Which Truth Tables Are the Same?

1)

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

2)

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

3)

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

The Idea of Min Term / Product Term

X	Y	F	A	B
0	0	1	1	0
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

X	Y	Min Term
0	0	X' Y'
0	1	X' Y
1	0	X Y'
1	1	X Y

- **Each row in a truth table represents a unique combination of variables**
- **Each row can be expressed as a logic combination specifying when that row combination is equal to a 1**
- **The term is called a MIN TERM or a PRODUCT TERM**
- **Thus F = A + B = X'Y' + XY**

Truth Table with Two Inputs

- **Two inputs X and Y; Output is F**
- **Logic Function:**
  - **F = 1 if and only if X = Y**
- **Truth Table:**

X	Y	F	Min Term
0	0	1	X'Y'
0	1	0	X'Y
1	0	0	XY'
1	1	1	XY

- **Logic Expression:**
  - **F=X'Y'.1 + X'Y.0 + XY'.0 + XY.1**
  - **=X'Y' + XY**

Min / Product terms for more variables

XYZ	Min Term
000	X' Y' Z'
001	X' Y' Z
010	X' Y Z'
011	X' Y Z
100	X Y' Z'
101	X Y' Z
110	X Y Z'
111	X Y Z

XYZW	Min Term
0000	X' Y' Z' W'
0001	X' Y' Z' W
0010	X' Y' Z W'
0011	X' Y' Z W
0100	X' Y Z' W'
0101	X' Y Z' W
0110	X' Y Z W'
0111	X' Y Z W
1000	X Y' Z' W'
1001	X Y' Z' W
1010	X Y' Z W'
1011	X Y' Z W
1100	X Y Z' W'
1101	X Y Z' W
1110	X Y Z W'
1111	X Y Z W

Truth Table with Three Inputs

- **Three inputs X, Y, and Z; Output is F**
- **Logic Function:**
  - **F = 1 if and only if there is a 0 to the left of a 1 in the input**
- **Truth Table:**

X	Y	Z	F	Min term
0	0	0	0	
0	0	1	1	
0	1	0	1	
0	1	1	1	
1	0	0	0	
1	0	1	1	
1	1	0	0	
1	1	1	0	

- **Logic Expression:**
  - **F=**

Truth Table with Four Inputs

- **Four inputs X, Y, Z, and W; Output is F**
- **Logic Function:**
  - **F = 1 if and only if number of variables with value 1 is more than the number of variables with value 0**
- **Truth Table:**

XYZW	F
0000	0
0001	0
0010	0
0011	0
0100	0
0101	0
0110	0
0111	1
1000	0
1001	0
1010	0
1011	1
1100	0
1101	1
1110	1
1111	1

- **Logic Expression:**
  - **F=**

Canonical Sum-of-Product Expression

- A product (min) term is a unique combination of variables:
  - It has a value of 1 for only one input combination
  - It is 0 for all the other combinations of variables
- To write an expression, we need not write the entire truth table
- We only need those combinations for which function output is 1
- For example, for the function below: f = x'yz'+xy'z'+xyz

x	y	z	f	Min term
0	0	0	0	
0	0	1	0	
0	1	0	1	x'yz'
0	1	1	0	
1	0	0	1	xy'z'
1	0	1	0	
1	1	0	0	
1	1	1	1	xyz

- This is called the Canonical Sum-of-Product (SOP) Expression

Shorthand Notation for Canonical SOP

- We can also assign an integer to represent each input combination
- Thus the function produces a 1 for input combinations 2, 4, 7
- Therefore, the function can be written as f(x,y,z)= $\sum m(2,4,7)$

x	y	z	f	Index for shorthand notation
0	0	0	0	0
0	0	1	0	1
0	1	0	1	2
0	1	1	0	3
1	0	0	1	4
1	0	1	0	5
1	1	0	0	6
1	1	1	1	7

Max / Sum Terms

- A max (sum) term is also a unique combination of variables
  - However, it is opposite of a min term
  - It has a value of 0 for only one input combination
  - It is 1 for all the other combinations of variables
  - That is why it is called a max (sum) term
  - Each row in truth table has a max term corresponding to it
- Example, a max term (x+y+z) is 0 for combination xyz=000 only

X	Y	Max Term
0	0	X + Y
0	1	X + Y'
1	0	X' + Y
1	1	X' + Y'

Canonical Product-of-Sum Expression

- A function can also be written in terms of max terms
- The function is product of all max terms for which function is 0
- For example, the same function of three variable x, y, and z produces 0 for xyz=000, 011, 101, then
  - F = (x+y+z).(x+y'+z').(x'+y+z')
- This is called the Canonical Product-of-Sum (POS) Expression
- The function can also be written as F(x,y,z)= $\prod M\,(0,3,5)$

X	Y	Z	F	F'
0	0	0	0	1
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	1	0
1	1	1	1	0

F = (F')'
= (x'y'z' + x'yz + xy'z)'
= (x+y+z).(x+y'+z').(x'+y+z')

Truth Tables and Logic Expression for Adder

A	B	C	X	Y
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$X(A,B,C) = \sum m(1,2,4,7)$$
$$Y(A,B,C) = \sum m(3,5,6,7)$$

$$X(A,B,C) = \prod M(\qquad\qquad)$$
$$Y(A,B,C) = \prod M(\qquad\qquad)$$

X = A'B'C + A'BC' + AB'C' + ABC

Y = A'BC + AB'C + ABC' + ABC

Multiple Forms and Equivalence

- Canonical Sum-of-Product form
- Canonical Product-of-sum form
- How to convert one from other?
- Minterm expansion of X to minterm expansion of X'
  - Just take the terms that are missing

$$X(A,B,C) = \sum m(1,2,4,7) \qquad X'(A,B,C) = \sum m(\qquad\qquad)$$

- Maxterm expansion of X to maxterm expansion of X'
  - Just take the terms that are missing

$$X(A,B,C) = \prod M(\qquad\qquad) \qquad X'(A,B,C) = \prod M(\qquad\qquad)$$

Boolean Algebra

George Boole
1815-1864

- **An algebraic structure consists of**
  - **a set of elements {0, 1}**
  - **binary operators {+, .}**
  - **and a unary operator { ' }**
- **Introduced by George Boole in 1854**
- **An effective means of describing circuits built with switches**
- **A powerful tool that can be used for designing and analyzing logic circuits**

Axioms of Boolean Algebra

- **1a: 0.0 = 0**
  1b: 1+1 = 1
- **2a: 1.1 = 1**
  2b: 0+0 = 0
- **3a: 0.1 = 1.0 = 0**
  3b: 1+0 = 0+1 = 1
- **4a: If x=0, then x' = 1**
  4b: If x=1, then x' = 0

Single-Variable Theorems

- **5a: x.0 = 0** — **Null**
  5b: x+1 = 1
- **6a: x.1 = x** — **Identity**
  6b: x+0 = x
- **7a: x.x = x** — **Idempotency**
  7b: x+x = x
- **8a: x.x' = 0** — **Complementarity**
  8b: x+x' = 1
- **9: (x')' = x** — **Involution**

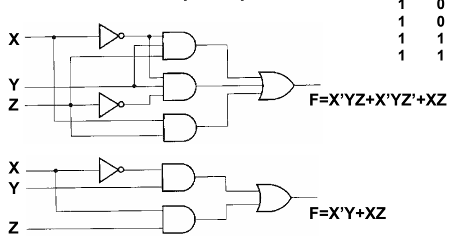
Two- and Three-Variable Properties

- **10a: x.y = y.x** — **Commutative**
  10b: x+y = y+x
- **11a: x.(y.z) = (x.y).z** — **Associative**
  11b: x+(y+z) = (x+y)+z
- **12a: x.(y+z) = x.y + x.z** — **Distributive**
  12b: x+y.z = (x+y).(x+z)
- **13a: x+x.y = x** — **Absorption**
  13b: x.(x+y) = x
- **14a: x.y+x.y' = x** — **Combining**
  14b: (x+y).(x+y') = x
- **15a: (x.y)' = x'+y'** — **DeMorgan's Theorem**
  15b: (x+y)' = x'.y'
- **16a: x+x'.y = x+y** — **Another form of Absorption**
  16b: x.(x'+y) = x.y

Simplify Logic Function by Algebraic Manipulation

- **F =X'YZ+X'YZ'+XZ**
  =X'Y(Z+Z')+XZ — **by Distributive**
  =X'Y(1)+XZ — **by Complementarity**
  =X'Y+XZ — **by Identity**

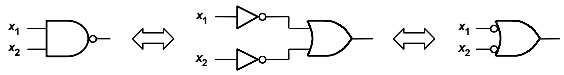
X	Y	Z	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1



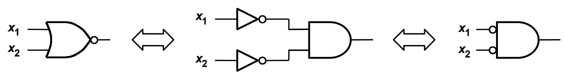
Principle of Duality

- **Dual:**
  - **A dual of a Boolean expression is derived by replacing . by +, + by ., 0 by 1, and 1 by 0 and leaving variables unchanged**
  - **In general duality:** f^D**(x1,x2,…,xn,0,1,+,.) = f(x1,x2,…,xn,1,0,.,+)**
- **Principle of Duality:**
  - **If any theorem can be proven, the dual theorem can also be proven.**
  - **A meta-theorem (a theorem about theorems)**
- **Examples:**
  - **Multiplication and factoring:**
    - (x+y).(x'+z) = x.z+x'.y and x.y+x'.z=(x+z).(x'+y)
  - **Consensus:**
    - (x.y)+(y.z)+(x'.z)=x.y+x'z and

DeMorgan's Theorem in Terms of Logic Gates

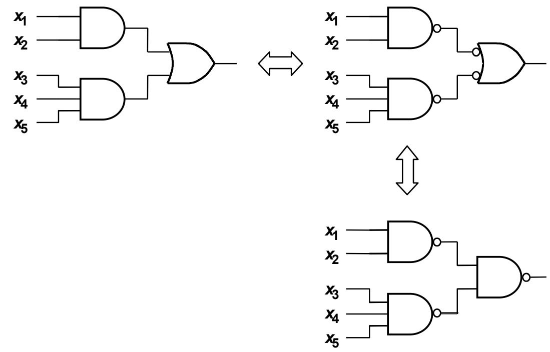


(a) $\overline{x_1 x_2} = \bar{x}_1 + \bar{x}_2$

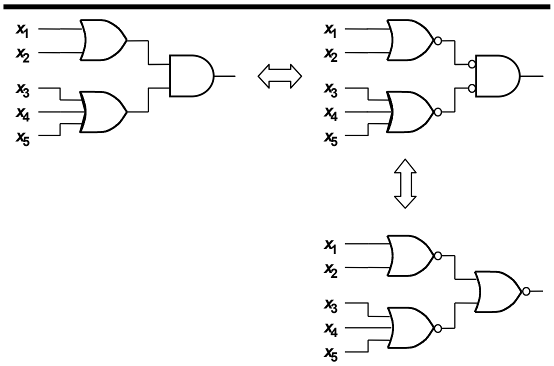


(b) $\overline{x_1 + x_2} = \bar{x}_1 \bar{x}_2$

Using NAND to Implement SOP



Using NOR to Implement POS

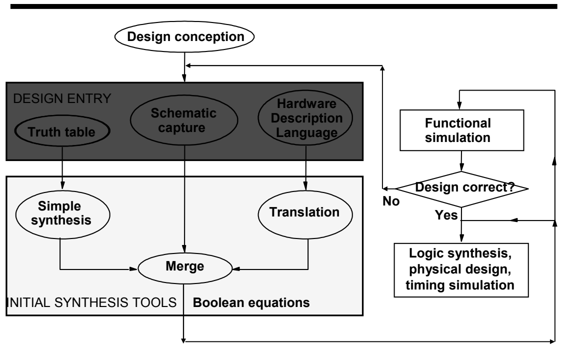


Order of Precedence of Logic Operators

- **From highest precedence to lowest: NOT, AND, OR**
- **We can use parenthesis to change the order**

- **Examples:**
  - **f = X'+X.Y is the same as**
  - **f = ((X')+(X.Y))**
  - **f = X.(Y+Z) is NOT the same as**
  - **f = X.Y+Z**

A Typical CAD (Computer-Aided Design) System



Verilog HDL

Popular Hardware Description Languages (HDLs):

- **Verilog HDL**
  - More popular with US companies
  - Similar to C / Pascal programming language in syntax
- **VHDL**
  - More popular with European companies
  - Similar to Ada programming language in syntax
  - More "verbose" than Verilog

Uses of Verilog:

- **Synthesis**
- **Simulation**
- **Verification**

Verilog Syntax

- **Module / Signal names:**
  - **Start with a letter**
  - **Follow by any sequence of letter, number, _ and $**
  - **Case sensitive**
- **Comment by // or /* */**
- **White spaces (SPACE, TAB, blank line) are ignored.**

```
// An example
module example1 (x1, x2, x3, f);
        input x1, x2, x3;
        output f;

        and (g, x1, x2);
        not (k, x2);
        and (h, k, x3);
        or (f, g, h);

endmodule
```

The following version, which has no line breaks, is crossed out on the slide:

```
// An example
module example1 (x1, x2, x3
, f);input x1, x2, x3;output
f;and (g, x1, x2); not (k, x2
);and (h, k, x3); or (f, g,
h);endmodule
```


Multiplexer Circuit

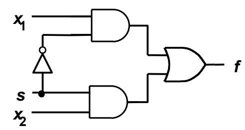
sx_1x_2	$f(s, x_1, x_2)$
0 0 0	0
0 0 1	0
0 1 0	1
0 1 1	1
1 0 0	0
1 0 1	1
1 1 0	0
1 1 1	1

Truth Table

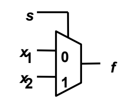
s	$f(s, x_1, x_2)$
0	x_1
1	x_2

More compact truth-table representation

$f = s'.x_1 + s.x_2$

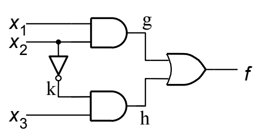


Circuit



Graphical symbol

Structural Specification of Logic Circuit

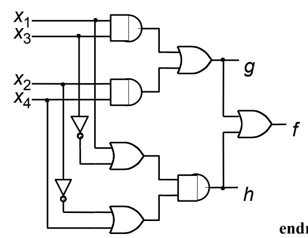



```
module example1 (x1, x2, x3, f);
        input x1, x2, x3;
        output f;

        and (g, x1, x2);
        not (k, x2);
        and (h, k, x3);
        or (f, g, h);

endmodule
```


Another Example of Structural Specification

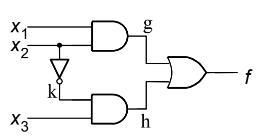



```
module example2 (x1, x2, x3, x4, f, g, h);
        input x1, x2, x3, x4;
        output f, g, h;

        and (z1, x1, x3);
        and (z2, x2, x4);
        or (g, z1, z2);
        or (z3, x1, ~x3);
        or (z4, ~x2, x4);
        and (h, z3, z4);
        or (f, g, h);

endmodule
```


Behavioral Specification Continuous Assignment




```
// Structural Specification
module example1 (x1, x2, x3, f);
        input x1, x2, x3;
        output f;

        and (g, x1, x2);
        not (k, x2);
        and (h, k, x3);
        or (f, g, h);

endmodule
```

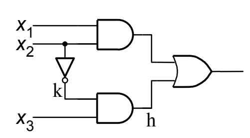
```
// Behavioral Specification
module example3 (x1, x2, x3, f);
        input x1, x2, x3;
        output f;

        assign f = (x1 & x2) | (~x2 & x3);

endmodule
```

Concurrent statement (arrows point to `and (h, k, x3);` and `assign`)

Behavioral Specification Procedural (Sequential) Statement




```
module example5 (x1, x2, x3, f);
    input x1, x2, x3;
    output f;
    reg f;           <- Declared as variable (register) if assigned a value in a procedural statement

    always @(x1 or x2 or x3)    <- Sensitivity list
        if (x2 == 1)
            f = x1;
        else
            f = x3;                   (always block)

endmodule
```

Declared as variable (register) if assigned a value in a procedural statement

Sensitivity list

always *block*