1.) Produce the minimum Sum of Products equation of the following K-Map.

\[
\begin{array}{cccc}
A & B & C & D \\
0 & 0 & 1 & 1 \\
0 & X & 1 & 0 \\
1 & 1 & X & X \\
0 & 0 & 0 & 1 \\
\end{array}
\]
2.) For the Boolean equation below:

\[ F = ((ADE + E \cdot (D + \overline{A})) \cdot C) \cdot (\overline{BC} + CD) \]

Using Boolean algebra, minimize this function. You don’t need to list the rules, but you must show the steps you use.
3.) For the following circuit, draw the corresponding K-map. **DO NOT solve the K-map, DO NOT draw the circuit**, merely set it up so we could then start circling items.

A family has two parents (P1 and P2) and two kids (K1 and K2). They want to vote on whether to eat out (input TRUE) or eat at home (input FALSE). Each person gets a vote, but a parent’s vote counts twice as much as a kid’s (i.e. consider it as TWO votes at once). Please specify the logic that will decide if the family will eat out (output TRUE).

The resulting circuit should be as simple as possible, so give the K-map solver as many possibilities as possible.
4.) For the following circuit, fill out the timing diagram for all gate outputs (C, D, F). The vertical lines in the timing diagram are 1ns apart, and each gate has a delay of 1ns. Also, two copies of the same diagram are given, in case you make a mistake. Make it clear which version you want graded.

Duplicate (if you make a mistake above):

---

4.

4.) For the following circuit, fill out the timing diagram for all gate outputs (C, D, F). The vertical lines in the timing diagram are 1ns apart, and each gate has a delay of 1ns. Also, two copies of the same diagram are given, in case you make a mistake. Make it clear which version you want graded.

Duplicate (if you make a mistake above):
5.) For the following Verilog of an FSM, draw the **state diagram**.

```verilog
module twiddler (clk, reset, w, out);
    input  logic  clk, reset, w;
    output logic [1:0] out;

    // State variables.
    enum { A, B, C } ps, ns;

    // Next State logic
    always_comb begin
        case (ps)
            A: if (w) ns = C;
               else ns = B;
            B: ns = C;
            C: if (w) ns = A;
               else ns = C;
        endcase
        if (ps == A && ns == B)
            out = 2'b01;
        else if (ps == C)
            out = 2'b10;
        else
            out = 2'b00;
    end

    // DFFs
    always_ff @(posedge clk)
        if (reset)
            ps <= C;
        else
            ps <= ns;

endmodule
```
6.) For the following state diagram, implement the circuit. You can use premade DFFs, and any basic gates. Your circuit should be as simple as possible.

The state encoding is UP = 0, DOWN = 1