Chapter 3

Combinational Logic Design
Chapter Overview

3-1 Design Concepts and Automation
- Hierarchy & top-down design
- Computer-aided design (HDL & logic synthesis)

3-2 The Design Space
- Properties of logic gates
- Technology parameters: fan-in, fan-out, & propagation delay
- Positive and negative logic concepts

* 補充資料：Analysis Procedure of Combinational Circuit

3-3 Design Procedure
- Specification, Formulation, Optimization

3-4 Technology Mapping

3-5 Verification

3-6 Programmable Implementation Technologies
- ROM, PLA, PAL

3-7 Chapter Summary
3-1 Design Concepts and Automation

- **Design concepts:**
  - Design hierarchy
  - Top-down design

- **Design automation:**
  - Computer-aided design (CAD) tools
  - Hardware description languages (HDLs)
  - Logic synthesis
Logic Circuits

Logic ckts for digital systems:

- Combinational ckt
  - consists of logic gates whose outputs at any time are determined by logic ops on the input values

- Sequential ckt (Ch6)
  - whose outputs are functions of the inputs and the bit values in the storage elements, which are a function of previously applied inputs and stored values.
Combinational Circuits

**Combinational ckt:**
- consists of input variables, output variables, logic gates, and interconnections

```
  n inputs                      m outputs
               Combinaional
               circuit
```

- can be specified by
  i. a truth table: lists the output values for each combination of the input variables
  ii. $m$ Boolean functions: one for each output variable
      Each function is expressed as a function of the $n$ input variables.
A. Design Hierarchy

- Hierarchical design: “divide and conquer” approach
  - For a complex digital system:
    
    Ckt $\rightarrow$ Blocks $\rightarrow$ Smaller blocks $\rightarrow$ …
    
    $\rightarrow$ Predefined blocks
    
    (some of which may be primitive blocks)
    
    AND, OR, NOT, …
Example: A 9-Input Odd Function

Divide & conquer approach:
Diagrams representation:

(a) 9-input odd function
   ├── 3-input odd function
   │    └── XOR
   │         └── NAND
   └── 3-input odd function
       └── XOR

(b) 9-input odd function
    └── 3-input odd function
        └── XOR
Important Properties

- Important properties of hierarchical design:
  - A hierarchy reduces the complexity required to represent the schematic diagram of a ckt.
  - A hierarchy ends at a set of predefined blocks.
  - The block is reusable.
    - It can be used in multiple places in the ckt design and in the design of other ckt designs as well.
B. Top-Down Design

- **Top-down design:**
  - The ckt function is specified text or a hardware description language (HDL), plus constraints on cost, performance, and reliability.
  - At high levels of the design, the ckt is repeatedly divided into blocks as necessary until the blocks are small enough to perform logic design.

- **Bottom-up design** (Ch4, 5, …)
C. Computer-Aided Design (CAD)

- CAD tools:
  - Schematic capture tools:
    - support the drawing of blocks and interconnections at all levels of the hierarchy.
    - At the level of primitives and functional blocks, libraries of graphics symbols are provided.
  - Logic simulator:
    - verify the behavior and the timing of the hierarchical blocks and the entire ckt
  - Logic synthesizer:
    - optimize design being generated automatically from HDL specifications in physical area or delay
D. Hardware Description Languages

- **Hardware description language (HDL):**
  - Two widely-used HDLs: VHDL & Verilog
  - resembles programming language, but is specifically oriented to describing hardware structures and behavior
    - Main difference:
      - HDL: represents extensive parallel op
      - Most programming languages: represent serial op
  - can represent
    - schematic information: called structure description or netlist
    - Boolean equations
    - truth tables
    - complex ops: e.g., arithmetic ops
Levels of a HDL:
- Behavior level
- Register-transfer level (RTL)
- Gate level
- Transistor level

- can be logic synthesized
E. Logic Synthesis

- Logic synthesis:
  - transforms an RTL (register transfer level) description of a ckt in an HDL into an optimized netlist representing storage elements and combinational logic.
  - allows exploration of the cost/performance trade-offs
High-Level Flow for Logic Synthesis Tool

1. HDL Description of Circuit
2. Translation
3. Intermediate Representation
4. Preoptimization
5. Optimization
6. Technology Mapping
7. Netlist
8. Electronic, Speed, and Area Constraints
9. Technology Library
Design space:

- IC: gate properties, levels of integration, ckt technologies
- Technology parameters: fan-in, fan-out, & propagation delay
- Positive and negative logic concepts
A. Gate Properties

- Digital ckts are constructed w/ integrated ckts.

- Integrated circuit (IC):
  - is a silicon semiconductor crystal (chip) containing the electronic components for the digital gates and storage elements.
  - The various components are interconnected on the chip.
  - The chip is mounted in a ceramic or plastic container, and connections are welded from the chip to the external pins.
B. Levels of Integration

- Levels of integration: ckt complexity
  (# of logic gates in a single silicon chip)
  - SSI: small-scale integration, <10 gates
  - MSI: medium-scale integration, 10 ~ 100 (Ch 4 & 5)
    - performs specific elementary digital functions
    - e.g.: addition of 4 bits
  - LSI: large-scale integration: 100 ~ x000
    - e.g.: small processors, small memories, & programmable modules
  - VLSI: very large-scale integration, x000 ~
    - e.g.: complex microprocessors, digital signal processing chips
C. Circuit Technologies

- **Implementation technology:**
  - TTL: transistor-transistor logic
  - ECL: emitter-coupled logic (speed)
  - MOS: metal-oxide semiconductor (density)
  - CMOS: complementary MOS (high density, high performance, & low power consumption)
  - GaAs: Gallium Arsenide (very high speed)
  - SiGe: Silicon Germanium (very high speed)

- The basic ckt in each family is a **NAND**, **NOR**, or **inverter** gate.
D. Technology Parameters

- **Fan-in**: # of inputs available on a gate
- **Fan-out**: # of standard loads driven by a gate output
  - *max fan-out*: the fan-out that the output can drive w/o impairing gate performance
- **Noise margin**: the max external noise voltage superimposed on a normal input value that will not cause an undesirable change in the ckt output
- **Cost** for a gate: is usually based on the area occupied by the layout cell (∝ the size of the transistors & the wiring in the gate layout)
- **Propagation delay**: the time required for a change in value of a signal to propagate from input to output
  - The operating speed of a ckt is inversely related to the longest propagation delays through the gates of the ckt.
- **Power dissipation**: the power drawn from the power supply and consumed by the gate
  - must be considered in relation to the operating temperature and cooling

J.J. Shann 3-20
For high-speed technologies, fan-in is often restricted on gate primitives to \( \leq 4 \) or 5.

E.g.: Implementation of a 7-input NAND gate using NAND gates with 4 or fewer inputs
Propagation Delay

- 3 propagation delay parameters:
  - high-to-low propagation time $t_{PHL}$
  - low-to-high propagation time $t_{PLH}$
  - propagation delay $t_{pd} : \max\{t_{PHL}, t_{PLH}\}$

E.g.:
2 different simulation models of propagation delay:

- **Transport delay:**
  - The change in an output in response to the change of an input occurs after a specified propagation delay.

- **Inertial delay:**
  - is similar to transport delay, except that if the input changes cause the output to change twice in an interval less than the rejection time, then the first of the two output changes does not occur.
  - **Rejection time:** is a specified value no larger than the propagation delay and is often equal to the propagation delay.

- E.g.: (next page)
Examples of behavior of transport and inertial delays:

- Propagation delay, $= 2$ ns
- Rejection time, $= 1$ ns
Fan-Out

- **Fan-out**: # of standard loads driven by a gate output
- **Max fan-out**: the fan-out that the output can drive w/o impairing gate performance
- For CMOS gates:
  - The load on the output of a gate determines the time required for the output of the gate to change from L to H and from H to L. → *transition time*
Example 3-1: Calculation of gate delay based on fan-out

A 4-input NAND gate output is attached to the inputs of the following gates with the given # of standard loads representing their inputs:

- 4-input NOR gate – 0.80 standard load
- 3-input NAND gate – 1.00 standard load
- Inverter – 1.00 standard load

The formula for the delay of the 4-input NAND gate is

$$t_{pd} = 0.07 + 0.021 \times SL \, ns$$

$SL$: the sum of the standard loads driven by the gate

<Ans.> Ignoring the wiring delay

$$t_{pd} = 0.07 + 0.021 \times (0.80 + 1.00 + 1.00) = 0.129 \, ns$$
- Fan-in & Fan-out:
  - must be dealt w/ in the technology mapping step of the design process. (§3-4)
  - Gate w/ fan-ins larger than available ⇒ Multiple gates
  - Gate w/ fan-outs either exceed its max allowable fan-out or have too high a delay ⇒ Multiple gates or added buffers at its output
E. Positive and Negative Logic

- Logic-value assignment: polarity assignment
  - Positive logic
  - Negative logic

<table>
<thead>
<tr>
<th>Signal value</th>
<th>Logic value</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>1</td>
</tr>
<tr>
<td>L</td>
<td>0</td>
</tr>
</tbody>
</table>
(a) Positive logic

<table>
<thead>
<tr>
<th>Signal value</th>
<th>Logic value</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>0</td>
</tr>
<tr>
<td>L</td>
<td>1</td>
</tr>
</tbody>
</table>
(b) Negative logic
Demonstration of positive & negative logic:

(a) Truth table with H and L

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>L</td>
<td>H</td>
<td>L</td>
</tr>
<tr>
<td>H</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td>H</td>
</tr>
</tbody>
</table>

(b) Gate block diagram

(c) Truth table for positive logic

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

(d) Positive-logic AND gate

(e) Truth table for negative logic

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

(f) Negative-logic OR gate
Conversion b/t positive and negative logic:

- Change 1’s to 0’s and 0’s to 1’s in both inputs and output of a gate (truth table)

⇒ take the dual of a function

(AND → OR, OR → AND)
F. Design Trade-Offs

- **Cost/performance trade-off:** area/delay
  - the most common of the trade-offs
  - E.g.:
    | Ckt                           | Cost | Propagation delay |
    |-------------------------------|------|-------------------|
    | a gate G (w/ fan-out = 16 standard loads) | 2.0  | 0.406 ns          |
    | gate G + a buffer             | 3.0  | 0.323 ns          |

- **Possible ckt constraints:**
  - Max input-to-output delay
  - Max area units
  - Max power dissipation
  - Max standard loads presented to ckt inputs
  - Min standard load and drive provided by the ckt outputs
  * not all are specified for a given ckt
Analysis Procedure

- Analysis procedure of a combinational ckt:
  Logic diagram → Output Boolean functions, a truth table, or a verbal explanation of the ckt operation

* Make sure that the given ckt is combinational. (not esq.)
Logic diagram → Output Boolean functions

Procedure:

1. Label all gate outputs that are a function of input variables w/ arbitrary symbols.
   Determine the Boolean function for each gate output.
2. Label the gates that are a function of input variables and previously labeled gates w/ other arbitrary symbols.
   Find the Boolean functions for these gates.
3. Repeat step 2 until the outputs of the ckt are obtained in terms of input variables.
Example

Analyze the following logic diagram:

3 inputs: A, B, C; 2 outputs: F₁, F₂

F₁ = T₂ + T₃
= A'B'C + A'BC' + AB'C + ABC

T₃ = T₁F₂'
= (A+B+C)(A'B' + A'C' + B'C')
= AB'C' + A'BC' + A'B'C

F₂' = (AB + AC + BC)'
= A'B' + A'C' + B'C'

F₂ = AB + AC + BC

J.J. Shann 3-34
Logic diagram $\rightarrow$ Truth table

- **Procedure:**
  1. Logic diagram $\rightarrow$ Output Boolean functions $\rightarrow$ Truth table
  2. Logic diagram $\rightarrow$ Truth table
    1. Determine the # of input variables in the ckt.
       For $n$ inputs, form the $2^n$ possible input combinations and list the binary numbers from 0 to $2^n - 1$ in a table.
    2. Label the outputs of selected gates w/ arbitrary symbols.
    3. Obtain the truth table for the outputs of those gates that are a function of the input variables only.
    4. Proceed to obtain the truth table for the outputs of those gates that are a function of previously defined values until the columns for all outputs are determined.

J.J. Shann 3-35
Example

\[ F_2 = AB + AC + BC \]

\[ T_3 = T_1 F_2' \]

\[ F_1 = T_2 + T_3 \]

Truth Table for the Logic Diagram of Fig. 4.2

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>( F_2 )</th>
<th>( F_2' )</th>
<th>( T_1 )</th>
<th>( T_2 )</th>
<th>( T_3 )</th>
<th>( F_1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
3-3 Design Procedure

Design procedure of a combinational ckt:

1. Specification:
   - Write a specification for the ckt.

2. Formulation:
   - Derive the truth table or initial Boolean eqs that define the required relationships b/t inputs and outputs.

3. Optimization:
   - Apply two-level and multiple-level optimization.
   - Draw a logic diagram or provide a netlist for the resulting ckt using ANDs, ORs, and inverters.

4. Technology Mapping: (§3-4)
   - Transform the logic diagram or netlist to a new diagram or netlist using the available implementation technology.

5. Verification: (§3-5)
   - Verify the correctness of the final design.
Example 3-2

E.g.: Design of a BCD-to-excess-3 code converter

**Specification:**
- Excess-3 code for a decimal digit: is the binary combination corresponding to the decimal digit plus 3.
- Inputs: BCD code; A, B, C, D
- Outputs: Excess-3 code; W, X, Y, Z

**Formulation:**

<table>
<thead>
<tr>
<th>Decimal Digit</th>
<th>Input BCD</th>
<th>Output Excess-3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A B C D</td>
<td>W X Y Z</td>
</tr>
<tr>
<td>0</td>
<td>0 0 0 0</td>
<td>0 0 1 1</td>
</tr>
<tr>
<td>1</td>
<td>0 0 0 1</td>
<td>0 1 0 0</td>
</tr>
<tr>
<td>2</td>
<td>0 0 1 0</td>
<td>0 1 0 1</td>
</tr>
<tr>
<td>3</td>
<td>0 0 1 1</td>
<td>0 1 1 0</td>
</tr>
<tr>
<td>4</td>
<td>0 1 0 0</td>
<td>0 1 1 1</td>
</tr>
<tr>
<td>5</td>
<td>0 1 0 1</td>
<td>1 0 0 0</td>
</tr>
<tr>
<td>6</td>
<td>0 1 1 0</td>
<td>1 0 0 1</td>
</tr>
<tr>
<td>7</td>
<td>0 1 1 1</td>
<td>1 0 1 0</td>
</tr>
<tr>
<td>8</td>
<td>1 0 0 0</td>
<td>1 0 1 1</td>
</tr>
<tr>
<td>9</td>
<td>1 0 0 1</td>
<td>1 1 0 0</td>
</tr>
</tbody>
</table>

1010 ~ 1111: don’t-care conditions
Optimization:
Initial optimization
(Two-level)
GIC = 26

<table>
<thead>
<tr>
<th>Input BCD</th>
<th>Output Excess-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C D</td>
<td>W X Y Z</td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>0 0 1 1</td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>0 1 0 0</td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>0 1 0 1</td>
</tr>
<tr>
<td>0 0 1 1</td>
<td>0 1 1 0</td>
</tr>
<tr>
<td>0 1 0 0</td>
<td>0 1 1 1</td>
</tr>
<tr>
<td>0 1 0 1</td>
<td>1 0 0 0</td>
</tr>
<tr>
<td>0 1 1 0</td>
<td>1 0 0 1</td>
</tr>
<tr>
<td>0 1 1 1</td>
<td>1 0 1 0</td>
</tr>
<tr>
<td>1 0 0 0</td>
<td>1 0 1 1</td>
</tr>
<tr>
<td>1 0 0 1</td>
<td>1 1 0 0</td>
</tr>
</tbody>
</table>

W = A + BC + BD

X = C\overline{BC} + \overline{BD} + B\overline{CD}

Y = CD + \overline{CD}

Z = \overline{D}
Optimization: (cont’d)

2nd optimization
(Multiple-level)

GIC = 22

\[ T_1 = C + D \]
\[ W = A + BC + BD = A + BT_1 \]
\[ X = \overline{BC} + \overline{BD} + \overline{BCD} = BT_1 + \overline{BCD} \]
\[ Y = CD + C\overline{D} \]
\[ Z = \overline{D} \]
Example 3-3

E.g.: Design of a BCD-to-seven-segment decoder

**Specification:**
- BCD-to-seven-segment decoder: a combinational ckt that accepts a decimal digit in BCD and generates the appropriate outputs for the segments of the display for that decimal digit.
- Inputs: BCD code; A, B, C, D
- Outputs: 7 segments; a, b, c, d, e, f, g

![Diagram of 7-segment display](image)

(a) Segment designation  (b) Numeric designation for display
**Formulation:**

truth table

<table>
<thead>
<tr>
<th>BCD Input</th>
<th>Seven-Segment Decoder</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C D</td>
<td>a b c d e f g</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>1 1 1 1 1 1 0</td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>0 1 1 0 0 0 0</td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>1 1 0 1 1 0 1</td>
</tr>
<tr>
<td>0 0 1 1</td>
<td>1 1 1 1 0 0 1</td>
</tr>
<tr>
<td>0 1 0 0</td>
<td>0 1 1 0 0 1 1</td>
</tr>
<tr>
<td>0 1 0 1</td>
<td>1 0 1 1 0 1 1</td>
</tr>
<tr>
<td>0 1 1 0</td>
<td>1 0 1 1 1 1 1</td>
</tr>
<tr>
<td>0 1 1 1</td>
<td>1 1 1 0 0 0 0</td>
</tr>
<tr>
<td>1 0 0 0</td>
<td>1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>1 0 0 1</td>
<td>1 1 1 1 0 1 1</td>
</tr>
<tr>
<td>All other inputs</td>
<td>0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>
Optimization:

Two-level optimization

\[ a = \overline{AC} + \overline{ABD} + \overline{BCD} + \overline{ABC} \]
\[ b = \overline{AB} + \overline{ACD} + \overline{ACD} + \overline{ABC} \]
\[ c = \overline{AB} + \overline{AD} + \overline{BCD} + \overline{ABC} \]
\[ d = \overline{ACD} + \overline{ABC} + \overline{BCD} + \overline{ABC} + \overline{ABCD} \]
\[ e = \overline{ACD} + \overline{BCD} \]
\[ f = \overline{ABC} + \overline{ACD} + \overline{ABD} + \overline{ABC} \]
\[ g = \overline{ACD} + \overline{ABC} + \overline{ABC} + \overline{ABC} \]
Example 3-4

E.g.: Design of a 4-bit equality comparator

**Specification:**
- Inputs: 2 vectors A(3:0) & B(3:0)  
  Vector A consists of 4 bits, A(3), A(2), A(1), A(0)  
  Vector B consists of 4 bits, B(3), B(2), B(1), B(0)  
- Output: E  
  Output E = 1 if A = B, E = 0 if A ≠ B

**Formulation:**
- Truth table: 8 inputs → impractical  
- E = 1 (i.e., A = B) if the bit values in each of the respective positions, 3 down to 0, of A and B are equal; otherwise, E = 0.
Optimization:

Develop a multiple level ckt using hierarchy by intuition:

For bit position $i$, ckt output $E_i = 0$ if $A_i = B_i$ and $E_i = 1$ if $A_i \neq B_i$

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

Output $E = 1$ only if all of the $E_i$ values are 0:

$$E = E_0 + E_1 + E_2 + E_3$$

* Alternative solution:

$$E_i = \overline{A_i}\overline{B_i} + A_iB_i \quad (E_i = 1 \text{ if } A_i = B_i)$$

$$E = E_0 \cdot E_1 \cdot E_2 \cdot E_3$$
\[ E_i = \overline{A_i}B_i + A_i\overline{B_i} \quad \text{and} \quad E = E_0 + E_1 + E_2 + E_3 \]
3 primary ways of designing VLSI ckt:

- Full custom design: expensive
  - An entire design of the chip, down to the smallest detail of the layout, is performed.
  - is very expensive \( \Rightarrow \) only for dense, fast ICs w/ high sales volume

- Standard cell design: (§3-4)
  - Large parts of the design have been performed ahead of time or possibly, used in previous designs.
  - intermediate cost; less density and speed compared to full custom

- Gate array: (§3-6)
  - uses a regular pattern of gates fabricated in silicon
  - only the interconnections b/t gates are specify to a design
  - can be used for numerous different designs.
  - lowest cost; less density compared to the above two
Cell Library

- **Cell**: a pre-designed primitive block
  - Cells are used for gate array, standard cell, and in some cases, full custom chip design.

- **Cell library**:
  - the collection of cells available for a given 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
A. Cell Specification

Cell specification:
1. Schematic or logic diagram for the function of the cell
2. Area of cell: often normalized to the area of a common, small cell (e.g., an inverter)
3. Input loading (in standard loads) that each input of a cell presents to the output driving it
4. Delays from each input to each output
5. One or more cell templates for technology mapping
6. One or more hardware description language models
   (If automatic layout is to be used: )
7. Physical layout of the cell ckt
8. A floorplan layout providing the location of inputs, outputs, power and ground connections on the cell
B. Libraries

- Library:
  - a collection of cell specification

- Technology mapping:
  - converts a ckt that initially consists of AND, OR and NOT gates to one that used only cells from the applicable libraries.

  Ckt w/ AND, OR, NOT gates → Library cells
Example Cell Library

**E.g.:** p.3-51~3-52

<table>
<thead>
<tr>
<th>Cell Name</th>
<th>Cell Schematic</th>
<th>Normalized Area</th>
<th>Typical Input Load</th>
<th>Typical Input-to-Output Delay</th>
<th>Basic Function Templates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inverter</td>
<td>![Inverter Schematic]</td>
<td>1.00</td>
<td>1.00</td>
<td>0.04 + 0.012 × SL</td>
<td>![Inverter Template]</td>
</tr>
<tr>
<td>2NAND</td>
<td>![2NAND Schematic]</td>
<td>1.25</td>
<td>1.00</td>
<td>0.05 + 0.014 × SL</td>
<td>![2NAND Template]</td>
</tr>
<tr>
<td>3NAND</td>
<td>![3NAND Schematic]</td>
<td>1.50</td>
<td>1.00</td>
<td>0.06 + 0.017 × SL</td>
<td>![3NAND Template]</td>
</tr>
<tr>
<td>4NAND</td>
<td>![4NAND Schematic]</td>
<td>2.00</td>
<td>0.95</td>
<td>0.07 + 0.021 × SL</td>
<td>![4NAND Template]</td>
</tr>
</tbody>
</table>

*SL:* the sum of all of the standard loads presented by the inputs of cells driven by the cell output
E.g.: (cont’d)

<table>
<thead>
<tr>
<th>Cell Name</th>
<th>Cell Schematic</th>
<th>Normalized Area</th>
<th>Typical Input Load</th>
<th>Typical Input-to-Output Delay</th>
<th>Basic Function Templates</th>
</tr>
</thead>
<tbody>
<tr>
<td>2NOR</td>
<td><img src="image1" alt="2NOR Schematic" /></td>
<td>1.25</td>
<td>1.00</td>
<td>0.06 + 0.018 × SL</td>
<td><img src="image2" alt="2NOR Templates" /></td>
</tr>
<tr>
<td>3NOR</td>
<td><img src="image3" alt="3NOR Schematic" /></td>
<td>2.00</td>
<td>0.95</td>
<td>0.15 + 0.012 × SL</td>
<td><img src="image4" alt="3NOR Templates" /></td>
</tr>
<tr>
<td>4NOR</td>
<td><img src="image5" alt="4NOR Schematic" /></td>
<td>3.25</td>
<td>0.80</td>
<td>0.17 + 0.012 × SL</td>
<td><img src="image6" alt="4NOR Templates" /></td>
</tr>
<tr>
<td>2-2 AOI</td>
<td><img src="image7" alt="2-2 AOI Schematic" /></td>
<td>2.25</td>
<td>0.95</td>
<td>0.07 + 0.019 × SL</td>
<td><img src="image8" alt="2-2 AOI Templates" /></td>
</tr>
</tbody>
</table>
Example 3-5: Calculation of Cell Delay

E.g.:

Calculate the delay of a 2NAND driving the following cells: an inverter, a 4NAND, and a 4NOR.

<Ans.>

The sum of the standard loads:

$$SL = 1.00 + 0.95 + 0.80$$

$$= 2.75$$

The delay of the 2NAND:

$$t_p = 0.05 + 0.014 \times 2.75$$

$$= 0.089 \text{ ns}$$
C. Mapping Techniques

- Mapping techniques:
  - Mapping to NAND gates
  - Mapping to NOR gates
  - Mapping Multiple gate types
(a) Mapping to \textit{NAND} Gates

- Implementing a Boolean function w/ NAND gates:
  
  i. Obtain the optimized Boolean function in terms of the Boolean operators AND, OR, and NOT and then
  
  ii. Map the function to NAND logic.

- Conversion of an algebraic expression from AND, OR, and NOT to NAND:
  
  - Procedure: p.3-56\textendash3-57
Mapping to *NAND* Gates

**Procedure:**

1. Replace each AND and OR gate with the NAND gate and inverter equivalent circuits:

2. Cancel all inverter pairs.
3. W/o changing the logic function
   a) push all inverters lying b/t (i) either a ckt input or a driving NAND gate output and (ii) the driven NAND gate inputs toward the driven NAND gate inputs.

   ❚ Cancel pairs of inverters in series whenever possible.

   b) Replace inverters in parallel w/ a single inverter that drives all of the outputs of the parallel inverters. (Inverse of (a))

   c) Repeat a) and b) until there is at most one inverter b/t the ckt input or driving NAND gate output and the attached NAND gate inputs.
Example 3-6

Example 3-6: Implementation w/ NAND Gates

Implement the following optimized function w/ NAND gates:

\[ F = AB + (\overline{AB})C + (\overline{AB})\overline{D} + E \]

<Ans.>

1. Replace each AND and OR gate w/ the NAND gate and inverter equivalent ckts:
2. Cancel all inverter pairs.
3. a) Push Inverter 5 through dot X:
Special case: two-level ckts

SoP forms (AND-OR ckts) $\rightarrow$ NAND logic diagrams
(b) Mapping to *NOR* Gates

- Implementing a Boolean function w/ NOR gates:
  1. Obtain the optimized Boolean function in terms of the Boolean operators AND, OR, and NOT and then
  2. Map the function to NOR logic.

- Conversion of an algebraic expression from AND, OR, and NOT to NOR:
  
  - Procedure: p.3-64~3-65
Mapping to *NOR* Gates

**Procedure:**

1. Replace each AND and OR gate w/ the NOR gate and inverter equivalent ckts:

![Diagram of AND and OR gates being replaced with NOR gates and inverters.]

2. Cancel all inverter pairs.
3. Similar to Step 3 for NAND.
Example 3-7: Implementation w/ NOR Gates

Implement the same optimized function w/ NOR gates:

\[ F = AB + (\overline{AB})C + (\overline{AB})\overline{D} + E \]

<Ans.>

1. Replace each AND and OR gate w/ the NOR gate and inverter equivalent ckts:
2. Cancel all inverter pairs: the pair of inverters on the D input line
3. Push inverter 1 through dot X to cancel w/ inverters 2 & 3:
Special case: two-level ckts
PoS forms (OR-AND ckts) → NOR logic diagrams
(c) Mapping Multiple Gate Types

Procedure:

1. Replace each AND and OR gate w/ an optimum equivalent ckt consisting only of 2-input NAND gates and inverters.
   - breaks the ckt up into small pieces ⇒ provide the max flexibility in mapping cells to achieve an optimized result

2. In each line in the ckt attached to a ckt input, a NAND gate input, a NAND gate output, or a ckt output in which no inverter appears, insert a serial pair of inverters.

3. Perform a replacement of connections of NAND gates and inverters by the available library cells s.t. the gate input cost which results within *fan-out free subcircuits* is optimized.
   - *fan-out free subckt*: a ckt in which each gate output drives a single gate input
4. W/o changing the logic function
   a) push all inverters lying b/t (i) either a ckt input or a driving gate output and (ii) the driven gate inputs, toward the driven gate inputs.
      ➢ Cancel pairs of inverters in series whenever possible.
   b) Replace inverters in parallel w/ a single inverter that drives all of the outputs of the parallel inverters.
   c) Repeat a) and b) until there is at most one inverter b/t the ckt input or driving gate output and the attached driven gate inputs.
Example 3-8

- **Example 3-8: Implementation w/ a small cell library**

Implement the same optimized function

\[ F = AB + (AB) C + (AB)D + E \]

w/ a cell library containing a 2-input NAND gate, 3-input NAND gate, a 2-input NOR gate, and an inverter.

<table>
<thead>
<tr>
<th>Cell Name</th>
<th>Cell Schematic</th>
<th>Normalized Area</th>
<th>Typical Input Load</th>
<th>Typical Input-to-Output Delay</th>
<th>Basic Function Templates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inverter</td>
<td><img src="image" alt="Inverter Schematic" /></td>
<td>1.00</td>
<td>1.00</td>
<td>0.04 + 0.012 × SL</td>
<td><img src="image" alt="Inverter Icon" /></td>
</tr>
<tr>
<td>2NAND</td>
<td><img src="image" alt="2NAND Schematic" /></td>
<td>1.25</td>
<td>1.00</td>
<td>0.05 + 0.014 × SL</td>
<td><img src="image" alt="2NAND Icon" /></td>
</tr>
<tr>
<td>3NAND</td>
<td><img src="image" alt="3NAND Schematic" /></td>
<td>1.50</td>
<td>1.00</td>
<td>0.05 + 0.017 × SL</td>
<td><img src="image" alt="3NAND Icon" /></td>
</tr>
<tr>
<td>2NOR</td>
<td><img src="image" alt="2NOR Schematic" /></td>
<td>1.25</td>
<td>1.00</td>
<td>0.06 + 0.018 × SL</td>
<td><img src="image" alt="2NOR Icon" /></td>
</tr>
</tbody>
</table>
<Ans.>

1. Replace each AND and OR gate w/ an optimum equivalent ckt consisting only of 2-input NAND gates and inverters.

2. In each line in the ckt attached to a ckt input, a NAND gate input, a NAND gate output, or a ckt output in which no inverter appears, insert a serial pair of inverters.
3. Perform replacement of connections of NAND gates and inverters by the available library cells s.t. the gate input cost which results within \textit{fan-out free subcircuits} is optimized.
4. Push inverter 1 through dot X to cancel with inverters 2 & 3:
Comparison

- Example 3-6: Mapping to NAND gates
  - GIC = 12 (use a 4-input NAND gate)
  - GIC = 14 w/o this cell

- Example 3-7: Mapping to NOR gates
  - GIC = 14

- Example 3-8: Mapping to multiple types of gates
  - GIC = 12

* The use of more diverse cell library has provided a cost benefit.
Example 3-9: Technology mapping for BCD-to-excess-3 code converter

Cell library: Table 3-3

<Ans.>
3-5 Verification

- Verification: (Analysis)
  Circuit → Boolean equations for ckt outputs or
  Truth table for the ckt
  - determination of whether or not a given ckt implements its specified function

- Ways for verification:
  - Manual Logic analysis
  - Computer simulation-based logic analysis
A. Manual Logic Analysis

- Manual logic analysis:
  - finding Boolean equations for the ckt outputs or truth table for the ckt
  - Method: Break up the ckt into subckts by defining intermediate variables at selected points in the ckt.
    - *fan-out point*: point at which a gate output drives $\geq 2$ gate inputs
Example 3-10: Manual verification of BCD-to-excess-3 code converter

<table>
<thead>
<tr>
<th>Input BCD</th>
<th>Output Excess-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C D</td>
<td>W X Y Z</td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>0 0 1 1</td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>0 1 0 0</td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>0 1 0 1</td>
</tr>
<tr>
<td>0 0 1 1</td>
<td>0 1 1 0</td>
</tr>
<tr>
<td>0 1 0 0</td>
<td>0 1 1 1</td>
</tr>
<tr>
<td>0 1 0 1</td>
<td>1 0 0 0</td>
</tr>
<tr>
<td>0 1 1 0</td>
<td>1 0 0 1</td>
</tr>
<tr>
<td>0 1 1 1</td>
<td>1 0 1 0</td>
</tr>
<tr>
<td>1 0 0 0</td>
<td>1 0 1 1</td>
</tr>
<tr>
<td>1 0 0 1</td>
<td>1 1 0 0</td>
</tr>
</tbody>
</table>

Original truth table
\[ T1 = C + D = C + D \]
\[ W = A \cdot (T1 \cdot B) = A + B \cdot T1 \]
\[ \Rightarrow X = (B \cdot \overline{T1}) \cdot (B \cdot \overline{C} \cdot \overline{D}) = \overline{B} \cdot T1 + B \overline{C} \cdot \overline{D} \]
\[ Y = CD + \overline{CD} = CD + \overline{CD} \]
\[ Z = \overline{D} \]
\[ T1 = C + D \]
\[ W = A + B \cdot T1 \]
\[ X = \overline{B} \cdot T1 + B\overline{C} \cdot \overline{D} \]
\[ Y = CD + \overline{CD} \]
\[ Z = \overline{D} \]

\[ \downarrow \text{substitution} \]

\[ W = A + BC + BD \]
\[ X = \overline{BC} + \overline{BD} + B\overline{CD} \]
\[ Y = CD + \overline{CD} \]
\[ Z = \overline{D} \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>W</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
### Original truth table

<table>
<thead>
<tr>
<th>Input BCD</th>
<th>Output Excess-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C D</td>
<td>W X Y Z</td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>0 0 1 1</td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>0 1 0 0</td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>0 1 0 1</td>
</tr>
<tr>
<td>0 0 1 1</td>
<td>0 1 1 0</td>
</tr>
<tr>
<td>0 1 0 0</td>
<td>0 1 1 1</td>
</tr>
<tr>
<td>0 1 0 1</td>
<td>1 0 0 0</td>
</tr>
<tr>
<td>0 1 1 0</td>
<td>1 0 0 1</td>
</tr>
<tr>
<td>0 1 1 1</td>
<td>1 0 1 0</td>
</tr>
<tr>
<td>1 0 0 0</td>
<td>1 0 1 1</td>
</tr>
<tr>
<td>1 0 0 1</td>
<td>1 1 0 0</td>
</tr>
</tbody>
</table>

### Derived truth table

<table>
<thead>
<tr>
<th>Input BCD</th>
<th>Output Excess-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C D</td>
<td>W X Y Z</td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>1 1</td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>1</td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>1 1</td>
</tr>
<tr>
<td>0 0 1 1</td>
<td>1</td>
</tr>
<tr>
<td>0 1 0 0</td>
<td>1 1 1</td>
</tr>
<tr>
<td>0 1 0 1</td>
<td>1</td>
</tr>
<tr>
<td>0 1 1 0</td>
<td>1</td>
</tr>
<tr>
<td>0 1 1 1</td>
<td>1 1</td>
</tr>
<tr>
<td>1 0 0 0</td>
<td>1 1 1</td>
</tr>
<tr>
<td>1 0 0 1</td>
<td>1</td>
</tr>
<tr>
<td>1 0 1 0</td>
<td>1</td>
</tr>
</tbody>
</table>

ann 3-81
B. Simulation

- Computer simulation:
  - permits truth table verification to be done for a significantly larger # of variables and greatly reduces the tedious analysis effort required.
  - If possible, it is desirable for thorough verification to apply all possible input combinations.
Example 3-11: Simulation-based verification of BCD-to-excess-3 code converter

- Xilinx ISE4.2i FPGA development tools
- XE II Modelsim simulator

<table>
<thead>
<tr>
<th>Input BCD</th>
<th>Output Excess-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C D</td>
<td>W X Y Z</td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>0 0 1 1</td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>0 1 0 0</td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>0 1 0 1</td>
</tr>
<tr>
<td>0 0 1 1</td>
<td>0 1 1 0</td>
</tr>
<tr>
<td>0 1 0 0</td>
<td>0 1 1 1</td>
</tr>
<tr>
<td>0 1 0 1</td>
<td>1 0 0 0</td>
</tr>
<tr>
<td>0 1 1 0</td>
<td>1 0 0 1</td>
</tr>
<tr>
<td>0 1 1 1</td>
<td>1 0 1 0</td>
</tr>
<tr>
<td>1 0 0 0</td>
<td>1 0 1 1</td>
</tr>
<tr>
<td>1 0 0 1</td>
<td>1 1 0 0</td>
</tr>
</tbody>
</table>
<Ans.>

Simulation results
3-6 Programmable Implementation Technologies

- Programmable logic device (PLD):
  - is an IC w/ programmable gates divided into an AND array and an OR array to provide an AND-OR sum of product implementation.

- 3 major types of combinational PLDs:
  - Differ in the placement of the programmable connections in the AND-OR array.
  1. ROM: read-only memory
  2. PAL: programmable array logic
  3. PLA: programmable logic array
(a) Programmable read-only memory (PROM)

(b) Programmable array logic (PAL) device

(c) Programmable logic array (PLA) device
Programming technologies of PLDs:

1. establish or break interconnections
2. build lookup tables
3. control transistor switching
A. Read-Only Memory (ROM)

- ROM:
  - a memory device in which permanent binary information is stored

**ROM Block diagram:**

k inputs (address) → $2^k \times n$ ROM → n outputs (data)
Internal logic of a $2^k \times n$ ROM: comb. ckt.

- have an internal $k \times 2^k$ decoder & $n$ OR gates
- E.g.: a $32 \times 8$ ROM

![Diagram of a 5-to-32 decoder and OR gates](image)
Types of ROMs

- **Types of ROM:**
  - Mask programming
  - PROM: programmable ROM
  - EPROM: erasable PROM
  - EEPROM: electrically-erasable PROM
B. Programmable Logic Array (PLA)

PLA:

- The array of AND gates can be programmed to generate any product terms of the input variables.
- The product terms are then connected to OR gates to provide the sum of products for the required Boolean functions.
**Internal logic of a PLA:**

- E.g.: a $3 \times 4 \times 2$ PLA

3 inputs
4 product terms
2 outputs

$$F_1 = \overline{A}B + AC + \overline{A}BC$$

$$F_2 = AC + BC$$

$T_2 + T_3$
C. Programmable Array Logic (PAL)

- **PAL:**
  - Only the AND gates are programmable.
  - The PAL is easier to program, but is not as flexible as the PLA.
Internal logic of a PAL:
- E.g.:
  4 inputs
  4 outputs (4 sections)
  3-wide AND-OR structure

\[ F_1 = \overline{AB} + AC + \overline{ABC} \]
\[ F_2 = AC + BC = \overline{AB} + C \]
3-7 Chapter Summary

- Design hierarchy & Top-down design
- Gate technology
- 5-step design procedure
  - Specification
  - Formulation
  - Optimization
  - Technology mapping:
    - map to NAND, NOR, or multiple types of gates
  - Verification
- Programmable logic technologies
### Problems

<table>
<thead>
<tr>
<th>Sections</th>
<th>Exercises</th>
</tr>
</thead>
<tbody>
<tr>
<td>§3-1</td>
<td>3-1 ~ 3-2</td>
</tr>
<tr>
<td>§3-2</td>
<td>3-2 ~ 3-9</td>
</tr>
<tr>
<td>§3-3</td>
<td>3-10 ~ 3-19</td>
</tr>
<tr>
<td>§3-4</td>
<td>3-20 ~ 3-23</td>
</tr>
<tr>
<td>§3-5</td>
<td>3-24 ~ 3-28</td>
</tr>
<tr>
<td>§3-6</td>
<td></td>
</tr>
</tbody>
</table>
Homework

- 3-1 --> chapter 3-1 design hierarchy 觀念的應用
- 3-5 --> chapter 3-2 考慮不同 delay 狀況下的 waveform 輸出
- 3-13 --> chapter 3-3 電路設計。給定一個問題，設計出對應這個問題的 logic circuit
- 3-14 --> chapter 3-3 電路設計。給定一個問題，設計出對應這個問題的 logic circuit（難度高一點）
- 3-22 --> chapter 3-4 circuit mapping
- 3-23 --> chapter 3-5 using manual methods, verify a circuit.

J.J. Shann 3-97