Logic and Computer Design Fundamentals

Chapter 3 – Combinational Logic Design

Part 1 – Implementation Technology and Logic Design

Yueming Wang (王跃明)

ymingwang@zju.edu.cn

2018

College of Computer Science, Zhejiang University

Qiushi Academy for Advanced Studies, Zhejiang University

Overview

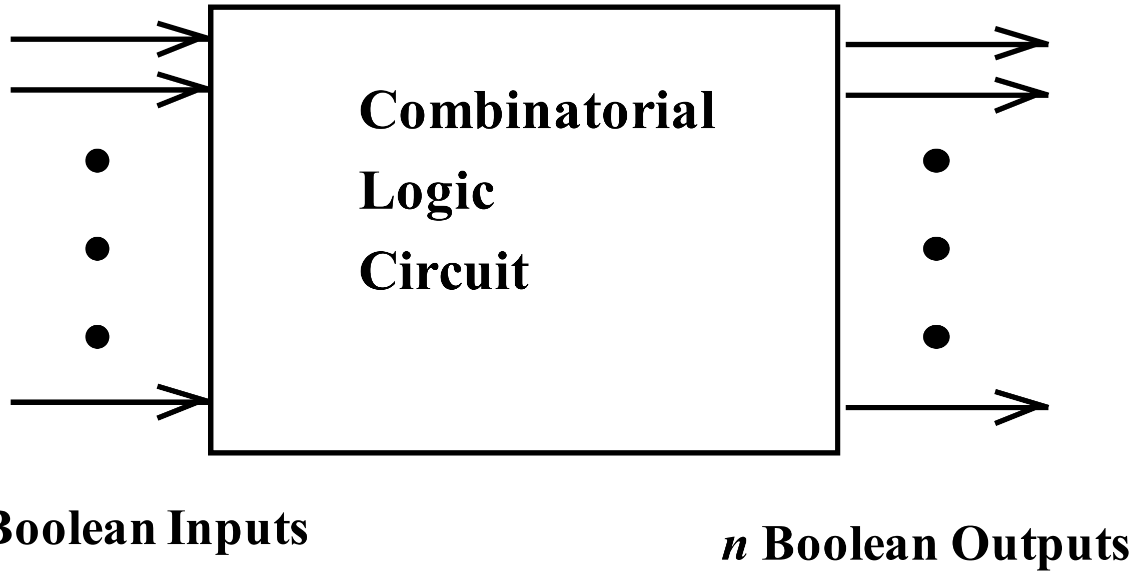
- Part 1 – Implementation Technology and Logic Design
  - Some design topics and concepts
  - Design Procedure
    - Specification
    - Formulation
    - Optimization
    - Technology Mapping
  - Beginning Hierarchical Design
  - Technology Mapping
  - Verification
    - Manual
    - Simulation

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

Combinational Circuits

- 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:



Integrated Circuits

- Integrated circuit (informally, a “chip”) is a semiconductor crystal (most often silicon) containing the electronic components for the digital gates and storage elements which are interconnected on the chip.
- Terminology - Levels of chip integration
  - *SSI* (*small-scale integrated*) - fewer than 10 gates
  - *MSI* (*medium-scale integrated*) - 10 to 100 gates
  - *LSI* (*large-scale integrated*) - 100 to thousands of gates
  - *VLSI* (*very large-scale integrated*) - thousands to 100s of millions of gates

Technology Parameters

- **Specific gate implementation technologies are characterized by the following parameters:**
  - ***Fan-in*** **– the number of inputs available on a gate**
  - ***Fan-out*** **– the number of standard loads driven by a gate output**
  - ***Logic Levels*** **– the signal value ranges for 1 and 0 on the inputs and 1 and 0 on the outputs (see Figure 1-1)**
  - ***Noise Margin*** **– the maximum external noise voltage superimposed on a normal input value that will not cause an undesirable change in the circuit output**
  - ***Cost for a gate*** **- a measure of the contribution by the gate to the cost of the integrated circuit**
  - ***Propagation Delay*** **– The time required for a change in the value of a signal to propagate from an input to an output**
  - ***Power Dissipation*** **– the amount of power drawn from the power supply and consumed by the gate**

Fan-out

- **Fan-out can be defined in terms of a standard load**
  - **Example: 1 standard load equals the load contributed by the input of 1 inverter.**
  - ***Transition time*** **-the time required for the gate output to change from H to L, t_{HL}, or from L to H, t_{LH}**
  - **The *maximum fan-out* that can be driven by a gate is the number of standard loads the gate can drive without exceeding its specified *maximum transition time***

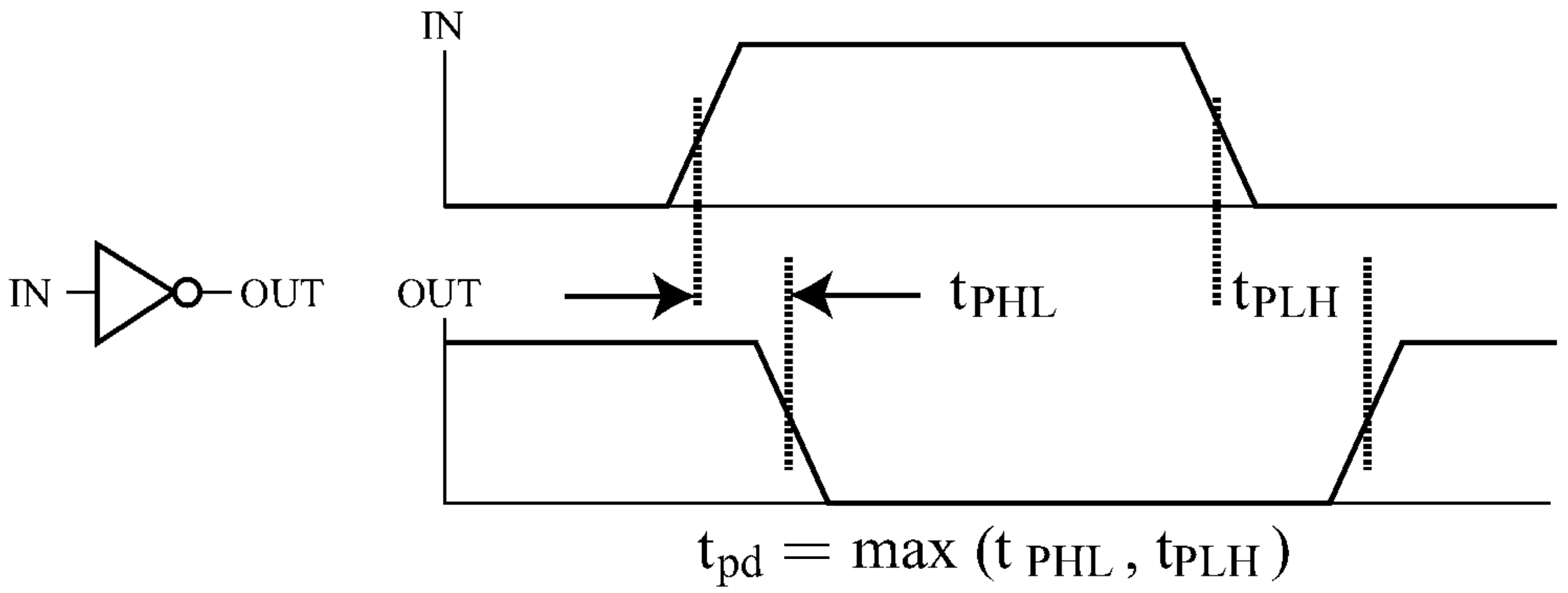
Cost

- In an integrated circuit:
  - The cost of a gate is proportional to the <u>chip area</u> occupied by the gate
  - The gate area is roughly proportional to the <u>number and size of the transistors</u> and the <u>amount of wiring</u> connecting them
  - Ignoring the wiring area, the gate area is roughly proportional to the <u>gate input count</u>
  - So gate input count is a rough measure of gate cost
- If the actual chip layout area occupied by the gate is known, it is a far more accurate measure

Propagation Delay

- *Propagation delay* is the time for a change on an input of a gate to propagate to the output.
- Delay is usually measured at the 50% point with respect to the H and L output voltage levels.
- High-to-low (t_{PHL}) and low-to-high (t_{PLH}) output signal changes may have different propagation delays.
- High-to-low (HL) and low-to-high (LH) transitions are defined with respect to the output, <u>not</u> the input.
- An HL input transition causes:
  - an LH output transition if the gate inverts and
  - an HL output transition if the gate does not invert.

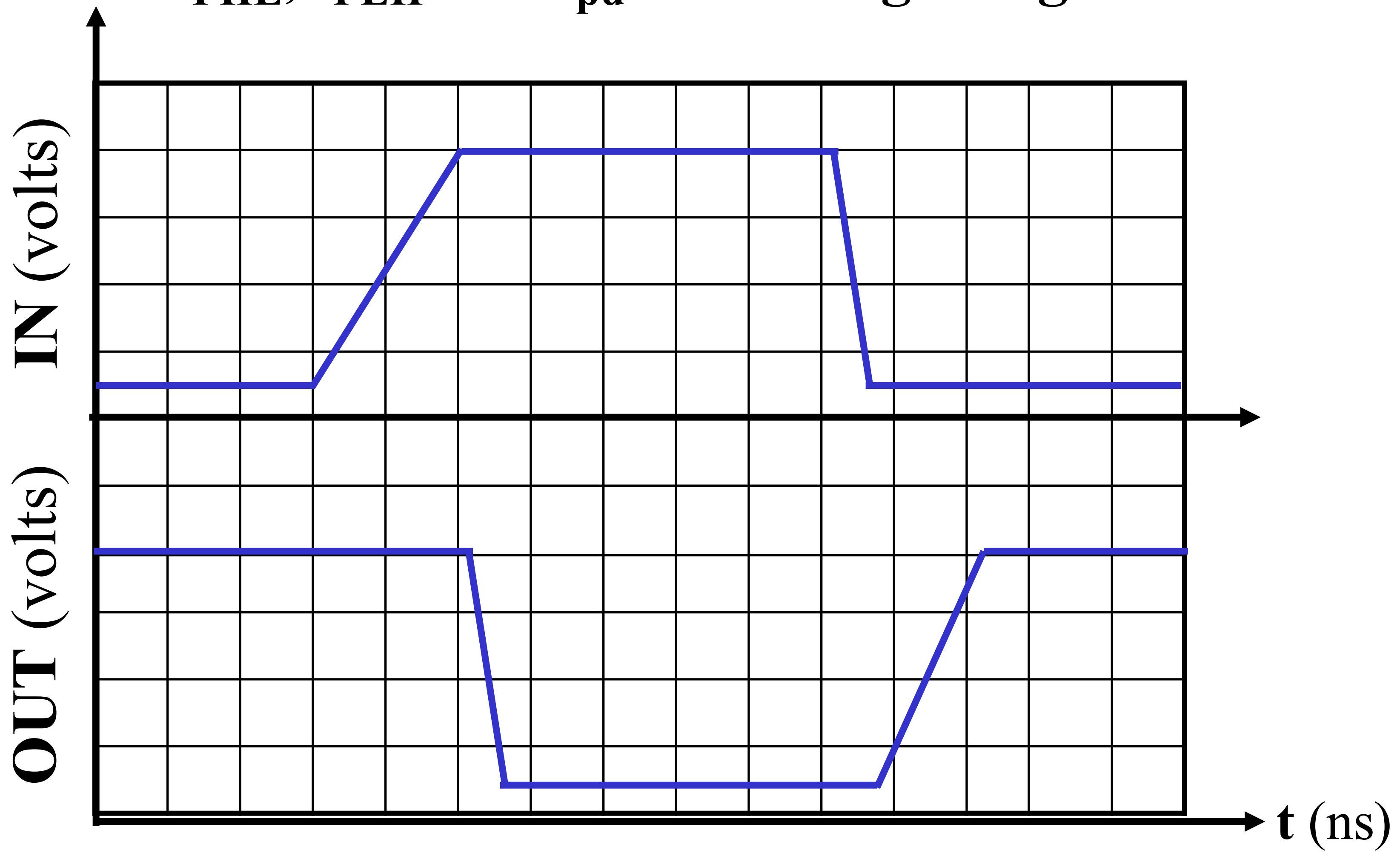
Propagation Delay (continued)



- **Propagation delays measured at the midpoint between the L and H values**

Propagation Delay Example

- Find t_{PHL}, t_{PLH} and t_{pd} for the signals given

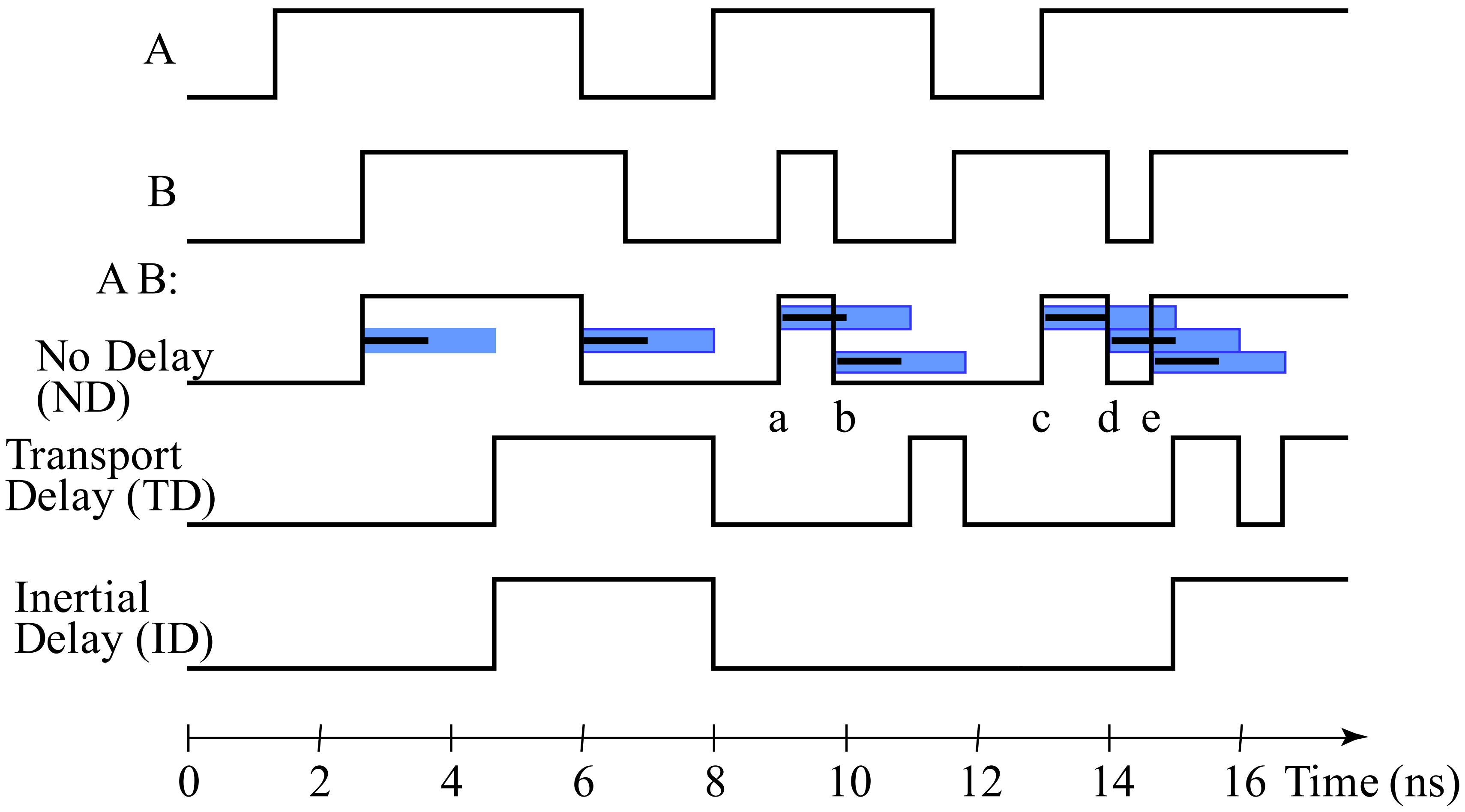


1.0 ns per division

Delay Models

- ***Transport delay*** **- a change in the output in response to a change on the inputs occurs after a fixed specified delay**
- ***Inertial delay*** **- similar to transport delay, except that if the input changes such that the output is to change twice in a time interval less than the** ***rejection time*****, the output changes do not occur. Models typical electronic circuit behavior, namely, rejects narrow "pulses" on the outputs**

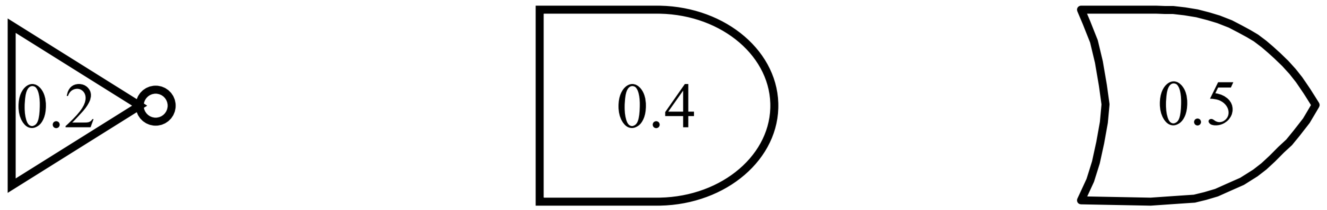
Delay Model Example



Propagation Delay = 2.0 ns Rejection Time = 1 .0 ns

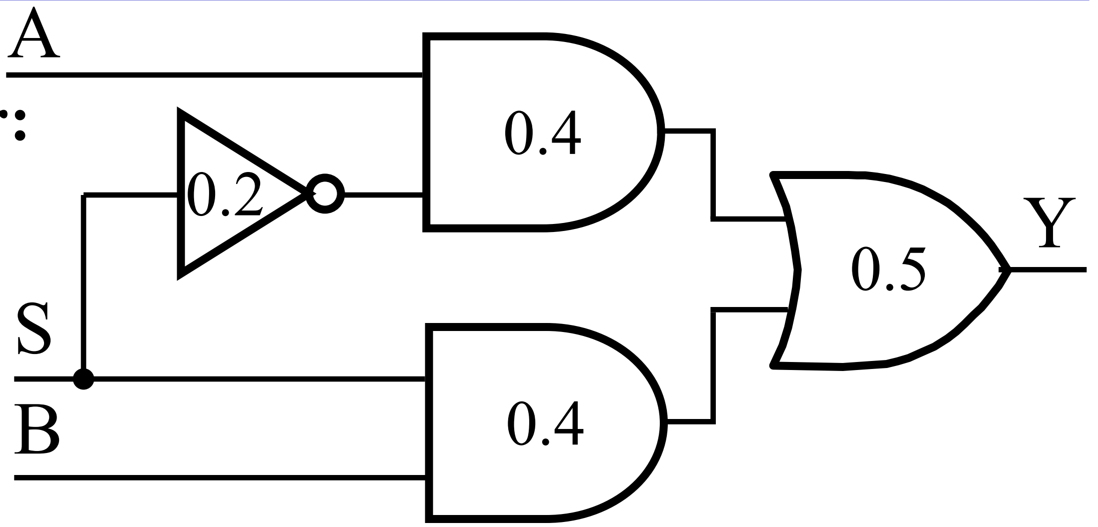
Circuit Delay

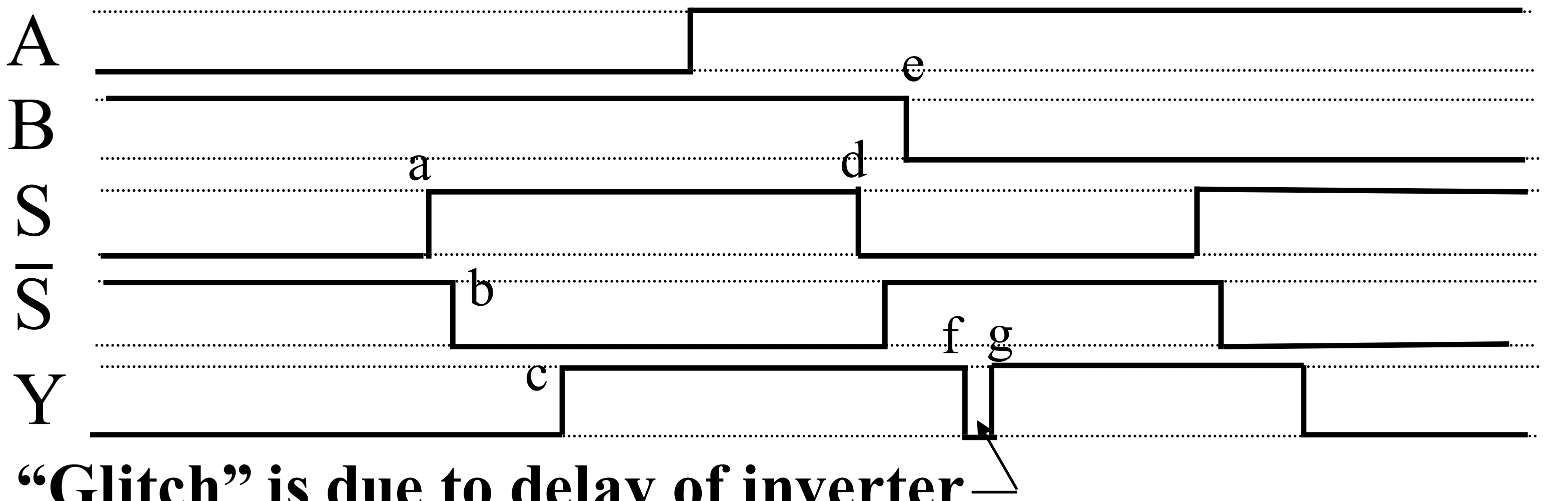
- Suppose gates with delay *n* ns are represented for *n* = 0.2 ns, *n* = 0.4 ns, *n* = 0.5 ns, respectively:



Circuit Delay

- Consider a simple 2-input multiplexer:
- With function:
  - Y = A for S = 0
  - Y = B for S = 1





- "Glitch" is due to delay of inverter

Fan-out and Delay

- The fan-out loading a gate's output affects the gate's propagation delay
- Example:
  - One realistic equation for t_{pd} for a NAND gate with 4 inputs is:

    $$t_{pd} = 0.07 + 0.021 \text{ SL ns}$$
  - SL is the number of standard loads the gate is driving, i. e., its fan-out in standard loads
  - For SL = 4.5, t_{pd} = 0.165 ns
- If this effect is considered, the delay of a gate in a circuit takes on different values depending on the circuit load on its output.

Cost/Performance Tradeoffs

- **Gate-Level Example:**
  - **NAND gate G with 20 standard loads on its output has a delay of 0.45 ns and has a normalized cost of 2.0**
  - **A buffer H has a normalized cost of 1.5. The NAND gate driving the buffer with 20 standard loads gives a total delay of 0.33 ns**
  - **In which if the following cases should the buffer be added?**
    1. **The cost of this portion of the circuit cannot be more than 2.5**
    2. **The delay of this portion of the circuit cannot be more than 0.40 ns**
    3. **The delay of this portion of the circuit must be less than 0.40 ns and the cost less than 3.0**
- **Tradeoffs can also be accomplished much higher in the design hierarchy**
- **Constraints on cost and performance have a major role in making tradeoffs**

Design Procedure

1. Specification
   - Write a specification for the circuit if one is not already available
2. Formulation
   - 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
3. Optimization
   - Apply 2-level and multiple-level optimization
   - Draw a logic diagram or provide a netlist for the resulting circuit using ANDs, ORs, and inverters

Design Procedure

4. Technology Mapping
   - Map the logic diagram or netlist to the implementation technology selected
5. Verification
   - Verify the correctness of the final design manually or using simulation

Design Example

1. Specification
   - <u>BCD to Excess-3 code converter</u>
   - 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 1 0 0

Design Example (continued)

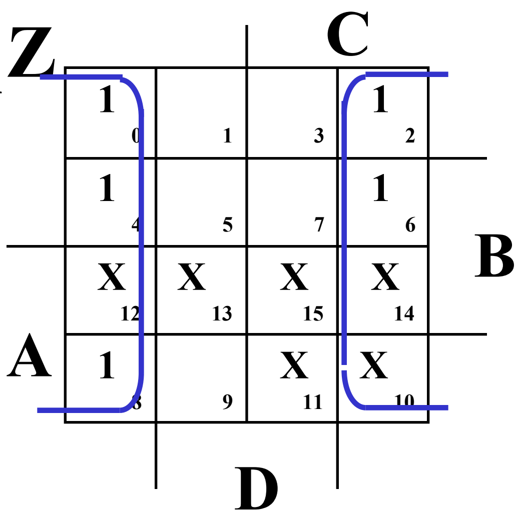
3. **Optimization**
   a. **2-level using K-maps**

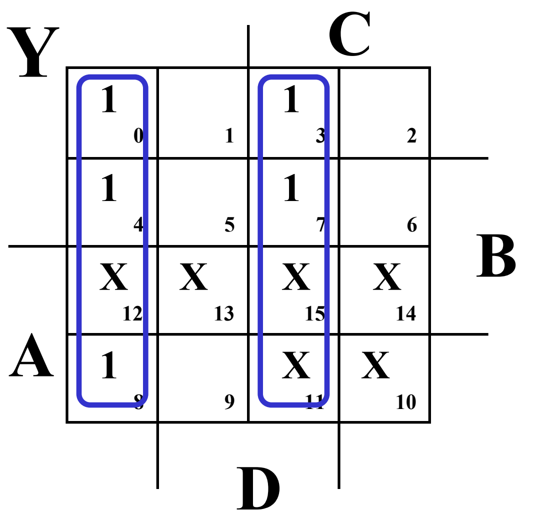
$W = A + BC + BD$

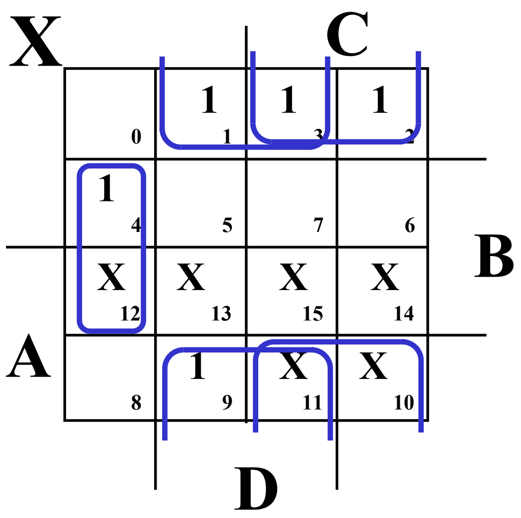
$X = \overline{B}C + \overline{B}D + B\overline{C}\,\overline{D}$

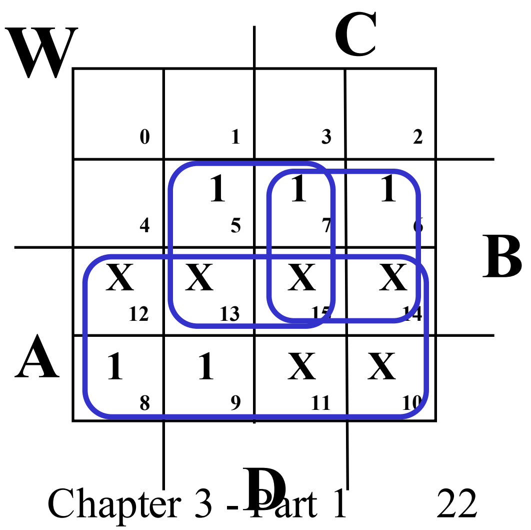
$Y = CD + \overline{C}\,\overline{D}$

$Z = \overline{D}$









Design Example (continued)

3. **Optimization (continued)**

   b. **Multiple-level using transformations**

   $W = A + BC + BD$

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

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

   $Z = \overline{D}$ $\qquad G = 7 + 10 + 6 + 0 = 23$

- **Perform extraction, finding factor:**

  $T_1 = C + D$

  $W = A + BT_1$

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

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

  $Z = \overline{D}$ $\qquad G = 2 + 4 + 7 + 6 + 0 = 19$

Design Example (continued)

3. Optimization (continued)

b. **Multiple-level using transformations**

$T_1 = C + D$

$W = A + BT_1$

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

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

$Z = \overline{D}$ $\qquad G = 19$

- An additional extraction not shown in the text since it uses a Boolean transformation: ($\overline{C}\overline{D} = \overline{C + D} = \overline{T}_1$):

$W = A + BT_1$

$X = \overline{B}T_1 + B\overline{T}_1$

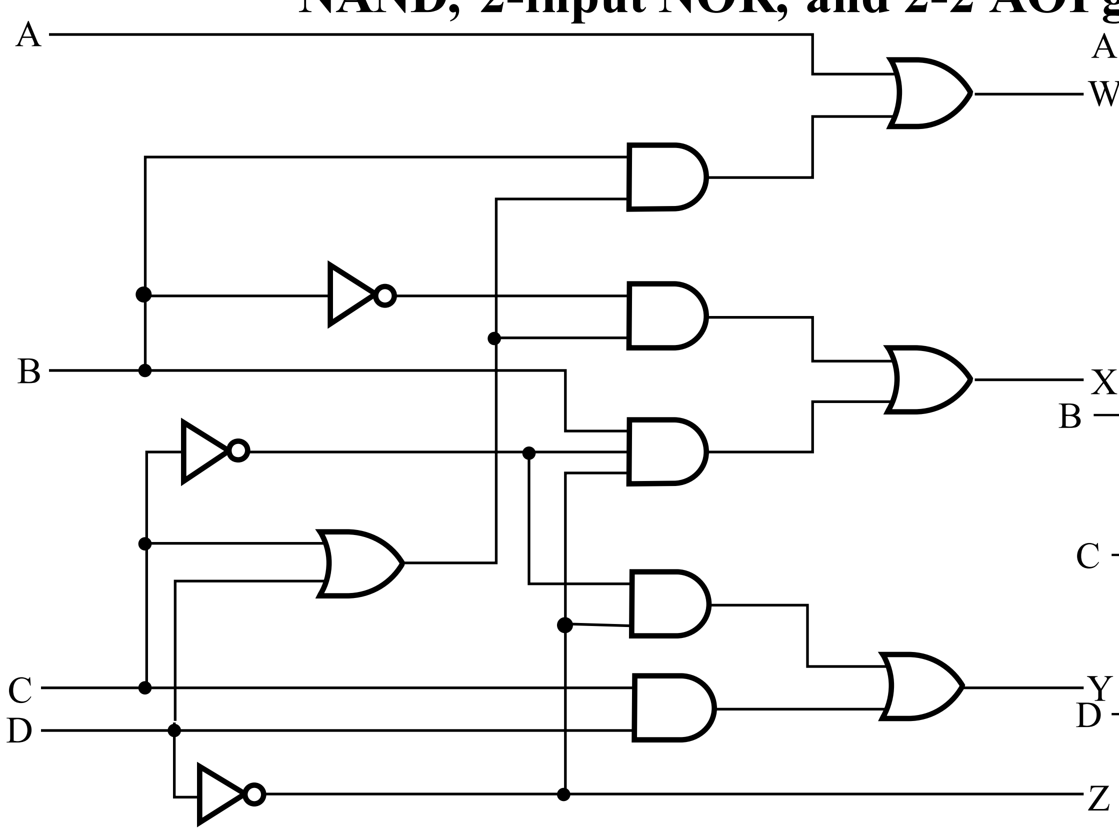
$Y = CD + \overline{T}_1$

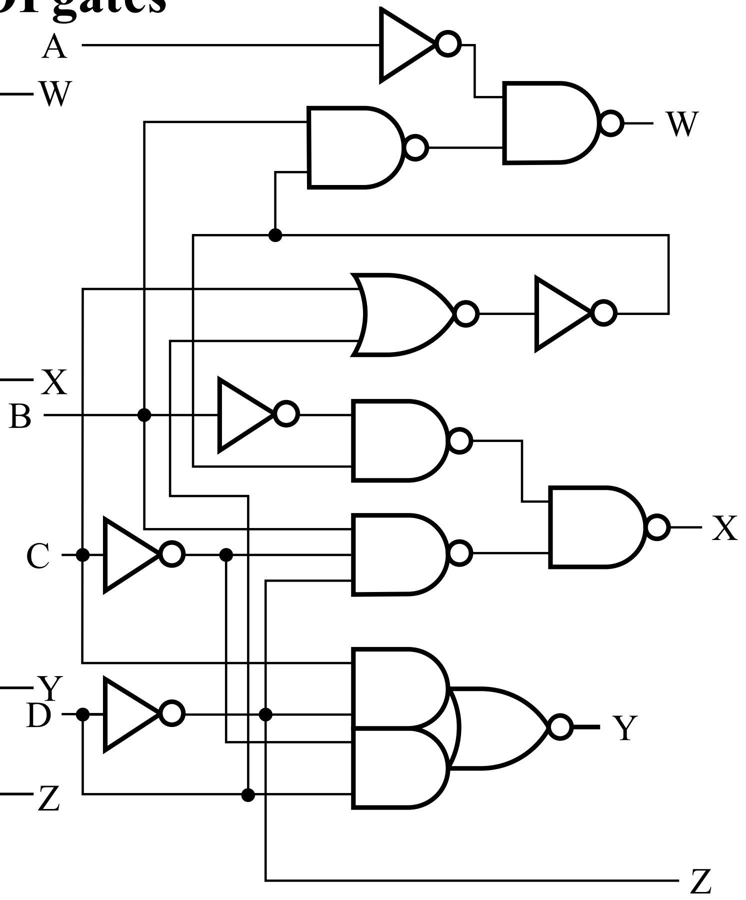
$Z = \overline{D}$ $\qquad G = 2 + 4 + 6 + 4 + 0 = 16!$

Design Example (continued)

4. Technology Mapping

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

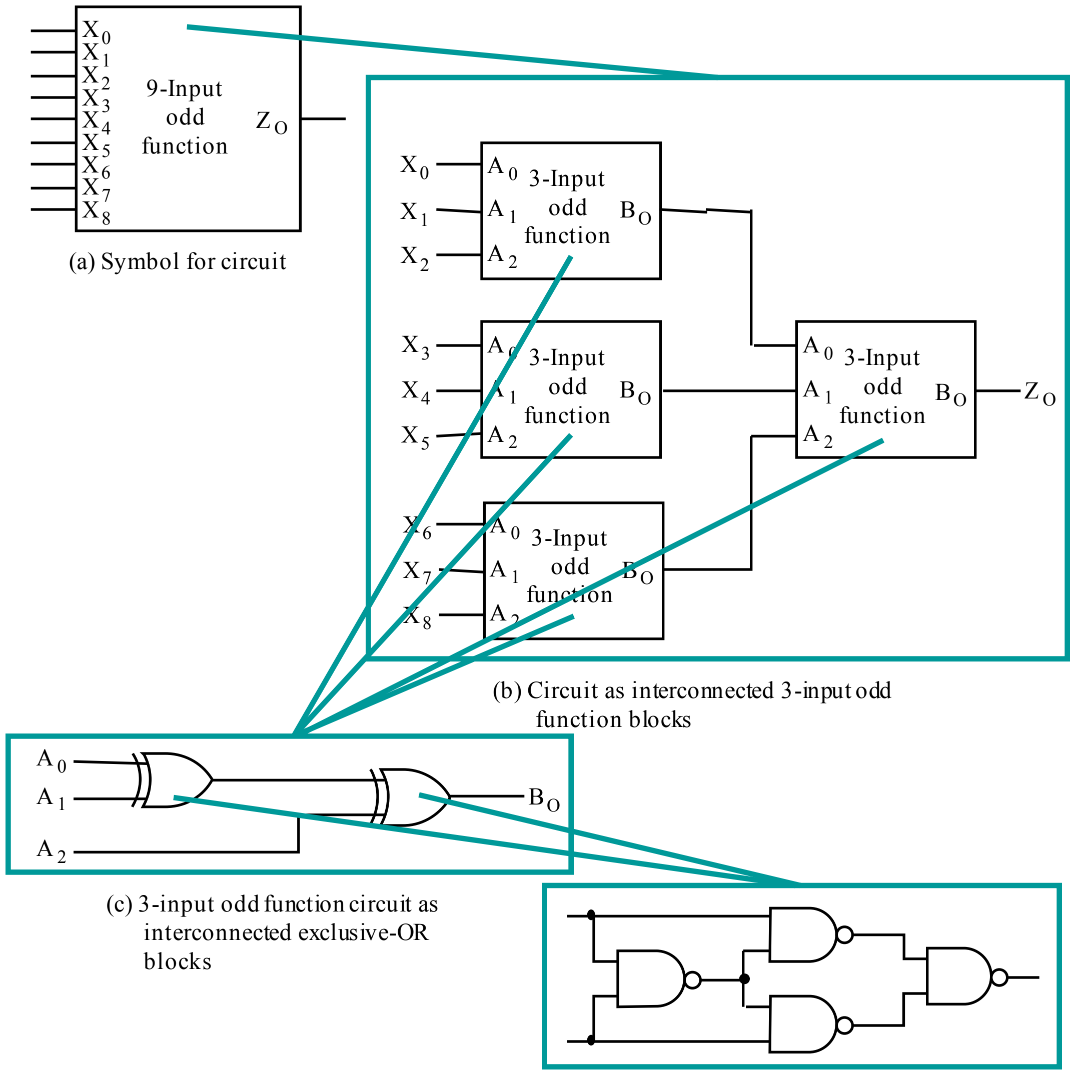




Beginning Hierarchical Design

- 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*
- Example: (next slide)

Hierarchy for Parity Tree Example



X0 X1 X2 X3 X4 X5 X6 X7 X8
9-Input odd function
ZO
(a) Symbol for circuit
A0 A1 A2
3-Input odd function
BO
(b) Circuit as interconnected 3-input odd function blocks
(c) 3-input odd function circuit as interconnected exclusive-OR blocks
(d) Exclusive-OR block as interconnected NANDs

- ➢ 9-input parity tree
- Top Level: 9 inputs, one output
- 2nd Level: Four 3-bit odd parity trees in two levels
- 3rd Level: Two 2-bit exclusive-OR functions
- Primitives: Four 2-input NAND gates
- 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**
- **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 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.**

Technology Mapping

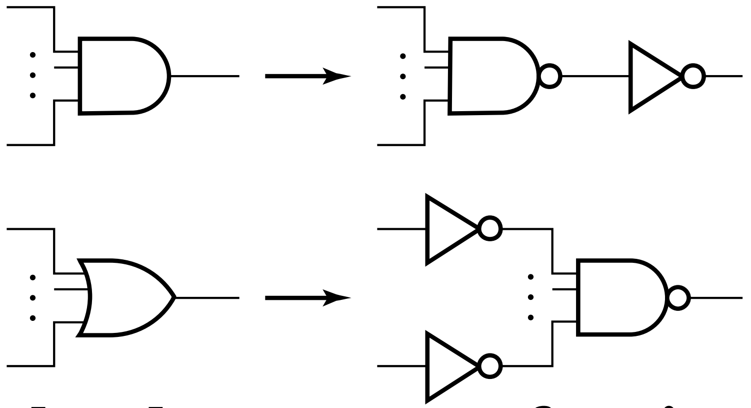
- **Mapping Procedures**
  - **To NAND gates**
  - **To NOR gates**
  - **Mapping to multiple types of logic blocks in covered in the reading supplement: Advanced Technology Mapping.**

Mapping to NAND gates

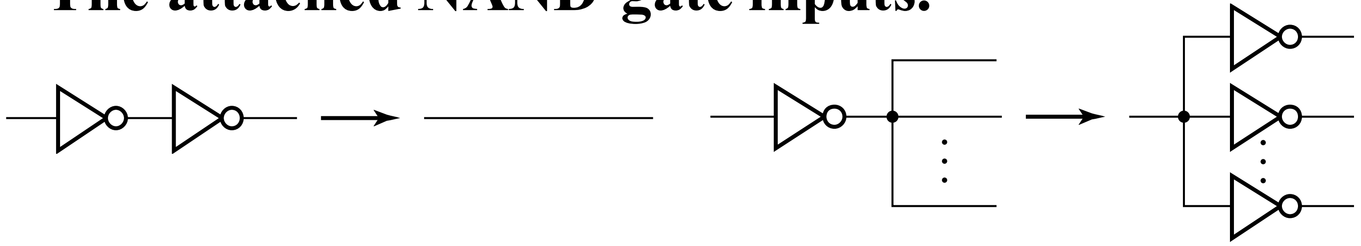
- **Assumptions:**
  - Gate loading and delay are ignored
  - Cell library contains an inverter and *n*-input NAND gates, *n* = 2, 3, …
  - An AND, OR, inverter schematic for the circuit is available
- **The mapping is accomplished by:**
  - Replacing AND and OR symbols,
  - Pushing inverters through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

NAND Mapping Algorithm

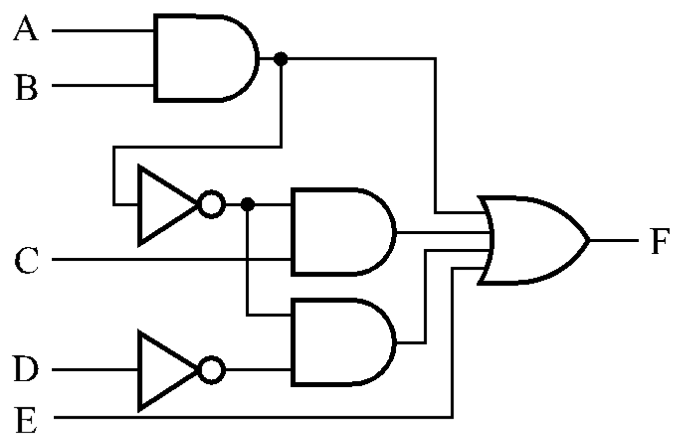
1. **Replace ANDs and ORs:**



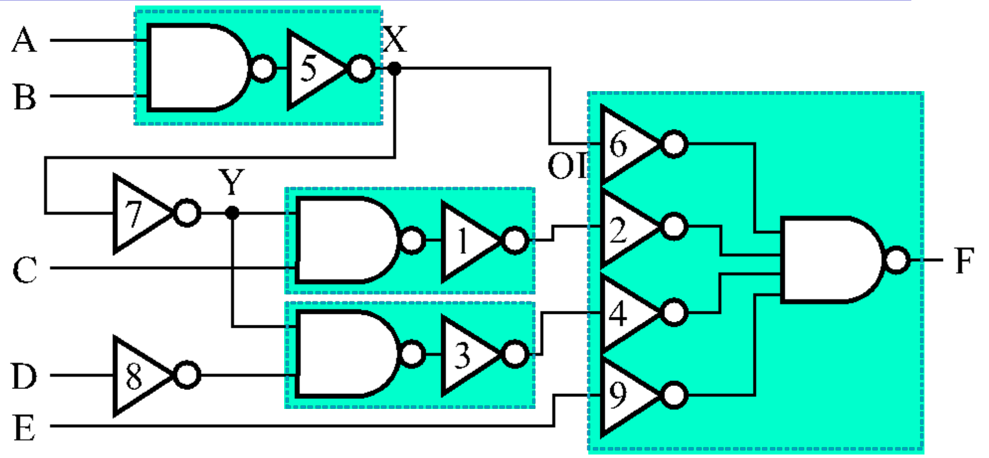
2. **Repeat the three steps of actions until there is at most one inverter between :**
   a. **A circuit input or driving NAND gate output, and**
   b. **The attached NAND gate inputs.**



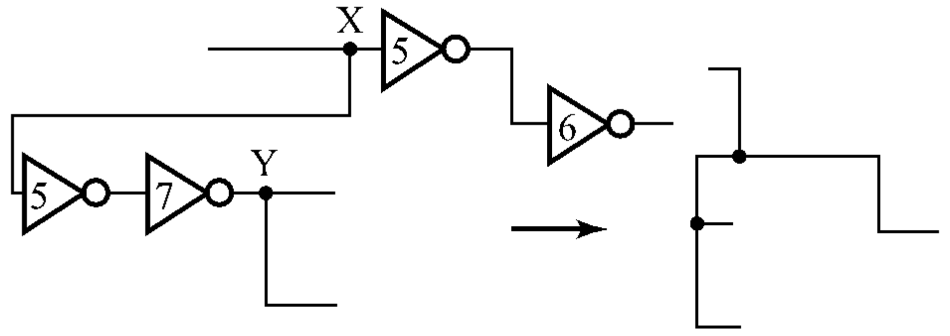
NAND Mapping Example



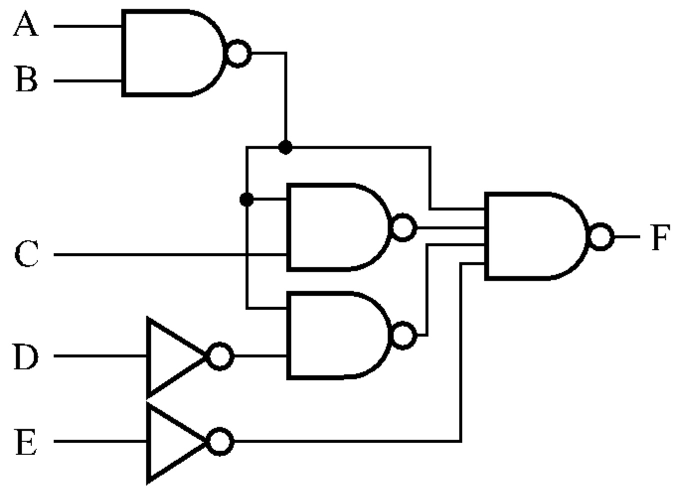
(a)



(b)



(c)



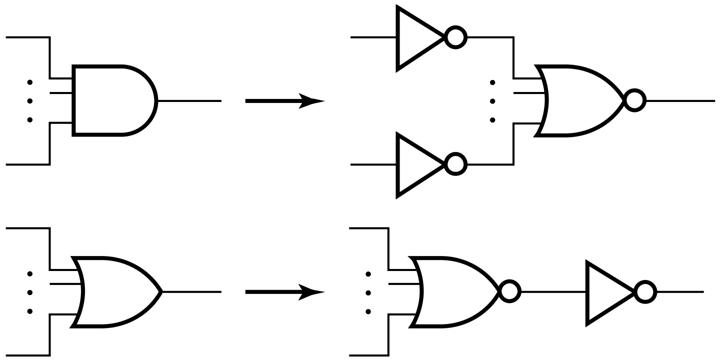
(d)

Mapping to NOR gates

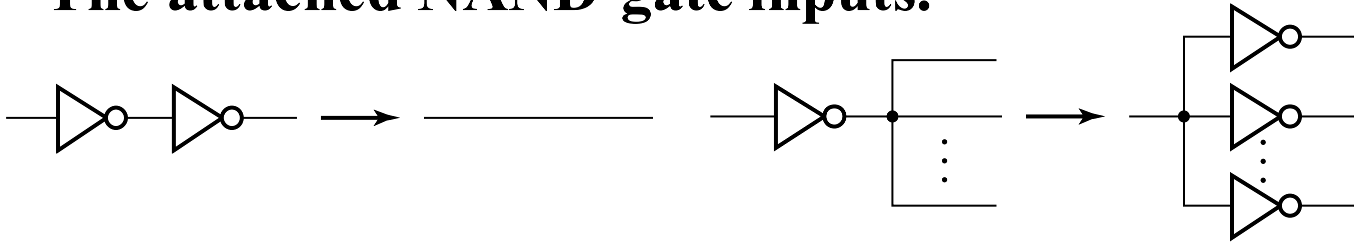
- **Assumptions:**
  - Gate loading and delay are ignored
  - Cell library contains an inverter and *n*-input NOR gates, *n* = 2, 3, …
  - An AND, OR, inverter schematic for the circuit is available
- **The mapping is accomplished by:**
  - Replacing AND and OR symbols,
  - Pushing inverters through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

NOR Mapping Algorithm

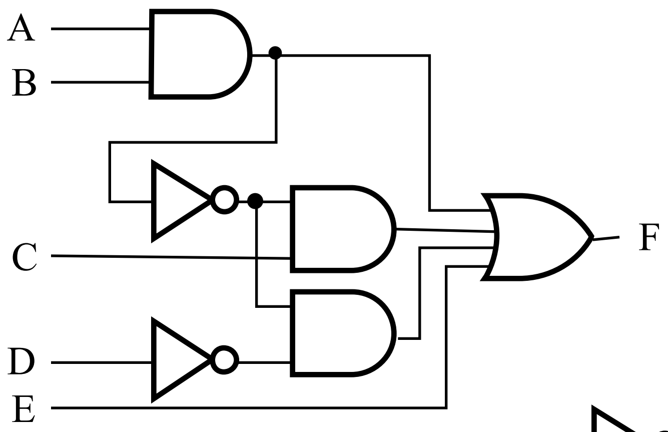
1. **Replace ANDs and ORs:**



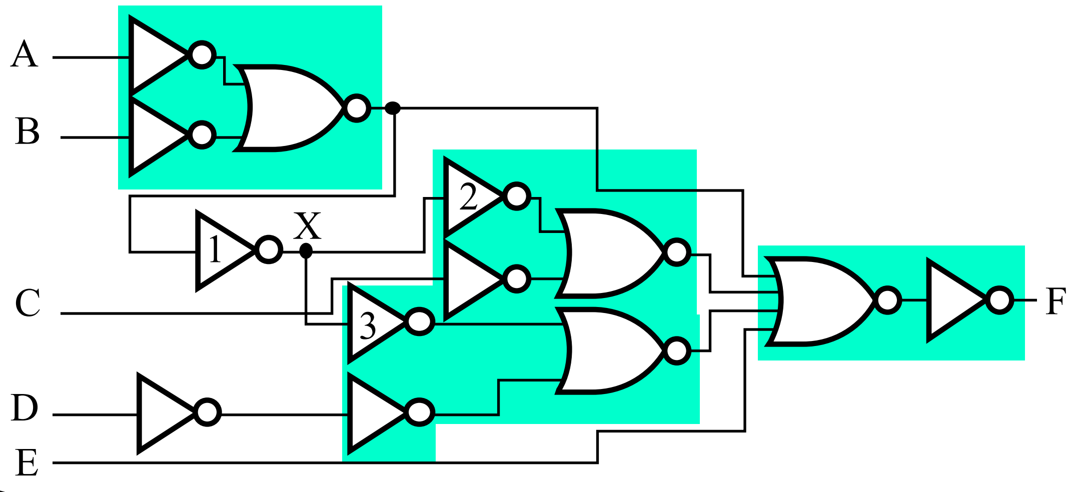
2. **Repeat the three steps of actions until there is at most one inverter between :**
   a. **A circuit input or driving NAND gate output, and**
   b. **The attached NAND gate inputs.**



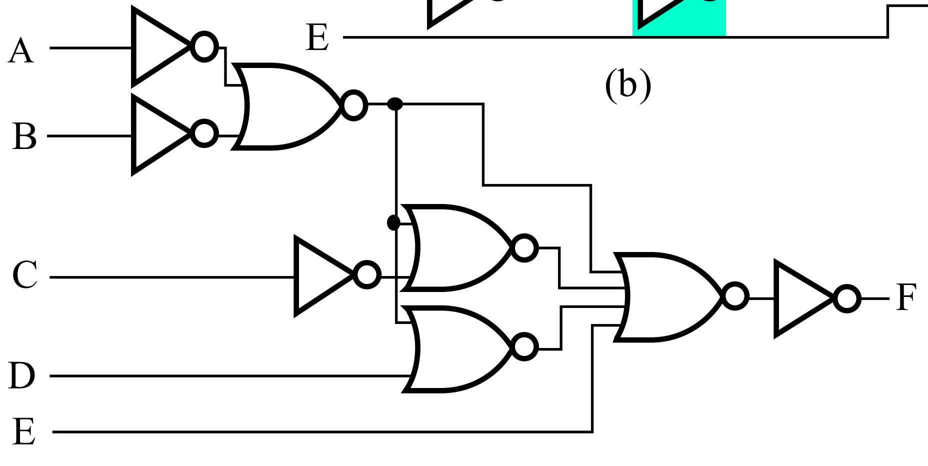
NOR Mapping Example



(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:**

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

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

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

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

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

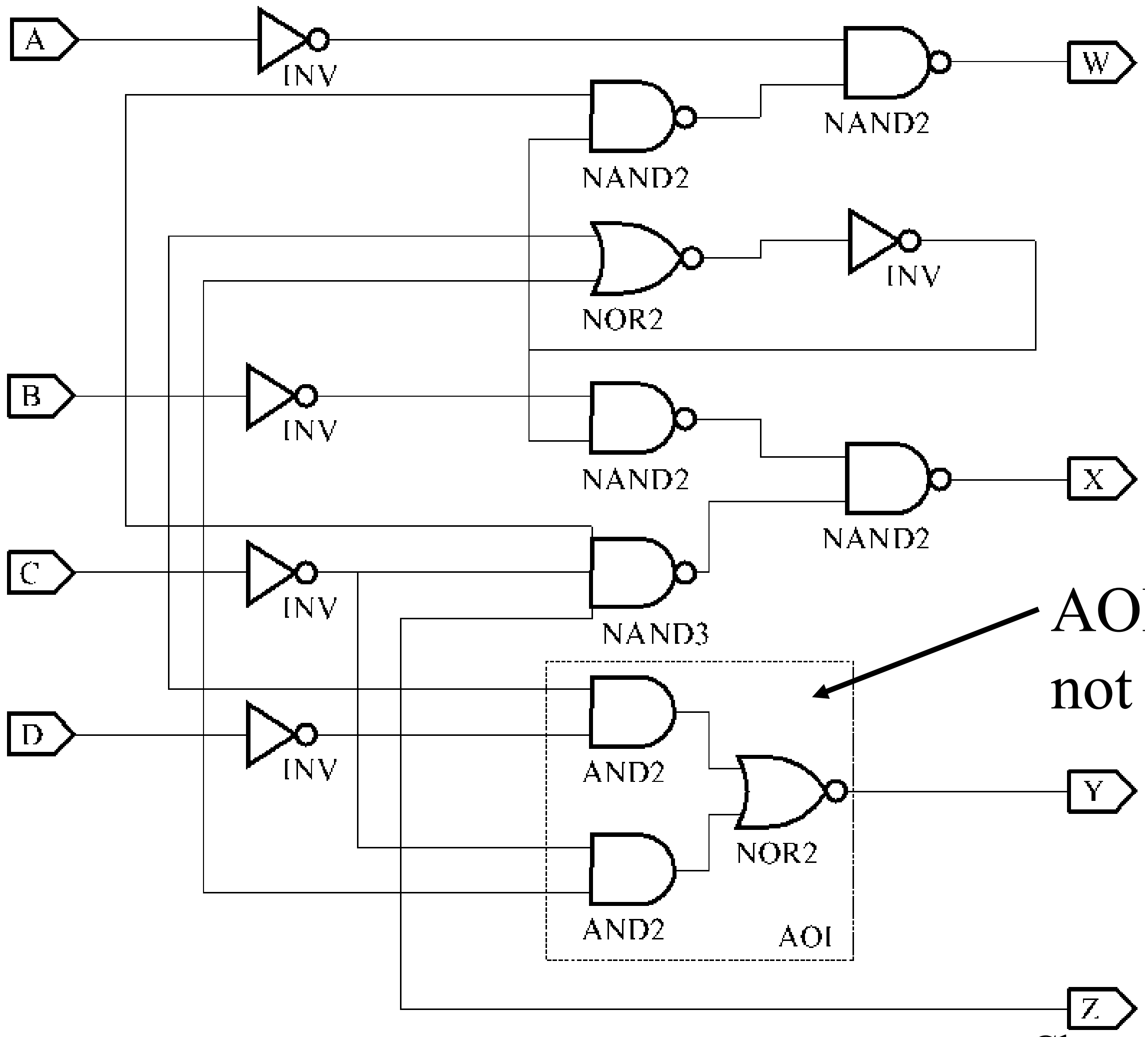
The tables match!

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

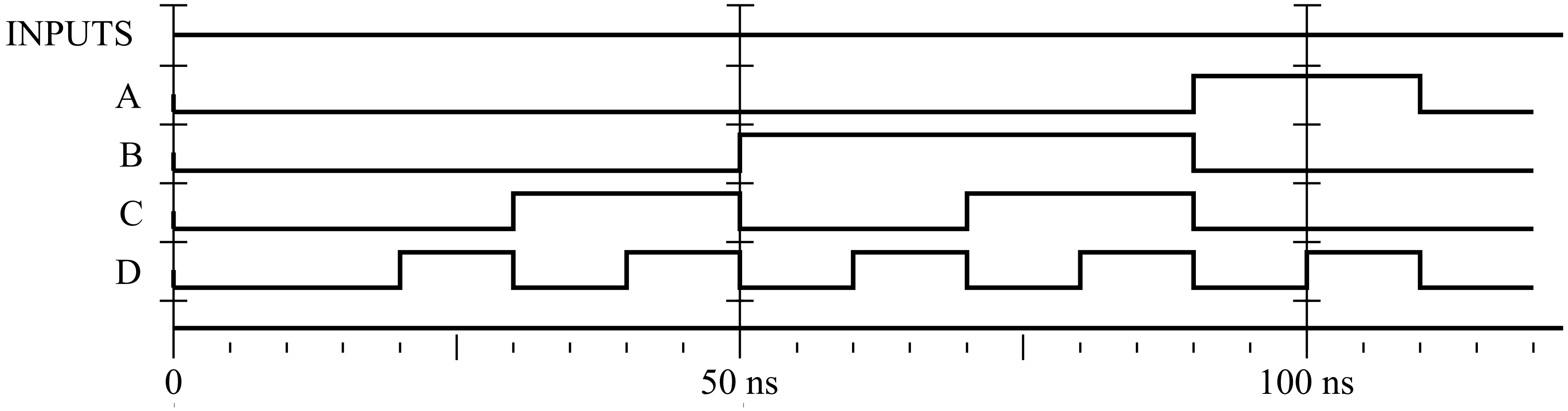
- **Enter BCD-to-Excess-3 Code Converter Circuit Schematic**



AOI symbol not available

Verification Example: Simulation

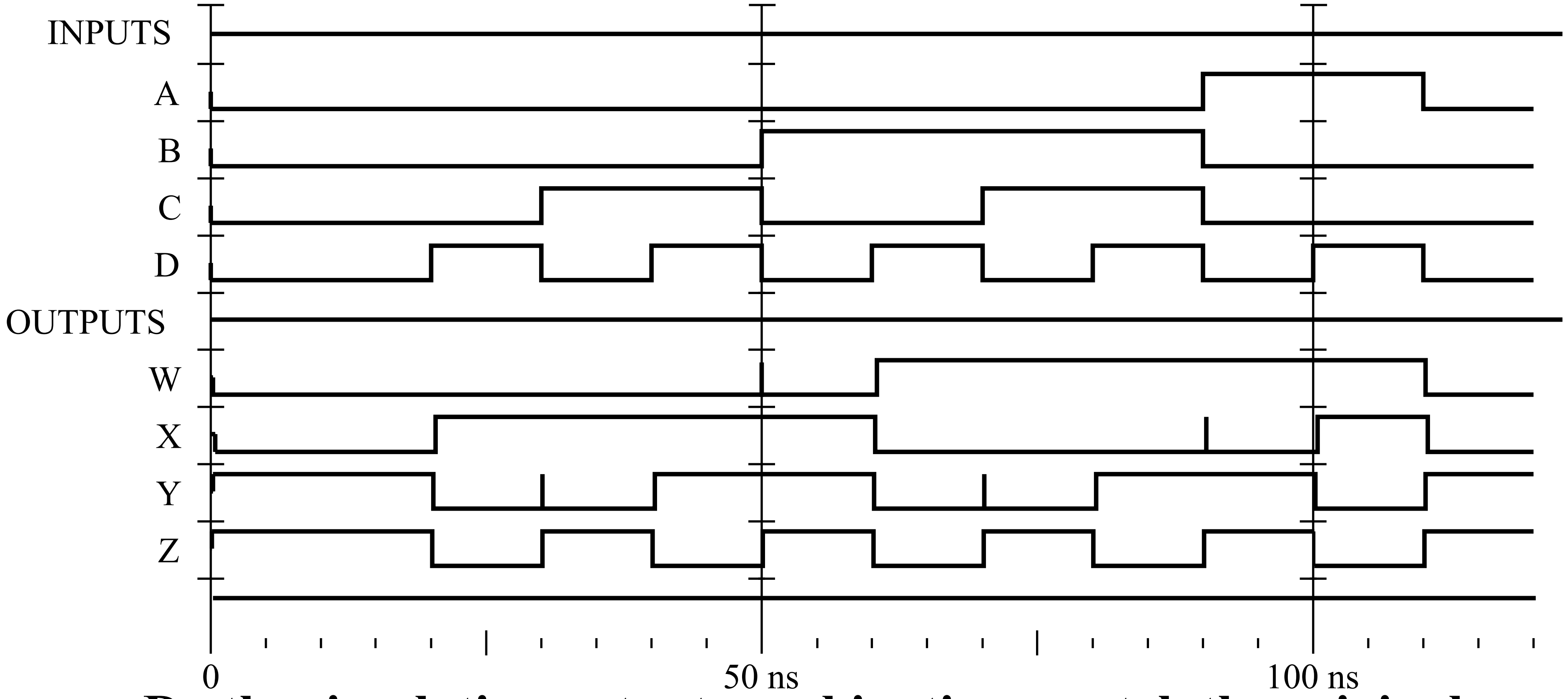
- **Enter waveform that applies all possible input combinations:**



- **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?**

Assignment

- 3-7, 3-8, 3-11, 3-13, 3-14, 3-16, 3-27, 3-28, 3-29, 3-37, 3-44, 3-47, 3-50, 3-51, 3-52, 3-59