```
 1  module top_tb();
 2     reg clk = 1'b0;
 3
 4     wire [1:0] pwm;
 5     wire [11:0] vec;
 6
 7
 8     top dut(.clk(clk), .pwm(pwm), .vec(vec));
 9
10     always
11       #5 clk = ~clk;
12
13     initial begin
14        $dumpfile("dump.vcd");
15        $dumpvars();
16        #10000000 $finish();
17     end
18
19  endmodule
20
21  module top
22     (input         clk,
23      output [12:1] vec,
24      output [2:1]  pwm);
25
26     wire [4:1] pwm_tab [1:2];
27
28     sequencer seq(.clk(clk), .vec(vec), .pwm(pwm_tab));
29     pwm pwm_gen(.clk(clk), .pwm_in(pwm_tab), .pwm(pwm));
30
31  endmodule
32
33  module sequencer
34    (input         clk,
35     output [12:1] vec,
36     output [4:1]  pwm [1:2]);
37
38     parameter OP_WAIT = 1'b0;
39     parameter OP_JUMP = 1'b1;
40
41     reg fetch = 1'b1;
42
43     wire opcode;
44     wire [9:1] addr_or_cycles;
45     wire [9:1] daddr;
46     wire [2:1] UNUSED;
47     wire [32:1] dout;
48     wire [9:1]  next_addr = (opcode == OP_JUMP ? addr_or_cycles : (daddr + 1)
);
49     rom ucode(.clk(clk), .en(fetch), .addr(next_addr), .daddr(daddr), .dout(d
out));
50     assign {pwm[2], pwm[1], vec, UNUSED, opcode, addr_or_cycles} = dout;
51
52     wire cout;
53     prescaler prs(.clk(clk), .cout(cout), .reset(fetch));
54     wire finished;
55     counter ctr(.clk(clk), .en(cout), .threshold(addr_or_cycles), .reset(fetc
h), .finished(finished));
56
57     always @(negedge clk) begin
58       case (opcode)
59         OP_WAIT : begin
60            if (finished)
61              fetch <= 1'b1;
62            else if (fetch)
63              fetch <= 1'b0;
64         end
65         OP_JUMP : begin
66            fetch = ~fetch;
67         end
68       endcase
69     end
70
```

```
 71  endmodule
 72
 73  module rom
 74    #(parameter FILE = "memory.hex",
 75      parameter WIDTH = 32,
 76      parameter DEPTH = 9)
 77    (input                clk,
 78     input                en,
 79     input [DEPTH:1]      addr,
 80     output reg [DEPTH:1] daddr,
 81     output reg [WIDTH:1] dout);
 82
 83     reg [WIDTH:1] mem [0:((1<<DEPTH)-1)];
 84
 85     initial
 86       $readmemh(FILE, mem);
 87
 88     always @(posedge clk) begin
 89        if (en) begin
 90           daddr <= addr;
 91           dout <= mem[addr];
 92        end
 93
 94     end
 95
 96  endmodule
 97
 98  module counter
 99    #(parameter WIDTH = 9)
100     (input           clk,
101      input           en,
102      input [WIDTH:1] threshold,
103      input           reset,
104      output reg      finished);
105
106     reg [WIDTH:1] ctr;
107
108     always @(posedge clk)
109       if (reset) begin
110         {ctr, finished} <= {{WIDTH{1'b0}}, 1'b0};
111       end else if (en) begin
112          if (ctr >= threshold)
113            finished <= 1'b1;
114          else
115            ctr <= ctr + 1;
116       end
117
118  endmodule
119
120  module prescaler
121    #(parameter WIDTH = 10)
122     (input  clk,
123      input  reset,
124      output reg cout);
125
126     reg [WIDTH:1] ctr = 0;
127
128     always @(posedge clk)
129       if (!reset)
130         {cout, ctr} <= ctr + 1;
131       else
132         {cout, ctr} <= {1'b0, {WIDTH{1'b0}}};
133
134  endmodule
135
136  module pwm
137    #(parameter PWM_WIDTH = 4)
138     (input        clk,
139      input [PWM_WIDTH:1]  pwm_in [1:2],
140      output [2:1] pwm);
141
142     wire en;
143
```

```
144     reg [PWM_WIDTH:1] cnt;
145
146     prescaler #(.WIDTH(2)) div(.clk(clk), .reset(1'b0), .cout(en));
147
148     always @(posedge clk) begin
149        if (en)
150          cnt <= cnt + 1;
151     end
152
153     wire [PWM_WIDTH:1] cnt_180 = cnt + {(PWM_WIDTH-1){1'b1}};
154
155     assign pwm[1] = pwm_in[1] > cnt;
156     assign pwm[2] = pwm_in[2] > cnt_180;
157  endmodule
158
```