• You are allowed 75 minutes.

• You may consult your own 8.5′′ × 11′′ double-sided sheet of notes, but nothing else (e.g., no textbooks, no other notes, calculators, etc.).

• Perform all work on the midterm itself. Use the backs of the pages if necessary. Do not use scratch paper.

• Explain your answers.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Value</th>
<th>Score</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15</td>
<td>Logic Minimization</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>Boolean and Combinational Logic</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>Programmable Logic</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>Arithmetic Logic</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td>Combinational Logic Design</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>40</td>
<td>FSM Implementation</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>100</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
1. (15 pts.) Logic Minimization. Give a **minimal sum-of-products form** expression for F. Show your work.

\[ F = \overline{A}(\overline{D} + B) + B(D + \overline{A}) + D(\overline{C} + B) \]

\[ F = \overline{AD} + BD + \overline{CD} \]

2. (10 pts.) Boolean and Combinational Logic. Show how the 2:1 mux below can be used to implement G by giving expressions in the dashed boxes to indicate what values must be placed on the corresponding wires.

\[ G = AB + \overline{AC} + BC \]

Splitting on A:
\[ = AB + \overline{A}C + BC(A + \overline{A}) \]
\[ = \overline{A}(C + BC) + A(B + BC) \]
\[ = \overline{A}(C + B\overline{C}) + A(B) \]
\[ = \overline{A}(C + B) + A(B) \]

Splitting on B or C would have yielded:
\[ G = \overline{B}(\overline{A}C) + B(1) \]
\[ G = \overline{C}(B) + C(B + \overline{A}) \]
3. (5 pts.) Programmable Logic. Add connections to the ROM below so that it implements $H$.

$$H = A_2A_1 + A_2\overline{A_0} + A_2A_1A_0$$

Expanding to get minterms:

$$= A_2A_1A_0 + A_2A_1\overline{A_0} + A_2A_1A_0 + A_2A_1\overline{A_0} + A_2A_1A_0$$

$$= A_2A_1A_0 + A_2A_1\overline{A_0} + A_2A_1A_0$$

4. (10 pts.) Arithmetic Logic. In the dashed boxes below, indicate what logic value (0, 1, or X) must be on the corresponding wires.
5. (20 pts.) Combinational Logic Design. Design a circuit that takes a year and indicates whether the year is divisible by 4. The input year is encoded with four, 4-bit BCD digits, as follows:

\[ THOUSANDS_{3:0}, HUNDREDS_{3:0}, TENS_{3:0}, ONES_{3:0} \]

For example, 2014 would appear as:

\[
\begin{align*}
THOUSANDS_{3:0} &= 0010 \\
HUNDREDS_{3:0} &= 0000 \\
TENS_{3:0} &= 0001 \\
ONES_{3:0} &= 0100 \\
\end{align*}
\]

Your circuit should produce a single-bit output, \( DIV4 \), which is true if and only if the year is divisible by 4.

You may specify your design using boolean expressions or a schematic.

HINT: Look for a repeating pattern in the numbers you wish to detect. Years that are divisible by 4 end in 00, 04, 12, 16, 20, 24, 28, 32, 36, ...

The pattern we're looking are numbers ending in 00, 04, 08, 12, 16, 20, 24, 28, 32, 36, etc. In other words:

- if TENS is even and ONES is 0, 4, or 8, or
- if TENS is odd and ONES is 2 or 6

Let

\[
\begin{align*}
ONESIS0 &= ONES_3 ONES_2 ONES_1 ONES_0 \\
ONESIS2 &= ONES_3 ONES_2 ONES_1 ONES_0 \\
ONESIS4 &= ONES_3 ONES_2 ONES_1 ONES_0 \\
ONESIS6 &= ONES_3 ONES_2 ONES_1 ONES_0 \\
ONESIS8 &= ONES_3 ONES_2 ONES_1 ONES_0 \\
\end{align*}
\]

Then

\[
DIV4 = TENS_0 \cdot (ONESIS0 + ONESIS4 + ONESIS8) + TENS_0 \cdot (ONESIS2 + ONESIS6)
\]
6. FSM Implementation. Examine this state transition diagram for a 2-bit up-down counter, and then answer the two questions that follow.

This counter takes two inputs

- count: C=0 to hold, C=1 to count
- direction: D=0 for up, D=1 for down

(a) (20 pts.) Using the given state encoding, provide expressions for the counter’s next state logic ($D_3, D_2, D_1, D_0$). You may assume $\overline{RST}$ will be handled via preset/clear on the flip-flops.

<table>
<thead>
<tr>
<th>State</th>
<th>$Q_3$</th>
<th>$Q_2$</th>
<th>$Q_1$</th>
<th>$Q_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</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>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

$$D_0 = Q_0 \overline{C} + Q_1 CD + Q_3 C \overline{D}$$
$$D_1 = Q_1 \overline{C} + Q_2 CD + Q_0 C \overline{D}$$
$$D_2 = Q_2 \overline{C} + Q_3 CD + Q_1 C \overline{D}$$
$$D_3 = Q_3 \overline{C} + Q_0 CD + Q_2 C \overline{D}$$
(b) (20 pts.) Using this state encoding instead, again give expressions for the machine's next state logic \((D_1, D_0)\). You may again assume \(RST\) will be handled via preset/clear on the flip-flops.

<table>
<thead>
<tr>
<th>State</th>
<th>(Q_1)</th>
<th>(Q_0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[
\begin{array}{cccc|cc}
Q_1 & Q_0 & C & D & D_1 & D_0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
& & 0 & 1 & 0 & 0 \\
& & 1 & 0 & 0 & 1 \\
& & 1 & 1 & 1 & 1 \\
0 & 1 & 0 & 0 & 0 & 1 \\
& & 0 & 1 & 0 & 1 \\
& & 1 & 0 & 1 & 0 \\
& & 1 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 \\
& & 0 & 1 & 1 & 0 \\
& & 1 & 0 & 1 & 1 \\
& & 1 & 1 & 0 & 1 \\
1 & 1 & 0 & 0 & 1 & 1 \\
& & 0 & 1 & 1 & 1 \\
& & 1 & 0 & 0 & 0 \\
& & 1 & 1 & 1 & 0 \\
\end{array}
\]

For \(D_1\):
\[
D_1 = Q_1\overline{C} + Q_1Q_0D + Q_1\overline{Q}_0D + \overline{Q}_1\overline{Q}_0CD + \overline{Q}_1Q_0\overline{C}D
\]

\[
D_0 = \overline{Q}_0C + Q_0\overline{C}
\]