module foo (clk, reset, in, out);
  input logic clk, reset, in;
  output logic out;
  enum { A=0, B=1 } ps, ns;

  always_comb begin
    case (ps)
      A: ns = B;
      B: if (in) ns = B;
      else ns = A;
    endcase
    out = ~ps;
  end

  always_ff @(posedge clk) begin
    if (reset) ps <= A;
    else ps <= ns;
  end
endmodule
module ... ...
input/output/logic ...

initial ...
always_comb ...
always_ff ...
assign ...

cendmodule
FSM Testbench

module simple_testbench();
logic clk, reset, w, out;

simple dut (.clk, .reset, .w, .out);

// Set up the clock.
parameter CLOCK_PERIOD=100;

initial begin
  clk <= 0;
  forever #(CLOCK_PERIOD/2) clk <= ~clk;
end
FSM Testbench (cont.)

// Design inputs. Each line is a clock cycle.

// ONLY USE THIS FORM for testbenches!!!
initial begin

reset <= 1; @(posedge clk); ^1
reset <= 0; w <= 0; @(posedge clk); ^2
@(posedge clk); ^3
@(posedge clk); ^4
@(posedge clk); ^5
@(posedge clk); ^6
w <= 1; @(posedge clk); ^7
w <= 0; @(posedge clk); ^8
w <= 1; @(posedge clk); ^9
@(posedge clk); ^10
@(posedge clk); ^11
@(posedge clk); ^12
w <= 0; @(posedge clk); ^13
@(posedge clk); ^14
$stop; // End the simulation.

end

endmodule
Testbench Waveforms

clk
reset
w
ps[1:0]
out

0/0 1/0
A B C
1/1

Reset