Chap 3. Combinational Logic Design

Overview

Part 1 - 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?

Part 2 - Programmable Implementation Technologies
- Read-Only Memories, Programmable Logic Arrays, Programmable Array Logic
- Technology mapping to programmable logic devices

3.1 Combinational Circuits

- Logic circuits for digital systems: combinational vs sequential
- Combinational Circuit (Chap 3, 4, 5)
  - Outputs are determined by the present applied inputs
  - Performs an operation, which can be specified logically by a set of Boolean expressions
- Sequential Circuit (Chap 6)
  - Logic gates + storage elements (called flip-flops)
  - Outputs are a function of the inputs & bit values in the storage elements
  - Output depends on the present values of inputs & past inputs

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

Hierarchy for Parity Tree Example
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

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.

Logic Synthesis

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

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

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 (tPHL) and low-to-high (tPLH) 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.

Propagation Delay (continued)

- Propagation delays measured at the midpoint between the L and H values
- What is the expression for the tPHL 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

Delay Models

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

Delay Model Example

- **Propagation Delay** = 2.0 ns **Rejection Time** = 1.0 ns
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

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:
      \[ t_{pd} = 0.07 + 0.021 \times SL \text{ ns} \]
    - SL is the number of standard loads the gate is driving, i.e., its fan-out in standard loads
    - Case 1:
      - For \( SL = 4.5 \), \( t_{pd} = 0.165 \text{ ns} \)
    - Case 2:
      - For 4-input NOR: \( 0.8 \times SL \)
      - 3-input NAND: \( 1 \times SL \)
      - inverter: \( 1 \times SL \)
      \[ t_{pd} = 0.07 + 0.021(0.8 \times 4.5 + 1 + 1) = 0.129 \text{ ns} \]

Cost

- In an integrated circuit:
  - The cost of a primitive 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

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:

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

What logic function is implemented?

<table>
<thead>
<tr>
<th>Positive Logic (( H = 1 )) (( L = 0 ))</th>
<th>Negative Logic (( H = 0 )) (( L = 1 ))</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>0 1</td>
</tr>
<tr>
<td>0 1</td>
<td>1 0</td>
</tr>
<tr>
<td>1 0</td>
<td>0 0</td>
</tr>
<tr>
<td>1 1</td>
<td>1 1</td>
</tr>
</tbody>
</table>

Positive and Negative Logic (continued)

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

<table>
<thead>
<tr>
<th>Positive Logic (( H = 1 )) (( L = 0 ))</th>
<th>Negative Logic (( H = 0 )) (( L = 1 ))</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>0 1</td>
</tr>
<tr>
<td>0 1</td>
<td>1 0</td>
</tr>
<tr>
<td>1 0</td>
<td>0 0</td>
</tr>
<tr>
<td>1 1</td>
<td>1 1</td>
</tr>
</tbody>
</table>
Logic Symbol Conventions

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

<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>H</td>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td>H</td>
</tr>
</tbody>
</table>

Logic Circuit

Positive Logic Negative Logic

Design Trade-Offs

- Cost - performance tradeoffs
- Gate-Level Examples:
  - 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 of 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

3.2 Design Topics

- Functional Blocks
  - predefined, reusable blocks
  - that typically lie in the lower levels of logic design hierarchies
  - provide functions that are broadly useful in digital design
  - available for decades in MSI circuits
- Top-Down Design
  - functional blocks: Ch 5, Ch 6
  - (cf) bottom up
- Computer-Aided Design
  - Schematic Capture, Library
  - HDL (hardware description language)
  - Verification
    - logic simulator
    - Logic Synthesizer
    - Physical area, delay

HDL (hardware description languages)

- VHSIC (VHSI/Very High Speed Integrated Circuits) Hardware Description Language
- Verilog: Cadence Design Systems, Inc.

Feature

- Programming Language
- Parallel operation
  - contrary to conventional serial operation
- Structural description
  - Alternative to schematic; low-level description
- Behavioral description
  - high-level description
- Logic Synthesis: RTL (Register transfer level)

3-3 Design Procedure

- Determining the function that the circuit implements
  1) make sure that the circuit is combinational, not sequential
  - no feedback or storage elements
  2) obtain the output Boolean functions or the truth table
  - interpret the operation of the circuit
- Derivation of Boolean Functions
  - to obtain the output Boolean functions from a logic diagram
    1) label all gate outputs that are a function only of input variables or their complements & determine the Boolean functions for each gate
    2) label the gates that are a function of input variables and previously labeled gates & find Boolean functions for each gate
    3) repeat (2) until the outputs of the circuits are obtained
### 3.3 Analysis Procedure

**Ex)**

4 inputs, A, B, C, D; 2 outputs, F₁, F₂

- \( T₁ = B' C \)
- \( T₂ = A' B \)
- \( T₃ = A + T₁ = A + B' C \)
- \( T₄ = T₂ \oplus D = (A' B) \oplus D = A'B'D' + AD + B'D \)

So,

- \( F₁ = T₃ \oplus T₄ = A'B + D \)
- \( F₂ = T₂ + D = A'B + D \)

**So,**

- \( F₁ = T₃ + T₄ = A + B'C + A'BD' + AD + B'D \)
- \( F₂ = T₂ + D = A'B + D \)

### 3.3 Analysis Procedure

Derivation of the Truth Table from the Logic Diagram

1. Determine the number of input variables in the circuit.
2. List all possible binary numbers for the inputs (2^n).
3. Label the outputs of selected gates.
4. Obtain the truth table for the outputs of those gates.

### 3.3 Analysis Procedure

**Ex)**

3 inputs, X, Y, Z; 2 outputs C, S (00 ~ 11)

- C is 1 if \( XY, XZ, \) or \( YZ = 11 \);
- C is 0 otherwise

### 3.4 Design Procedure

- **Procedure to design combinational circuits**
  1. Determine the required number of inputs and outputs, and assign a letter symbol to each.
  2. Derive the truth table.
  3. Obtain simplified Boolean functions for each output.
  4. Draw the logic diagram.
  5. Verify the correctness of the design.

- **Truth Table**
  - n input variables: \( 2^n \) binary numbers.
  - Output functions give the exact definition of combinational circuits.
  - Output functions are simplified by methods such as algebraic manipulation, mapping, and computer programs.

### 3.4 Design Procedure

**Ex 3.1)**

Design a combinational circuit with 3 inputs and 1 output.

Output is 1 when input is less than 3(011), otherwise 0.
3.4 Design Procedure

- Code Converter
  - a circuit that translates info from one binary code to another
  - Ex 3.2) BCD to Excess-3 Code Converter
    - Excess-3 code: decimal digit + 3
    - desirable to implementing decimal subtraction

<table>
<thead>
<tr>
<th>Decimal Digit</th>
<th>Input (BCD)</th>
<th>Output (Excess-3)</th>
</tr>
</thead>
<tbody>
<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 0</td>
<td>0 1 0 0</td>
</tr>
<tr>
<td>2</td>
<td>0 0 0 0</td>
<td>0 1 1 0</td>
</tr>
<tr>
<td>3</td>
<td>0 0 0 1</td>
<td>0 1 1 1</td>
</tr>
<tr>
<td>4</td>
<td>0 0 1 0</td>
<td>0 1 0 1</td>
</tr>
<tr>
<td>5</td>
<td>0 0 1 0</td>
<td>0 1 1 0</td>
</tr>
<tr>
<td>6</td>
<td>0 1 0 0</td>
<td>0 1 0 0</td>
</tr>
<tr>
<td>7</td>
<td>0 1 0 0</td>
<td>0 1 0 1</td>
</tr>
<tr>
<td>8</td>
<td>1 0 0 0</td>
<td>1 1 0 0</td>
</tr>
<tr>
<td>9</td>
<td>1 0 0 0</td>
<td>1 1 0 1</td>
</tr>
</tbody>
</table>

(Don’t care conditions!!)

- two-level AND-OR logic diagram
  - W = A + BC + BD = A + B(C + D)
  - X = B'C + B'D + BC'D' = B'(C + D) + BC'D'
  - Y = CD + C'D' = (C D)'
  - Z = D'

- Logic Diagram

Ex 3.3) BCD to Seven-Segment Decoder
  - LED (light emitting diodes), or LCD (liquid crystal display)
  - accept a decimal digit in BCD
  - & generate the appropriate outputs

<table>
<thead>
<tr>
<th>BCD Input</th>
<th>Seven-Segment Decoder</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0</td>
<td>0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>0 0 0 0 0 0 0 1 0</td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>0 0 0 0 1 0 0 0 0</td>
</tr>
<tr>
<td>0 0 1 1</td>
<td>0 0 0 0 1 0 0 1 0</td>
</tr>
<tr>
<td>0 1 0 0</td>
<td>0 0 0 1 0 0 0 0 0</td>
</tr>
<tr>
<td>0 1 0 1</td>
<td>0 0 0 1 0 0 0 1 0</td>
</tr>
<tr>
<td>0 1 1 0</td>
<td>0 0 0 1 0 0 1 0 0</td>
</tr>
<tr>
<td>0 1 1 1</td>
<td>0 0 0 1 0 0 1 1 0</td>
</tr>
<tr>
<td>1 0 0 0</td>
<td>0 0 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>1 0 0 1</td>
<td>0 0 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>1 0 1 0</td>
<td>0 0 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>1 0 1 1</td>
<td>0 0 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>1 1 0 0</td>
<td>0 0 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>1 1 0 1</td>
<td>0 0 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>1 1 1 0</td>
<td>0 0 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>1 1 1 1</td>
<td>0 0 1 1 1 1 1 1 1</td>
</tr>
</tbody>
</table>

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

2. Formulation
   • Conversion of 4-bit codes can be most easily formulated by a truth table
     | Input BCD | Output Excess-3 |
     |-----------|-----------------|
     | A B C D   | 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 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 1         |
     | 1 0 0 0   | 1 0 1 1         |
     | 1 0 0 1   | 1 0 1 1         |

3. Optimization
   a. 2-level using K-maps
      \[ W = A + BC + BD \]
      \[ X = BC + BD + BC \]
      \[ Y = CD + CB \]
      \[ Z = D \]

   b. Multiple-level using transformations
      \[ T_1 = C + D \]
      \[ W = A + BT_1 \]
      \[ X = BT_1 + BCD \]
      \[ Y = CD + CB \]
      \[ Z = D \]
      \[ G = 7 + 10 + 6 + 0 = 23 \]
      \[ G = 2 + 1 + 4 + 7 + 6 + 0 = 19 \]

   An additional extraction not shown in the text since it uses a Boolean transformation: \( C \overline{D} = C + D = T_1 \):

      \[ W = A + BT_1 \]
      \[ X = BT_1 + B T_1 \]
      \[ Y = CD + T_1 \]
      \[ Z = D \]
      \[ G = 2 + 1 + 4 + 6 + 4 + 0 = 16 \]

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

- Chip design styles
- Cells and cell libraries
- Mapping Techniques
  - NAND gates
  - NOR gates
  - Multiple gate types
  - Programmable logic devices
  - The subject of Chapter 3 - Part 2

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

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

<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></td>
<td>1.00 1.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2NAND</td>
<td></td>
<td>1.25 1.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2NOR</td>
<td></td>
<td>1.25 1.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2-2 AOI</td>
<td></td>
<td>2.25 0.95</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Mapping to NAND gates

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

1. Replace ANDs and ORs:

```
  A --\   /-- B
     \   /  
      \ /   
       X
```

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.

```
  A --\   /-- B
     \   /  
      \ /   
       X
```

NAND Mapping Example

```
(a)  
(b)  
```

Mapping to NOR gates

- Assumptions:
  - Gate loading and delay are ignored
  - Cell library contains an inverter and $n$-input NOR gates, $n = 2, 3, \ldots$
  - 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

```
  A --\   /-- B
     \   /  
      \ /   
       X
```

NOR Mapping Algorithm

1. Replace ANDs and ORs:

```
  A --\   /-- B
     \   /  
      \ /   
       X
```

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.

```
  A --\   /-- B
     \   /  
      \ /   
       X
```

NOR Mapping Example

```
(a)  
(b)  
```

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
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”

Example: Mapping Multiple Gate Types

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

Example: Mapping Multiple Gate Types

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!

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:
  - \[ T_1 = C \overline{D} + D = C + D \]
  - \[ W = A (T_1 \overline{B}) = A + B \overline{T_1} \]
  - \[ X = (T_1 B) (B C \overline{D}) = \overline{B} T_1 + BCD \]
  - \[ Y = C \overline{D} + C D = CD + \overline{CD} \]
Verification Example: Manual Analysis

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

<table>
<thead>
<tr>
<th>Input BCD</th>
<th>Output Excess3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABCD</td>
<td>WXYZ</td>
</tr>
<tr>
<td>0000</td>
<td>0011</td>
</tr>
<tr>
<td>0001</td>
<td>0100</td>
</tr>
<tr>
<td>0010</td>
<td>0101</td>
</tr>
<tr>
<td>0011</td>
<td>0110</td>
</tr>
<tr>
<td>0100</td>
<td>0111</td>
</tr>
<tr>
<td>0101</td>
<td>1000</td>
</tr>
<tr>
<td>0110</td>
<td>1001</td>
</tr>
<tr>
<td>0111</td>
<td>1010</td>
</tr>
<tr>
<td>1000</td>
<td>1011</td>
</tr>
<tr>
<td>1011</td>
<td>1011</td>
</tr>
</tbody>
</table>

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

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?

Overview

- Part 1 - 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
    - Techniques for logic specification, minimization, simplification, technology mapping, and implementation
  - Technology Mapping
    - Mapping from AND, OR, and NOT to other gate types
  - Verification
    - Does the designed circuit meet the specifications?

- Part 2 - Programmable Implementation Technologies
  - Read-Only Memories, Programmable Logic Arrays, Programmable Array Logic
    - Technology mapping to programmable logic devices
Overview

- Why programmable logic?
- Programmable logic technologies
  - Read-Only Memory (ROM)
  - Programmable Logic Array (PLA)
  - Programmable Array Logic (PAL)
- VLSI Programmable Logic Devices - covered in VLSI Programmable Logic Devices reading supplement

Why Programmable Logic?

- Facts:
  - It is most economical to produce an IC in large volumes
  - Many designs required only small volumes of ICs
- Need an IC that can be:
  - Produced in large volumes
  - Handle many designs required in small volumes
- A programmable logic part can be:
  - made in large volumes
  - programmed to implement large numbers of different low-volume designs

Programmable Logic - Additional Advantages

- Many programmable logic devices are field-programmable, i.e., can be programmed outside of the manufacturing environment
- Most programmable logic devices are erasable and reprogrammable:
  - Allows “updating” a device or correction of errors
  - Allows reuse the device for a different design - the ultimate in re-usability!
  - Ideal for course laboratories
- Programmable logic devices can be used to prototype design that will be implemented for sale in regular ICs:
  - Complete Intel Pentium designs were actually prototype with specialized systems based on large numbers of VLSI programmable devices!

Programming Technologies

- Programming technologies are used to:
  - Control connections
  - Build lookup tables
  - Control transistor switching
- The technologies:
  - Control connections
    - Mask programming
    - Fuse
    - Antifuse
    - Single-bit storage element

The technologies (continued)

- Build lookup tables
  - Storage elements (as in a memory)
- Transistor Switching Control
  - Stored charge on a floating transistor gate
    - Erasable
    - Electrically erasable
    - Flash (as in Flash Memory)
  - Storage elements (as in a memory)

Technology Characteristics

- Permanent - Cannot be erased and reprogrammed
  - Mask programming
  - Fuse
  - Antifuse
- Reprogrammable
  - Volatile - Programming lost if chip power lost
    - Single-bit storage element
  - Non-Volatile
    - Erasable
    - Electrically erasable
    - Flash (as in Flash Memory)
Programmable Configurations

- **Read Only Memory (ROM)** - a fixed array of AND gates and a programmable array of OR gates.
- **Programmable Array Logic (PAL)** - a programmable array of AND gates feeding a fixed array of OR gates.
- **Programmable Logic Array (PLA)** - a programmable array of AND gates feeding a fixed array of OR gates.
- **Complex Programmable Logic Device (CPLD) /Field-Programmable Gate Array (FPGA)** - complex enough to be called “architectures” - See VLSI Programmable Logic Devices reading supplement.

Read Only Memory

- **Read Only Memories (ROM) or Programmable Read Only Memories (PROM)** have:
  - \(N\) input lines,
  - \(M\) output lines, and
  - \(2^N\) decoded minterms.
- **Fixed AND array** with \(2^N\) outputs implementing all \(N\)-literal minterms.
- **Programmable OR Array** with \(M\) outputs lines to form up to \(M\) sum of minterm expressions.

Programmable Array Logic (PAL)

- **The PAL** is the opposite of the ROM, having a programmable set of ANDs combined with fixed ORs.
- **Disadvantage**
  - ROM guaranteed to implement any \(M\) functions of \(N\) inputs. PAL may have too few inputs to the OR gates.
- **Advantages**
  - For given internal complexity, a PAL can have larger \(N\) and \(M\).
  - Some PALs have outputs that can be complemented, adding POS functions.
  - No multilevel circuit implementations in ROM (without external connections from output to input). PAL has outputs from OR terms as internal inputs to all AND terms, making implementation of multi-level circuits easier.

ROM, PAL and PLA Configurations

- **ROM**
  - Read Only Memory (ROM) or Programmable Read Only Memories (PROM) have:
    - \(N\) input lines,
    - \(M\) output lines, and
    - \(2^N\) decoded minterms.
  - **Fixed AND array** with \(2^N\) outputs implementing all \(N\)-literal minterms.
  - **Programmable OR Array** with \(M\) outputs lines to form up to \(M\) sum of minterm expressions.

- **PAL**
  - Programmable Array Logic (PAL) is the opposite of the ROM, having a programmable set of ANDs combined with fixed ORs.
  - **Disadvantage**
    - ROM guaranteed to implement any \(M\) functions of \(N\) inputs. PAL may have too few inputs to the OR gates.
  - **Advantages**
    - For given internal complexity, a PAL can have larger \(N\) and \(M\).
    - Some PALs have outputs that can be complemented, adding POS functions.
    - No multilevel circuit implementations in ROM (without external connections from output to input). PAL has outputs from OR terms as internal inputs to all AND terms, making implementation of multi-level circuits easier.
Programmable Logic Array (PLA)

- Compared to a ROM and a PAL, a PLA is the most flexible having a programmable set of ANDs combined with a programmable set of ORs.

- **Advantages**
  - A PLA can have large N and M permitting implementation of equations that are impractical for a ROM (because of the number of inputs, N, required).
  - A PLA has all of its product terms connectable to all outputs, overcoming the problem of the limited inputs to the PAL ORs.
  - Some PLAs have outputs that can be complemented, adding POS functions.

- **Disadvantage**
  - Often, the product term count limits the application of a PLA. Two-level multiple-output optimization reduces the number of product terms in an implementation, helping to fit it into a PLA.

Programmable Logic Array Example

- What are the equations for F₁ and F₂?
- Could the PLA implement the functions without the XOR gates?

3-input, 3-output PLA with 4 product terms