(T)EE2026

Digital Fundamentals

Logic Gates

Prof. Massimo Alioto

Dept of Electrical and Computer Engineering

Email: ***massimo.alioto@nus.edu.sg***

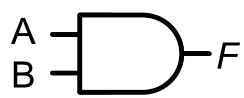
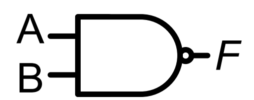
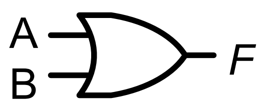
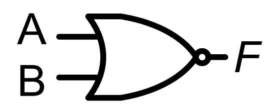
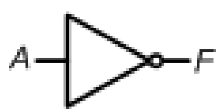
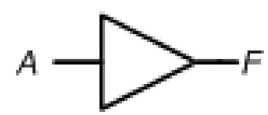
Outline

- Logic gate introduction
  - AND/NAND, OR/NOR, NOT/Buffer, XOR/NXOR
  - different levels of description (Boolean, truth table, graphical, Verilog)
- Implementation of Boolean function using gates
  - different levels of description (Boolean, graphical, Verilog)
- Design simplification by algebra manipulation
- Positive and negative logic
- Commercial logic gates

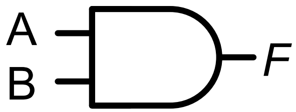
Logic Gate Introduction

- Logic gates are digital circuits that implement the Boolean operations

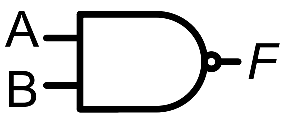
Basic Logic Gates:

Gate	Symbol	Function (*F*)	Gate	Symbol	Function (*F*)
AND		$A \cdot B$	NAND		$\overline{A \cdot B}$
OR		$A + B$	NOR		$\overline{A + B}$
NOT		$\bar{A}$	Buffer		A

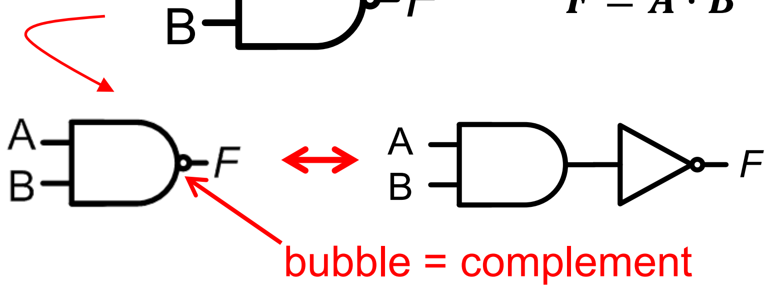
AND and NAND Gates



$$F = A \cdot B$$



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



Truth Table (AND, NAND):

A	B	$A \cdot B$	$\overline{A \cdot B}$
0	0	0	1
1	0	0	1
0	1	0	1
1	1	1	0

AND

- ***F*** is TRUE only when both ***A*** and ***B*** are TRUE

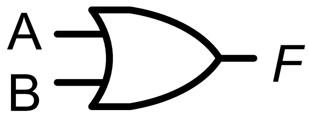
```
module andgate(A, B, F);
    input A, B;
    output F;
    assign F = A & B;
endmodule
```

NAND

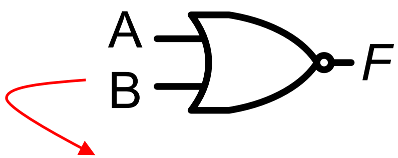
- ***F*** is FALSE only if both ***A*** and ***B*** are TRUE

```
module nandgate(A, B, F);
    input A, B;
    output F;
    assign F = ~(A & B);
endmodule
```

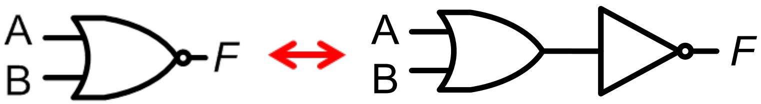
OR and NOR Gates



$$F = A + B$$



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



Truth Table (OR, NOR):

A	B	$A + B$	$\overline{A + B}$
0	0	0	1
1	0	1	0
0	1	1	0
1	1	1	0

OR

- ***F*** is FALSE only when both ***A*** and ***B*** are FALSE

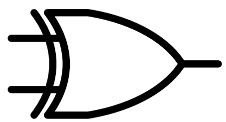
```
module orgate(A, B, F);
    input A, B;
    output F;
    assign F = A | B;
endmodule
```

NOR

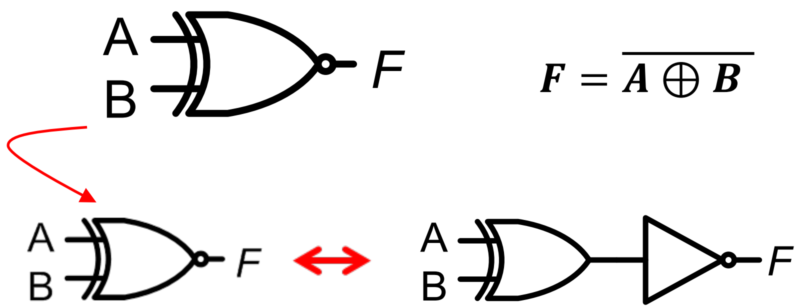
- ***F*** is TRUE only if both ***A*** and ***B*** are FALSE

```
module norgate(A, B, F);
    input A, B;
    output F;
    assign F = ~(A | B);
endmodule
```

XOR and XNOR Gates



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



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

Truth Table (XOR, XNOR):

A	B	$A \oplus B$	$\overline{A \oplus B}$
0	0	0	1
1	0	1	0
0	1	1	0
1	1	0	1

XOR

- ***F*** is TRUE if ***A* ≠ *B***

```
module xorgate(A, B, F);
    input A, B;
    output F;
    assign F = A ^ B;
endmodule
```

XNOR

- ***F*** is TRUE if ***A* = *B***

```
module xnorgate(A, B, F);
    input A, B;
    output F;
    assign F = ~(A ^ B);
endmodule
```

What function does the NOR gate implement?

A NOR B = not(A) OR not(B)

A NOR B = A OR B

A NOR B = NOT(A OR B)

Implementation of Boolean Function using Logic Gates

- Implement the following Boolean functions to logic gates, assume that the maximum number of inputs of a gate is 4.

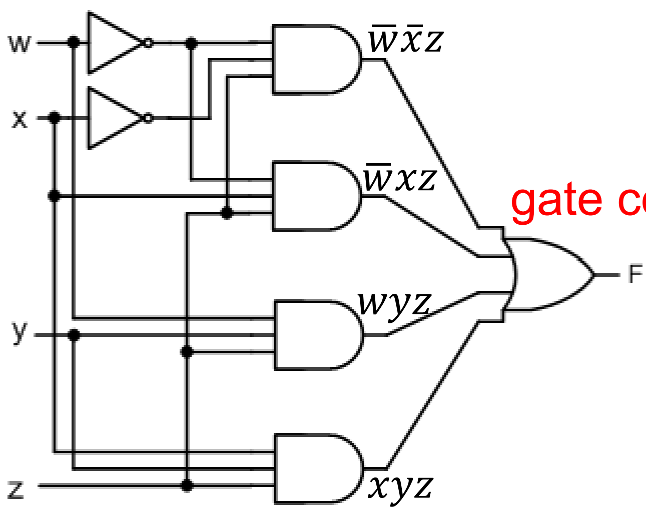
$$F(w,x,y,z) = \bar{w}\bar{x}z + \bar{w}xz + wyz + wxz$$

Implementation of Boolean Function using Logic Gates

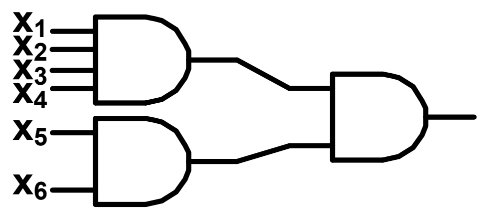
- Implement the following Boolean functions to logic gates, assume that the maximum number of inputs of a gate is 4.

$$F(w,x,y,z) = \bar{w}\bar{x}z + \bar{w}xz + wyz + xyz$$

if AND5 or more is needed: two-level ANDing (same for OR):
$x_1 \cdot x_2 \cdot x_3 \cdot x_4 \cdot x_5 \cdot x_6 = (x_1 \cdot x_2 \cdot x_3 \cdot x_4) \cdot (x_5 \cdot x_6)$



gate count = 7



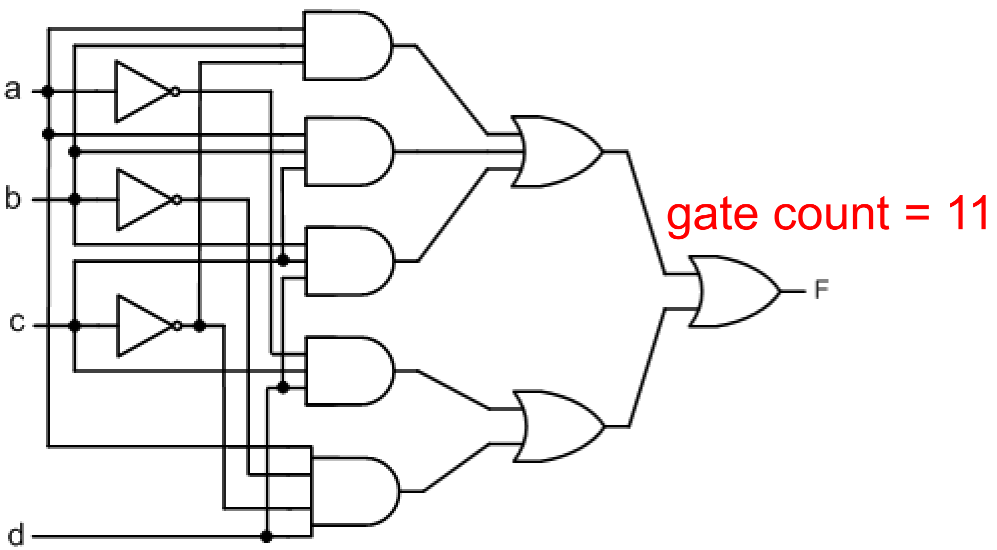
parentheses (~w & x & z) not needed in SOP, as precedence order is ~, &, ^, |

```
module func(w,x,y,z,F);
    input w, x, y, z;
    output F;
    assign F = ~w & ~x & z | ~w & x & z | w & y & z | x & y & z;
endmodule
```

Implementation of Boolean Function using Logic Gates

- Implement the following Boolean functions to logic gates, assume that the maximum number of inputs of a gate is 4.

$$F(a,b,c,d) = ab\bar{c} + abc + bcd + \bar{a}cd + a\bar{b}\bar{c}d$$




```
module func(a,b,c,d,F);
    input a, b, c, d;
    output F;
    assign F = a & b & ~c | a & b & c | b & c & d | ~a & c & d | a & ~b & ~c & d;
endmodule
```

Implementation of Boolean Function using Logic Gates

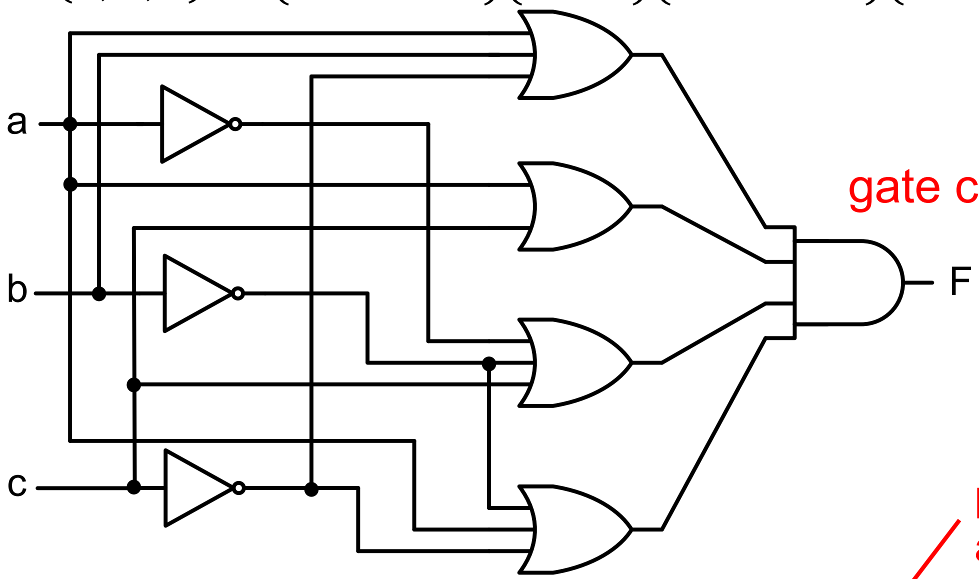
- Implement the following Boolean functions to logic gates, assume that the maximum number of inputs of a gate is 4.

$$F(a,b,c) = (a+b+\bar{c})(a+c)(\bar{a}+\bar{b}+c)(a+\bar{b}+\bar{c})$$

Implementation of Boolean Function using Logic Gates

- Implement the following Boolean functions to logic gates, assume that the maximum number of inputs of a gate is 4.

$$F(a,b,c) = (a+b+\bar{c})(a+c)(\bar{a}+\bar{b}+c)(a+\bar{b}+\bar{c})$$



gate count = 8

parentheses (~a | ~b | c) needed in POS, as precedence order is ~, &, ^, |

```
module func(a,b,c,F);
    input a, b, c;
    output F;
    assign F = (a | b | ~c) & (a | c) & (~a | ~b | c) & (a | ~b | ~c);
endmodule
```

Verilog Description of Boolean Expressions and Digital Systems

- Three main description styles
  - style affects implementation (not exactly equivalent)

behavioral → algorithmic (sequence)

```
always @ (posedge clk)
    begin
        a = ~b; // procedural assignment
    end
```

dataflow → assumes synchronous logic (describes combinational logic)

```
assign a = ~b; // continuous assignment
```

structural → gate-level structure

```
// modules instantiated and connected
and2 u0 (a, b, out);
nor2 u1 (out, c, d);
```

Verilog Description of Boolean Expressions and Digital Systems

- **Behavioral** includes registers (see part II)
- **Dataflow**
  - continuous assignment (in the body of the module)
  - example of logic gate

```
module nandgate(A, B, F);
    input A, B;
    output F;
    assign F = ~(A & B);
endmodule
```

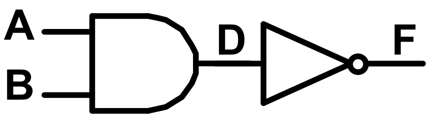
  - example of complex combinational function

```
module func(a,b,c,d,F);
    input a, b, c, d;
    output F;
    assign F = a & b & ~c | a & b & c | b & c & d | ~a & c & d | a & ~b & ~c & d;
endmodule
```

Verilog Description of Boolean Expressions and Digital Systems

- **Structural**
  - specifies exact gate-level structure
  - constrains synthesis tool (no automated optimization)

```
module nand2struct(F, A, B);
      output F;
      input A, B;
      wire D;
      and2 u1(D,A,B); // AND2 gate, instance u1
      inv u2(F,D); // inverter gate, instance u2
endmodule
```



  - Verilog primitives
    - INVERTER, BUFFER: not, buf
    - TRISTATE: bufif0 (active lo), bufif1 (active hi), notif0, notif1
    - COMB: and, nand, or, nor, xor, xnor

As designers, when should we adopt a structural Verilog style of description?

When poll is active, respond at **PollEv.com/massimoaliot866**

Text **MASSIMOALIOT866** to **+61 429 883 481** once to join

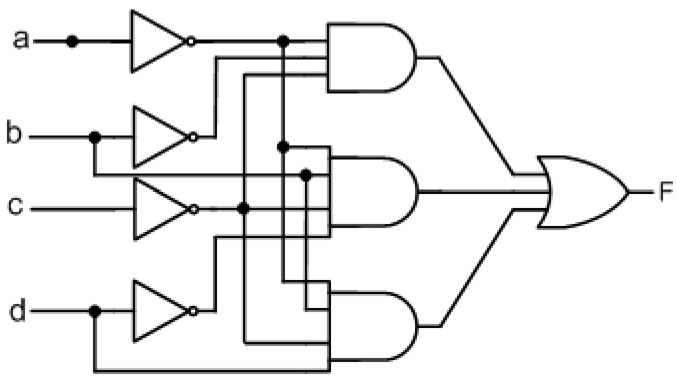
Boolean Function Simplification using Algebra Manipulation

- To reduce the hardware cost, the Boolean function can be simplified before implemented using logic gates
- A simplified Boolean Function contains a minimal number of literals and terms such that no other expression with fewer literals and terms will represent the original function
- Simplification can be done by
  - Algebra manipulation using postulates and theorem
  - Karnaugh Map

Boolean Function Simplification

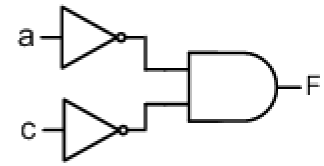
$$
\begin{aligned}
F(a,b,c,d) &= \bar{a}\bar{b}\bar{c} + \bar{a}b\bar{c}\bar{d} + \bar{a}b\bar{c}d \\
&= \bar{a}\bar{b}\bar{c} + \bar{a}b\bar{c}\left(\bar{d} + d\right) \quad \longleftarrow \quad (A + \bar{A} = 1) \\
&= \bar{a}\bar{c}\left(\bar{b} + b\right) \quad \longleftarrow \quad (A + \bar{A} = 1) \\
&= \bar{a}\bar{c} \quad \longleftarrow \quad (A + \bar{A} = 1)
\end{aligned}
$$

Before simplification:



gate count = 8

After simplification:



gate count = 3

(62.5% reduction!)

Boolean Function Simplification

(Relook at the first examples on Slide 7):

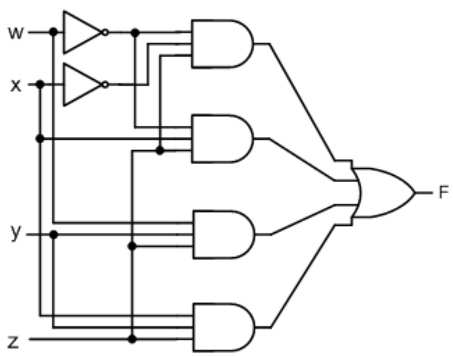
$F(x, y, z) = \bar{w}\bar{x}z + \bar{w}xz + xyz + wxy$

$= \bar{w}z(\bar{x} + x) + w(xy) + z(xy)$ $\longleftarrow$ $(A + \bar{A} = 1)$

$= \bar{w}z + w(xy) + z(xy)$ $\longleftarrow$ $(A + \bar{A} = 1)$

$= \bar{w}z + wxy$ $\longleftarrow$ $(AB + \bar{A}C + BC = AB + \bar{A}C)$ - consensus

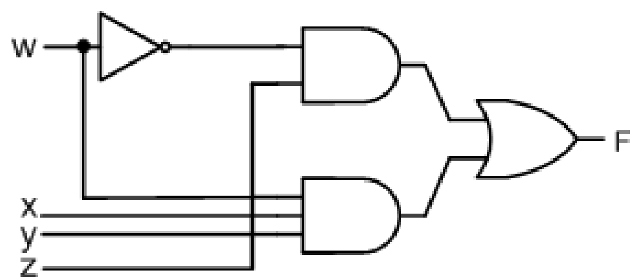
Before simplification:



gate count = 7

$\rightarrow$

After simplification:



gate count = 4

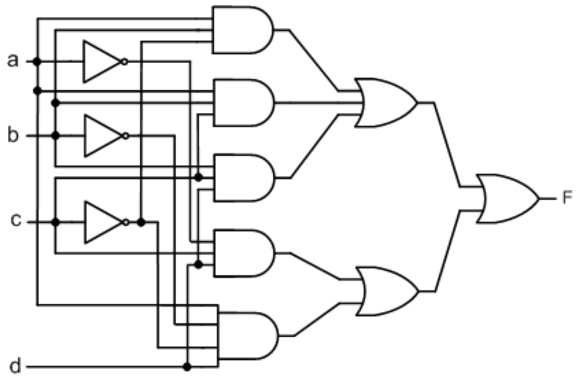
(43% reduction!)

Boolean Function Simplification

(Relook at the second examples on Slide 9):

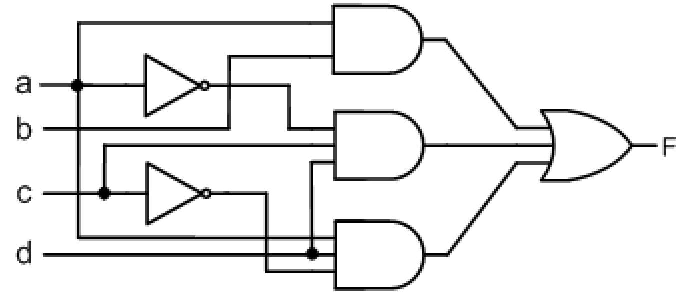
$$F(a,b,c,d) = ab\bar{c} + abc + bcd + \bar{a}cd + a\bar{b}\bar{c}d$$
$$= ab(\bar{c} + c) + bcd + \bar{a}cd + a\bar{b}\bar{c}d \qquad \longleftarrow \quad (A + \bar{A} = 1)$$
$$= a[b + \bar{b}(\bar{c}d)] + bcd + \bar{a}cd \qquad \longleftarrow \quad (A + \bar{A} \cdot B = A + B)$$
$$= a(b + \bar{c}d) + bcd + \bar{a}cd$$
$$= [ab + \bar{a}(cd) + b(cd)] + a\bar{c}d \qquad \longleftarrow \quad (AB + \bar{A}C + BC = AB + \bar{A}C) \text{ - consensus}$$
$$= ab + \bar{a}cd + a\bar{c}d$$

Before simplification:



gate count = 11

After simplification:



gate count = 6

(45.5% reduction!)

Some Guidelines for Simplification of Boolean Function (in SOP)

- Three most used theorems:

  (1) $AB + A\bar{B} = A$ (Logical adjacency)

  (2) $A + \bar{A} \cdot B = A + B$

  (3) $AB + \bar{A}C + BC = AB + \bar{A}C$ (Consensus)

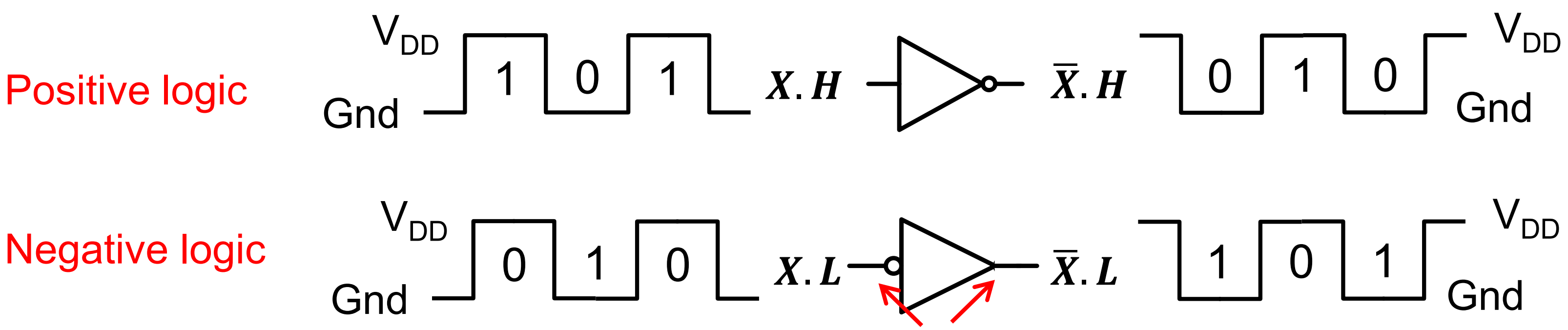
- Apply (1) until it cannot be applied further
- Apply (2) until it cannot be applied further
- Go back to (1) and then (2) until they can no longer be applied
- Apply (3) until it cannot be applied further
- Go back to (1), (2) and then (3) until none of them can be applied
- It can then be assumed that the function is simplified
- Empirical: the result is usually close to minimal, but may not be the minimal
- Cumbersome: other methods are much easier and quicker

What are the drawbacks of function simplification via Boolean algebra manipulations?

Positive and Negative Logic

- Positive and negative logic map the physical voltage (H, L) in a gate correspondence to a logic value
- Positive logic (Active high)
  - Voltage "**H**" (i.e. V_{DD}) → interpreted as logic "**1**" or "**True**"
  - Voltage "**L**" (i.e. Gnd or 0V) → interpreted as logic "**0**" or "**False**"
- Negative Logic (Active low)
  - Voltage "**L**" (i.e. Gnd or 0V) → interpreted as logic "**1**" or "**True**"
  - Voltage "**H**" (i.e. V_{DD}) → interpreted as logic "**0**" or "**False**"

Example:

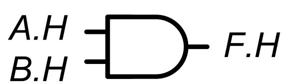


graphically: put a bubble at each active-low I/O signal
(for any voltage level, X.L is the complement of X.H)

Positive and Negative Logic – cont.

A physical gate implements two different functions when using positive or negative logic (to have negative logic, both inputs and output need to be complemented)

$X.H$ → Logic value X represented in positive logic
$X.L$ → Logic value X represented in negative logic



A.H	B.H	F.H
0	0	0
0	1	0
1	0	0
1	1	1

AND

Pos. logic ←

physical gate

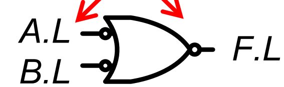
Physical truth table

A	B	F
L	L	L
L	H	L
H	L	L
H	H	H

Physical voltage levels

→ Neg. logic

bubbles to specify active-low I/Os



A.L	B.L	F.L
1	1	1
1	0	1
0	1	1
0	0	0

OR

If I use a physical positive AND gate, I actually get an OR gate when applying/reading active-low signals

Conversion of Inverter between Positive and Negative Logic

Systematic approach to convert gates between positive/negative logic:

Voltage level	Positive logic value	Negative logic value
H	1	0
L	0	1

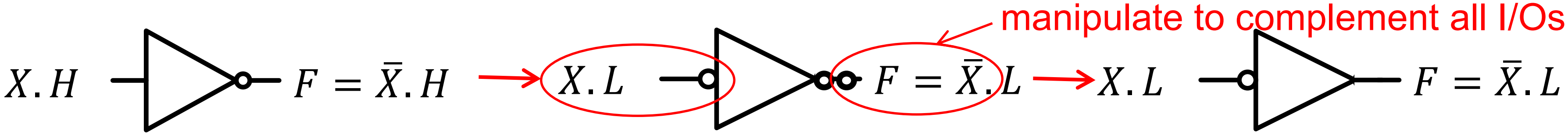
Conversion of a signal:

$$\boldsymbol{X.H = \overline{X}.L}$$
$$\boldsymbol{X.L = \overline{X}.H}$$

(add bubble at each active-low signals)
$\Rightarrow$ add bubble to all inputs/outputs to represent a gate in negative logic

Graphical approach to find equivalent gate in negative logic from positive:
which physical gate should we use to achieve same function but in negative logic?
(apply active-low inputs, express output as active-low signal)

manipulate to complement all I/Os



Equivalently, same can be done through Boolean algebra:

$$F.H = \overline{X.H} \rightarrow \overline{F.H} = X.H = \overline{\overline{X}}.H \rightarrow F.L = \overline{X}.L$$

Similarly, it could be done through truth tables.

Note that there is only ONE physical inverter

Conversion between Positive and Negative Logic of Other Gates

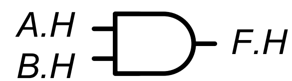
AND gate:

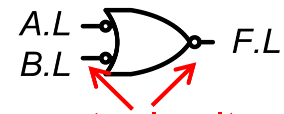
if I have positive physical gates, what function do I implement with them in negative logic? (hint: complement all I/Os, find new function)

$F.H = A.H \cdot B.H$ ← AND (in positive logic)

$\overline{F.H} = \overline{A.H \cdot B.H} = \overline{A.H} + \overline{B.H}$

$F.L = A.L + B.L$ ← OR (in negative logic)





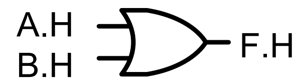
OR gate:

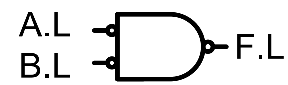
bubbles to specify active-low I/Os (i.e., the actual voltage is the complement)

$F.H = A.H + B.H$ ← OR (in positive logic)

$\overline{F.H} = \overline{A.H + B.H} = \overline{A.H} \cdot \overline{B.H}$

$F.L = A.L \cdot B.L$ ← AND (in negative logic)

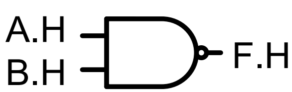


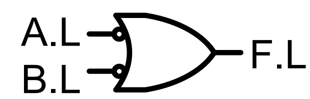


NAND gate:

NOR in negative logic

$F.H = \overline{A.H \cdot B.H} \rightarrow \overline{F.H} = A.H \cdot B.H = \overline{\overline{A.H} + \overline{B.H}} \rightarrow F.L = \overline{A.L + B.L}$

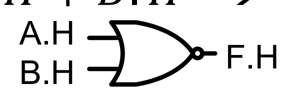


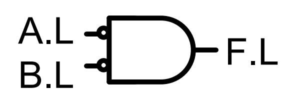


NOR gate:

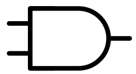
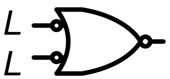
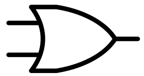
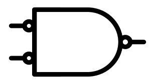
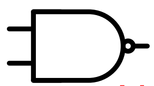
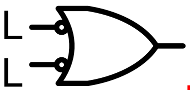
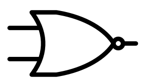
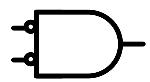
NAND in negative logic

$F.H = \overline{A.H + B.H} \rightarrow \overline{F.H} = A.H + B.H = \overline{\overline{A.H} \cdot \overline{B.H}} \rightarrow F.L = \overline{A.L \cdot B.L}$





Summary of Positive/Negative Logic and Mixed Logic

Physical Gate	Positive logic	Negative logic	Remark
AND	*A.H* *B.H*  *F.H* AND	*A.L* *B.L*  *F.L* OR	Both are physical AND gate
OR	A.H B.H  F.H OR	A.L B.L  F.L AND	Both are physical OR gate
NAND	A.H B.H  F.H NAND	A.L B.L  F.L NOR	Both are physical NAND gate
NOR	A.H B.H  F.H NOR	A.L B.L  F.L NAND	Both are physical NOR gate

- Above table allows conversion btwn positive/negative logic
- Mixed logic combines active-high/active-low signals
- Systematic procedure to find physical gates in positive / negative / mixed logic, based on active-high/low signal assignment?

Bubble Pushing Rule to Rearrange Logic and Transform (N)AND/(N)OR

Practical rule to account for active-low signals and mix with active-high: think in terms of positive logic, and complement active-low inputs/outputs.

How to rearrange logic through graphic manipulations in the presence of bubbles:

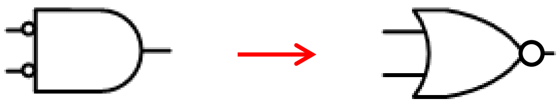
- two adjacent bubbles gets simplified

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

- bubbles at the input of an AND gate can be "pushed" at its output, and the gate is transformed into a NOR gate (similarly, NAND becomes OR)

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

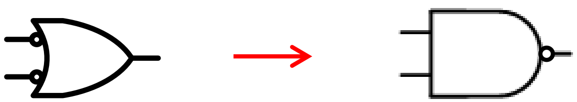
De Morgan's law



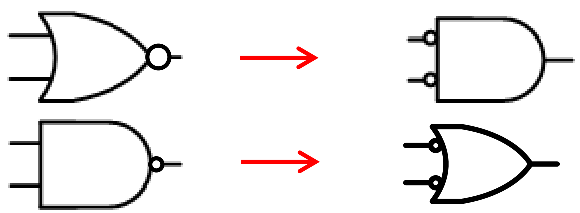
- bubbles at the input of an OR gate can be "pushed" at its output, and the gate is transformed into a NAND gate (similarly, NOR becomes AND)

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

De Morgan's law



- and vice versa, of course:

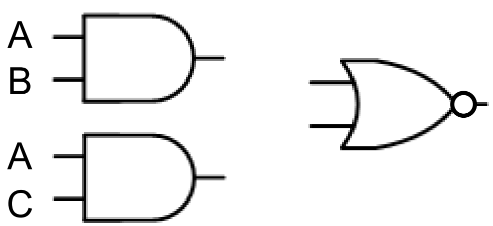


Example – Implementation in Positive Logic

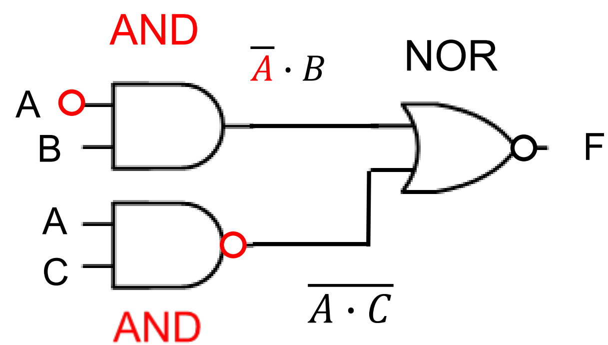
Implement the following Boolean function in positive logic using only **NOR** gates and inverters

$$F = \overline{\left(\overline{A} \cdot B + \overline{A \cdot C}\right)}$$

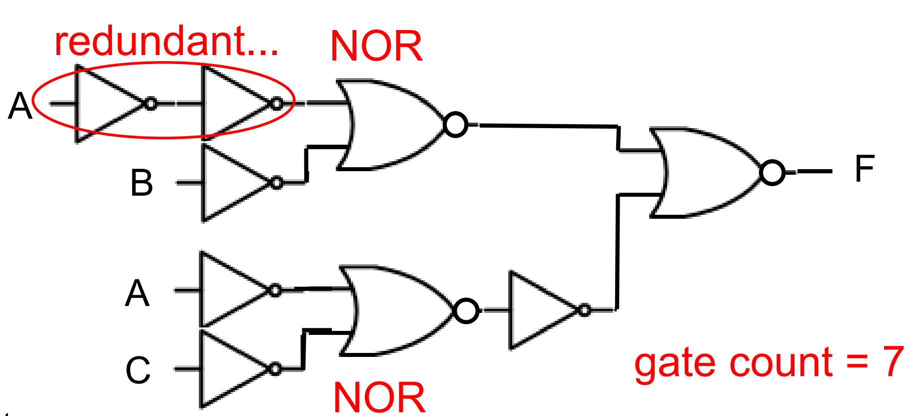
Step 1:



Step 2:
(add the negation where needed for the correct function)



Step 3:
(i) Replace AND gate with NOR gate
(ii) balance the bubbles using inverters to maintain the correct functionality

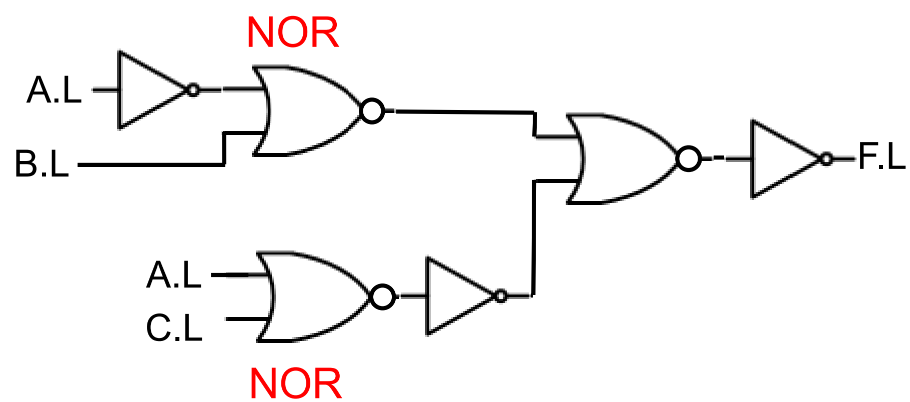


Example – Implementation in Negative and Mixed Logic

Implement the same Boolean function in negative logic (i.e., all input and output signals of the function are active low), using only NOR gates and inverters

$$F = \overline{\left(\overline{A} \cdot B + \overline{A \cdot C}\right)}$$

Just perform same steps as previous slide and insert inverters (bubbles) at inputs and outputs:



In case of mixed logic at inputs or outputs (positive & negative), just add inverters (bubbles) as needed and rearrange according to the same rules

What Boolean algebra property translates into bubble pushing?

Commercial logic gate ICs

- 74xxx Series
  - **TTL** family (**T**ransistor-**T**ransistor **L**ogic)
  - Use Bipolar or CMOS technology
- Name convention
  - 1st field: 2 or 3 letters → Manufacturer (sometimes omitted)
  - 2nd field: 74 → Commercial temperature range (54 → Military)
  - 3rd field: 4 letters → Logic sub-family
  - 4th field: 2 or more digits → Type of device
  - 5th field: Type of package or other information (sometimes omitted)

DM 74 LS 14 N

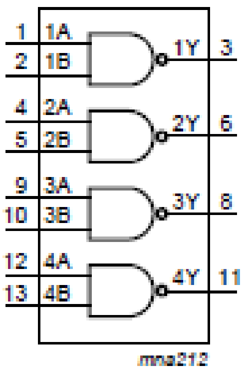
- DM → National Semiconductor (SN = Texas instruments)
- 74 → Commercial temperature range
- LS → Low power Schottky
- 14 → Hex inverters with Schmitt trigger inputs
- N → Plastic package

74 series Logic Sub-families (3rd field)

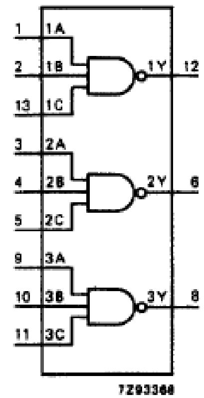
- TTL (Bipolar)
  - 74L → Low power
  - 74H → High speed
  - 74LS → Low power Schotty
  - 74AS → Advanced low power schotty
  - 74ALS → Advanced low power schotty
  - ....
- CMOS (not TTL, but retains some compatibility)
  (same part numbers as bipolar are retained to identify the function)
  - 74C → CMOS 4-15V
  - 74HC → High speed
  - 74AC → Advanced CMOS
  - 74LVC → Low voltage, 1.65 to 3.3V
  - 74LVX → 3.3V with 5V tolerant inputs
  - ……

Some 74 series Logic gates

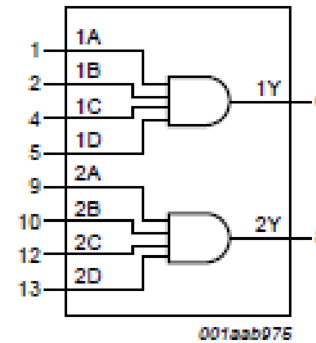
7400 (74HC00)
(Quad 2-input NAND gate)



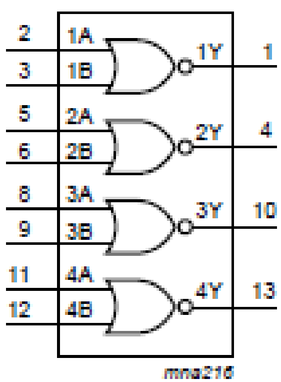
7410(74HC10)
(Dual 3-input AND gate)



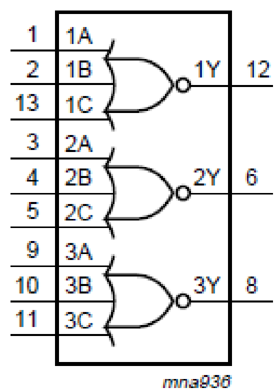
7421(74HC21)
(Dual 4-input AND gate)



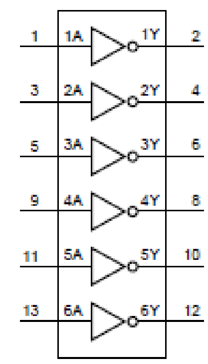
7402(74HC02)
(Quad 2-input NOR gate)



7427(74HC27)
(Quad 3-input NOR gate)



7404(74HC04)
(Hex inverters)



Summary

- Logic gate is a circuit that implement Boolean operations
- AND and NAND gates
- OR and NOR gates
- XOR and XNOR gates
- Boolean function implementation using logic gates
- Boolean function simplification using algebra postulates and theorems
- Positive and negative logics
  - Definition
  - Physical gates with positive and negative logics
  - Physical truth table and logic truth table
  - Conversion between positive and negative logics
  - Gates with mixed logic

Suggestions for Self-Improvement

- In addition to the lecture/tutorials/lab sessions on Verilog, you may want to read chapter 4 of the textbook (see IVLE Workbin)
  - simple introduction to Verilog
  - description of logic gates
  - description of logic functions