

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

```
1 `default_nettype none
2
3 module BusDriver
4   #(parameter WIDTH = 8)
5   (input logic en,
6    input logic [WIDTH-1:0] data,
7    output logic [WIDTH-1:0] buff,
8    inout tri [WIDTH-1:0] bus
9   );
10
11   assign bus = (en) ? data : 'bz;
12   assign buff = bus;
13
14 endmodule : BusDriver
15
16 module Memory
17   #(parameter DW = 16,
18           W = 256,
19           AW = $clog2(W))
20   (input logic re, we, clock,
21    input logic [AW-1:0] addr,
22    inout tri [DW-1:0] data);
23
24   logic [DW-1:0] M[W];
25   logic [DW-1:0] rData;
26
27   assign data = (re) ? rData : 'bz;
28
29   always_ff @(posedge clock)
30     if (we)
31       M[addr] <= data;
32
33   always_comb
34     rData = M[addr];
35
36 endmodule : Memory
37
38 `default_nettype none
39
40 // Magnitude comparator
41 module MagComp
42   #(parameter WIDTH = 16)
43   (input logic [WIDTH-1:0] A, B,
44    output logic AltB, AeqB, AgtB);
45
46   assign AltB = A < B;
47   assign AeqB = A == B;
48   assign AgtB = A > B;
49
50 endmodule : MagComp
51
52 // Adder
53 module Adder
54   #(parameter WIDTH = 16)
55   (input logic cin,
56    input logic [WIDTH-1:0] A, B,
57    output logic [WIDTH-1:0] S,
58    output logic cout);
59
60   assign {cout, S} = A + B + cin;
61
62 endmodule : Adder
63
64 // Mux
65 module Multiplexer
66   #(parameter WIDTH = 16)
67   (input logic [WIDTH-1:0] I,
68    input logic [$clog2(WIDTH)-1:0] S,
```

```
70     output logic Y);
71
72     assign Y = I[S];
73
74 endmodule : Multiplexer
75
76 // Mux2to1
77 module Mux2to1
78     #(parameter WIDTH = 16)
79     (input logic [WIDTH-1:0] I0, I1,
80      input logic S,
81      output logic [WIDTH-1:0] Y);
82
83     assign Y = S ? I1 : I0;
84
85 endmodule : Mux2to1
86
87 // Decoder
88 module Decoder
89     #(parameter WIDTH = 16)
90     (input logic [$clog2(WIDTH)-1:0] I,
91      input logic en,
92      output logic [WIDTH-1:0] D);
93
94     assign D = en ? 1'b1 << I : 1'b0;
95
96 endmodule : Decoder
97
98 // DFlipFlop
99 module DFlipFlop
100     (input logic D,
101      input logic clock, preset_L, reset_L,
102      output logic Q);
103
104     always_ff @(posedge clock, negedge reset_L, negedge preset_L)
105         // when reset_L and preset_L asserted at the same time, output x
106         if (~reset_L && ~preset_L)
107             Q <= 1'bx;
108         else if (~reset_L)
109             Q <= 1'b0;
110         else if (~preset_L)
111             Q <= 1'b1;
112         else
113             Q <= D;
114
115 endmodule : DFlipFlop
116
117 // Register
118 module Register
119     #(parameter WIDTH = 16)
120     (input logic en, clear, clock,
121      input logic [WIDTH-1:0] D,
122      output logic [WIDTH-1:0] Q);
123
124     always_ff @(posedge clock)
125         if (en)
126             Q <= D;
127         else if (clear)
128             Q <= 1'b0;
129
130 endmodule : Register
131
132 // Counter
133 module Counter
134     #(parameter WIDTH = 16)
135     (input logic en, clear, load, up, clock,
136      input logic [WIDTH-1:0] D,
137      output logic [WIDTH-1:0] Q);
138
139     always_ff @(posedge clock)
140         if (clear)
```

```
141     Q <= 1'b0;
142     else if (load)
143         Q <= D;
144     else if (en) begin
145         if (up)
146             Q <= Q + 1;
147         else
148             Q <= Q - 1;
149     end
150
151 endmodule : Counter
152
153 // Sync
154 module Synchronizer
155     (input logic async, clock,
156      output logic sync);
157
158     always_ff @(posedge clock)
159         sync <= async;
160
161 endmodule : Synchronizer
162
163 // ShiftRegister_SIPO
164 module ShiftRegister_SIPO
165     #(parameter WIDTH = 16)
166     (input logic serial, en, left, clock,
167      output logic [WIDTH-1:0] Q);
168
169     always_ff @(posedge clock)
170         if (en) begin
171             if (left)
172                 Q <= {Q[WIDTH-2:0], serial};
173             else
174                 Q <= {serial, Q[WIDTH-1:1]};
175         end
176
177 endmodule : ShiftRegister_SIPO
178
179 // ShiftRegister_PIP0
180 module ShiftRegister_PIP0
181     #(parameter WIDTH = 16)
182     (input logic en, left, load, clock,
183      input logic [WIDTH-1:0] D,
184      output logic [WIDTH-1:0] Q);
185
186     always_ff @(posedge clock)
187         if (load)
188             Q <= D;
189         else if (en) begin
190             if (left)
191                 Q <= Q << 1;
192             else
193                 Q <= Q >> 1;
194         end
195
196 endmodule : ShiftRegister_PIP0
197
198 // BarrelShiftRegister
199 module BarrelShiftRegister
200     #(parameter WIDTH = 16)
201     (input logic en, load, clock,
202      input logic [1:0] by,
203      input logic [WIDTH-1:0] D,
204      output logic [WIDTH-1:0] Q);
205
206     always_ff @(posedge clock)
207         if (load)
208             Q <= D;
209         else if (en) begin
210             Q <= Q << by;
211         end
```

212

213 **endmodule** : BarrelShiftRegister

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

```
1 `default_nettype none
2
3 module Memory_test();
4     logic [7:0] data;
5     tri [7:0] bus;
6     logic [1:0] addr;
7     logic re, we, clock;
8     BusDriver #(8) BUS (.en(we),
9                          .data(data),
10                         .bus(bus));
11
12     Memory #(8,4,2) DUT (.addr(addr),
13                         .re(re),
14                         .we(we),
15                         .clock(clock),
16                         .data(bus));
17
18     initial begin
19         clock = 'b0;
20         forever #10 clock = ~clock;
21     end
22
23     initial begin
24         $monitor($time,, "[%s]      BUS: %d | addr: %b data: %d",
25                  (we) ? "WRITE" : "READ", bus, addr, data);
26     end
27
28     initial begin
29         addr <= 2'b00;
30         data <= 8'd240;
31         re <= 1'b1;
32         we <= 1'b0;
33         @(posedge clock);
34         re <= 1'b0;
35         we <= 1'b1;
36         @(posedge clock);
37         re <= 1'b1;
38         we <= 1'b0;
39         @(posedge clock);
40         addr <= 2'b01;
41         data <= 8'd220;
42         @(posedge clock);
43         re <= 1'b0;
44         we <= 1'b1;
45         @(posedge clock);
46         re <= 1'b1;
47         we <= 1'b0;
48         addr <= 2'b00;
49         @(posedge clock);
50         re <= 1'b0;
51         we <= 1'b1;
52         addr <= 2'b10;
53         data <= 8'd250;
54         @(posedge clock);
55         re <= 1'b1;
56         we <= 1'b0;
57         @(posedge clock);
58         re <= 1'b0;
59         we <= 1'b1;
60         addr <= 2'b11;
61         data <= 8'd213;
62         @(posedge clock);
63         re <= 1'b1;
64         we <= 1'b0;
65         @(posedge clock);
66         addr <= 2'b00;
67         @(posedge clock);
68         addr <= 2'b01;
69         @(posedge clock);
```

```
70     addr <= 2'b10;
71     @(posedge clock);
72     addr <= 2'b11;
73     @(posedge clock);
74     #1 $finish;
75 end
76
77 endmodule : Memory_test
```

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

```
1 `default_nettype none
2
3 module MemoryController
4   #(parameter logic [7:0] page = 8'h02)
5   (inout tri [15:0] addressData,
6    input logic addressValid, read, clock, reset);
7
8   // indicating controller to drive the bus
9   logic send;
10  // word to be sent
11  logic [15:0] toSend;
12  // word received
13  logic [15:0] received;
14
15  // init the bus driver
16  BusDriver #(16) BUS (.en(send),
17                      .data(toSend),
18                      .bus(addressData),
19                      .buff(received));
20
21  // current address
22  logic [7:0] addr;
23
24  // init memory
25  Memory #(.AW(8), .DW(16)) M(.addr(addr),
26                              .re(~send),
27                              .we(send),
28                              .clock(clock),
29                              .data(addressData));
30
31  FSM fsm (.*);
32
33 endmodule : MemoryController
34
35 module FSM
36   (input logic [15:0] addressData,
37    input logic [7:0] page,
38    input logic read, addressValid,
39    output logic [7:0] addr,
40    output logic send);
41   enum logic [3:0] {IDLE, READ1, READ2,
42                   READ3, READ4, WRITE1, WRITE2, WRITE3,
43                   WRITE4, DONE} cur_state, n_state;
44
45   always_comb begin
46     case (cur_state)
47       IDLE: begin
48         if (addressValid && (addressData[15:8] == page)) begin
49           addr = addressData[7:0];
50           if (read)
51             n_state = READ1;
52           else
53             n_state = WRITE1;
54         end
55       else
56         n_state = IDLE;
57     end
58     READ1: begin
59       n_state = READ2;
60       send = 'b1;
61       addr = addr + 1;
62     end
63     READ2: begin
64       n_state = READ2;
65       addr = addr + 1;
66     end
67     READ3: begin
68       n_state = READ2;
69       addr = addr + 1;
```

```
70     end
71     READ4: begin
72         n_state = DONE;
73     end
74     DONE:
75         send = 'b0;
76     WRITE1: begin
77         n_state = WRITE2;
78         addr = addr + 1;
79     end
80     WRITE2: begin
81         n_state = WRITE3;
82         addr = addr + 1;
83     end
84     WRITE3: begin
85         n_state = WRITE4;
86         addr = addr + 1;
87     end
88     WRITE4: begin
89         n_state = DONE;
90     end
91 endcase
92 end
93
94 endmodule : FSM
```