Design Hierarchy

- Combinatorial Circuits
  - A combinatorial logic circuit has:
    - A set of $m$ Boolean inputs,
    - A set of $n$ Boolean outputs, and
    - $n$ switching functions mapping the $2^n$ input combinations to a output such that the current output depends only on the current inputs.
  - A block diagram:
Hierarchical Design

- The function mapping inputs to outputs may be very complex
  - To control complexity, we decompose the function into smaller pieces called blocks
  - The blocks are subdivided into finer blocks
  - The "leaves" in the hierarchy are called primitive blocks
- Example: 16 input parity tree
  - Top Level: 16 inputs, one output
  - 2nd Level: Five 4-bit parity trees in two levels
  - 3rd Level: Three 2-bit exclusive-OR functions
  - Primitive level: Four 2-input NANDs
  - The design requires 5 X 3 X 4 = 60 two-input NAND gates

Reusable Functions and Design

- Whenever possible, we try to decompose a complex design into common, reusable function blocks
- These blocks are tested and well documented
- Computer-aided design (CAD) tools might include them in libraries
- Computer-aided manufacturing (CAM) tools might know how to manufacture and test them
- Other tools:
  - Schematic Capture
  - Logic Simulators
  - Timing Verifiers
  - Hardware Description Languages (HDL)
Top-Down verses 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 elements and combines them into larger and larger and more complex functions.
- 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 "sweats" the details.

Analysis Procedure

- Switching Functions from Logic Diagrams
- Given a logic diagram, the analysis process provides a set of Boolean equations, a truth table, or a verbal explanation of circuit behavior.
- Procedure:
  1. Determine that the circuit is combinational (no feedback loops), then:
  2. Identify and label all gate outputs that are a function of the input variables. Obtain the Boolean functions for these labeled gate outputs.
  3. Identify and label all gate outputs that are a function of inputs or previously labeled gates. Obtain Boolean functions for them.
  4. Repeat Step 2 until all outputs are completed.
  5. Back substitute until all functions are specified in terms of inputs only.
Analysis Example

- **Step 2:** Label all outputs of gates near inputs.

- **Write Boolean equations for them:**
  \[ T1 = \overline{B} + C \]
  \[ T2 = B \cdot \overline{E} \]

- **Analysis (Continued)**

- **Step 3:** Identify and label all gate outputs that are a function of inputs or previously labeled gates. Obtain Boolean functions for them.
  \[ T3 = \overline{D} + T2 \]

- **Step 4:** Repeat Step 3 until all done
  \[ T4 = T1 \cdot T3 \]
  \[ F = A + T4 \]
### Analysis (Continued)

- **Step 4:** Back substitute until all functions are specified in terms of inputs only
  - \[ F = A + T4 \]
  - \[ T4 = T1 \]
  - \[ T3 = D + T2 \]
  - \[ T2 = B \cdot E \]
  - \[ T1 = \overline{B} + C \]

- Substituting:
  - \[ T3 = \overline{D} + (B \cdot \overline{E}) \]
  - \[ T4 = (B + C) \cdot (\overline{D} + (B \cdot \overline{E})) \]
  - \[ F = A + (B + C) \cdot (\overline{D} + (B \cdot \overline{E})) \]

---

### Analysis Example: Code Converter

- **Step 2:** Label gates derived from inputs and develop Boolean functions.

- **Step 3:** Label the next stage of gates and develop Boolean functions.
Code Converter Analysis (Cont.)

- The process terminates with all gate outputs defined. Proceeding with Step 4, substituting,

Truth Tables from Logic Diagrams

1. Determine the number of input variables, n. There will be $2^n$ input vectors from zero to $2^{n-1}$. Enter them in the table.
2. Label the outputs of selected gates with symbols and enter a column for each one in the table.
3. Obtain the truth table for the outputs of those gates that are a function of only input variables.
4. Proceed to fill in the outputs of all gates that are derived from inputs and previously calculated terms.
Truth Tables from Logic Diagrams

- **Procedure:**
  - Determine the number of input variables, \( n \). There will be \( 2^n \) input vectors from zero to \( 2^n - 1 \). Enter them in the table.
  - Label the outputs of selected gates with symbols and enter a column for each one in the table.
  - Obtain the truth table for the outputs of those gates that are a function of only input variables.
  - Proceed to fill in the outputs of all gates that are derived from inputs and previously calculated terms.

- **Example:** Find the function table for the code converter.

### Code Converter Truth Table

- **Four inputs give 16 input vectors.**
- **Start with \( F_0, F_1 \) and \( z \).

<table>
<thead>
<tr>
<th>ABCD</th>
<th>( F_0 )</th>
<th>( F_1 )</th>
<th>( F_2 )</th>
<th>( w )</th>
<th>( x )</th>
<th>( y )</th>
<th>( z )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0010</td>
<td>0</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0011</td>
<td>0</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0100</td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0101</td>
<td>1</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0110</td>
<td>0</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0111</td>
<td>0</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1001</td>
<td>1</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1010</td>
<td>0</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1011</td>
<td>0</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1100</td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1101</td>
<td>1</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1110</td>
<td>0</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1111</td>
<td>0</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Truth Table Fill-In

- Now we can calculate $x$, $y$, and $F_2$.

<table>
<thead>
<tr>
<th>ABCD</th>
<th>F0</th>
<th>F1</th>
<th>F2</th>
<th>w</th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0010</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0011</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0100</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0101</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0110</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0111</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1001</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1010</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1011</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1100</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1101</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1110</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1111</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Complete Entries

- Finally we can fill in $w$ to complete the table:

<table>
<thead>
<tr>
<th>ABCD</th>
<th>F0</th>
<th>F1</th>
<th>F2</th>
<th>w</th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0010</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0011</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0100</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0101</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0110</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0111</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1000</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1001</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1010</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1011</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1100</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1101</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1110</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1111</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
What Does the Circuit Do?

- By inspection, the output variable vector $(w,x,y,z)$ is just the input variable vector $(A,B,C,D)$ plus three.
- The function(s) $F(A,B,C,D) = (w,x,y,z)$ are: "ADD THREE TO THE INPUT VECTOR"
- Function $F1$ has the meaning: "ADD ONE TO THE UPPER TWO BITS"
- Similarly, function $F2$ has the meaning: "ADD ONE TO THE UPPER BIT"
- Generally, it is not this obvious to figure out what the functions mean!

Final Note (and warning)

- The use of "Don't Cares" in the original specification can cloud the analysis.
  - Note that the functions for the "$w$" bit differ from the implementation in Ex. 3-2 of the book.
  - The book used "Don't Cares" to simplify the logic. The example here did not.
  - This can be seen by inspecting the two K-maps for the function $w$: 

---

Logic and Computer Design Fundamentals
© 2001 Prentice Hall, Inc
Logic Design: Functional Blocks

- **Analysis:** From a **design** to a **specification** of the behavior
  - Logic diagram to equations
  - Logic diagram to function table
  - "Word description" of circuit operation
- **Synthesis:** From a **specification** to design implementation
  - Define the problem
  - Generate function table or equations
  - Minimize the Boolean function
  - Implement the circuit

Combinatorial Logic Implementation

- A combinatorial logic circuit has:
  - A set of $m$ Boolean inputs,
  - A set of $n$ Boolean outputs, and
  - A function mapping inputs to outputs.
- We think of the function as $n$ separate Boolean functions of $m$ inputs
- **Procedure:**
  - Treat each output as a separate function
  - Minimize the equations for each function
  - Implement each function independently
  - Sometimes an implementation can share product or sum logic terms to arrive at a lower literal cost solution.
Design Procedure

- First, start with the specification of the circuit to be designed.
  - Note: this can sometimes require a lot of work to complete the specification process, especially if it is poorly specified initially.
- Second, follow these steps: We will study the design of a code converter to see these steps.
  - Identify the inputs and outputs
  - Derive truth table
  - Obtain simplified Boolean equations
  - Draw the logic diagram
  - Check your work to verify correctness.

Code Converter Design Example

- A code converter transforms one internal representation of data to another
- We will start with a table of the desired conversion and minimize the resulting multiple output Boolean function
- Sometimes terms can be shared to minimize the implementation cost
- The Problem:
  - Design a BCD to Excess-3 code converter
  - Specification:
    - BCD code -- 4-bit patterns "0000" to "1001" for digits 0 to 9 base 10
    - Excess-3 -- BCD code plus binary "0011" for digits 0 to 9 base 10
Example: BCD to Excess 3

Function 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 0 1 1</td>
</tr>
</tbody>
</table>

Note:

All BCD codes greater than "9" can be assigned "Don't Cares" in the K-Map. Such BCD codes are never possible.

Example (Cont.): BCD to Excess 3

- Map functions and find minimum cost SOP equations for each
Example (Cont.): BCD to Excess 3

- Next, we will manipulate the equations to expose some shared terms:

- The term \((C + D)\) can be used more than once to simplify the implementation
- See Fig. 3-10 in Mano and Kime for the implementation

An Alternative: BCD to Excess 3

- Another Approach: Excess-3 is defined as BCD plus 3.
- Adding 3 to BCD to Excess-3:

  \[
  \begin{array}{cccc}
  A & B & C & D \\
  + & 0 & 0 & 1 & 1 \\
  \hline
  W & x & y & z
  \end{array}
  \]

Here HA is a Half-Adder and FA is a Full-Adder (We will discuss these later in the chapter).