Problem 1: [12 points] Drill problem
Filename: library.sv

```
`default_nettype none

module MagComp
  # (parameter WIDTH = 1)
  (input logic [WIDTH-1:0] A, B,
   output logic AltB, AeqB, AgtB);

  always_comb begin
    if (A < B)
      begin
        AeqB = 1'b0;
        AltB = 1'b1;
        AgtB = 1'b0;
      end
    else if (A == B)
      begin
        AeqB = 1'b1;
        AltB = 1'b0;
        AgtB = 1'b0;
      end
    else
      begin
        AgtB = 1'b1;
        AltB = 1'b0;
        AeqB = 1'b0;
      end
  end

endmodule: MagComp


module Multiplexer
  # (parameter WIDTH = 2)
  // parameter S_WIDTH = $clog2(WIDTH);
  (input logic [WIDTH-1:0] I,
   input logic [$clog2(WIDTH)-1:0] S,
   output logic Y);

  always_comb begin
    if (S < WIDTH)
      Y = I[S];
  end

endmodule: Multiplexer

module Mux2to1
  // input width
  # (parameter WIDTH = 1)
  (input logic [WIDTH-1:0] I0, I1,
   input logic S,
   output logic [WIDTH-1:0] Y);

  assign Y = S ? I1 : I0;

endmodule: Mux2to1

module Decoder
    # (parameter WIDTH = 1)
    // parameter I_WIDTH = $clog2(WIDTH);
    (input logic [$clog2(WIDTH)-1:0] I,
     input logic en,
     output logic [WIDTH-1:0] D);

    always_comb begin
      D = 0;
      if (en)
        begin
          if (I < WIDTH)
            D[I] = 1'b1;
        end
      else
        D = I;
    end

endmodule: Decoder

module Adder
  # (parameter WIDTH = 1)
  (input logic [WIDTH-1:0] A, B,
   input logic Cin,
   output logic [WIDTH-1:0] S,
   output logic Cout);

  assign {Cout, S} = A + B + Cin;

endmodule: Adder

module Register
  # (parameter WIDTH = 1)
  (input logic [WIDTH-1:0] D,
   input logic en, clear,
   input logic clock,
   output logic [WIDTH-1:0] Q);

   always_ff @(posedge clock)
     if (en)
       Q <= clear ? 0 : D;

endmodule: Register
```

Problem 1: [12 points] Drill problem
Filename: library_tests.sv

```
module MagComp_test ();
  logic [4:0] A1, B1;
  logic AltB, AgtB, AeqB;

  MagComp #(5) five (.A(A1), .B(B1), .AltB, .AgtB, .AeqB);

  initial begin
    $monitor($time,, "A1=%d, B1=%d, AltB=%b, AgtB=%b, AeqB=%b",
            A1, B1, AltB, AgtB, AeqB);
    #5 A1 = 5'b10000;
       B1 = 5'b00010;
    #5 A1 = 5'b00001;
       B1 = 5'b00001;
    #5 B1 = 5'b01111;
    #1 $finish;
  end
endmodule: MagComp_test

module Multiplexer_test ();
  logic [6:0] I;
  logic Y;
  logic [2:0] S;

  Multiplexer #(7) m1 (.*);

  initial begin
    $monitor($time,, "I=%b, Y=%b, S=%d",
            I, Y, S);
    #5 I = 7'b0000000;
       S = 3'b000;
    #5 S = 3'b101;
    #5 I = 7'b0100000;
    #5 S = 3'b111;
    #5 I = 7'b1111000;
       S = 3'b001;
    #5 S = 3'b100;
    #1 $finish;
  end

endmodule: Multiplexer_test

module Mux2to1_test ();
  logic [6:0] I0, I1, Y;
  logic S;

  Mux2to1 #(7) m1 (.*);

  initial begin
    $monitor($time,, "I0=%b, I1=%b, Y=%b, S=%b",
            I0, I1, Y, S);
    #5 I0 = 7'b0000000;
       I1 = 7'b1111111;
       S = 1'b1;
    #5 S = 1'b0;
    #1 $finish;
  end
endmodule: Mux2to1_test

module Decoder_test ();
  // `define WIDTH1 3
  // `define I_WIDTH1 2

  logic [2:0] D;
  logic [1:0] I;
  logic en;

  Decoder #(3) dec (.*);

  initial begin
    $monitor($time,, "I=%d, en=%b, D=%b", I, en, D);
    #5 I = 2'b11;
       en = 0;
    #5 en = 1;
    #5 I = 2'b01;
    #1 $finish;
  end
endmodule: Decoder_test

module Adder_test ();
  // `define 8 8
  logic Cin, Cout;
  logic [7:0] A, B, S;

  Adder #(8) add (.*);

  initial begin
    $monitor($time,, "A=%d, B=%d, Cin=%b, Cout=%b, S=%d", A, B, Cin, Cout, S);
    #5 A = 27;
       B = 0;
       Cin = 1'b0;
    #5 A = 38;
       B = 200;
    #5 Cin = 1'b1;
    #5 A = 129;
       B = 129;
    #1 $finish;
  end
endmodule: Adder_test


module Register_test ();
  //`define 3 3
  logic [2:0] D, Q;
  logic en, clear;
  logic clock, reset_L;

  Register #(3) regis (.*);

  initial begin
    $monitor($time,, "Q=%d, D=%d, en=%d, clear=%b",
             Q, D, en, clear);
    clock = 0;
    forever #5 clock = ~clock;
  end

  initial begin
    D = 3'b000;
    en = 0;
    clear = 0;
    reset_L <= 0;
    @(posedge clock);
    reset_L <= 1;
    @(posedge clock);
    D = 3'b100;
  // reset is synchronous, so must wait for a clock edge
    @(posedge clock);
    en = 1;
    @(posedge clock);
    D = 3'b010;
    @(posedge clock);
    D = 3'b011;
    clear = 1;
    @(posedge clock);
    en = 0;
    @(posedge clock);
    clear = 0;
    @(posedge clock);
    @(posedge clock);
    D = 3'b111;
    en = 1;
    @(posedge clock);
    D = 3'b101;
    en = 0;
    @(posedge clock);
    #1 $finish;
  end

endmodule: Register_test
```

Problem 5: [20 points]
Filename: hw7prob5.sv

```
`default_nettype none

module fsm
  (input logic clock, reset_L, start, inputB, inputC,
   output logic en, done, clear);

  enum logic [3:0] {init=4'b1000, checkC=4'b0100,
                    checkB=4'b0010, stop=4'b0000} Q, D;

  always_comb
    unique case (Q)
      init: D <= start ? checkC : init;
      checkC: D <= inputC ? checkB : stop;
      checkB: D <= checkC;
      stop: D <= stop;
    endcase

  always_ff @(posedge clock, negedge reset_L)
    if (~reset_L) Q <= init;
    else Q <= D;


  assign clear = (Q == init) ? 1 : 0;
  assign en = (((Q == checkB) && inputB) || (Q == init)) ? 1 : 0;
  assign done = (Q == stop) ? 1 : 0;
endmodule: fsm

module prob5FSMD
  // #(parameter W = 8)
  (input logic start, inputC, inputB,
   input logic clock, reset_L,
   input logic [7:0] inputA,
   output logic done,
   output logic [7:0] value);

  logic en, dontUse, clear;
  logic [7:0] addOut;

  fsm dut (.*);
  Adder #(8) a1 (.A(inputA), .B(value), .Cin(1'b0),
                 .S(addOut), .Cout(dontUse));
  Register #(8) r1 (.D(addOut), .en, .clear,
                    .clock, .Q(value));
endmodule: prob5FSMD
```

Problem 6: [12 points]
Filename: hw7prob6.sv

```
`default_nettype none
module fsm
  (input logic clock, reset_L,
   output logic firstInput, clear);

  enum logic [1:0] {init=2'b00, round1=2'b01, next=2'b10} D, Q;

  always_comb
    unique case(Q)
      init: D <= round1;
      round1: D <= next;
      next: D <= next;
      default: D <= init;
    endcase


  always_ff @(posedge clock, negedge reset_L)
    if (~reset_L) Q <= init;
    else Q <= D;

  always_comb begin
    unique case (Q)
      next: begin
              firstInput = 0;
              clear = 0;
            end
      init: begin
              firstInput = 1;
              clear = 1;
            end
      round1: begin
               firstInput = 1;
               clear = 0;
              end
    endcase
  end
endmodule: fsm


module serialTwosComp
  (input logic A, clock, reset_L,
   output logic B);

  logic Cin, Cout, en, clear, firstInput, flippedA;
  // logic [1:0] nextState;
  assign flippedA = ~A;
  Adder #(1) a1 (.A(flippedA), .B(firstInput), .Cin, .Cout, .S(B));
  Register #(1) coutReg (.D(Cout), .en(1'b1), .clear,
                         .clock, .Q(Cin));
  fsm dut (.*);

endmodule: serialTwosComp

module serialTwosComp_test ();
  // number is 10111011
  logic A, B, Cin, Cout;
  logic reset_L, clock;

  serialTwosComp stc (.*);
  initial begin
    $monitor($time,, "A=%b, B=%b, Cin=%b, Cout=%b, \
                      firstInput=%b, reset=%b, s=%s",
                      A, B, stc.Cin, stc.Cout,
                      stc.firstInput, reset_L, stc.dut.Q.name);
    clock = 0;
    forever #5 clock = ~clock;
  end

  initial begin
    reset_L = 0;
    A = 1;
    Cin = 0;
    Cout = 0;
    @(posedge clock);
    reset_L = 1;
    @(posedge clock);
    A = 1;
    @(posedge clock);
    @(posedge clock);
    A = 0;
    @(posedge clock);
    A = 1;
    @(posedge clock);
    @(posedge clock);
    @(posedge clock);
    A = 0;
    @(posedge clock);
    A = 1;
    @(posedge clock);
    reset_L = 0;
    @(posedge clock);
    A = 0;
    reset_L = 1;
    @(posedge clock);
    @(posedge clock);
    @(posedge clock);
    @(posedge clock);
    @(posedge clock);
    @(posedge clock);
    @(posedge clock);
    @(posedge clock);
    #1 $finish;
  end

endmodule: serialTwosComp_test
```