Lecture 2: Boolean Algebra and Logic Gates

CS207: Digital Logic

Jialin Liu

Department of Computer Science and Engineering (CSE)
Southern University of Science and Technology (SUSTech)

16 September 2022

Acknowledgement

These slides were prepared based on the slides by Dr. Jianqiao Yu and the ones by Prof. Georgios Theodoropoulos of the Department of CSE at the SUSTech, as well as the contents of the following book:
M. M. Mano and M. Ciletti, *Digital design: with an introduction to the Verilog HDL*. Pearson, 2013

Recap: Last Week

- **Number systems**: decimal, binary, octal, hexadecimal systems.
- **Number-based conversions**:
  - Base-r system to decimal: for $(a_n a_{n-1} \ldots a_2 a_1 . b_1 b_2 \ldots b_m)_r$,

$$\begin{aligned} & a_n \cdot r^{n-1} + a_{n-1} \cdot r^{n-2} + a_{n-2} \cdot r^{n-3} + \cdots + a_2 \cdot r^1 + a_1 \cdot r^0 \\ & + b_1 r^{-1} + b_2 \cdot r^{-2} + \cdots + b_m \cdot r^{-m} \end{aligned}$$

  - Decimal to base-r system: division for integer part, multiplication for fraction
  - Conversion between binary and octal systems
- **Complements of numbers**: r's complement & $r-1$'s complement
  - For substraction
  - For representing signed binary numbers
- **Representation of data: code**

Exercise

1. Convert $(110.5)_{10}$ to base 3. For repeating decimals, write down four reptends (循环节).

Exercise

Exercise

1. Convert $(110.5)_{10}$ to base 3. For repeating decimals, write down four reptends (循环节).

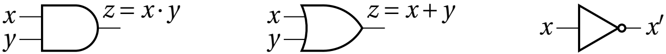
$$
\begin{aligned}
(110)_{10} &= 1*3^4+1*3^3+0*3^2+0*3^1+2*3^0=(11002)_3 \\
(0.5)_{10} &= 1*3^{-1}+1*3^{-2}+1*3^{-3}+1*3^{-4}+1*3^{-5}+1*3^{-6}+\cdots=(0.111111\ldots)_3 \\
(110.5)_{10} &= (11002.1111)_3 \qquad \textit{// write down four reptends}
\end{aligned}
$$

Exponent	Division	Quotient	Remainder
0	110/3	36	2
1	36/3	12	0
2	12/3	4	0
3	4/3	1	1
4	1/3	0	1

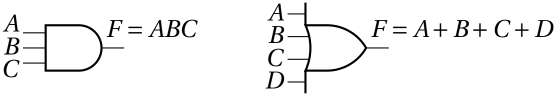
Exponent	Multiplication	Product
-1	$0.5*3$	1.5
-2	$0.5*3$	1.5
-3	$0.5*3$	1.5
-4	$0.5*3$	1.5
-5	$0.5*3$	1.5
-6	$0.5*3$	1.5
…	…	…

Logic Gate (逻辑门)

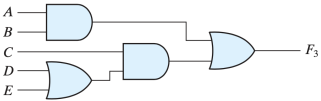
- **Logic gates** are electronic circuits that operates on one or more inputs signals to produce an output.



- It is fine to have more than two inputs for AND/OR.



- Any desired circuit can be realised through various combinations of logic gates.



Why Boolean Algebra (逻辑代数)?

- **Binary logic** is used in all of todays's digital computers and devices.
  - In a digital system, input is given with the help of switches: usually have two distinct discrete levels or values: `ON` and `OFF`.
  - Binary logic deals with variables that take on two discrete values (1 for true or switch `ON` and 0 for false or switch `OFF`) and with operations that assume logical meaning.

- The cost of the digital circuits is an important factor addressed by designers.
  Aim: Reduce cost.
  ← Finding simpler and cheaper, but equivalent, realizations of a circuit.
  ← Mathematical methods to simplify circuits.
  ← Key: Boolean algebra, a deductive mathematical system.

Outline of This Lecture

- Boolean Algebra
- Boolean Function
- Canonical Forms, Minterms, Maxterms
- Other Logic Operations
- Digital Logic Gates
- Summary of this Lecture

Boolean Algebra (逻辑代数)

- **Boolean algebra**, a deductive mathematical system developed by George Boole in 1854, deals with the rules by which logical operations are carried out.

- Boolean algebra is an algebraic structure defined by
  - a set of elements S: binary inputs;
  - a set of binary operators: rules;
  - and a number of unproved postulates.

Postulates (公理)

1. `Closure` (闭包): A set S is closed with respect to a binary operator if, for every pair of elements of S, the binary operator specifies a rule for obtaining a unique element of S.
2. `Associative law` (结合律): $A+(B+C)=(A+B)+C$ and $A\cdot(B\cdot C)=(A\cdot B)\cdot C$.
3. `Commutative law` (交换率): $A+B=B+A$ and $A\cdot B=B\cdot A$.
4. `Identity element` (单比特): A set S is to have an identity element with respect to a binary operation $*$ on S, if there exists an element $E\in S$ with the property $E*A=A*E=A$. Examples:
   - Element 0 is an identity element of + as $A+0=0+A=A$.
   - Element 1 is an identity element of · as $A\cdot 1=1\cdot A=A$.
5. `Distributive law` (分配律): $A\cdot(B+C)=A\cdot B+A\cdot C$ and $A+B\cdot C=(A+B)\cdot(A+C)$.
6. For every element $\in S$, there exists and element $x'\in S$ (called the `complement` of x) such that $x+x'=1$ and $x\cdot x'=0$.

Two-valued Boolean Algebra

- A **two-valued Boolean algebra** is defined on
  - a set of two elements: $\{0, 1\}$;
  - two binary operators $+$ and $\cdot$, and a complement operator $'$.
- It satisfied the aforementioned postulates.

AND

x	y	$x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

OR

x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1

NOT

x	x'
0	1
1	0

In this course, we only deal with two-valued Boolean algebra. From now on, we use "Boolean algebra" to refer to "two-valued Boolean algebra" for short.

Properties of Boolean Algebra

Operator Precedence

- Operator precedence for evaluating Boolean expression:
  1. Parentheses $()$,
  2. NOT $'$;
  3. AND $\cdot$;
  4. OR $+$.

Properties of Boolean Algebra

Duality Property (对偶性)

- Every algebraic expression deducible from the postulates of Boolean algebra remains valid if the operators and identity elements are interchanged.
  - Change $+$ to $\cdot$ and vice versa.
  - Change 0 to 1 and vice versa.
- The **dual** of a Boolean expression is the expression one obtains by interchanging $+$ and $\cdot$ and interchanging 0s and 1s.
- Examples:
  - $A + A' = 1 \rightarrow A \cdot A' = 0$.
  - $A + B = B + A \rightarrow A \cdot B = B \cdot A$.
  - $A \cdot (B + C) = A \cdot B + A \cdot C \rightarrow A + B \cdot C = (A + B) \cdot (A + C)$.
  - $(A + B)' = A' \cdot B' \rightarrow (A \cdot B)' = A' + B'$.

Properties of Boolean Algebra

Basic Postulates and Theorems

Postulate 2	(a)	$x + 0 = x$	(b)	$x \cdot 1 = x$
Postulate 5	(a)	$x + x' = 1$	(b)	$x \cdot x' = 0$
Theorem 1	(a)	$x + x = x$	(b)	$x \cdot x = x$
Theorem 2	(a)	$x + 1 = 1$	(b)	$x \cdot 0 = 0$
Theorem 3, involution		$(x')' = x$		
Postulate 3, commutative	(a)	$x + y = y + x$	(b)	$xy = yx$
Theorem 4, associative	(a)	$x + (y + z) = (x + y) + z$	(b)	$x(yz) = (xy)z$
Postulate 4, distributive	(a)	$x(y + z) = xy + xz$	(b)	$x + yz = (x + y)(x + z)$
Theorem 5, DeMorgan	(a)	$(x + y)' = x'y'$	(b)	$(xy)' = x' + y'$
Theorem 6, absorption	(a)	$x + xy = x$	(b)	$x(x + y) = x$

Figure: Screenshot of Table 2.1 in [1].

Outline of This Lecture

Boolean Algebra

Boolean Function

Canonical Forms, Minterms, Maxterms

Other Logic Operations

Digital Logic Gates

Summary of this Lecture

Boolean Function

Algebraic Expression (代数表达式)

- A Boolean function is an expression formed with **binary variables**, the two **binary operators** `AND` and `OR`, one **unary operator** `NOT`, **parentheses** and **equal sign**.

- A Boolean function describes how to determine the binary output given binary variables as inputs and binary operators.

- The value of a function may be `0` or `1`, depending on the values of variables present in the Boolean function or expression.

- Example: $F = A \cdot B' \cdot C$.
  - $F = 1$ when $A = C = 1$ and $B = 0$,
  - otherwise $F = 0$.

- Boolean functions can also be represented by **truth tables**.
  - Tabular form of the values of a Boolean function according to the all possible values of its variables. n number of variables $\rightarrow 2^n$ combinations of 1's and 0'.s
  - Example: $F = A \cdot B + C$.

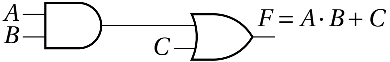
A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

- Each Boolean function has one representation in truth table, but a variety of ways in algebraic form.

Boolean Function Simplification

Logic Diagram (逻辑图)

- A Boolean function from an algebraic expression can be realised to a **logic diagram** composed of logic gates.



- Minimisation of the number of literals and the number of terms leads to less complex circuits as well as lower number of gates.
- We first try use postulates and theorems of Boolean algebra to simplify.

$$
\begin{aligned}
F &= A\cdot B + B\cdot C + B'\cdot C \\
&= A\cdot B + C\cdot(B + B') \\
&= A\cdot B + C
\end{aligned}
$$

$$
\begin{aligned}
F &= A'\cdot B'\cdot C + A'\cdot B\cdot C + A\cdot B' \\
&= A'\cdot C\cdot(B' + B) + A\cdot B' \\
&= A'\cdot C + A\cdot B'
\end{aligned}
$$

Exercise

Exercise

1. Use postulates and theorems of Boolean algebra to simplify $F = A \cdot B \cdot C + A \cdot B' \cdot C + A \cdot B \cdot C'$

Exercise

Exercise

1. Use postulates and theorems of Boolean algebra to simplify $F = A{\cdot}B{\cdot}C + A{\cdot}B'{\cdot}C + A{\cdot}B{\cdot}C'$

$$
\begin{aligned}
F &= A{\cdot}B{\cdot}C + A{\cdot}B'{\cdot}C + A{\cdot}B{\cdot}C' \\
&= A{\cdot}B{\cdot}C + A{\cdot}B'{\cdot}C + A{\cdot}B{\cdot}C + A{\cdot}B{\cdot}C' \\
&= A{\cdot}C{\cdot}(B' + B) + A{\cdot}B{\cdot}(C + C') \\
&= A{\cdot}C + A{\cdot}B \\
&= A{\cdot}(C + B)
\end{aligned}
$$

Before simplifying: 3 terms, 9 literals
After simplifying: 2 terms, 3 literals

Algebraic Manipulation

- Reduce the total number of terms and literals.
- Usually not possible by hand for complex functions, use computer minimisation program.
- More advanced techniques in the next lectures.

Boolean Function Complement

- Complement a Boolean function from F to F'.
  - Change `0`'s to `1`'s and vice versa in the truth table.
  - Use DeMorgan's theorem for multiple variables.
    Recap: $(x+y)' = x' \cdot y'$, $(x \cdot y)' = x' + y'$

- Example: $F = x'yz' + x'y'z$.

Complement:

$$\begin{aligned} F' &= (x'yz' + x'y'z)' \\ &= (x'yz')'(x'y'z)' \\ &= (x+y'+z)(x+y+z') \end{aligned}$$

Dual:

$$F^* = (x'+y+z')(x'+y'+z)$$

Outline of This Lecture

Canonical Forms

- Logical functions are generally expressed in terms of different combinations of logical variables with their true forms as well as the complement forms: x and x'.

- An arbitrary logic function can be expressed in the following forms, called **canonical forms**:
  - **Sum of products** (SOP), and
  - **Product of sums** (POS).

- What are the products and sums?

Canonical Forms

- The logical product of several variables on which a function depends is considered to be a **product term**.
  - Called **minterms** when all variables are involved: For x and y, xy, $x'y$, xy', and $x'y'$ are all the minterms.
- The logical sum of several variables on which a function depends is considered to be a **sum term**.
  - Called **maxterms** when all variables are involved: For x and y, $x+y$, $x'+y$, $x+y'$, and $x'+y'$ are all the maxterms.
- `SOP`: The logical sum of two or more logical product terms is referred to as a **sum of products expression (SOP)**.
- `POS`: The logical product of two or more logical sum terms is referred to as a **product of sums expression (POS)**.

Minterms

- In the minterm, a variable will possess the value 1 if it is in true or uncomplemented form, whereas, it contains the value 0 if it is in complemented form.

A	B	C	Minterm
0	0	0	$A'B'C'$
0	0	1	$A'B'C$
0	1	0	$A'BC'$
0	1	1	$A'BC$
1	0	0	$AB'C'$
1	0	1	$AB'C$
1	1	0	ABC'
1	1	1	ABC

- It possesses the value of 1 for only one combination of n input variables. The rest of the $2^n - 1$ combinations have the logic value of 0.

Sum of Minterms

- **Canonical SOP** expression, or **sum of minterms**:
  A Boolean function expressed as the logical sum of all the minterms from the rows of a truth table with value 1.
- A compact form by listing the corresponding decimal-equivalent codes of the minterms:
  - Example: $F = AB + C = A'B'C + A'BC + AB'C + ABC' + ABC = \sum(1,3,5,6,7)$.

Decimal	A	B	C	F	Minterms
0	0	0	0	0	$A'B'C'$
1	0	0	1	1	$A'B'C$
2	0	1	0	0	$A'BC'$
3	0	1	1	1	$A'BC$
4	1	0	0	0	$AB'C'$
5	1	0	1	1	$AB'C$
6	1	1	0	1	ABC'
7	1	1	1	1	ABC

Procedure of Determining SOP

The canonical sum of products form of a logic function can be obtained by using the following procedure.

1. Check each term in the given logic function.
   - If it is a minterm,
     continue to examine the next term in the same manner.
   - Otherwise,
     - examine for the variables that are missing in each product which is not a minterm.
     - if the missing variable in the minterm is X, multiply that minterm with $(X + X')$.
       Example: $A + B \rightarrow A \cdot (B + B') + (A + A') \cdot B$
2. Combine all the products and discard the redundant terms.

Procedure of Determining SOP

Example

$$
\begin{aligned}
F(A,B,C,D) &= AB + ACD \\
&= AB(C + C')(D + D') + ACD(B + B') \\
&= (ABC + ABC')(D + D') + ABCD + AB'CD \\
&= ABCD + ABCD' + ABC'D + ABC'D' + ABCD + AB'CD \\
&= ABCD + ABCD' + ABC'D + ABC'D' + AB'CD
\end{aligned}
$$

Recap:

1. Check each term in the given logic function.
   - If it is a minterm,
     continue to examine the next term in the same manner.
   - Otherwise,
     - examine for the variables that are missing in each product which is not a minterm.
     - if the missing variable in the minterm is X, multiply that minterm with $(X + X')$.
       Example: $A + B \rightarrow A \cdot (B + B') + (A + A') \cdot B$
2. Combine all the products and discard the redundant terms.

Maxterms

- In the maxterm, a variable will possess the value 0, if it is in true or uncomplemented form, whereas, it contains the value 1, if it is in complemented form.

A	B	C	Maxterm
0	0	0	$A+B+C$
0	0	1	$A+B+C'$
0	1	0	$A+B'+C$
0	1	1	$A+B'+C'$
1	0	0	$A'+B+C$
1	0	1	$A'+B+C'$
1	1	0	$A'+B'+C$
1	1	1	$A'+B'+C'$

- It possesses the value of 0 for only one combination of n input variables. The rest of the $2^n - 1$ combinations have the logic value of 1.

Product of Maxterms

- **Canonical POS** expression, or **product of maxterms**:
  A Boolean function expressed as the logical product of all the maxterms from the rows of a truth table with value 0.

- A compact form by listing the corresponding decimal-equivalent codes of the maxterms:
  - Example: $F = (A+B+C)(A+B'+C)(A'+B+C') = \prod(0,2,5)$.

Decimal	A	B	C	Maxterm
0	0	0	0	$A+B+C$
1	0	0	1	$A+B+C'$
2	0	1	0	$A+B'+C$
3	0	1	1	$A+B'+C'$
4	1	0	0	$A'+B+C$
5	1	0	1	$A'+B+C'$
6	1	1	0	$A'+B'+C$
7	1	1	1	$A'+B'+C'$

- Example: $F(A,B,C) = A + B'C$.

$$\begin{aligned}
F(A,B,C) &= A + B'C \\
&= AA + B'C \\
&= (A+B')(A+C) \text{ using the distributive property: } X+YZ=(X+Y)(X+Z) \\
&= (A+B'+CC')(A+C+BB') \\
&= (A+B'+C)(A+B'+C')(A+B+C)(A+B'+C) \text{ using the distributive property: } X+YZ=(X+Y)(X+Z) \\
&= (A+B'+C)(A+B'+C')(A+B+C)
\end{aligned}$$

Derive from A Truth Table

Decimal	A	B	C	F	Minterm	Maxterm
0	0	0	0	0		$A+B+C$
1	0	0	1	0		$A+B+C'$
2	0	1	0	1	$A'BC'$	
3	0	1	1	0		$A+B'+C'$
4	1	0	0	1	$AB'C'$	
5	1	0	1	1	$AB'C$	
6	1	1	0	1	ABC'	
7	1	1	1	0		$A'+B'+C'$

- The final canonical SOP for the output F is derived by summing or performing an OR operation of the four product terms as shown below:
  - $F = A'BC' + AB'C' + AB'C + ABC' = \sum(2,4,5,6)$.
- The final canonical POS for the output F is derived by summing or performing an AND operation of the four sum terms as shown below:
  - $F = (A+B+C)(A+B+C')(A+B'+C')(A'+B'+C') = \prod(0,1,3,7)$.

Conversion between Minterms and Maxterms

- Minterms are the complement of corresponding maxterms: $m_i = M_i'$.
- Example: $A' + B' + C' = (ABC)'$.

$$
\begin{aligned}
F(A,B,C) &= \sum(2,4,5,6) = m_2 + m_4 + m_5 + m_6 \\
&= A'BC' + AB'C' + AB'C + ABC' \\
F'(A,B,C) &= \sum(0,1,3,7) = m_0 + m_1 + m_3 + m_7 \\
F(A,B,C) &= \left(F'(A,B,C)\right)' = (m_0 + m_1 + m_3 + m_7)' \\
&= m_0' m_1' m_3' m_7' \\
&= M_0 M_1 M_3 M_7 \\
&= \prod(0,1,3,7) \\
&= (A+B+C)(A+B+C')(A+B'+C')(A'+B'+C').
\end{aligned}
$$

Outline of This Lecture

Boolean Algebra

Boolean Function

Canonical Forms, Minterms, Maxterms

Other Logic Operations

Digital Logic Gates

Summary of this Lecture

Other Logic Operations

- When the three operators AND, OR, and NOT are applied on two variables x and y, they form 16 Boolean functions:

Boolean Expressions for the 16 Functions of Two Variables

Boolean Functions	Operator Symbol	Name	Comments
$F_0 = 0$		Null	Binary constant 0
$F_1 = xy$	$x \cdot y$	AND	x and y
$F_2 = xy'$	x/y	Inhibition	x, but not y
$F_3 = x$		Transfer	x
$F_4 = x'y$	y/x	Inhibition	y, but not x
$F_5 = y$		Transfer	y
$F_6 = xy' + x'y$	$x \oplus y$	Exclusive-OR	x or y, but not both
$F_7 = x + y$	$x + y$	OR	x or y
$F_8 = (x + y)'$	$x \downarrow y$	NOR	Not-OR
$F_9 = xy + x'y'$	$(x \oplus y)'$	Equivalence	x equals y
$F_{10} = y'$	y'	Complement	Not y
$F_{11} = x + y'$	$x \subset y$	Implication	If y, then x
$F_{12} = x'$	x'	Complement	Not x
$F_{13} = x' + y$	$x \supset y$	Implication	If x, then y
$F_{14} = (xy)'$	$x \uparrow y$	NAND	Not-AND
$F_{15} = 1$		Identity	Binary constant 1

Figure: Screenshot of Table 2.8 in [1].

Outline of This Lecture

Boolean Algebra

Boolean Function

Canonical Forms, Minterms, Maxterms

Other Logic Operations

Digital Logic Gates

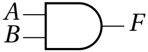
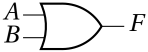
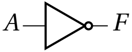
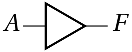
Summary of this Lecture

Construct Logic Gates

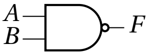
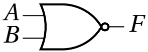
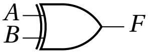
- As Boolean functions are expressed in terms of `AND`, `OR`, and `NOT` operations, it is easier to implement the Boolean functions with these basic types of gates.
  - It is possible to construct other types of logic gates.

- The following factors are to be considered for construction of other types of gates.
  - The `feasibility` and economy of producing the gate with physical parameters.
  - The possibility of `extending` to more than two inputs.
  - The basic properties of the binary operator such as `commutability` and `associability`.
  - The ability of the gate to `implement Boolean functions` alone or in conjunction with other gates.

Digital Logic Gates (1/2)

Name	Graphic symbol	Algebraic function	A	B	F
AND		$F = AB$	0	0	0
			0	1	0
			1	0	0
			1	1	1
OR		$F = A + B$	0	0	0
			0	1	1
			1	0	1
			1	1	1
NOT		$F = A'$	0	-	1
			1	-	0
Buffer		$F = A$	0	-	0
			1	-	1

Digital Logic Gates (2/2)

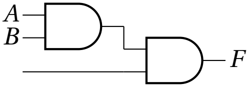
Name	Graphic symbol	Algebraic function	A	B	F
NAND		$F = (AB)'$	0	0	1
			0	1	1
			1	0	1
			1	1	0
NOR		$F = (A + B)'$	0	0	1
			0	1	0
			1	0	0
			1	1	0
XOR		$F = AB' + A'B$ $= A \oplus B$	0	0	0
			0	1	1
			1	0	1
			1	1	0

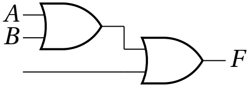
Multiple Input Logic Gates

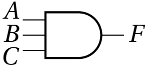
AND and OR

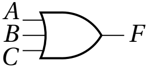
- A gate can be extended to have multiple inputs if its binary operation is commutative and associative.

- AND and OR gates are both commutative and associative.
  - $F = ABC = (AB)C$.
  - $F = A + B + C = (A + B) + C$.







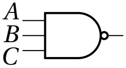


Multiple Input Logic Gates

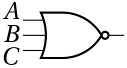
NAND and NOR

- The NAND and NOR functions are the complements of AND and OR functions respectively.
  - They are commutative, but not associative.
  - $((AB)'C)' \neq (A(BC)')'$: does not follow associativity.
  - $((A+B)'+C)' \neq (A+(B+C)')'$: does not follow associativity.

- We modify the definition of multi-input NAND and NOR:



$F = (ABC)' = A' + B' + C'$

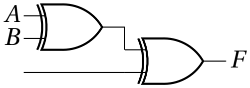


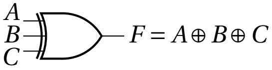
$F = (A+B+C)' = A'B'C'$

Multiple Input Logic Gates

XOR

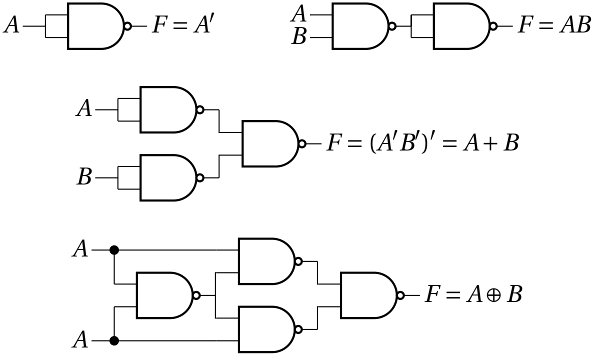
- The XOR gates and equivalence gates both possess commutative and associative properties.
  - Gate output is low when even numbers of 1's are applied to the inputs, and when the number of 1's is odd the output is logic 0.
  - Multiple-input exclusive-OR and equivalence gates are uncommon in practice.





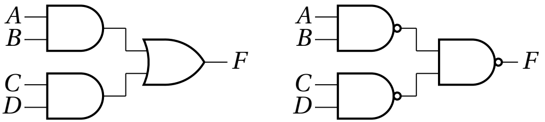
Universal Gates (1/2)

- NAND gates and NOR gates are called **universal gates** or **universal building blocks**.
  - Any type of gates or logic functions can be implemented by these gates.



Universal Gates (2/2)

- Universal gates are easier to fabricate with electronic components.
- Also reduce the number of varieties of gates.
- Example: $F = AB + CD$ requires
  - two AND and one OR gates,
  - or three NAND gates: $F = AB + CD = ((AB + CD)')' = ((AB)'(CD)')'$



Outline of This Lecture

Boolean Algebra

Boolean Function

Canonical Forms, Minterms, Maxterms

Other Logic Operations

Digital Logic Gates

Summary of this Lecture

Summary

- Boolean algebra help reduce the complexity and size of digital circuit (reduce the total number of terms and literals).
  - Postulates of Boolean algebra are key to it.
- Boolean functions can be expressed by algebraic expressions, true tables or logic diagrams. (We will see other representations in the future.)
- An arbitrary logic function can be expressed in canonical forms, thus sum of products (SOP) and product of sums (POS), determined by minterms or maxterms.
- It is possible to construct other types of logic gates (and of multiple inouts) with `AND`, `OR`, and `NOT` operations.

Next week: Gate-Level Minimisation (逻辑化简) using Karnaugh map.

Essential Reading

- Essential reading for this lecture: pages 38-68 of the textbook.
- Essential reading for next lecture: pages 73-118 of the textbook.

[1] M. M. Mano and M. Ciletti, *Digital design: with an introduction to the Verilog HDL*. Pearson, 2013