1. [25 points.]

Consider the Moore Finite State Machine in Figure 1. Recall, circles represent states and edges represent transitions from one state to another. The $x$ on an edge from state $S_i$ to state $S_{i+1}$ indicates the value of the input must be $x$ for that transition to occur. Outputs are shown in the bottom half of each state circle.

(a) (10 points) Write a Verilog module (behavioral) to implement the FSM shown in Figure 1. Assume a synchronous reset signal such that when reset=1, the FSM should return to state “S0”
at the beginning of the next clock cycle. Use only synthesizable Verilog and be certain to avoid any unnecessary latches.
(Note: Remember the guidelines from lecture about separating combinatorial logic from sequential logic in behavioral verilog. Failure to do so will result in lost points)
(b) (10 points) Write a testbench to test the FSM. Ensure that all the states are visited at least once.
(c) (5 points) Sketch the clock, input, states and output waveforms for reset = “0”, starting state = “S0” and input is the procession of values 1,1,1,0,1. The input changes just before the next positive edge of the clock.

2. [25 points.]

Consider the following Verilog code:

```verilog
module MyModule(out, a, b, c);
    output out;
    input a;
    input b;
    input c;
    reg out;
    wire temp1;
    reg temp2;
    always @ (a or b or c or temp1) begin
        if (c == 1) begin
            out = temp1;
            temp2 = a;
        end
        else
            out = b;
    end
    assign #2 temp1 = temp3;
endmodule
```

(a) (5 points) Is this code syntactically correct? If not list any errors you see.
(b) (5 points) What parts of this code cannot be synthesized?
(c) (10 points) Draw a block diagram of the hardware this code would produce if synthesized. Assume any non-synthesizable parts are ignored.
(d) (5 points) Write an improved version of this code given the assumption the designer intended to produce solely combinatorial logic.