Logic and Computer Design Fundamentals

Chapter 3 – Combinational Logic Design

Part 1 – Implementation Technology and Logic Design

Ming Cai
cm@zju.edu.cn
College of Computer Science and Technology
Zhejiang University

Overview

- **Part 1 – Design Procedure**
  - **Design Concepts and Automation**
  - **Design Space - Technology parameters for gates, positive and negative logic and design tradeoffs**
  - **Design Steps – Specification, formulation, optimization, technology mapping**
  - **Technology Mapping - AND, OR, and NOT to NAND or NOR**
  - **Verification**
    - **Manual**
    - **Simulation**

Overview (continued)

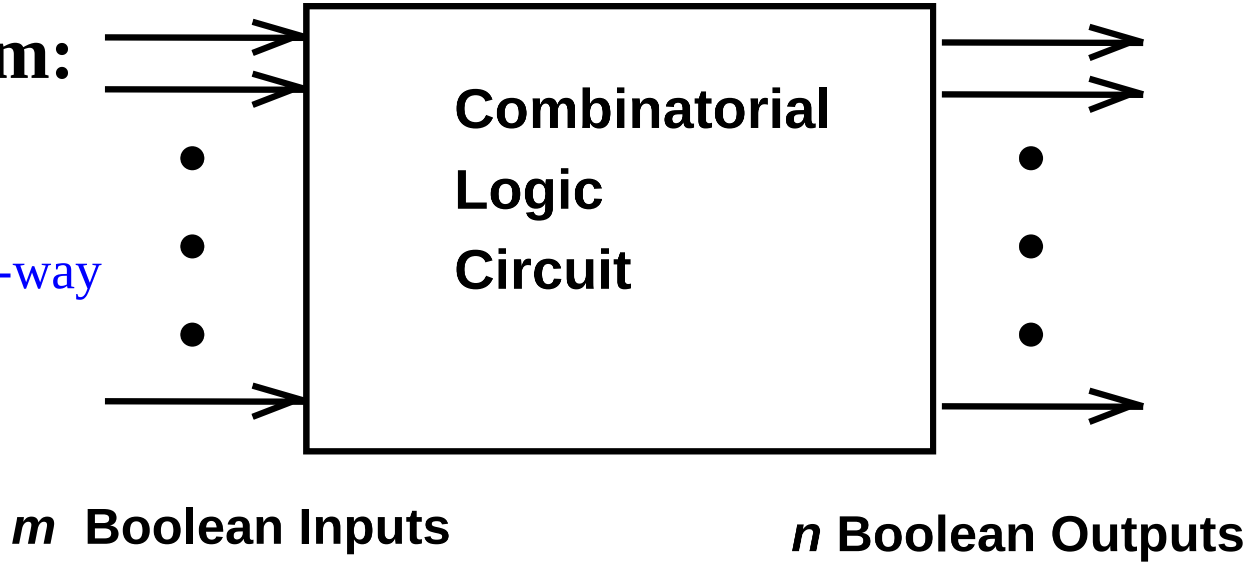
- Part 2 – Combinational Logic
  - Functions and functional blocks
  - Rudimentary logic functions
  - Decoding using Decoders
    - Implementing Combinational Functions with Decoders
  - Encoding using Encoders
  - Selecting using Multiplexers
    - Implementing Combinational Functions with Multiplexers

Two types of logic circuits

- **Two types of logic circuits:**
  - **Combinational Logic Circuit**
  - **Sequential Logic Circuit**
- **Definition of Combinational Circuit**
  - **A combinational circuit consists of logic gates whose output is a function of only the present input.**
- **Definition of Sequential Logic Circuit**
  - **Sequential logic is a type of logic circuit whose output depends not only on the present value of its input signals but on the sequence of past inputs (state or memory).**

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:**



Note: Without any memory devices, feedback loops, one-way transmission of input signal

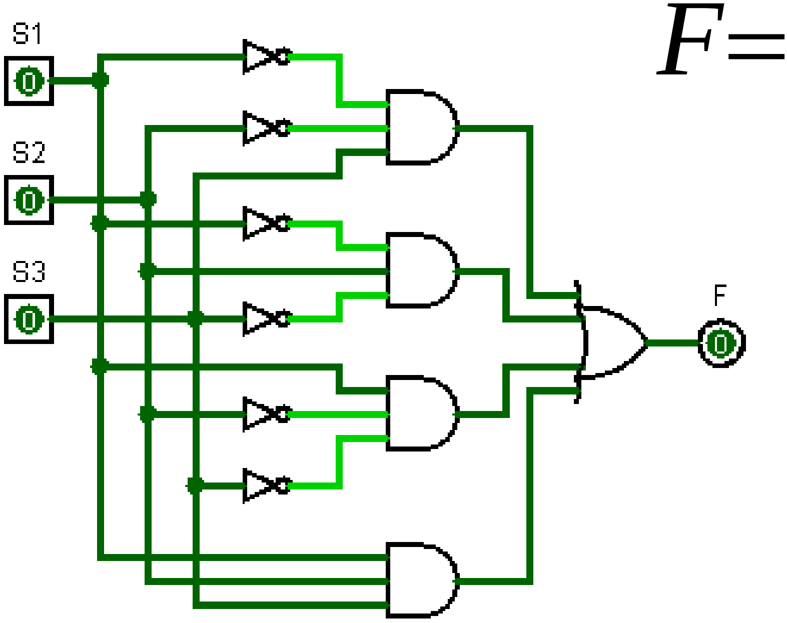
Method of Describing Logic Events

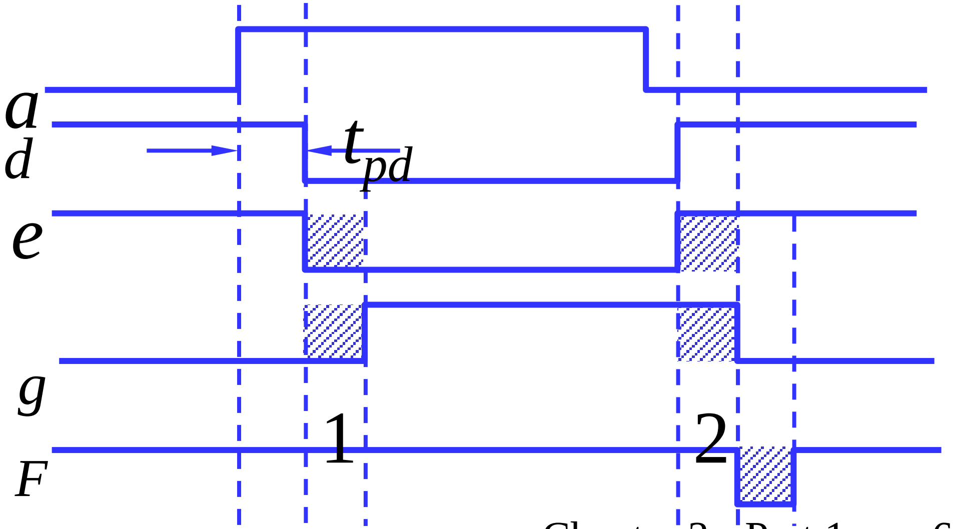
1. **Truth Table**
2. **Boolean Function**
3. **Karnaugh Maps**
4. **Timing Diagram**
5. **Logic Circuit**

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

$$F = S_3 S_2 S_1 + S_3 S_2 S_1 + S_3 S_2 S_1 + S_3 S_2 S_1$$





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: 9-input parity tree (see next slide)**
  - **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 $4 \times 2 \times 4 = 32$ 2-input NAND gates**

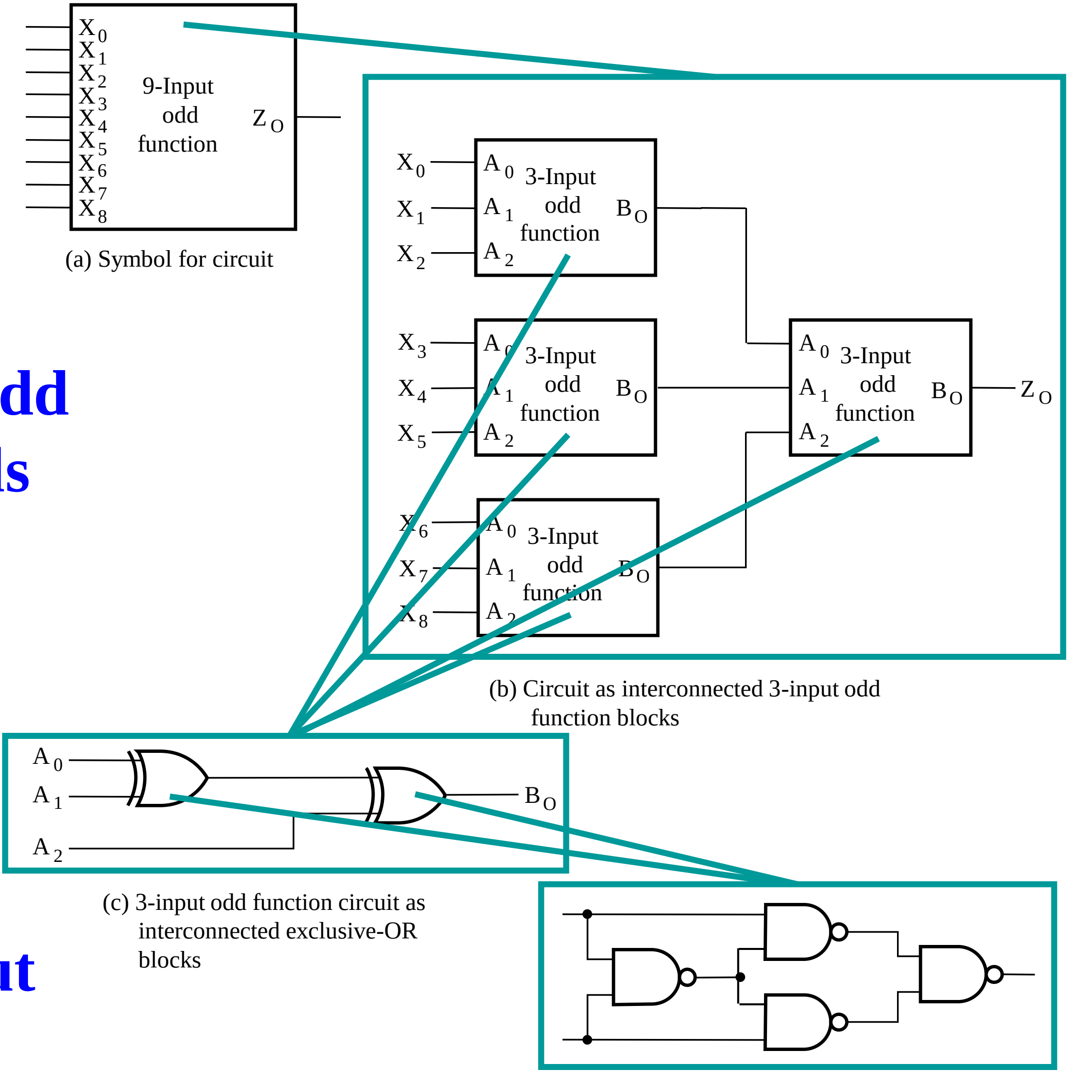
Hierarchy for Parity Tree Example

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



(a) Symbol for circuit

(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

Reusable Function Blocks and CAD

- **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**
- **Representatives of Computer-Aided Design Tools:**
  - **Schematic Capture**
  - **Logic Simulators**
  - **Tools for Timing Verification**
  - **Hardware Description Languages**
    - **Verilog and VHDL**
  - **Logic Synthesizers**
  - **Integrated Circuit Layout**

Top-Down versus Bottom-Up

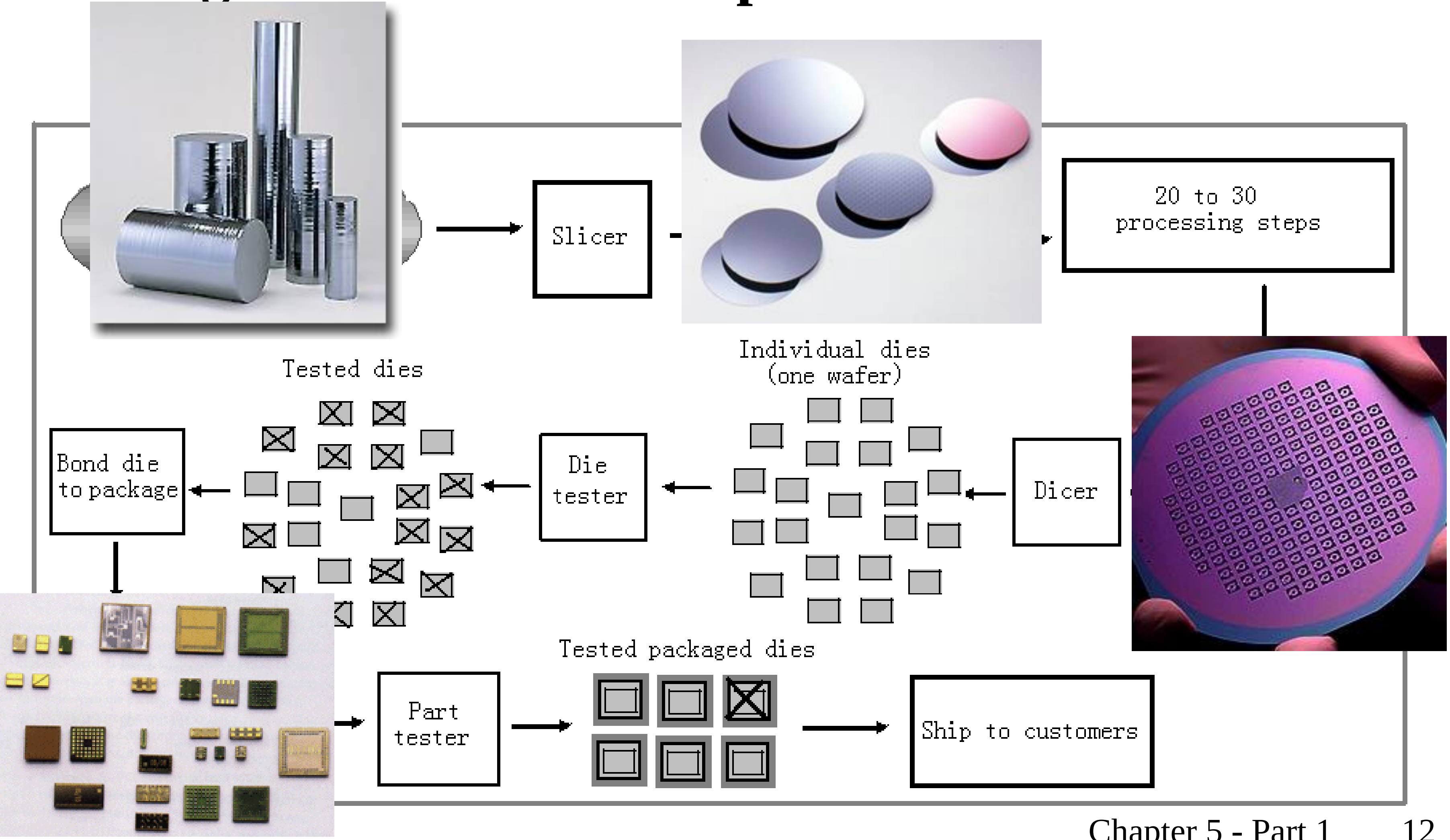
- A top-down design proceeds from an abstract, high-level specification to a more and more detailed design by decomposition and successive refinement
- A bottom-up design starts with detailed primitive blocks and combines them into larger and more complex functional blocks
- Designs usually proceed from both directions simultaneously
  - Top-down design answers: What are we building?
  - Bottom-up design answers: How do we build it?
- Top-down controls complexity while bottom-up focuses on the details

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**

Integrated Circuits

- **Design and manufacture procedure**



Slicer

20 to 30 processing steps

Individual dies (one wafer)

Dicer

Die tester

Tested dies

Bond die to package

Part tester

Tested packaged dies

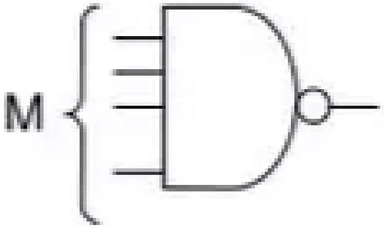
Ship to customers

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**
  - ***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**

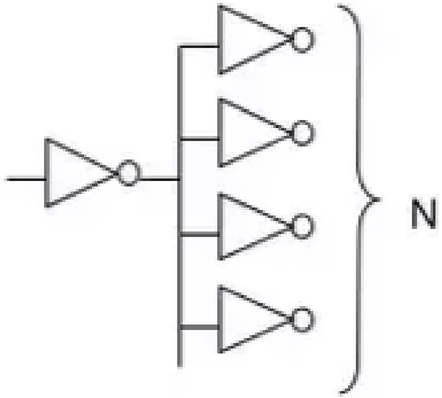
Fan-in and Fan-out

- **Fan-in: The fan-in is defined as the maximum number of inputs that a logic gate can accept. If number of input exceeds, the output will be undefined or incorrect.**



Fan-in = M

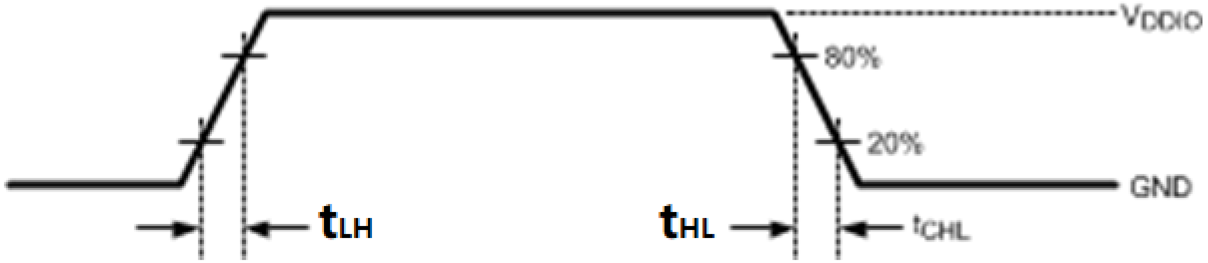
- **Fan-out: The fan-out is defined as the maximum number of inputs (load) that can be connected to the output of a gate without degrading the normal operation.**



Fan-out = N

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}**

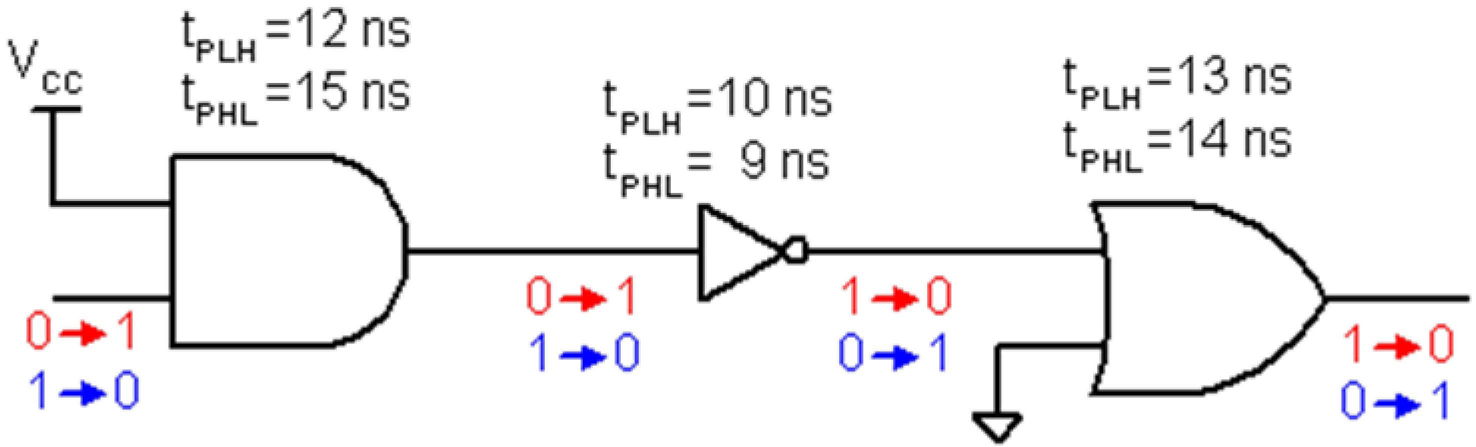


  - **The *maximum fan-out* that can be driven by a gate is the number of standard loads the gate can drive without exceeding its specified *maximum transition time***

Cost

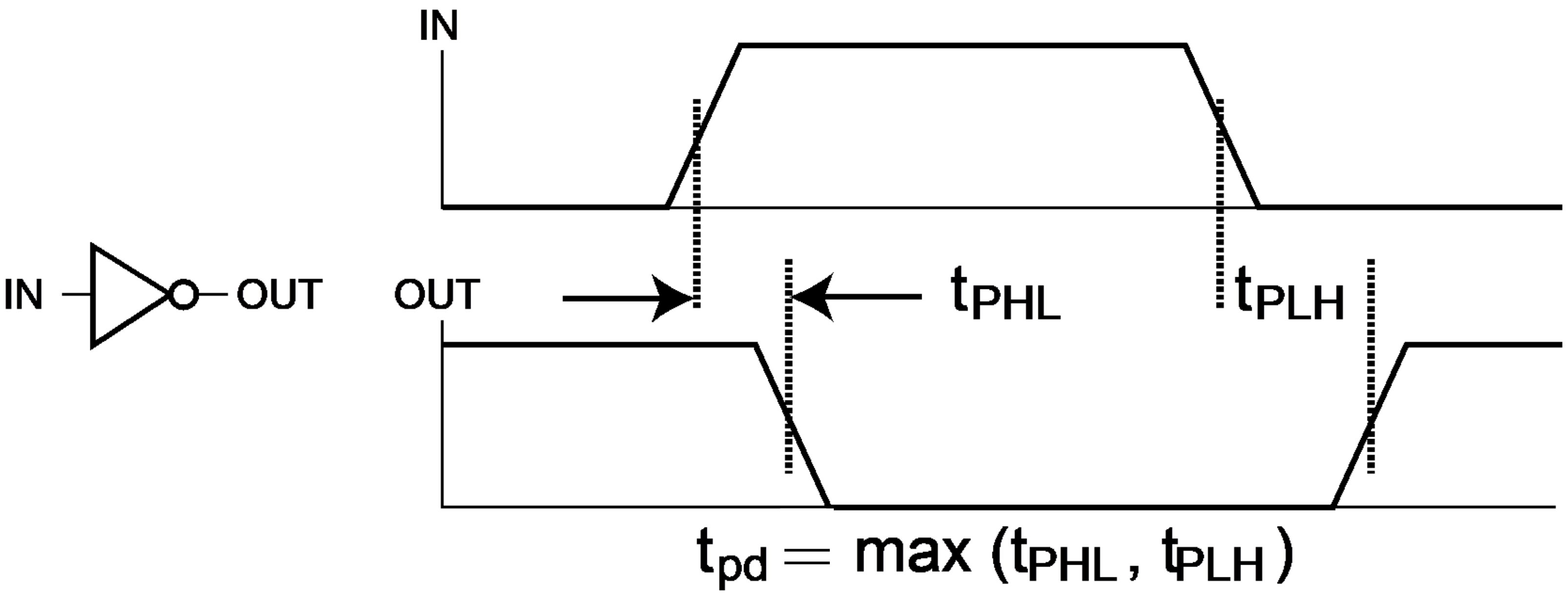
- In an integrated circuit:
  - The cost of a gate is proportional to the chip area occupied by the gate
  - The gate area is roughly proportional to the number and size of the transistors and the amount of wiring connecting them
  - Ignoring the wiring area, the gate area is roughly proportional to the gate input count
  - 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.**
- **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.**

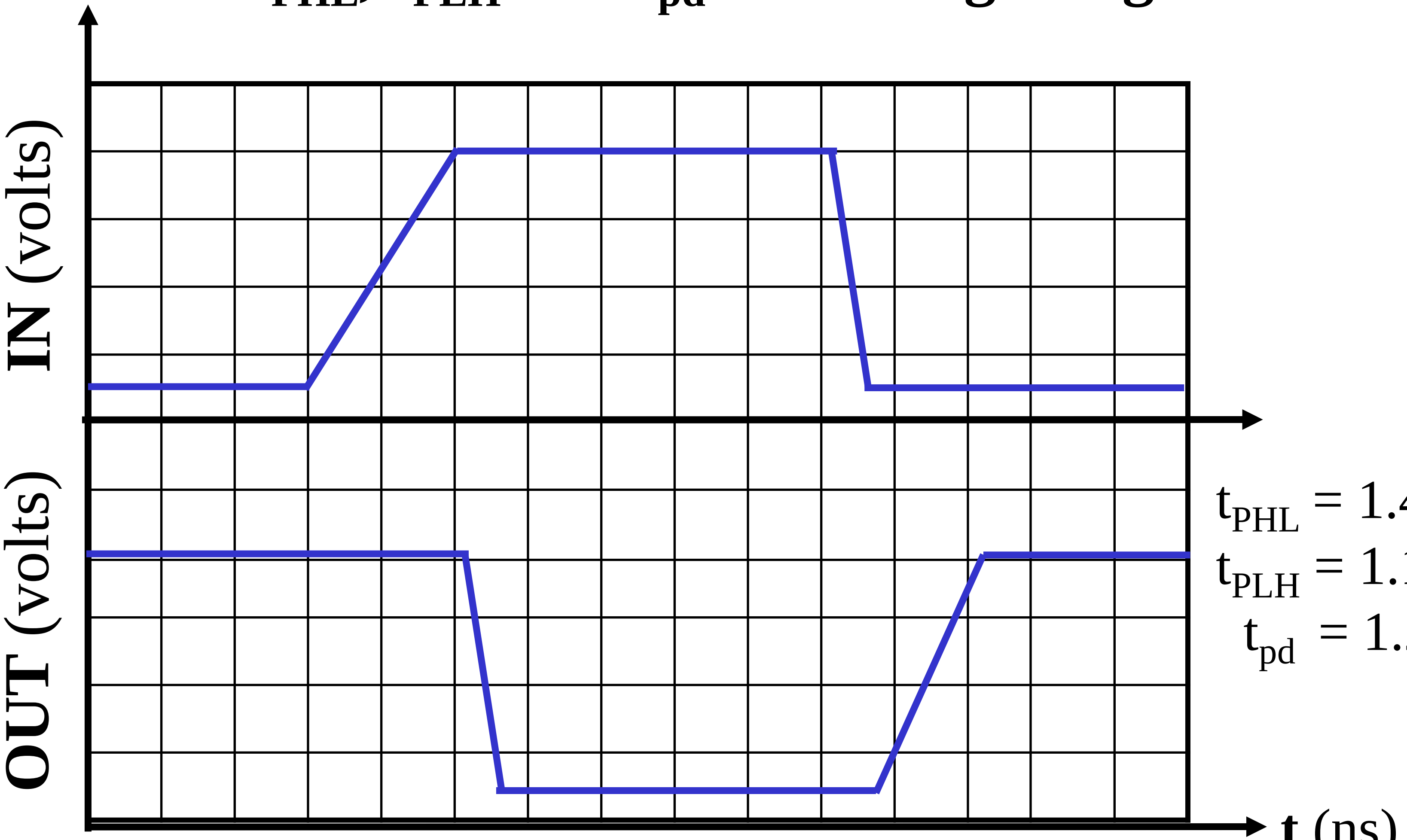
Propagation Delay (continued)



- Propagation delays measured at the midpoint (50% point) between the L and H values
- What is the expression for the t_{PHL} delay for:
  - a string of *n* identical buffers?
  - a string of *n* identical inverters?

Propagation Delay Example

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



t_{PHL} = 1.4 ns

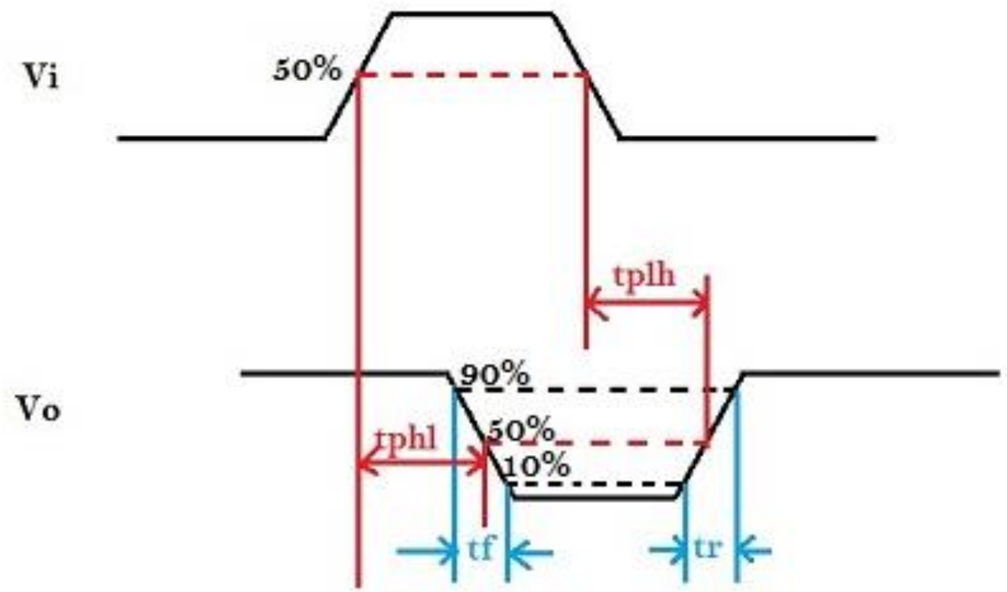
t_{PLH} = 1.1 ns

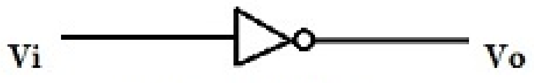
t_{pd} = 1.25 ns

1.0 ns per division

Propagation Delay vs. Transition time

- ***Propagation delay*** **is the time for a change on an input of a gate to propagate to the output.**
- ***Transition time*** **is the time required for gate's output to reach the final value.**



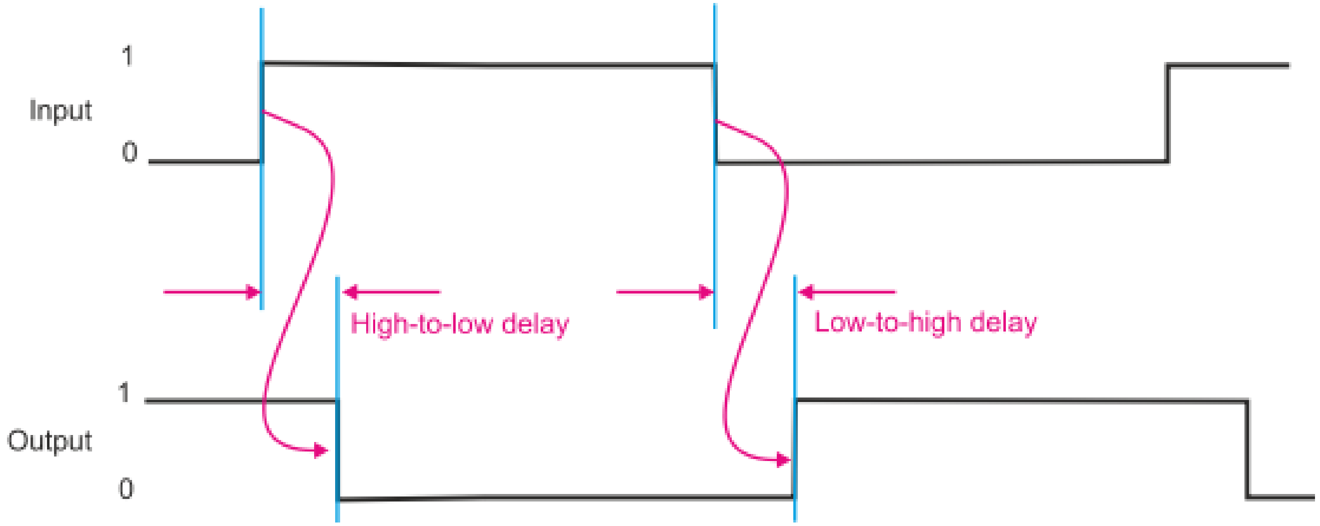


Inverter (NOT gate)

tr = Rise transition time = t_{LH}
tf = Fall transition time = t_{HL}
tphl = Propagation delay high-low
tplh = Propagation delay low-high

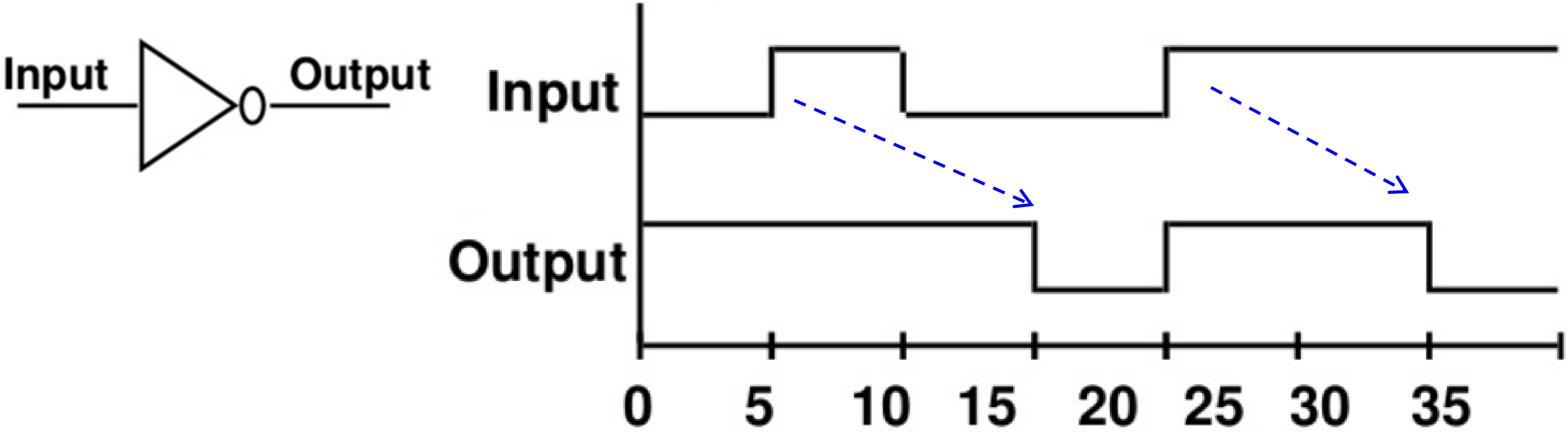
Delay Models

- **Two different models are usually employed in modeling inherent gate delays during simulation:**
  - **Transport delay**
  - **Inertial delay**



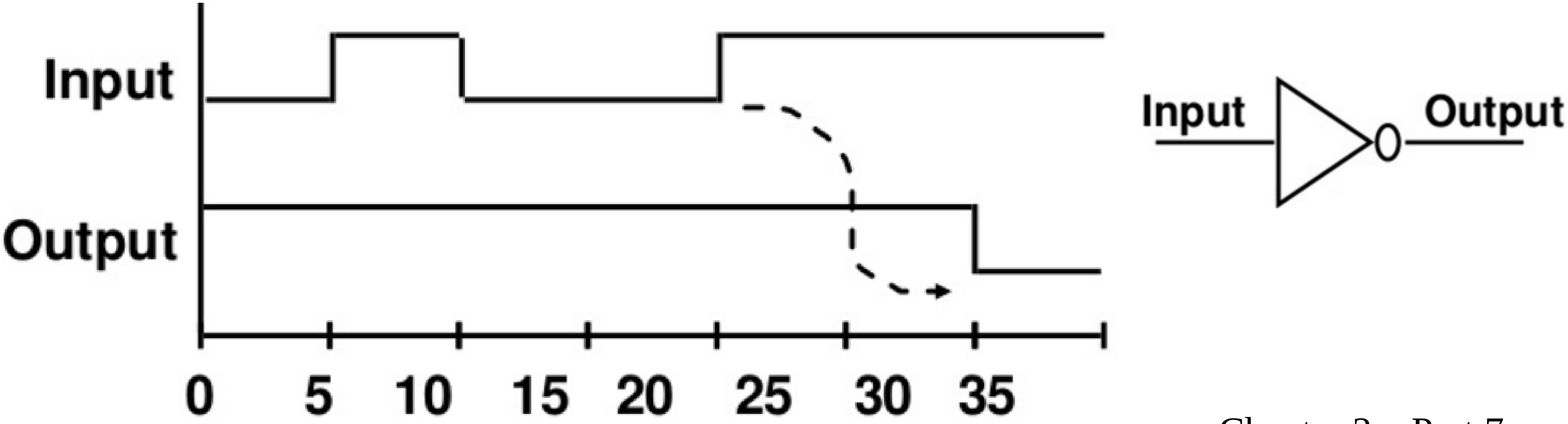
Delay Models (continued)

- *Transport delay*
  - A change in the output in response to a change on the inputs occurs after a fixed specified delay.
  - Circuits are like ideal conductors; that is, they are modeled as having no resistance.

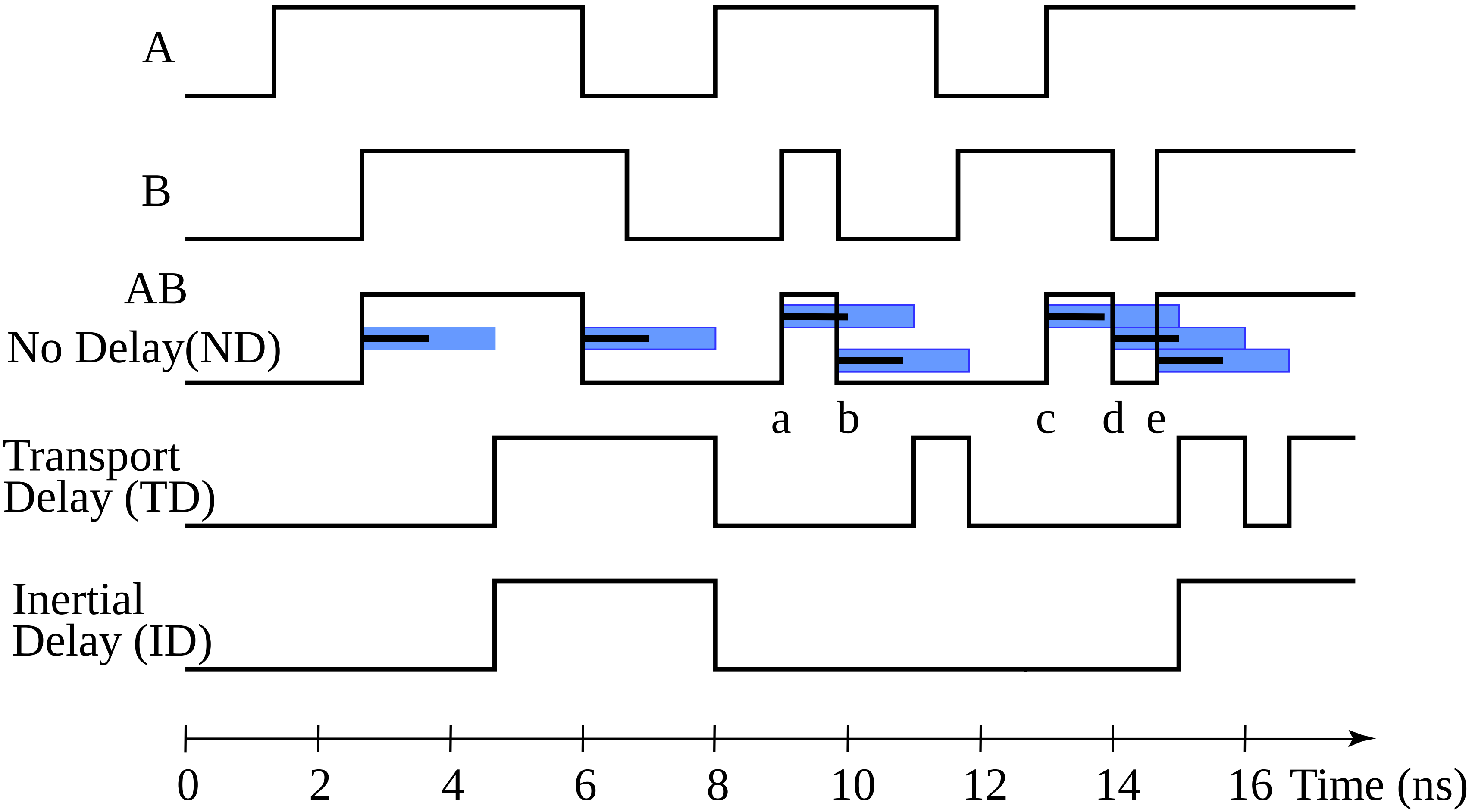


Delay Models (continued)

- *Inertial delay* is a measure of the elapsed time during which a signal must persist at an input of a device in order for a change to appear at an output.
- A pulse of duration less than the inertial delay (*rejection time*) does not contain enough energy to cause the device to switch.
- Inertial delay rejects narrow "pulses" on the outputs.



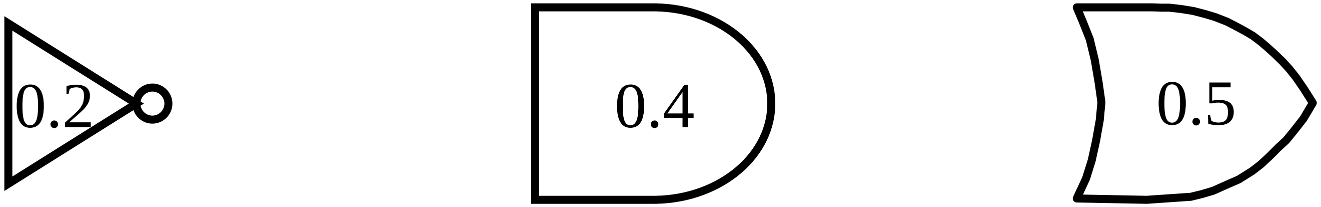
Delay Model Example



Transport Delay = 2.0 ns Rejection Time = 1.0 ns

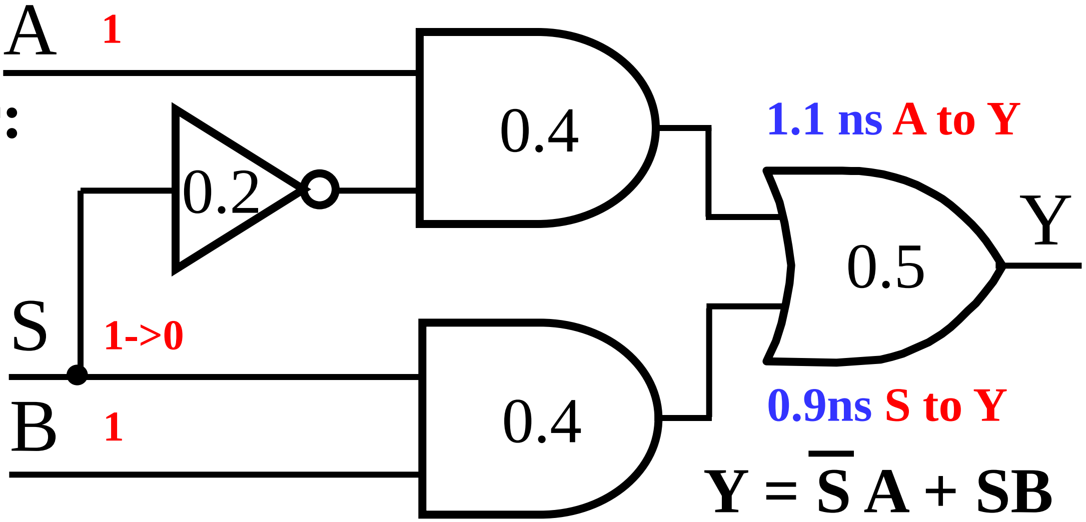
Circuit Delay

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

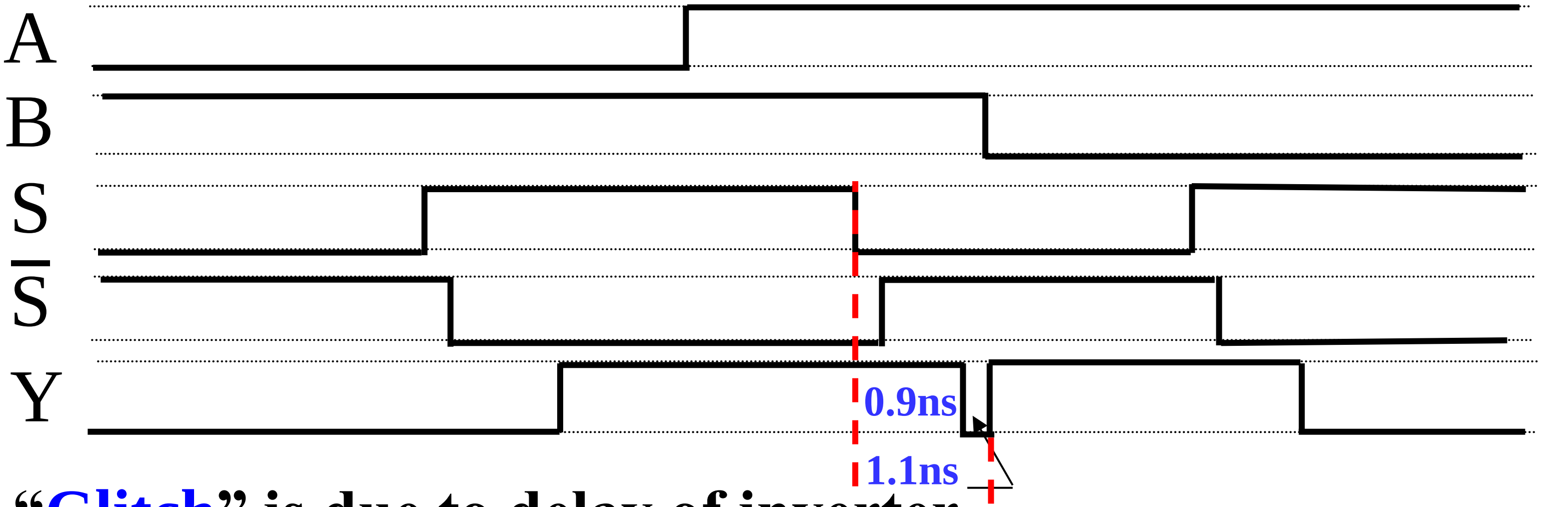


Circuit Delay

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



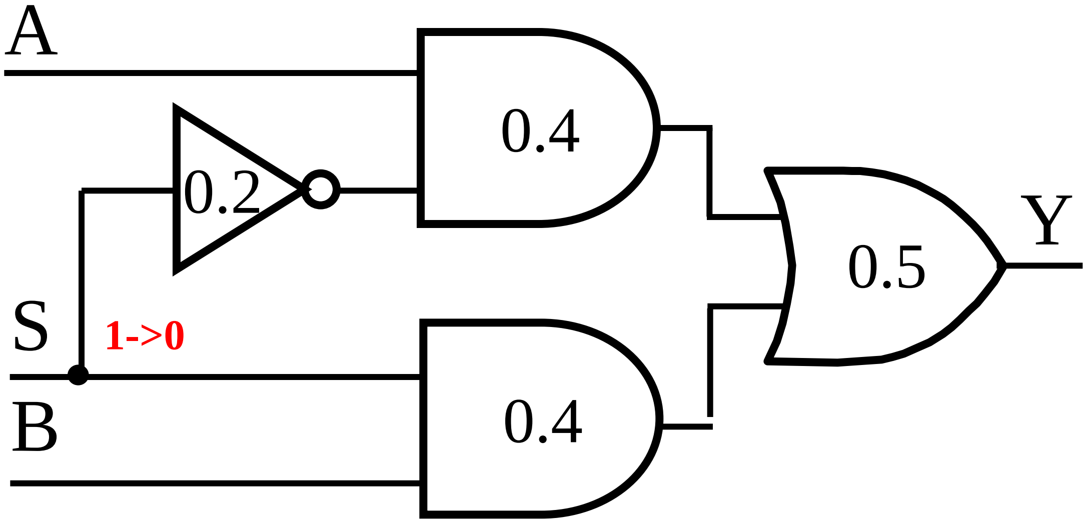
$$Y = \overline{S}A + SB$$

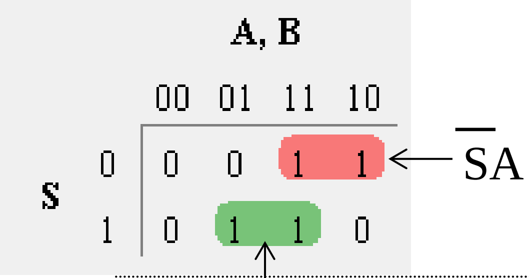


- **"Glitch" is due to delay of inverter**

Circuit Delay (continued)

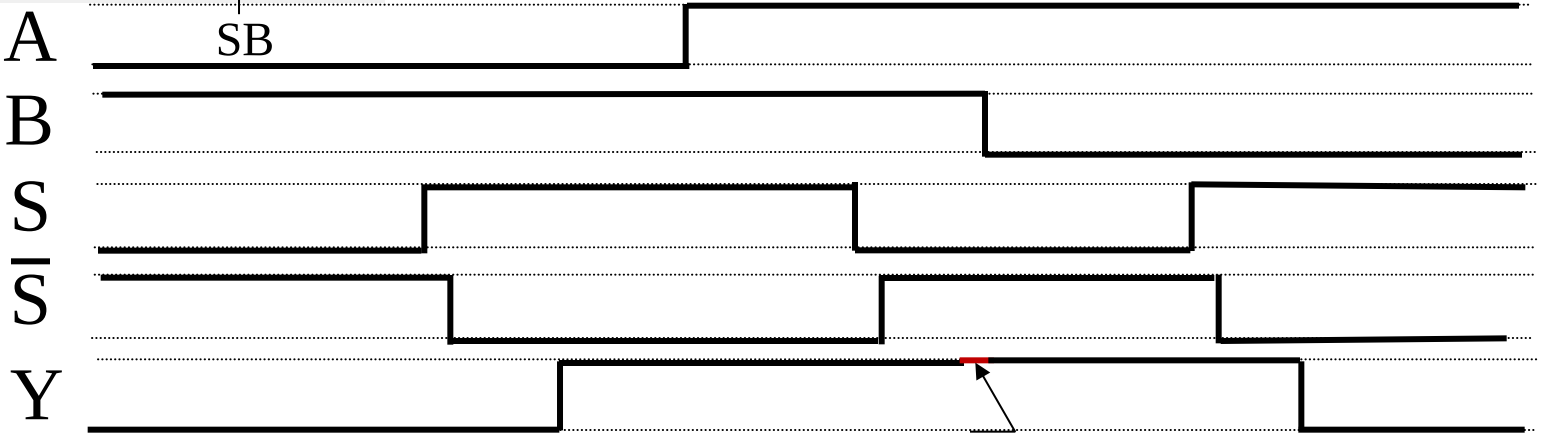
- **Y = SB + $\overline{\text{S}}$A**
- **The hazard appears when A=B=1, S=1->0**





		A, B			
		00	01	11	10
S	0	0	0	1	1
	1	0	1	1	0

$\overline{\text{S}}$A
SB

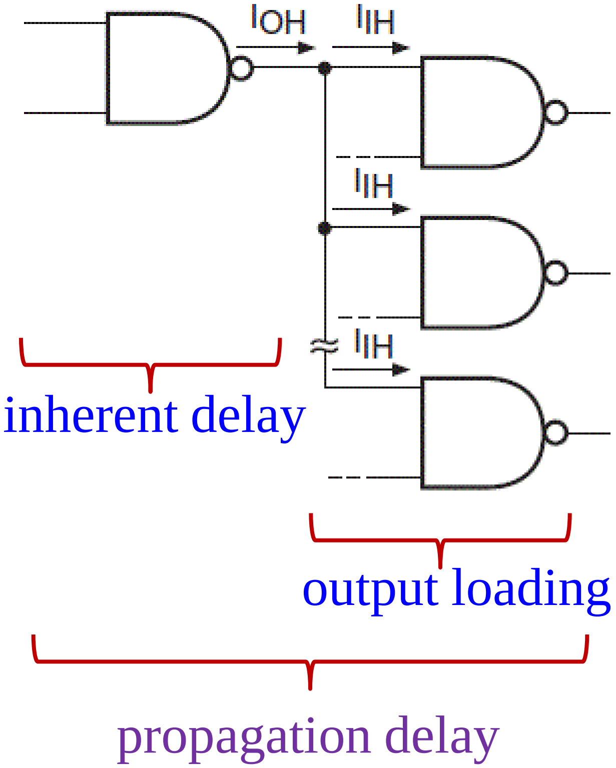


redundant term AB

- **Y = SB + $\overline{\text{S}}$A + AB**

Fan-out and Delay

- Apart from the inherent delay, the **fan-out** of a gate also affects the propagation delay.



- Example:
  - One realistic equation for t_{pd} for a NAND gate with 4 inputs is:

    $$t_{pd} = 0.07 + 0.021 * SL \quad ns$$

  - SL is the number of standard loads the gate is driving
  - For SL = 4.5
  - $t_{pd} = 0.07 + 0.021*4.5 = 0.165$ ns

Cost/Performance Tradeoffs

- Gate-Level Example:

Gate type	Standard load	Delay (ns)	Cost
NAND	20	0.45	2.0
Buffer	20	0.33	1.5

- 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

Positive and Negative Logic

- **The same physical gate has different logical meanings depending on interpretation of the signal levels.**
- ***Positive Logic***
  - **HIGH (more positive) signal levels represent Logic 1**
  - **LOW (less positive) signal levels represent Logic 0**
- ***Negative Logic***
  - **LOW (more negative) signal levels represent Logic 1**
  - **HIGH (less negative) signal levels represent Logic 0**
- **A gate that implements a Positive Logic AND function will implement a Negative Logic OR function, and vice-versa.**

Positive and Negative Logic (continued)

- Given this signal level table:

Input X Y	Output
L L	L
L H	H
H L	H
H H	H

- What logic function is implemented?

Positive Logic	(H = 1) (L = 0)
0 0	0
0 1	1
1 0	1
1 1	1

Negative Logic	(H = 0) (L = 1)
1 1	1
1 0	0
0 1	0
0 0	0

Positive and Negative Logic (continued)

- Rearranging the negative logic terms to the standard function table order:

Positive Logic	(H = 1) (L = 0)
0 0	0
0 1	1
1 0	1
1 1	1

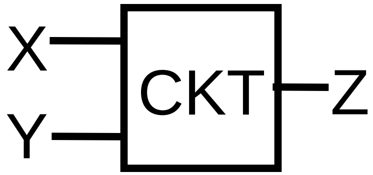
Negative Logic	(H = 0) (L = 1)
0 0	0
0 1	0
1 0	0
1 1	1

F=A+B

F=AB

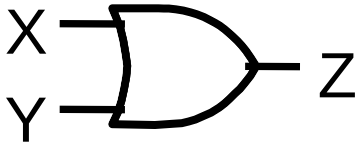
Logic Symbol Conventions

- Use of *polarity indicator* to represent use of negative logic convention on gate inputs or outputs

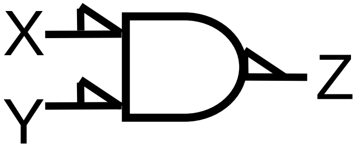


Logic Circuit

X	Y	Z
L	L	L
L	H	H
H	L	H
H	H	H



Positive Logic



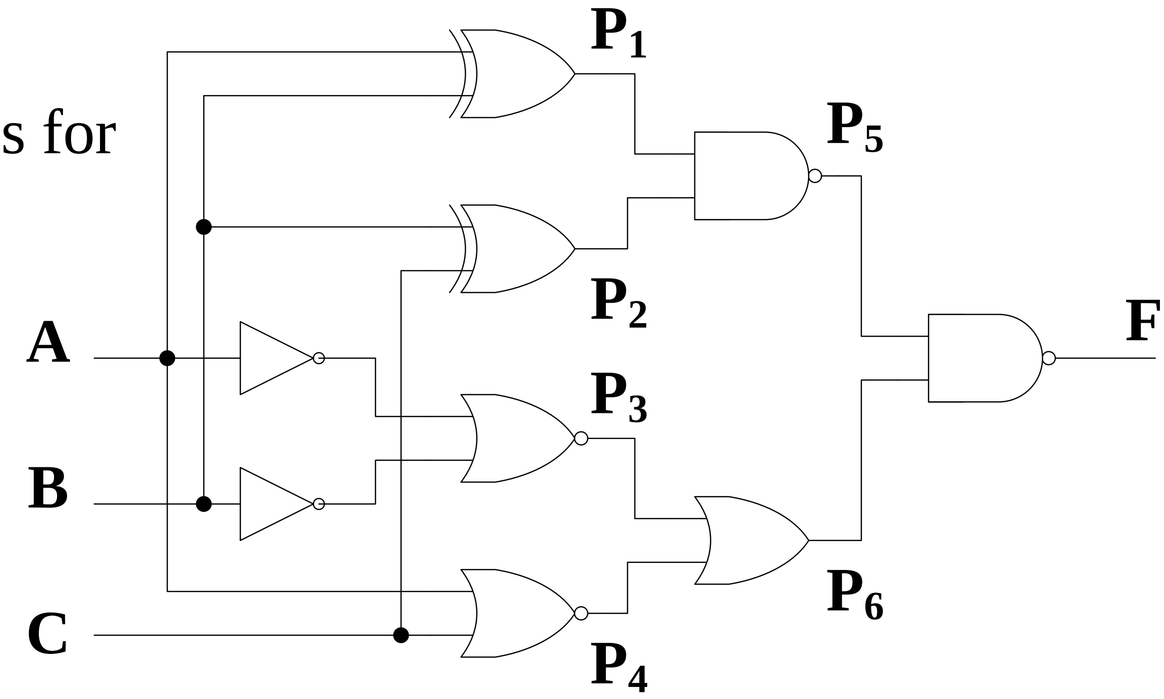
Negative Logic

Analysis of Logic Circuits

- **Write the Boolean function for the circuit**
  - **Begin with the input signal**
  - **Define the relationship of each gate**
  - **Optimization**
- **Derive a truth table**
  - **Define the relationships between the inputs and outputs**
- **Functional Analysis**
  - **Define the function of each signal and the whole circuit**
  - **Draw the timing diagram of the circuit**
- **Verification**
  - **Verify the correctness of the final design**

Analyze the Function of the Logic Circuit

List the Boolean functions for all signals



$$P_1 = A \oplus B$$

$$P_2 = B \oplus C$$

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

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

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

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

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

Boolean Function Simplification

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

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

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

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

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

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

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

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

Derive 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

Boolean Function Simplification

$F=($

$=(A \qquad A+C)$

$=A\overline{B}$

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

$=$

$=$

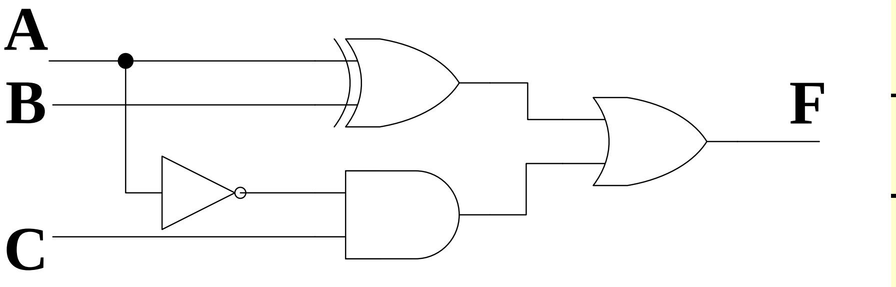
$=\overline{A}C+A\overline{B}+\overline{A}B$

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

Functional analysis:
From the truth table, we can see that F=1 if A≠B or B<C. it's a condition determination circuit.

Evaluation:
The original design is not the best. Use the following circuits instead.

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
				0	1
				0	1
				0	0
1	1	1	0	0	0

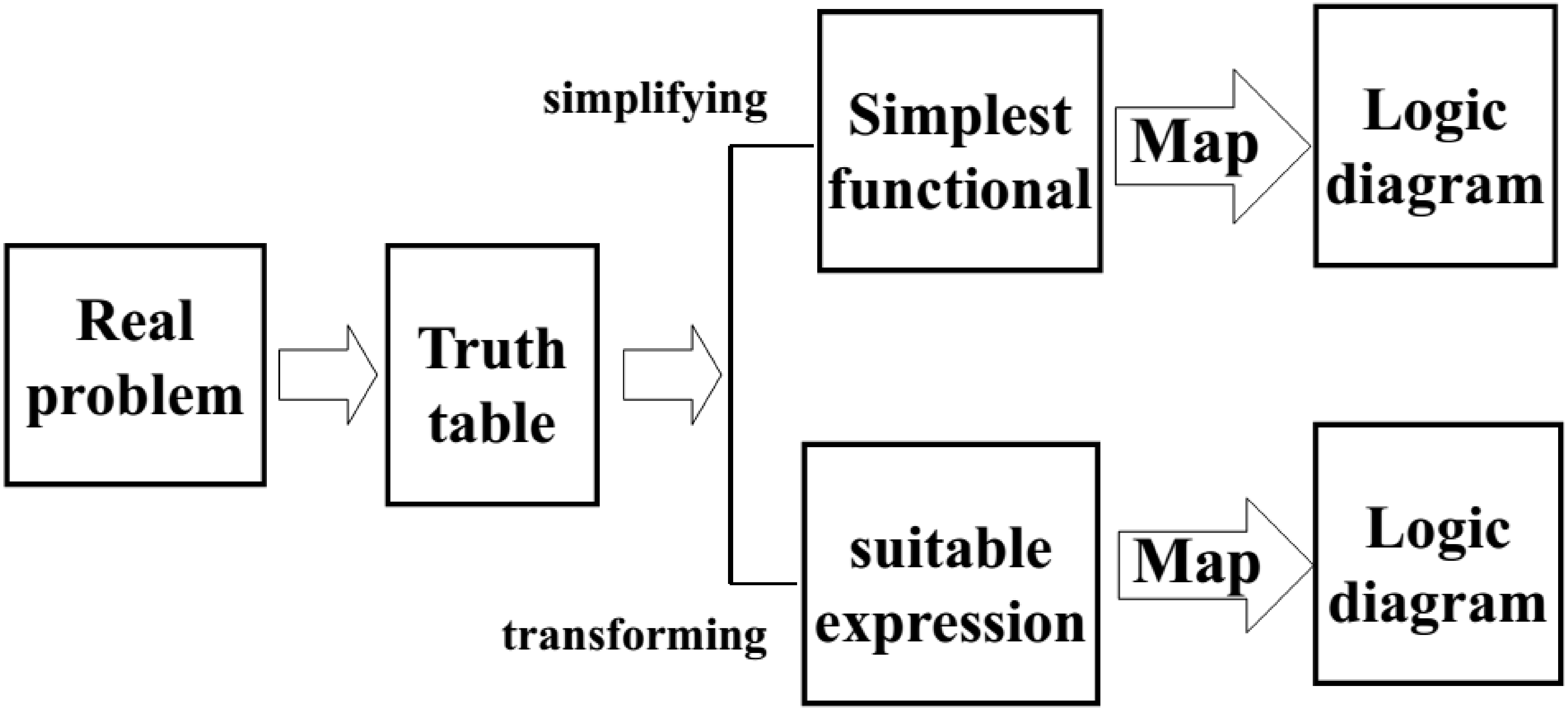


Design Procedure

1. Specification
   - Write a specification for the circuit if one is not already available (text, HDL)
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**



Real problem

Truth table

simplifying

Simplest functional

Map

Logic diagram

transforming

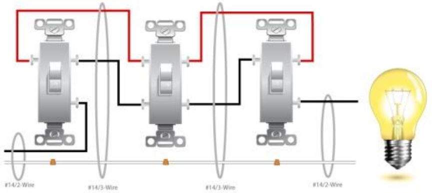
suitable expression

Map

Logic diagram

Design Example 1

Example: The only light in the room is controlled by three switches, and each switch can control the light separately. Please design the logic circuit for the light and switches with least gates.



Solutions: 1. Specification

Analysis result:

input signal: switches S_1,S_2,S_3
output signal: light F
“1” for switch closed and “0” for opened
“1” for light on and “0” for light off

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

2. Formulation

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

Design Example 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

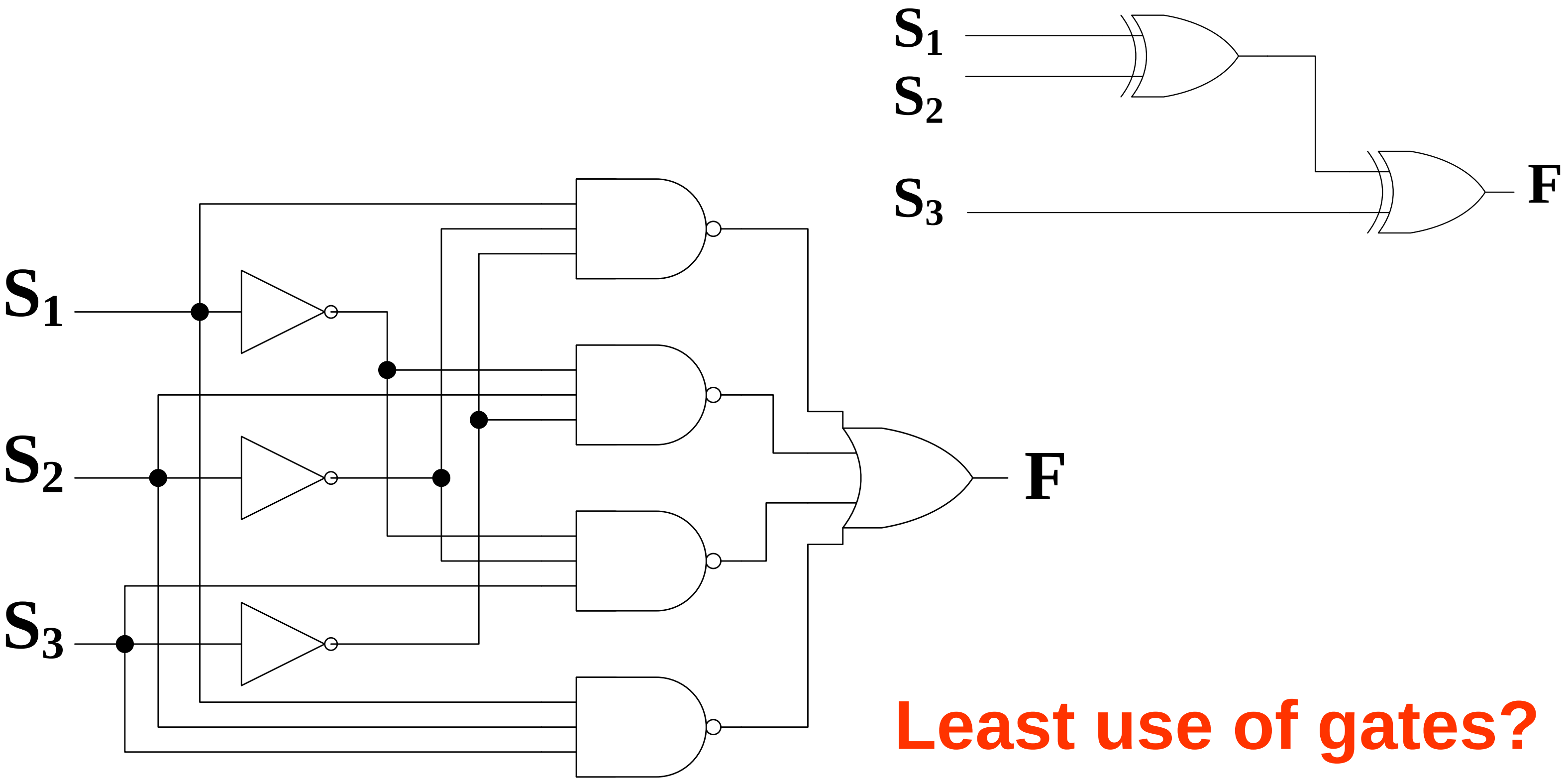
3. Optimization

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

Already optimized

Design Example 1

4. Technology Mapping



Least use of gates?

Design Example 1

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

5. Verilog Programming

```
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) | (s3&s2&s1) ;

endmodule
```

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)**

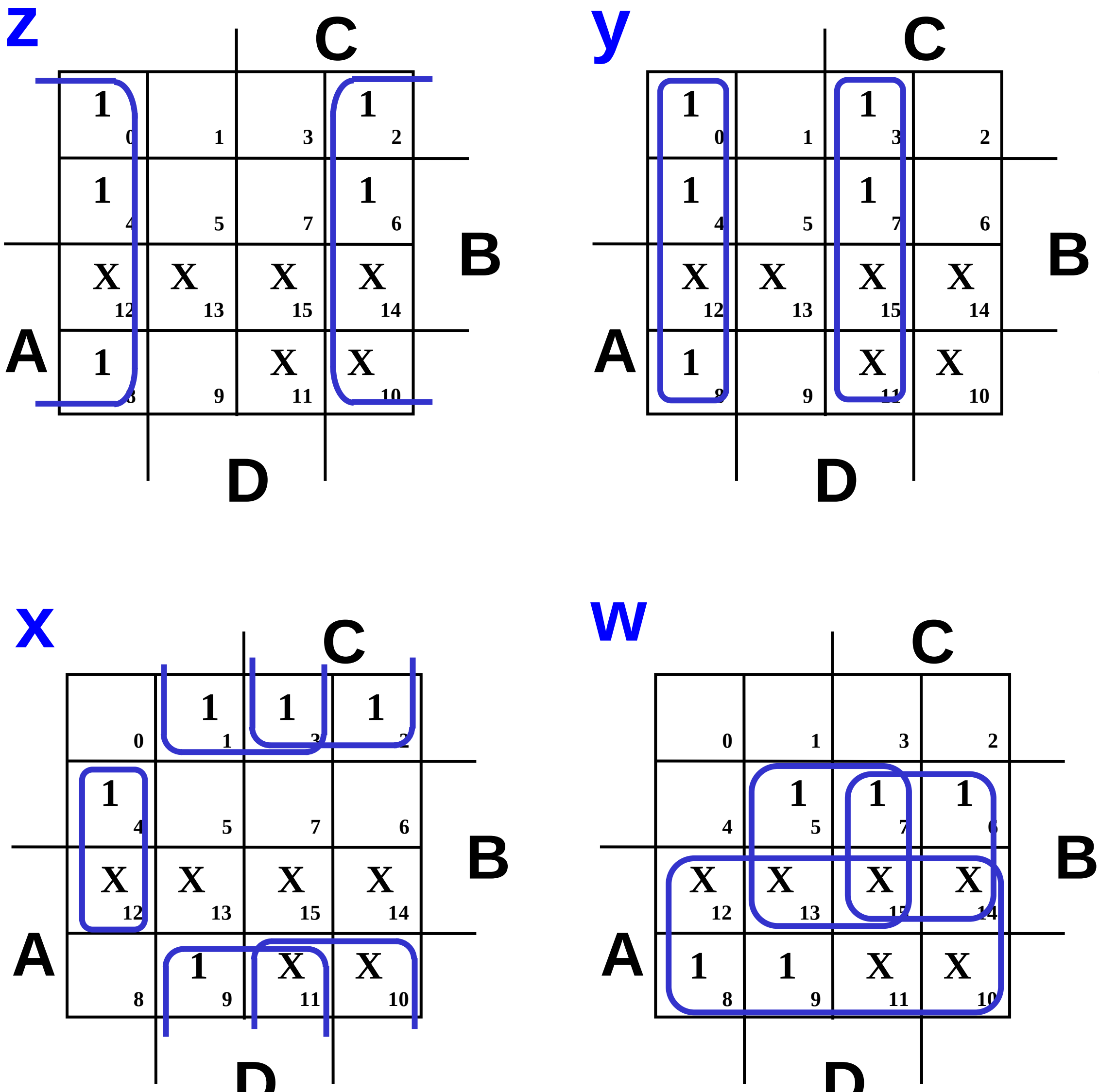
Design Example 2 (continued)

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	Output Excess-3
A B C D	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 2 (continued)



z

1 0	1	3	1 2
1 4	5	7	1 6
X 12	X 13	X 15	X 14
1 8	9	X 11	X 10

y

1 0	1	1 3	2
1 4	5	1 7	6
X 12	X 13	X 15	X 14
1 8	9	X 11	X 10

x

0	1 1	1 3	1 2
1 4	5	7	6
X 12	X 13	X 15	X 14
8	1 9	X 11	X 10

w

0	1	3	2
4	1 5	1 7	1 6
X 12	X 13	X 15	X 14
1 8	1 9	X 11	X 10

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 2 (continued)

3. Optimization (continued)

b. Multiple-level using transformations

$W = A + BC + BD$

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

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

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

- **Perform extraction, finding factor:**

$T_1 = C + D$

$W = A + BT_1$

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

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

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

Design Example 2 (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: ($\overline{C}\,\overline{D} = \overline{C + D} = \overline{T_1}$):

$W = A + BT_1$

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

$Y = CD + \overline{T_1}$

$Z = \overline{D}$

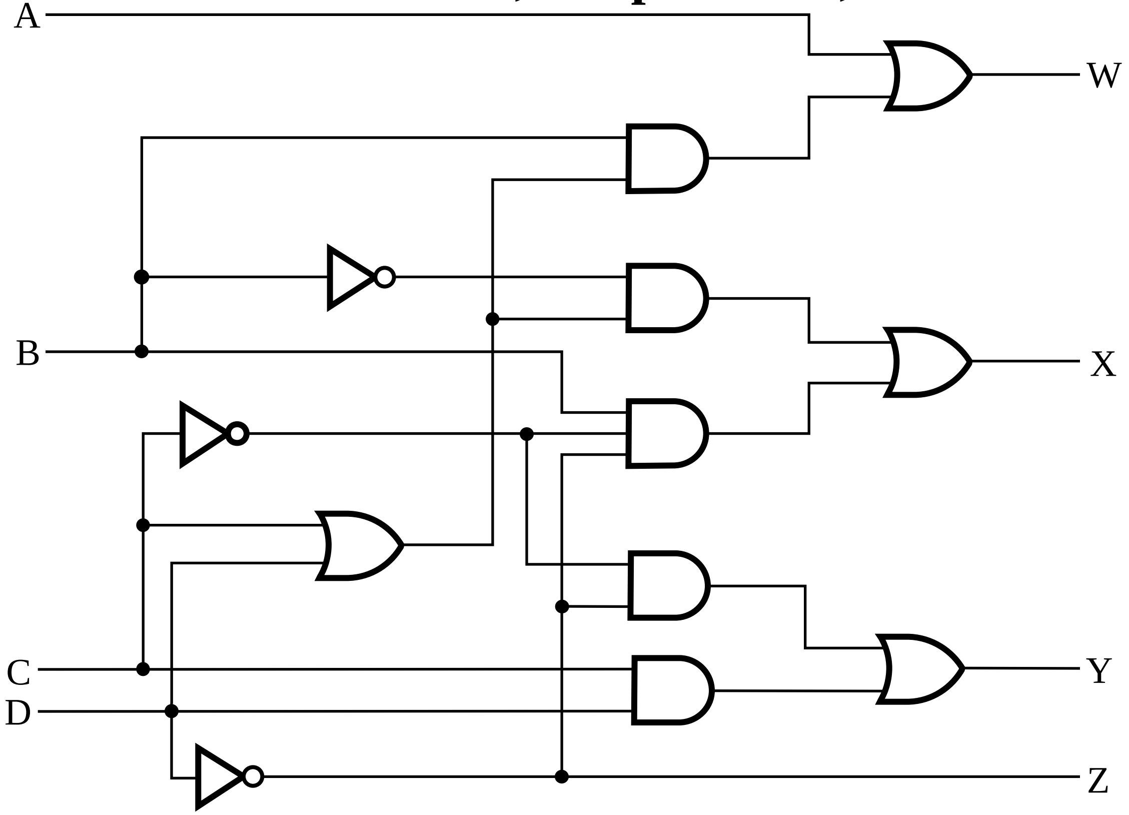
How many levels in the circuit?

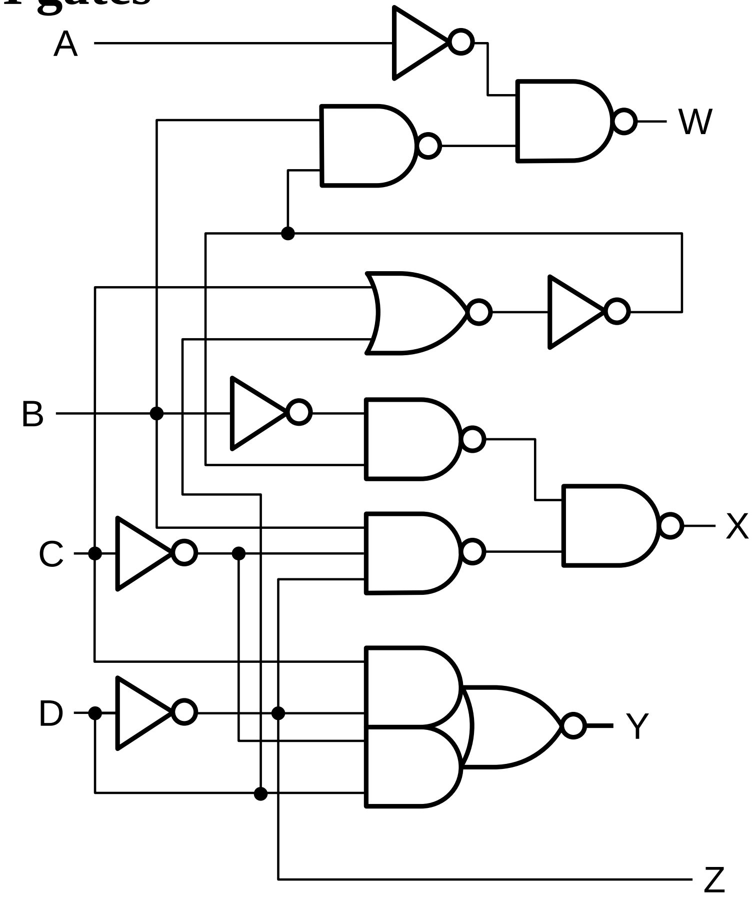
$G = \underset{\underset{T_1}{\uparrow}}{2} + \underset{\underset{\overline{T_1}}{\uparrow}}{0} + 4 + 6 + 4 + 0 = 16!$

Design Example 2 (continued)

4. Technology Mapping

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





Design Example 2 (continued)

5. Verilog Programming

```
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&~T1;
        assign  Y = C&D|~T1;
        assign  Z= ~D ;

endmodule
```

Technology Mapping

- Chip design styles
- Cells and cell libraries
- Mapping Procedures
  - To NAND gates
  - To NOR gates
  - Mapping to multiple types of logic blocks in covered in the reading supplement: Advanced Technology Mapping.

Chip Design Styles

- **Full custom** - the entire design of the chip down to the smallest detail of the layout is performed
  - Extremely labor-intensive and expensive
  - Justifiable only for dense, fast chips with high sales volume
- **Standard cell** - cells that are pre-designed can be picked up from cell libraries for designing the circuit
  - Intermediate cost
  - Less density and speed compared to full custom
- **Gate array** - an array of gates that can be used in many designs built into chip - only the interconnections between gates are specific to a design
  - Lowest cost
  - Less density compared to full custom and standard cell

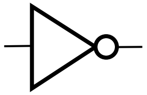
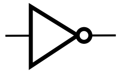
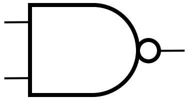
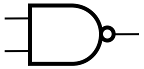
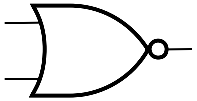
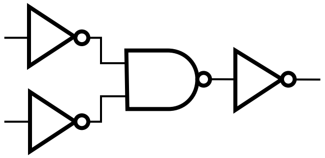
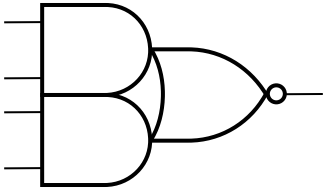
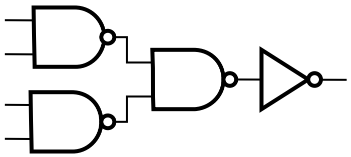
Cell Libraries

- ***Cell*** **- a pre-designed primitive block**
- ***Cell library*** **- a collection of cells available for design using a particular implementation technology**
- ***Cell characterization*** **- a detailed specification of a cell for use by a designer - often based on actual cell design and fabrication and measured values**
- **Cells are used for gate array, standard cell, and in some cases, full custom chip design**

Typical Cell Characterization Components

- Schematic or logic diagram
- Area of cell
  - Often normalized to the area of a common, small cell such as an inverter
- Input loading (in standard loads) presented to outputs driving each of the inputs
- Delays from each input to each output
- One or more cell templates for technology mapping
- One or more hardware description language models
- If automatic layout is to be used:
  - Physical layout of the cell circuit
  - A floorplan layout providing the location of inputs, outputs, power and ground connections on the cell

Example Cell Library

Cell Name	Cell Schematic	Normalized Area	Typical Input Load	Typical Input-to-Output Delay	Basic Function Templates
Inverter		1.00	1.00	0.04 + 0.012 * SL	
2NAND		1.25	1.00	0.05 + 0.014 * SL	
2NOR		1.25	1.00	0.06 + 0.018 * SL	
2-2 AOI		2.25	0.95	0.07 + 0.019 * SL	

Technology Mapping

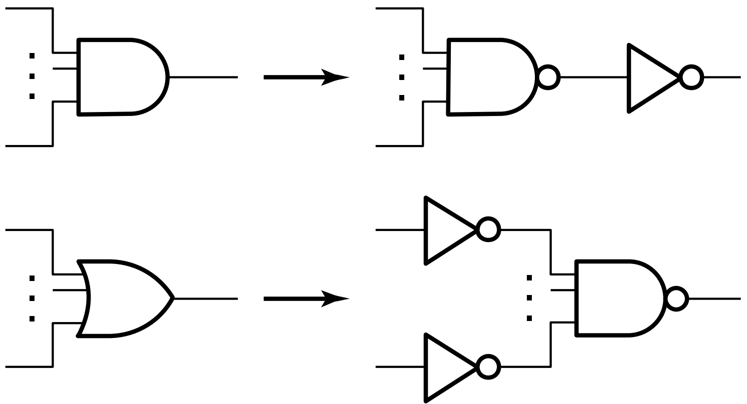
- **Technology mapping is an important step in the process of logic synthesis, which transforms a technology-independent logic description into a particular technology specification.**
- **One of the key operations during technology mapping is to recognize logic equivalence between a portion of the initial logic description and an element of the target technology.**

Mapping to NAND gates

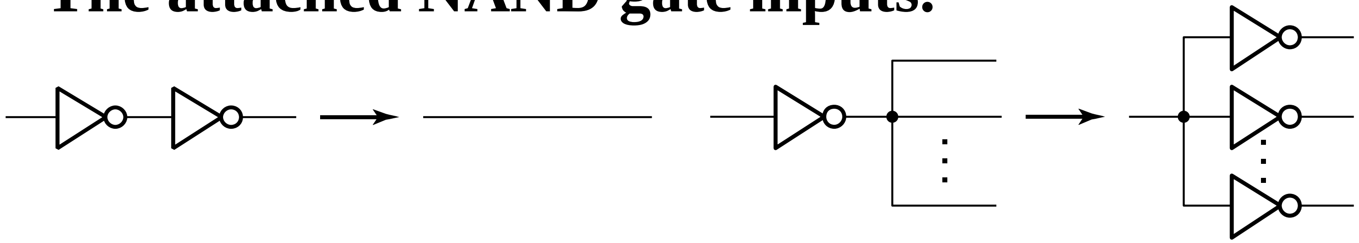
- **Assumptions:**
  - **An AND, OR, inverter schematic for the circuit is available**
  - **Cell library contains an inverter and *n*-input NAND gates, *n* = 2, 3, …**
  - **Gate loading and delay are ignored**
- **The mapping is accomplished by:**
  - **Replacing AND and OR symbols,**
  - **Pushing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

NAND Mapping Algorithm

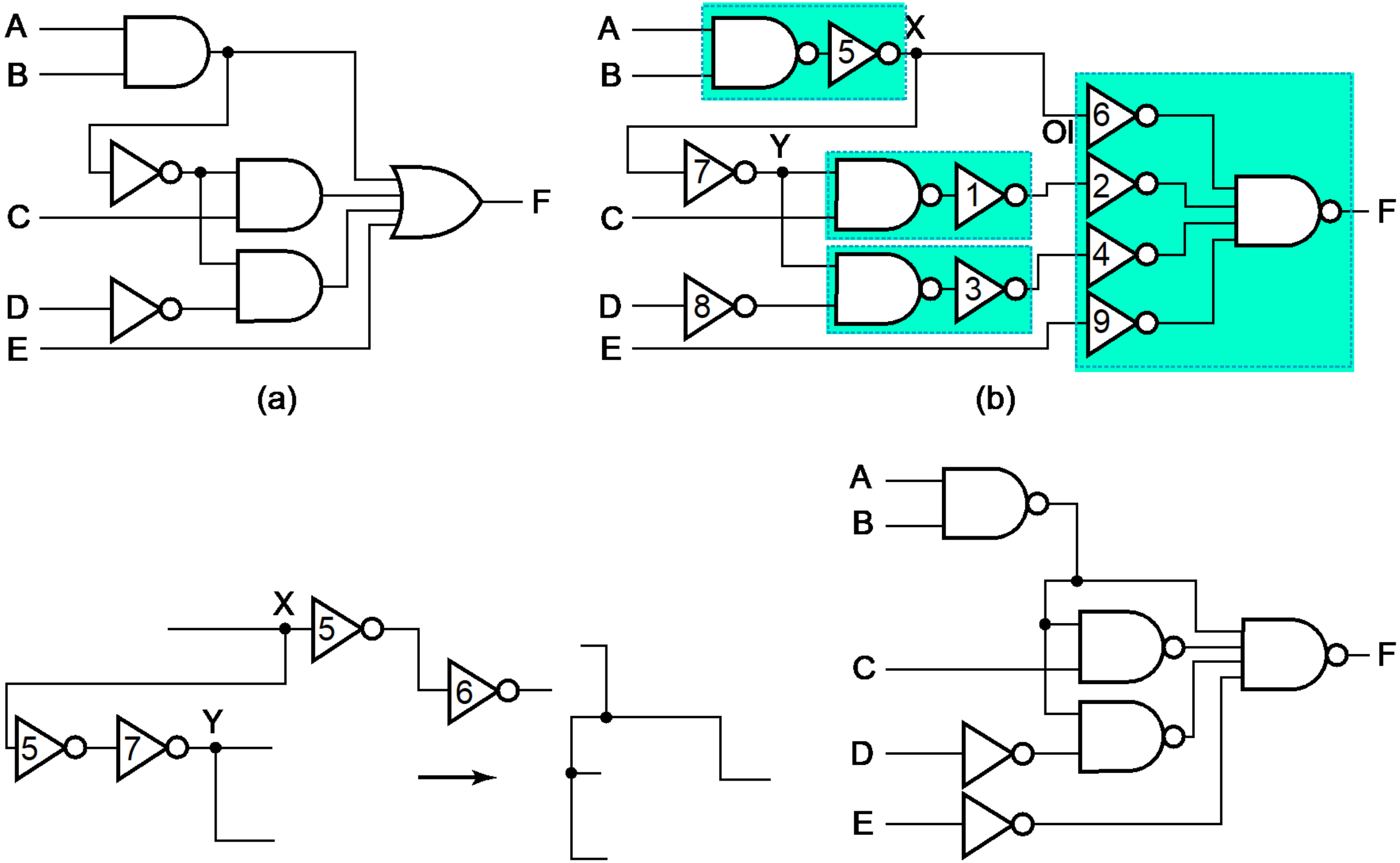
1. Replace ANDs and ORs:



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)

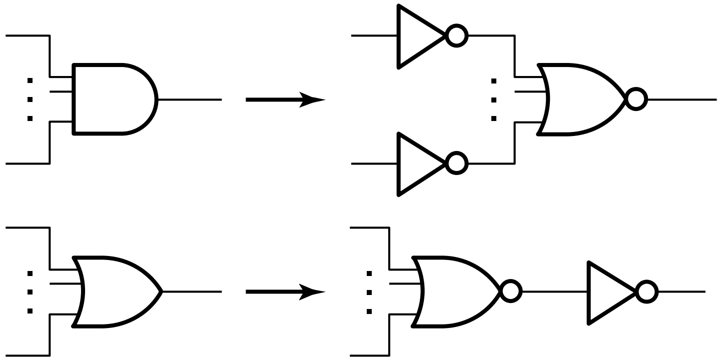
(d)

Mapping to NOR gates

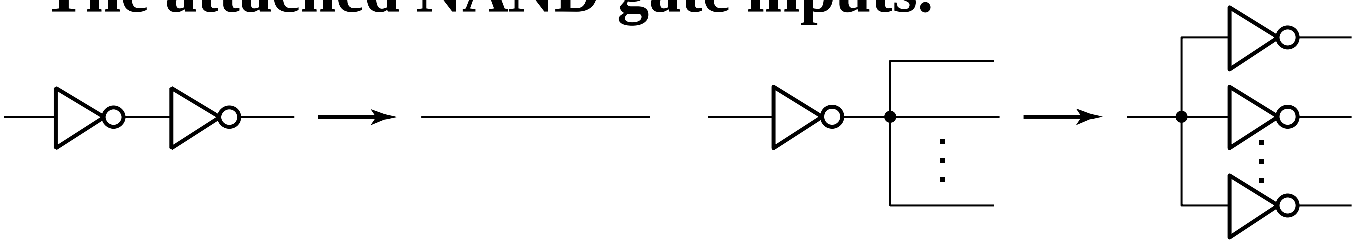
- **Assumptions:**
  - **An AND, OR, inverter schematic for the circuit is available**
  - **Cell library contains an inverter and *n*-input NOR gates, *n* = 2, 3, …**
  - **Gate loading and delay are ignored**
- **The mapping is accomplished by:**
  - **Replacing AND and OR symbols,**
  - **Pushing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

NOR Mapping Algorithm

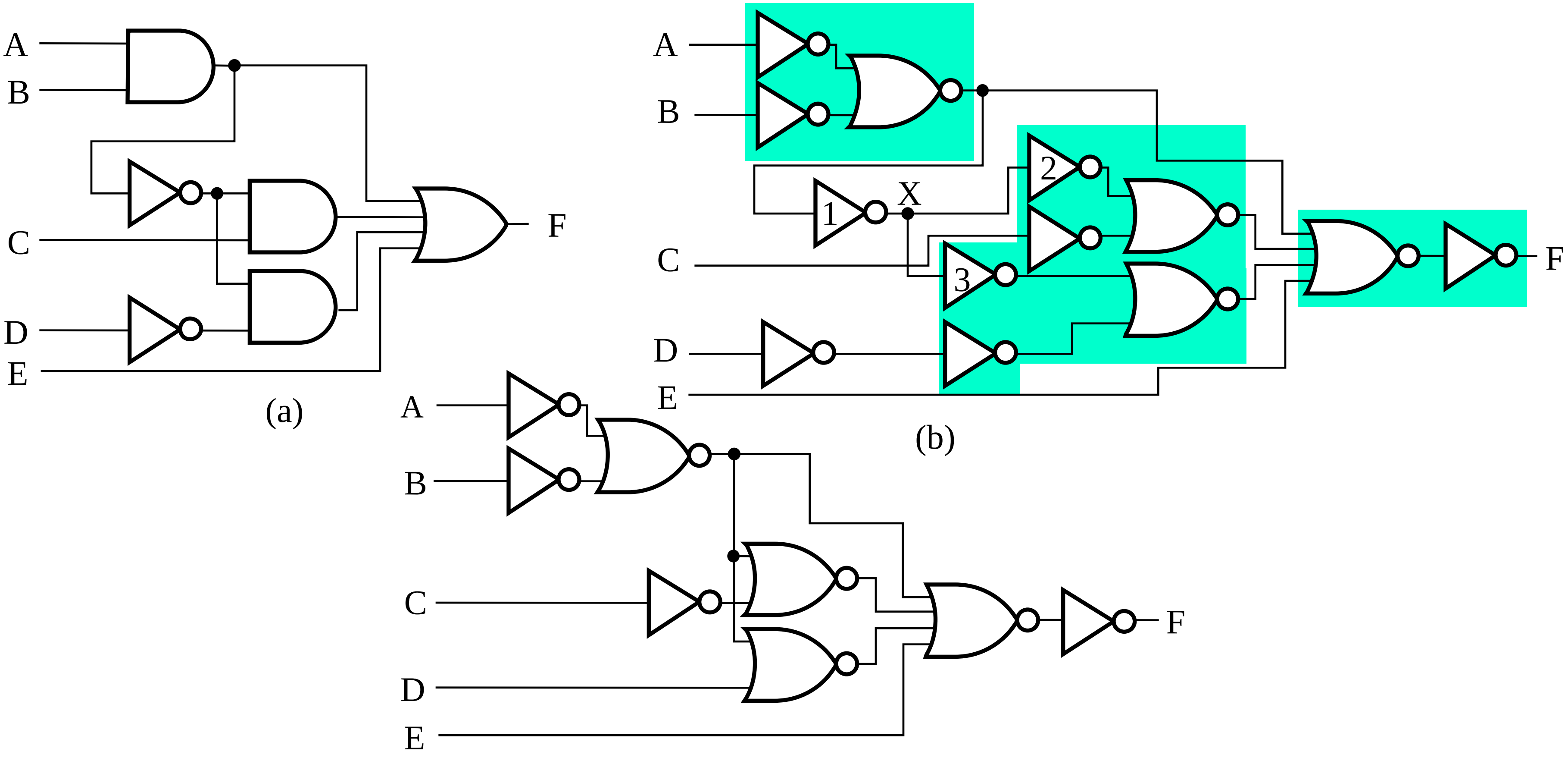
1. **Replace ANDs and ORs:**



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 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:

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

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

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

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

$$Z = \overline{D}$$

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 1 0 0

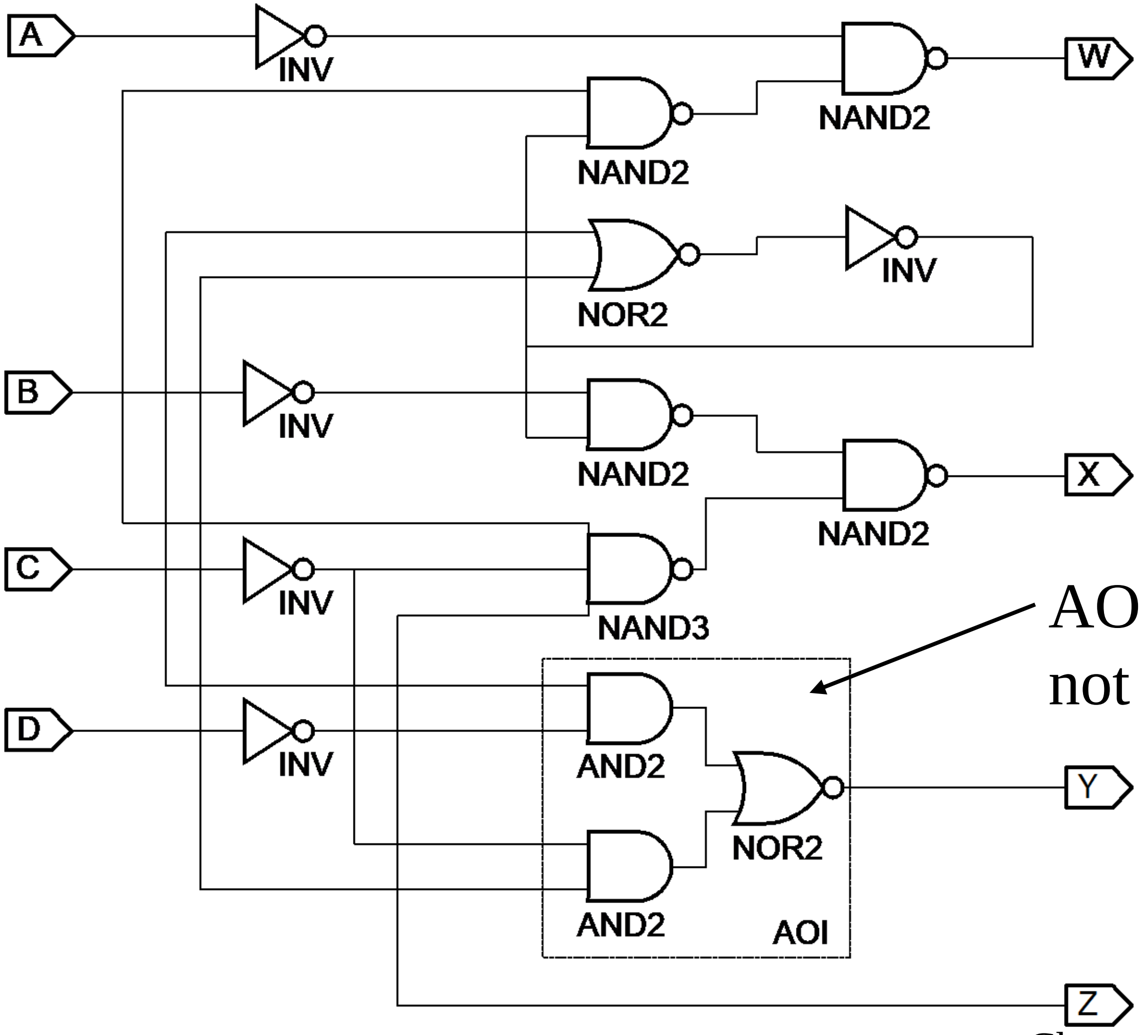
The tables match!

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

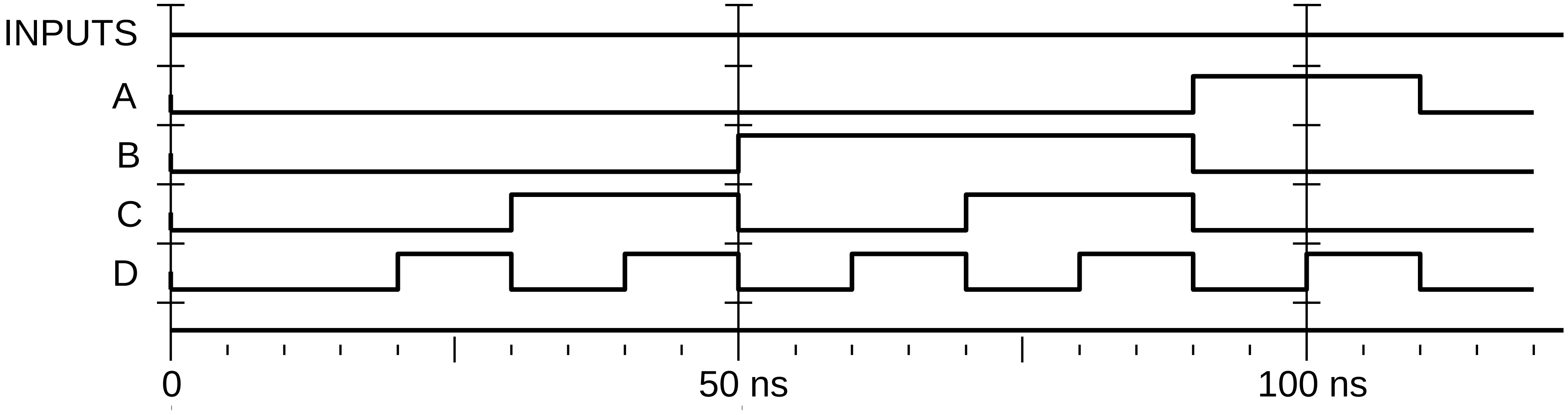
- **Enter BCD-to-Excess-3 Code Converter Circuit Schematic**



AOI symbol not available

Verification Example: Simulation

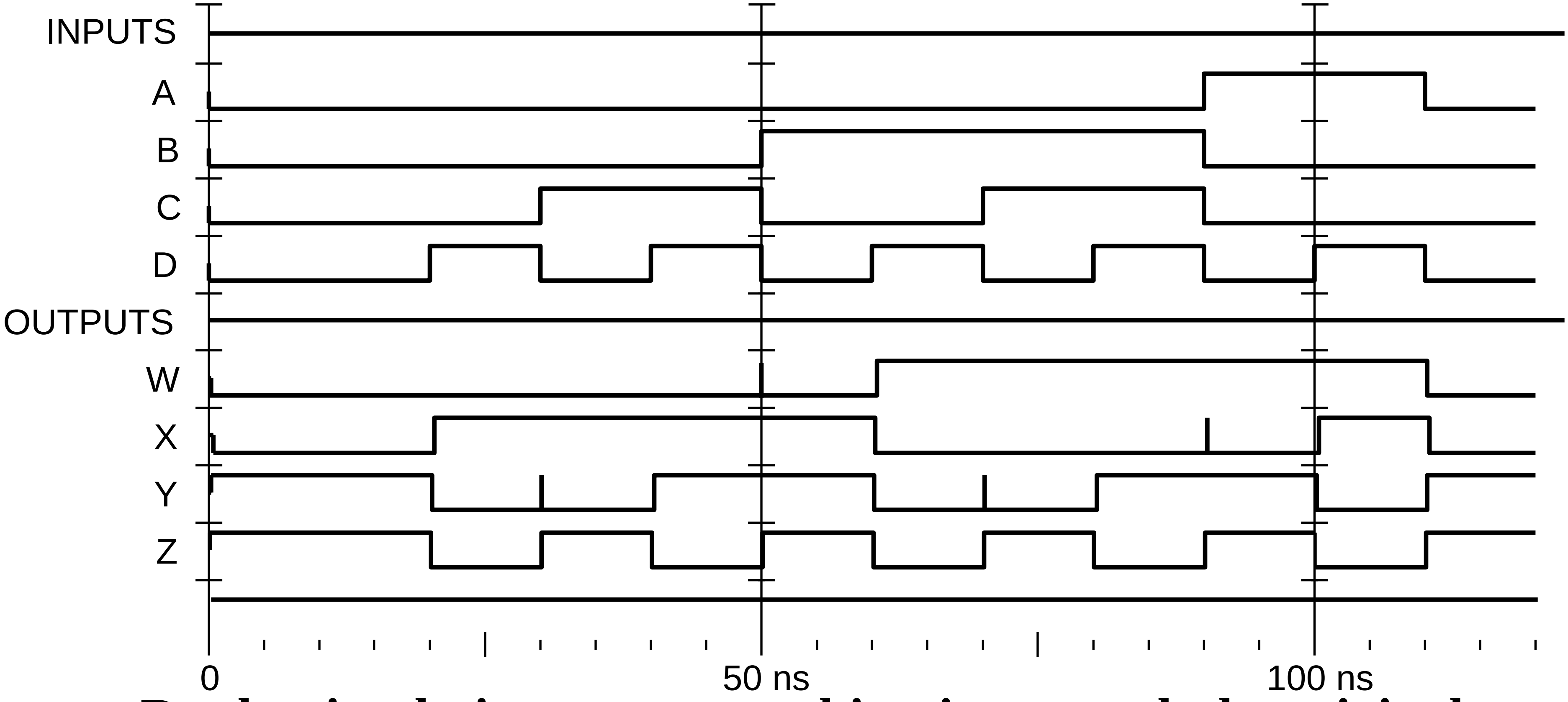
- Enter waveform that applies all possible input combinations:



- 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

Reading:

- 2.7, 3.1, 3.2, 5.1

Problem assignment:

- 2-29, 2-30 (fig.2-40中时间轴横坐标为0, 0.08, 0.16, 0.24, ……), 2-31, 5-3, 3-7, 3-8, 3-11, 3-13, 3-14, 3-16, 3-27