Q1: H & H 4.2

Part a

First, we will want to write out a truth table from the if-else logic. We will use X to denote don't care, as in the case of, if \( a[0] \) is high, we don't care what any other input is, but we know \( y \) will be \( y[1:0] = 11 \). This helps simplify our truth table substantially.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

\[2'6\ldots\]
\[\uparrow\uparrow\]
\[y[1] \quad y[0]\]

This hierarchy of \( a[0] \rightarrow a[3] \) follows how the if-else statement is evaluated.

\[\rightarrow \text{this final case is the default statement, and in this case } a[1:0] \text{ is 0,0, which gives our y values.}\]

If we had gone into K-maps you could step through to prove the optimal result, but we can readily see it here,

First consider \( y[1] \), its simplified truth table is

\[\uparrow\uparrow\]

\[\text{We see that } y[1] \text{ is only high if } a[0] \text{ or } a[1] \text{ are high, } y[0] \text{ is only low if } a[0] \text{ and } a[1] \text{ are low.}\]

Therefore, \( a[0] \rightarrow \quad y[1] \)

\( a[1] \rightarrow \quad y[1] \)

Completely describes the behavior in the Verilog code for \( y[1] \).
Now, doing the same for y[0],

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>0</td>
</tr>
</tbody>
</table>

We see y[0] is 1 only when a[0] is high or when a[2] is high and a[1] is low.

We have to specify the and when a[1] is low because if a[1] is high then y[0] should be low and we want to reflect that in our logic. We don't have to specify for a[0] or a[3] in this case because we don't care, as they don't alter the high y[0] when a[2] is high.

Then,

\[ a[0] \quad \land \quad a[2] \quad \rightarrow \quad y[1] \]

So combining it all gives the Verilog if statement,

\[ \begin{align*}
    a[0] & \quad \land \\
    a[1] & \\
    a[2] & \quad \rightarrow \\
    a[3] & \quad \text{don't care} \\
\end{align*} \]

Q1 part b.

Specify the pinout for our made-up 7 segment display.

Let's say we have `led[6:0]`, 7 bits, each corresponding to a single segment, where setting it high turns on the segment, and low turns it off.

```
   led[0]
  __________|
 |  led[5]  |
 |     led[6]     |
 |__ led[1]__|
    |
   led[4]|
    \
     led[3]
```

Now we assign values in a lookup case statement for our 4-bit input, `a[3:0]`, representing the input 0-F, to make unique representations:

- Use lower case `b, c, d` and upper case for `A, E, F`.

Verilog next page.
module display(
    input [3:0]in,
    output [6:0]out);

//for pins,
//a[0] is top  a[1] is top right  a[2] is bottom right
//a[6] is center

//b'1 is high for pin
//b'0 is low

//Decoder will implement as combo logic
assign out = (in == 4'h0) ? 7'b011_1111 :
              (in == 4'h1) ? 7'b000_0110 :
              (in == 4'h2) ? 7'b101_1011 :
              (in == 4'h3) ? 7'b100_1111 :
              (in == 4'h4) ? 7'b110_0110 :
              (in == 4'h5) ? 7'b110_1101 :
              (in == 4'h6) ? 7'b111_1101 :
              (in == 4'h7) ? 7'b000_0111 :
              (in == 4'h8) ? 7'b111_1111 :
              (in == 4'h9) ? 7'b110_1111 :
              (in == 4'hA) ? 7'b111_0111 :
              (in == 4'hB) ? 7'b111_1100 :
              (in == 4'hC) ? 7'b010_0111 :
              (in == 4'hD) ? 7'b101_1110 :
              (in == 4'hE) ? 7'b111_1001 :
              (in == 4'hF) ? 7'b111_0011 : 7'b000_0000;
endmodule
Q2:
For the circuit below, assume that all gates have the same delay (including inverters) of 5 ns, complete the timing diagrams.
Q3: Complete the timing diagram for the circuit below for 10 clock cycles. Is the reset signal active low or active high? **Active low**

You have learned from class that this circuit is a crucial component in sequential logic design. What is it and why do you think it might be useful? **This is a DFF, a main component in building registers, state machines and ALU.**