Two-level Implementation

CS207 Chapter 4

James YU
yujq3@sustech.edu.cn

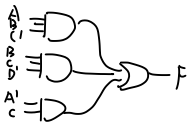
Department of Computer Science and Engineering
Southern University of Science and Technology

Sept. 28, 2022

NAND and NOR implementation

- Digital circuits are frequently constructed with NAND or NOR gates rather than with AND and OR gates.
  - NAND and NOR gates are easier to fabricate with electronic components.
  - They are the basic gates used in all IC digital logic families.
- Rules and procedures have been developed for the conversion from Boolean functions given in terms of AND, OR, and NOT into equivalent NAND and NOR logic diagrams.

eg. $AB(C)' + B(C)' + A'C$

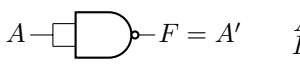


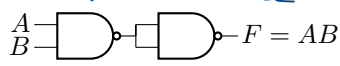
2 level implementation
(in theory)

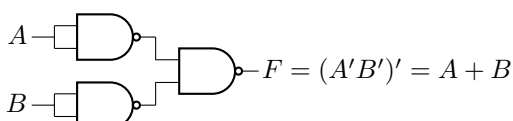
NAND circuits

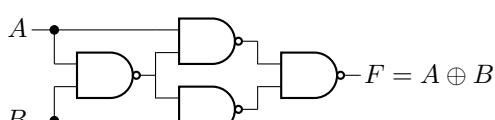
- The NAND gate is said to be a universal gate.
- A convenient way to implement a Boolean function with NAND gates:
  - Obtain the simplified Boolean function in terms of Boolean operators;
  - Convert the function to NAND logic.
- Recall that:

$(AB)' = A' + B'$ $\quad [(A'+B')(A'+B')]' = (A'+B')' = A''B'' = AB$









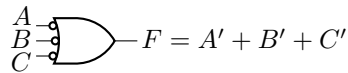
NAND circuits

- To facilitate the conversion to NAND logic, it is convenient to define an alternative graphic symbol for the gate.

AND-invert:

=

Invert-OR:

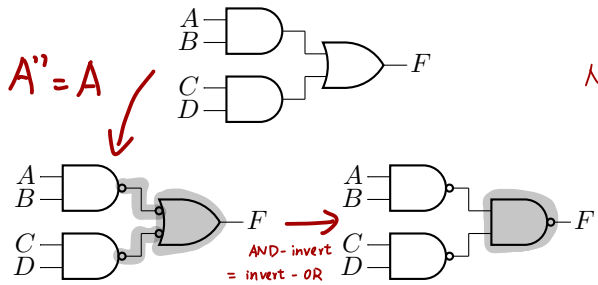


only difference is where to place invert

NAND circuits

- The implementation of Boolean functions with NAND gates requires that the functions be in sum-of-products form.
- Take $F = AB + CD$ as an example:

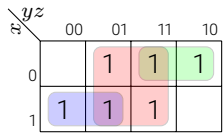
AND & OR gates
↓ pairs of bubbles
NAND & NOR gates



- $F = AB + CD = ((AB)'(CD)')'$ according to DeMorgan property.

NAND circuits

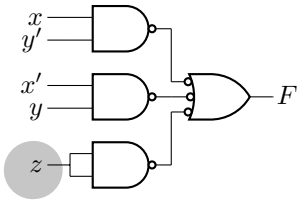
- Example: Implement the following Boolean function with NAND gates: $F(x, y, z) = \sum(1, 2, 3, 4, 5, 7)$.

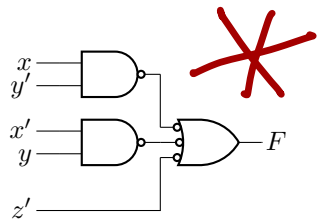


- $F = xy' + x'y + z$.

NAND circuits

- Example: Implement the following Boolean function with NAND gates: $F = xy' + x'y + z$.





the input of z/z' are both acceptable

bubbles NOT need to always appear in pair

NAND circuits

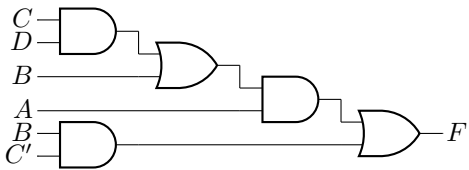
- A Boolean function can be implemented with two levels of NAND gates.
  1. Simplify the function and express it in **sum-of-products form**.
  2. Draw **a NAND gate for each product term** of the expression that has at least two literals. The inputs to each NAND gate are the literals of the term. This procedure produces a group of first-level gates.
  3. Draw **a single gate** using the AND-invert or the invert-OR graphic symbol **in the second level**, with inputs coming from outputs of first-level gates.
  4. A term with **a single literal requires an inverter** in the first level. However, if the single literal is complemented, it can be connected directly to an input of the second level NAND gate.

a thing that completes or brings to perfection:

Multilevel NAND circuits

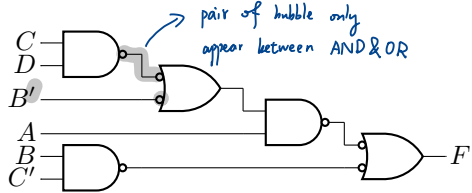
rarely used because gate delay

- The standard form of expressing Boolean functions results in a two-level implementation.
  - There are occasions when the design of digital systems results in gating structures with three or more levels.
  - Example: $F = A(CD + B) + BC'$.



Multilevel NAND circuits

- The standard form of expressing Boolean functions results in a two-level implementation.
  - There are occasions when the design of digital systems results in gating structures with three or more levels.
  - Example: $F = A(CD + B) + BC'$.



Multilevel NAND circuits

- The general procedure for converting a multilevel AND–OR diagram into an all-NAND diagram using mixed notation is as follows:
  1. Convert all AND gates to NAND gates with AND-invert graphic symbols.
  2. Convert all OR gates to NAND gates with invert-OR graphic symbols.
  3. Check all the bubbles in the diagram. For every bubble that is not compensated by another small circle along the same line, insert an inverter (a one-input NAND gate) or complement the input literal.

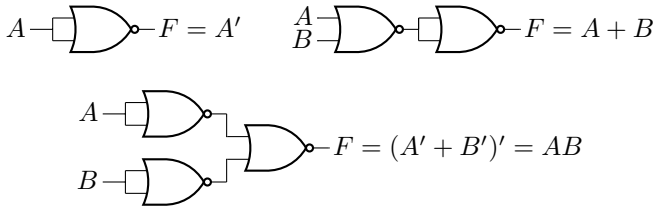
give (someone) something, typically money, in recognition of loss, suffering, or injury incurred;

convert AND & OR gates to NAND

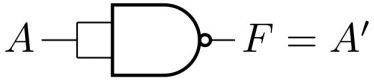
By adding pair of bubbles or bubble & inverter

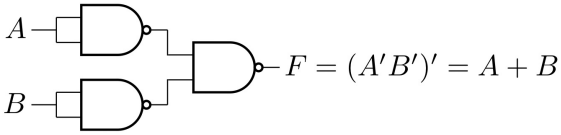
NOR circuits

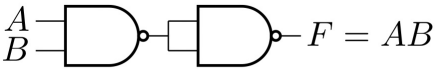
- The NOR operation is the dual of the NAND operation.
- All procedures and rules for NOR logic are the duals of the corresponding procedures and rules developed for NAND logic.



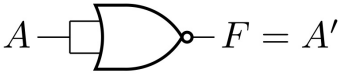
NAND

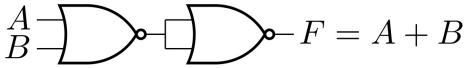


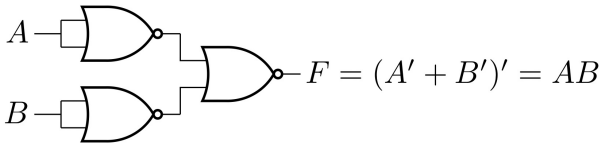




NOR



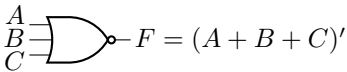




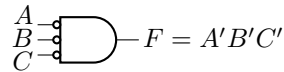
NOR circuits

- To facilitate the conversion to NOR logic, it is convenient to define an alternative graphic symbol for the gate.

OR-invert:



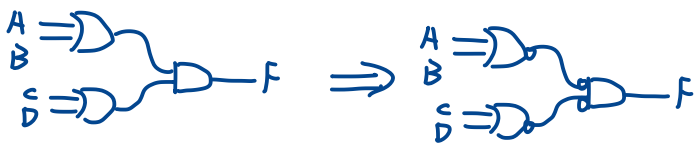
Invert-AND:



NOR circuits

- A two-level implementation with NOR gates requires that the function be simplified into product-of-sums form.
- Change the OR gates to NOR gates with OR-invert graphic symbols and the AND gate to a NOR gate with an invert-AND graphic symbol.

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



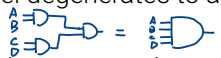
$(A')' = A$

Non-degenerate forms

Something that is informative gives you useful information.
useful and informative:

- It will be instructive from a theoretical point of view to find out how many two-level combinations of gates are possible.
- We consider four types of gates: AND, OR, NAND, and NOR.
  - There are 16 possible combinations of two-level forms.
- Eight of these combinations are said to be *degenerate* forms.
  - They degenerate to a single operation.
  - Example: AND in the first level and second level degenerates to an AND of all inputs.

If you say that someone or something degenerates, you mean that they become worse in some way, for example weaker, lower in quality, or more dangerous.



- The remaining eight nondegenerate forms produce an implementation in sum-of-products form or product-of-sums form.
  1) AND-OR 2) OR-AND 3) NAND-NAND 4) NOR-NOR
  5) NOR-OR 6) NAND-AND 7) OR-NAND 8) AND-NOR

NOT - AND = OR - NOT

$A' \cdot B' = (A+B)'$

AND-OR-INVERT implementation

AND-NOT-AND → NOT-AND = OR-NOT → AND-OR-NOT

- The two forms, NAND–AND and AND–NOR, are equivalent.
  - Both perform the AND–OR–INVERT function.
  - Example: $F = (AB + CD + E)'$.
- An AND–OR implementation requires an expression in sum-of-products form.
- The AND–OR–INVERT implementation is similar, except for the inversion.
  - If the complement of the function is simplified into sum-of-products form (by combining the 1's in the map), it will be possible to implement F with the AND–OR part of the function.

inversion: the action of inverting something or the state of being inverted

NOT-OR-NOT = AND

$= (A' + B')'$

$= A'' \cdot B''$

$= AB$

NOT-AND-NOT = OR

$= (A' \cdot B')'$

$= A'' + B''$

$= A + B'$

OR-AND-INVERT implementation

- The two forms, OR-NAND and NOR-OR, are equivalent.
  - Both perform the OR-AND-INVERT function.
  - Example: $F = [(A + B)(C + D)E]'$.
- The AND-OR-INVERT implementation requires an expression in product-of-sums form.

Exclusive-OR function

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

not universal

Something that is exclusive is used or owned by only one person or group, and not shared with anyone else.

- Exclusive-OR, *XOR*: $x \oplus y = xy' + x'y.$
- Exclusive-NOR, *XNOR* or *equivalency*: $(x \oplus y)' = xy + x'y'.$
- The following identities apply to the XOR operation:
  - $x \oplus 0 = x.$
  - $x \oplus 1 = x'.$ ⟹ serve us NOR gate
  - $x \oplus x = 0.$
  - $x \oplus x' = 1.$
  - $x \oplus y' = x' \oplus y = (x \oplus y)'.$

XOR $x \oplus y$ { $x=y$: 0; $x \neq y$: 1

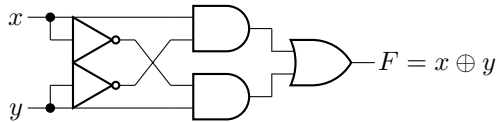
XNOR $(x \oplus y)'$ { $x=y$: 1; $x \neq y$: 0

A	B	XOR	XNOR
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

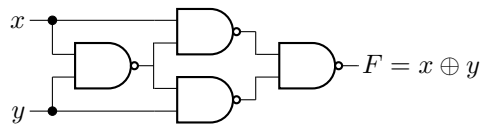
XOR

If someone fabricates information, they invent it in order to deceive people
If something is fabricated from different materials or substances, it is made out of those materials or substances.

- XOR is hard to fabricate, so it is typically constructed by other gates.
  - $(x' + y')x + (x' + y')y = xy' + xy' = x \oplus y.$



- Or use NAND gates:



  - The first NAND gate performs the operation $(xy)' = x' + y'$.
  - The other two-level NAND circuit produces the sum of products.

XOR

- Only a limited number of Boolean functions can be expressed in terms of XOR operations.
- Particularly useful in arithmetic operations and error detection/correction circuits.

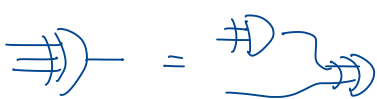
party cheering

help message to be write

communitive & associative

$$A \oplus B = B \oplus A$$

$$(A \oplus B) \oplus C = A \oplus (B \oplus C) = A \oplus B \oplus C$$



异或

The Ex-or gate plays an important role in constructing digital circuits that perform arithmetic operations and calculations. Especially Adders and Half-Adders,

as they can provide a "carry-bit" function or as a controlled inverter, where one input passes the binary data, and the other input is supplied with a control signal.

Something that is exclusive is used or owned by only one person or group, and not shared with anyone else.

2. Pseudo-random Number Generation

It can be arranged in such a manner so that they form a linear feedback type shift register. The linear shift register is also known as a Pseudo-random number generator.

3. Sequence Detection and Correlation

It has the functionality to produce the low output (0) when both the inputs are high are both the inputs are low. When we have a long data sequence, and we have a requirement to search any particular bit sequence, It can be used for finding the data bit sequence. The accuracy can also be determined by calculating the 0 counts in the target sequence. It is used for correlation as they are attached in the decoder to extract parity from it.

Odd function

- The XOR operation with three or more variables can be converted into an ordinary Boolean function by replacing the $\oplus$ with its equivalent Boolean expression.

$(AB'+A'B)' = (AB')' \cdot (A'B)'$

$= (A'+B)(A+B') = AB + A'B'$

$x \oplus y$

$= xy' + x'y$

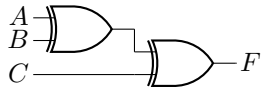
$$\begin{aligned} A \oplus B \oplus C &= (AB' + A'B)C' + (AB + A'B')C \\ &= AB'C' + A'BC' + ABC + A'B'C \\ &= \sum(1, 2, 4, 7) \end{aligned}$$

- The three-variable XOR function is equal to 1 if only one variable is equal to 1 or if all three variables are equal to 1.
  - An **odd** number of variables are equal to 1.
  - *Odd function*.

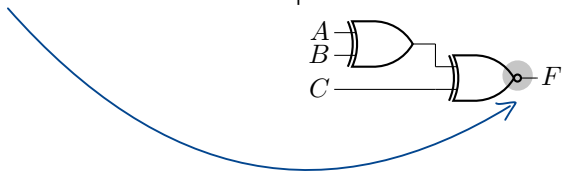
only when odd number of variables equals to 1 the function is equal to 1

Odd function

- The three-input odd function is implemented by means of two-input XOR gates.



- *Even function* can also be implemented:



Parity generation and checking

the state or condition of being equal, especially regarding status or pay:

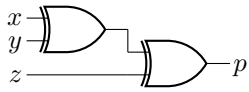
- XOR functions are very useful in systems requiring error detection and correction codes.
  - A parity bit is an extra bit included with a binary message to make the number of 1's either odd or even.
- The circuit that generates the parity bit in the transmitter is called a *parity generator*.
- The circuit that checks the parity in the receiver is called a *parity checker*.

Parity generation and checking

- Consider a three-bit message to be transmitted together with an even-parity bit.

x	y	z	Parity bit p
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

if the message has odd number of 1, then p=1



- p constitutes an odd function.

If x&y&z have odd number of 1
the p=1, make number of in x&y&z&p is even.

Parity generation and checking

- The three bits in the message, together with the parity bit, are transmitted to their destination.
- The four bits received must have an even number of 1's with even parity.