浙江大學
ZHEJIANG UNIVERSITY

Digital Logic Design

Chapter 3 – Combinational Logic Design

Implementation Technology and Logic Design

浙江大学计算机学院　王总辉
zhwang@zju.edu.cn
http://course.zju.edu.cn
2023年10月

Overview

- **Part 1 – Design Procedure**
  - Steps
    - Specification
    - Formulation
    - Optimization
    - Technology Mapping
  - Beginning Hierarchical Design
  - Technology Mapping - AND, OR, and NOT to NAND or NOR
  - Verification
    - Manual
    - Simulation

Overview (continued)

- **Part 2 – Combinational Logic**
  - Functions and functional blocks
  - Rudimentary logic functions
  - Decoding using Decoders
    - Implementing Combinational Functions with Decoders
  - Encoding using Encoders
  - Selecting using Multiplexers
    - Implementing Combinational Functions with Multiplexers

learning demands:

- Understanding the characteristics of combinational logic circuits
- Skilled to master the basic method of combination circuit analysis and design of the basic method
- Understanding gate properties and main technological parameters
- Understanding the programmable chip structure Mastered programmable implement technology

Combinational Circuits

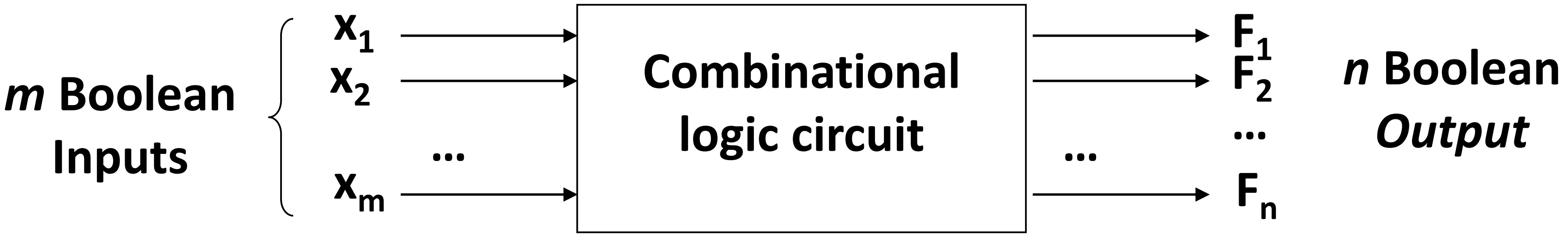
- **Logic Circuits**

major categories: Combinational Logic Circuits / Sequential Logic Circuits

Combinational circuit is defined:
--If a stable **output value** of logic circuit, at any given moment, the value depends only on set of the **input values** of the moment, regardless of the past input value, then the circuit is a combinational logic circuit.

Combinational circuit characteristics

- **A combinational logic circuit has:**
  - A set of *m* Boolean inputs,
  - A set of *n* Boolean outputs, and
  - *n* switching functions, each mapping the 2^m input combinations to an output such that the current output depends only on the current input values
- **A block diagram:**



$F_i=f_i(x_1, x_2, \ldots x_m), \quad i=1, 2, \ldots, n$

（ **without any memory element and feedback loop.**
input signal unidirectional transmission ）

Combinational circuit analysis method

1. To write logic function according to circuit

—From input to the output, level by level, to write expression of each door

—Then, write a logical expression of the entire circuit, according to the logical relationship among each door

—simplifying function expression

2. Lists the truth table

—analyzing the variables causal relationship and logic functions between input and output

3. Event functional analysis

—assigning variable to the actual meaning, sum actual functionality up the circuit

—To draw logic waveform diagram of input and output

4. Evaluation or improvement

—Evaluate of the design rationality

Analyzes circuits logic functions

List the expression level by level

$$P_1 = A \oplus B$$

$$P_2 = B \oplus C$$

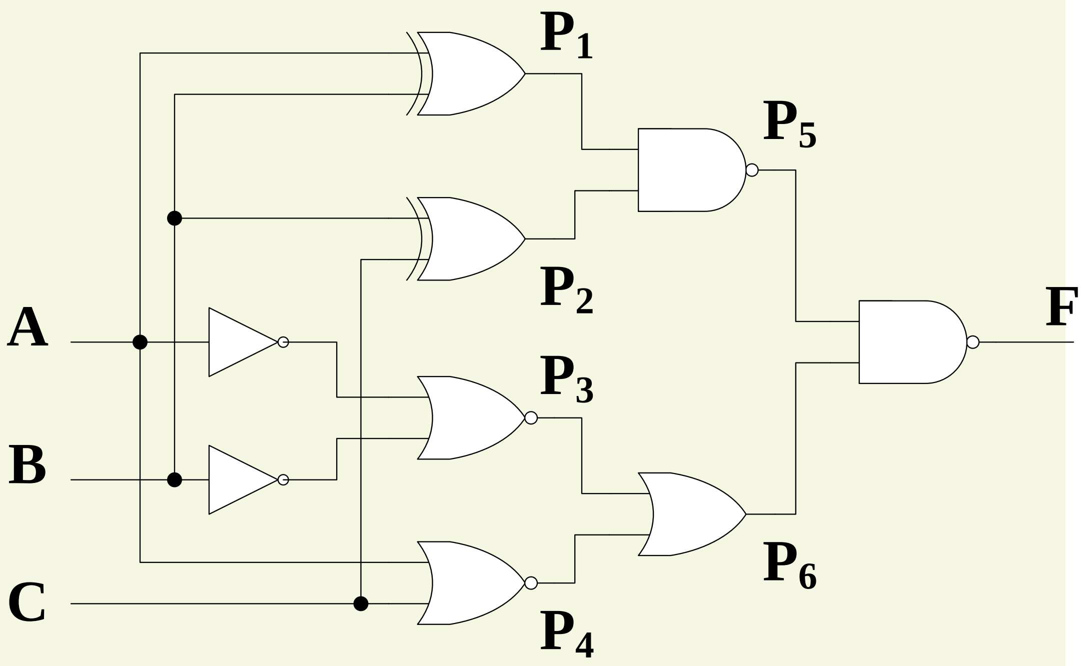
$$P_3 = \overline{\overline{A} + \overline{B}} = AB$$

$$P_4 = \overline{A + C}$$

$$P_5 = \overline{P_1 P_2} = \overline{(A \oplus B) \cdot (B \oplus C)}$$

$$P_6 = P_3 + P_4 = AB + \overline{A + C}$$

$$F = \overline{P_5 \cdot P_6} = \overline{\overline{(A \oplus B) \cdot (B \oplus C)} \cdot (AB + \overline{A + C})}$$



Logic circuit diagram

simplifying function expression

$$F = (A \oplus B)(B \oplus C) + \overline{AB + \overline{A + C}}$$

$$= (A\overline{B} + \overline{A}B)(B\overline{C} + \overline{B}C) + (\overline{A} + \overline{B})(A + C)$$

$$= A\overline{B}C + \overline{A}B\overline{C} + \overline{A}C + A\overline{B} + \overline{B}C$$

$$= \overline{A}B\overline{C} + \overline{A}C + A\overline{B} + \overline{B}C$$

$$= \overline{A}C + A\overline{B} + \overline{A}B\overline{C}$$

$$= \overline{A}(C + B\overline{C}) + A\overline{B}$$

$$= \overline{A}C + \overline{A}B + A\overline{B}$$

$$= \overline{A}C + (A \oplus B)$$

Lists the truth table

A	B	C	$A \oplus B$	$\overline{A}C$	F
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	1	0	1
0	1	1	1	1	1
1	0	0	1	0	1
1	0	1	1	0	1
1	1	0	0	0	0
1	1	1	0	0	0

simplifying function expression

$F = ($

$= (A$ $+C)$

$= A\bar{B}$

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

Functional Analysis :

Seen from the truth table, F = 1, the condition is A ≠ B, or B <C, **it's a condition determination circuit**

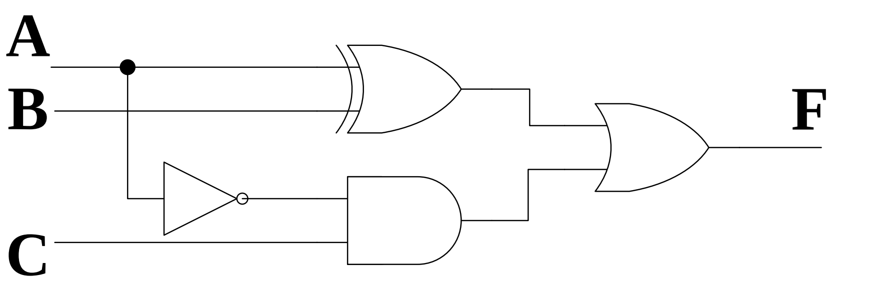
Evaluation:

The original design is not the best, replaced by the following simplest circuit:

$= \bar{A}C + (A \oplus B)$

Lists the truth table

A	B	C	$A \oplus B$	$\bar{A}C$	F
0	0	0	0	0	0
0	0	1	0	1	1
					1
					1
					1
					1
1	1	0	0	0	0
1	1	1	0	0	0



Design Procedure

1. **Specification (Logical abstraction)**
   - Write a specification for the circuit if one is not already available
2. **Formulation(Logical abstraction to write logic expression)**
   - Derive a truth table or initial Boolean equations that define the required relationships between the inputs and outputs, if not in the specification
   - Apply hierarchical design if appropriate

Design Procedure (continued)

3. **Optimization(The simplification)**
   - Apply 2-level and multiple-level optimization
   - Draw a logic diagram or provide a netlist for the resulting circuit using ANDs, ORs, and inverters
4. **Technology Mapping(Converted into the suitable expression)**
   - Map the logic diagram or net-list to the implementation technology selected
5. **Verification**
   - Verify the correctness of the final design manually or using simulation

Combinational circuit design method

1. **Logical abstraction**

   1） Analyzes Causality of events, Determine:

      Input variables –-Independent variables

      Output variable –- Function

   2） Defining meaning of the logical variables (state):

      assigning values to variables

   3） Lists the Truth table according to the Causality

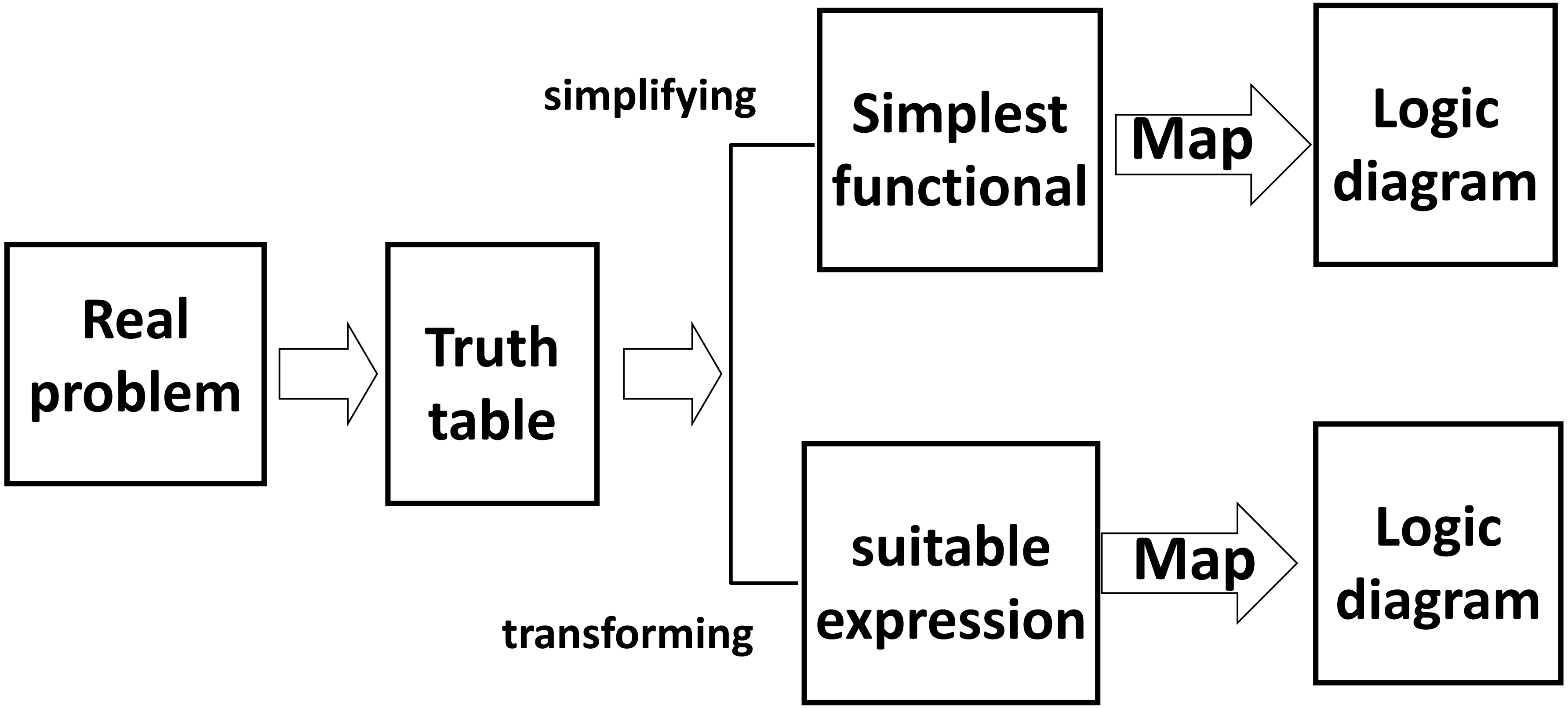
2． **To write logic expression (Minterm)**

3． **The simplification**—K-map

4． **Technology Mapping**

   Converted into the suitable expression, and draw the logical connection diagram

5. **Designing circuit module with HDL description**


simplifying
Real problem
Truth table
Simplest functional
Map
Logic diagram
suitable expression
Map
Logic diagram
transforming

Design Example 1

Only one lamp in the dormitory, but there are three beds. Now at the side of each bed install a switch that can independently control the light. Please design this circuit with the least gates.

Step 1: Logical abstraction

Analyzes Causality :

Switch as Input variables : S_1、S_2、S_3

Control the light as Output variable: F

Assigning Switch : Pressed is "1", pounce to "0"

Assigning Output: Brighten is "1", dimmed to "0"

Step 2: Derive logic expression

$$F = \overline{S}_3\overline{S}_2S_1 + \overline{S}_3S_2\overline{S}_1 + S_3\overline{S}_2\overline{S}_1 + S_3S_2S_1$$

Truth table

S_3	S_2	S_1	F
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

Design Example

$$F = \overline{S}_3\overline{S}_2 S_1 + \overline{S}_3 S_2 \overline{S}_1 + S_3 \overline{S}_2 \overline{S}_1 + S_3 S_2 S_1$$

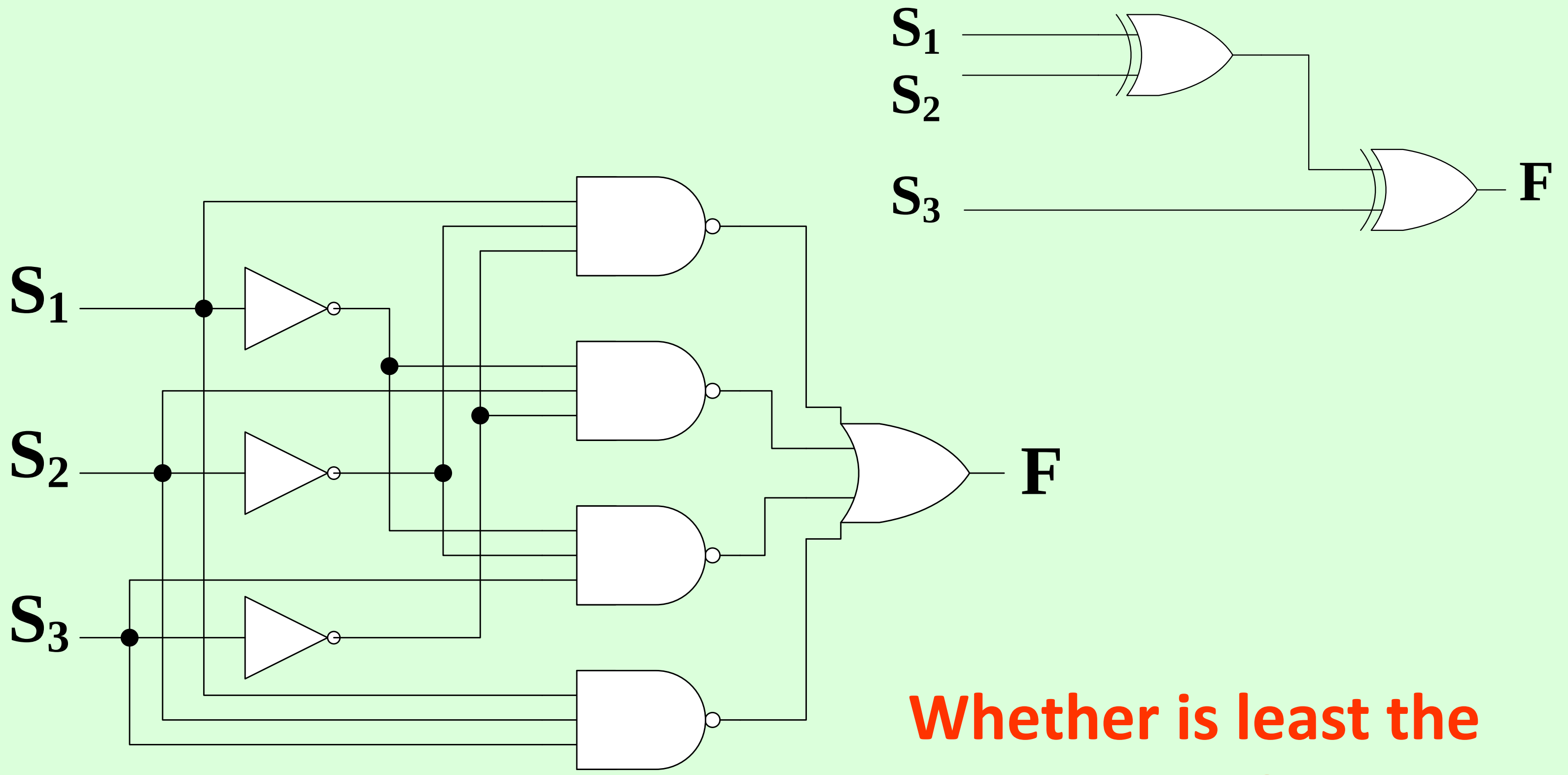
S_3	S_2	S_1	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Step 3： simplifying

S_3 \ $S_2 S_1$	00	01	11	10
0		1		1
1	1		1	

Has been the most simple

Step 4： Logical connection diagram



Whether is least the gates?

Step 5: Designing circuit module with HDL description

$$F = \overline{S}_3\overline{S}_2S_1 + \overline{S}_3S_2\overline{S}_1 + S_3\overline{S}_2\overline{S}_1 + S_3S_2S_1$$

```
module lamp_control(s1,s2,s3,F );
    input   s1,s2,s3;
    output  F;
    wire    s1,s2,s3,f;

    assign  F= (~s3&~s2&s1) | (~s3&s2&~s1) | (s3&s2&s1) ;

endmouule
```

Beginning Hierarchical Design

- **To control the complexity of the function mapping inputs to outputs:**
  - Decompose the function into smaller pieces called *blocks*
  - Decompose each block's function into smaller blocks, repeating as necessary until all blocks are small enough
  - Any block not decomposed is called a *primitive block*
  - The collection of all blocks including the decomposed ones is a *hierarchy*

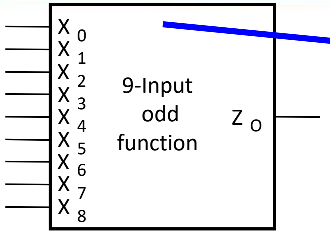
Module definition

```
module lamp_control(s1,s2,s3,F );
        input   s1,s2,s3;
        output  F;
```

- **Example: 9-input parity tree (see next slide)**

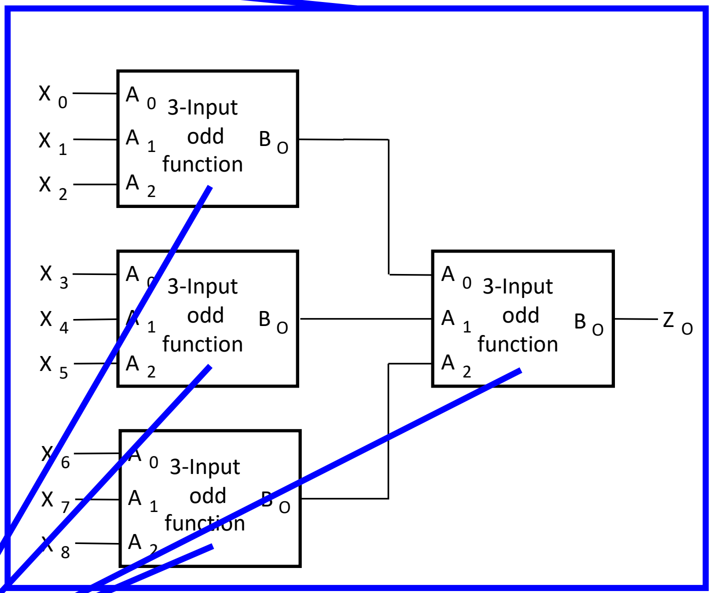
Hierarchy for Parity Tree Example

Top Level: 9 inputs, one output



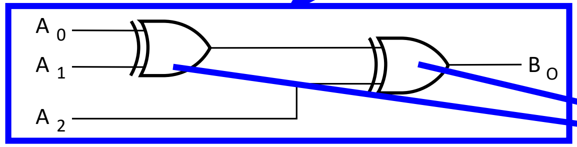
(a) Symbol for circuit

2nd Level: Four 3-bit odd parity trees in two levels



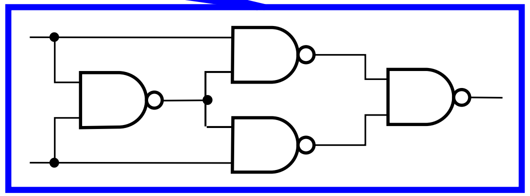
(b) Circuit as interconnected 3-input odd function blocks

3rd Level: Two 2-bit exclusive-OR functions



(c) 3-input odd function circuit as interconnected exclusive-OR blocks

Primitives: Four 2-input NAND gates



(d) Exclusive-OR block as interconnected NANDs

Design requires 4 X 2 X 4 = 32 2-input NAND gates

Reusable Functions

- **Whenever possible, we try to decompose a complex design into common, *reusable* function blocks**

- **These blocks are**
  - verified and well-documented
  - placed in libraries for future use

Top-Down versus Bottom-Up

- A ***top-down design*** proceeds from an abstract, high-level specification to a more and more detailed design by decomposition and successive refinement ***What do we build?***
- A ***bottom-up design*** starts with detailed primitive blocks and combines them into larger and more complex functional blocks ***What we use to build?***
- Design usually proceeds **top-down** to known building blocks ranging from **complete** CPUs **to primitive** logic gates or electronic components.
- Much of the material in this chapter is devoted to learning about combinational blocks used in top-down design.

Design Example 2

1. **Specification**
   - BCD to Excess-3 code converter
   - Transforms BCD code for the decimal digits to Excess-3 code for the decimal digits
   - BCD code words for digits 0 through 9: 4-bit patterns 0000 to 1001, respectively
   - Excess-3 code words for digits 0 through 9: 4-bit patterns consisting of 3 (binary 0011) added to each BCD code word
   - Implementation:
     - multiple-level circuit
     - NAND gates (including inverters)

Design Example (continued)

2. Formulation

- Conversion of 4-bit codes can be most easily formulated by a truth table
- Variables
  - <u>BCD</u>:
    A,B,C,D
- Variables
  - <u>Excess-3</u>
    W,X,Y,Z
- Don't Cares
  - BCD 1010
    to 1111

Input BCD A B C D	Output Excess-3 WXYZ
0 0 0 0	0 0 1 1
0 0 0 1	0 1 0 0
0 0 1 0	0 1 0 1
0 0 1 1	0 1 1 0
0 1 0 0	0 1 1 1
0 1 0 1	1 0 0 0
0 1 1 0	1 0 0 1
0 1 1 1	1 0 1 0
1 0 0 0	1 0 1 1
1 0 0 1	1 0 1 1 1100

Design Example (continued)

3. **Optimization**

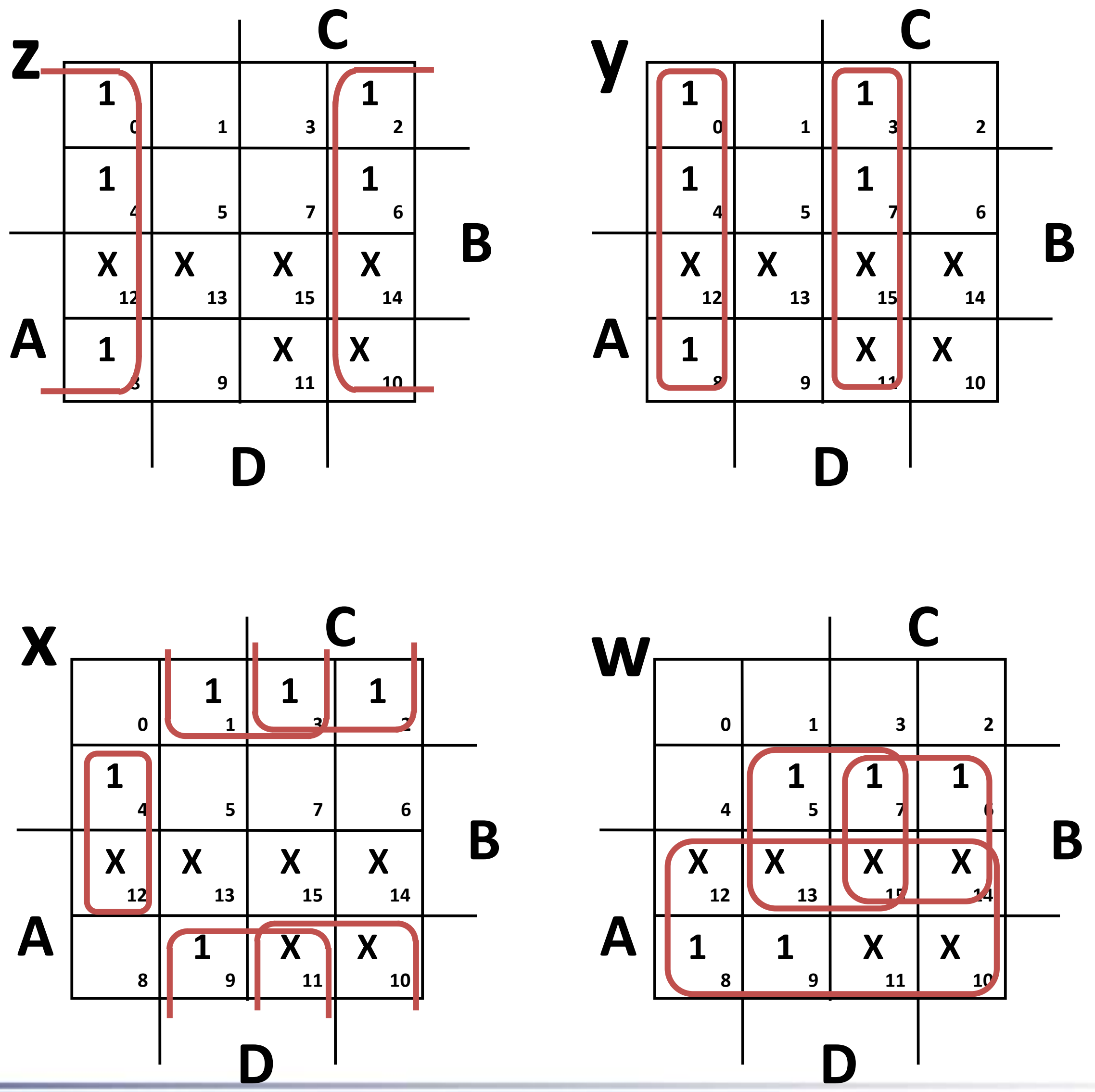
   a. 2-level using K-maps

$$W = A + BC + BD$$

$$X = \overline{B}C + \overline{B}D + B\overline{C}\,\overline{D}$$

$$Y = CD + \overline{C}\,\overline{D}$$

$$Z = \overline{D}$$



Design Example (continued)

3. Optimization (continued)

b. Multiple-level using transformations

$$
\begin{aligned}
W &= A + BC + BD \\
X &= \overline{B}C + \overline{B}D + B\overline{C}\,\overline{D} \\
Y &= CD + \overline{C}\,\overline{D} \\
Z &= \overline{D}
\end{aligned}
\qquad G = 7 + 10 + 6 + 0 = 23
$$

- Perform extraction, finding factor: $T_1 = C + D$

$$
\begin{aligned}
T_1 &= C + D \\
W &= A + BT_1 \\
X &= \overline{B}T_1 + B\overline{C}\,\overline{D} \\
Y &= CD + \overline{C}\,\overline{D} \\
Z &= \overline{D}
\end{aligned}
$$

Design Example (continued)

3. **Optimization (continued)**

   b. Multiple-level using transformations

$$T_1 = C + D$$
$$W = A + BT_1$$
$$X = \overline{B}T_1 + B\overline{C}\,\overline{D}$$
$$Y = CD + \overline{C}\,\overline{D}$$
$$Z = \overline{D}$$

G = 2+4+7+6+0=19

- An additional extraction not shown in the text since it uses a Boolean transformation:

$$\overline{T}_1 = \overline{C + D} = \overline{C}\,\overline{D}$$

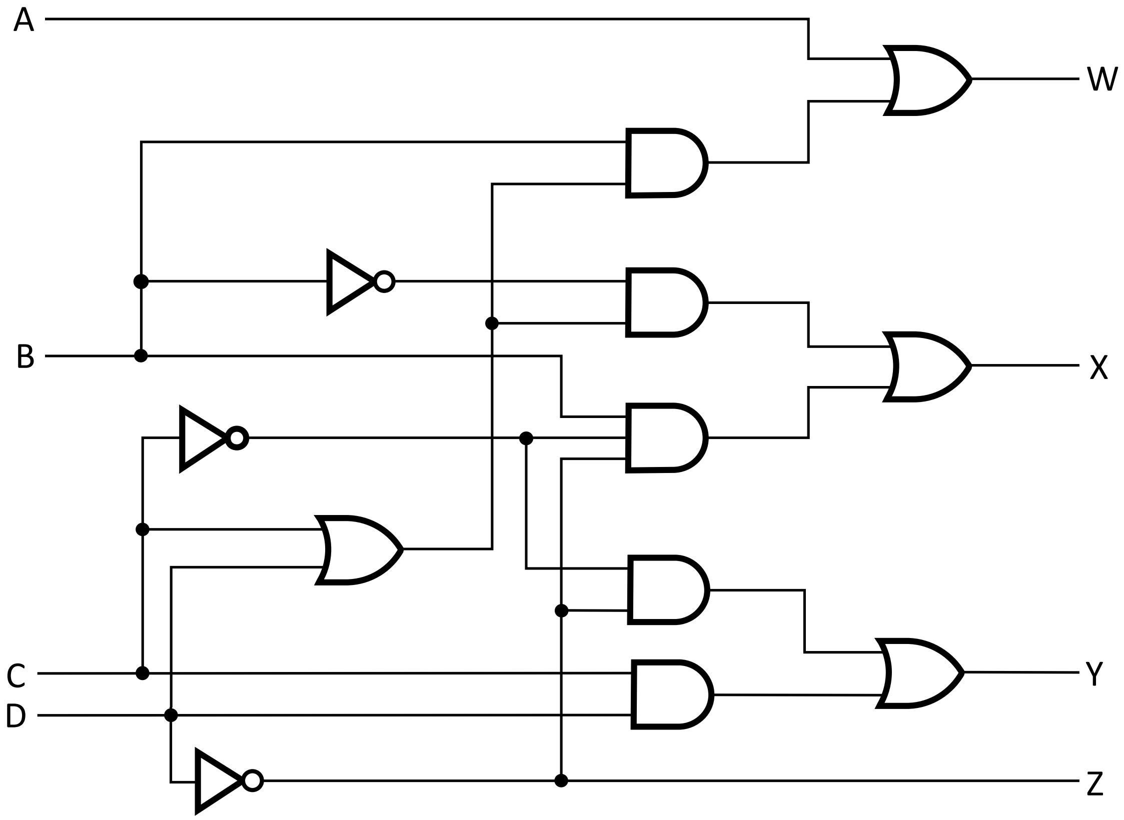
$$W = A + BT_1$$
$$X = \overline{B}T_1 + B\overline{T}_1$$
$$Y = CD + \overline{T}_1$$
$$Z = \overline{D}$$

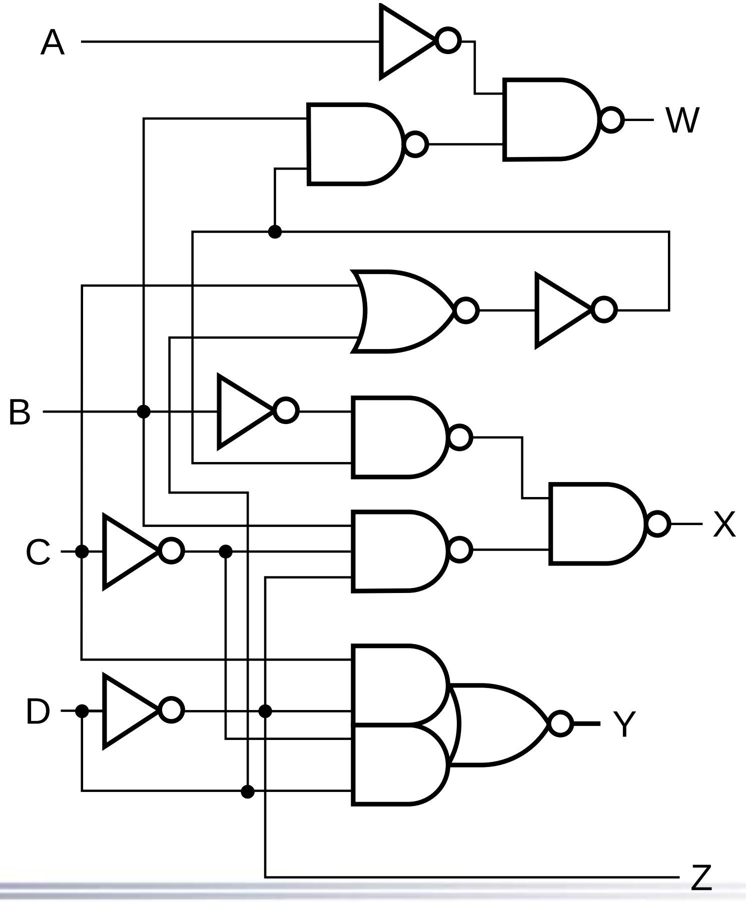
G = 2 +1 + 4 + 6 + 4 + 0 = 17!

Design Example (continued)

4. Technology Mapping

- Mapping with a library containing inverters and 2-input NAND, 2-input NOR, and 2-2 AOI gates





5. Hardware Description module design

```
module BCD_ Excess_3( A,B,C,D,W,X,Y,Z );
    input   A,B,C,D;
    output  W,X,Y,Z;
    wire A,B,C,D , W,X,Y,Z,T1;

    assign  T1= C|D;
    assign  W= A|B&C|B&D ;
    assign  X = ~B&T1|B&(~C&~D);
    assign  Y = C&D|~C&~D;
    assign  Z= ~D ;

endmodule
```

Technology Mapping

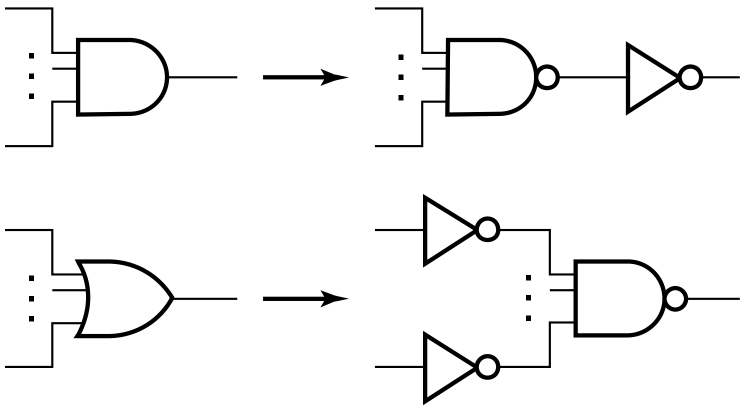
- **Mapping Procedures**
  - To NAND gates
  - To NOR gates
  - Mapping to multiple types of logic blocks in covered in the reading supplement: Advanced Technology Mapping.

Mapping to NAND gates

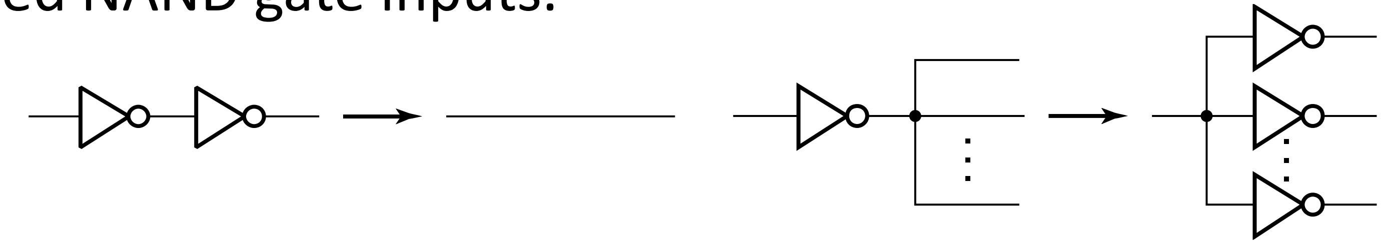
- **Assumptions:**
  - Gate loading and delay are ignored
  - Cell library contains an inverter and *n*-input NAND gates, *n* = 2, 3, …
  - An AND, OR, inverter schematic for the circuit is available
- **The mapping is accomplished by:**
  - Replacing AND and OR symbols,
  - Pushing inverters through circuit fan-out points, and
  - Canceling inverter pairs

NAND Mapping Algorithm

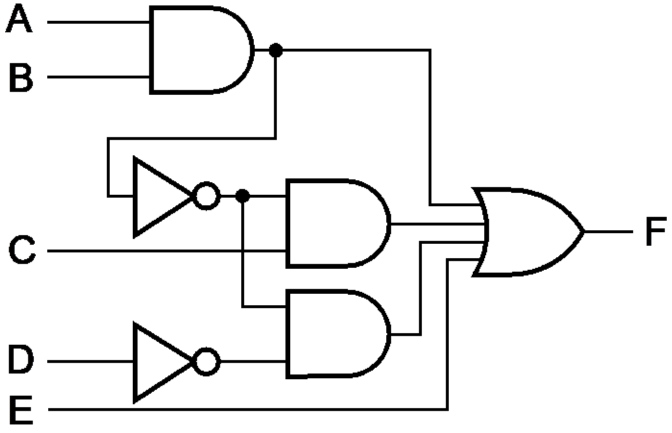
1. **Replace ANDs and ORs:**



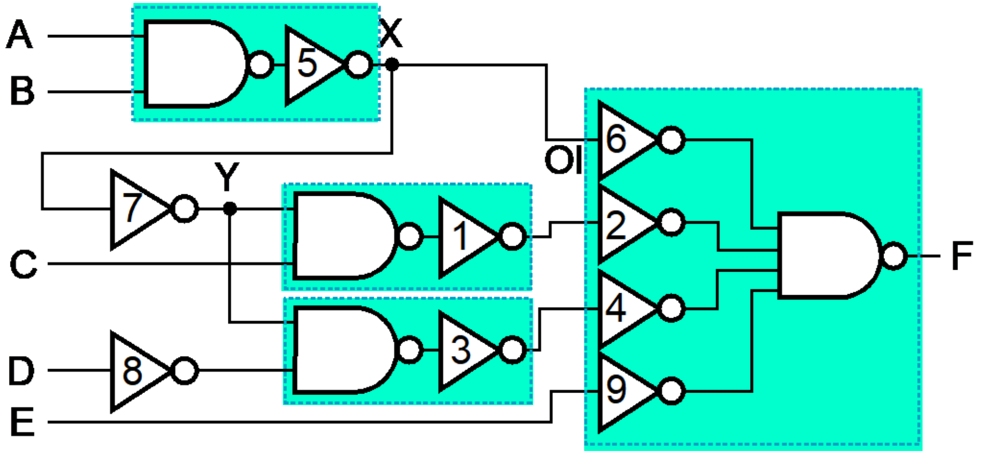
2. **Repeat the following pair of actions until there is at most one inverter between :**
   a. A circuit input or driving NAND gate output, and
   b. The attached NAND gate inputs.



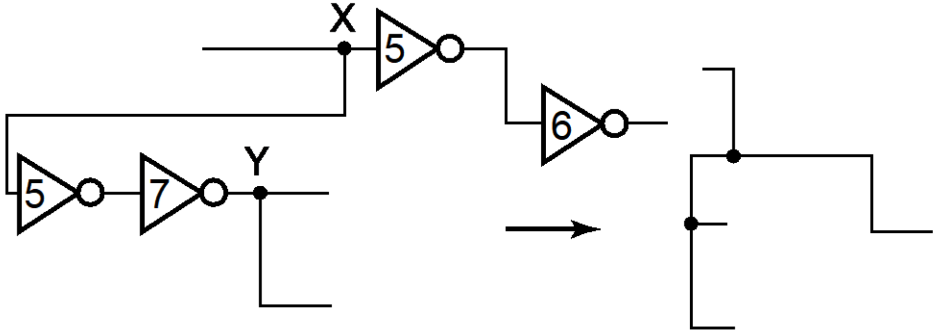
NAND Mapping Example



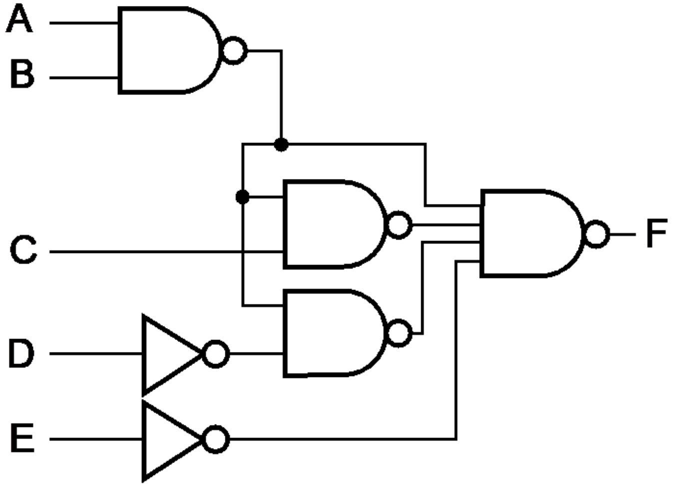
(a)



(b)



(c)



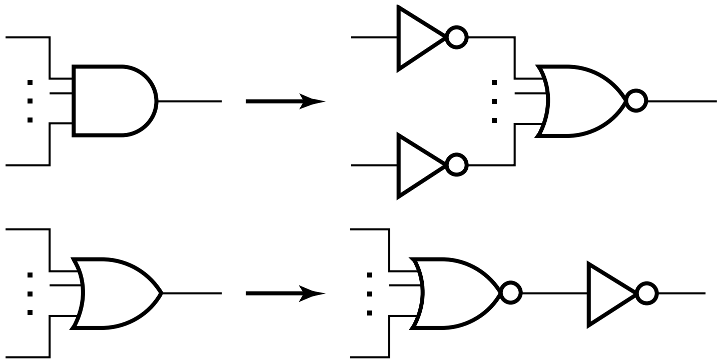
(d)

Mapping to NOR gates

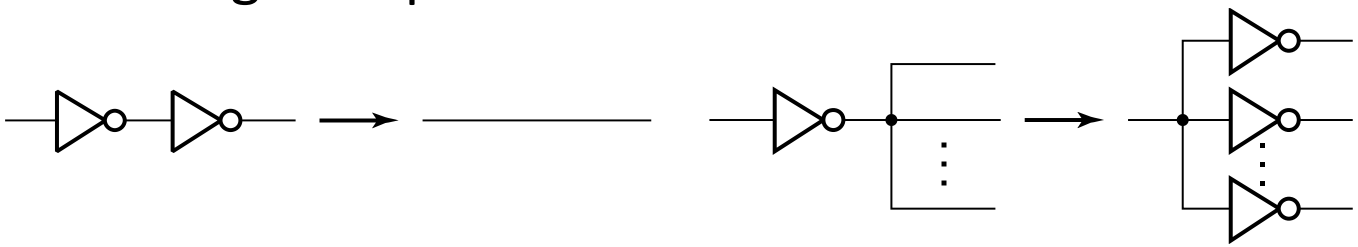
- **Assumptions:**
  - Gate loading and delay are ignored
  - Cell library contains an inverter and n-input NOR gates, n = 2, 3, …
  - An AND, OR, inverter schematic for the circuit is available
- **The mapping is accomplished by:**
  - Replacing AND and OR symbols,
  - Pushing inverters through circuit fan-out points, and
  - Canceling inverter pairs

NOR Mapping Algorithm

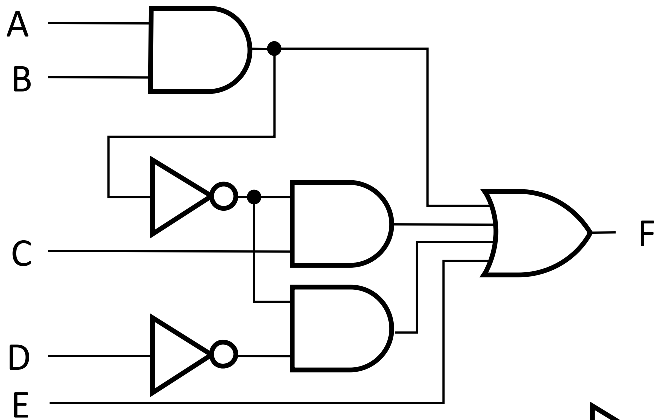
1. **Replace ANDs and ORs:**



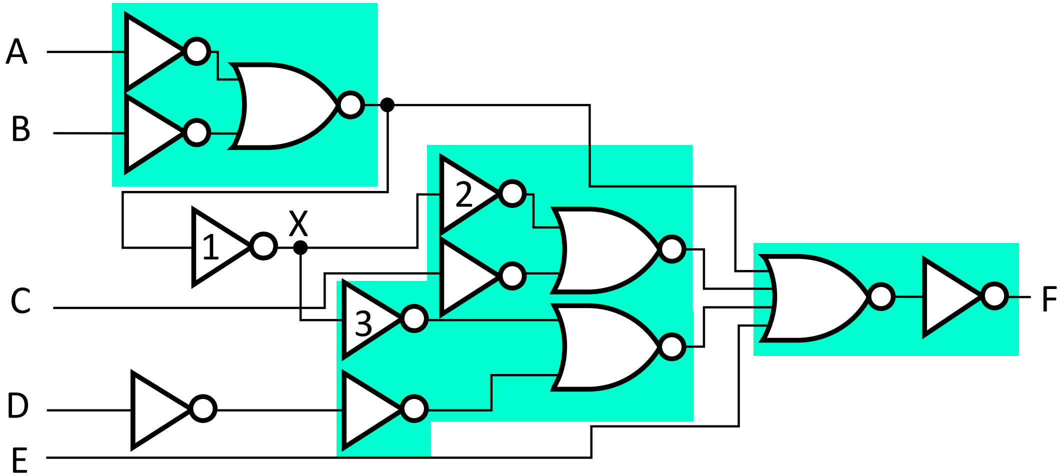
2. **Repeat the following pair of actions until there is at most one inverter between :**
   a. A circuit input or driving NAND gate output, and
   b. The attached NAND gate inputs.



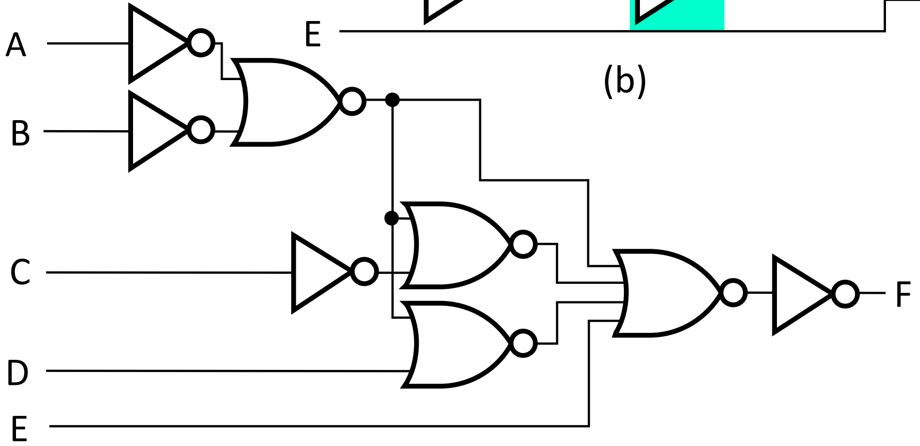
NOR Mapping Example



(a)



(b)



(c)

Verification

- **Verification - show that the final circuit designed implements the original specification**
- **Simple specifications are:**
  - truth tables
  - Boolean equations
  - HDL code
- **If the above result from <u>formulation</u> and are not the <u>original specification</u>, it is critical that the formulation process be flawless for the verification to be valid!**

Basic Verification Methods

- **Manual Logic Analysis**
  - Find the truth table or Boolean equations for the final circuit
  - Compare the final circuit truth table with the specified truth table, or
  - Show that the Boolean equations for the final circuit are equal to the specified Boolean equations
- **Simulation**
  - Simulate the final circuit (or its netlist, possibly written as an HDL) and the specified truth table, equations, or HDL description using test input values that fully validate correctness.
  - The obvious test for a combinational circuit is application of all possible "care" input combinations from the specification

Verification Example: Manual Analysis

- **BCD-to-Excess 3 Code Converter**
  - Find the SOP Boolean equations from the final circuit.
  - Find the truth table from these equations
  - Compare to the formulation truth table
- **Finding the Boolean Equations:**

$$T_1 = \overline{\overline{C + D}} = C + D$$

$$W = \overline{\overline{A}\,\overline{(T_1 B)}} = A + BT_1$$

$$X = \overline{\overline{(T_1\overline{B})}\,\overline{(B\overline{C}\,\overline{D})}} = \overline{B}T_1 + B\overline{C}\,\overline{D}$$

$$Y = \overline{C\overline{D} + \overline{C}D} = CD + \overline{C}\,\overline{D}$$

$$Z = \overline{D}$$

Verification Example: Manual Analysis

- **Find the circuit truth table from the equations and compare to specification truth table:**

Input BCD A B C D	Output Excess -3 WXYZ
0 0 0 0	0 0 1 1
0 0 0 1	0 1 0 0
0 0 1 0	0 1 0 1
0 0 1 1	0 1 1 0
0 1 0 0	0 1 1 1
0 1 0 1	1 0 0 0
0 1 1 0	1 0 0 1
0 1 1 1	1 0 1 0
1 0 0 0	1 0 1 1
1 0 0 1	1 0 1 1

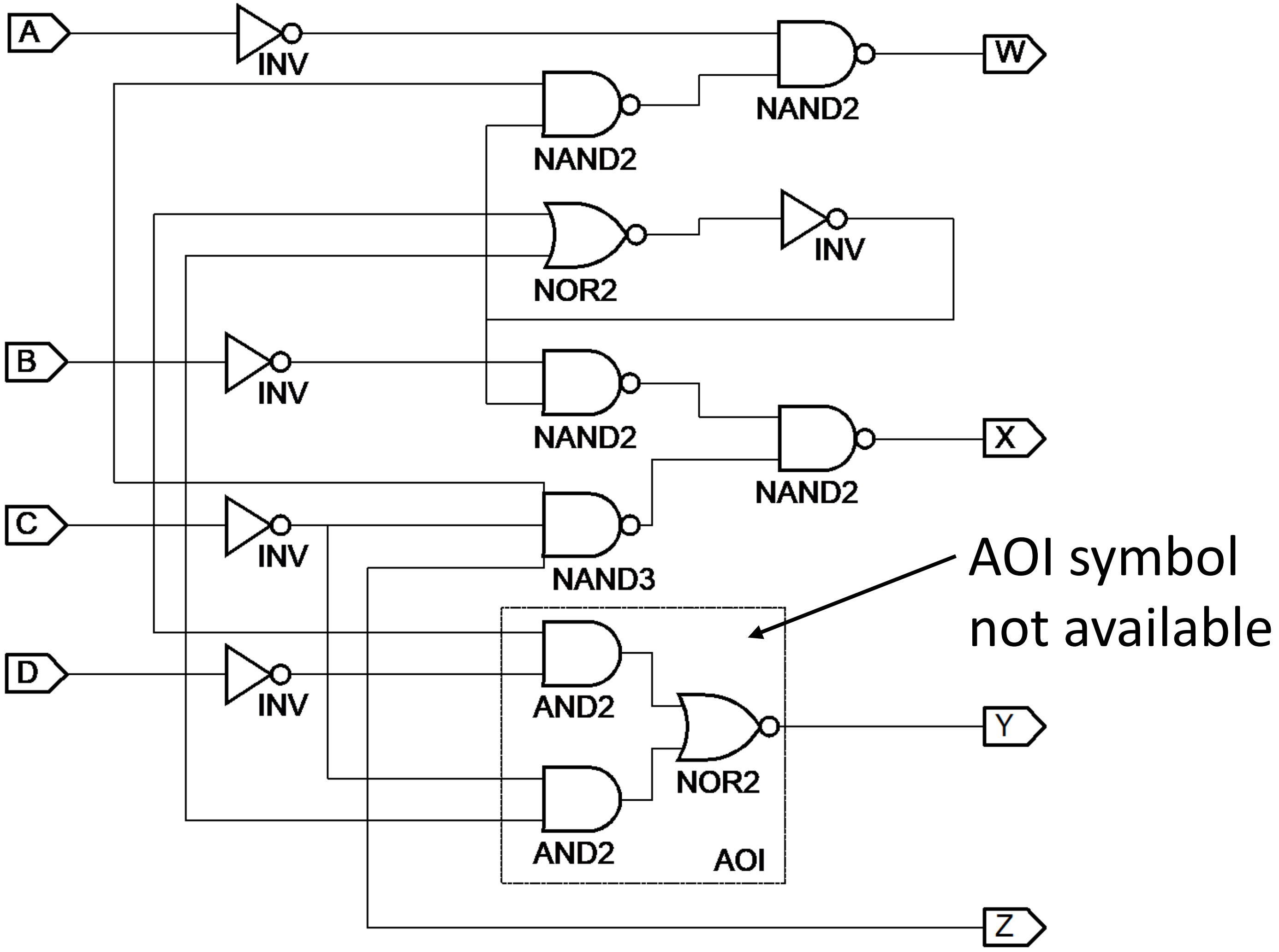
The tables match!

Verification Example: Simulation

- **Simulation procedure:**
  - Use a schematic editor or text editor to enter a gate level representation of the final circuit
  - Use a waveform editor or text editor to enter a test consisting of a sequence of input combinations to be applied to the circuit
    - This test should guarantee the correctness of the circuit if the simulated responses to it are correct
    - Short of applying all possible “care” input combinations, generation of such a test can be difficult

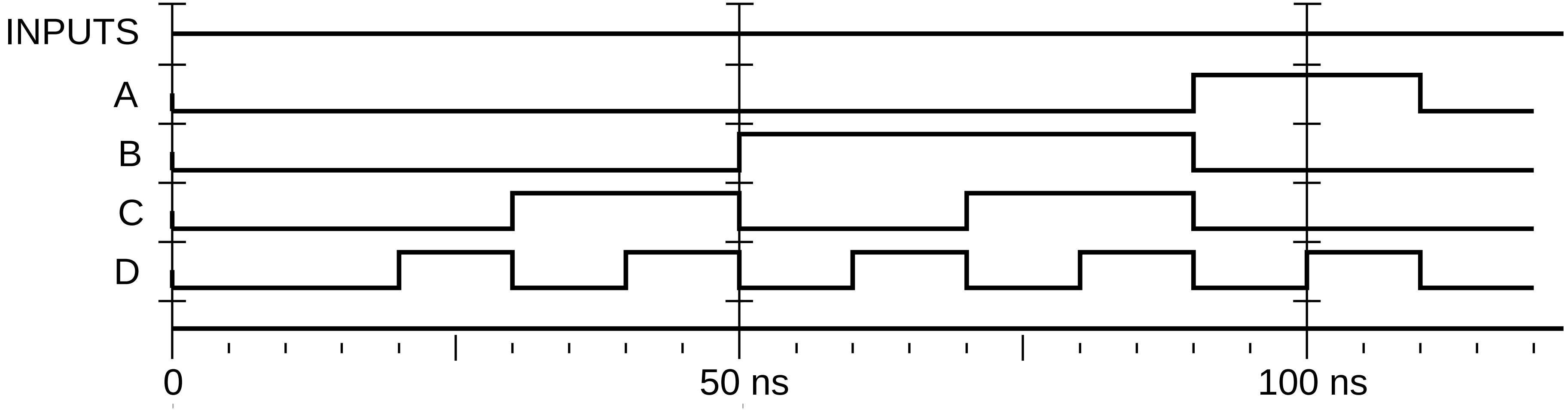
Verification Example: Simulation

- **Enter BCD-to-Excess-3 Code Converter Circuit Schematic**



Verification Example: Simulation

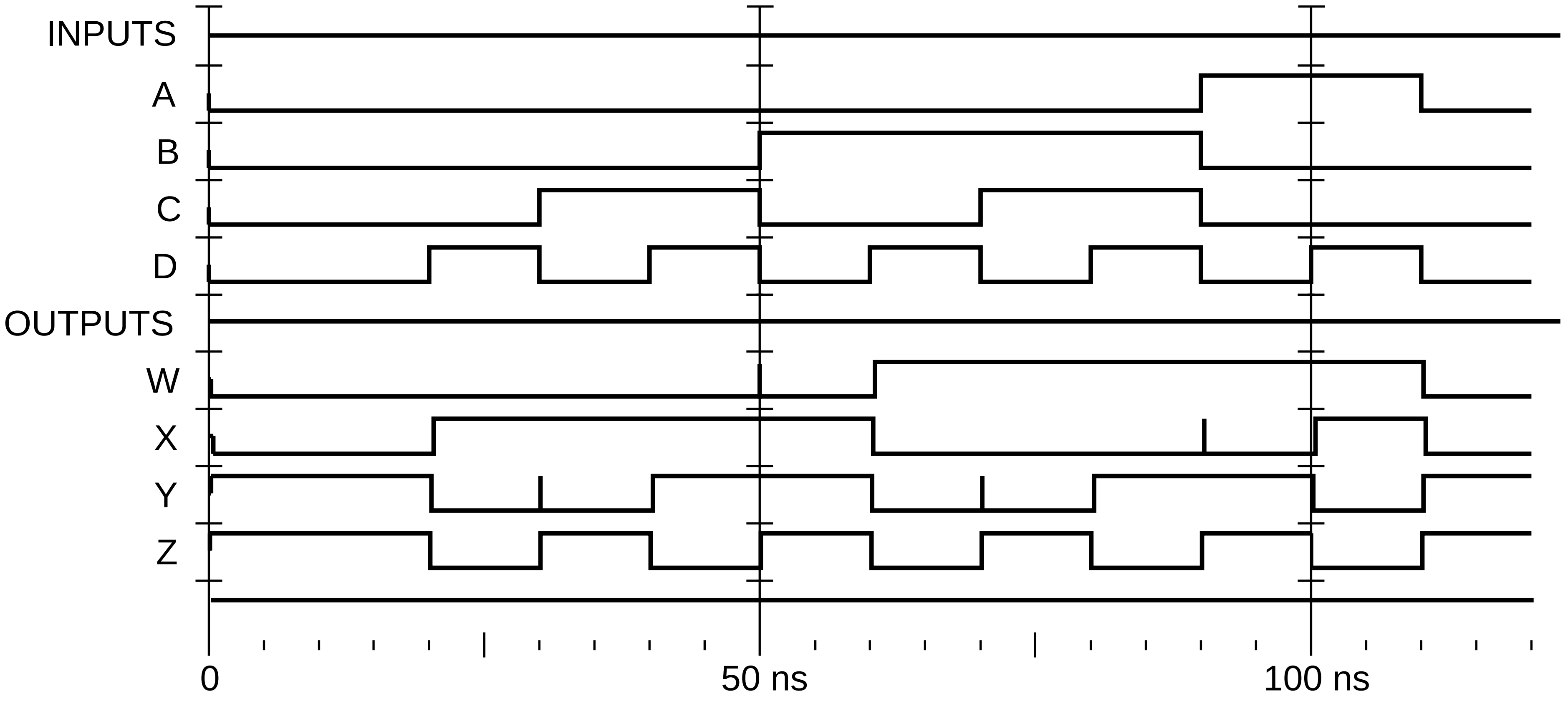
- **Enter waveform that applies all possible input combinations:**



- **Are all BCD input combinations present? (Low is a 0 and high is a one)**

Verification Example: Simulation

- **Run the simulation of the circuit for 120 ns**



- **Do the simulation output combinations match the original truth table?**

□谢 谢！