In this homework, you will be asked to use binary-encoded or one-hot-encoded states. In binary encoding, state \( n \) corresponds to a state where the FFs hold \( n \) in a binary number. For example, in a 4-state machine, the FFs hold 00 in state 0, 01 in state 1, 10 in state 2, and 11 in state 3. On the other hand, in one-hot encoding, state \( n \) is a state where the \( n \)-th FF holds 1 and the other FFs hold 0. Using the same example, the value of FFs is 0001 in state 0, 0010 in state 1, 0100 in state 2, and 1000 in state 3 in one-hot encoding.

**Problem 1: Combinational Lock**

**IMPORTANT:** Use a skeleton at [https://inst.eecs.berkeley.edu/~eecs151/sp23/files/verilog/comb_lock.v](https://inst.eecs.berkeley.edu/~eecs151/sp23/files/verilog/comb_lock.v) for this problem.

Consider the following state transition diagram of a combinational lock.

The following is the additional specification of this combinational lock.

- It receives one 2-bit code at a time.
- The correct code sequence is \( 11 \rightarrow 10 \).
- It outputs two signals: OPEN and ERROR. OPEN is 1 iff it is in OK2, whereas ERROR is 1 iff it is in BAD2.
- It is guaranteed that ENTER and RESET do not take 1 at the same time.
• It uses the following state assignment: \{START = state 0, OK1 = state 1, OK2 = state 2, BAD1 = state 3, BAD2 = state 4\}.

(a) Write a Verilog module with binary-encoded states. \((Hint: \text{Use}\ case)\.)

(b) Write a Verilog module with one-hot-encoded states, without using always blocks. \((Hint: \text{Use continuous assignment})\.)

Solution:

(a) `include "EECS151.v"

```verilog
module comb_lock_binary(  
    input   CLK, ENTER, RESET,  
    input [1:0] CODE,  
    output reg OPEN, ERROR  
);  

localparam START = 3'b000;  
localparam OK1 = 3'b001;  
localparam OK2 = 3'b010;  
localparam BAD1 = 3'b011;  
localparam BAD2 = 3'b100;  
localparam CODE1 = 2'b11;  
localparam CODE2 = 2'b10;  
wire [2:0] cur_state;  
reg [2:0] next_state;  
REGISTER #(.N(3)) r(.q(cur_state), .d(next_state), .clk(CLK));  

always @(*) begin  
OPEN = 0;  
ERROR = 0;  
next_state = cur_state;  
case(cur_state)  
START: begin  
if(ENTER) begin  
if(CODE == CODE1) next_state = OK1;  
else next_state = BAD1;  
end  
end  
OK1: begin  
if(RESET) next_state = START;  
if(ENTER) begin  
if(CODE == CODE2) next_state = OK2;  
else next_state = BAD2;  
end  
end
```
end
OK2: begin
  OPEN = 1;
  if(RESET) next_state = START;
end
BAD1: begin
  if(RESET) next_state = START;
  if(ENTER) next_state = BAD2;
end
BAD2: begin
  ERROR = 1;
  if(RESET) next_state = START;
end
endcase
end

endmodule

(b) module comb_lock_onehot(
  input CLK, ENTER, RESET,
  input [1:0] CODE,
  output OPEN, ERROR
);

wire [4:0] cur_state, next_state;
REGISTER #(.N(5)) r(.q(cur_state), .d(next_state), .clk(CLK));

assign OPEN = cur_state[2];
assign ERROR = cur_state[4];

assign next_state[0] = (cur_state[0] & ~ENTER) | RESET;
assign next_state[1] = (cur_state[0] & ENTER & CODE[1] & CODE[0]) |
  (cur_state[1] & ~ENTER & ~RESET);
  (cur_state[2] & ~RESET);
assign next_state[3] = (cur_state[0] & ENTER & (~CODE[1] | ~CODE[0])) |
  (cur_state[3] & ~ENTER & ~RESET);
  (cur_state[3] & ENTER) |
  (cur_state[4] & ~RESET);
endmodule

Problem 2: STD to Gate-Level Circuits

Consider the following state transition diagram with input $a, b$ and output $o$. Labels of the arrow specify the values of input and output as $ab/o$. 
(No need to write Verilog for this problem.)

(a) Draw a gate-level circuit diagram with binary-encoded states. Use K-maps to simplify the circuit.

(b) Draw a gate-level circuit diagram with one-hot-encoded states.

(Note: You may omit wires as long as they are named. Example is shown below.)

Solution:

(a) Truth Table:
<table>
<thead>
<tr>
<th>$s_1$</th>
<th>$s_0$</th>
<th>$a$</th>
<th>$b$</th>
<th>$n_1$</th>
<th>$n_0$</th>
<th>$o$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<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>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<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>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

K-map for $n_1$:

\[
\begin{array}{cccc}
\text{ab} & 00 & 01 & 11 & 10 \\
\hline
00 & 0 & 0 & 0 & 1 \\
01 & 0 & 1 & 0 & 0 \\
11 & 0 & 1 & 1 & 1 \\
10 & 1 & 1 & 1 & 1 \\
\end{array}
\]

K-map for $n_0$:
K-map for $o$:

<table>
<thead>
<tr>
<th></th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

SOPs:

$$n_1 = s_1 s_0' + s_1 a + s_0' a b' + s_0 a' b$$
$$n_0 = s_1' s_0 + s_0 b + s_1' a' b + s_1 a b'$$
$$o = s_1' a b + s_0' a b'$$

Circuit Diagram:
(b) Boolean Expression:

\[ n_0 = s_0(a \oplus b)' + s_3a'b' \]
\[ n_1 = s_0a'b + s_1(a'b)' \]
\[ n_2 = s_0ab' + s_2(ab')' \]
\[ n_3 = s_1a'b + s_2ab' + s_3(a + b) \]
\[ o = (s_0 + s_1)a'b + (s_0 + s_2)ab' \]

Circuit Diagram:
Problem 3: Alternative Counters

A 2-bit counter is a state machine with 4 states. It transitions from state \( n \) to state \( (n + 1 \mod 4) \) at each positive edge of the clock signal. Assume the states are binary-encoded.

(a) Write a Boolean expression of the input of each FF.

(b) Imagine a situation that we need to count 4 cycles but no need to know which cycle we are in of the 4 cycles. In this situation, we can use an alternative counter, which transitions among 4 states in an arbitrary order. For example, it may transition from state 0 to state 2, state 2 to state 1, state 1 to state 3, and state 3 to state 0. Find a best alternative counter in terms of number of 2-input gates, and write the order of states and the Boolean expressions of FF inputs. You cannot use more than two FFs and they are initialized to 0.

Solution:

(a)

\[
\begin{align*}
n_0 &= s'_0 \\
n_1 &= s_0 \oplus s_1
\end{align*}
\]

(b) Order: state 0 → state 1 → state 3 → state 2.
Boolean Expression:

\[
\begin{align*}
  n_0 &= s_1' \\
  n_1 &= s_0
\end{align*}
\]

**Problem 4: Vending Machine**

You are to design a control module in a vending machine. The items inside the machine cost $15, and the machine accepts five dollar and ten dollar bills only. The control module receives two signals that indicate what kind of bill has been deposited.

The control module has three output signals. One signal causes the item to be delivered, while the other two signals cause a bill to be dispensed.

The vending machine has another module that counts the number of five dollar bills inside. The control module receives a signal from it that indicates whether it is out of change or not.

Whenever the amount of money received becomes $15, the machine delivers the item and resets to its initial state. When it has received $20, it delivers the item with a change, or returns two ten dollar bills if it is out of change. After that, it goes back to the initial state.

Identify your inputs and outputs, and draw a state transition diagram that implements the control module. (No need to write Verilog.)

Assume that only one bill can deposited or dispensed at a time. No bills can be deposited when delivering the item.

**Solution:**

**Inputs:**

- FIVE (when a five dollar bill is deposited)
- TEN (when a ten dollar bill is deposited)
- OUT (when the machine is out of change)

**Outputs:**

- ITEM (deliver the item)
- RETURN5 (dispense a five dollar bill)
- RETURN10 (dispense a ten dollar bill)