EE 459/500 – HDL Based Digital Design with Programmable Logic

Lecture 3

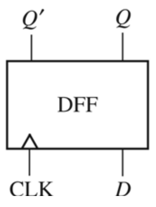
Digital Design Fundamentals

Read before class:

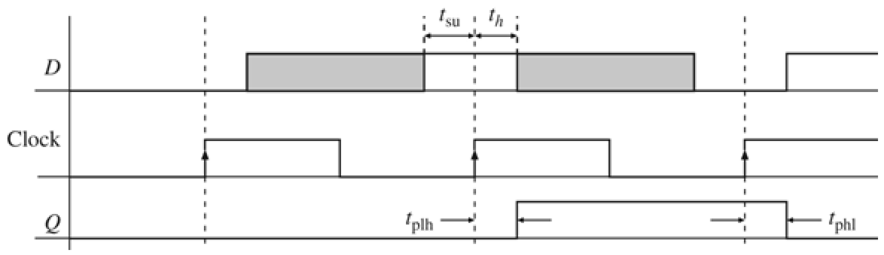
- *Chapter 1 (second half). First part of Chapter 2.*
- *Textbook used in your introductory digital logic design course*

D Flip-Flop (DFF)

- Rising edge triggered D Flip-Flop
- Timing parameters:
  - Setup time t_{su}: input must be stable before the clock edge
  - Hold time t_h: input must stay stable after the clock edge
  - Clock to Q $t_{c\text{-}q}$: maximum time for output to be stable after the clock edge

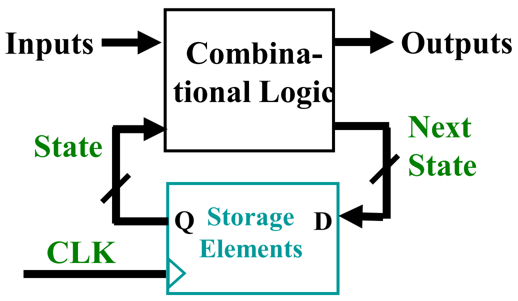


D	Q	Q^+
0	0	0
0	1	0
1	0	1
1	1	1



Finite-State Machines (FSMs)

- Finite State Machines are clocked sequential circuits

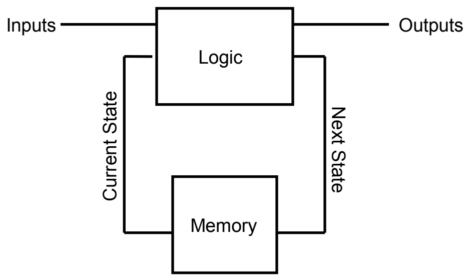


- After a clock edge, the system assumes a new state that depends on where it was before the edge (old state) and the inputs just before the edge

3

State Machines

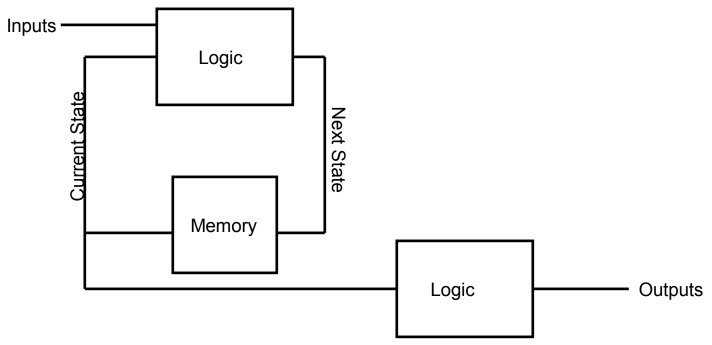
- Mealy Machine
  - Outputs are dependent on current state and inputs
  - Outputs change asynchronously with inputs



4

State Machines

- Moore Machine
  - Outputs are dependent only on current state
  - Outputs are fixed during clock cycle



5

State Graphs (state diagrams)

- Symbolic abstract, graphic representation of behavior
- Consists of:
  - Nodes – A node represents a unique state; has unique symbolic name
  - Arcs – An arc represents a transition from one state to another; labeled with condition that will cause the transition
- Output values also specified:
  - Under the condition expression of transition arcs – for Mealy machines whose output depend on input and state
  - Inside the state bubble – for Moore machines whose output depend on state only

FIGURE 1-20: Mealy State Graph for Sequence Detector

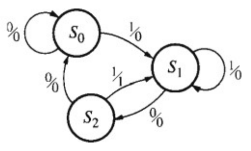
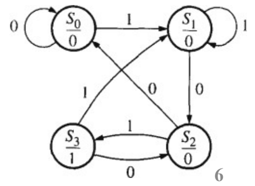


FIGURE 1-28: State Graph of the Moore Sequence Detector



6

State Machine Design

- Make sure
  - all states are represented
  - all possible inputs are taken into account for state transitions
  - there is an exit out of each state
  - there are no conflicts in state transitions
- Encodings:
  - Binary
  - One-Hot
  - …

7

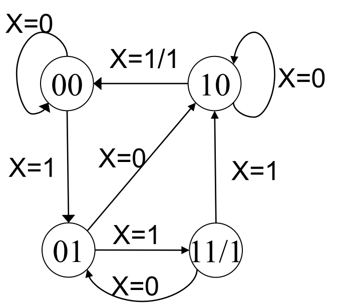
State Encoding

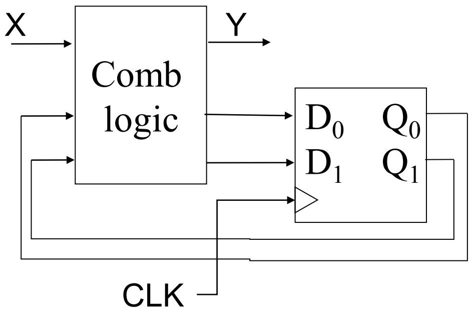
- Each state Si is represented by a binary pattern Pi, where i is an arbitrary index.
- A mapping from the state index i to Pi is the state encoding function E.
- Binary (sequential) encoding: E(i) = i; i = 1,2,…,n
- One-hot encoding: E(i) = 2i; i = 0,1,…,n-1
- Others: grey, johnson, hamming-2, etc.

8

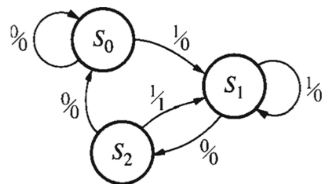
Finite State Machine: Generic Example

- There are automated procedures to build (synthesize) the logic for finite state machines
- One way of describing a FSM, in terms of transitions on each clock edge





Mealy machine design example1: Sequence detector (detect sequence "101")



State graph

Present State	Next State $X=0$	Next State $X=1$	Present Output $X=0$	Present Output $X=1$
S_0	S_0	S_1	0	0
S_1	S_2	S_1	0	0
S_2	S_0	S_1	0	1

State transition table

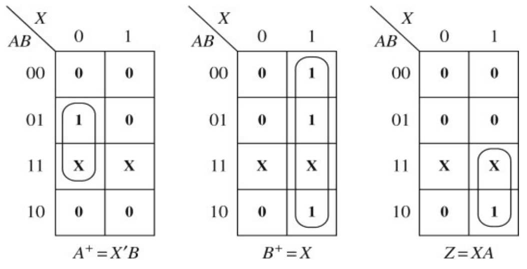
State encoding

AB	A^+B^+ $X=0$	A^+B^+ $X=1$	Z $X=0$	Z $X=1$
00	00	01	0	0
01	10	01	0	0
10	00	01	0	1

Transition table with encoded states

Mealy machine design example1: Sequence detector

K-Maps for Next States and Output of Sequence Detector



AB \ X	0	1
00	0	0
01	1	0
11	X	X
10	0	0

$A^+ = X'B$

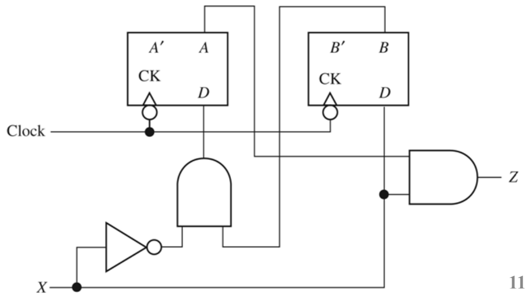
AB \ X	0	1
00	0	1
01	0	1
11	X	X
10	0	1

$B^+ = X$

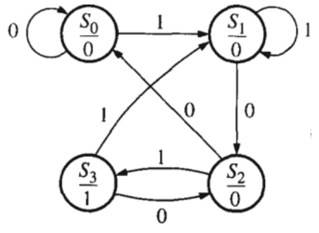
AB \ X	0	1
00	0	0
01	0	0
11	X	X
10	0	1

$Z = XA$

Hardware implementation with two DFFs



Moore machine design example1: Sequence detector



State graph

Present State	Next State $X = 0$	Next State $X = 1$	Present Output (Z)
S_0	S_0	S_1	0
S_1	S_2	S_1	0
S_2	S_0	S_3	0
S_3	S_2	S_1	1

State transition table

State encoding

AB	A^+B^+ $X = 0$	A^+B^+ $X = 1$	Z
00	00	01	0
01	11	01	0
11	00	10	0
10	11	01	1

Transition table with encoded states

Moore machine design example1: Sequence detector

- Exercise:
  - Build K-Maps and find equations of Z, (D1,D2) or (J1,K1,J2,K2) when:
    - DFFs are used
    - JKFFs are used
  - Draw the circuit diagrams in both cases

13

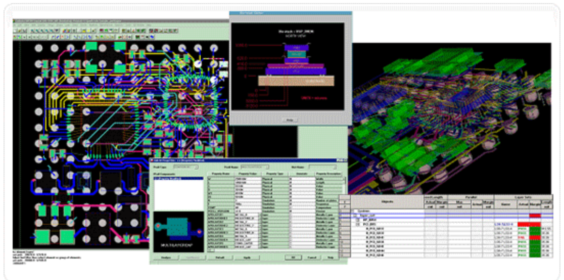
Part 3

- Logic Design and Implementation Technology
  - Design concepts and design automation
  - Design space: parameters and tradeoffs
  - Design procedure (design flow)
    - Major design steps: specification, formulation, optimization, technology mapping, and verification

14

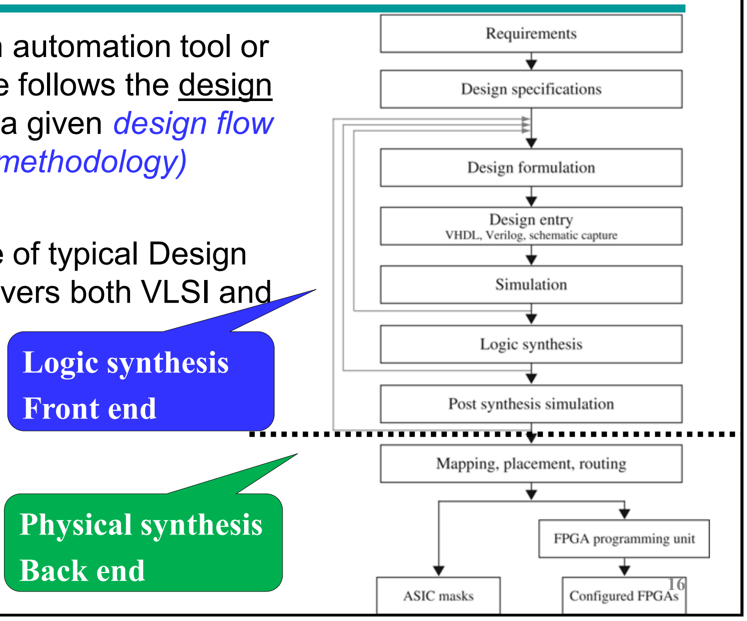
Design Automation

- Design automation: the process (activities) of developing/architecting and implementing EDA tools
- Electronic design automation (EDA) is a category of software tools for designing electronic systems such as printed circuit boards and integrated circuits (ICs). The tools work together in a design flow that chip designers use to design and analyze entire chips
- Use of EDA tools effectively automate the design process (much of it done manually in the old days)
- EDA companies: Cadence, Synopsis/Magma, Mentor Graphics, etc.



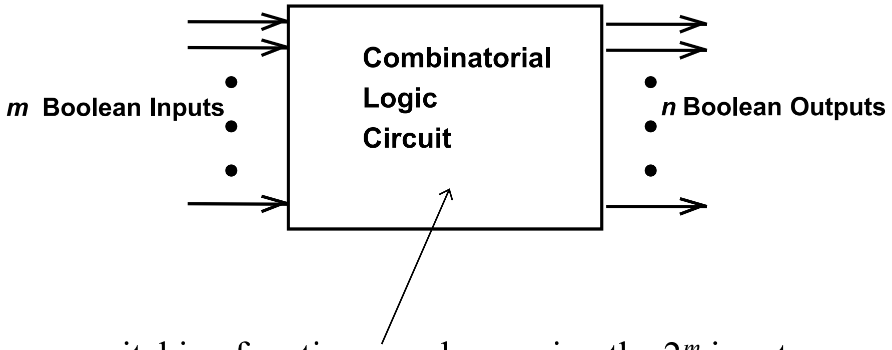
Design Flow (Design Methodology)

- A design automation tool or tool-suite follows the <u>design steps</u> of a given *design flow (design methodology)*
- Example of typical Design Flow (covers both VLSI and FPGA):



A) Combinational Circuits

- A block diagram of combinational logic circuit:



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

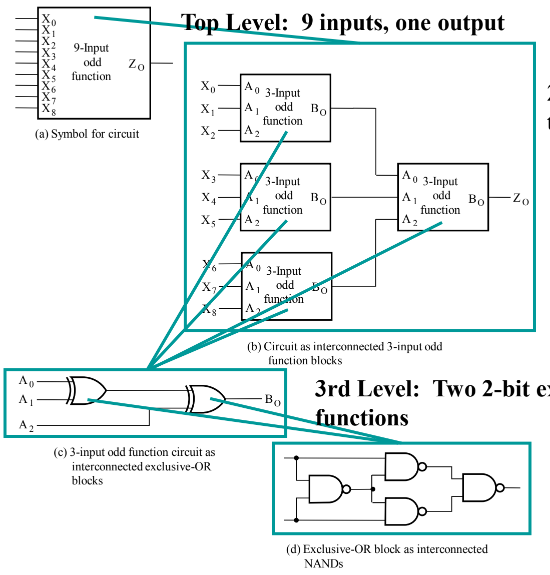
17

Concept: Hierarchical Design

- To control the complexity of the function mapping inputs to outputs:
  - Decompose the function into smaller pieces – *blocks*
    - *ALU, Multiplier and Accumulator, etc*
  - Decompose each block's function into smaller blocks, repeating as necessary until all blocks are small enough
    - Adder ➔ Gates
  - Any block not decomposed is called a *primitive block*
  - The collection of all blocks including the decomposed ones is a **hierarchy**

18

Example: Hierarchy for Parity Tree



(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

Top Level: 9 inputs, one output

2nd Level: Four 3-bit odd parity trees in two levels

Design requires:
4 X 2 X 4 = 32
2-input NAND gates

3rd Level: Two 2-bit exclusive-OR functions

Primitives:
Four 2-input NAND gate

19

(Technology) Parameters

- Specific characteristic parameters for gate implementation technologies:
  - *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
  - *Propagation Delay* – The time required for a change in the value of a signal to propagate from an input to an output
  - *Cost for a gate* - a measure of the contribution by the gate to the "cost" of the integrated circuit
  - *Power Dissipation* – the amount of power drawn from the power supply and consumed by the gate

20

Fan-out & Delay

- Fan-out can be defined in terms of a standard load (SL)
  - 1 standard load equals the load contributed by the input of 1 inverter.
- Maximum fan-out is the number of standard loads the gate can drive without exceeding its specified maximum transition time
- Gate's propagation delay depends on the fan-out loading at the gate's output
- Example:
  - Equation to estimate propagation delay t_{pd} for a NAND gate with 4 inputs is:

    $t_{pd} = 0.07 + 0.021$ SL ns
  - SL: the number of standard loads the gate is driving, i.e., its fan-out in standard loads

Cost

- In an IC:

$$\boxed{\text{Cost of a gate}} \propto \boxed{\text{Chip area of the gate}} \propto \left\{ \begin{array}{c} \boxed{\text{\#/size of transistors}} \\ + \\ \boxed{\text{wiring area}} \end{array} \right. \propto \boxed{\text{Gate input count}}$$

- If the actual chip layout area occupied by the gate is known, it is a far more accurate measure

Gate Input Cost

- Gate input costs - the #of inputs to the gates corresponding exactly to the given equations. (G - inverters not counted, GN - inverters counted)
- For SOP and POS equations, it can be found by the sum of:
  - all literal appearance – literal cost
  - the number of terms excluding terms consisting only of a single literal, (G)
- Example:
  - $F = BD + A\bar{B}C + A\bar{C}\bar{D}$ G=11,GN=14
  - $F = BD + A\bar{B}C + A\bar{B}\bar{D} + AB\bar{C}$ G= ,GN=
  - $F = (A + \bar{B})(A + D)(B + C + \bar{D})(\bar{B} + \bar{C} + D)$ G= ,GN=
  - Which solution is best?

23

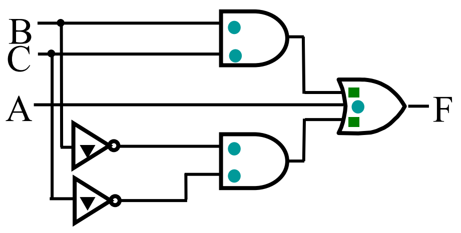
Cost Criteria (contd.)

$F = \overset{\bullet}{A} + \overset{\bullet}{B}\underset{\blacksquare}{\,}\overset{\bullet}{C} + \overset{\blacktriangledown}{\overset{\bullet}{\bar{B}}}\underset{\blacksquare}{\,}\overset{\blacktriangledown}{\overset{\bullet}{\bar{C}}}$

$L = 5$

$G = L + 2 = 7$

$GN = G + 2 = 9$



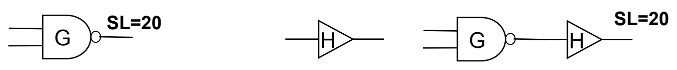
- L: counts the AND inputs and the single literal OR input
- G: adds the remaining OR gate inputs
- GN: adds the inverter inputs

24

Design Trade-Offs

- Cost - performance tradeoffs

  Gate-level example:

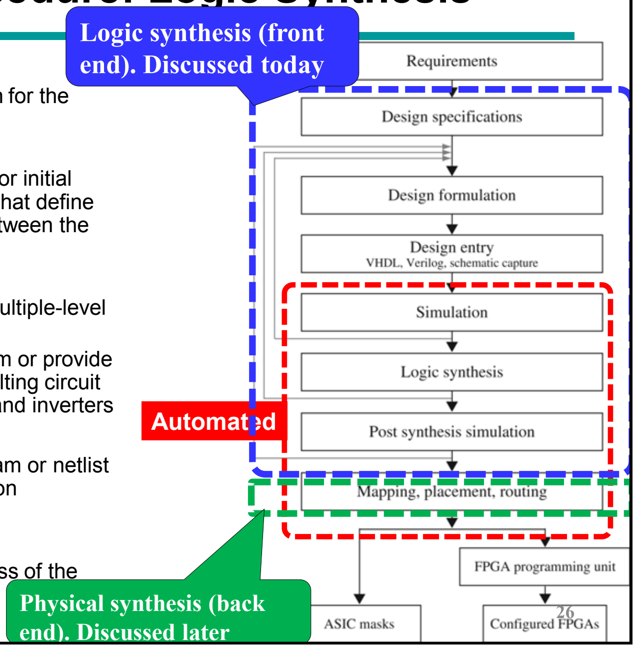


T_{pd}=0.45ns, Cost=2.0 Cost=1.5 T_{pd}=0.33ns, Cost=2.0+1.5=3.5

- Tradeoffs can be accomplished at much higher design level in the hierarchy
- Constraints on cost and performance have a major role in making tradeoffs

Design Procedure: Logic Synthesis

1. Specification
   - Write a specification for the circuit
2. Formulation
   - Derive a <u>truth table</u> or initial Boolean <u>equations</u> that define the relationships between the inputs and outputs
3. Optimization
   - Apply 2-level and multiple-level optimization
   - Draw a logic diagram or provide a <u>netlist</u> for the resulting circuit using ANDs, ORs, and inverters
4. Technology Mapping
   - Map the logic diagram or netlist to the implementation technology selected
5. Verification
   - Verify the correctness of the final design


Logic synthesis (front end). Discussed today
Requirements
Design specifications
Design formulation
Design entry
VHDL, Verilog, schematic capture
Simulation
Logic synthesis
Automated
Post synthesis simulation
Mapping, placement, routing
FPGA programming unit
ASIC masks
Configured FPGAs
Physical synthesis (back end). Discussed later

Design Example

1. Specification

BCD to Excess-3 code converter: Transforms BCD code for the decimal digits to Excess-3 code

- BCD code words for digits 0-9: 4-bit patterns 0000 to 1001, respectively
- Excess-3 code words for digits 0-9: 4-bit patterns consisting of 3 (binary 0011) added to each BCD code word

- *Note: because we assume inputs and outputs are provided and implemented in parallel, our circuit can be designed as a simple combinational circuit. If, instead inputs are available in series, then we must design a sequential circuit instead (like described on page 19 in textbook)!*

Design Example (Contd.)

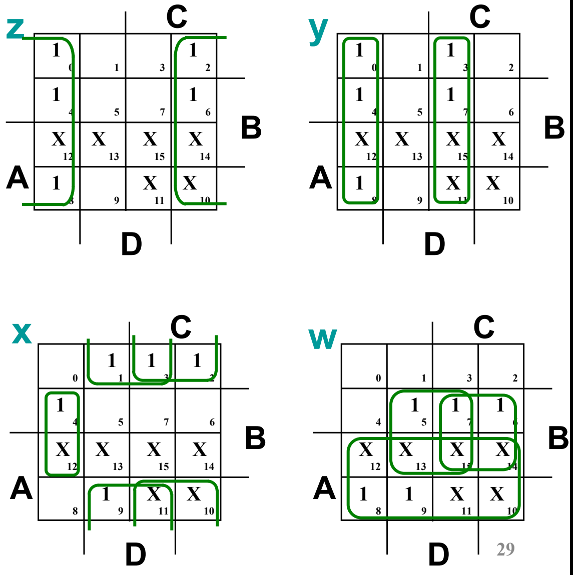
2. Formulation

- Conversion of 4-bit codes can be easily formulated by a truth table
- BCD Variables: A,B,C,D
- Excess-3 Variables: W,X,Y,Z
- BCD Don't Cares - 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 0 1 1

Design Example (Contd.)

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 (Contd.)

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

$W = A + BC + BD$
$X = \overline{B}C + \overline{B}D + B\overline{C}\overline{D}$
$Y = CD + \overline{C}\overline{D}$
$Z = \overline{D}$ **G = 7 + 10 + 6 + 0 = 23**

- Perform extraction, finding factor:

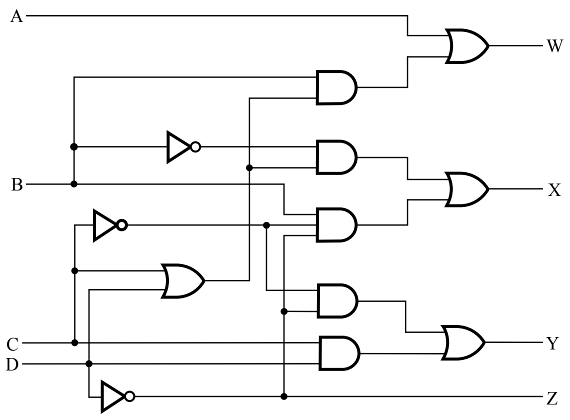
$T_1 = C + D$
$W = A + BT_1$
$X = \overline{B}T_1 + B\overline{C}\overline{D}$
$Y = CD + \overline{C}\overline{D}$
$Z = \overline{D}$ **G = 2 + 4 + 7 + 6 + 0 = 19**

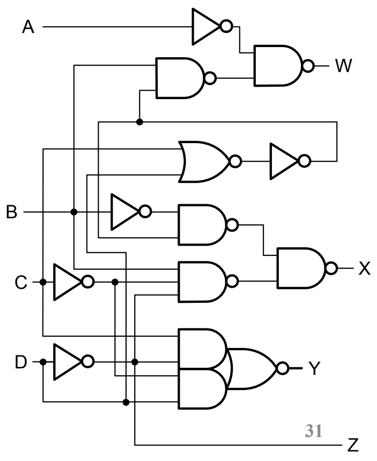
- An additional extraction using a Boolean transformation: ($\overline{C}\overline{D} = \overline{C + D} = \overline{T_1}$)

$W = A + BT_1$
$X = \overline{B}T_1 + B\overline{T_1}$
$Y = CD + \overline{T_1}$
$Z = \overline{D}$ **G = 2 + 1 + 4 + 5 + 4 + 0 = 16**

Design Example (Contd.)

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



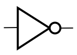
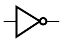
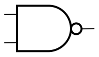
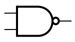
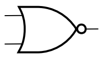
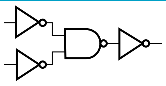
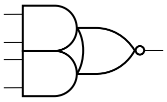
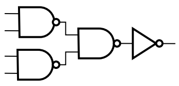


Concept: Cell Libraries

- *A* collection of *cells* using a particular implementation *technology*
- *Cell characterization* - a detailed *specification* of a cell - often based on actual cell design and fabrication and measured values
  - Function: Schematic or logic diagram
  - Parameters: Area, Input loading, Delays
  - 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

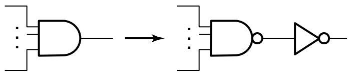
functions parameters templates

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	

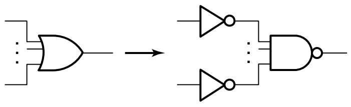
33

Mapping to NAND gates

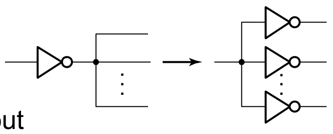
- Assumptions:
  - Cell library contains an inverter and *n*-input NAND gates, *n* = 2, 3, …



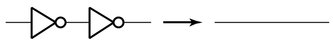
- NAND Mapping algorithms



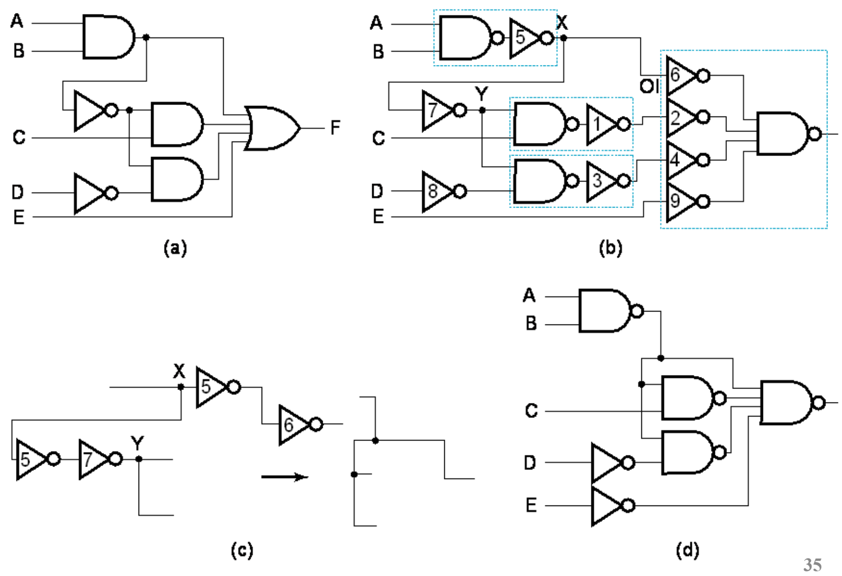
1. Replace ANDs and ORs:
2. Repeat the following pair of actions until there is at most one inverter between:
   - A circuit input or driving NAND gate output
   - The attached NAND gate inputs



- Pushing inverters through circuit fan-out points
- Canceling inverter pairs



NAND Mapping Example



Verification Example: Manual Analysis

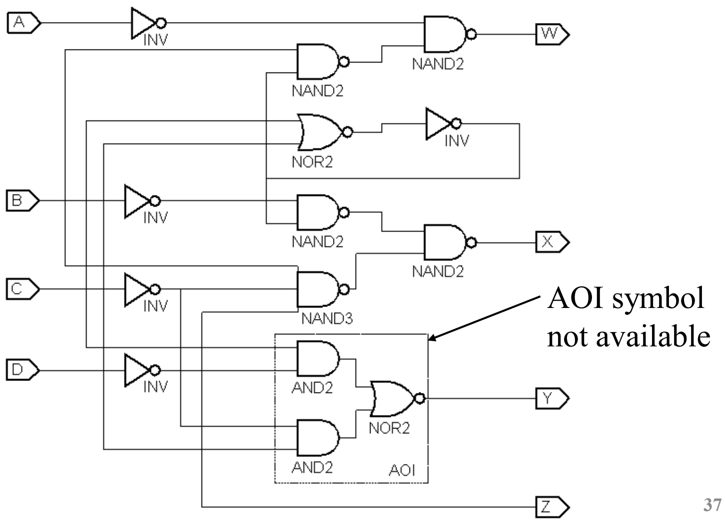
- Find the circuit truth table from the equations and compare to specification truth table:

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

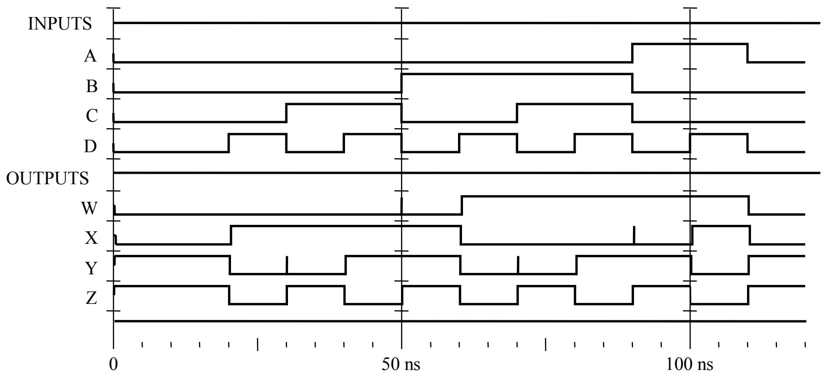
- Enter BCD-to-Excess-3 Code Converter Circuit Schematic



37

Verification Example: Simulation

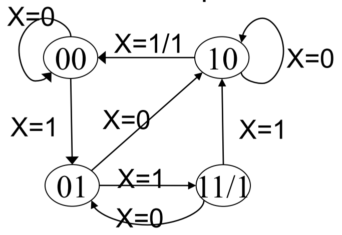
- Enter waveform that applies all possible input combinations
  - Are all BCD input combinations present?
- Run the simulation of the circuit for 120 ns

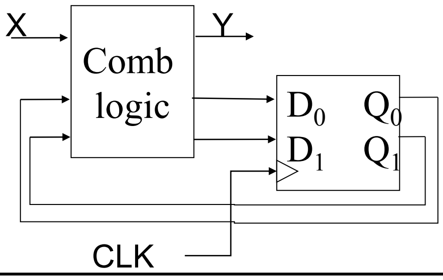


- Do simulation output match the original truth table?

B) Sequential Circuits (FSMs)

- Next state and output determination: specification





Q_1	Q_0	X	D_1	D_0	Y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	0	0
0	1	1	1	1	0
1	0	0	1	0	0
1	0	1	0	0	1
1	1	0	0	1	1
1	1	1	1	0	1

$D_1=XQ_0+Q_1'Q_0+X'Q_1Q_0$

$D_0=XQ_1'+X'Q_1Q_0$

$Y=XQ_1+Q_1Q_0$

Exercise

- For the “BCD to Excess-3” Mealy machine design example on page 9 of the textbook, identify and discuss each of the design steps: *specification, formulation, optimization, technology mapping, and verification*
- If some of these steps is missing, then investigate and propose how to do it

Summary

- Most of real digital systems are sequential circuits
- Design process follows a set of typical steps of given design flow (design methodology)
- EDA tools automate most of the design steps
  - However, user has a lot of flexibility to manually interfere or tune “tool knobs” to drive the design process towards achieving certain design goals/costs

41