistors

Logic Chips – 7400 NAND

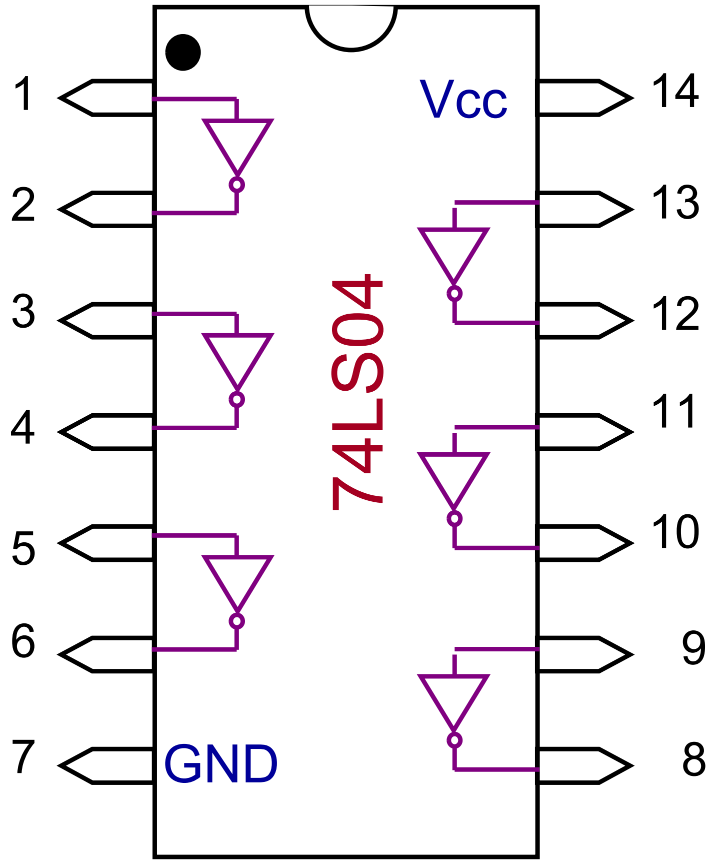
- Part number – 74xx00 is a 7400 Quad Nand Gate
- There are four 2-input NAND gates in a 7400.

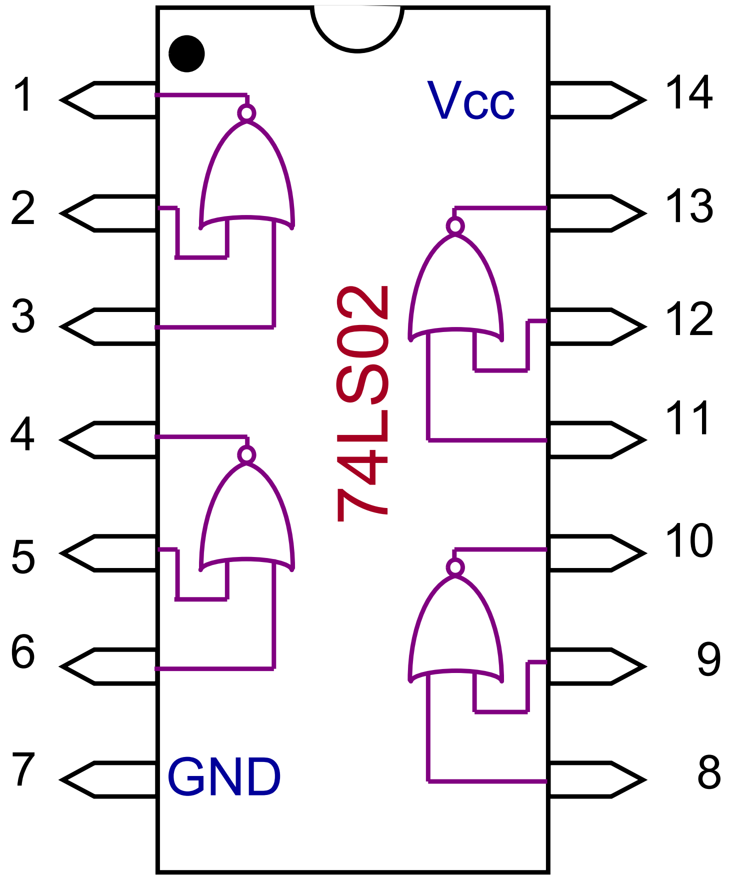
LS	– Low-Power Schottky TTL
HC	– CMOS
HCT	– TTL-compatible CMOS
F	– Fast TTL



- All chips have connections to Vcc (+5) and GND

Logic Chips – Inverters and NORs





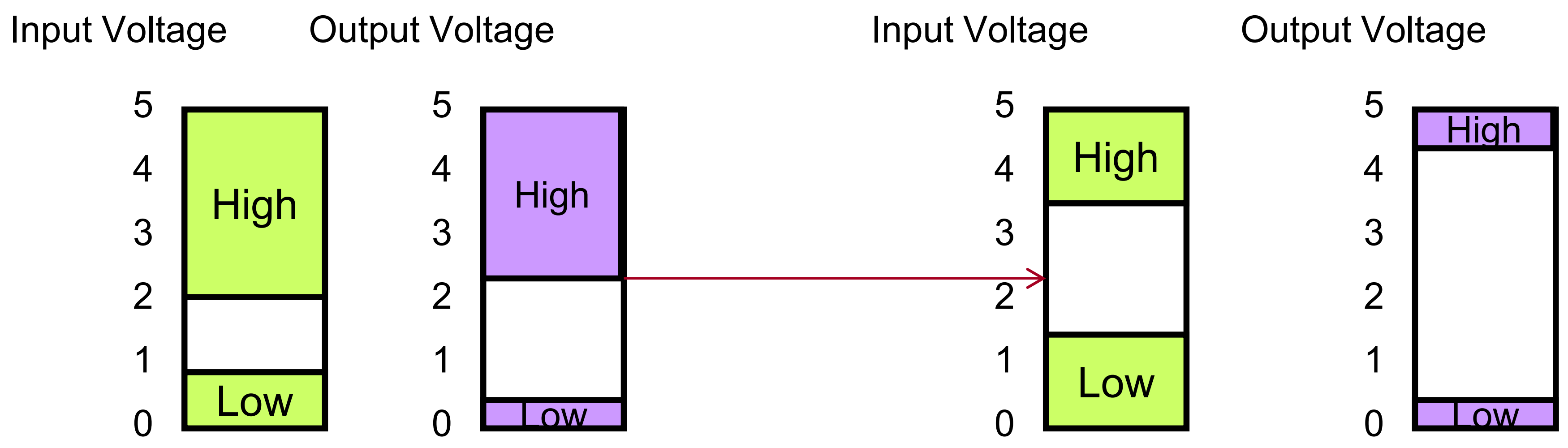
Logic Families

5V TTL

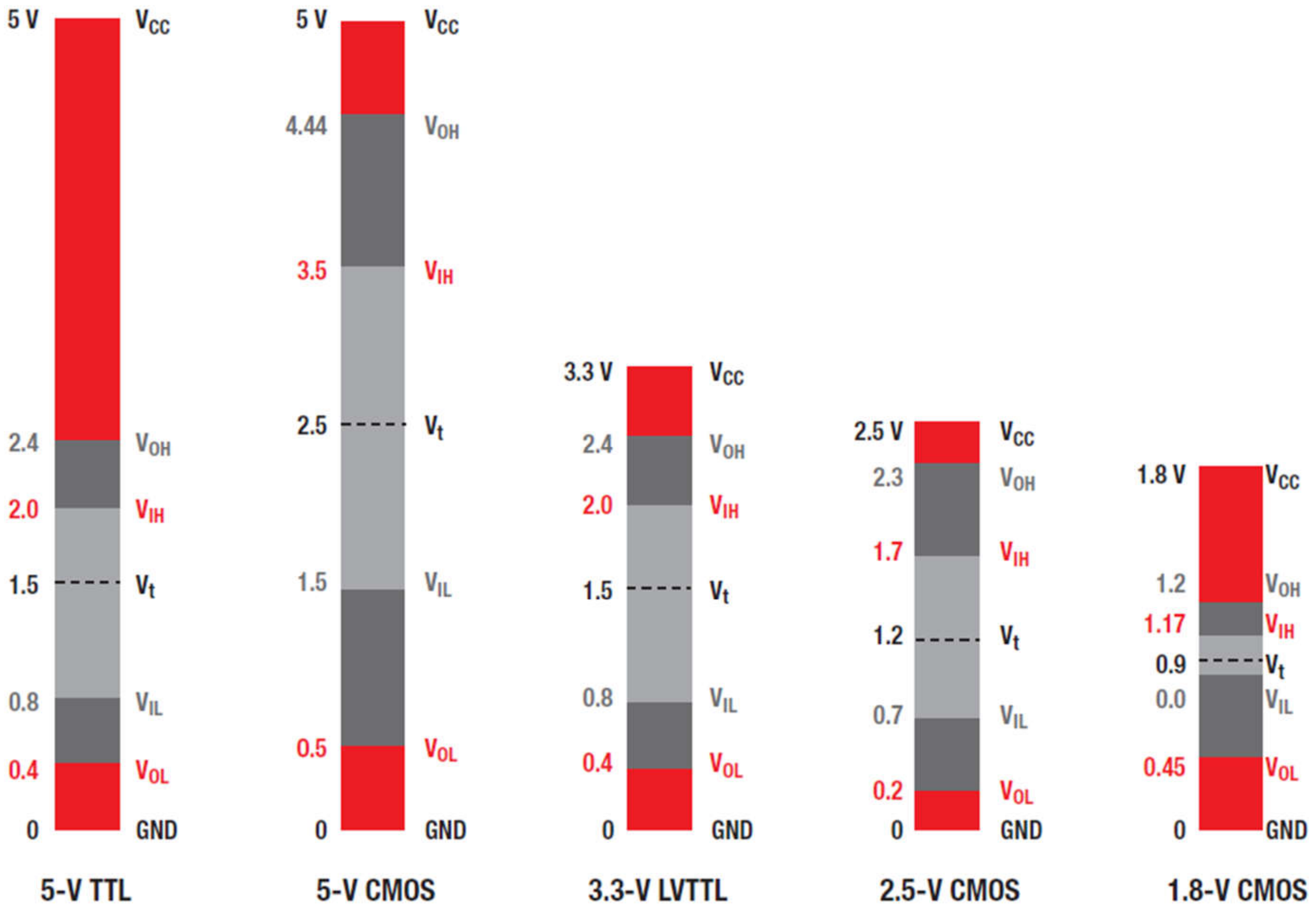
Standard
Low-Power Schottky – LS
Fast - F

5V CMOS

Standard - HC
HCT – TTL compatible



Logic Families



Positive and Negative logic

- Recall it is up to us humans to assign meaning to the two voltage levels
  - Thus, far we've used (unknowingly) the positive logic convention where 1 (High voltage) means true and 0 (Low voltage) means false
  - In negative logic 0 means true and 1 means false

x_1	x_2	x_3	x_o
0V	0V	0V	0V
0V	0V	5V	0V
0V	5V	0V	0V
0V	5V	5V	0V
5V	0V	0V	0V
5V	0V	5V	0V
5V	5V	0V	0V
5V	5V	5V	5V

Voltages

x_1	x_2	x_3	x_o
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Positive Logic

x_1	x_2	x_3	x_o
1	1	1	1
1	1	0	1
1	0	1	1
1	0	0	1
0	1	1	1
0	1	0	1
0	0	1	1
0	0	0	0

Negative Logic

Acknowledgments

- Credit is acknowledged where credit is due. Please refer to the full list of references.