**B441/541 – Fall 2004 – Homework One**

*NOTE: Work your answers out on scratch paper first. Sloppy and/or disorganized answers are penalized*

1. Give an efficient mixed-logic synthesis of

\[ Y = (A \cdot B + C) \cdot (D \oplus E + B) \]

using two-input *nor* gates (LS02), two-input *exclusive-or* (\( \oplus \)) gates (LS86) and inverters (LS04), and signals A.L, B.H, C.L, D.H, E.L and Y.L. Try to minimize the number of chips.

**ANSWER:**

1. *First, draw a schematic of the logic:*

   ![Schematic](image1)

2. *Add the I/O constraints*

   ![I/O Constraints](image2)

3. *Implement the gates. We don’t have LS00 nand gates available, so all of the ands and ors must be implemented with LS02s. In this solution, I started at the output side and worked backwards. I might have started on the input side, or somewhere in the middle, but not with the xor gate because the LS86 can implement xor with four different polarity combinations).*

   ![Gate Implementation](image3)

4. *Use inverters to fix polarity mis-matches. Even though there are a lot of inverters, the design is implemented using three ICs, with one xor gate to spare.*

   ![Inverter Implementation](image4)

*Solutions that alter the logic of the implementation do not receive full credit, even if they result in fewer gates. Implementation should not be confused with design.*
2. Consider the logic schematic

(a) Express this logic as a system of boolean equations. The system below only identifies the internal signals that fan out.

\[ X = C + D \]
\[ Y = B + X \]
\[ W = [(A \cdot Y) \cdot (A + Y)] + X \cdot (E \cdot \overline{C}) \]

(b) Implement this specification with LS00, LS02 and LS04 devices. The I/O constraints are A.H, B.H, E.L and W.L. C and D may be either polarity.

Here’s one way to do it. The unlabeled inverters could be realized with the remaining unused LS00 and LS02 gates.
3. (a) Using the K-map below, derive a reduced (near-minimal) sum-of-products expression for the function whose truth table is shown to the right. Unspecified or conflicting cases, if any, may be regarded as don't-cares.

(b) Using boolean algebra, further optimize to an equivalent multi-level term with fewer operations than the \( SoP \) form.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>X</td>
<td>0</td>
<td>X</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>X</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>e</td>
<td>τ</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

The map-entered K-map below shows how the information in the truth table is used to fill in the map. It is also valid, but more time consuming, to build two regular K-maps for the cases in which \( E = 1 \) and \( E = 0 \). In this case, it is unnecessary to use any 1s in the covering of the \( E \)s.

The answers below are not unique.

(a) \[ \overline{A}DE + \overline{A}BE + ABC + BCD + \overline{ABC} \]

(b) \[ \overline{AE}(D + B) + A(B \oplus C) + BCD \]
4.

(a) Give a mixed-logic synthesis of the logic expression \( A \cdot (B \cdot C + \overline{B} \cdot D) + B \cdot \overline{C} \cdot \overline{D} \) using only 74LS06 open-collector inverters and pull-up resistors (which you may assume are appropriately sized). Assume all inputs and outputs must have positive (\( T = H \)) polarity.

*Open collector circuits give us positive logic wired-ands and negative logic wired-ors.*

Incorporate the inverters to match the polarities.
(b) A SoP form of the original expression, \( A B C + A \overline{B} D + B \overline{C} \overline{D} \). Can also be implemented with '06s, using a more regular logic array layout structure (possibly at the cost of more gates). Draw such an implementation.

We need SOP form of the original function, \( Y = A B C + A \overline{B} D + B \overline{C} \overline{D} \). Since we have only LS06 inverters, two in series are needed to get an open-collector voltage for \( A, B \) and \( C \). There is also also use a pull-up on the output to get a proper digital value.
5. Design a one-bit full adder-subtractor taking operand inputs \( A \) and \( B \); carry/borrow input \( C/B_{in} \); and add/subtract mode input \( M \); and generating outputs \( S \) for the sum/difference and \( C/B_{out} \) for the propagated carry/borrow bit.

When \( M = 1 \) the device should behave as a full adder as described in class; and when \( M = 0 \) it should behave as a full subtractor. The functions should compose in a ripple-carry way to get \( n \)-bit adder/subtractor combinations.

From the truth table one can see that the \( S/D \) output is independent of \( M \).

\[
\begin{array}{cccc|cc}
M & A & B & C/B_i & S/D & C/B_o \\
\hline
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 \\
1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 \\
\end{array}
\]

**subtraction**

**addition**

\[
S/D = A \oplus B \oplus C/B_i \\
C_o = B C_i + \overline{M} \overline{A} B + \overline{M} A C_i + M A B + M C_i A \\
= B C_i + \overline{M} \overline{A} (B + C_i) + M A (B + C_i) \\
= B C_i + (\overline{M} A + M A) (B + C_i) \\
= B C_i + (\overline{M} A + M A) (B + C_i) \\
= \text{etc.}
\]
6. In certain technologies (to be described later in this course) the only gate provided is a \(2^n\)-input multiplexer. Suppose you have only 2-input multiplexers for implementing the expression

\[ C \cdot D + A \cdot C + B \cdot \overline{C} + A \cdot C \cdot \overline{C} \]

That is, you must convert this expression to one composed of nested combinations of:

\[ \text{mux}(s, x, y) \overset{\text{def}}{=} (s \cdot x) + (\overline{s} \cdot y) \]

(a) Design a mux-based implementation of \(E\). If you employed a specific technique (e.g. boolean algebra, etc.) to accomplish this task, explain it for extra credit. For even more extra credit, implement your strategy in software.

(b) Synthesize a solution using only 74LS125 tri-state buffers and 74LS04 inverters.

As discussed in class, three approaches could be taken to solving this problem.

1. Implement each of and, or and not with a multiplexor,
2. Build a tree of 15 muxes to implement a four-variable look-up table.
3. Algebraically manipulate the specification expression to place it in a nested-mux form.