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)

(b) The logical OR function (parallel connection)

Figure 2.4. A series-parallel connection.

Figure 2.5. An inverting circuit.

Figure 2.6. A truth table for AND and OR.
<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_1 \cdot x_2 \cdot x_3$</th>
<th>$x_1 \oplus x_2 \oplus x_3$</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>1</td>
<td>0</td>
<td>0</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>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>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</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>1</td>
</tr>
</tbody>
</table>

Figure 2.7. Three-input AND and OR.

Figure 2.8. The basic gates.

(a) AND gates

(b) OR gates

(c) NOT gate

Figure 2.9. An OR-AND function.

Figure 2.10a. Logic network.

(a) Network that implements $f = (x_1 \cdot x_2) \cdot x_3$

(b) Truth table for $f$

(c) Timing diagram

(d) Network that implements $g = x_1 \cdot x_2$

Figure 2.10b. Logic network.

Figure 2.11. Proof of DeMorgan’s theorem.
Figure 2.15. A function to be synthesized.

Figure 2.16. Two implementations of the function in Figure 2.15.

Figure 2.17. Three-variable minterms and maxterms.

Figure 2.18. A three-variable function.

Figure 2.19. Two realizations of the function in Figure 2.18.

Figure 2.20. NAND and NOR gates.

(a) Canonical sum-of-products

(b) Minimal-cost realization

(a) NAND gates

(b) NOR gates

(b) A minimal product-of-sums realization

Figure 2.20. NAND and NOR gates.
Figure 2.21. DeMorgan’s theorem in terms of logic gates.

Figure 2.22. Using NAND gates to implement a sum-of-products.

Figure 2.23. Using NOR gates to implement a product-of-sums.

Figure 2.24. Truth table for a three-way light control.

Figure 2.25a. SOP implementation of the three-way light control.

Figure 2.25b. POS implementation of the three-way light control.
Figure 2.26. Multiplexer.

Figure 2.27. Screen capture of the Waveform Editor.

Figure 2.28. Screen capture of the Graphic Editor.

Figure 2.30. A simple logic function.

module example1 (x1, x2, x3, f);
    input x1, x2, x3;
    output f;
    and (g, x1, x2);
    not (k, x2);
    and (h, k, x3);
    or (f, g, h);
endmodule

Figure 2.31. Verilog code for the circuit in Figure 2.30.

module example2 (x1, x2, x3, x4, f, g, h);
    input x1, x2, x3, x4;
    output f, g, h;
    and (z1, x1, x3);
    and (z2, x2, x4);
    or (g, z1, z2);
    or (z3, x1, ~x3);
    or (z4, ~x2, x4);
    and (h, z3, z4);
    or (f, g, h);
endmodule

Figure 2.32. Verilog code for a four-input circuit.
module example3 (x1, x2, x3, f);
input x1, x2, x3;
output f;
assign f = (x1 & x2) | (~x2 & x3);
endmodule

module example4 (x1, x2, x3, x4, f, g, h);
input x1, x2, x3, x4;
output f, g, h;
assign g = (x1 & x3) | (x2 & x4);
assign h = (x1 | ~x3) & (~x2 | x4);
assign f = g | h;
endmodule

// Behavioral specification
module example5 (x1, x2, x3, f);
input x1, x2, x3;
output f;
reg f;
always @(x1 or x2 or x3)
  if (x2 == 1)
    f = x1;
  else
    f = x3;
endmodule

module example6 (x1, x2, x3, f);
input x1, x2, x3;
output f;
reg f;
always @(x1 or x2 or x3)
  if (x2 == 1)
    f = x1;
  else
    f = x3;
endmodule
Figure P2.3. A timing diagram representing a logic function.

Figure P2.4. A timing diagram representing a logic function.