Review Problem 19

- Extend this Verilog code to also show the letter "A" on input pattern 1010 (ten) and "F" on pattern 1111 (fifteen).

```verilog
module seg7 (bcd, leds);
    input logic [3:0] bcd;
    output logic [6:0] leds;

    always_comb begin
        case (bcd)
            // BCD[]
            // 3210
            4'b0000: leds = 7'b0111111;
            4'b0001: leds = 7'b0000110;
            4'b0010: leds = 7'b1011011;
            4'b0011: leds = 7'b1001111;
            4'b0100: leds = 7'b1100110;
            4'b0101: leds = 7'b1101101;
            4'b0110: leds = 7'b1111101;
            4'b0111: leds = 7'b0000111;
            4'b1000: leds = 7'b1111111;
            4'b1001: leds = 7'b1101111;
            default: leds = 7'b1X;
        endcase
    end
endmodule
```
Process Line Control Example (cont.)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Meaning</th>
<th>Accept</th>
<th>Long</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Start/Short</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Entering</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Start/Spec</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Entering</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Short</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Forbidden</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>In Spec</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Too Long</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Rule:

Rods must be spaced by \( > \text{Spec} + 5\% \)

\[
\begin{array}{ccc}
\text{Accept} & A \\
\hline
C1 & 0 & 0 & 1 & 0 \\
C1 & 0 & 0 & 0 & X
\end{array}
\]

Accept = ABC

\[
\begin{array}{ccc}
\text{Long} & A \\
\hline
C1 & 0 & 0 & 0 & 0 \\
C1 & 0 & 0 & 1 & X
\end{array}
\]

Long = AC
Logical Function Unit

Create a unit that can compute the AND, OR, or XOR of two inputs A and B, based upon control lines C0 and C1.

<table>
<thead>
<tr>
<th>C1</th>
<th>C0</th>
<th>A</th>
<th>B</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>??</td>
<td>??</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>AND</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>0</td>
</tr>
<tr>
<td>XOR</td>
<td>1</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>1</td>
</tr>
</tbody>
</table>

Similar to the main computation unit in a Microprocessor
Logical Function Unit (cont.)

- Implementation:
Debugging Complex Circuits

- Complex circuits require careful debugging
  - Rip up and retry?
- Ex. Debug a 9-input odd parity circuit
  - True if an odd number of inputs are true
Debugging Complex Circuits (cont.)

\[ \text{A} \]
\[ \text{B 3-Parity Out} \]
\[ \text{C} \]

\[ \begin{array}{cccc}
A & \quad & B & \quad & \text{Out} \\
\quad & \quad & \quad & \quad & \\
\quad & \quad & \quad & \quad & \\
\quad & \quad & \quad & \quad & \\
\end{array} \]
Debugging Approach

- Test all behaviors.
  - All combinations of inputs for small circuits, subcircuits.

- Identify any incorrect behaviors.

- Examine inputs and outputs to find earliest place where value is wrong.
  - Typically, trace backwards from bad outputs, forward from inputs.
  - Look at values at intermediate points in circuit.

- DO NOT RIP UP, DEBUG!