Chapter 2 :: Combinational Logic Design

Digital Design and Computer Architecture
David Money Harris and Sarah L. Harris

Chapter 2 :: Topics

• Introduction
• Boolean Equations
• Boolean Algebra
• From Logic to Gates
• Multilevel Combinational Logic
• X’s and Z’s, Oh My
• Karnaugh Maps
• Combinational Building Blocks
• Timing

Introduction

A logic circuit is composed of:
• Inputs
• Outputs
• Functional specification
• Timing specification

Circuits

• Nodes
  – Inputs: A, B, C
  – Outputs: Y, Z
  – Internal: n1
• Circuit elements
  – E1, E2, E3
  – Each circuit element is a circuit
Types of Logic Circuits

- **Combinational Logic**
  - Memoryless
  - Outputs determined by current values of inputs
- **Sequential Logic**
  - Has memory
  - Outputs determined by previous and current values of inputs

Rules of Combinational Composition

- Every circuit element is itself combinational
- Every node of the circuit is either designated as an input to the circuit or connects to exactly one output terminal of a circuit element
- The circuit contains no cyclic paths: every path through the circuit visits each circuit node at most once
- Example:

Boolean Equations

- Functional specification of outputs in terms of inputs
- Example:
  \[
  S = F(A, B, C_{in}) \\
  C_{out} = F(A, B, C_{in}) 
  \]

- Example:
  \[
  A \rightarrow \begin{array}{c}
  \oplus \\
  S \\
  \end{array} \quad B \rightarrow \begin{array}{c}
  \oplus \\
  C_{in} \\
  \end{array} \\
  A \oplus B \oplus C_{in} \\
  S = A \oplus B \oplus C_{in} \\
  C_{out} = AB + AC_{in} + BC_{in} 
  \]

Some Definitions

- Complement: variable with a bar over it \( A, B, C \)
- Literal: variable or its complement \( A, \bar{A}, B, \bar{B}, C, \bar{C} \)
- Implicant: product of literals \( AB\bar{C}, \bar{A}C, BC \)
- Minterm: product that includes all input variables \( ABC, ABC, ABC \)
- Maxterm: sum that includes all input variables \( (A+B+C), (\bar{A}+\bar{B}+\bar{C}), (\bar{A}+B+C) \)
Sum-of-Products (SOP) Form

- All Boolean equations can be written in SOP form
- Each row in a truth table has a minterm
- A minterm is a product (AND) of literals
- Each minterm is TRUE for that row (and only that row)
- The function is formed by ORing the minterms for which the output is TRUE
- Thus, a sum (OR) of products (AND terms)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Y</th>
<th>minterm</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>( \overline{A} \overline{B} )</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>( \overline{A} B )</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>( A \overline{B} )</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>( A B )</td>
</tr>
</tbody>
</table>

Thus, \( Y = \overline{A} \overline{B} + \overline{A} B + A \overline{B} + A B \).  

Boolean Equations Example

- You are going to the cafeteria for lunch
  - You won’t eat lunch (\( E \))
    - If it’s not open (\( O \)) or
    - If they only serve corn dogs (\( C \))
- Write a truth table for determining if you will eat lunch (\( E \)).

<table>
<thead>
<tr>
<th>O</th>
<th>C</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>( E )</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>( \cdot )</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>( \cdot )</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>( E )</td>
</tr>
</tbody>
</table>

Boolean Algebra

- Set of axioms and theorems to simplify Boolean equations
- Like regular algebra, but in some cases simpler because variables can have only two values (1 or 0)
- Axioms and theorems obey the principles of duality:
  - ANDs and ORs interchanged, 0’s and 1’s interchanged

<table>
<thead>
<tr>
<th>Axiom</th>
<th>Dual</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1: ( B = 0 ) if ( B \neq 1 )</td>
<td>A1’</td>
<td>( B = 1 ) if ( B \neq 0 )</td>
</tr>
<tr>
<td>A2: ( 0 = 1 )</td>
<td>A2’</td>
<td>( T = 0 )</td>
</tr>
<tr>
<td>A3: ( A \cdot 0 = 0 )</td>
<td>A3’</td>
<td>( 1 + 1 = 1 )</td>
</tr>
<tr>
<td>A4: ( A \cdot 1 = A )</td>
<td>A4’</td>
<td>( 0 + 0 = 0 )</td>
</tr>
<tr>
<td>A5: ( A \cdot 1 = A \cdot 0 = A )</td>
<td>A5’</td>
<td>( 1 + 0 = 0 + 1 = 1 )</td>
</tr>
</tbody>
</table>
T1: Identity Theorem

• \( B \cdot 1 = B \)
• \( B + 0 = B \)

T3: Idempotency Theorem

• \( B \cdot B = B \)
• \( B + B = B \)

T4: Identity Theorem

• \( \overline{B} = B \)

T5: Complement Theorem

• \( B \cdot \overline{B} = 0 \)
• \( B + \overline{B} = 1 \)
Boolean Theorems: Summary

<table>
<thead>
<tr>
<th>Theorem</th>
<th>Dual</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>B • 1 = B</td>
<td>T1'</td>
</tr>
<tr>
<td>T2</td>
<td>B • 0 = 0</td>
<td>T2'</td>
</tr>
<tr>
<td>T3</td>
<td>B + B = B</td>
<td>T3'</td>
</tr>
<tr>
<td>T4</td>
<td>B = B'</td>
<td>T4'</td>
</tr>
<tr>
<td>T5</td>
<td>B • 0 = 0</td>
<td>T5'</td>
</tr>
<tr>
<td></td>
<td>B + 1 = 1</td>
<td></td>
</tr>
</tbody>
</table>

Simplifying Boolean Expressions: Example 1

• \( Y = \overline{AB} + AB \)

Simplifying Boolean Expressions: Example 2

• \( Y = A(AB + ABC) \)
DeMorgan's Theorem

- \( Y = AB = \overline{A} + \overline{B} \)

- \( Y = \overline{A + B} = \overline{A} \cdot \overline{B} \)

Bubble Pushing

- Backward:
  - Body changes
  - Adds bubbles to inputs

\[
\begin{aligned}
& A & & Y \\
& B & \Downarrow & \\
& \overline{A} & \Leftrightarrow & Y
\end{aligned}
\]

\[
\begin{aligned}
& A & & Y \\
& B & \Downarrow & \\
& \overline{B} & \Leftrightarrow & Y
\end{aligned}
\]

- Forward:
  - Body changes
  - Adds bubble to output

\[
\begin{aligned}
& A & & Y \\
& B & \Downarrow & \\
& A & \Leftrightarrow & \overline{Y}
\end{aligned}
\]

- Begin at the output of the circuit and work toward the inputs.
- Push any bubbles on the final output back toward the inputs.
- Draw each gate in a form so that bubbles cancel.

Bubble Pushing Rules

\[
\begin{aligned}
& A & & Y \\
& B & \Downarrow & \\
& C & \Leftrightarrow & Y
\end{aligned}
\]

\[
\begin{aligned}
& A & & Y \\
& B & \Downarrow & \\
& D & \Leftrightarrow & Y
\end{aligned}
\]

What is the Boolean expression for this circuit?
**Bubble Pushing Example**

A bubble diagram showing a logic circuit with inputs A, B, C, and D, and output Y.

**From Logic to Gates**

- Two-level logic: ANDs followed by ORs
- Example: \( Y = \overline{ABC} + ABC + ABC \)

**Circuit Schematics with Style**

- Inputs on the left (or top)
- Outputs are on right (or bottom)
- Gates flow from left to right
- Straight wires are best

**Circuit Schematic Rules (cont.)**

- Wires always connect at a T junction
- A dot where wires cross indicates a connection between the wires
- Wires crossing without a dot make no connection
Multiple Output Circuits

- Output asserted corresponding to most significant TRUE input:

<table>
<thead>
<tr>
<th>A3</th>
<th>A2</th>
<th>A1</th>
<th>A0</th>
<th>Y3</th>
<th>Y2</th>
<th>Y1</th>
<th>Y0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0</td>
<td>0 0 0 1</td>
<td>0 0 1 0</td>
<td>0 0 1 1</td>
<td>0 1 0 0</td>
<td>0 1 0 1</td>
<td>0 1 1 0</td>
<td>0 1 1 1</td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>0 0 1 1</td>
<td>0 1 0 0</td>
<td>0 1 0 1</td>
<td>0 1 1 0</td>
<td>0 1 1 1</td>
<td>1 0 0 0</td>
<td>1 0 0 1</td>
</tr>
<tr>
<td>0 1 0 0</td>
<td>0 1 0 1</td>
<td>0 1 1 0</td>
<td>0 1 1 1</td>
<td>1 0 0 0</td>
<td>1 0 0 1</td>
<td>1 0 1 0</td>
<td>1 0 1 1</td>
</tr>
<tr>
<td>1 0 0 0</td>
<td>1 0 0 1</td>
<td>1 0 1 0</td>
<td>1 0 1 1</td>
<td>1 1 0 0</td>
<td>1 1 0 1</td>
<td>1 1 1 0</td>
<td>1 1 1 1</td>
</tr>
</tbody>
</table>

Priority Circuit Hardware

Don't Cares

<table>
<thead>
<tr>
<th>A3</th>
<th>A2</th>
<th>A1</th>
<th>A0</th>
<th>Y3</th>
<th>Y2</th>
<th>Y1</th>
<th>Y0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0</td>
<td>0 0 0 1</td>
<td>0 0 1 0</td>
<td>0 0 1 1</td>
<td>0 1 0 0</td>
<td>0 1 0 1</td>
<td>0 1 1 0</td>
<td>0 1 1 1</td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>0 0 1 1</td>
<td>0 1 0 0</td>
<td>0 1 0 1</td>
<td>0 1 1 0</td>
<td>0 1 1 1</td>
<td>1 0 0 0</td>
<td>1 0 0 1</td>
</tr>
<tr>
<td>0 1 0 0</td>
<td>0 1 0 1</td>
<td>0 1 1 0</td>
<td>0 1 1 1</td>
<td>1 0 0 0</td>
<td>1 0 0 1</td>
<td>1 0 1 0</td>
<td>1 0 1 1</td>
</tr>
<tr>
<td>1 0 0 0</td>
<td>1 0 0 1</td>
<td>1 0 1 0</td>
<td>1 0 1 1</td>
<td>1 1 0 0</td>
<td>1 1 0 1</td>
<td>1 1 1 0</td>
<td>1 1 1 1</td>
</tr>
</tbody>
</table>

Contention: X

- Contention: circuit tries to drive the output to 1 and 0
  - Actual value may be somewhere in between
  - Could be a legal 0, a legal 1, or in the forbidden zone
  - Might change with voltage, temperature, time, noise
  - Often causes excessive power dissipation

- Contention usually indicates a bug.
  - Fix it unless you are sure you know what you are doing.
- Warning: X is used for “don’t care” and contention - look at the context to tell them apart
**Floating: Z**

- Floating, high impedance, open, high Z
- Floating output might be 0, 1, or somewhere in between
  - A voltmeter won’t indicate whether a node is floating

**Tristate Buffer**

```
E
A  Y

E  A  Y
0  0  Z
0  1  Z
1  0  0
1  1  1
```

**Tristate Busses**

- Floating nodes are used in tristate busses
  - Many different drivers
  - Exactly one is active at any time

**Karnaugh Maps (K-Maps)**

- Boolean expressions can be minimized by combining terms
- K-maps minimize equations graphically
- \( PA + P\overline{A} = P \)

**K-map**

- Circle 1’s in adjacent squares
- In the Boolean expression, include only the literals whose true and complement form are not in the circle

\[
Y = \overline{AB}
\]
3-input K-map

\[
\begin{array}{|c|c|c|c|c|}
\hline
& A & B & C & Y \\
\hline
0 & 0 & 0 & 0 & 0 \\
& 1 & 1 & 1 & 1 \\
1 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 \\
\hline
\end{array}
\]

K-map Definitions

• Complement: variable with a bar over it \( \bar{A}, \bar{B}, \bar{C} \)
• Literal: variable or its complement \( A, \bar{A}, B, \bar{B}, C, \bar{C} \)
• Implicant: product of literals \( ABC, \bar{A}C, BC \)
• Prime implicant: implicant corresponding to the largest circle in a K-map

K-map Rules

• Every 1 in a K-map must be circled at least once
• Each circle must span a power of 2 (i.e. 1, 2, 4) squares in each direction
• Each circle must be as large as possible
• A circle may wrap around the edges of the K-map
• A “don't care” (X) is circled only if it helps minimize the equation

4-input K-map

\[
\begin{array}{|c|c|c|c|c|c|}
\hline
& A & B & C & D & Y \\
\hline
0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 1 & 1 & 1 \\
1 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 & 1 & 1 \\
1 & 0 & 1 & 0 & 1 & 1 \\
1 & 0 & 1 & 0 & 1 & 1 \\
1 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 \\
\hline
\end{array}
\]

\[
Y = \bar{A}C + ABD + ABC + BD
\]
K-maps with Don't Cares

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</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>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</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>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>X</td>
</tr>
</tbody>
</table>

Combinational Building Blocks

- Multiplexers
- Decoders

Multiplexer (Mux)

- Selects between one of N inputs to connect to the output
- \( \log_2 N \)-bit select input – control input

**Example:**

2:1 Mux

\[
\begin{align*}
S & \quad D_0 \quad 0 \\
D_1 & \quad 1
\end{align*}
\]

<table>
<thead>
<tr>
<th>S</th>
<th>D_0</th>
<th>D_1</th>
<th>Y</th>
<th>S</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Multiplexer Implementations

- **Logic gates**
  - Sum-of-products form

- **Tristates**
  - For an N-input mux, use N tristates
  - Turn on exactly one to select the appropriate input
Logic using Multiplexers

- Using the mux as a lookup table

\[
\begin{array}{c|c|c}
A & B & Y \\
0 & 0 & 0 \\
0 & 1 & 0 \\
1 & 0 & 0 \\
1 & 1 & 1 \\
\end{array}
\]

\[ Y = AB \]

- Reducing the size of the mux

\[
\begin{array}{c|c|c|c}
A & B & Y & A & Y \\
0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 1 \\
1 & 1 & 0 & 0 & 0 \\
\end{array}
\]

Decoders

- \( N \) inputs, \( 2^N \) outputs
- One-hot outputs: only one output HIGH at once

\[
\begin{array}{c|c|c|c|c|c}
A_1 & A_0 & Y_3 & Y_2 & Y_1 & Y_0 \\
0 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 \\
\end{array}
\]

Decoder Implementation
Logic using Decoders

- OR minterms

<table>
<thead>
<tr>
<th>Encoder</th>
<th>Minterm</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>( AB )</td>
</tr>
<tr>
<td>10</td>
<td>( AB )</td>
</tr>
<tr>
<td>01</td>
<td>( A\bar{B} )</td>
</tr>
<tr>
<td>00</td>
<td>( \bar{A}\bar{B} )</td>
</tr>
</tbody>
</table>

\[ Y = AB + \bar{AB} = A \oplus \bar{B} \]

Timing

- Delay between input change and output changing
- How to build fast circuits?

Propagation & Contamination Delay

- **Propagation delay**: \( t_{pd} = \text{max delay from input to output} \)
- **Contamination delay**: \( t_{cd} = \text{min delay from input to output} \)

- Delay is caused by
  - Capacitance and resistance in a circuit
  - Speed of light limitation

- Reasons why \( t_{pd} \) and \( t_{cd} \) may be different:
  - Different rising and falling delays
  - Multiple inputs and outputs, some of which are faster than others
  - Circuits slow down when hot and speed up when cold
Critical (Long) and Short Paths

Critical (Long) Path: \( t_{pd} = 2t_{pd_{\text{AND}}} + t_{pd_{\text{OR}}} \)
Short Path: \( t_{cd} = t_{cd_{\text{AND}}} \)

Glitches

• When a single input change causes multiple output changes

Glitch Example

• What happens when \( A = 0, C = 1, B \) falls?

Y = \( \overline{A}B + BC \)

Glitch Example (cont.)
Fixing the Glitch

Why Understand Glitches?

- Glitches don’t cause problems because of synchronous design conventions (which we’ll talk about in Chapter 3)
- But it’s important to recognize a glitch when you see one in simulations or on an oscilloscope
- Can’t get rid of all glitches – simultaneous transitions on multiple inputs can also cause glitches