Binary Logic Circuits

- Logic circuits perform operations on digital signals.
- Implemented using electronic circuits.
- Binary logic circuits take only two values: 0 and 1.

---

**Figure 2.1** A binary switch

- **(a)** Two states of a switch
- **(b)** Symbol for a switch

**Figure 2.2** A light controlled by a switch

- **(a)** Simple connection to a battery
- **(b)** Using a ground connection as the return path

**Figure 2.3** Two basic functions

- **(a)** The logical AND function (series connection)
  - \( L(x_1, x_2) = x_1 \cdot x_2 \)
  - \( L = 1 \) if \( x_1 = 1 \) and \( x_2 = 1 \)
  - \( L = 0 \) otherwise
- **(b)** The logical OR function (parallel connection)
  - \( L(x_1, x_2) = x_1 + x_2 \)
  - \( L = 1 \) if \( x_1 = 1 \) or \( x_2 = 1 \)
  - \( L = 0 \) otherwise

**Figure 2.4** A series-parallel connection

- \( L(x_1, x_2, x_3) = (x_1 + x_2) \cdot x_3 \)
Figure 2.5  An inverting circuit

Figure 2.6 A truth table for AND and OR

Figure 2.7 Three-input AND and OR

Figure 2.8 The basic gates

Figure 2.9 An OR-AND function

Figure 2.10 a Logic network
Axioms of Boolean Algebra

1. $0 \cdot 0 = 0$
2. $1 \cdot 1 = 1$
3. $0 \cdot 1 = 1 \cdot 0 = 0$
4. if $x = 0$ then $\overline{x} = 1$
5. if $x = 1$ then $\overline{x} = 0$

Single-Variable Theorems

5. $x \cdot 0 = 0$
6. $x \cdot 1 = x$
7. $x \cdot x = x$
8. $x = 0$
9. $\overline{x} = x$

Principle of Duality

- Axioms and theorems listed in pairs to show principle of duality.
- Given a logic expression, its dual is found by exchanging + operators and · operators and 0s and 1s.
- The dual of any true statement is true.

2- and 3-Variable Properties

10. $x \cdot y = y \cdot x$  
11. $x \cdot (y \cdot z) = (x \cdot y) \cdot z$  
12. $x \cdot (y + z) = x \cdot y + x \cdot z$

10b. $x + y = y + x$
11b. $x + (y + z) = (x + y) + z$
12b. $x + y \cdot z = (x + y) \cdot (x + z)$

Boolean Algebra

- 1849, George Boole published a scheme for describing logical thought and reasoning.
- In 1930s, Claude Shannon applied Boolean algebra to describe circuits built w/switches.
- Boolean algebra provides the theoretical foundation for digital design.
2- and 3-Variable Properties

13a. $x + x \cdot y = x$ Absorption
13b. $x \cdot (x + y) = x$
14a. $x \cdot y + x \cdot \overline{y} = x$ Combining
14b. $(x + y) \cdot (x + \overline{y}) = x$
15a. $x \cdot y = x + \overline{y}$ DeMorgan’s Thm
15b. $x + \overline{y} = \overline{x} \cdot \overline{y}$
16. $x + \overline{x} \cdot y = x + y$ $x \cdot (\overline{x} + y) = x \cdot y$

Figure 2.11 Proof of DeMorgan’s theorem

$$(x_1 + x_3) \cdot (\overline{x}_1 + \overline{x}_3) = x_1 \cdot \overline{x}_3 + \overline{x}_1 \cdot x_3$$

Figure 2.13 Verification of the distributive property

Venn diagram representation

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>$\overline{x}$</th>
<th>$\overline{y}$</th>
<th>$x \cdot y$</th>
<th>$x + y$</th>
<th>$\overline{x} \cdot \overline{y}$</th>
<th>$\overline{x} + \overline{y}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</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>0</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>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

FHS | RHS
Notation

• $\overline{x} = x' = \neg x = \text{NOT } x$
• $\overline{(x_1 \cdot x_2)} = x_1' + x_2 = (x_1 + x_2)' = \neg (x_1 + x_2) = \text{NOT}(x_1 + x_2)$
• $x_1 \cdot x_2 = x_1 \land x_2 = \neg (x_1' + x_2')$
• $x_1 + x_2 = x_1 \lor x_2$

Precedence of Operations

• In absence of parentheses, operations are performed in this order: NOT, AND, OR.

$x_1 \cdot x_2 + x_1' \cdot x_2' = (x_1 \cdot x_2) + ((x_1') \cdot (x_2'))$

Figure 2.14: Verification example

Figure 2.15: A function to be synthesized

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$f(x_1, x_2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 2.16: Two implementations of a function

Figure 2.17: Three-variable Minterms and Maxterms

<table>
<thead>
<tr>
<th>$m_i$</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>Min. minterm</th>
<th>Max. minterm</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$m_0 = x_1 \cdot x_2 \cdot x_3$</td>
<td>$M_0 = \neg x_1 + \neg x_2 + \neg x_3$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>$m_1 = x_1 \cdot x_2 \cdot x_3$</td>
<td>$M_1 = \neg x_1 + x_2 + \neg x_3$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>$m_2 = x_1 \cdot x_2 \cdot x_3$</td>
<td>$M_2 = x_1 + x_2 + \neg x_3$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>$m_3 = x_1 \cdot x_2 \cdot x_3$</td>
<td>$M_3 = x_1 + x_2 + \neg x_3$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>$m_4 = x_1 \cdot x_2 \cdot x_3$</td>
<td>$M_4 = x_1 + x_2 + \neg x_3$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>$m_5 = x_1 \cdot x_2 \cdot x_3$</td>
<td>$M_5 = \neg x_1 + x_2 + \neg x_3$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>$m_6 = x_1 \cdot x_2 \cdot x_3$</td>
<td>$M_6 = \neg x_1 + \neg x_2 + x_3$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>$m_7 = x_1 \cdot x_2 \cdot x_3$</td>
<td>$M_7 = \neg x_1 + \neg x_2 + x_3$</td>
</tr>
</tbody>
</table>
Figure 2.18 A three-variable function

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$f(x_1, x_2, x_3)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

(a) A minimal sum-of-products realization

(b) A minimal product-of-sums realization

Figure 2.20 Truth table for a three-way light controller

<table>
<thead>
<tr>
<th>$y_1$</th>
<th>$y_2$</th>
<th>$y_3$</th>
<th>$f(x_1, x_2, x_3)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

(a) Sum-of-products realization

(b) Product-of-sums realization

Figure 2.21 SOP implementation of the three-way light controller

Figure 2.22 Implementation of the three-way light controller
Design Entry

- Truth tables
  – Practical for only small circuits.
- Schematic capture
  – Interconnect symbols in some library.
  – Facilitates hierarchical design.
  – Good for larger circuits.
  – Difficult to use for very large circuits.

Design Entry (cont)

- Hardware description languages (HDLs).
  – Similar to a programming language.
  – VHDL and Verilog HDL are IEEE standards.
  – Provide design portability.
  – Allow for sharing and design reuse.
  – Support hierarchical design.
  – Can be combined with schematics.
Synthesis

- Logic synthesis, or logic optimization, is process to translate a truth table, schematic, or VHDL code into a network of logic gates.
- What makes a circuit good depends on the application.
- Converting logic description to a physical design entails technology mapping and layout synthesis.

Functional Simulation

- A functional simulator is used to determine if designed circuit operates correctly.
- User provides inputs values to the circuit.
- Simulator determines circuits response.
- User checks responses against required.
- A timing simulator can be used to check the performance of a design.

Representation of Digital Signals

- Each logic signal in a circuit is a data object in the VHDL code.
- Data objects in VHDL are assigned types.
- A simple type is BIT which is used for objects that can only take 2 values: 0 and 1.
- Other data types are introduced later.

VHDL - Very high speed integrated circuit hardware description language

- Revised standard in 1993.
- Originally used for documentation and simulation.
- Now, it also used for synthesis.
- Very complex language, but only a subset is needed to design wide range of circuits.

Representation of Digital Signals

- Each logic signal in a circuit is a data object in the VHDL code.
- Data objects in VHDL are assigned types.
- A simple type is BIT which is used for objects that can only take 2 values: 0 and 1.
- Other data types are introduced later.
How NOT to Write VHDL Code

- Novice tempted to write code with lots of variables and loops.
- This code style is difficult to relate to the circuit and should be avoided.
- Good guideline is that if designer cannot determine what circuit is doing from code, then circuit synthesized likely will be wrong.

Concluding Remarks

- Introduced concept of logic circuits.
  - Implemented using logic gates.
  - Described with Boolean algebra.
- Briefly introduced CAD tools.