Problem 1: [4 points] Drill problem
Filename: library.sv
AndrewID: xinyew

```
`default_nettype none

module BusDriver
 #(parameter WIDTH = 8)
  (input  logic en,
   input  logic [WIDTH-1:0] data,
   output logic [WIDTH-1:0] buff,
   inout  tri   [WIDTH-1:0] bus
   );

   assign bus = (en) ? data : 'bz;
   assign buff = bus;

endmodule : BusDriver

module Memory
 #(parameter DW = 16,
             W  = 256,
             AW = $clog2(W))
  (input  logic re, we, clock,
   input  logic [AW-1:0] addr,
   inout  tri   [DW-1:0] data);

  logic [DW-1:0] M[W];
  logic [DW-1:0] rData;

  assign data = (re) ? rData : 'bz;

  always_ff @(posedge clock)
    if (we)
      M[addr] <= data;

  always_comb
    rData = M[addr];

endmodule : Memory


`default_nettype none

// Magnitude comparator
module MagComp
 #(parameter WIDTH = 16)
  (input  logic [WIDTH-1:0] A, B,
   output logic AltB, AeqB, AgtB);

  assign AltB = A < B;
  assign AeqB = A == B;
  assign AgtB = A > B;

endmodule : MagComp

// Adder
module Adder
 #(parameter WIDTH = 16)
  (input  logic cin,
   input  logic [WIDTH-1:0] A, B,
   output logic [WIDTH-1:0] S,
   output logic cout);

  assign {cout, S} = A + B + cin;

endmodule : Adder

// Mux
module Multiplexer
 #(parameter WIDTH = 16)
  (input  logic [WIDTH-1:0] I,
   input  logic [$clog2(WIDTH)-1:0] S,
    output logic Y);

  assign Y = I[S];

endmodule : Multiplexer

// Mux2to1
module Mux2to1
 #(parameter WIDTH = 16)
  (input  logic [WIDTH-1:0] I0, I1,
   input  logic S,
   output logic [WIDTH-1:0] Y);

   assign Y = S ? I1 : I0;

endmodule : Mux2to1

// Decoder
module Decoder
 #(parameter WIDTH = 16)
  (input  logic [$clog2(WIDTH)-1:0] I,
   input  logic en,
   output logic [WIDTH-1:0] D);

  assign D = en ? 1'b1 << I : 1'b0;

endmodule : Decoder

// DFlipFlop
module DFlipFlop
  (input  logic D,
   input  logic clock, preset_L, reset_L,
   output logic Q);

  always_ff @(posedge clock, negedge reset_L, negedge preset_L)
    // when reset_L and preset_L asserted at the same time, output x
    if (~reset_L && ~preset_L)
      Q <= 1'bx;
    else if (~reset_L)
      Q <= 1'b0;
    else if (~preset_L)
      Q <= 1'b1;
    else
      Q <= D;

endmodule : DFlipFlop

// Register
module Register
 #(parameter WIDTH = 16)
  (input  logic en, clear, clock,
   input  logic [WIDTH-1:0] D,
   output logic [WIDTH-1:0] Q);

  always_ff @(posedge clock)
    if (en)
      Q <= D;
    else if (clear)
      Q <= 1'b0;

endmodule : Register

// Counter
module Counter
 #(parameter WIDTH = 16)
  (input  logic en, clear, load, up, clock,
   input  logic [WIDTH-1:0] D,
   output logic [WIDTH-1:0] Q);

  always_ff @(posedge clock)
    if (clear)
```

```
      Q <= 1'b0;
    else if (load)
      Q <= D;
    else if (en) begin
      if (up)
        Q <= Q + 1;
      else
        Q <= Q - 1;
    end

endmodule : Counter

// Sync
module Synchronizer
  (input  logic async, clock,
   output logic sync);

  always_ff @(posedge clock)
    sync <= async;

endmodule : Synchronizer

// ShiftRegister_SIPO
module ShiftRegister_SIPO
 #(parameter WIDTH = 16)
  (input  logic serial, en, left, clock,
   output logic [WIDTH-1:0] Q);

  always_ff @(posedge clock)
    if (en) begin
      if (left)
        Q <= {Q[WIDTH-2:0], serial};
      else
        Q <= {serial, Q[WIDTH-1:1]};
    end

endmodule : ShiftRegister_SIPO

// ShiftRegister_PIPO
module ShiftRegister_PIPO
 #(parameter WIDTH = 16)
  (input  logic en, left, load, clock,
   input  logic [WIDTH-1:0] D,
   output logic [WIDTH-1:0] Q);

  always_ff @(posedge clock)
    if (load)
      Q <= D;
    else if (en) begin
      if (left)
        Q <= Q << 1;
      else
        Q <= Q >> 1;
    end

endmodule : ShiftRegister_PIPO

// BarrelShiftRegister
module BarrelShiftRegister
 #(parameter WIDTH = 16)
  (input  logic en, load, clock,
   input  logic [1:0] by,
   input  logic [WIDTH-1:0] D,
   output logic [WIDTH-1:0] Q);

  always_ff @(posedge clock)
    if (load)
      Q <= D;
    else if (en) begin
      Q <= Q << by;
    end
```

```

endmodule : BarrelShiftRegister
```

Problem 1: [4 points] Drill problem
Filename: library_tests.sv
AndrewID: xinyew

```
`default_nettype none

module Memory_test();
  logic [7:0] data;
  tri [7:0] bus;
  logic [1:0] addr;
  logic re, we, clock;
  BusDriver #(8) BUS (.en(we),
                      .data(data),
                      .bus(bus));

  Memory #(8,4,2) DUT (.addr(addr),
                       .re(re),
                       .we(we),
                       .clock(clock),
                       .data(bus));

  initial begin
    clock = 'b0;
    forever #10 clock = ~clock;
  end

  initial begin
    $monitor($time,, "[%s]     BUS: %d | addr: %b  data: %d",
                      (we) ? "WRITE" : "READ", bus, addr, data);
  end

  initial begin
    addr <= 2'b00;
    data <= 8'd240;
    re <= 1'b1;
    we <= 1'b0;
    @(posedge clock);
    re <= 1'b0;
    we <= 1'b1;
    @(posedge clock);
    re <= 1'b1;
    we <= 1'b0;
    @(posedge clock);
    addr <= 2'b01;
    data <= 8'd220;
    @(posedge clock);
    re <= 1'b0;
    we <= 1'b1;
    @(posedge clock);
    re <= 1'b1;
    we <= 1'b0;
    addr <= 2'b00;
    @(posedge clock);
    re <= 1'b0;
    we <= 1'b1;
    addr <= 2'b10;
    data <= 8'd250;
    @(posedge clock);
    re <= 1'b1;
    we <= 1'b0;
    @(posedge clock);
    re <= 1'b0;
    we <= 1'b1;
    addr <= 2'b11;
    data <= 8'd213;
    @(posedge clock);
    re <= 1'b1;
    we <= 1'b0;
    @(posedge clock);
    addr <= 2'b00;
    @(posedge clock);
    addr <= 2'b01;
    @(posedge clock);
    addr <= 2'b10;
    @(posedge clock);
    addr <= 2'b11;
    @(posedge clock);
    #1 $finish;
  end

endmodule : Memory_test
```

Problem 4: [20 points]
Filename: hw7prob4.sv
AndrewID: xinyew

```
`default_nettype none

module MemoryController
 #(parameter logic [7:0] page = 8'h02)
  (inout tri [15:0] addressData,
   input logic addressValid, read, clock, reset);

  // indicating controller to drive the bus
  logic send;
  // word to be sent
  logic [15:0] toSend;
  // word received
  logic [15:0] received;

  // init the bus driver
  BusDriver #(16) BUS (.en(send),
                       .data(toSend),
                       .bus(addressData),
                       .buff(received));

  // current address
  logic [7:0] addr;

  // init memory
  Memory #(.AW(8), .DW(16)) M(.addr(addr),
                              .re(~send),
                              .we(send),
                              .clock(clock),
                              .data(addressData));

  FSM fsm (.*);

endmodule : MemoryController

module FSM
  (input logic [15:0] addressData,
   input logic [7:0] page,
   input logic read, addressValid,
   output logic [7:0] addr,
   output logic send);
  enum logic [3:0] {IDLE, READ1, READ2,
                    READ3, READ4, WRITE1, WRITE2, WRITE3,
                    WRITE4, DONE} cur_state, n_state;

  always_comb begin
    case (cur_state)
      IDLE: begin
        if (addressValid && (addressData[15:8] == page)) begin
          addr = addressData[7:0];
          if (read)
            n_state = READ1;
          else
            n_state = WRITE1;
        end
        else
          n_state = IDLE;
      end
      READ1: begin
        n_state = READ2;
        send = 'b1;
        addr = addr + 1;
      end
      READ2: begin
        n_state = READ2;
        addr = addr + 1;
      end
      READ3: begin
        n_state = READ2;
        addr = addr + 1;
      end
      READ4: begin
        n_state = DONE;
      end
      DONE:
        send = 'b0;
      WRITE1: begin
        n_state = WRITE2;
        addr = addr + 1;
      end
      WRITE2: begin
        n_state = WRITE3;
        addr = addr + 1;
      end
      WRITE3: begin
        n_state = WRITE4;
        addr = addr + 1;
      end
      WRITE4: begin
        n_state = DONE;
      end
    endcase
  end

endmodule : FSM
```