Homework 2 covers the combinational logic specification, and implementation. For the first three problems, we practice more on the specification and design using some popular functions. For the last two problems, we practice logic design using Karnaugh maps to derive the minimal sum of products and product of sums expressions.

1 Full Subtractor

• 30 points, graded by correctness

A full subtractor reads three binary values \((x, y, b_{in})\) and produces two binary outputs \((b_{out}, d)\). The relation between all variables can be formulated with an arithmetic equation \(-2b_{out} + d = x - y - b_{in}\).

A. Write the truth table of the subtractor.
B. Express the functions in a minimal sum of products form.
C. Use a minimal number of full subtractors (no other gates) to implement a 4-bit binary subtractor. Depict your design with a schematic diagram. Note that a 4-bit binary subtractor reads two 4-bit binary numbers \(X = (x_3, x_2, x_1, x_0)\), \(Y = (y_3, y_2, y_1, y_0)\) and a borrow-in bit \(b_0\), and produces the subtraction result with a borrow-out bit \(b_4\) and the difference, i.e. a 4-bit binary number \(D = (d_3, d_2, d_1, d_0)\). For example, when \(X = (0110), Y = (1011)\) and \(b_0 = 1\), the result is \(b_4 = 1\) and \(D = (1010)\).

<table>
<thead>
<tr>
<th>(x)</th>
<th>(y)</th>
<th>(b_{in})</th>
<th>(b_{out})</th>
<th>(d)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</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>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</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>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>

\[ b_{out}(x, y, b_{in}) = x'b_{in} + x'y + yb_{in} \]
\[ d(x, y, b_{in}) = xy'b_{in} + x'y'b_{in} + xyb_{in} + x'yb'_{in} \]
Figure 1: Schematic Diagram
2 Majority Function

- 10 points, graded by completeness

A majority function produces output \( f = 1 \), when half of all, or more input bits are true, otherwise, the output \( f = 0 \). For a 4-input majority function example, we have \( f(1, 1, 0, 0) = 1 \), and \( f(0, 1, 0, 0) = 0 \).

A. Write the truth table of the 4-input majority function.
B. Express the 4-input majority function using a minimal sum of product expression. Illustrate your design with a logic diagram.
C. Construct a 3-input majority function using 4-input majority function modules as basic building blocks (no other gates except 0, 1 constants, and inverters).
D. Can we construct a 5-input majority function using 4-input majority function modules as basic building blocks (no other gates except 0, 1 constants, and inverters)? Use one sentence to explain your answer.

\[
\begin{array}{cccc|c}
 a & b & c & d & f(a, b, c, d) \\
 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 1 & 0 \\
 0 & 0 & 1 & 0 & 0 \\
 0 & 0 & 1 & 1 & 1 \\
 0 & 1 & 0 & 0 & 0 \\
 0 & 1 & 0 & 1 & 1 \\
 0 & 1 & 1 & 0 & 1 \\
 0 & 1 & 1 & 1 & 1 \\
 1 & 0 & 0 & 0 & 0 \\
 1 & 0 & 0 & 1 & 1 \\
 1 & 0 & 1 & 0 & 1 \\
 1 & 0 & 1 & 1 & 1 \\
 1 & 1 & 0 & 0 & 1 \\
 1 & 1 & 0 & 1 & 1 \\
 1 & 1 & 1 & 0 & 1 \\
 1 & 1 & 1 & 1 & 1 \\
\end{array}
\]

(B) Any two or more input bits equal to 1 will generate the output being 1. Therefore, we have \( ab + ac + ad + bc + bd + cd \). We can also find the same expression through the K-map.
(C) We can construct a 3-input majority function by inputting the 3 input bits along with a 0 to the 4-bit majority function module.

(D) Yes, inverters and majority function modules together are a universal set. We can implement OR using the 4-input majority function module and NOT using the inverter. Since we are able to implement OR and NOT, we can construct a 5-input majority function using 4-input majority function modules, inverters, 0 and 1.

Elaboration: We output 1 whenever 3 or more input bits are 1, so we arrive at the expression $abc + abd + abc + acd + ace + ade + bcd + bce + bde + cde$. Using the DeMorgan’s law, we can show that the expression is equivalent to $(a' + b' + c')' + (a' + b' + d')' + (a' + b' + e')' + (a' + c' + d')' + (a' + c' + e')' + (a' + d' + e')' + (b' + c' + d')' + (b' + c' + e')' + (b' + d' + e')' + (c' + d' + e')'$, which can be implemented with multiples of OR and NOT.

3 Priority Encoder

- 10 points, graded by completeness

A priority encoder reads a vector of four binary bits $(a_3, a_2, a_1, a_0)$ and produces a binary number $(d_1, d_0)$ that presents the highest index of the input bit which is true. For example, when $(a_3, a_2, a_1, a_0) = (1, 1, 1, 0)$, the output $(d_1, d_0) = (1, 1)$; when
(a_3, a_2, a_1, a_0) = (0, 1, 0, 1), the output (d_1, d_0) = (1, 0); and when (a_3, a_2, a_1, a_0) = (0, 0, 0, 1), the output (d_1, d_0) = (0, 0). However, when none of the input bits asserts a true value, i.e. (a_3, a_2, a_1, a_0) = (0, 0, 0, 0), the output remains to be (d_1, d_0) = (0, 0).

A. Write the truth table of the priority encoder.
B. Express the encoder in a minimal sum of product form. Illustrate your design with a logic diagram.
C. How do we differentiate the cases when (d_1, d_0) = (0, 0)? Use one sentence to explain your answer.

<table>
<thead>
<tr>
<th>a_3</th>
<th>a_2</th>
<th>a_1</th>
<th>a_0</th>
<th>d_1</th>
<th>d_0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</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>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

\[d_1(a_3, a_2, a_1, a_0) = a_3 + a_2\]
\[d_0(a_3, a_2, a_1, a_0) = a_3 + a_2' a_1\]

We can distinguish the two scenarios with extra outputs. For example, we can feed \(d_1\) and \(d_0\) into a NOR gate and then AND the output with \(a_0\) to differentiate the two cases as demonstrated in Figure 4.
Figure 3: Logic Diagram

Figure 4: Design

\(\chi(d_1, d_0)\)

- \(x = 1\) when \(d_1, d_0\) are both 0
- \(y = 1\) when \(a_0 = 1\)
- \(y = 0\) when \(a_0 = 0\)
4 Minimal Sum of Products Expression

- 20 points, graded by completeness

Implementation from truth tables to sum of products expressions.
1. Use Karnaugh map to simplify function
   \[ f(a, b, c) = \sum m(1, 2, 4) + \sum d(3, 5, 6). \]
   List all possible minimal two-level sum of products expressions. Show the switching functions. No need for the schematic diagram.
2. Use Karnaugh map to simplify function
   \[ f(a, b, c, d) = \sum m(1, 4, 6, 7, 11, 12, 15) + \sum d(3, 5, 9). \]
   List all possible minimal two-level sum of products expressions. Show the switching functions. No need for the schematic diagram.
3. Use Karnaugh map to simplify function
   \[ f(a, b, c, d, e) = \sum m(0, 2, 5, 7, 8, 13, 15, 16, 18, 21, 24, 28, 29, 31) + \sum d(10, 22, 27). \]
   List all minimal two-level sum of products expressions. Show the switching functions. No need for the schematic diagram.

4.A

\[
\begin{array}{c|c|c}
\text{a} & \text{b} & \text{c} \\
0 & 0 & 0 \\
0 & 0 & 1 \\
0 & 1 & 1 \\
1 & 1 & 0 \\
1 & 1 & 1 \\
\end{array}
\]

\[ f(a, b, c) = b'c' + bc' + ac' \]
\[ = b'c + bc' + ab' \]
\[ = b'c + a'b + ac' \]
\[ = b'c + a'b + ab' \]
\[ = a'c + bc' + ac' \]
\[ = a'c + bc' + ab' \]
\[ = a'c + a'b + ac' \]
\[ = a'c + a'b + ab' \]

Figure 5: 4.1 solution

5 Minimal Product of Sums Expression

- 20 points, 5.2 is graded by correctness
- 10 points, 5.1 & 5.3 are graded by completeness

Implementation from truth table to product of sums expressions.
1. Use Karnaugh map to simplify function
   \[ f(a, b, c) = \sum m(2, 6, 7) + \sum d(1, 4). \]
   List all possible minimal two-level product of sums expressions. Show the switching functions. No need for the schematic diagram.
2. Use Karnaugh map to simplify function
4.B

![Diagram](Image)

\[ f(a, b, c, d) = bc'd' + a'b + cd + a'd \]
\[ = bc'd' + a'b + cd + b'd \]

Figure 6: 4.2 solution

4.C

![Diagram](Image)

\[ f(a, b, c, d, e) = a'ce + cd'e + bce + c'd'e' + b'c'e' + ab'd'e' \]
\[ = a'ce + cd'e + bce + c'd'e' + b'c'e' + ab'd'e' + abc'd \]
\[ = ab'd'e' + cd'e + bce + b'c'e' + a'c'e' + a'ce \]

Figure 7: 4.3 solution
\[ f(a, b, c, d) = \sum m(1, 4, 5, 9, 13, 14) + \sum d(3, 6, 11, 12, 15). \]

List all possible minimal two-level product of sums expressions. Show the switching functions. No need for the schematic diagram.

3. Use Karnaugh map to simplify function

\[ f(a, b, c, d, e) = \sum m(3, 11, 12, 13, 14, 19, 22, 23, 25, 28, 29, 30, 31) + \sum d(7, 9, 15, 27) \]

List all minimal two-level product of sums expressions. Show the switching functions. No need for the schematic diagram.

5.A

Figure 8: 5.1 solution

5.B

Figure 9: 5.2 solution
Figure 10: 5.3 solution