Lecture 4: Gate-level Implementation

CS207: Digital Logic

Jialin Liu

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

30 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: Lecture 2 - Complement of Boolean Function

- **Complement of a Boolean function** ← A vocabulary / definition.

- Determining the complement of a Boolean function:
  - Change 0's to 1's and vice versa in the truth table.
  - Use *DeMorgan's theorem* for multiple variables.
    DeMorgan's theorem: $(x+y)' = x' \cdot y'$, $(x \cdot y)' = x' + y'$

Recap: Lecture 2 - Complement of Boolean Function

- **Complement of a Boolean function** ← A vocabulary / definition.
- Determining the complement of a Boolean function:
  - Change 0's to 1's and vice versa in the truth table.
  - Use *DeMorgan's theorem* for multiple variables.
    DeMorgan's theorem: $(x+y)' = x' \cdot y'$, $(x \cdot y)' = x' + y'$
- Example: $F_1(A, B, C) = A + B'C$

$$\begin{aligned} \text{Its complement } F_1'(A, B, C) &= (A + B'C)' \\ &= A'(B'C)' \\ &= A'(B + C') \end{aligned}$$

Recap: Lecture 2 - Dual of Boolean Expression

- **Dual of a Boolean expression**: ← A vocabulary / definition.
- Determining the dual of a Boolean expression:
  - interchanging + and ·, and interchanging 0s and 1s.
- Example: $F_1(A, B, C) = A + B'C$

Its dual $F_1^*(A, B, C) = A(B' + C)$

Decimal	A	B	C	$F_1 = A + B'C$	$F_1' = A'(B + C')$	$F_1^* = A(B' + C)$
0	0	0	0	0	1	0
1	0	0	1	1	0	0
2	0	1	0	0	1	0
3	0	1	1	0	1	0
4	1	0	0	1	0	1
5	1	0	1	1	0	1
6	1	1	0	1	0	0
7	1	1	1	1	0	1

Recap: Lecture 2 - Duality Property

- **Duality property**: ← A property.

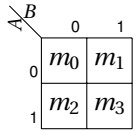
  Every algebraic expression deducible from the postulates of Boolean algebra remains valid if the operators and identity elements are interchanged.

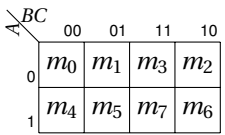
  - Example: $F_1(A,B,C) = A + B'C$ and $F_2(A,B,C) = (A+B')(A+C)$
    According to *Duality property*: If $F_1(A,B,C) = F_2(A,B,C)$ holds, then $F_1^*(A,B,C) = F_2^*(A,B,C)$ holds.

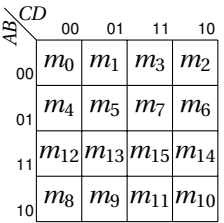
Decimal	A	B	C	$F_1 = A + B'C$	$F_2 = (A+B')(A+C)$	$F_1^* = A(B'+C)$	$F_2^* = AB' + AC$
0	0	0	0	0	0	0	0
1	0	0	1	1	1	0	0
2	0	1	0	0	0	0	0
3	0	1	1	0	0	0	0
4	1	0	0	1	1	1	1
5	1	0	1	1	1	1	1
6	1	1	0	1	1	0	0
7	1	1	1	1	1	1	1

Recap: Lecture 3 - The Map Method for Gate-level Minimisation I

- **Karnaugh map**
  - The map is a diagram consisting of **square**s or **cell**s. For n variables on a Karnaugh map there are 2^n numbers of squares.
  - Each square represents one of the **minterm**s of the function that is to be minimised.
  - Since any Boolean function can be expressed as a sum of minterms, it is possible to recognise a Boolean function graphically in the map from the area enclosed by those squares whose minterms appear in the function.







Recap: Lecture 3 - The Map Method for Gate-level Minimisation II

- Properties of the adjacent squares:
  - Any two adjacent squares in the Karnaugh map differ by only one variable, which is complemented in one square and uncomplemented in one of the adjacent squares.
  - The sum of two minterms in adjacent squares can be simplified to a single `AND` term consisting of fewer literals.

- Remarks:
  - The simplified expressions produced by the map are always in one of the two standard forms: **sum of products** or **product of sums**.
    → So we can use either a map of minterms or a map of maxterms.
  - The **simplest algebraic expression** is an algebraic expression with *a minimum number of terms and with the smallest possible number of literals in each term*.
  - The simplest expression may not be unique. In that case, either solution is satisfactory.

Recap: Lecture 3 - K-map of Minterms

- Simplify $F(A,B,C,D) = \sum(0,1,2,4,5,6,8,9,12,13,14)$.

Recap: Lecture 3 - K-map of Minterms

- Simplify $F(A,B,C,D) = \sum(0,1,2,4,5,6,8,9,12,13,14)$.
  - **Step 1**: Fill the K-map with the presence of minterms.

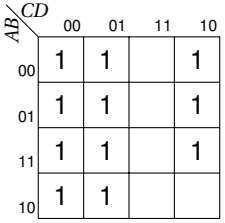
AB \ CD	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

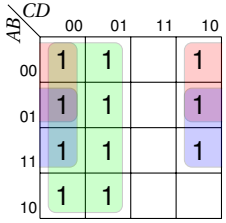
AB \ CD	00	01	11	10
00	1	1		1
01	1	1		1
11	1	1		1
10	1	1		

Recap: Lecture 3 - K-map of Minterms

- Simplify $F(A,B,C,D) = \sum(0,1,2,4,5,6,8,9,12,13,14)$.
  - **Step 1**: Fill the K-map with the presence of minterms.
  - **Step 2**: Simplify the expression using properties of the adjacent squares (combining the pairs, quads, octets of adjacent squares of 1's).

AB \ CD	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}





$\sum(0,2,4,6) = A'D'$, $\sum(4,6,12,14) = BD'$, $\sum(0,1,4,5,12,13,8,9) = C'$.

$F(A,B,C,D) = \sum(0,1,2,4,5,6,8,9,12,13,14) = A'D' + BD' + C'$

Recap: Lecture 3 - K-map of Maxterms

- When an expression is given in respect to the maxterms. In a K-map, 0's are to placed instead of 1's at the corresponding maxterm squares.

- Two ways of simplifying an expression:
  - **Way 1:** Simplify the expression using properties of the adjacent squares (combining the pairs, quads, octets of adjacent squares of `0`'s).
  - **Way 2:** Consider the 1's of the K-map. Thus, combining the pairs, quads, octets of adjacent squares of `1`'s.

Quiz

1. Simplify $F(W,X,Y,Z) = \prod(0,1,2,3,4,5,6,7,9,13,14,15)$ using **K-map of maxterms**.
2. Simplify the above expression using **K-map of minterms**.

- **Count as attendance in your final mark of this course.**
  - A wrong answer will be treated equally as a correct answer.
  - A meaningless answer receives 0 mark.
- **Requirement:** Take a piece of paper, write down
  1. your name,
  2. your student number,
  3. your answer on a piece of paper. **Your answer should be justified**, thus, you should also write down the procedure of finding the answer.
- **Submission:**
  - **Onsite students**: I will collect your paper in the classroom 10 minutes later.
  - **International students**: Send a photo of your solution to *liujl@sustech.edu.cn* while I am collecting the answers of onsite students.
  - Late submissions receive 0 mark.

Quiz

1. Simplify $F(W,X,Y,Z) = \prod(0,1,2,3,4,5,6,7,9,13,14,15)$ using **K-map of maxterms**.
2. Simplify the above expression using **K-map of minterms**.

RECAP Lecture 2:

Minterms are the complement of corresponding maxterms, i.e., $m_i = M_i'$.
Maxterms are the complement of corresponding minterms, i.e., $M_i = m_i'$.

$$
\begin{aligned}
F &= \prod(0,1,2,3,4,5,6,7,9,13,14,15) = M_0 \cdot M_1 \cdot M_2 \cdot M_3 \cdot M_4 \cdot M_5 \cdot M_6 \cdot M_7 \cdot M_9 \cdot M_{13} \cdot M_{14} \cdot M_{15} \\
F' &= (M_0 \cdot M_1 \cdot M_2 \cdot M_3 \cdot M_4 \cdot M_5 \cdot M_6 \cdot M_7 \cdot M_9 \cdot M_{13} \cdot M_{14} \cdot M_{15})' \\
&= M_0' + M_1' + M_2' + M_3' + M_4' + M_5' + M_6' + M_7' + M_9' + M_{13}' + M_{14}' + M_{15}' \\
&= m_0 + m_1 + m_2 + m_3 + m_4 + m_5 + m_6 + m_7 + m_9 + m_{13} + m_{14} + m_{15}
\end{aligned}
$$

Quiz

1. Simplify $F(W,X,Y,Z) = \prod(0,1,2,3,4,5,6,7,9,13,14,15)$ using **K-map of maxterms**.
2. Simplify the above expression using **K-map of minterms**.

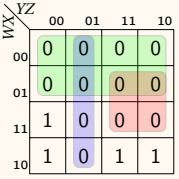
Answer:

Quiz

1. Simplify $F(W,X,Y,Z) = \prod(0,1,2,3,4,5,6,7,9,13,14,15)$ using **K-map of maxterms**.
2. Simplify the above expression using **K-map of minterms**.

Answer:

- Using K-map of maxterms:



$$
\begin{aligned}
F &= \prod(0,1,2,3,4,5,6,7,9,13,14,15) \\
F' &= m_0 + m_1 + m_2 + m_3 + m_4 + m_5 + m_6 + m_7 + m_9 + m_{13} + m_{14} + m_{15} \\
&= W' + Y'Z + XY \\
F &= (W' + Y'Z + XY)' = W(Y + Z')(X' + Y')
\end{aligned}
$$

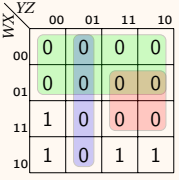
Quiz

1. Simplify $F(W,X,Y,Z) = \prod(0,1,2,3,4,5,6,7,9,13,14,15)$ using **K-map of maxterms**.
2. Simplify the above expression using **K-map of minterms**.

Answer:

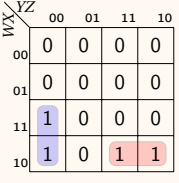
- Using K-map of maxterms:

$$
\begin{aligned}
F &= \prod(0,1,2,3,4,5,6,7,9,13,14,15) \\
F' &= m_0 + m_1 + m_2 + m_3 + m_4 + m_5 + m_6 + m_7 + m_9 + m_{13} + m_{14} + m_{15} \\
&= W' + Y'Z + XY \\
F &= (W' + Y'Z + XY)' = W(Y + Z')(X' + Y')
\end{aligned}
$$



- Using K-map of minterms:

$$
\begin{aligned}
F &= \prod(0,1,2,3,4,5,6,7,9,13,14,15) = \sum(8,10,11,12) \\
&= m_8 + m_{10} + m_{11} + m_{12} \\
&= WY'Z' + WX'Y
\end{aligned}
$$



Lecture 4: Gate-level Implementation

NAND and NOR Implementation
- NAND Circuits
- NOR Circuits

Other Two-Level Implementations

Exclusive-OR Function
- XOR
- XOR for Error Detection

Summary

Outline of This Lecture

NAND and NOR Implementation

- NAND Circuits
- NOR Circuits

Other Two-Level Implementations

Exclusive-OR Function

Summary

NAND and NOR Implementation

- Recap: NAND gates (与非门) and NOR gates (或非门) are called **universal gates** or **universal building blocks**.
  - Any type of gates or logic functions can be implemented by these gates.
- 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.

Outline

NAND and NOR Implementation

- NAND Circuits
- NOR Circuits

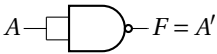
Other Two-Level Implementations

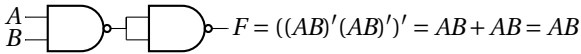
Exclusive-OR Function

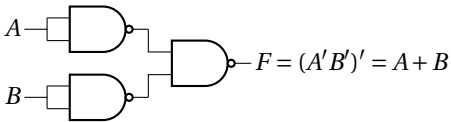
Summary

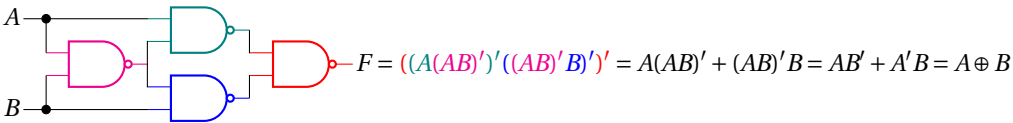
NAND Circuits

Examples





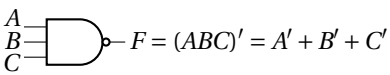




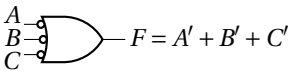
- A convenient way to implement a Boolean function with NAND gates:
  1. Obtain its simplified Algebraic expression.
  2. Convert the Algebraic expression to NAND logic.

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

AND-invert:

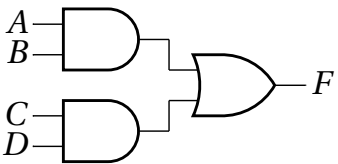


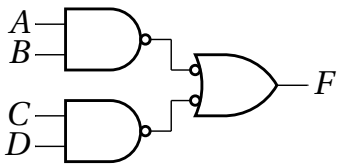
Invert-OR:

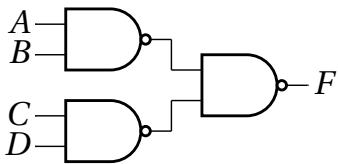


NAND Circuits

- The implementation of Boolean functions with `NAND` gates requires that the expressions be in sum-of-products form.
- Example: $F = AB + CD$
  - $F = AB + CD = ((AB)'(CD)')'$ according to **DeMorgan property**.







NAND Circuits

Example 1/2

- Example: Implement the following Boolean function with NAND gates:

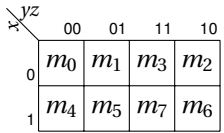
$$F(x, y, z) = \sum(1, 2, 3, 4, 5, 7).$$

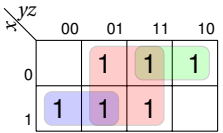
- Example: Implement the following Boolean function with NAND gates:

$$F(x,y,z) = \sum(1,2,3,4,5,7).$$

  - Step 1: Obtain its simplified Algebraic expression.

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

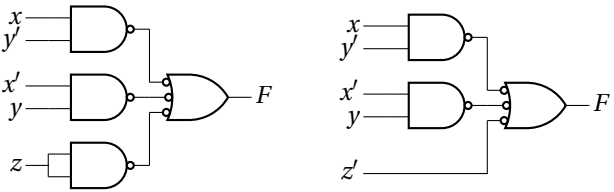




- Example: Implement the following Boolean function with NAND gates:

$$F(x, y, z) = \sum(1, 2, 3, 4, 5, 7) = xy' + x'y + z.$$

  - Step 1: Obtain the simplified Algebraic expression.
  - Step 2: Convert $F = xy' + x'y + z$ to NAND logic. ← Procedure detailed in the next slides.



Two-level Implementation with NAND Circuits

Example

- Implement $F(x, y, z) = \sum(1, 2, 3, 4, 5, 7)$ with two levels of NAND gates.

Two-level Implementation with NAND Circuits

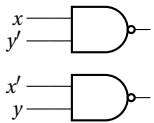
Example

- Implement $F(x, y, z) = \sum(1, 2, 3, 4, 5, 7)$ with two levels of NAND gates.
  1. Simplify the function and express it in sum-of-products form: $F(x, y, z) = xy' + x'y + z$.

Two-level Implementation with NAND Circuits

Example

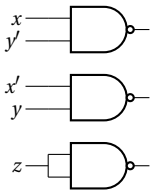
- Implement $F(x, y, z) = \sum(1, 2, 3, 4, 5, 7)$ with two levels of NAND gates.
  1. Simplify the function and express it in sum-of-products form: $F(x, y, z) = xy' + x'y + z$.
  2. Draw a NAND gate for each product term of the expression that has at least two literals. The inputs to each NAND gate are the literals of the term. This procedure produces a group of **first-level gates**.



Two-level Implementation with NAND Circuits

Example

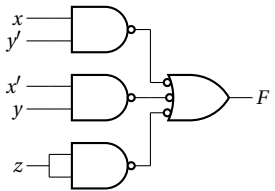
- Implement $F(x, y, z) = \sum(1, 2, 3, 4, 5, 7)$ with two levels of NAND gates.
  1. Simplify the function and express it in sum-of-products form: $F(x, y, z) = xy' + x'y + z$.
  2. Draw a NAND gate for each product term of the expression that has at least two literals. The inputs to each NAND gate are the literals of the term. This procedure produces a group of **first-level gates**.
  3. 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.



Two-level Implementation with NAND Circuits

Example

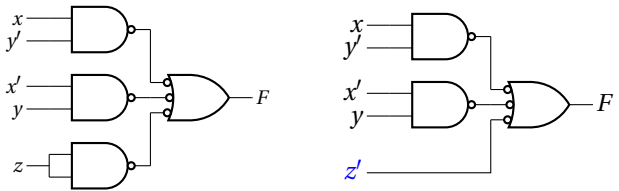
- Implement $F(x, y, z) = \sum(1, 2, 3, 4, 5, 7)$ with two levels of NAND gates.
  1. Simplify the function and express it in sum-of-products form: $F(x, y, z) = xy' + x'y + z$.
  2. Draw a NAND gate for each product term of the expression that has at least two literals. The inputs to each NAND gate are the literals of the term. This procedure produces a group of **first-level gates**.
  3. 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.
  4. 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.



Two-level Implementation with NAND Circuits

Example

- Implement $F(x, y, z) = \sum(1, 2, 3, 4, 5, 7)$ with two levels of NAND gates.
  1. Simplify the function and express it in sum-of-products form: $F(x, y, z) = xy' + x'y + z$.
  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. 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.
  4. 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.



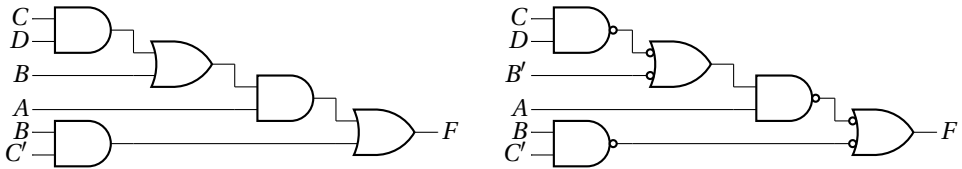
Two-level Implementation with NAND Circuits

Procedure

- A Boolean function can be implemented with **two levels of NAND gates**.
  1. Simplify the Algebraic expression 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. 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.
  4. 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.

Multilevel NAND Circuits

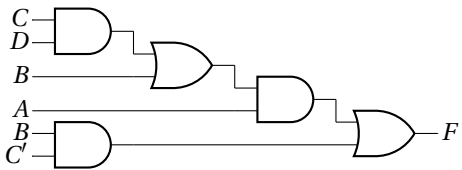
- The standard form of expressing Boolean functions results in a two-level implementation.
- There are occasions when the design of digital systems results in gating structures with three or more levels.
  - Example: $F = A(CD + B) + BC'$.

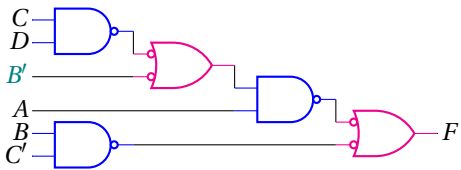


Multilevel NAND Circuits

Example

- Convert the following multilevel AND–OR diagram into an all-NAND diagram using mixed notation.
  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.





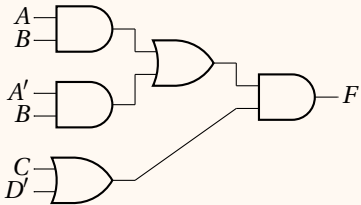
Multilevel NAND Circuits

Procedure

- General procedure for converting a multilevel AND–OR diagram into an all-NAND diagram using mixed notation:
  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.

Exercise

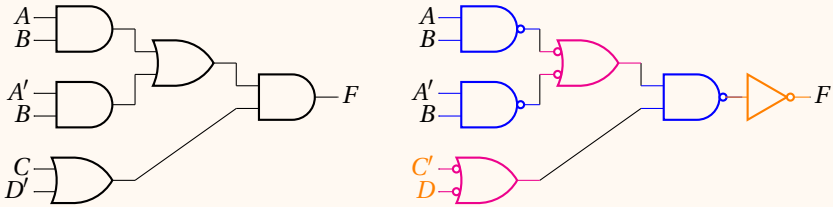
- Convert the following multilevel AND–OR diagram into an all-NAND diagram using mixed notation.



- Hints:
  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.

Exercise

- Convert the following multilevel AND–OR diagram into an all-NAND diagram using mixed notation.



- Hints:
  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.

Outline

NAND and NOR Implementation

- NAND Circuits
- NOR Circuits

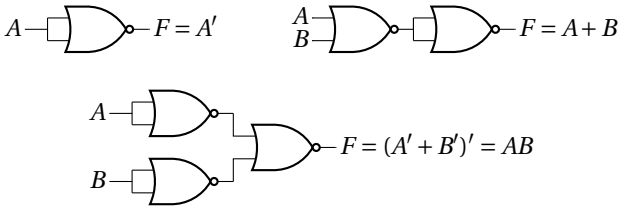
Other Two-Level Implementations

Exclusive-OR Function

Summary

NOR Circuits

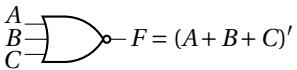
- The NOR operation is the dual of the NAND operation.
- All procedures and rules for NOR logic are the duals of the corresponding procedures and rules developed for NAND logic.



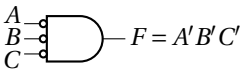
NOR Circuits

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

OR-invert:

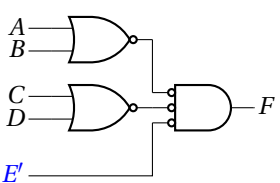


Invert-AND:

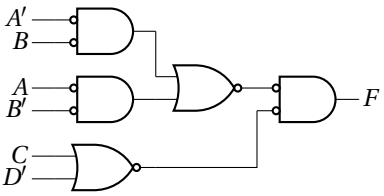


NOR Circuits

- A two-level implementation with NOR gates requires that the function be simplified into product-of-sums form.
- **Procedure:** 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.
- **Remark:** The equivalent AND-OR diagram can be recognised from the NOR diagram by removing all the bubbles.



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



$F(A,B,C,D) = (AB' + A'B)(C+D')$

Outline of This Lecture

NAND and NOR Implementation

Other Two-Level Implementations

Exclusive-OR Function

Summary

Non-degenerate Forms

- 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** (退化形).
    - They degenerate to a single operation.
    - Example: AND in the first level and second level degenerates to an AND of all inputs.
  - The remaining eight non-degenerate 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

AND-OR-INVERT Implementation

- The two forms, NAND–AND and AND–NOR, are equivalent.
- Both perform the AND-OR-INVERT function.
- An AND–OR implementation requires an expression in sum-of-products form.The AND-OR-INVERT implementation is similar, except for the inversion.
  - Therefore, if the complement of the function is simplified into sum-of-products form (by combining the 0's in the map), it will be possible to implement F' with the AND–OR part of the function.

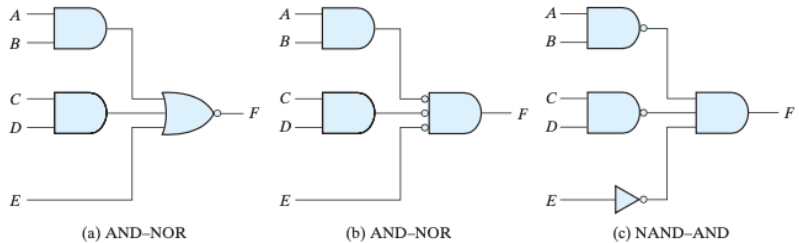


Figure: $F(A,B,C,D,E) = (AB + CD + E)'$. Figure from [1].

OR-AND-INVERT Implementation

- The two forms, `OR-NAND` and `NOR-OR`, are equivalent.
- Both perform the `OR-AND-INVERT` function.
- The `OR-AND-INVERT` implementation requires an expression in product-of-sums form.
  - If the complement of the function is simplified into product-of-sums form, we can implement F' with the `OR-AND` part of the function.

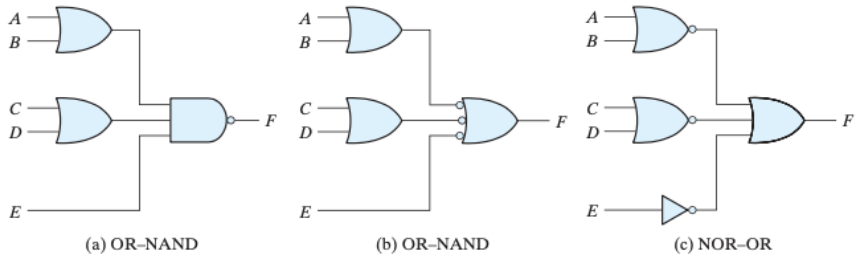


Figure: $F = ((A+B)(C+D)E)'$. Figure from [1].

Outline of This Lecture

NAND and NOR Implementation

Other Two-Level Implementations

Exclusive-OR Function

- XOR
- XOR for Error Detection

Summary

Outline

NAND and NOR Implementation

Other Two-Level Implementations

Exclusive-OR Function

XOR

XOR for Error Detection

Summary

Exclusive-OR Function

- Exclusive-OR (异或门), **XOR**:

$$x \oplus y = xy' + x'y.$$

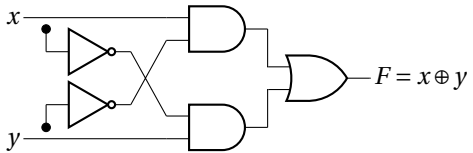
- Exclusive-NOR (同或门), **XNOR** or **equivalency**:

$$(x \oplus y)' = xy + x'y'.$$

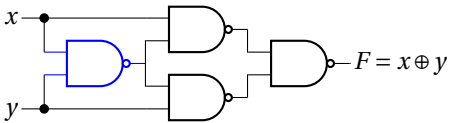
- The following identities apply to the XOR operation:
  - $x \oplus 0 = x$.
  - $x \oplus 1 = x'$.
  - $x \oplus x = 0$.
  - $x \oplus x' = 1$.
  - $x \oplus y' = x' \oplus y = (x \oplus y)'$.

XOR

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



  - Or use NAND gates: $x \oplus y = (x' + y')x + (x' + y')y = (xy)'x + (xy)'y$



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

Outline

NAND and NOR Implementation

Other Two-Level Implementations

Exclusive-OR Function

XOR

XOR for Error Detection

Summary

XOR for Error Detection

- 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.

Odd Function

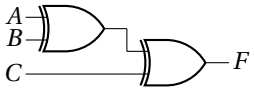
- The XOR operation with three or more variables can be converted into an ordinary Boolean function by replacing the $\oplus$ with its equivalent Boolean expression.

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

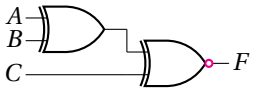
- The three-variable XOR function is equal to 1 if only one variable is equal to 1 or if all three variables are equal to 1.
  - An odd number of variables are equal to 1.
  - **Odd function**.

Odd Function and Even Function

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



- **Even function** can also be implemented:



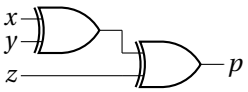
Parity Generation and Checking

- `XOR` functions are very useful in systems requiring error detection and correction codes.
  - Recap: 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

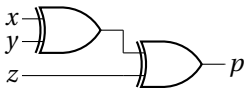


3-bit even parity generator

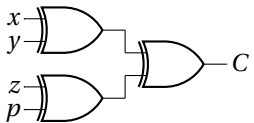
- p constitutes an odd function.

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.



3-bit even parity generator



4-bit even parity checker

Outline of This Lecture

NAND and NOR Implementation

Other Two-Level Implementations

Exclusive-OR Function

Summary

Summary

- Today's lecture
  - Two- and multi-level implementations:
    - Universal gates: `NAND` and `NOR` gates.
    - Procedure of two-level implementations using `NAND` or `NOR` gates.
    - Procedure of multi-level implementations using `NAND` or `NOR` gates.
  - `Exclusive-OR` gate for constructing error detection circuits.
- Today's lab: Verilog (loop, primitive, structured design)
- Next lecture (Oct 9th, Sunday): Combinational Logic.

Essential Reading

- Essential reading for this lecture: pages 91-118 of the textbook.
- Essential reading for next lecture: pages 125-148 of the textbook.

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