Finite state machine design

You’ve been tasked by an airline to build a device that smashes every 3rd suitcase that is checked.

To implement this circuit, design a finite-state machine with a 1-bit input, \textit{sensor}, and a 1-bit output, \textit{div3}, which is 1 when the number of cycles that \textit{sensor} has been 1 is a number divisible by 3.

\[
\begin{array}{|c|c|c|}
\hline
\text{Current State} & \text{(Input) sensor} & \text{Next State} \\
\hline
\hline
\hline
\hline
\hline
\end{array}
\]
Finite state machine (FSM) design

A universal serial bus (USB) transmits data across a cable one-bit at a time. Sometimes we can process the data as it is received from the cable. For example, rather than wait for all 32 bits to arrive and then process all of the data through a single large 32-bit ALU, we could process the data using a bit-serial design, processing each bit as it arrives. The design lets us use a 1-bit ALU to process 32-bits of data and in turn lets us run the circuit that processes incoming data at an extremely high clock speed.

In this problem, you will design a bit-serial comparator FSM that compares 2 data-inputs from two serial buses \(a\) and \(b\). These data inputs should be interpreted as \(N\)-bit unsigned binary integers \(A\) and \(B\) and receive bits starting from the least significant bit to the most significant bit. A reset signal resets your comparator FSM to a starting state that starts with the assumption that \(A\) and \(B\) are equal. Your FSM generates two output bits based on the current state of the circuit: \(c_1c_0\). These bits should encode the following information.

<table>
<thead>
<tr>
<th>(c_1c_0)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>(A == B)</td>
</tr>
<tr>
<td>01</td>
<td>(A &lt; B)</td>
</tr>
<tr>
<td>10</td>
<td>(A &gt; B)</td>
</tr>
</tbody>
</table>

The following is an example input and output stream (\(A = 5, B = 12\) followed by \(A = 23, B = 3\)).

\[
\begin{array}{cccccccccccc}
\text{time} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\
\text{reset} & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
\text{a} & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\
\text{b} & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
\text{c1} & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
\text{c0} & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]
Scaling Decoders

Decoders are an important circuit for implementing memories (like register files). Below is the truth table for a 1-to-2 decoder.

![Decoder symbol](image1)

**Figure 1. Decoder symbol**

<table>
<thead>
<tr>
<th>EN</th>
<th>A</th>
<th>E[0:1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2'b00</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2'b00</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2'b01</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2'b10</td>
</tr>
</tbody>
</table>

**Figure 2. Decoder truth table**

```verilog
module decoder2 (E, A, EN);
    input A;
    input EN;
    output [1:0] E;

    and a0(E[0], EN, ~A);
    and a1(E[1], EN, A);
endmodule // decoder2
```

**Figure 3. Decoder Verilog**

Design a 2-to-4 decoder **using only 1-to-2 decoders**.

![Decoder symbol](image2)

**Continued on the following page.**
Design a 3-to-8 decoder using only 1-to-2 and 2-to-4 decoders.