EE2026 (Part 1)
Tutorial 3 - Questions

Boolean Algebra Manipulation and Logic gates

- Boolean algebra is ultimately described by
  - a set of elements B={0,1} (binary symbols)
  - operator $\overline{...}$ (NOT) on single operand
  - operators $\cdot$ (AND), + (OR) on multiple operands

binary constants or variables
(as in common algebra)

$\cdot$ has precedence over +

priority: NOT has highest precedence, followed by AND and OR
→ NOT(A · B + C) = NOT((A · B) + C)

1. Closure
   - $\forall x, y \in B, x + y \in B$ (outcome of operation is still in B, obviously)
   - $\forall x, y \in B, x \cdot y \in B$ (outcome of operation is still in B, obviously)

2. Neutral elements of $+$ and $\cdot$
   - There exists a 0 and 1 element in B, such that
     - $x + 0 = x$ ← 0 is neutral elements for +
     - $x \cdot 1 = x$ ← 1 is neutral elements for $\cdot$

3. Commutative Law
   - $x + y = y + x$
   - $x \cdot y = y \cdot x$

4. Distributive Law
   - $x \cdot (y + z) = x \cdot y + x \cdot z$ ($\cdot$ over $+$, easy to remember)
   - $x + (y \cdot z) = (x + y) \cdot (x + z)$ ($+$ over $\cdot$ not intuitive, still true)

} duality

5. Complement
   - $\forall x \in B$, there exists an element $\bar{x} \in B$ (complement of x) such that
     - $x + \bar{x} = 1$
     - $x \cdot \bar{x} = 0$

6. There exist at least two distinct elements in the set B (obvious, it was introduced for more general algebras with more than two symbols)

#	Theorem		
1	$A + A = A$	$A \cdot A = A$	Tautology Law
2	$A + 1 = 1$	$A \cdot 0 = 0$	Union Law
3	$\overline{(\bar{A})} = A$		Involution Law
4	$A + (B + C)$ $= (A + B) + C$	$A \cdot (B \cdot C) = (A \cdot B) \cdot C$	Associative Law
5	$\overline{A + B} = \bar{A} \cdot \bar{B}$	$\overline{A \cdot B} = \bar{A} + \bar{B}$	De Morgan's Law
6	$A + A \cdot B = A$	$A \cdot (A + B) = A$	Absorption Law
7	$A + \bar{A} \cdot B = A + B$	$A \cdot (\bar{A} + B) = A \cdot B$	
8	$AB + A\bar{B} = A$	$(A + B)(A + \bar{B}) = A$	Logical adjacency
9	$AB + \bar{A}C + BC$ $= AB + \bar{A}C$	$(A + B)(\bar{A} + C)(B + C)$ $= (A + B)(\bar{A} + C)$	Consensus Law

duality (swap OR and AND, swap 0 and 1)

- A Boolean expression can be rewritten in many formally different (but equivalent) forms
  - Sum of products (SOP)
    - Logic sum of product terms
    - Example:

$$F_1(A,B,C) = \overbrace{\bar{A}\bar{B}C + \bar{A}B\bar{C} + \mathrm{A}\bar{B}C}^{\text{sum}}$$

products

  - Product of sums (POS)
    - Logic product of sum terms
    - Example:

$$F_2(A,B,C) = \overbrace{(A + B + C) \cdot (A + \bar{B} + \bar{C}) \cdot (\bar{A} + B + C) \cdot (\bar{A} + \bar{B} + C) \cdot (\bar{A} + \bar{B} + \bar{C})}^{\text{product}}$$

sums

Boolean Algebra Manipulation and Logic gates

1. Use algebraic manipulation to find the most simplified expression (MSOP, minimum SOP) for

$$F = x_1x_3 + x_1\bar{x}_2 + \bar{x}_1x_2x_3 + \bar{x}_1\bar{x}_2\bar{x}_3$$

Write the Verilog code that describes the above function as a module (use $\underline{\text{dataflow}}$ description style).

F=

$$\underline{x_1x_3} + x_1\bar{x}_2 + \underline{\bar{x}_1x_2x_3} + \bar{x}_1\bar{x}_2\bar{x}_3$$

$$= x_3(x_1 + \bar{x}_1x_2) + \bar{x}_2(x_1 + \bar{x}_1\bar{x}_3)$$ distributive law

$$= x_3(x_1 + x_2) + \bar{x}_2(x_1 + \bar{x}_3) \quad \{\text{using } A + \bar{A}B = A + B\}$$ $\bar{A}$ is redundant here (Theorem 7)

$$= \boxed{x_1\underline{x_3}} + x_2x_3 + \cancel{\boxed{x_1\bar{x}_2}} + \boxed{\bar{x}_2\underline{\bar{x}_3}}$$ consensus law.

$$= x_1x_3 + x_2x_3 + \bar{x}_2\bar{x}_3 \quad \{\text{using } AB + \bar{A}C + BC = AB + \bar{A}C;\ A \rightarrow x_3,\ B \rightarrow x_1,\ C \rightarrow \bar{x}_2\}$$

$$\text{or} \quad x_1\bar{x}_2 + x_2x_3 + \bar{x}_2\bar{x}_3 \quad \{\text{using } AB + \bar{A}C + BC = AB + \bar{A}C;\ A \rightarrow \bar{x}_2,\ B \rightarrow x_1,\ C \rightarrow x_3\}$$

$$\cancel{\boxed{x_1x_3}} + \boxed{x_2x_3} + \boxed{x_1\bar{x}_2} + \bar{x}_2\bar{x}_3$$

```
module func(x1,x2,x3,F);
        input x1,x2,x3;
        output F;
        assign F = x1 & x3 | x1 & ~x2 | ~x1 & x2 & x3 | ~x1 & ~x2 & ~x3;
        // no parentheses are needed
endmodule
```

wire

continuous assignment

automatically evaluate order precedence.

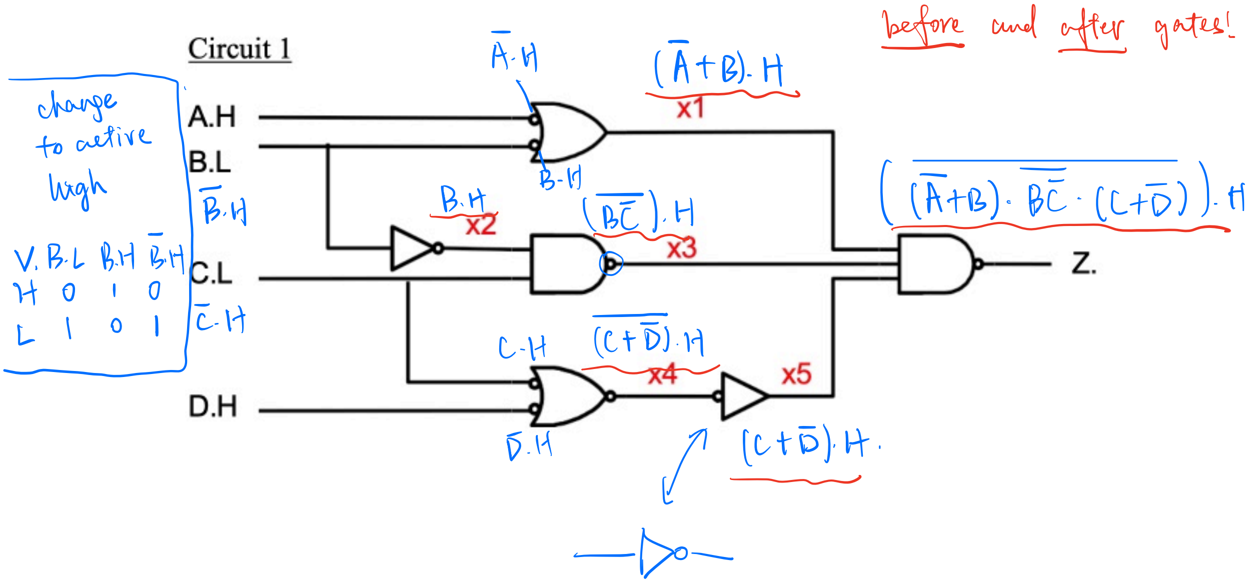
· > + .

☆. ① Standardize logic system. ② evaluate one by one

2. Express all inputs and outputs as active high (i.e., complement active-low signals). The resulting circuit is in positive logic, and the expression of all intermediate nodes x1...x5 is found immediately by proceeding from inputs to output:

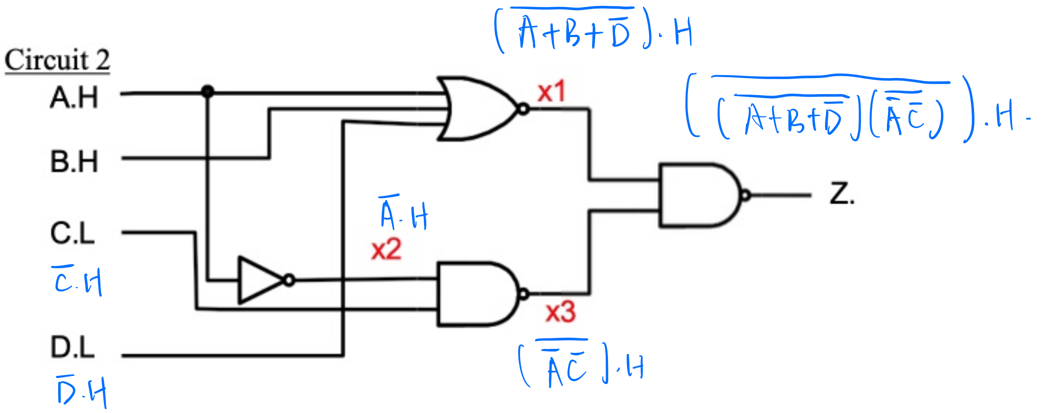
Evaluate each bubble before and after gates!

Circuit 1



V.	B.L	B.H	$\bar{B}$.H
H	0	1	0
L	1	0	1

Circuit 2


A.H
B.H
C.L
$\bar{C}$.H
D.L
$\bar{D}$.H
$(\overline{A+B+\bar{D}})$.H
x1
$\bar{A}$.H
x2
x3
$(\overline{\bar{A}\bar{C}})$.H
Z.
$(\overline{(\overline{A+B+\bar{D}})(\overline{\bar{A}\bar{C}})})$.H.

describe gates in verilog

(c) Using structural modeling style, write the Verilog code that describes Circuit 2. Assume that Verilog primitives are available: not(out,in), nand(out,in1,in2,...), nor(out,in1,in2,...).

wire

```
module func(Z,A,B,C,D); // Cb and Db are complemented (active-low)
        input A, B, C, D;
        output Z;
        wire x1, x2, x3, Cb, Db;                         output      input
gate  →     not u1(Cb,C); // inverter gate to generate Cb=NOT(C) (see omitted inverter)
type        not u2(Db,D); // inverter gate to generate Db=NOT(D) (see omitted inverter)
            nor u3(x1,A,B,Db); // NOR3 gate, instance u3
            not u4(x2,A); // inverter gate
            nand u5(x3,x2,Cb); // NAND2 gate
            nand u6(Z,x1,x3); // NAND2 gate
endmodule
```

Voltage logic values.

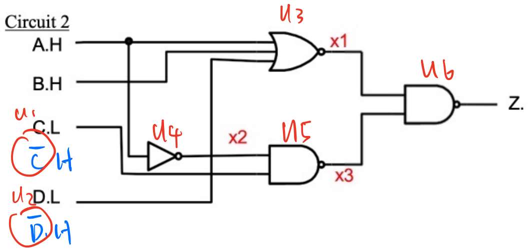
	V.	X.H	X.L
A	L	0	1
B	L	0	1
C	L	0	1
D	L	0	1

omitted inverter gates

Cb : $\bar{C}$.H / C.L

Db : $\bar{D}$.H / D.L

Circuit 2



3. Design a logic circuit implementing the following Boolean function, using a minimum number of NOR gates. NOT gates can also be used if required.

$$X = A \oplus B \oplus C,$$

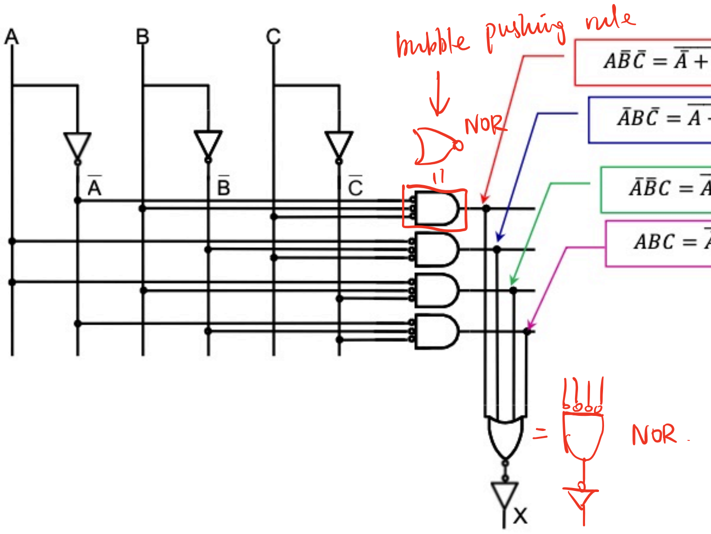
where ⊕ represents the 2-operand XOR operation.
Use alternate gate representation if necessary for clear circuit diagrams.

$$\overbrace{(A\bar{B}+\bar{A}B)}^{D}\oplus C \rightarrow D\bar{C}+\bar{D}C$$

$$X = A \oplus B \oplus C = (A\bar{B}+\bar{A}B)\cdot\bar{C}+\overline{(A\bar{B}+\bar{A}B)}\cdot C = A\bar{B}\bar{C}+\bar{A}B\bar{C}+\bar{A}\bar{B}C+ABC$$

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

bubble pushing rule



$A\bar{B}\bar{C} = \overline{\bar{A}+B+C}$ (De Morgan's)

$\bar{A}B\bar{C} = \overline{A+\bar{B}+C}$ (De Morgan's)

$\bar{A}\bar{B}C = \overline{A+B+\bar{C}}$ (De Morgan's)

$ABC = \overline{\bar{A}+\bar{B}+\bar{C}}$ (De Morgan's)

$Z \cdot H = (\bar{A}B + \bar{B}\bar{C}D + \bar{B}\bar{D}) \cdot H$

4. Design a circuit to realize $Z = \bar{A}B + \bar{B}\bar{C}D + \bar{B}\bar{D}$ in the positive logic convention.

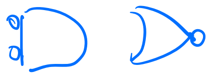
A.L, B.L   C·H, D·H, Z·H

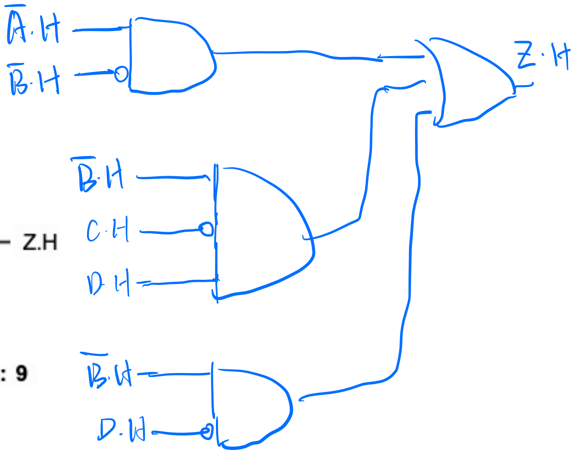
A and B are active low signals while C, D and Z are active high.

Use a minimum of number of **only** NAND or **only** NOR gates. You may assume that the gates have no limits on the number of inputs. NOT gates can be used if required.

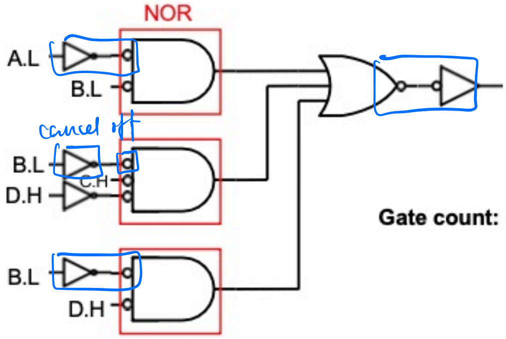
Draw clear circuit diagrams with alternate gate representations where necessary.

☆ $A \cdot L = \bar{A} \cdot H$.





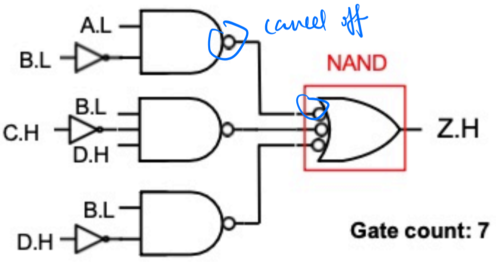
Using NOR gates only:



Gate count: 9

All NOR gates + Not gates (inverters)

Using NAND gates only:



All NAND gates + Not gates (inverters)

5. [Verilog Behavioural Style Modeling]

In the EE2026 design project, an OLED screen with 16-bit RGB colour resolution is used as a display device. The 16-bit signal colour resolution is represented through 5 bits for the red colour component, 6 bits for the green colour component, and 5 bits for the blue colour component. Write a Verilog program to switch between 4 display colours according to the table below.

The program receives a 2-bit input named SEL and produces a 16-bit output signal named PIXEL_COLOUR.

SEL	Colour To Display	PIXEL_COLOUR
“00”	Red	“11111 000000 00000”
“01”	Blue	“00000 000000 11111”
“10”	Green	“00000 111111 00000”
“11”	Magenta	“11111 000000 11111”

high level

behavioral

data flow

structural

low level

procedural assignment

// Solution 1a – Behavioral Style of Modeling using if-else statements

```
module oled (input [1:0] SEL, output reg [15:0] PIXEL_COLOUR);

  always @ (SEL)
  begin
        if   ( SEL[1] == 0 && SEL[0] == 0 )
            PIXEL_COLOUR = {5’b11111, 6’b000000, 5’b00000};
        else if ( SEL[1] == 0 && SEL[0] == 1)
            PIXEL_COLOUR = {5’b00000, 6’b000000, 5’b11111};
        else if ( SEL[1] == 1 && SEL[0] == 0)
            PIXEL_COLOUR = {5’b00000, 6’b111111, 5’b00000};
        else
            PIXEL_COLOUR = { 5’b11111, 6’b000000, 5’b11111};
  end

endmodule
```

procedural assignment

equivalence of procedural and continuous assign (L3. pg 17)

// Solution 1b – Behavioral Style of Modeling using case statements

```
  always @ (*)
  begin
        case ({SEL[1],SEL[0]})
           2’b00 : PIXEL_COLOUR = 16’hF800;
           2’b01 : PIXEL_COLOUR = 16’h001F;
           2’b10 : PIXEL_COLOUR = 16’h07E0;
           2’b11 : PIXEL_COLOUR = 16’hF81F;
           default : PIXEL_COLOUR = 16’h0000;
           //the default statement is used to capture all other cases!
        endcase
  end
```

continuous assignment

conditional operator (similar to C..)

// Solution 2 – Dataflow Style of Modeling using continuous assignments

wire

check left bit

```
assign PIXEL_COLOUR = (SEL[1]) ? ( SEL[0] ?  16’hF81F :  16’h07E0):
                                 ( SEL[0] ?  16’h001F :  16’hF800);
```

check left bit { 1: check right bit { 1: 16'hF81F
0: 16'h07E0
0: check right bit { 1: 16'h001F
0: 16'hF800