1. Design a combinational logic subsystem with three inputs, $I_3$, $I_2$, $I_1$, and two outputs, $O_1$, $O_0$, that behaves as follows. The outputs indicate the highest index of the inputs that is driven high. For example, if $I_3$ is 0, $I_2$ is 1, $I_1$ is 1, then $O_1$, $O_0$ would be 10 (i.e., $I_2$ is the highest input set to 1). If none of the inputs is 1, then the outputs would be 00.
   a. Specify the subsystem by filling out a complete truth table for the two outputs.

<table>
<thead>
<tr>
<th>$I_3$</th>
<th>$I_2$</th>
<th>$I_1$</th>
<th>$O_1$</th>
<th>$O_0$</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>0</td>
<td>1</td>
</tr>
<tr>
<td>0</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>0</td>
</tr>
<tr>
<td>1</td>
<td>0</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>
</tr>
<tr>
<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>
</tr>
</tbody>
</table>

   b. Write a specification of this function in Verilog.
   (note, this is using simplified product-of-sums as listed in part c)

   ```verilog
module f(o1, o0, i3, i2, i1);
    input i3, i2, i1;
    output o1, o0;
    assign o1 = !i2 & i1 | i3;
    assign o0 = i3 | i2;
endmodule
```
c. Find the minimized Sum of Products description using K-maps.

K-map of O0

<table>
<thead>
<tr>
<th>I3</th>
<th>I2'I1</th>
<th>I3</th>
<th>I2'I1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

O0 = I2'I1 + I3

K-map of O1

<table>
<thead>
<tr>
<th>I3</th>
<th>I2'I1</th>
<th>I3</th>
<th>I2'I1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

O1 = I3 + I2

d. Implement the subsystem using 2 x 4:1 multiplexers.

![Multiplexer Diagram]

e. If implemented using a ROM, what size ROM is required? Why?

The ROM would require 16 bits total. Each O0 and O1 is a function of I3, I2, and I1, which have a total of 8 combinations each.

f. Compare your solutions in parts (c) and (d). Which is simpler and why (i.e., what criteria are you using to measure complexity)?

The multiplexor circuit is simpler because its implementation would use less transistors than a look-up table.
2. Scientists have discovered that the Venusians use a base 16 number system. However, the digits are quite different than the ones to which we are accustomed. The first row below is 0 through 7, and the second row is 8 through F. Your task is to design a combinational logic subsystem to decode a hexadecimal digit in the range of 0 (0000) through F (1111) to drive a seven-segment display for the Martian version of the hexadecimal digits (0-7 in the top row, 8-F in the bottom row). The LED segments are numbered counter clockwise starting at the horizontal LED at the bottom (LED₀). The middle LED is LED₆.

\[
\begin{array}{ccccccc}
4 & 3 & 2 & 1 & 0 & 5 & 6 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 0 & 0 & 0 & 0 & 0 & 1 \\
3 & 0 & 0 & 0 & 0 & 1 & 1 \\
4 & 0 & 0 & 0 & 1 & 1 & 0 \\
5 & 0 & 0 & 1 & 1 & 1 & 0 \\
6 & 0 & 1 & 1 & 1 & 1 & 0 \\
7 & 1 & 1 & 1 & 1 & 1 & 1 \\
\end{array}
\]

a. Specify the function by filling out a complete truth table for each of the seven segment drivers.

<table>
<thead>
<tr>
<th>O0</th>
<th>O1</th>
<th>O2</th>
<th>O3</th>
<th>O4</th>
<th>O5</th>
<th>O6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0001</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0010</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0011</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0100</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0101</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0110</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0111</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1000</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1001</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1010</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1011</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1100</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1101</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1110</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1111</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Write each as a Verilog specification.

```
module o0(o0, i0, i1, i2, i3, i4);
    input i4, i3, i2, i1;
    output o0;

    assign o0 =
        (!i3 & i2 & i1 & i0) |
        (i3 & !i2 & !i1 & !i0) |
        (i3 & !i2 & !i1 & i0) |
        (i3 & !i2 & i1 & !i0) |
        (i3 & !i2 & i1 & i0) |
        (i3 & i2 & !i1 & !i0) |
        (i3 & i2 & !i1 & i0) |
        (i3 & i2 & i1 & !i0) |
        (i3 & i2 & i1 & i0);
endmodule

module o1(o1, i0, i1, i2, i3, i4);
    input i4, i3, i2, i1;
    output o1;

    assign o1 =
        (!i3 & i2 & !i1 & !i0) |
        (!i3 & i2 & i1 & !i0) |
        (!i3 & i2 & i1 & i0) |
        (i3 & i2 & !i1 & !i0) |
        (i3 & i2 & !i1 & i0) |
        (i3 & i2 & i1 & !i0) |
        (i3 & i2 & i1 & !i0) |
        (i3 & i2 & i1 & i0);
endmodule
```
module o2(o2, i0, i1, i2, i3, i4);
  input i4, i3, i2, i1;
  output o2;

  assign o2 =
    (!I3 & I2 & !I1 & !I0) |
    (!I3 & I2 & !I1 & I0) |
    (!I3 & I2 & I1 & !I0) |
    (!I3 & I2 & I1 & I0) |
    (I3 & !I2 & !I1 & !I0) |
    (I3 & !I2 & !I1 & I0) |
    (I3 & I2 & !I1 & I0) |
    (I3 & I2 & I1 & !I0);
endmodule

module o3(o3, i0, i1, i2, i3, i4);
  input i4, i3, i2, i1;
  output o3;

  assign o3 =
    (!I3 & !I2 & I1 & I0) |
    (!I3 & I2 & !I1 & !I0) |
    (!I3 & I2 & !I1 & I0) |
    (!I3 & I2 & I1 & !I0) |
    (!I3 & I2 & I1 & I0) |
    (I3 & !I2 & !I1 & !I0) |
    (I3 & !I2 & !I1 & I0) |
    (I3 & !I2 & I1 & !I0) |
    (I3 & !I2 & I1 & I0);
endmodule
module o4(o4, i0, i1, i2, i3, i4);
    input i4, i3, i2, i1;
    output o4;
    assign o4 =
        (!I3 & !I2 & I1 & !I0) | 
        (!I3 & !I2 & I1 & I0) | 
        (!I3 & I2 & !I1 & !I0) | 
        (!I3 & I2 & !I1 & I0) | 
        (!I3 & I2 & I1 & !I0) | 
        (!I3 & I2 & I1 & I0) | 
        (I3 & !I2 & !I1 & !I0) | 
        (I3 & !I2 & !I1 & I0) | 
        (I3 & !I2 & I1 & !I0) | 
        (I3 & !I2 & I1 & I0) | 
        (I3 & I2 & !I1 & !I0) | 
        (I3 & I2 & !I1 & I0) |
endmodule

module o5(o5, i0, i1, i2, i3, i4);
    input i4, i3, i2, i1;
    output o5;
    assign o5 =
        (!I3 & !I2 & !I1 & I0) | 
        (!I3 & !I2 & !I1 & I0) | 
        (!I3 & !I2 & I1 & !I0) | 
        (!I3 & !I2 & I1 & !I0) | 
        (!I3 & !I2 & !I1 & !I0) | 
        (!I3 & !I2 & !I1 & I0) | 
        (!I3 & !I2 & !I1 & !I0) | 
        (!I3 & !I2 & !I1 & !I0) | 
        (!I3 & !I2 & !I1 & I0) | 
        (!I3 & !I2 & !I1 & I0) | 
        (!I3 & !I2 & !I1 & I0) | 
        (!I3 & !I2 & I1 & !I0) | 
        (!I3 & !I2 & I1 & !I0) |
endmodule
module o6(o6, i0, i1, i2, i3, i4);
    input i4, i3, i2, i1;
    output o6;

    assign o6 =
        (!i3 & i2 & i1 & !i0) |
        (!i3 & i2 & i1 & i0) |
        (i3 & !i2 & !i1 & !i0) |
        (i3 & i2 & i1 & !i0) |
        (i3 & i2 & i1 & i0) |
        (i3 & i2 & i1 & !i0);

endmodule

c. Develop the minimized gate-level implementation using the K-map method, minimizing each K-map independently.

\[
\begin{array}{|c|c|c|c|}
\hline
I4 & I3 & O0 = I4 + I3 I2 I1 \\
\hline
00 & 00 & 0 & 0 & 0 & 0 \\
\hline
01 & 00 & 0 & 0 & 0 & 0 \\
\hline
11 & 11 & 0 & 1 & 1 & 1 \\
10 & 10 & 1 & 1 & 1 & 1 \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|c|c|}
\hline
I4 & I3 & O1 = I4 I3 + I3 I1 + I3 I2 \\
\hline
00 & 00 & 0 & 0 & 0 & 0 \\
\hline
01 & 00 & 0 & 0 & 0 & 0 \\
\hline
11 & 11 & 0 & 1 & 1 & 1 \\
10 & 10 & 1 & 1 & 1 & 1 \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|c|c|}
\hline
I4 & I3 & O2 = I4' I3 + I4 I2 I1 + I4 I3 I2' + I3 I2 I1' \\
\hline
00 & 00 & 0 & 0 & 0 & 0 \\
\hline
01 & 01 & 1 & 1 & 1 & 1 \\
\hline
11 & 11 & 0 & 1 & 0 & 1 \\
10 & 10 & 1 & 1 & 0 & 0 \\
\hline
\end{array}
\]
<table>
<thead>
<tr>
<th></th>
<th>I2 I1</th>
<th>I4 I3</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

O3 = I3 I2" + I4’ I3 + I4’ I2 I1 + I3 I2 I1’

<table>
<thead>
<tr>
<th></th>
<th>I2 I1</th>
<th>I4 I3</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

O4 = I4 I3’ + I4’ I3 + I2 I4

<table>
<thead>
<tr>
<th></th>
<th>I2 I1</th>
<th>I4 I3</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

O5 = I2 I1 + I4’ I3 + I4 I3’ + I2’ I1’ I3’ + I4’ I1

<table>
<thead>
<tr>
<th></th>
<th>I2 I1</th>
<th>I4 I3</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

O6 = I4 I3’ I2’ I1’ + I2 I3

O4 = I4 I3’ + I4’ I3 + I2 I4

O5 = I2 I1 + I4’ I3 + I4 I3’ + I2’ I1’ I3’ + I4’ I1

O6 = I4 I3’ I2’ I1’ + I2 I3

O6 = I4 I3’ I2’ I1’ + I2 I3
d. Repeat (c), but this time, minimize so as to exploit shared product terms wherever possible, as though the implementation target is a PLA. (There may be several solutions)

### O0

<table>
<thead>
<tr>
<th>I2 I1</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ O0 = I4' I3 + I4 I3' + I3 I2 I1 \]

### O1

<table>
<thead>
<tr>
<th>I2 I1</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ O1 = I4 I3 + I3 I2' I1 + I3 I2 I1 + I3 I2 I1' \]

### O2

<table>
<thead>
<tr>
<th>I2 I1</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ O2 = I4' I3 + I4 I2 I1 + I4 I3 I2' + I3 I2 I1' \]

### O3

<table>
<thead>
<tr>
<th>I2 I1</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ O3 = I3 I2' I1' + I3 I2' I1 + I4' I2 I1 + I3 I2 I1' \]
e. How does the complexity of your answers to (c) and (d) compare? Which is simpler and why? What criteria are you using to measure the complexity of these two different implementations?

Although D may not be the most simplified sum of products expression, it will be easier to implement than C because many of the terms may be shared between the outputs.

3. Your textbook presents the concept of a Master-Slave flip-flop, constructed from two cascaded stages of R-S flip-flops (see Figure 6.18 on page 269).
   a. Master-Slave flip-flops implemented in this way exhibit the phenomena of “ones catching.” Explain what it is and why it happens. Is it possible for a Master-Slave flip-flop to “catch zeros”? Justify your answer!
   "One’s catching” means the output of the Master-Slave flip-flop will be a 0, until a 1 is sensed at the input. Once this happens, the output will become 1 at the next clock cycle and remain at 1 for the remaining clock cycles or until a reset signal is received. “zero-catcher” can be constructed
b. Suppose that you implement a Master-Slave flip-flop as two cascaded D flip-flops, the first stage a positive edge triggered D FF and the second stage a negative edge triggered D FF. Draw a timing chart with a clock waveform and a D input that oscillates from 0 to 1 or 1 to 0 each time the clock is low. Make sure you show how the two flip-flops’ outputs change in response to input and clock changes.

4. Your textbook presents a negative edge-triggered D flip-flop using NOR gates (see Figure 6.24 on page 272).
   a. Implement a positive-edge triggered D flip-flop function using NAND gates only.
b. Explain the timing behavior of this circuit. Label your internal circuit nodes, show their waveforms on the timing chart, and use these waveforms to briefly explain why the triggering works.

A changes on the rising edge of CLK, and Q changes on the falling edge of CLK.

5. Design a 3-bit counter that implements the following sequence: 000, 010, 100, 110, 111, 101, 011, 001, and repeat. Design the counter with a reset input that causes the counter to enter the 000 state.

   a. Write a Verilog specification for this counter.

   module weirdcounter(clk, rst, out);
   
   parameter s0 = 3’b000;
   parameter s1 = 3’b010;
   parameter s2 = 3’b100;
   parameter s3 = 3’b110;
   parameter s4 = 3’b111;
   parameter s5 = 3’b101;
   parameter s6 = 3’b011;
   parameter s7 = 3’b001;
   input clk, rst;
   output [2:0] out;
   reg out;
   reg [2:0] ps, ns;
   
   assign out = ps;
   
   always@(posedge clk) begin
     if(rst) ps <= s0;
     else ps <= ns;
   end
   always@(ps)
     case(ps)
       s0: ns = s1;
s1: ns = s2;
s2: ns = s3;
s3: ns = s4;
s4: ns = s5;
s5: ns = s6;
s6: ns = s0;
default: ns = s0;
end
endmodule

b. Design the next state functions and minimize for PLA-based implementation.

<table>
<thead>
<tr>
<th>PS2</th>
<th>PS1</th>
<th>PS0</th>
<th>NS2</th>
<th>NS1</th>
<th>NS0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</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>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<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>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>
<td>0</td>
</tr>
</tbody>
</table>

(The PLA optimized form is the same as the minimized Sum-of Products form)

NS0

<table>
<thead>
<tr>
<th>PS2</th>
<th>PS1</th>
<th>PS0</th>
<th>NS0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

NS0 = PS1 PS0 + PS2 PS0 + PS2 PS1

NS1

<table>
<thead>
<tr>
<th>PS2</th>
<th>PS1</th>
<th>PS0</th>
<th>NS1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

NS1 = PS1’ PS0’ + PS2 PS1’ + PS2 PS0’

NS2

<table>
<thead>
<tr>
<th>PS2</th>
<th>PS1</th>
<th>PS0</th>
<th>NS2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

NS2 = PS1 PS0’ + PS2 PS1 + PS2 PS0’
c. Show how to implement this counter using 2:1 multiplexers and D flip-flops only. You may assume that the D FFs have a reset input.

d.