Lecture 3: Combinational Logic Design

Chapter 3.1 – 3.2

Objectives

- Discuss the procedures in designing a combinational circuit based on the problem specification
- Discuss the techniques in *technology mapping* of a circuit to NAND and NOR gate technologies
- Verify a logic circuit implementation through logic analysis process

Outline

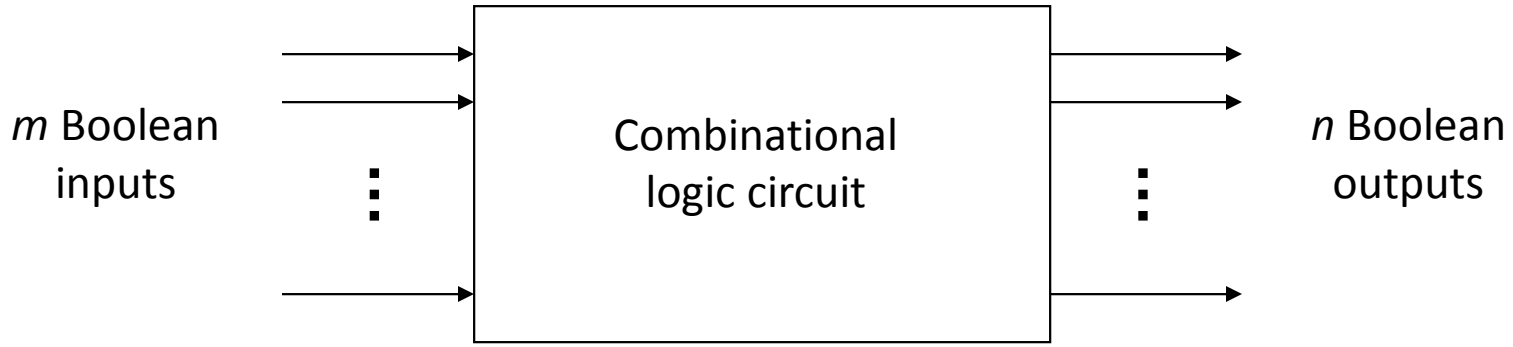
- Combinational circuit design procedures
- BCD-to-excess-3 code converter
- BCD-to-seven-segment decoder
- Hierarchical design
- Technology mapping and verification

Outline

- Combinational circuit design procedures
- BCD-to-excess-3 code converter
- BCD-to-seven-segment decoder
- Hierarchical design
- Technology mapping and verification

Design procedures (1)

- Five procedures to design a combinational circuit
  - Combinational circuit: a circuit made up of logic gates, which produces an output given some inputs
  - Specification → Formulation → Optimization → Technology mapping → Verification
- Objective
  - Given the problem specification, draw the logic circuit diagram of the combinational circuit



Design procedures (2)

- Specification
  - Write a specification for the circuit
  - i.e., the problem statement that describes the input-output requirement)

- Formulation
  - Derive the truth table or initial Boolean equations that define the required relationships between inputs and outputs
  - Convert problem statement to these form for optimization of circuit

Design procedures (3)

- Optimization
  - Apply two-level and/or **multiple-level** optimization, and
  - Draw a logic circuit diagram for the resulting circuit using AND, OR and NOT gates
  - Techniques: algebraic manipulation, K-map
- Technology mapping
  - Transform the logic circuit diagram to a new diagram using the available implementation technology
- Verification
  - Verify the correctness of the final design

Outline

- Combinational circuit design procedures
- BCD-to-excess-3 code converter
- BCD-to-seven-segment decoder
- Hierarchical design
- Technology mapping and verification

BCD-to-excess-3 code converter (1)

- Specification
  - BCD: binary-coded decimals
  - Excess-3-code
    - Each decimal has its *excess-3-code*
    - Binary combination of the decimal **plus 3**
    - Example: Excess-3-code of 5 is 1000
  - Requirement
    - A converter which converts the BCD of a digit to its excess-3-code
    - Input and output digits represented in binary format
      - Input: ABCD
      - Output: WXYZ
    - I.e., Input of the converter: A, B, C and D;
    - Output of the converter: W, X, Y and Z

BCD-to-excess-3 code converter (2)

- Formulation
  - Derive the truth table for the converter
  - 4 variables as BCD **input**, $2^4 = 16$ possible input combinations
    - However, combinations from 1010 to 1111 are invalid BCD code; assume they never happen and are don't care conditions
  - Add 3, i.e., 0011 in binary, to the BCD input to obtain the excess-3-code, as **output**

BCD-to-excess-3 code converter (3)

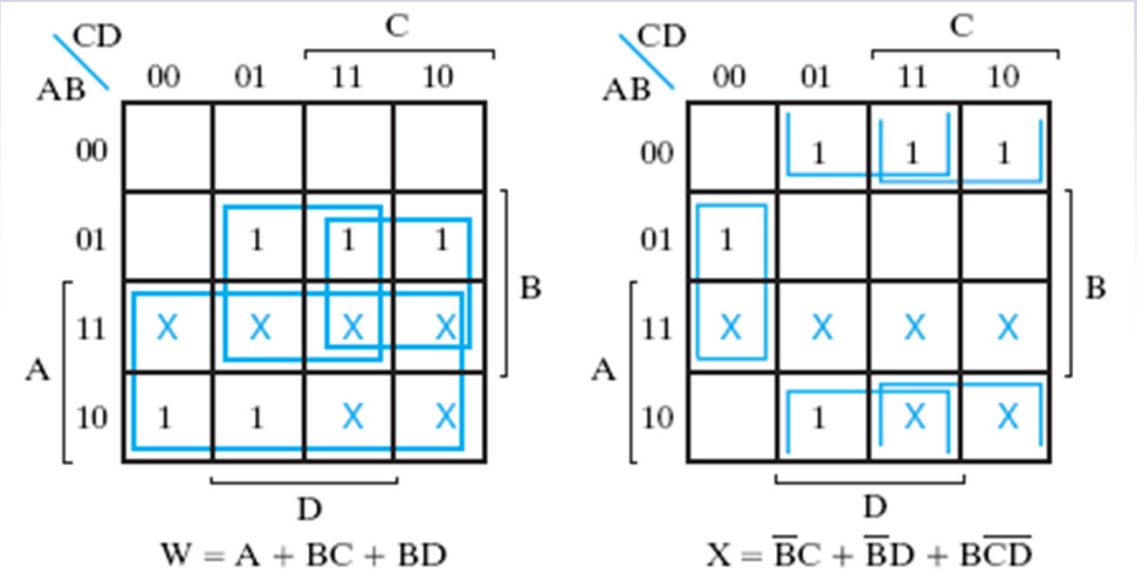
TABLE 3-1
Truth Table for Code Converter Example

Decimal Digit	Input BCD				Output Excess-3			
	A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

BCD-to-excess-3 code converter (4)

- Optimization
  - 4 input variables, hence use 4-variable K-map
  - One output requires one K-map
    - i.e., 4 K-maps are required
    - Reason: one output is represented by one Boolean function
  - Place 1s into the squares of K-map according to the 1s in the truth table
  - Place Xs into the squares of K-map which represent don't care minterms
  - Optimize the four Boolean functions
    - Place rectangles to include all 1s in the K-maps

BCD-to-excess-3 code converter (5)



$W = A + BC + BD$

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

$Y = CD + \overline{CD}$

$Z = \overline{D}$

BCD-to-excess-3 code converter (6)

- Optimization
  - Apply **multiple-level** optimization to reduce the gate input cost
    - Sharing of same terms among outputs

Boolean functions from K-map result

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

Gate-input cost:
26 (incl. inverters)
23 (excl. inverters)

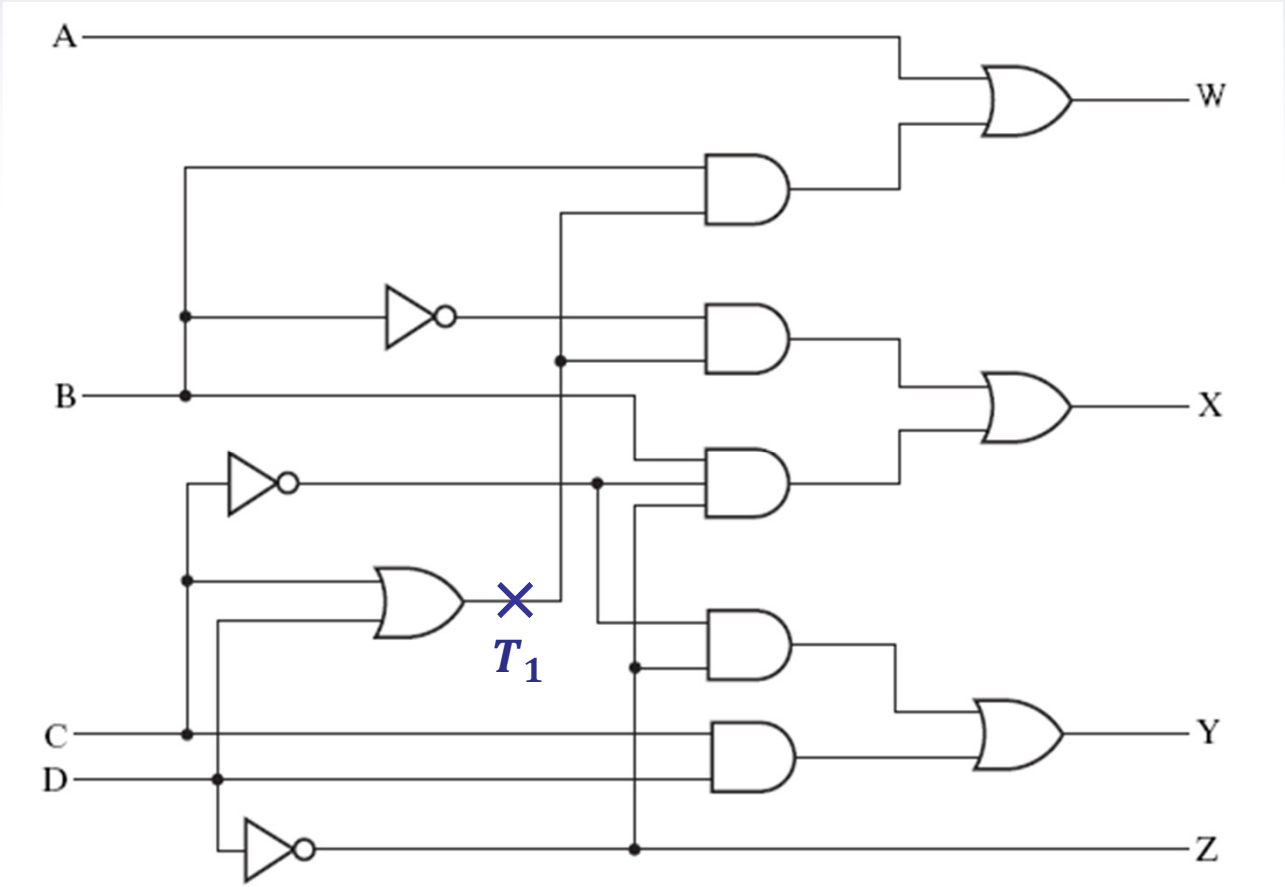
Sharing terms among four outputs

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

Gate-input cost:
22 (incl. inverters)
19 (excl. inverters)

BCD-to-excess-3 code converter (7)

Logic circuit diagram of BCD-to-excess-3 code converter

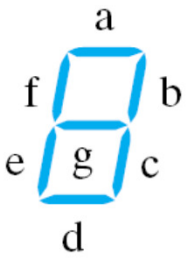


Outline

- Combinational circuit design procedures
- BCD-to-excess-3 code converter
- BCD-to-seven-segment decoder
- Hierarchical design
- Technology mapping and verification

BCD-to-seven-segment decoder (1)

- Specification
  - BCD: binary-coded decimals
  - Seven-segment
    - A display consists of seven LED segments
    - Each LED segment can be turned on by a digital signal
    - E.g., alarm clock, calculator
  - Requirement
    - A combinational circuit which accepts BCD digits as input, and generates appropriate outputs for the segment display to show the decimal digit
    - Inputs: A, B, C and D; outputs: a, b, c, d, e, f and g



(a) Segment designation

(b) Numeric designation for display

BCD-to-seven-segment decoder (2)

- Formulation
  - Derive the truth table for the decoder
  - 4 variables as BCD **input**, $2^4 = 16$ possible input combinations
    - Combinations 1010 through 1111 have no meaning in BCD; can assume them to be don't care minterms
    - Now we use a safer choice – turn off all segments for these unused combinations
  - **Output** 1 or 0 at the segment output (*a* to *g*) based on BCD input, so that decimal digit is displayed correctly
    - Assumption: output logic 1 means turning on the segment, logic 0 means turning off the segment

BCD-to-seven-segment decoder (3)

Truth Table for BCD- to- Seven-Segment Decoder

BCD Input				Seven-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
All other inputs				0	0	0	0	0	0	0

Turn off all LEDs if BCD input is invaild

BCD-to-seven-segment decoder (4)

- Optimization
  - 4 input variables, hence use 4-variable K-maps
  - 7 K-maps are required, one for each output
  - Use normal K-map manipulation method to simplify the output functions

$$a = \overline{A}C + \overline{A}BD + \overline{B}\,\overline{C}\,\overline{D} + A\overline{B}\,\overline{C}$$
$$b = \overline{A}\,\overline{B} + \overline{A}\,\overline{C}\,\overline{D} + \overline{A}CD + A\overline{B}\,\overline{C}$$
$$c = \overline{A}B + \overline{A}D + \overline{B}\,\overline{C}\,\overline{D} + A\overline{B}\,\overline{C}$$
$$d = \overline{A}C\overline{D} + \overline{A}\,\overline{B}C + \overline{B}\,\overline{C}\,\overline{D} + A\overline{B}\,\overline{C} + \overline{A}B\overline{C}D$$
$$e = \overline{A}C\overline{D} + \overline{B}\,\overline{C}\,\overline{D}$$
$$f = \overline{A}B\overline{C} + \overline{A}\,\overline{C}\,\overline{D} + \overline{A}B\overline{D} + A\overline{B}\,\overline{C}$$
$$g = \overline{A}C\overline{D} + \overline{A}\,\overline{B}C + \overline{A}B\overline{C} + A\overline{B}\,\overline{C}$$

BCD-to-seven-segment decoder (5)

- Optimization
  - Gates required: 27 AND gates and 7 OR gates
    - Number of product terms = number of AND gates required
    - Each output function requires an OR gate
  - Possible to reduce the number of required gates
    - Sharing product terms
    - Reduction of 13 AND gates

Shared product terms	Occurrence	# gates reduced
$A\,\overline{B}\,\overline{C}$	6 (a, b, c, d, f, g)	5
$\overline{B}\,\overline{C}\,\overline{D}$	4 (a, c, d, e)	3
$\overline{A}\,C\,\overline{D}$	3 (d, e, g)	2
$\overline{A}\,\overline{C}\,\overline{D}$	2 (b, f)	1
$\overline{A}\,\overline{B}\,C$	2 (d, g)	1
$\overline{A}\,B\,\overline{C}$	2 (f, g)	1

Outline

- Combinational circuit design procedures
- BCD-to-excess-3 code converter
- BCD-to-seven-segment decoder
- Hierarchical design
- Technology mapping and verification

Hierarchical design

- Divide-and-conquer approach
- Use a *hierarchy* of symbols to represent the circuit
  - Entire circuit broken down into pieces called *blocks*
  - At lower level, blocks are implemented using primitive logic gates
  - Blocks are interconnected, to form a circuit which satisfies the problem specification

- Suitable for problem that
  - Has large number of inputs;
    - Too many inputs → using K-map approach is difficult
  - Can be broken down into smaller pieces

4-bit equality comparator (1)

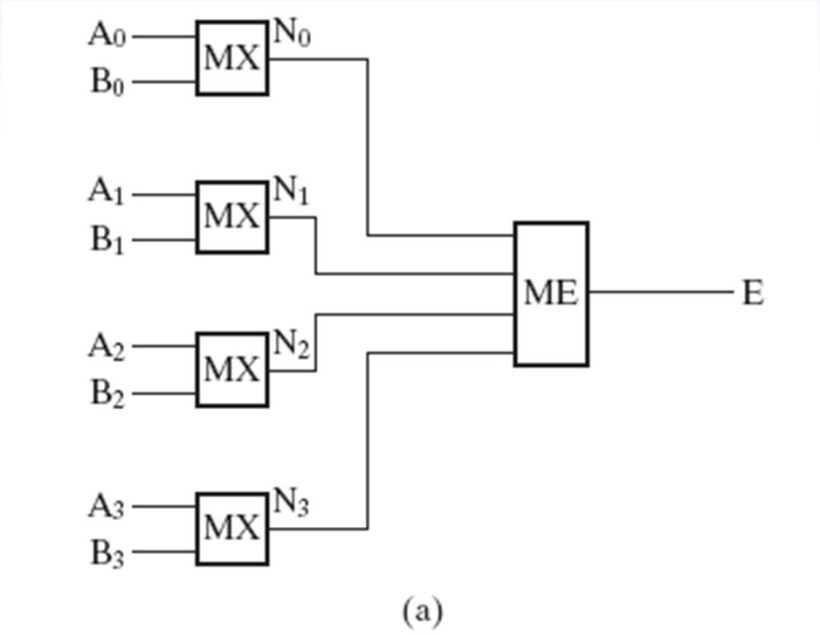
- Specification
  - Equality comparator compares two *binary vectors* (i.e., binary number)
  - Returns 1 if the two input vectors are equal, 0 otherwise
  - **Inputs**: A(3:0) and B(3:0)
    - i.e., two 4-bit numbers
    - $A_3A_2A_1A_0$ and $B_3B_2B_1B_0$
  - **Output**: E (one bit), the comparison result

- Example
  - If A = 1000 and B = 1000, then E = 1
  - If A = 1001 and B = 1000, then E = 0

4-bit equality comparator (2)

- Formulation
  - Possible to break down the problem…
    - If each bit position of A and B are equal in ALL four positions, then E = 1
    - Compare a bit from A with the corresponding bit from B
    - Repeat the comparison for every bit positions
    - Combine the result and output E
  - Not necessary to construct truth tables
    - Otherwise, we have 8 inputs
      - (2^8 = 256 input combinations!)

4-bit equality comparator (3)



(a)

- MX: one-bit comparison circuit
  - Input: A_i, B_i
  - Output: N_i
- ME: combines N_i and gives E

4-bit equality comparator (4)

- Optimization, two approaches
  - Approach 1
    - In each MX circuit, $N_i = 1$ if A_i and B_i are different
    - In ME circuit, output E = 0 if one of the N_i is 1
  - Approach 2
    - In each MX circuit, $N_i = 1$ if A_i and B_i are the same
    - In ME circuit, output E = 1 if all N_i is 1

Approach 1

$$N_i = \overline{A_i}\, B_i + A_i \overline{B_i}$$

$$E = \overline{N_0 + N_1 + N_2 + N_3}$$

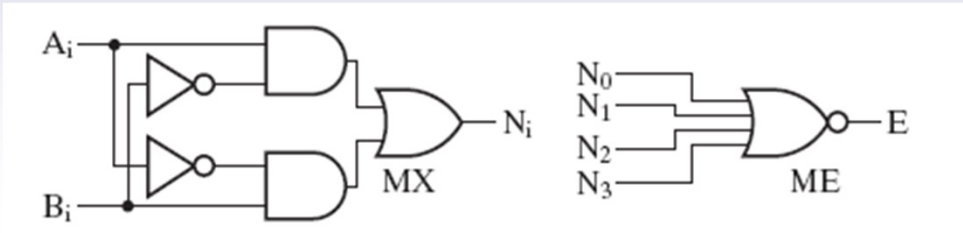
Approach 2

$$N_i = A_i B_i + \overline{A_i}\ \overline{B_i}$$

$$E = N_0 N_1 N_2 N_3$$

4-bit equality comparator (5)

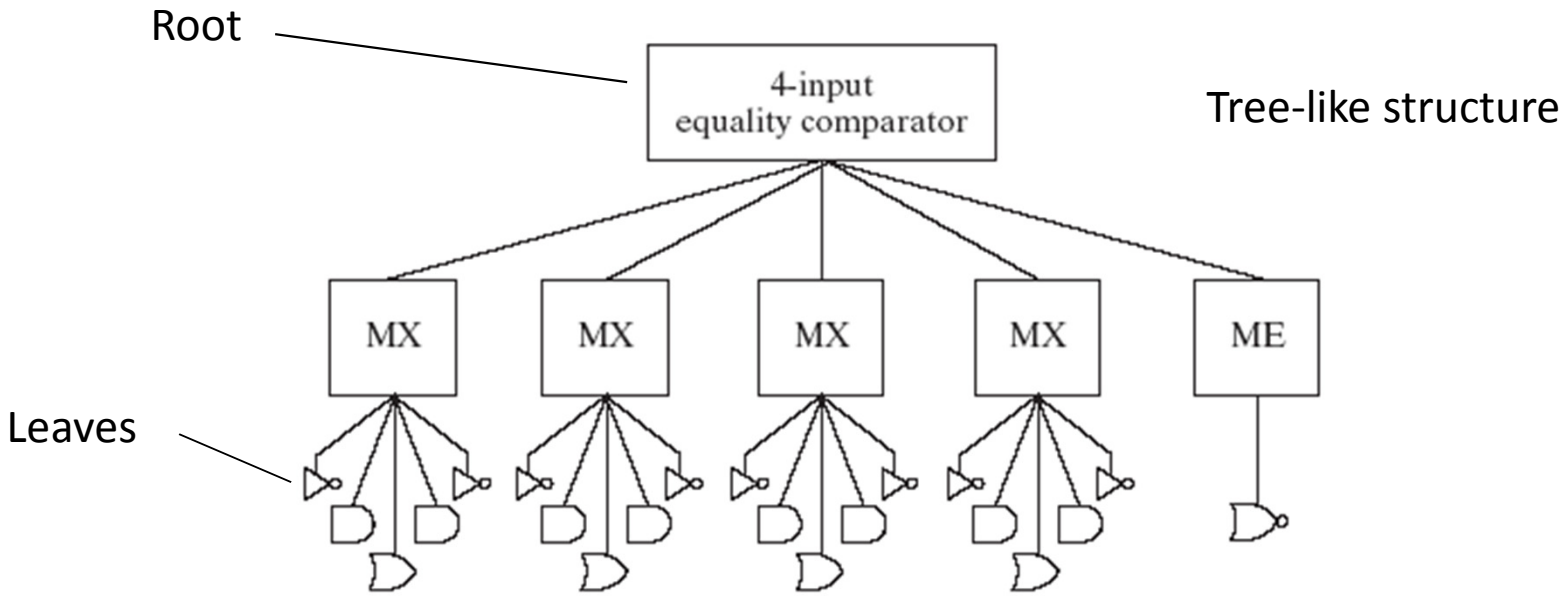
Logic circuit diagram using approach 1



MX circuit for one bit comparison

ME circuit for output E

Structure of hierarchy



Hierarchical design

- Simpler circuit diagram can be drawn
  - An MX circuit block represents a circuit that uses 5 gates
  - Reduces the number of symbols in the diagram

- Allows reuse of blocks
  - In the previous example, the MX circuit is reused four times

Outline

- Combinational circuit design procedures
- BCD-to-excess-3 code converter
- BCD-to-seven-segment decoder
- Hierarchical design
- Technology mapping and verification

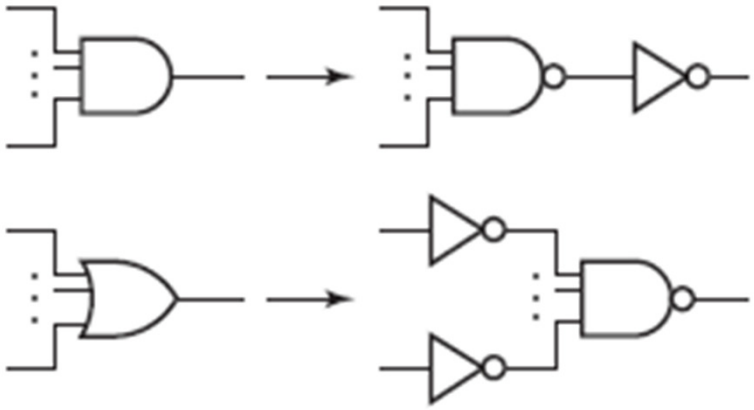
Technology mapping (1)

- Mapping of a logic circuit diagram to a description that uses the available implementation technology
  - NAND or NOR gates are mostly available

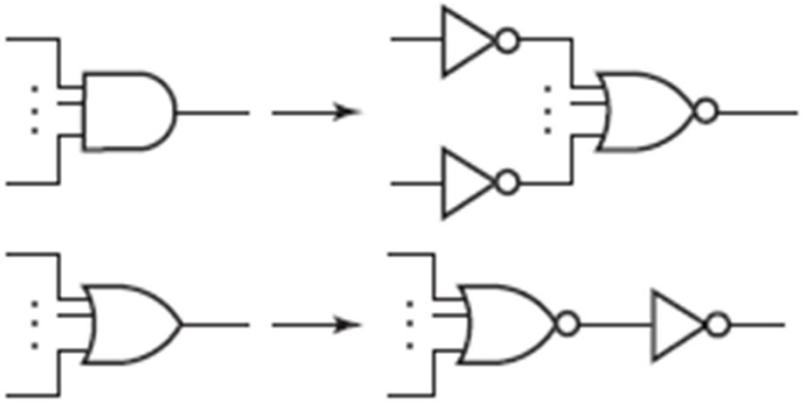
- Map AND, OR and NOT gates to either NAND, or NOR gate description
  - Fixed number of input (n) in NAND or NOR gates
  - If n = 1, the NAND gate is represented by an inverter (NOT gate)

Technology mapping (2)

- With an optimized logic diagram with AND, OR gates and inverters, convert the logic functions to NAND logic (or NOR logic)
- Procedures
  - (1) Replace AND and OR gate with NAND (NOR) gate and inverter



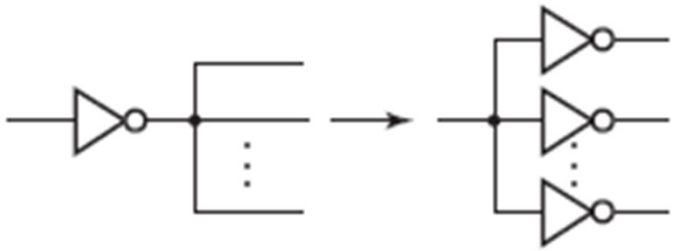
(a) Mapping to NAND gates



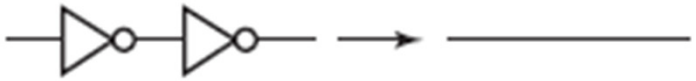
(b) Mapping to NOR gates

Technology mapping (3)

- Procedures
  - (2) Push inverters through a dot
  - (3) Cancel inverter pairs
    - So that between two NAND gates there is at most one inverter



(c) Pushing an inverter through a "dot"



(d) Cancelling inverter pairs

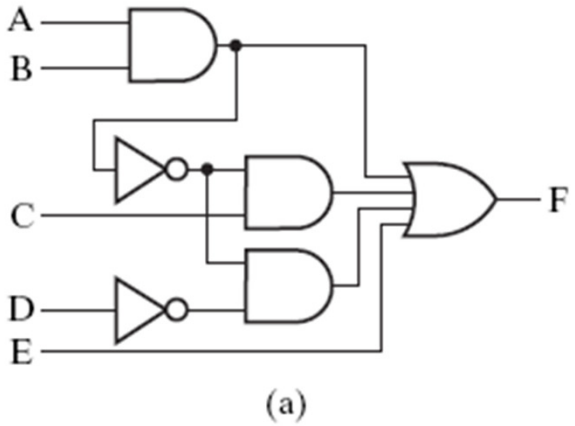
Technology mapping (4)

- Example: implement F with NAND gates

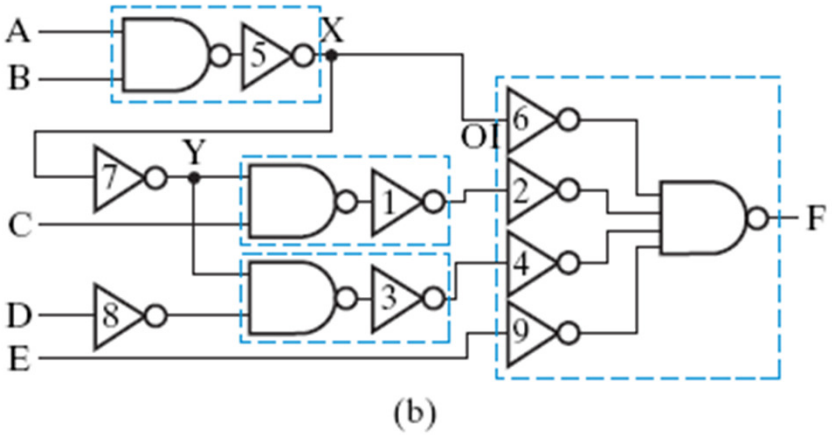
$$F = AB + \overline{AB}C + \overline{AB}\,\overline{D} + E$$

  - Step 1: Replace AND and OR gates with NAND gates and inverters (figure b)

(a) Logic diagram of F using AND gates, OR gates and inverters

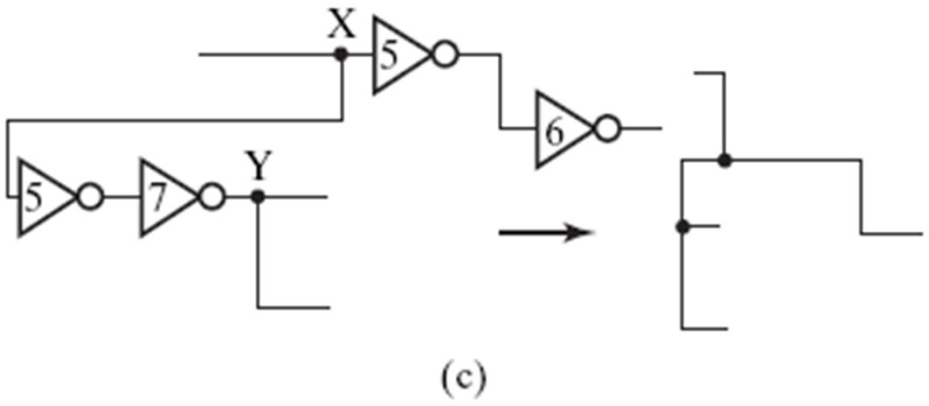


(b) Logic diagram of F after step 1

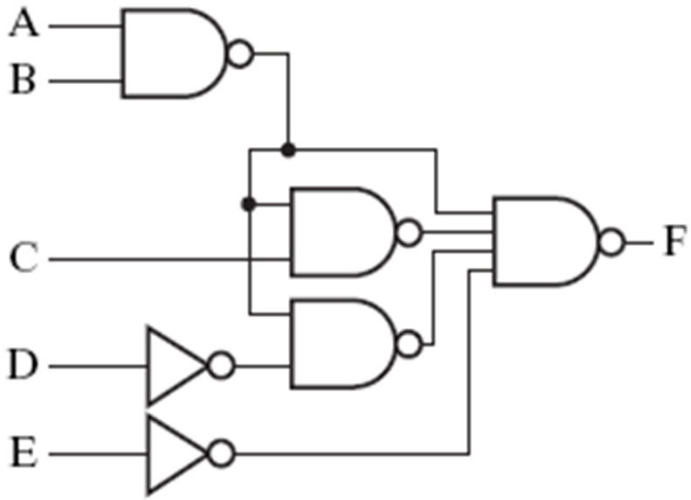


Technology mapping (5)

- Step 2: Push inverter 5 through a dot next to inverters 6 and 7 (figure c)
- Step 3: Cancel inverter pairs (5, 6) and (5, 7); cancel inverter pairs (1, 2) and (3, 4) (figure d)



(c)



(d)

Verification

- Verification of circuit function
  - Determine whether a given circuit implements its specified function by logic analysis
  - Important in preventing incorrect circuit design from being manufactured and used
- Two approaches
  - Manual logic analysis
  - Computer simulation-based logic analysis

Manual logic analysis

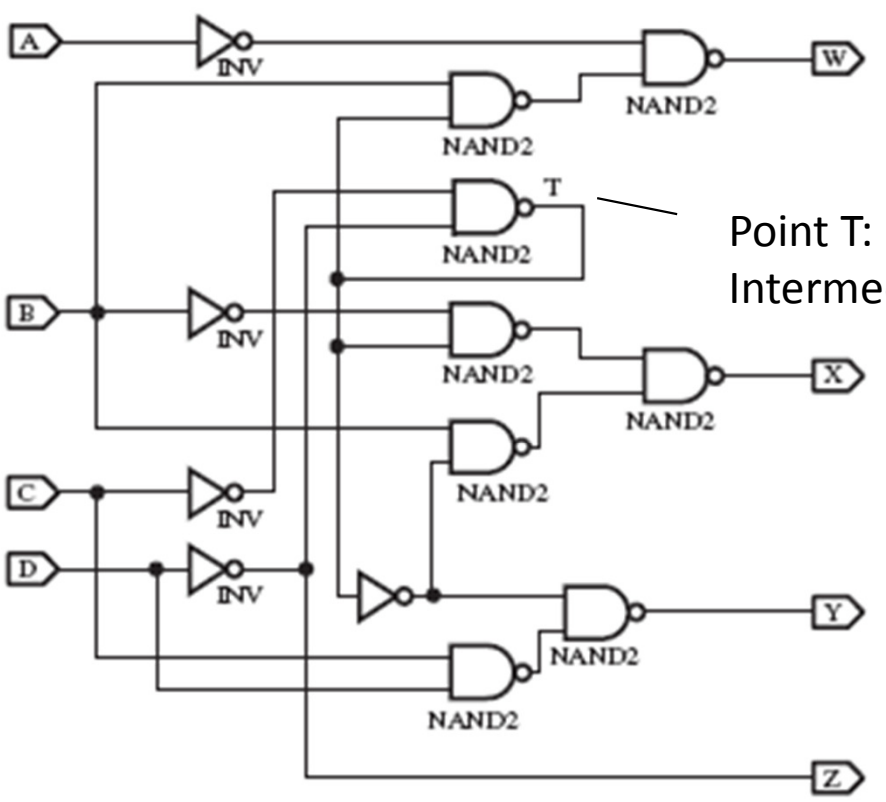
- Based on a logic circuit diagram in NAND gate implementation
  - (1) Find the Boolean equations for each output of the circuit
  - (2) Find the truth table for the circuit
  - (3) Verify the derived truth table, compare with the original truth table specification

BCD-to-excess-3 code converter verification (1)

(a) Original truth table specification; (b) final NAND-mapped circuit implementation

Input BCD				Output 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

(a)



Point T:
Intermediate result

(b)

BCD-to-excess-3 code converter verification (2)

- Determine equations for outputs W, X, Y and Z
  - Intermediate point T is used
  - Ease the verification process
- Substitute expression for T in equations for W, X and Y

NAND operation:

$$\overline{XY} = \overline{X} + \overline{Y}$$

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

$$W = \overline{\overline{A}\left(\overline{TB}\right)} = A + BT$$

$$X = \overline{\left(\overline{\overline{B}T}\right)\left(\overline{B\overline{T}}\right)} = \overline{B}T + B\overline{T}$$

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

$$Z = \overline{D}$$

$$W = A + B(C + D) = A + BC + BD$$

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

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

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

$$Z = \overline{D}$$

BCD-to-excess-3 code converter verification (3)

- Derive the truth table, and verify with the original truth table
  - Determine output W, X, Y and Z under each possible input combinations
  - Place 1s in the truth table which map with the product term in output equations W, X, Y and Z

Partially filled truth table, only the following product terms are placed:

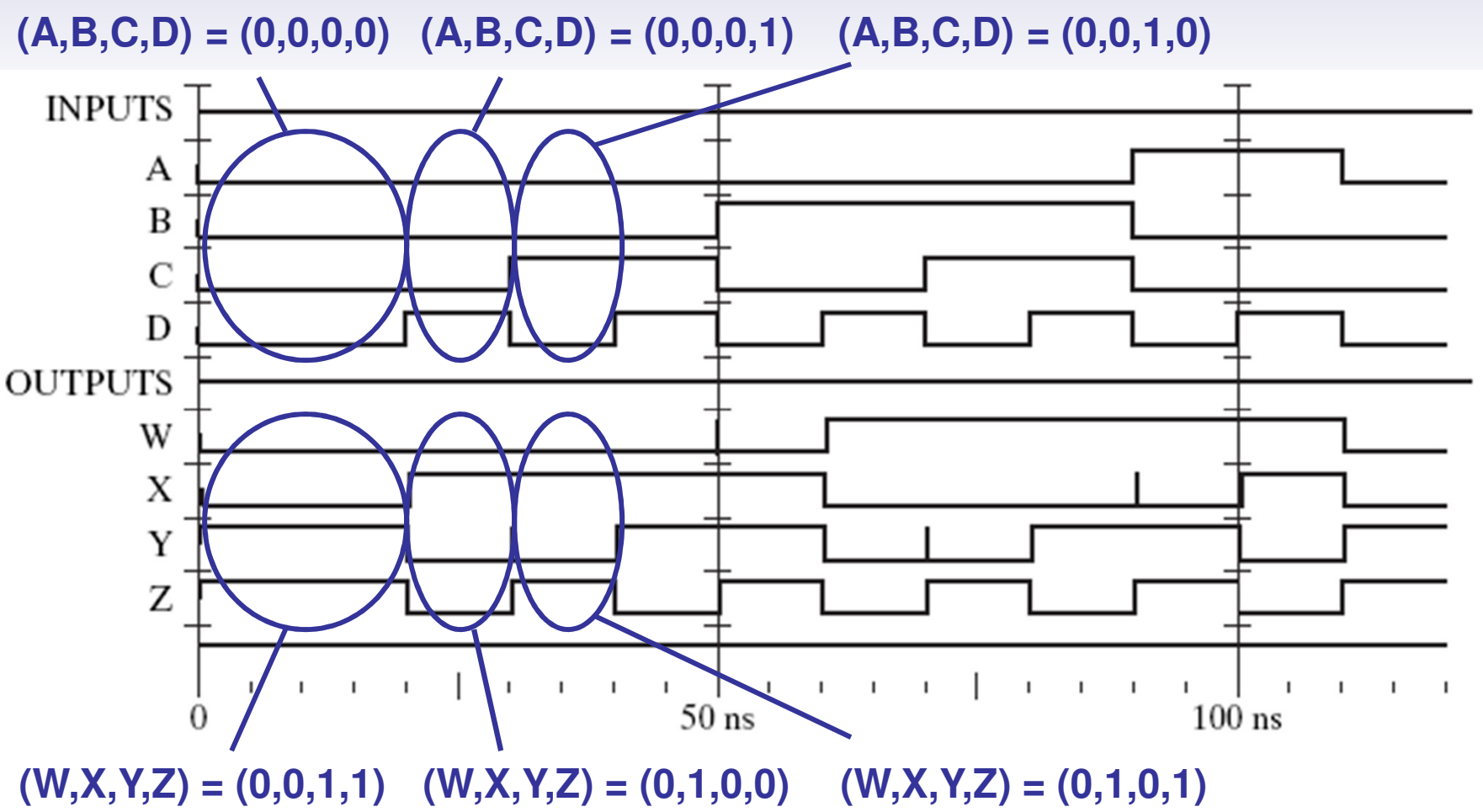
$A, BC, BD, \overline{B}C, CD, \overline{D}$

Input BCD				Output Excess-3			
A	B	C	D	W	X	Y	Z
0	0	0	0				1
0	0	0	1				
0	0	1	0		1		1
0	0	1	1		1	1	
0	1	0	0				1
0	1	0	1	1			
0	1	1	0	1			1
0	1	1	1	1		1	
1	0	0	0	1			1
1	0	0	1	1			

Simulation-based logic analysis (1)

- Output of a circuit generated by computer through simulation
  - Suitable for circuit with large number of input variables
- Verification process
  - **Input circuit** implementation to simulator
  - **Enter input combinations** in the form of a waveform (INPUT section)
  - **Simulator gives** corresponding **output** waveform (OUTPUT section)
  - **Verify** whether output match with original truth table

Simulation-based logic analysis (2)



Summary and acknowledgment

- A five-step design procedure for combinational circuit was discussed and illustrated with examples
- Concept of design hierarchy was introduced
- Optimization step applies two-level and multiple-level optimization
- Technology mapping converts the circuit into one which uses NAND or NOR gates only
- Verification through logic analysis assures the final circuit satisfies the initial specification
- Reference
  - Logic and Computer Design Fundamental (5/e)