Midterm Histogram

Average: 17.8  3/4 >= 16  1/2 >= 19  1/4 >= 21
1.) Simplify the following K-maps, and write the corresponding Boolean Equation. You should create the simplest Sum of Products form possible.

a.)

\[ F = \overline{AC} + ABD + BC \]

b.)

\[ F = BD + \overline{AC} + BC \]
2.) 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.

\[ F = (B \cdot (CDE + D \cdot (\overline{C} + E))) \cdot (AB + BC) \]

\[
\begin{align*}
F &= B \cdot D \cdot (E + \overline{E} + \overline{E}) \cdot (\overline{A} + \overline{B})(\overline{B} + \overline{C}) \\
&= BD (E + \overline{E}) (\overline{A}B + \overline{A}C + BB + BC) \\
&= BD (B + \overline{A}C) \\
&= \overline{B}BD + \overline{A}BCD \\
&= \overline{A}BCD
\end{align*}
\]
3.) Draw the state diagram for the following circuit. YOU DO NOT HAVE TO DO THE STATE TABLE. YOU DO NOT HAVE TO IMPLEMENT THE CIRCUIT. Your state diagram should be as simple as possible.

Your machine has two inputs: A and B. Your machine should output TRUE once it has seen at least one 1 on input A, and at least one 1 on input B, and continue outputting TRUE thereafter. The 1's can be seen on the same or different cycles. The output reacts as soon as you see the proper pattern, including the current inputs to the circuit.
4.) For the following state diagram, implement the circuit. The state encoding is given to you in each state. You may use premade D-flipflops and any other basic gates. Your implementation should be as simple as possible.

\[ \begin{array}{c|c|c|c|c} \hline PS & IN & OUt & NS \\ \hline 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & \uparrow \text{Same} \\ \hline \end{array} \]
5.) For the following circuit, fill in the timing diagram below. All DFFs are positive-edge-triggered, like we’ve been using in class. Note that the time for any gate is very small compared to the grid lines in the timing diagram, and you can assume $T_{\text{setup}}$ and $T_{\text{hold}}$ are 0.

Assume that the value of all DFFs starts at 0.

Timing Diagram:

2nd copy (in case you make a mistake. CLEARLY indicate which one I should grade):
Half Adder

A_1

+ B_1

S_1

A_2

+ B_2

S_2

A_3

+ B_3

S_3

Full Adders
Half Adder

<table>
<thead>
<tr>
<th>Ai</th>
<th>Bi</th>
<th>Carry</th>
<th>Sum</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>0</td>
</tr>
</tbody>
</table>

Carry = Ai Bi

Sum = \( \overline{A_i} B_i + A_i \overline{B_i} \) = \( A_i \oplus B_i \)

Half-adder Schematic
## Full Adder

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Ci</th>
<th>CO</th>
<th>S</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>0</td>
<td>1</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>0</td>
<td>1</td>
</tr>
<tr>
<td>1</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>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ S = A \oplus B \oplus C_i \]

\[ C_o = AB + AC_i + BC_i \]

![Full Adder Circuit Diagram](image-url)
Full Adder Implementation
Multi-Bit Addition

A_3 A_2 A_1 A_0
+ B_3 B_2 B_1 B_0
-------------------
  A_3 A_2 A_1 A_0
  B_3 B_2 B_1 B_0

Diagram showing the addition of two multi-bit numbers, with carry and sum outputs for each bit position.
Multi-Bit Addition in Verilog, Parameters

module uadd #(parameter WIDTH=8) (out, a, b);
    output logic [WIDTH-1:0] out;
    input logic [WIDTH-1:0] a, b;

    always_comb begin
        out = a + b;
    end
endmodule

module add4 #(parameter W=22) (out, a, b, c, d);
    output logic [W+1:0] out;
    input logic [W-1:0] a, b, c, d; // W wide

    logic [W:0] ab, cd;
    uadd #(C.WIDTH(W)) addAB (.out(ab), .a(a), .b(b));
    uadd #(C.WIDTH(W)) addCD (.out(cd), .a(c), .b(cd));
    uadd #(C.WIDTH(W+1)) addABCD (.out(out), .a(ab), .b(cd));
endmodule