1.) Using the rules of Boolean Algebra, minimize this function. You do not need to list the rules, but you must show the steps you use.

\[
\frac{\overline{A\overline{B} + C)}D + \overline{A} \cdot \overline{B} + \overline{D}E}{(A\overline{B} + C)D + \overline{A} \cdot \overline{B} + \overline{D}E}
\]

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

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

\[
= \overline{A} + \overline{C} + B + \overline{D}
\]
2.) Draw the **state diagram** for the following circuit – your diagram should be as simple as possible:

Create a string recognizer that determines whether the last three input values, including the current one, are 000 or 111. Remember that at startup, the system has not yet seen any inputs.
3.) For the given decimal value, convert it into a 4-bit number in the given format. If the number CANNOT be represented in that format, write “FAIL” in the corresponding square.

<table>
<thead>
<tr>
<th></th>
<th>4-bit unsigned</th>
<th>4-bit sign-magnitude</th>
<th>4-bit 2’s complement</th>
</tr>
</thead>
<tbody>
<tr>
<td>+8</td>
<td>1000</td>
<td>FAIL</td>
<td>FAIL</td>
</tr>
<tr>
<td>+1</td>
<td>0001</td>
<td>0001</td>
<td>0001</td>
</tr>
<tr>
<td>-1</td>
<td>FAIL</td>
<td>1001</td>
<td>1111</td>
</tr>
<tr>
<td>-8</td>
<td>FAIL</td>
<td>FAIL</td>
<td>1000</td>
</tr>
</tbody>
</table>
4.) In class we showed how to make big muxes out of small muxes, and big decoders out of small decoders. Do the same for encoders. Specifically, build an 8:3 basic encoder out of two 4:2 basic encoders, and any basic gates you need. Your circuit should be as simple as possible, and use the provided 4:2 encoders.
5.) For the following circuits, assume $T_{setup}$ and $T_{hold}$ are 0.1 ns, $Clk\rightarrow Q$ is 1.0 ns, all gate delays are 1.0 ns, and the clock period is 100 ns. "A" is an external input to the circuit, and all DFFs were initialized to 0 before the circuit started running. Figure out whether F can be metastable. If it can, give a specific scenario when this can happen. If it cannot, explain why.

Yes.

A goes from 0 to 1 either

1. Between 0.9 ns... 1.1 ns before the clock edge
2. Between 1.9 ns... 2.1 ns before the clock edge
6.) Implement the following circuits using only 3-LUTs. Your circuit should be as simple as possible:

\[ Y = A \times B \times C \times D \]  
\[ t = A \times B \times C \]  
\[ y = t \times D \]

\[ Z = E + F \times G \]

As an example of this process, I have implemented \( W = \text{XOR}(J,K,L) \) for you below.
7.) A saturating up-counter is the same as a normal up-counter, but instead of rolling over to 0 after the maximum value, it stays at that maximum value. That is, it goes 0, 1, 2, ..., MaxVal-1, MaxVal, MaxVal, MaxVal, ...

Implement a 2-bit saturating up-counter. You can assume you are given premade D flip-flops, muxes, decoders, encoders, AND, OR, XOR, Invert, and any other basic gates. Your counter will operate as a saturating up-counter when the “enable” signal is true, and will hold its current value when the “enable” signal is false. However, if the “load” signal is true you will load a new parallel value regardless of the value of the “enable” signal.
8.) On slide 183 we implemented serial communication by hooking the shift_out of the left shift register to the shift_in of the right shift register, and have the left shift register do “Load, shift, shift, shift” repeatedly.

Imagine we ALSO hook the shift_out of the right shift register to the shift_in of the left shift register. We have both shift registers do “Load, shift, shift, shift” repeatedly.

This does something useful. **Give a simple, direct summary of what this circuitry accomplishes.** As an example, slide 183 accomplishes “*moves 4-bit data serially from the left system to the right system over 4 clock cycles*”.

*Moves 4-bit data serially from the left system to the right, AND from the right system to the left, over 4 clock cycles.*