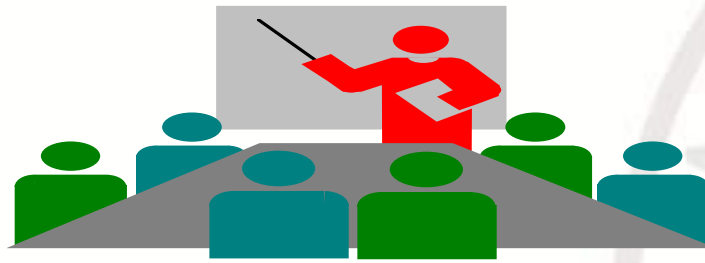




浙江大学
ZHEJIANG UNIVERSITY



数字逻辑设计

LOGIC and Computer Design Fundamentals

CHAPTER 3

Combinational Logic Design

Logic Design and Implementation Technology

施青松

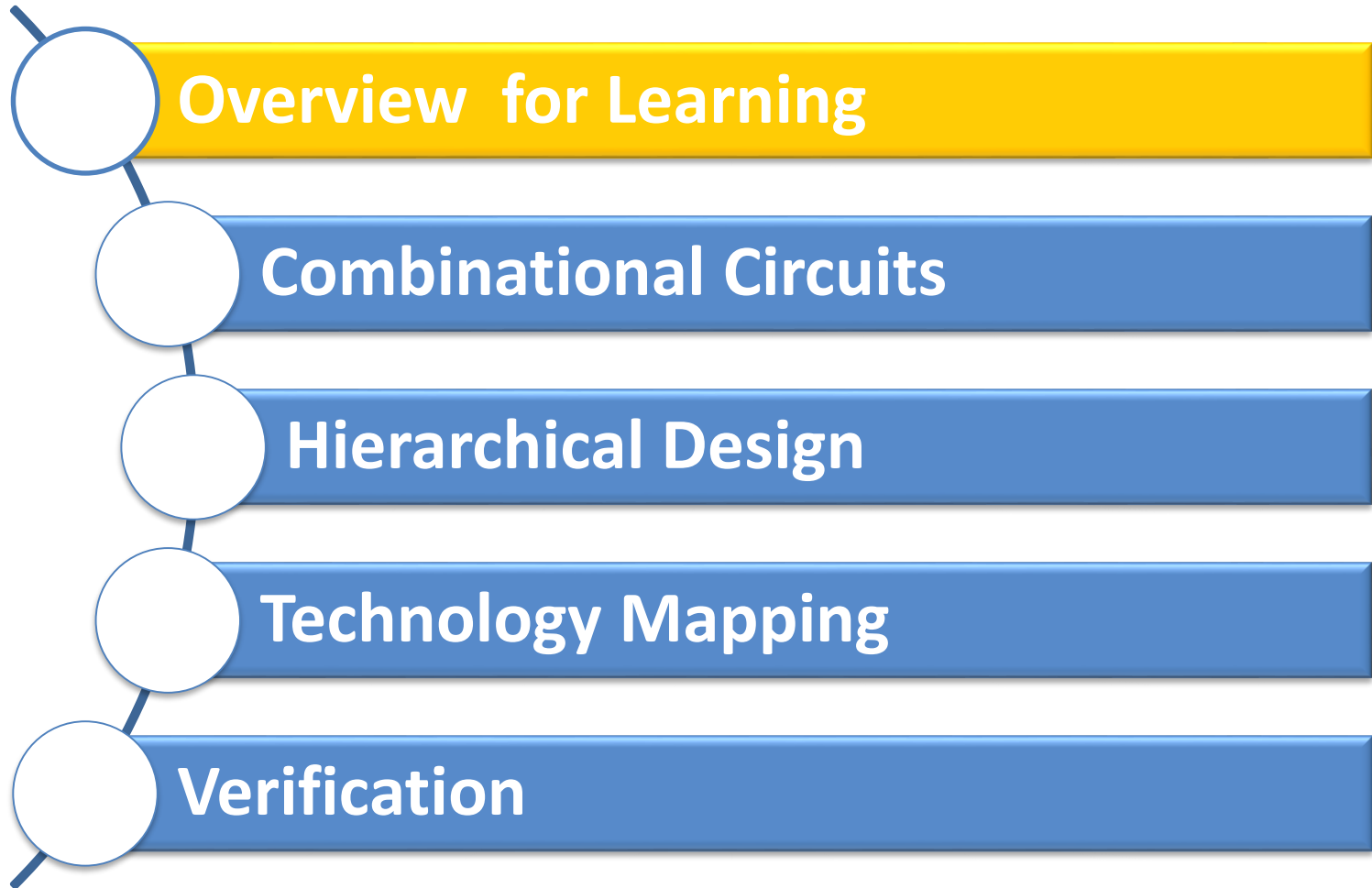
Asso. Prof. Shi Qingsong

College of Computer Science and Technology, Zhejiang University

zjsqs@zju.edu.cn



Course Outline





learning demands:

- Understanding the characteristics of combinational logic circuits
- **Skilled to master** 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

□ 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



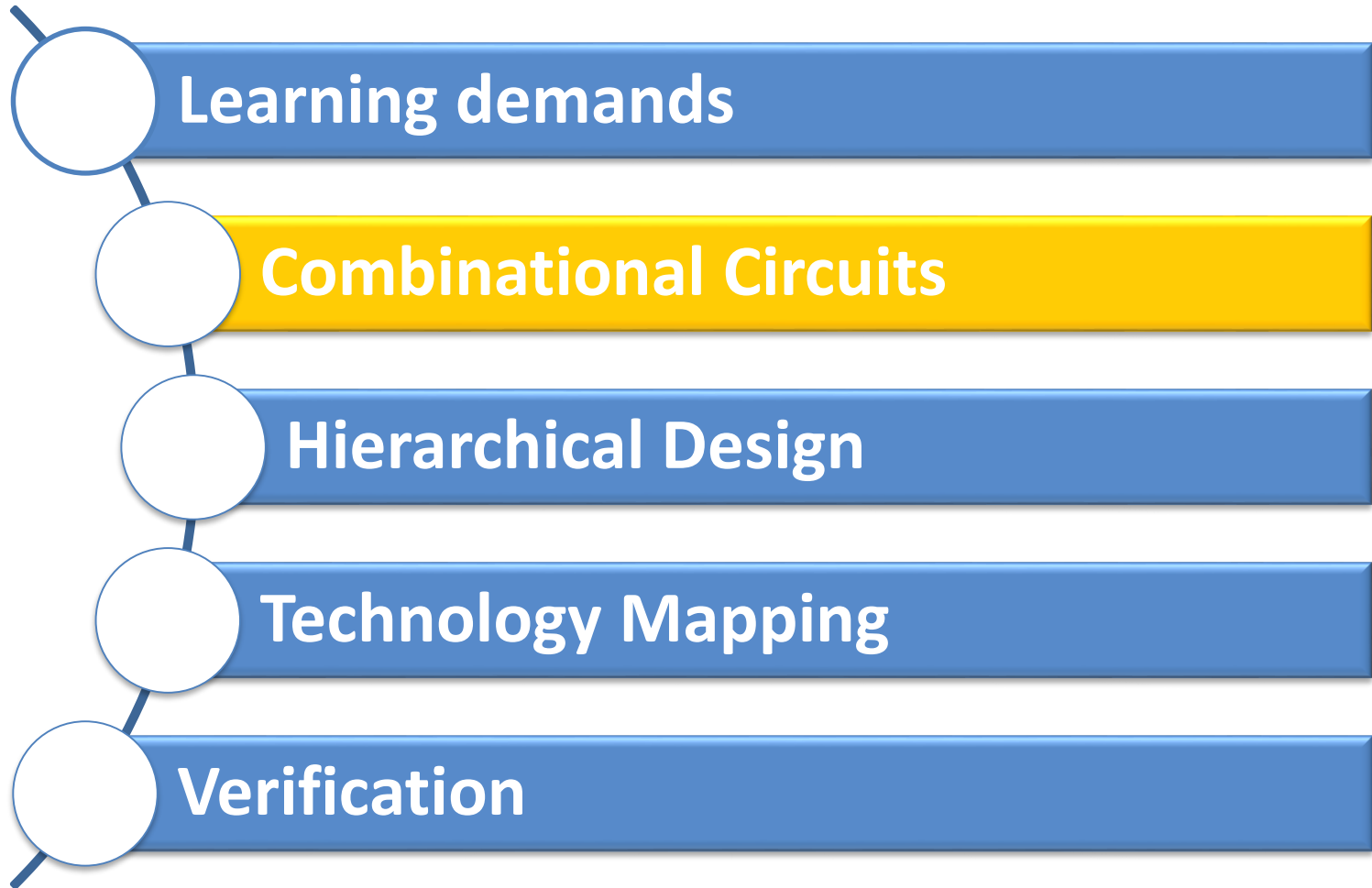
Overview(continued)

□ Part 2 – Commonly Function Design

- Functions and functional blocks
- Rudimentary logic functions
- Commonly Logic Function Design
 - Decoding using **Decoders**
 - Encoding using **Encoders**
 - Selecting using **Multiplexers**
- Implementing Combinational Functions with
 - Decoders and OR-Gate
 - Multiplexers and inverter
 - ROMs
 - PLAs
 - PALs
 - Lookup Tables



Course Outline





Combinational Circuits

□ Logic Circuits

major categories { **Combinational Logic Circuits**
Sequential Logic Circuits

□ Combinational circuit is defined

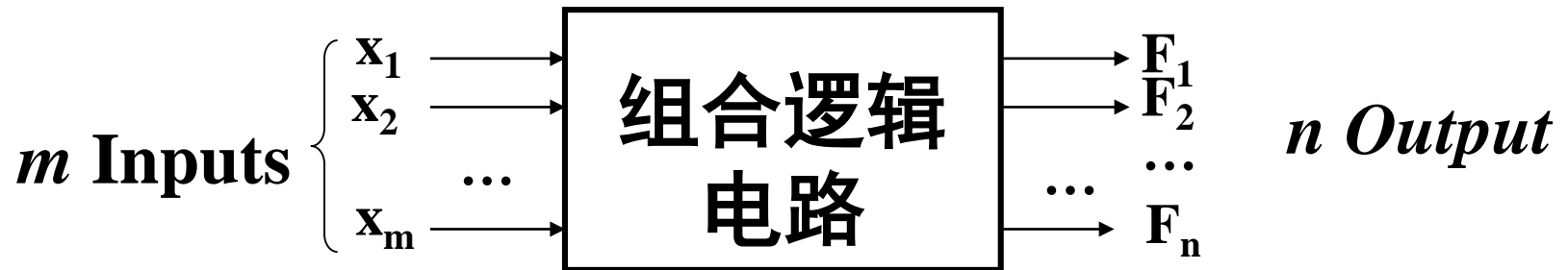
- If a stable **output value** of logic circuit, at any given moment, the value **depends only** on set of the **input values** of the moment, regardless of the past input value, then the circuit is a **combinational logic circuit**.

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:



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

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

$$P_1 = A \oplus B$$

$$P_2 = B \oplus C$$

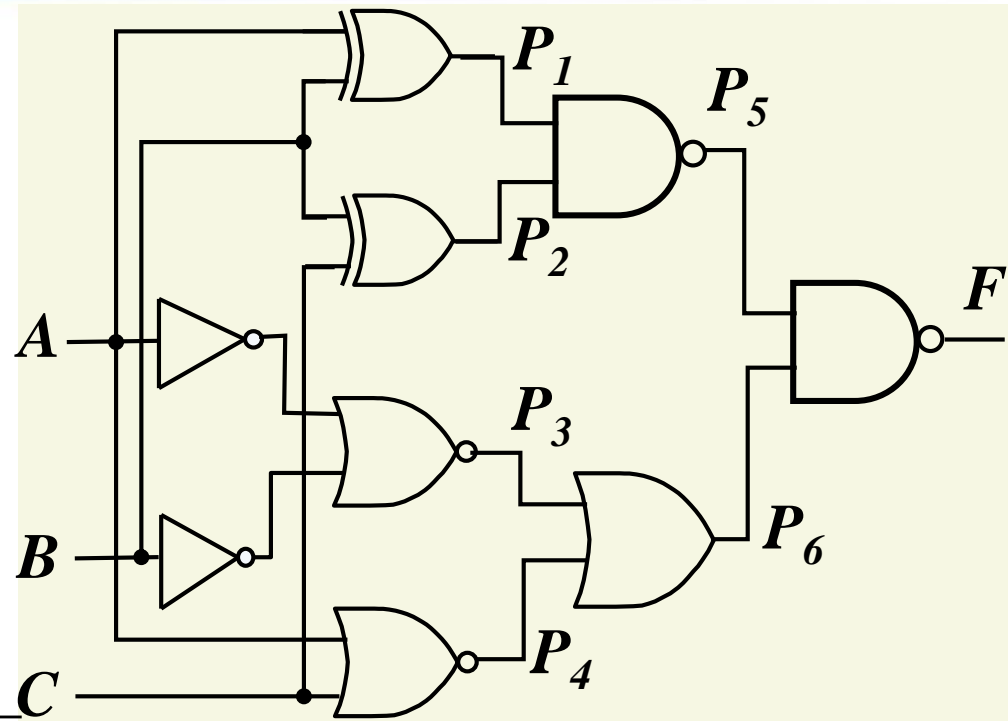
$$P_3 = \overline{\overline{A} + \overline{B}} = AB$$

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

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

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

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



Logic circuit diagram

simplifying function expression

$$\begin{aligned}
 F &= (A \oplus B)(B \oplus C) + \overline{AB + A + C} \\
 &= (\overline{A}B + A\overline{B})(\overline{B}C + B\overline{C}) + (\overline{A} + \overline{B})(A + C) \\
 &= \overline{A}B\overline{C} + \overline{A}B\overline{C} + \overline{A}C + \overline{A}B + \overline{B}C \\
 &= \overline{A}B\overline{C} + \overline{A}C + \overline{A}B + \overline{B}C \\
 &= \overline{A}C + \overline{A}B + \overline{A}B\overline{C} \\
 &= \overline{A}(C + B\overline{C}) + \overline{A}B \\
 &= \overline{A}C + \overline{A}B + \overline{A}B \\
 &= \overline{A}C + (A \oplus B)
 \end{aligned}$$

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

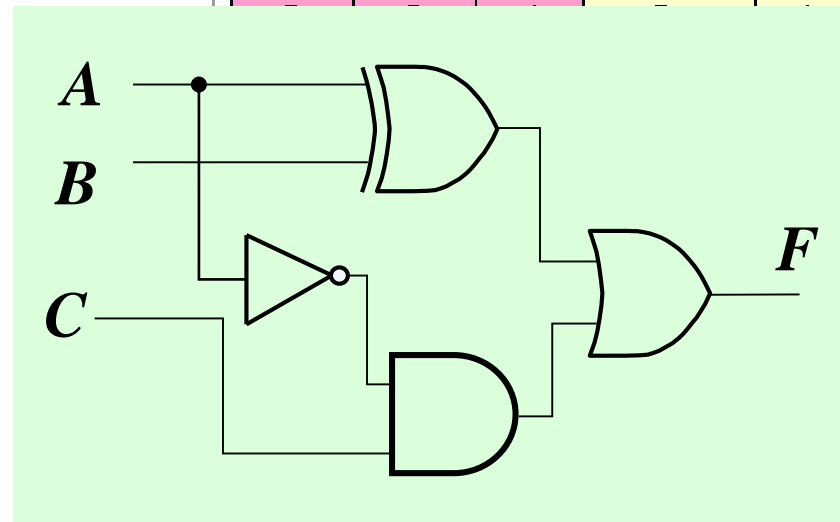
simplifying function expression

Functional Analysis :

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

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
0	1	0	1	0	1
0	1	1	1	0	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



Evaluation:

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

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



Design Procedure

1. Specification (**Logical abstraction -1**)

- Write a specification for the circuit if one is not already available

2. Formulation(**Logical abstraction -2,3** **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

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

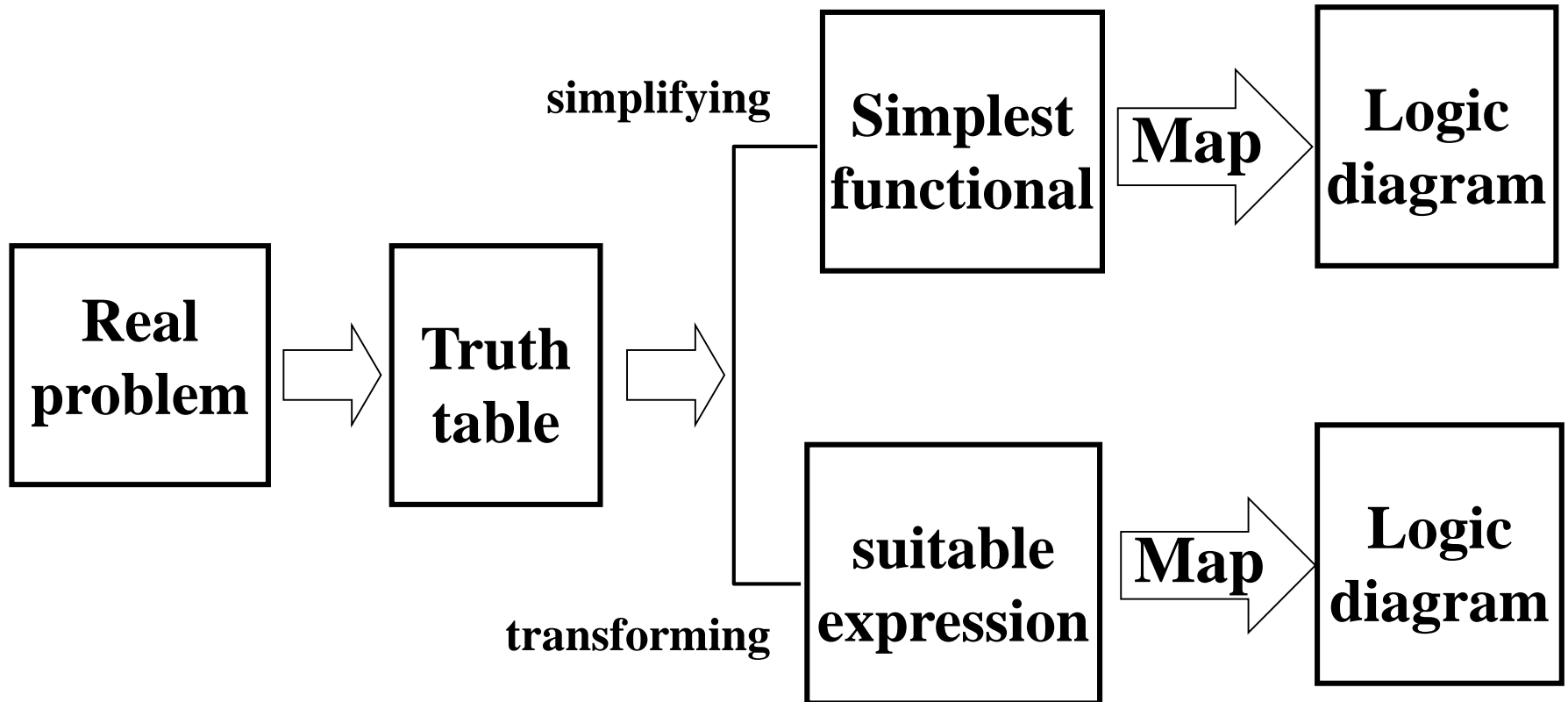
2. To write logic expression (Minterm)

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



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"

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

Step 2: Derive logic expression

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

Design Example

- Only one lamp in the dormitory but there are three beds. Now at the

Step 3: simplifying	Step 1: truth table				Step 2: Karnaugh map				Step 4: simplified expression	
	S_3	S_2	S_1	F	S_3	S_2	S_1	F	S_1	F
	0	0	0	0	0	0	0	0	0	0
	0	0	1	1	0	0	1	1	1	1
	0	1	0	1	0	1	0	1	0	1
	0	1	1	0	0	1	1	0	1	0
	1	0	0	1	1	0	0	1	0	1
	1	0	1	0	1	0	1	0	0	1
	1	1	0	0	1	1	0	0	1	0
	1	1	1	1	1	1	1	1	0	0
	1	1	1	1	1	1	1	1	1	1

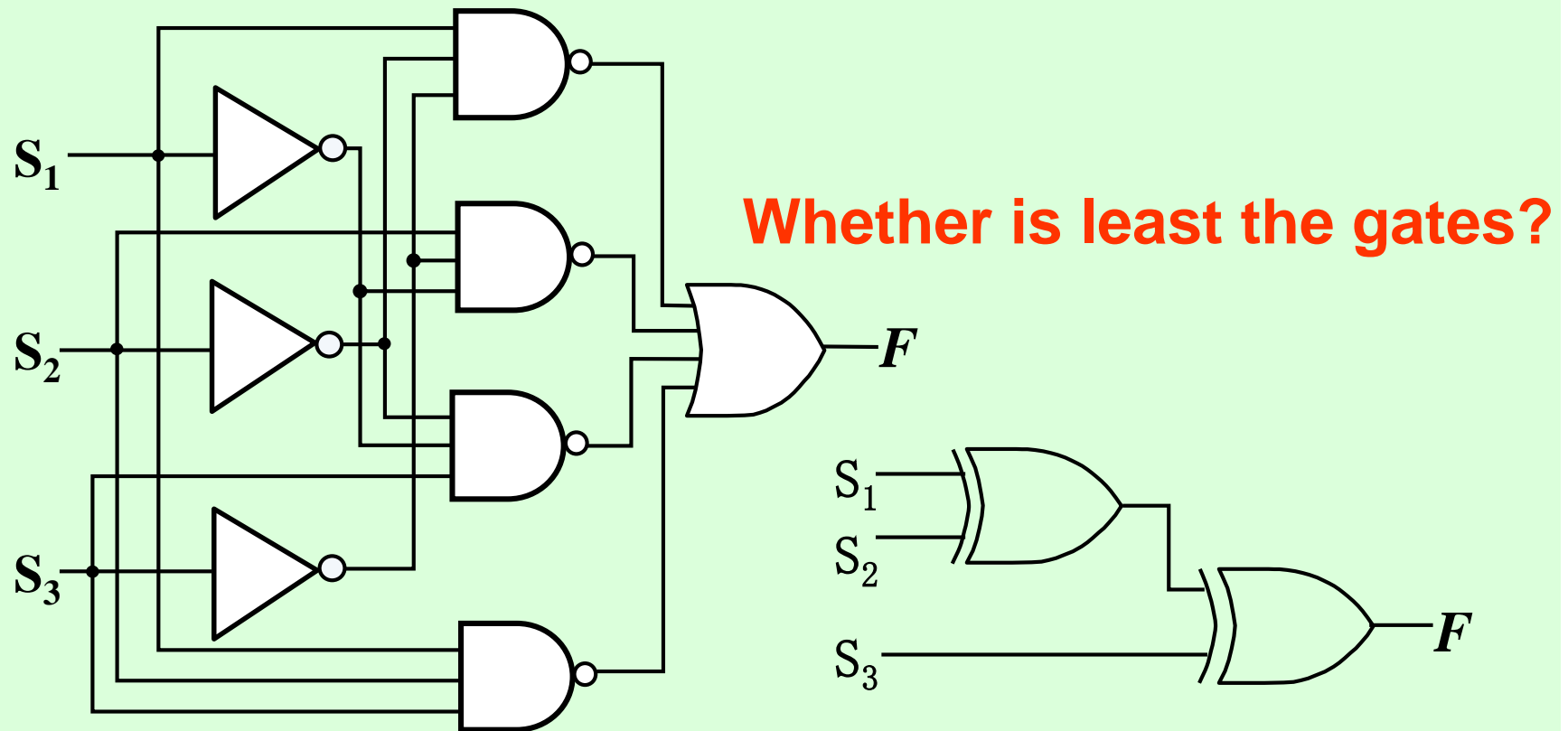
Step 3: simplifying

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

Has been the most simple

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

Step 4: Logical connection diagram



Step 5: Designing circuit module with HDL description

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

```
    input  s1,s2,s3;
```

```
    output F;
```

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

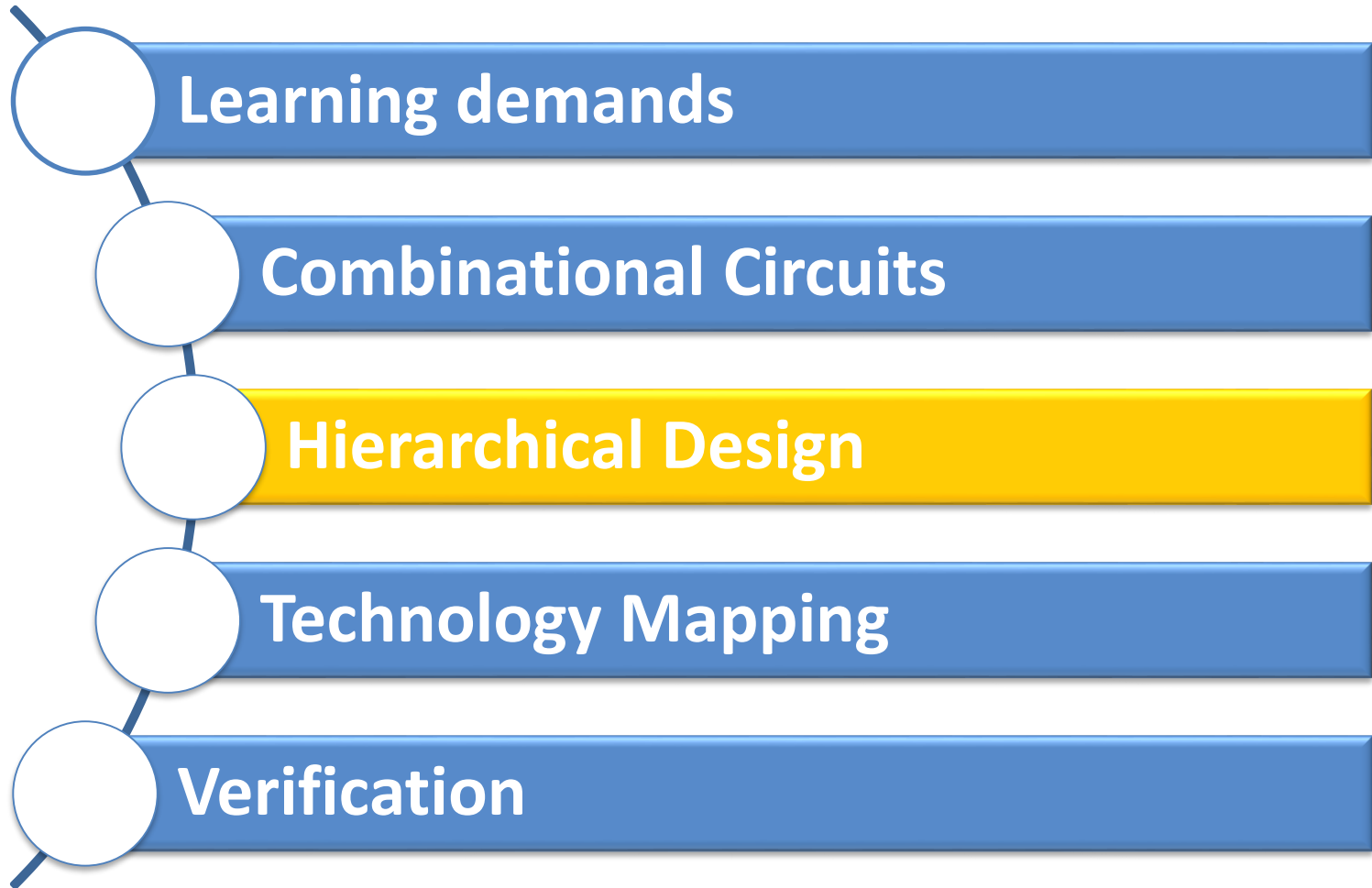
```
    wire  s1,s2,s3,f;
```

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

```
endmodule
```



Course Outline





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

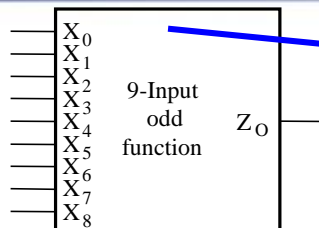
```
module lamp_control (s1, s2, s3, F);  
    input  s1,s2,s3;  
    output F;  
    .....  
endmodule
```

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

Hierarchy for Parity Tree Example

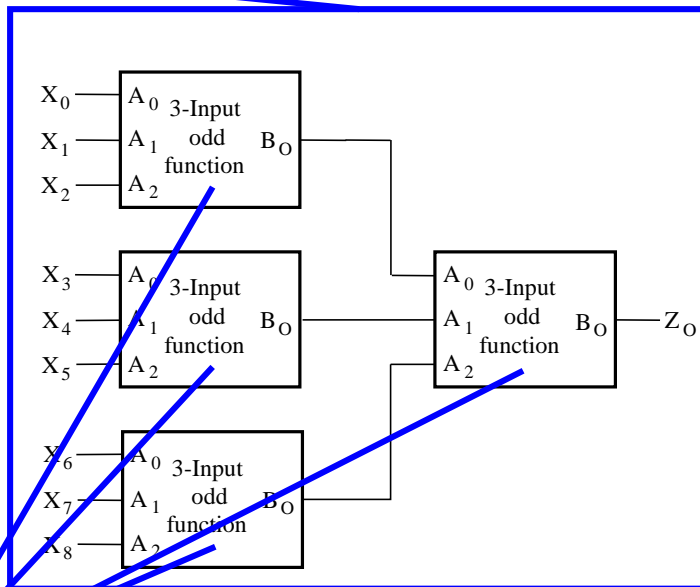


Top Level: 9 inputs, one output



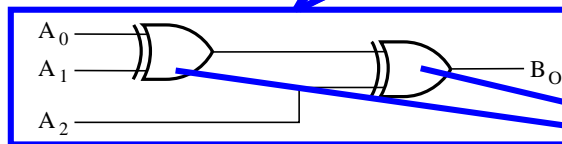
(a) Symbol for circuit

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



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

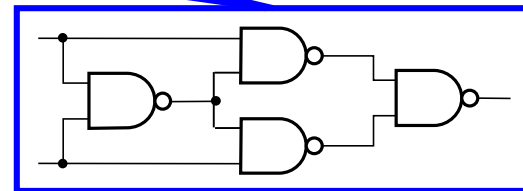
3rd Level: Two 2-bit exclusive-OR functions



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

Primitives: Four 2-input NAND gates

Design requires $4 \times 2 \times 4 = 32$ 2-input NAND gates



(d) Exclusive-OR block as interconnected NANDs



Reusable Functions

□ Function Block

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

□ These blocks are(IP Core)

- Intellectual Property Core
- 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
 - BCD: A,B,C,D
- Variables
 - Excess-3: W,X,Y,Z
- **Don't Cares**
 - BCD 1010 to 1111

Input BCD A B C D	Output Excess-3 W X Y Z
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

Design Example (continued)

3. Optimization

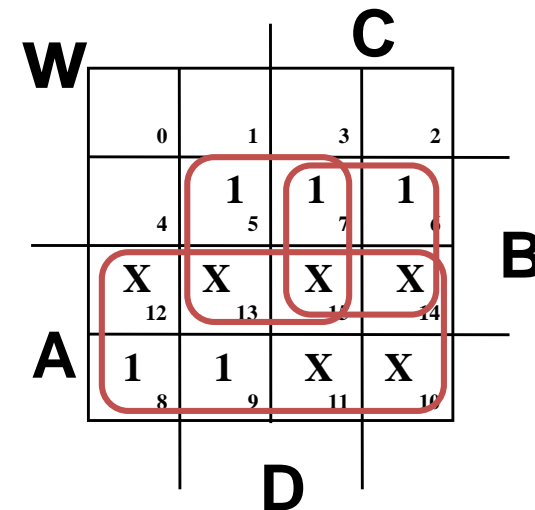
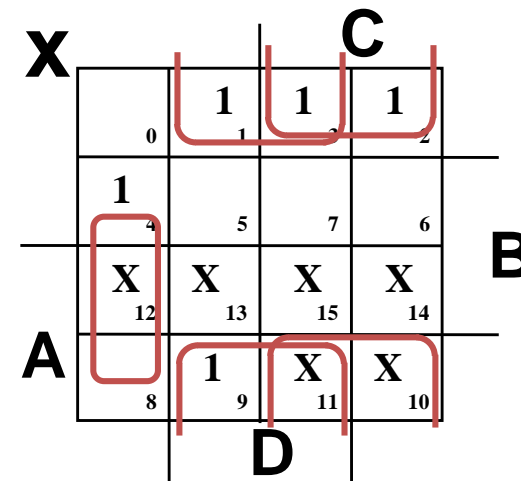
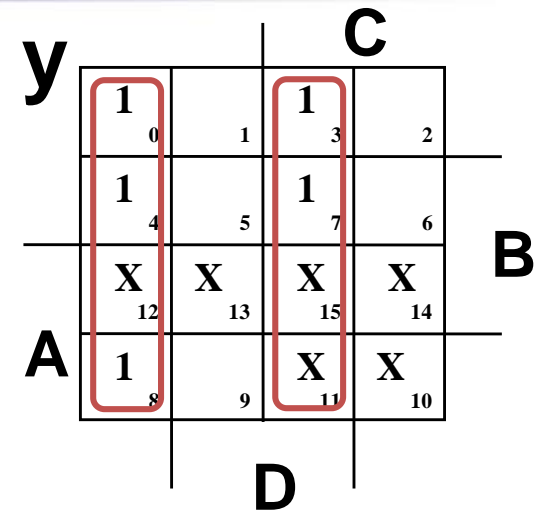
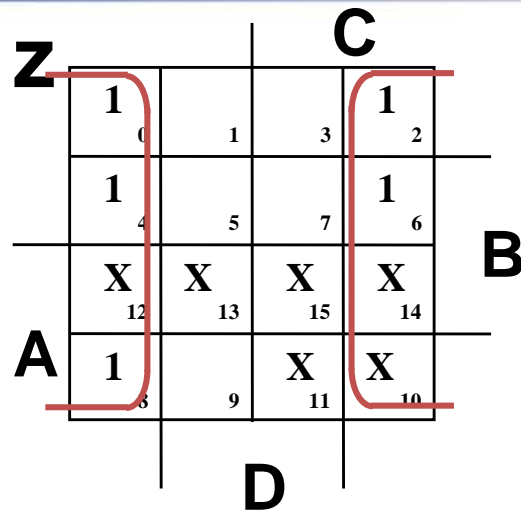
- 2-level using K-maps

$$W = A + BC + BD$$

$$X = \bar{B}C + \bar{B}D + B\bar{C}\bar{D}$$

$$Y = CD + \bar{C}\bar{D}$$

$$Z = \bar{D}$$





Design Example (continued)

3. Optimization (continued)

- Multiple-level using transformations

$$W = A + BC + BD$$

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

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

$$Z = \overline{D}$$

$$G = 7 + 10 + 6 + 0 = 23$$

- Perform extraction, finding factor:

$$T_1 = C + D$$

$$W = A + BT_1$$

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

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

$$Z = \overline{D}$$

$$G = 2 + 1 + 4 + 7 + 6 + 0 = 19$$



Design Example (continued)

3. Optimization (continued)

- Multiple-level using transformations

$$T_1 = C + D$$

$$W = A + BT_1$$

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

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

$$Z = \overline{D}$$

$$G = 19$$

- An additional extraction not shown in the text since it uses a Boolean transformation: ($\overline{C}\overline{D} = \overline{C + D} = \overline{T_1}$):

$$W = A + BT_1$$

$$X = \overline{B}T_1 + B\overline{T_1}$$

$$Y = CD + \overline{T_1}$$

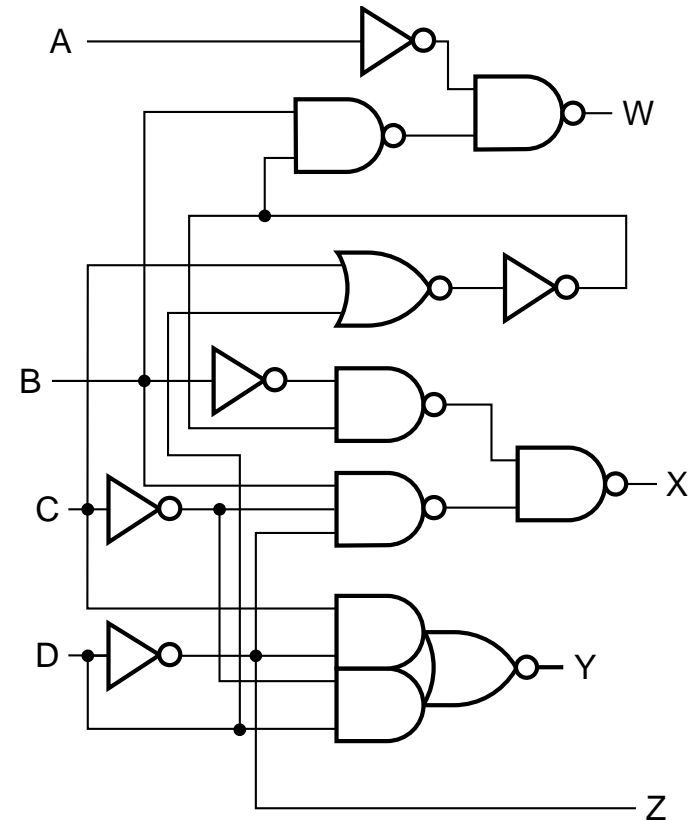
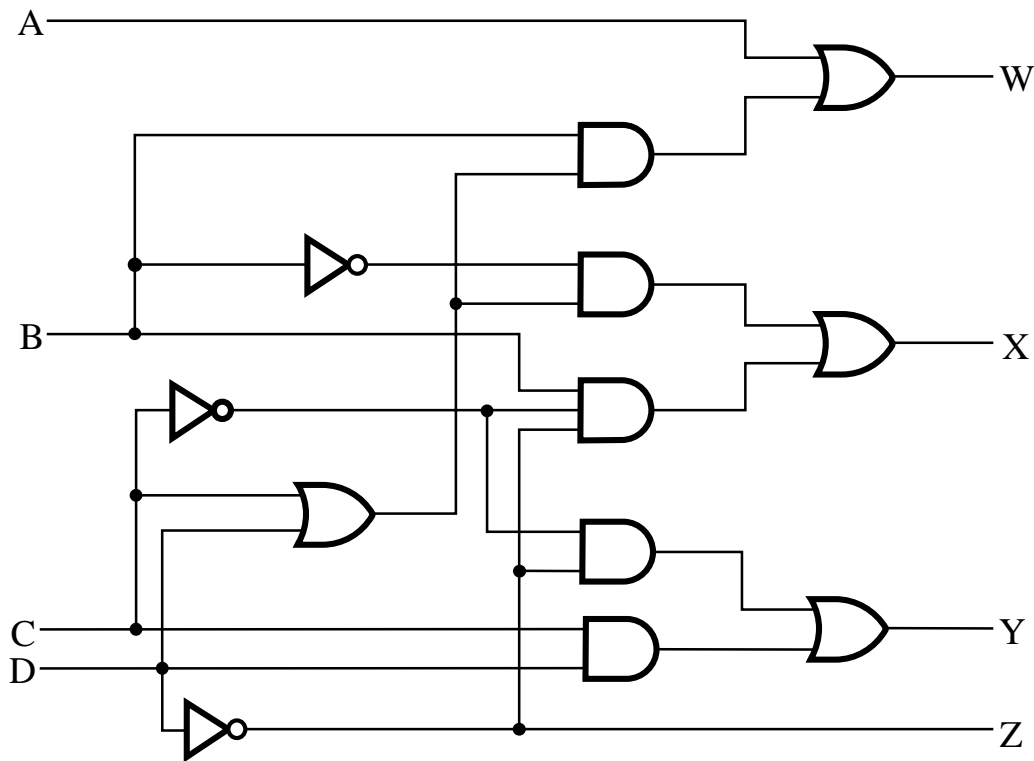
$$Z = \overline{D}$$

$$G = 2 + 1 + 4 + 6 + 4 + 0 = 16!$$

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;
```

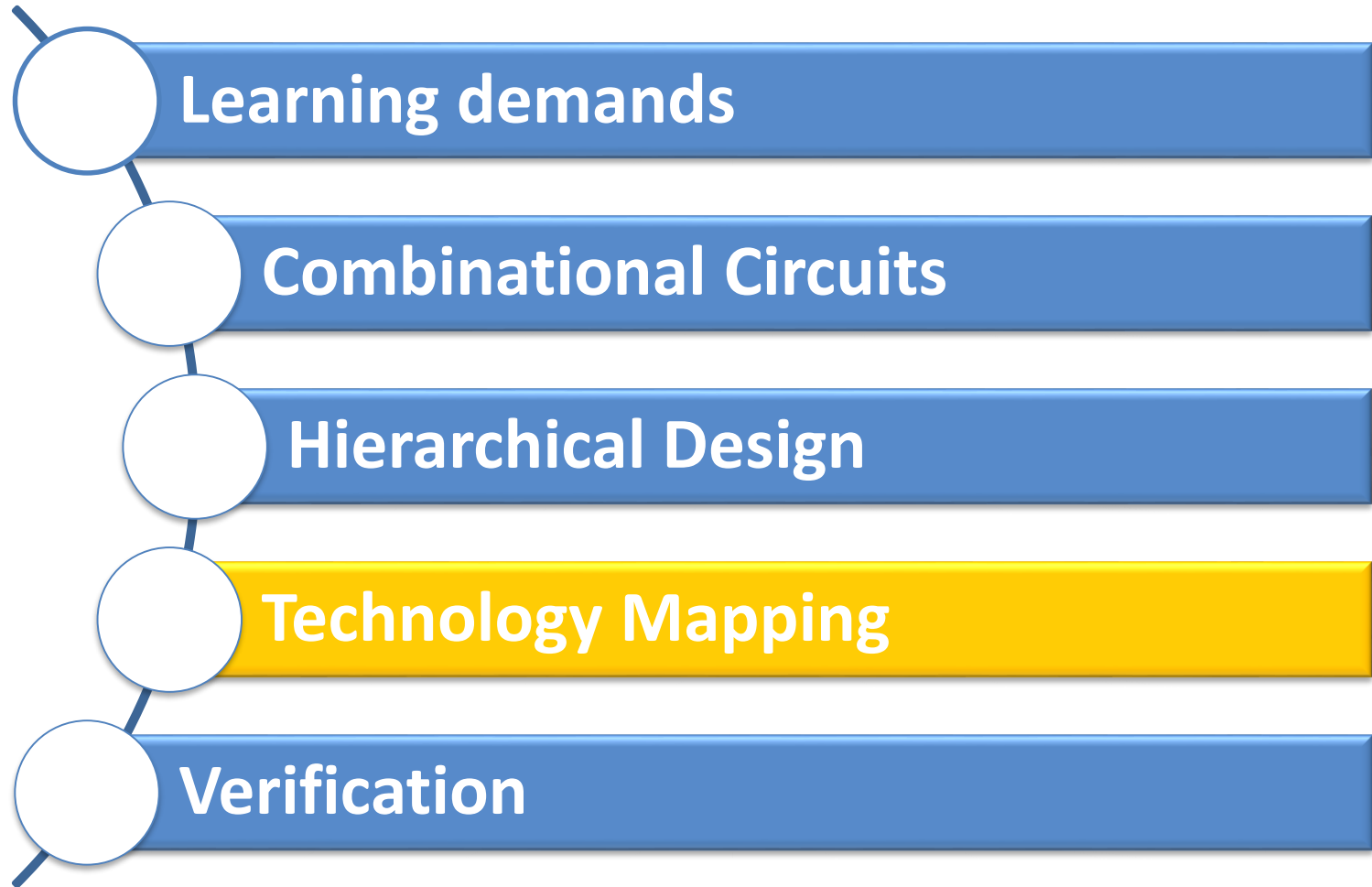
```
    assign Z  = ~D ;
```

```
endmouule
```

也可调用原语描述！



Course Outline





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:

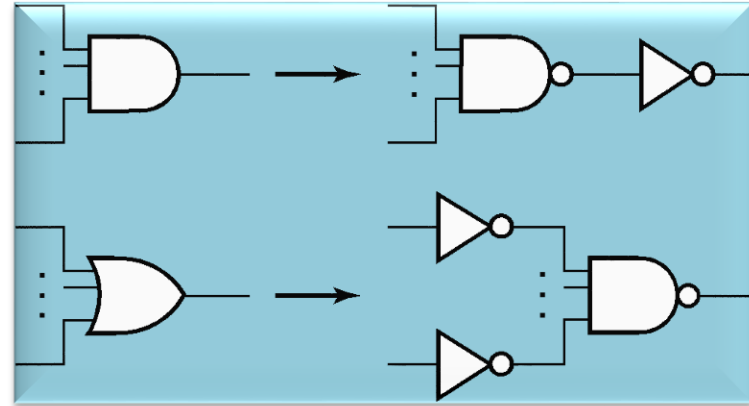
- Gate loading and delay are ignored
- Cell library contains **an inverter** and n -input **NAND** gates, $n = 2, 3, \dots$
- An AND, OR & Inverter schematic for the **circuit** is available

□ The mapping is accomplished by:

- Replacing AND and OR symbols,
- Pushing **Inverters** through circuit *fan-out* points, and
- Canceling **Inverter pairs**

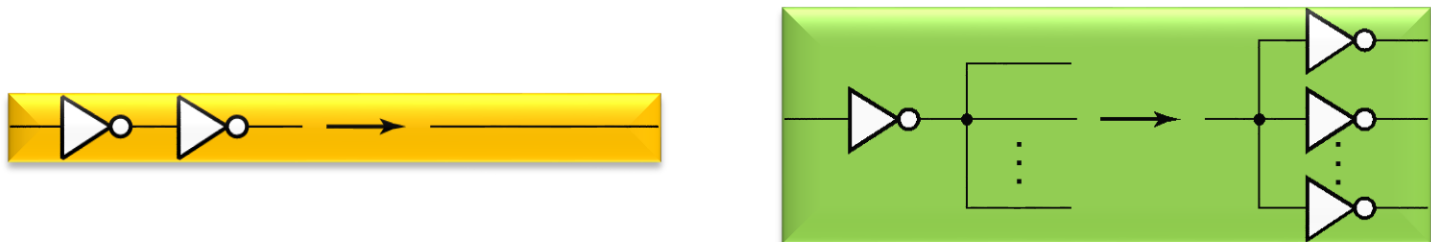
NAND Mapping Algorithm

1. Replace ANDs and ORs:

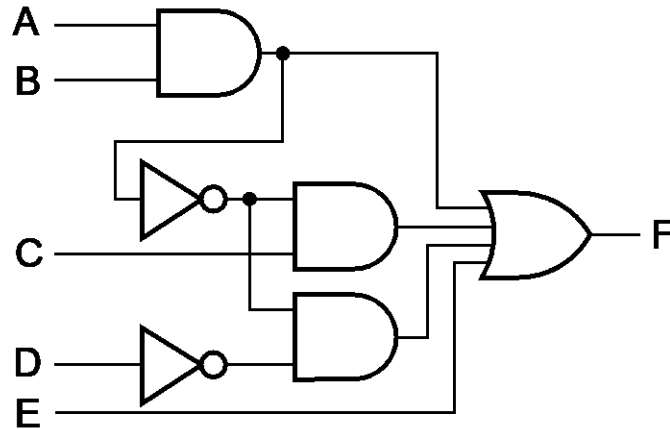


2. Repeat the following pair of actions until there is at most one inverter between :

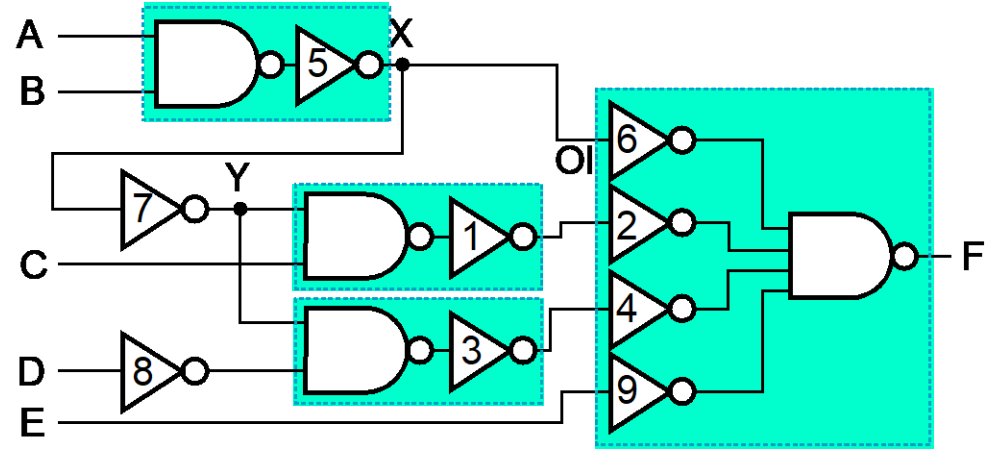
- A circuit input or driving NAND gate output, and
- The attached NAND gate inputs.



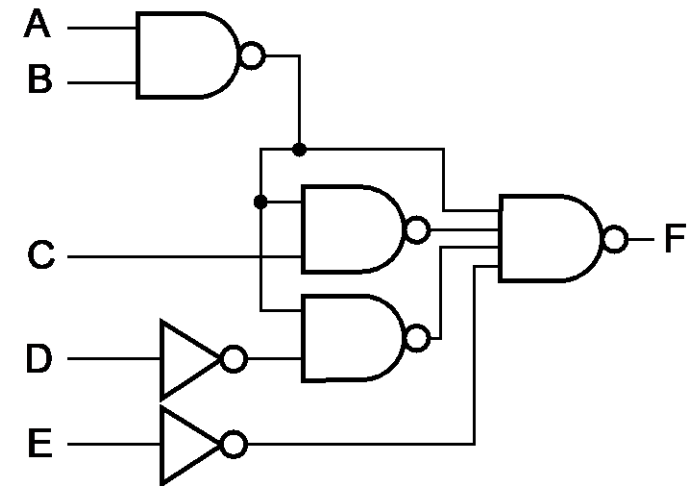
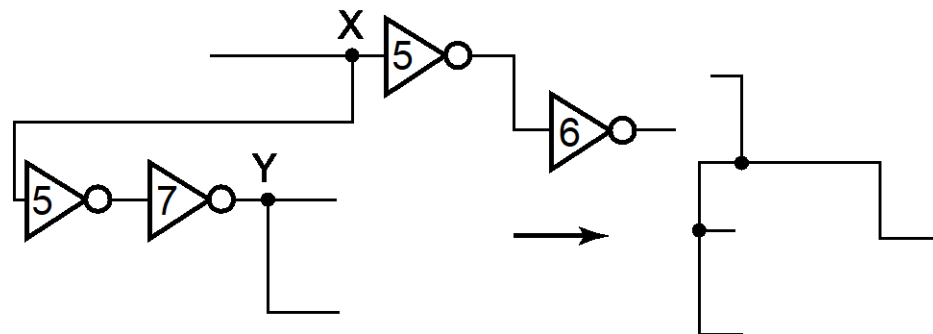
NAND Mapping Example



(a)



(b)



(d)



Mapping to NOR gates

□ Assumptions:

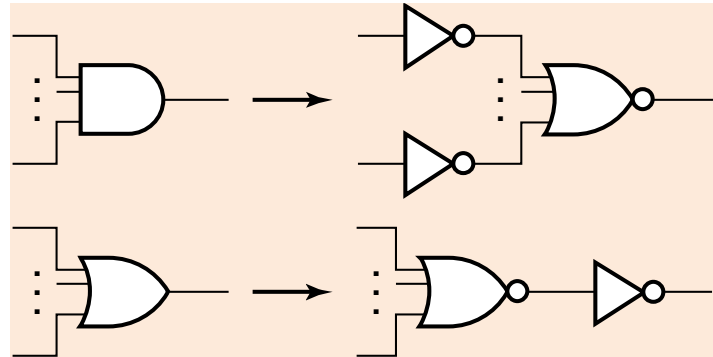
- Gate loading and delay are ignored
- Cell library contains an inverter and n -input NOR gates, $n = 2, 3, \dots$
- An AND, OR, inverter schematic for the circuit is available

□ The mapping is accomplished by:

- Replacing AND and OR symbols,
- Pushing inverters through circuit fan-out points, and
- Canceling inverter pairs

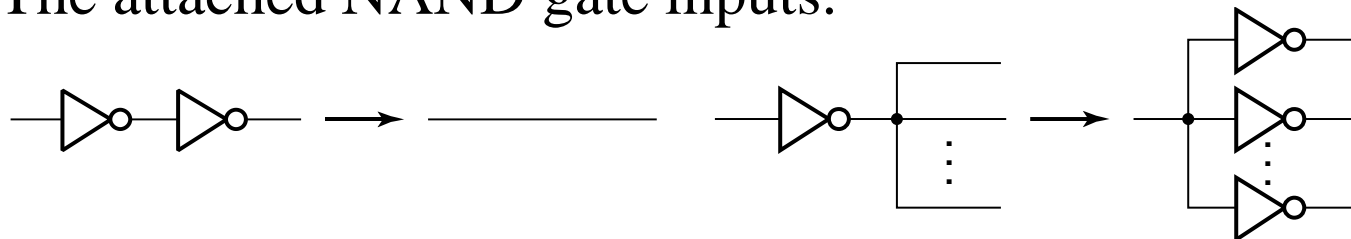
NOR Mapping Algorithm

1. Replace ANDs and ORs:

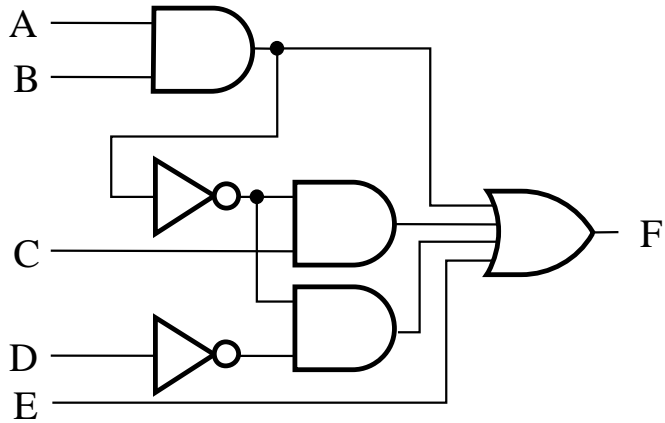


2. Repeat the following pair of actions until there is at most one inverter between :

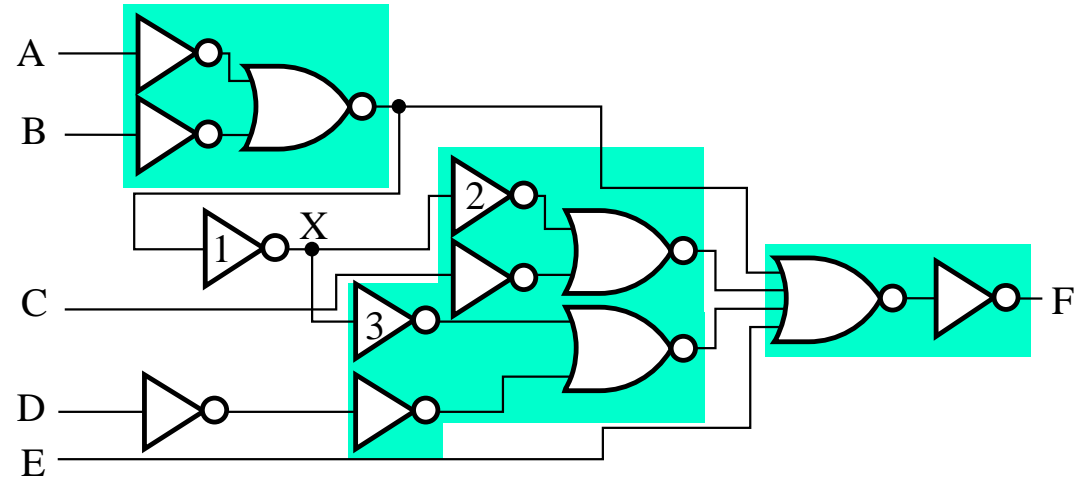
- A circuit input or driving NAND gate output, and
- The attached NAND gate inputs.



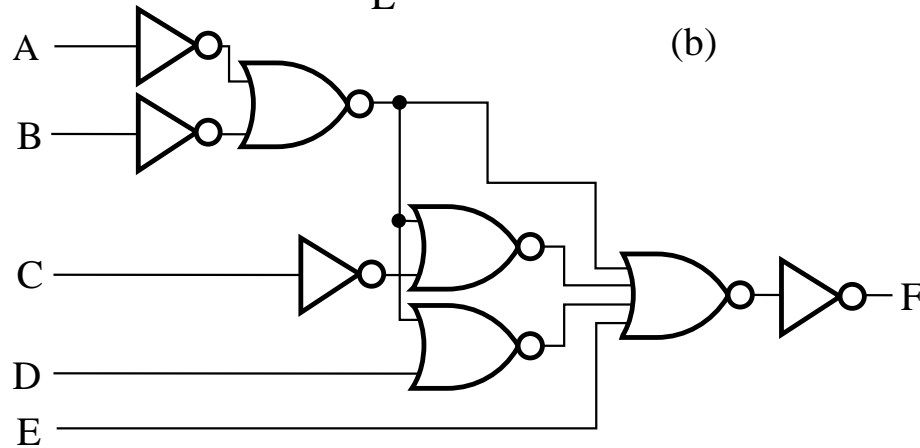
NOR Mapping Example



(a)



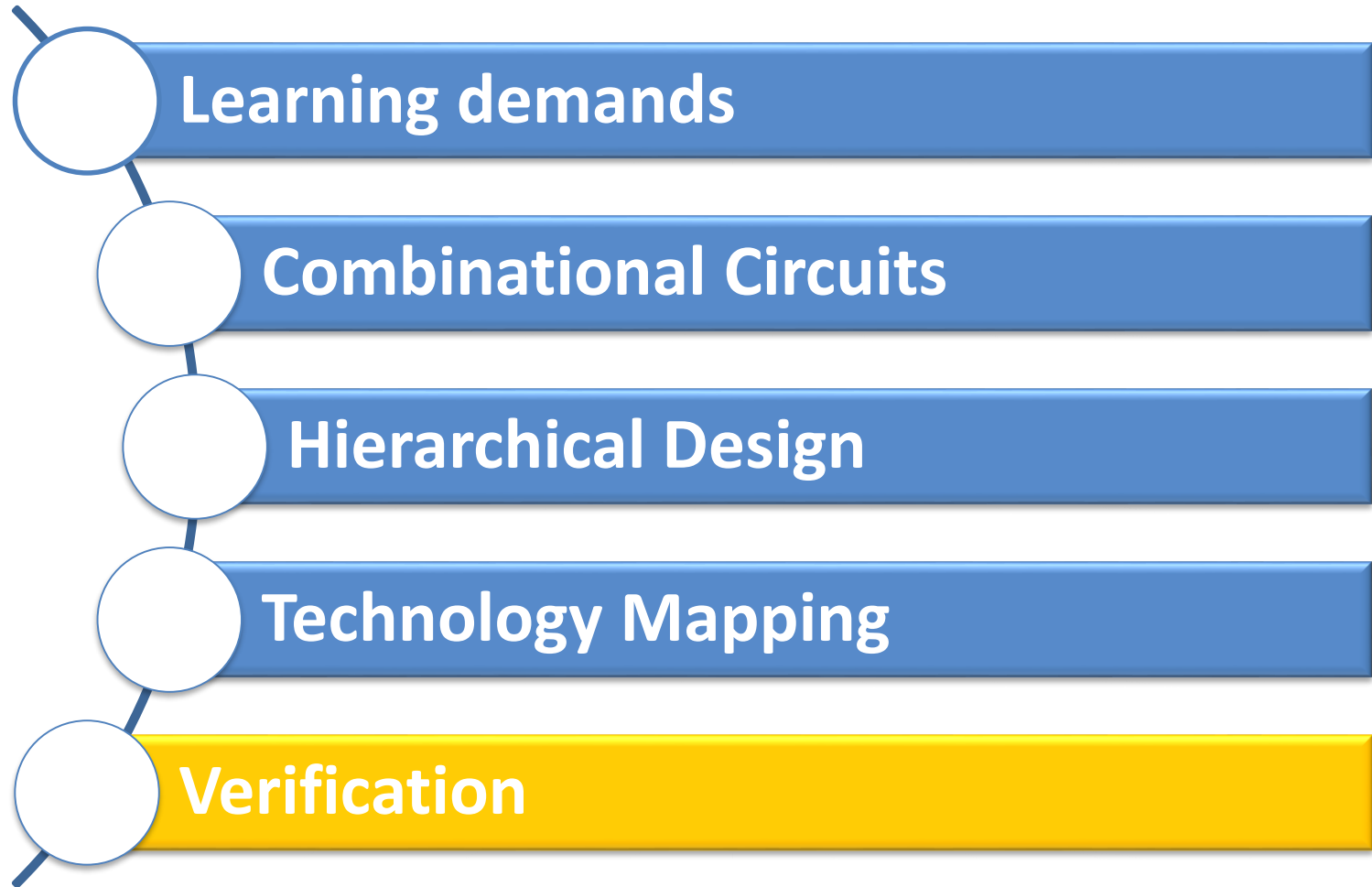
(b)



(c)



Course Outline





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 **formulation** and are **not** the **original specification**, 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:

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

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

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

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

$$Z = \overline{D}$$

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 W X Y Z
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



Verification Example: Simulation

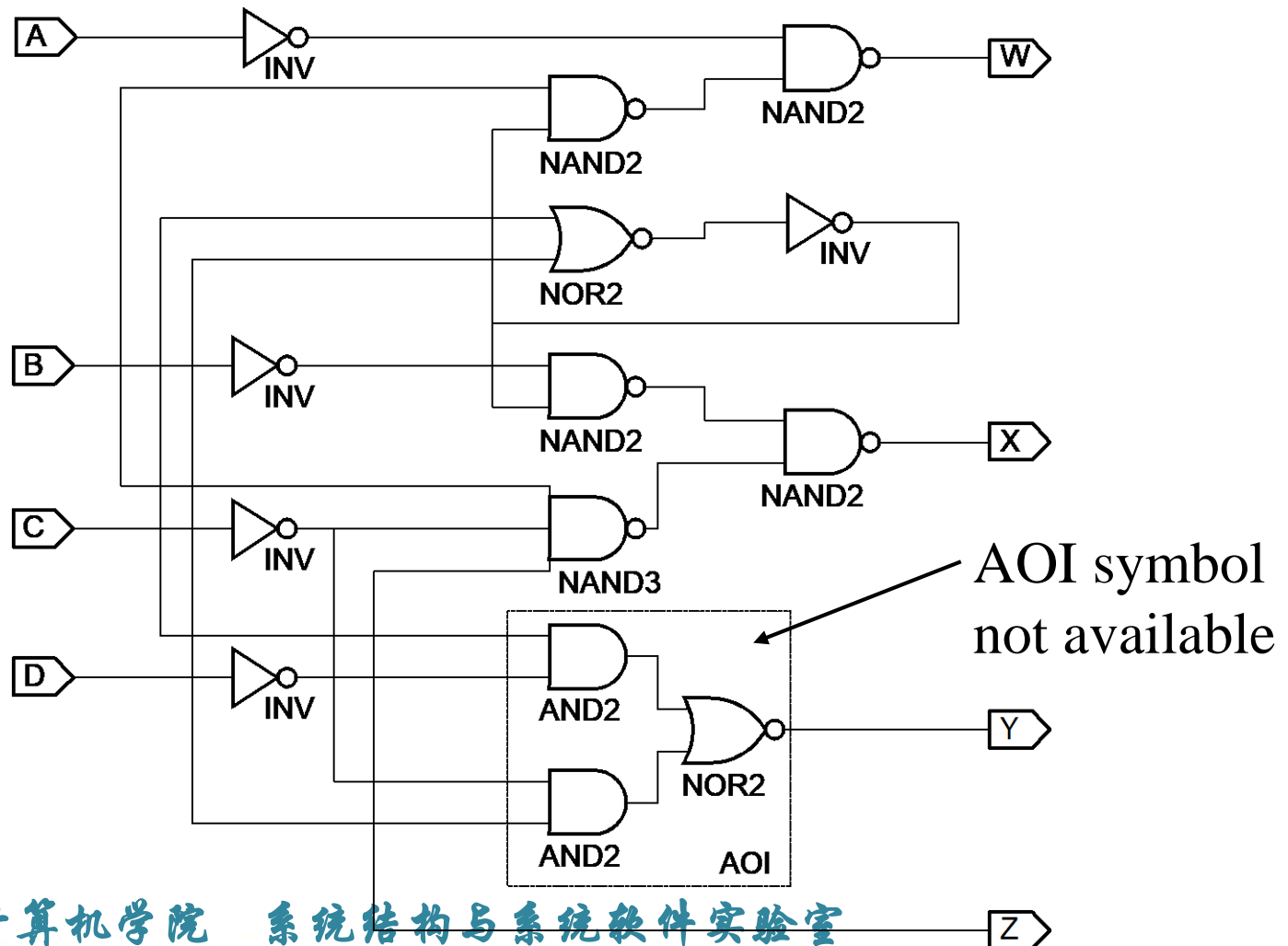
□ Simulation procedure:

- 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



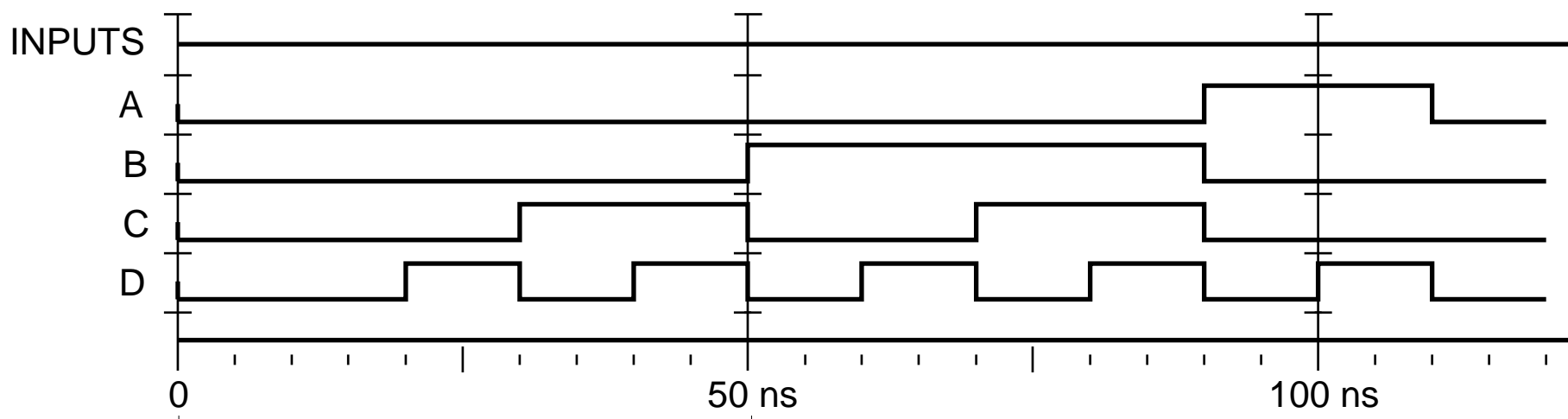
Enter BCD-to-Excess-3 Code Converter Circuit Schematic



Verification Example: Simulation



- Enter waveform that applies all possible input combinations:

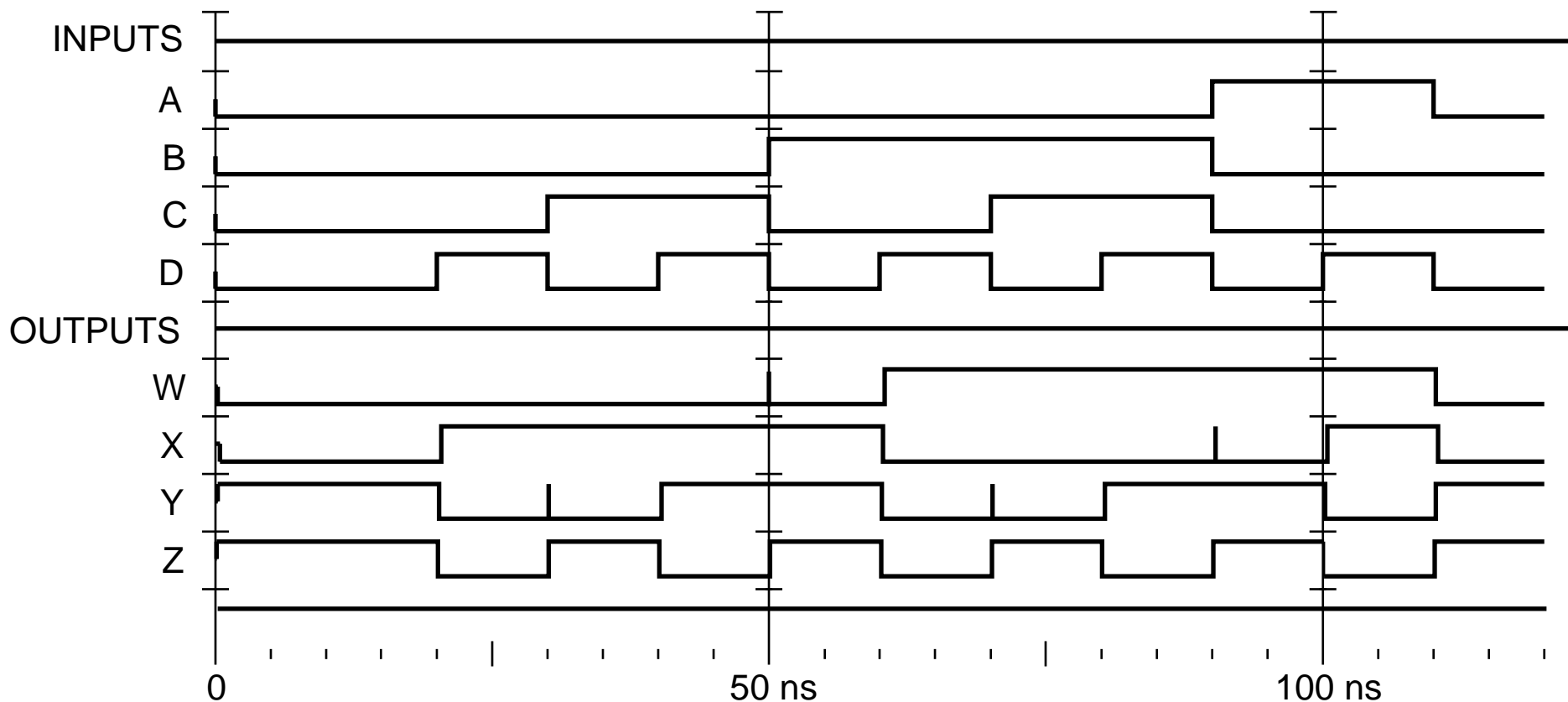


- Are all BCD input combinations present? (Low is a 0 and high is a one)

Verification Example: Simulation



- Run the simulation of the circuit for 120 ns



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



第三章作业1

Ch3

page184-187: 3-7, 3-8, 3-11, 3-13, 3-14

More and more, It is simple also!

Thank you!

