Name: *Solution*

<table>
<thead>
<tr>
<th>Problem</th>
<th>Max Score</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>2A</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>2B</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>2C</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td><strong>100</strong></td>
<td><strong>100</strong></td>
</tr>
</tbody>
</table>
1. Combinational Logic (25 points)

What function is implemented by the multiplexer below. S2 is the most significant bit of the control signals, S0 is the least significant (e.g., S2=1, S1=1, S0=0 selects input #6).

\[ Z = A'B'C' + AB'D' + ABC' + ABCD' + A'B'C + A'BC \]

where the underlined terms are don't cares

\[ Z \text{ can be simplified to:} \]

\[ Z = A'B'C' + ABC' + C'D' \] (but this is not necessary for the problem)
2. Sequential Logic (75 points)

The following Verilog was found among old papers in a dusty drawer of a now defunct dot-com company. Unfortunately, there were no comments in the code.

```verilog
module Mystery (In, Clk, Reset, A, B, Out);
    input   In, Clk, Reset;
    output  A, B, Out;
    reg     [5:0] state;
    reg     [5:0] next_state;
    wire    [2:0] count;

    parameter S0 = 6'b000000;
    parameter S1 = 6'b000001;
    parameter S2 = 6'b000010;
    parameter S3a = 6'b100011;
    parameter S3b = 6'b001011;
    parameter S4aa = 6'b100100;
    parameter S4ab = 6'b010100;
    parameter S4bb = 6'b001100;

    always @(posedge Clk) begin
        if (Reset) begin state = `S0; end
        else begin state = next_state; end
    end

    always @(In or state) begin
        case(state)
            `S0:   next_state = `S1;
            `S1:   next_state = `S2;
            `S2:   if (In) next_state = `S3b; else next_state = `S3a;
            `S3a:  if (In) next_state = `S4ab; else next_state = `S4aa;
            `S3b:  if (In) next_state = `S4bb; else next_state = `S4ab;
            `S4aa: next_state = `S0;
            `S4ab: next_state = `S0;
            `S4bb: next_state = `S0;
        endcase
    end

    assign count = state[2:0];
    assign A   = state[5];
    assign B   = state[3];
    assign Out = state[4];
endmodule
```
A. (25 points) Complete its state diagram and clearly label all transitions with the input value. For the values for A, B, and Out in each state, use Output = [A, B, Out]. You can write the triplet within the state bubble. Use the template below. Clearly label all transitions and outputs.

Output = \{A, B, Out\}
B. (25 points) Simulate the state machine for the following sample input waveforms. You are provided the values for the input signals, Reset and In. Fill in the details for the signals A, B, and Out. Also, please indicate the state the FSM is in for each clock cycle (i.e., write the symbolic name of the state in the space provided). Assume that the FSM is initially in an unknown state. Assume that all FFs are positive edge-triggered.
C. (25 points) When we look to see how this state machine was actually implemented, we find that there is a 3-bit binary counter in the circuit and 2 individual D flip-flops with some very minimal logic around them. We need to figure out if we have an up to date description for the module. The Verilog describing the actual circuit implementation is below.

```verilog
module MysteryImplementation (In, Clk, Reset, A, B, Out);
  input   In, Clk, Reset;
  output  A, B, Out;
  reg     [2:0]counter, A, B;

  always @(posedge Clk) begin
    if (Reset | counter[2]) counter = 0;
    else                 counter = counter + 1;
  end

  always @(posedge Clk) begin
    if (Reset | counter[2]) A = 0;
    else                   A = A | (counter[1] & ~In);
  end

  always @(posedge Clk) begin
    if (Reset | counter[2]) B = 0;
    else                   B = B | (counter[1] & In);
  end

  assign Out = A & B;
endmodule
```

It turns out this is a correct implementation of the Mystery module? In fact, Mystery and MysteryImplementation describe the same state diagram. What is the correspondence of states between them? For each of the states of the Mystery module list the states for the sequential elements of the MysteryImplementation module – note that there are 3 state elements in the MysteryImplementation: the counter (3 bits), the A FF (1 bit), and the B FF (1 bit). Fill in the table below.

<table>
<thead>
<tr>
<th>Mystery</th>
<th>MysteryImplementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0 = 6'b000000</td>
<td>000</td>
</tr>
<tr>
<td>S1 = 6'b000001</td>
<td>001</td>
</tr>
<tr>
<td>S2 = 6'b000010</td>
<td>010</td>
</tr>
<tr>
<td>S3a = 6'b100011</td>
<td>011</td>
</tr>
<tr>
<td>S3b = 6'b001011</td>
<td>011</td>
</tr>
<tr>
<td>S4aa = 6'b100100</td>
<td>100</td>
</tr>
<tr>
<td>S4ab = 6'b010100</td>
<td>100</td>
</tr>
<tr>
<td>S4bb = 6'b001100</td>
<td>100</td>
</tr>
</tbody>
</table>
Since there are 3 separate state machines in MysteryImplementation (the counter with 8 states and 2 FFs with 2 states each) and the product of the states in each of the three would imply there are 32 states in this implementation. But that would only be true if all combinations of states are possible. That is clearly not the case as the counter goes right back to 0 when it reaches 4, so it only reaches 5 of its 8 states. The FF for A can only change to 1 if the counter is at 2 or 3. Thus, it can only be two different values for the counter’s states 3 and 4. Similarly for the B FF. However, both A and B can’t change in the same cycle as they both rely on In. In the end, we have 5 states for the counter. In counter states 0, 1, and 2, the A and B FFs will both be 0. In counter state 3, the A and B FFs could be 01 or 10. In counter state 4, the A and B FFs could be 01, 10, or 11. That means we have a total of 3+2+3 states for a total of 8, which is the same as our original state diagram.