Combinational Circuit Design

EE 200

Digital Logic Circuit Design

Dr. Muhamed Mudawar

King Fahd University of Petroleum and Minerals

Presentation Outline

- ❖ How to Design a Combinational Circuit
- ❖ Designing a BCD to Excess-3 Code Converter
- ❖ Designing a BCD to 7-Segment Decoder
- ❖ Hierarchical Design
- ❖ Iterative Design

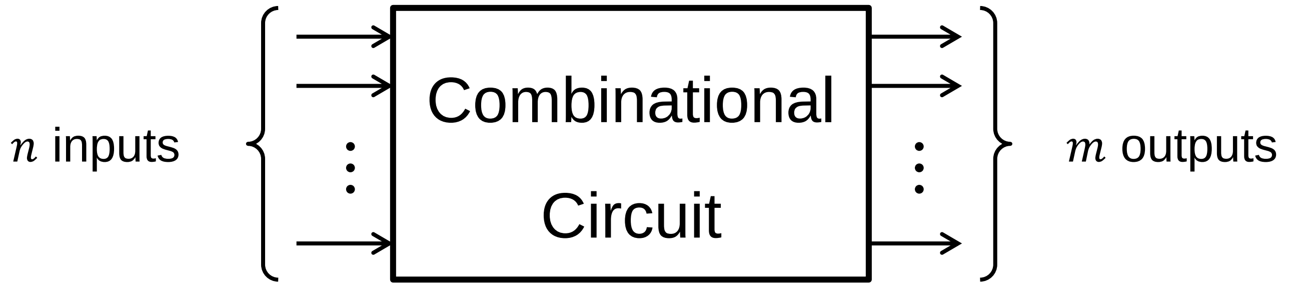
Combinational Circuit

- A combinational circuit is a block of logic gates having:

  n inputs: $x_1, x_2, \dots, x_n$

  m outputs: $f_1, f_2, \dots, f_m$

- Each output is a function of the input variables
- Each output is determined from **present combination** of inputs
- Combination circuit performs operation specified by logic gates



How to Design a Combinational Circuit

1. **Specification**
   - ✧ Specify the inputs, outputs, and what the circuit should do
2. **Formulation**
   - ✧ Convert the specification into truth tables or logic expressions for outputs
3. **Logic Minimization**
   - ✧ Minimize the output functions using K-map or Boolean algebra
4. **Technology Mapping**
   - ✧ Draw a logic diagram using ANDs, ORs, and inverters
   - ✧ Map the logic diagram into the selected technology
   - ✧ Considerations: cost, delays, fan-in, fan-out
5. **Verification**
   - ✧ Verify the correctness of the design, either manually or using simulation

Designing a BCD to Excess-3 Code Converter

1. Specification

- ✧ Convert BCD code to Excess-3 code
- ✧ Input: BCD code for decimal digits 0 to 9
- ✧ Output: Excess-3 code for digits 0 to 9

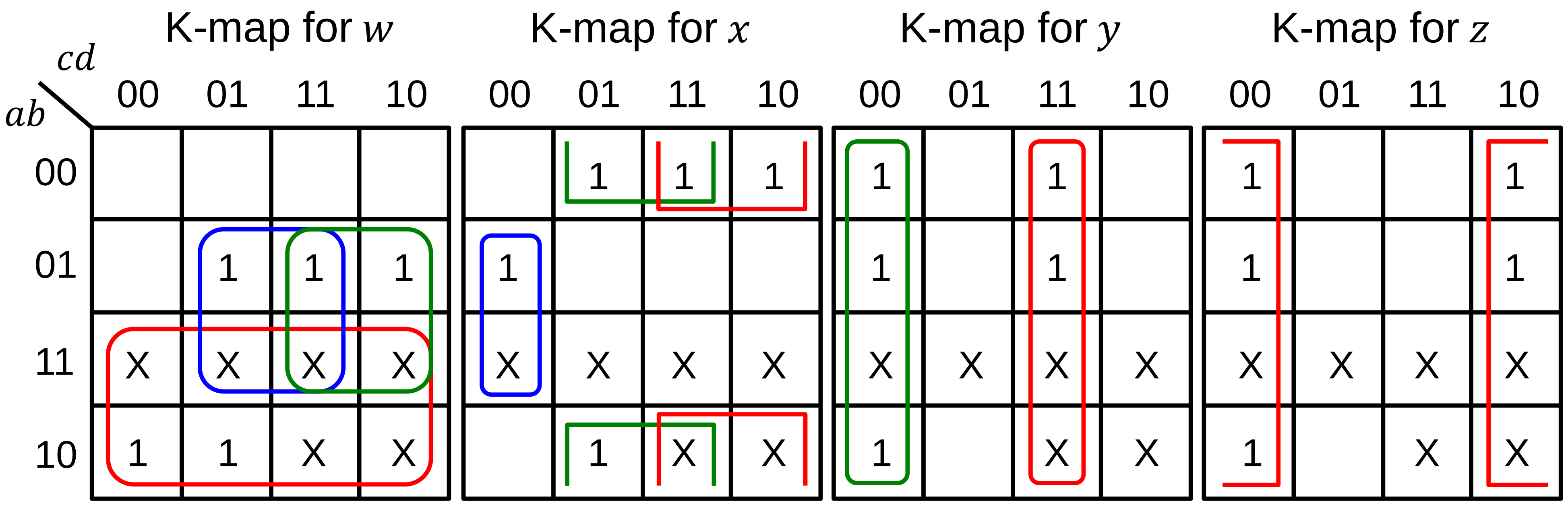
2. Formulation

- ✧ Done easily with a truth table
- ✧ BCD input: a, b, c, d
- ✧ Excess-3 output: w, x, y, z
- ✧ Output is don't care for 1010 to 1111

BCD	Excess-3
a b c d	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 1 0 0
1010 to 1111	X X X X

Designing a BCD to Excess-3 Code Converter

3. Logic Minimization using K-maps



K-map for w

$ab \backslash cd$	00	01	11	10
00				
01		1	1	1
11	X	X	X	X
10	1	1	X	X

K-map for x

$ab \backslash cd$	00	01	11	10
00		1	1	1
01	1			
11	X	X	X	X
10		1	X	X

K-map for y

$ab \backslash cd$	00	01	11	10
00	1		1	
01	1		1	
11	X	X	X	X
10	1		X	X

K-map for z

$ab \backslash cd$	00	01	11	10
00	1			1
01	1			1
11	X	X	X	X
10	1		X	X

Minimal Sum-of-Product expressions:

$w = a + bc + bd$, $x = b'c + b'd + bc'd'$, $y = cd + c'd'$, $z = d'$

Additional 3-Level Optimizations: extract common term $(c + d)$

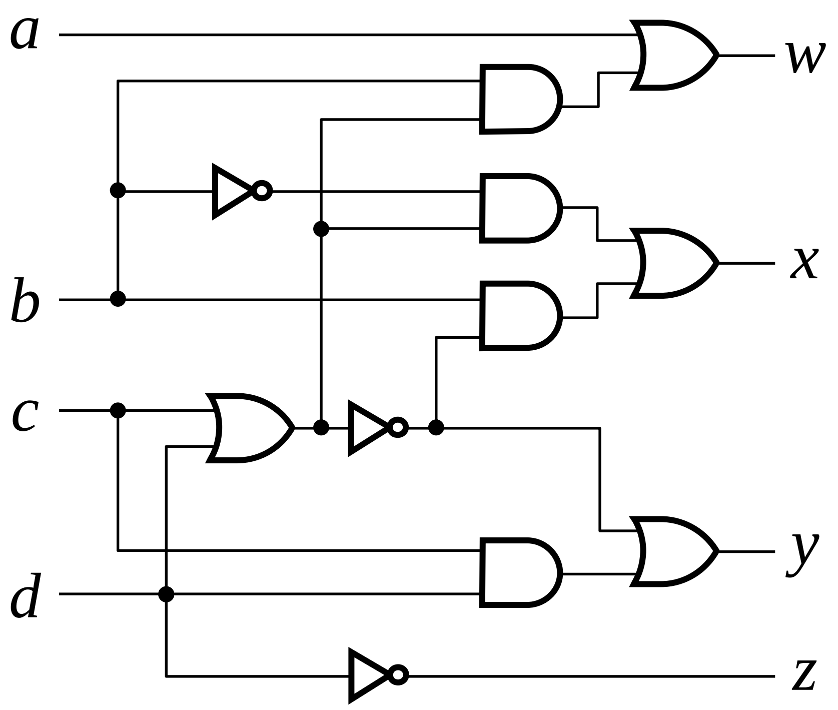
$w = a + b(c + d)$, $x = b'(c + d) + b(c + d)'$, $y = cd + (c + d)'$

Designing a BCD to Excess-3 Code Converter

4. Technology Mapping

Draw a logic diagram using ANDs, ORs, and inverters

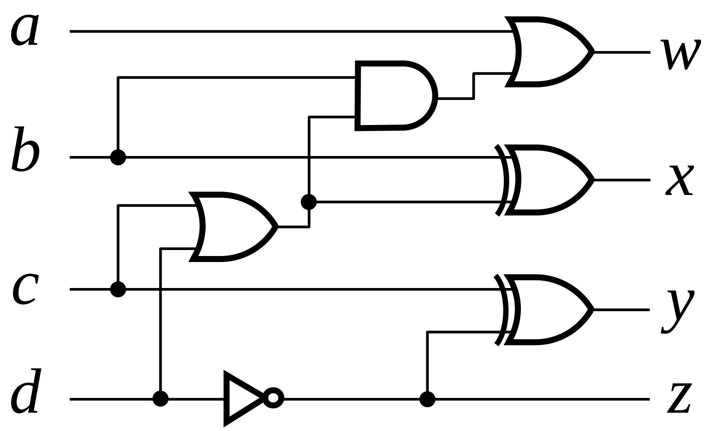
Other gates can be used, such as NAND, NOR, and XOR



Using XOR gates

$$x = b'(c+d) + b(c+d)' = b \oplus (c+d)$$

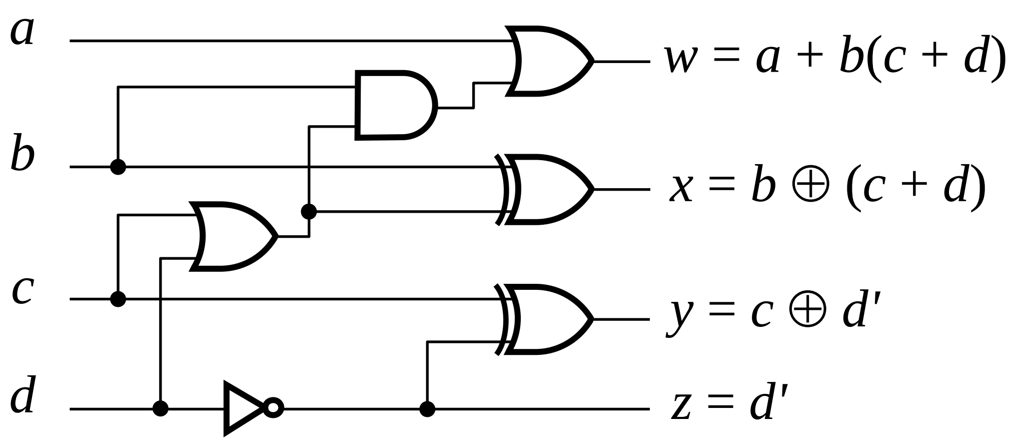
$$y = cd + c'd' = (c \oplus d)' = c \oplus d'$$



Designing a BCD to Excess-3 Code Converter

5. Verification

- Can be done manually
  - Extract output functions from circuit diagram
  - Find the truth table of the circuit diagram
  - Match it against the specification truth table
- Verification process can be automated
  - Using a simulator for complex designs

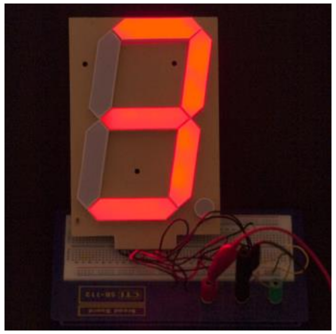


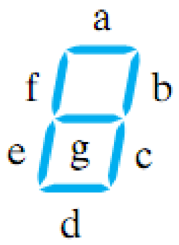
Truth Table of the Circuit Diagram

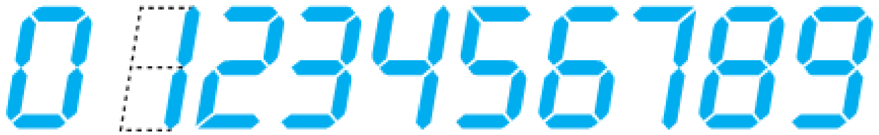
BCD a b c d	c+d	b(c+d)	Excess-3 w x y z
0 0 0 0	0	0	0 0 1 1
0 0 0 1	1	0	0 1 0 0
0 0 1 0	1	0	0 1 0 1
0 0 1 1	1	0	0 1 1 0
0 1 0 0	0	0	0 1 1 1
0 1 0 1	1	1	1 0 0 0
0 1 1 0	1	1	1 0 0 1
0 1 1 1	1	1	1 0 1 0
1 0 0 0	0	0	1 0 1 1
1 0 0 1	1	0	1 1 0 0

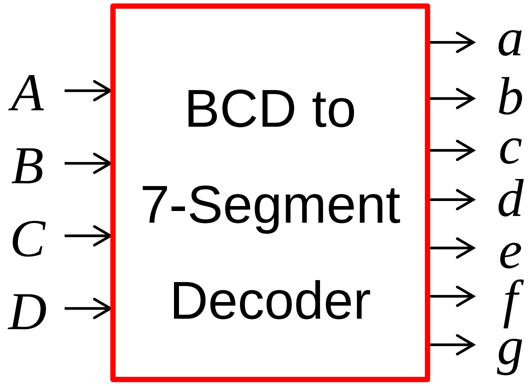
BCD to 7-Segment Decoder

- Seven-Segment Display:
  - Made of Seven segments: light-emitting diodes (LED)
  - Found in electronic devices: such as clocks, calculators, etc.









- BCD to 7-Segment Decoder
  - Accepts as input a BCD decimal digit (0 to 9)
  - Generates output to the seven LED segments to display the BCD digit
  - Each segment can be turned on or off separately

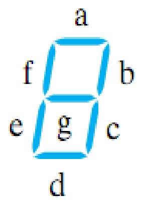
Designing a BCD to 7-Segment Decoder

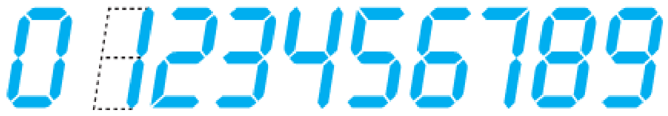
1. Specification:

- ✧ Input: 4-bit BCD (A, B, C, D)
- ✧ Output: 7-bit (a, b, c, d, e, f, g)
- ✧ Display should be OFF for Non-BCD input codes

2. Formulation

- ✧ Done with a truth table
- ✧ Output is zero for 1010 to 1111





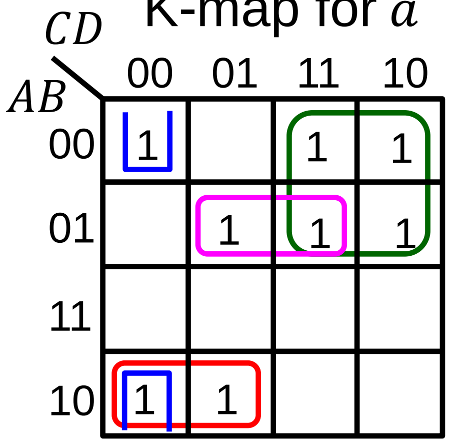
Truth Table

BCD input	7-Segment decoder
A B C D	a b c d e f g
0 0 0 0	1 1 1 1 1 1 0
0 0 0 1	0 1 1 0 0 0 0
0 0 1 0	1 1 0 1 1 0 1
0 0 1 1	1 1 1 1 0 0 1
0 1 0 0	0 1 1 0 0 1 1
0 1 0 1	1 0 1 1 0 1 1
0 1 1 0	1 0 1 1 1 1 1
0 1 1 1	1 1 1 0 0 0 0
1 0 0 0	1 1 1 1 1 1 1
1 0 0 1	1 1 1 1 0 1 1
1010 to 1111	0 0 0 0 0 0 0

Designing a BCD to 7-Segment Decoder

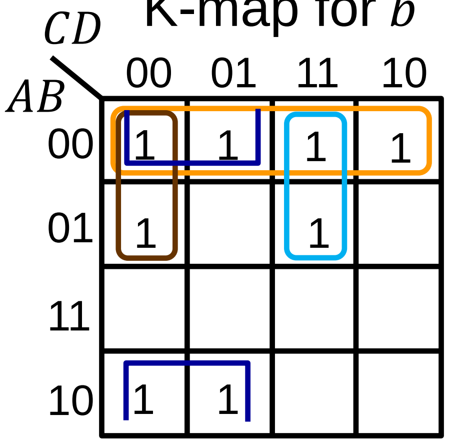
3. Logic Minimization Using K-Maps

K-map for a



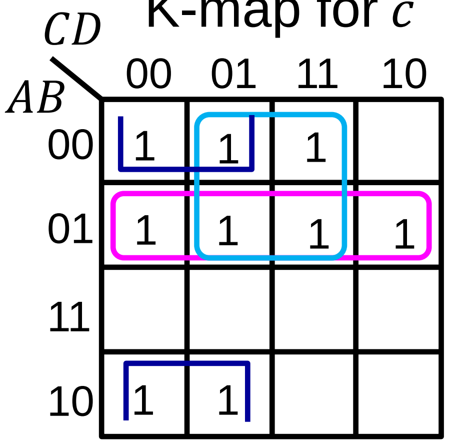
$AB \backslash CD$	00	01	11	10
00	1		1	1
01		1	1	1
11				
10	1	1		

K-map for b



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

K-map for c



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

$$a = A'C + A'BD + AB'C' + B'C'D'$$

$$b = A'B' + B'C' + A'C'D' + A'CD$$

$$c = A'B + B'C' + A'D$$

Extracting common terms

Let $T_1 = A'B,\ T_2 = B'C',\ T_3 = A'D$

Optimized Logic Expressions

$$a = A'C + T_1\,D + T_2\,A + T_2\,D'$$

$$b = A'B' + T_2 + A'C'D' + T_3C$$

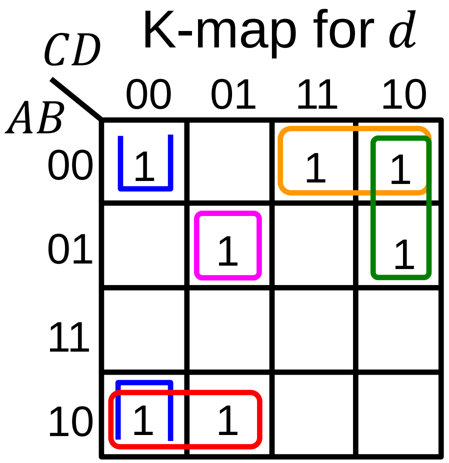
$$c = T_1 + T_2 + T_3$$

T_1, T_2, T_3 are **shared gates**

Designing a BCD to 7-Segment Decoder

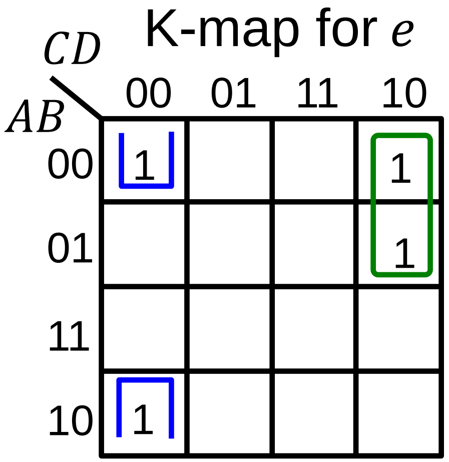
3. Logic Minimization Using K-Maps

K-map for d



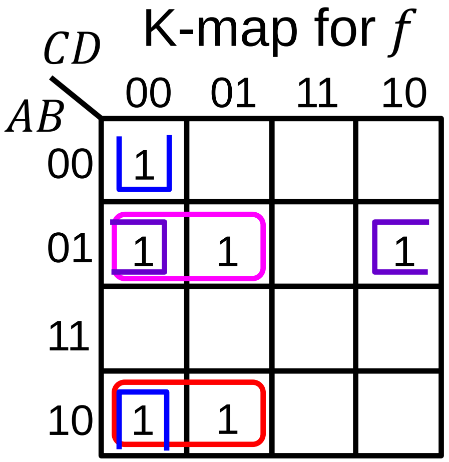
$AB \backslash CD$	00	01	11	10
00	1		1	1
01		1		1
11				
10	1	1		

K-map for e



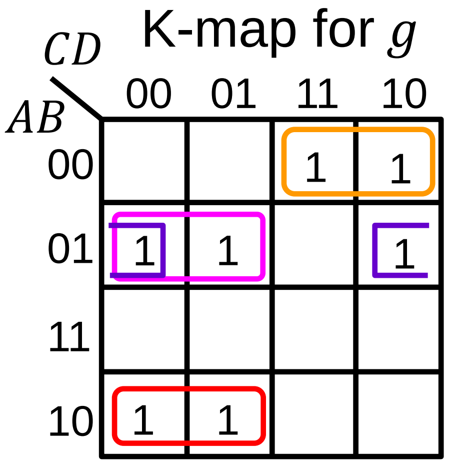
$AB \backslash CD$	00	01	11	10
00	1			1
01				1
11				
10	1			

K-map for f



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

K-map for g



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

Common AND Terms

➔ Shared Gates

$T_4 = AB'C'$, $T_5 = B'C'D'$

$T_6 = A'B'C$, $T_7 = A'CD'$

$T_8 = A'BC'$, $T_9 = A'BD'$

Optimized Logic Expressions

$d = T_4 + T_5 + T_6 + T_7 + T_8\, D$

$e = T_5 + T_7$

$f = T_4 + T_5 + T_8 + T_9$

$g = T_4 + T_6 + T_8 + T_9$

Designing a BCD to 7-Segment Decoder

4. Technology Mapping

Many Common AND terms: T_0 thru T_9

$T_0 = A'C$, $T_1 = A'B$, $T_2 = B'C'$

$T_3 = A'D$, $T_4 = AB'C'$, $T_5 = B'C'D'$

$T_6 = A'B'C$, $T_7 = A'CD'$

$T_8 = A'BC'$, $T_9 = A'BD'$

Optimized Logic Expressions

$a = T_0 + T_1 D + T_4 + T_5$

$b = A'B' + T_2 + A'C'D' + T_3 C$

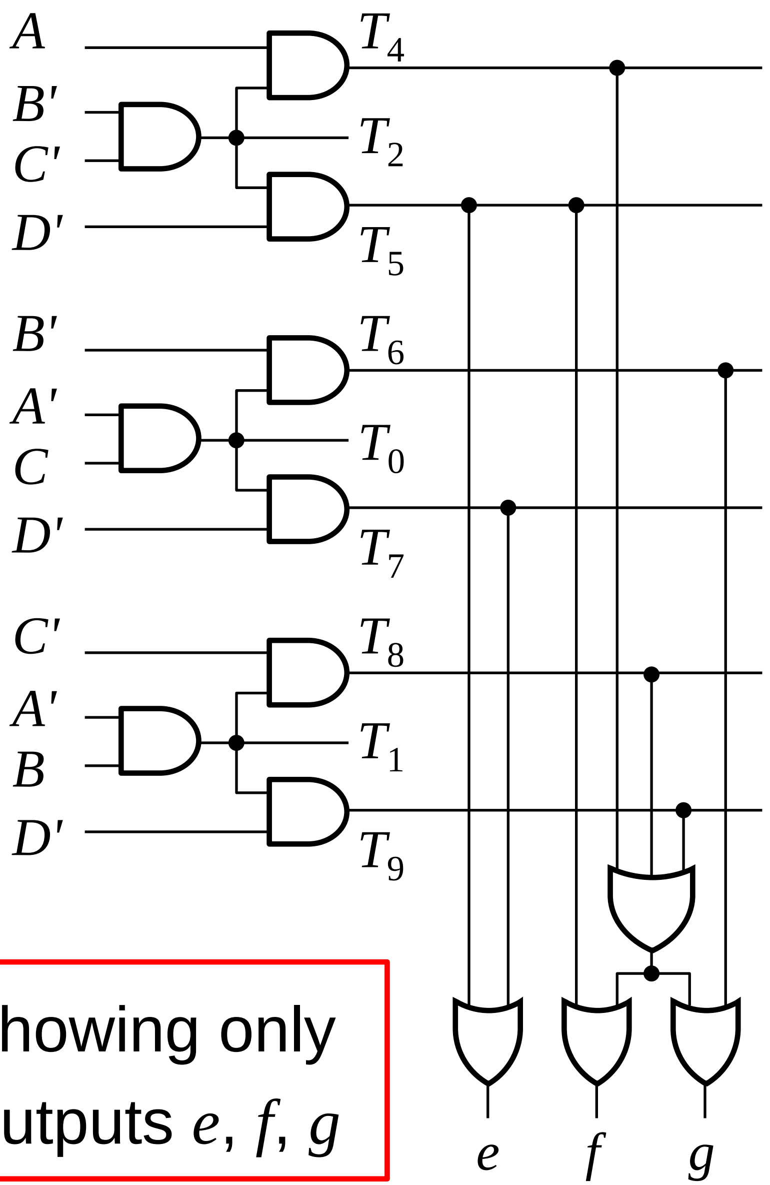
$c = T_1 + T_2 + T_3$

$d = T_4 + T_5 + T_6 + T_7 + T_8 D$

$e = T_5 + T_7$

$f = T_4 + T_5 + T_8 + T_9$

$g = T_4 + T_6 + T_8 + T_9$



Showing only Outputs e, f, g

Verification Methods

- Manual Logic Analysis
  - Find the logic expressions and truth table of the final circuit
  - Compare the final circuit truth table against the specified truth table
  - Compare the circuit output expressions against the specified expressions
  - Tedious for large designs + Human Errors
- Simulation
  - Simulate the final circuit, possibly written in HDL (such as Verilog)
  - Write a test bench that automates the verification process
  - Generate test cases for ALL possible inputs (exhaustive testing)
  - Verify the output correctness for ALL input test cases
  - Exhaustive testing can be very time consuming for many inputs

Hierarchical Design

- ❖ Why Hierarchical Design?

  To simplify the implementation of a complex circuit

- ❖ What is Hierarchical Design?

  Decompose a complex circuit into smaller pieces called blocks

  Decompose each block into even smaller blocks

  Repeat as necessary until the blocks are small enough

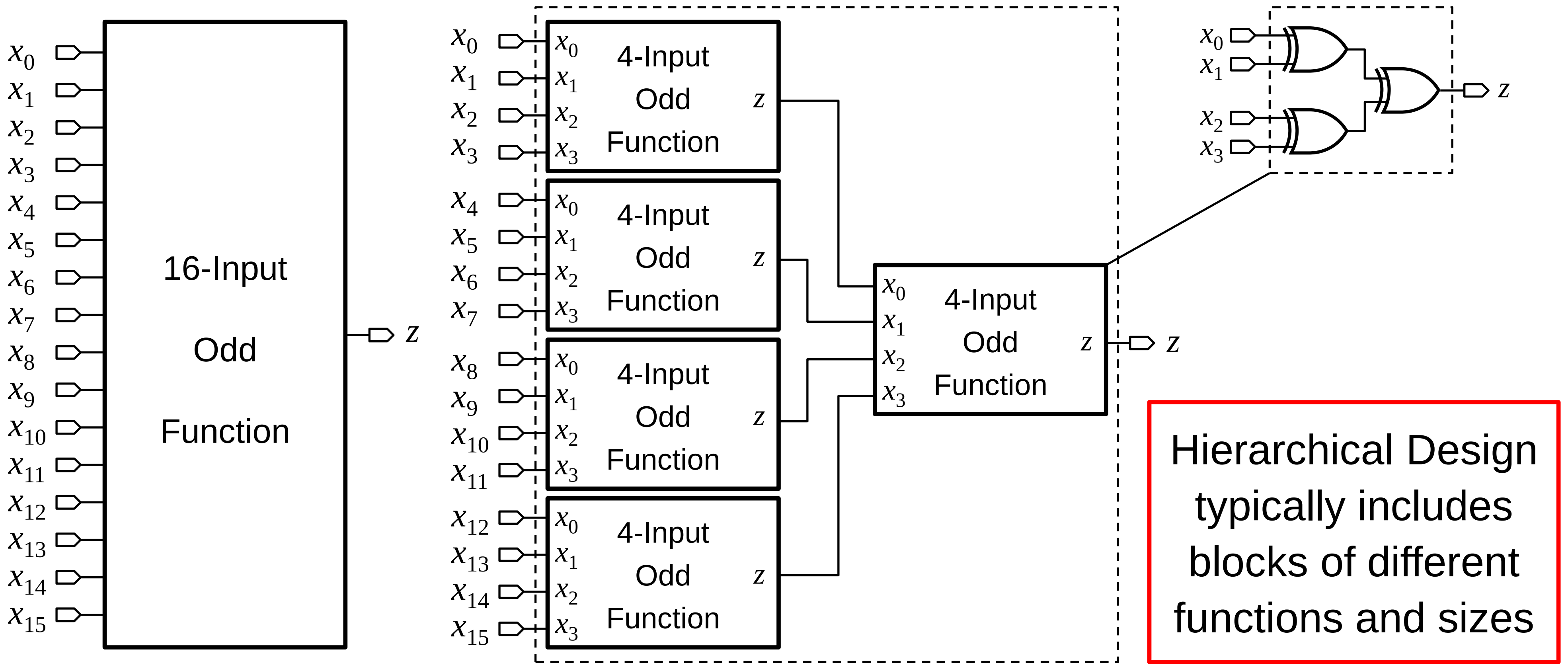
  Any block not decomposed is called a primitive block

  The hierarchy is a tree of blocks at different levels

- ❖ The blocks are verified and well-document
- ❖ They are placed in a library for future use

Example of Hierarchical Design

- ❖ Top Level: 16-input odd function: 16 inputs, one output
  - ✧ Implemented using Five 4-input odd functions
- ❖ Second Level: 4-input odd function that uses three XOR gates



Hierarchical Design typically includes blocks of different functions and sizes

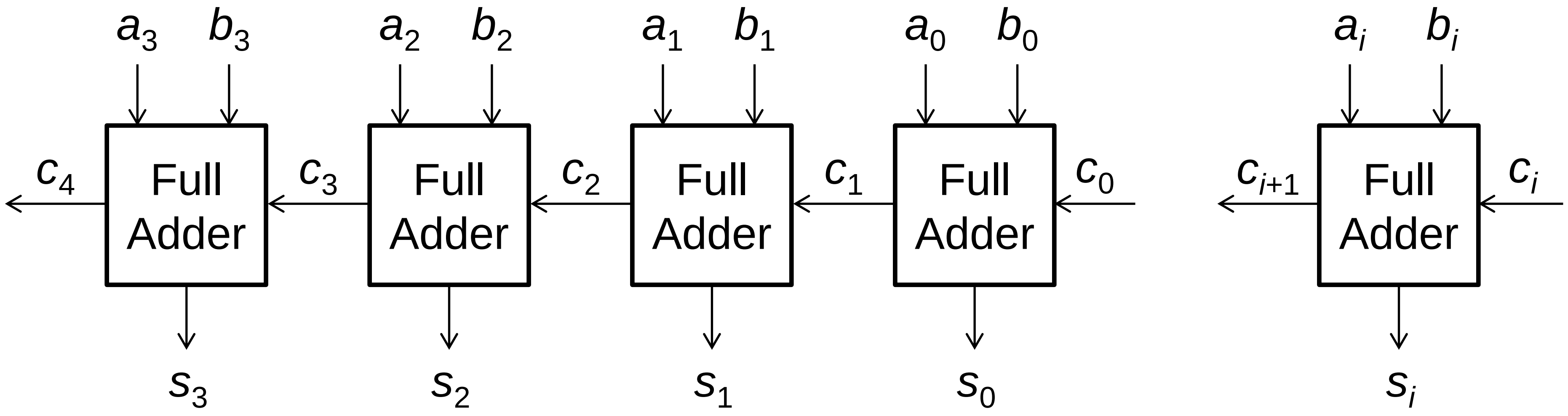
Top-Down versus Bottom-Up Design

- A **top-down design** proceeds from a high-level specification to a more and more detailed design by decomposition and successive refinement
- A **bottom-up design** starts with detailed primitive blocks and combines them into larger and more complex functional blocks
- Design usually proceeds top-down to a known set of building blocks, ranging from complete processors to primitive logic gates

Iterative Design

- Using **identical copies** of a smaller circuit to build a large circuit
- Example: Building a 4-bit adder using 4 copies of a full-adder
- The **cell** (iterative block) is a **full adder**

  Adds 3 bits: a_i, b_i, c_i, Computes: Sum s_i and Carry-out c_{i+1}
- Carry-out of cell i becomes carry-in to cell (i+1)



Full Adder

- Full adder adds 3 bits: **a**, **b**, and **c**
- Two output bits:
  1. Carry bit: **cout**
  2. Sum bit: **sum**
- Sum bit is 1 if the number of 1's in the input is odd (odd function)

  $\mathbf{sum} = (a \oplus b) \oplus c$
- Carry bit is 1 if the number of 1's in the input is 2 or 3

  $\mathbf{cout} = a \cdot b + (a \oplus b) \cdot c$

Truth Table

a b c	cout	sum
0 0 0	0	0
0 0 1	0	1
0 1 0	0	1
0 1 1	1	0
1 0 0	0	1
1 0 1	1	0
1 1 0	1	0
1 1 1	1	1