// Dual-Port RAM Module

module dual\_port\_ram #(parameter ADDR\_WIDTH = 4, DATA\_WIDTH = 8)(

input wr\_clk,

input rd\_clk,

input wr\_en,

input rd\_en,

input [ADDR\_WIDTH-1:0] wr\_addr,

input [ADDR\_WIDTH-1:0] rd\_addr,

input [DATA\_WIDTH-1:0] din,

output reg [DATA\_WIDTH-1:0] dout

);

reg [DATA\_WIDTH-1:0] mem [(1<<ADDR\_WIDTH)-1:0];

always @(posedge wr\_clk) begin

if (wr\_en)

mem[wr\_addr] <= din;

end

always @(posedge rd\_clk) begin

if (rd\_en)

dout <= mem[rd\_addr];

end

endmodule

// 2-Flip-Flop Synchronizer

module sync\_2ff #(parameter WIDTH = 4)(

input clk,

input [WIDTH-1:0] d\_in,

output reg [WIDTH-1:0] d\_out

);

reg [WIDTH-1:0] sync1;

always @(posedge clk) begin

sync1 <= d\_in;

d\_out <= sync1;

end

endmodule

// Testbench for Dual-Port RAM

module tb\_dual\_port\_ram;

reg wr\_clk = 0, rd\_clk = 0;

reg wr\_en = 0, rd\_en = 0;

reg [3:0] wr\_addr = 0, rd\_addr = 0;

reg [7:0] din = 0;

wire [7:0] dout;

dual\_port\_ram dut(

.wr\_clk(wr\_clk), .rd\_clk(rd\_clk),

.wr\_en(wr\_en), .rd\_en(rd\_en),

.wr\_addr(wr\_addr), .rd\_addr(rd\_addr),

.din(din), .dout(dout)

);

always #5 wr\_clk = ~wr\_clk;

always #7 rd\_clk = ~rd\_clk;

initial begin

$display("Starting Dual-Port RAM Testbench");

#10 wr\_en = 1; din = 8'hA5; wr\_addr = 4'd2;

#10 wr\_en = 0;

#20 rd\_en = 1; rd\_addr = 4'd2;

#20 rd\_en = 0;

#10 $finish;

end

endmodule

// Testbench for 2-FF Synchronizer

module tb\_sync\_2ff;

reg clk = 0;

reg [3:0] d\_in = 0;

wire [3:0] d\_out;

sync\_2ff #(4) dut(

.clk(clk), .d\_in(d\_in), .d\_out(d\_out)

);

always #5 clk = ~clk;

initial begin

$display("Starting Synchronizer Testbench");

#10 d\_in = 4'b1010;

#20 d\_in = 4'b1100;

#20 d\_in = 4'b0001;

#20 $finish;

end

endmodule