Introduction to Digital Logic

Prof. Nizamettin AYDIN

naydin@yildiz.edu.tr
naydin@ieee.org

1

Course Outline

1. Digital Computers, Number Systems, Arithmetic Operations, Decimal, Alphanumeric, and Gray Codes
2. Binary Logic, Gates, Boolean Algebra, Standard Forms
3. Circuit Optimization, Two-Level Optimization, Map Manipulation, Multi-Level Circuit Optimization
4. Additional Gates and Circuits, Other Gate Types, Exclusive-OR Operator and Gates, High-Impedance Outputs
5. Implementation Technology and Logic Design, Design Concepts and Automation, The Design Space, Design Procedure, The major design steps
6. Programmable Implementation Technologies: Read-Only Memories, Programmable Logic Arrays, Programmable Array Logic,Technology mapping to programmable logic devices
7. Combinational Functions and Circuits
8. Arithmetic Functions and Circuits
9. Sequential Circuits Storage Elements and Sequential Circuit Analysis
10. Sequential Circuits, Sequential Circuit Design State Diagrams, State Tables
11. Counters, register cells, buses, & serial operations
12. Sequencing and Control, Datapath and Control, Algorithmic State Machines (ASM)
13. Memory Basics

2

Introduction to Digital Logic

Lecture 5

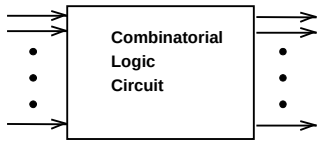
Implementation Technology and Logic Design

- Design Concepts and Automation
  - Fundamental concepts of design and computer-aided design techniques
- The Design Space
  - Technology parameters for gates, positive and negative logic and design tradeoffs
- Design Procedure
  - The major design steps: specification, formulation, optimization, technology mapping, and verification
- Technology Mapping
  - From AND, OR, and NOT to other gate types
- Verification
  - Does the designed circuit meet the specifications?

3

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:



m **Boolean Inputs** ***n*** **Boolean Outputs**

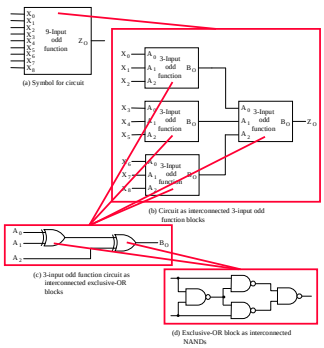
4

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 X 2 X 4 = 32 2-input NAND gates

5

Hierarchy for Parity Tree Example



6

Reusable Functions 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
- Representative Computer-Aided Design Tools:
  - Schematic Capture
  - Logic Simulators
  - Timing Verifiers
  - Hardware Description Languages
    - Verilog and VHDL
  - Logic Synthesizers
  - Integrated Circuit Layout

7

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

8

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

9

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

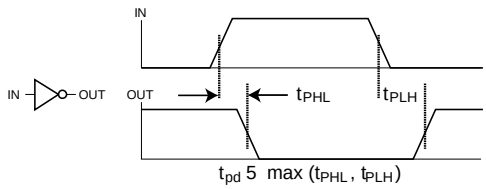
10

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, not 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.

11

Propagation Delay (continued)



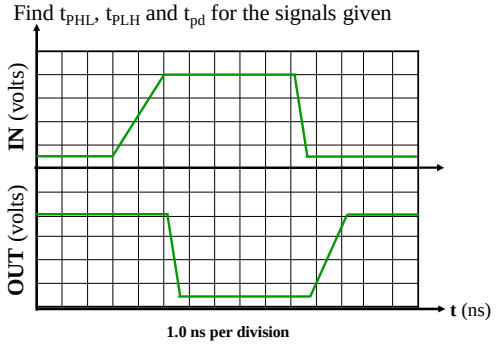
t_{pd} 5 max (t_{PHL}, t_{PLH})

- Propagation delays measured at the midpoint 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?

12

Propagation Delay Example

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



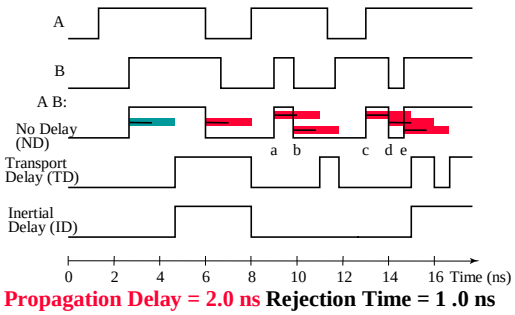
13

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

14

Delay Model Example



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

15

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*

16

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:

    $t_{pd} = 0.07 + 0.021$ 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

17

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

18

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.

19

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

20

Positive and Negative Logic (continued)

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

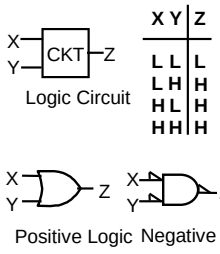
Positive Logic	(H = 1) (L = 0)
0 0	0
0 1	1
1 0	1
1 1	1

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

21

Logic Symbol Conventions

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



22

Design Trade-Offs

- 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 0.30 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

23

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
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

24

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

25

Design Example

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)

26

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 1 0 0
1 0 1 0	x x x x
1 0 1 1	x x x x
1 1 0 0	x x x x
1 1 0 1	x x x x
1 1 1 0	x x x x
1 1 1 1	x x x x

27

Design Example (continued)

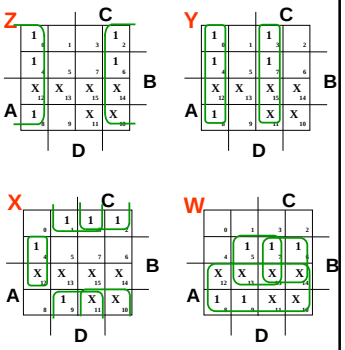
3. Optimization
   a. 2-level using K-maps

$W = A + BC + BD$

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

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

$Z = \overline{D}$



28

Design Example (continued)

3. Optimization (continued)
   b. Multiple-level using transformations

$W = A + BC + BD$

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

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

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

   - Perform extraction, finding factor:

$T_1 = C + D$

$W = A + BT_1$

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

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

$Z = \overline{D}$ $\quad G = 2 + 1 + 4 + 7 + 6 + 0 = 20$

29

Design Example (continued)

3. Optimization (continued)
   b. Multiple-level using transformations

$T_1 = C + D$

$W = A + BT_1$

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

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

$Z = \overline{D}$ $\quad G = 20$

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

$W = A + BT_1$

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

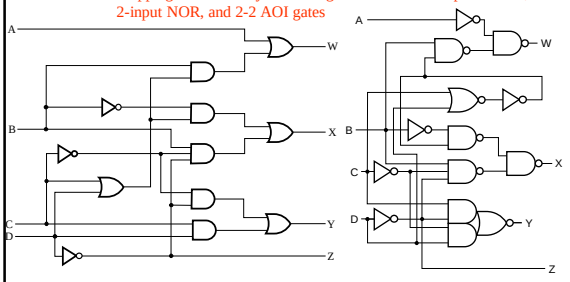
$Y = CD + \overline{T}_1$

$Z = \overline{D}$ $\quad G = 2 + 1 + 4 + 6 + 4 + 0 = 17$

30

Design Example (continued)

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



31

Technology Mapping

- Chip design styles
- Cells and cell libraries
- Mapping Techniques
  - NAND gates
  - NOR gates
  - Multiple gate types
  - Programmable logic devices

32

Chip Design Styles

- Full custom - the entire design of the chip down to the smallest detail of the layout is performed
  - Expensive
  - Justifiable only for dense, fast chips with high sales volume
- Standard cell - blocks have been design ahead of time or as part of previous designs
  - Intermediate cost
  - Less density and speed compared to full custom
- Gate array - regular patterns of gate transistors 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

33

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

34

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

35

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 1 0.012 3 SL	
2NAND		1.25	1.00	0.05 1 0.014 3 SL	
2NOR		1.25	1.00	0.06 1 0.018 3 SL	
2-2 AOI		2.25	0.95	0.07 1 0.019 3 SL	

36

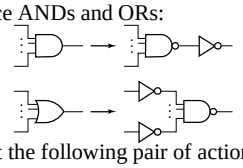
Mapping to NAND gates

- 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

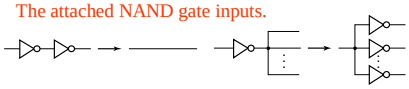
37

NAND Mapping Algorithm

1. Replace ANDs and ORs:

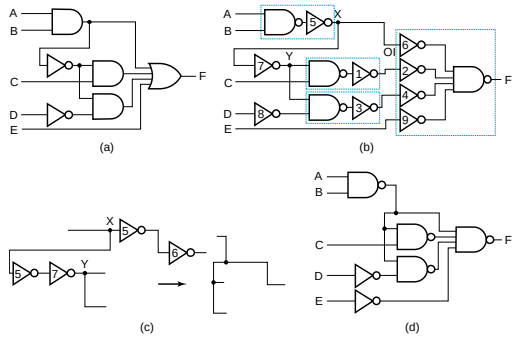


2. Repeat the following pair of actions until there is at most one inverter between :
   a. A circuit input or driving NAND gate output, and
   b. The attached NAND gate inputs.



38

NAND Mapping Example



39

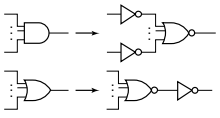
Mapping to NOR gates

- 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

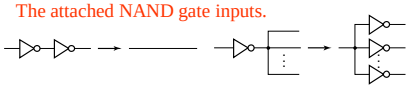
40

NOR Mapping Algorithm

1. Replace ANDs and ORs:

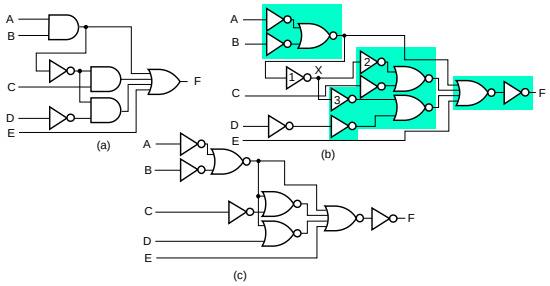


2. Repeat the following pair of actions until there is at most one inverter between :
   a. A circuit input or driving NAND gate output, and
   b. The attached NAND gate inputs.



41

NOR Mapping Example



42

Mapping Multiple Gate Types

- Algorithm is available in the Advanced Technology Mapping reading supplement
- Cell library contains gates of more than one "type"
- Concept Set 1
  - Steps
    - Replace all AND and OR gates with optimum equivalent circuits consisting only of 2-input NAND gates and inverters.
    - Place two inverters in series in each line in the circuit containing NO inverters
  - Justification
    - Breaks up the circuit into small standardize pieces to permit maximum flexibility in the mapping process
    - For the equivalent circuits, could use any simple gate set that can implement AND, OR and NOT and all of the cells in the cell library

43

Mapping Multiple Gate Types

- Concept Set 2
  - *Fan-out free subcircuit* - a circuit in which a single output cell drives only one other cell
  - Steps
    - Use an algorithm that guarantees an optimum solution for "fan-out free" subcircuits by replacing interconnected inverters and 2-input NAND gates with cells from the library
    - Perform inverter "canceling" and "pushing" as for the NAND and NOR
  - Justification
    - Steps given optimize the total cost of the cells used within "fan-out free" subcircuits of the circuit
- End result: An optimum mapping solution within the "fan-out free subcircuits"

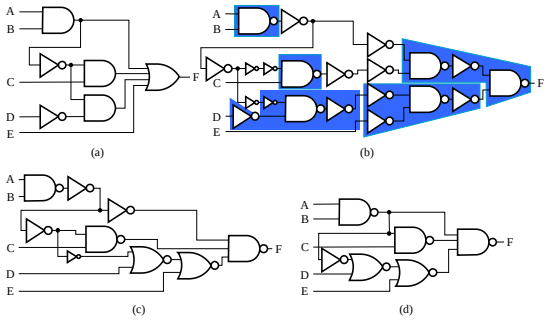
44

Example: Mapping Multiple Cell Types

- Uses same example circuit as NAND mapping and NOR mapping
- Cell library: 2-input and 3-input NAND gates, 2-input NOR gate, and inverter
- Circuits on next slide
  (a) Optimized multiple-level circuit
  (b) Circuit with AND and OR gates replaced with circuits of 2-input NAND gates and inverters (Outlines show 2-input NANDs and inverter sets mapped to library cells in next step)
  (c) Mapped circuit with inverter pairs cancelled
  (d) Circuit with remaining inverters minimized

45

Example: Mapping Multiple Gate Types



46

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!

47

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

48

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:

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

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

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

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

49

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!

50

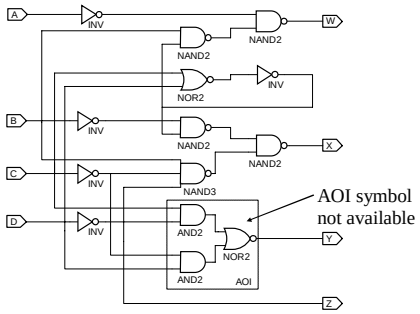
Verification Example: Simulation

- 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

51

Verification Example: Simulation

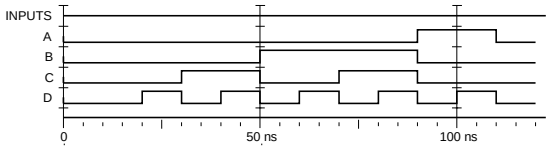
- Enter BCD-to-Excess-3 Code Converter Circuit Schematic



52

Verification Example: Simulation

- Enter waveform that applies all possible input combinations:

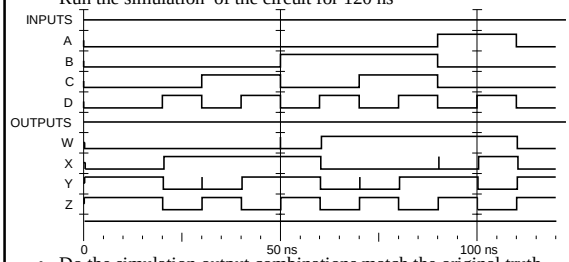


- Are all BCD input combinations present? (Low is a 0 and high is a one)

53

Verification Example: Simulation

- Run the simulation of the circuit for 120 ns



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

54